Možnosti vývoje aplikací v CRM systému Salesforce

Bc. Lukáš Hurtík

Diplomová práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav počítačových a komunikačních systémů

Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Lukáš Hurtík**
Osobní číslo:	**A21816**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Počítačové a komunikační systémy**
Forma studia:	**Kombinovaná**
Téma práce:	**Možnosti vývoje aplikací v CRM systému Salesforce**
Téma práce anglicky:	**Application Development Options in Salesforce CRM**

Zásady pro vypracování

1. Popište současný stav možnosti vývoje aplikací v různých CRM systémech.
2. Zaměřte se na systém Salesforce a popište možnosti vývoje aplikací včetně vytváření automatizačních procesů v tomto systému.
3. Navrhněte ukázkové projekty s využitím systému Salesforce.
4. Realizujte vývoj navržených automatizačních procesů v systému Salesforce a popište klíčové části řešení a doporučené postupy.
5. Demonstrujte výsledky a formulujte závěr.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. GOODEY, Paul. Salesforce CRM – The Definitive Admin Handbook: Build, configure, and customize Salesforce CRM and mobile solutions. 5th Edition. United Kingdom: Packt Publishing, 2019. ISBN 1789619785.
2. PEARCE, Michael. Customer Relationship Management: How To Develop and Execute a CRM Strategy. USA: Business Expert Press, 2021. ISBN 1953349641.
3. APPLEMAN, Dan. Advanced Apex Programming in Salesforce. 4th Edition. USA: Desaware Publishing, 2018. ISBN 1936754126.
4. BATTISSON, Paul. Mastering Apex Programming: A developer's guide to learning advanced techniques and best practices for building robust Salesforce applications. United Kingdom: Packt Publishing, 2020. ISBN 1800200927.
5. SHRIVASTAVA, Mohith. Learning Salesforce Lightning Application Development: Build and test Lightning Components for Salesforce Lightning Experience using Salesforce DX. United Kingdom: Packt Publishing, 2018. ISBN 1787124673.

Vedoucí diplomové práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **2. prosince 2022**
Termín odevzdání diplomové práce: **24. května 2023**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

Ing. Miroslav Matýsek, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 8. prosince 2022

Jméno, příjmení: Lukáš Hurtík

Název bakalářské/diplomové práce: Možnosti vývoje aplikací v CRM systému Salesforce

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 14.05.2023

Lukáš Hurtík, v. r.
podpis diplomanta

ABSTRAKT

Cieľom tejto práce je priblížiť čitateľom systémy riadenia vzťahov so zákazníkmi. Postupne budú čitateľom predstavené rôzne CRM systémy a možnosti vývoja aplikácii pre dané systémy. Čitatelia budú oboznámení s momentálne najpoužívanejším CRM systémom na trhu, so systémom Salesforce. Možnosti vývoja aplikácii v systéme Salesforce budú demonštrované na praktických príkladoch. Okrem iného budú predstavené deklaratívne a programové nástroje na vytváranie aplikácií v Salesforce.

Klíčová slova:

CRM systémy, Salesforce, programovací jazyk Apex

ABSTRACT

The objective of the thesis is to introduce readers to customer relationship management systems. Gradually, the reader will be introduced to different CRM systems and the possibilities of developing applications for these systems. Readers will be introduced to the most widely used CRM system on the market at the moment, Salesforce. The possibilities of application development in Salesforce will be demonstrated with practical examples. Among other things, declarative and programmatic tools for creating applications in Salesforce will be demonstrated.

Keywords:

CRM systems, Salesforce, Apex programming language

Rád by som poďakoval mojej manželke, mame a celej rodine, všetkým priateľom a vyučujúcim, ktorí ma v priebehu štúdia podporovali. Avšak osobitné poďakovanie patrí pánovi Ing. et Ing. Erikovi Královi, Ph.D. za odborné vedenie, cenné rady a čas, ktorý mi venoval počas vypracovávania tejto práce.

Vyhlasujem, že predložená verzia práce a elektronická verzia nahraná do IS/STAG sú totožné.

„To, ako zhromažďujete, spravujete a používate informácie,

rozhoduje o tom, či vyhráte alebo prehráte.“

Bill Gates

OBSAH

ÚVOD

Systémy riadenia vzťahov so zákazníkmi (CRM) sa stali neoddeliteľným nástrojom podnikov na efektívne riadenie ich interakcií so zákazníkmi. Tieto systémy umožňujú firmám zefektívniť predajné procesy, zvýšiť angažovanosť zákazníkov a v konečnom dôsledku zvýšiť ich spokojnosť. V oblasti systémov CRM sa ako vedúca platforma presadila spoločnosť Salesforce, ktorá ponúka komplexný balík funkcií a rozsiahle možnosti vývoja aplikácií.

Cieľom tejto práce je preskúmať možnosti vývoja aplikácií v rámci systému Salesforce a poskytnúť čitateľom cenné poznatky o využití jeho funkcií. Na dosiahnutie tohto cieľa práca kombinuje dôkladný prehľad literatúry, analýzu teoretických základov CRM systémov a vývoja aplikácií a praktické ukážky metód vývoja v systéme Salesforce.

Výskum realizovaný v tejto práci je vedený potrebou preklenúť medzeru medzi teoretickými poznatkami a praktickou implementáciou v kontexte vývoja aplikácií v systéme Salesforce. Poskytnutím komplexných poznatkov a praktických návodov chce táto práca umožniť čitateľom optimalizovať ich stratégie CRM a zvýšiť angažovanosť ich zákazníkov.

V rámci tejto práce čitatelia nájdu hĺbkové preskúmanie teoretických základov, historického vývoja a súčasných trendov v oblasti systémov CRM a vývoja aplikácií. Následne sa pozornosť zužuje na Salesforce, najpoužívanejší CRM systém na trhu. Podrobné diskusie o dátovom modeli Salesforce a možnostiach vývoja aplikácií poskytujú čitateľom dôkladné pochopenie možností tejto platformy.

V tejto práci sú uvedené aj praktické príklady vývoja aplikácií v rámci Salesforce, pričom sú prezentované deklaratívne aj programové prístupy. Prostredníctvom týchto príkladov čitatelia získajú praktické skúsenosti a prehľad o efektívnosti rôznych metód vývoja, čo im umožní prijímať informované rozhodnutia na základe reálnych požiadaviek klientov.

Zistenia tejto práce poskytujú cenné poznatky o optimálnych metódach vývoja aplikácií v rámci Salesforce. Zapracovaním týchto zistení do svojich stratégií CRM môžu čitatelia efektívne využívať možnosti Salesforce na uspokojenie vyvíjajúcich sa potrieb svojich klientov.

Na záver možno konštatovať, že táto práca slúži ako komplexný zdroj informácií pre pochopenie vývoja aplikácií Salesforce v kontexte systémov CRM. Jej prínos spočíva v syntéze teoretických poznatkov, praktických ukážok a hodnotiacich analýz. Tým, že táto práca poskytuje čitateľom potrebné vedomosti a nástroje, umožňuje čitateľom naplno využiť

potenciál Salesforce, optimalizovať svoje stratégie CRM a dosiahnuť úspech v konkurenčnom podnikateľskom prostredí.

I. TEORETICKÁ ČASŤ

1 CRM A MOŽNOSTI VÝVOJA APLIKÁCII

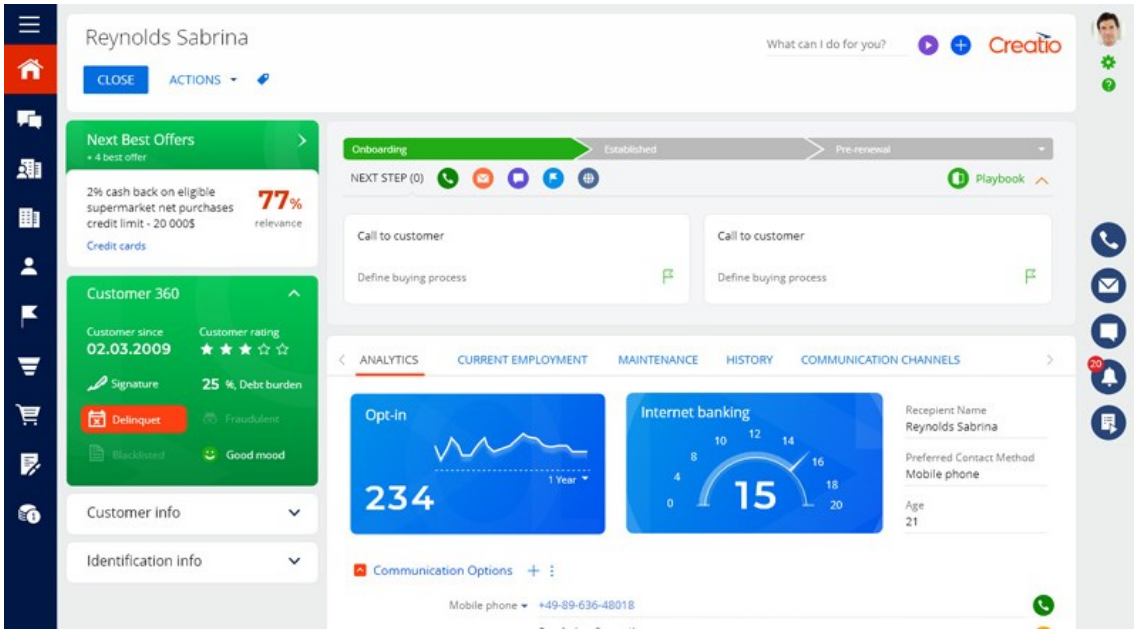
Koncepcia riadenia vzťahov so zákazníkmi (CRM) zahŕňa rôzne pohľady a interpretácie odborníkov v tejto oblasti, čo odráža mnohostrannú povahu spôsobu, akým spoločnosti riadia svoje interakcie so zákazníkmi. Hoci sa názory líšia, väčšina odborníkov sa zhoduje v tom, že CRM zahŕňa riadenie vzťahov so zákazníkmi v rámci spoločností. Vo všeobecnosti sa diskutuje o štyroch rôznych definíciách, pričom Peppers a Rogers a Peelen a Beltman ponúkajú dve významné perspektívy. Z týchto tvrdení vyplýva, že CRM je stratégia, ktorú musí prijať celá firma. Peppers a Rogers definujú riadenie vzťahov so zákazníkmi ako celopodnikovú stratégiu na dosiahnutie cieľov špecifických pre zákazníka prostredníctvom prijatia opatrení špecifických pre zákazníka. Podľa Peelena a Beltmana je riadenie vzťahov so zákazníkmi obchodná stratégia a ovplyvňuje organizáciu ako celok, tzn. aj všetky jej časti: marketing, IT, služby, logistiku, financie, výrobu a vývoj, ľudské zdroje, manažment atď. [1] [2]

Ďalšie definície poukazujú na konkrétnejšie využitie technológii, ako je vidieť v definícii Boseho a Xu. Bose definuje riadenie vzťahov so zákazníkmi ako integráciu technológií a obchodných procesov používaných na uspokojenie potrieb zákazníka počas akejkoľvek interakcie. Xu spolu so spoluautormi dokonca posúva definíciu ešte ďalej, keď zahŕňa online aspekt spracovania vzťahov so zákazníkmi. Podľa neho je riadenie vzťahov so zákazníkmi termín informačného priemyslu pre metodiky, softvér a zvyčajne aj internetové možnosti, ktoré pomáhajú podniku organizovane riadiť vzťahy so zákazníkmi. Tiež tvrdí, že ak sa CRM používa správne, môže zvýšiť schopnosť organizácie dosiahnuť konečný cieľ, ktorým je udržať si zákazníkov, a v dôsledku toho získať strategickú výhodu nad konkurenciou. Avšak je to tiež spôsob, ako naplniť marketingový cieľ lepšie slúžiť zákazníkovi. [3] [4]

Všetky vyššie spomenuté definície je možné zosumarizovať do jednej všeobecnej. CRM poskytuje mnohé zlepšenia procesov, aby sa predalo viac tovaru a služieb. Podnik musí zarábať peniaze, inak nemá zmysel ho prevádzkovať, takže základným cieľom CRM sú efektívne a účinné vzťahy so zákazníkmi, ktoré majú zlepšiť ziskovosť organizácie. Efektívnosť pri zvyšovaní ziskovosti sa dosahuje prostredníctvom získavania, analýzy a zvyšovania znalostí o zákazníkoch.

Informačné technológie sú veľmi dôležité pre fungovanie systému CRM a preto by sa dal CRM systém z hľadiska IT definovať ako o špecifický informačný systém, ktorý zbiera a ukladá informácie o firemných zákazníkoch. Poskytuje tak ucelený prehľad o komunikácii

medzi zákazníkom a predávajúcim, o nákupoch a všetkých aktivitách zákazníka. Je vhodný pre podnikanie typu B2B a aj B2C. Možno ho použiť samostatne alebo aj ako súčasť podnikového informačného systému (ERP). Práve preto si podnikatelia často tieto dva systémy zamieňajú. Kým CRM je systém určený iba na správu zákazníkov a aktivít firmy smerom k nim, tak ERP je oveľa komplexnejšie riešenie, ktoré obsahuje navyše aj množstvo ďalších modulov – napríklad nákup tovaru, sklad, účtovníctvo. [5]



Obrázok 1 Príklad moderného CRM systému [6]

Systémy CRM môžu zahŕňať širokú škálu nástrojov, od základnej správy kontaktov a sledovania potenciálnych zákazníkov až po pokročilejšiu analytiku a automatizáciu marketingu. Moderný systém CRM zahŕňa funkcie, ako sú:

- **Správa kontaktov:** Ukladanie všetkých relevantných informácií o zákazníkoch vrátane kontaktných údajov, histórie nákupov a histórie komunikácie.
- **Monitorovanie potenciálnych zákazníkov:** Správa a sledovanie potenciálnych zákazníkov od prvého kontaktu až po predaj tovaru.
- **Automatizácia predaja:** Zjednodušený proces predaja automatizáciou úloh, ako je priraďovanie potenciálnych zákazníkov, následné pripomenutia a sledovanie pokroku pri vybavovaní objednávky.
- **Automatizácia marketingu:** Plánovanie a realizovanie cielenej marketingovej kampane s využitím údajov o zákazníkoch a analýzy ich správania.
- **Analytika:** Prehľad o správaní zákazníkov, výkonnosti predaja a účinnosti marketingu pomocou nástrojov na analýzu údajov. [5]

Napriek množstvu vstavaných funkcií, ktoré väčšina moderných CRM systémov ponúka, často sa stáva, že organizácia, ktorá si od dodávateľa chce zakúpiť licenciu na CRM systém, vyžaduje dodatočnú špecifickú úpravu systému. Vtedy prichádzajú na rad programátori, ktorí upravia daný systém, prípadne ho doplnia o nové funkcie tak, aby spĺňal dané kritéria zákazníka.

1.1 História CRM

Existuje mylná predstava, že riadenie vzťahov so zákazníkmi je niečo nové, čo vzniklo koncom dvadsiateho storočia. Tento pojem je síce nový, ale táto praktika je stará ako obchodovanie a dokonca staršia ako peniaze. Prvý zaznamenaný príklad obchodu sa datuje do obdobia pred 20 000 rokmi, keď sa na území dnešnej Papuy-Novej Guiney obchodovalo s obsidiánovými nožmi medzi ostrovmi, ktoré mali cenné ložiská obsidiánu, a tými, ktoré ich nemali. Obsidián bol v týchto raných dobách celosvetovo cenený pre svoju vhodnosť na výrobu nožov a iných rezných nástrojov a obchodovalo sa s ním v ranej Európe a Ázii, ako aj v Amerike.

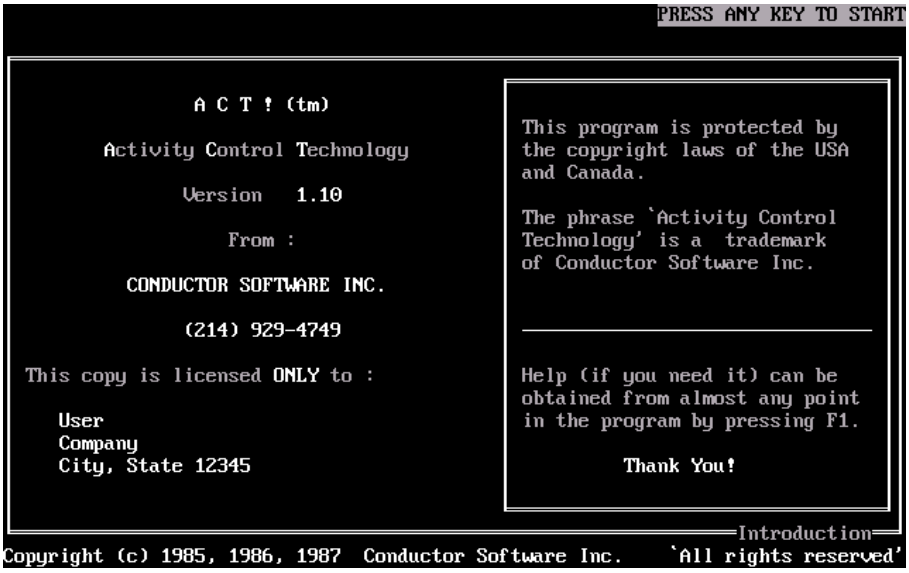
Už v paleolite musela existovať myšlienka, že je jednoduchšie predať niečo existujúcemu zákazníkovi ako nájsť nového a že je výhodné so zákazníkom udržovať vzťah. Vedenie záznamov o tom, kto čo vlastnil a kto komu čo dlhoval, si vyžadovalo určitú formu zápisu a trvalej evidencie, takže vlastne sa jednalo o určitú formu účtovníctva. Prirodzene, súčasťou účtovných záznamov boli mená a miesta, ktoré boli najstaršou formou databázy zákazníkov. Uplatňovala sa aj určitá raná forma segmentácie, pravdepodobne na základe osobného bohatstva a platobnej schopnosti. [5]

Tento proces zostal v priebehu storočí viac alebo menej rovnaký a zmenil sa až s príchodom prvým komerčných počítačov. Koncom 50. a začiatkom 60. rokov 20. storočia sa hlavný dôraz kládol na vedenie účtovných záznamov, či už na účtovných strojoch alebo počítačoch, v bankách, na burzách cenných papierov a vo vládnych inštitúciách. Keďže cena počítačov sa neskôr markantne znížila, v 70. rokoch si mohli už aj malé podniky dovoliť zapojiť sa do počítačovej revolúcie. Postupne sa ukázalo, že automatizácia by mohla byť užitočná aj v iných oblastiach podnikania, najmä v predaji, a za niekoľko krátkych rokov sa objavilo CRM v naozaj digitálnej podobe.

Začiatky CRM, ako ho poznáme dnes, sa začali v 80. rokoch minulého storočia. Robert a Kate Kestnbaumovci boli priekopníkmi databázového marketingu. Databázový marketing

bola forma priameho marketingu, ktorá štatisticky analyzovala databázu zákazníkov s cieľom určiť, ktorí zákazníci budú s najväčšou pravdepodobnosťou reagovať na marketingovú kampaň. O tom, kto skutočne vynašiel moderné CRM sa veľa diskutuje, avšak s prihliadnutím na veľký prínos Boba Kestnbauma modernému marketingu a marketingovým stratégiám, možno deklarovať že „otcom“ myšlienky moderného CRM je práve on. [7]

Avšak niektorí odborníci polemizujú o prínose Boba Kestnbauma a tvrdia, že moderné CRM vynašli dvaja Texasania, Pat Sullivan a Mike Muhney, so svojím produktom ACT!, vydaným v roku 1986. ACT!, teda Automated Contact Tracking, čo v preklade znamená automatizované sledovanie kontaktov možno považovať za prvý automatizovaný CRM. Tento nástroj si v priebehu rokov prešiel mnohými zmenami názvov a akvizíciami, ale existuje dodnes. [7]



Obrázok 2 Jedna z prvých verzií CRM systému ACT! [8]

S nezastaviteľným rozvojom informačných technológií v 90. rokoch 20. storočia vzniklo mnoho nových produktov, ktoré spravovali údaje o zákazníkoch. Tieto produkty, ktoré boli spojením databázového marketingu a správy kontaktov, sa označovali skratkou SFA (Sales Force Automation – v preklade automatizácia predajnej sily). Jeden z prvých takýchto produktov bol Siebel založený Tomom Siebelom, ktorý pracoval v spoločnosti Oracle, z ktorej v roku 1993 odišiel, aby založil svoju vlastnú spoločnosť Siebel Systems, ktorá sa vo svojej dobe stala lídrom na trhu. [7]

Spoločnosť Salesforce nevyvolala pri svojom uvedení na trh v roku 1999 veľký rozruch, väčšina konkurentov považovala cloudové služby za výstrelok a nie za dobrý prostriedok pre CRM. Počiatočná pozornosť spoločnosti Salesforce bola zameraná na malé a stredné

podniky a v čase, keď sa konkurenti prebudili a zistili, že zákazníci prechádzajú na cloudové riešenia, sa spoločnosť Salesforce stala najväčším svetovým dodávateľom CRM.

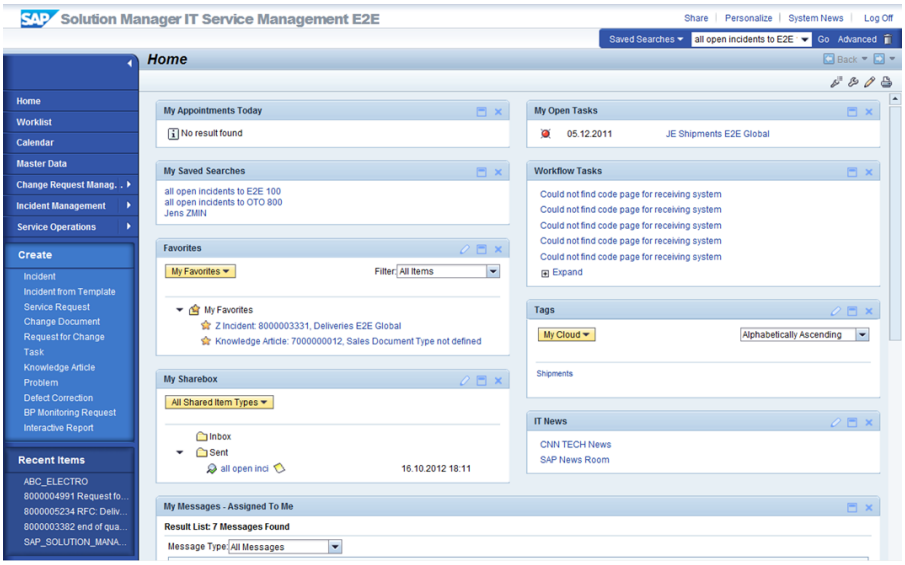
V súčasnosti sa zdá, že trh s novými produktmi CRM ešte nedosiahol bod nasýtenia. Nové spoločnosti naďalej prichádzajú na trh s cloudovými produktmi, zatiaľ čo existujúci dodávatelia zmenili svoje licenčné modely, aby ponúkli cloudové alternatívy k už predstaveným produktom. Zdá sa teda, že budúcnosťou CRM systémov sú práve cloudové riešenia.

Avšak s príchodom smartfónov začali mnohé spoločnosti ponúkať aj mobilné verzie svojich produktov. Tempo akým napreduje vývoj nových technológií je také rýchle, že mnohí dodávatelia bojujú o to, aby udržali krok s najnovším výdobytkami doby, od chatbotov až po umelú inteligenciu. Nedá sa teda jednoznačne určiť smer akým sa bude trh s CRM systémami uberať v budúcnosti. [9]

1.2 SAP CRM

SAP CRM je softvérové riešenie vyvinuté spoločnosťou SAP AG v roku 1992, ktoré ponúka svojim klientom zvýšenie efektivity riadenia vzťahov so zákazníkmi. Toto riešenie umožňuje spoločnostiam riadiť interakcie so zákazníkmi prostredníctvom viacerých kanálov vrátane call centier, e-mailu, sociálnych sietí a tiež mobilných zariadení. Poskytuje komplexný pohľad na údaje o zákazníkoch a umožňuje spoločnostiam sledovať interakcie so zákazníkmi, spravovať zákaznícke účty a analyzovať správanie zákazníkov v reálnom čase.

K dispozícii je lokálna verzia ale aj cloudová, čo spoločnostiam poskytuje flexibilitu pri výbere modelu nasadenia, ktorý najlepšie vyhovuje ich potrebám. [10]



Obrázok 3 Ukážka užívateľského prostredia SAP CRM [11]

1.2.1 Dátový model SAP CRM

Dátový model SAP CRM predstavuje jeho základnú štruktúru, ktorá definuje spôsob ukladania a organizácie údajov v systéme SAP CRM. Pozostáva z tabuliek, polí, vzťahov a ďalších prvkov, ktoré spoločne predstavujú rôzne objekty a entity v systéme. SAP CRM používa hierarchický dátový model, kde sú údaje uložené v stromovej štruktúre.

Základné rozdelenie dátového modelu SAP CRM:

- **Obchodní partneri:** Obchodní partneri sú jadrom dátového modelu SAP CRM. Predstavujú zákazníkov, predajcov a iné subjekty, ktoré komunikujú s organizáciou.
- **Účty:** Účty sú organizačné entity, ktoré sú spojené s obchodnými partnermi.
- **Produkty:** Produkty sú tovary a služby, ktoré organizácia predáva svojim zákazníkom.
- **Objednávky predaja:** Objednávky predaja predstavujú predajné transakcie medzi organizáciou a jej zákazníkmi.

Toto je len niekoľko príkladov objektov a entít v dátovom modeli SAP CRM. Dátový model je vysoko konfigurovateľný a možno ho prispôsobiť špecifickým potrebám každej organizácie. Pochopením dátového modelu môžu vývojári vytvárať vlastné aplikácie a integrácie, ktoré využívajú možnosti systému SAP CRM. [10]

1.2.2 Možnosti vývoja aplikácií v SAP CRM

Vývoj aplikácií pre SAP CRM je realizovaný v jazyku ABAP CRM (Advanced Business Application Programming - Pokročilé programovanie podnikových aplikácií). ABAP CRM je proprietárny jazyk vyvinutý spoločnosťou SAP a používa sa pri vývoji aplikácií v SAP CRM. ABAP CRM je rozšírením jazyka ABAP, ktorý sa používa na programovanie v iných moduloch SAP, ako sú FI/CO (Finance/Controlling - Financie/kontrolovanie), MM (Materials Management - Manažment materiálov) a SD (Sales and Distribution - Predaj a distribúcia).

Niektoré z kľúčových vlastností programovacieho jazyka ABAP CRM sú:

- **Objektové programovanie:** ABAP CRM poskytuje framework na vytváranie a správu obchodných objektov v systéme SAP CRM. Tieto objekty môžu predstavovať zákazníkov, produkty, objednávky a iné entity v systéme.

- **Vývoj používateľského rozhrania:** ABAP CRM poskytuje celý rad nástrojov a rozhraní API na vývoj používateľských rozhraní v systéme SAP CRM. Tieto nástroje zahŕňajú webové rozhrania, desktopové aplikácie a mobilné aplikácie.
- **Integrácia:** ABAP CRM poskytuje celý rad integračných nástrojov a rozhraní API na prepojenie systému SAP CRM s inými modulmi SAP a externými systémami.
- **Vykazovanie:** ABAP CRM obsahuje výkonný nástroj na vytváranie správ, ktorý umožňuje vývojárom vytvárať správy na mieru a informačné panely v systéme SAP CRM.

Predpokladajme, že spoločnosť chce vytvoriť aplikáciu na mieru v systéme SAP CRM, ktorá umožní servisným agentom spravovať zákaznícke požiadavky. Aplikácia by mala servisným agentom umožniť zobrazovať a spravovať servisné požiadavky, sledovať stav každej požiadavky a vytvárať správy o počte a typoch prijatých požiadaviek. [12]

Nižšie je príklad kódu ABAP CRM, ktorý by sa mohol použiť na vytvorenie tejto aplikácie.

```
DATA: lt_requests TYPE crmt_service_request_t,
      ls_request TYPE crmt_service_request.

CALL FUNCTION 'CRM_ORDER_READ_OW'
  EXPORTING
    iv_object_id    = lv_object_id
    iv_object_type  = 'BUS2000116'
  TABLES
    et_service      = lt_requests
  EXCEPTIONS
    no_authorization      = 1
    communication_failure = 2
    object_not_found      = 3
    quantity_limit_error  = 4
    sytem_failure         = 5
    OTHERS                = 6.

IF sy-subrc EQ 0.
  LOOP AT lt_requests INTO ls_request.
    WRITE: / ls_request-guid,
             ls_request-description,
             ls_request-status,
             ls_request-created_at.
  ENDLOOP.
ENDIF.
```

Obrázok 4 Ukážka použitia programovacieho jazyku ABAP CRM

Na začiatku kódu sa deklarujú dva dátové objekty: tabuľka servisných požiadaviek a jedna servisná požiadavka. Následne sa zavolá funkčný modul CRM_ORDER_READ_OW, ktorý sa používa na načítanie údajov pre zadaný objekt v systéme SAP CRM. Potom sú zadefinované parametre funkčného modulu, ktoré slúžia na určenie objektu, pre ktorý sa majú načítať údaje. Následne sa deklaruje parameter, ktorý určuje tabuľku, do ktorej sa majú pridať vyhľadané servisné požiadavky. Neskôr sú uvedené výnimky, ktoré môže vyvolať funkčný modul CRM_ORDER_READ_OW. Potom sa vykoná podmienka, v ktorej sa kontroluje, či pole sy-subrc má hodnotu 0, čo znamená, že volanie funkcie CRM_ORDER_READ_OW bolo úspešné. Sy-subrc je to celočíselná systémová premenná, ktorá obsahuje stav posledného vykonaného príkazu alebo volania funkcie. Po volaní modulu funkcie CRM_ORDER_READ_OW sa premenná sy-subrc automaticky aktualizuje o návratový kód funkcie. Hodnota 0 znamená úspech, nenulová hodnota znamená, že došlo k chybe. Ak teda pri volaní funkčného modelu CRM_ORDER_READ_OW nedošlo k chybe, spustí sa cyklus v tabuľke, ktorá obsahuje načítané servisné požiadavky. Na konci programu sa postupne po jednom na výstupnú obrazovku vypíšu polia objektu.

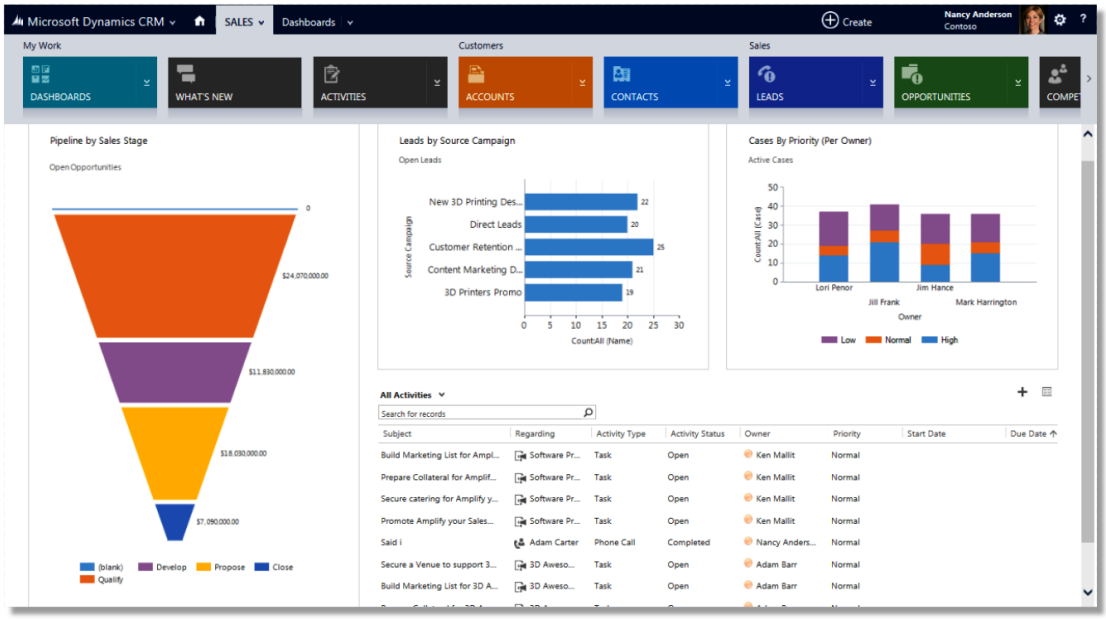
1.3 Microsoft Dynamics CRM

Microsoft Dynamics CRM je softvér na riadenie vzťahov so zákazníkmi, ktorý vyvinula spoločnosť Microsoft v roku 2003. Microsoft Dynamics CRM ponúka celý rad funkcií, ktoré možno prispôsobiť špecifickým potrebám rôznych organizácií. Medzi kľúčové funkcie patria:

- **Riadenie predaja:** Pomáha organizáciám spravovať ich predajné kanály, sledovať potenciálnych zákazníkov a potenciálne príležitosti, spravovať prognózy predaja a analyzovať výkonnosť predaja.
- **Automatizácia marketingu:** Umožňuje organizáciám vytvárať a spravovať marketingové kampane, a tiež analyzovať výkonnosť kampaní.
- **Zákaznícka podpora:** Umožňuje organizáciám spravovať otázky zákazníkov a požiadavky na podporu a analyzovať výkonnosť služieb zákazníkom.
- **Integrácia so sociálnymi médiami:** Umožňuje organizáciám monitorovať a komunikovať so zákazníkmi na platformách sociálnych médií, ako sú Twitter, Facebook a LinkedIn.

- **Mobilná verzia:** Poskytuje mobilný prístup k systému CRM, čo umožňuje predajcom a zástupcom zákazníckeho servisu pristupovať k dôležitým informáciám na cestách.
- **Integrácia s inými produktmi Microsoft:** Zrejme najvýraznejšia kľúčová funkcia Microsoft Dynamics CRM, ktorá tento systém odlišuje od ostatných je integrácia s ďalšími produktmi spoločnosti Microsoft, ako sú Outlook, Excel, SharePoint a Power BI.

Microsoft Dynamics CRM je k dispozícii v lokálnej, a tiež cloudovej verzii (Microsoft Dynamics 365), vďaka čomu je flexibilný a škálovateľný podľa potrieb rôznych organizácií. Okrem toho ponúka používateľsky prívetivé rozhranie, ktoré používateľom uľahčuje navigáciu a používanie softvéru. [13]



Obrázok 5 Ukážka užívateľského prostredia systému Microsoft Dynamics CRM [14]

1.3.1 Dátový model Microsoft Dynamics CRM

Dátový model Microsoft Dynamics CRM je v podstate relačná databáza, ktorá uchováva informácie o entitách a ich vzťahoch. Entita v systéme Microsoft Dynamics CRM je podobná tabuľke v relačnej databáze a predstavuje špecifický typ údajov, ktoré je potrebné sledovať. Medzi príklady entít v systéme Microsoft Dynamics CRM patria účty, kontakty, potenciálni zákazníci, príležitosti a prípady.

Každá entita v systéme Microsoft Dynamics CRM má súbor atribútov, ktoré definujú údaje, ktoré môže uchovávať. Atribúty sú podobné stĺpcom v tabuľke a predstavujú špecifické

informácie, ktoré je potrebné o entite sledovať. Napríklad entita účet môže mať atribúty ako názov účtu, adresa, telefónne číslo a webová stránka.

Entity v systéme Microsoft Dynamics CRM môžu mať vzťahy s inými entitami. Vzťahy definujú, ako sú dve alebo viac entít navzájom prepojené a aký typ vzťahu majú. V systéme Microsoft Dynamics CRM existujú tri typy vzťahov: jeden k viacerým, viac k jednému a viac k viacerým.

Dátový model v aplikácii Microsoft Dynamics CRM možno rozšíriť tak, aby spĺňal špecifické obchodné požiadavky. Na sledovanie ďalších údajov alebo na definovanie nových vzťahov medzi existujúcimi entitami možno vytvoriť vlastné entity, atribúty a vzťahy. Okrem toho Microsoft Dynamics CRM poskytuje celý rad nástrojov a rozhraní API na interakciu s údajmi v systéme, čo umožňuje vývojárom vytvárať aplikácie na mieru. [13]

1.3.2 Možnosti vývoja aplikácií v Microsoft Dynamics CRM

Microsoft Dynamics CRM používa na vývoj a prispôsobenie rôznych komponentov platformy viacero programovacích jazykov a frameworkov, napríklad:

- **C#** - C# je často používaný programovací jazyk na vývoj prídavných modulov na mieru, pracovných postupov a iných komponentov na mieru v systéme Microsoft Dynamics CRM. Jazyk C# sa používa aj na vývoj webových aplikácií na mieru pomocou webového rozhrania Microsoft Dynamics CRM Web API.
- **.NET Framework** - .NET Framework sa používa na vývoj aplikácií na platforme Windows. Microsoft Dynamics CRM je postavený na frameworku .NET, takže vývojári môžu na vytváranie aplikácií na mieru a integrácií používať rôzne knižnice a frameworky .NET, ako napríklad Entity Framework.
- **JavaScript** - JavaScript sa v systéme Microsoft Dynamics CRM vo veľkej miere používa na vývoj logiky na mieru na strane klienta, napríklad na validáciu údajov zadaných užívateľom cez používateľské rozhranie. Dynamics CRM poskytuje aj JavaScript API pre interakciu s údajmi a funkciami platformy.
- **HTML/CSS -** Microsoft Dynamics CRM poskytuje webové používateľské rozhranie a vývojári môžu používať HTML a CSS na prispôsobenie vzhľadu aplikácie.
- **FetchXML -** FetchXML je dožadovací jazyk, ktorý sa používa na získavanie údajov zo systému Microsoft Dynamics CRM. Vývojári môžu používať jazyk FetchXML na vytváranie vlastných požiadaviek na získavanie a manipuláciu s údajmi.

Celkovo Microsoft Dynamics CRM ponúka širokú škálu nástrojov a frameworkov na vývoj a prispôsobenie platformy, takže vývojári si môžu vybrať najvhodnejšie programovacie jazyky a frameworky na základe svojich požiadaviek a odborných znalostí.

Predpokladajme, že klient chce vytvoriť aplikáciu na mieru v systéme Microsoft Dynamics CRM, ktorá načíta údaje o účtoch zo systému Microsoft Dynamics CRM webového rozhrania Microsoft Dynamics CRM Web API. Nižšie, na obrázku č. 6, sa nachádza príklad kódu v jazyku C#, ktorý by sa mohol použiť na vytvorenie tejto aplikácie. [13]

```
using System;
using System.Net.Http;
using System.Net.Http.Headers;
using Newtonsoft.Json;

// Define the URL of the Dynamics CRM organization and the API version
string crmUrl = "https://myorganization.crm.dynamics.com/api/data/v9.1/";

// Define the credentials for the Dynamics CRM user
string crmUsername = "myusername";
string crmPassword = "mypassword";

// Create an HttpClient object to send requests to the Dynamics CRM web API
var httpClient = new HttpClient();
httpClient.BaseAddress = new Uri(crmUrl);

// Set the authentication header for the HttpClient object
var authHeader = new AuthenticationHeaderValue("Basic",
                    Convert.ToBase64String(System.Text.Encoding.ASCII.GetBytes(
                        $"{crmUsername}:{crmPassword}")));
httpClient.DefaultRequestHeaders.Authorization = authHeader;

// Define the query to retrieve account data from Dynamics CRM
string query = "/accounts?$select=name,primarycontactid";

// Send the request to the Dynamics CRM web API
HttpResponseMessage response = await httpClient.GetAsync(query);

// Check if the response was successful
if (response.IsSuccessStatusCode)
{
    // Parse the response content as a JSON string
    string responseContent = await response.Content.ReadAsStringAsync();

    // Deserialize the JSON string into a dynamic object
    dynamic accounts = JsonConvert.DeserializeObject(responseContent);

    // Loop through the accounts and print the name and primary contact ID
    foreach (var account in accounts.value)
    {
        Console.WriteLine($"Account name: {account.name}");
        Console.WriteLine($"Primary contact ID: {account.primarycontactid}");
    }
}
else
{
    Console.WriteLine($"Error retrieving accounts: {response.ReasonPhrase}");
}
```

Obrázok 6 Ukážka použitia programovacieho jazyku C# v systéme Microsoft Dynamics

Na prvých niekoľko riadkoch sa importujú potrebné knižnice vrátane System.Net.Http a Newtonsoft.Json. Neskôr sa definuje adresa URL organizácie v systéme Microsoft Dynamics CRM spolu s verziou webového rozhrania API, ktoré sa má použiť, a prihlasovacie údaje používateľa Microsoft Dynamics CRM. Ďalej sa vytvorí inštancia triedy HttpClient, ktorá sa bude používať na odosielanie požiadaviek na Microsoft Dynamics CRM web API. Potom sa nastaví autentifikačná hlavička pre objekt HttpClient pomocou prihlasovacích údajov zadaných predtým. Následne sa definuje reťazec požiadavky, ktorý určuje údaje, ktoré sa majú získať zo systému Microsoft Dynamics CRM (v tomto prípade meno a ID primárneho kontaktu pre všetky účty). Pomocou objektu HttpClient sa potom odošle požiadavka GET nna Microsoft Dynamics CRM web API spolu so zadaným reťazcom požiadavky. Ďalej sa skontroluje, či bola odpoveď z webového rozhrania API úspešná. Ak bola odpoveď úspešná, obsah odpovede (ktorý je vo formáte JSON) sa načíta a deserializuje do dynamického objektu. Nakoniec sa použije cyklus na iteráciu cez účty v dynamickom objekte a meno a ID primárneho kontaktu pre každý účet sa vypíše na konzolu. [13]

1.4 Oracle CRM

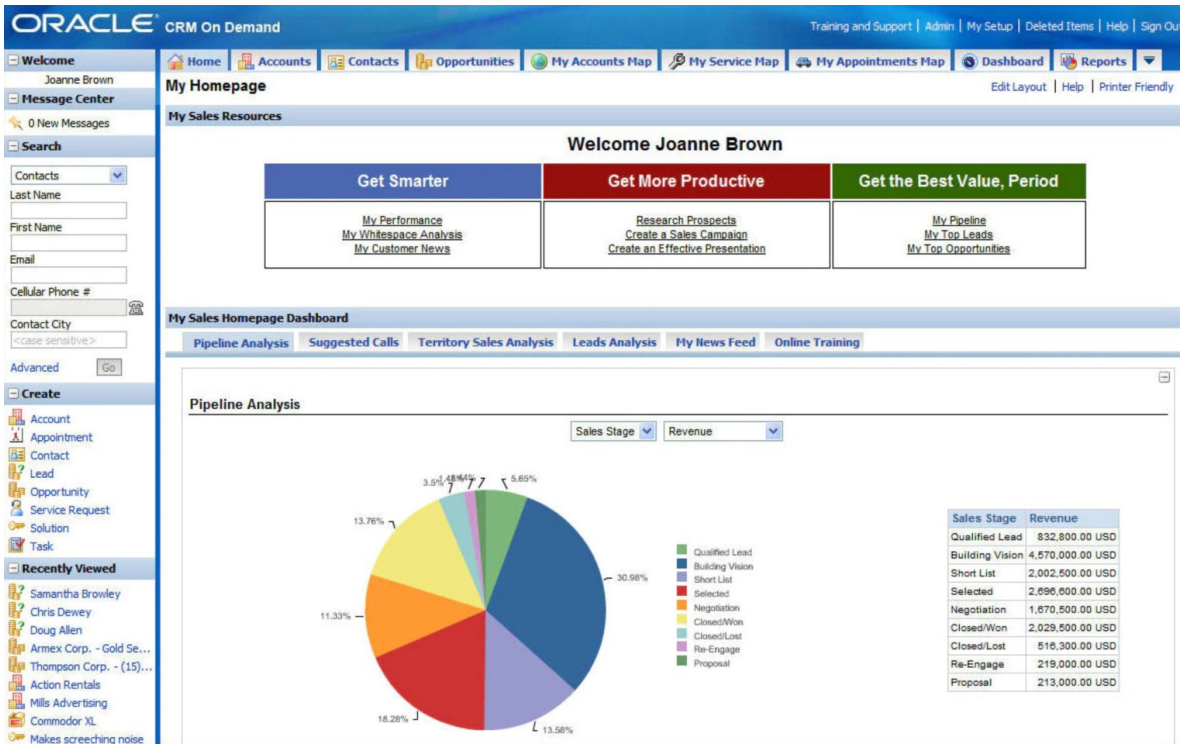
Oracle CRM, systém vydaný v roku 1998, je súbor softvérových aplikácií, ktoré organizácie používajú na riadenie interakcií so zákazníkmi, predajných aktivít a marketingových kampaní. Poskytuje celý rad nástrojov a funkcií, ktoré pomáhajú podnikom získavať, udržiavať a rozširovať svoju zákaznícku základňu.

Balík Oracle CRM obsahuje rôzne moduly, ktoré možno prispôsobiť konkrétnym potrebám podniku. Medzi tieto moduly patria:

- **Predaj:** Pomáhajú obchodným tímom riadiť ich predajné procesy, od získavania potenciálnych zákazníkov cez správu obchodných príležitostí až po realizáciu objednávok.
- **Marketing:** Poskytuje nástroje na plánovanie, realizáciu a vyhodnocovanie marketingových kampaní vrátane e-mailového marketingu, marketingu v sociálnych médiách a marketingových podujatí.
- **Zákaznícky servis:** Umožňuje organizáciám poskytovať konzistentný a vysokokvalitný zákaznícky servis prostredníctvom viacerých kanálov vrátane telefónu, e-mailu a živého chatu.
- **Obchod:** Podporuje online a offline obchodné činnosti vrátane správy objednávok a riadenia zásob

- **Vernostné služby:** Umožňuje podnikom navrhovať a spravovať vernostné programy, propagačné akcie a stimuly na zvýšenie angažovanosti a udržania zákazníkov.

Oracle CRM je postavený na robustnej a škálovateľnej architektúre, ktorá dokáže podporovať potreby veľkých podnikov. Integruje sa aj s ďalšími aplikáciami Oracle, ako sú Oracle ERP a Oracle HCM, aby poskytoval jednotný pohľad na údaje o zákazníkoch v rámci rôznych podnikových funkcií. Spoločnosť Oracle ponúka svoj CRM systém v lokálnej aj cloudovej verzii. Lokálna verzia systému Oracle CRM obsahuje ďalšie „podsystémy“ tzn. v minulosti zakúpene CRM systémy PeopleSoft Enterprise CRM a Siebel CRM, ktoré spoločnosť Oracle zakúpila aby rozšírila použiteľnosť svojho CRM systému. [15]



Obrázok 7 Ukážka užívateľského prostredia systému Oracle CRM [16]

1.4.1 Dátový model Oracle CRM

Základom dátového modelu Oracle CRM sú tabuľky, ktoré sú usporiadané do modulov zodpovedajúcich rôznym obchodným funkciám. Každá tabuľka predstavuje špecifický typ údajov a tabuľky v rámci modulu sú navzájom prepojené prostredníctvom tzv. kľúčových polí, rovnako ako v relačnej databáze. Napríklad modul Predaj môže obsahovať tabuľky príležitostí, kontaktov a účtov s kľúčovými poľami, ktoré ich spájajú.

Medzi základné moduly v dátovom modeli Oracle CRM patria:

- **Predaj:** Tento modul obsahuje tabuľky na správu predajných príležitostí, potenciálnych zákazníkov, kontaktov a účtov.
- **Marketing:** Tento modul obsahuje tabuľky na správu marketingových kampaní vrátane e-mailu, pošty a telemarketingu. Obsahuje aj tabuľky na sledovanie výkonnosti kampaní a miery odozvy.
- **Služby:** Tento modul obsahuje tabuľky na správu interakcií so zákazníkmi, vrátane servisných požiadaviek a zmlúv. Obsahuje aj tabuľky na sledovanie ukazovateľov výkonnosti služieb a spokojnosti zákazníkov.
- **Call centrum**: Tento modul obsahuje tabuľky na správu interakcií call centra vrátane prichádzajúcich, odchádzajúcich hovorov a smerovania hovorov.
- **Správa partnerov:** Tento modul obsahuje tabuľky na správu partnerských vzťahov vrátane profilov partnerov, partnerských zmlúv a ukazovateľov výkonnosti partnerov.

Oracle CRM tiež podporuje vytváranie špecifických objektov na mieru, a tiež polí na mieru, ktoré môžu používatelia konfigurovať tak, aby spĺňali špecifické obchodné požiadavky. Všeobecne je dátový model Oracle CRM navrhnutý tak, aby podporoval komplexné a vzájomne prepojené obchodné procesy spojené so správou vzťahov so zákazníkmi. Usporiadaním údajov do modulov a tabuliek poskytuje systém podnikom flexibilný a škálovateľný rámec na riadenie ich interakcií so zákazníkmi a partnermi. [15] [17]

1.4.2 Možnosti vývoja aplikácií v Oracle CRM

Programovací jazyk používaný v Oracle CRM závisí od konkrétnej aplikácie alebo súčasti, ktorá sa používa. Spoločnosť Oracle poskytuje celý rad nástrojov a technológií na vývoj a prispôsobenie aplikácií CRM vrátane:

- **Oracle Application Development Framework (ADF)** - Framework založený na jazyku Java, ktorý sa používa na vývoj a prispôsobenie aplikácií Oracle CRM.
- **Oracle Forms** - Vývojový nástroj používaný na vytváranie klasických aplikácií typu klient/server v systéme Oracle CRM.
- **Oracle PL/SQL** - Procedurálny jazyk používaný na vytváranie databázových objektov v Oracle CRM.
- **JavaScript** - Používa sa na vytváranie skriptov a prispôsobovanie komponentov používateľského rozhrania v systéme Oracle CRM.

- **HTML/CSS** - Používa sa na vytváranie a štylizovanie webových stránok a používateľskom rozhraní v systéme Oracle CRM.

V závislosti od konkrétnych potrieb projektu alebo aplikácie sa môžu používať aj iné jazyky a technológie.

Predpokladajme, že klient chce vytvoriť aplikáciu na mieru v systéme Oracle CRM, ktorá načíta vytvorí nový špecifický objekt. Nižšie, na obrázku č. 8, sa nachádza príklad kódu v jazyku Java, frameworku Oracle Application Development, ktorý by sa mohol použiť na vytvorenie tejto aplikácie. [17] [18]

```
import oracle.apps.fnd.ext.common.*;
import oracle.apps.fnd.ext.common.Session;
import oracle.apps.fnd.ext.common.SessionImpl;
import oracle.apps.fnd.ext.common.AppsContext;
import oracle.apps.fnd.ext.common.AppsContextImpl;
import oracle.apps.fnd.ext.common.EBiz;

public class CreateCustomObject {

    public static void main(String[] args) {

        // Set the EBiz credentials
        String userName = "username";
        String password = "password";
        String url = "http://mycrm.com";
        int appId = 123; // Replace with the actual application ID

        // Create a new session
        Session session = new SessionImpl();

        // Set the session context
        AppsContext context = new AppsContextImpl();
        context.setUserName(userName);
        context.setPassword(password);
        context.setResponsibilityId(-1);
        context.setApplicationId(appId);
        session.setAppsContext(context);

        // Create a new EBiz object
        EBiz ebiz = new EBiz(userName, password, url);

        // Create a new custom object
        String objectName = "CUSTOM_OBJECT_NAME";
        String objectTableName = "CUSTOM_OBJECT_TABLE_NAME";
        String objectDescription = "Custom object description";
        int objectApplicationId = appId;

        String result = ebiz.createCustomObject(objectName, objectTableName, objectDescription, objectApplicationId);

        // Print the result of the create custom object call
        System.out.println(result);

        // Release the session
        session.destroy();
    }
}
```

Obrázok 8 Ukážka použitia frameworku Oracle ADF v systéme Oracle CRM

V úvode kódu sa importujú potrebné triedy z Oracle CRM Java API. Potom sa nastavia prihlasovacie údaje EBiz pre inštanciu Oracle CRM, ku ktorej sa bude pripájať. Meno používateľa a heslo sa nahradí vlastnými prihlasovacími údajmi EBiz, adresa http://mycrm.com sa nahradí URL adresou inštancie Oracle CRM, ku ktorej sa bude pripájať a číslo 123 sa nahradí skutočným ID aplikácie Oracle CRM, v ktorej bude vytvorený špecifický objekt na mieru. Následne sa vytvorí nový objekt Session, ktorý sa používa na vytvorenie spojenia s inštanciou Oracle CRM. Potom sa vytvorí objekt AppsContext na nastavenie kontextu danej relácie. Neskôr sa definujú vlastnosti userName, password, responsibilityId a applicationId objektu AppsContext a potom sa spojí objekt AppsContext s objektom Session. Následne sa vytvorí nový objekt EBiz, ktorému sa priradia prihlasovacie údaje EBiz. Objekt EBiz sa používa na vykonávanie rôznych operácií na inštancii Oracle CRM, napríklad na vytváranie špecifických objektov na mieru. Potom sa vytvorí nový špecifický objekt na mieru v Oracle CRM pomocou metódy createCustomObject objektu EBiz. V tomto prípade sa objektu zadefinuje názov, názov tabuľky, popis a ID aplikácie špecifického objektu na mieru, ktorý chceme vytvoriť. Metóda createCustomObject vráti reťazec obsahujúci informácie o novovytvorenom vlastnom objekte. Na záver sa vypíše výsledok volania metódy createCustomObject na konzolu a následne sa objekt Session zničí, aby sa uvoľnilo spojenie s inštanciou Oracle CRM.

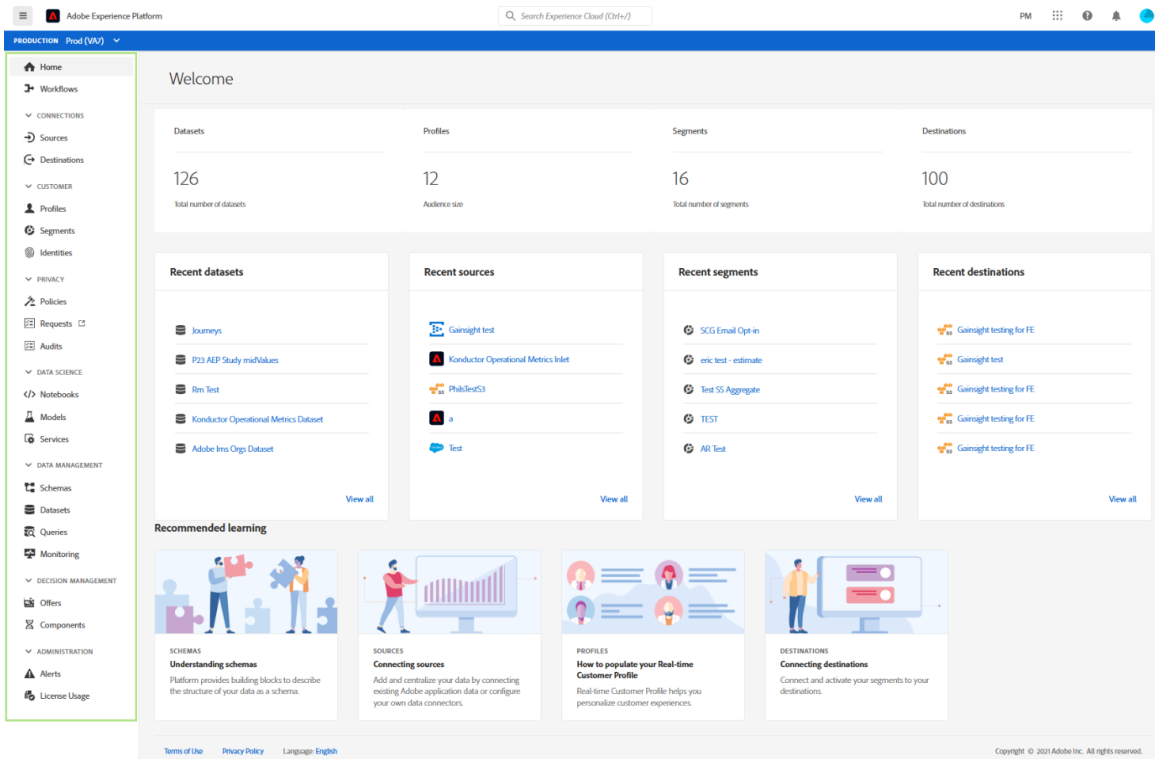
1.5 Adobe Experience Cloud

Adobe Experience Cloud nie je systémom pre riadenie vzťahov so zákazníkmi (CRM) v tradičnom zmysle, hoci poskytuje mnoho funkcií a možností, ktoré sa bežne spájajú so systémami CRM. Balík Adobe Experience Cloud obsahuje niekoľko produktov, ktoré možno použiť na správu vzťahov so zákazníkmi a zlepšenie zákazníckych skúseností, ako napríklad

- **Adobe Analytics** - Výkonný analytický nástroj, ktorý podnikom poskytuje prehľad o správaní a angažovanosti zákazníkov
- **Adobe Target** - Nástroj na personalizáciu zákazníckych skúseností, ktorý využíva údaje a poznatky na poskytovanie prispôsobeného obsahu, ponúk a správ.
- **Adobe Campaign** - Nástroj na automatizáciu marketingu, ktorý pomáha podnikom vytvárať a realizovať cielené, personalizované marketingové kampane.
- **Adobe Experience Manager** - Systém na správu obsahu, ktorý podnikom umožňuje spravovať ich digitálny obsah vrátane webových stránok, mobilných aplikácií a ďalších.

- **Adobe Advertising Cloud** - Platforma na správu a optimalizáciu reklamných kampaní.
- **Adobe Experience Platform** - Platforma na spracovanie zákazníckych údajov, ktorá pomáha podnikom zjednotiť profily zákazníkov a vytvoriť kompletný prehľad o svojich zákazníkoch.

Na rozdiel od tradičných systémov CRM sa však Adobe Experience Cloud nezameriava výlučne na riadenie interakcií so zákazníkmi s predajným oddelením, marketingovým oddelením a zákazníckym servisom. Namiesto toho je navrhnutý tak, aby poskytoval komplexný súbor nástrojov a služieb digitálneho marketingu, ktoré pomáhajú podnikom vytvárať a spravovať skúsenosti so zákazníkmi vo všetkých kontaktných bodoch, od počiatočnej fázy informovanosti a akvizície až po udržanie a lojalitu. Celkovo avšak možno povedať, že hoci Adobe Experience Cloud poskytuje mnoho funkcií a možností, ktoré tradičné CRM neposkytujú, možno ho do tejto kategórie zaradiť. [19]



Obrázok 9 Ukážka užívateľského prostredia systému Adobe Experience Cloud [20]

1.5.1 Dátový model Adobe Experience Cloud

Dátový model systému Adobe Experience Cloud je navrhnutý tak, aby organizáciám pomáhal získavať, ukladať a analyzovať údaje o zákazníkoch v rôznych kontaktných bodoch, kanáloch a zariadeniach. Ide o flexibilný a škálovateľný model, ktorý sa dokáže prispôsobiť

jedinečným potrebám rôznych podnikov a odvetví. Pozostáva z troch hlavných zložiek: zdrojov údajov, súborov údajov a dátových prvkov.

- **Zdroje údajov** - Zdroje údajov sú východiskovým bodom na získavanie údajov o zákazníkoch. Môže to byť akýkoľvek systém, platforma alebo aplikácia, ktorá generuje údaje o zákazníkoch. Medzi zdroje údajov v službe Adobe Experience Cloud patria napríklad webové stránky, mobilné aplikácie, platformy sociálnych médií a ďalšie.
- **Dátové súbory** - Súbory údajov sú zbierky dátových prvkov, ktoré sú usporiadané do logických skupín. Predstavujú konkrétne obchodné subjekty alebo interakcie so zákazníkmi. Príklady dátových súborov v Adobe Experience Cloud zahŕňajú profily zákazníkov, interakcie s webovými stránkami, nákupy produktov a údaje o marketingových kampaniach.
- **Dátové prvky -** Dátové prvky sú jednotlivé časti údajov, ktoré tvoria súbor údajov. Môžu to byť atribúty, udalosti alebo metriky. Príklady dátových prvkov v službe Adobe Experience Cloud: meno zákazníka, e-mailová adresa, výška nákupu, kategória produktu a ďalšie.

Dátový model služby Adobe Experience Cloud je navrhnutý tak, aby bol rozšíriteľný a prispôsobiteľný, čo podnikom umožňuje pridávať vlastné zdroje údajov, dátové súbory a dátové prvky podľa potreby. Podporuje tiež možnosti integrácie a transformácie údajov, vďaka čomu je možné ľahko kombinovať údaje z rôznych zdrojov a normalizovať ich do konzistentného formátu na účely analýzy a reportovania. [19]

1.5.2 Možnosti vývoja aplikácií v Adobe Experience Cloud

Špecifické aplikácie na mieru v službe Adobe Experience Cloud možno vyvíjať v niekoľkých programovacích jazykoch v závislosti od konkrétnych požiadaviek klienta. Tu sú niektoré z najčastejšie používaných programovacích jazykov v systéme Adobe Experience Cloud:

- **JavaScript** - JavaScript sa v systéme Adobe Experience Cloud široko používa na vytváranie dynamických webových aplikácií a používateľských rozhraní. Používa sa aj na vytváranie skriptov, ktoré môžu automatizovať rôzne úlohy v rámci platformy.

- **Java** - Java sa v systéme Adobe Experience Cloud používa napríklad na vytváranie škálovateľných a robustných podnikových aplikácií, konkrétnejšie napríklad na vytváranie špecifických integrácií na mieru s externými systémami.
- **C#** - V službe Adobe Experience Cloud možno použiť jazyk C# na tvorbu špecifických desktopových aplikácií a integrácií s technológiami spoločnosti Microsoft.

Okrem týchto programovacích jazykov podporuje služba Adobe Experience Cloud aj ďalšie jazyky, napríklad Ruby, Go a Swift. Výber programovacieho jazyka závisí od konkrétnych požiadaviek na špecifickú aplikáciu na mieru a od zručností a odborných znalostí vývojového tímu. [19]

Predpokladajme, že klient chce vytvoriť jednoduchú aplikáciu, ktorá bude posielať údaje do systému Adobe Experience Cloud. Nižšie, na obrázku č. 10, sa nachádza príklad kódu v jazyku Java Script, ktorý by sa mohol použiť na vytvorenie tejto aplikácie.

```
function captureCustomerData() {
  // Get the customer data from the website (e.g., user ID, page views, product purchases, etc.)
  const userData = {
    userId: '12345',
    pageViews: 5,
    purchases: [
      {
        productId: 'P001',
        quantity: 2,
        price: 19.99
      },
      {
        productId: 'P002',
        quantity: 1,
        price: 29.99
      }
    ]
  };

  // Send the customer data to Adobe Experience Cloud using the Adobe Analytics send() method
  const adobeAnalytics = window.adobe && window.adobe.analytics;
  if (adobeAnalytics) {
    adobeAnalytics.send({
      hitType: 'event',
      category: 'Customer Data',
      action: 'Capture',
      data: userData
    });
  }
}

// Call the captureCustomerData function when the website loads or when the user performs a specific action
window.addEventListener('load', captureCustomerData);
```

Obrázok 10 Ukážka použitia jazyka Java Script v systéme Adobe Experience Cloud

Na začiatku kódu sa definuje objekt s názvom userData, ktorý obsahuje vzorové údaje o zákazníkovi. Tieto údaje obsahujú userId, pageViews a pole nákupov, ktoré obsahuje informácie o každom nákupe, napríklad productId, množstvo a cenu. Následne prebehne kontrola, či je načítaná knižnica Adobe Analytics, a to tak, že sa skontroluje prítomnosť objektu window.adobe.analytics. Ak je knižnica načítaná, zavolá sa metóda send() knižnice Adobe Analytics a odovzdá objekt, ktorý špecifikuje typ zásahu (hitType), kategóriu pre údaje (category), akciu, ktorá sa má vykonať (action), a objekt userData. Atribút hitType sa vzťahuje na typ požiadavky na sledovanie odoslanej na server Adobe Analytics. Je to povinné pole, ktoré určuje typ údajov zhromažďovaných v aktuálnej požiadavke na sledovanie. Medzi ďalšie možné hodnoty pre hitType v službe Adobe Analytics patria "pageview", používa sa na sledovanie zobrazení stránok, a "transaction", ktorá sa používa na sledovanie transakcií. Zadaním atribútu hitType v požiadavke na sledovanie môže služba Adobe Analytics správne kategorizovať a spracovať odosielané údaje, čo pomáha zabezpečiť presné a použiteľné analytické údaje. Nakoniec sa funkcia nastaví tak, aby sa spustila po dokončení načítania objektu okna pridaním tzv. poslucháča udalostí, ktorý „počúva“ udalosť "load" a vykoná funkciu captureCustomerData(), keď sa táto udalosť spustí.

2 SALESFORCE CRM

Salesforce CRM je cloudová softvérová platforma navrhnutá tak, aby pomáhala podnikom riadiť interakcie so zákazníkmi, predaj, marketing, zákaznícky servis a analytiku. Ide o komplexné riešenie, ktoré poskytuje centralizovaný prehľad o údajoch zákazníkov a umožňuje podnikom prijímať rozhodnutia na základe týchto údajov, ktoré zlepšujú vzťahy so zákazníkmi, podporujú rast príjmov a zefektívňujú obchodné procesy.

Jednou zo základných funkcií systému Salesforce CRM sú jeho možnosti riadenia predaja. Poskytuje množstvo nástrojov na riadenie celého procesu predaja, od správy potenciálnych zákazníkov a správu obchodných príležitostí až po prognózovanie a správu obchodných spoluprác. Nástroje platformy na správu potenciálnych zákazníkov umožňujú podnikom zachytávať a klasifikovať potenciálnych zákazníkov, zatiaľ čo nástroje na správu obchodných príležitostí umožňujú obchodným tímom efektívnejšie riadiť predajné procesy a uzatvárať obchodné dohody. Salesforce CRM poskytuje aj funkcie, ako je analýza výkonnosti predaja a prognózovanie predaja, ktoré poskytujú podnikom prehľad o ich výkonnosti predaja a pomáhajú im prijímať rozhodnutia o budúcich predajných aktivitách. [21]

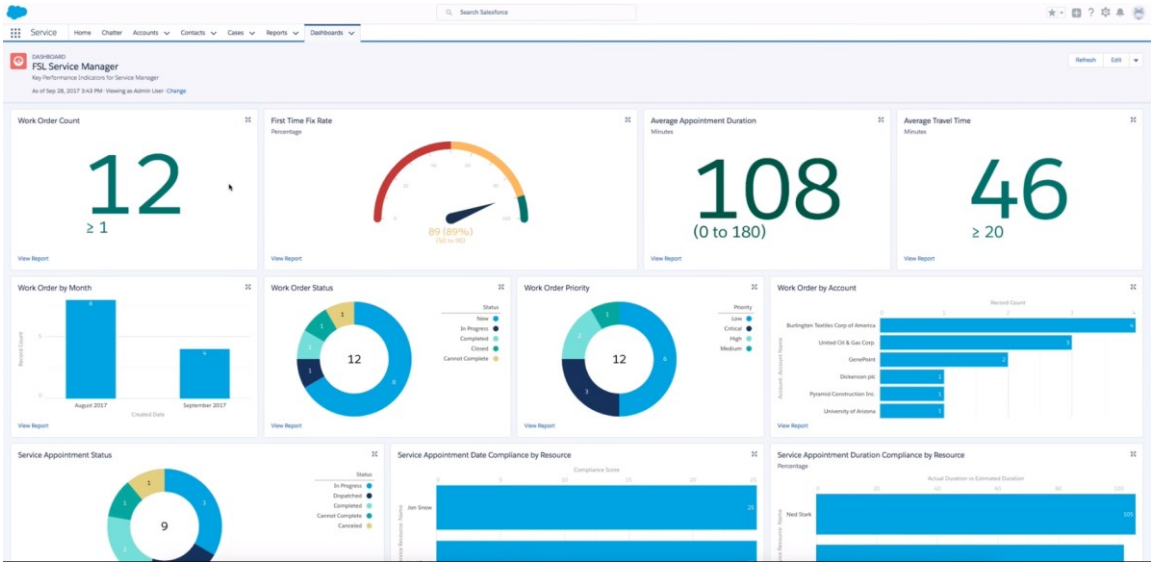
Ďalšou kľúčovou funkciou systému Salesforce CRM sú jeho možnosti automatizácie marketingu. Patria sem nástroje na e-mailový marketing, správu sociálnych médií a marketingovú analýzu. Pomocou Salesforce CRM môžu podniky automatizovať a optimalizovať svoje marketingové kampane, sledovať správanie a preferencie zákazníkov a poskytovať personalizovaný marketingový obsah s cieľom zlepšiť angažovanosť zákazníkov.

Salesforce CRM poskytuje aj celý rad foriem zákazníckeho servisu vrátane správy servisných požiadaviek a samoobslužných portálov. Pomocou týchto nástrojov môžu podniky efektívnejšie spravovať otázky a problémy zákazníkov, poskytovať zákazníkom možnosti samoobsluhy a monitorovať spokojnosť zákazníkov s cieľom identifikovať oblasti, ktoré je potrebné zlepšiť. Salesforce CRM obsahuje aj funkcie umožňujúce spoluprácu, ako sú zdieľané kalendáre, zoznamy úloh a nástroje na riadenie projektov, ktoré umožňujú členom tímu efektívnejšie spolupracovať. [21]

Analytické funkcie systému Salesforce CRM poskytujú podnikom množstvo poznatkov o zákazníkoch a ich aktivitách na základe dát. Platforma poskytuje informačné panely v reálnom čase a prispôsobiteľné reporty, ktoré podnikom umožňujú sledovať kľúčové ukazovatele výkonnosti a sledovať pokrok pri dosahovaní obchodných cieľov. Salesforce CRM

obsahuje aj nástroje prediktívnej analýzy, ktoré umožňujú podnikom identifikovať trendy a vzorce správania zákazníkov a predvídať ich budúce potreby a preferencie.

Vďaka možnostiam mobilného prístupu je Salesforce CRM dostupný odkiaľkoľvek, čo umožňuje členom tímu pristupovať k údajom o zákazníkoch, spolupracovať na projektoch a spravovať svoje úlohy z mobilných zariadení. Možnosti prispôsobenia systému Salesforce umožňujú podnikom prispôsobiť systém svojim špecifickým potrebám, napr. vytvárať vlastné pracovné postupy a tiež integrácie s inými podnikovými systémami.



Obrázok 11 Ukážka užívateľského prostredia systému Salesforce [22]

2.1 História

Spoločnosť Salesforce bola založená vo februári 1999 v San Franciscu s jasnou víziou byť internetovou spoločnosťou svetovej triedy. Zakladateľmi spoločnosti boli Marc Benioff, Parker Harris, Frank Dominguez a Dave Moellenhoff.

Salesforce ihneď niekoľko mesiacov po založení predstavil svoj plán V2MOM (Vision, Values, Methods, Obstacles, and Measures, preklade Vízia, Hodnoty, Metódy, Prekážky a Opatrenia), ktorého cieľom bolo poskytnúť zamestnancom jasnú víziu a zosúladiť celú spoločnosť okolo spoločných cieľov. V2MOM zostáva aj v súčasnosti jadrom toho, ako spoločnosť Salesforce riadi svoje podnikanie, a tiež naďalej usmerňuje každé rozhodnutie, ktoré spoločnosť prijíma. Na konci prvého roka existencie sa spoločnosť rozrástla na 40 zamestnancov.

V marci 2000 spoločnosť Salesforce získava väčšiu popularitu vďaka prvej významnejšej správe v tlači v denníku The Wall Street Journal, keď si prenajala hercov, aby zinscenovali fingovaný protest pred sídlom konkurenčnej spoločnosti Siebel Systems. "Protestujúci" niesli transparenty s antisoftvérovými heslami, aby zdôraznili marketingový slogan spoločnosti Salesforce "Koniec softvéru". V apríli 2001 spoločnosť Salesforce oznámila svoju expanziu na celosvetový trh so sídlom v Dubline a Tokiu. Spoločnosť Salesforce v roku 2001 prekročila hranicu 3 000 zákazníkov, čím sa stala najrýchlejšie rastúcou spoločnosťou v oblasti CRM. V novembri 2001 je Marc Benioff vymenovaný za výkonného riaditeľa a predsedu predstavenstva Salesforce. Príjmy Salesforce za rok 2001 dosiahli 5,4 milióna dolárov, za 2002 už 51 miliónov USD. Do konca roka 2002 mala spoločnosť už 5 740 zákazníkov, 70 000 používateľov v 107 krajinách, ktorí mali prístup k službe v ôsmich jazykoch. Spoločnosť v roku 2003 zamestnávala viac ako 400 zamestnancov, začala pôsobiť na celom svete a otvorila pobočky v Austrálii, Francúzsku, Nemecku, Írsku, Japonsku, Španielsku a Veľkej Británii. Salesforce v roku 2003 dosiahla tržby takmer 100 miliónov dolárov a v decembri toho istého roka vstúpila na burzu. [23]

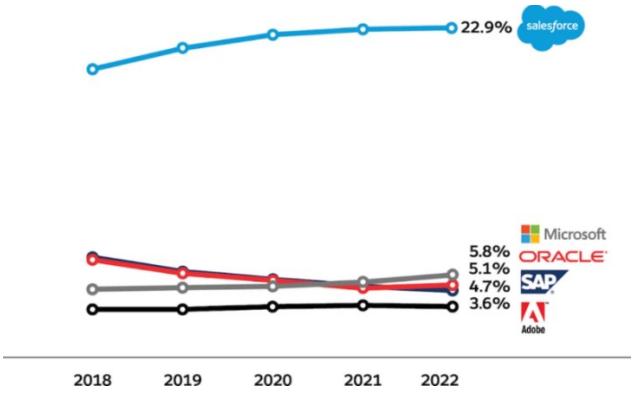
V septembri 2005 Salesforce predstavila službu AppExchange, ktorá ponúka vývojárom tretích strán miesto, kde môžu pracovať na svojich špecifických aplikáciách na mieru a sprístupniť ich ostatným zákazníkom Salesforce. O rok neskôr sa ekosystém AppExchange rozrástol do dynamického trhu s 575 aplikáciami od 250 nezávislých dodávateľov softvéru. Salesforce v roku 2006 dosiahla tržby takmer 497 miliónov dolárov, v roku 2008 to už bola 1 miliarda dolárov, čím sa Salesforce stala prvou spoločnosťou v oblasti cloud computingu, ktorej ročné tržby prekročili hranicu miliardy dolárov. V tom istom roku časopis Forbes zaradil spoločnosť Salesforce medzi najrýchlejšie rastúce technologické spoločnosti na svete.

V roku 2011 Salesforce spúšťa Chatter ako službu sociálnej spolupráce pre podniky; v prvom roku si Chatter osvojí viac ako 80 000 zákazníkov. Spoločnosť Salesforce dosiahla v roku 2012 ročné príjmy vo výške viac ako 3 miliardy amerických dolárov a zamestnávala 9 800 zamestnancov v 22 pobočkách po celom svete. V roku 2014 spoločnosť Salesforce spustila službu Trailhead, ktorá umožňuje každému bez ohľadu na úroveň vzdelania rozvíjať zručnosti potrebné pre prácu v systéme Salesforce. V roku 2015 Salesforce vstupuje do rebríčka Fortune 500. [23]

V roku 2016 časopis Forbes označil generálneho riaditeľa Salesforce Marca Benioffa za "najlepšieho inovátora desaťročia". O rok neskôr, spoločnosť Salesforce uvádza na trh produkt Einstein, prvú komplexnú technológiu umelej inteligencie (AI) pre CRM, ktorá

sprístupňuje AI každému používateľovi systému Salesforce. V roku 2019 spoločnosť Salesforce dosiahla tržby viac ako 17 miliárd amerických dolárov. [23]

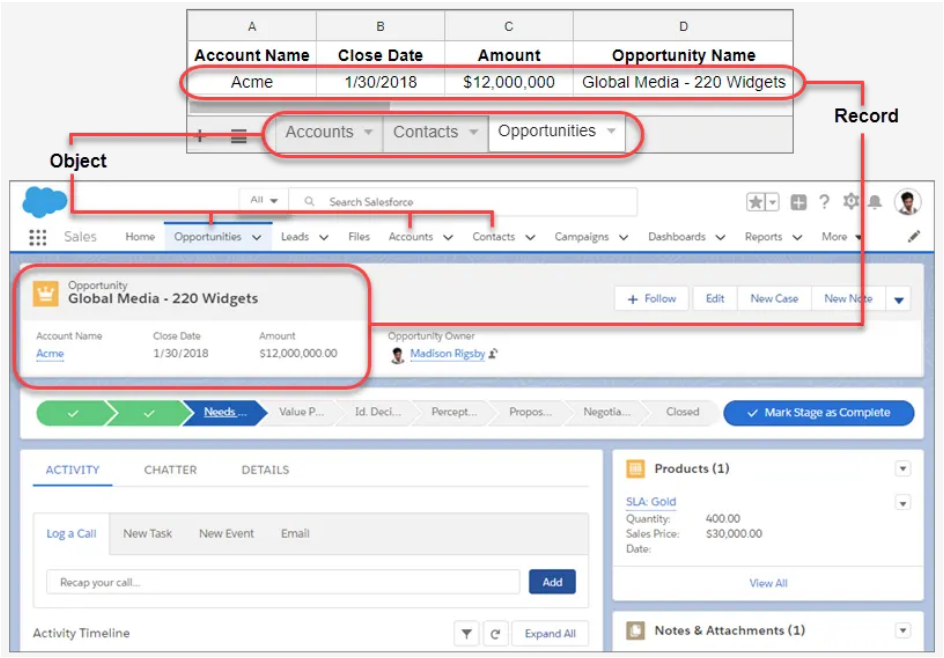
V roku 2022 sa spoločnosť Salesforce umiestnila na prvom mieste v rebríčku CRM systémov už 9. rok po sebe. Spoločnosť Salesforce sa taktiež aktívne angažuje v charitatívnej činnosti a tiež v oblasti enviromentalistiky. Nižšie, na obrázku č. 12 je znázornený vývoj percentuálneho podielu v posledných 4 rokoch najpoužívanejších CRM systémov na trhu CRM systémov.



Obrázok 12 Podiel najpoužívanejších CRM systémov na trhu CRM systémov [24]

2.2 Dátový model

Salesforce organizuje všetky údaje do objektov a záznamov. Objekty je možné si predstaviť ako tabuľku v databáze a záznam ako jeden riadok tabuľky.



Obrázok 13 Ukážka dátového modelu Salesforce [25]

K objektom je možné pristupovať z navigačného panela. Výberom ľubovoľného záznamu sa používateľ dostane do detailnejšieho zobrazenia konkrétneho účtu, kontaktu, príležitosti alebo akéhokoľvek iného záznamu v Salesforce. Medzi základné pojmy potrebné pre orientáciu v dátovom modeli Salesforce patria:

- **Záznam** - Položka, ktorú používateľ vytvára a s ktorou pracuje; ak sa o dátach v Salesforce uvažuje ako o relačnej databáze, potom je záznam riadkom v tabuľke. Každý záznam má jedinečný identifikátor známy ako ID záznamu, ktorý sa používa na identifikáciu a odkazovanie na záznam v systéme.
- **Pole** - Miesto, kde je uchovávaná hodnota, napríklad meno alebo adresa; ak sa o dátach v Salesforce uvažuje ako o relačnej databáze, pole reprezentuje stĺpec v tabuľke.
- **Objekt** - Ak sa o dátach v Salesforce uvažuje ako o relačnej databáze, objekt reprezentuje tabuľku v databáze.
- **Org** - Skratka pre "organizáciu", miesto, kde sa nachádzajú všetky dáta a konfiguračné súbory. Používatelia sa do nej prihlasujú a niekedy sa tiež nazýva aj "inštancia Salesforce".
- **Aplikácia** - Súbor polí, objektov, oprávnení a funkcií na podporu obchodného procesu

Záznamy sú celkovo základnou zložkou dátového modelu Salesforce, pretože umožňujú ukladať, organizovať a spravovať informácie o reálnych entitách v systéme. Záznamy v systéme Salesforce môžu vytvárať, upravovať a odstraňovať používatelia, ktorí majú príslušné oprávnenia. Tieto bezpečnostné oprávnenia pomáhajú zabezpečiť, aby boli citlivé údaje v bezpečí a prístupné naozaj len tým používateľom, ktorí by ich mali byť schopní vidieť. [26]

Je možné povoliť konkrétnym používateľom zobrazenie objektu, ale potom obmedziť jednotlivé záznamy objektu, ktoré môžu vidieť. Napríklad anketár môže vidieť a upravovať svoje vlastné recenzie, ale nie recenzie iných anketárov. Prístup na úrovni záznamov je možné spravovať týmito štyrmi spôsobmi:

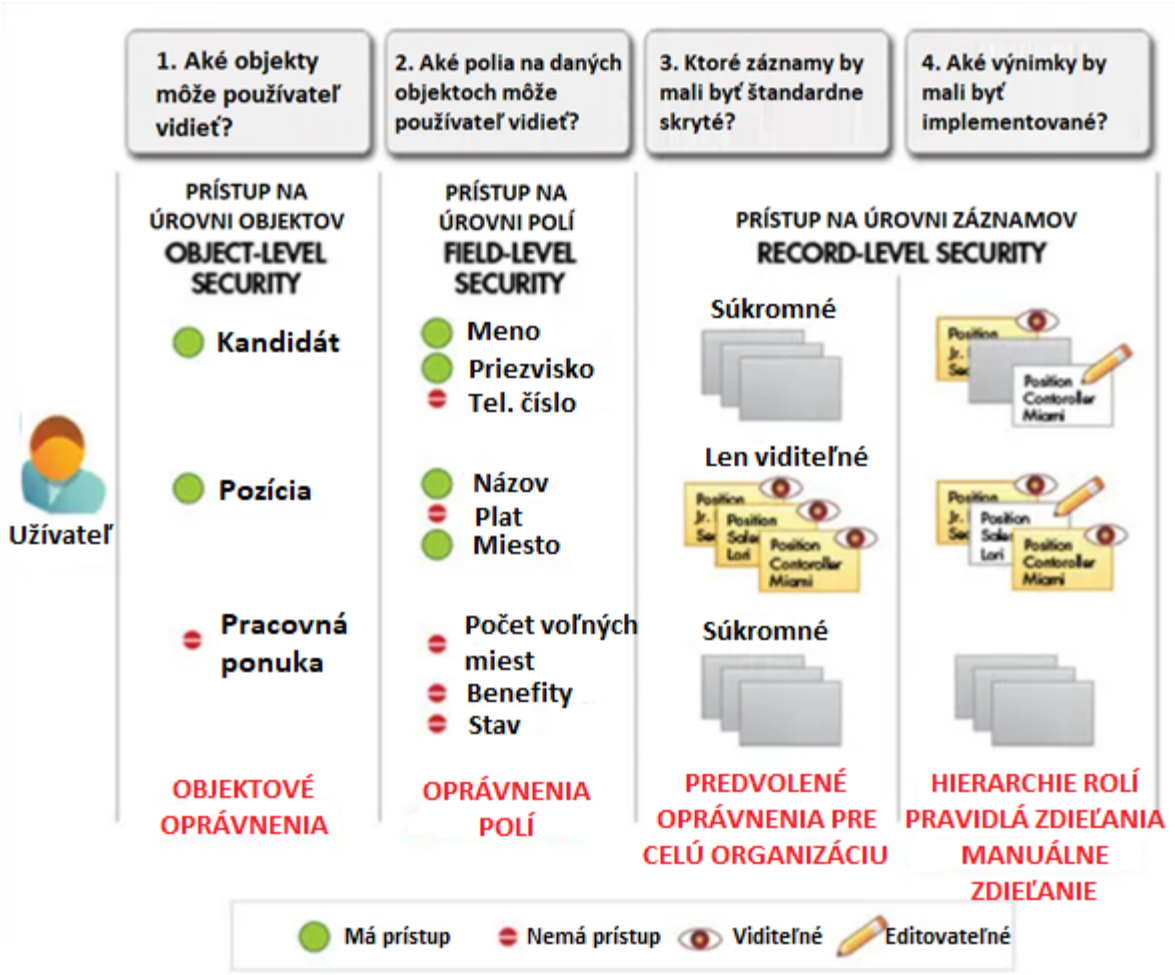
- **Predvolené nastavenia pre celú organizáciu (Organization-wide defaults)** - Určujú predvolenú úroveň prístupu používateľov k záznamom ostatných používateľov. Pomocou nastavenia zdieľania na úrovni celej organizácie je možné uzamknúť údaje na najprísnejšiu úroveň (takže každý používateľ bude mať prístup iba k záznamom, ktoré vytvoril on sám) a potom pomocou ostatných nástrojov zabezpečenia a zdieľania na úrovni záznamov selektívne poskytnúť prístup ostatným používateľom.

- **Hierarchie rolí** - Poskytujú používateľom, ktorí sú v hierarchii vyššie, prístup ku všetkým záznamom, ktoré vytvorili používatelia, ktorí sú v hierarchii pod nimi. Každá rola v hierarchii by mala predstavovať úroveň prístupu k údajom, ktorú používateľ alebo skupina používateľov potrebuje.
- **Pravidlá zdieľania (Sharing rules)** - Sú automatické výnimky z predvolených nastavení pre celú organizáciu pre konkrétne skupiny používateľov, aby sa mohli dostať k záznamom, ktoré nevlastnia alebo ktoré bežne nevidia. Pravidlá zdieľania, podobne ako hierarchie rolí, sa používajú len na to, aby ďalší používatelia mali prístup k záznamom.
- **Manuálne zdieľanie** - Umožňuje vlastníkom konkrétnych záznamov zdieľať ich s inými používateľmi. Hoci manuálne zdieľanie nie je automatizované ako celoorganizačné nastavenia zdieľania, hierarchie rolí alebo pravidlá zdieľania, môže byť užitočné v niektorých situáciách, napríklad keď náborový pracovník, ktorý odchádza na dovolenku, potrebuje dočasne prideliť vlastníctvo žiadosti o zamestnanie niekomu inému.

Prístup k údajom na úrovni objektov je kontrolovaný nastavením oprávnení na konkrétny typ objektu. Týmito oprávneniami je možné skupine používateľov zabrániť vytvárať, zobrazovať, upravovať alebo odstraňovať akékoľvek záznamy tohto objektu. Pomocou objektových oprávnení je možné napríklad zabezpečiť, aby si účastníci pohovorov mohli pozície a žiadosti o zamestnanie prezerať, ale nie upravovať alebo vymazávať. Dva dôležité nástroje používané na riadenie prístupu k objektom v systéme Salesforce sú profily a sady oprávnení. Profily aj sady oprávnení sa používajú na riadenie prístupu používateľov a oprávnení v Salesforce, ale majú odlišné úlohy. Profily sa používajú na určenie základnej úrovne prístupu a oprávnení pre používateľa, zatiaľ čo sady oprávnení sa používajú na udelenie ďalších oprávnení nad rámec tejto základnej úrovne. [26] [27] [28]

Prístup k určitým poliam je možné obmedziť, aj keď má používateľ prístup k objektu. Napríklad je možné nastaviť, aby pole plat v objekte pozície bolo neviditeľné pre uchádzačov o zamestnanie, ale viditeľné pre náborových manažérov a personalistov.

Prehľad riadenia prístupu k údajom v systéme Salesforce je znázornený na obrázku nižšie.



Obrázok 14 Prehľad riadenia prístupu k údajom v systéme Salesforce

Objekty v systéme Salesforce môžu byť buď štandardné alebo špecifické, na mieru. Štandardné objekty poskytuje Salesforce v základnej licencii a sú predpripravené na podporu bežných obchodných procesov, zatiaľ čo špecifické objekty na mieru vytvárajú používatelia, aby spĺňali ich jedinečné obchodné potreby. Medzi najpoužívanejšie štandardné objekty patria:

- **Účty** - Objekt účet ukladá informácie o spoločnosti alebo organizácii vrátane jej názvu, adresy a kontaktných informácií.
- **Kontakty** - Ukladajú informácie o jednotlivcoch spojených s účtom vrátane ich mena, e-mailu, telefónneho čísla a názvu pracovnej pozície.
- **Príležitosti** - Ukladá informácie o potenciálnych obchodných transakciách vrátane sumy transakcie, očakávaného dátumu uzavretia a štádia transakcie.
- **Príkazy** - V objekte príkaz sa ukladajú informácie o potenciálnych zákazníkoch, ktorí prejavili záujem o výrobok alebo službu, ale ešte sa nestali zákazníkmi.

- **Prípady** - Ukladá informácie o požiadavkách alebo problémoch zákazníkov vrátane mena zákazníka, kontaktných informácií a podrobností o prípade.

Polia sa v systéme Salesforce používajú na ukladanie konkrétnych údajov v rámci objektu. Každý objekt má sadu polí, ktoré zodpovedajú typu uložených údajov. Polia môžu byť rovnako ako aj objekty buď štandardné alebo špecifické polia na mieru, vytvorené používateľmi na splnenie ich špecifických obchodných potrieb. [26] [29]

Polia v Salesforce môžu byť rôznych dátových typov. Najrozšírenejšie dátové typy sú:

- **Text** - Používa sa na ukladanie textových údajov, ako sú mená, adresy a popisy.
- **Číslo -** Používa sa na ukladanie číselných údajov, ako sú sumy, množstvá a percentá.
- **Dátum a čas** - Používa sa na ukladanie údajov o dátume a čase, ako je dátum vytvorenia záznamu alebo dátum splatnosti faktúry.
- **Výberový zoznam (Picklist)** - Používa sa na ukladanie preddefinovaných hodnôt, ktoré môžu používatelia vyberať zo zoznamu. Pole výberového zoznamu sa môže použiť napríklad na zaznamenanie odvetvia vedúceho pracovníka alebo priority prípadu.
- **Vyhľadávanie (Lookup)** - Slúži na prepojenie jedného záznamu s iným záznamom. Napríklad vyhľadávacie pole sa môže použiť na prepojenie kontaktu s účtom.
- **Vzorec (Formula)** - Používa sa na vykonávanie výpočtov alebo kombinovanie údajov z iných polí v zázname. Polia vzorcov sú určené len na čítanie a používatelia ich nemôžu upravovať.
- **Zaškrtávacie políčko (Checkbox)** - Používa sa na ukladanie hodnôt true/false. Pole typu checkbox sa môže použiť napríklad na označenie toho, či bol vedúci kontaktovaný alebo nie.

Na prepojenie rôznych objektov v systéme Salesforce sa používajú tzv. vzťahy. V Salesforce existujú dva hlavné typy vzťahov: vzťahy vyhľadávania (Lookup relationships) a vzťahy master-detail (Master-detail relationships). [26]

Vzťahy vyhľadávania sú typom vzťahov, pri ktorých existuje voľné spojenie medzi dvoma objektmi. Pomocou vzťahu vyhľadávania je možné prepojiť dva objekty vytvorením poľa v jednom objekte, ktoré odkazuje na záznam v inom objekte. Napríklad vzťah lookup medzi objektmi účet a kontakt by umožnil prepojiť záznam kontakt so súvisiacim záznamom účet.

Vzťah master-detail je pevnejšie prepojený vzťah medzi dvoma objektmi ako je vzťah lookup. Pri vzťahu master-detail je jeden objekt označený ako "master" a druhý objekt je označený ako "detail". Objekt "detail" je závislý od objektu "master" a nemôže existovať bez neho. Napríklad vzťah "master-detail" medzi objektmi príležitosť a položka príležitosti by umožnil priradiť k jednému záznamu o príležitosti viacero záznamov o položke príležitosti. [30]

Medzi kľúčové rozdiely medzi vzťahmi lookup a master-detail patria:

- Vzťahy typu lookup možno vytvoriť medzi ľubovoľnými dvoma objektmi, zatiaľ čo vzťahy typu master-detail možno vytvoriť len medzi určitými typmi objektov.
- Pri vzťahu master-detail zdedí objekt detailu určité vlastnosti od master objektu, napríklad nastavenia zdieľania a správanie pri vymazávaní.
- Pomocou vzťahu master-detail je možné vytvárať na master zázname aj tzv. súhrnné polia na sumarizáciu údajov z detailných záznamov, napríklad pole hmotnosť na detailnom zázname je možné sumarizovať na master zázname a získať priemernú hmotnosť všetkých detail záznamov.

Okrem týchto dvoch hlavných typov vzťahov existujú v Salesforce aj niektoré špeciálne typy vzťahov, napríklad vzťahy typu mnoho k mnohým (many-to-many) a hierarchické vzťahy. Vzťahy typu many-to-many umožňujú priradiť viaceré záznamy z jedného objektu k viacerým záznamom z iného objektu. Hierarchické vzťahy sa používajú na vytvorenie vzťahov rodič-dieťa medzi záznamami v rámci objektu používateľ (User), napríklad medzi manažérom a jeho priamymi podriadenými. [26] [30]

Celkovo sú vzťahy výkonným nástrojom v Salesforce, ktorý umožňuje vytvárať komplexné dátové modely a spájať súvisiace údaje vo viacerých objektoch. Pochopenie toho, ako efektívne vytvárať a používať vzťahy, je nevyhnutné na vytváranie škálovateľných a efektívnych riešení v systéme Salesforce.

2.3 Možnosti vývoja aplikácií v Salesforce

Možnosť veľmi detailne prispôsobiť systém Salesforce konkrétnym potrebám klienta je hlavnou výhodou, ktorá ho odlišuje od iných CRM systémov. Aby bolo možné dosiahnuť toto detailné prispôsobenie, Salesforce poskytuje dve hlavné možnosti vývoja špecifických aplikácií na mieru: deklaratívnu možnosť a programovaciu.

Deklaratívny vývoj zahŕňa využívanie rozhrania Salesforce typu "ukáž a klikni" (point and click) na konfiguráciu a prispôsobenie platformy konkrétnym obchodným potrebám. Táto metóda si nevyžaduje programovanie ani programátorské zručnosti a často sa označuje ako "klikanie, nie programovanie". Pomocou deklaratívneho vývoja môžu používatelia vytvárať špecifické objekty na mieru, polia, reporty, informačné panely (dashboards) a pod. Deklaratívny vývoj je zvyčajne rýchlejší a používateľsky prívetivejší ako programovací vývoj, ale má určité obmedzenia z hľadiska prispôsobenia a funkčnosti.

Programovací vývoj zahŕňa písanie kódu na vytvorenie zložitejších a viac špecifických aplikácií na mieru v rámci platformy Salesforce. Tento prístup si vyžaduje programátorské zručnosti, napríklad znalosti jazykov Apex, SOQL, Visualforce a Lightning component framework jazyku JavaScript. Programatický vývoj umožňuje väčšiu flexibilitu a prispôsobenie, ale vyžaduje si aj viac času, zdrojov a odborných znalostí.

Deklaratívny vývoj je teda rýchlejší a jednoduchší spôsob vytvárania jednoduchých aplikácií, ktoré nevyžadujú zložité prispôsobenie alebo pokročilé funkcie, zatiaľ čo programovací vývoj je účinnejší spôsob vytvárania zložitých a špecifických aplikácií na mieru, ktoré vyžadujú špecifickú obchodnú logiku alebo integráciu s externými systémami. [31]

2.3.1 Deklaratívne spôsoby vývoja aplikácií

Salesforce poskytuje celý rad deklaratívnych vývojových nástrojov, ktoré umožňujú používateľom vytvárať špecifické aplikácie na mieru bez potreby písania kódu. Medzi najpoužívanejšie deklaratívne vývojové nástroje patria:

- Rozloženie stránky - Page Layout
- Typ záznamu - Record Type
- Lightning App Builder
- Process Builder
- Flow

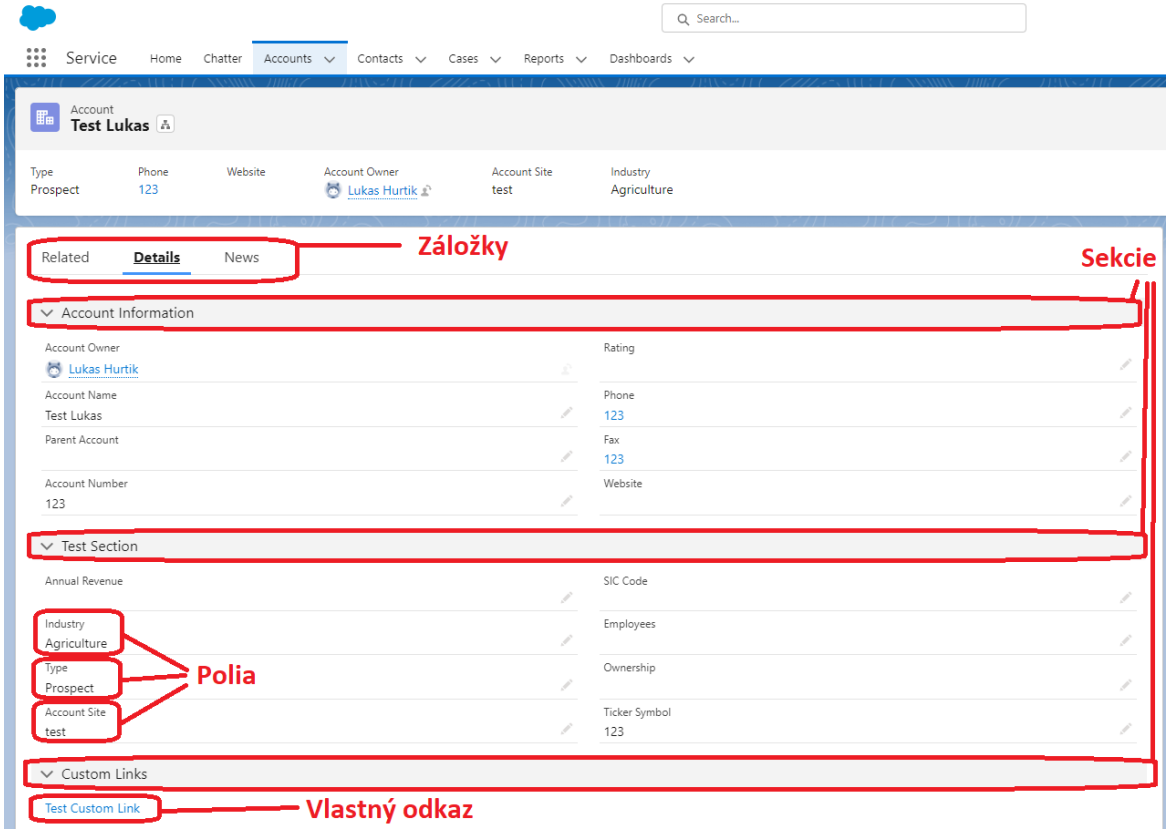
2.3.1.1 Rozloženie stránky

V Salesforce je rozloženie stránky usporiadaním polí, súvisiacich zoznamov a vlastných odkazov na stránke s podrobnosťami záznamu. Určuje, aké informácie sa zobrazia používateľom pri zobrazení, úprave alebo vytvorení záznamu. Rozloženie stránky možno prispôsobiť tak, aby vyhovovalo konkrétnym obchodným potrebám, a možno ho priradiť rôznym profilom a typom záznamov. Licencia Salesforce sa dodáva so štandardnými rozloženiami

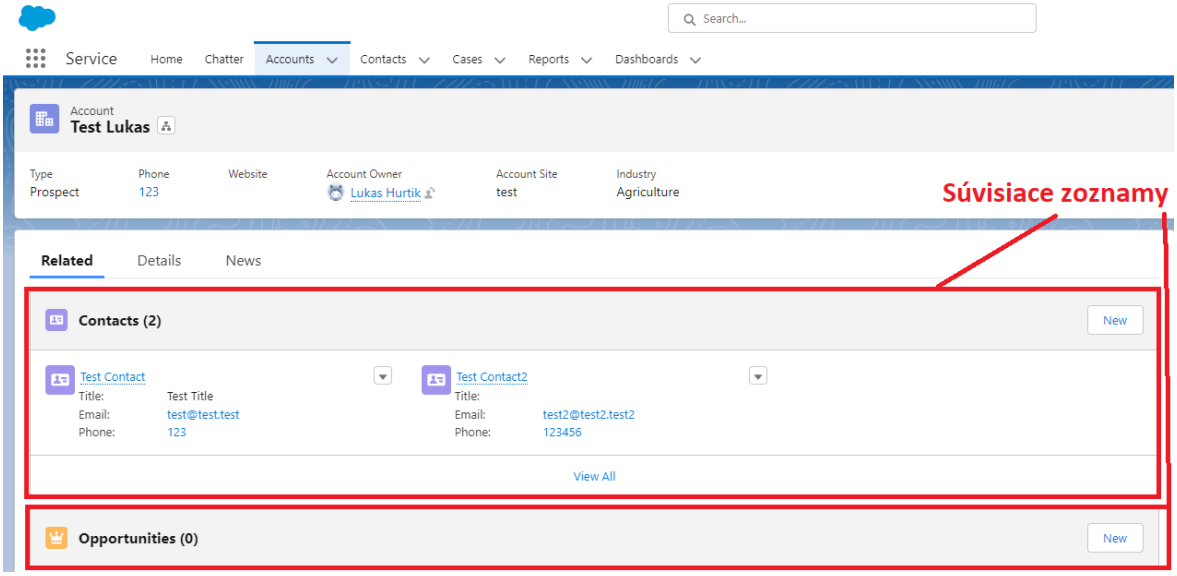
stránok pre rôzne typy objektov, ako sú napríklad Účty, Kontakty, Príležitosti a Prípady. Tieto rozloženia stránok sú vopred nakonfigurované spoločnosťou Salesforce a možno ich používať tak, ako sú, alebo ich prispôsobiť tak, aby spĺňali špecifické obchodné požiadavky. Klienti môžu vytvárať svoje vlastné rozloženia stránok pre rôzne profily používateľov a typy záznamov. Vlastné rozloženia stránok umožňujú podnikom prispôsobiť informácie zobrazované používateľom na základe ich rolí a povinností. [31]

Rozloženie stránky tiež obsahujú:

- **Polia** - Polia zobrazujú informácie o zázname. Klienti môžu pridávať, odstraňovať alebo meniť usporiadanie polí v rozložení stránky tak, aby sa najdôležitejšie informácie zobrazovali ako prvé. Polia možno tiež nastaviť ako polia určené len na čítanie, povinné alebo voliteľné v závislosti od potrieb klienta.
- **Súvisiace zoznamy** - Súvisiace zoznamy sú zbierkou záznamov súvisiacich s hlavným záznamom. V súvisiacom zozname pri zázname účtu sa môžu napríklad zobraziť všetky kontakty súvisiace s týmto účtom. Súvisiace zoznamy možno prispôsobiť tak, aby zobrazovali konkrétne polia, a možno ich pridať alebo odstrániť z rozloženia stránky.
- **Vlastné odkazy -** Vlastné odkazy sú odkazy, ktoré sa pridávajú do rozloženia stránky a môžu sa použiť na prístup k externým webovým lokalitám, dokumentom alebo iným zdrojom.
- **Sekcie** - Sekcie rozdeľujú rozloženie stránky do rôznych sekcií na zoskupenie súvisiacich polí a súvisiacich zoznamov. Sekcie možno zložiť alebo rozbaliť, aby sa zobrazili alebo skryli informácie na základe preferencií používateľa. [32] [31]



Obrázok 15 Ukážka rozloženia stránky



Obrázok 16 Ukážka rozloženia stránky - Súvisiace zoznamy

2.3.1.2 Typ záznamu

Typy záznamov v Salesforce umožňujú definovať rôzne sady hodnôt výberových zoznamov, rozloženia stránok a obchodných procesov, ktoré možno priradiť rôznym skupinám

používateľov alebo profilom. Typy záznamov sa zvyčajne používajú na prispôsobenie správania a vzhľadu štandardných a vlastných objektov v Salesforce.

Každý typ záznamu je spojený so špecifickou sadou rozložení stránky, ktoré určujú polia a súvisiace zoznamy, ktoré sa zobrazujú na stránke s detailom záznamu. To umožňuje vytvárať rôzne rozloženia stránok pre rôzne skupiny používateľov alebo obchodné procesy.

Typy záznamov sa používajú aj na definovanie rôznych obchodných procesov, napríklad fáz predaja alebo stavov prípadov podpory. To umožňuje prispôsobiť aplikáciu rôznym obchodným požiadavkám. Okrem toho možno typy záznamov použiť na riadenie viditeľnosti a prístupu k určitým poliam, súvisiacim zoznamom a akciám. To umožňuje obmedziť určitým používateľom alebo profilom možnosť vidieť alebo upravovať určité údaje.

Pre lepšie pochopenie typov záznamu je v tabuľke č. 1 uvedený príklad. V systéme Salesforce je vytvorený špecifický objekt na mieru s názvom Prípad zákazníckeho servisu. Na základe toho aký typ záznamu používateľ vyberie pri vytváraní záznamu sú v jednotlivých poliach dostupné rôzne hodnoty. [31]

Tabuľka 1 Príklad použitia typov záznamu

		Typy záznamu		
		Návrh	**Problém**	**Otázka**
Pole	**Status**	Nový, V procese, Akceptovaný, Odmietnutý, Dokončený	Nový, V procese, Dokončený	Nový, V procese, Dokončený
	Priorita	Nízka	Nízka, Stredná, Vysoká, Urgentná	Nízka, Stredná
	Požadovaný dátum vyriešenia	Pole je skryté	Pole je viditeľné	Pole je skryté

2.3.1.3 Lightning App Builder

Používateľské rozhranie systému Salesforce zostalo počas prvých 16 rokov jeho fungovania v podstate nezmenené. Aby boli zákazníci aj naďalej spokojní a aby si systém Salesforce aj v budúcnosti zachoval popredné postavenie na trhu CRM systémov, spoločnosť Salesforce si uvedomila, že potrebuje modernizovať svoje rozhranie. V roku 2015 spoločnosť predstavila nové užívateľské prostredie s názvom Lightning experience, skrátene Lightning.

Cieľom implementácie Lightning do Salesforce bolo zlepšiť celkový používateľský zážitok a pomôcť používateľom pracovať efektívnejšie poskytnutím moderného a citlivého rozhrania, ktoré sa ľahko ovláda a používa.

Súčasťou tohto prostredia bol aj nástroj s názvom Lightning App Builder. Lightning App Builder je vizuálne rozhranie, ktoré umožňuje používateľom vytvárať a prispôsobovať stránky Lightning bez potreby písania kódu. Poskytuje funkciu drag-and-drop, ktorá používateľom umožňuje pridávať, odstraňovať a usporadúvať komponenty Lightning na rozložení stránky, čo uľahčuje vytváranie špecifických aplikácií a stránok na mieru. [33] [31]

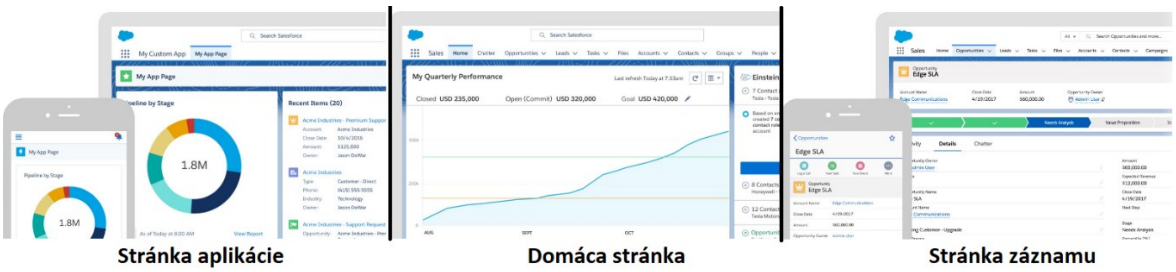
Nižšie sú uvedené niektoré ďalšie kľúčové aspekty nástroja Lightning App Builder v Salesforce:

- **Lightning Pages** - Lightning App Builder umožňuje používateľom vytvárať a prispôsobovať stránky Lightning, ktoré sú stavebnými prvkami pre špecifické aplikácie na mieru v systéme Salesforce. Lightning Pages sa skladajú z komponentov Lightning, ktoré môžu byť buď štandardné alebo špecifické na mieru.
- **Predpripravené komponenty -** Lightning App Builder sa dodáva s knižnicou vopred pripravených komponentov Lightning, ktoré možno pridať do rozloženia stránky, vrátane grafov, tabuliek, formulárov a ovládacích panelov. Tieto komponenty je možné prispôsobiť tak, aby spĺňali špecifické obchodné potreby.
- **Špecifické komponenty na mieru** - Podniky si môžu vytvoriť aj vlastné komponenty Lightning, ktoré vykonávajú špecifické funkcie, a pridať ich na stránku Lightning pomocou nástroja Lightning App Builder.
- **Šablóny stránok** - Lightning App Builder poskytuje vopred pripravené šablóny stránok, ktoré možno použiť na rýchle vytvorenie vlastných stránok. Tieto šablóny obsahujú predpripravené komponenty usporiadané v špecifickom rozložení, čo uľahčuje rýchle vytvorenie funkčnej stránky.

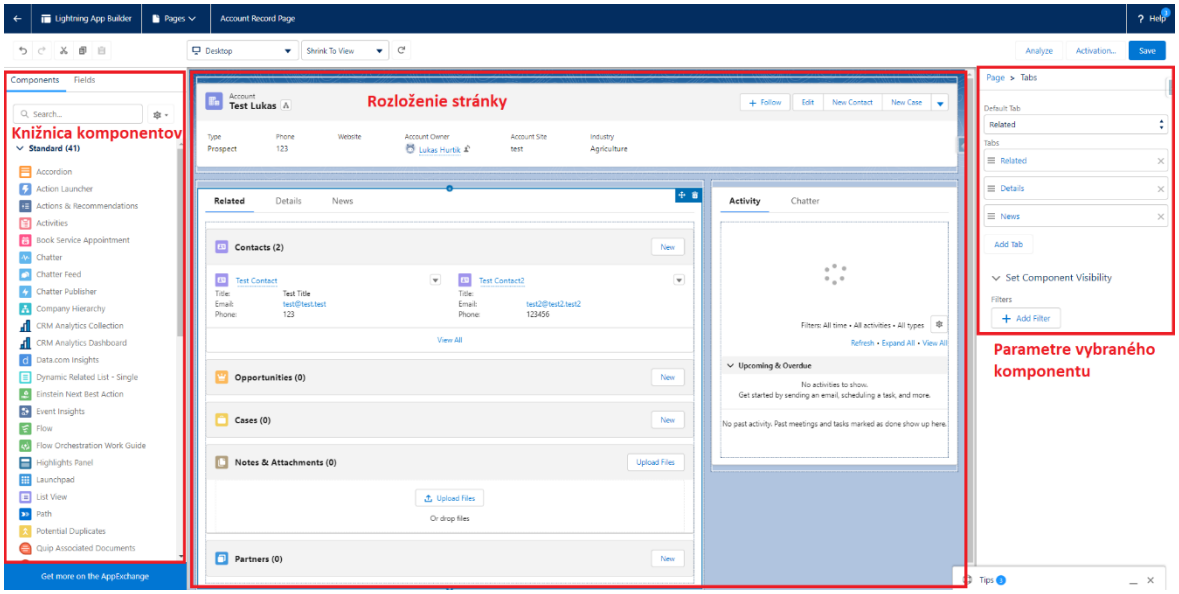
- **Dynamické stránky -** Lightning App Builder umožňuje používateľom vytvárať dynamické stránky, ktoré zobrazujú rôzne komponenty na základe profilu používateľa, typu záznamu alebo iných kritérií. To uľahčuje personalizáciu používateľského prostredia a poskytovanie relevantných informácií každému používateľovi.

V Lightning App Builder je možné vytvoriť 3 typy stránok:

- **Stránka aplikácie** - Používateľ vidí túto stránku po otvorení aplikácie.
- **Domáca stránka -** Používateľ vidí túto stránku po kliknutí na tlačidlo Domov, v rámci aplikácie.
- **Stránka záznamu** - Používateľ vidí túto stránku po otvorení daného záznamu



Obrázok 17 Typy stránok v Lightning App Builder



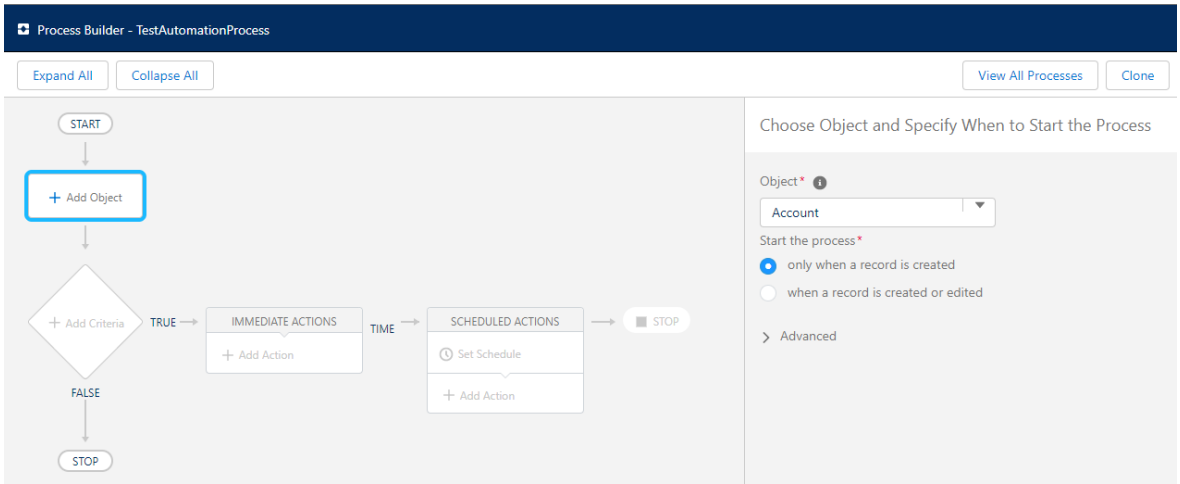
Obrázok 18 Používateľské rozhranie Lightning App Builder

2.3.1.4 Process Builder

Process Builder je vizuálny nástroj v Salesforce, ktorý umožňuje automatizovať obchodné procesy. Pomocou nástroja Process Builder je možné jednoducho vytvárať automatizované akcie, ako je napríklad vytvorenie záznamu, aktualizácia záznamu alebo odoslanie e-mailu.

Medzi hlavné funkcie nástroja Process Builder patria:

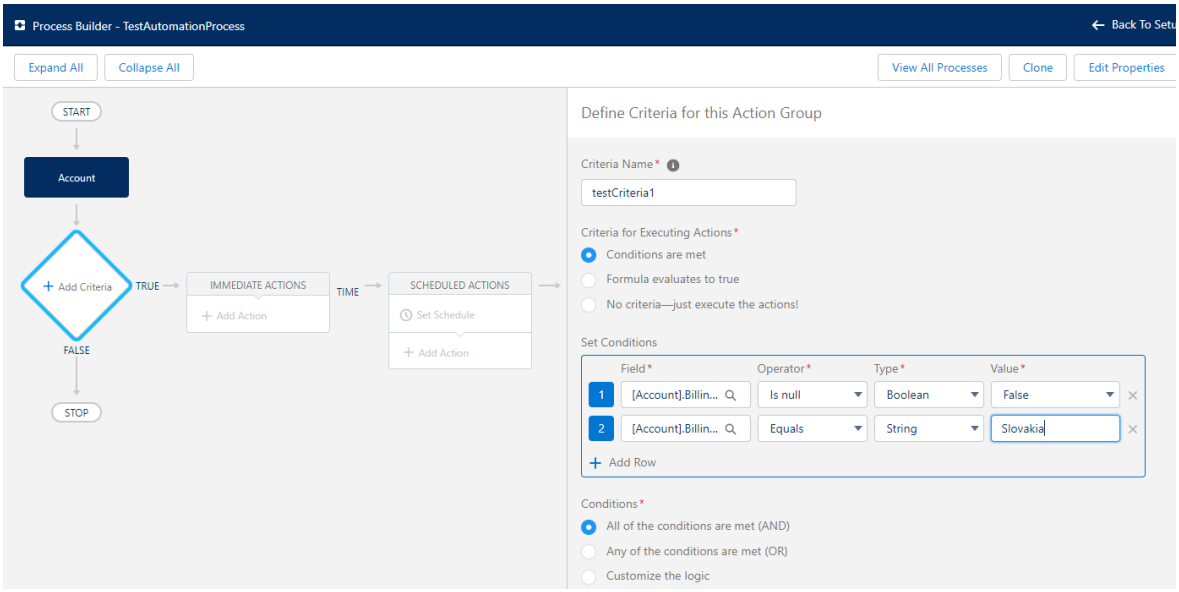
- **Vizuálne rozhranie** - Process Builder používa na vytváranie a správu automatizovaných pracovných postupov vizuálne rozhranie, ktoré uľahčuje vytváranie pracovných postupov aj technicky menej zdatným používateľom, bez potreby programátorských zručností.
- **Spúšťacie podmienky** - Je možné nastaviť spúšťacie podmienky, ktoré spustia proces, napríklad keď sa vytvorí alebo aktualizuje záznam.
- **Akcie** - Je možné nastaviť viacero akcií, ktoré sa vykonajú po spustení procesu. Je možné napríklad vytvoriť nový záznam, aktualizovať záznam alebo odoslať e-mailové oznámenie.
- **Logika** - Process Builder umožňuje nastaviť logiku if/then, čo znamená, že je možné vytvárať podmienené pracovné postupy na základe konkrétnych kritérií.
- **Testovanie** - Pred aktiváciou procesu je možné ho otestovať, čo umožňuje uistiť sa, že funguje tak, ako má. [34] [31]



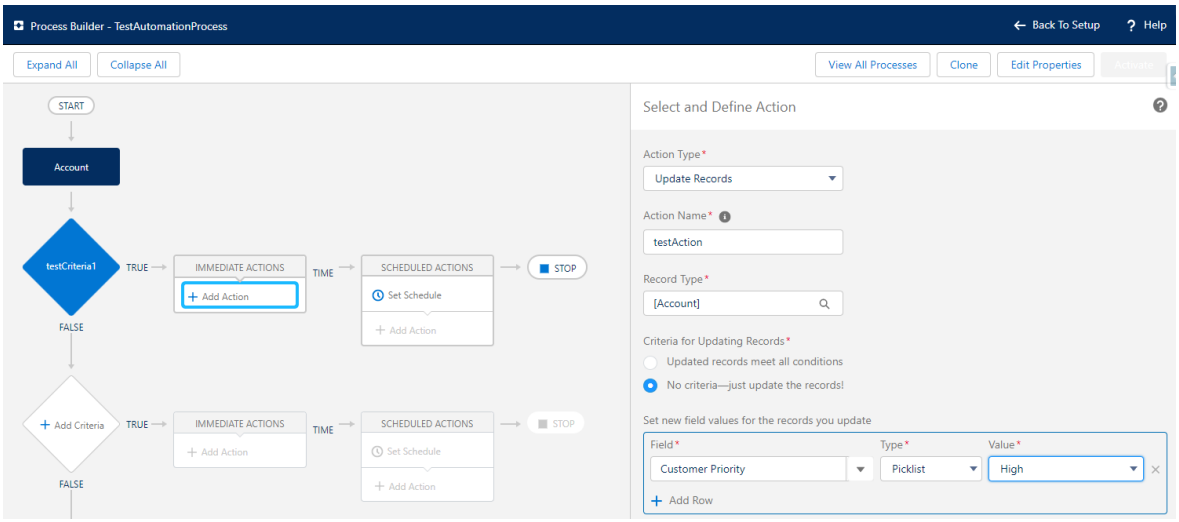
Obrázok 19 Process Builder - Špecifikovanie spúšťacej podmienky

Na obrázku č. 19 je znázornená ukážka vytvárania procesu v Process Builderi. Prvým krokom pri vytváraní procesu je špecifikovanie objektu a spúšťacej podmienky, v tomto prípade sa jedná o objekt Účet a spúšťacia podmienka je nastavená tak, že sa proces spustí vždy iba ak bude vytvorený záznam na danom objekte. Nižšie, na obrázku č. 20 je znázornené definovanie kritérií. V danej ukážke sú kritéria definované nasledovne: ak pole fakturačná krajina (Billing Country) na objekte Účet nie je prázdne a súčasne sa hodnota tohto poľa rovná Slovensko, vyhodnoť kritériá ako pravdivé. V tomto kroku je aj možné vybrať možnosť žiadne kritériá, vtedy sa kritérium vyhodnotí ako pravdivé vždy. Prípadne je možné pri

vyhodnocovaní kritérií uplatniť vlastnú logiku pomocou logických operátorov (AND, OR, NOT).



Obrázok 20 Process Builder - Špecifikovanie kritérií



Obrázok 21 Process Builder - Definovanie akcie

Na obrázku č. 21 je znázornené definovanie akcie. Daná akcia sa vykoná iba v prípade ak príslušné kritérium bolo vyhodnotené pravdivo. V tomto konkrétnom príklade bola akcia nastavená tak, že sa vykoná aktualizácia záznamu na objekte Účet, pole zákaznícka priorita (Customer Priority) sa aktualizuje na hodnotu Vysoká. V tomto kroku je možné nastaviť aj ďalšie doplňujúce kritéria pre vykonanie akcie, tiež je možné aktualizovať niekoľko polí na jednom zázname naraz. Akcia nemusí iba aktualizovať záznam, môže napríklad aj spustiť kód v jazyku Apex, vytvoriť záznam, odoslať email, spustiť flow, spustiť iný proces

a mnoho iného. Je tiež možné nastaviť aby sa akcia nevykonala ihneď ale aby sa vykonala v konkrétnom dátume a čase.

Poradie spúšťania procesov nie je definované, preto aby sa predišlo neplánovanému správaniu systému je odporúčané mať pre jeden objekt iba jeden proces. Pre jeden proces je možné definovať až 200 rôznych kritérií.

Napriek tomu, že je Proces Builder naozaj účinný nástroj, má niekoľko limitácií. Napríklad Process Builder spracováva záznamy po jednom, čo môže viesť k problémom s výkonom a chybám pri práci s veľkými objemami údajov. Limitácie nástroja Proces Builder motivovali spoločnosť Salesforce k tomu aby priniesla ešte výkonnejší nástroj na automatizáciu procesov. V roku 2019 spoločnosť Salesforce predstavila nástroj Flow, ktorý posunul Process Builder do úzadia. Používatelia boli s novým nástrojom veľmi spokojní a aj preto spoločnosť Salesforce oznámila, že v roku 2025 plánuje prestať podporovať nástroj Process Builder. V roku 2022 bola tiež predstavená beta verzia nástroja, ktorý dokáže pretransformovať proces na flow.

2.3.1.5 Flow

Flow je nástroj na automatizáciu procesov v systéme Salesforce, ktorý používateľom umožňuje vytvárať vlastné procesy a automatizovať obchodnú logiku. Flow je podobný nástroju Process Builder, ale ponúka väčšiu flexibilitu a dokáže spracovať zložitejšie prípady použitia. Flow sa môže používať na rôzne účely vrátane vytvárania záznamov, aktualizácie záznamov, odosielania e-mailov a zavolania kódu Apex. Flow tiež môže pracovať s externými službami a spracovávať viac záznamov naraz (na rozdiel od Process Builderu).

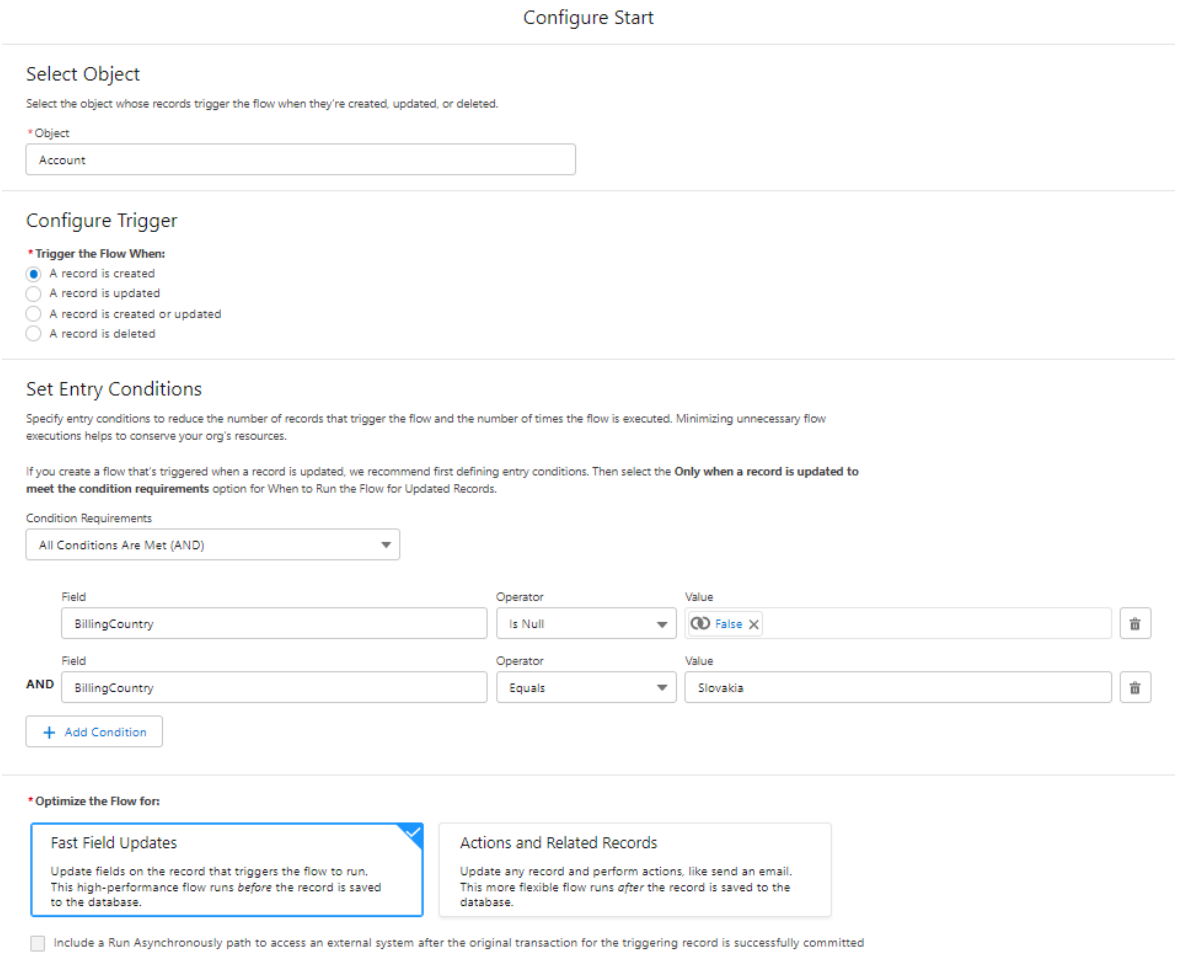
Flow možno vytvárať pomocou nástroja Flow Builder, vizuálneho nástroja, ktorý používateľom umožňuje pridávať rôzne prvky (pomocou drag and drop funkcionality) a definovať ich vlastnosti. Flow možno spúšťať aj rôznymi udalosťami, ako sú zmeny záznamov alebo udalosti platformy. [35] [31]

Existuje niekoľko typov Flowu:

- **Flow obrazovky** - Sústava na seba nadväzujúcich obrazoviek, ktorá vedie používateľov obchodným procesom, ktorý sa spúšťa zo stránok Lightning, rýchlych akcií a ďalších.
- **Flow spustený záznamom** - Spúšťa sa pri vytvorení, aktualizácii alebo odstránení záznamu.

- **Plánovane spustený flow** - Spúšťa sa v určenom čase a frekvencii (napríklad každý pondelok) pre každý záznam.
- **Flow spustený udalosťou platformy** - Spustí sa po prijatí správy o udalosti platformy.
- **Automaticky spúšťaný flow (bez spúšťača)** - Spustí sa, keď ho vyvolá Apex kód, proces, REST API a pod.

Výber typu flowu je prvý krok pri vytváraní flowu. Po vybraní typu, používateľ uvidí konfiguračné okno. Konfiguračné okno flowu, ktorý je spustený záznamom je znázornené na obrázku č. 22.



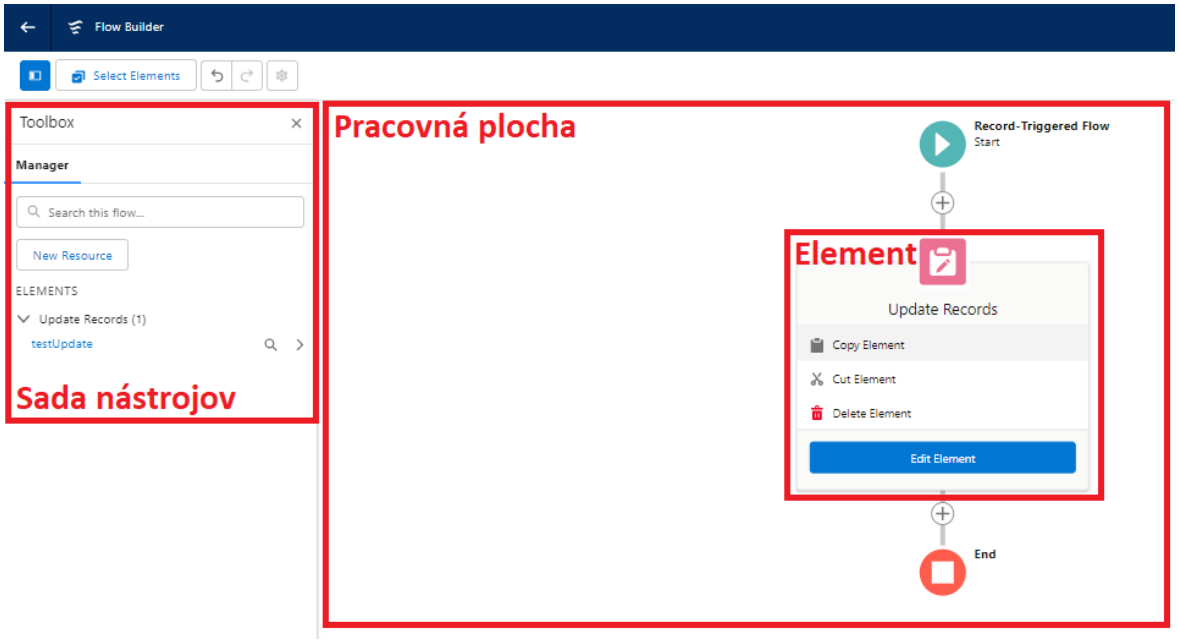
Obrázok 22 Konfiguračné okno flowu, ktorý je spustený záznamom

V konfiguračnom okne používateľ najprv vyberie objekt, ktorého záznamy budú spúšťať flow. Následne vyberie akciu pri ktorej sa flow spustí. Na sú možnosti:

- Keď je záznam vytvorený.
- Keď je záznam aktualizovaný.

- Keď je záznam vytvorený alebo aktualizovaný.
- Keď je záznam vymazaný.

Následne používateľ definuje vstupné kritériá podobne ako aj v Process Builderi. Potom používateľ vyberie typ optimalizácie flowu. Rýchla aktualizácia poľa je možnosť optimalizácie flowu, ktorá je vhodná pre aktualizáciu polia v zázname, ktorý spúšťa flow. Tento vysoko výkonný flow sa spustí pred uložením záznamu do databázy. Akcie a súvisiace záznamy je možnosť optimalizácie flowu, ktorá je vhodná pre aktualizáciu ľubovoľného záznamu a vykonanie akcie, napríklad odoslanie e-mailu. Tento flexibilnejší typ flowu sa spustí po uložení záznamu do databázy. Na konci konfiguračného okna môže používateľ zahrnúť možnosť spustiť asynchrónne, na prístup k externému systému po úspešnom vykonaní pôvodnej transakcie pre spúšťajúci záznam.



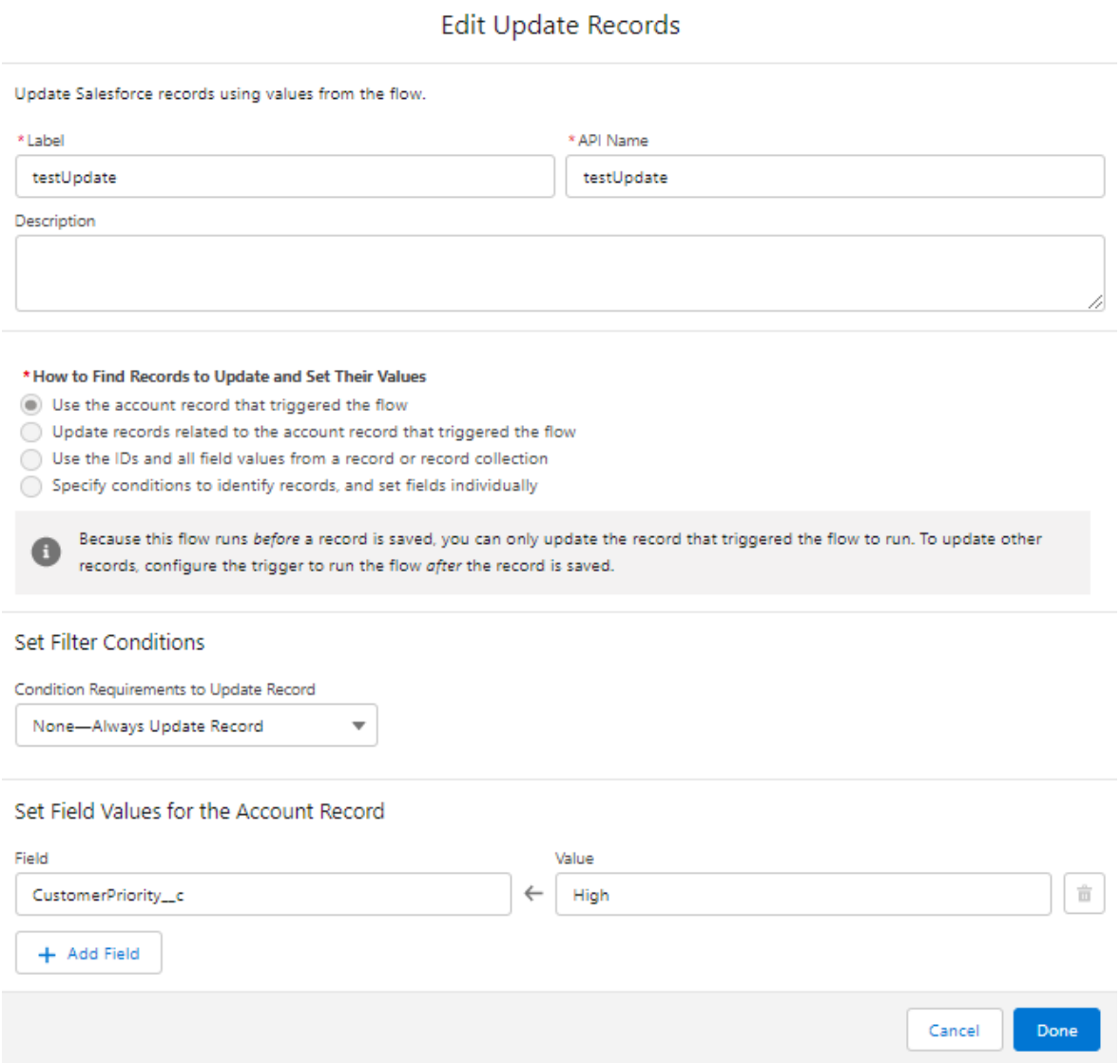
Obrázok 23 Používateľské rozhranie Flow Builder

Po zatvorení konfiguračného okna používateľ uvidí používateľské rozhranie Flow Builder. Na ľavej strane strane sa nachádza sada nástrojov, kde sú uvedené všetky použité elementy a zdroje. V ukážke na obrázku č. 23 sa v sade nástrojov nachádza jeden element s názvom testUpdate, typu aktualizuj záznam. Element je možné pridať kliknutím na symbol + v pravej časti obrazovky, na pracovnej ploche. Po kliknutí na názov elementu, alebo po kliknutí na tlačidlo Edit element v pravej časti obrazovky, používateľ uvidí konfiguračné okno daného elementu. Na obrázku č. 24 je znázornené konfiguračné okno elementu aktualizuj záznam.

Používateľ môže definovať, ktoré záznamy chce aktualizovať. Na výber je hneď niekoľko možností:

- Aktualizovať záznam, ktorý spustil flow
- Aktualizovať záznamy súvisiace so záznamom, ktorý spustil flow
- Použiť unikátne identifikátory (Id) a všetky hodnoty polí zo záznamu alebo kolekcie záznamov
- Zadať podmienky na identifikáciu záznamov

Následne je možné podobne ako v Process Builderi definovať doplňujúce kritériá a na záver je potrebné zadefinovať polia, ktoré budú aktualizované a tiež ich hodnoty.



Obrázok 24 Konfiguračné okno elementu aktualizuj záznam

Rôzne typy flowov poskytujú rôzne typy elementov. Flow spustený záznamom poskytuje tieto elementy:

- **Priradenie** - Priradí premennej hodnotu.
- **Rozhodnutie** - Vyhodnocuje jednu alebo viac podmienok a určuje cestu, ktorou by sa mal flow ďalej uberať.
- **Slučka** - Iteruje nad kolekciou záznamov a vykoná súbor akcií pre každý záznam v kolekcii.
- **Triedenie kolekcie** - Triedi kolekciu záznamov na základe zadaného poľa. Prvok možno použiť na usporiadanie kolekcie záznamov na zobrazenie alebo na spracovanie záznamov v určitom poradí.
- **Filtrovanie kolekcie** - Umožňuje filtrovať kolekciu záznamov na základe zadanej podmienky. Prvok možno použiť na vyčlenenie podmnožiny záznamov z väčšej kolekcie na základe špecifických kritérií.
- **Získaj záznam** - Umožňuje získať kolekciu záznamov z objektu Salesforce. Prvok možno použiť na získanie údajov zo Salesforce na spracovanie vo flowe alebo na získanie údajov na zobrazenie v rozhraní.
- **Aktualizuj záznam** - Slúži na aktualizáciu záznamov

Všetky dostupné elementy sú podrobne popísané v Salesforce dokumentácii spolu s ukážkami použitia.

2.3.2 Programovacie spôsoby vývoja aplikácií

Salesforce poskytuje široké spektrum programovacích vývojových nástrojov, ktoré umožňujú používateľom vytvárať špecifické aplikácie na mieru pomocou písania kódu. Programovacie spôsoby vývoja aplikácií sa zvyčajne používajú ak požiadavky klienta nie je možné splniť použitím deklaratívnych spôsobov vývoja aplikácií. Medzi najpoužívanejšie programovacie vývojové nástroje patria:

- Programovací jazyk Apex
- Visualforce
- Lightning component framework programovacieho jazyka Java Script

2.3.2.1 Programovací jazyk Apex

Apex je proprietárny objektovo orientovaný programovací jazyk používaný v platforme Salesforce na vytváranie špecifickej logiky a integrácií na mieru. Je veľmi podobný jazyku Java a je určený na prácu s dátovým modelom a API rozhraniami spoločnosti Salesforce. Medzi kľúčové vlastnosti programovacieho jazyka Apex patria:

- **Objektovo orientované programovanie** - Apex je objektovo orientovaný programovací jazyk, čo znamená, že používa objekty na zapuzdrenie údajov. Podporuje triedy, dedičnosť, rozhrania a ďalšie koncepty objektovo orientovaného programovania.
- **Dátový model Apex** - Apex je navrhnutý na prácu s dátovým modelom Salesforce, ktorý pozostáva z objektov, polí a vzťahov medzi objektami. Apex poskytuje súbor zabudovaných tried a metód na vyhľadávanie a manipuláciu s údajmi Salesforce.
- **Spúšťače (Triggers)** - Apex podporuje spúšťače, čo sú bloky kódu, ktoré sa vykonajú pred alebo po vytvorení, aktualizácii alebo odstránení záznamu. Spúšťače možno použiť na aktualizáciu súvisiacich záznamov alebo vykonanie iných akcií na základe zmien záznamov.
- **Správcovské limity** - Všetok kód Apex beží na platforme Lightning, ktorá je zdieľaným zdrojom používaným všetkými inštanciami Salesforce. Na zaručenie konzistentného výkonu a škálovateľnosti je vykonávanie Apexu obmedzené správcovskými limitmi, ktoré zabezpečujú, že žiadne jednotlivé vykonávanie Apexu neovplyvní celkovú činnosť služby Salesforce. To znamená, že všetok kód Apex je obmedzený počtom operácií (napríklad DML alebo SOQL), ktoré môže vykonať v rámci jednej transakcie. Návratová hodnota pre všetky Apex požiadavky je kolekcia, ktorá obsahuje od 1 do 50 000 záznamov. Pri písaní kódu je neprípustné predpokladať, že kód bude pracovať vždy len s jedným záznamom. Preto je potrebné implementovať programovacie schémy, ktoré zohľadňujú hromadné spracovanie záznamov.
- **Integrácia s inými jazykmi** - Apex je možné integrovať s inými programovými jazykmi pomocou webových služieb alebo API rozhraní. Apex môže napríklad využívať rozhrania REST API, SOAP API alebo iné externé služby.
- **Testovací framework Apex** - Apex poskytuje testovací framework na písanie jednotkových testov pre kód Apex. Testy Apex možno použiť na zabezpečenie toho,

aby kód fungoval podľa očakávaní, a na zabránenie regresiám pri zmenách v kóde. [36] [37] [38]

Pri pomenovaní premenných, metód alebo tried nie je možné použiť žiadne z rezervovaných kľúčových slov Apexu. Patria sem slová, ktoré sú súčasťou platformy Apex a Lightning, ako napríklad list, test, if, select alebo account. Apex je silne typový jazyk, to znamená, že pri prvom odkaze na premennú je potrebné deklarovať jej dátový typ. Dátové typy jazyka Apex zahŕňajú základné typy, ako sú Integer, Date a Boolean, ako aj pokročilejšie typy, ako sú zoznamy, mapy, objekty a sObjects. Premenné sa deklarujú pomocou názvu a dátového typu. Pri deklarácii je možné premennej priradiť hodnotu, avšak tiež je možné premennej priradiť hodnotu aj neskôr. Pri deklarovaní premenných sa používa nasledujúca syntax:

```
// Nasledujúca premenná má dátový typ Integer
// s názvom Count a má hodnotu 0.
Integer Count = 0;
// Nasledujúca premenná má dátový typ Decimal
// s názvom Total a nebola jej priradená žiadna hodnota
Decimal Total;
// Nasledujúca premenná je účet,
// ktorý sa označuje aj ako sObject
Account MyAcct = new Account();
```

Obrázok 25 Príklad syntaxe pri pomenovávaní premenných

V Apexe je sObject skratkou pre "Salesforce objekt", čo je typ objektu, ktorý predstavuje záznam v databáze Salesforce. Objektom sObject môže byť štandardný objekt, napríklad účet alebo kontakt, alebo vlastný objekt, ktorý používateľ vytvoril vo svojej organizácii. Objekt sObject je podobný tabuľke v databáze, kde každý záznam predstavuje riadok v tabuľke a každé pole predstavuje stĺpec v tabuľke. Jednou z kľúčových výhod používania sObjects v Apexe je, že sú dynamicky typované, čo znamená, že je možné pristupovať k poliam a vlastnostiam sObject pomocou bodkového zápisu (dot notation) bez toho, aby bolo potrebné vopred špecifikovať dátový typ poľa. To uľahčuje prácu s rôznymi typmi objektov v jednej kódovej základni a umožňuje písať kód, ktorý je flexibilnejší a opakovane použiteľný. [38] [39]

Dátové štruktúry, ktoré umožňujú zoskupiť viacero hodnôt do jedného objektu sa v jazyku Apex nazývajú kolekcie. Existujú tri hlavné typy kolekcií:

- **Zoznamy** - Zoznamy sú usporiadané kolekcie prvkov rovnakého dátového typu. Prvky zoznamu je možné pridávať, odstraňovať a načítavať pomocou indexových pozícií. Zoznamy môžu obsahovať až 4 milióny prvkov.
- **Sady** - Sady sú neusporiadané kolekcie jedinečných prvkov rovnakého dátového typu. Je možné pridávať a odstraňovať prvky zo sady a kontrolovať, či sada obsahuje konkrétny prvok. Sady sú užitočné, keď je potrebné zabezpečiť, aby kolekcia obsahovala len jedinečné prvky.
- **Mapy** - Mapy sú kolekcie dvojíc kľúč-hodnota, kde každý kľúč zodpovedá príslušnej hodnote. Kľúče môžu byť ľubovoľného dátového typu, ale musia byť v rámci mapy jedinečný. Do mapy je možné pridávať, načítavať a odstraňovať páry kľúč-hodnota. [38] [40]

Na získavanie dát z databázy Salesforce používa dva vyhľadávacie jazyky, SOQL (Salesforce Object Query Language) a SOSL (Salesforce Object Search Language).

Jazyk SOQL (Salesforce Object Query Language) sa používa na vyhľadávanie záznamov z jedného objektu alebo súvisiacich objektov v Salesforce. Je podobný jazyku SQL a používa syntax podobnú príkazu SELECT jazyka SQL. Pomocou jazyka SOQL je možné filtrovať, usporadúvať a zoskupovať záznamy, ako aj získavať polia zo súvisiacich objektov pomocou vzťahov, ako je napríklad vzťah rodič-dieťa, lookup a master-detail. Nižšie, na obrázku č. 25 je príklad základného dotazu SOQL na získanie polí Id, Meno a Priemysel všetkých účtov.

Jazyk SOSL (Salesforce Object Search Language) umožňuje vykonávať textové vyhľadávanie vo viacerých poliach a objektoch. SOSL je ideálny na vyhľadávanie záznamov na základe kľúčových slov a fráz a dokáže vrátiť výsledky z viacerých objektov v rámci jedného dotazu. Nižšie, na obrázku č. 25 je príklad základného dotazu SOSL na vyhľadávanie účtov a kontaktov, ktorých hodnota ktoréhokoľvek poľa obsahuje vyhľadávané kľúčové slovo. [37] [41]

```
//SOQL
List<Account> accounts = [SELECT Id, Name, Industry FROM Account];
//SOSL
List<List<SObject>> searchResults = [FIND 'Acme' IN ALL FIELDS RETURNING
                                     Account (Id, Name),
                                     Contact (Id, FirstName, LastName)];
```

Obrázok 26 Príklad použitia jazykov SOSL a SOQL

Jazyk SOQL sa odporúča použiť vtedy, keď je známe, v ktorých objektoch sa údaje nachádzajú, a ak je potrebné:

- Získať údaje z jedného objektu alebo z viacerých objektov, ktoré sú navzájom prepojené.
- Spočítať počet záznamov, ktoré spĺňajú zadané kritériá.
- Zoradiť výsledky ako súčasť dotazu.
- Získať údaje z číselných polí, dátumových polí alebo zaškrtávacích polí.

Jazyk SOSL sa odporúča použiť vtedy, keď nie je známe, v ktorom objekte alebo poli sa údaje nachádzajú, a ak je potrebné:

- Získať údaje pre konkrétny výraz, o ktorom je známe, že existuje v rámci poľa. Keďže SOSL dokáže tokenizovať viacero termínov v rámci poľa a vytvoriť z nich index vyhľadávania, vyhľadávanie pomocou SOSL je rýchlejšie a môže vrátiť relevantnejšie výsledky.
- Efektívne vyhľadávať viaceré objekty a polia, pričom tieto objekty môžu, ale nemusia byť navzájom prepojené.
- Vyhľadávať údaje, ktoré sú v čínštine, japončine, kórejčine alebo thajčine. Morfologická tokenizácia pre výrazy CJKT (Chinese, Japan, Korean, Thai) pomáha zabezpečiť presné výsledky. [37]

V Salesforce sa na manipuláciu s údajmi v databáze používa jazyk DML (Data Manipulation Language). Používa sa na vykonávanie operácií CRUD (Create, Read, Update, Delete) s údajmi uloženými v objektoch. Štyri hlavné operácie DML sú:

- **Vložiť (Insert)** - Táto operácia sa používa na vytvorenie nového záznamu v objekte. Na vykonanie operácie vloženia je potrebné vytvoriť novú inštanciu objektu a nastaviť hodnoty jeho polí. Potom je možné zavolať metódu insert(), aby sa nový záznam uložil do databázy.
- **Aktualizácia (Update)** - Táto operácia sa používa na úpravu existujúceho záznamu v objekte. Pre vykonanie operácie aktualizácie, je potrebné získať záznam (buď pomocou SOQL alebo SOSL), ktorý bude aktualizovaný, upraviť hodnoty jeho polí a potom zavolať metódu update() na objekte, aby sa uložili zmeny.
- **Odstránenie (Delete)** - Táto operácia sa používa na odstránenie záznamu z objektu. Pre vykonanie operácie odstránenia, je potrebné získať záznam (buď pomocou

SOQL alebo SOSL), ktorý bude vymazaný, a potom zavolať metódu delete() na objekte, aby bol odstránený z databázy.

- **Upsert** - Táto operácia sa používa buď na vloženie nového záznamu, alebo na aktualizáciu existujúceho záznamu na základe jedinečného identifikátora (Id). Pre vykonanie operácie upsert, je potrebné vytvoriť novú inštanciu objektu, nastaviť hodnoty jeho polí a potom zavolať metódu upsert() na objekte so zadaným poľom jedinečného identifikátora. Na určenie, či záznam už existuje, príkaz upsert používa ID záznamu ako kľúč na porovnanie záznamov. Ako kľúč je možné použiť aj externé pole ID alebo štandardné pole s atribútom idLookup nastaveným na hodnotu true. Ak sa kľúč nezhoduje, vytvorí sa nový záznam objektu. Ak sa kľúč aspoň raz zhoduje, potom sa aktualizuje existujúci záznam objektu. Ak sa kľúč zhoduje viackrát, potom sa vygeneruje chyba a záznam objektu sa nevloží ani neaktualizuje. [38] [42]

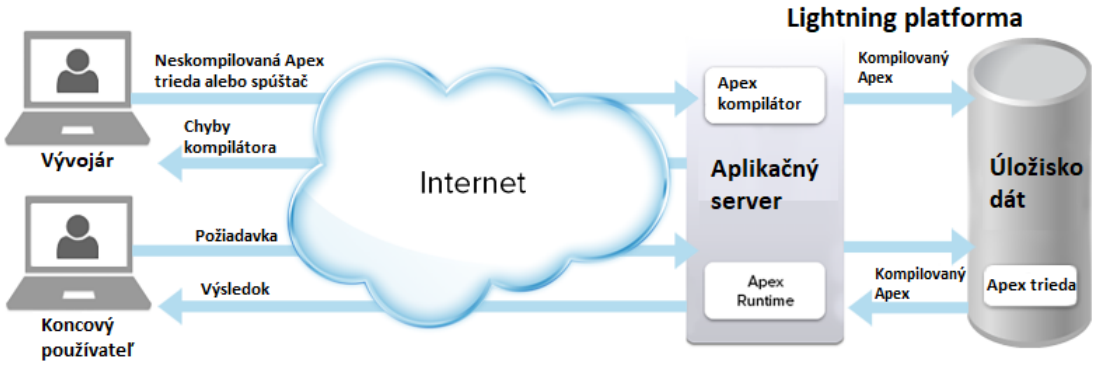
Príklady použitia jednotlivých DML operácií sú uvedené nižšie na obrázku č. 27.

```
//insert
Account newAcct = new Account(name = 'Acme');
insert newAcct;

//update
Account myUpdateAcct = [SELECT Id, Name, BillingCity FROM Account WHERE Id = :newAcct.Id];
myUpdateAcct.BillingCity = 'San Francisco';
update myUpdateAcct;

//delete
Account myDeleteAcct = [SELECT Id, Name, BillingCity FROM Account WHERE Id = :newAcct.Id];
delete myDeleteAcct;

//upsert - aktualizuje názov mesta pre všetky existujúce účty nachádzajúce sa v meste,
//         ktoré bolo predtým známe ako Bombaj,
//         a tiež vloží nový účet nachádzajúci sa v San Franciscu
Account[] acctsList = [SELECT Id, Name, BillingCity FROM Account WHERE BillingCity = 'Bombay'];
for (Account a : acctsList) {
    a.BillingCity = 'Mumbai';
}
Account newAcct = new Account(Name = 'Acme', BillingCity = 'San Francisco');
acctsList.add(newAcct);
upsert acctsList;
```

Obrázok 27 Príklady jednotlivých DML operácií

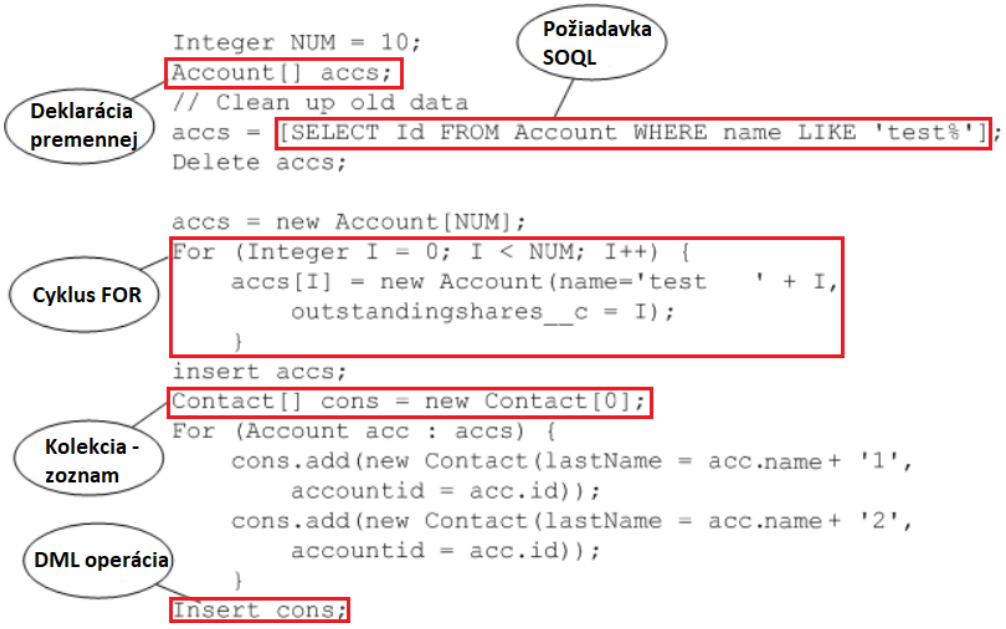
Všetok Apex kód sa kompiluje, ukladá a spúšťa výhradne na platforme Lightning. Vývojári píšu a ukladajú kód Apex na platformu a koncoví používatelia spúšťajú vykonávanie kódu Apex prostredníctvom používateľského rozhrania. Keď vývojár napíše a uloží kód Apex na platformu, aplikačný server platformy najprv skompiluje kód do abstraktnej sady inštrukcií, ktorým rozumie Apex runtime interpretátor, a potom tieto inštrukcie uloží ako metadáta. Keď koncový používateľ spustí vykonávanie Apexu, napríklad kliknutím na

tlačidlo alebo prístupom na stránku Visualforce, aplikačný server platformy načíta skompilované inštrukcie z metadát a odošle ich cez runtime interpretátor pred vrátením výsledku. Koncový používateľ nepozoruje žiadne rozdiely v čase vykonávania oproti štandardným požiadavkám platformy, v ktorých sa Apex nepoužíva. [43]



Obrázok 28 Apex architektúra [44]

V používateľskom rozhraní Salesforce je možné určiť verziu rozhrania Salesforce API, na základe ktorej sa má uložiť trieda alebo spúšťač Apex. Rozhranie API spoločnosti Salesforce je súbor protokolov a nástrojov, ktoré môžu vývojári používať na interakciu s platformou Salesforce, pričom každá verzia rozhrania API poskytuje rôzne funkcie a možnosti. Pri vytváraní novej triedy alebo spúšťača Apex je potrebné určiť verziu API, ktorá bude použitá. To určuje súbor funkcií a správania, ktoré sú k dispozícii pre kód napísaný v danej triede alebo v spúšťači. Verziu je možné zmeniť po uložení. Každý názov triedy alebo spúšťača musí byť jedinečný. Tú istú triedu alebo spúšťač nie je možné uložiť voči rôznym verziám.



Obrázok 29 Príklad Apex kódu

2.3.2.2 Visualforce

Visualforce je webový framework, ktorý sa používa na vytváranie vlastných používateľských rozhraní v Salesforce. Je to značkovací jazyk podobný HTML a používa sa na vytváranie stránok a komponentov, ktoré komunikujú s údajmi a obchodnou logikou Salesforce. Umožňuje vývojárom vytvárať stránky, ktoré vyzerajú a pôsobia ako súčasť platformy Salesforce, ale majú špecifickú funkcionalitu a vzhľad na mieru. Tieto stránky možno použiť na zobrazenie, úpravu alebo vytvorenie záznamov, ako aj na zobrazenie údajov z externých zdrojov. Visualforce poskytuje aj výkonnú sadu komponentov, ako sú vstupné polia, tlačidlá a grafy, ktoré možno použiť na vytvorenie zložitejších používateľských rozhraní.

Jednou z kľúčových výhod nástroja Visualforce je jeho úzka integrácia s platformou Salesforce. To znamená, že stránky Visualforce môžu pristupovať k údajom a metadátam Salesforce a manipulovať s nimi a môžu tiež využívať kód Apex a ďalšie rozhrania API Salesforce na implementáciu zložitejšej obchodnej logiky.

Visualforce podporuje aj architektúru založenú na komponentoch, ktorá umožňuje vývojárom vytvárať opakovane použiteľné komponenty používateľského rozhrania, ktoré možno zdieľať na viacerých stránkach a v rôznych aplikáciách. To pomáha skrátiť čas vývoja a zjednodušiť údržbu. [45] [46]

```
<apex:page controller="AccountController">
    <apex:pageBlock title="Account List">
        <apex:pageBlockTable value="{!accounts}" var="acc">
            <apex:column headerValue="Account Name" value="{!acc.Name}"/>
            <apex:column headerValue="Phone" value="{!acc.Phone}"/>
            <apex:column headerValue="Website" value="{!acc.Website}"/>
        </apex:pageBlockTable>
    </apex:pageBlock>
</apex:page>
```

```
public class AccountController {
    public List<Account> accounts { get; set; }

    public AccountController() {
        accounts = [SELECT Id, Name, Phone, Website FROM Account];
    }
}
```

Obrázok 30 Príklad Visualforce stránky spolu s Apex ovládačom

Vyššie, na obrázku č. 30 je znázornená stránka Visualforce definovaná pomocou komponentu apex:page. Stránka je spojená s Apex ovládačom s názvom AccountController, ktorý

je zodpovedný za poskytovanie údajov, ktoré sa majú na stránke zobraziť. Trieda AccountController obsahuje jedinú premennú s názvom accounts, čo je zoznam záznamov objektu Účet. Premenná accounts je definovaná ako verejná premenná s metódami getter a setter, ktoré umožňujú iným triedam alebo komponentom prístup k nej a jej modifikáciu.

Trieda AccountController tiež obsahuje konštruktor metódu, ktorá je zodpovedná za inicializáciu premennej accounts so zoznamom záznamov objektu Účet. V tomto prípade metóda konštruktora vyhľadáva záznamy objektu Účet z databázy Salesforce pomocou dotazu SOQL a napĺňa premennú accounts výsledkami daného dotazu. Pripojením tejto triedy AccountController k stránke Visualforce pomocou atribútu controller môže stránka pristupovať k premennej accounts a zobrazovať záznamy objektu Účet v tabuľke.

Komponent apex:pageBlock sa používa na definovanie časti stránky s názvom "Zoznam účtov". V rámci tejto sekcie sa komponent apex:pageBlockTable používa na vytvorenie tabuľky, ktorá zobrazuje zoznam účtov. Atribút value komponentu apex:pageBlockTable je nastavený na premennú accounts ovládača, čo je zoznam záznamov objektu Účet v Salesforce. Atribút var sa používa na definovanie názvu premennej acc, ktorá predstavuje každý jednotlivý záznam objektu Účet v tabuľke.

Nakoniec komponenty apex:column definujú stĺpce tabuľky, pričom atribút headerValue definuje text záhlavia stĺpca a atribút value určuje dátové pole, ktoré sa má zobraziť v každej bunke.

Napriek tomu, že bol Visualforce predstavený ešte v roku 2006, je stále podporovanou technológiou v rámci Salesforce a existuje mnoho aplikácií, ktoré ju naďalej efektívne využívajú. Keďže sa však Salesforce naďalej vyvíja a zavádza nové technológie, Visualforce nemusí byť vždy najlepšou voľbou na vytváranie moderných, flexibilných a citlivých používateľských rozhraní.

Nižšie je uvedených niekoľko dôvodov, prečo Visualforce nemusí byť najlepšou voľbou pre určité prípady použitia:

- **Obmedzená podpora moderných frameworkov používateľského rozhrania -** Visualforce bol navrhnutý pred tým, ako sa stali populárne moderné frameworky pre vývoj webových stránok, ako sú React a Angular, takže neponúka rovnakú úroveň flexibility a responzivity ako tieto frameworky. To spôsobuje, že je náročné vytvoriť moderné, používateľsky prívetivé rozhrania, ktoré budú dobre fungovať na rôznych zariadeniach.

- **Obmedzená integrácia s externými systémami** - Visualforce je úzko integrovaný so Salesforce, čo má za následok, že je náročné integrovať ho s externými systémami alebo vytvárať vlastné integrácie, ktoré nezapadajú do dátového modelu Salesforce.
- **Nedostatočná podpora pre Lightning Experience** - Moderné používateľské rozhranie Salesforce, Lightning Experience, nepodporuje Visualforce v plnej miere. Hoci existujúce stránky Visualforce možno v rámci Lightning Experience stále používať, nemusia ponúkať rovnakú úroveň funkčnosti alebo odozvy ako natívne komponenty Lightning. [45] [46]

Pri vývoji nových komponentov, pre dosiahnutie najmodernejších a najresponzívnejších funkcií Salesforce odporúča používať low-code nástroje Lightning Experience a webové komponenty Lightning namiesto Visualforce. Platforma Lightning poskytuje vývojárom rôzne spôsoby vytvárania vlastných funkcií a podporuje novšie, komplexné obchodné procesy, ktoré nie sú k dispozícii v nástroji Visualforce.

Napriek mnohým nevýhodám Salesforce stále podporuje Visualforce, pretože je to stále populárna technológia v komunite vývojárov Salesforce a existuje mnoho aplikácií, ktoré ju naďalej efektívne využívajú. V blízkej budúcnosti sa preto nepredpokladá, že by Salesforce prestal podporovať Visualforce.

2.3.2.3 Lightning component framework programovacieho jazyka Java Script

Lightning Component Framework je framework pre vývoj používateľského rozhrania (UI) vytvorený spoločnosťou Salesforce na vytváranie aplikácií v rámci platformy Salesforce. Lightning Component Framework využíva model založený na komponentoch, v ktorom je každý komponent samostatný a zapuzdruje konkrétnu funkcionalitu, napríklad zobrazovanie údajov, spracovanie interakcií s používateľom alebo vykonávanie výpočtov. Komponenty sa dajú kombinovať do väčších a zložitejších komponentov a dajú sa opakovane používať vo viacerých aplikáciách a stránkach. [47] [48]

Framework je založený na návrhovom vzore Model-View-Controller (MVC), ktorý oddeľuje používateľské rozhranie (View) od obchodnej logiky (Model) a ovládača (Controller), ktorý spravuje interakciu medzi nimi. Toto oddelenie umožňuje jednoduchšiu údržbu, testovanie a škálovateľnosť aplikácie. [48]

Lightning Component Framework ponúka dva typy komponentov: Aura komponenty a Lightning Web komponenty. Hoci oba typy komponentov umožňujú vývojárom vytvárať

opakovane použiteľné, vysoko výkonné komponenty pre platformu Salesforce, existujú medzi nimi niektoré kľúčové rozdiely. [49]

Komponenty Aura boli pôvodným typom komponentov, ktoré spoločnosť Salesforce predstavila pre Lightning Component Framework v roku 2014. Používajú framework Aura, čo je framework používateľského rozhrania vyvinutý spoločnosťou Salesforce. Aura komponenty sú tvorené kombináciou značkovaného jazyka podobného HTML, JavaScriptu a CSS. Komponenty Aura podporujú obojsmerné viazanie údajov, programovanie riadené udalosťami (event-driven) a akcie na strane servera. [50]

Nižšie, na obrázku č. 31 je uvedený príklad Aura komponentu, ktorý zobrazuje tabuľku účtov s ich názvami, odvetviami a telefónnymi číslami. Súbor testComponent.cmp je súbor ktorý obsahuje značkovací jazyk podobný HTML. Obsahuje atribút s názvom accounts typu zoznam (List), ktorý bude obsahovať zoznam účtov, ktoré sa zobrazia v tabuľke. Následne je použitý tag aura handler, ktorý pri inicializácii komponentu zavolá funkciu "getAccounts" definovanú v JavaScript ovládači. Atribút value je nastavený na {!this}, ktorý odkazuje na aktuálnu inštanciu komponentu. Následne je definovaná tabuľka pomocou štandardných HTML prvkov tabuľky, údaje o účtoch sa zobrazia pomocou tagu apex iterate, čo je v podstate cyklus, ktorý iteruje cez zoznam účtov uložených v atribúte accounts. Tento Aura komponent využíva aj Salesforce Lightning Design System (SLDS) na definovanie štýlu tabuľky a jej prvkov. SLDS je framework CSS poskytovaný spoločnosťou Salesforce, ktorý pomáha vývojárom vytvárať používateľské rozhrania, ktoré zodpovedajú vzhľadu a štýlu Salesforce.

JavaScriptový ovládač testComponent.js obsahuje funkciu getAccounts. Vo funkcii sa vytvorí akcia, ktorá zavolá metódu fetchAccounts definovanú v Apex ovládači. Odpoveď volania ovládača Apex sa potom nastaví na atribút accounts komponentu.

V Apex ovládači MyController.cls je definovaná metóda fetchAccounts, čo je statická metóda, ktorá vracia zoznam účtov, na ktoré sa dotazuje z databázy pomocou SOQL dotazu. Anotácia "@AuraEnabled" sa používa na označenie toho, že metódu možno volať z JavaScript ovládača na strane klienta.

V CSS súbore je súbor štýlov, ktorý štylizuje tabuľku zobrazenú v komponente. Pridáva do buniek tabuľky výplň, nastavuje šírku tabuľky na 100 % a centruje text v bunkách tabuľky. Štýly CSS sa vzťahujú na komponent pomocou selektora .THIS, ktorý zabezpečuje, že štýly sa vzťahujú len na tento komponent a nie na iné komponenty alebo prvky na tej istej stránke.

Component Markup (testComponent.cmp):

```
<aura:component controller="MyController">
    <aura:attribute name="accounts" type="List" />
    <aura:handler name="init" value="{!this}" action="{!c.getAccounts}" />
    <table class="slds-table slds-table_bordered slds-table_cell-buffer">
        <thead>
            <tr class="slds-text-title_caps">
                <th scope="col">
                    <div class="slds-truncate">Account Name</div>
                </th>
                <th scope="col">
                    <div class="slds-truncate">Industry</div>
                </th>
                <th scope="col">
                    <div class="slds-truncate">Phone</div>
                </th>
            </tr>
        </thead>
        <tbody>
            <aura:iteration items="{!v.accounts}" var="account">
                <tr>
                    <td>
                        <div class="slds-truncate">{!account.Name}</div>
                    </td>
                    <td>
                        <div class="slds-truncate">{!if(account.Industry, account.Industry, '')}</div>
                    </td>
                    <td>
                        <div class="slds-truncate">{!if(account.Phone, account.Phone, '')}</div>
                    </td>
                </tr>
            </aura:iteration>
        </tbody>
    </table>
</aura:component>
```

JavaScript Controller (testComponent.js):

```
({
    getAccounts: function(component, event, helper) {
        var action = component.get("c.fetchAccounts");
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                component.set("v.accounts", response.getReturnValue());
            }
        });
        $A.enqueueAction(action);
    }
})
```

Apex Controller (MyController.cls):

```
public with sharing class MyController {
    @AuraEnabled
    public static List<Account> fetchAccounts() {
        return [SELECT Name, Industry, Phone FROM Account];
    }
}
```

CSS:

```
.THIS .slds-table_cell-buffer {
    padding: 10px;
}

.THIS .slds-table {
    width: 100%;
}

.THIS .slds-table td, .THIS .slds-table th {
    text-align: center;
}
```

Obrázok 31 Ukážka Aura komponentu

Na druhej strane Lightning Web komponenty (LWC) sú novším typom komponentov, ktoré spoločnosť Salesforce predstavila v roku 2019. Využívajú moderné webové štandardy, poskytujú lepší výkon a podporujú aj obojsmerné viazanie údajov aj jednosmerný tok údajov. LWC má tiež jednoduchší a konzistentnejší programovací model a možno ho použiť na vytváranie či už samostatných aplikácií alebo aj zabudovaných komponentov v rámci platforiem.
Nižšie, na obrázku č. 32 je uvedený príklad Lightning Web komponentu, ktorý zobrazuje tabuľku účtov s ich názvami, odvetviami a telefónnymi číslami. Komponent sa skladá zo štyroch súborov: súbor HTML (testComponent.html), súbor JavaScript ovládača (testComponent.html.js), súbor ovládača Apex (AccountController.cls) a súbor CSS (testComponent.html.css).

Súbor HTML definuje komponent lightning-card, ktorý obsahuje komponent lightning-datatable. Komponent lightning-datatable zobrazuje zoznam účtov so stĺpcami pre názov účtu, odvetvie a telefónne číslo. Údaje pre tabuľku sa získavajú z JavaScript ovládača a stĺpce sú definované pomocou konštantného poľa záznamov objektu Účet.

Súbor ovládača JavaScript exportuje predvolenú triedu testComponent, ktorá rozširuje triedu LightningElement, ktorá je základnou triedou pre všetky Lightning Web komponenty. Trieda má dve premenné: accounts a columns. Premenná accounts je pole, ktoré obsahuje údaje pre tabuľku, a premenná columns je pole, ktoré definuje stĺpce pre tabuľku. Trieda definuje aj tzv. wire adaptér, ktorý získava údaje z metódy getAccounts ovládača Apex.

Súbor ovládača Apex definuje verejnú statickú metódu getAccounts, ktorá vracia zoznam záznamov objektu Účet s poľami pre názov účtu, odvetvie a telefónne číslo. Metóda je označená ako cacheable=true, aby sa zlepšil výkon ukladaním výsledku do vyrovnávacej pamäte. Metóda getAccounts() je jednoduchá a pri každom volaní vracia rovnaký výsledok, takže je vhodným kandidátom na ukladanie do vyrovnávacej pamäte. Ak sa však údaje vrátené metódou môžu často meniť alebo závisia od kontextu používateľa alebo iných premenných, ukladanie do vyrovnávacej pamäte nemusí byť vhodné a namiesto toho by sa mala byť metóda označená ako cacheable=false.

Súbor CSS definuje niektoré štýly pre komponent lightning-card a komponent lightning-datatatable. Štýly zahŕňajú okraj, farbu pozadia, veľkosť písma a výplň pre hlavičku karty a výplň pre bunky tabuľky.

Component Markup (testComponent.html):

```
<template>
    <lightning-card title="Account List">
        <lightning-datatable
            key-field="Id"
            data={accounts}
            columns={columns}
            hide-checkbox-column>
        </lightning-datatable>
    </lightning-card>
</template>
```

JavaScript Controller (testComponent.js):

```
import { LightningElement, wire } from 'lwc';
import getAccounts from '@salesforce/apex/AccountController.getAccounts';

const COLUMNS = [
    { label: 'Name', fieldName: 'Name' },
    { label: 'Industry', fieldName: 'Industry' },
    { label: 'Phone', fieldName: 'Phone', type: 'phone' }
];

export default class testComponent extends LightningElement {
    accounts = [];
    columns = COLUMNS;

    @wire(getAccounts)
    wiredAccounts({ error, data }) {
        if (error) {
            console.error(error);
            return;
        }
        if (data) {
            this.accounts = data;
        }
    }
}
```

Apex Controller (AccountController.cls):

```
public with sharing class AccountController {
    @AuraEnabled(cacheable=true)
    public static List<Account> getAccounts() {
        try {
            return [SELECT Name, Industry, Phone FROM Account];
        } catch (Exception ex) {
            System.debug(ex);
            throw new AuraHandledException(ex.getMessage());
        }
    }
}
```

CSS:

```
.THIS .slds-card {
    margin: 1rem;
}

.THIS .slds-card__header {
    background-color: #f5f5f5;
    border-bottom: 1px solid #d8dde6;
    font-size: 1.25rem;
    font-weight: bold;
    padding: 0.75rem 1rem;
}

.THIS .slds-table td, .THIS .slds-table th {
    padding: 0.75rem;
}
```

Obrázok 32 Ukážka Lightning Web komponentu

Existuje niekoľko dôvodov, prečo sa komponenty Lightning Web (LWC) považujú za lepšie ako komponenty Aura:

- **Výkon** - LWC sú vytvorené pomocou moderných webových štandardov, ako sú webové komponenty a moduly ES6, ktoré poskytujú lepší výkon ako komponenty Aura. LWC má rýchlejšie vykresľovanie a inicializáciu ako komponenty Aura, čo vedie k plynulejšiemu používateľskému zážitku.
- **Opätovná použiteľnosť** - Komponenty LWC sú navrhnuté tak, aby boli modulárnejšie ako komponenty Aura. To znamená, že vývojári môžu vytvárať zložitejšie komponenty kombinovaním menších komponentov, čím sa skracuje čas vývoja a zvyšuje udržiavateľnosť kódu.
- **Jednoduché učenie -** LWC používa štandardné webové technológie, ako sú HTML, CSS a JavaScript, čo vývojárom uľahčuje učenie a vytváranie komponentov. Vývojári s predchádzajúcimi skúsenosťami s vývojom webových aplikácií sa môžu rýchlo naučiť LWC a začať vytvárať komponenty.
- **Lepšie nástroje** - LWC má lepšiu podporu nástrojov ako komponenty Aura. LWC poskytuje súbor nástrojov na vývoj a testovanie komponentov, čo vývojárom uľahčuje vytváranie a nasadzovanie komponentov.
- **Kompatibilita** - LWC je postavený na moderných webových štandardoch, ktoré sú kompatibilné s najnovšími webovými technológiami a predpokladá sa, že táto kompatibilita pretrvá aj v budúcnosti. Komponenty Aura sú postavené na proprietárnom princípe a preto nemusia byť kompatibilné s webovými technológiami vyvinutými v budúcnosti. [47]

Celkovo Lightning Web Components poskytujú modernejší, modulárnejší a výkonnejší prístup k budovaniu webových komponentov ako komponenty Aura. Je však dôležité poznamenať, že komponenty Aura sa v ekosystéme Salesforce stále široko používajú a podporujú.

II. PRAKTICKÁ ČASŤ

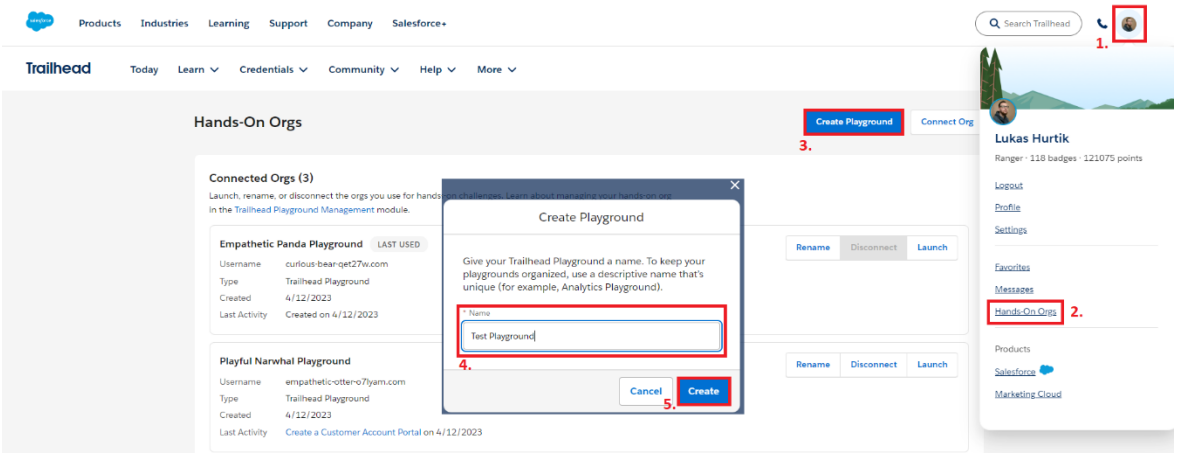
3 PRÍPRAVA SALESFORCE

Pre potreby praktickej časti tejto práce je potrebné vytvoriť novú inštanciu Salesforce a následne prepojiť danú inštanciu s vývojovým prostredím Visual Studio Code.

3.1 Tvorba Salesforce inštancie

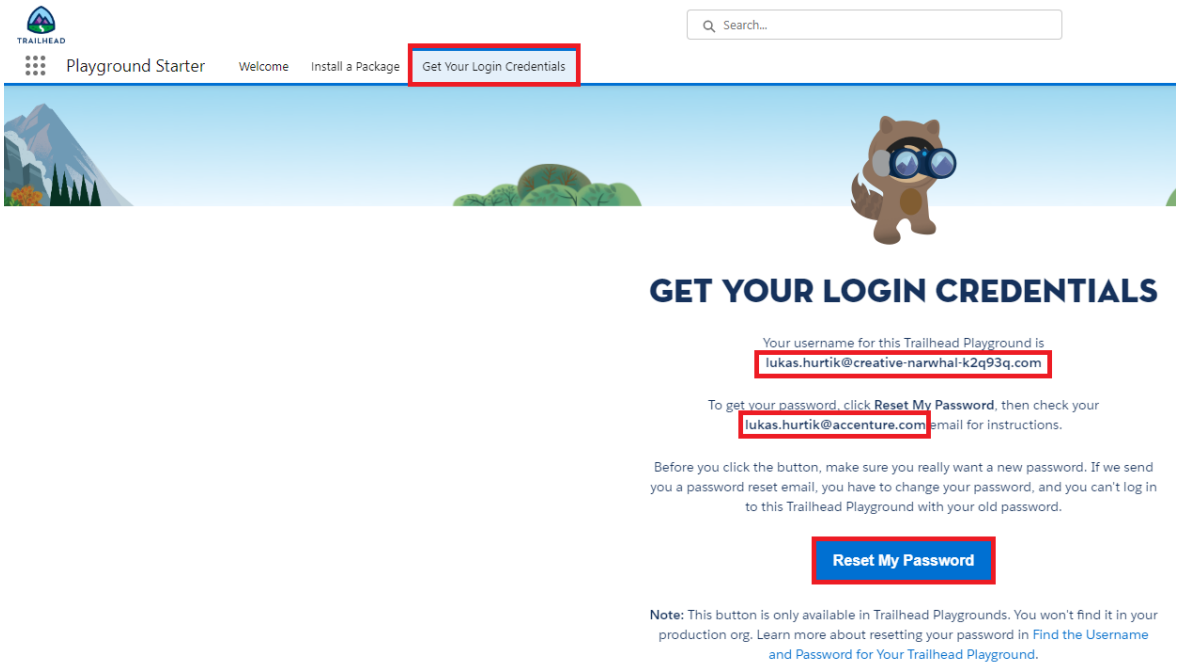
Salesforce ponúka túto možnosť pre nekomerčné účely zadarmo na webovom portáli s názvom Trailblazer. Pre registráciu na portáli Trailblazer je potrebná iba aktívna emailová adresa. Po úspešnej registrácii používateľ dostane prístup do komunitného fóra Trailblazer, kde Salesforce vývojári komunikujú medzi sebou a pomáhajú si navzájom pri riešení rôznych problémov spojených s vývojom aplikácií v Salesforce. Spolu s prístupom do komunitného fóra Trailblazer používateľ po registrácii získa aj prístup do portálu Trailhead. Trailhead je výukový portál od spoločnosti Salesforce, ktorý obsahuje široké spektrum výukových modulov, vhodných nie len pre vývojárov ale napríklad aj administrátorov alebo konzultantov. Používateľ vypracováva výukové moduly v tzv. Salesforce ihrisku, čo je bezplatná Salesforce inštancia pre nekomerčné účely. Každý používateľ môže mať v jeden moment vytvorených až 10 Salesforce inštancií.

Nižšie na obrázku č. 33 je znázornený postup vytvárania Salesforce inštancie. Používateľ si otvorí stránku na ktorej sú zobrazené všetky inštancie vytvorené pre jeho profil, klikne na tlačidlo Vytvoriť ihrisko, uvedie želaný názov inštancie Salesforce a následne klikne na tlačidlo *Create*. Tvorba inštancie Saleforce trvá niekoľko minút, po vytvorení sa inštancia zobrazí v zozname všetkých inštancií vytvorených pre profil používateľa. Po kliknutí na tlačidlo *Launch* sa spustí daná inštancia Salesforce.



Obrázok 33 Vytváranie inštancie Salesforce

Po spustení Salesforce inštancie používateľ uvidí uvítaciu obrazovku. Následne je potrebné sa prekliknúť na obrazovku s názvom *Get Your Login Credential* a kliknúť na tlačidlo *Reset My Password*. Po kliknutí na tlačidlo, sa používateľovi odošle email s odkazom na resetovanie hesla. Po kliknutí na odkaz v emaily, si používateľ zvolí nové heslo. Heslo a používateľské meno je potrebné pre prihlásenie sa do Salesforce inštancie.

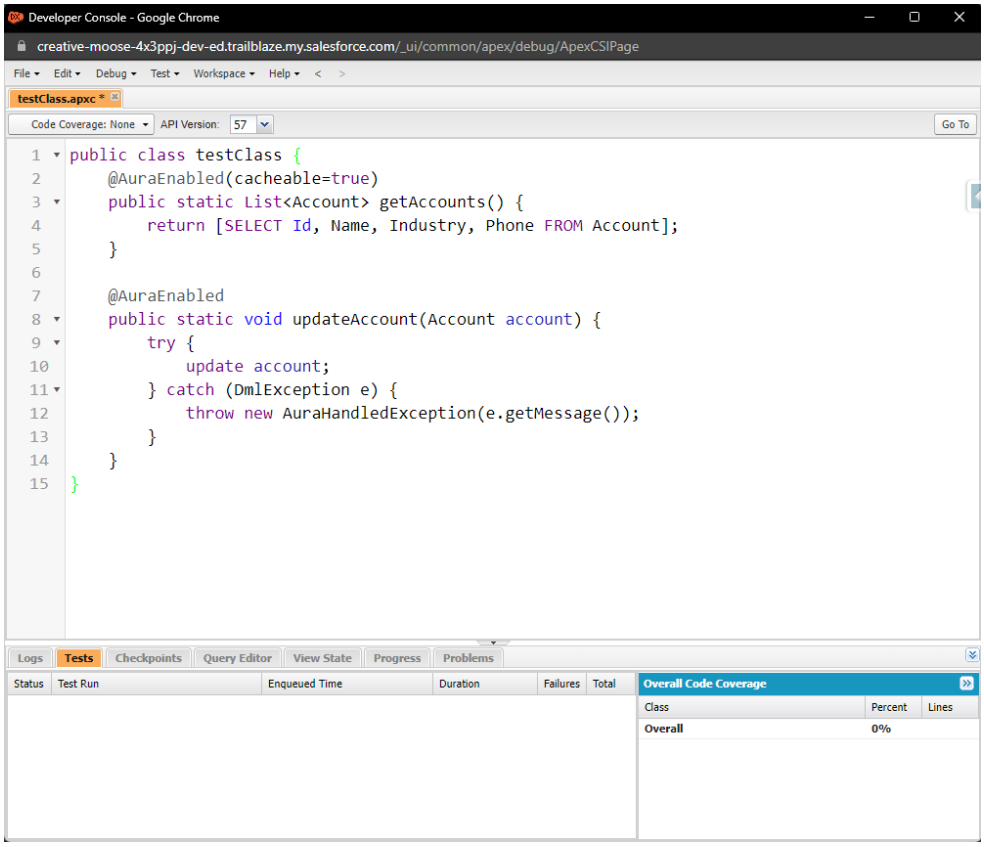


Obrázok 34 Okno Salesforce inštancie pre resetovanie hesla

3.2 Prepojenie s vývojovým prostredím

Vývoj aplikácií programovacím spôsobom v Salesforce je možný buď priamo v inštancii Salesforce, použitím nástroja konzola vývojára (Developer console) alebo prepojením inštancie Salesforce s externým vývojovým prostredím (napr. Visual Studio Code).

Konzola vývojárov v Salesforce je integrované vývojové prostredie (IDE), ktoré umožňuje vývojárom písať, testovať a ladiť kód v prostredí Salesforce. Vývojárom poskytuje webové rozhranie na písanie, vykonávanie a testovanie kódu Apex, stránok Visualforce a komponentov Lightning.



Obrázok 35 Konzola vývojárov

Vývojári Salesforce avšak namiesto konzoly vývojára z rôznych dôvodov čoraz častejšie používajú externé integrované vývojové prostredia (IDE), ako napríklad Visual Studio Code. Externé integrované vývojové prostredia majú oproti konzole vývojárov niekoľko výhod, medzi ktoré patria napríklad:

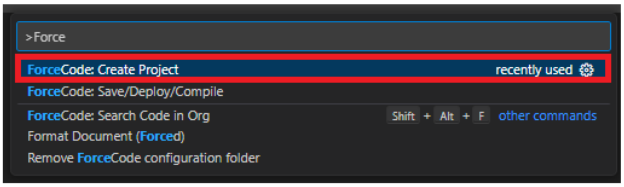
- Externé integrované vývojové prostredia ponúkajú robustnejšie a prispôsobiteľnejšie vývojové prostredie ako konzola vývojárov. Vývojári si môžu prostredie prispôsobiť pomocou rozšírení a prídavných modulov s cieľom zvýšiť produktivitu.
- Externé integrované vývojové prostredia poskytujú lepšie možnosti kontroly verzií a spolupráce. S VS Code môžu vývojári používať Git na kontrolu verzií, spolupracovať s inými vývojármi pomocou nástrojov, ako je napríklad GitHub.
- Externé integrované vývojové prostredia sú efektívnejšie pre väčšie projekty so zložitými kódovými základňami. Konzola pre vývojárov sa pri veľkých projektoch môže stať pomalou a ťažkopádnou, zatiaľ čo externé IDE môžu ponúknuť lepší výkon a spracovanie veľkých kódových báz.
- Používanie externého IDE, ako je VS Code, poskytuje štandardizovanejšie vývojové prostredie v rôznych tímoch a organizáciách. To môže pomôcť presadzovať

osvedčené postupy, zabezpečiť konzistentnosť a zjednodušiť proces nástupu nových vývojárov.

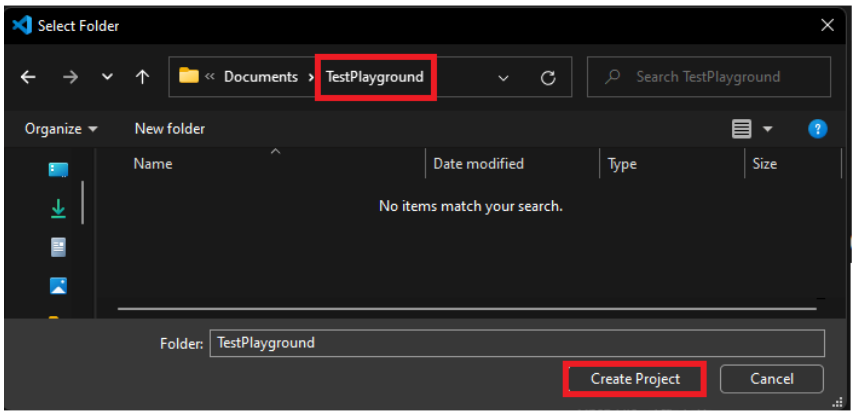
Vzhľadom k vyššie spomenutým nevýhodám konzoly vývojára, v tejto práci bude pre vývoj aplikácií v Salesforce použité externé integrované vývojové prostredie Visual Studio Code.

Visual Studio Code, často označované skratkou VS Code, je bezplatný open-source editor kódu vyvinutý spoločnosťou Microsoft. Je navrhnutý ako ľahký a prispôsobiteľný nástroj na programovanie, ladenie a vytváranie softvérových aplikácií pre širokú škálu platforiem a programovacích jazykov. VS Code podporuje mnoho programovacích jazykov vrátane jazykov JavaScript, Python, Java, C++ a mnohých ďalších. Obsahuje funkcie, ako je zvýrazňovanie syntaxe, dokončovanie kódu a nástroje na ladenie. K dispozícii má aj veľké množstvo rozšírení a prídavných modulov, ktoré možno použiť na prispôsobenie editora a pridanie ďalších funkcií.

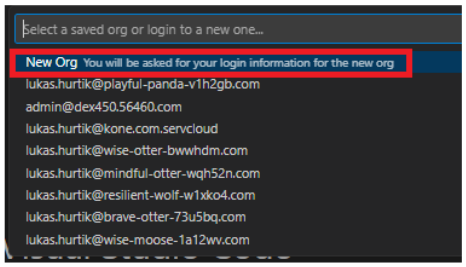
Stiahnutie a proces inštalácie Visual Studio Code je pomerne jednoduchý, a preto v tejto práci nebude podrobne opísaný. Po prvom spustení je potrebné nainštalovať dve doplňujúce rozšírenia, ktoré umožňujú prepojenie VS Code a Salesforce. Rozšírenie, ktoré je potrebné nainštalovať ako prvé sa nazýva Salesforce Extension Pack (Expanded). Po úspešnej inštalácii je potrebné nainštalovať druhé rozšírenie, s názvom ForceCode. Po dokončení inštalácie ForceCode je potrebné reštartovať VS Code. Po opätovnom spustení VS Code a po načítaní rozšírení, stlačením kombinácie kláves CRTl+SHIFT+P sa otvorí paleta príkazov (command palette). Pre vytvorenie nového projektu je potrebné zadať do palety príkazov príkaz *ForceCode: Create Project*. Po zadaní príkazu sa automaticky otvorí vyskakovacie okno, v ktorom je potrebné určiť miesto kde bude nový projekt uložený. Pre každý nový projekt sa odporúča vytvoriť novú zložku. Po vybraní zložky sa kliknutím na tlačidlo *Create Project* vytvorí nový projekt. Predošlé okno VS Code sa automaticky zatvorí a otvorí sa nové, už v rámci novo vytvoreného projektu. Po opätovnom načítaní rozšírení sa automaticky otvorí okno v rámci VS Code, v ktorom je potrebné vybrať Salesforce inštanciu, ktorá bude prepojená s novo vytvoreným projektom vo VS Code. V tomto prípade je potrebné vybrať možnosť *New Org*, čo znamená že bude vybraná nová inštancia Salesforce, ktorá ešte v minulosti na tomto počítači nebola prepojená s VS Code prostredím. Doterajší postup pri prepájaní inštancie Salesforce a VS Code je znázornený na obrázku č. 36.



Paleta príkazov



Vytvorenie zložky pre nový projekt

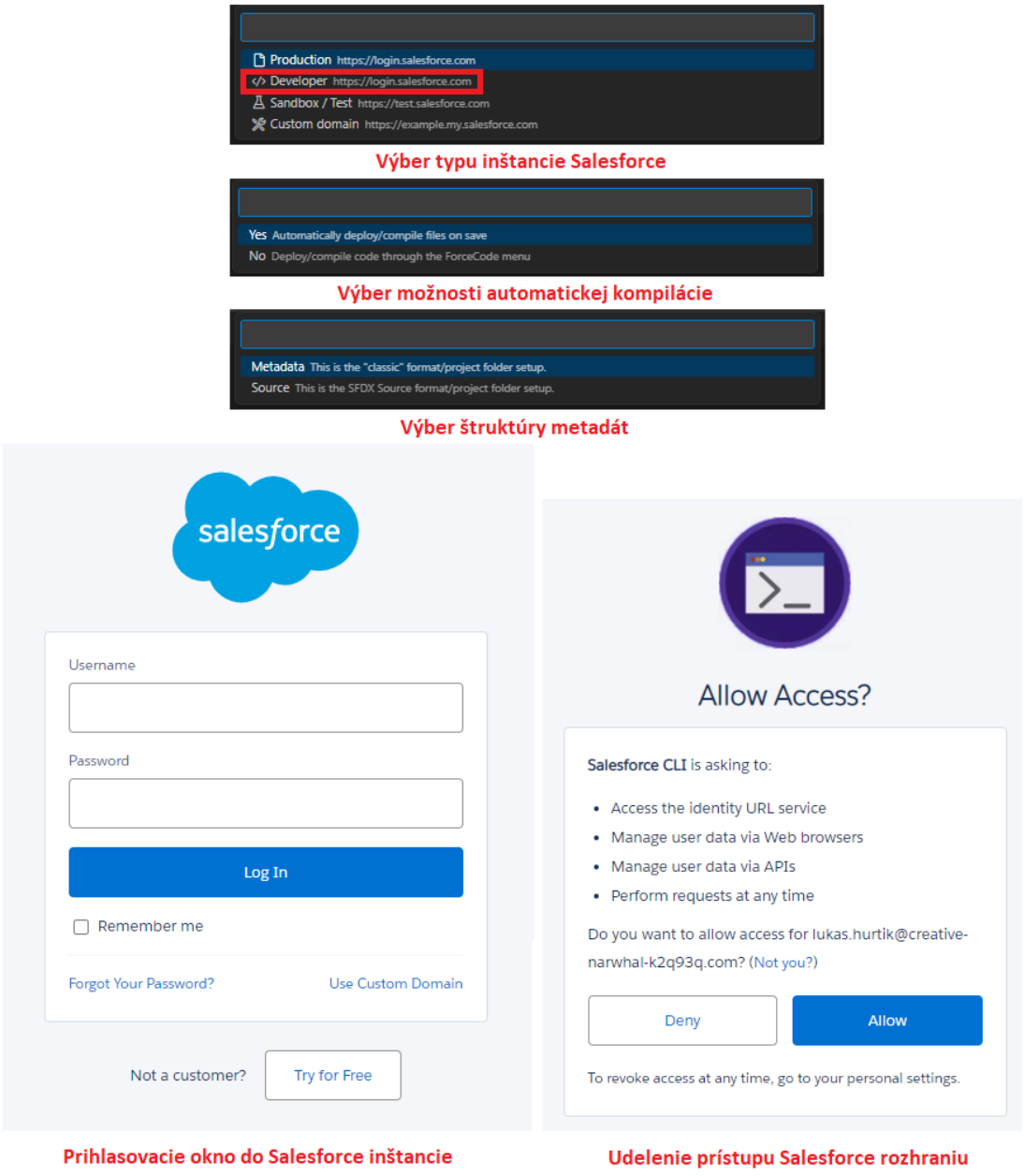


Výber Salesforce inštancie

Obrázok 36 Prepojenie inštancie Salesforce a VS Code - časť prvá

Po zvolení možnosti *New Org*, je potrebné vybrať typ inštancie Salesforce. V tomto prípade je najlepšie zvoliť možnosť *Developer*. Následne je potrebné vybrať štruktúru metadát v projekte. Na výber sú možnosti *Metadata* a *Source*. Najväčším rozdielom týchto dvoch štruktúr je štruktúra metadát objektov. Kým pri *Metadata* štruktúre má objekt jeden súbor v ktorom sú uložené všetky informácie o ňom (polia, typy záznamov, validačné pravidlá a pod.), v *Source* štruktúre má objekt niekoľko podsúborov, takže napríklad polia objektu sú uložené v jednom podsúbore, typy záznamov v ďalšom podsúbore a validačné pravidlá tiež v oddelenom podsúbore. Nedá sa jednoznačne určiť, ktorá štruktúra je lepšia a preto si štruktúru vyberá vývojár sám podľa svojej preferencie. Avšak, odporúča sa vybrať štruktúra, ktorá je použitá aj v git repozitári pretože to jednoznačne zjednoduší a urýchli proces pridávania novej logiky do git repozitára. V tomto prípade bola zvolená štruktúra *Metadata*. Po zvolení štruktúry metadát sa automaticky otvorí okno v predvolenom internetovom prehliadači, v ktorom je potrebné sa prihlásiť do Salesforce inštancie. Po prihlásení je ešte potrebné udeliť povolenia Salesforce rozhraniu vykonávať požiadavky a získavať používateľské dáta.

Po odsúhlasení prístupu sa otvorí Salesforce inštancia a proces jej prepojenia s VS Code je dokončený. Postup prepojenia VS Code a Salesforce inštancie od výberu typu Salesforce inštancie až po udelenie povolenia Salesforce rozhraniu je znázornený na obrázku č. 37.



Obrázok 37 Prepojenie inštancie Salesforce a VS Code - časť druhá

Následne je vhodné si stiahnuť všetky metadáta zo Salesforce inštancie do VS Code. Vo VS Code v ľavom dolnom rohu sa nachádza tlačidlo s nápisom *ForceCode Menu*. Stlačením tohto tlačidla sa otvorí paleta príkazov dostupných pre ForcCode. Pre stiahnutie všetkých metadát z inštancie Salesforce je potrebné zadať príkaz *Retrieve Package/Metadata* a následne zvoliť možnosť *Get All Files from org*. Po dokončení sťahovania metadát je vývojové prostredie pripravené na vývoj novej funkcionality.

4 NÁVRH A REALIZÁCIA APLIKÁCII V SYSTÉME SALESFORCE

V tejto časti práce bude uvedených 5 príkladov aplikácií vyvinutých v systéme Salesforce.

4.1 Príklad číslo 1

V príklade číslo 1 bude navrhnuté riešenie vyvinuté pomocou deklaratívneho ale aj programovacieho spôsobu vývoja aplikácií v Salesforce.

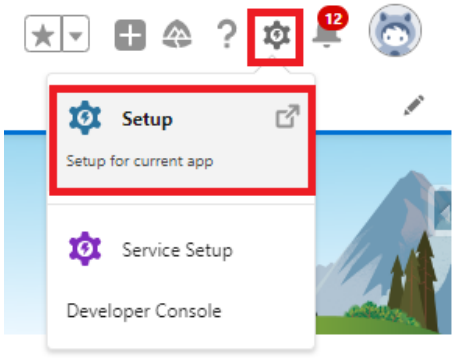
4.1.1 Zadanie

Škola chce implementovať do svojho školského systému novú funkcionalitu. Po ohodnotení testu učiteľom, príde študentovi emailová notifikácia s výsledkom testu. Jeden učiteľ môže ohodnotiť niekoľko testov, študent môže absolvovať niekoľko testov, avšak konkrétny test musí byť napísaný jedným žiakom a ohodnotený jedným učiteľom. Každý používateľ bude môcť upravovať iba záznamy ktoré sám vytvoril.

4.1.2 Dátový model

Pre splnenie požiadaviek uvedených v zadaní je najvhodnejšie vytvoriť tri špecifické objekty na mieru: objekt učiteľ, objekt žiak a objekt test. Objekt test bude spojovací objekt medzi objektami učiteľ a žiak. V Salesforce je spojovací objekt objektom, ktorý slúži na vytvorenie vzťahu mnoho k mnohým (many to many) medzi dvoma inými objektmi. Spojovací objekt obsahuje dva vzťahy master-detail, jeden ku každému z objektov, ktoré spája.

Pre vytvorenie objektu je potrebné otvoriť si inštanciu Salesforce, kliknúť v pravom hornom rohu na tretiu ikonu z prava a následne na možnosť *Setup* (viď. Obrázok číslo 38).



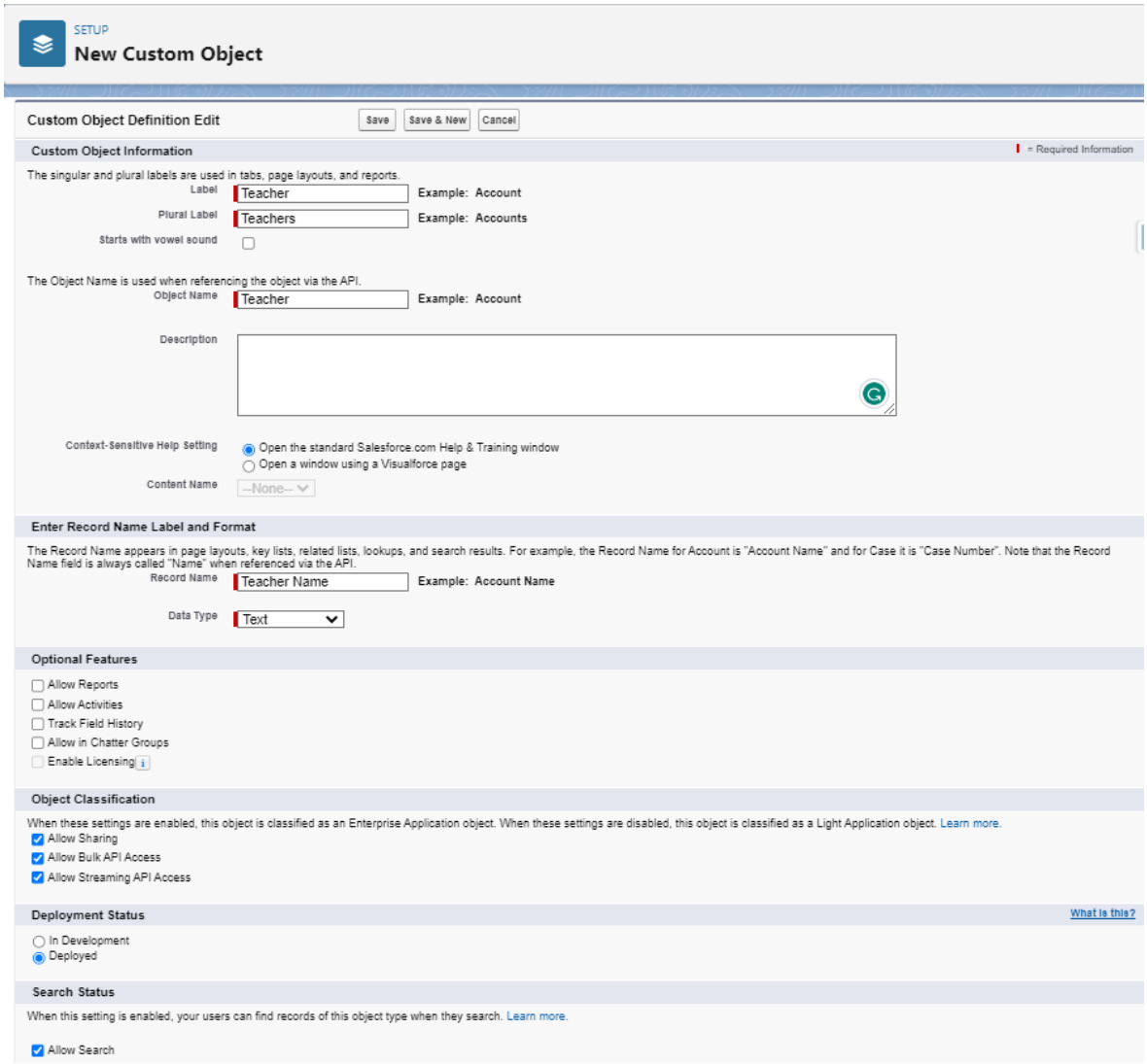
Obrázok 38 Nastavenia v inštancii Salesforce

Následne sa na novej karte v internetovom prehliadači otvorí stránka nastavení (Setup). Pre vytvorenie nového objektu je potrebné kliknúť na záložku *Object Manager*, t. j. objektový

manažér. V objektovom manažéri je zoznam všetkých objektov, po kliknutí na niektorý objekt sa zobrazia informácie týkajúce sa daného objektu (napríklad zoznam polí, validačných pravidiel a pod.). Následne je potrebné kliknúť na tlačidlo *Create* a vybrať možnosť *Custom Object*. Ako prvý bude vytvorený objekt Učiteľ, po anglicky Teacher. Informácie o objekte budú vyplnené nasledovne:

- Label: Teacher
- Plural Label: Teachers
- Allow Search: checked

Všetky ostatné informácie budú ponechané nezmenené, podľa predvoleného nastavenia.

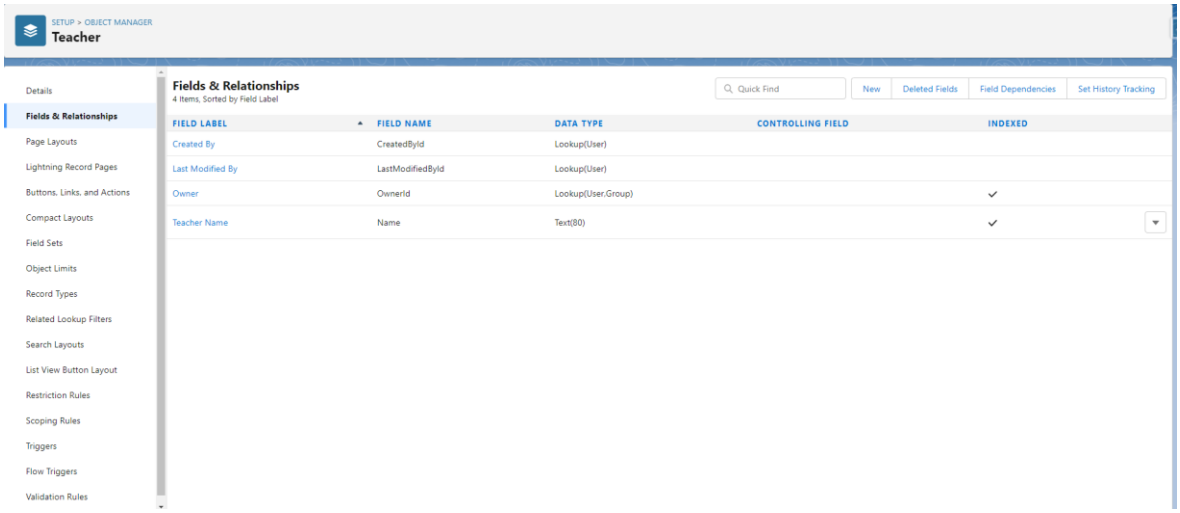


Obrázok 39 Stránka vytvárania objektu

Po uložení sa zobrazí stránka novo vytvoreného objektu, na ktorej môže používateľ vidieť všetky informácie o objekte. Napríklad po kliknutí na záložku *Fields & Relationships* uvidí

používateľ zoznam všetkých polí vytvorených pre daný objekt. Salesforce pri vytvorení objektu vytvorí automaticky štyri systémové polia s názvami:

- Created By - Toto pole odkazuje na používateľa, ktorý vytvoril záznam.
- Last Modified By - Toto pole odkazuje na používateľa, ktorý ako posledný upravoval záznam.
- Owner - Toto pole odkazuje na používateľa alebo skupinu, ktorý vlastní daný záznam.
- Teacher Name - Toto pole reprezentuje názov záznamu, v tomto prípade meno učiteľa



Obrázok 40 Stránka objektu Teacher - záložka Polia a vzťahy

Obdobne je potrebné vytvoriť aj dva ďalšie objekty: Test a Žiak. Informácie o objektoch budú vyplnené nasledovne:

- Objekt Test
    a. Label: Test
    b. Plural Label: Tests
    c. Data Type: Auto Number
    d. Display Format: TEST-{0000}
    e. Starting Number: 1
    f. Allow Search: checked
- Objekt Žiak
    a. Label: Student
    b. Plural Label: Students

c. Allow Search: checked

Je potrebné tiež určiť, ktoré profily budú mať prístup na novo vytvorené objekty. Profil System Administrator má automaticky prístup na novo vytvorené objekty, pretože má nastavené administrátorské povolenie *View all* (vidieť všetko) a tiež *Modify all* (modifikovať všetko). Odporúča sa používateľom nepriraďovať administrátorský profil, preto bude vytvorený nový používateľský profil. V nastaveniach sa vyhľadaním slova *Profiles* otvorí zoznam všetkých profilov. Je potrebné otvoriť si profil s názvom Standard User a kliknúť na tlačidlo *Clone* (klonovať). Ako meno nového profilu bude určené Test User.

Následne je potrebné vytvoriť na objekte Test dva vzťahy master-detail, jeden k objektu Učiteľ a druhý k objektu Žiak. Na stránke objektu Test sa kliknutím na záložku *Fields & Relationships* zobrazí zoznam polí a vzťahov. Pre vytvorenie vzťahu je potrebné kliknúť na tlačidlo *New* a vybrať možnosť *Master-Detail Relationship*. Následne je potrebné zvoliť objekt ku ktorému bude vytvorený vzťah z objektu Test, je potrebné vybrať možnosť Žiak a kliknúť na tlačidlo Uložiť. Obdobne bude vytvorený aj vzťah k objektu Učiteľ.

Na novo vytvorených objektoch budú vytvorené aj ďalšie polia uvedené v tabuľke nižšie.

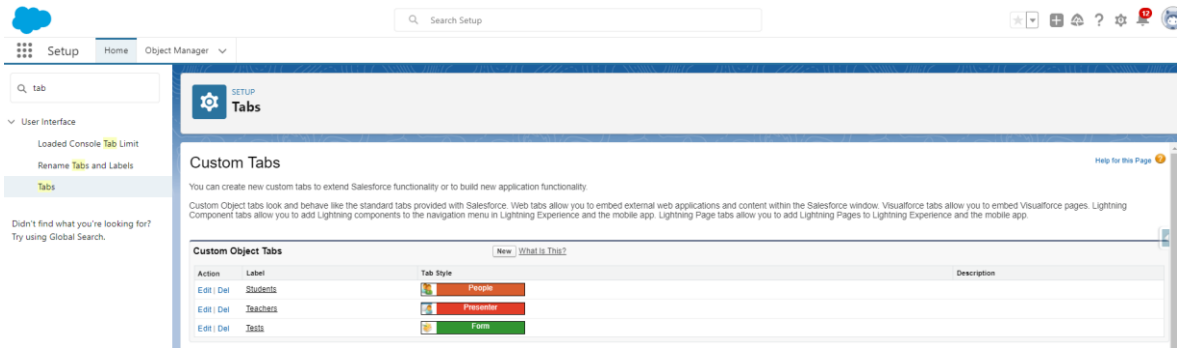
Tabuľka 2 Polia vytvorené pre objekty Učiteľ, Žiak, Test

Objekt	**Názov poľa (Field label)**	**Typ poľa (Field Type)**
Test	Grade	Text(2)
Test	Comment	Text Area
Test	Status	Picklist (Graded, Ungraded)
Teacher	Faculty	Picklist
Teacher	Amount of ungraded	Roll-Up Summary
Student	Faculty	Picklist
Student	Email address	Email

Prístup na novo vytvorené polia bol udelený tak, že profil System Administrator a Test User majú pre dané polia prístup čítaj a upravuj (Read+Edit). Pole *Email address* bolo označené ako povinné, to znamená že každý jeden záznam objektu Študent musí mať toto pole vyplnené, inak záznam nebude uložený do databázy. Pole *Amount of ungraded* je Roll-Up Summary pole, ktoré zobrazuje počet testov daného učiteľa, ktoré ešte nie sú ohodnotené. Pole typu *Roll-Up Summary* je podrobnejšie opísané v teoretickej časti tejto práce.

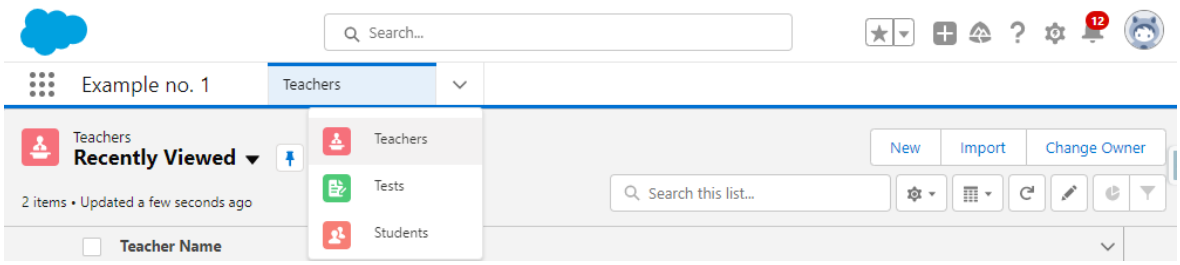
Aby používateľ mohol s novo vytvorenými objektami pohodlne pracovať je potrebné vytvoriť najprv záložky pre novo vytvorené objekty a potom aplikáciu. Záložky pre objekty je možné vytvoriť v nastaveniach, stačí do vyhľadávača v ľavej časti obrazovky nastavení vyhľadať slovo *Tabs*. Kliknutím na možnosť *Tabs* sa otvorí obrazovka záložiek. V sekcii

Custom Object Tabs, teda záložky špecifických objektov na mieru je potrebné kliknúť na tlačidlo *New*. Následne je nutné vybrať objekt pre ktorý bude vytvorená záložka, teda napríklad Žiak. Okrem objektu treba vybrať aj štýl záložky, pre objekt Žiak sa hodí štýl People avšak je možné vytvoriť aj vlastný štýl. Kliknutím na tlačidlo Uložiť sa vytvorí záložka objektu. Obdobne budú vytvorené záložky aj pre objekty Učiteľ a Test (viď obrázok č. 41).



Obrázok 41 Záložky pre novo vytvorené objekty

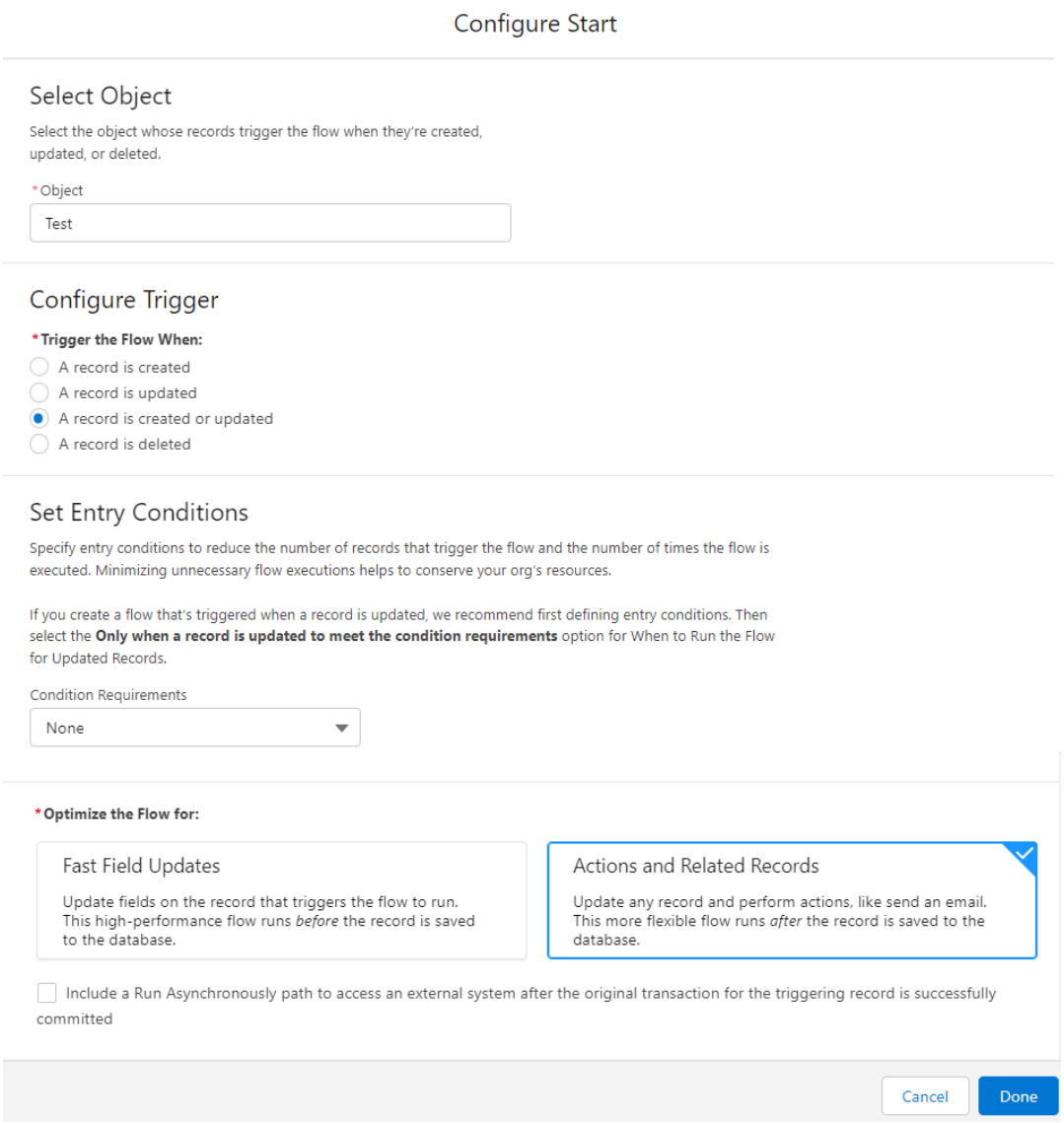
Pre vytvorenie aplikácie je potrebné vyhľadať v nastaveniach spojenie *App Manager* a kliknúť na tlačidlo *New Lightning App*. Názov aplikácie bude Example no. 1, čiže príklad číslo 1. V sekcii *Navigation Style* je potrebné zvoliť možnosť *Console navigation*, konzolová navigácia je modernejšia a poskytuje používateľovi väčší prehľad aj pri viacerých otvorených záznamoch súčasne. Následne je potrebné vybrať, ktoré záložky objektov budú dostupné v danej aplikácii, teda budú to objekty Učiteľ, Žiak a Test. Posledným krokom pri vytváraní aplikácie je určenie profilov pre ktoré bude aplikácia dostupná. Je potrebné vybrať profily Test User a System Administrator. Kliknutím na uložiť sa aplikácia vytvorí a je pripravená na použitie. Aplikácia sa otvára kliknutím na ikonu 9 bodiek v ľavom rohu obrazovky. Táto ikona sa nazýva *App launcher*, takže v preklade spúšťač aplikácií. Kliknutím na ikonu sa zobrazí vyhľadávacie pole, do ktorého je potrebné zadať názov aplikácie, teda Example no. 1.



Obrázok 42 Aplikácia Example no. 1

4.1.3 Deklaratívny spôsob vývoja aplikácie

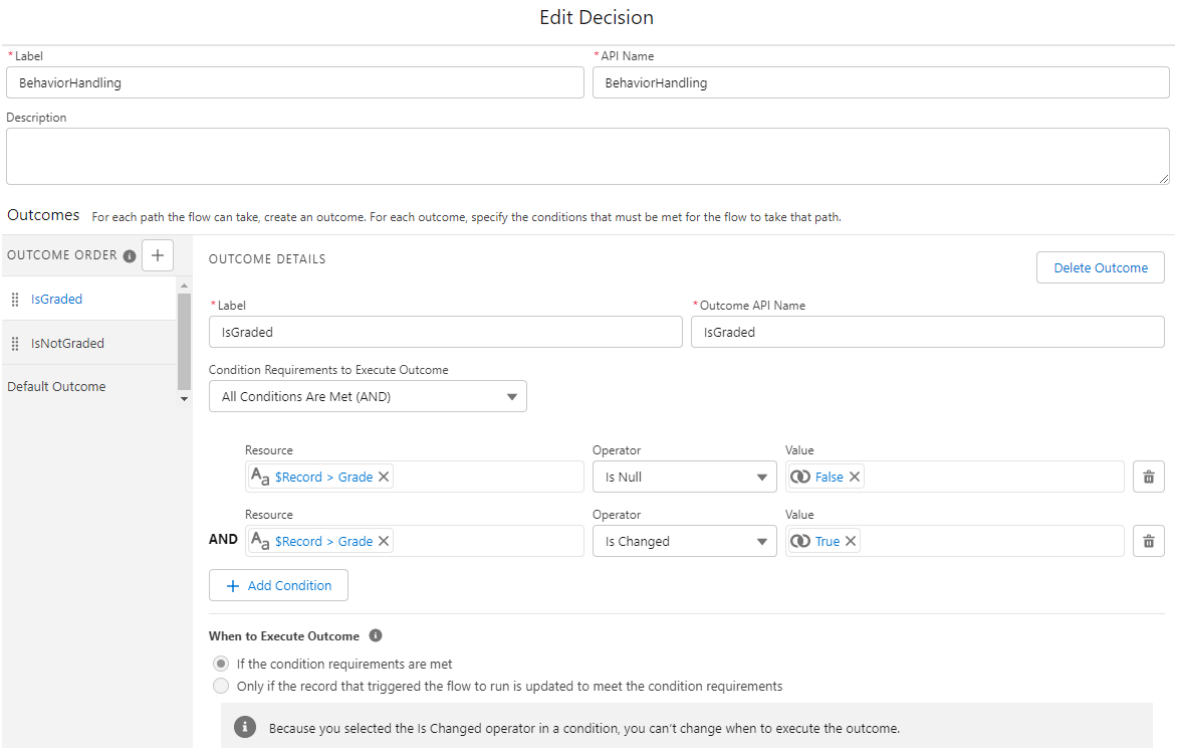
Vývoj aplikácie pre príklad č. 1 bude prebiehať použitím nástroja Flow Builder. V nastaveniach, vyhľadaním slova Flows, sa zobrazí zoznam všetkých flowov. Kliknutím na tlačidlo *New Flow* v pravej hornej časti obrazovky sa otvorí sprievodca vytváraním flowu. Prvým krokom je výber typu flowu. V návrhu príkladu č. 1 je uvedené, že emailová notifikácia sa má odoslať po tom ako učiteľ ohodnotí test, takže ako typ flowu je potrebné vybrať *Record-Triggered Flow*, čiže Flow spustený záznamom. Pre potvrdenie výberu je potrebné kliknúť na tlačidlo *Create* (vytvoriť). Následne je potrebné určiť objekt, ktorého zmena záznamu spustí flow, v tomto prípade je to objekt Test. Tiež je potrebné určiť kedy sa má spustiť flow. Pre tento príklad je najvhodnejšie vybrať možnosť *A record is created or updated*. Na obrázku č. 43 je uvedená obrazovka vytvárania flowu.



Obrázok 43 Obrazovka vytvárania flowu

Po kliknutí na tlačidlo *Done*, sa zobrazí pracovná plocha flowu. Kliknutím na ikonu +, ktorá je umiestnená medzi elementami *Start* a *End*, je možné pridať nový element. Pri spustení flowu bude odoslaná emailová notifikácia a zároveň bude status testu zmenený na *Graded*. Zároveň ak sa hodnotenie z testu zmazané (napríklad z dôvodu chyby učiteľa), status sa zmení naspäť na *Ungraded*.

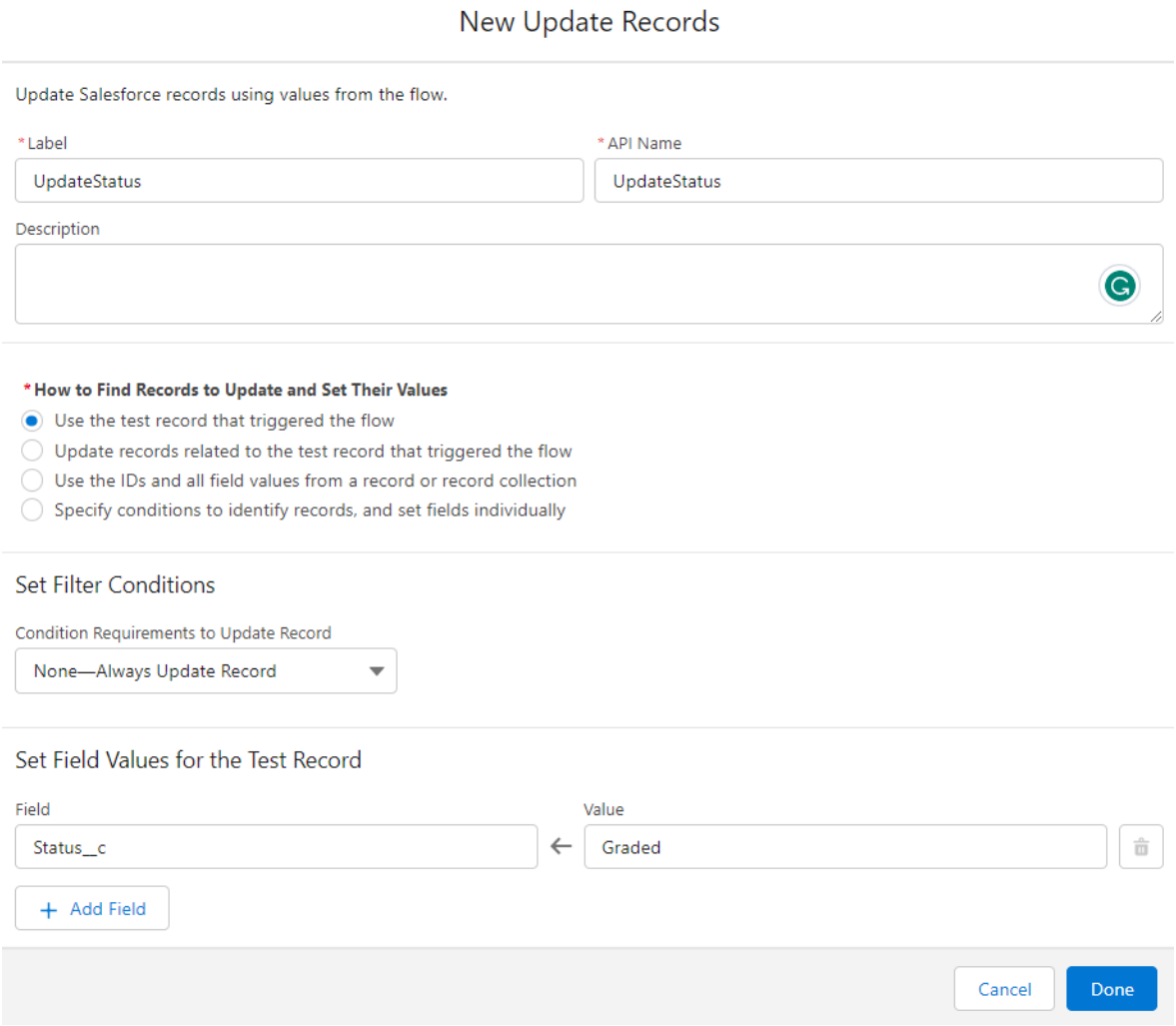
Ako prvý element bude pridaný element *Decision* (rozhodnutie). Názov nového elementu bude *BehaviorHandling*. Na ľavej strane obrazovky elementu sa nachádza zoznam tzv. *Outcomes*. Pre každú cestu, ktorou sa flow môže uberať, by mal byť vytvorený *Outcome*. Pre každý *Outcome* je potrebné uviesť podmienky, ktoré musia byť splnené, aby sa flow vybral danou cestou. *Outcome* je vlastne rozhodovacia vetva. Prvá rozhodovacia vetva bude nazvaná *IsGraded*, a bude obsahovať podmienku, ktorá sa vyhodnotí pravdivo ak pole *Grade* nebude prázdne, ale zároveň jeho hodnota bude zmenená. Druhá rozhodovacia vetva bude nazvaná *IsNotGraded*, a bude obsahovať podmienku, ktorá sa vyhodnotí pravdivo ak pole *Grade* bude prázdne, ale zároveň jeho hodnota bude zmenená.



Obrázok 44 Element Decision, rozhodovacia vetva IsGraded

Po uložení zmien v elemente *Decision*, sa na obrazovke objavili tri čiary, ktoré vychádzajú z elementu *Decision* a reprezentujú rozhodovacie vetvy.

Následne je potrebné pridať element *Update Records* (Aktualizuj záznamy) v rozhodovacej vetve *IsGraded*. Názov elementu bude *UpdateStatusToGraded*, a vnútri elementu je potrebné určiť, že bude aktualizované pole *Status* na hodnotu *Graded* (viď Obrázok č. 45).



Obrázok 45 Element Update Records

Obdobne bude vytvorený aj ďalší element *Update Records*, tento krát v rozhodovacej vetve *IsNotGraded*. Názov elementu bude *UpdateStatusToUngraded*, a vnútri elementu je potrebné určiť, že bude aktualizované pole *Status* na hodnotu *Ungraded*. Zároveň bude aktualizované aj pole *Comment*, do ktorého bude vložený prázdny textový reťazec tzn. jeho predošlá hodnota bude zmazaná.

Pred vytvorením posledného elementu je potrebné vytvoriť tri tzv. *resources*, čo sú vlastné premenné. Tieto premenné budú použité pre posielanie emailovej notifikácie. V ľavej hornej časti obrazovky sa po kliknutí na tlačidlo *Toggle Toolbox* zobrazí *Toolbox*, čiže sada nástrojov. Po otvorení sady nástrojov je potrebné kliknúť na tlačidlo *New Resource*. Typ prvej

premennej bude *Formula*, čo znamená v preklade vzorec. Názov prvej premennej bude *Comment* a dátový typ bude Text. Do poľa *Formula* bude vložený tento reťazec: *IF(ISBLANK({!$Record.Comment__c}),"","Teacher also wrote a comment to your test:"&BR()&{!$Record.Comment__c&BR()})*. Táto podmienka vyhodnocuje či je pole *Comment* na zázname objektu Test prázdne alebo nie. V prípade ak je prázdne, bude hodnota tejto premennej prázdny reťazec. V prípade ak nie je prázdne, bude sa bude hodnota tejto premennej skladať zo štyroch častí:

1. Časť - Textový reťazec *Teacher also wrote a comment to your test:*, čo v preklade znamená Učiteľ tiež napísal komentár k vášmu testu.
2. a 4. Časť - *BR()*, čo je funkcia v Salesforce ktorá vkladá do textu zlom riadku.
3. Časť - *{!$Record.Comment__c}*, čo je odkaz na hodnotu poľa *Comment* na zázname objektu Test

Všetky štyri časti sú spojené pomocou ampersandu, ktorý slúži na spájanie textových reťazcov v Salesforce.

Typ druhej premennej bude *Text template*, čo znamená v preklade textová šablóna. Názov druhej premennej bude *EmailBody* a do poľa Body bude uložený tento textový reťazec: *Dear student {!$Record.Student__r.Name},*

Test {!$Record.Name} was graded by teacher {!$Record.Teacher__r.Name}.

Your grade is {!$Record.Grade__c}.

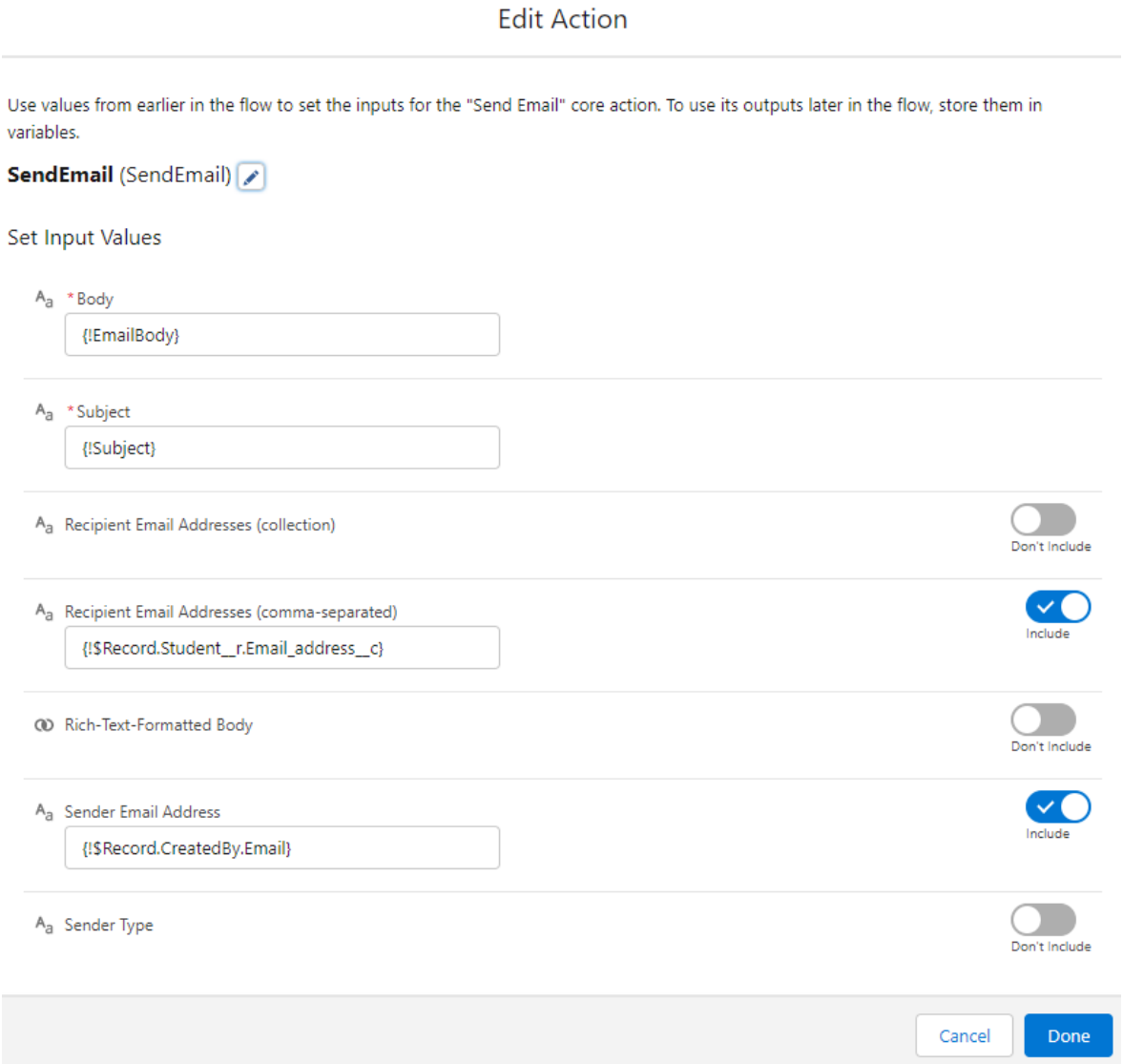
{!Comment}

Have a nice day!

{!$Record.Student__r.Name}* je odkaz na hodnotu poľa *Name* na zázname objektu *Student*, teda Žiak, ktorý je prepojený so záznamom objektu Test vzťahom master-detail. Obdobne *{!$Record.Name} je odkaz na hodnotu poľa *Name* na zázname objektu *Test* a *{!$Record.Teacher__r.Name}* je odkaz na hodnotu poľa *Name* na zázname objektu *Teacher*, teda Učiteľ, ktorý je prepojený so záznamom objektu Test vzťahom master-detail. *{!$Record.Grade__c}* je odkaz na hodnotu poľa *Grade* na zázname objektu *Test*. *{!Comment}* je odkaz na premennú *Comment*.

Typ tretej premennej bude *Text template*, jej názov bude *Subject* a do poľa Body bude uložený tento textový reťazec: *Test {!$Record.Name} was graded!*.

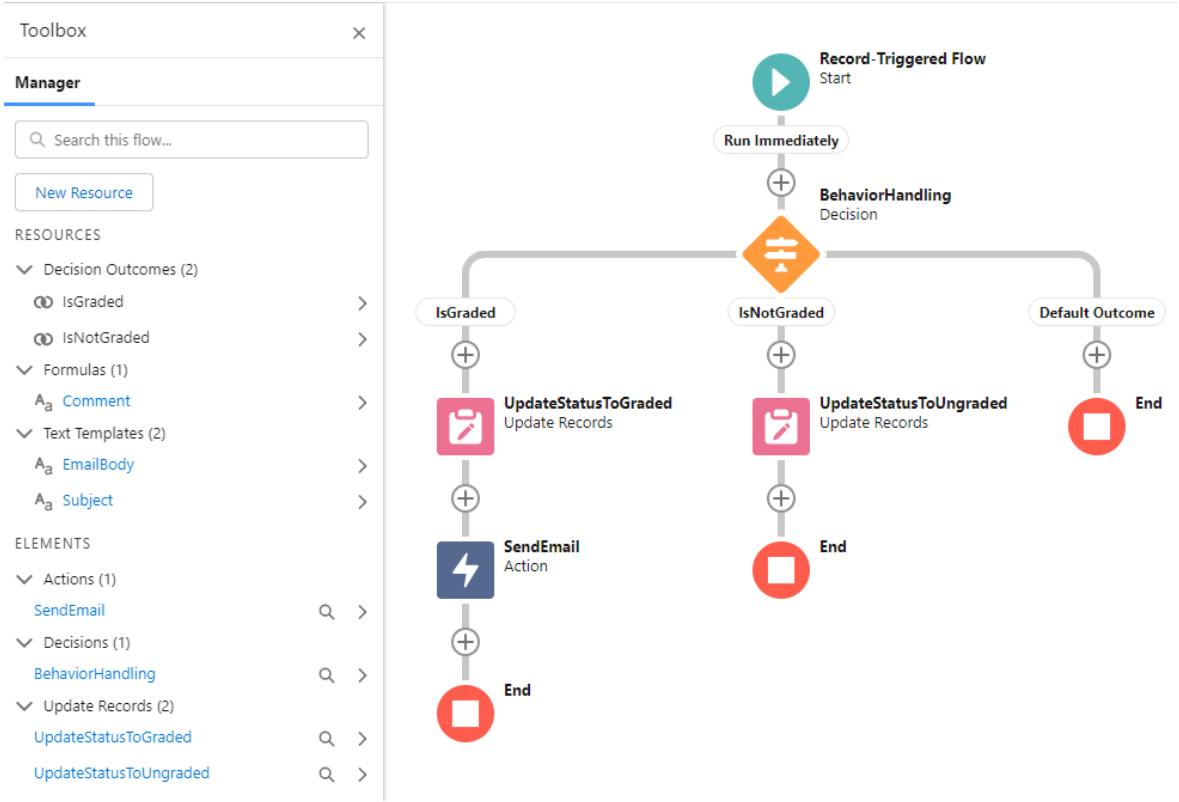
Po vytvorení potrebných premenných je potrebné pokračovať vytvorením posledného elementu, ktorý bude typu *Action - SendEmail.* Názov elementu bude *SendEmail*. Do poľa *Body* bude priradená premenná *EmailBody*, do poľa *Subject* zase premenná *Subject*. Do poľa *Recipient Email Addresses (comma-separated)* bude vložený odkaz na hodnotu poľa *Email* na zázname objektu *Student*, teda Žiak, ktorý je prepojený so záznamom objektu Test vzťahom master-detail. Do poľa *Sender Email Address* bude vložený odkaz na emailovú adresu používateľa, ktorý vytvoril daný záznam objektu Test (viď obrázok č. 46).



Obrázok 46 Element typu Action – SendEmail

Tým je vývoj flowu ukončený, je potrebné ho ešte uložiť a aktivovať. Pri ukladaní je potrebné zadať názov flowu, ktorý bude v tomto prípade *Example1Flow*. V nástroji *Flow Builder* je tiež vstavaný nástroj na ladenie flowu. Obrazovka tohto nástroja sa otvorí po kliknutí na tlačidlo *Debug*. Po otvorení obrazovky je možné nakonfigurovať podmienky pri

ktorých bude flow ladený. Je možné určiť či majú byť vstupné podmienky pri ladení preskočené alebo nie, či sa má flow správať ako keby bol záznam vytvorený alebo aktualizovaný a tiež je možné určiť záznam objektu Test pre ktorý bude spustené ladenie flowu. Tento nástroj je veľmi užitočný hlavne ak flow obsahuje viacero rozhodovacích vetiev. Vizualizácia finálnej verzie flowu spolu s premennými je znázornená na obrázku č. 47.



Obrázok 47 Príklad č.1 vypracovaný pomocou nástroja Flow Builder

4.1.4 Programovací spôsob vývoja aplikácie

V príklade č. 1 je definované, že emailová notifikácia sa má odoslať študentovi po tom ako učiteľ ohodnotí test. Ohodnotenie testu je vykonávané aktualizáciou záznamu. Preto najvhodnejším spôsobom ako vytvoriť danú aplikáciu pomocou programovania, je vytvoriť Apex spúšťač (Apex Trigger). Spúšťač je definovaný na objekte Test a je nastavený tak, aby sa vykonal pred a po vložení do databázy a pred a po aktualizácii záznamov v databáze.

Spúšťač obsahuje podmienku na určenie kontextu, v ktorom práve spúšťač beží. V tomto prípade kontroluje, či sa spúšťač spúšťa v kontexte vkladania pomocou *Trigger.isInsert* alebo aktualizácie pomocou *Trigger.isUpdate*.

Ak je spúšťač spustený v kontexte vkladania, zavolá metódu *updateStatusAndSendEmail* triedy *TestObjectTriggerHandler*, s parametrom *Trigger.new*, čo je zoznam nových

záznamov. Druhý parameter je prázdny, pretože v kontexte vkladania neexistuje stará verzia záznamov.

Ak je spúšťač spustený v kontexte aktualizácie, tiež zavolá metódu *updateStatusAndSendEmail* triedy *TestObjectTriggerHandler*, s parametrom *Trigger.new*, avšak druhý parameter v tomto kontexte nie je prázdny, ale predstavuje mapu starých záznamov *Trigger.oldMap*. To znamená, že pri aktualizácii existujúcich záznamov sa obsluha spúšťača zavolá s novými aj starými záznamami, čo umožní porovnať staré a nové hodnoty.

```
trigger TestObjectTrigger on Test__c (before insert, before update, after insert, after update) {
    //When trigger runs in insert context there is no old value of the record
    //For testing Example no. 1 via flow, comment code below
    if (Trigger.isInsert) {
        TestObjectTriggerHandler.updateStatusAndSendEmail(Trigger.new, null);
    } else if (Trigger.isUpdate) {
        TestObjectTriggerHandler.updateStatusAndSendEmail(Trigger.new, Trigger.oldMap);
    }
}
```

Obrázok 48 Príklad č. 1 - Apex spúšťač TestObjectTrigger

Apex trieda, ktorá sa volá zo spúšťača sa nazýva aj obsluha spúšťača. V tomto prípade je obsluha spúšťača Apex trieda *TestObjectTriggerHandler*. Trieda obsahuje jednu statickú metódu s názvom *updateStatusAndSendEmail*. Táto metóda prijíma dva parametre: zoznam záznamov objektu Test s názvom *newTestList* a mapu záznamov s názvom *oldTestMap*, kde kľúč mapy je identifikátor záznamu objektu Test a hodnota je samotný záznam objektu Test.

Metóda iteruje nad zoznamom *newTestList* a kontroluje, či je každý záznam testu nový alebo či bolo pole *Grade__c* aktualizované od predchádzajúcej hodnoty uloženej v *oldTestMap*. Ak je táto podmienka splnená, záznam sa pridá sa do zoznamu *changedTests*.

Potom nasleduje ďalší cyklus, v ktorom prebieha iterácia cez zoznam *changedTests*. Vo vnútri cyklu je podmienka, ktorá kontroluje či je pole *Grade__c* prázdne alebo nie.

Ak nie je prázdne, identifikátor záznamu sa pridá do zoznamu *testsToSendEmail* (Pozn.: Môže sa zdať, že tento príkaz sa vykoná dva krát, teda aj v kontexte pred vložením do databázy a aj v kontexte po vložení do databázy. Avšak v kontexte pred vložením do databázy záznam nemá žiadny identifikátor, takže do zoznamu nie je uložená žiadna hodnota). Nasleduje podmienka, ktorá kontroluje či spúšťač beží v kontexte pred uložením do databázy. Ak áno, pole *Status__c* je aktualizované na hodnotu *Graded*. Aktualizácia záznamov sa vykonáva len v kontexte pred vložením do databázy (*Trigger.isBefore*), pretože v kontexte po

vložení do databázy (*Trigger.isAfter*) sú polia dostupné len na čítanie a nie je možné ich aktualizovať.

Ak je pole *Grade__c* prázdne, vykoná sa podmienka, ktorá kontroluje či spúšťač beží v kontexte pred uložením do databázy. Ak áno, hodnota poľa *Status__c* sa aktualizuje na *Ungraded* a hodnota poľa *Comment__c* sa vymaže.

Nakoniec, ak spúšťač beží v kontexte po uložení do databázy (*Trigger.isAfter*) a zoznam *testsToSendEmail* nie je prázdny, zavolá sa metóda *sendEmailToStudent* triedy *SendEmail* na odoslanie emailov študentom.

```
public without sharing class TestObjectTriggerHandler {
    public static void updateStatusAndSendEmail(List<Test__c> newTestList, Map<Id, Test__c> oldTestMap) {
        List<Test__c> changedTests = new List<Test__c>();
        List<Id> testsToSendEmail = new List<Id>();

        for (Test__c newTest : newTestList) {
            Test__c oldTest = oldTestMap != null ? oldTestMap.get(newTest.Id) : null;
            //Only new records and records where grade is updated are in scope
            if (oldTest == null || newTest.Grade__c != oldTest.Grade__c) {
                changedTests.add(newTest);
            }
        }

        for (Test__c testRecord : changedTests) {
            //Check if field Grade is not empty
            //If it is not empty, Status is moved to Graded and Id of record is added to testsToSendEmail
            //If it is empty, Status is moved to Ungraded and field Comment is erased.
            if (!String.isEmpty(testRecord.Grade__c)) {
                //Separated list was created for records for which email should be sent
                //Send email logic should not be in the for loop due to apex governor limits
                testsToSendEmail.add(testRecord.Id);
                //Update of the records is done only in Before context
                //It is because in after context fields are in read only mode
                if (Trigger.isBefore) {
                    testRecord.Status__c = 'Graded';
                }
            } else {
                if (Trigger.isBefore) {
                    testRecord.Status__c = 'Ungraded';
                    testRecord.Comment__c = null;
                }
            }
        }

        //Sending email is done only in After context and if testsToSendEmail is not empty
        //Trigger context have to be After because IDs of related records (Student, Teacher) are needed
        if (Trigger.isAfter && !testsToSendEmail.isEmpty()) {
            SendEmail.sendEmailToStudent(testsToSendEmail);
        }
    }
}
```

Obrázok 49 Príklad č. 1 - Apex trieda TestObjectTriggerHandler

Každá trieda Apex by mala mať zodpovedajúcu testovaciu triedu, aby sa zabezpečilo, že trieda funguje tak, ako má, a aby sa overilo jej správanie v rôznych scenároch. Salesforce

vyžaduje minimálne pokrytie potrebné pre triedu Apex 75 %. To znamená, že aspoň 75 % spustiteľných riadkov triedy musí byť pokrytých testami. Okrem toho musia byť aspoň raz otestované všetky spúšťače. Dosiahnutie tohto minimálneho pokrytia pomáha zabezpečiť, aby sa testovali najkritickejšie časti kódu, čím sa znižuje riziko neočakávaných problémov v produkcii. Vo všeobecnosti sa však odporúča usilovať sa o vyššie pokrytie, ideálne 100 %, aby sa dôkladne otestovala trieda a jej okrajové prípady. Pre potreby príkladu č. 1 bola vytvorená testovacia trieda *TestObjectTriggerHandlerTest*. Táto testovacia trieda pokrýva na 100% triedu *TestObjectTriggerHandler* a spúšťač *TestObjectTrigger*. Triedu *SendEmail* pokrýva táto testovacia trieda na 86%, čo je dostačujúce. Nepokrytá časť kódu je vetva *catch* v *try-catch* bloku. Do tejto časti sa kód dostane iba v prípade ak je dosiahnutý povolený maximálny počet odoslaných emailov. V testovacom kontexte tento limit nie je možné dosiahnuť, pretože pre testovací kontext sa nevzťahuje väčšina Salesforce limitov.

Trieda *SendEmail* obsahuje jednu statickú metódu s názvom *sendEmailToStudent*. Táto metóda preberá parameter *testRecordIds*, čo je zoznam identifikátorov reprezentujúcich záznamov objektu Test. Na začiatku metódy sa vykoná požiadavka na databázu na získanie potrebných polí záznamov objektu Test, pomocou SOQL na základe vstupného parametra *testRecordIds*.

Následne sa vykoná iterácia nad zoznamom *newTestList,* získaným z SOQL požiadavky. Na začiatku cyklu sa nachádza blok *try-catch* na spracovanie prípadných výnimiek pri rezervovaní kapacity emailov pomocou *Messaging.reserveSingleEmailCapacity(1)*, čo vlastne znamená, že je dopredu definované, že sa email nepošle viacerým adresátom ale iba jednému. Salesforce povoľuje poslať v jeden deň len určité množstvo emailov, presný počet záleží podľa typu licencie a typu Salesforce inštancie. Preto ak by hrozilo, že poslaním ďalšieho emailu bude prekročený limit, v bloku *try-catch* sa vykoná vetva *catch*, v ktorej sa nachádza logika, ktorá zobrazí informácie o limitoch do konzoly. Konzolu je možné zobraziť v nástroji Vývojárska konzola, kliknutím na záložku *Logs*. Avšak je potrebné najprv otvoriť konzolu a až potom spustiť triedu.

Za *try-catch* blokom sa pre každý záznam zo zoznamu *newTestList* vytvorí objekt *Messaging.SingleEmailMessage* s názvom *mail* na zostavenie emailu. Adresa príjemcu emailu je nastavená na emailovú adresu študenta, ktorý písal daný test. Ako meno odosielateľa je zobrazené meno používateľa, ktorý vytvoril daný záznam. Následne je vytvorený predmet a telo emailu pomocou kombinácie textu a hodnôt polí zo záznamu. Zložený email sa pridá do

zoznamu emailov na odoslanie. Po skončení cyklu sa použije metóda *Messaging.sendEmail* na odoslanie zoznamu emailov uložených v *emailsToSend*.

```
public without sharing class SendEmail {
    public static void sendEmailToStudent(List<Id> testRecordIds) {
        //Needed fields of the record are queried based on Id
        List<Test__c> newTestList = [SELECT Id, Name, Grade__c, Comment__c, CreatedBy.Name, CreatedBy.Email,
                                     Student__r.Name, Student__r.Email_address__c, Teacher__r.Name
                                     FROM Test__c WHERE Id IN :testRecordIds];

        List<Messaging.SingleEmailMessage> emailsToSend = new List<Messaging.SingleEmailMessage>();
        for (Test__c newTest : newTestList) {
            try {
                //Apex has limit of single message emails that can be send
                //Reserves email capacity to send single email to the specified number of email addresses.
                //If limit is hit, code fails and code continues in catch part, where Limits are displayed in console
                Messaging.reserveSingleEmailCapacity(1);
            } catch (Exception e) {
                Map<String,System.OrgLimit> limitsMap = OrgLimits.getMap();
                System.OrgLimit apiRequestsLimit = limitsMap.get('SingleEmail');
                System.debug('Limit Name: ' + apiRequestsLimit.getName());
                System.debug('Usage Value: ' + apiRequestsLimit.getValue());
                System.debug('Maximum Limit: ' + apiRequestsLimit.getLimit());
            }

            //Setting ToAddress, ReplyTo address, Sender Display Name and Subject using methods from Messaging class
            Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
            mail.setToAddresses(new List<String>{newTest.Student__r.Email_address__c});
            mail.setReplyTo(newTest.CreatedBy.Email);
            mail.setSenderDisplayName(newTest.CreatedBy.Name);
            mail.setSubject('Test ' + newTest.Name + ' was graded!');

            //Building email body
            String emailBody = 'Dear student ' + newTest.Student__r.Name + ',\nTest ' + newTest.Name +
            ' was graded by teacher ' + newTest.Teacher__r.Name + '.\n' +
            'Your grade is ' + newTest.Grade__c + '.\n';
            String emailBodyComment = String.isEmpty(newTest.Comment__c) ? '\n' : 'Teacher also wrote a comment to your test:\n' + newTest.Comment__c + '\n\n';
            String emailBodyFooter = 'Have a nice day!';
            mail.setPlainTextBody(emailBody + emailBodyComment + emailBodyFooter);

            //adding single email to list
            emailsToSend.add(mail);
        }

        //sending list of emails
        Messaging.sendEmail(emailsToSend);
    }
}
```

Obrázok 50 Príklad č. 1 - Apex trieda SendEmail

4.1.5 Zhodnotenie

Pomocou nástroja Flow Builder bol vytvorený prehľadný flow avšak vývoj pomocou programovania bol zložitejší, pretože aktualizáciu záznamov a posielanie emailov nie je možné vykonať v jednom kontexte spúšťača. Aktualizáciu záznamov je možné vykonať iba v kontexte pred uložením do databázy, pretože v kontexte po uložení do databázy sú záznamy uzamknuté a je možné ich iba čítať. Naopak posielanie emailov je možné iba v kontexte po uložení do databázy, pretože pre poslanie emailu je potrebné poznať identifikátor záznamu, ktorý sa avšak vytvorí až po uložení do databázy.

Posielanie emailov v Salesforce je limitované. Presný denný limit sa líši podľa licencie a typu Salesforce inštancie. Pre potreby tejto práce bola vytvorená vývojárska bezplatná inštancia, v ktorej sú veľmi nízke limity. Maximálny denný možný počet poslaných jednoduchých emailov (menej ako 250 adresátov) z Apexu je 15. Z deklaratívnych nástrojov (Flow Builder, Process Builder) je to 405, čo je 27 krát viac ako z Apexu. Pre tzv. Full inštanciu (úplná kópia produkčnej inštancie) je tento rozdiel ešte markantnejší. Denný limit

pre jednoduché emaily z Apexu je 5 tisíc. Avšak pre deklaratívne nástroje je tento limit až 2 milióny, čo je 400 násobne viac.

Existujú ale aj spôsoby ako sa vyhnúť prekročeniu denného limitu pre emaily z Apexu. Napríklad ak sa email spojí s identifikátorom používateľa pomocou štandardnej metódy *setTargetObjectId*, email sa nezapočíta do denného limitu. Avšak táto možnosť nevyhovovala požiadavkám príkladu č. 1.

Po zhodnotení oboch riešení, je možné konštatovať, že aplikáciu pre príklad č. 1 je jednoznačne vhodnejšie vytvoriť pomocou deklaratívneho spôsobu vývoja aplikácii.

4.2 Príklad číslo 2

V príklade číslo 2 bude navrhnuté riešenie vyvinuté pomocou deklaratívneho ale aj programovacieho spôsobu vývoja aplikácií v Salesforce.

4.2.1 Zadanie

Spoločnosť, ktorá sa zaoberá obchodovaním s kryptomenami chce zlepšiť svoj zákaznícky servis. Pilotná verzia aplikácie bude dostupná iba pre klientov, ktorí investovali do kryptomeny Bitcoin. V databáze klientov bude mať každý klient vyplnené základné údaje ako meno a emailovú adresu, tiež množstvo zakúpenej kryptomeny Bitcoin a hodnotu danej kryptomeny v eurách v momente nákupu. Klient tiež uvedenie svoj plánovaný zisk v percentách. Každý deň bude prebiehať kontrola všetkých účtov klientov. Bude porovnávaná súčasná hodnota konta klienta s počiatočnou hodnotou konta. Súčasná hodnota konta bude vypočítaná po zistení aktuálnej trhovej hodnoty kryptomeny Bitcoin. V prípade ak bude pomer súčasného stavu konta väčší ako plánovaný zisk klienta, klientovi sa odošle emailová notifikácia s informáciou o súčasnom stave konta. V prípade ak plánovaný zisk nebude dosiahnutý, emailová notifikácia sa neodošle, avšak na zázname klienta bude uvedená informácia o tom, že plánovaný zisk nebol dosiahnutý.

4.2.2 Dátový model

Pre splnenie požiadaviek uvedených v zadaní je najvhodnejšie vytvoriť špecifický objekt na mieru s názvom Investor. Postup ako sa vytvára objekt v systéme Salesforce je podrobne vysvetlený v príklade č. 1. Informácie o objekte budú vyplnené nasledovne:

- Label: Investor
- Plural Label: Investors

- Allow Search: checked

Všetky ostatné informácie budú ponechané nezmenené, podľa predvoleného nastavenia. Následne je potrebné vytvoriť polia na objekte Investor.

Tabuľka 3 Polia vytvorené pre objekt Investor

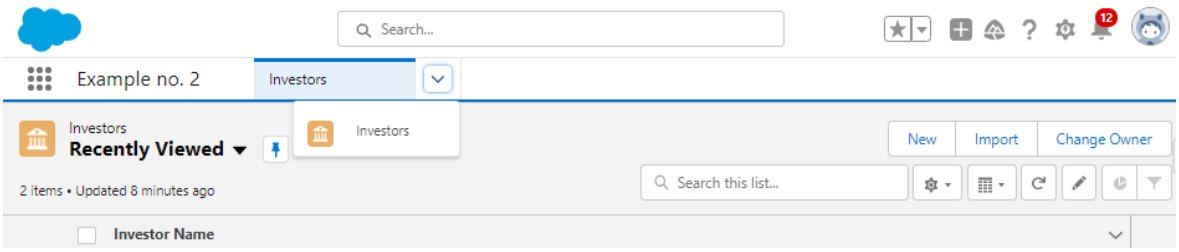
Názov poľa (Field label)	**Typ poľa (Field Type)**
Actual balance	Formula(Number)
Actual value	Number(10, 2)
Amount of BTC	Number(9, 9)
Desired profit	Percent(5, 2)
Email address	Email
Initial balance	Formula (Number)
Initial value	Number (10, 2)
Status	Picklist (Not yet been checked., Desired profit not reached., Desired profit reached., Error in HTTP Request)

Pole *Actual balance*, typu Formula, je vypĺňané automaticky súčinom hodnôt polí *Actual value* a *Amount of BTC*. Podobne je vypĺňané aj pole *Initial balance*, súčinom hodnôt polí *Initial value* a *Amount of BTC*. Pole *Actual value* slúži iba ako pomocné pole, takže nie je zobrazené v užívateľskom prostredí.

Prístup na novo vytvorené polia bol udelený tak, že profil System Administrator a Test User majú pre dané polia prístup čítaj a upravuj (Read+Edit). Ako povinné boli označené polia: *Amount of BTC*, *Desired profit*, *Email address*, *Initial value* a *Status*.

Aby používateľ mohol s novo vytvoreným objektom pohodlne pracovať je potrebné vytvoriť najprv pre daný objekt záložku a potom samotnú aplikáciu. Podrobný postup vytvárania záložky je opísaný v príklade č. 1. Pri vytváraní novej záložky sa teda ako objekt vyberie objekt Investor. Okrem objektu treba vybrať aj štýl záložky, pre objekt Investor sa hodí štýl Bank avšak je možné vytvoriť aj vlastný štýl.

Následne je potrebné vytvoriť aplikáciu. Podrobný postup vytvárania aplikácie je opísaný v príklade č. 1. Názov aplikácie bude Example no. 2, čiže príklad číslo 2. V sekcii *Navigation Style* je potrebné zvoliť možnosť *Console navigation*. Následne je potrebné vybrať, ktoré záložky objektov budú dostupné v danej aplikácii, v tomto prípade to bude iba objekt Investor. Posledným krokom pri vytváraní aplikácie je určenie profilov pre ktoré bude aplikácia dostupná. Je potrebné vybrať profily Test User a System Administrator. Kliknutím na uložiť sa aplikácia vytvorí a je pripravená na použitie.

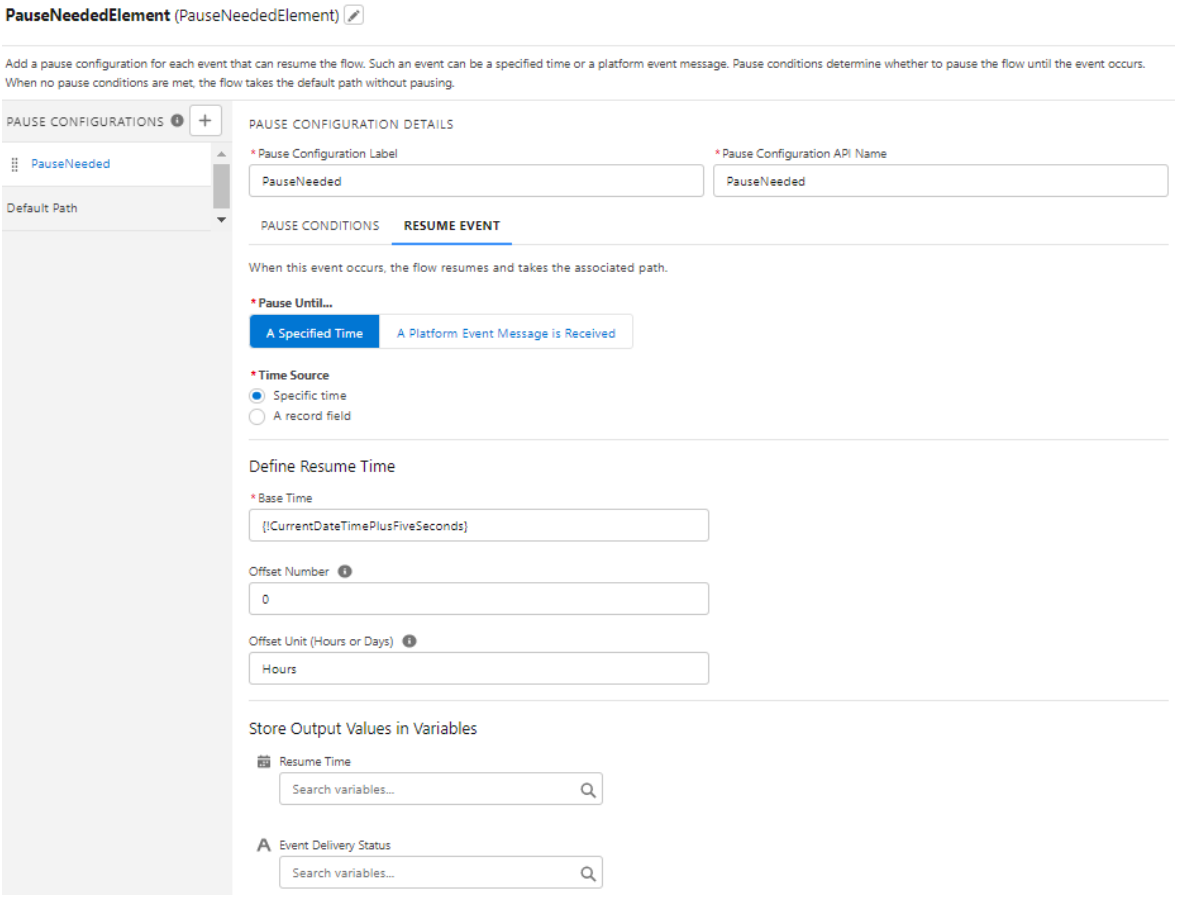


Obrázok 51 Aplikácia Example no. 2

4.2.3 Deklaratívny spôsob vývoja aplikácie

Vývoj aplikácie pre príklad č. 2 bude prebiehať použitím nástroja Flow Builder, pretože Salesforce odporúča tento nástroj pre deklaratívny spôsob vývoja aplikácie ako overený postup (best practice). Prvým krokom je výber typu flowu. V zadaní príkladu č. 2 je uvedené, že emailová notifikácia má byť odoslaná každý deň, takže ako typ flowu je potrebné vybrať *Schedule-Triggered Flow*, čiže Flow spustený podľa harmonogramu. Následne je potrebné určiť kedy a ako často bude flow spustený. Ako frekvenciu je potrebné vybrať možnosť denne a ako počiatočný čas je možné určiť napríklad 8:00. Potom je potrebné zadať objekt a podmienky, ktoré musí každý záznam spĺňať. V tomto prípade bude vybraný objekt Investor a vstupné podmienky nebudú definované, pretože v zadaní príkladu č. 2 je uvedené, že majú byť skontrolované všetky záznamy objektu Investor.

Ako prvý element bude pridaný element typu *Pause* (pauza). Názov nového elementu bude *PauseNeededElement*. Pri flowe, ktorý je typu *Schedule-Triggered Flow* je potrebné tento element použiť pred tým ako bude použitý element typu *Action-Create HTTP Callout (Beta)*. Z názvu tejto akcie vyplýva, že je to iba beta verzia, tzn. skúšobná verzia. Táto akcia vie pracovať bezchybne aj bez použitia elementu typu *Pause* zatiaľ iba v prípade ak sa jedná o flow typu *Record-Triggered Flow*. V elemente *PauseNeededElement* je potrebné definovať konfiguráciu elementu *Pause*. Názov konfigurácie bude *PauseNeeded*. V konfigurácii sa určuje kedy sa má vykonávanie flowu zastaviť a kedy má pokračovať. V sekcii *Pause conditions* bude vybraná možnosť aby sa flow zastavil vždy. V sekcii *Resume Event* bude vybraná možnosť aby flow pokračoval v špecifický čas. Čas kedy má flow pokračovať je definovaný v premennej typu *Formula-Date/Time* s názvom *CurrentDateTimePlusFiveSeconds*. Hodnota premennej je: *{!$Flow.CurrentDateTime}+5/(24*60*60)*. *{!$Flow.CurrentDateTime}* označuje aktuálny čas v momente kedy je táto premenná použitá. *5/(24*60*60)* označuje 5 sekúnd. Element *PauseNeededElement* je znázornený na obrázku č. 52.



Obrázok 52 Element typu Pause

Ešte pred tým ako bude pridaný element typu element typu *Action-Create HTTP Callout (Beta)* je potrebné vykonať niekoľko úprav v nastaveniach. Najprv bude vytvorený súbor povolení (*Permission Set*). V nastaveniach, vyhľadaním slovného spojenia *Permission Sets*, sa otvorí zoznam súborov povolení. Po kliknutí na tlačidlo *New* sa otvorí okno vytvorenia nového súboru povolení. Názov nového súboru povolení bude *CreateCallout*, ako licencia bude zvolená možnosť Salesforce. Kliknutím na tlačidlo *Save* sa vytvorí nový súbor povolení. Kliknutím na tlačidlo *Manage Assignments* sa otvorí zoznam používateľov, ktorí majú priradený daný súbor povolení. Je potrebné do tohto zoznamu pridať používateľov, ktorí budú manipulovať s flowom.

Následne je potrebné v nastaveniach vyhľadať slovné spojenie *Named Credentials*. Otvorí sa zoznam menovitých poverení, je potrebné kliknúť na záložku *External Credentials* (externé poverenia). Po kliknutí na tlačidlo *New* sa otvorí okno vytvorenia nového externého poverenia. Názov externého poverenia bude *NoAuthentication* a ako *Authentication Protocol* bude zvolená možnosť *Custom*, keďže pre získanie aktuálnej hodnoty kryptomeny bude použitá API, ktorá nevyžaduje žiadnu autentizáciu. Kliknutím na tlačidlo *Save* sa

vytvorí nové externé poverenie a zobrazí sa okno tohto poverenia. V sekcii *Permission Set Mapping* je potrebné kliknúť na tlačidlo New, vybrať novo vytvorený súbor povolení a uložiť výber. Opätovným vyhľadaním slovného spojenia *Named Credentials* v nastaveniach sa otvorí zoznam menovitých poverení. Následne je potrebné vytvoriť menovité poverenie. Názov menovitého poverenia bude *GetPriceCredential*, do poľa *URL* bude vložená hodnota *https://api.gemini.com*. Tento portál poskytuje bezplatné informácie o aktuálnych trhových hodnotách kryptomien. V poli *External Credential* bude vybrané externé poverenie *No-Authentication*. Kliknutím na tlačidlo *Save* sa vytvorí nové menovité poverenie. Následne je možné pokračovať v konfigurácii v nástroji Flow Builder.

Ako druhý element bude pridaný element typu *Action-Create HTTP Callout (Beta)*. Názov nového elementu bude *getPrice* a v poli menovité poverenia bude vybrané menovité poverenie s názvom *GetPriceCredential*. Kliknutím na tlačidlo *Next* sa otvorí okno vytvorenia externej služby. Názov externej služby bude *GetPriceCallout*, a ako metóda bude vybraná metóda GET. Keďže je akcia typu *Create HTTP Callout* iba v skúšobnej verzii, momentálne je podporovaná iba metóda GET. Avšak práva metóda GET je tá, ktorá je potrebná pre získanie aktuálnej hodnoty kryptomeny. Následne je potrebné definovať cestu URL, t. j. časť URL za doménou. V tomto prípade to bude */v2/ticker/btceur*, pretože podľa zadania príkladu č. 2 je potrebné zistiť aktuálnu hodnotu kryptomeny Bitcoin v eurách. V sekcii *Provide Sample Response* je po kliknutí na tlačidlo *New* potrebné uviesť vzorovú odpoveď z API. Vzorovú odpoveď je možné získať z webového prehliadača zadaním URL: *https://api.gemini.com/v2/ticker/btceur*. Vzorová odpoveď vyzerá nasledovne:

{"sym-
bol":"BTCEUR","open":"26228.48","high":"27949.54","low":"26056.5","close":"26056.5","changes":["26228.48","26261.01","26309.62","26244.26","26312.27","26417.39","26514.98","26391.32","26390.98","26293.84","26293.84","26225.7","26258.83","26258.83","26185.23","26185.23","26185.23","26223.92","26267.67","26195.69","26236.68","26208.11","26056.5","26184.14"],"bid":"26190.45","ask":"26203.76"}

Následne kliknutím na tlačidlo Done je vytvorený element *getPrice* typu *Action-Create HTTP Callout (Beta)*.

Ako tretí element bude pridaný element Decision (rozhodnutie). Názov nového elementu bude *ResponseCheck*. Na ľavej strane obrazovky elementu sa nachádza zoznam tzv. *Outcomes*. Pre každú cestu, ktorou sa flow môže uberať, by mal byť vytvorený *Outcome*. Pre

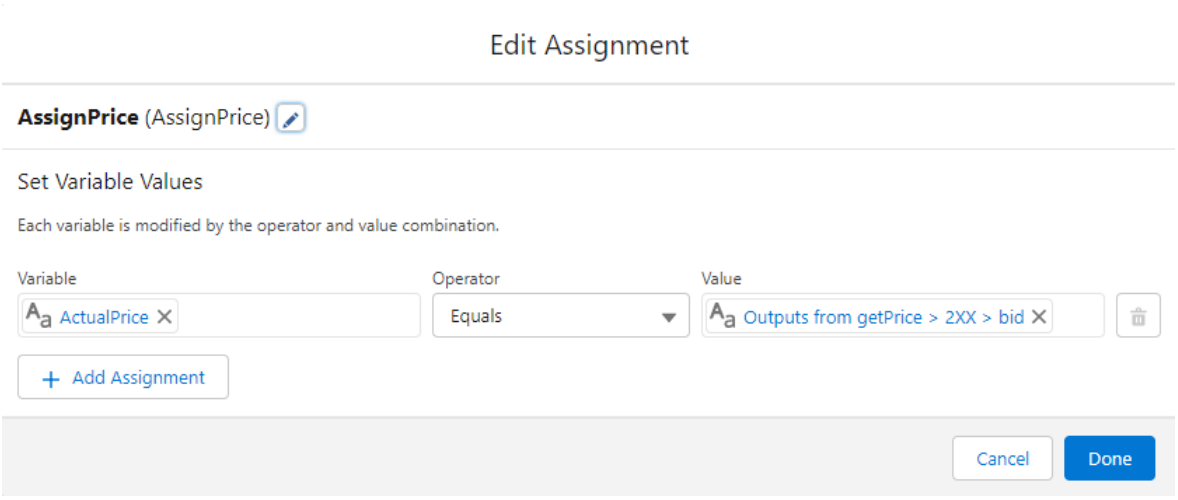
každý *Outcome* je potrebné uviesť podmienky, ktoré musia byť splnené, aby sa flow vybral danou cestou. *Outcome* je vlastne rozhodovacia vetva.

Prvá rozhodovacia vetva bude nazvaná *ResponseIs200*, a bude obsahovať podmienku, ktorá sa vyhodnotí či kód odpovede z API je 200, čo znamená, že požiadavka bola úspešná.

Druhá rozhodovacia vetva bude nazvaná *ResponseIsNot200* a nebude obsahovať žiadnu podmienku, čo znamená že sa vykoná iba v prípade ak požiadavka na API nebude úspešná. Po uložení zmien v elemente *Decision*, sa na obrazovke objavia dve čiary, ktoré vychádzajú z elementu *Decision* a reprezentujú rozhodovacie vetvy.

Do rozhodovacej vetvy *ResponseIsNot200* bude pridaný element typu aktualizuj záznam s názvom *UpdateInvestor3*. Tento element aktualizuje pole *Status* na hodnotu *Error in HTTP Request*.

Do rozhodovacej vetvy *ResponseIs200* bude pridaný element typu priradenie (Assignemt) s názvom *AssignPrice*. Avšak pred konfiguráciou daného elementu je potrebné vytvoriť premennú typu Text s názvom *ActualPrice*. Do tejto premennej bude v elemente *AssignPrice* uložená odpoveď z API.



Obrázok 53 Element typu Assignment

Následne je potrebné vytvoriť premennú typu formula-number s názvom *ActualValueFormula*. Táto premenná bude slúžiť na konverziu premennej *ActualPrice* na číslo. Do rozhodovacej vetvy *ResponseIs200*, za element *AssignPrice* bude pridaný element typu aktualizuj záznam s názvom *UpdateActualValue*. Tento element aktualizuje pole *Actual_value__c* na hodnotu premennej *ActualValueFormula*.

Za element *UpdateActualValue* bude pridaný element *Decision* (rozhodnutie). Názov nového elementu bude *PriceDecision*. Tento element bude obsahovať dve rozhodovacie vetvy.

Prvá rozhodovacia vetva bude nazvaná *IsDesiredProfitReached*, a bude obsahovať podmienku, ktorá skontroluje či bol dosiahnutý požadovaný profit. Avšak aby táto podmienka vôbec mohla byť vyhodnotená, je potrebné zistiť aktuálny stav konta klienta a stav konta pri dosiahnutí požadovaného zisku.

Pre zistenie aktuálneho stavu konta je potrebné vytvoriť premennú typu formula-number s názvom *ActualBalance*. V tejto premennej je vypočítaný aktuálny stav konta súčinom hodnoty premennej *ActualValueFormula* a hodnoty poľa *Amount of BTC*.

Pre zistenie stavu konta pri dosiahnutí požadovaného zisku je potrebné vytvoriť premennú typu formula-number s názvom *DesiredBalance*. V tejto premennej je vypočítaný stav konta pri dosiahnutí požadovaného zisku súčinom hodnoty poľa *Initial balance* a upravenej hodnoty poľa *Desired profit*. Hodnota poľa *Desired profit* bola vydelená číslom 100 a následne k výsledku podielu bolo pričítané číslo 1. Takže napríklad ak hodnota poľa *Desired profit* bola 30, upravená hodnota tohto poľa pre potreby výpočtu stavu konta pri dosiahnutí požadovaného zisku bola 1,3.

Druhá rozhodovacia vetva bude nazvaná *DesiredProfitNotReached* a nebude obsahovať žiadnu podmienku, čo znamená že sa vykoná iba v prípade ak hodnota premennej *ActualBalance* bude menšia ako hodnota premennej *DesiredBalance*.

Po uložení zmien v elemente *Decision*, sa na obrazovke objavia dve čiary, ktoré vychádzajú z elementu *Decision* a reprezentujú rozhodovacie vetvy.

Do rozhodovacej vetvy *DesiredProfitNotReached* bude pridaný element typu aktualizuj záznam s názvom *UpdateInvestor2*. Tento element aktualizuje pole *Status* na hodnotu *Desired profit not reached*.

Do rozhodovacej vetvy *IsDesiredProfitReached* bude pridaný element typu typu aktualizuj záznam s názvom *UpdateInvestor1*. Tento element aktualizuje pole *Status* na hodnotu *Desired profit reached.* Za element *UpdateInvestor1* bude pridaný element typu *Action – SendEmail* s názvom *SendEmailNotification*. Pre telo emailu bola vytvorená premenná typu Text template s názvom *EmailBody*, podobne ako aj v príklade č. 1. Hodnota tejto premennej je nasledovná:

Dear {!$Record.Name},

Congratulation! You have reached desired profit.

Your desired profit was {!$Record.Desired_profit__c}%.

Your actual profit is {!ActualProfit}%.

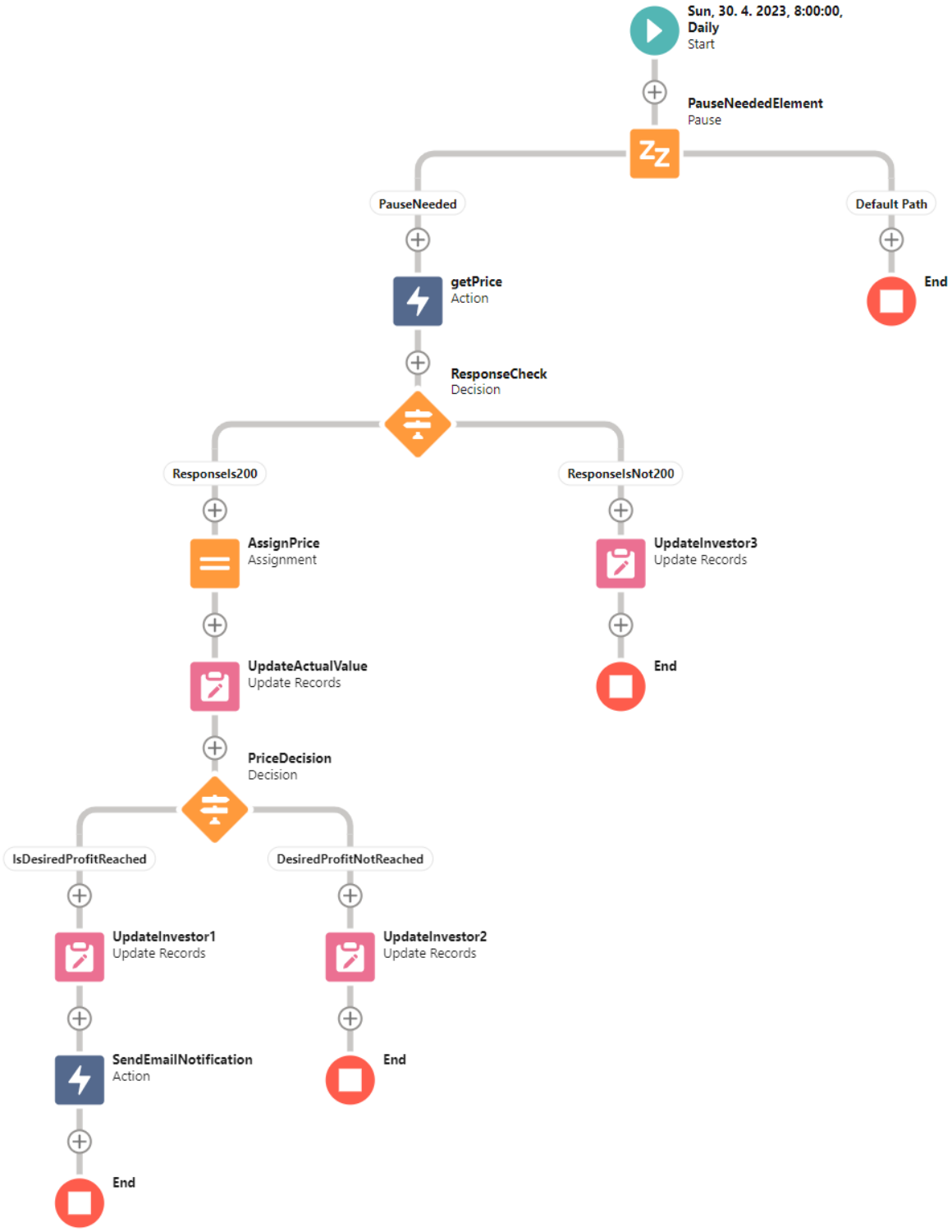
Value of your balance is now {!ActualBalance}.

Have a nice day!

{!$Record.Name}* je odkaz na názov záznamu objektu Investor. *{!$Record.Desired_profit__c} je odkaz na hodnotu poľa *Desired profit*. *{!ActualProfit}* je odkaz na premennú *ActualProfit*. *{!ActualBalance}* je odkaz na premennú *ActualBalance*.

Do poľa *Subject* bude vložený textový reťazec: *Desired Profit was reached!* Do poľa *Recipient Email Addresses (comma-separated)* bude vložený odkaz na hodnotu poľa *Email address* na zázname objektu Investor. Do poľa *Sender Email Address* bude vložená emailová adresa *testsender@test.com*.

Tým je vývoj flowu ukončený, je potrebné ho ešte uložiť a aktivovať. Pri ukladaní je potrebné zadať názov flowu, ktorý bude v tomto prípade *Example2Flow*. V nástroji *Flow Builder* je tiež vstavaný nástroj na ladenie flowu. Obrazovka tohto nástroja sa otvorí po kliknutí na tlačidlo *Debug*. Po otvorení obrazovky je možné nakonfigurovať podmienky pri ktorých bude flow ladený. Vizualizácia finálnej verzie flowu spolu s premennými je znázornená na obrázku č. 54.



Obrázok 54 Príklad č. 2 vypracovaný pomocou nástroja Flow Builder

4.2.4 Programovací spôsob vývoja aplikácie

V zadaní príkladu č. 2 je uvedené, že emailová notifikácia má byť odoslaná každý deň, takže pre splnenie zadania je potrebné vytvoriť Apex triedu, ktorá bude implementovať rozhranie

Schedulable. V programovacom jazyku Apex sa rozhranie *Schedulable* používa na definovanie triedy, ktorú možno naplánovať tak, aby sa spúšťala v určitých intervaloch alebo v určitých dátumoch a časoch.

Pre príklad č. 2 bola vytvorená Apex trieda s názvom *TestScheduler*, ktorá implementuje rozhranie *Schedulable*. Metóda *execute* je vstupným bodom pre naplánovaný dávkový proces, v každej triede ktorá implementuje rozhranie *Schedulable* musí byť definovaná metóda *execute*. Z metódy *execute* sa zavolá metóda *processPrice*, ktorá vykonáva logiku spracovania ceny kryptomeny.

Metóda *processPrice* je anotovaná pomocou *@future(callout=true)*, aby sa vykonalo asynchrónne volanie na externý koncový bod HTTP. Metóda načíta zoznam všetkých záznamov objektu Investor__c z databázy Salesforce pomocou dotazu SOQL. Následne sa zavolá metóda *getPrice* pomocou ktorej je získaná aktuálna cena tak, že sa vykoná HTTP požiadavka metódou GET na zadaný externý koncový bod. Ak je kód odpovede 200 (čo znamená úspešnú požiadavku), z odpovede v tvare JSON je extrahovaná hodnota *bid* a uložená do premennej typu *Double*, ktorá je návratovou hodnotou. Ak kód odpovede nie je 200, metóda vráti 0. Následne v metóde *processPrice*, v prípade ak je aktuálna hodnota vyššia ako 0, prebehne iterácia nad zoznamom všetkých záznamov objektu Investor. Pre každý záznam objektu Investor bude pole *Actual Value* aktualizované aktuálnou hodnotou kryptomeny. Následne bude rovnako ako vo flowe Example2Flow vypočítaný aktuálny stav konta klienta a stav konta pri dosiahnutí požadovaného zisku. Ak je aktuálny stav konta väčší alebo rovný ako stav konta pri dosiahnutí požadovaného zisku, pole *Status* bude aktualizované na hodnotu *Desired profit reached*. Spolu so zmenou poľa *Status* bude odoslaná emailová notifikácia zavolaním metódy *sendEmail*. Ak aktuálny stav konta nie je väčší alebo rovný ako stav konta pri dosiahnutí požadovaného zisku, pole *Status* bude aktualizované na hodnotu *Desired profit not reached*. Ak nie je aktuálna hodnota kryptomeny získaná z metódy *getPrice* väčšia ako 0, pole *Status* bude aktualizované na hodnotu *Error in HTTP Request*.

Vstupnými parametrami do metódy *sendEmail* je záznam objektu Investor a hodnota aktuálneho stavu konta klienta. Na začiatku metódy sa nachádza blok *try-catch* na spracovanie prípadných výnimiek pri rezervovaní kapacity emailov pomocou *Messaging.reserveSingleEmailCapacity(1)*, čo vlastne znamená, že je dopredu definované, že sa email nepošle viacerým adresátom ale iba jednému. Salesforce povoľuje poslať v jeden deň len určité množstvo emailov, presný počet záleží podľa typu licencie a typu Salesforce inštancie. Preto ak by hrozilo, že poslaním ďalšieho emailu bude prekročený limit, v bloku *try-catch* sa

vykoná vetva *catch*, v ktorej sa nachádza logika, ktorá zobrazí informácie o limitoch do konzoly. Za *try-catch* blokom je definovaná premenná typu Double s názvom *actualProfit*. Táto premenná slúži pre výpočet aktuálneho profitu klienta. Následne je vytvorený email typu *Messaging.SingleEmailMessage* s názvom *mail*. Adresa príjemcu emailu je nastavená na emailovú adresu klienta, takže na hodnotu poľa *Email address* záznamu objektu Investor. Následne je vytvorený predmet a telo emailu pomocou kombinácie textu a hodnôt polí zo záznamu. Na záver sa použije metóda *Messaging.sendEmail* na odoslanie emailu.

Každá trieda Apex by mala mať zodpovedajúcu testovaciu triedu, preto pre potreby príkladu č. 2 bola vytvorená testovacia trieda *TestSchedulerTest*. Táto testovacia trieda pokrýva triedu *TestScheduler* na 93%, čo je dostačujúce keďže Salesforce vyžaduje pokrytie minimálne 75%. Nepokrytá časť kódu je vetva *catch* v *try-catch* bloku. Do tejto časti sa kód dostane iba v prípade ak je dosiahnutý povolený maximálny počet odoslaných emailov. V testovacom kontexte tento limit nie je možné dosiahnuť, pretože pre testovací kontext sa nevzťahuje väčšina Salesforce limitov.

Odoslanie HTTP požiadavky bolo simulované pomocou rozhrania *HttpCalloutMock*. Implementáciou tohto rozhrania je možné definovať maketu odpovede HTTP, ktorá bude vrátená počas vykonávania testu, namiesto skutočnej odpovede požiadavky HTTP. Pre testovaciu triedu *TestSchedulerTest* boli vytvorené dve pomocné triedy *MockHttpResponseGenerator* a *MockHttpResponseGeneratorErrorResponse*, ktoré implementujú rozhranie *HttpCalloutMock*.

Po dokončení vývoja Apex triedy *TestScheduler* je ešte potrebné naplánovať spúšťanie tejto triedy. Plánovanie spúšťania triedy prebieha pomocou konzoly vývojára. V konzoli vývojára je potrebné otvoriť okno *Open Execute Anonymous Window* (klávesová skratka CRTL+E). Do tohto okna je potrebné zadať príkaz *System.schedule('TestScheduler'+System.now(), '0 0 13 * * ? ', new TestScheduler());*. Prvý parameter označuje názov pre naplánovaný dávkový proces. Druhý parameter je výron cron. Výraz cron je reťazec pozostávajúci z piatich alebo šiestich polí oddelených bielou medzerou. Každé pole predstavuje časovú jednotku v určenom poradí (minúta, hodina, deň v mesiaci, mesiac, deň v týždni, rok (nepovinné)). Výrazy cron sa používajú na definovanie plánu, ktorý sa spúšťa v pevnom intervale, napríklad každú hodinu alebo každý deň v určitom čase. Tretí parameter je názov Apex triedy, ktorá implementuje rozhranie *Schedulable*.

```
public with sharing class TestScheduler implements Schedulable {
    public void execute(SchedulableContext ctx)
    {
        processPrice();
    }

    //Asynchronous method to process price received via HTTP Request
    @future (callout=true)
    public static void processPrice() {
        List<Investor__c> listOfInvestors = [SELECT Id, Actual_value__c, Amount_of_BTC__c, Desired_profit__c,
                                             Email_address__c, Initial_value__c, Initial_balance__c, Name, Status__c
                                             FROM Investor__c];
        Double price = getPrice();
        for(Investor__c inv : listOfInvestors) {
            if(price > 0) {
                inv.Actual_value__c = price;
                Double actualBalance = (inv.Actual_value__c * inv.Amount_of_BTC__c).setScale(2);
                Double desiredBalance = inv.Initial_balance__c * ((inv.Desired_profit__c / 100) + 1);
                if (actualBalance >= desiredBalance) {
                    inv.Status__c = 'Desired profit reached.';
                    sendEmail(inv, actualBalance);
                } else {
                    inv.Status__c = 'Desired profit not reached.';
                }
            } else {
                inv.Status__c = 'Error in HTTP Request';
            }
        }
        update listOfInvestors;
    }

    //Method for sending HTTP request and receiving response
    public static Double getPrice() {
        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:GetPriceCredential'+'/v2/ticker/btceur');
        req.setMethod('GET');

        Http http = new Http();
        HTTPResponse res = http.send(req);
        if (res.getStatusCode() == 200) {
            Map<String, Object> m = (Map<String, Object>) JSON.deserializeUntyped(res.getBody());
            Double strToDec = Double.valueOf(m.get('bid'));
            return strToDec;
        } else { return 0; }
    }

    public static void sendEmail(Investor__c inv, Double actualBalance) {
        List<Messaging.SingleEmailMessage> emailsToSend = new List<Messaging.SingleEmailMessage>();
        try {
            //Apex has limit of single message emails that can be send
            //Reserves email capacity to send single email to the specified number of email addresses.
            //If limit is hit, code fails and code continues in catch part, where Limits are displayed in console
            Messaging.reserveSingleEmailCapacity(1);
        } catch (Exception e) {
        Map<String,System.OrgLimit> limitsMap = OrgLimits.getMap();
        System.OrgLimit apiRequestsLimit = limitsMap.get('SingleEmail');
        System.debug('Limit Name: ' + apiRequestsLimit.getName());
        System.debug('Usage Value: ' + apiRequestsLimit.getValue());
        System.debug('Maximum Limit: ' + apiRequestsLimit.getLimit());
    }

    Double actualProfit = (((inv.Actual_value__c / inv.Initial_value__c) - 1 ) * 100).setScale(2);
    //Setting ToAddress, ReplyTo address, Sender Display Name and Subject using methods from Messaging class
    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
    mail.setToAddresses(new List<String>{inv.Email_address__c});
    mail.setReplyTo('testsender@test.com');
    mail.setSubject('Desired Profit was reached!');

    //Building email body
    String emailBody = 'Dear ' + inv.Name + ',\nCongratulation! You have reached desired profit.\n' +
    'Your desired profit was ' + inv.Desired_profit__c + '%.\nYour actual profit is ' + actualProfit + '%.\n' +
    'Value of your balance is now ' + actualBalance + '.\n\nHave a nice day!';

    mail.setPlainTextBody(emailBody);

    //adding single email to list
    emailsToSend.add(mail);

    //sending list of emails
    Messaging.sendEmail(emailsToSend);
}
```

Obrázok 55 Príklad č. 2 - Apex trieda TestScheduler

4.2.5 Zhodnotenie

V nástroji Flow Builder bol vytvorený funkčný flow, ktorý spĺňa požiadavky príkladu č. 2. Pre poslanie požiadavky HTTP bol použitý element typu *Action-Create HTTP Callout (Beta)*. Tento element je stále iba v skúšobnej verzií takže Salesforce zatiaľ nezaručuje jeho sto percentnú funkčnosť. Pre správne fungovanie vo flowe typu *Schedule-Triggered Flow* bolo navyše potrebné pred daný element pridať element typu *Pause*.

Pre splnenie požiadaviek príkladu č. 2 bola vytvorená Apex trieda *TestScheduler*. Odoslanie požiadavky HTTP bolo vykonané pomocou štandardných Apex tried *HttpRequest* a *HttpResponse*.

Rovnako ako v príklade č. 1 aj v tomto príklade zohráva veľkú úlohu denný limit na posielanie emailov. Ak by spoločnosť, ktorá si objednala danú aplikáciu bola malá spoločnosť s počtom klientov menej ako 5000, bolo by najvhodnejšie použiť pre danú aplikáciu programovacie riešenie. Dôvodom je hlavne fakt, že element typu *Action-Create HTTP Callout (Beta)* je stále iba v skúšobnej verzii a preto sa na jeho bezchybnú funkčnosť neodporúča spoliehať.

Avšak ak by spoločnosť, ktorá si objednala danú aplikáciu bola veľká spoločnosť s počtom klientov viac ako 5000, bolo by najvhodnejšie navrhnúť kombináciu deklaratívneho a programovacieho riešenia. Bol by vytvorený flow z ktorého by sa zavolala Apex trieda v ktorej by sa vykonalo odoslanie požiadavky HTTP. Odpoveď by bola odoslaná naspäť do flowu a následne by bola vyhodnotené či sa má poslať emailová notifikácia. Posielanie emailu by prebiehalo z flowu takže maximálny denný limit v produkčnej inštancii by bol 2 milióny.

4.3 Príklad číslo 3

V príklade číslo 3 bude navrhnuté riešenie vyvinuté pomocou deklaratívneho ale aj programovacieho spôsobu vývoja aplikácií v Salesforce.

4.3.1 Zadanie

Spoločnosť, ktorá prevádzkuje rôzne nákupné centra chce zlepšiť proces riešenia opráv výťahov a eskalátorov. V databáze bude uložený zoznam všetkých výťahov a eskalátorov v nákupných centrách prevádzkovaných danou spoločnosťou. Ak sa niektoré zo zariadení pokazí, bude možné vytvoriť tzv. servisný kontrakt priamo zo stránky záznamu zariadenia. Každé zariadenie bude môcť mať vytvorených viac servisných kontraktov, ale jeden

servisný kontrakt bude môcť byť priradený k iba jednému zariadeniu. Pri vytváraní servisného kontraktu bude potrebné vyplniť podrobný opis vady a tiež priložiť prílohy k servisnému kontraktu (napríklad fotografie). Prílohy budú prepojené so záznamom servisného kontraktu. Po vytvorení servisného kontraktu bude danému servisnému kontraktu automaticky priradená firma ktorá sa zaoberá opravovaním daného typu zariadenia. Zoznam firiem bude uložený v databáze. Firma bude môcť mať naraz priradené maximálne 3 servisné kontrakty, avšak k jednému servisnému bude môcť byť priradená vždy iba jedna firma.

4.3.2 Dátový model

Pre splnenie požiadaviek uvedených v zadaní je najvhodnejšie vytvoriť tri špecifické objekty na mieru: objekt Equipment (zariadenie), objekt Repair contract (servisný kontrakt) a objekt Company (firma). Objekt Repair contract bude spojovací objekt medzi objektami Equipment a Company. V Salesforce je spojovací objekt objektom, ktorý slúži na vytvorenie vzťahu mnoho k mnohým (many to many) medzi dvoma inými objektmi. Spojovací objekt obsahuje dva vzťahy master-detail, jeden ku každému z objektov, ktoré spája. Postup ako sa vytvára objekt v systéme Salesforce je podrobne vysvetlený v príklade č. 1. Ako prvý bude vytvorený objekt Equipment. Informácie o objekte budú vyplnené nasledovne:

- Label: Equipment
- Plural Label: Equipments
- Data Type: Auto Number
- Display Format: E-{0000}
- Starting Number: 1
- Allow Search: checked

Obdobne je potrebné vytvoriť aj dva ďalšie objekty: Repair contract a Company. Informácie o objektoch budú vyplnené nasledovne:

- Objekt *Repair contract*
    a. Label: Repair contract
    b. Plural Label: Repair contracts
    c. Data Type: Auto Number
    d. Display Format: RepairContract-{0000}
    e. Starting Number: 1
    f. Allow Search: checked

- Objekt *Company*
    a. Label: Company
    b. Plural Label: Companies
    c. Allow Search: checked

Všetky ostatné informácie budú ponechané nezmenené, podľa predvoleného nastavenia.

Následne je potrebné vytvoriť na objekte Repair contract dva vzťahy master-detail, jeden k objektu Company a druhý k objektu Equipment. Podrobný postup tvorby vzťahov je popísaný v príklade č. 1.

Na novo vytvorených objektoch budú vytvorené aj ďalšie polia uvedené v tabuľke nižšie.

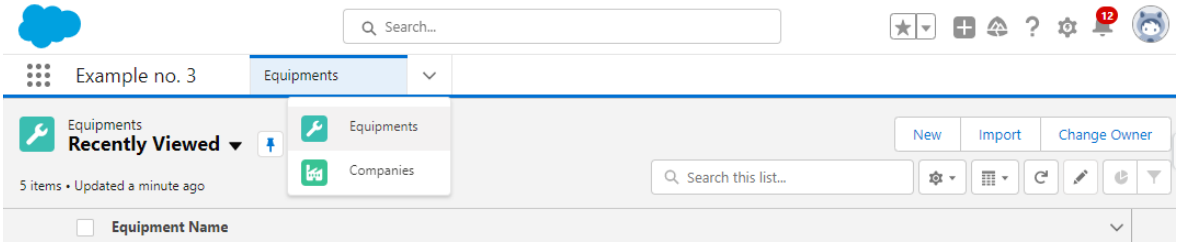
Tabuľka 4 Polia vytvorené pre objekty Equipment, Repair contract, Company

Objekt	**Názov poľa (Field label)**	**Typ poľa (Field Type)**
Equipment	Name	Text(50)
Equipment	Type	Picklist (Elevator, Escalator)
Repair contract	Description	Long Text Area(32768)
Company	Address	Text Area(255)
Company	Email address	Email
Company	Number of related repair contracts	Roll-Up Summary (COUNT Repair contract)
Company	Phone number	Phone
Company	Type	Picklist (Elevator, Escalator)

Prístup na novo vytvorené polia bol udelený tak, že profil System Administrator a Test User majú pre dané polia prístup čítaj a upravuj (Read+Edit). Pole *Name* a *Type* na objekte Equipment boli označené ako povinné. Na objekte Company boli označené ako povinné polia *Company Name* a *Type*.

Aby používateľ mohol s novo vytvoreným objektom pohodlne pracovať je potrebné vytvoriť najprv pre dané objekty záložky a potom samotnú aplikáciu. Podrobný postup vytvárania záložiek je opísaný v príklade č. 1. Pri vytváraní prvej záložky sa ako objekt vyberie objekt Equipment. Okrem objektu treba vybrať aj štýl záložky, pre objekt Equipment sa hodí štýl Wrench avšak je možné vytvoriť aj vlastný štýl. Pri vytváraní druhej záložky sa ako objekt vyberie objekt Repair contract. Okrem objektu treba vybrať aj štýl záložky, pre objekt Repair contract sa hodí štýl Form. Pri vytváraní tretej záložky sa ako objekt vyberie objekt Company. Okrem objektu treba vybrať aj štýl záložky, pre objekt Company sa hodí štýl Factory.

Následne je potrebné vytvoriť aplikáciu. Podrobný postup vytvárania aplikácie je opísaný v príklade č. 1. Názov aplikácie bude Example no. 3, čiže príklad číslo 3. V sekcii *Navigation Style* je potrebné zvoliť možnosť *Console navigation*. Následne je potrebné vybrať, ktoré záložky objektov budú dostupné v danej aplikácii, v tomto prípade to záložky pre objekty Equipment, Repair contract a Company. Posledným krokom pri vytváraní aplikácie je určenie profilov pre ktoré bude aplikácia dostupná. Je potrebné vybrať profily Test User a System Administrator. Kliknutím na uložiť sa aplikácia vytvorí a je pripravená na použitie.

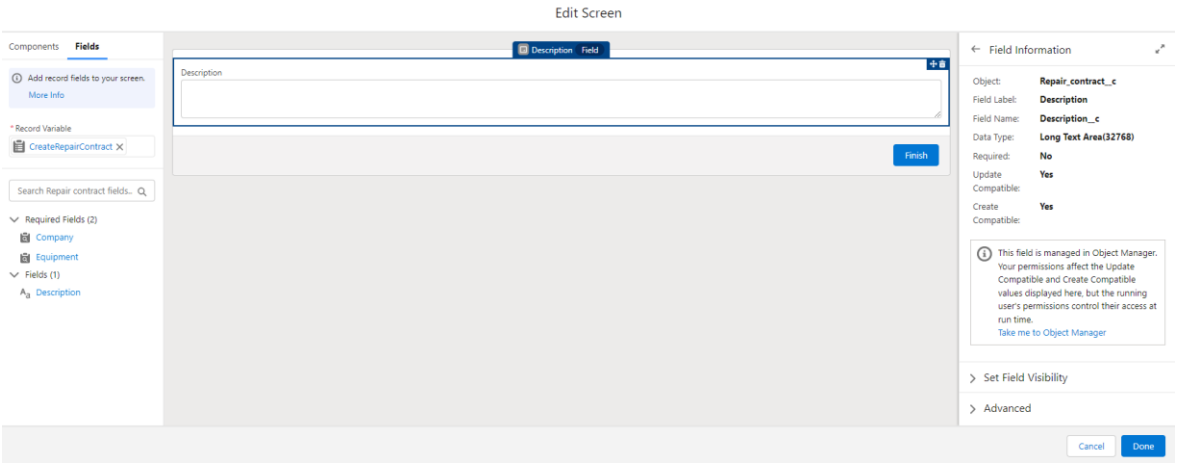


Obrázok 56 Aplikácia Example no. 3

4.3.3 Deklaratívny spôsob vývoja aplikácie

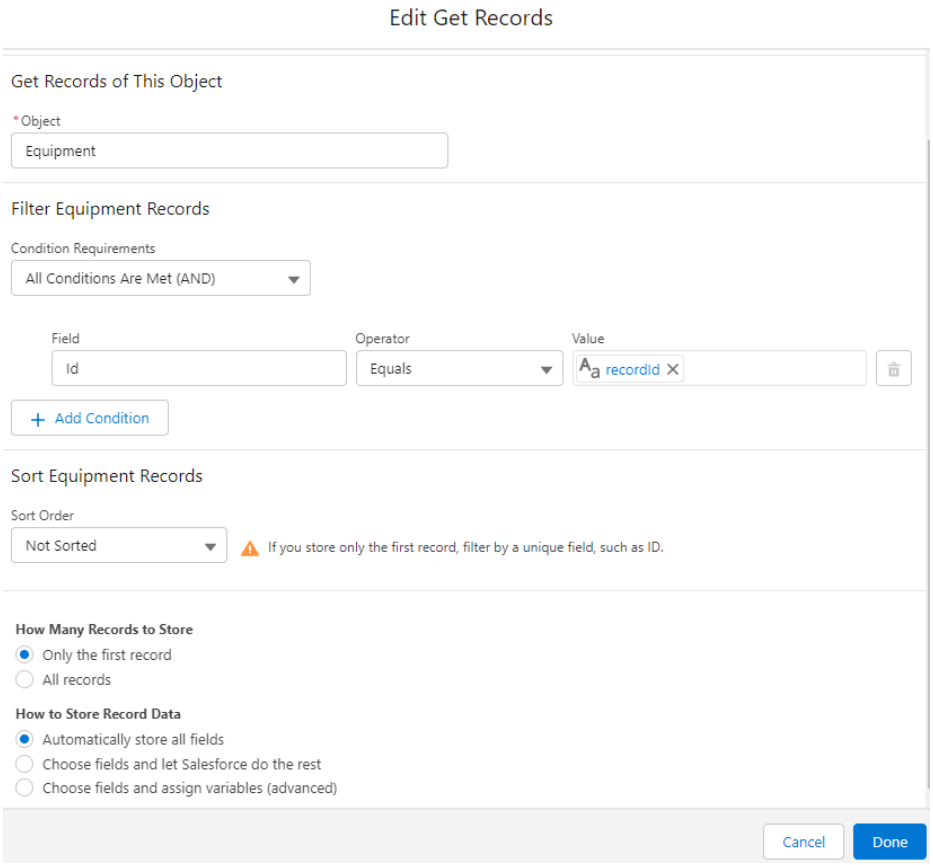
Vývoj aplikácie pre príklad č. 3 bude prebiehať rovnako ako v predošlých dvoch príklad, použitím nástroja Flow Builder, pretože Salesforce odporúča tento nástroj pre deklaratívny spôsob vývoja aplikácie ako overený postup (best practice). Prvým krokom je výber typu flowu. V zadaní príkladu č. 3 je uvedené, že možnosť vytvorenia nového servisného kontraktu má byť dostupná priamo zo stránky záznamu zariadenia. Preto ako typ flowu je potrebné vybrať *Screen Flow*, čiže Flow obrazoviek.

Predtým ako bude pridaný prvý element do flowu je potrebné vytvoriť premennú typu záznam s názvom *CreateRepairContract*. Pri vytváraní premennej bude do poľa objekt zadaná hodnota Repair contract. Ako prvý element bude pridaný do flowu element typu *Screen* s názvom *DescriptionScreen*. Na pravej strane okna elementu sa nachádza panel vlastností elementu. V záložke *Configure Footer* je potrebné zvoliť aby tlačidlá *Previous* a *Pause* boli skryté. Na ľavej strane okna elementu sa nachádza knižnica komponentov. Do elementu bude pridaný komponent typu *Long Text Area(32768)* s názvom *Description*. Do tohto komponentu je potrebné vložiť premennú záznamu *CreateRepairContract*. Prehľad okna komponentu *DescriptionScreen* je uvedený na obrázku č. 57.



Obrázok 57 Element typu Screen

Ako ďalší element bude do flowu pridaný element typu Get Records. Názov elementu bude *GetEquipment*. Tento element bude slúžiť na získanie záznamu objektu Equipment z ktorého bol flow spustený. Najprv je ale potrebné vytvoriť premennú typu Text s názvom *recordId*. Pri vytváraní tejto premennej je potrebné zaškrtnúť možnosť *Available for input*. Následne v elemente *GetEquipment* bude záznam nájdený na základe hodnoty premennej *recordId*. Konfigurácia elementu je znázornená na obrázku č. 58.



Obrázok 58 Element typu Get Records

Následne bude pridaný element typu Decision s názvom *EquipmentCheck*. Tento element bude obsahovať dve rozhodovacie vetvy.

Prvá rozhodovacia vetva bude nazvaná *EquipmentIsNull*, a bude obsahovať podmienku, ktorá skontroluje či vôbec bol úspešne získaný nejaký záznam v elemente *GetEquipment*.

Ak je prázdny, čo znamená, že žiadny záznam nebol získaný, flow pokračuje v rozhodovacej vetve *EquipmentIsNull*. V tejto vetve sa nachádza element typu Screen s názvom *EquipmentIsNullScreen*. Do tohto elementu bude pridaný komponent typu Display Text s názvom *EquipmentIsNullText* a hodnotou *Equipment was not set!*

Druhá rozhodovacia vetva bude nazvaná *EquipmentIsNotNull*, a nebude obsahovať žiadnu podmienku, čo znamená že sa vykoná iba v prípade ak záznam získaný z elementu *GetEquipment* nebude prázdny. V tejto vetve je vykonávaný zvyšok flowu.

Ako ďalší element bude do flowu pridaný element typu Get Records. Názov elementu bude *GetCompany*. Tento element slúži na získanie vhodného záznamu objektu Company, ktorí bude neskôr priradený k novo vytvorenému záznamu objektu Repair Contract. V zadaní príkladu č. 3 je uvedené, že po vytvorení servisného kontraktu bude danému servisnému kontraktu automaticky priradená firma ktorá sa zaoberá opravovaním daného typu zariadenia a tiež že, firma bude môcť mať naraz priradené maximálne 3 servisné kontrakty. Preto záznamy objektu Company budú vyfiltrované nasledovne:

1. Hodnota poľa *Type* záznamu objektu Company musí byť rovná hodnote poľa *Type* záznamu objektu Equipment.
2. Hodnota poľa *Number of related repair contracts* musí byť menšia ako tri

V možnosti s názvom *How Many Records to Store* musí byť vybraná možnosť *Only the first record*, pretože novo vytvorenému záznamu objektu Repair contract chceme priradiť iba jeden záznam objektu Company.

Následne bude pridaný element typu Decision s názvom *CompanyCheck*. Tento element bude obsahovať dve rozhodovacie vetvy.

Prvá rozhodovacia vetva bude nazvaná *CompanyIsNull*, a bude obsahovať podmienku, ktorá skontroluje či vôbec bol úspešne získaný nejaký záznam v elemente *GetCompany*.

Ak je prázdny, čo znamená, že žiadny záznam nebol získaný, flow pokračuje v rozhodovacej vetve *CompanyIsNull*. V tejto vetve sa nachádza element typu Screen s názvom

CompanyIsNullScreen. Do tohto elementu bude pridaný komponent typu Display Text s názvom *CompanyIsNullText* a hodnotou *Company was not set!*

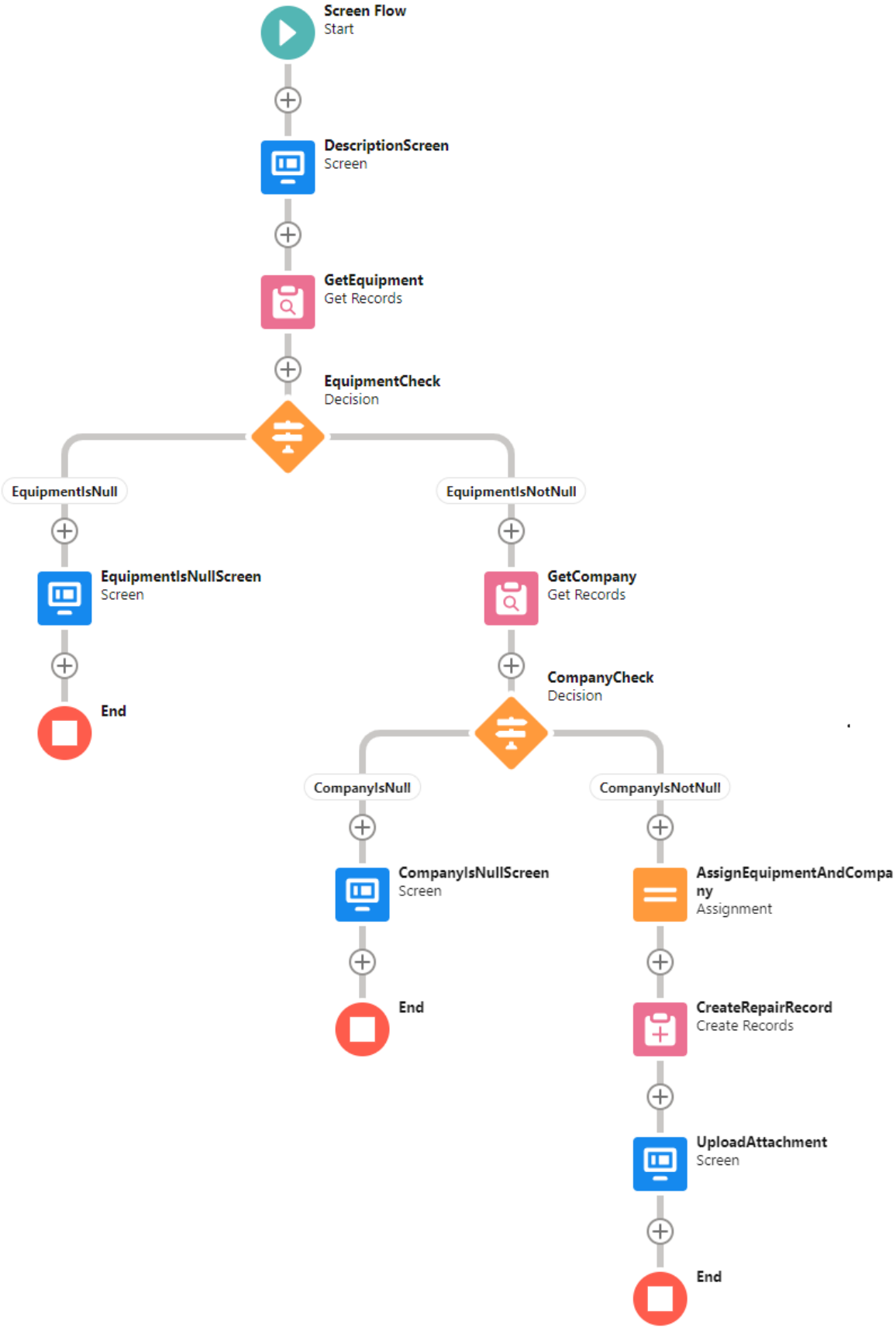
Druhá rozhodovacia vetva bude nazvaná *CompanyIsNotNull*, a nebude obsahovať žiadnu podmienku, čo znamená že sa vykoná iba v prípade ak záznam získaný z elementu *GetCompany* nebude prázdny. V tejto vetve je vykonávaný zvyšok flowu.

Ako ďalší element bude do flowu pridaný element typu Assignment s názvom *AssignEquipmentAndCompany*. V tomto elemente budú vyplnené polia *Equipment* a *Company* premennej *CreateRepairContract* hodnotami premenných získaných v elementoch *GetEquipment* a *GetCompany*.

Následne bude do flowu pridaný element type Create Record s názvom *CreateRepairRecord*. V možnosti *How Many Records to Create* je potrebné vybrať možnosť *One*, pretože je potrebné vytvoriť iba jeden záznam. V možnosti *How to Set the Record Fields* je potrebné vybrať možnosť *Use separate resources, and literal values*. Ako objekt je potrebné vybrať Repair contract a následne je potrebné naplniť polia Company, Description a Equipment objektu Repair contract hodnotami polí s rovnakými názvami premennej *CreateRepairDescription*. Je tiež potrebné zaškrtnúť možnosť *Manually assign variables* a do poľa *Store Repair contract ID in Variable* uviesť premennú v ktorej bude uložený identifikátor nového záznamu objektu Repair contract. Typ tejto premennej bude Text a názov *repairContractId*.

Ako posledný element bude do flowu pridaný element typu Screen s názvom *UploadAttachment*. V záložke *Configure Footer* je potrebné zvoliť aby tlačidlo *Pause* bolo skryté. Do elementu bude pridaný komponent typu File Upload s názvom *UploadFile*. Na pravej strane okna elementu sa nachádza panel vlastností elementu. Do poľa *File Upload Label* bude vložený text *Upload Attachment*. Do poľa *Allow Multiple Files* bude vložená hodnota *true*, pretože je potrebné aby bolo možné vkladať súčasne viac súborov. Do poľa *Related Record ID* bude vložená hodnota premennej *repairContractId*, pretože je potrebné aby nahrané súbory boli prepojené s novo vzniknutým záznamom objektu Repair contract.

Tým je vývoj flowu ukončený, je potrebné ho ešte uložiť a aktivovať. Pri ukladaní je potrebné zadať názov flowu, ktorý bude v tomto prípade *Example3Flow*. V nástroji *Flow Builder* je tiež vstavaný nástroj na ladenie flowu. Obrazovka tohto nástroja sa otvorí po kliknutí na tlačidlo *Debug*. Po otvorení obrazovky je možné nakonfigurovať podmienky pri ktorých bude flow ladený. Vizualizácia finálnej verzie flowu spolu s premennými je znázornená na obrázku č. 59.



Obrázok 59 Príklad č. 3 vypracovaný pomocou nástroja Flow Builder

Avšak aby bolo možné spustiť flow priamo zo stránky záznamu objektu Equipment je potrebné vytvoriť *Quick Action*, čiže tzv. rýchlu akciu. V nastaveniach, v záložke Object Manager je potrebné vyhľadať objekt Equipment. Následne je potrebné sa na stránke objektu prepnúť na záložku *Buttons, Links, and Actions* a kliknúť na tlačidlo *New Action*. Ako *Action Type* je potrebné vybrať možnosť *Flow*. V možnosti *Flow* je potom potrebné vybrať daný flow, v tomto prípade *Example3Flow*. Meno akcie bude *Create Repair contract via Flow*.

Po vytvorení akcie je potrebné ju ešte pridať do užívateľského rozhrania, na stránku záznamu objektu Equipment. Na stránke objektu sa kliknutím na záložku *Page Layouts* zobrazí zoznam stránok rozložení. Po kliknutí na názov stránky je potrebné z kategórie *Mobile & Lightning Actions* vybrať akciu *Create Repair contract via Flow* a pridať ju medzi ostatné štandardné akcie. Po uložení bude akcia dostupná na stránke záznamu objektu Equipment.

4.3.4 Programovací spôsob vývoja aplikácie

V zadaní príkladu č. 3 je uvedené, že možnosť vytvorenia nového servisného kontraktu má byť dostupná priamo zo stránky záznamu zariadenia, takže bude potrebné vytvoriť komponent v užívateľskom prostredí. Systém Salesforce podporuje 3 typy komponentov: Visualforce komponenty, Aura komponenty a Lightning Web komponenty. Salesforce ako overený postup (best practice) odporúča použiť pre vytváranie komponentov na užívateľskom prostredí Lightning Web komponenty (LWC).

Pre príklad č. 3 bol vytvorený Lightning Web komponent s názvom *CreateRepairContractLWC*, ktorý bude umožňovať používateľom vytvoriť záznam objektu Repair contract prostredníctvom rýchlej akcie umiestnenej na stránke záznamu objektu Equipment. Komponent sa skladá zo súboru HTML (*createRepairContractLWC.html*), súboru JavaScript (*createRepairContractLWC.js*) a súboru metadát (*createRepairContractLWC.js-meta.xml*).

Súbor HTML definuje štruktúru používateľského rozhrania komponentu. Na štylizáciu prvkov používa framework Lightning Design System (SLDS). Vo vnútri HTML súboru sa nachádza štandardný komponent *lightning-quick-action-panel*, ktorý slúži ako kontajner pre obsah. Panel má hlavičku s textom *Create Repair contract via LWC*. Obsah panela sa podmienečne vykresľuje na základe hodnoty vlastnosti *isFirstScreen*.

Ak je hodnota premennej *isFirstScreen* rovná boolean hodnote *true*, tak sa na obrazovke, ktorú je možno teda nazvať aj prvá obrazovka, vykreslí štandardný komponent *lightning-textarea*, do ktorého môžu používatelia zadať opis zmluvy o oprave (hodnota zadaná do

tohto komponentu bude neskôr priradená poľu *Description* na novo vzniknutom zázname objektu Repair contract.). Hodnota vložená do komponentu je viazaná na premennú *repairContractDescription* z JavaScript súboru. Pri zmene hodnoty, ktorá bola vložená do komponentu sa zavolá metóda v JavaScript súbore s názvom *handleDescriptionChange*.

Ak je hodnota premennej *isFirstScreen* rovná boolean hodnote *false*, tak sa na obrazovke, ktorú je možno nazvať aj druhá obrazovka, vykreslí štandardný komponent *lightning-file-upload,* ktorý umožňuje používateľom nahrávať súbory ako prílohy k zmluve o oprave. Komponent je nakonfigurovaný tak, aby bolo umožnené nahrávanie viacerých súborov. Metóda v JavaScript súbore s názvom *handleUploadFinished* sa zavolá, keď je nahrávanie súborov dokončené.

Spodná časť panela obsahuje dve tlačidlá. Obe tlačidlá sa zobrazujú iba na prvej obrazovke. Tlačidlo *Cancel* po kliknutí vyvolá metódu v JavaScript súbore s názvom *handleClose* a tlačidlo *Next* po kliknutí vyvolá metódu v JavaScript súbore s názvom *handleValidate*. Na tlačidlo *Next* nie je možné kliknúť ak je hodnota premennej *saveDisabled* rovná boolean hodnote *true*.

```
<template>
    <lightning-quick-action-panel header="Create Repair contract via LWC">
        <!-- First screen where Description input is displayed -->
        <template if:true={isFirstScreen}>
            <div class="slds-m-around_small">
                <lightning-textarea label="Description" placeholder="type here..." required value={repairContractDescription} onchange={handleDescriptionChange}></lightning-textarea>
            </div>
        </template>
        <!-- Second screen where upload file option is displayed -->
        <template if:false={isFirstScreen}>
            <div class="slds-m-around_small">
                <lightning-file-upload label="Attachment" name="fileUploader" record-id={repairContractId} onuploadfinished={handleUploadFinished} multiple></lightning-file-upload>
            </div>
        </template>
        <!-- Footer buttons that are displayed only on first screen -->
        <div slot="footer">
            <template if:true={isFirstScreen}>
                <lightning-button variant="neutral" label="Cancel" onclick={handleClose}></lightning-button>
                <lightning-button variant="brand" label="Next" type="submit" onclick={handleValidate} disabled={saveDisabled} class="slds-m-left_x-small"></lightning-button>
            </template>
        </div>
    </lightning-quick-action-panel>
</template>
```

Obrázok 60 Príklad č. 3 - HTML súbor Lightning Web komponentu *createRepairContractLWC*

Súbor JavaScript vykonáva funkciu kontrolera komponentu. Trieda definuje niekoľko premenných vrátane *repairContractDescription*, *equipmentRecord*, *companyRecordId*, *repairContractId* a *isFirstScreen*. Tieto premenné sa používajú na ukladanie údajov a riadenie správania komponentu. Komponent obsahuje aj niekoľko metód:

- *getEquipmentRecord* - Táto metóda je napojená na funkciu *getRecord* zo služby Lightning Data Service (LDS) a načíta záznam objektu Equipment na základe premennej *recordId* (premenná *recordId* obsahuje identifikátor aktuálneho záznamu z ktorého bol komponent spustený).

- *handleDescriptionChange* - Táto metóda sa zavolá, keď sa zmení hodnota vložená do komponentu *lightning-textarea*, a zodpovedajúcim spôsobom aktualizuje premennú *repairContractDescription*.
- *handleValidate* - Táto metóda sa volá, keď používateľ klikne na tlačidlo *Next*. Overí vstupný komponent a zavolá metódu *getCompanyRecordAndCreateRecord*, ak sú všetky vstupné komponenty platné (platnosť vstupných komponentov znamená že napríklad ak je vstupný komponent označený ako povinný, metóda *getCompanyRecordAndCreateRecord* nebude zavolaná pokiaľ vo vstupných komponentoch nebude vyplnená nejaká hodnota).
- *getCompanyRecordAndCreateRecord* - Táto metóda načíta záznam objektu Company z Apex triedy s názvom *CreateRepairContractLWCController* na základe typu zariadenia. Pre načítanie záznamu použije metódu *getCompany* v Apex triede a následne zavolá JavaScript metódu *createRepairContract*.
- *createRepairContract* - Táto metóda vytvára nový záznam objektu Repair contract pomocou funkcie *createRecord* z LDS. Následne metóda nastaví potrebné hodnoty polí záznamu objektu Repair contract. Do poľa *Description* bude uložená hodnota vložená používateľom do komponentu *lightning-textarea*, do poľa *Equipment* bude uložená hodnota premennej *recordId* a do poľa *Company* bude vložená hodnota získaná z metódy *getCompany* Apex triedy *CreateRepairContractLWCController*.
- *handleUploadFinished* - Táto metóda sa zavolá, keď sa ukončí nahrávanie súboru. Zobrazí hlásenie o úspešnom vytvorení záznamu objektu Repair contract, zavolá *handleClose* metódu na vynulovanie polí a zatvorenie panelu rýchlej akcie a obnoví stránku záznamu.
- *handleClose* - Táto metóda vynuluje vstupné polia a odošle akčnú udalosť *close* na zatvorenie panelu rýchlej akcie.
- *refreshPage*: Táto metóda obnoví stránku záznamu.

```
public with sharing class CreateRepairContractLWCController {
  //Retrieve company record that has Type equal to Equipment Type and Number of related repair contract records less than 3
  @AuraEnabled
  public static Id getCompany(String equipmentTypeField) {
    List<Company__c> listOfCompanies = [SELECT Id FROM Company__c WHERE Type__c = :equipmentTypeField AND Number_of_related_repair_contracts__c < 3 LIMIT 1];
    if (!listOfCompanies.isEmpty()) {
      return listOfCompanies[0].Id;
    } else {
      AuraHandledException e = new AuraHandledException('No company record found with matching criteria.');
      e.setMessage('No company record found with matching criteria.');
      throw e;
    }
  }
}
```

Obrázok 61 Príklad č. 3 - Apex trieda CreateRepairContractLWCController

```
import { LightningElement, api, wire } from 'lwc';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import { CloseActionScreenEvent } from 'lightning/actions';
import { getRecord } from 'lightning/uiRecordApi';
import { createRecord } from 'lightning/uiRecordApi';
import EQUIPMENT_TYPE from '@salesforce/schema/Equipment__c.Type__c';
import REPAIR_CONTRACT_OBJECT from '@salesforce/schema/Repair_contract__c';
import getCompany from '@salesforce/apex/CreateRepairContractLWCController.getCompany';

export default class CreateRepairContractLWC extends LightningElement {
    //Api decorator is used to expose a variable or functions publically and make properties reactive.
    @api recordId;
    repairContractDescription;
    equipmentRecord;
    companyRecordId;
    repairContractId;
    isFirstScreen = true;

    //Method to retrieve equipment record
    //To read Salesforce data, Lightning web components use a reactive wire service.
    //When the wire service provisions data, the component rerenders.
    @wire(getRecord, {recordId: "$recordId", fields: EQUIPMENT_TYPE})
    getEquipmentRecord({error, data}) {
        if (data) {
            this.equipmentRecord = data;
        } else if (error) {
            this.showToast("Error retrieving equipment record", error.body.message, "error");
        }
    }

    //This method is used for storing description from UI
    handleDescriptionChange(event){
        this.repairContractDescription = event.target.value;
    }

    //This methods validates the input, then calls method for retrieving company record and creating repair contract record
    handleValidate(){
        const textareaCmp = this.template.querySelectorAll("lightning-textarea");
        const fileUploadCmp = this.template.querySelector("lightning-file-upload");

        const allValid = [...textareaCmp, fileUploadCmp]
            .reduce((validSoFar, inputCmp) => {
                inputCmp.reportValidity();
                return validSoFar && inputCmp.checkValidity();
            }, true);
        if (allValid) {
            this.getCompanyRecordAndCreateRecord();
        }
    }

    //This method retrieves company record from Apex class, then call method for creating repair contract record
    getCompanyRecordAndCreateRecord() {
        const equipmentTypeField = this.equipmentRecord.fields.Type__c.value;
        getCompany({equipmentTypeField})
            .then(result => {
                this.companyRecordId = result;
                this.createRepairContract();
            }).catch(error => {
                this.showToast("Error retrieving company record", error.body.message, "error");
            });
    }
```

Obrázok 62 Príklad č. 3 - Lightning Web komponent CreateRepairContractLWC - 1. časť

Pre Lightning Web komponent *CreateRepairContractLWC* bola vytvorená Apex trieda *CreateRepairContractLWCController*, ktorá v metóde *getCompany* pred vytvorením záznamu objektu Repair contract načíta záznam objektu Company na základe hodnoty poľa Type záznamu objektu Equipment. Na strane servera vykonáva potrebné načítanie a overovanie údajov, aby sa zabezpečila integrita a konzistentnosť aplikačnej logiky. Pre danú Apex triedu

bola vytvorená aj testovacia Apex trieda s názvom *CreateRepairContractLWCController-Test*, ktorá pokrýva *CreateRepairContractLWCController* na sto percent.

```
    //This method creates new repair contract record
    createRepairContract() {
        const fields = {'Description__c' : this.repairContractDescription,
                        'Equipment__c' : this.recordId,
                        'Company__c' : this.companyRecordId};
        const recordInput = { apiName: REPAIR_CONTRACT_OBJECT.objectApiName, fields };
        createRecord(recordInput)
            .then(response  => {
                this.repairContractId = response.id;
                this.isFirstScreen = false;
            })
            .catch(error => {
                this.showToast("Error creating record", error.body.message, "error");
            });
    }

    //This method handles events after upload is finished
    handleUploadFinished() {
        this.showToast("Success!", "Record was created", "success");
        this.handleClose();
        this.refreshPage();
    }

    //This method resets input field and close the quick action
    handleClose() {
        this.resetFields();
        this.dispatchEvent(new CloseActionScreenEvent());
    }

    //This method refreshes page
    refreshPage() {
        setTimeout(() => {
            eval("$A.get('e.force:refreshView').fire();");
        }, 1000);
    }

    //This method resets input fields
    resetFields() {
        const inputFields = [...this.template.querySelectorAll("lightning-textarea")];
        if (inputFields) {
            inputFields.forEach(field => {
                field.value = "";
            });
        }
    }

    //This method show toast message on the UI
    showToast(title, message, type, behavior, messageData) {
        const variant = type || "error";
        const mode = behavior || (variant === "error" ? "sticky" : "dismissible");
        const evt = new ShowToastEvent({title, message, variant, mode, messageData});
        this.dispatchEvent(evt);
    }
}
```

Obrázok 63 Príklad č. 3 - Lightning Web komponent CreateRepairContractLWC - 2. časť

Avšak aby bolo možné spustiť Lightning Web komponent priamo zo stránky záznamu objektu Equipment je potrebné vytvoriť *Quick Action*, čiže tzv. rýchlu akciu. Ako *Action Type* je potrebné vybrať možnosť *Lightning Web Component*. V možnosti *Lightning Web Component* je potom potrebné vybrať daný komponent, v tomto prípade *CreateRepairContractLWC*. Meno akcie bude *Create Repair contract via LWC* .

Po vytvorení akcie je potrebné ju ešte pridať do užívateľského rozhrania, na stránku záznamu objektu Equipment. Na stránke objektu sa kliknutím na záložku *Page Layouts* zobrazí zoznam stránok rozložení. Po kliknutí na názov stránky je potrebné z kategórie *Mobile & Lightning Actions* vybrať akciu *Create Repair contract via LWC* a pridať ju medzi ostatné štandardné akcie. Po uložení bude akcia dostupná na stránke záznamu objektu Equipment.

4.3.5 Zhodnotenie

Pomocou nástroja Flow Builder bol vytvorený flow typu Screen Flow, ktorý spĺňa požiadavky príkladu č. 3 z pohľadu deklaratívneho spôsobu vývoja. Pre zobrazenie dát na užívateľskom prostredí bol použitý element typu Screen. Nahrávanie súborov prebieha pomocou štandardného komponentu *Upload File*, ktorý bol vložený do elementu Screen.

Pre splnenie požiadaviek príkladu č. 3 programovacím spôsobom bol vytvorený Lightning Web komponent s názvom *CreateRepairContractLWC* a Apex trieda s názvom *CreateRepairContractLWCController*. Nahrávanie súborov prebieha pomocou štandardného komponentu *lightning-file-upload*, ktorý bol vložený do HTML súboru Lightning Web komponentu.

Aby bolo možné spustiť funkcionalitu priamo zo stránky záznamu objektu Equipment, bolo potrebné vytvoriť tzv. rýchlu akciu pre flow ako aj pre Lightning Web komponent.

Po analýze riešení je možné skonštatovať, že zadanie príkladu č. 3 neobsahuje tak špecifickú funkcionalitu, ktorá by vyžadovala vytvorenie Lightning Web komponentu, preto by bolo najvhodnejšie pre tento príklad použiť deklaratívny spôsob vývoja aplikácií. V prípade ak by bolo v zadaní špecifikované, že má byť daná funkcionalita dostupná napríklad z iného komponentu, bolo by nutné použiť Lightning Web komponent. Navyše Salesforce odporúča použiť vždy deklaratívny spôsob vývoja aplikácií ak výsledné riešenie je jednoduchšie a prehľadnejšie ako riešenie programovacím spôsobom vývoja aplikácií.

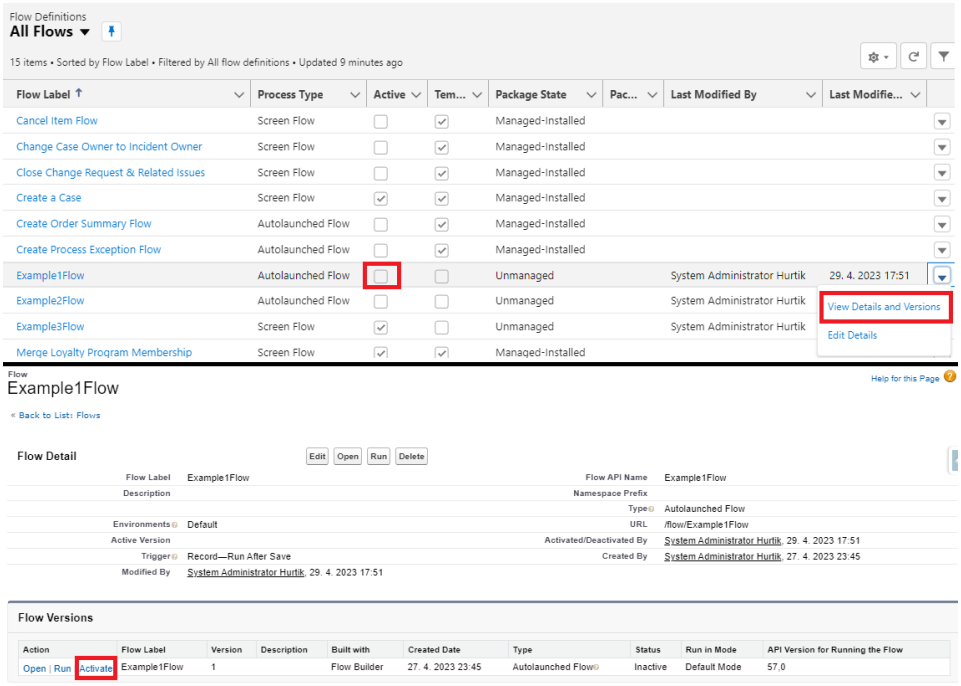
4.4 Testovanie

Aby čitatelia tej práce mohli sami otestovať funkcionalitu vyvinutých aplikácií bol v Salesforce inštancii vytvorený testovací používateľ. Pre prihlásenie sa pomocou tohto používateľa do Salesforce inštancie je potrebné v internetovom prehliadači prejsť na webovú stránku https://login.salesforce.com/. Prihlasovacie meno používateľa je testusersalesforce2023@seznam.cz.diplomovapracehurtik a heslo je test123*test123*. Toto heslo bude zmenené z bezpečnostných dôvodov dňa 01.07.2023. Po prihlásení je možné otvoriť jednu z troch aplikácií pomocou Spúšťača aplikácií (9 bodiek v ľavej hornej časti obrazovky).

4.4.1 Testovanie príkladu č. 1

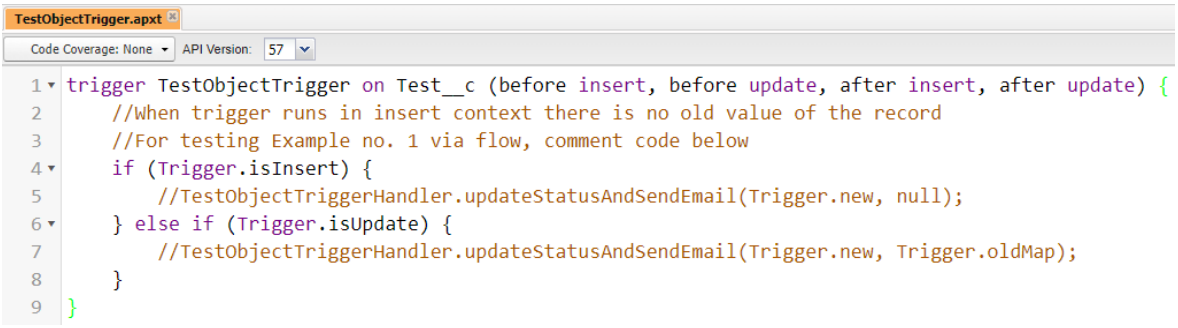
Ešte pred začatím testovania príkladu č. 1 je potrebné vybrať ktorý spôsob vývoja bude otestovaný. Ak má byť otestovaný deklaratívny spôsob vývoja aplikácie je potrebné sa uistiť či je daný flow aktívny a či je kód daného spúšťača zakomentovaný. V prípade že má byť testovaný programovací spôsob vývoja aplikácie, flow musí byť neaktívny, kód daného spúšťača odkomentovaný.

V nastaveniach, vyhľadaním a zvolením možnosti *Flows* sa zobrazí zoznam flowov. Stĺpec *Active* označuje či je daný flow aktívny alebo nie. Kliknutím na šípku v pravej časti riadku daného flowu sa zobrazia možnosti. Je potrebné kliknúť na možnosť *View Details and Versions*. Následne kliknutím na tlačidlo Activate sa flow aktivuje (viď. Obrázok č. 64).



Obrázok 64 Aktivácia/Deaktivácia flowu

Kód spúšťača je možné zakomentovať/odkomentovať vo vývojárskej konzole (Developer Console). Po otvorení vývojárskej konzole je potrebné kliknúť na možnosť *File* a následne na *Open Resource*. Do vyhľadávača je potrebné uviesť názov spúšťača, čiže *TestObjectTrigger*. Po kliknutí na tlačidlo Open sa spúšťač otvorí a je možné ho zakomentovať. Jednoriadkový komentár v jazyku Apex sa označuje dvomi lomkami za sebou (viď obrázok č. 65).



Obrázok 65 Zakomentovaný spúšťač TestObjectTrigger vo vývojárskej konzole

Po zakomentovaní/odkomentovaní je potrebné spúšťač uložiť klávesovou skratkou CTRL + S. Pri testovaní príkladu č. 1 je potrebné brať do úvahy, že maximálny počet všetkých emailov odoslaných z Apexu v jeden deň je v danej Salesforce inštancii iba 15.

4.4.2 Testovanie príkladu č. 2

Pre príklad číslo 2 bol vytvorený flow *Example2Flow*, ktorý sa spustí každý deň o 20:00 podľa časového pásma predvoleného pre danú Salesforce inštanciu. V tomto prípade ide o časové pásmo *(GMT+02:00) Central European Summer Time (Europe/Bratislava)*. Zmena z letného na zimný čas prebieha v Salesforce inštancii automaticky, takže sa flow spustí vždy o 20:00 bez ohľadu na to či je práve letný čas alebo zimný.

Programovacím spôsobom bola pre príklad číslo 2 vytvorená Apex trieda *TestScheduler*, ktorej spustenie je naplánované každý deň o 8:00 podľa časového pásma používateľa, ktorý spustenie danej Apex triedy naplánoval. V tomto prípade ide rovnako ako aj pri flowe o časové pásmo *(GMT+02:00) Central European Summer Time (Europe/Bratislava)*.

Pri testovaní príkladu č. 2 je potrebné brať ohľad na to, že maximálny počet všetkých emailov odoslaných z Apexu v jeden deň je v danej Salesforce inštancii iba 15. V prípade ak by bol počet všetkých záznamov objektu Investor desať, a všetci by dosiahli požadovaný profit, ráno o 8:00 by bolo odoslaných desať emailových notifikácií avšak večer o 20:00 by bolo odoslaných iba päť emailových notifikácií, keďže limit je 15. Taktiež je potrebné brať ohľad na to, že daný limit sa vzťahuje na celú Salesforce inštanciu, takže pre všetky tri príklady.

4.4.3 Testovanie príkladu č. 3

Pre príklad č. 3 boli vytvorené dve rôzne rýchle akcie, ktoré boli pridané na stránku záznamu objektu Equipment a preto je možné testovať oba spôsoby vývoja aplikácie bez obmedzení. Avšak treba brať ohľad na to, že pri vytváraní záznamu objektu Repair contract sa na základe hodnoty poľa *Type* priraďuje novovznikajúcemu záznamu príslušný záznam objektu Company. Avšak to sa stane iba v prípade ak daný záznam objektu Company má priradené menej ako 3 záznamy objektu Repair contract. V prípade ak nebude nájdený žiadny záznam objektu Company vyhovujúci kritériám, záznam objektu Repair contract nebude vytvorený.

Oba varianty príkladu č. 3 používajú štandardné nahrávanie súborov. Súčasne je možné nahrať maximálne 10 súborov. Maximálna veľkosť jedného súboru je 2 GB avšak je potrebné brať ohľad na to že testovacia inštancia Salesforce má dostupné úložisko s kapacitou iba 20 MB.

ZÁVER

Cieľom tejto práce bolo oboznámiť čitateľov tejto práce s možnosťami vývoja aplikácií v CRM systéme Salesforce. Pre dosiahnutie tohto cieľa bola vypracovaná literárna rešerš, ktorej výsledky boli zhrnuté v teoretickej časti tejto práce. V úvode teoretickej časti bol definovaný pojem CRM systém a následne bol prezentovaný súhrn histórie CRM systémov. V ďalšej časti tejto práce boli postupne predstavené štyri CRM systémy, ktoré patria medzi najpoužívanejšie na trhu. Postupne boli opísané systémy SAP CRM, Microsoft Dynamics CRM, Oracle CRM a Adobe Experience cloud. Následne bol pre každý zo systémov vysvetlený jeho dátový model a definované možnosti vývoja aplikácií v danom systéme. Pre každý systém bol uvedený aj jednoduchý príklad aplikácie s jej následným vypracovaním.

V nasledujúcej kapitole bol predstavený suverénne najpoužívanejší CRM systém na trhu, Salesforce. Postupne bol čitateľ oboznámený s históriou a dátovým modelom daného CRM systému. Následne boli prezentované možnosti vývoja aplikácií v Salesforce. Z pomedzi deklaratívnych spôsobov vývoja aplikácií boli okrem iných opísané nástroje Flow a Process Builder. Z programovacích spôsobov vývoja aplikácií bol ako prvý predstavený programovací jazyk Apex, následne webový framework Visualforce a Lightning component framework programovacieho jazyka Java Script.

Poznatky získané z teoretickej časti tejto práce boli demonštrované v jej praktickej časti. V úvode praktickej časti bol detailne popísaný postup tvorby Salesforce inštancie a následne bol uvedený podrobný postup prepájania Salesforce inštancie a vývojového prostredia VS Code. V nasledujúcej kapitole boli predstavené tri príklady aplikácií v systéme Salesforce. Pre každý z príkladov bolo uvedené zadanie, po ktorom bol podrobne opísaný dátový model daného príkladu. Následne bol každý príklad vypracovaný pomocou deklaratívneho a programovacieho spôsobu vývoja aplikácií. Ku každému príkladu bolo uvedené aj jeho zhodnotenie, kde boli porovnané oba spôsoby vypracovania a následne bolo konštatované, ktorý spôsob vypracovania by bol najvhodnejší ak by sa jednalo o skutočnú požiadavku klienta. Konštatovania boli podložené pádnymi dôvodmi a argumentami.

V poslednej kapitole praktickej časti boli uvedené prihlasovacie údaje do Salesforce inštancie vytvorenej v úvode praktickej časti. Následne bol vysvetlený postup testovania každého z príkladov vypracovaných v praktickej časti.

Po prečítaní tejto práce získajú čitatelia znalosti ako efektívne využívať možnosti systému Salesforce na uspokojenie vyvíjajúcich sa potrieb svojich klientov.

ZOZNAM POUŽITEJ LITERATÚRY

[1] PEPPERS, Don a Martha ROGERS. *Managing Customer Relationships: A Strategic Framework*. Druhá edícia. USA: Wiley, 2011. ISBN 978-0470423479.

[2] PEELEN, Ed a Rob BELTMAN. *Customer Relationship Management*. Druhá edícia. Spojené kráľovstvo: Pearson Education, 2013. ISBN 978-0273774952.

[3] BOSE, Ranjit. Customer relationship management: Key components for IT success. *Industrial Management & Data Systems*. Spojené Kráľovstvo: Emerald Group Publishing, 2002, (102), 89-97. ISSN 0263-5577. Dostupné z: doi:10.1108/02635570210419636

[4] XU, Yurong, Binshan LIN, David C. CHOU a David C. YEN. Adopting customer relationship management technology. *Industrial Management & Data Systems*. Spojené Kráľovstvo: Emerald Group Publishing, 2002, (102), 442-452. ISSN 0263-5577. Dostupné z: doi:10.1108/02635570210445871

[5] PEARCE, Michael. *Customer Relationship Management: How To Develop and Execute a CRM Strategy*. USA: Business Expert Press, 2021. ISBN 1953349641.

[6] *Creatio* [online]. [cit. 2023-05-15]. Dostupné z: https://www.creatio.com/industries/banking

[7] BARAN, Roger J. a Robert J. GALKA. *Customer Relationship Management: The Foundation of Contemporary Marketing Strategy*. Druhá edícia. USA: Routledge, 2016. ISBN 978-1138919525.

[8] *WinWorld* [online]. [cit. 2023-05-15]. Dostupné z: https://winworldpc.com/product/act/110-dos

[9] PINGREY, Jess. The History of CRM From the 1950s to Today. *Fit Small Business* [online]. USA, August 31, 2022 [cit. 2023-02-22]. Dostupné z: https://fitsmallbusiness.com/history-of-crm/

[10] JOHANNES, Stephen. *SAP CRM Technical Principles and Programming*. Nemecko: SAP PRESS, 2013. ISBN 978-1592294398.

[11] *SAP Community* [online]. [cit. 2023-05-15]. Dostupné z: https://blogs.sap.com/2014/11/12/enable-favorites-and-tags-by-default-in-crmui/

[12] BANDARI, Kiran. *Complete ABAP: The Comprehensive Guide to SAP ABAP*. Tretia edícia. Nemecko: SAP PRESS, 2022. ISBN 978-1493223053.

[13] BLOKDYK, Gerardus. *Microsoft Dynamics CRM A Complete Guide*. 5STARCooks, 2021. ISBN 978-1493223053.

[14] *CRM SoftwareBlog* [online]. [cit. 2023-05-15]. Dostupné z: https://www.crmsoftwareblog.com/2014/01/4-noticeable-ways-microsoft-dynamics-crm-can-lighten-your-workload/

[15] BLOKDYK, Gerardus. *Oracle CRM Complete Self-Assessment Guide*. 5STARCooks, 2022. ISBN 978-0655154778.

[16] *DiscoverCRM* [online]. [cit. 2023-05-15]. Dostupné z: https://www.discovercrm.com/oracle-customer-experience-crm-software-profile.html

[17] VESTERLI, Sten. *Oracle ADF Survival Guide: Mastering the Application Development Framework*. USA: Apress, 2017. ISBN 978-1484228197.

[18] BLOKDYK, Gerardus. *Oracle Application Development Framework Complete Self-Assessment Guide*. 5STARCooks, 2022. ISBN 978-0655154778.

[19] *Adobe Experience Cloud A Complete Guide: 2021 Edition*. USA: The Art of Service - Adobe Experience Cloud Publishing, 2020. ISBN 978-1867438342.

[20] *Adobe Experience League* [online]. [cit. 2023-05-15]. Dostupné z: https://experienceleague.adobe.com/docs/experience-platform/landing/platform-ui/ui-guide.html?lang=en

[21] JYOTI, Dipanker a James A. HUTCHERSON. *Salesforce Architect's Handbook: A Comprehensive End-to-End Solutions Guide*. USA: Apress, 2021. ISBN 978-1484266304.

[22] *Cargas* [online]. [cit. 2023-05-15]. Dostupné z: https://cargas.com/software/salesforce-crm/salesforce-crm-features/

[23] The History of Salesforce: 23 Years of Salesforce News Highlights. *Salesforce.com* [online]. 19.03.2020 [cit. 2023-03-21]. Dostupné z: https://www.salesforce.com/news/stories/the-history-of-salesforce/

[24] *CX Today* [online]. [cit. 2023-05-15]. Dostupné z: https://www.cxtoday.com/crm/salesforce-still-leads-the-crm-market-and-by-some-distance/

[25] *Trailhead Salesforce* [online]. [cit. 2023-05-15]. Dostupné z: https://www.cxtoday.com/crm/salesforce-still-leads-the-crm-market-and-by-some-distance/

[26] ZAFAR, Ahsan. *Salesforce Data Architecture and Management: A pragmatic guide for aspiring Salesforce architects and developers to manage, govern, and secure their data effectively*. Spojené kráľovstvo: Packt Publishing, 2021. ISBN 978-1801073240.

[27] Sharing Settings. *Salesforce Help* [online]. [cit. 2023-03-21]. Dostupné z: https://help.salesforce.com/s/articleView?id=sf.managing_the_sharing_model.htm&type=5

[28] User Permissions and Access. *Salesforce Help* [online]. [cit. 2023-03-21]. Dostupné z: https://help.salesforce.com/s/articleView?id=sf.managing_the_sharing_model.htm&type=5

[29] Field Types. *Salesforce Developer* [online]. [cit. 2023-03-19]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.object_reference.meta/object_reference/field_types.htm

[30] Object Relationships Overview. *Salesforce Help* [online]. [cit. 2023-03-21]. Dostupné z: https://help.salesforce.com/s/articleView?id=sf.overview_of_custom_object_relationships.htm&type=5

[31] GOODEY, Paul. *Salesforce CRM - The Definitive Admin Handbook: Build, configure, and customize Salesforce CRM and mobile solutions*. Piata edícia. Spojené Kráľovstvo: Packt Publishing, 2019. ISBN 1789619785.

[32] Page Layouts. *Salesforce* [online]. [cit. 2023-04-08]. Dostupné z: https://help.salesforce.com/s/articleView?id=sf.customize_layout.htm&type=5

[33] Lightning App Builder. *Salesforce* [online]. [cit. 2023-04-08]. Dostupné z: https://help.salesforce.com/s/articleView?id=sf.lightning_app_builder_overview.htm&type=5

[34] Process Builder. *Salesforce* [online]. [cit. 2023-04-08]. Dostupné z: https://help.salesforce.com/s/articleView?id=process_overview.htm&language=en_US&type=5

[35] Flow Builder. Salesforce [online]. [cit. 2023-04-08]. Dostupné z: https://help.salesforce.com/s/articleView?id=sf.flow.htm&type=5

[36] What is Apex?. *Salesforce* [online]. [cit. 2023-04-09]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_what_is_apex.htm

[37] APPLEMAN, Dan. *Advanced Apex Programming in Salesforce*. Štvrtá edícia. USA: Desaware Publishing, 2018. ISBN 1936754126.

[38] BATTISSON, Paul. *Mastering Apex Programming: A developer's guide to learning advanced techniques and best practices for building robust Salesforce applications*. Spojené Kráľovstvo: Packt Publishing, 2020. ISBN 1800200927.

[39] Understanding Apex Core Concepts. *Salesforce* [online]. [cit. 2023-04-09]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_qs_core_concepts.htm

[40] Collections. *Salesforce* [online]. [cit. 2023-04-09]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_collections.htm

[41] Introduction to SOQL and SOSL. *Salesforce* [online]. [cit. 2023-04-09]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_sosl_intro.htm

[42] Introduction to SOQL and SOSL. *Salesforce* [online]. [cit. 2023-04-09]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_dml_section.htm

[43] How Does Apex Work?. *Salesforce* [online]. [cit. 2023-04-09]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_how_does_apex_work.htm

[44] *XRay* [online]. [cit. 2023-05-15]. Dostupné z: https://docs.getxray.app/display/XRAY400/Testing+Salesforce+apps+using+Apex

[45] BOWDEN, Keir. *Visualforce Development Cookbook*. Druhá edícia. Spojené Kráľovstvo: Packt Publishing, 2016. ISBN 978-1786468086.

[46] What is Visualforce?. *Salesforce* [online]. [cit. 2023-04-10]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.pages.meta/pages/pages_intro_what_is_it.htm

[47] SHRIVASTAVA, Mohith. *Learning Salesforce Lightning Application Development: Build and test Lightning Components for Salesforce Lightning Experience using Salesforce DX*. Spojené Kráľovstvo: Packt Publishing, 2018. ISBN 1787124673.

[48] What Is the Lightning Component Framework?. *Salesforce* [online]. [cit. 2023-04-13]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/intro_framework.htm

[49] Introducing Lightning Web Components. *Salesforce* [online]. [cit. 2023-04-13]. Dostupné z: https://developer.salesforce.com/docs/component-library/documentation/en/lwc

[50] Aura Components. *Salesforce* [online]. [cit. 2023-04-13]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/intro_components.htm

ZOZNAM OBRÁZKOV

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

ABAP	Advanced Business Application Programming - Pokročilé programovanie podnikateľských aplikácií
API	Application Programming Interface - Programové rozhranie aplikácie
atď.	a tak ďalej
B2B	Business to Business - podnik-podnik
B2C	Business to Consumer - podnik-zákazník
CRM	Customer Relationship Management - riadenie vzťahov so zákazníkmi
CSS	Cascading Style Sheets - Kaskádové súbory štýlov
ERP	Enterprise Resource Planning - plánovanie podnikových zdrojov
HTML	HyperText Markup Language - Hypertextový značkovací jazyk
ID	Identifier - Identifikátor
IT	Informačné technológie
JSON	JavaScript Object Notation - Zápis objektov v jazyku JavaScript
LWC	Lightning Web Component - Lightning web komponent
napr.	napríklad
REST	Representational State Transfer - Prenos reprezentatívneho stavu
SFA	Sales Force Automation - Automatizácia predajnej sily
SLDS	Salesforce Lightning Design Systém - dizajnový systém Salesforce Lightning
SOAP	Simple Object Access Protocol - Protokol jednoduchého prístupu k objektom
SOQL	Salesforce Object Query Language - Jazyk dopytovania objektov Salesforce
SOSL	Salesforce Object Search Language - Jazyk vyhľadávania objektov Salesforce
t. j.	to jest
tzn.	to znamená

ZOZNAM TABULIEK

ZOZNAM PRÍLOH

Example_no_1	Zdrojový kód aplikácie vytvorenej v príklade č. 1
Example_no_2	Zdrojový kód aplikácie vytvorenej v príklade č. 2
Example_no_3	Zdrojový kód aplikácie vytvorenej v príklade č. 3
CreateRepairContractLWCController	Zdrojový kód Apex triedy vytvorenej v príklade č. 3
CreateRepairContractLWCControllerTest	Zdrojový kód Apex triedy vytvorenej v príklade č. 3
MockHttpResponseGenerator	Zdrojový kód Apex triedy vytvorenej v príklade č. 2
MockHttpResponseGeneratorErrorResponse	Zdrojový kód Apex triedy vytvorenej v príklade č. 2
SendEmail	Zdrojový kód Apex triedy vytvorenej v príklade č. 1
TestObjectTriggerHandler	Zdrojový kód Apex triedy vytvorenej v príklade č. 1
TestObjectTriggerHandlerTest	Zdrojový kód Apex triedy vytvorenej v príklade č. 1
TestScheduler	Zdrojový kód Apex triedy vytvorenej v príklade č. 2
TestSchedulerTest	Zdrojový kód Apex triedy vytvorenej v príklade č. 2
NoAuthentication	Zdrojový kód externého poverenia vytvoreného v príklade č. 2
GetPriceCallout	Zdrojový kód externého služby vytvorenej v príklade č. 2

Example1Flow	Zdrojový kód flowu vytvoreného v príklade č. 1
Example2Flow	Zdrojový kód flowu vytvoreného v príklade č. 2
Example3Flow	Zdrojový kód flowu vytvoreného v príklade č. 3
Company__c-Company Layout	Zdrojový kód rozloženia stránky objektu Company vytvorenej v príklade č. 3
Equipment__c-Equipment Layout	Zdrojový kód rozloženia stránky objektu Equipment vytvorenej v príklade č. 3
Investor__c-Investor Layout	Zdrojový kód rozloženia stránky objektu Investor vytvorenej v príklade č. 2
Repair_contract__c-Repair contract Layout	Zdrojový kód rozloženia stránky objektu Repair contract vytvorenej v príklade č. 3
Student__c-Student Layout	Zdrojový kód rozloženia stránky objektu Student vytvorenej v príklade č. 1
Teacher__c-Teacher Layout	Zdrojový kód rozloženia stránky objektu Teacher vytvorenej v príklade č. 1
Test__c-Test Layout	Zdrojový kód rozloženia stránky objektu Test vytvorenej v príklade č. 1
createRepairContractLWC	Zdrojový kód Lightning Web komponentu vytvoreného v príklade č. 3
GetPriceCredential	Zdrojový kód menovitého poverenia vytvoreného v príklade č. 2
Company__c	Zdrojový kód objektu Company vytvoreného v príklade č. 3

Equipment__c	Zdrojový kód objektu Equipment vytvoreného v príklade č. 3
Investor__c	Zdrojový kód objektu Investor vytvoreného v príklade č. 2
Repair_contract__c	Zdrojový kód objektu Repair contract vytvoreného v príklade č. 3
Student__c	Zdrojový kód objektu Student vytvoreného v príklade č. 1
Teacher__c	Zdrojový kód objektu Teacher vytvoreného v príklade č. 1
Test__c	Zdrojový kód objektu Test vytvoreného v príklade č. 1
CreateCallout	Zdrojový kód súboru povolení vytvoreného v príklade č. 2
Admin	Zdrojový kód používateľského profilu
Test User	Zdrojový kód používateľského profilu
Equipment__c.Create_Repair_contract_via_Flow	Zdrojový kód rýchlej akcie vytvorenej v príklade č. 3
Equipment__c.Create_Repair_contract_via_LWC	Zdrojový kód rýchlej akcie vytvorenej v príklade č. 3
Company__c	Zdrojový kód záložky objektu Company vytvorenej v príklade č. 3
Equipment__c	Zdrojový kód záložky objektu Equipment vytvorenej v príklade č. 3
Investor__c	Zdrojový kód záložky objektu Investor vytvorenej v príklade č. 2
Repair_contract__c	Zdrojový kód záložky objektu Repair contract vytvorenej v príklade č. 3

Student__c	Zdrojový kód záložky objektu Student vytvorenej v príklade č. 1
Teacher__c	Zdrojový kód záložky objektu Teacher vytvorenej v príklade č. 1
Test__c	Zdrojový kód záložky objektu Test vytvorenej v príklade č. 1
TestObjectTrigger	Zdrojový kód Apex spúšťača vytvoreného v príklade č. 1