Tvorba webových aplikací pomocí Frameworku Next.js

Oliver Ludvík

Bakalářská 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 BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Oliver Ludvík**
Osobní číslo:	**A21269**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Tvorba webových aplikací pomocí frameworku Next.js**
Téma práce anglicky:	**Creating Web Applications Using the Next.js Framework**

Zásady pro vypracování

1. Popište základní stavební prvky, strukturu a principy tvorby webových aplikací pomocí frameworku Next.js.
2. Zdokumentujte postup nastavení prostředí pro vývoj v daném frameworku.
3. Stanovte funkcionální a nefunkcionální požadavky pro demonstrační aplikaci tak, aby byly vhodně pokryty příklady důležitých prvků a postupů, které vývojový framework nabízí.
4. Implementujte demonstrační aplikaci dle stanovených požadavků.
5. Shrňte důležité kroky procesu implementace a na jeho základě vytvořte porovnání s dalšími populárními frameworky.

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

Seznam doporučené literatury:

1. The React Framework for the Web [online]. [cit. 2023-11-10]. Dostupné z: https://nextjs.org
2. React The library for web and native user interfaces [online]. [cit. 2023-11-10]. Dostupné z: https://react.dev
3. PocketBase [online]. [cit. 2023-11-10]. Dostupné z: https://pocketbase.io
4. Tailwind CSS [online]. [cit. 2023-11-10]. Dostupné z: https://tailwindcss.com
5. TypeScript [online]. [cit. 2023-11-10]. Dostupné z: https://www.typescriptlang.org
6. VYSTAVĚL, Radek. Myslete databázově, myslete v SQL!. Ondřejov: Radek Vystavěl, 2023. ISBN 978-80-908144-1-7.
7. RICHARDSON, Leonard a AMUNDSEN, Michael. RESTful Web APIs. Sebastopol: O'Reilly, 2013. ISBN 9781449358068.

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské 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 bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

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

Ve Zlíně, dne

Oliver Ludvík, v. r.
podpis studenta

ABSTRAKT

Bakalářská práce popisuje vývoj webové aplikace pomocí frameworku Next.js. Práce ukazuje, jak framework řeší klasické paradigma webového vývoje. Například abstrakci databáze, získávání dat z veřejných API, autorizaci a autentizaci uživatele a směřování požadavků. Dále práce popisuje nasazení aplikace na internet pomocí nástrojů nabízených společností Vercel. Na konci se práce zabývá porovnáním frameworku Next.js s ostatními fullstack frameworky, jako jsou Django, Ruby on Rails či ASP.NET.

Klíčová slova: Next, Next.js, webová aplikace, vývoj webové aplikace, Vercel

ABSTRACT

This bachelor thesis describes development of a web application using the framework Next.js. Theses shows us, how the framework deals with classical paradigms of web development, namely database abstraction, data fetching from public APIs, user authorization and authentication and routing. Thesis also gives details of application deployment to the world wide web via tools offered by the company Vercel. At the end, the thesis looks at other available frameworks like Django, Ruby on Rails or ASP.NET and compares them against Next.js.

Keywords: Next, Next.js, web application, web application development, Vercel

Tímto bych rád poděkoval Ing. Radku Valovi Ph.D. za možnost zpracovat práci pod jeho vedením a za cenné rady při tvorbě práce.

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

OBSAH

ÚVOD

Webové stránky se posunuly od dob statických stránek, a proto je dneska potřeba použít dostatečně silné nástroje na vývoj kompetentních stránek. První krok při tvorbě webové aplikace by kvůli tomu měl být právě výběr toho správného nástroje, v našem případě webového frameworku.

Tato práce rozebírá moderní pojetí webových aplikací na ty nejzákladnější komponenty. Každá komponenta následně popisuje, ukazuje, kde se vyskytuje, a pokud možno, nabízí alternativy. O tyto data se práce následně opírá v dalších kapitolách.

Hlavní cíl práce je přestavení frameworku Next.js. Práce ukazuje, jaké technologie framework využívá a jak je používá. Mezi důležité složky frameworku patří například knihovna React, pro tvorbu uživatelského rozhraní. Framework má také unikátní přístup k problematikám routování uživatelských požadavků a nebo rozdělení zátěže mezi klienta a serveru. Všechny specifikace frameworku jsou v práci popsány a vysvětleny.

Konec teoretické části popisuje přípravu aplikace pomocí generátorů. Je zde vysvětleno, co generátor nastaví a vytvoří.

Součástí práce je i vytvoření demonstrační webové aplikace pro chatování mezi uživateli. Na začátku praktické části je ukázáno vytvoření splnitelných požadavků na danou aplikaci a jak tyto požadavky budu následně řešit pomocí konvencí frameworku Next.js.

Velkou částí práce je tvorba dané demonstrační aplikace. Práce popisuje tvorbu aplikace od samotného začátku, tj. tvorba počáteční struktury, využití verzovacího systému, napojení na infrastrukturu společnosti Vercel a připojení databáze. Demonstrační aplikace slouží hlavně pro poukázání doporučených způsobů řešení problematik vývoje aplikace. Například jak nejlépe dosáhnout autentizace a autorizace pomocí Next.js konvencí. Část práce zaměřená na tvorbu aplikace poukazuje hlavně na tyto konvence a nejlepší doporučené způsoby, to ovšem neznamená, že postup ukázaný v práci je jediný správný.

Práce si tedy dává za cíl seznámit čtenáře s úplnými základy webového vývoje. Po přečtení by čtenář měl být schopný vytvořit libovolnou malou aplikaci ve frameworku Next.js a dostat ji bezplatně na internet prostřednictvím nástrojů nabízených společností Vercel.

V poslední části práce porovnávám framework Next.js s dalšími populárními frameworky na trhu. Tento subjektivní popis by měl pro čtenáře sloužit, jako reference, který framework se chce vlastně naučit. Next.js určitě není ten nejlepší nástroj na všechny druhy aplikace. Práce má proto za cíl i rychlé představení alternativ.

I. TEORETICKÁ ČÁST

1 WEBOVÁ APLIKACE

Webová aplikace je aplikace uložená na serveru a předávána přes internet do rozhraní prohlížeče. Díky tomuto nemusí být stažena přímo do počítače. K jejímu použití stačí uživateli internet a webový prohlížeč. [1]

1.1 Základní rozdělení webových aplikací

Webová aplikace je kombinace spousty navzájem komunikujících technologií. Technologie je široký pojem popisující ať už programovací jazyk nebo třeba část architektonického vzoru. V této kapitole se podíváme na rozdělení těchto technologií. Podle toho, jestli obstarávají front-end či back-end aplikace, jakou část architektonického vzoru obsluhují, jak ovlivňují rozložení zátěže aplikace nebo jak řeší komunikaci mezi serverem a prohlížečem.

1.1.1 Front-end a back-end

Rozdělení technologií podle toho, jestli jejich kód řeší to, co se děje v prohlížeči uživatele nebo to, co se děje na vzdáleném serveru.

1.1.1.1 Front-end

Front-end je část webové aplikace, která obstarává všechnu interakci s uživatelem. Od zobrazení textu ve webovém prohlížeči po reaktivitu tlačítek a odkazů. Hlavní technologie pro front-end jsou HTML pro strukturu, CSS pro vzhled stránky a JavaScript pro reaktivitu. Front-end je část aplikace, která se načítá na straně klienta, tedy v prohlížeči uživatele.

1.1.1.1.1 HTML

HTML neboli HyperText Markup Language je značkovací jazyk pro tvorbu struktury webové stránky. Nestará se o žádnou logiku, čistě definuje, kde se jaké prvky nacházejí. Prvkům také přidává identifikátory pro pozdější použití při tvorbě vzhledu stránky či její funkcionality. Mezi stavební bloky patří např. text, nadpis, tabulka, odkaz, formulář a mnohé další.

1.1.1.1.2 CSS

CSS neboli Cascading style sheets, v češtině také kaskádové styly je jazyk pro tvorbu vzhledu webové stránky. Má za úkol zachytit jednotlivé elementy stránky, ať už podle je-

jich druhu, nebo identifikátoru a vytvořit jim vzhled. Ve zkratce řeší estetiku webové stránky.

1.1.1.1.3 JavaScript

JavaScript je jednovláknový, asynchronní programovací jazyk pro web.[2] Zajišťuje veškerou funkcionalitu od responsivních tlačítek po načítání a zpracovávání dat z externích zdrojů. Jelikož webový prohlížeč dokáže nabídnout jen jedno vlákno pro veškerou práci, používá JS tzv. event loop. JavaScript tímto cyklem neustále prochází a kousek po kousku zpracovává každý příkaz zavěšený v tomto cyklu. Díky této technologii dokáže asynchronně pracovat na více úkolech. Bez JavaScriptu by byl web statický.

1.1.1.2 Back-end

Back-end obstarává tu část webové aplikace, kterou uživatel ve svém prohlížeči neuvidí. Back-end totiž sídlí někde daleko na serveru a veškerou komunikaci s klientskou částí aplikace provádí přes internet. Komunikace probíhá tak, že na back-end přijde požadavek pro určitá data, která back-end načte například z databáze či dalšího externího serveru a pošle je zpátky tomu, kdo je chtěl.

1.1.1.2.1 Server

Server řeší komunikaci s front-endem nebo jinými externími službami. Komunikuje pomocí HyperText Transfer Protokol (HTTP). HTTP používá pro komunikaci klíčová slovesa. Například sloveso GET říká, že po serveru něco chceme, třeba webovou stránku na zobrazení. Mezi další slovesa patří POST, PUT, PATCH, …

1.1.1.2.2 Logika

Tato část back-endu zpracovává informace z příchozích zpráv a podle nich generuje odpověď. Pro vytvoření odpovědi využívá naprogramovanou logiku, přístup k databázi a možnost komunikace s externími službami.

1.1.1.2.3 Databáze

Databáze je externí systém pro uložení a správu persistentních dat pro jakoukoliv službu. Databáze uchovávají širokou škálu informací, včetně uživatelských dat, informací o produktech, obsahu webových stránek atd. Umožňují webovým vývojářům efektivně organi-

zovat a spravovat velké množství dat a poskytují spolehlivý a bezpečný přístup k těmto datům. [3]

1.1.1.2.4 SQL

Standardizovaný jazyk pro práci s databází. Drtivá většina databází používá jazyk SQL pro všechny operace, jako jsou například tvorby tabulek či zobrazení dat. [4]

1.1.1.2.4.1 ORM

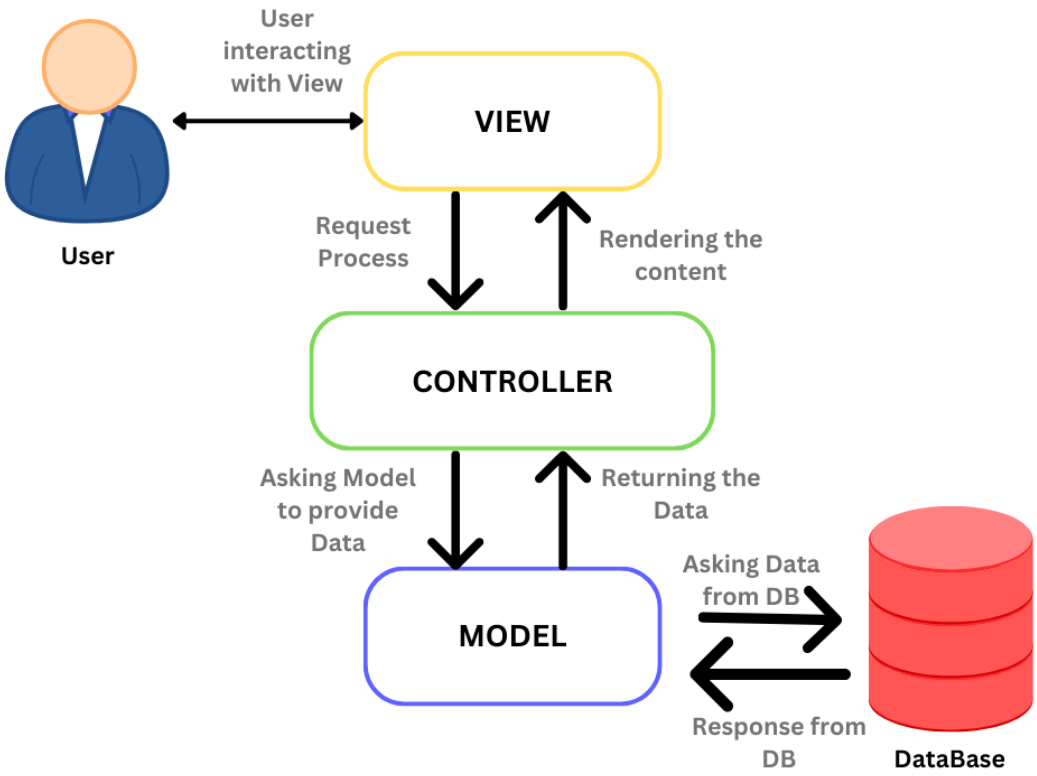
Object Relational Mapping (ORM) vytváří abstrakci nad databází pro použití v kódu. Ulehčuje jednak manipulaci s daty díky mapováním dat, tabulek a relací do datových struktur programovacího jazyka, ale také uložení jednoduchých SQL příkazů do těla funkcí použitelných v kódu. Takže v našem programu už pouze voláme funkce a projíždíme list objektů.

1.1.2 MVC a alternativy

Rozdělení technologií webových aplikací pomocí architektonických návrhových vzorů. Architektura webové aplikace je model interakce mezi komponenty webové aplikace. Konkrétní druh architektury pro webové aplikace striktně závisí na způsobu, jakým bude aplikační logika rozdělena mezi klientskou a serverovou stranu.

1.1.2.1 MVC

MVC je architektonický návrhový vzor, který si bere za úkol zlepšit organizaci aplikace díky „oddělení zodpovědnosti“. Tohoto cíle dosahuje pomocí separace částí řešící data (model), UI (pohled) a logiku (kontrolér). [5]



Obrázek 1 – MVC architektura [6]

1.1.2.1.1 Model

Model definuje, jaké data bude aplikace obsahovat. Například online obchod bude mít tabulku zboží, kde každá položka obsahuje název, cenu, atd. Díky nadefinování těchto modelů dat v aplikaci je můžeme jednodušeji implementovat do kódu. Model také definuje relace mezi daty a ukládá v sobě databázové migrace, což je seznam úprav provedených v databázi od jejího vytvoření. V programu si model můžeme představit jako článek mezi kontrolérem a databází.

1.1.2.1.2 View

View neboli pohled se stará o zobrazení stránky v prohlížeči, zobrazení dat na stránce a o posílání uživatelských vstupů do kontroléru. Pokud uživatel klikne na stránce na tlačítko pro zobrazení dat, pohled pošle na kontrolér požadavek pro data a jakmile je dostane, zobrazí je na stránce.

1.1.2.1.3 Controller

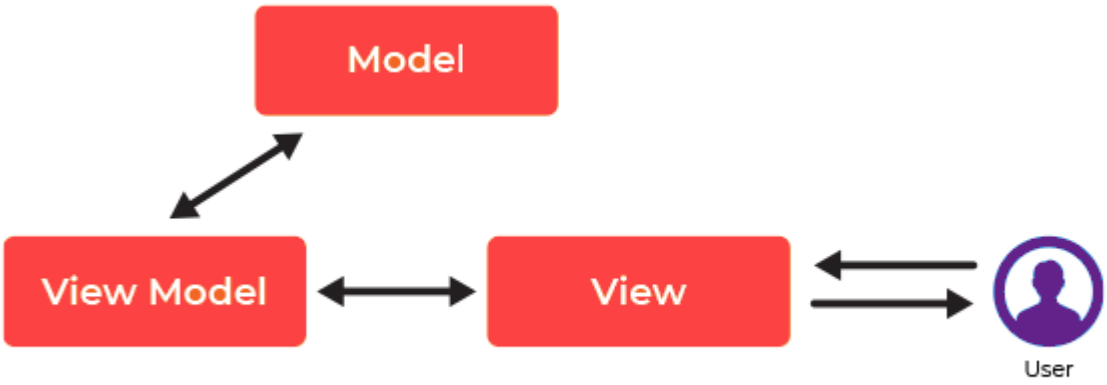
Controller neboli kontrolér obsahuje veškerou logiku aplikace. Aktualizuje model a pohled v reakci na vstupy uživatele. [7] V této sekci se také řeší routing neboli jaká stránka se má uživateli zobrazit, na jaké adrese, popř. jaké data se mají poslat, jsou-li ve webové aplikaci API endpointy.

1.1.2.2 Alternativy

Přestože je MVC nejpopulárnější vzor, má nějaké nevýhody a určitě není vhodný na všechny webové aplikace.

1.1.2.2.1 MVVM

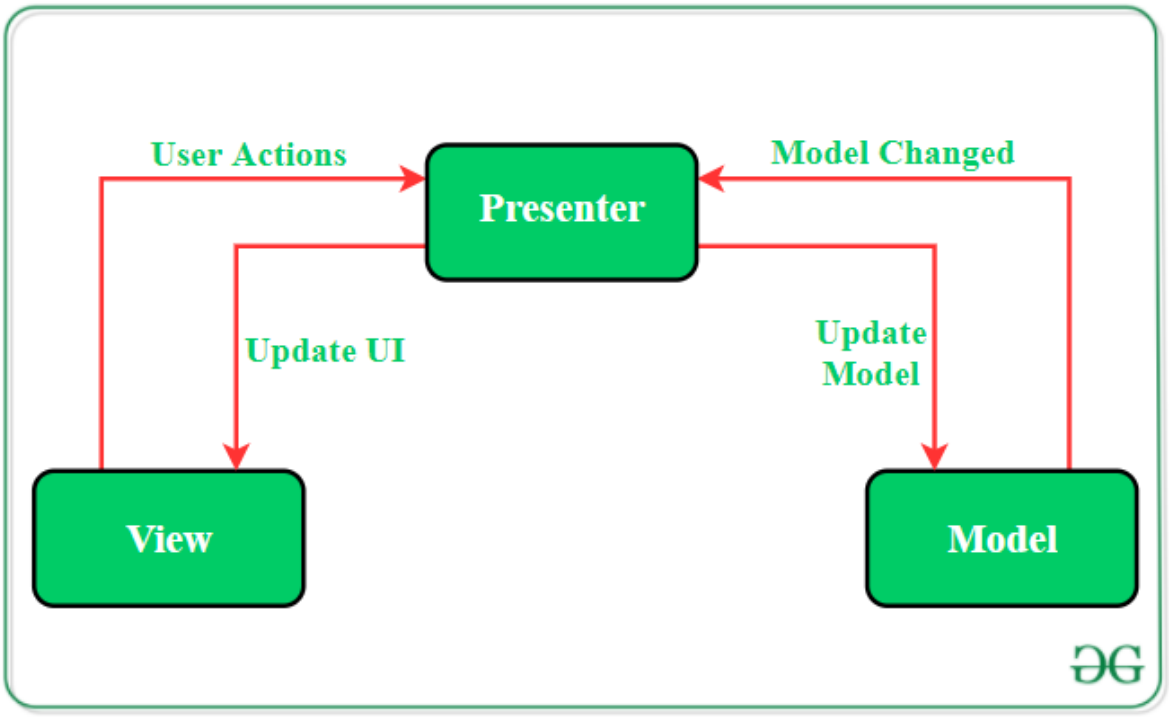
MVVM (Model-View-ViewModel) říká, že MVC má problém s tím, že komponenta pohledu dělá dvě úlohy. Prezentaci a datovou logiku prezentace. [8] MVVM čistě odděluje obchodní logiku aplikace od uživatelského rozhraní. Konečným cílem architektury MVVM je učinit pohled zcela nezávislým na aplikační logice. [9]



Obrázek 2 – MVVC architektura [10]

1.1.2.2.2 MVP

MVP (Model-View-Presenter) řeší podobnou problematiku jako MVVM. Pohled v této architektuře pouze vykresluje data. Je tedy zodpovědný za pasivní prezentaci na základě dat od uživatele. Posílání dat mezi modelem a pohledem řeší prezentující. [8]



Obrázek 3 – MVP architektura [11]

1.1.3 Rozdělení podle zátěže

Rozděluje webové aplikace podle poměru zátěže aplikace. Zda je výkonná část spíše na straně serveru, či klienta.

1.1.3.1 Rich client, thin server

Většinu práce dělá klient, znamená to pro nás delší čas načítání stránky, ale jakmile se načte, bude posílání požadavků na server minimální. Můžeme naplno využívat zdroje prohlížeče, jako je paměť cache. V případě získávání dat se stránka nejdříve podívá, zda má tyto data uložené v paměti cache. Jestli ano, zobrazí je, a až poté začne zjišťovat, jestli jsou relevantní. Pokud ne, stáhne nové a přepíše stránku.

1.1.3.2 Thin client, rich server

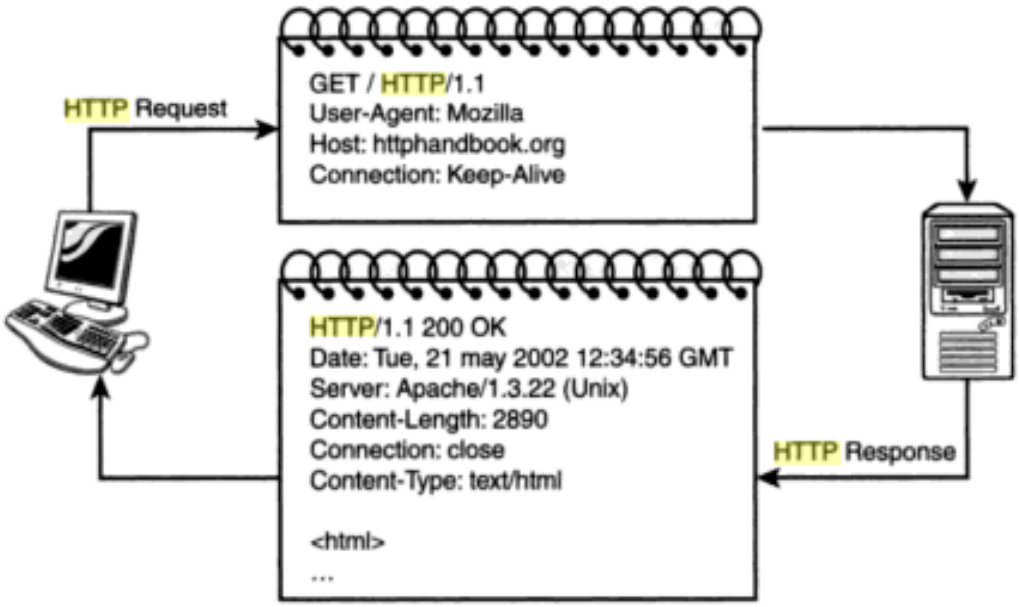
Většinu práce tady odvádí server. Ten má nejlépe většinu zdrojů již načtených, zpracovaných a uložených ve volatilní paměti. Na straně klienta jde tedy jen o dotazování serveru s požadavkem. V těle odpovědi už může očekávat vše hotové a připravené na zobrazení.

1.1.4 Rozdělení podle komunikace se serverem

Rozděluje webové aplikace podle formy komunikace se serverem. Pokaždé, když proběhne načtení stránky, ať už za účelem načtení nových dat, či jenom přesměrování na jinou adresu webové aplikace, je nutné komunikovat se serverem.

1.1.4.1 HTTP protokol

HTTP protokol používá vzor požadavek-odpověď. Klient vytvoří požadavek a pošle jej přes internet na server. Server požadavek zpracuje a posléze odešle odpověď zpátky. Každý http požadavek obsahuje všechny informace o klientovy a serveru. Obsahuje také informace důležité pro zpracování požadavku. Tento bez stavový design je výhodný, protože umožňuje nasadit další servery pro zpracování požadavků bez nutnosti synchronizace stavové logiky mezi servery.[12]

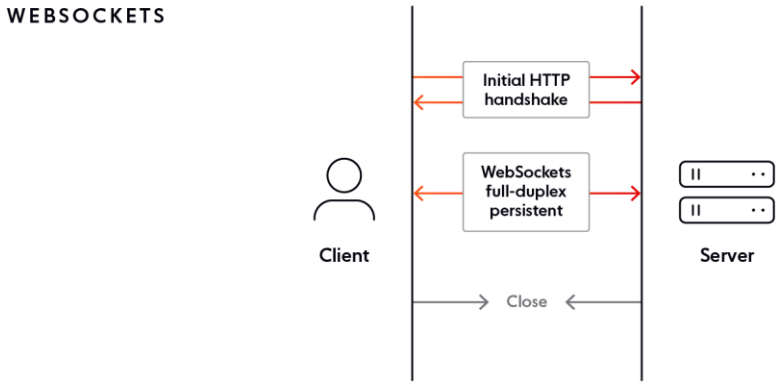


Obrázek 4 – způsob komunikace klienta a serveru přes http [13]

1.1.4.2 Websocket

Websocket nabízí plně-duplexní (oboustranné) spojení mezi klientem a serverem, což nám dovoluje jednoduchý datový přenos kdykoliv je potřeba. Websocket otevře spojení mezi klientem a serverem, takže s klasickým dotazováním klienta serveru o data můžeme data i

server poslat na klienta kdykoliv bude potřebovat. Klient se nemusí periodicky dotazovat serveru, jestli se například něco nezměnilo. Jakmile se něco změní, server to na klienta pošle. [14]



Obrázek 5 – způsob komunikace klienta a serveru přes websocket [11]

2 NEXT.JS

Next.js je framework pro tvorbu webových aplikací založený na knihovně React. React komponenty jsou zde použity nejenom k vytváření uživatelského rozhraní, ale i k dalším funkcím a optimalizacím. Framework Next.js přichází s kompletní sadou nástrojů pro práci s knihovnou React, jako jsou nástroje pro nastavení, kompilaci a další. [15]

Next.js je vyvinut a udržován společností Vercel, která nabízí spoustu služeb pro vývojáře webových aplikací v daném frameworku, například nasazení aplikace na jejich servery, nebo velice obsáhlou dokumentaci.

Next.js můžeme použít ve dvou módech, App router a Pages router. Pages router je originální nastavení Next.js aplikací a i přes to, že je starší, je stále aktualizovaný a podporovaný. Nicméně pro tuto práci se budeme zabývat pouze novějším App routerem, který podporuje nejnovější funkce knihovny React, jako jsou komponenty na serveru či streamování komponent. [15]

2.1 Použité technologie

V první části práce jsme si rozebrali, jak fungují technologie v jednotlivých částech webové aplikace a jejím vývoji. Nyní si řekneme, jaké konkrétní technologie řeší jakou část Next.js webové aplikace.

2.1.1 Front-end

React je hlavní stavební kámen celého frameworku, proto není divu že řeší kompletní UI a vlastně celý front-end aplikace. React je knihovna napsána v jazyku JavaScript pro tvorbu modulárních a reaktivních UI. [16] Tohoto dosahuje pomocí komponent. Komponenta je element, například text, tlačítko, tabulka, nebo skupina komponent. Dokonce celé kompletní okno webové aplikace je komponenta poskládaná z mnoha dalších komponent. React nám dovoluje tvořit znovupoužitelné komponenty, které dokážou být reaktivní, což znamená, že se uživateli mění data ve webové aplikaci před očima. [17] React dokáže sám o sobě kompletně nahradit HTML, CSS a JS na front-endu, nicméně je doporučeno použít HTML pro části webu, které jsou například stejné na každé stránce, jako může být navbar či footer. Také je doporučováno používat externí CSS soubor pro znovupoužitelnost stylů a přehlednost výsledného kódu.

2.1.2 Back-end

Back-end Next.js aplikace obstarává Node.js. Node.js nám dovoluje spustit Javascript jinde než jenom ve webovém prohlížeči, třeba na serveru. Node využívá technologie JavaScriptu jako náhradu za nástroje klasických serverových jazyků, například vlákna jsou nahrazena callbacky a eventy, asynchronnost je zařízena tzv. event loopem, atd. Další a snad i nejdůležitější aspekt Node.js leží v jeho ekosystému. Npm manažer balíčků a obrovská, stále rostoucí databáze nástrojů a modulů nabízejí knihovnu snad na všechno, co vývojář může potřebovat. [18]

2.1.2.1 Databáze

Vercel nabízí velice jednoduché propojení Next.js aplikace s PostgreSQL databází přes stránky Vercel a Github. Nabízí také knihovnu @vercel/postgres pro jednoduché ovládání databáze v aplikaci. [19]

2.1.3 Architektura

Framework neřeší, jakou architekturu programátor použije. Je to všechno v rukách vývojáře podle druhu a funkcionality vyvíjené aplikace. Next.js podporuje architekturu MVC.

2.1.4 Rozdělení zátěže

Opět záleží na řešeném problému. Framework nám umožňuje si zátěž rozložit jak je libo. Ovlivnit, zda se logika aplikace provádí na serveru nebo u klienta, můžeme hlavně u získávání externích dat a jejich ukládání. Pokud se rozhodneme získávat data a ukládat je na straně klienta, můžeme tak dělat zároveň s chodem webové aplikace. Dokud se však data nenačtou, stránka nebude kompletní. Při tomto stylu data ukládáme ve webové paměti, neboli cache. Pokud data získáváme na straně serveru, můžeme si je jednoduše přednačíst a na stranu klienta už jen posílat výsledky. Při použití tohoto způsobu se data ukládají nejčastěji do volatilní paměti, nicméně můžeme i do databáze.

2.1.5 Komunikace se serverem

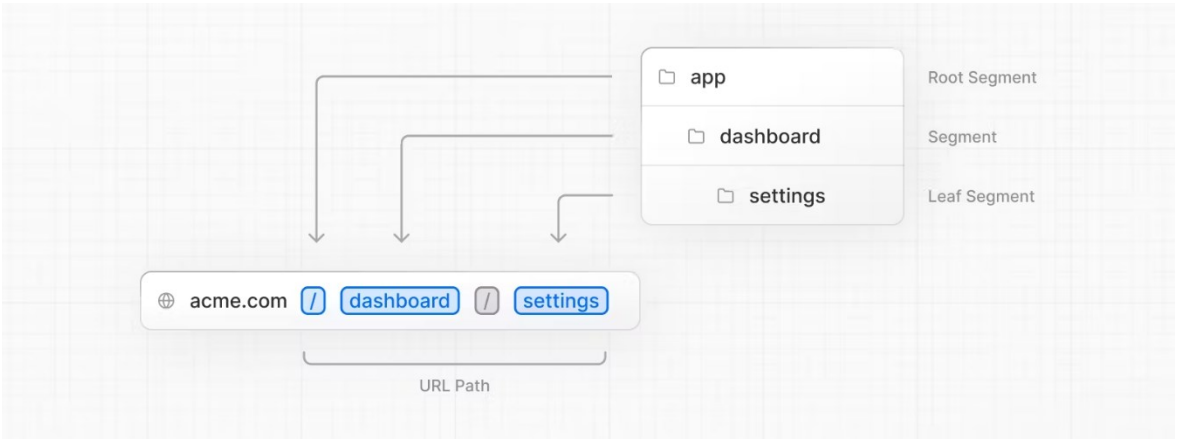
Framework používá pro interní komunikaci mezi front-endem a back-endem metodu http. Dovoluje ovšem vývojáři pro své účely využít i http i websockety. Metody a funkcionalita http je již zabalena v Node.js a npm nabízí knihovny pro websocket komunikaci. Technologie můžeme dokonce i nakombinovat a pro každou část aplikace použít jinou, podle potřeby.

2.2 Jak framework řeší klasické paradigma webového vývoje

Webové aplikace nabízí spoustu již definovaných funkcionalit. Je tedy na každém frameworku zařídit, aby tyto funkcionality, na které je již uživatel zvyklý, fungovaly. Nejlépe také, aby byly co nejrychlejší, nejsnadnější na implementaci a dobře zdokumentované.

2.2.1 Routing – směrování požadavků

Směrování požadavků zajišťuje, že každá specifická url naší aplikace ví, kde najde svůj kód. Jakmile uživatel otevře jakoukoliv adresu v naší webové aplikaci, pošle na server požadavek a je na frameworku zařídit, která část kódu (obvykle kontrolér) obstará tento požadavek. V Next.js frameworku je směrování řešeno pomocí adresářové struktury. Složka App v sobě schovává všechny cesty, které naše webová aplikace má, každá další podsložka koresponduje s textem za lomítkem v url.



Obrázek 6 – ukázka routingu v frameworku Next.js [20]

2.2.1.1 Konvence jmenování souborů

V adresářích následně vytvoříme React komponenty řešící jednotlivé části a možné stavy stránky podle jmenné konvence frameworku. Pro stránku jako takovou použijeme jméno souboru page.js (jsou povoleny také soubory typu .jsx nebo .tsx). Pro sdílené UI této routy a všech jejích podadresářů použijeme jméno souboru layout.js. Framework definuje spoustu dalších předdefinovaných souborů.

2.2.1.2 Dynamické routy

Většinou nám nestačí, aby stránka zobrazovala pokaždé to samé, občas chceme, aby například zobrazovala informace o různých prvcích v databázi. Toho dosáhneme pomocí dynamického směrování. Jméno složky dáme do hranatých závorek a pojmenujeme jak chceme,

nejčastěji id. Jméno, které jsme zvolili, se nám nakonec objeví v kódu jako proměnná s hodnotou, kterou uživatel napsal do url.

2.2.1.3 Middleware

Routy často používají tzv. middleware, což je nějaký nástroj, který stojí před samotným kontrolérem a vyhodnocuje či upravuje požadavek na něj. Může například řešit, jestli je url ve správném formátu ať se nikdo nesnaží do části, kde má přijít id, napsat nějaké písmena. Také může upravovat požadavek. Pokud posíláme data v JSON formátu a náš kontrolér je očekává ve formátu JavaScript objektu, použijeme middleware na formátování. Next.js využívá jmennou konvenci i pro soubor s middleware funkcemi, stačí soubor vytvořit v domovském adresáři projektu a pojmenovat jej middleware.ts.

2.2.2 Získávání a manipulace externích dat

Webové aplikace často používají externí data k vlastnímu účelu, ať už si berou aktuální počasí pro zobrazení na stránce, či noviny nebo zajímavé fakty o kočkách, nějak tyto informace musí získat.

2.2.2.1 Získávání dat pomocí fetch

Fetch je metoda implementovaná v nodejs, tudíž běží na straně serveru. Tato metoda nám dokáže asynchronně získat data z jakéhokoliv veřejného API. [21] Také se dá použít s jakoukoliv http metodou pro manipulaci dat, pokud nám to cílová stránka dovolí. Next.js funkcionalitu fetch API obohacuje o automatické a nastavitelné cáchování a revalidaci dat. Také automaticky vykresluje a převykresluje React komponenty na základě příchodu či změny dat. [22] Toto jsou všechno výhody získávání dat pomocí nativního fetche na straně serveru. Next.js také umožňuje využívat fetch v klientských komponentách.

2.2.2.2 Cachování

Framework umožňuje na každém fetch požadavku nastavit způsob a dobu ukládání dat. Nejčastěji se data ukládají do paměti prohlížeče, neboli cache. Data zde můžou zůstat i mezi načteními stránky a díky tomu můžeme urychlit právě načtení stránky, jelikož jsou data již stažené. Při načtení tedy proběhne pouze ověření dat podle nějakých námi stanovených podmínek. Například pokud jsou data starší než 10 min, stáhni je znova, jinak zobraz uložené. [22]

2.2.3 Optimalizace

Framework nabízí spoustu nástrojů pro zlepšení výkonu a klíčových faktorů kvality webové aplikace. Nabízí například sadu předpřipravených React komponent, Metadata API, Nástroje pro analýzu a monitoring, atd. [23] Dovoluje nám také jednoduše použít tzv. Lazy loading strategii, která zařídí, že se klientovi vždy pošlou pouze ty části webové aplikace, které zrovna potřebuje. [24]

2.2.4 Testování

Testování Next.js webové aplikace se provádí pomocí externích nástrojů. Typy testů, co lze napsat jsou Unit testy, End-to-End testy a Snapshot testy. Nástroje používané na testování jsou Vitest, Jest, Playwright, atd. [25]

2.2.5 Nasazení

Po vytvoření webové aplikace přichází nasazení na server a dostání na produkci. Framework podporuje nasazení pomocí samohostování na náš vlastní server, nasazení na vzdálený Node.js server, vypuštění aplikace do světa jako Docker kontejner či dokonce i nasazení jako statické html soubory. Doporučené je však nasadit Next.js aplikace na servery Vercelu. Výhody nasazení aplikace na servery Vercelu jsou, že pro nasazení je potřeba nulová konfigurace a můžeme si být jistí, že všechny funkce budou podporovány. [26]

2.3 Tvorba aplikace

Tvorba webové aplikace je velice komplexní úkon a nejtěžší část může být právě začátek. Je potřeba nakonfigurovat prostředí, stáhnout všechny knihovny a závislosti, vytvořit adresářovou strukturu atd. Naštěstí máme nástroje jako jsou generátory, které všechny tyto zdlouhavé úlohy udělají za nás.

2.3.1 Generátor

Next.js přichází s kompletním instalačním i nastavovacím generátorem. Jediné co je potřeba je Node.js verze 18.17+. Poté už stačí jen napsat do příkazového řádku příkaz
npx create-next-app@latest

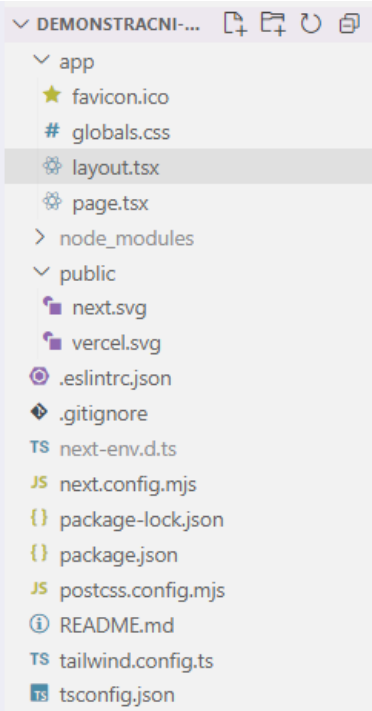
Tento příkaz nás provede instalací a nastavením našeho projektu. Zeptá se nás na konfigurační otázky, například jak se naše aplikace bude jmenovat a jestli chceme použít Ty-

peScript nebo JavaScript. [27] Pro účely aplikace zvolíme již před vybrané hodnoty. To znamená, že aplikace bude nakonfigurovaná pro TypeScript. [28]

Existují také externí generátory vytvořené uživateli, popřípadě lze stáhnout předvytvořenou strukturu a kód například z githubu. Pro účely této práce budeme používat oficiální generátor Next.js a zvolíme základní nastavení.

2.3.2 Struktura projektu

Po použití generátoru s defaultními hodnotami se nám vytvoří hned několik složek a souborů.



Obrázek 7 – adresářová struktura projektu otevřená ve vscode

Jméno	Typ	Úloha
app	složka	Veškerá logika aplikace, složka také slouží jako router
node_modules	složka	Stažené knihovny a moduly npm
public	složka	Statické asety aplikace
.gitignore	soubor	Říká gitu, které soubory a složky ignorovat. Například soubor enviromentálních proměnných
package.json	soubor	Hlavní konfigurační soubor node aplikací, popisuje aplikace,

		definuje scripty a má seznam externích knihoven

Tabulka 1 – důležité složky a soubory vygenerované Next.js aplikace

2.3.3 Vygenerované scripty v package.json

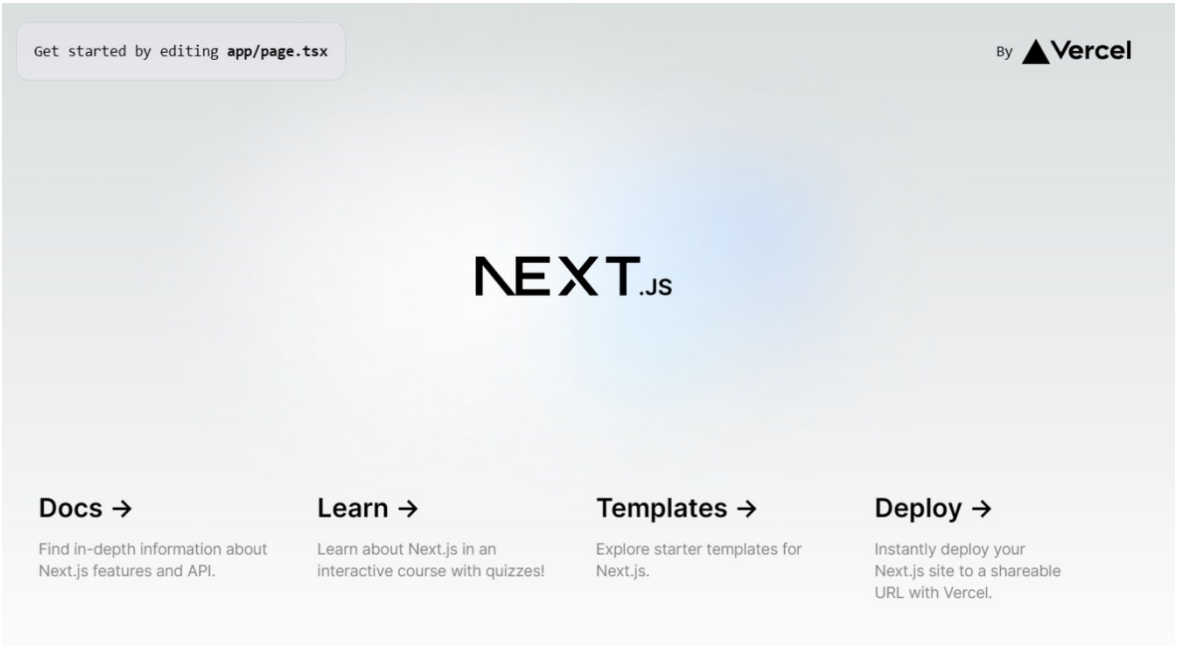
V souboru package.json se nachází předdefinované scripty pro ulehčení vývoje v node.js. Scripty se spouštějí v příkazovém řádku pomocí příkazu npm run a jméno scriptu. Ze začátku máme 4 scripty.

Script	Použití
npm run dev	Spustí aplikaci ve vývojovém módu. Disponuje funkcí hot reload.
npm run build	Zkompiluje aplikaci do produkčního stavu
npm run start	Spustí aplikaci z produkčního stavu.
npm run lint	Spustí eslint pro analýzu kódu.

Tabulka 2 – Základní scripty vygenerované Next.js aplikace

2.3.4 Vygenerovaná aplikace

Po spuštění scriptu npm run dev můžeme na adrese http://localhost:3000 najít rozběhlou vygenerovanou aplikaci. Aplikace pouze obsahuje loga a odkazy na dokumentaci.



Obrázek 8 – vygenerovaná a rozběhlá aplikace Next.js frameworku

II. PRAKTICKÁ ČÁST

3 PŘEDSTAVENÍ TVOŘENÉ WEBOVÉ APLIKACE

Ukázková aplikace bude sloužit jako chatovací aplikace obohacená o hlavní stránku s novinkami a nastavitelnými widgety. Bez přihlášení bude možné zobrazit noviny a nějaké zajímavosti. Po přihlášení bude možné psát si zprávy s ostatními uživateli. Možná bude i úprava vlastní profilu. Práce se bude nazývat demonstrační webová aplikace pro chatování mezi uživateli.

3.1 Funkcionální a nefunkcionální požadavky

Analýza požadavků je důležitý proces, který musí proběhnout před tvorbou jakékoliv aplikace. Pomocí nadefinovaných požadavků dokážeme určit, v jaké stavu vývoje aplikace jsme a co ještě musíme udělat, pro dokončení projektu.

3.1.1 Funkcionální požadavky

Funkcionální požadavky definují chování a funkcionalitu aplikace.

Uživatel	
F01	Uživatel si bude moct vytvořit účet
F02	Uživatel se bude moct přihlásit a odhlásit
F03	Uživatel si bude moct upravit vlastní profil
F04	Uživatel si bude moct psát s ostatními uživateli
F05	Uživatel bude moct aktualizovat widgety na hlavní stránce
Systém	
F06	Systém udržuje veškeré informace o uživatelích v databázi
F07	Systém bude při znovunačtení stránky aktualizovat widgety
F08	Systém zobrazuje uživateli zprávy hned po obdržení
F09	Systém udržuje záznam o přihlášeném uživateli a neodhlásí jej po zavření stránky

Tabulka 3 - Funkcionální požadavky

3.1.2 Nefunkcionální požadavky

Nefunkcionální požadavky definují vlastnosti aplikace. Jako je jeho rychlost načítání či škálovatelnost databáze

Dostupnost	
N01	Webová aplikace je optimalizovaná pro počítač
N02	Webová aplikace lze spustit na mobilním zařízení a tabletu
Zabezpečení	
N03	Webová aplikace šifruje uživatelská hesla
Výkon	
N04	Webová aplikace maximalizuje výkon využitím doporučených technologií frameworku Next.js

Tabulka 4 - Nefunkcionální požadavky

3.2 Řešení funkcionálních požadavků pomocí next.js konvencí

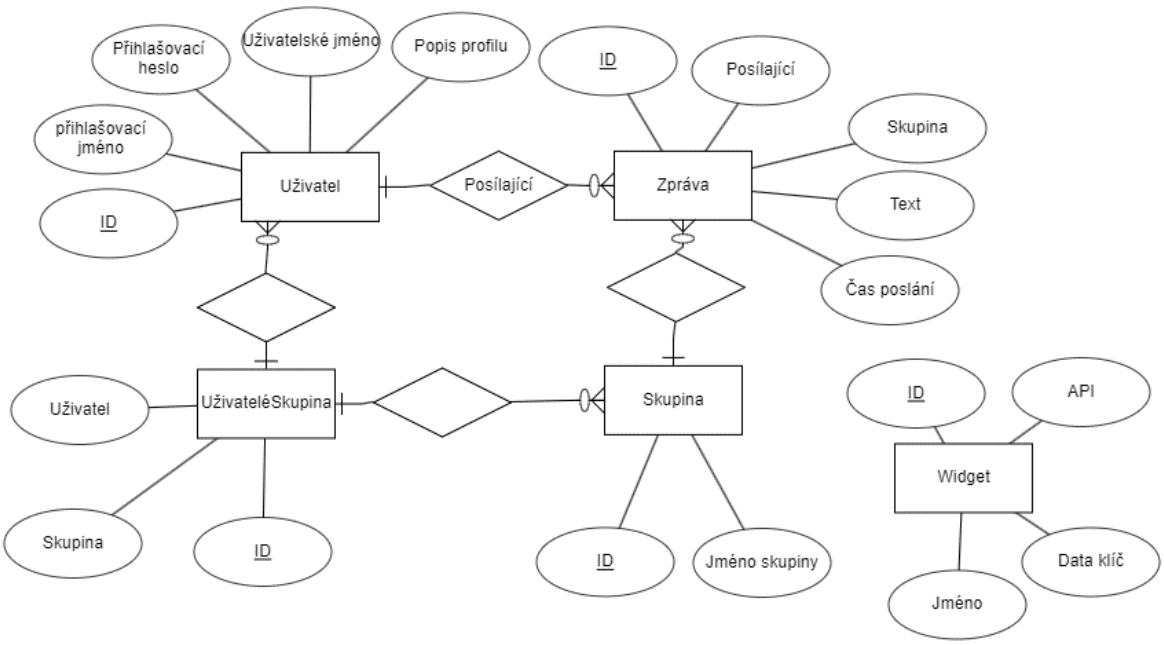
Tato kapitola obsahuje rozepsané způsoby řešení požadavků, spolu s čísly požadavků, které řeší.

- Vytvoření a úprava profilu – F01, F03, F06
    - Vytvoření profilu bude probíhat pomocí formuláře na straně klienta, který volá serverovou akci na vytvoření zápisu v databázi.
    - Úprava profilu bude probíhat přes API route. Formulář na straně klienta odešle data na API route, která se postará o úpravu záznamu v databázi.
- Autentizace a autorizace – F02, F06
    - Autentizaci bude uživatel provádět pomocí přihlašovacího jména a hesla. Obě data budou uložena v externí databázi. Heslo před uložením projde hashovací funkcí a do databáze se následně bude posílat až upravená verze hesla.
    - Autorizace bude probíhat pomocí middleware funkcí na každé stránce aplikace.
- Widgety – F05, F07
    - Widget bude vytvořen jako React komponenta. Budou zobrazovat informace obdržitelé zdarma na internetu.

- Komponenta widgetu bude obsahovat tlačítko pro aktualizaci dat.

- Komunikace mezi uživateli – F04
  - Komunikace bude probíhat přes technologii websocket, kvůli instantnímu zobrazení na stránce bez nutnosti obnovení stránky.
- Udržování informací – F06
  - Informace budou uskladněny v postgres databázi běžící na serveru společnosti Vercel
- Udržení přihlášeného uživatele – F09
  - Data o přihlášeném uživateli se budou uschovávat v cookies webového prohlížeče.

3.3 Model databáze



Obrázek 9 - ERD databáze webové aplikace [29]

V ukázkovém diagramu jsou jména entit a atributů v češtině. V popisování prvků databáze budu psát české názvy, nicméně v samotné aplikaci budu psát jména anglicky. Je to hlavně z důvodů, že některé frameworky jsou pro tvorbu webových aplikací zvyklé používat anglické skloňování ve své jmenné konvenci.

3.3.1 Popis entit

Tato kapitola se zabývá popisem tabulek v databázi aplikace.

Jméno tabulky	Anglický název	popis
Uživatel	users	Uchovává data o uživatelských účtech
Zpráva	messages	Uchovává data o všech zprávách
Skupina	threads	Uchovává data všech chatovacích skupin
UživatelSkupina	users_threads	Many-to-many tabulka pro uživatele ve skupinách
Widget	widgets	Uchovává data o widgetech.

Tabulka 5 - Popis tabulek v databázi

3.3.2 Popis relací

Tato kapitola se zabývá relacemi mezi tabulkami v databázi. Relace definují vztahy jednotlivých zápisů.

Tabulky	Relace	Popis
users-messages	one-to-many	Uživatel může mít několik zpráv. Zpráva může mít pouze jednoho posílajícího uživatele.
users-threads	many-to-many	Uživatel může patřit do několik skupin a každá skupina může mít několik uživatelů.
threads-messages	one-to-many	Skupina může obsahovat několik zpráv. Zpráva může být pouze v jedné skupině.

Tabulka 6 - Popis relací v databázi

3.3.3 Popis atributů

Tato poslední kapitola popisující návrh databáze se zabývá popisem atributů tabulek v databázi. Tabulky atributů ukazují názvy atributů v češtině i angličtině. Dále ukazují jejích datové typy a to jak v kódu tak v databázi. Nakonec ještě popisují doplňkové vlastnosti. Na konci kapitoly je tabulka vysvětlující doplňkové vlastnosti.

Atribut v češtině	Atribut v angličtině	Datový typ v kódu	Datový typ v databázi	Doplňkové vlastnosti
ID	id	string	integer	not null; auto-increment

Přihlašovací jméno	login	string	text	not null
Přihlašovací heslo	password	string	text	not null
Uživatelské jméno	username	string	text	not null
Popis profilu	description	string	text	

Tabulka 7 - Popis atributů pro tabulku Uživatel

Atribut v češtině	Atribut v angličtině	Datový typ v kódu	Datový typ v databázi	Doplňkové vlastnosti
ID	id	string	integer	not null; auto-increment
Posílající	user_id	string	integer	not null; foreign key
Skupina	thread_id	string	integer	not null; foreign key
Text	text	string	text	not null
Čas poslání	send_time	DateTime	timestamp	not null

Tabulka 8 - Popis atributů pro tabulku Zpráva

Atribut v češtině	Atribut v angličtině	Datový typ v kódu	Datový typ v databázi	Doplňkové vlastnosti
ID	id	string	integer	not null; auto-increment
Jméno skupiny	name	string	text	not null

Tabulka 9 - Popis atributů pro tabulku Skupina

Atribut v češtině	Atribut v angličtině	Datový typ v kódu	Datový typ v databázi	Doplňkové vlastnosti
ID	id	string	integer	not null; auto-increment

Uživatel	user_id	string	integer	not null; foreign key
Skupina	thread_id	string	integer	not null; foreign key

Tabulka 10 - Popis atributů pro tabulku UživatelSkupina

Atribut v češtině	Atribut v angličtině	Datový typ v kódu	Datový typ v databázi	Doplňkové vlastnosti
ID	id	string	integer	not null; auto-increment
API	api	string	text	not null
Data klíč	data_key	string	text	not null
Jméno	name	string	text	not null

Tabulka 11 - Popis atributů pro tabulku Widget

Doplňková vlastnost	Popis
not null	Sloupec musí obsahovat nějaký zápis, nemůže být nulový.
auto-increment	Data v sloupci se automaticky inkrementují.
foreign key	Odkazuje na ID zápisu v jiné tabulkce

Tabulka 12 - Popis doplňkových vlastností

4 TVORBA WEBOVÉ APLIKACE

Níže budou popsány všechny kroky pro vytvoření funkční webové aplikace. Popis generování základní struktury jsem vysvětlil u konce teoretické části. Tuto strukturu použijeme jako základní kámen naší aplikace.

4.1 Prerekvizity

Níže budou popsány jednotlivé kroky, které je potřeba udělat ještě před samotným programováním.

4.1.1 Vývojové prostředí

Pro vývoj aplikace jsem použil bezplatné vývojové prostředí Visual Studio Code v základním nastavení a bez jakýchkoliv doplňků. Visual Studio Code lze stáhnout ze stránky https://code.visualstudio.com.

4.1.2 Node.js

O běh back-endu aplikace se stará Node.js. Node se dá stáhnout ze stránky https://nodejs.org/en/download. V instalačním balíčku jsou i nástroje npm a npx.

4.1.2.1 Stažení závislostí

Součástí této kapitoly je seznam knihoven, které jsem nainstaloval příkazem npm install

Jméno knihovny	Obsah a funkcionalita
dotenv	Knihovna pro práci s enviromentálními proměnnými [30]
@vercel/postgres	Práce s databází v aplikaci [31]
jose	Podepisování a verifikace JWT [32]
zod	Validace dat ve formulářích [33]
bcrypt	Hashovací funkce [34]
luxon	Práce s datumy a časem [35]
socket.io	Vytvoření serveru pro komunikaci technologií websocket [36]
socket.io-client	Klientská komunikace technologií websocket [37]

Tabulka 13 - Seznam použitých knihoven

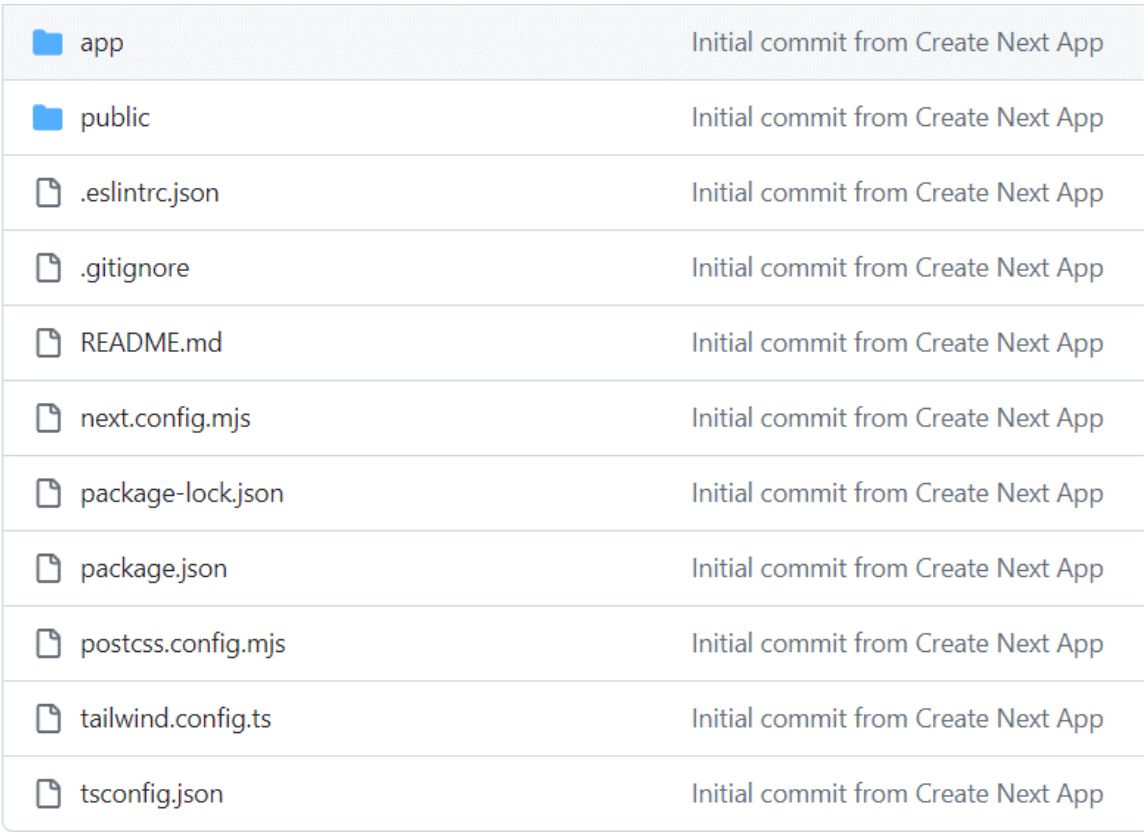
4.1.3 Git repozitář

Pro vývoj jakékoliv aplikace se hodí používat nástroj pro kontrolu verzí, jako je například git. Pro aplikaci jsem použil github, hlavně proto, že jej pozdější propojím s účtem na Vercelu. Propojení s Vercelem je důležité pro to, že mezi služby společnosti Vercel patří bezplatné nasazení aplikace na internet a také propojení s databází. Pro vytvoření účtu či přihlášení k účtu na Github je tato stránka https://github.com/login

4.1.3.1 Tvorba repozitáře

Na stránce Github jsem si vytvořil repozitář a pomocí pár jednoduchých kroků jsem jej propojil s vygenerovanou aplikací [38]

1) Vytvořil jsem privátní repozitář se jménem nextApp
2) V kořenu aplikace jsem do příkazového řádku napsal:
    a. git remote add origin https://github.com/mujUserName/ nextApp.git,
    b. git branch -M main
    c. git push -u origin main

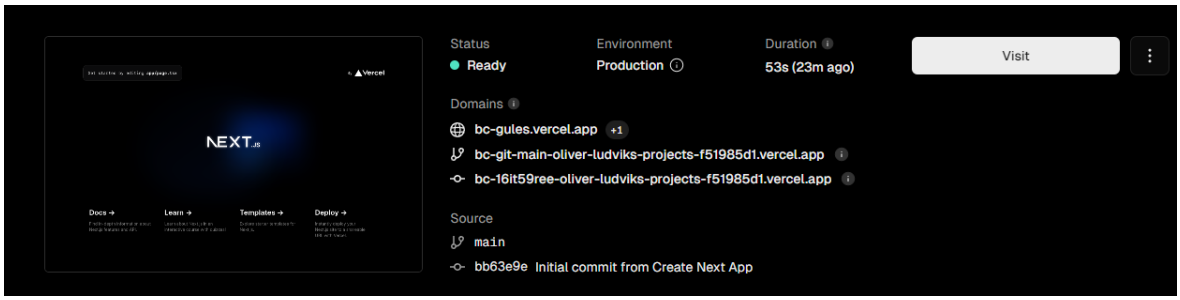


Obrázek 10 - Stav repozitáře po úspěšném vytvoření repozitáře

4.1.4 Účet na Vercel.com

Na stránce https://vercel.com/signup jsem si vytvořil hobby účet a ten propojil s Github účtem. Následně ještě stáhl aplikaci git a zvolil můj nextApp repozitář pro nasazení. Tyto kroky zařídí automatické postavení aplikace a její nasazení na servery Vercelu. Pokaždě, co upravím kód a ten následně pošlu na Github pomocí příkazů git commit a git push, Vercel automaticky přestaví aplikaci.

Hobby účet na stránce Vercel je zdarma a mířen na malé projekty [39]. Pro ukázkovou aplikaci je hobby účet dostačující.



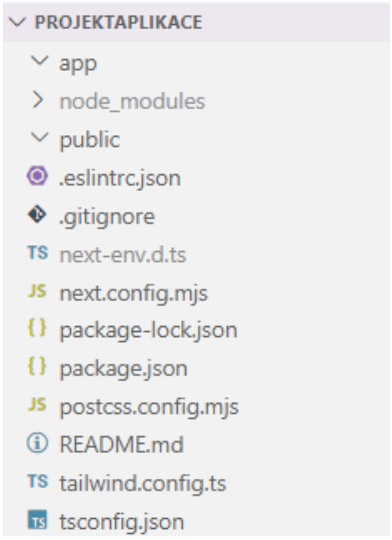
Obrázek 11 - Stav stránky Vercel.com po sestavení projektu

4.2 Úprava struktury vygenerované generátorem

Úpravy, které je potřeba udělal na vygenerované aplikace.

4.2.1 Smazání zbytečností

Generátor vytvořil základní kód pro otestování správného fungování aplikace. Nicméně tento kód ve finální verzi určitě nebude, proto jsem smazal většinu věcí a ponechal jenom základní složkovou strukturu.

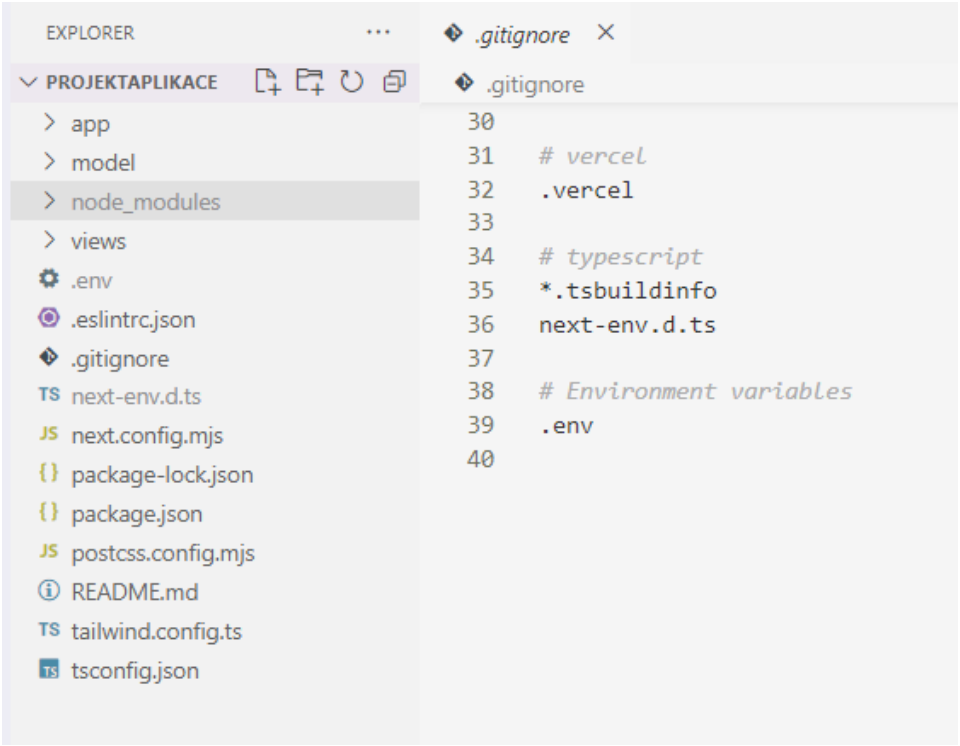


Obrázek 12 - Adresářová struktura po odmazání nepotřebných souborů

Složky app a public zůstaly po smazání nepotřebných souborů prázdné.

4.2.2 Příprava adresářů a tvorba důležitých souborů

V kořenovém adresáři jsem vytvořil složky views a model. Podle MVC konvencí bude složka views zastřešovat react komponenty s složka model se bude starat o databázi a přístup k ní. Přidal jsem také soubor .env pro proměnné prostředí, ten je potřeba zapsat do souboru .gitignore, protože do něj budu vkládat citlivé údaje. V adresáři app jsem vytvořil soubory layout.tsx a global.css. Soubor layout.tsx slouží pro strukturu, která půjde vidět na každé stránce aplikace. Zároveň definuje věci jako globální soubor css nebo informace o stránce. Soubor global.css mimo klasické využití pro stylování aplikace také importuje knihovnu tailwind css [40], která je součástí vygenerovaného next.js projektu. Do adresáře app budu dále přidávat složky a soubory pro vytváření cest a logiku kontroléru.



Obrázek 13 - připravená struktura a soubor .gitignore

Obrázek 14 - Soubor layout.tsx

```
app > # global.css
1  @tailwind base;
2  @tailwind components;
3  @tailwind utilities;
4
```

Obrázek 15 - Soubor global.css

Do konfiguračního souboru knihovny Tailwind jsem napsal adresáže, které mají knihovnu využívat. Jsou to adresáře model, views a app.

```
const config: Config = {
  content: [
    "./model/**/*.{js,ts,jsx,tsx,mdx}",
    "./views/**/*.{js,ts,jsx,tsx,mdx}",
    "./app/**/*.{js,ts,jsx,tsx,mdx}",
  ],
```

Obrázek 16 - Změna v souboru tailwind.config.ts

4.3 Tvorba a integrace databáze

V počátečním plánu tvorby aplikace bylo využít jednoduchou databázi PocketBase. Databáze PocketBase je jednosouborová databáze vložená přímo do kódu aplikace. [41] Nicméně od tohoto původního plánu jsem přešel k propojení s databází Postgres přes společnost Vercel.

Vercel nabízí také databázové služby k propojeným next.js aplikacím. Na stránce svého projektu jsem otevřel záložku Storage a zvolil si Postgres. Umístění databáze jsem dal co nejblíž k sobě a označil jsem databázi produkční a vývojovou. Vytvořila se mi prázdná databáze. Z její stránky jsem zkopíroval hodnoty v záložce .env.local a vložil je do souboru .env. To mi zařídí, že SDK pro postgres od Vercelu bude automaticky vědět, kam se připojit. Můžeme tedy databázi plně ovládat z aplikace.

4.3.1 Počáteční seed databáze

Pro vytvoření struktury databáze jsem napsal script seed.js do složky db. Ve scriptu se nachází potřebné funkce pro vytvoření každé tabulky a finální funkce pro spuštění všech funkcí.

```
const { db } = require("@vercel/postgres");
async function seedUsers(client) {
  const createTable = await client.sql`
    CREATE TABLE IF NOT EXISTS users (
      id SERIAL PRIMARY KEY,
      login TEXT NOT NULL,
      password TEXT NOT NULL,
      username TEXT NOT NULL UNIQUE,
      description TEXT
    );
  `;
  return createTable;
}
```

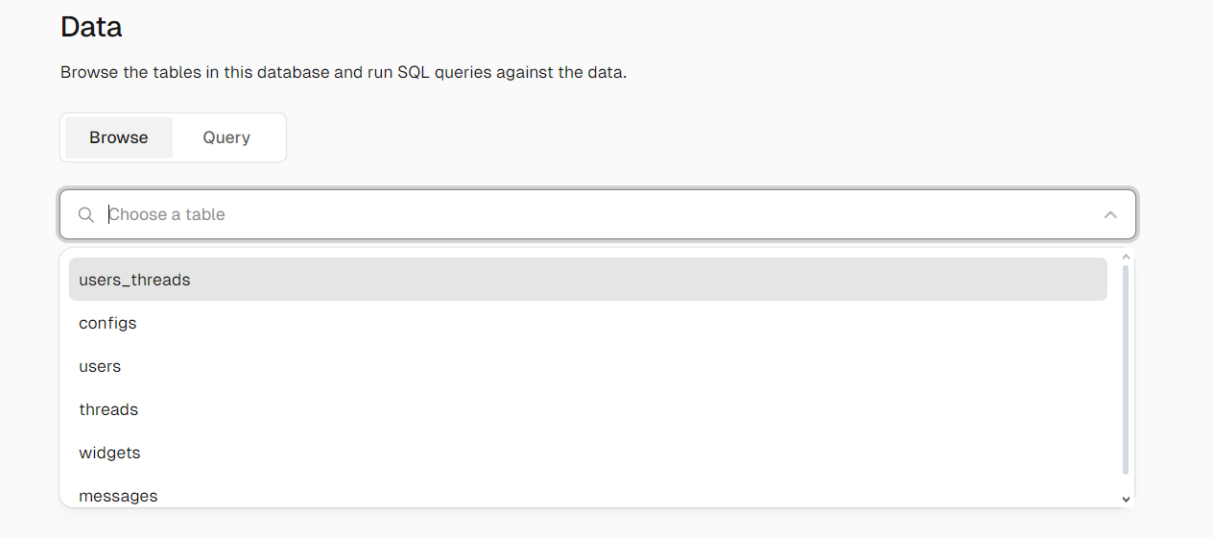
Obrázek 17 - Ukázková funkce pro vytvoření jedné tabulky

```
async function main() {
  const client = await db.connect();

  await seedUsers(client);
  await seedThreads(client);
  await seedMessages(client);
  await seedUsersThreads(client);
  await seedWidgets(client);
  await seedConfigurations(client);

  await client.end();
}

main().catch((err) => {
  console.log(err);
});
```

Obrázek 18 - Kód pro vytvoření struktury databáze

Next.js v základu nabízí knihovnu pro pracování s databází na úrovni SQL queries. Je možné si stáhnout hned několik ORM pro ulehčení práce, nicméně v aplikaci budu používat pouze knihovnu vercel/postgres.

Následně jsem do souboru package.json do objektu scripts přidal script na seed, který obsahuje příkaz "node -r dotenv/config ./db/seed.js". Po úspěšném běhu bychom měli být schopní na stránkách Vercelu zobrazit tabulky.



Obrázek 19 - Zobrazení databáze

4.4 Důležité kroky v procesu tvorby

V této kapitole budu popisovat důležité kroky při tvorbě a také prvky frameworku Next.js tak, jak se budou objevovat.

4.4.1 Tvorba hlavní stránky

V prvním kroku jsem vytvořil komponenty pro widget a nabídkové menu. Tyto komponenty pro svou funkcionalitu potřebují data z databáze, kvůli tomu jsem v složce model vytvořil soubor widgetData.ts.

4.4.1.1 Widget komponenta

```
'use client'
import { useEffect, useState } from "react"
export default function Widget(props:any) {
    const { url, name, dataKey } = props
    const [data, setData] = useState("loading")
    const fetchData = async () => {
        const req = await fetch(url);
        const newData = await req.json();
        return setData(newData[dataKey]);
    };
    const handleClick = (event:any) => {
        event.preventDefault();
        fetchData();
    };
    useEffect(() => {
        fetchData()
    },[])
    return (
        <div className="m-6 h-64 w-64 rounded overflow-hidden shadow-lg bg-slate-200 relative">
            <div className="border border-b-black text-center">{name}</div>
            <div>{data}</div>
            <button
            className="border h-16 border-t-black w-full bottom-0 absolute" onClick={handleClick}>
                FETCH DATA
            </button>
        </div>
    );
}
```

Obrázek 20 - Klientská komponenta widget

React komponenty se píší jako funkce začínající velkým písmenem. Na serveru můžeme použít synchronní nebo asynchronní komponenty, na straně klienta pouze synchronní. To, jestli je komponenta renderování na straně serveru nebo klienta určuje první řádek souboru, kde je buď napsáno ‘use client’, nebo ‘use server’. Díky tomu, že jsem komponentu definoval jako klientskou, můžu využít reaktivní elementy, jako je třeba tlačítko, ale také tzv. React hooky. V této komponentě využívám dva hooky a to sice useState a useEffect. Hook useState zařizuje perzistentnost dat mezi načtením komponenty a hook useEffect vytváří funkci, která se provede pokaždé, co se změní obsah proměnných v hranatých závorkách. V mojem případě se funkce provede jen při načtení komponenty. Data z rodičovské komponenty se předávají ve formě argumentu funkce pojmenovaného props. Tato proměnná se následně dekonstruuje.

Soubor widgetData.ts

```
model > TS widgetData.ts > ...
1   import { sql } from '@vercel/postgres';
2   import { unstable_noStore } from 'next/cache';
3   async function fetchWidgets() {
4       unstable_noStore()
5       const data = await sql`
6       SELECT * FROM widgets`;
7       return data.rows
8   }
9   export { fetchWidgets }
```

Obrázek 21 - Soubor model/widgetData.ts

Všechny soubory modelu budou exportovat asynchronní funkce, které provádí jednoduché SQL queries. Je to vlastně jako bych psal vlastní jednoduché ORM. Funkce využívá metodu unstable_noStore(), ta zařizuje, že výsledky z databáze nebudou uloženy v paměti cache aplikace. Využívám ji pro to, abych mohl bezstarostně přidávat widgety a ty se instantně objevovali na stránce. Jinak by se uložil výsledek volání SQL query a ten se použil místo získání nového výsledku.

Tyto kroky řeší požadavky F05 a F07.

4.4.1.2 Nabídkové menu

Komponentu nabídkového menu jsem vytvořil do souboru view/sidebar.jsx. Komponentu následně používám v souboru app/layout.tsx, a to proto, aby byla vidět na všech stránkách aplikace.

```tsx
app > layout.tsx > ...
import './global.css'
import Sidebar from '@/views/sidebar'
export const metadata = {
  title: 'Demonstrační webová aplikace pro chatování mezi uživateli',
}
export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="en">
      <body className='flex'>
        <Sidebar />
        <div className='flex-none'>
          {children}
        </div>
      </body>
    </html>
  )
}
```

Obrázek 22 - Soubor app/layout.tsx

4.4.2 Autentizace a autorizace

Při tvorbě části kódu pro autentizaci jsem pracoval podle návodu na stránce https://nextjs.org/docs/app/building-your-application/authentication. Autentizace má za úkol vytvoření účtu, přihlášení do něj a odhlášení. Také ale se stará o to, aby uživatel zůstal přihlášený napříč stránkami aplikace. Autorizace potom zajišťuje to, že stránky uživatelů jsou ochráněny před cizími lidmi.

4.4.2.1 Autentizace

Jako první jsem vytvořil klientskou komponentu s formulářem pro registraci nového uživatele. Komponenta obsahuje pouze základní html formulář, ovšem při odeslání posílá data do serverové akce. Serverové akce jsou speciální funkce, které lze volat právě z formulářů.

```tsx
"use client";

import { register } from "@/app/actions/auth";

export function RegisterForm() {
  return (
    <form
      action={register}
      className="bg-white shadow-md rounded px-8 pt-6 pb-8 m-4"
    >
```

Obrázek 23 - Způsob volání serverové akce z klientské komponenty

Serverová akce register sídlí ve složce app/actions/auth.ts

```
export async function register(formData: FormData) {
  //Validace dat
  const validatedFields = RegisterFormSchema.safeParse({
    login: formData.get("login"),
    username: formData.get("username"),
    password: formData.get("password"),
  });
  if (!validatedFields.success) {
    return {
      errors: validatedFields.error.flatten().fieldErrors,
    };
  }
  //hash hesla a vložení do databáze
  const { login, username, password } = validatedFields.data;
  const hashedPassword = await bcrypt.hash(password, 10);
  const user = await createUser(login, username, hashedPassword);
  if (!user) {
    return {
      message: "An error occurred while creating your account.",
    };
  }
  //Uložení dat do cookies
  await createSession(user.id);
  redirect(`/`);
}
```

Obrázek 24 - Serverová akce register

Funkce register dělá hned několik věcí. Na začátku validuje data pomocí knihovny zod. Knihovna zod definuje schéma dat z formuláře, například minimální počet znaků.

```
import { z } from "zod";
export const RegisterFormSchema = z.object({
  login: z.string().min(2, { message: 'Name must be at least 2 characters long.' }).trim(),
  username: z.string().min(2, { message: 'Name must be at least 2 characters long.' }).trim(),
  password: z.string()
  .min(8, { message: 'Be at least 8 characters long' })
  .regex(/[a-zA-Z]/, { message: 'Contain at least one letter.' })
  .regex(/[0-9]/, { message: 'Contain at least one number.' })
  .regex(/[^a-zA-Z0-9]/, {
    message: 'Contain at least one special character.',
  })
  .trim(),
});
```

Obrázek 25 - Schéma pro registrační formulář

Po validaci dat funkce register přejde k uložení nového uživatele do databáze. V sobě rovnou využívá hashovací funkci z knihovny bcrypt. Díky tomuto se nezpracované heslo nikdy nedostane ven z aplikace. Funkce následně vytvoří uživatelskou session.

```
const secretKey = process.env.SESSION_SECRET;
const encodedKey = new TextEncoder().encode(secretKey);
export async function encrypt(payload: SessionPayload) {
  return new SignJWT(payload)
    .setProtectedHeader({ alg: "HS256" })
    .setIssuedAt()
    .setExpirationTime("7d")
    .sign(encodedKey);
}
export async function createSession(userId: string) {
  const expiresAt = new Date(Date.now() + 7 * 24 * 60 * 60 * 1000);
  const session = await encrypt({ userId, expiresAt });
  cookies().set("session", session, {
    httpOnly: true,
    secure: true,
    expires: expiresAt,
    sameSite: "lax",
    path: "/",
  });
}
```

Obrázek 26 - Funkce pro podepsání tokenu a vytvoření uživatelské session

Než můžeme uložit do prohlížeče cookies s informacemi o uživatelské session, musíme data zahashovat a podepsat. Data hashujeme a podepisujeme kvůli tomu, protože cookies jsou nástroj webu, ke kterému má každý uživatel přístup. Bez hashování by si je mohl každý přečíst a bez podepisování by mohl každý vytvořit svoje. Pro obojí využijeme knihovnu Jose. Knihovna Jose potřebuje vygenerovaný tajný klíč. Ten jsem vygeneroval v terminálu pomocí příkazu ‘openssl rand -base64 32’. Výsledek z tohoto příkazu jsem následně uložil do souboru .env pod názvem SESSION_SECRET.

Po dokončení vytváření session nás funkce register přesměruje na domovskou stránku aplikace, už jako přihlášeného uživatele.

Funkce pro přihlášení funguje na stejné bázi, jen místo zapsání nového uživatele do databáze si pomocí přihlašovacího jméno najde uživatele a pomocí funkce bcrypt.compare porovná hesla. Funkce pro odhlášení pouze maže cookies, kde je uložená uživatelské session.

4.4.2.2 Autorizace

Autorizace zabraňuje neoprávněným uživatelům přístup k chráněným stránkám. Využívá k tomu soubor middleware.ts. Funkce exportovaná ze souboru middleware.ts se spustí pokaždé, když uživatel načte novou stránku.

```
import { NextRequest, NextResponse } from "next/server";
import { getUserFromSession } from "@/app/lib/session";

export default async function middleware(req: NextRequest) {
  const path = req.nextUrl.pathname;
  if(path.split('/')[1] == "message") {
    return NextResponse.next();
  }
  if (["/", "/login", "/register", "/logout"].includes(path)) {
    return NextResponse.next();
  }
  try {
    const user = await getUserFromSession();
    if (path != `/${user.id}` && path != `/${user.id}/edit`) {
      return NextResponse.redirect(new URL("/", req.nextUrl));
    }
  } catch (error) {
    return NextResponse.next();
  }
  return NextResponse.next();
}

export const config = {
  matcher: ["/((?!api|_next/static|_next/image|.*\\.png$).*)"],
};
```

Obrázek 27 - Soubor middleware.ts

Funkce nejdříve zkontroluje, zda uživatel nepřistupuje k nechráněným stránkám. Pokud ano, nechá jej pokračovat. Pokud ne, načte si uživatelovo ID z session. Jestli uživatelovo ID je stejné jako ID stránky ke které se snaží přistoupit, projde bez problémů. Jestli ne, je uživatel odkázán zpátky na hlavní stránku aplikace. Soubor middleware.ts také exportuje konfiguraci stránek, pro které se nepoužívá. Například stránky pro veřejné API.

Tyto kroky řeší požadavky F01, F02 a F09.

4.4.3 Upravení profilu

Pro upravení profilu jsem využil druhou možnou strategii propojení formuláře na straně klienta a funkcí na straně serveru, a to sice metodu s API.

```
'use client'

import { useRouter } from "next/navigation"
import { useState } from "react"

export default function EditForm(params:any) {
    const {user} = params
    const [username, setUsername] = useState("")
    const [desc, setDesc] = useState("")
    const router = useRouter()
    async function onSubmit(e:any) {
        e.preventDefault()
        const data = new FormData()
        data.set('username', username)
        data.set('description', desc)
        data.set('userId', user.id)
        await fetch('/api/edit', {
            method: 'POST',
            body: data
        })
        router.push('/')
    }
```

Obrázek 28 - První část formuláře pro úpravu profilu

```
  return (
    <form
      onSubmit={onSubmit}
      className="bg-white shadow-md rounded px-8 pt-6 pb
    >
      <div>Current Username: {user.username}</div>
      <label className="block text-gray-700 text-sm font
        New Username
      </label>
      <input
        className="shadow appearance-none border rounded w
        type="text"
        name="username"
        onChange={(e) => setUsername(e.target.value)}
      />
      <div>Current Description: {user.description}</div>
      <label className="block text-gray-700 text-sm font
        New Description
      </label>
      <input
        className="shadow appearance-none border rounded w
        type="text"
        name="description"
        onChange={(e) => setDesc(e.target.value)}
      />
      <input
        className="bg-blue-500 hover:bg-blue-700 te
        type="submit"
      />
    </form>
  );
}
```

Obrázek 29 - Druhá část formuláře pro úpravu profilu

Formulář funguje na jiné bázi, než formulář pro registaci. Data z vstupních polí formuláře se při změně ukládají do useState proměnných. Při kliknutí na tlačítko odeslat formulář se zabrání výchozímu chování html formuláře pomocí funkce preventDefault(). Data se namísto toho pošlou pomocí funkce fetch s metodou POST na API pro úpravu profilu.

```
import { editUser } from "@/model/users";
import { NextRequest, NextResponse } from "next/server";

export async function POST(request: NextRequest) {
  const data = await request.formData();
  await editUser(
    data.get("username") as string,
    data.get("description") as string,
    data.get("userId") as string
  );
  return NextResponse.json({ success: true });
}
```

Obrázek 30 - Soubor app/api/edit/route.ts

U souborů definujících API cesty se nepoužívá jako vstupní soubor page.tsx ale route.ts. Soubor route.ts načítá příchozí data z formuláře a pomocí SQL upraví samotný zápis v tabulce.

Tyto kroky řeší požadavek F03

4.4.4 Psaní si s ostatními uživateli

Tvorbu funkcionality pro psaní si mezi uživateli rozděluji na dva milníky. V prvním kroku jsem se zaměřil na zprovoznění chatovacího okna a možnosti napsat zprávu. Tato zpráva bude zapsána do databáze, nicméně na straně přijímajícího zprávy se zobrazí až po obnovení stránky. V druhém kroku použiji propojení technologie websocket s React komponentou chatovací okna pro natlačení zprávy do okna přijímajícího uživatele. Nejdříve jsem ale upravil komponentu Sidebar, aby po přihlášení zobrazovala všechny ostatní uživatele stránky.

4.4.4.1 Úprava komponenty Sidebar

```
{isUser ? (
  users.map(user2 => {
    return (
      <li key={user2.id} className="flex border ■border-b-black h-15">
  <a
        href={`/message/${user2.id}`}
        className="flex-1 ■text-gray-900 p-3 □hover:bg-gray-300 group"
      >
        {user2.username}
      </a>
  </li>
    )
  })
```

Obrázek 31 – Úprava v komponentě Sidebar

Komponenta využívá podmíněného zobrazení dat na základě toho, jestli je uživatel přihlášený. Následně používá funkci map pro projetí všech uživatel a zobrazení odkazu na chat s nimi. U zobrazování dat pomocí jakékoliv formy cyklu je důležité, aby každý nový rodičovský prvek měl unikátní identifikátor key. V mém případě je to id ostatních uživatelů.

4.4.4.2 Vytvoření logiky pro jednoduché psaní mezi uživateli

Pro psaní zpráv jsem nejdříve vytvořil serverovou komponentu pro načtení dat z databáze.

```
'use server'

import { getUserFromSession } from "@/app/lib/session"
import { findUser, getMessagesFromThreadId, getThreadForUsers } from '@/model/users';
import MessangerWindow from "@/views/messangerWindow";

export default async function Page({ params }: { params: { userId: string } }) {
    const currUser = await getUserFromSession()
    const counterUser = await findUser(params.userId)

    const thread = await getThreadForUsers(currUser.id, counterUser.id)
    const messages = await getMessagesFromThreadId(thread.id)

    return <MessangerWindow thread={thread} user={currUser} user2={counterUser} messages={messages} />
}
```

Obrázek 32 - Soubor app/message/[userId] /page.tsx

Komponenta využívá dynamické cesty. To znamená, že část cesty se může libovolně měnit. V mém případě se mění část uživatelského ID podle toho, kdo je zrovna přihlášený. Tato proměnná se následně objeví v kódu jako argument funkce Page. Komponenta načítá data jako oba dva uživatelé v chatovací místnosti, všechny zprávy místnosti a místnost samotnou. Komponenta pro posílání zpráv následně využívá již popsaných technologií.

```
export default function MessangerWindow(props: any) {
  const { thread, user, user2, messages } = props;

  const [ms, setMessages] = useState(messages);
  const [newMs, setNewMs] = useState("");

  async function onSubmit(e: any) {
    e.preventDefault();
    const date = DateTime.now().toUTC().toISO();
    const nm = await addMessageS(user.id, thread.id, newMs, date);
    setNewMs("");
    setMessages(ms.concat(nm));
  }
```

Obrázek 34 - První část komponenty chatovacího okna

```
  return (
    <div className="h-screen w-screen">
      <div className="h-screen text-center w-4/6 border ■border-black overflow-auto">
        {thread.name}
        <div className="border-t ■border-t-black grid grid-cols-1">
          {ms.map((message: any) => {
            if (message.user_id == user.id) {
              return (
                <div
                  key={message.id}
                  className="justify-self-end w-fit m-1 px-3 py-1 border rounded-3xl"
                >
                  {message.text}
                </div>
              );
            } else {
              return (
                <div
                  key={message.id}
                  className="w-fit m-1 px-3 py-1 border rounded-3xl"
                >
                  {message.text}
                </div>
              );
            }
          })}
        </div>
        <form
          onSubmit={onSubmit}
          className="absolute bottom-0 h-8 border ■border-black"
        >
          <input type="text" onChange={(e) => setNewMs(e.target.value)} />
          <button type="submit"> heelo </button>
        </form>
      </div>
    </div>
  );
}
```

Obrázek 33 - Druhá část komponenty chatovacího okna

Pro uložení typu timestamp do databáze využívám knihovnu luxon a její třídu DateTime.

Tyto kroky řeší požadavek F04.

4.4.4.3 Implementace technologie websocket

Pro správné fungování technologie websocket potřebujeme vytvořit soubor server.js v domovském adresáři projektu.

```
const express = require('express')
const app = express()
const http = require('http').createServer(app)

const io = require('socket.io')(http, {
  cors: {
    origin: '*',
  },
})

io.on('connection', (socket) => {
  socket.on('newMessage', (state) => {
    socket.broadcast.emit(`${state.thread_id}`, state)
  })
})

http.listen(3001, () => {
  console.log('✔ Server listening on port 3001')
})
```

Obrázek 35 - Soubor server.js

Soubor využívá knihoven express, http a socket.io pro vytvoření serveru. Server zachytává zprávy od klientů, kde tělo obsahuje informace o napsané zprávě klientem. Zprávu následně rozešle všem připojeným klientům. Aby klienti poznali, koho se to týká, je název zprávy ID místnosti.

```
socket.on(`${thread.id}`, (data) => {
  if (data.user_id != user.id && !ms.includes(data)) {
    setMessages(ms.concat([data]));
  }
});
async function onSubmit(e: any) {
  e.preventDefault();
  const date = DateTime.now().toUTC().toISO();
  const nm = await addMessageS(user.id, thread.id, newMs, date);
  socket.emit("newMessage", nm[0]);
  setNewMs("");
  setMessages(ms.concat(nm));
}
```

Obrázek 36 - Změny v komponentě chatovacího okna

Klientská komponenta po zachycení zprávy s id její místnosti natlačí novou zprávu do useState proměnné, a ta ji instantně vepíše do místnosti.

Přestože tato metoda posílání zpráv funguje skvěle ve vývojovém prostředí za použití dvou rozběhlých aplikací, nefunguje na produkční verzi. Pro zprovoznění by byl potřeba další externí server.

Tyto kroky řeší požadavek F08.

4.4.5 Testy

Framework Next.js nabízí dokumentaci pro mnoho testovacích knihoven. Pro ukázku jsem využil knihovnu jest. Knihovnu jsem nastavil podle návodu zde

https://nextjs.org/docs/app/building-your-application/testing/jest

```
import "@testing-library/jest-dom";
import Widget from "@/views/widget";
import { act } from "react";
import React from "react";
import ReactDOM from "react-dom/client";

const unmockedFetch = global.fetch;
let container = document.createElement("div");

beforeAll(() => {
  global.fetch = () => Promise.resolve({json: () => Promise.resolve([]),});
  container = document.createElement("div");
  document.body.appendChild(container);
});

afterAll(() => {
  global.fetch = unmockedFetch;
  document.body.removeChild(container);
  container = null;
});

describe("Test", () => {
  it("render testovaci komponenty", async () => {
    act(() => {
      ReactDOM.createRoot(container).render(
        <Widget url={"https://catfact.ninja/fact"} name={"Cat facts"} dataKey={"fact"} />
      );
    });
    const name = container.querySelector("#name");
    expect(name.textContent).toBe("Cat facts")
  });
});
```

Obrázek 37 - Test pro komponentu Widget

Komponentu Widget testuji velice jednoduše, a to sice jestli se zobrazovaný požadovaný nadpis zadaný v props komponenty.

4.5 Zprovoznění aplikace v lokálním vývojovém prostředí

Jak už jsem zmínil v předchozí kapitole, aplikaci díky technologii websocket nefunguje na produkčních serverech společnosti Vercel. To ale neznamená, že aplikace nejde zprovoznit v lokálním vývojovém prostředí. Stačí v jednom okně konzole rozběhnout server.js příkazem node server.js a v druhém okně rozběhnout samotnou aplikaci příkazem npm run dev.

4.6 Rekapitulace konceptů implementovaných ve frameworku Next.js

Tato kapitola slouží k rychlé rekapitulaci konvencí, použitých ve vývoji demonstrační aplikace. Poukazuje, na výhody a nevýhody těchto řešení.

4.6.1 Jazyk JavaScript na front-endu i back-endu

Framework využívá knihovnu React pro front-end a technologii Node.js pro back-end. Psaní v jazyku JavaScript napříč celým kódem aplikace ulehčuje vývoj a dělá z Next.js jednoduchý framework na naučení.

4.6.2 ORM

Framework v základu neobsahuje krom čistého SQL žádný nástroj pro práci s databází. Díky tomuto se může vývojář rozhodnout, jestli ušetří místo a vystačí si s čistým SQL, nebo využije jednu z mnoha knihoven z ekosystému NPM. Stránky frameworku doporučují použití knihovny a nabízejí pár otestovaných a ozkoušených knihoven.

4.6.3 Konfigurace rozdělení zátěže

Framework jako jeden z mála na trhu nabízí jednoduchou konfiguraci, zda se logika komponenty stane na straně serveru, či na straně klienta. Díky propojení React komponent a této technologie, jde jednoduše stavět aplikace, přesně tak jak chceme. Framework umožňuje komponenty různě kombinovat. Klientská komponenta může obsahovat serverovou komponentu a naopak.

4.6.4 Serverové akce

Pokud potřebujeme vykonat v klientské komponentě akci na straně serveru, typicky třeba vyřešení formuláře, může nastat problém. Serverové akce řeší tuto problematiku ne úplně elegantním způsobem. Serverové akce se dají také využit v cestách API.

4.6.5 Routing

Spousta webových frameworků využívá soubor pro definování všech cest aplikace. Next.js využívá adresářové struktury. Toto řešení funguje perfektně pro malé aplikace, ale dokáže být velice zmatené při růstu aplikace. Dynamické routes, neboli cesty, definující hodnotu proměnné, taky využívají tuto konvencu,

5 POROVNÁNÍ FRAMEWORKŮ PRO TVORBU WEBOVÉ APLIKACE

Porovnání frameworku Next.js s frameworky Ruby on Rails a ASP.NET. Porovnání je subjektivní na základě zkušeností a oficiálních zdrojů na stránkách jednotlivých frameworků.

5.1 Základní představení frameworků

Součástí kapitoly je tabulka s velice základními informacemi o porovnávaných frameworcích.

Framework	Společnost	Programovací jazyk	Datum vytvoření
Next.js	Vercel	JavaScript	25.10.2016
Ruby on Rails	Komunita	Ruby	25.7.2005
ASP.NET	Microsoft	C#	5.1.2002

Tabulka 14 - Základní představení frameworků

Z tabulky lze vyčíst například to, že framework Ruby on Rails není zastoupen nijakou velkou firmou. To může mít za důsledek nekonzistentní verze, více chyb, nebo pomalejší odpovědi podpory. Také tabulka říká, že Next.js je z frameworků nejnovější.

5.2 Popis klíčových charakteristik jednotlivých frameworků

Tato kapitula porovnána frameworky podle jejich dokumentace, konvence routování požadavků, dostupných ORM, technologií na front-endu a ekosystému. Porovnání je založeno čistě na zkušenostech a osobním názoru.

5.2.1 Dokumentace

Next.js – Dokumentace je rozdělena na části podle postupu tvorby. Popisuje tvorbu aplikace od instalace po nasazení a udržování. Dokumentace ukazuje a vysvětluje vše, co Next.js nabízí a obsahuje. Nicméně návody v dokumentaci dokážou být matoucí a občasně i nefunkční pro nejnovější verze frameworku.

Ruby on Rails – Dokumentace je rozdělena na části podle architektonického vzoru. Každá část má obsáhlý popis všeho, co je potřeba. Dokumentace je často nepřehledná a pro najití

potřebných dat je potřeba skákat z místa na místo. Součástí dokumentace je tvorba ukázkové aplikace.

ASP.NET – Dokumentace frameworku je neskutečně komplikovaná a zmatená. Obsahuje však popis instalace a spoustu tutoriálů.

5.2.2 Routing

Next.js využívá jmenných konvencí a struktury složek. Tento styl je jednoduchý na naučení, nicméně začne být nepřehledný u velkých projektů.

Ruby on Rails využívá konfigurační soubor routes.rb. Se souborem je velice těžké pracovat, ale za to je přehledný a hlavně velmi chytrý. Rails nabízí spoustu metod jak routes ovládat.

ASP.NET využívá pro veškerou komunikaci soubor Program.cs. Tento způsob se o moc neliší oproti implementaci Ruby on Rails, nicméně soubor Program.cs konfiguruje ještě další části aplikace.

5.2.3 ORM

Next.js v základu má pouze lehkou knihovnu pro tvorbu SQL queries a komunikaci s databází. Npm však nabízí spoustu externích knihoven, které využívají v základu stejnou knihovnu, jako Vercel/postgres. Takže není potřeba mít strach, že by knihovny byly nekompatibilní nebo nepoužitelné.

Ruby on Rails – Asi největší výhoda frameworku Ruby on Rails je jejich ActiveRecord ORM. V Ruby on Rails je všechno řešeno podle jmenných konvencí. Databázová tabulka users bude mít vytvořený model User.rb, ve kterém jsou popsány všechny jeho relace. Díky tomuto jde kdekoliv v kódu najít tabulku User a zavolat na ni jakoukoliv z mnoha metod ActiveRecord ORM.

ASP.NET – Základní ORM pro framework ASP.NET je Entity Framework Core. Je trochu složitější na zprovoznění, ale pak funguje jako klasické ORM s jednoduchými metodami nad tabulkami a možností zavolání vlastní SQL query.

5.2.4 Front-end

Next.js používá na front-endu knihovnu React. Knihovna je velice dobře zdokumentována. Základy knihovny React jsou velice jednoduché a příjemné na používání, ale obsahuje i komplikované nástroje pro tvorbu velice zajímavých komponent.

Ruby on Rails používají embedded ruby na front-endu. Embedded ruby je v podstatě html dokument, kde do složených závorek můžeme psát kód v jazyce ruby.

ASP.NET používá Razor pages. Technologie funguje hodně podobně jako embedded ruby.

5.2.5 Ekosystém

Next.js využívá ekosystému npm pro svoje knihovny. Knihovny jsou často velmi dobře zdokumentované a udržované. Existují zde knihovny na všechno.

Ruby on Rails využívá pro svoje knihovny službu RubyGems. Knihoven je také spousta, ale často jsou nekompatibilní s novějšími verzemi Ruby on Rails.

APS.NET využívá nuget packages. Výhodou je, jak jednoduché je nainstalovat knihovny do aplikace a také, že seznam knihoven je dostupný ve vývojovém prostředí visual studio.

5.3 Finální slova k porovnání

Pokud se někdo chce dostat do světa webových aplikací je podle mě Next.js ta správná volba. Jazyk JavaScript je využit jak na back-endu tak na front-endu, takže není potřeba se učit moc nových jazyků. Framework všechny technologie webového vývoje dobře popisuje ve své dokumentaci. Používání klientských a serverových komponent může být občas matoucí, ale vždy je ta možnost používat pouze ty serverové. Největší výhodu vidím v počtu udržovaných externích knihoven. Framework Next.js silně doporučuji na začátek programování webových aplikací, hlavně pro osvojení technologií, které je potřeba v tomto odvětví programování umět.

Ostatní frameworky, tím myslím Ruby on Rails a ASP.NET jsou designované spíše na větší aplikace. Tady pak vidím největší rozdíl v tom, jestli chce vývojář raději programovat v jazyku C# nebo Ruby.

ZÁVĚR

Tato bakalářská práce měla za úkol tyto věci. Vysvětlit vše potřebné pro pochopení tvorby webových aplikací. Ukázat, jaké technologie používá framework Next.js a také jak řeší problematiky webového vývoje. Popsat a představit tvorbu jednoduché aplikace ve frameworku. A nakonec také představit další populární frameworky na trhu, porovnat je a ukázat, v čem každý z nich vyniká.

Teoretická část se zabývala hlavně představením světa webových aplikací. Popisovala všechny technologie, s kterými se při tvorbě aplikací pro web může vývojář potkat. Dále seznámila čtenáře s koncepty frameworku Next.js.

Velkou součástí praktické části byla ukázka tvorby aplikace ve frameworku Next.js. Popisovala tvorbu od začátku, vytvoření struktury projektu generátorem, po konec, nasazení aplikace na servery společnosti Vercel. Na tvorbě aplikace práce popsala hlavní konvence frameworku. Při tvorbě aplikace práce popisovala, jaké výhody a nevýhody nesou některé netradiční řešení frameworku Next.js. Tato aplikace byla testována při tvorbě mým častým zkoušením základních funkcionalit pro každou verzi.

Konec práce se zaobírá porovnáním s ostatními frameworky. Tato kapitola splnila cíl seznámení uživatele s alternativy k Next.js a také k ukázání, jak důležité je, vybrat si správný framework.

Výstupem praktické části bakalářské práce je i samotná demonstrační aplikace pro chatování mezi uživateli.

SEZNAM POUŽITÉ LITERATURY

[1] Web application (web app). Online. Techtarget. C2006–2024. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app. [cit. 2024-04-04].

[2] If Javascript Is Single Threaded, How Is It Asynchronous? Online. DEV.TO. © 2016 – 2024. Dostupné z https://dev.to/bbarbour/if-javascript-is-single-threaded-how-is-it-asynchronous-56gd. [cit. 2024-04-05].

[3] The Role of Databases in Web Development and How to Manage Them. Online. Timespro. C2024. Dostupné z: https://timespro.com/blog/the-role-of-databases-in-web-development-and-how-to-manage-them. [cit. 2024-04-05].

[4] VYSTAVĚL, Radek. Myslete databázově, myslete v SQL!. Ondřejov: Radek Vystavěl, 2023. ISBN 978-80-908144-1-7. [cit. 2023-11-10].

[5] OSMANI, Addy. Learning JavaScript Design Patterns. 2. O'Reilly Media, 2023. ISBN 1098139836, 9781098139834.

[6] The MVC Architecture. Online. Medium. 2023. Dostupné z: https://medium.com/@sadikarahmantanisha/the-mvc-architecture-97d47e071eb2. [cit. 2024-04-08].

[7] MVC. Online. Mdn web docs. ©1998–2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/MVC. [cit. 2024-04-09].

[8] FERRARA, Anthony. Alternatives To MVC. Online. Ircmaxell's blog. ©2011-2023. Dostupné z: https://blog.ircmaxell.com/2014/11/alternatives-to-mvc.html. [cit. 2024-04-09].

[9] GALLARDO, Estefanía García. What Is MVVM Architecture? Online. Builtin. ©2024. Dostupné z: https://builtin.com/software-engineering-perspectives/mvvm-architecture. [cit. 2024-04-09].

[10] MARTIN, Matthew. MVC vs MVVM – Difference Between Them. Online. Guru99. ©2024. Dostupné z: https://www.guru99.com/mvc-vs-mvvm.html. [cit. 2024-04-09].

[11] MVP (Model View Presenter) Architecture Pattern in Android with Example. Online. Geeksforgeeks. Dostupné z: https://www.geeksforgeeks.org/mvp-model-view-presenter-architecture-pattern-in-android-with-example/. [cit. 2024-04-09].

[12] BOOKER, Alex. WebSockets vs HTTP: Which to choose for your project in 2024. Online. Ably. Dostupné z: https://ably.com/topic/websockets-vs-http. [cit. 2024-04-09].

[13] SHIFLETT, Chris. HTTP Developer's Handbook. Ilustrované vydání. Sams Publishing, 2003. ISBN 0672324547, 9780672324543.

[14] CHOPRA, Varun. WebSocket Essentials – Building Apps with HTML5 WebSockets. Packt Publishing, 2015. ISBN 1784395005, 9781784395001.

[15] The React Framework for the Web [online]. [cit. 2023-11-10]. Dostupné z: https://nextjs.org

[16] React The library for web and native user interfaces [online]. [cit. 2023-11-10]. Dostupné z: https://react.dev. [cit. 2023-11-10].

[17] CHIARELLI, Andrea. Beginning React: Simplify your frontend development workflow and enhance the user experience of your applications with React. Packt Publishing, 2018. ISBN 1789534925, 9781789534924. [cit. 2024-05-01].

[18] CASCIARO, Mario a MAMMINO, Luciano. Node.js Design Patterns. 2. Packt Publishing, 2016. ISBN 1785887386, 9781785887383. [cit. 2024-05-01].

[19] Setting Up Your Database. Online. VERCEL, INC. Next.js. © 2024. Dostupné z: https://nextjs.org/learn/dashboard-app/setting-up-your-database. [cit. 2024-05-01].

[20] Routing Fundamentals. Online. VERCEL, INC. Nextjs. © 2024. Dostupné z: https://nextjs.org/docs/app/building-your-application/routing. [cit. 2024-04-09].

[21] RICHARDSON, Leonard a AMUNDSEN, Michael. RESTful Web APIs. Sebastopol: O'Reilly, 2013. ISBN 9781449358068. [cit. 2024-04-09].

[22] Data Fetching, Caching, and Revalidating. Online. VERCEL, INC. Nextjs. © 2024. Dostupné z: https://nextjs.org/docs/app/building-your-application/data-fetching/fetching-caching-and-revalidating. [cit. 2024-04-09].

[23] Optimizations. Online. VERCEL, INC. Nextjs. © 2024. Dostupné z: https://nextjs.org/docs/app/building-your-application/optimizing. [cit. 2024-04-09].

[24] Lazy Loading. Online. VERCEL, INC. Nextjs. © 2024. Dostupné z: https://nextjs.org/docs/app/building-your-application/optimizing/lazy-loading. [cit. 2024-04-09].

[25] Testing. Online. VERCEL, INC. Nextjs. © 2024. Dostupné z: https://nextjs.org/docs/app/building-your-application/testing. [cit. 2024-04-09].

[26] Deploying. Online. VERCEL, INC. Nextjs. © 2024. Dostupné z: https://nextjs.org/docs/app/building-your-application/deploying. [cit. 2024-04-09].

[27] Installation. Online. VERCEL, INC. Nextjs. © 2024. Dostupné z: https://nextjs.org/docs/getting-started/installation. [cit. 2024-04-09].

[28] TypeScript [online]. [cit. 2023-11-10]. Dostupné z: https://www.typescriptlang.org

[29] Thread messaging system database schema design. Online. In: Stack overflow. 2008. Dostupné z: https://stackoverflow.com/questions/6541302/thread-messaging-system-database-schema-design. [cit. 2024-05-01].

[30] Dotenv. Online. Npm Docs. © 2024. Dostupné z: https://www.npmjs.com/package/dotenv. [cit. 2024-05-02].

[31] @vercel/postgres. Online. VERCEL, INC. Vercel Documentation. © 2024. Dostupné z: https://vercel.com/docs/storage/vercel-postgres/sdk. [cit. 2024-05-02].

[32] jose. Online. Npm Docs. © 2024. Dostupné z: https://www.npmjs.com/package/jose. [cit. 2024-05-02].

[33] zod. Online. Npm Docs. © 2024. Dostupné z: https://www.npmjs.com/package/zod. [cit. 2024-05-02].

[34] node.bcrypt.js. Online. Npm Docs. © 2024. Dostupné z: https://www.npmjs.com/package/bcrypt. [cit. 2024-05-02].

[35] Luxon. Online. Npm Docs. © 2024. Dostupné z: https://www.npmjs.com/package/luxon#features. [cit. 2024-05-02].

[36] socket.io. Online. Npm Docs. © 2024. Dostupné z: https://www.npmjs.com/package/socket.io. [cit. 2024-05-02].

[37] socket.io-client. Online. Npm Docs. © 2024. Dostupné z: https://www.npmjs.com/package/socket.io-client. [cit. 2024-05-02].

[38] GITHUB, INC. Quickstart for repositories. Online. GitHub Docs. © 2024. Dostupné z: https://docs.github.com/en/repositories/creating-and-managing-repositories/quickstart-for-repositories. [cit. 2024-05-02].

[39] Vercel Hobby Plan. Online. VERCEL, INC. Vercel Documentation. © 2024. Dostupné z: https://vercel.com/docs/accounts/plans/hobby. [cit. 2024-05-03].

[40] Tailwind CSS [online]. [cit. 2023-11-10]. Dostupné z: https://tailwindcss.com

[41] PocketBase [online]. [cit. 2023-11-10]. Dostupné z: https://pocketbase.io

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	application programming interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
HTTP	Hypertext Transfer Protocol
SQL	Structured Query Language
ORM	Object Relational Mapping
MVC	Model-view-controller
MVVM	Model–view–viewmodel
MVP	Model–view–presenter
UI	User Interface
JSX	JavaScript XML
TSX	TypeScript XML
NPM	Node package manager
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
Vscode	Visual studio code
ERD	Entity Relationship Diagram
SDK	Software development kit
DB	Database
ID	Identification

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: PŘILOŽENÉ CD

CD obsahuje:

- Bakalářskou práci ve formátu PDF/A
- Složku se zdrojovým kódem webové aplikace