Interní webová aplikace pro administraci a schvalování finančních prostředků

Bc. Jiří Zenzinger

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jiří Zenzinger**
Osobní číslo:	**A22287**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Interní webová aplikace pro administraci a schvalování finančních prostředků**
Téma práce anglicky:	**Internal Web Application for Financial Resource Administration and Approval**

Zásady pro vypracování

1. Vypracujte literární rešerši na zadané téma.
2. Analyzujte dosavadní verzi interní aplikace a také technologie, které budou použity pro implementaci řešení.
3. Navrhněte technologické úpravy webové aplikace, včetně nového uživatelského rozhraní s podporou mobilního rozhraní.
4. Implementujte aplikaci dle návrhu.
5. Věnujte pozornost zabezpečení webové aplikace.
6. Otestujte funkčnost webové aplikace.
7. Implementaci a testování vhodně popište.

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

Seznam doporučené literatury:

1. ROLDÁN, Carlos Santana. *React 18 Design Patterns and Best Practices - Fourth Edition*. 31.červenec 2023. Packt Publishing, 2023. ISBN 1803233109.
2. STEFANOV, Stoyan. *React: Up & Running: Building Web Applications*. December 21, 2021. O'Reilly Media, 2021. ISBN 1492051462.
3. MICHÁLEK, Martin. *Vzhůru do (responzivního) webdesignu*. Verze 1.1. Praha: vlastním nákladem autora, 2021. ISBN 978-808-8253-006.
4. *NN/g Nielsen Norman Group* [online]. 2023 [cit. 2023-11-07]. Dostupné z: https://www.nngroup.com/.
5. *Webdesigner Depot* [online]. 2023 [cit. 2023-11-07]. Dostupné z: https://www.webdesignerdepot.com/.
6. *React* [online]. 2023 [cit. 2023-11-07]. Dostupné z: https://react.dev/.

Vedoucí diplomové práce: **Ing. Tomáš Vogeltanz, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

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

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

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové 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 práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou 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 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 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 práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové 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 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 10.5.2024 Jiří Zenzinger, v. r.

podpis studenta

ABSTRAKT

Diplomová práce je zaměřena na návrh a vytvoření interní webové aplikace pro schvalování a administraci finančních prostředků ve společnosti NXP Semiconductors. V teoretické části je popsán vývoj webových aplikací a k tomu určené technologie, které byly posléze využity v praktické části této práce. Dále je probrán webový design a jeho principy a diskutována problematika spojená s UX/UI a responzivním designem. Součástí teoretické části je vysvětlení principů zabezpečení webových aplikací včetně jejich testování. Na závěr teoretické části je představena interní webová aplikace *Get Approval Tool*.

V praktické části je interní webová aplikace *Get Approval Tool* detailně analyzována jak z hlediska uživatelského rozhraní, tak i funkčnosti jednotlivých částí aplikace. Na základě analýzy byl vytvořen nový návrh uživatelského rozhraní včetně technologických úprav. Podle návrhu byla webová aplikace implementována a zabezpečena. Implementace, zabezpečení a následné otestování funkčnosti webové aplikace bylo detailně popsáno.

Klíčová slova: vývoj webové aplikace, React.js, Next.js, TypeScript, interní webová aplikace, UX/UI Design

ABSTRACT

The thesis is focused on designing and developing an internal web application for the approval and administration of funds at NXP Semiconductors. The theoretical part describes the development of the web application and the technologies used for this purpose, which have been subsequently used in the practical part of this thesis. Furthermore, web design and its principles and issues related to UX/UI and responsive design are discussed. The theoretical part explains the principles of web application security and its testing. Lastly, the internal web application *Get Approval Tool* is introduced.

In the practical part, the internal web application *Get Approval Tool* is analyzed in detail from the perspective of user interface and functionality of individual parts of the application. Based on the analysis, a new user interface design, including technological modifications, was created. According to the design, the web application was implemented and secured. The implementation, security and subsequent testing of the web application's functionality were described in detail.

Keywords: web application development, React.js, Next.js, TypeScript, internal web application, UX/UI Design

Tímto bych chtěl velmi poděkovat vedoucímu mé diplomové práce, panu Ing. Tomáši Vogeltanzovi, Ph.D. Také děkuji za ochotu vždy zodpovědět všechny mé dotazy, za trpělivost, a především za skvělý přístup.

Dále bych chtěl poděkovat společnosti NXP Semiconductors, která mi poskytla téma pro vypracování diplomové práce. Chtěl bych poděkovat kolegům, rodině, kteří mi v průběhu práce byli oporou a pomohli mi přijít na správná řešení.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Vývoj webových aplikací prošel významnou evolucí, která za posledních několik desetiletí změnila podobu digitálního světa. Od počátku jednoduchých statických webových stránek až po dynamické, interaktivní a poutavé aplikace, které vidíme dnes, byl vývoj poháněn spojením technologického pokroku, měnících se požadavků uživatelů a inovativních vývojových řešení. Tato významná evoluce reflektuje i rostoucí trend digitalizace, kdy se stále více administrativních záležitostí přesouvá do digitální podoby, což významně ovlivňuje způsob, jakým žijeme a pracujeme.

Ruku v ruce s rozvíjející se digitalizací společnosti rostou nároky na funkčnost, použitelnost i vizuál webových aplikací. Zmíněné parametry významně ovlivňují uživatelský zážitek. V minulosti byly webové aplikace využívány primárně na stolních počítačích s podobným rozlišením, ale v poslední době významně převyšuje návštěvnost webových aplikací z mobilních zařízení, jejichž rozlišení se oproti stolním počítačům mění. Při vývoji webové aplikace je v závislosti na tomto trendu kladen velký důraz na její responzivitu.

Tato diplomová práce si klade za cíl analyzovat původní stav interní webové aplikace *Get Approval Tool* firmy NXP Semiconductors a implementovat ji podle nového návrhu odpovídajícího aktuálním trendům. Původní stav webové aplikace byl značně zastaralý a vizuálně neatraktivní, negativně ovlivňoval uživatelskou zkušenost a mobilitu během použití na jiných zařízeních kromě počítače. Analýza původního stavu funkčnosti a uživatelského rozhraní vede k návrhu a implementaci nového rozhraní. Nutností je následné zabezpečení webové aplikace a otestování jak v průběhu vývoje, tak i po implementaci.

Tato práce přispěje k efektivnější správě prostředků a zlepšení mobility zaměstnanců při používání aplikace nejen na počítačích, ale i na mobilních zařízeních. Mezi technologie, využívávané při implementaci, spadá React.js, Next.js, TailwindCSS a s tím související knihovna DaisyUI. Komunikace a validace na straně serveru, vytváření požadavků a operace s daty zprostředkovává GraphQL a Apollo Client.

I. TEORETICKÁ ČÁST

1 WEBOVÝ VÝVOJ A POUŽITÉ TECHNOLOGIE

Vývoj webových stránek spočívá jak ve vytváření webových stránek či aplikací, tak v jejich následné údržbě. Mezi jednotlivé kroky vytváření webových rozhraní patří programování, návrh, publikování webu i správa databází.

Fáze programování se dělí do dvou typů; frontend a backend. V případě frontendového typu vývojář tvoří vizuál a ovládání aplikace. Backendový typ spravuje nevizuální část aplikace, tedy data a databáze. Dříve byla veškerá logika aplikace spravována v backendové části, nyní se logika postupně přesunula spíše do frontendové části.

V současném webovém vývoji představují technologie jako TypeScript, React, Next a TailwindCSS nezbytné nástroje pro vytváření dynamických a uživatelsky atraktivních aplikací. K nástrojům lze zmínit i knihovnu DaisyUI, která sice není pro použití nezbytná, ale zjednodušuje vývoj a přináší konzistentní, atraktivní design, což pozitivně ovlivňuje uživatelský zážitek. Tyto moderní přístupy odrážejí přesun aplikační logiky na frontend, což vyžaduje, aby vývojáři měli nejen technické dovednosti, ale také porozumění UX/UI Designu. Zahrnutí zmíněných nástrojů do procesu vývoje podporuje vytváření aplikací, které jsou nejen efektivní a udržitelné, ale také intuitivní a vizuálně atraktivní pro konečné uživatele. Tento vývoj ukazuje, jak důležité je adaptovat se rychle se vyvíjejícím požadavkům digitálního světa, a zdůrazňuje roli těchto technologií.

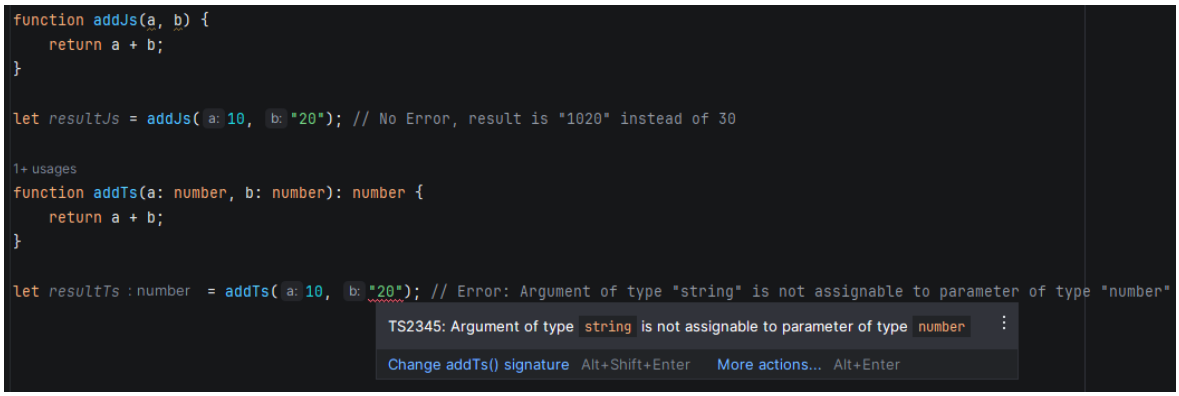
1.1 TypeScript

TypeScript je syntaktická nadstavba JavaScriptu, která nabízí možnost volitelného přidávání statického typování dat a další pokročilé funkce i pro objektově orientované programování.

TypeScript byl vyvinut společností Microsoft a byl poprvé vydán v roce 2012. Ke dnešnímu datu si rychle získal širokou základnu vývojářské komunity zaměřené na webový vývoj. TypeScript přidává syntaxi nad JavaScript; tato syntaxe umožňuje vývojářům vytvářet tzv. typy argumentů. [1]

Volba volitelného statického typování umožňuje vývojářům nastavit typy proměnných, argumentů, parametrů funkcí nebo návratových hodnot. Na Obrázku 1 je znázorněn rozdíl mezi JavaScriptem a TypeScriptem. Kód z JavaScriptu vytváří první funkci s názvem *addJs*, která má dynamicky typované argumenty *a*, *b*. Při předání textového řetězce místo čísla do argumentu *b* není vyvolána chyba a dochází ke špatnému výsledku. Z ukázky TypeScript funkce *addTs* je patrné, že typy argumentů *a*, *b* jsou pevně nastaveny jako čísla. V případě

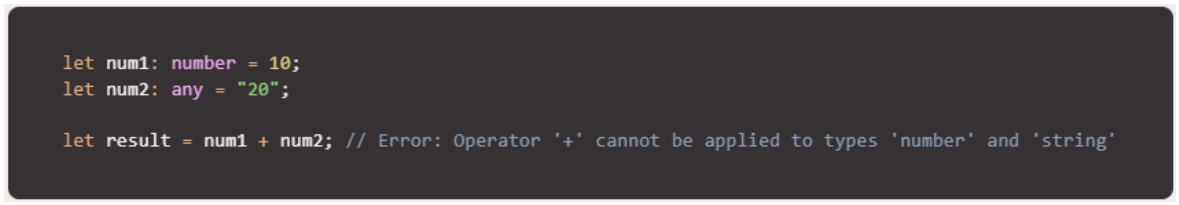
předání textového řetězce místo čísla do argumentu *b* TypeScript při kompilaci kódu vyvolá chybu, poskytující dostatečnou zpětnou vazbu k definování problémů. [1][2][3]



Obrázek 1: Rozdíl mezi JavaScriptem a TypeScriptem [1]

TypeScript přináší řadu vhodných funkcí zaměřených na webový vývoj, které řeší limitace JavaScriptu. Tyto funkce nabízejí nejen lepší organizaci kódu, ale i vylepšení z hlediska efektivity vývojáře. TypeScript má robustní systém typování, který umožňuje definovat typy proměnných a parametrů funkcí při kompilaci. TypeScript nabízí flexibilitu ve volbě použití statického typování. To znamená, že vývojář si může definovat typy proměnných a parametrů funkcí, nebo nechat TypeScript automaticky určit typy na základě přiřazených hodnot. [1][4]

Ukázkový kód (Obrázek 2) má typ proměnné *num2* jako řetězec, ale lze si vybrat, jestli bude daný typ specifikován. Typ může být definován i jako *any*; říká, že hodnota může být libovolného typu. [1]



Obrázek 2: TypeScript, ukázka volitelného typování [1]

TypeScript podporuje moderní funkce JavaScriptu, včetně těch, které byly zavedeny v ECMAScript 6 (ES6) a novějších verzích, umožňující psát čistý a srozumitelný kód s použitím např. šipkových funkcí, destrukturalizace, šablonových řetězců a jiných, a to vše s přidanou kontrolou typů. Ukázkový kód (Obrázek 3) využívá zmíněné arrow funkce a šablonové literálové typy. [1]

```typescript
const greeting = (name: string): string => {
  return `Hello, ${name}!`; // Use of arrow function and template literal
};

console.log(greeting("John")); // Output: Hello, John!
```

Obrázek 3: Ukázka TypeScript kódu využívající ES6+ funkcí [1]

TypeScript nabízí pro organizaci kódu vestavěnou podporu pro moduly a jmenné prostory, umožňující oddělení a organizování kódu do samostatných souborů a řešení závislostí mezi nimi, což usnadňuje správu velkých kódových základů (Obrázek 4). [1][4]

```typescript
// greeting.ts:
export function greet(name: string): string { // Export a function from a module
  return `Hello, ${name}!`;
}

// app.ts:
import { greet } from "./greeting"; // Import from a module

console.log(greet("John")); // Output: Hello, John!
```

Obrázek 4: Organizace kódu do více souborů v TypeScriptu [1]

Jmenný prostor, tzv. namespace, definuje způsob, jak seskupit spolu související kód a zabránit tak zhoršení čitelnosti a organizace kódu (Obrázek 5). Namespace může být použit k organizaci souborů, které obsahují rozhraní, třídy, proměnné nebo funkce, a k vytvoření globálně jedinečných typů. [1]

```typescript
namespace Utilities {
  export function greet(name: string): string {
    return `Hello, ${name}!`;
  }

  export function capitalize(str: string): string {
    return str.toUpperCase();
  }
}

console.log(Utilities.greet("John")); // Output: Hello, John!
console.log(Utilities.capitalize("hello")); // Output: HELLO
```

Obrázek 5: Organizace souborů pomocí namespace v TypeScriptu [1]

Velkou výhodou je navržení TypeScriptu jako rozšíření JavaScriptu, což znamená, že jakýkoli funkční kód JavaScriptu je také funkčním kódem TypeScriptu. To umožňuje snadnou integraci TypeScriptu do existujících JavaScriptových projektů bez nutnosti zdlouhavého přepisování celého kódu. [2][4]

1.2 React.js

React.js vznikl účelem rozdělení složitých bloků komponent na menší jednodušší komponenty a zajištění jejich opětovné použitelnosti na více místech. Pro příklad, pokud je potřeba vytvořit tlačítko na více stránkách, dříve byli vývojáři nuceni vypisovat podobný kód přes několik stránek. Tuto problematiku React.js vyřešil. Jeden z dalších důvodů vzniku React.js byla nízká rychlost pracování Document Object Modelu (zkr. DOM). Přestože DOM představuje hlavní prvek pro tvorbu dynamických webových aplikací, jeho přímá manipulace často vede k výkonnostním problémům při aktualizaci uživatelského rozhraní. [5][6][7]

React.js je často mylně považován za framework, ale jde o jednu z nejoblíbenějších knihoven JavaScriptu, určenou pro vytváření dynamických uživatelských rozhraní. Rozdíl mezi frameworkem a knihovnou je v jejich volnosti psaní kódu. Framework je pevně řízený způsob struktury pro psaní kódu, zatímco knihovna má volný způsob zápisu bez jakéhokoliv strukturálního omezení. Knihovnu React.js vytvořila společnost Meta, dříve známá pod názvem Facebook, v roce 2011, ale pro širokou veřejnosti ji poprvé zveřejnila o dva roky později, v květnu 2013. Aktuálně jde o open-source projekt. [6][7][8]

Klíčovými vlastnostmi React.js jsou komponentový a deklarativní přístup k programování, který značně usnadňuje vývoj složitějších aplikací. React.js umožňuje vývojářům vytvářet opakovaně použitelné UI komponenty, které mohou být snadno integrovány a opětovně využity v různých částech aplikace, což vede k lepší efektivitě a organizaci kódu. Díky těmto vlastnostem React.js nejenže umožňuje rychlý vývojový proces, ale také díky efektivní aktualizaci uživatelského rozhraní zajišťuje vysokou výkonnost aplikace. [6][7]

React.js je v dnešní době považován za jednu z klíčových technologií ve vývoji webových aplikací.

1.2.1 Virtual Document Object Model

Document Object Model, zkr. DOM, je nejdůležitější prvek webu, dělí se na moduly a spouští kód. DOM poskytuje programovatelné rozhraní pro JavaScript, které umožňuje přistupovat k jednotlivým prvkům na stránce a měnit jejich vlastnosti nebo obsah. JavaScript

tak může dynamicky aktualizovat obsah stránky na základě interakce uživatele nebo událostí ve webové aplikaci. JavaScript obvykle aktualizuje celý DOM najednou, což webovou aplikaci výrazně zpomaluje. Virtuální DOM (zkr. VDOM) pak přichází jako řešení této problematiky, která je spojená s častými aktualizacemi DOM. VDOM je virtuální reprezentace UI uchovávaná v paměti prohlížeče a odpovídá přesné kopii skutečného DOM vytvořené pomocí knihovny React.js. Kdykoli dojde ke změně ve webové aplikaci, aktualizuje se nejprve celý virtuální DOM a zjistí se rozdíl mezi skutečným a virtuálním DOM. [6][9]

JavaScript je sám o sobě rychlý nástroj, ale operace aktualizace DOM je náročná; při každé změně dat musí prohlížeč vypočítat pozice všech prvků ve webové aplikaci. Rychlost byla ještě více zhoršována, protože většina frameworků v době, kdy vznikal React.js, aktualizovala DOM mnohem častěji, než bylo potřeba. Pro lepší představu problému je zde jednoduchý příklad: existuje seznam, který obsahuje 10 položek s možností selekce a jedna z položek seznamu má být změněna. Většina frameworků by při této akci aktualizovala všech 10 položek, což je 9krát více operací, než je nutné, jelikož se ze seznamu změnila pouze jedna položka. [6][9]

Virtuální DOM umožňuje efektivnější aktualizaci skutečného DOM, protože minimalizuje počet operací potřebných k vykreslení změn. Místo aktualizace každého jednotlivého prvku se aktualizují pouze ty části, které se doopravdy změnily. To vede k výraznému zrychlení procesu aktualizace a optimalizaci výkonu webových aplikací. [6][9][10]

1.2.2 JavaScript XML

JavaScript XML, zkr. JSX, představuje syntaktické rozšíření jazyka JavaScript. Rozšíření syntaxe bylo vydáno souběžně s React.js za účelem zjednodušení psaní a zlepšení čitelnosti kódu při vytváření uživatelských rozhraní. Rozšíření není v React.js podmínkou, ale využití se doporučuje, kvůli jeho schopnosti kombinovat HTML atributy a JavaScript výrazy přímo v kódu komponent. Vytvořené elementy jsou při zpracovávání převáděny do JavaScript objektů pomocí nástroje jako je Babel, který zvyšuje flexibilitu vývoje překládáním JSX formátu do formátu podporovaného prohlížeči. Základní použití JSX syntaxe na Obrázku č.6 ukazuje názornou deklaraci dvou JavaScript proměnných v kódu. [6][7][11]

```
const name = "GeekforGeeks";
const ele = <h1>Welcome to {name}</h1>;
```

Obrázek 6: JSX deklarace elementů [7]

1.2.3 React.js komponenty

Komponenta představuje jednu ze základních stavebních prvků v React.js. Aplikace jsou rozděleny na menší, nezávislé bloky kódu, nazývané komponenty. Komponenty by měly být navrhovány tak, aby byly znovupoužitelné napříč aplikací. Tento přístup značně usnadňuje vytváření složitějších uživatelských rozhraní. [6][7]

V praxi to znamená, že místo psaní většího množství kódu do jednoho souboru je vytvořeno více menších částí, které mohou představovat např. formuláře, tlačítka nebo navigační prvky. Části lze pak kombinovat a skládat dohromady do větších komponent až po úroveň celé aplikace. Je tedy velice efektivní, pokud lze definovat obecnou část aplikace jako komponentu, která se může použít nespočetněkrát kdekoli v kódu. S tím se i pojí dva přístupy vytváření těchto komponent. [7][12]

První přístup, více používaný, je vytváření funkčních komponent (Obrázek 7). Ty si lze jednoduše představit jako JavaScriptové funkce. Funkce pak mohou přebírat parametry. Parametry umožňují rodičovským komponentám předávat data nebo funkce svým potomkům, tzv. props. Ve funkčních komponentách je návratovou hodnotou kód JSX, který se vykresluje do stromu DOM. [6][7][13]

```
function Greeting() {
  return (
    <h1 className="heading">Welcome to React </h1>
  );
}
```

Obrázek 7: Funkční komponenta v React.js [6]

Přístup vytváření komponent přes třídy (Obrázek 8) je o něco složitější než vytváření funkčních komponent. Na rozdíl od funkčních komponent spolu komponenty vytvořené pomocí tříd mohou vzájemně komunikovat. Pokud je potřeba předávat data z jedné komponenty do druhé, je potřeba třídy deklarovat s *Extend React.Component*. [6][13]

```
class HelloWorld extends Component {
  constructor(props) {
    super(props);
    // Initialize state
    this.state = {
      greeting: 'Hello, World!'
    };
  }

  render() {
    return (
      <div>
        <h1>{this.state.greeting}</h1>
      </div>
    );
  }
}
```

Obrázek 8: Komponenta třídy v React.js [6]

Funkční komponenta je nejvhodnější pro případy, kdy nemusí daná komponenta komunikovat s jinými komponentami nebo spravovat složité stavy. Funkční komponenty jsou ideální pro zobrazení statických prvků uživatelského rozhraní nebo pro seskupení více jednoduchých komponent pod jednu rodičovskou komponentu. Komponenty založené na třídách sice mohou dosahovat stejného výsledku, ale ve srovnání s funkčními komponentami jsou obecně méně efektivní. Proto se nedoporučuje používat komponenty tříd pro obecné použití. [6][7][13]

1.2.4 React Hooks

React Hooks, dále jen Hooks, jsou znovupoužitelné funkce, které byly poprvé představeny ve verzi React.js 16.8. Hooks poskytují přístup k tzv. stavům a dalším vestavěným funkcím. Hooks slouží jako funkce JavaScriptu nabízející vývojářům jednodušší a přehlednější způsob zpracování stavů a vedlejších událostí v aplikacích. Všechny funkce a stavy lze využívat bez nutnosti vytvářet komponenty založené na třídách. Hooks mají i další výhody, jako je lepší organizace kódu, opakovaná použitelnost a významně zjednodušená správa logiky komponent. [14][15]

Existuje řada vestavěných Hooks, z nichž každá metoda slouží ke specifickým účelům. Mezi nejznámější metody patří *useState*, *useContext*, *useEffect* a mnoho dalších. [10][15]

State Hooks (Obrázek 9), v překladu stavové Hooks, umožňují komponentám zapamatovat si data jakéhokoliv typu, například e-mailovou adresu, kterou uživatel napsal do textového pole v prohlížeči nebo celkový počet produktů zobrazených na e-shopu. O logiku se stará metoda *useState*, která zajišťuje vytváření, aktualizování a manipulaci se stavy uvnitř

funkčních komponent. Jednoduše lze stavy popsat jako proměnné uchovávající zmíněná data. Data se mohou v průběhu životního cyklu komponent měnit a kdykoli k tomu dojde, React.js aktualizuje své uživatelské rozhraní a znovu vykreslí komponenty s aktuálními daty. [14][15][16]

```
const [index, setIndex] = useState(0);
```

Obrázek 9: React *useState* Hook [16]

React.js je postaven na základě principu jednosměrného datového toku. Když je potřeba předat data z rodičovské komponenty do potomka, musí se data ručně posílat jako props. Tím se data předávají do každé následující úrovně, i tam, kde nejsou potřeba, což vede k tzv. props drillingu. Tento problém je řešen pomocí kontextu. Kontextové Hooks (Obrázek 10) zpřístupňují specifická data všem komponentám napříč aplikací bez ohledu na to, do jaké komponenty jsou vnořeny. Metoda *useContext* načte vytvořený kontext a přihlásí se k odběru jeho aktuálních dat. Tento přístup umožňuje přenášet informace, např. o preferovaném jazyce uživatele nebo zvoleném tématu, světlém nebo tmavém rozhraní. Výhoda použití kontextu spočívá v jednoduchosti předávání dat přes všechny komponenty v rámci aplikace. [14][15][16]

```
function Button() {
  const theme = useContext(ThemeContext);
```

Obrázek 10: React *useContext* Hook [16]

Poslední nejznámější metodou je *useEffect* (Obrázek 11), který umožňuje spouštět vedlejší události ve funkčních komponentách. Metodu si lze představit jako spojení třech částí životního cyklu komponenty: *componentDidMount*, *componentDidUpdate* a *componentWillunmount*. Hook slouží pro úpravu dat na základě změny stavu v rodičovské komponentě, načítání dat z databáze, aktualizaci DOM i aktualizaci časovače. I když je metoda velice známá, používá se spíš jako poslední záchranné východisko. Nedoporučuje se používat *useEffect*, pokud nejsou prováděny interakce s externím rozhraním, jako třeba Application Programming Interface (zkr. API), jelikož *useEffect* se hodí například na provádění asynchronních operací. Bez takové interakce by mohlo použití *useEffect* vést k neefektivnímu vyvolávání funkcí a zbytečnému zatížení aplikace. [14][15][16]

```
function ChatRoom({ roomId }) {
  useEffect(() => {
    const connection = createConnection(roomId);
    connection.connect();
    return () => connection.disconnect();
  }, [roomId]);
```

Obrázek 11: React *useEffect* Hook [16]

1.3 Next.js

Next.js, vyvinut společnosti Vercel, je open-source framework postaven na knihovně React.js. Umožňuje vytvářet rychlé React aplikace optimalizované pro vyhledávače bez nutnosti konfigurace. React.js aplikace se tradičně vykreslují na straně klienta. [17][18]

Prohlížeč začíná se základní stránkou HTML, která neobsahuje žádný vykreslovaný obsah. Z něj prohlížeč získá soubory typu JavaScript, které se skládají ze souborů React.js, a zobrazí obsah na webové stránce. Next.js umožňuje vytvářet aplikace připravené pro produkční nasazení díky schopnosti vykreslovat obsah na straně serveru (server-side rendering), předběžnému načítání adres (route pre-fetching) a chytrému sdružování (smart bundling). Tyto možnosti nejsou v React.js běžně k dispozici, pro jejich získání je potřeba nainstalovat externí knihovny, například React Router, sloužící k routování SPA aplikací. [10][18][19]

Next.js nabízí také okamžité načítání kódu, automatické rozdělení kódu, kompatibilitu s ekosystémem a automatické routování. Next.js má dva typy routování: App Router a Pages Router. App Router je modernější router, který umožňuje používat nejnovější funkce Reactu; serverové komponenty a streamování. Pages Router je prvotní router Next.js, který umožňoval vytvářet serverově renderované aplikace a je i nadále podporován pro starší verze Next.js aplikací. [17][19][20]

Next.js navíc podporuje statické generování (SSG) i vykreslování na straně serveru (SSR), umožňující efektivní předvykreslování. SSR vytváří obsah stránek na serveru, zatímco vykreslování na straně klienta (CSR) se používá pro jednostránkové React.js aplikace. Vykreslování na straně klienta může být problematické pro optimalizaci pro vyhledávače (zkr. SEO), protože vyhledávače nevidí skutečný obsah stránky. Načítání stránek na straně serveru dokáže tomuto problému předejít, čímž zabrání problikávání stránky během načítání dat a poskytne obsah vhodný pro SEO. [10][18]

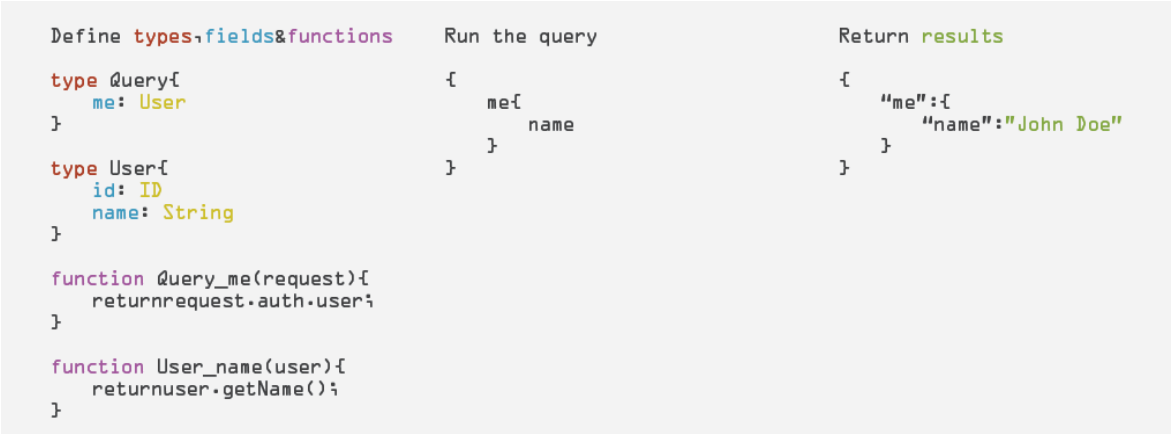
1.4 GraphQL

GraphQL je open-source dotazovací jazyk definující, jakou cestou klient prostřednictvím rozhraní API získává informace. V širším slova smyslu přestavuje jazyk GraphQL syntaxi, kterou mohou programátoři použít k vyžádání konkrétních dat a jejich vracení z různých zdrojů. Jakmile klient definuje strukturu požadovaných dat, server data vrátí s použitím identické struktury. Technologie vyniká schopností redukovat nadměrné a nedostatečné načítání dat, což je běžný problém tradičních rozhraní REST API. [21][22]

Díky otevřeným možnostem jazyka GraphQL pro procesy čtení, mutace dat a monitorování, mohou vývojáři aktualizovat data v reálném čase. Servery GraphQL byly vyvinuty pro práci s populárními kódovacími jazyky, jako jsou JavaScript, TypeScript, Python, Ruby, C# a jiné. Cílem GraphQL je poskytnout vývojářům komplexní přehled o datech uložených v rámci rozhraní API. To zahrnuje schopnost přijímat výhradně data přímo odpovídající konkrétním dotazům a vytvořit tak architekturu, která usnadní škálování a přizpůsobování rozhraní API v dlouhodobém horizontu. [21][22][27]

Pro využití jazyka GraphQL definují vývojáři schéma, které popisuje vlastnosti a strukturu rozhraní API. Schéma funguje jako plán, podle kterého jsou sestavovány přesně definované a efektivní datové příkazy. Toto schéma zahrnuje typy, které popisují datové struktury a jejich pole, a resolvery, které určují, jak a kde je možné k datům získat přístup. Dotazy definované vývojáři dávají serveru instrukce, jaká data má načíst, a podporují složité struktury, např. vnořená pole. Tento přístup zvyšuje nejen efektivitu získávání dat, ale také zlepšuje celkový komfort vývojářů tím, že poskytuje jasnou a organizovanou techniku pro vytváření datových dotazů. [21][26][27]

Po spuštění služby GraphQL (obvykle na adrese URL webové služby) může GraphQL přijímat dotazy, které jsou ověřeny a následně spuštěny. Služba nejprve zkontroluje dotaz, aby se ujistila, že se týká pouze definovaných typů a polí. Poté se spustí poskytnuté funkce a zobrazí se tak výsledek, jak je názorně ukázáno v Obrázku 12:



Obrázek 12: Ukázkový GraphQL dotaz s výsledkem [27]

1.5 Apollo Client

Apollo Client je komplexní knihovna určená pro správu stavů v React.js, ulehčující řízení lokálních i vzdálených dat pomocí dotazovacího jazyku GraphQL. Kromě získávání dat nabízí Apollo Client i jejich logické ukládání do mezipaměti a deklarativní načítání, čímž výrazně zvyšuje výkon aplikace a snižuje zátěž na server. Jelikož bylo Apollo vyvinuto právě pro React.js, tak používá k vytvoření dotazu na komponentu *useQuery* API Hook. To umožňuje okamžité vykreslení výsledků vytvořeného dotazu do zvolené komponenty. Deklarativní způsob vytváření dotazů napomáhá psaní menšího množství kódu a může pomoci zefektivnit vývojový proces. [28][29]

Apollo Client pokrývá celkový cyklus získávání dat včetně sledování jednotlivých stavů. Vývojář se nemusí starat o manuální ukládání dat do mezipaměti, sledování stavů ani transformaci dat. Apollo Client sleduje jak načítání a chybové stavy, tak i konečný stav, kdy byla data z dotazu úspěšně načtena. Díky zapouzdření stavů pomocí *useQuery* Hook je integrace dotazů do aplikace velice jednoduchá. Při spouštění dotazu Apollo Client informuje vývojáře o odeslání dotazu na server a o spuštění načítání dat; vývojář může na uživatelském rozhraní zobrazit komponentu indikující načítání dat. V případě chybného dotazu lze jednoduše zobrazit chybnou hlášku nebo komponentu. Na konci cyklu získávání dat, pokud je dotaz úspěšný, dojde k zobrazení nových dat do uživatelského rozhraní. [28][29]

V případech, kdy by vývojář potřeboval vlastní funkce, si může vytvořit vlastního klienta a jeho vlastnosti si upravit, avšak toto rozšíření by probíhalo nad Apollo Client a chování by muselo být přesně definováno. Apollo Client podporuje zmíněné ukládání a správu mezipaměti (Obrázek 13) jak automaticky, tak manuálně; vývojář může sám procházet, upravovat

a mazat data uložená v mezipaměti. Tato funkce může zrychlit aplikaci, jelikož v určitých situacích nebude potřeba opět vyvolávat všechna data z databáze, ale dotaz se bude týkat jen těch, která se změnila. [28]

Velkou výhodou Apollo Client je jeho řešení normalizace dat, která je klíčem k udržení konzistence dat napříč aplikací. Kdykoliv Apollo Client obdrží data z odpovědi dotazu, automaticky se pokusí identifikovat změny a objekty, které byly aktualizovány, uložit. Identifikace probíhá dle položky *typename*; pokud je položka nalezena a objekt se v něčem liší, tak jej uloží. [28]

```
1 import { ApolloClient, InMemoryCache } from '@apollo/client';
2
3 const client = new ApolloClient({
4   cache: new InMemoryCache(),
5 });
```

Copy

Obrázek 13: Ukázkové vytvoření mezipaměti v Apollo Client [28]

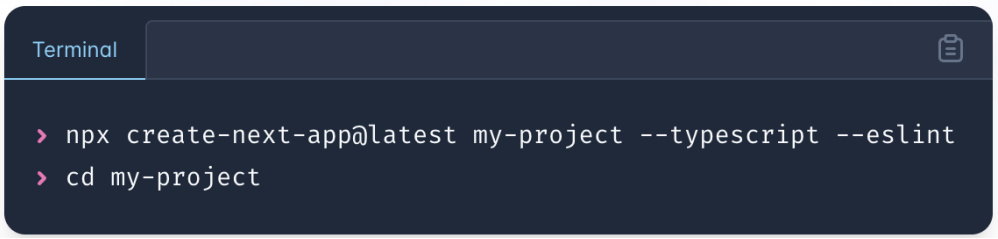
1.6 TailwindCSS

TailwindCSS umožňuje vývojářům efektivněji a rychleji definovat styly aplikace bez nutnosti pracně spravovat vlastní statické soubory CSS. Funguje na principu procházení HTML souborů, JavaScriptových komponent a jiných šablon, kde hledá názvy tříd. Poté, co jednotlivé třídy v souborech vyhledá, tak vygeneruje statické soubory CSS odpovídající použitým stylům. Framework poskytuje sadu předdefinovaných tříd, které přímo odpovídají jednotlivým CSS vlastnostem. Nabízí možnost velmi flexibilně upravovat vzhled aplikace a komponent přímo na úrovni HTML nebo JSX kódu, a to bez běhu v reálném čase. [31]

1.6.1 Konfigurace pro Next.js

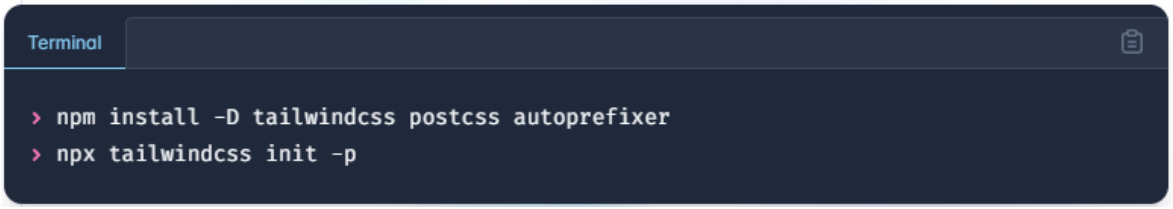
Ukázková konfigurace je představena přímo pro technologii Next.js, ta bude použita v praktické části. Veškeré příkazy budou spouštěny přímo v terminálu. V rámci instalace a konfigurace je v prvé řadě vhodné mít nainstalovaný Node.js a i správce knihoven npm, které jsou součástí.

Prvním krokem konfigurace je vytvoření projektu přes nejčastěji používaný nástroj Create Next App. Pod část *my-project* je projekt pojmenován; ostatní části říkají, že bude projekt nastavený pro TypeScript a ESLint (Obrázek 14). Po vytvoření projektu se do něj přesuneme, abychom mohli TailwindCSS dále nastavit. [30]



Obrázek 14: Vytvoření projektu Create Next App [30]

Přes npm správce knihoven je posléze nainstalován modul TailwindCSS spolu s PostCSS a AutoPrefixer (Obrázek 15). Tyto dva moduly slouží k optimalizaci a zefektivnění procesu vytváření stylů. PostCSS slouží pro transformaci CSS pomocí Javascriptu a pomáhá převádět Tailwindové direktivy a konfigurace na čisté CSS. AutoPrefixer je specifický plugin pro PostCSS, který automaticky přidává specifické CSS vlastnosti, prefixy, k CSS pravidlům. Ty jsou nezbytně nutné pro zajištění kompatibility mezi různými prohlížeči. [30][31]



Obrázek 15: Instalace TailwindCSS [30]

Inicializace TailwindCSS pak probíhá v rámci druhého řádku, který vygeneruje dva soubory. Jeden je pro konfiguraci samotného TailwindCSS souboru (Obrázek 16) a druhý pro konfiguraci PostCSS modulu. [30][31]

Další krok je zaměřen na nastavení cest k daným souborům a šablonám. Jaká je zvolena struktura projektu je jen na vývojáři. Všechny parametry jsou nastavovány v rámci konfiguračního souboru *tailwind.config.js*. Cesty jsou nastaveny v objektu *content*, který byl dříve známý pod názvem *purge*. Konfigurací cest je zajištěno, že TailwindCSS bude správně pracovat se soubory. [30][31]

tailwind.config.js

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./app/**/*.{js,ts,jsx,tsx,mdx}",
    "./pages/**/*.{js,ts,jsx,tsx,mdx}",
    "./components/**/*.{js,ts,jsx,tsx,mdx}",

    // Or if using `src` directory:
    "./src/**/*.{js,ts,jsx,tsx,mdx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Obrázek 16: Konfigurace Tailwind souboru [30]

Aby TailwindCSS mohl správně pracovat, musí být naimportován do konfiguračního souboru PostCSS a spolu s AutoPrefixer (Obrázek 17). Toho lze dosáhnout tak, že do vytvořeného *postccs.config.js* souboru v kořenovém adresáři projektu jsou do objektu plugins vloženy zmíněné prefixy. [31]

postcss.config.js

```
module.exports = {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  }
}
```

Obrázek 17: Přidání TailwindCSS do PostCSS konfigurace [31]

Jeden z posledních kroků potřebných pro nastavení TailwindCSS je vytvoření souboru *globals.css* a vložení direktivy pro každou vrstvu (Obrázek 18). První *base* resetuje a nastavuje základní styly tak, aby byl vzhled webových stránek konzistentní napříč různými webovými prohlížeči. *Components* vkládá styly, které jsou předdefinované pro námi vybrané komponenty v konfiguračním souboru. Poslední direktiva, *utilities*, přidává třídy, které umožní rychlé stylování prvků pomocí HTML tříd. [30][31]

globals.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Obrázek 18: Přidání direktiv do globálního CSS souboru [30]

Po úspěšném dokončení všech kroků popsaných výše lze začít plně využívat efektivní vytváření stylování pomocí TailwindCSS. Ukázka kódu (Obrázek 19) znázorňuje využití stylovacích tříd do HTML kódu. Tímto způsobem jsou pak aplikovány styly na vytvořený HTML element. [30][31]

index.tsx

```
export default function Home() {
  return (
    <h1 className="text-3xl font-bold underline">
      Hello world!
    </h1>
  )
}
```

Obrázek 19: Stylovací třídy TailwindCSS [30]

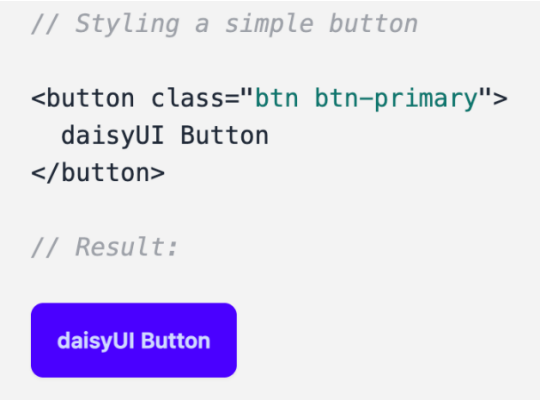
1.7 DaisyUI

DaisyUI je open source UI knihovna vytvořená na základech TailwindCSS. Knihovna poskytuje širokou kolekci předem navržených UI komponent, které lze jednoduše přizpůsobit a integrovat do projektu. Komponenty navíc zajišťují konzistenci uživatelského rozhraní napříč jednotlivými částmi aplikace. Pod komponentami si lze představit tlačítka různých velikostí, menu, zápatí, responzivní karty, vyskakovací okna, a to vše v různých barevných variantách. Součástí jsou nejen popsané vizuální prvky, ale i komponenty, které se starají o rozložení webových aplikací. Mezi ně patří kontejnery, mřížky (grid), vizuální děliče a jiné. Přes TailwindCSS lze upravovat velikost prvků, prostor okolo prvků a jejich responzivní chování. [32][34]

Kromě rychlejšího vývoje a atraktivního vzhledu komponent knihovna zmenšuje velikost souborů. Pro příklad, kdyby vývojář chtěl pomocí TailwindCSS stylovat obyčejné tlačítko, by musel napsat velké množství tříd, aby dosáhl totožného vzhledu tlačítka.

Kombinace DaisyUI a TailwindCSS umožní vývojářům použít jen jednu třídu, a to *btn*. Tímto způsobem je vygenerováno tlačítko, které přesně odpovídá stylování vytvořenému knihovnou (Obrázek 20). S rostoucí popularitou použití více motivů v jedné aplikaci tuto možnost nabízí i knihovna DaisyUI. Má předdefinovanou sadu motivů, podporující jak různé barevné varianty, tak i rozdělení na světlé a tmavé motivy. Samozřejmě vývojářům umožní vytvořit i vlastní nastavení motivů přes generátor. [32][33]

Je potřeba říct, že DaisyUI bohužel není vhodná pro všechny případy. V momentě, kdy by vývojář potřeboval vytvořit speciální designovou stránku nebo aplikaci s vlastním brandingem, tak se mu nutně předdefinované komponenty nemusí hodit. Naopak, bylo by potřeba předělat kompletně celé stylování všech využitých prvků. [32][34]



Obrázek 20: Ukázka vytvoření tlačítka pomocí DaisyUI [33]

2 WEBOVÝ DESIGN

Webový design je specializované odvětví, které se zabývá vytvářením uživatelských rozhraní a vzhledu webových aplikací. Tato oblast je nezbytnou součástí procesu vývoje webových aplikací a jeho cílem je vytvořit webovou stránku, která je vizuálně atraktivní, přehledná a jednoduchá na ovládání. Bez těchto jednotlivých kroků by nebyly webové stránky pro uživatele intuitivní. To jen podtrhuje důležitost webového designu, bez něj jsou stránky prakticky odsouzeny k zániku. [23][35][36][37]

Webový design však neznamená pouze grafický design a vytváření jednotlivých komponent stránky. Zahrnuje mnoho dalších aspektů, jako je SEO, marketing, copywriting, ale i branding a design společnosti, pro kterou je stránka vytvářena. Pro efektivní přizpůsobení webu potřebám a očekáváním koncového zákazníka je nezbytné jim dobře porozumět. [5][35][36]

Webový design zaštiťuje tři hlavní oblasti: vizuální komunikaci, obsahovou strategii a interakční design s cílem maximalizovat uživatelskou odezvu a zvýšit celkovou spokojenost uživatelů. Vizuální komunikace zahrnuje vše, co je součástí vizuální prezentace stránky; od barev a typografie po animace a celkové rozložení stránky. Obsahová strategie se naopak zaměřuje na vytváření a správu obsahu, který je pro uživatele relevantní a zajímavý. Interakční design se zabývá způsobem, jakým uživatelé pracují se stránkou a jak stránka reaguje na jejich akce. [35][37]

Kvalitní webový design je jedním z hlavních důvodů, proč se uživatelé na webovou stránku opakovaně vrací. Pokud je stránka atraktivní, intuitivní a uživatelsky přívětivá, uživatelé se na ni budou chtít vracet. Naopak, pokud stránka není dobře navržena, uživatelé ji mohou opustit po několika sekundách používání a již se na ni nevrátit. Proto je webový design tak kritický pro úspěch jakékoliv webové stránky nebo aplikace. Přesně za tyto cíle zodpovídají dva obory, a to UX a UI Design. [35][36]

2.1 UX Design

Uživatelská zkušenost, angl. user experience, známá jako UX Design, je obor, opírající se o principy kognitivní vědy. UX Design se zaměřuje na celkový zážitek uživatele při interakci s produktem. Klíčovým aspektem při návrhu a vývoji jakékoliv aplikace, ať už webové, mobilní či desktopové je, aby nebyla zaměřena pouze na vzhled stránky, ale především na interakci uživatele. Je nezbytné zajistit, aby uživatelé mohli v aplikaci snadno nalézt to, co hledají, a aby byla její použitelnost jednoduchá a intuitivní. Cílem je vytvořit řešení, které

nejlépe vyhovuje potřebám koncovým uživatelům a zajištění, aby se uživatelé necítili ztraceni. [24][35][39]

V tomto kontextu je důležité zdůraznit, že hlavním cílem není vizuální stránka aplikace, ale celkový uživatelský zážitek a pocity z interakce s aplikací. Úspěch aplikace je definován její schopností plnit svůj účel a funkci co nejefektivněji. To vyžaduje od UX Designéra schopnost přesně identifikovat cílovou skupinu uživatelů, pro které je aplikace určena. Na základě této cílové skupiny se poté rozhoduje o přístupu k designu. Co může být pro jednu skupinu uživatelů naprosto srozumitelné, například specifické termíny, texty, obrázky nebo ikony, může být pro jinou skupinu matoucí a nejasné. [24][38][39]

2.2 UI Design

Design uživatelského rozhraní, ang. user interface, UI Design, se zaměřuje na vizuální prvky a interakce, které vytvářejí uživatelské prostředí digitálních produktů. Pod digitálními produkty si lze představit aplikace mobilní, webové a samotné webové stránky. Na rozdíl od UX Designu, který se soustředí výhradně na celkovou uživatelskou zkušenost, se UI Design zaměřuje na aspekty jako jsou vzhled, styl a interakci uživatele s produktem. [38][39][40]

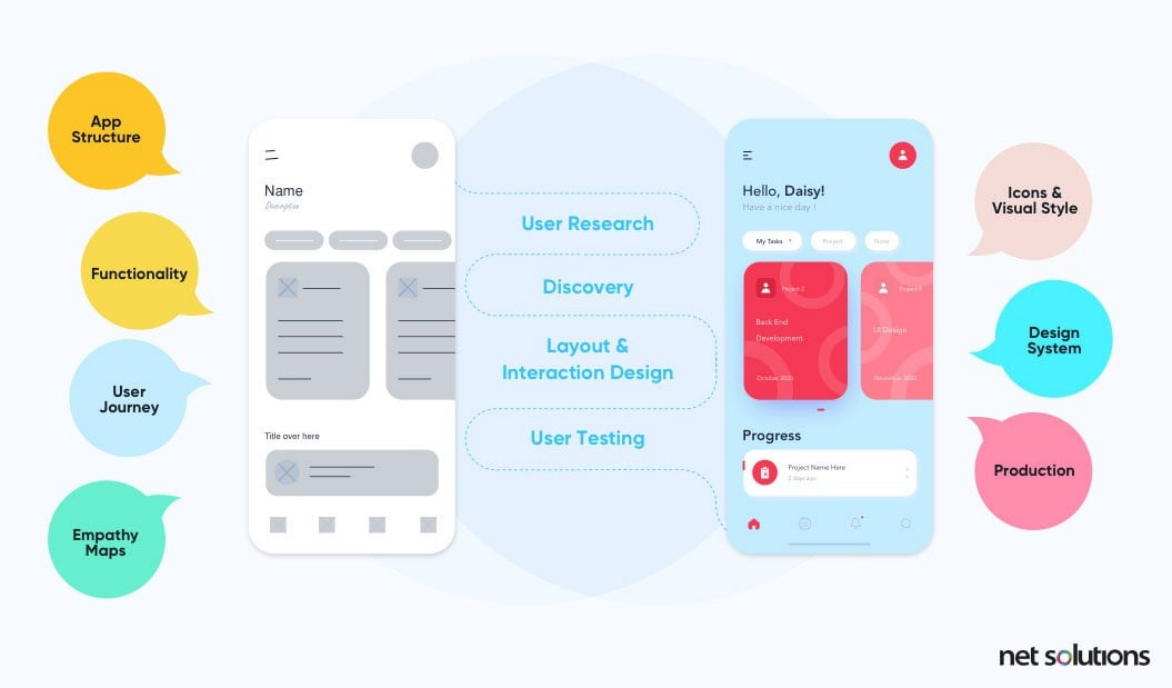
Zatímco UX Design pokrývá širší spektrum faktorů ovlivňujících uživatelskou interakci s produktem (včetně jeho struktury, logiky a funkčnosti), UI Design je striktně digitální a zahrnuje konkrétní bod interakce mezi uživatelem a digitálním produktem. Tento bod interakce může být reprezentován dotykovým rozhraním smartphonu, designem webové stránky nebo rozložením aplikace, kde vizuální a interaktivní prvky hrají klíčovou roli v tom, jak je produkt vnímán a používán. [24][38][39][40]

UI Design zahrnuje výběr a uspořádání ikon, tlačítek, typografie, barevných schémat a dalších vizuálních prvků, stejně jako řešení responzivního designu, tzn. aby se produkt správně zobrazil a byl použitelný na různých velikostech zařízeních a obrazovkách. Důraz je kladen na intuitivnost, vizuálně přitažlivé a snadno použitelné rozhraní. UI Designéři musí pečlivě zvážit každý aspekt vizuálního designu a interakce, aby uživatelé mohli snadno pracovat s produktem. [25][38][39][40]

Dalším klíčovým aspektem UI Designu je zohlednění základních principů, jako jsou jasnost, obeznámenost, efektivita a konzistence. Tyto principy zajišťují, že uživatelé nebudou zbytečně ztrácet čas přemýšlením o tom, jak jednotlivé prvky fungují, ale budou moci využívat

svých předchozích interakčních zkušeností a budou mít z používání produktu pozitivní zážitek. [25][39][40]

V porovnání s UX Designem, který může zahrnovat i netechnické aspekty a postupy pro zlepšení uživatelské spokojenosti, se UI Design zaměřuje striktně na digitální rozhraní a jeho prvky (Obrázek 21). Proces návrhu UI Designu zahrnuje vytvoření náčrtů, prototypů a jejich testování, aby finální produkt splňoval jak estetické, tak funkční požadavky. Dobře vytvořený UI Design pak vede k rozhraní, které je nejen atraktivní, ale je také uživatelsky přívětivé, což přispívá k pozitivní uživatelské zkušenosti. [38][40]



Obrázek 21: UX vs. UI [40]

2.3 Responzivní design

Responzivní webový design (zkr. RWD) je metoda navrhování webových stránek tak, aby se dokázaly přizpůsobit zařízení uživatele. Cílem je zajistit optimální použitelnost a vzhled bez ohledu na to, z jakého typu zařízení se na webové stránky přistupuje. Responzivní webový design toho dosahuje pomocí flexibilních vrstev a rozvržení, responzivních obrázků a mediálních dotazů CSS, které se přizpůsobují různým velikostem obrazovek, orientacím, rozvržením a platformám. [41][43]

V minulosti byly pro zobrazování webových stránek používány pouze stolní počítače a mobilní zařízení, ale nyní existují různé typy zařízení, včetně notebooků, tabletů a hodinek,

kterým je potřeba se přizpůsobovat. Dříve, než se RWD stal běžným řešením k zajištění funkčnosti webové stránky na různých typech zařízení, používali vývojáři označení mobilní webový design, vývoj mobilních webových stránek nebo design vhodný pro mobilní zařízení. Jejich cílem, totožně jako RWD, je zajistit, aby webové stránky řádně fungovaly na zařízeních s různými fyzickými vlastnostmi, jako je velikost obrazovky, rozlišení a výkon. [41][42][43]

2.4 Figma

Figma je grafický nástroj sloužící k jednoduchému a rychlému vytváření prototypů aplikace. Prostředí je primárně dostupné na webu, ale existuje volba, kdy si uživatel stáhne aplikaci přímo do počítače. Figma také nabízí mobilní aplikaci, která je vhodná k testování a interakci s vytvořeným prototypem přímo na zařízení v reálném čase. Jelikož jde primárně o webovou aplikaci, není důležité, zda je projekt vytvářen v operačním systému Windows, macOS nebo open-source Linux. Stejné možnosti platí i pro mobilní aplikaci. [44]

Jelikož je Figma cloudová, umožňuje spolupracovat až 100 lidem najednou na jednom designovém souboru. Už tento fakt ji odlišuje od klasických editorů, které většinou ukládají soubory přímo do paměti počítače. Cloudová výhoda, pro kterou je Figma často používaný nástroj v mnoha firmách, je v některých případech naopak i nevýhoda. V případě off-line práce na projektu se soubory z cloudu nenačítají a tím pádem ani neukládají. Tudíž pokud se jiný spolupracovník v danou chvíli chce podívat na změny provedené v projektu, žádné neuvidí, zobrazí se mu pouze stará verze beze změn. [44]

Komunita uživatelů se neustále rozšiřuje nejen díky možnosti využití tzv. pluginů, ale i samotné ceně nástroje. Běžně stačí program Figma používat v rámci balíčku *Starter pack*, která je zcela zdarma a nabízí většinu důležitých funkcí. Placená varianta má neomezenou verzovací historii, u varianty *Starter pack* ukládá verze pouhých 30 dní. [44]

3 ZABEZPEČENÍ WEBOVÝCH APLIKACÍ

V moderním digitálním světě přineslo rozšíření webových aplikací novou úroveň konektivity a komfortu, která způsobila revoluci ve vzájemné interakci, transakcích a komunikacích mezi jednotlivci. Současně s tímto rozvojem však vznikají kybernetické hrozby, které zneužívají zranitelnosti obsažené v architektuře webových aplikací, čímž ohrožují bezpečnost citlivých dat a kontinuitu základních služeb. V důsledku se nutnost zabezpečit webové aplikace stala prioritou, která nutí společnosti investovat do robustní obrany a zkušeného personálu schopného odvrátit kybernetické hrozby.

Jako příklad lze uvést zranitelnost osobních a finančních údajů uložených v databázích webových aplikací. Bez spolehlivých bezpečnostních opatření jsou tyto informace náchylné ke zneužití, což může vést ke krádeži identity a finančnímu zneužití. Příkladem jsou tzv. SQL injection útoky, při nichž útočníci zneužívají zranitelnosti v databázích, aby získali přístup k citlivým údajům. Phishingové útoky, jejichž cílem je přimět uživatele k vyzrazení důvěrných informací, podtrhují rozšířené riziko, které představují jak pro jednotlivce, tak pro celé společnosti. [45]

K nutnosti spolehlivého zabezpečení webových aplikací přispívá i požadavek na dodržování právních předpisů, jehož příkladem je dodržování přísných rámců, jako je obecné nařízení o ochraně osobních údajů (zkr. GDPR). Nedodržení těchto předpisů nejenže vystavuje firmy značným finančním pokutám, ale také podkopává důvěru uživatelů, čímž způsobuje vážné poškození reputace. [45][46]

3.1 Autentizace a autorizace

Autentizace a autorizace jsou dva zásadní bezpečnostní procesy, které administrátoři používají k ochraně systémů a dat. Každá z těchto metod je důležitá a obvykle souvisí s internetovými službami jako klíčovými prvky jejich infrastruktury. Autentizace a autorizace hrají při zabezpečení aplikací a dat odlišné, ale stejně významné role. Při autentizaci je ověřována totožnost uživatelů, kteří chtějí získat přístup do systému. Ověření identity probíhá obvykle v určité formě, např. přihlášení aplikace pomocí zadání uživatelského jména a hesla, které zná pouze daný uživatel. Naopak v procesu autorizace jsou kontrolována oprávnění osoby nebo uživatele, zda má dostatečná práva pro přístup ke zdrojům. Pro příklad si lze představit jednoduché popsání rolí uživatele. Běžný uživatel může mít obvykle práva na vytváření nových požadavků, ale administrátoři mohou nastavit práva i tak, že uživatel bude mít práva

jen pro čtení požadavků, které se jej přímo týkají. Proces autentizace se uskutečňuje před procesem autorizace. Jejich kombinací se určuje zabezpečení systému a bez správného nastavení autentizace i autorizace není možné dosáhnout spolehlivé bezpečnosti. [47][48]

3.1.1 Lightweight Directory Access Protocol

Lightweight Directory Access Protocol, zkr. LDAP, je softwarový protokol, který je používán k vyhledávání informací nebo zařízení v síti, ověřování uživatelů nebo ke kontrole přístupů. Lze jej použít k vytvoření centrálního ověřovacího serveru pro organizaci nebo ke zjednodušení přístupu k interním serverům a zařízením. Síť může uchovávat informace nejen o zařízeních, ale i uživatelská jména a hesla, které mohou sloužit k autorizaci uživatelů. [49][50]

LDAP je standardní protokol určený k údržbě a přístupu k adresářovým službám v rámci sítě. Adresářová služba lze připodobnit k telefonnímu seznamu síťových zdrojů (soubory, tiskárny, uživatelé, zařízení a servery). LDAP umožňuje uživatelům vyhledat konkrétní tiskárnu v síti a bezpečně se k ní připojit. Adresáře LDAP obvykle uchovávají data, ke kterým se přistupuje, ale která se zřídka mění. LDAP je optimalizován pro rychlý výkon čtení dat, a to i u velkých datových sad, ale jeho rychlost v rámci zápisu do databáze je výrazně pomalejší. [49]

LDAP je běžně používán k vytváření centrálních ověřovacích serverů, které obsahují uživatelská jména a hesla všech uživatelů v síti. K serveru LDAP se pak mohou připojovat aplikace a služby pro ověřování a autentizace uživatelů. LDAP autentizace je proces ověřování uživatelů dle jejich jména a hesla uložených v adresářové službě, jako je OpenLDAP nebo Active Directory. Správci mohou v rámci adresáře vytvářet uživatelské účty a udělovat jim oprávnění. Když se uživatel pokusí získat přístup ke zdroji, je odeslán požadavek na server. Server zjistí, zda je zadané uživatelské jméno a heslo platné na základě údajů v adresáři a pokud dojde ke shodě, je nutné ověřit, jestli uživatel oprávněn k přístupu k požadovaným informacím. [50]

3.1.2 Single Sign-On

Jednotné přihlášení, angl. Single Sign-On (zkr. SSO), je technologie, která sdružuje několik různých možností přihlášení k aplikacím. Díky SSO stačí, když uživatel zadá své přihlašovací údaje pouze jednou na webové stránce, aby získal přístup ke všem aplikacím. [51]

Jednotné přihlášení je důležitým aspektem mnoha řešení správy identit a přístupu. Ověření identity uživatele je klíčové pro zjištění, jaká oprávnění by měl mít každý uživatel. Jednotné přihlášení se často používá v podnikovém kontextu, kdy uživatelské aplikace spravuje interní IT tým. Používání této technologie je výhodné nejen pro zaměstnance podniku, ale i pro externí uživatele, kteří používají dané aplikace. [52]

Jednotné přihlášení funguje na základě vztahu založeném na důvěře mezi aplikací, známou jako poskytovatel služeb, a poskytovatelem identit, jako je OneLogin. Tento vztah je často založen na certifikátu, který si poskytovatel identit a poskytovatel služeb vyměňují. Tento certifikát lze použít k podpisu informací o identitě, které jsou odesílány od poskytovatele identity poskytovateli služeb. V SSO mají tyto údaje o identitě podobu tokenů, které obsahují identifikační informace o uživateli, např. e-mailová adresa nebo uživatelské jméno. [51][52]

4 TESTOVÁNÍ WEBOVÝCH APLIKACÍ

V současné době vzniká stále více webových aplikací a s každým napsaným řádkem kódu se zvyšuje riziko vzniku chyb. Obecně platí, že náklady na opravu chyb rostou exponenciálně s časem, než dojde k objevení chyb. Testování webových aplikací se obvykle skládá z několika kroků, které zajišťují, že je aplikace plně funkční a běží bez problémů a bezpečně. Je nezbytnou součástí vývoje webových aplikací a zajišťuje, aby aplikace před vydáním správně fungovala.

Testování webové aplikace rovněž neznamená pouze hledání běžných chyb nebo omylů, ale také zkoumání rizik spojených s kvalitou aplikace. Testeři musí do hloubky chápat architekturu a klíčové oblasti webové aplikace, aby mohli efektivně plánovat a provádět operace testování.

Testování webové aplikace zahrnuje analýzu závad webu ve srovnání s obecnými závadami softwaru. Webové aplikace je nutné zkoušet v různých internetových prohlížečích, aby testeři mohli identifikovat oblasti, na které je nutné se při testování webové aplikace speciálně zaměřit.

4.1 Typy testování

V praxi existují čtyři základní typy testování webových aplikací: testování statických nebo dynamických webových stránek, testování internetových obchodů a testování internetových aplikací na mobilních zařízeních. Prověření jednotlivých oblastí během testování zajistí, že webová aplikace bude robustní, plně funkční a připravena pro použití koncovými uživateli. [54]

První testování je zaměřeno na statické webové stránky. Pod nimi si lze představit stránky, na kterých se zobrazuje totožný obsah jako je uložen v databázi. Stránky pak postrádají dynamické funkce, ale vyniknou svým atraktivním uživatelským rozhraním, tudíž jde často o designové webové stránky. Při testování statických stránek je pozornost věnována především posouzení atraktivity uživatelského rozhraní. Testování se skládá z kontroly vizuálních prvků, jako jsou barvy, velikost písma a rozložení komponent. Mimo jiné je potřeba i ověřit správnou funkčnost kontaktních formulářů a ostatních odkazů použitých na webu. [53][54][55]

Testováním dynamických webových stránek je kontrolována nejen atraktivita webové stránky, ale i celková funkčnost dynamických prvků. Dynamické stránky se skládají jak z frontendové tak i backendové části. Stránky se na základě proměnlivých dat pravidelně aktualizují a mění podle požadavků uživatele. Součástí dynamických stránek je mnoho funkcí zahrnujících například to, jak se má chovat tlačítko, pokud je stisknuto uživatelem. Kontrolována je i funkčnost backendu, například zda jsou po stisknutí zmíněného tlačítka navrácena zpět uživateli na frontend správná data. [53][54][55]

Velmi časově náročné testování vyžadují i elektronické obchody. Jelikož se elektronické obchody skládají z mnoho podstránek a složitých funkcí, jsou náročné na údržbu i na testování. Tester musí různými způsoby kontrolovat například funkčnost nákupního košíku, mimo jiné, zda správně funguje autentizace uživatele, a hlavně bezpečné provádění plateb. Oblastí, které je potřeba ověřit, je mnoho, ale nejdůležitější je celková bezpečnost a funkčnost. [53][54][55]

Jednodušší, ale stejně důležité, je i testování webových stránek na mobilních zařízeních. Tester kontroluje kompatibilitu na různých velikostech zařízení. Je nutno ověřit, zda se webová stránka nebo aplikace vhodně zobrazuje a je responzivní na všech zařízeních a platformách. [54]

4.2 Nástroje pro testování

Testování hraje při vývoji webových aplikací klíčovou úlohu, neboť zajišťuje správnost funkcí, optimální výkon a bezchybnou uživatelskou zkušenost.

Nástroje pro automatizované testování, jako je například Selenium, zajišťují funkčnost a správnou integritu webové aplikace v průběhu vývoje při vytváření opakovaných a systematických testů. Selenium je open-source projekt určený k automatizaci interakcí s prohlížečem a vytváření funkčních i regresních testů. Je známý svou všestranností a flexibilitou vůči programovacím jazykům, takže eliminuje nutnost osvojit si znalosti specifického testovacího skriptovacího jazyka. [55]

Cypress je moderním end-to-end testovacím frameworkem navrženým speciálně pro webové aplikace. End-to-end testy ověřují celou škálu možných problémů v aplikaci z pohledu uživatele. Díky funkcím načítání aplikace v reálném čase a robustním metodám pro ladění zvyšuje Cypress kvalitu testování a poskytuje tak důkladné vyhodnocení funkčnosti. [55]

Pro hodnocení výkonnosti webové aplikace za různých provozních podmínek jsou klíčové nástroje pro testování vytížení. Nástroje pomáhají identifikovat slabiny výkonnosti a zajistit, že aplikace unese maximální vytížení. Apache JMeter je jedním z předních nástrojů pro testování. Jde o open-source nástroj, který je schopen simulovat různé scénáře a úrovně zatížení. Další nástroj, Gatling, je opět open-source nástroj, který využívá jazyk Scala pro skriptování realistických scénářů. Nástroj je velmi škálovatelný a zvládá velké objemy provozu. [55]

Při vývoji webových aplikací je bezpečnost nepřehlédnutelným aspektem. Identifikace potenciálních zranitelností a slabých míst je klíčovým krokem k zajištění bezpečnosti aplikace. Pro tento účel jsou využívány specializované nástroje. Mezi nejvýznamnější nástroje v této oblasti patří OWASP ZAP a Burp Suite. OWASP ZAP je široce využívaný open-source nástroj pro identifikaci bezpečnostních zranitelností během vývoje a testování aplikací. Na druhé straně Burp Suite poskytuje komplexní sadu výkonných nástrojů určených k testování bezpečnosti webových aplikací, včetně funkcí pro skenování, průzkum a využití identifikovaných zranitelností. [55]

Pro udržení integrovaného stavu webové aplikace je nezbytné používat nástroje pro údržbu kvality kódu a kontinuální integraci. Využitím těchto nástrojů mohou vývojáři s jistotou zajistit kvalitu a spolehlivost svých webových aplikací. Významným nástrojem v této kategorii je Jenkins, open-source automatizační server, který podporuje vytváření, nasazování a automatizaci různých aspektů vývoje softwaru, včetně testování. Pomocí cloudových služeb, jako jsou Travis CI a CircleCI, lze testování a nasazování kódu snadno automatizovat. Začlenění nástrojů pro testování do procesu vývoje webových aplikací může výrazně zvýšit kvalitu, bezpečnost a výkon aplikace a zároveň urychlit vývojový cyklus. [55]

5 GET APPROVAL TOOL

Nástroj *Get Approval Tool*, zkr. GAT, je interní webová aplikace určena pro české pobočky společnosti NXP Semiconductors, sloužící k zefektivnění procesu schvalování financí a nákupu. Umožňuje zaměstnancům vytvářet finanční požadavky, které jsou následně přezkoumány určenými schvalovateli, aby byl zajištěn soulad s firemními zásadami a rozpočtovými pokyny.

Recepční oddělení mohou nástroj využívat pro finanční činnosti související s administrativními záležitostmi, jako je schvalování žádostí o půjčení firemního vozidla nebo půjčení vozidla z autopůjčovny. Prostřednictvím na míru vytvořených funkcí nástroj zvyšuje provozní efektivitu, dodržování předpisů a možnosti rozhodování napříč různými organizačními funkcemi.

Na rozdíl od přímého nákupu zaměstnancem, pověřená osoba provede skutečný nákup od určených dodavatelů až po schválení požadavku. To zajišťuje kontrolované výdaje v souladu s firemními finančními protokoly. Jakmile je nákup proveden, zaměstnanec nahraje fakturu k evidenci. Na konci procesu je požadavek buď uzavřen, pokud je transakce úspěšně zdokumentována, nebo zamítnut, pokud nesplňuje požadovaná kritéria.

Tento nástroj má zásadní význam pro udržení finanční disciplíny a odpovědnosti prostřednictvím řízení přidělování finančních prostředků v rámci poboček v České republice. Bližší informace o této aplikaci jsou uvedeny v kapitole 6, kde je také provedena analýza jejího stavu.

5.1 NXP Semiconductors

NXP Semiconductors je mezinárodní společnost, která je jedním z největších a nejvýznamnějších světových výrobců polovodičů a procesorů. Společnost se zaměřuje na automobilový průmysl, chytrá města, komunikační infrastrukturu, chytré domácnosti, bezpečné datové připojení a další. [56]

Společnost NXP Semiconductors má bohatou historii sahající až do 40. let 20. století, kdy se na vývoji polovodičů významně podílela jak společnost Motorola v oblasti vesmírných technologií, tak společnost Philips. Společnost NXP, která vznikla v roce 2006 sloučením společnosti Philips Semiconductors a dalších subjektů, se stala lídrem v rozmanitých technologiích, jako jsou mikroprocesory, rádia a NFC. [57]

Od svého vstupu na burzu NASDAQ v roce 2010 a zařazení do indexu NASDAQ-100 v roce 2013 je společnost NXP oceňována za svou průkopnickou práci a získala cenu European Inventor Award za pokrok v oblasti NFC. Svou pozici na trhu s polovodiči společnost dále utvrdila fúzí se společností Freescale v roce 2015, díky níž se NXP stala čtvrtou největší polovodičovou společností na světě a největším dodavatelem pro automobilový průmysl. [57]

II. PRAKTICKÁ ČÁST

6 ANALÝZA PŮVODNÍHO STAVU APLIKACE

Ještě před navrhováním nového prototypu uživatelského rozhraní je nezbytné provést analýzu a zhodnocení současného stavu webové aplikace, která disponuje webovým rozhraním s různými oprávněními. Kromě analýzy uživatelského rozhraní je nutné zkoumat také funkcionality aplikace pro získání celkového obrazu o fungování a možných nedostatcích.

Úroveň oprávnění určuje, jaký obsah je uživateli prezentován. Manažer má přístup ke stránkám s podrobnějšími informacemi o požadavcích, zatímco běžný zaměstnanec vidí pouze omezené informace. Tato aplikace je určena pro prezentaci zaměstnancům, proto byla pro diplomovou práci zvolena registrace běžného zaměstnance pomocí firemního e-mailu nebo identifikačního čísla.

Na začátek analýzy je nutné zdůraznit, že některá data nemohla být zveřejněna. Toto omezení v přístupu k datům mělo vliv na detailnost analýzy, avšak bylo nezbytné pro dodržení smluvních podmínek a zachování důvěrnosti dat.

6.1 Uživatelské rozhraní

Analýza původního vzhledu webové aplikace byla provedena na všech částech webu s primárním zaměřením na nejpoužívanější, a to je vytváření požadavku z pohledu zaměstnance. Většina sekcí potřebuje nutnou modernizaci uživatelského rozhraní pro zjednodušení využívání aplikace.

Počáteční fáze analýzy byla zaměřena na část, která je uživateli zobrazena ihned po otevření aplikace. Domovskou stránku aplikace je možné vidět na Obrázku 22.

Analýza domovské stránky odhalila několik zásadních nedostatků, které mohou vést ke zmatení uživatelů. Při prvním načtení aplikace je uživateli prezentována stránka, jejíž design a uspořádání nedává jasně najevo, že se jedná o úvodní stránku. Jsou zde viditelné hlavní prvky webové aplikace, jako je navigační menu, odkazy na další interní aplikace a možnost přejít rovnou k vytváření požadavku.

Na úvodní stránku se uživatel dostane v případě zadání adresy domény do prohlížeče a na první pohled neposkytuje uživateli známky identity nebo účelu. Jediná komponenta, která uživatele informuje o tom, že se nachází na webové aplikaci, je titulek *GET APPROVAL TOOL internal* vedle loga společnosti. Podle části titulku *internal* je jasné, že webová aplikace slouží jen pro interní použití, a tudíž by k ní externí uživatelé neměli mít přístup.

Navíc je již na první pohled jasné, že celkový vizuální dojem z domovské stránky je značně neuspokojivý a nesouladný s aktuálními designovými trendy. Absence jasně definované struktury komponent ztěžuje orientaci na úvodní stránce a její špatná intuitivnost používání. Tyto nedostatky v prezentaci klíčových prvků společně se zastaralým designem jsou hlavní překážky, které mohou negativně ovlivnit uživatelskou zkušenost.

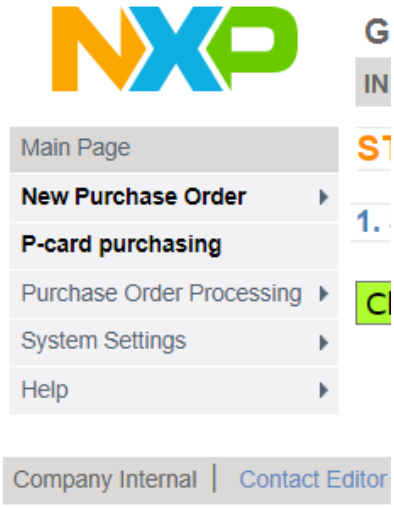
Je potřeba také zmínit neaktuálnost některých částí webu. Na pravé straně webu se nachází komponenta pro vyhledávání, ve které text napovídající možnosti vyhledávání obsahuje neaktuální informaci ohledně názvu firmy. V textu je uveden *All FreeScale Search,* což není aktuální název firmy. Po akvizici se firma stala NXP Semiconductors a tudíž je důležité tento text aktualizovat, aby odpovídal současnému stavu.

Taky by bylo vhodné aktualizovat cílovou adresu prokliknutí odkazu s logem firmy NXP nad navigačním menu. Loga firem nebo daných aplikací, která se často nachází v levém horním rohu, běžně odkazují na úvodní stránku aplikace. Tady tomu tak není, po kliknutí na logo je aplikace přesměrována na úvodní stránku společnosti NXP.

Obrázek 22: Původní stav úvodní stránky aplikace

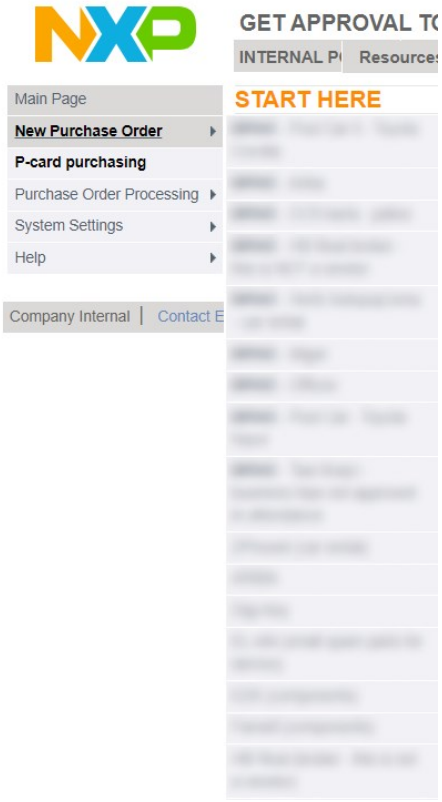
Druhá analýza byla zaměřena na navigační menu, které je vyobrazeno v levé části webové aplikace (Obrázek 23). Jedná se o jeden z nejdůležitějších uživatelských scénářů. Menu v aplikaci funguje jako hlavní ovládací centrála, kterou uživatel používá pro pohybování se na ostatní podstránky. Obsahuje odkazy na zmíněné podstránky, jako je například zobrazení všech vytvořených požadavků aktuálního uživatele, vytvoření nového požadavku podle šablony a další odkazy, které jsou dostupné jen s potřebnou úrovní oprávnění.

Nadcházející uživatelský scénář popisuje vytvoření požadavku. V prvním kroku je potřeba rozhodnout, o jaký typ požadavku se bude jednat, jelikož výběr definuje následující kroky a potřebné informace, které uživatel musí poskytnout. Pro tento výběr slouží v navigačním menu dva odkazy, první je *New Purchase Order* a druhý *P-card purchasing*.



Obrázek 23: Původní stav navigačního menu

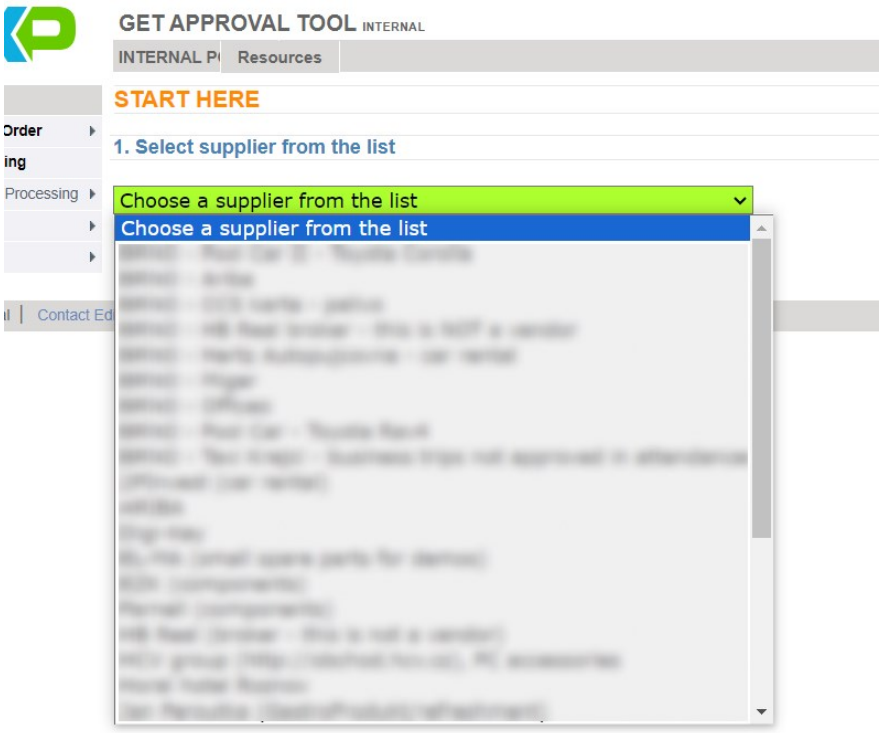
První odkaz *New Purchase Order* definuje typ požadavku přímo od smluvního partnera, po jeho kliknutí poskytne uživateli seznam dostupných dodavatelů (Obrázek 24). Celkový seznam obsahuje velké množství dodavatelů, ve kterém je náročné najít ten, který zrovna uživatel potřebuje pro vytvoření požadavku. Dle typografie se v seznamu nachází velký objem textu, který je zařazen blízko pod sebe a není nijak prostorově rozdělen. Uživateli se při čtení text po kratší době začne slučovat, takže je potřeba zaručit, aby bylo procházení seznamu lépe čitelné a uživatel se v něm neztratil.



Obrázek 24: Původní stav seznamu dodavatelů v navigačním menu

K seznamu dodavatelů lze přistoupit nejen přes odkaz v navigačním menu, ale i přes úvodní stránku *Main Page.* Tuto možnost reprezentuje komponenta s rozbalovacím seznamem (Obrázek 25), kde si uživatel může vybrat ze stejných dodavatelů, jako jsou v odkazu *New Purchase Order* v navigačním menu. Z typografického hlediska je text seznamu stejně špatně čitelný jako ten v navigačním menu. Další kritická chyba související s velkým počtem prvků v seznamu je neschopnost v něm vyhledávat.

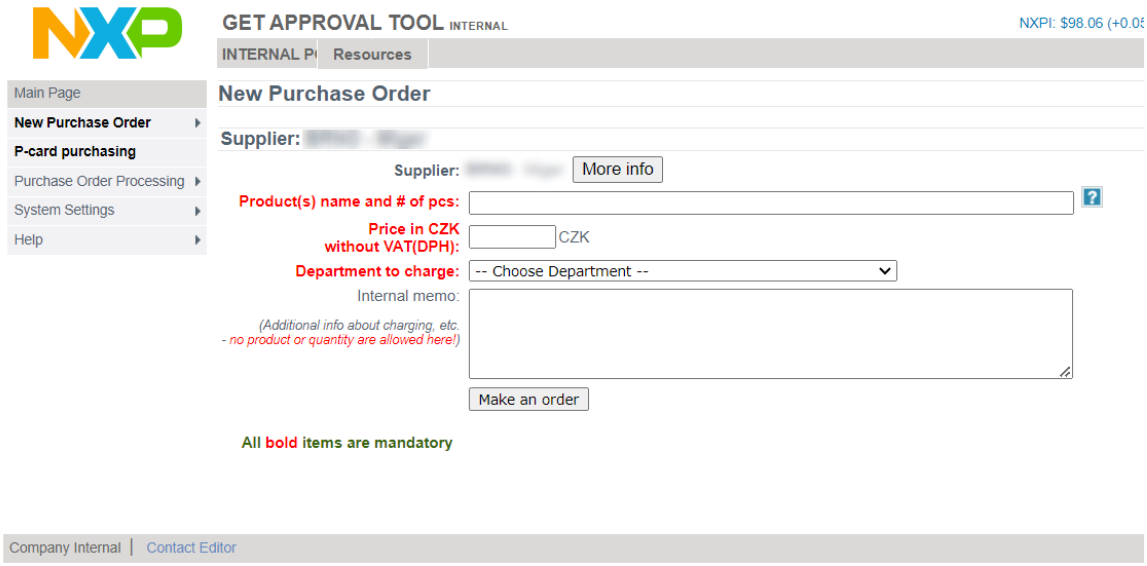
Selekci dodavatele lze zjednodušit pomocí podpoření funkce pro vyhledávání přímo v komponentě sloužící pro výběr. Uživatel by při otevření seznamu zadal určitý počet znaků a byl by mu navrácen seznam dodavatelů odpovídající jeho vyhledávání. Navíc by bylo vhodné každého člena seznamu opatřit dostatkem vizuálního prostoru a celkově seznam abecedně seřadit, aby se text lépe četl.



Obrázek 25: Původní vzhled seznamu dodavatelů na úvodní stránce

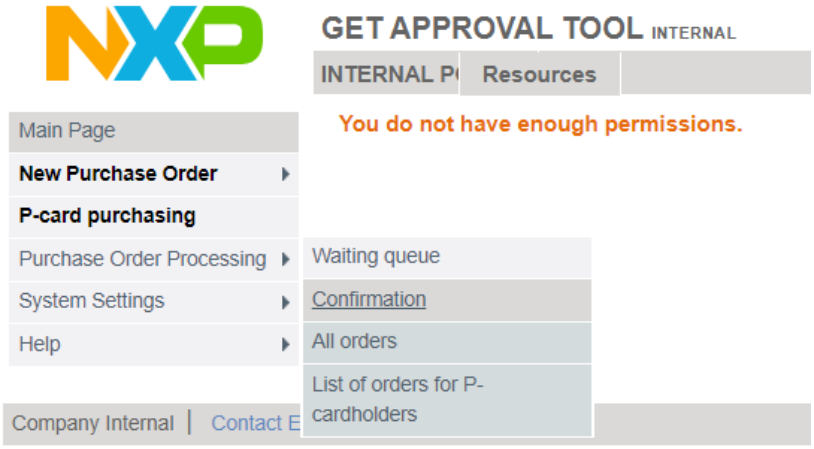
Jakmile si uživatel vybere dodavatele ze seznamu, je přesměrován na stránku pro vytváření požadavků (Obrázek 26). Dle selekce je uživateli vyobrazena šablona formuláře s potřebnými textovými vstupy. Každá zmíněná šablona potřebuje konkrétní informace, které musí uživatel zadat. Pro příklad, dodavatel typu *Karta palivo* nepotřebuje zadávat počet produktů, ale pouze název *Tankování benzín*, datum uskutečnění a výslednou částku.

Tato část není z hlediska UX/UI tak kritická, ale z vizuálního hlediska je potřeba vytvořit modernější návrh, který by odpovídal novodobým standardům vývoje webových aplikací. Jak bylo již uvedeno v teoretické části, v dnešní době rychlé digitalizace a velké množství konkurence na trhu je klíčové neustále vylepšovat produkty nejen z pohledu použitelnosti, ale i z pohledu atraktivity uživatelského rozhraní.



Obrázek 26: Původní vzhled vytváření požadavku

Jako další nedostatek navigačního seznamu lze brát zobrazování odkazů, na které zaměstnanec nemá dostatečné oprávnění. Běžný uživatel by měl správně vidět na stránce pouze ty odkazy a informace, ke kterým má práva, a to např. přehled o všech svých požadavcích. Nemá pak právo zobrazovat například všechny požadavky, které čekají na schválení, i když je odkaz *Confirmation* v navigačním menu viditelný (Obrázek 27). Po jeho kliknutí na daný odkaz je sice uživateli správně zobrazena hláška, že na zobrazení vybrané stránky nemá dostatečné oprávnění, ale mnohem lepší pro by bylo daný odkaz ani nezobrazovat. Zmíněné odkazy jsou pak pro běžného uživatele zcela nepoužitelné, tudíž je uživateli prezentováno UI a funkce, ke kterým nemá přístup. Tímto nedostatkem může uživatele vést ke zbytečnému zmatku při používání aplikace a ke zvýšení složitosti navigace.



Obrázek 27: Původní vzhled odkazu *Confirmation*

Stránka, která shromažďuje a prezentuje veškeré vytvořené požadavky daného uživatele, je v původní verzi pojmenována jako *All Your Purchase Order List* (Obrázek 28). Na stránku uživatel přistoupí při kliknutí odkazu *All Orders* v rozbalovací části navigačního menu. Tělo stránky je rozděleno do třech částí, kde jsou jednotlivé komponenty řazeny pod sebe. První je zmíněný titulek stránky a hned pod ním se nachází dva selektory nabízející filtrování; první pro filtrování podle dodavatele a druhý pro filtrování podle divize požadavku. Druhá část nabízí uživateli vytvoření tisknutelné verze tzv. reportu požadavků. Poslední a nejdůležitější částí stránky je tabulka, která prezentuje veškerá data požadavků uživatele.

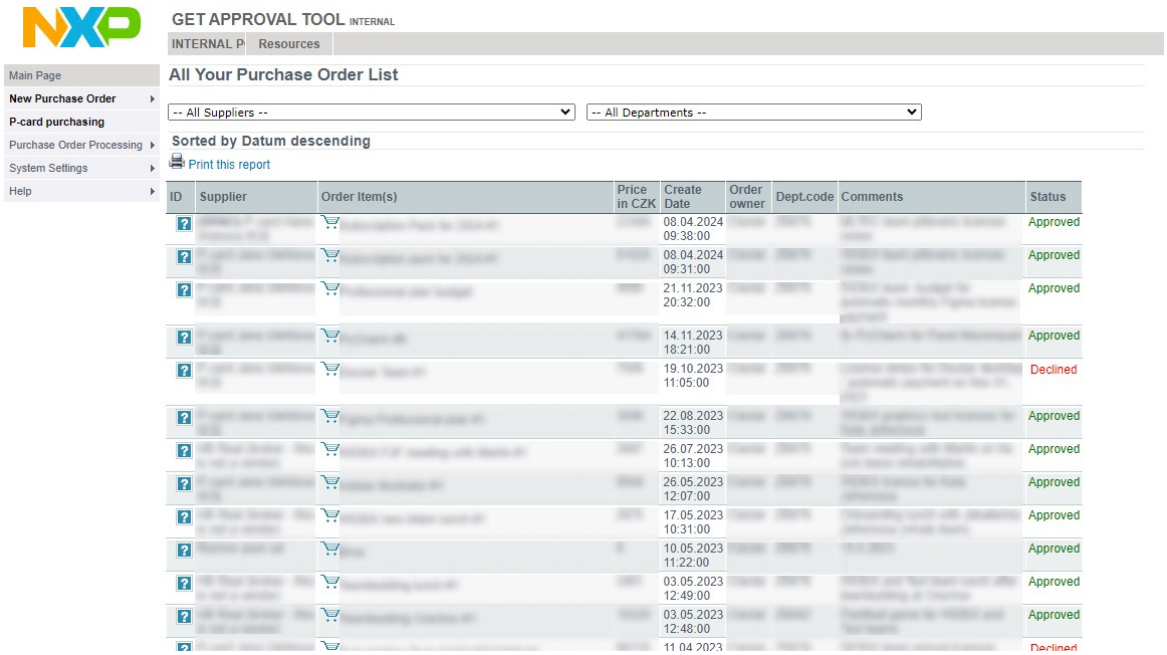
První část analýzy stránky je zaměřena na možnost filtrování požadavků. Zastaralost nabízených filtrů je už od prvního pohledu jasná, ale primárním problémem je zde chybějící prvek, který by umožnil uživateli vyhledávat požadavky, např. podle komentářů nebo názvů položek. Bez této funkce může být pro uživatele komplikované najít hledaný požadavek i přes možnost filtrování. Samotné filtry nejsou z hlediska UX/UI tak problematické, ale působí zastarale a při novém navrhování bylo by vhodné zapracovat na jejich vizuální atraktivitě.

Další část dává uživateli najevo, jak je tabulka seřazena a nabízí i funkci pro vygenerování tisknutelného reportu. V seřazení je chyba v nesourodosti textu. Text zmiňuje, že tabulka je seřazena podle data sestupně, text je napsaný napůl v anglickém a českém jazyce. Je potřeba zaručit srozumitelnost stránky tak, že bude veškerý text konzistentní a jednojazyčný. Tlačítko pro vytvoření reportu je s ikonou symbolizující tiskárnu v pořádku, pouze by bylo vhodnější umístit jej do pravé části stránky, ideálně vedle dalšího tlačítka *Create New Order*, které by sloužilo pro přesunutí na stránku vytváření požadavku. To by výrazně zlepšilo celkovou navigaci uživatele napříč aplikací.

Nejvíc kritickou částí stránky je tabulka, která je určena k reprezentaci dat požadavků (Obrázek 28). Informace jsou od sebe sice vizuálně odděleny, ale na jednom řádků je jich mnoho a obsahují mnoho textu. To znamená, že při čtení se uživateli text začne slučovat a bude obtížné najít klíčové informace, zejména při větším počtu požadavků v tabulce. Hlavní problém tabulky je zobrazení mnoha sloupců a veškerých informací, takže tabulka nemůže být responzivní. Zvětšení rozestupů mezi řádky a sloupci by pomohlo, aby byly jednotlivé údaje čitelnější a méně seskupené. Zobrazují se zde i komentáře k požadavkům, které zabírají v tabulce skoro 25 %, ale nemají takovou prioritu. Jako jedno z možných řešení se nabízí zobrazení komentářů buď po kliknutí na ikonu symbolizující komentář nebo jejich zobrazení

v detailnějším popisu. Takový přístup by v tabulce uvolnil více místa a zlepšil celkovou přehlednost.

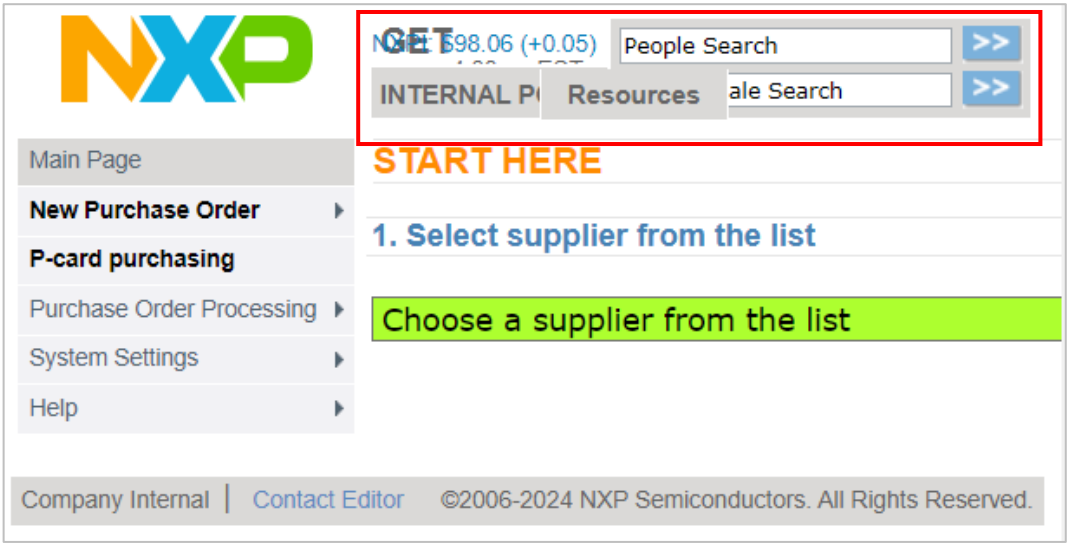
Co v tabulce není špatně, je dostatečná viditelnost aktuální stavu požadavku. Schválené požadavky jsou označeny kontrastně zelenou barvu a ty zamítnuté červenou. Bylo by vhodné pozměnit barvu vůči stavu, v jakém se požadavek aktuálně nachází. Zelená pro schválený požadavek by mohla zůstat totožná, ale zamítnutý požadavek by mohl mít šedou barvu a do budoucna by nemusel být tak významný. Naopak červená barva by byla přiřazena požadavku ve stavu tzv. *Needs Action*, který musí být viditelný, jelikož se očekává buď od uživatele nebo schvalovatele nějaká aktivita, a to v nejkratším možném čase. To by zdůraznilo prioritu různých stavů požadavků, a naopak snížilo prioritu těm, které již nevyžadují žádnou aktivitu.



Obrázek 28: Původní vzhled stránky zobrazující požadavky

Poslední nedostatek, který bude zařazen do analýzy původního stavu, je celková responzivita webové aplikace. Jelikož jeden z cílů implementace je zlepšení mobility zaměstnanců při používání aplikace, je zaručení responzivity klíčové, aby mohli zaměstnanci aplikaci využívat na jakémkoliv zařízení. Neresponzivní web výrazně zhoršuje použitelnost a přístupnost, zejména v dnešním světě, kde je používáno více různých typů zařízení. Takový web je špatně zobrazován na obrazovkách menších, než jsou obrazovky typických počítačů, například tabletech a chytrých telefonech. Špatné zobrazení nutí uživatele k přibližování a oddalování nebo horizontálnímu posouvání, aby si mohli zobrazit obsah, což může být nepohodlné.

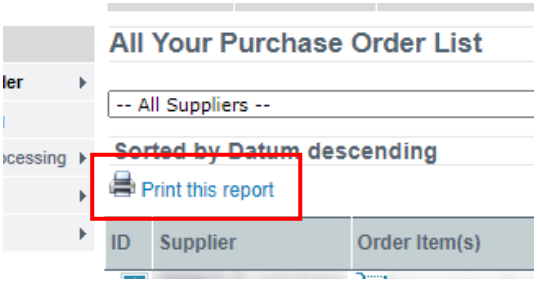
Přesně tyto chyby lze vidět na Obrázku 29 v označené části. Místo toho, aby byly komponenty na stránce navrhnuty tak, aby se dokázaly přizpůsobit aktuálnímu rozlišení a tím byly flexibilní, jsou pevně na své pozici. Tím pak vzniká nevhodné chování, kdy se komponenty ve vrchní části webové aplikace vzájemně překrývají. Jedno z vhodných řešení se nabízí mít flexibilní zobrazení komponent, to znamená že bude jiná pozice komponenty na počítači než na mobilu. Například dvě komponenty sloužící pro vyhledávání s textem *People Search* a *All Freescale Search* by se na telefonu přesunuly po titulek aplikace *GET APPROVAL TOOL*, tudíž by se předešlo danému překrytí, jak lze vidět na Obrázku 29.



Obrázek 29: Původní responzivita aplikace

6.2 Funkčnost aplikace

V rámci analýzy funkčnosti aplikace byl proveden důkladný průzkum již zavedených funkcí a celkového chodu webové aplikace. Přestože byly identifikovány určité nedostatky, většina klíčových funkcí vykazuje správnou funkčnost. Proces vytváření požadavků je bezchybný, uživatelům umožňuje vytváření požadavků dle vybraného typu. Systém kontroly práv uživatele funguje efektivně a zajišťuje, že jsou určité informace dostupné pouze oprávněným osobám. Aplikace dále nabízí přehledné zobrazení informací o vytvořených požadavcích uživatele a umožňuje tvorbu tisknutelných reportů pro finanční oddělení (Obrázek 30).



Obrázek 30: Odkaz na vytvoření reportu

Jedním z prvních a hlavních problémů je závislost na použití virtuální privátní sítě, dále jako VPN, pro přístup uživatelů do aplikace, což s sebou přináší několik komplikací. Aplikace je sice dostupná z jakéhokoli zařízení schopného připojit se přes VPN, ale prakticky to většinou znamená, že uživatelé jsou omezeni pouze na firemní zařízení. Toto omezení negativně ovlivňuje mobilitu uživatelů, kteří nemohou aplikaci používat mimo pracoviště nebo mimo firemní zařízení, což může zapříčinit snížení produktivity a efektivity práce. V dnešní době, kdy se flexibilita a možnost práce z různých lokací stává stále důležitější, představuje závislost na VPN významnou překážku.

Vyžadování použití VPN také omezuje možnosti rychlého a efektivního přístupu k aplikaci v případě, kdy uživatelé potřebují získat data mimo běžné pracovní hodiny nebo z míst mimo kancelář. Kromě toho způsobuje i bezpečnostní obavy, jelikož zaměstnanci mohou být nuceni vyhledávat alternativní způsoby, jak se k aplikaci dostat, pokud není VPN ihned dostupné. Tímto se potenciálně zvyšuje riziko bezpečnostních hrozeb. Pro zlepšení celkové efektivity a uživatelské spokojenosti by bylo vhodné přijmout opatření k zajištění bezpečnějšího a snadněji přístupného prostředí pro všechny uživatele, ať už pracují z kanceláře, z domova, či z jiné lokace.

Dalším významným nedostatkem je způsob přikládání faktur po schválení požadavků, který probíhal prostřednictvím zasílání e-mailů. Tento postup se ukázal jako neefektivní a náchylný k chybám, neboť vyžaduje další manuální práci s dokumenty, což může vést ke zdržení dokončení požadavku nebo ztrátě důležitých informací. Navíc tento proces znamená značné riziko v oblasti bezpečnosti dat, protože e-maily mohou být snadno napadeny nebo odeslány na nesprávnou adresu.

Sběrem zpětné vazby od zaměstnanců, kteří aplikaci pravidelně využívají, se potvrdilo, že tato funkcionalita je považována za zásadní nedostatek. Požadavkem je implementování funkce do samotné aplikace, která by umožňovala přidávání faktur přímo po schválení a realizaci požadavku. Toto by výrazně zvýšilo efektivitu procesu, snížilo riziko chyb,

zjednodušila by se auditovatelnost procesů a celkově by se zlepšila uživatelská zkušenost a bezpečnost.

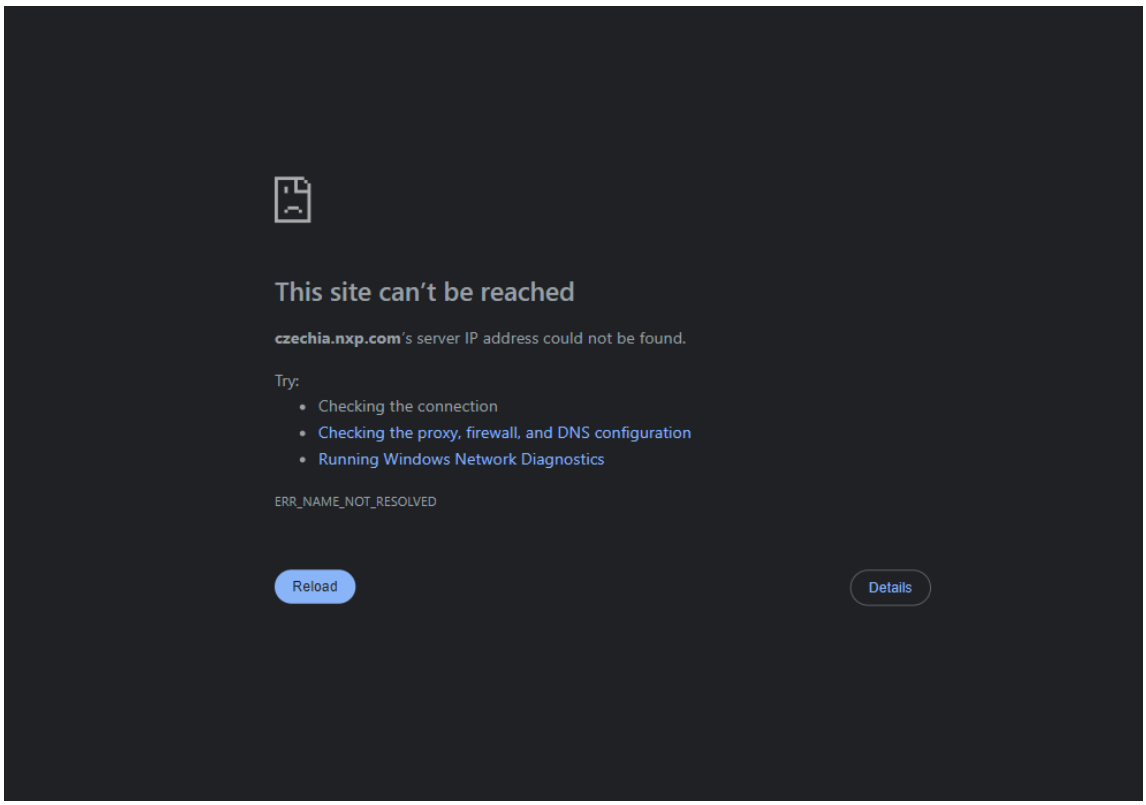
Zavedení této funkce by nejenom splnilo potřeby uživatelů, ale taky by přineslo významné výhody v podobě urychlení a zjednodušení procesů. Tato změna by také mohla významně přispět k celkové digitalizaci dokumentace a procesů v rámci organizace, což je v souladu s moderními trendy efektivních bezpečných podniků.

Absence funkce pro zobrazení notifikací přímo v aplikaci představuje další nedostatek současného systému. V původní verzi systému jsou uživatelé upozorňováni na změny stavu požadavků pouze prostřednictvím e-mailů, což může způsobit významné zpoždění ve chvíli, kdy je uživatel aktivně připojený v aplikaci. Implementace notifikací přímo v aplikaci by značně zlepšila zpětnou vazbu a urychlila by reakční dobu uživatele. Okamžité upozornění na změny by uživatele informovalo efektivněji a umožnilo rychlejší reakci na nové informace nebo potřebné akce, což by celkově zefektivnilo pracovní procesy.

V návaznosti na dříve zmíněné nedostatky spojené s nutností používání VPN pro přístup k aplikaci, další významný problém spočívá v nepřítomnosti jakéhokoliv funkce pro přihlášení nebo autentizaci uživatelů. Tato skutečnost může vést k významným bezpečnostním rizikům. V případě, že uživatel zůstane přihlášený k VPN a opustí své pracovní místo, jeho počítač zůstává zapnutý a přístupný jakékoli třetí osobě. Taková chyba umožňuje neoprávněný přístup k citlivým informacím, což může mít závažné důsledky pro ochranu osobních i firemních údajů.

Toto vážné bezpečnostní riziko zdůrazňuje potřebu zavedení robustního autentizačního mechanismu, který by zajistil, že je každý přístup k aplikaci řádně ověřen a autorizován. Implementace systému pro přihlášení by mohla významně přispět k zabezpečení aplikace, minimalizaci rizika neautorizovaného přístupu a taky zvýšit mobilitu zaměstnanců při používání aplikace.

Mimo jiné se v současné konfiguraci aplikace ukázal zásadní problém, jakým aplikace reaguje při pokusu o přístup bez aktivního VPN spojení. V takovém případě prohlížeč vrací uživatelům chybový stav se sdělením, že hledaná adresa neexistuje, jak je tomu vidět na Obrázku 31. Absence informativního hlášení vede k situaci, kdy uživatelé nemohou rozlišit mezi skutečným výpadkem aplikace a omezením přístupu z důvodu neaktivního VPN. Zavedení jasného chybového hlášení by pomohlo uživatelům lépe pochopit situaci a správně reagovat.



Obrázek 31: Výsledek přístupu do aplikace bez VPN

7 NOVÝ NÁVRH APLIKACE

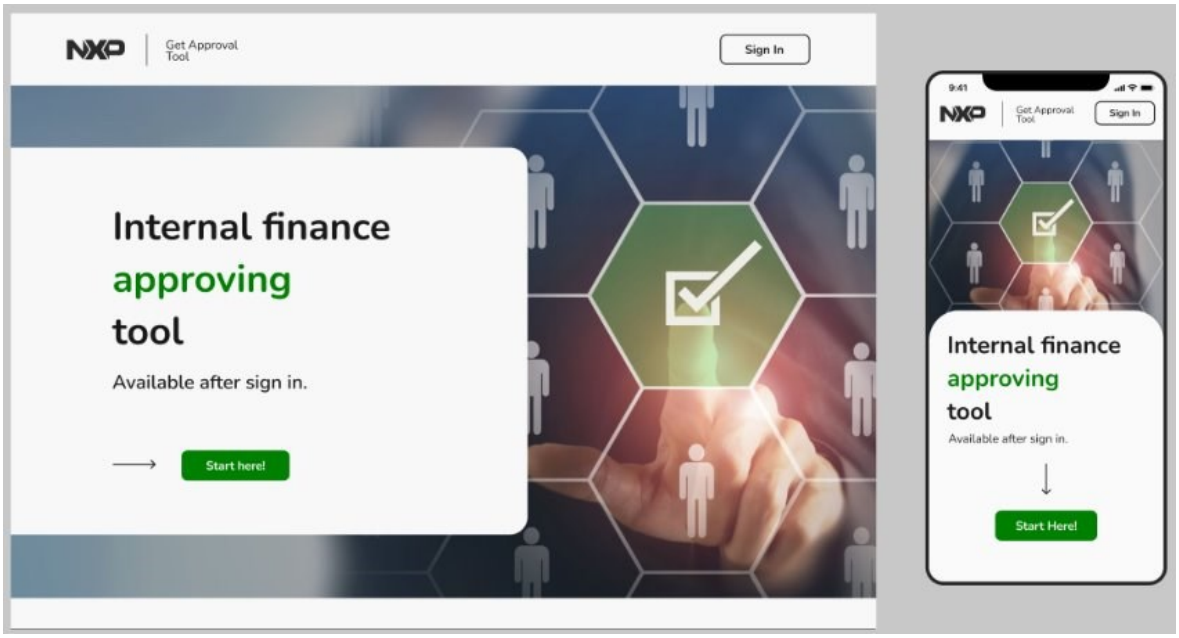
Nový návrh aplikace je zaměřen na vylepšení uživatelského rozhraní aplikace *Get Approval Tool*. V závislosti na předchozí kapitole, kde byla zhodnocena analýza kritických částí webu, byly navrhnuty nové změny odpovídající UX/UI standardům. Oproti původní verzi nové změny celkově vylepšují rozhraní webové stránky na daleko modernější a atraktivnější styl, který je primárně jednodušší na používání. Veškeré části webu byly navrhovány tak, aby jejich použití bylo jednodušší, intuitivní a pro uživatele více informativní. Jelikož UX a UI jsou vzájemně provázané, byl vytvořen návrh, který respektuje principy obou praxí.

7.1 Uživatelské rozhraní

Návrh UI webové aplikace byl tvořen pro všechny části webové aplikace, ale popsán bude pro ty, které byly zmíněny v analýze v předchozím bodě (kap. 6.1). Změna, která vyžadovala nejvíce pozornosti, se týkala nejdůležitějšího uživatelského scénáře, a to vytvoření požadavku z pohledu běžného zaměstnance. Ale první bylo nutné začít se stránkou, kterou uživatel uvidí ihned po načtení webové aplikace.

Pro úvodní stránku, jinak řečeno domovskou stránku, byl vytvořen návrh, který jasně definuje a zároveň informuje uživatele o tom, na jaké stránce se nachází. Nový návrh stránky byl vytvořen tak, aby působil moderně a vizuálně atraktivně. Stránka má dva typy rozložení komponent podle zařízení, na kterém je zobrazena. Na mobilních zařízeních jsou části stránky více zarovnané na střed, naopak pro počítače jsou textové části popisující aplikace na levé straně. V původním stavu této stránky existovala možnost rovnou přejít k vytváření požadavku, ale tato možnost byla odstraněna, protože podobné akce budou dostupné až po přihlášení.

Oproti původnímu stavu aplikace, kdy vzhled úvodní stránky nebyl moderní, informativní a vizuálně atraktivní se nový návrh zaměřuje právě na tuto problematiku. Název aplikace spolu s logem společnosti byl přesunut na vrchní část stránky do navigační lišty tak, aby bylo jasné, kde se uživatel nachází. Jelikož hlavní úkol aplikace je něco schvalovat, tak byl zvolen obrázek s ikonou „fajfky“ představující onen proces schválení, který vylepšuje atraktivitu stránky hned při prvním pohledu. Aby uživatel pochopil, jaký je účel aplikace *Get Approval Tool*, byla vytvořena sekce s krátkým textovým popisem z části překrývající daný obrázek (Obrázek 32).

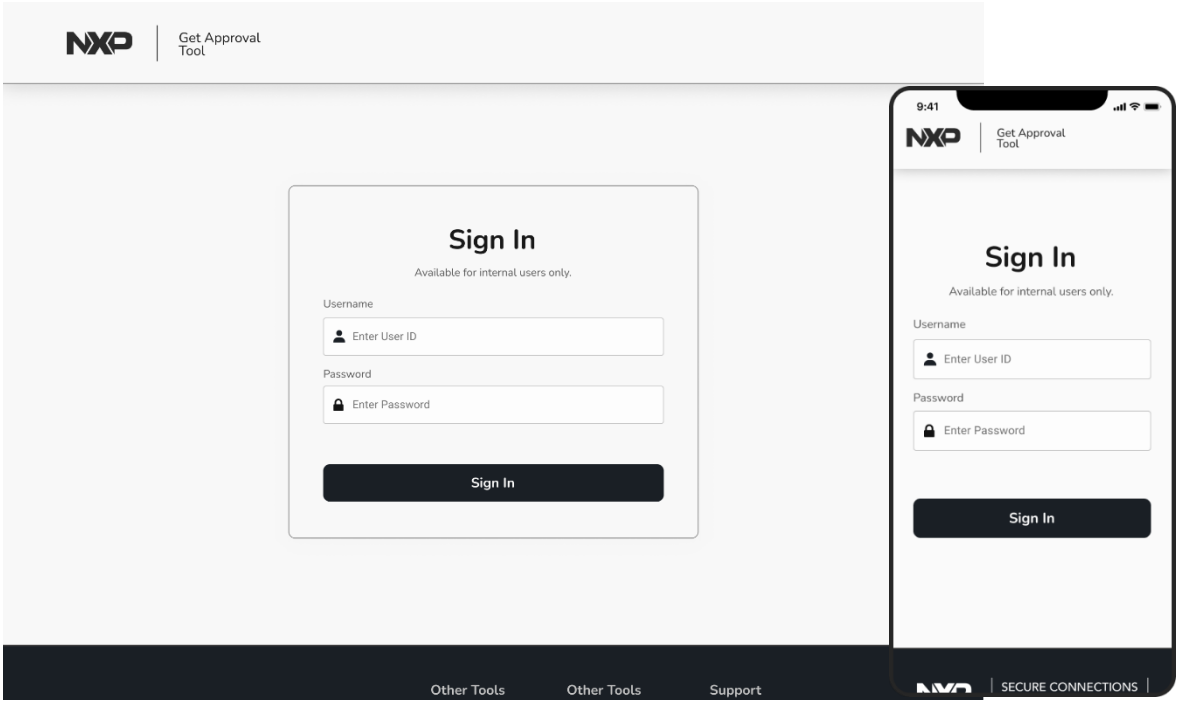


Obrázek 32: Návrh úvodní stránky aplikace

Hned po úvodní stránce následuje stránka pro přihlašování uživatelů. Ta byla navržena tak, aby byla přehledná a jednoduchá na pochopení i použití. Z úvodní stránky se uživatel dostane k přihlášení dvěma způsoby. Nabízí se možnost buď přes tlačítko *Sign In* dostupné v navigační liště, což je standartní pozice, kterou běžně uživatelé očekávají. Druhou možností je tlačítko *Start Here!* umístěné v sekci popisující aplikaci (Obrázek 32).

Na stránce se nachází formulář pro přihlášení, který je navržen tak, aby vypadal jako kartička s rámečkem. Tento rámeček je viditelný pouze na počítači a zařízeních s podobným rozlišením, na mobilních zařízení rámeček mizí a jde vidět pouze daný formulář. To působí moderně a esteticky; kvůli rozlišení telefonu by rámeček vypadal nevhodně a rušil by celkový dojem ze stránky. Uživatelé jsou obvykle zvyklý mít k dispozici dva druhy přihlášení. Buď obyčejné přihlášení nebo registrace. Jelikož jde o interní aplikaci dostupnou jen pro zaměstnance společnosti, tak zde návrh pro možnost registrace není potřeba.

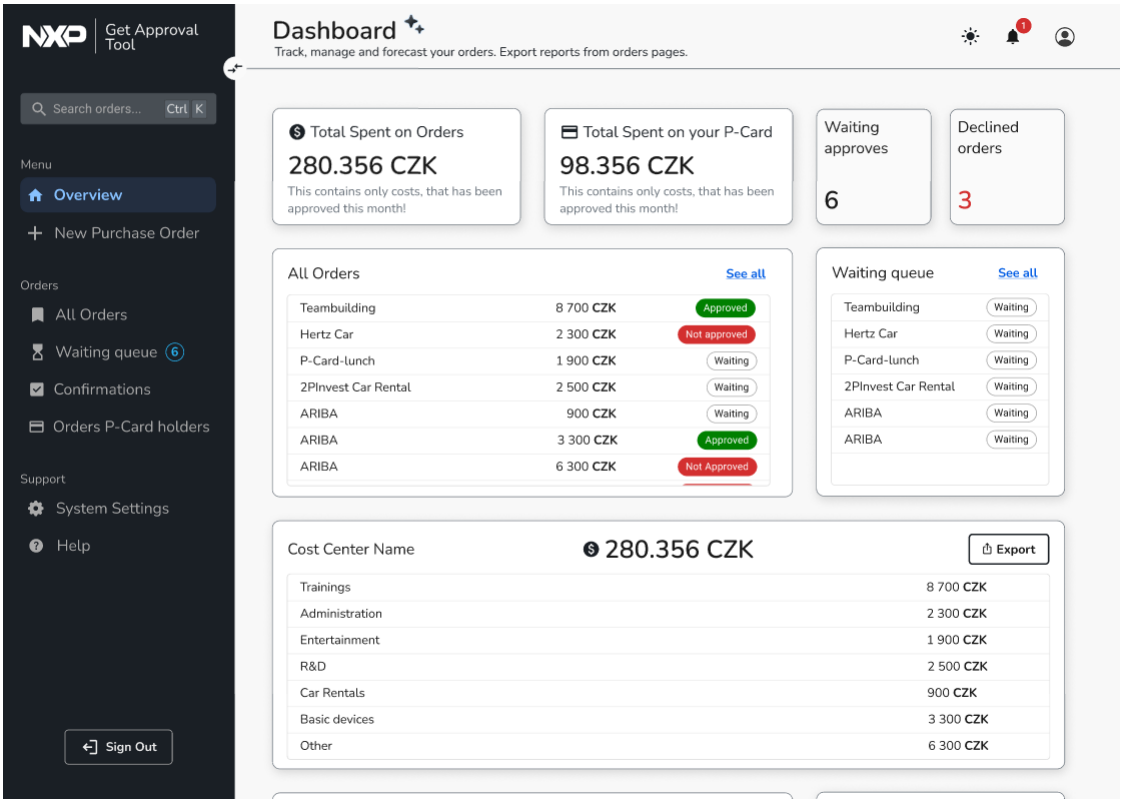
Součástí formuláře je i textová část, která napovídá uživateli, jaké údaje má použít pro přihlášení. Pro zlepšení UX je návrh tlačítka *Sign In* po jeho kliknutí vybaven animací točícího kolečka, který signalizuje probíhající přihlašování. Pokud dojde k úspěšnému přihlášení, je uživatel přesměrován do aplikace na stránku s přehledem nazvanou *Overview*.



Obrázek 33: Návrh responzivní přihlašovací stránky

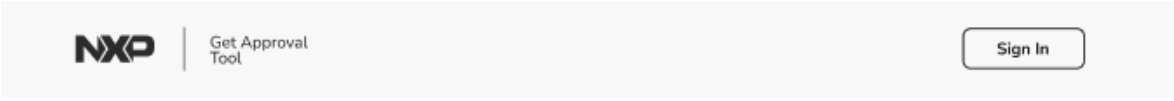
Návrh *Overview* stránky byl vytvořen tak, aby působil jako centrální bod aplikace. Poskytuje uživatelům aktuální přehled o všech podstatných informacích, jako jsou vytvořené požadavky čekající na schválení, zamítnuté požadavky, ale i odkaz na všechny požadavky uživatele nebo celkové útraty dle typu požadavku. V těle stránky jsou přehledně zobrazeny jednotlivé informace v kartičkách různých velikostí, které vytvářejí atraktivní vizuální reprezentaci dat. Zmíněné vytvořené požadavky, které čekají na schválení, využívají nejmenší typ kartiček, jelikož jde pouze o zobrazení titulku, o jakou informaci se jedná, a hodnotu, tedy daný počet požadavků.

Většina kartiček má pevně danou velikost, takže je pak jednodušší zaručit správné responzivní chování stránky. Každopádně, aby byla plně podpořena responzivita stránky, je potřeba definovat, jak se mají kartičky zobrazovat na mobilním zařízení nebo naopak na počítači. V rámci rozložení kartiček na mobilním zařízení ve většině případů bude jedna kartička právě na jeden řádek. Nejmenší velikosti kartiček ukazující pouze titulek a hodnotu, mohou vedle sebe být až dvě, ale musí být mezi nimi zaručena dostatečná mezera. Naopak u největší velikosti kartičky by měla být jejich šířka nastavena tak, aby využívala veškerý dostupný prostor na stránce.

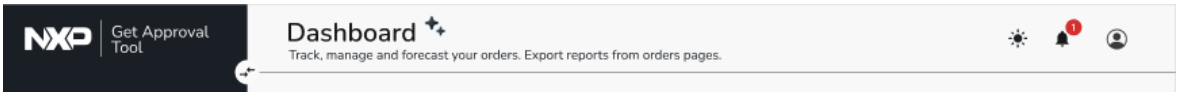


Obrázek 34: Návrh *Overview* stránky

Součástí nejen stránky *Overview*, ale i všech ostatních, jsou tři komponenty, ve kterých se v průběhu používání aplikace nemění jejich obsah. Jedná se o navigační lištu, která se zobrazuje pokaždé na vrchní části stránky a nese informace jako logo společnosti, název aplikace, a v případě mobilního zařízení tlačítko na otevření bočního menu. Jediná změna v obsahu navigační lišty je pouze na úvodní stránce. Místo dvou tlačítek na přepínání barevného módu a další (Obrázek 36), je zde tlačítko na přihlášení, jinak zbytek zůstává statický (Obrázek 35).



Obrázek 35: Navigační lišta na úvodní stránce

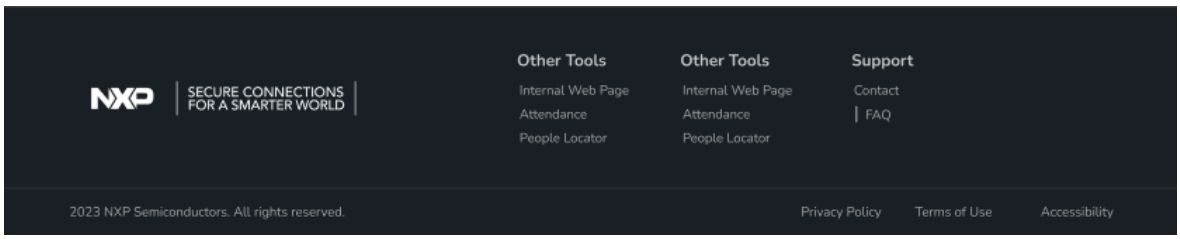


Obrázek 36: Navigační lišta po přihlášení do aplikace

Boční menu slouží jako komponenta pro navigaci napříč aplikací. Obsahuje odkazy na další podstránky, které uživatelům umožňují snadné přepínání mezi různými sekcemi aplikace. Pozadí této komponenty je účelně zvoleno kontrastně, aby byla pozvednuta její informační

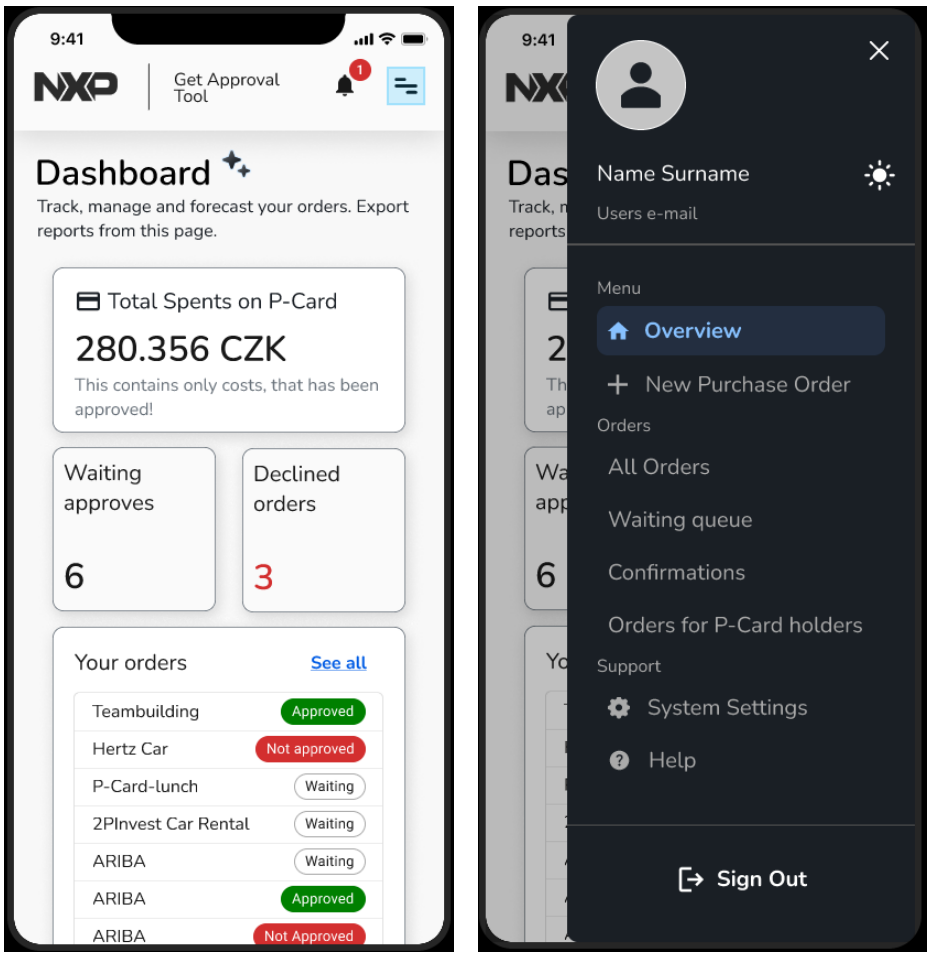
důležitost. Jednotlivé odkazy jsou barevně podsvíceny podle aktuální podstránky, na které se uživatel nachází, jak je vidět na odkazu *Overview* v pravé části Obrázku 38. Tímto způsobem má uživatel vždy jasnou představu o tom, kde právě je, což mu pomáhá udržet přehled o své pozici v aplikaci.

Poslední neměnná komponenta je zápatí, v angl. footer (Obrázek 37). Zápatí obsahuje detailnější informace o společnosti, odkazy na další interní aplikace, sekci autorských práv a práv využití nebo další užitečné odkazy. Komponenta poskytuje jasný způsob, jak najít důležité informace bez ohledu na to, na jaké podstránce se nachází. Dle návrhu se zápatí objeví vždycky na spodní části webové aplikace, jakmile uživatel dojde na konec hlavního obsahu skrz posunutí dolů.



Obrázek 37: Návrh komponenty zápatí

Návrh rozhraní klade důraz na responzivitu, což zajišťuje optimální zobrazení a funkčnost na různých zařízeních. Na mobilních telefonech se navigační menu automaticky skrývá, aby uvolnilo prostor pro hlavní obsah a zlepšilo přehlednost uživatelského rozhraní. Jakmile uživatel přejde na větší rozlišení, například na tablet, navigační menu se opět zobrazí a umožňuje snadný přístup ke všem potřebným odkazům na podstránky a funkce aplikace. Toto dynamické chování menu zvyšuje uživatelskou přívětivost a efektivitu práce s aplikací, protože se přizpůsobuje potřebám a možnostem různých uživatelů.



Obrázek 38: Návrh responzivního navigačního menu

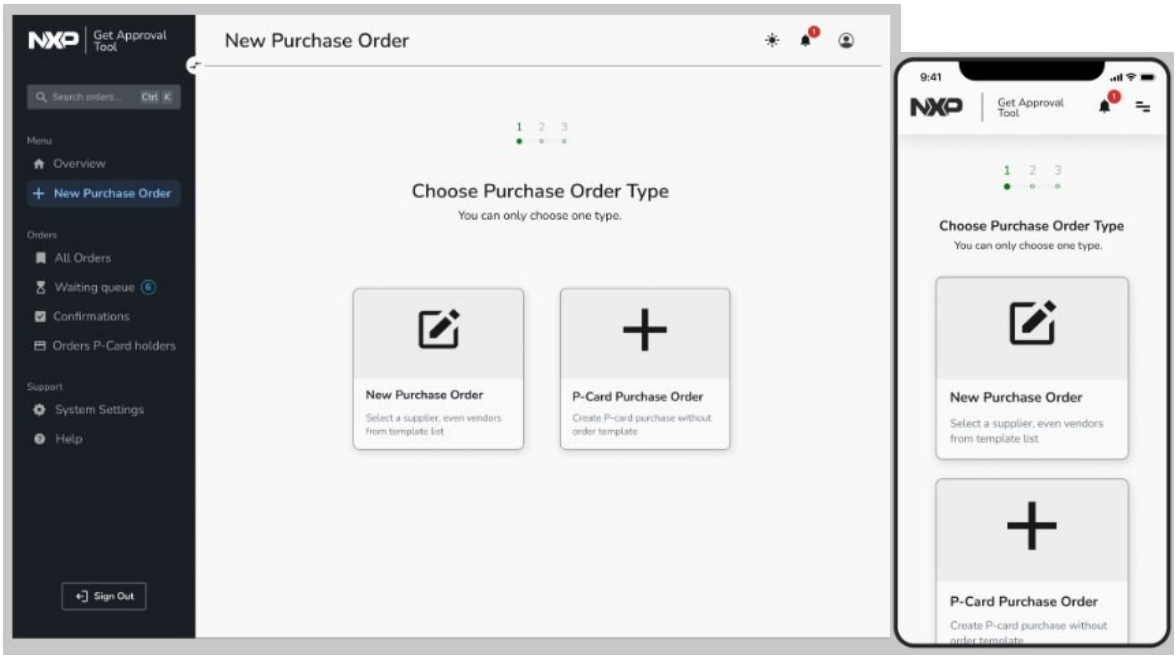
Jak již bylo avizováno dříve, jeden z nejdůležitějších uživatelských scénářů je vytvoření nového požadavku. Tato změna se jevila nejvíce komplikovanou, jelikož jde nejen o vytvoření vizuálně atraktivního návrhu, ale především o zjednodušení používání. Na základě předchozí analýzy původního stavu této části aplikace byl vytvořen nový návrh, kde bylo vytváření požadavku rozděleno do několika kroků. Na stránku se uživatel dostane přes boční navigační menu a odkaz *New Purchase Order*.

V každém kroku jsou komponenty, které informují uživatele o aktuálním stavu procesu. Jednotlivé kroky jsou indikovány řadou teček v horní části, kdy tečka, která je vyplněná nebo barevně zvýrazněná, označuje krok, ve kterém se uživatel momentálně nachází. Pro zaručení lepší orientace uživatele zde napomáhají textové pokyny určené pro popsání konkrétního kroku. Tyto prvky se mění podle zvolených možností a průběhu při vytváření požadavku.

Pro první krok byl vytvořen návrh, kdy si uživatel prvotně musí zvolit, jaký typ požadavku chce vytvářet (Obrázek 39). Každá z možností volby je vyobrazena ve velkých klikatelných kartách, které jsou snadno identifikovatelné a pro uživatele jednoduché. Karty obsahují

ikonu symbolizující možnost, titulek a krátký popis pro zaručení úplné srozumitelnosti. Celková struktura stránky je vytvořena tak, aby byla responzivní a uživatelsky přívětivá. Karty mají mezi sebou dostatek prostoru, takže nedojde k zahlcení rozhraní.

Pokud si uživatel zvolí první variantu *New Purchase Order*, tak má tato možnost dva další kroky, které jsou potřeba splnit pro vytvoření požadavku. Naopak druhá možnost *P-Card Purchase Order* se skládá pouze z jednoho dalšího kroku, který uživatele dělí od vytvoření požadavku. Podle volby požadavku se upravuje i počet indikátorů kroků, v případě první typu počet zůstane stejný, ale u druhého typu se počet změní ze tří na dva (Obrázek 41).

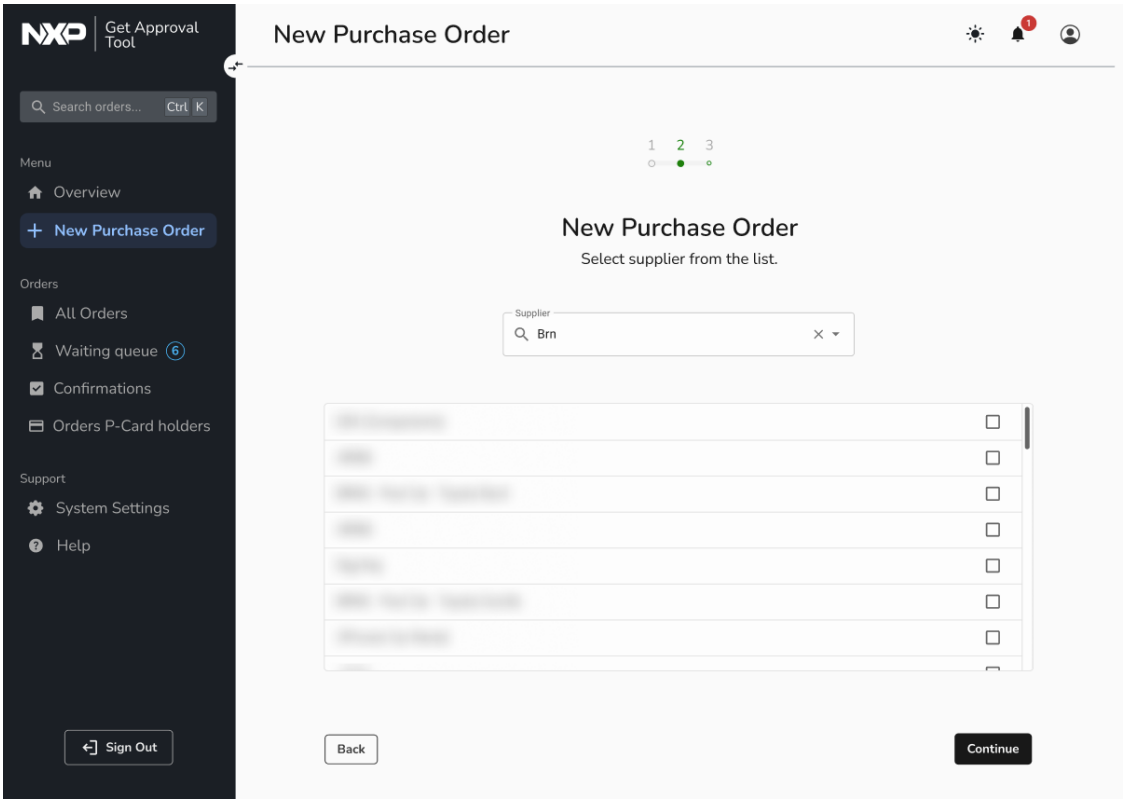


Obrázek 39: Návrh prvního kroku vytvoření požadavku

V případě volby požadavku typu *New Purchase Order* si v dalším kroku uživatel musí vybrat s jakým dodavatelem bude pokračovat do posledního kroku. Důvod oddělení tohoto selektoru do samostatného kroku je jednoduchý. Na základě volby dodavatele se v dalším kroku mění obsah stránky podle toho, které informace je nutné poskytnout k úspěšnému vytvoření požadavku.

Pro zaručení jednoduššího vybrání dodavatele ze seznamu byla vytvořena komponenta pro filtrování na základě textu. Nabízí se i možnost prohlédnout si všechny dodavatele v jednom obsažném seznamu, jelikož může dojít k situaci, kdy uživatel sám neví přesný název dodavatele, a tudíž filtrování podle názvu nemusí využít.

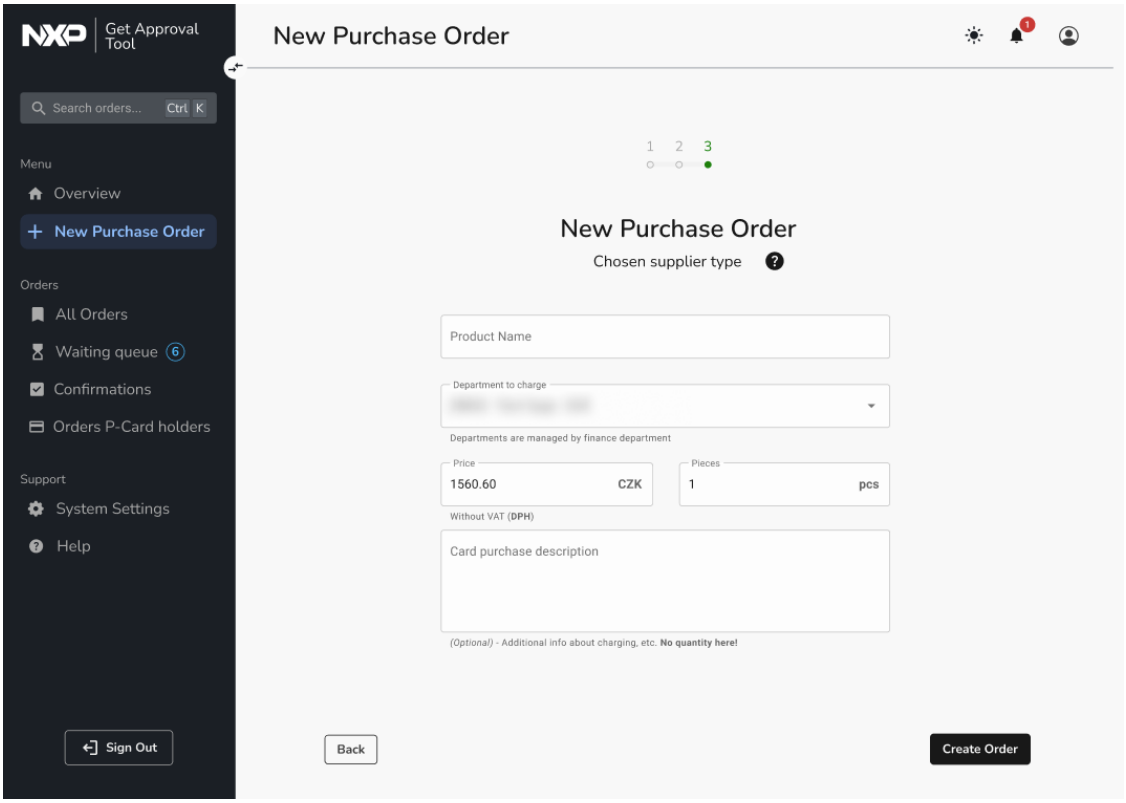
Pro zjednodušení navigace napříč jednotlivými kroky slouží ve spodní části dvě tlačítka, *Back* tlačítko pro vrácení k prvnímu kroku a *Continue* pro pokračování dál (Obrázek 40). Aby uživatel mohl pokračovat dál a tlačítko *Continue* mu bylo zprovozněno, je potřeba vybrat ze seznamu daného dodavatele. Bez toho bude tlačítko nedostupné, v tzv. zablokovaném stavu.



Obrázek 40: Návrh druhého kroku vytvoření požadavku

Jakmile uživatel úspěšné splní předchozí dva kroky, dostává se k poslednímu a tomu zásadnímu. V posledním kroku je potřeba na základě zvoleného typu dodavatele poskytnout všechny potřebné informace. Aby uživatel neztratil přehled o zvoleném dodavateli, je hned pod titulkem typu požadavku text, který jej o tom informuje. Pro poskytnutí všech potřebných informací je v návrhu vytvořen formulář, kde jsou všechny prvky potřeba vyplnit, kromě komponenty pro krátké objasnění požadavku, který má podtitulek *Optional*.

Podtitulkem jasně informuje uživatele, že tato část není potřeba vyplňovat, pokud se tak sám nerozhodne. Ostatní části jsou povinné a bez jejich vyplnění uživatel nebude moci vytvořit požadavek, jelikož tlačítko *Create Order* na spuštění této akce bude v zablokovaném stavu, dokud nesplní potřebné kroky.



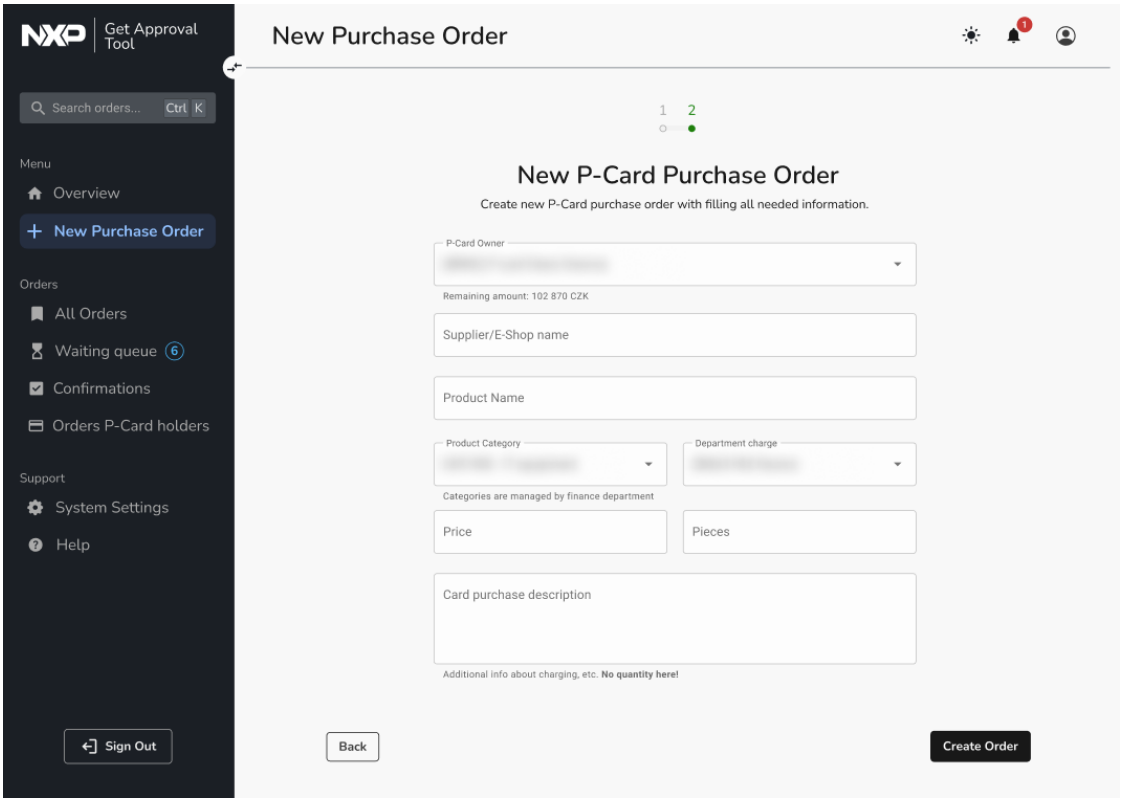
Obrázek 41: Návrh posledního kroku vytvoření požadavku

Pokud uživatel v prvním kroku zvolí druhou možnost požadavku typu *P-Card Purchase Order*, byl pro tuto stránku vytvořen návrh, který ukazuje řadu vylepšení UI (Obrázek 42). Nový návrh je primárně zaměřen na vylepšení efektivity vytváření požadavků a zlepšení uživatelské přívětivosti. Jelikož tato možnost nemá další krok, tak se indikátor správně změní ze tří kroků pouze na dva, protože po vyplnění všech potřebných informací dojde rovnou k vytvoření požadavku.

Jádro celé stránky je samotný formulář, který je oproti ostatním možnostem výrazně obsáhlejší, tudíž bylo nutné se zamyslet nad jeho srozumitelností. Formulář obsahuje jak textová pole např. pro zadávání názvu produktů, tak komponenty pro výběr ze seznamu. Všechny komponenty jsou navrženy tak, aby se přizpůsobily aktuálnímu prostoru kolem sebe, a tudíž byly responzivní na různých zařízení, což pozitivně podpoří uživatelskou mobilitu.

Většina informací je povinná pro vyplnění, ale ty, které nejsou, mají jasný podtitulek *Optional*, informující uživatele, že není potřeba tyto informace vyplňovat. Byl kladen důraz na jasné a srozumitelné popisy každého prvku tak, aby uživatelé přesně věděli, jaké informace mají vyplnit nebo vybrat. Tlačítko pro vytvoření požadavku, *Create Order*, je aktivní jen v případě, že uživatel vyplní veškerá povinná pole. Díky této kontrolní funkci je zabráněno

neúplnému nebo nesprávnému zadání údajů, a tím je zajištěna vyšší kvalita zasílaných údajů. Veškeré úpravy nového návrhu vedou ke zvýšení spokojenosti a mobilitě uživatelů, což je jeden z hlavních cílů implementace těchto změn.



Obrázek 42: Návrh vytvoření požadavku typu P-karta

Poslední část je zaměřena na návrh uživatelského rozhraní stránky *All Orders* podle přechozí analýzy nedostatků. Stránka prezentuje informace o všech vytvořených požadavcích přihlášeného uživatele, takže se dá říct, že se jedná o historii požadavků. Návrh je vytvořen tak, aby na uživatele působil pozitivně díky svému čistému vizuálnímu designu a barevnému schématu. Rozhraní je opět primárně zaměřeno na snadné používání, funkčnost a přístup k informacím.

Stránka je rozdělena do tří informačních částí. První sděluje uživateli zvolené období a nabídku funkcí, jako je třeba vytvoření reportu nebo odkaz na vytvoření nového požadavku. Druhá část prezentuje uživateli statistický přehled o svých požadavcích. Třetí část, a především jádro celé stránky, je přehledná tabulka, která zobrazuje potřebné informace o všech požadavcích (Obrázek 43). Logické uspořádání prvků na stránce pozitivně ovlivňuje uživatelskou zkušenost.

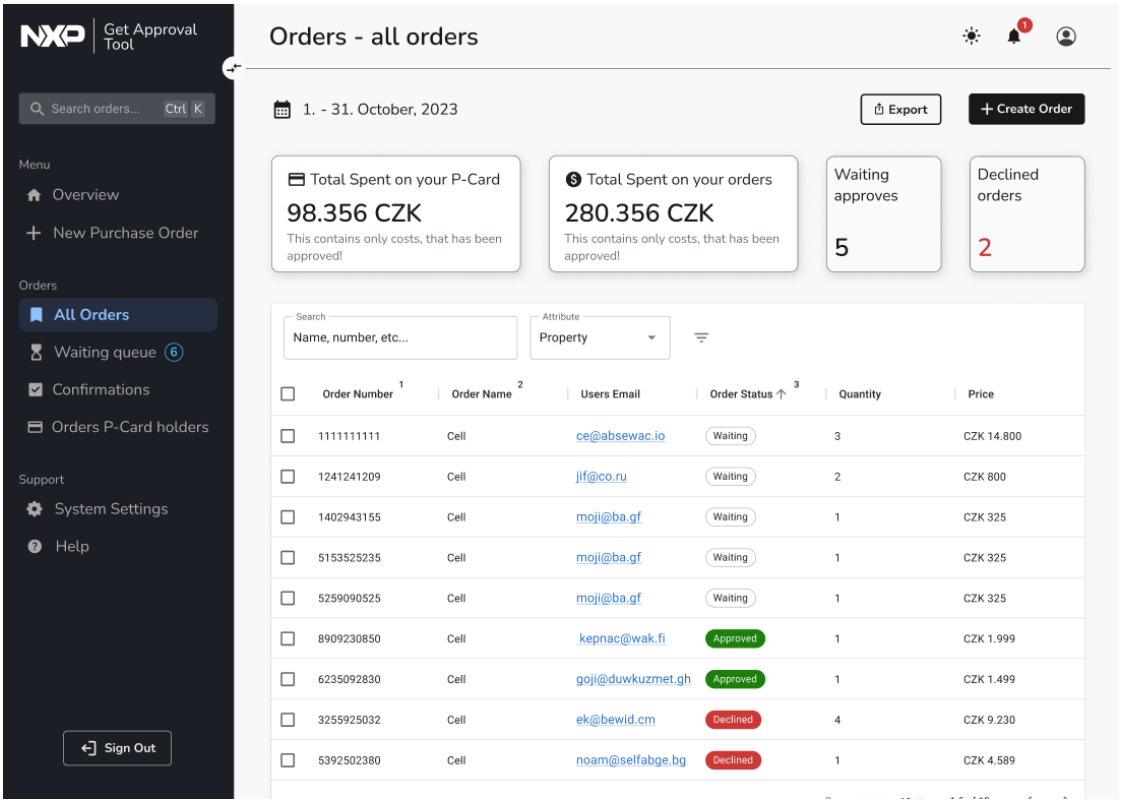
Pro vylepšení textové konzistence byl parametr informující uživatele o datu vložen do komponenty s ikonou prezentující kalendář a textem aktuálně zvoleného období. Aby byla vylepšena celková navigace napříč aplikací, byl do návrhu vložen odkaz v podobě tlačítka na vytvoření nového požadavku. Jelikož jde o stránku úzce související s požadavky, tak se tento odkaz na stránku hodí. Kombinace textového popisu prvků a ikon dává uživateli jasně najevo, co dané prvky znamenají. Tlačítko pro vytvoření reportu dostalo nový modernější a atraktivnější vzhled, a jelikož není tak důležité jako tlačítko *Create Order*, nemá barevnou výplň, pouze rámeček, kdy na uživatele působí jakožto sekundární funkce.

Další část prezentuje uživateli statistický přehled o požadavcích v podobně viditelných vizuálních kartiček, které se kterými se uživatel setkal již na stránce *Overview*. Na stránku byly kartičky vloženy z důvodu rychlého získání pozornosti uživatele, aby co v nejkratším čase mohl vyhodnotit svoje výdaje a počet požadavků v závislosti na aktuálním stavu. Kombinace textu, kontrastních kartiček a atraktivních ikon zlepšuje informační sdělení a celkovou zkušenost.

Hlavní komponenta a ústřední prvek celé stránky je tabulka obsahující seznam všech požadavků, zobrazující uživateli informace jako číslo požadavku, jeho produkty, celkovou cenu, a hlavně aktuální stav, ve kterém se požadavek nachází. Součástí tabulky je vyhledávací panel, který umožňuje filtrovat požadavky na základě zvolených atributů. Tato funkce je nezbytná pro zaručení dobré uživatelské zkušenosti, jelikož může být vyhledávání položek v potenciálně dlouhém seznamu obtížné. Vedle vyhledávací komponenty je umístěn filtr pro zpřesnění vyhledávaných atributů. Atributy mohou být například vyhledávání podle čísla požadavku, názvu produktů nebo i celkové ceny.

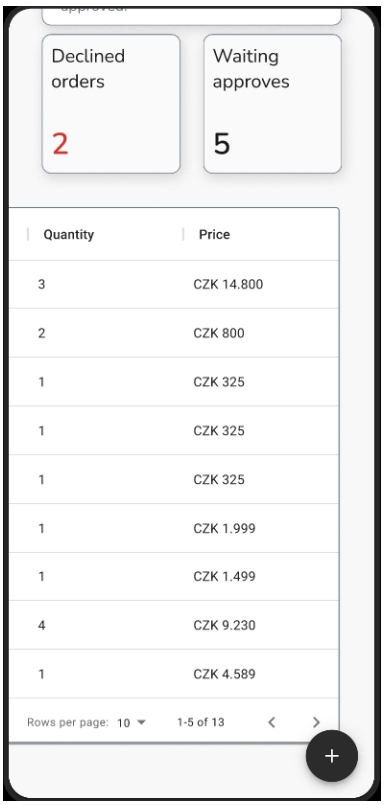
Jednotlivé záznamy v řádku jsou vizuálně odděleny, aby nedocházelo k typografickému slučování při čtení. Navíc sloupce popisující typ údaje nabízí možnost třízení sestupně nebo vzestupně, aby by bylo možné zobrazit sestupně ty požadavky se stavem, které čekají na schválení. Každý stav je prezentován komponentou odznaku a svou přiřazenou barvou pro lepší vizuální rozdělení. Barevné rozdělení stavů je jasným vizuálním vodítkem pro snadné pochopení stavu bez nutnosti číst text.

V momentě, kdy by seznam obsahoval velké množství požadavků, je zde funkce stránkování a volby, kolik řádků má tabulka obsahovat. Výchozí hodnota je zde nastavena na deset řádků. Jakmile by měl uživatel vytvořených více jak deset požadavků, nabízela by se mu funkce pro zobrazení další stránky v tabulce.



Obrázek 43: Návrh stránky zobrazující všechny požadavky

Pro zaručení responzivity má tabulka nastavenou minimální šířku tak, aby se veškeré prvky zobrazovaly správně a nedocházelo ke složitému čtení. Zobrazení na telefonu by pak dovolilo uživateli jednoduše horizontálně posunout a hledat údaje v tabulce, jak to lze vidět na Obrázku 44. Kromě tabulky se chová responzivně i zobrazení kartiček. Například na počítači jsou všechny karty seřazeny vedle sebe s dostatečnými mezerami, ale naopak na mobilním zařízení první dvě karty zabírají veškerý prostor. Další dvě, které mají poloviční velikost, mohou být pozicemi vedle sebe a budou mít i tak dostatečné množství prostoru.



Obrázek 44: Návrh stránky *All Orders* zobrazené na mobilním zařízení

8 IMPLEMENTACE WEBOVÉ APLIKACE

Primárním cílem praktické části bylo implementovat aplikaci pro správu požadavků ve společnosti NXP Semiconductors, a to právě díky technologiím React.js, Next,js, GraphQL a Apollo Client. Ještě před samotnou implementací proběhla analýza původního stavu aplikace, návrh nového uživatelského rozhraní a funkčnosti aplikace. Návrh nového uživatelského rozhraní a aplikace byly testovány při vývoji na různých zařízeních, aby se ověřilo, že webová aplikace bude fungovat správně. Pro ukázkové popsání implementace byly vybrány nejvýznamnější a zároveň nejzajímavější části aplikace. Vybrané části byly implementovány tak, aby vizuálně co nejpřesněji odpovídaly novému návrhu uživatelského rozhraní.

8.1 Nastavení projektu

Nastavení projektu je klíčovým faktorem implementace webové aplikace, je nezbytný pro úspěšný vývoj jak z technického, tak i z organizačního hlediska. Bez správného nastavení projektu není možné získávat aktuální data z databáze, komunikovat se serverem, poskytovat správné uživatelské rozhraní a zkrátka zajistit správné fungování celé aplikace.

Tento proces probíhá ještě před samotným programováním a patří do něj důkladná konfigurace a nastavení různých technologií, frameworků a nástrojů, jako jsou dříve zmíněné React.js, Next.js, TypeScript, TailwindCSS a další (viz kap. 1). Cílem této fáze je vytvořit stabilní základní strukturu, která poskytne vhodné prostředí pro vývoj a umožní systematické a koordinované programování.

Díky závislosti aplikace na technologii Node.js bylo možné instalovat externí balíčky a spouštět a testovat aplikaci na lokálním serveru. V první fázi byl projekt vytvořen pomocí nástroje *create-next-app*, který poskytl základní nastavení pro Next.js. V rámci nastavení bylo vybráno, že aplikace bude vytvořena bez *src/* struktury, bude využívat *app* routování, což je doporučený krok, mimo jiné taky technologie jako TypeScript, ESLint a TailwindCSS. Tato volba vytvořila základní konfiguraci a potřebné soubory pro spuštění projektu se zvolenými technologiemi.

Aby mohla aplikace používat konzistentní stylování komponent, spolu s TailwindCSS byla instalována i knihovna DaisyUI. Nejprve byl nastaven TailwindCSS dle kap. 1.6. a provedena konfigurace TailwindCSS do projektu postaveného na Next.js. Poté je potřeba vložit do konfiguračního souboru TailwindCSS je povinný modul s názvem *daisyui,* aby fungovala knihovna DaisyUI a mohly být využívány její vytvořené komponenty.

Nejdůležitější část konfigurace se týká získávání dat a komunikace se serverem. Od toho se odvíjí jak nastavení technologie Apollo Client, tak GraphQL spolu s Codegen pro generování spustitelných funkcí. Konfigurace není výrazně složitá, jelikož na oficiálních stránkách jak Apollo Client, tak GraphQL, je podrobný postup, jak zprovoznit v projektu tyto technologie.

Pro zprovoznění Apollo Client je potřeba nejprve nainstalovat balíček s názvem *@apollo/client* a zapouzdřit hlavní kořenovou komponentu do komponenty *ApolloProvider*, která poskytne aplikaci vytvořeného klienta. Pro nastavení klienta je nutné vytvořit instanci Apollo Client s použitím *httpLink*. Jelikož aplikace nabízí získávání dat pouze přihlášeným uživatelům je nutné získat z cookies uložený přihlašovací token uživatele a spolu s klíčovým slovem ho vložit do hlavičky. Výsledné vytvoření instance Apollo Client, Apollo Link a autorizace pro komunikaci se serverem lze vidět na Obrázku 45.

```
12  const $token : CookieValueTypes  = getCookie(TOKEN_STORAGE);
13
14  const auth : ApolloLink  = setContext( setter: (_ : GraphQLRequest<Record<string, … ) : {headers: {…}}  => {
15    return {
16      headers: {
17        Authorization:  + $token,
18      },
19    };
20  });
21
22  const httpLink : ApolloLink  = createHttpLink( linkOptions: {
23    uri: process.env.NEXT_LOCAL_GQL_URI,
24    fetch,
25  });
26
27  const apolloClient : ApolloClient<NormalizedCacheOb…  = new ApolloClient( options: {
28    link: auth.concat(httpLink),
29    cache: new InMemoryCache(),
30  });
```

Obrázek 45: Konfigurace Apollo Client

Komunikace s backendem probíhá pomocí nástroje GraphQL. Pro konfiguraci je potřeba nainstalovat balíček s názvem *graphql*, poté je možné vytvářet schémata, dotazovací soubory nebo soubory pro mutace. Při konfiguraci je nutné vytvořit i strukturu ukládání souborů. Soubory se budou ukládat do složky *graphql* v kořenovém adresáři, která je dále rozdělena do podsložek *queries* a *mutations*.

Za použití nástroje Codegen lze jednodušeji automatizovat vytvoření typů z definovaných GraphQL schémat. Po stáhnutí veškerých balíčků potřebných pro Codegen je nutné vytvořit konfigurační soubor *codegen.*yml, který specifikuje proces generování. V souboru je nutné

stanovit schéma, cestu pro dokumenty, pluginy, výslednou složku a soubor, do kterého se budou funkce vytvářet.

Pomocí nastavení skriptu *codegen generate* v souboru *package.json* lze automaticky generovat potřebné soubory, což zjednodušuje práci s daty a typy v projektu. Ukázka konfiguračního souboru *codegen.yml* je znázorněna na Obrázku 46.

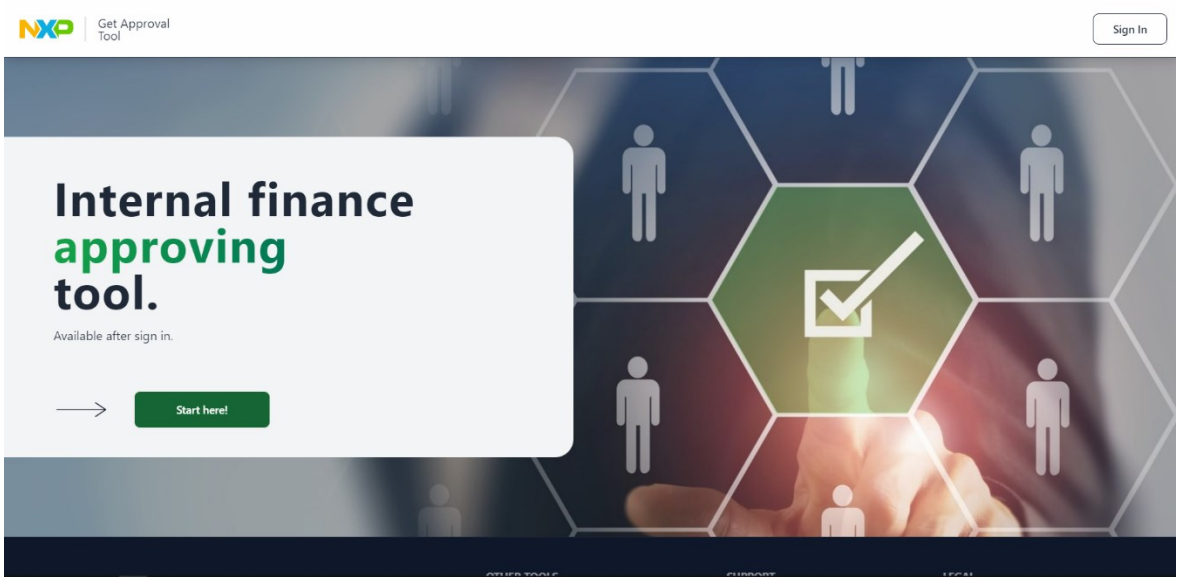
```yaml
overwrite: true
schema:
  - ${NEXT_PUBLIC_GQL_URI}
documents: "graphql/**/*.{ts,graphql}"
generates:
  get-approval-tool/api.tsx:
    plugins:
      - "typescript"
      - "typescript-operations"
      - "typescript-react-apollo"
      - "typescript-apollo-client-helpers"
    config:
      dedupeOperationSuffix: true
      enumsAsTypes: true
  ./graphql.schema.json:
    plugins:
      - "introspection"
```

Obrázek 46: Konfigurace nástroje Codegen

Celkovým nastavením jednotlivých technologií je připraven projekt, který je spustitelný na zařízení, testovatelný a funkční. Konfigurací je zajištěna správná funkcionalita pro vytváření nových záznamů do databáze, získávání dat, vytváření stránek za pomocí Next.js *app routování* a využití předdefinovaných komponent z knihovny DaisyUI, které mohou být dodatečně upraveny pomocí TailwindCSS.

8.2 Přihlášení

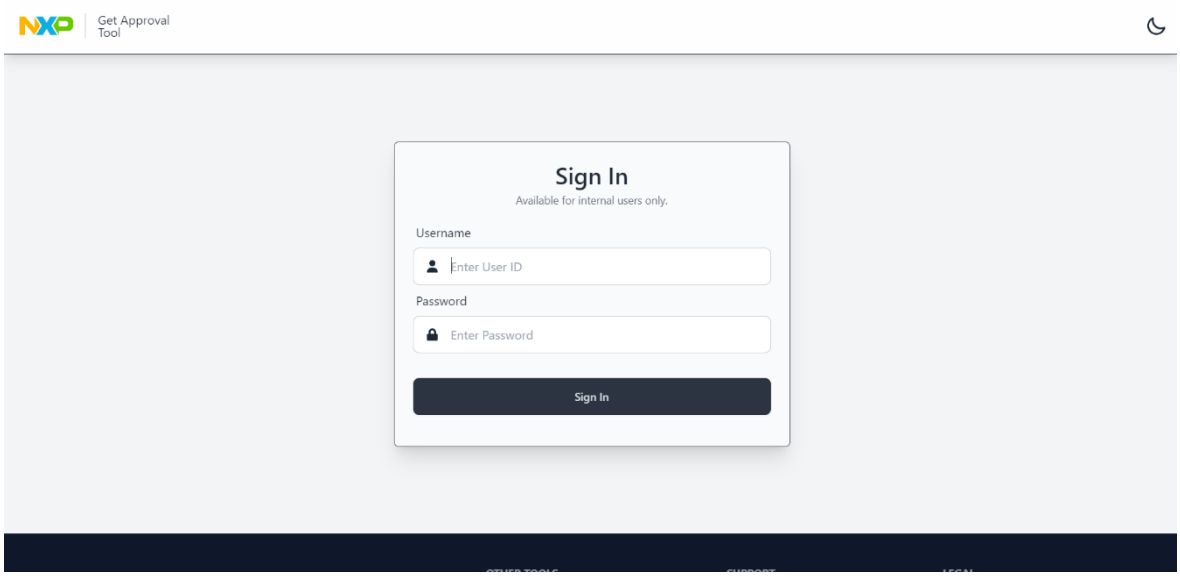
Aby mohli uživatelé pracovat s webovou aplikací, vytvářet nové požadavky a dále s nimi pracovat, je nutné každého uživatele autentizovat a identifikovat. Ještě před možností přihlášení se uživatel musí dostat na úvodní stránku aplikace (Obrázek 47). Zde jsou dvě tlačítka *Sign In* v navigační liště a *Start Here!* v těle stránky, kdy po kliknutí jednoho z nich bude uživatel přesměrován na přihlašovací stránku.



Obrázek 47: Ukázka úvodní stránky webové aplikace

Přihlašovací stránka od uživatele vyžaduje zadání potřebných informací, bez kterých se do aplikace nedostane; své firemní uživatelské identifikační jméno a heslo. Tyto dva vstupy uživatel zadává do formuláře, který je v aplikaci viditelný (Obrázek 48). V případě, že uživatel jeden ze dvou vstupů nevypíše, aplikace mu nedovolí odeslat přihlašovací dotaz na server.

Jakmile uživatel oba údaje zadá, na serveru je provedeno ověření, zda se uživatel s tímto uživatelským jménem v databázi již nenachází. Pokud se ukáže, že uživatel zadal validní údaje, je přesměrován na *Overview* stránku. V případě zadání špatných údajů mu aplikace zobrazí hlášku upozorňující, že buď uživatel s těmito iniciály neexistuje nebo zadal špatné heslo.



Obrázek 48: Ukázka přihlašovací stránky

Pro ověření a následné přihlášení interního uživatele slouží asynchronní funkce s názvem *LoginAuthentication*. Funkce byla vytvořena jako exportovatelná; v rámci přihlašovací stránky je tato metoda potom importována, aby se mohl vyvolat dotaz na server. Tato funkce přebírá jeden parametr v podobě objektu *userData*, který v sobě uchovává uživatelské jméno a heslo. Jde o požadavek typu *http*, metodou dotazu je *POST*.

V těle celého dotazu jsou pak zpracovávány zmíněné uživatelské údaje z objektu a, aby bylo zamezeno možným chybám, jsou údaje modifikovány tak, aby vždycky byly v podobě textového řetězce. Právě s tímto typem server očekává, že bude při validaci pracovat.

Jelikož se metoda dotazuje na server, očekává se ze strany serveru nějaká odpověď. Jakmile server obdrží uživatelské údaje, ověří, zda uživatel v databázi existuje nebo ne. V případě, že je dotaz úspěšný a uživatelské údaje byly validní, je uživateli navrácena hodnota přihlašovacího tokenu s datovým typem textového řetězce, který je později zpracováván. Pokud ale přihlašovací údaje nebyly správné, je navrácena odpověď v podobě chyby.

Veškerá kontrola výsledku funkce, zda je návratová hodnota opravdu přihlašovací token nebo naopak nějaká vyskytnutá chyba, probíhá v rámci přihlašovací stránky. Tento bod je přesněji popsán v kapitole zabezpečení webové aplikace (viz kap. 3). Funkce pro přihlášení a vyvolání dotazu na server vypadá následovně (Obrázek 49):

```
17  export default async function LoginAuthentication(userData: UserCredentials) : Promise<any>  {
18    return await fetch( input: `${process.env.NEXT_PUBLIC_API_URI}/login`, init: {
19      method: "POST",
20      mode: "cors",
21      headers: { "Content-Type": "application/json" },
22      body: JSON.stringify( value: {
23        user: userData.username,
24        pass: userData.password,
25      }),
26    }) Promise<Response>
27      .then((response : Response ) => response.json()) Promise<any>
28      .then((result) => {
29        return result;
30      })
31      .catch((error) => {
32        console.error(error);
33        return error;
34      });
35  }
36
```

Obrázek 49: Funkce pro získání přihlašovacího tokenu

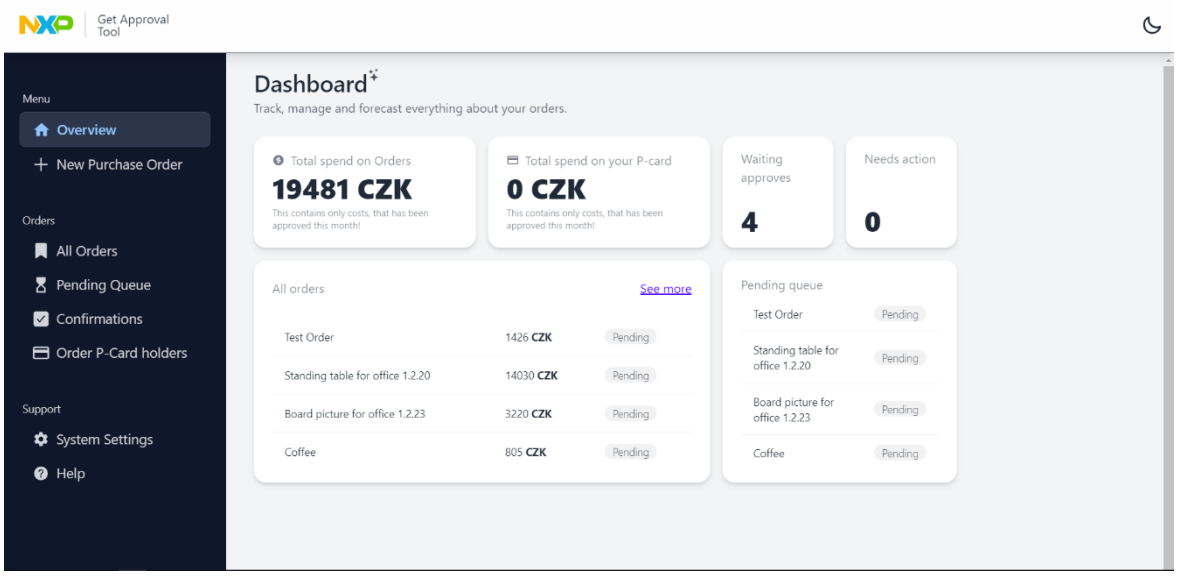
8.3 *Overview* stránka

Jak už bylo zmíněno v předchozím bodě, který popisoval přihlášení do aplikace, v momentě úspěšného přihlášení je uživatel automaticky přesměrován na centrální bod aplikace, *Overview* stránku (Obrázek 50). Stránka slouží jako přehled všech důležitých informacích o požadavcích, kde uživatel může vidět všechny své vytvořené požadavky, požadavky, které čekají na vyřízení a jednoduchý číselný přehled, aby okamžitě viděl aktuální útratu či počet požadavků čekajících na zprocesování.

Číselný přehled se skládá z komponent v podobě karet vytvořených na základě návrhu nového rozhraní. Karty zobrazují aktuální data o požadavcích, která jsou filtrována na základě stavu nebo typu požadavku. Ku příkladu první karta zobrazuje celkovou částku obecného typu požadavku. Naopak druhá karta prezentuje tu částku požadavků, které jsou typu *P-Card*. Ostatní, dvě menší karty, uchovávají celkový počet požadavků, které jsou filtrované na základě aktuálního stavu.

Pro vytvoření základu těchto vizuálních prvků byla použita předdefinovaná komponenta *Stat* z knihovny DaisyUI, která byla upravena pro vytvoření různých typů velikostí. Některý typ karty umožňuje vložit odkaz na případné přesměrování napříč aplikací, například karta zobrazující všechny vytvořené požadavky, která nabízí odkaz s textem *See More*, pomocí kterého je uživatel přesměrován na stránku *All* Orders. Tento přístup výrazně zlepšuje

uživatelskou zkušenost a jednoduchost používání aplikace. Stránka *All* Orders bude později v rámci popisu implementace také probrána.



Obrázek 50: Ukázka *Overview* stránky

Ještě před samotným vytvořením procesu pro získávání požadavků z databáze je klíčové vytvořit *GraphQL Query* s názvem *GetMyOrders* a vložit ji do souboru *Orders.graphql* do podsložky *queries* ve složce *graphql*. V tomto kroku je nutné specifikovat, jaké parametry v rámci požadavků jsou potřebné a relevantní. Pro stránku *Overview* je potřeba získávat víceméně veškerá dostupná data o požadavcích, primárně ta data, podle kterých je potřeba požadavky filtrovat a ta, která jsou potom vypsána do uživatelského rozhraní. Na Obrázku 51 je uvedený kód GraphQL dotazu *GetMyOrders*, který popisuje, jaké parametry byly specifikovány.

```
query GetMyOrders {
  getMyOrders {
    Id
    ProductName
    Status
    Type
    Created
    LastModified
    Description
    Price {
      WithVAT
      WithoutVAT
    }
  }
}
```

Obrázek 51: GraphQL dotaz pro získání požadavků

Pomocí příkazu *codegen generate* jsou vygenerovány potřebné části, díky kterým je možné spouštět dotazy pomocí Apollo Client a manipulovat s požadavky uživatele (Obrázek 52). Tímto způsobem je spouštění dotazů značně zjednodušeno a automatizováno.

```
export function useGetMyOrdersQuery(baseOptions?: Apollo.QueryHookOptions<GetMyOrdersQuery, GetMyOrdersQueryVariables>)
        const options :{}  = {...defaultOptions, ...baseOptions}
        return Apollo.useQuery<GetMyOrdersQuery, GetMyOrdersQueryVariables>(GetMyOrdersDocument, options);
      }
```

Obrázek 52: Ukázka vygenerované funkce pomocí nástroje Codegen

Jakmile přistoupí uživatel na stránku po přihlášení, automaticky je spuštěna vygenerovaná funkce s dotazem prostřednictvím Apollo Client, který se postará o jeho odeslání na server a správné zpracování. Apollo Client tímto způsobem komunikuje s backendovou části pomocí GraphQL. Po úspěšném dokončení dotazu a získáním dat, zachycené funkcí *onCompleted*, jsou požadavky přihlášeného uživatele uložena metodou *SetMyOrders* do proměnné s využitím React Hook *useState*.

Data jsou uchovávána ve formě datového pole typu *Order*, což umožňuje efektivní správu a manipulaci s požadavky napříč stránkou. V dané funkci je rovnou vypočítávána celková útrata, kde se požadavky filtrují na základě jejích typu (Obrázek 53). Stejným způsobem jsou získávána data i na stránce se všemi požadavky *All Orders*, která bude popsána v kap. 8.5.

```
18    const [myOrders : Order[] , setMyOrders : Dispatch<SetStateAction<Order[… ] = useState<Order[]>( initialState: []);
19    const [totalCostOrders : number , setTotalCostOrders : Dispatch<SetStateAction<number… ] = useState( initialState: 0);
20    const [totalCostPCard : number , setTotalCostPCard : Dispatch<SetStateAction<number… ] = useState( initialState: 0);
21
22    const { loading : boolean  } = useGetMyOrdersQuery( baseOptions: {
23      onCompleted: (res : GetMyOrdersQuery ) : void  => {
24        setMyOrders(res!.getMyOrders! as Order[]);
25
26        res!.getMyOrders!.map((item  : ) : void  => {
27          if (item!.Type === OrderType.ORDER) {
28            setTotalCostOrders(
29              value: (prevState : number ) =>
30                prevState + parseFloat(item!.Price!.WithVAT!.toString()),
31            );
32          }
33          if (item!.Type === OrderType.P_CARD) {
34            setTotalCostPCard(
35              value: (prevState : number ) =>
36                prevState + parseFloat(item!.Price!.WithVAT!.toString()),
37            );
38          }
39        });
40      },
41      client: apolloClient,
42      notifyOnNetworkStatusChange: true,
43    });
```

Obrázek 53: Funkce pro získání dat o požadavcích na stránce *Overview*

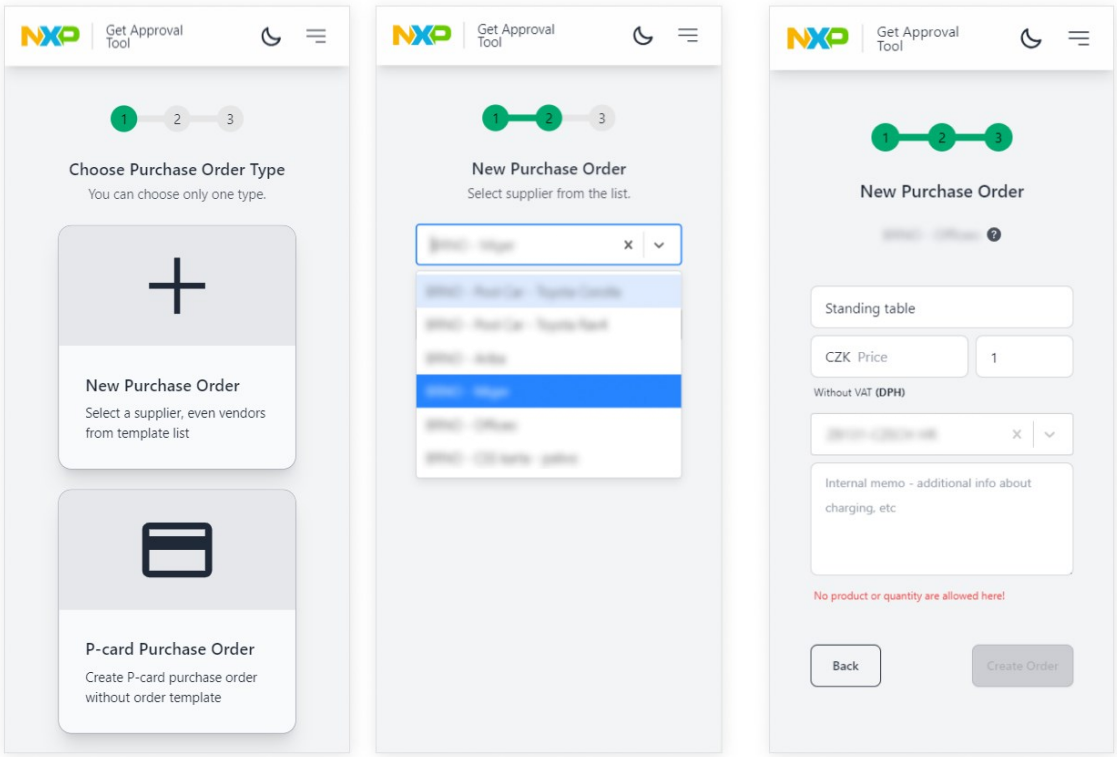
8.4 Vytvoření požadavku

V přechozí kapitole byla popsána implementace *Overview* stránky, ale především její získávání dat o požadavcích uživatele. Tato část se naopak zaměřuje na implementaci vytvoření požadavků, protože bez tohoto procesu by nebylo na ostatních stránkách co zobrazovat.

Podle návrhu nového rozhraní byla vytvořena stránka, která obsahuje dynamický formulář pro vytvoření požadavku. V prvním kroku jsou uživateli zobrazeny dvě možnosti v podobě velkých karet s popisem typu požadavku. Jednu z možností uživatel musí vybrat, jinak se nedostane na další krok.

V rámci ukázky implementace byla zvolena první možnost, a to vytvoření požadavku *New Purchase Order*. V druhém kroku si uživatel volí, s jakým typem dodavatele si chce požadavek vytvářet. Pro volbu slouží komponenta se seznamem, kde si uživatel může najít potřebného dodavatele i v seznamu vyhledávat podle názvu. Tlačítko pro přechod k dalšímu kroku zůstane zablokované do té doby, než si uživatel vybere dodavatele. Poté bude moci přejít k poslednímu kroku.

Ve třetím kroku je prvně uživateli pod titulkem zobrazen zvolený dodavatel, aby neztrácel přehled o vybraných možnostech. Poslední část opět zobrazuje formulář, ale ten už obsahuje veškeré potřebné prvky pro zadání informací a následné vytvoření požadavku. Jak je vidět v ukázce níže (Obrázek 54), stránka uživateli zobrazuje jak komponenty pro zadávání textu, tak pro výběr ze seznamu. Všechny prvky jsou povinné, tudíž je potřeba, aby je uživatel vyplnil, kromě posledního textového pole. V rámci ukázky na Obrázku 54 je vidět, že uživatel zadal skoro všechny informace kromě celkové ceny požadavku, takže tlačítko pro vytvoření požadavku je stále zablokované. Aktivní bude až tehdy, kdy uživatel doplní i celkovou cenu. Tato kontrola zabraňuje nevhodnému chování nebo vytvoření požadavků, které nemají vyplněné všechny potřebné informace.



Obrázek 54: Ukázka stránky pro vytvoření požadavku *New Purchase Order*

Požadavky lze vytvořit pomocí speciální GraphQL mutace s názvem *CreateOrder* (Obrázek 55), která umožňuje nejen data číst, ale hlavně vytvářet. Mutace přesně specifikuje, jaké parametry jsou potřeba odesílat s dotazem na server, aby se backendová část mohla postarat o jeho zpracování. Součástí mutace kromě parametrů je i odpověď v podobě konkrétních dat o vytvořeném požadavku. Jakmile je mutace vytvořena a vložena do souboru *Orders.graphql* do podsložky *mutations* ve složce *graphql*, vygeneruje se spustitelná funkce pomocí dříve zmíněného příkazu *codegen generate*.

```
mutation createOrder($ProductName: String, $Type: String, $Description: String, $Price: PriceInput, $Status: String) {
  CreateOrder(
    order: {ProductName: $ProductName, Type: $Type, Description: $Description, Price: $Price, Status: $Status}
  ) {
    Id
    ProductName
    Status
    Type
    Created
    LastModified
    Description
    Price {
      WithVAT
      WithoutVAT
    }
  }
}
```

Obrázek 55: GraphQL mutace pro vytvoření požadavku

Proces vytváření požadavku je zajištěn pomocí vygenerované funkce nástrojem Codegen React Hook *useCreateOrderMutation*, která poskytuje metodu spuštění a zachycení výsledků vytvořené mutace. Poté, co uživatel vyplní veškeré potřebné informace do UI, se mu aktivuje tlačítko *Create Order*, klikne na něj, a to vyvolá spuštění funkce *createOrderHandler*, která se stará o vytvoření požadavku (Obrázek 56). Tato funkce přijímá jako parametry všechny informace nutné pro vytvoření požadavku, jako je název produktu, typ, cenu nebo případně detailnější popis požadavku. Funkce má dvojité ošetření; i přes to, když by si uživatel ručně aktivoval v nástroji DevTools tlačítko, k pokusu o vytvoření požadavku bez potřebných informací nedojde. Probíhá zde jak kontrola parametrů přímo ve funkci, tak i na samotné stránce mají pro vytvoření požadavku textové vstupy atribut *required*.

Jakmile dojde k úspěšnému vytvoření požadavku, tak v části pro zachycení výsledku mutace dojde k přesměrování na stránku *Overview*. Přesměrování v této části zajišťuje React.js Hook *useRouter*.

Aby si byl uživatel jist, že opravdu došlo k vytvoření požadavku, zobrazí se mu překrývající komponenta sloužící jako zpětná vazba. V případě úspěšného dokončení mutace se mu zobrazí s textem na zeleném pozadí, že vytvoření požadavku bylo úspěšné, po několika sekundách zase zmizí. Naopak, pokud nastane nějaká chyba, tak se komponenta zobrazí na červeném pozadí s textem, že při vytváření požadavku nastala chyba.

```
23    const [createOrder] = useCreateOrderMutation();
24    const router : AppRouterInstance  = useRouter();
25
      1+ usages   ± jirizenzinger *
26    const createOrderHandler = (desc: string, price: PriceInput, productName: string, type: string, status: string) : null  => {
27      if (!productName || !price || !type || !status) return null;
28      createOrder( options: {
29        variables: {
30          Description: desc,
31          Price: price,
32          Type: type,
33          Status: status,
34          ProductName: productName,
35        },
36        onCompleted: () => router.push( href: "/overview"),
37      }) Promise<FetchResult<...>>
38        .then(() : void  => {
39          enqueueSnackbar("Order was successfully created.", {
40            variant: "success",
41          });
42        }) Promise<void>
43        .catch((err) : void  => {
44          enqueueSnackbar("Failed to finish creating order.", {
45            variant: "error",
46          });
47          console.error(err);
48        });
49    };
50
```

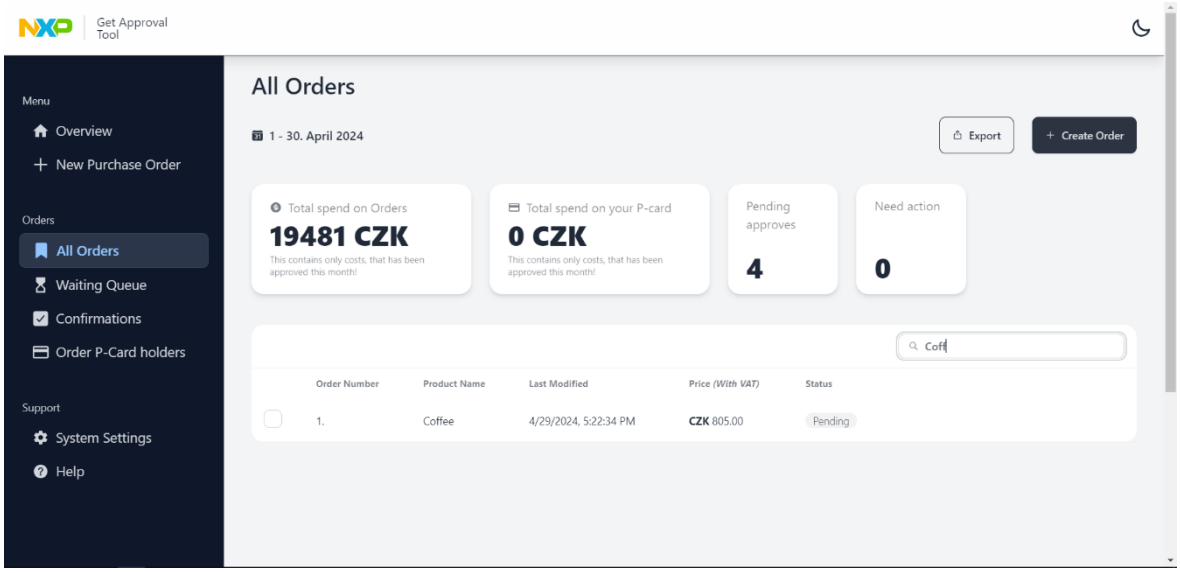
Obrázek 56: Funkce pro vytvoření požadavku

8.5 *All Orders* stránka

V situaci, kdy uživatel potřebuje najít v aplikaci konkrétní požadavek, využije stránku *All Orders* (Obrázek 57). Na stránku je možné se dostat přes stránku *Overview* nebo pomocí odkazu v navigačním menu na levé straně. Stránka slouží jako hlavní bod pro zobrazení veškerých vytvořených požadavků uživatele.

Jelikož byla stránka opět vytvořena na základě nového návrhu UI, primárně poskytuje celkový přehled o všech vytvořených požadavcích. Stránka obsahuje základní velké karty, které už byly viditelné na *Overview* stránce. Karty zobrazují totožné informace a slouží pro informační statistický přehled, který vylepšuje uživatelskou zkušenost, jelikož kromě vyhledávání konkrétních požadavků po celou dobu uživatel vidí i celkový přehled požadavků. Další část, která výrazně vylepšuje uživatelskou zkušenost, je tlačítko pro přesměrování na stránku pro vytvoření nového požadavku, což napomáhá i navigaci po aplikaci.

Ovšem nejdůležitější část této stránky je samotná tabulka. Tabulka je navržena tak, aby byla přehledná a zobrazovala nejdůležitější informace o daném požadavku. Pod nejdůležitější informace patří například název produktu, číslo pro přehled, název produktu, cena a aktuální stav požadavku. V momentě, kdy by uživatel měl v historii vytvořených mnoho požadavků, nabízí se zde hodně vhodná komponenta v podobě vyhledávače. Pomocí tohoto vyhledávače uživatel může jednodušeji najít konkrétní požadavek. To usnadňuje rychlost a jednoduchost nazelení hledaného požadavku.



Obrázek 57: Ukázka stránky *All Orders*

Když už byly zmíněny vizuální karty, je důležité vysvětlit i zjišťování dat, které tyto karty zobrazují. Konkrétní data o požadavcích jsou získávána stejným způsobem jako na *Overview* stránce, kde byl konkrétní popis funkce již probrán. Následný popis se naopak zaměří spíše na filtrování informací, které se zobrazí v kartičkách.

Každá karta zobrazuje jiné údaje v závislosti na základě typu. Zjišťování celkového počtu útraty podle typu požadavku bylo popsáno v kap. 8.3, kde bylo popisováno zjišťování a filtrování dat na základě typu požadavku ve zpracování výsledku dotazu. Jakmile *All Orders* získá totožná data, dojde ke stejnému zpracování výsledku, jako bylo popsáno ve zmíněné kapitole. Jediný rozdíl, který v kapitole popsán nebyl, byla problematika zaměřená na zjišťování aktuálního počtu požadavků, které čekají ke schválení nebo naopak potřebují nějakou aktivitu. V tomto případě je použita JavaScript funkce pro filtrování ze všech požadavků a dle aktuálního stavu buď vrátí data nebo ne. Jakmile dojde k dokončení tohoto filtrování, je zjištěn počet prvků v tomto poli a výsledné číslo je poté zobrazeno v kartě. Popsaná funkce pro filtrování a zobrazení počtu v kartách je uvedena na Obrázku 58.

```
<Stats
  title={"Pending approves"}
  type={"xs"}
  value={
    myOrders.filter((item : Order ) : boolean  => item.Status === OrderStatus.PENDING).length
  }
/>
<Stats
  title={"Need action"}
  type={"xs"}
  value={
    myOrders.filter((item : Order ) : boolean  => item.Status === OrderStatus.NEED_ACTION).length
  }
/>
```

Obrázek 58: Filtrování pro zobrazení počtu požadavků dle typu

8.6 Zabezpečení webové aplikace

Dříve, než bylo možné zahájit práci na zabezpečení webové aplikace, bylo potřeba učinit volbu, jakou metodu ověření uživatelů vybrat; jestli pomocí protokolu LDAP nebo SSO přístupu.

Použití protokolu LDAP oproti SSO přístupu je daleko výhodnější v rámci aplikací, které musí být přístupné pouze interním zaměstnancům. Protokol, na rozdíl od SSO, které se většinou propojuje s externími poskytovateli, je vytvořen speciálně pro správu interních

uživatelských identit. Administrátoři mohou regulovat práva uživatelů, členství ve skupinách a úrovně oprávnění napříč organizací.

Další výhoda využití protokolu LDAP je v celkové správě uživatelů. Protokol představuje centrální systém pro ověřování uživatelů, kromě nastavení práv umožňuje administrátorům blokovat uživatelské účty, zpřísňovat pravidla pro hesla nebo efektivně přidávat, upravovat a odebírat uživatele. Tímto způsobem je i jednodušší celková administrativa v porovnání se správou uživatelských účtů za použití externích poskytovatelů v SSO.

V rámci první části zabezpečení webové aplikace je nutno zajistit správnou a bezpečnou autentizaci uživatelů. Autentizace uživatelů probíhá na přihlašovací stránce, kde uživatel zadává své údaje, které je potřeba ověřit a na základě správnosti vydat přístupový token pro další operace v aplikaci. Tento proces je potřeba zabezpečit vůči nevhodným výsledkům, a proto je kontrola výsledků přihlášení ještě před povolením uživatele do aplikace klíčovou částí pro zaručení správné funkčnosti.

Kontrola probíhá v momentě, kdy přihlašovací funkce vrátí výsledek. Poté je ověřováno, jestli výsledek představuje přístupový token nebo nebylo přihlášení platné. Podle typu výsledku se uživateli buď zobrazí chybná hláška, že takový uživatel neexistuje nebo bylo zadáno špatné heslo. Pokud k takovému výsledku dojde, hodnota výsledku není uložena do cookies, a tudíž uživatel není připuštěn do aplikace a musí se přihlásit znovu. V momentě, kdy výsledek funkce vrátí daný přihlašovací token, tak ten je poté uložen do cookies a uživatel je přesměrován do aplikace. Když se chce uživatel odhlásit z aplikace, tak je přístupový token smazán z cookies. Tím pádem, jakmile se uživatel vrátí do aplikace, bude nucen se znova přihlásit. Popsaná kontrola výsledku funkce pro přihlášení je zobrazena na Obrázku 59.

```
24  const handleLoginSubmit = async (e: FormEvent<HTMLFormElement>) : Promise<null>  => {
25    setLoadingSignin( value: true);
26    setFormError( value: null);
27    e.preventDefault();
28    if (!username || !password) return null;
29
30    try {
31      const response = await LoginAuthentication( userData: {
32        username: username,
33        password: password,
34      });
35
36      if (!response || response.includes("ETIMEDOUT")) {
37        throw new Error( message: "Failed to submit login data. Please try again.");
38      }
39      if (response === LOGIN_ERRORS.invalidCred) {
40        throw new Error( message: "Invalid Credentials. Please try again.");
41      }
42      if (response === LOGIN_ERRORS.unknownUser || response.includes("REFUSED")
43      ) {
44        throw new Error( message: "Unknown user. Please enter valid username.");
45      }
46
47      setCookie(TOKEN_STORAGE, response.toString());
48      router.push( href: "/overview");
49    } catch (error) {
50      console.error(error);
51      setFormError((error as Error).message);
52    } finally {
53      setLoadingSignin( value: false);
54    }
55  };
```

Obrázek 59: Funkce pro kontrolu výsledku přihlašování

Zmíněný přístupový token je používán v aplikaci jak pro přístup k datům, tak pro vytváření nových požadavků. Aby mohl Apollo Client správně získávat data z databáze, server očekává, že v rámci hlavičky bude součástí spolu s klíčovým slovem i přístupový token. Bez tokenu pak není možné získávat data o požadavcích, jelikož takové dotazy server nepovolí a ve výsledku se uživatel ani nedostane do aplikace kvůli dalšímu zabezpečení kontrolujícímu přístup.

V aplikacích, které mohou být dostupné jen pro určité uživatele, je potřeba zajistit přísnou kontrolu přístupu do aplikace. Jelikož je interní aplikace postavena na React.js a Next.js, nabízí se zde použití funkce Next.js Middleware, která umožňuje spouštět kód ještě před dokončením příchozího požadavku (Obrázek 60). Typické použití tohoto mechanismu je v rámci ověřování a autorizace uživatelů. Často zde patří i kontrola identity uživatelů nebo souborů cookies ještě před udělením přístupu na konkrétní stránky nebo API.

V interní aplikaci je potřeba kontrolovat přístup, a to přes ověření, že uživatel obdržel přihlašovací token po úspěšném přihlášení a byl uložen do cookies. Proces ověřování funguje

následovně: jakmile se uživatel přesměrovává například ze stránky *Overview* na vytvoření požadavku, tak ještě před dokončením daného přesměrování ve funkci Middleware dojde ke kontrole, zda je v cookies pod zvoleným klíčem uložen přihlašovací token. Pokud je token nalezen, uživateli je umožněno přesměrování. V případě, že naopak token nalezen není a uživatel se zrovna nenachází na stránce pro přihlášení, Middleware ho přesměruje zpět na přihlašovací stránku. Kontrolou, že zrovna není na přihlašovací stránce, se zabrání nechtěným opakovaným přesměrováním.

Ve funkci je i definovaná stránka, na které ověření přístupu neprobíhá, a to na stránce úvodní. Na úvodní stránku a stránku pro přihlášení se uživatel dostane i bez přihlašovacího tokenu.

Aby mohl Middleware fungovat, je potřeba vytvořit soubor *middleware.ts*, který musí být uložen v kořenovém adresáři na stejné úrovni jako hlavní složka *app*. Níže uvedený kód ukazuje zmíněnou kontrolu cookies a možné přesměrování v případě nenalezení přihlašovacího tokenu.

```
9   export function middleware(request: NextRequest) : NextResponse<?> {
10    const res : NextResponse<?> = NextResponse.next();
11    const cookiesTokenExist : boolean = hasCookie(TOKEN_STORAGE, options: { res, req: request });
12
13    // Redirect to login page if not authenticated
14    if (!cookiesTokenExist && !request.url.includes("/login")) {
15      return NextResponse.redirect(new URL( url: "/login", request.url));
16    }
17
18    // If the user is authenticated and has token, continue as normal
19    return NextResponse.next();
20  }
21
      no usages  jirizenzinger
22  export const config : {matcher: string[]} = {
23    matcher: ["/((?!api|_next/static|favicon.ico).*.{1,})"],
24  };
```

Obrázek 60: Funkce NextJS Middleware

9 OTESTOVÁNÍ APLIKACE

Každá vytvořená část aplikace byla testována samostatně, aby byla zjištěna její dokonalá funkčnost. Testování probíhalo jak během vývoje aplikace, tak i po jeho dokončení na reálných zařízeních. Cílem otestování webové aplikace bylo identifikovat možné nedostatky a primárně ověřit celkovou provozuschopnost jak uživatelského rozhraní, tak veškerých funkcí.

Klíčové faktory testování zahrnují ověření kompatibility aplikace na různých platformách a velikostech zařízení, zabezpečení dat a celkovou stabilitu aplikace. Část, která nesměla být v testování opomenuta, je ověření správné funkčnosti integrovaných služeb.

9.1 Testování aplikace během vývoje

Webová aplikace byla v průběhu vývoje testována pomocí nástroje ESLint, který umožnil kontrolovat a identifikovat potencionální chyby v kódu. Tímto způsobem ESLint dokázal detekovat a opravovat nejen syntaktické a logické chyby v kódu, ale zároveň kontrolovat správnost typů při použití TypeScriptu.

Tento typ testování podpořilo vývojové prostředí PyCharm od společnosti JetBrains, které umožnilo rychle odhalit a opravit chyby, což zlepšilo celkovou kvalitu kódu.

Během vývoje bylo testováno, jakým způsobem komunikují integrované služby. To znamená, že veškerá data, která jsou získávána z databáze nebo naopak zapisována, jsou korektní. Veškerá ověření z databáze bylo možné sledovat ve webovém rozhraní GraphQL a samozřejmě konkrétní výsledky volání pro získávání dat ve webovém prostředí prohlížeče DevTools.

Prostředím DevTools bylo testováno i celkové chování a responzivita webové aplikace. V DevTools lze ověřit responzivitu napříč různými rozlišeními, kdy prostředí dovoluje vybrat si jak určitou velikost zařízení, tak si vlastními silami přizpůsobit výslednou velikost rozlišení.

Tímto způsobem bylo jednoznačné, jestli se webová aplikace chová adaptivně vůči velikosti rozlišení či nikoliv. Jelikož bylo vše testováno v průběhu vývoje, tyto chyby bylo možné velice rychle identifikovat a napravit.

9.2 Testování uživatelského rozhraní

Testování uživatelského rozhraní tvořilo významnou součást v celém procesu vývoje webové aplikace, a to jak v průběhu návrhu nového rozhraní, tak i po dokončení implementace jednotlivých částí před tím, než bude aplikace spuštěna pro koncové uživatele. Testy byly nezbytné pro ověření nejen technické stránky návrhu aplikace, ale i skrz zajištění, že výsledný produkt bude intuitivní, vizuálně atraktivní, a především jednoduše použitelný pro všechny zaměstnance společnosti. Správná funkčnost je nezbytná pro zaručení hladkého každodenního používání a celkové spokojenosti uživatelů.

Na začátku vývojové fáze, při tvorbě nového návrhu UI, byl design pravidelně modifikován díky zpětné vazbě od zaměstnanců, kteří s původní verzí aplikace denně pracují. Tato spolupráce umožnila lépe identifikovat potřeby přímo od koncových uživatelů a přizpůsobit nového rozhraní tak, aby co nejvíce odpovídalo jednoduchému používání v reálných situacích. Tímto způsobem byl například upraven nový návrh stránky s vytvářením nového požadavku, kdy původní návrh nepůsobil jednoduše a uživatele často mátl.

Další fáze testování byla uskutečněna po dokončení implementace, ve které se testy zaměřily na ověření celkové funkčnosti integrovaných funkcí. Veškeré testy byly řízeny možným scénářem, který reprezentoval očekávané chování koncového uživatele ve webové aplikaci.

Ověřování pak probíhalo krok po kroku od přihlášení zaměstnance do aplikace, správné přesměrování, vytvoření požadavků, po kontroly všech vytvořených požadavků či jiná manipulace s nimi. Bylo testováno i neočekávané chování, například pokus o přístup do aplikace bez autentizace.

Testování bylo prováděno ve webových prohlížečích *Google Chrome*, *Brave*, *Opera* i *Microsoft Edge*. V daných prohlížečích byla aplikace testována z pohledu běžného uživatele a byl primárně kladen důraz na její výslednou responzivitu, uživatelskou přívětivost a celkovou funkčnost.

ZÁVĚR

Cílem diplomové práce bylo analyzovat dosavadní stav interní webové aplikace *Get approval tool*, sloužící k administraci a schvalování finančních prostředků ve firmě NXP Semiconductors. Na základě analýzy, a z ní vyplývajících nedostatků, byl vytvořen nový návrh odpovídající aktuálním trendům ve webovém designu.

Při vytváření návrhu byla využita JavaScript knihovna React.js kvůli své jednoduchosti a dělení na znovupoužitelné komponenty, frameworková nadstavba React.js, Next.js, kvůli app routování a škálovatelnosti a stylovací nástroj TailwindCSS spolu s knihovnou DaisyUI. Komunikace a validace na straně serveru, vytváření požadavků a operace s daty byly zprostředkovány za pomoci dotazovacího jazyku GraphQL a knihovny Apollo Client.

Dle návrhu byla vytvořena nová webová aplikace, která byla testována jak během vývoje, tak i po vytvoření. Během vývoje byla kontrolována správná komunikace webové aplikace se serverem, zabezpečení, vytváření požadavků, a jejich následné zobrazování včetně jejich zapisování do databáze. Responzivita byla testována jak v průběhu vývoje, tak po dokončení. Po dokončení byla testována celková funkčnost webové aplikace obsahující neočekávané chování uživatele.

Výsledkem této diplomové práce je nová webová aplikace, zvyšující efektivitu zaměstnanců při správě požadavků týkajících se finančních prostředků. Webová aplikace se stala intuitivnější, dodržuje nejnovější trendy ve webdesignu a zároveň umožňuje práci na jakémkoliv zařízení. Zaměstnanec se takto stává plně mobilním bez nutnosti být odkázán pouze na počítač.

Do budoucna je nutné zaměřit se na možnost nahrávání fyzických faktur k již vytvořenému požadavku, notifikování i zabývat se možností vytvoření progresivní webové aplikace.

SEZNAM POUŽITÉ LITERATURY

[1] What Is TypeScript? A Comprehensive Guide. Online. Kinsta. Dostupné z: https://kinsta.com/knowledgebase/what-is-typescript/

[2] TypeScript for the New Programmer. Online. TypeScript. Dostupné z: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

[3] TypeScript Introduction. Online. W3Schools. Dostupné z: https://www.w3schools.com/typescript/typescript_intro.php

[4] TypeScript for JavaScript Programmers. Online. TypeScript. Dostupné z: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.

[5] React. Online. Dostupné z: https://react.dev/

[6] TYAGI, Ankur. Learn React – A Guide to the Key Concepts. Online. FreeCodeCamp. 2024. Dostupné z: https://www.freecodecamp.org/news/learn-react-key-concepts/#key-concepts-to-understand-in-react

[7] React Introduction. Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/reactjs-introduction/

[8] Why You Should Use React.js For Web Development. Online. FreeCodeCamp. Dostupné z: https://www.freecodecamp.org/news/why-use-react-for-web-development/

[9] Virtual DOM and Internals. Online. React. Dostupné z: https://legacy.reactjs.org/docs/faq-internals.html

[10] ROLDÁN, Carlos Santana. *React 18 Design Patterns and Best Practices*. Fourth Edition. Packt Publishing, 2023. ISBN 1803233109.

[11] Writing Markup with JSX. Online. React. Dostupné z: https://react.dev/learn/writing-markup-with-jsx

[12] Thinking in React. Online. React. Dostupné z: https://react.dev/learn/thinking-in-react

[13] STEFANOV, Stoyan. *React: Up & Running: Building Web Applications*. O'Reilly Media, 2021. ISBN 1492051462

[14] React Hooks. Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/reactjs-hooks/

[15] Learn React Hooks – A Beginner's Guide. Online. FreeCodeCamp. Dostupné z: https://www.freecodecamp.org/news/the-beginners-guide-to-react-hooks/

[16] Built-in React Hooks. Online. React. Dostupné z: https://react.dev/reference/react/hooks

[17] Next.js. Online. Dostupné z: https://nextjs.org/docs

[18] Next VS React. Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/nextjs-vs-reactjs-which-one-to-choose/

[19] Introduction: Welcome to the Next.js documentation! Online. Next.js. Dostupné z: https://nextjs.org/docs

[20] About React and Next.js. Online. Next.js. Dostupné z: https://nextjs.org/learn/react-foundations/what-is-react-and-nextjs

[21] Introduction to GraphQL. Online. GraphQL. Dostupné z: https://graphql.org/learn/

[22] What is GraphQL. Online. GraphQL. Dostupné z: https://graphql.com/learn/what-is-graphql/

[23] Webdesigner Depot. Online. Dostupné z: https://www.webdesignerdepot.com/

[24] NN/G NIELSEN NORMAN GROUP. Online. Dostupné z: https://www.nngroup.com/

[25] MICHÁLEK, Martin. *Vzhůru do (responzivního) webdesignu*. Verze 1.1. Praha: vlastním nákladem autora, 2021. ISBN 978-808-8253-006.

[26] Introducing Types. Online. GraphQL. Dostupné z: https://graphql.com/learn/introducing-types/#introducing-scalar-types

[27] GraphQL. Online. TechTarget. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/GraphQL

[28] Why Apollo Client. Online. Apollo GraphQL. Dostupné z: https://www.apollographql.com/docs/react/why-apollo/#zero-config-caching

[29] Introduction to Apollo Client. Online. Apollo GraphQL. Dostupné z: https://www.apollographql.com/docs/react/

[30] Install Tailwind CSS with Next.js. Online. Tailwind Labs. Dostupné z: https://tailwindcss.com/docs/guides/nextjs

[31] Get started with Tailwind CSS. Online. Tailwind Labs. Dostupné z: https://tailwindcss.com/docs/installation/using-postcss

[32] What is daisyUI. Online. DaisyUI. Dostupné z: https://daisyui.com/blog/what-is-daisyui/

[33] DaisyUI. Online. DaisyUI. Dostupné z: https://daisyui.com/

[34] PUROHIT, Rakesh. How to Use DaisyUI to Create Stunning User Interfaces. Online. DhiWise. Dostupné z: https://www.dhiwise.com/post/how-to-use-daisyui-to-create-stunning-user-interfaces

[35] What is web design? A comprehensive guide. Online. Wix.com. Dostupné z: https://www.wix.com/blog/web-design

[36] Web Design. Online. Tiller Digital. Dostupné z: https://tillerdigital.com/glossary/web-design/

[37] What is web design (and how do I get it right). Online. 99designs. Dostupné z: https://99designs.com/blog/web-digital/what-is-web-design/

[38] What is the difference between UI and UX? Online. Figma. Dostupné z: https://www.figma.com/resource-library/difference-between-ui-and-ux/

[39] A peek into the UI/UX universe (with examples). Online. Specbee. Dostupné z: https://www.specbee.com/blogs/peek-ui-ux-universe-examples

[40] The Difference between UX & UI Design: 9 Things to Know. Online. Net Solutions. Dostupné z: https://www.netsolutions.com/insights/ux-vs-ui-design/

[41] Responsive Web Design: What It Is And How To Use It. Online. Smashing. Dostupné z: https://www.smashingmagazine.com/2011/01/guidelines-for-responsive-web-design/

[42] What Is Responsive Web Design? And How to Get Started. Online. Coursera. Dostupné z: https://www.coursera.org/articles/responsive-web-design

[43] Responsive design. Online. Mdn. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design

[44] Figma. Online. Figma. Dostupné z: https://www.figma.com/

[45] What is web application security? Online. Cloudflare. Dostupné z: https://www.cloudflare.com/learning/security/what-is-web-application-security/

[46] Web Application Security. Synopsys. Dostupné z: https://www.synopsys.com/glossary/what-is-web-application-security.html

[47] Authentication vs. Authorization. Online. OneLogin. Dostupné z: https://www.onelogin.com/learn/authentication-vs-authorization

[48] Difference between Authentication and Authorization. Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/difference-between-authentication-and-authorization/

[49] What is lightweight directory access protocol (LDAP) authentication? Online RedHat. Dostupné z: https://www.redhat.com/en/topics/security/what-is-ldap-authentication

[50] What is LDAP? Online. OneLogin. Dostupné z: https://www.onelogin.com/learn/what-is-ldap

[51] How Does Single Sign-On Work? Online. OneLogin. Dostupné z: https://www.onelogin.com/learn/how-single-sign-on-works

[52] What is SSO. Online. Cloudflare. Dostupné z: https://www.onelogin.com/learn/how-single-sign-on-works

[53] 5 Best practices for testing web applications. Online. Global App Testing. Dostupné z: https://www.globalapptesting.com/blog/best-practices-for-testing-web-applications

[54] Web Based Testing – Software Testing. Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/software-testing-web-based-testing/

[55] Web application testing: 6-step guide how to test a website. Online. UsersNap. Dostupné z: https://usersnap.com/blog/web-application-testing/

[56] About NXP. Online. NXP Semiconductors. Dostupné z: https://www.nxp.com/company/about-nxp:ABOUT-NXP

[57] NXP History. Online. NXP Semiconductors. Dostupné z: https://www.nxp.com/company/about-nxp/history:NXP-HISTORY

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
DOM	Document Object Model
GAT	Get Approval Tool
JSX	JavaScript XML
LDAP	Lightweight Directory Access Protocol
RWD	Responzivní webový design
SEO	Search engine optimization
SPA	Single Page Application
SSO	Single Sing-On
SQL	Structured Query Language
UI	User Interface
UX	User eXperience

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

Struktura obsahu přiloženého CD:

- Soubor **fulltext.pdf** – obsahuje text diplomové práce ve formátu PDF