Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

Izomorfní webové aplikace s využitím frameworků Next.js a Meteor.js

Isomorphic web applications using Next.js and Meteor.js frameworks

Bakalářská práce

Vypracoval: Jan Štrosser

Vedoucí práce: PaedDr. Petr Pexa, Ph.D.

České Budějovice 2023

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Pedagogická fakulta

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan ŠTROSSER**
Osobní číslo:	**P19451**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie a e-learning**
Téma práce:	**Izomorfní webové aplikace s využitím frameworků Next.js a Meteor.js**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování

Cílem bakalářské práce bude představit a otestovat principy tvorby tzv. izomorfních webových aplikací s využitím nových frameworků Next.js a Meteor.js. Základní charakteristikou izomorfní webové aplikace je sdílení kódu mezi její serverovou a klientskou stranou, v tomto případě se jedná o použití JavaScriptu nejen pro webový prohlížeč, ale také pro server. Realizace tématu bude zaměřena na tvorbu izomorfních aplikací pomocí dvou nových frameworků Next.js a Meteor.js, které zefektivňují a usnadňují vývoj aplikací. Bude především zpracován jejich význam a porovnání s klasickou tvorbou webových aplikací a budou otestovány výhody a nevýhody jejich použití. V praktické části práce bude vytvořena izomorfní Single page webová aplikace (SPA) s využitím server-side renderingu (SSR), který oba frameworky využívají, a to ve dvou variantách v obou zmíněných frameworcích. Opět bude následovat porovnání obou použitých postupů, zjištěny rozdíly, výhody a nevýhody použití Nextu a Meteoru z praktického pohledu vývojáře. Při tvorbě komplexní aplikace budou také využívány tradiční JavaScriptové frameworky React a Node.js

Rozsah pracovní zprávy:	**40**
Rozsah grafických prací:	**CD ROM**
Forma zpracování bakalářské práce:	**tištěná**

Seznam doporučené literatury:

1. Create a Next.js App | Learn Next.js. Next.js by Vercel – The React Framework [online]. Copyright © [cit. 02.04.2021]. Dostupné z: https://nextjs.org/learn/basics/create-nextjs-app?utm_source=next-site&utm_medium=homepage-cta&utm_campaign=next-website
2. Meteor Software: A Platform to Build, Host, Deploy and Scale Full-Stack Javascript Applications [online]. Copyright © 2021 Meteor [cit. 02.04.2021]. Dostupné z: https://www.meteor.com/
3. MeteorJS – platforma pre super rýchly vývoj real-time aplikácií – Zdroják. Zdroják – o tvorbě webových stránek a aplikací [online]. Dostupné z: https://zdrojak.cz/clanky/meteorjs-platforma/
4. Getting Started – React. React – A JavaScript library for building user interfaces [online]. Copyright © 2021 Facebook Inc. [cit. 02.04.2021]. Dostupné z: https://reactjs.org/docs/getting-started.html
5. ASP.NET MVC – Single Page Application. itnetwork.cz – Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další. [online]. Copyright © 2021 itnetwork.cz. Veškerý obsah webu [cit. 02.04.2021]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net-mvc/single-page-application

Vedoucí bakalářské práce: **PaedDr. Petr Pexa, Ph.D.**
Katedra informatiky

Datum zadání bakalářské práce: **7. dubna 2021**
Termín odevzdání bakalářské práce: **30. dubna 2022**

L.S.

doc. RNDr. Helena Koldová, Ph.D.
děkanka

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 7. dubna 2021

Prohlášení

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne 18. dubna 2023.

Jan Štrosser

Abstrakt/Anotace

Cílem bakalářské práce bude představit a otestovat principy tvorby tzv. izomorfních webových aplikací s využitím nových frameworků Next.js a Meteor.js. Základní charakteristikou izomorfní webové aplikace je sdílení kódu mezi její serverovou a klientskou stranou, v tomto případě se jedná o použití JavaScriptu nejen pro webový prohlížeč, ale také pro server. Realizace tématu bude zaměřena na tvorbu izomorfních aplikací pomocí dvou nových frameworků Next.js a Meteor.js, které zefektivňují a usnadňují vývoj aplikací. Bude především zpracován jejich význam a porovnání s klasickou tvorbou webových aplikací a budou otestovány výhody a nevýhody jejich použití. V praktické části práce budou vytvořeny dvě izomorfní Single page webové aplikace (SPA) s využitím server-side renderingu (SSR), který oba frameworky využívají. Opět bude následovat porovnání obou použi- tých postupů, zjištěny rozdíly, výhody a nevýhody použití Nextu a Meteoru z praktického pohledu vývojáře. Při tvorbě komplexní aplikace budou také využívány tradiční JavaScriptové frameworky React a Node.js.

Klíčové slova

Single page aplikace (SPA), server-side rendering (SSR), framework, JavaScript, React, Node.js, Next.js, Meteor.js, Blaze

Abstract

The point of the bachelor thesis will be to present and test the principles of creating so-called isomorphic web applications using the new Next.js and Meteor.js frameworks. The basic characteristic of an isomorphic web application is the sharing of code between its server and client side, in this case it is the use of JavaScript not only for the web browser, but also for the server. The implementation of the topic will focus on the creation of isomorphic applications using two new frameworks Next.js and Meteor.js, which streamline and facilitate the development of applications. Their significance and comparison with the classical creation of web applications will be elaborated, and the advantages and disadvantages of their use will be tested. In the practical part of the work, two isomorphic Single page web applications (SPA) will be created using server-side rendering (SSR), which both frameworks use. Again, a comparison of both used procedures will follow, identifying the differences, advantages and disadvantages of using Next and Meteor from a practical developer's point of view. Traditional JavaScript frameworks React and Node.js will also be used when creating a complex application.

Keywords

Single page application (SPA), server-side rendering (SSR), framework, JavaScript, React, Node.js, Next.js, Meteor.js, Blaze

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé bakalářské práce panu PaedDr. Petru Pexovi, Ph.D. za nápomocné rady, ochotu a vedení při zpracování mé závěrečné práce.

Obsah

1 Úvod

Frameworků na tvorbu izomorfních aplikací je celá řada. V mé bakalářské práci se budu zabývat konkrétně frameworky Next.js a Meteor.js.

1.1 Východiska práce

Javascript byl původně jazykem webového prohlížeče a prováděl výpočty a změny až na straně klienta. S příchodem knihovny Node.js se Javascript stal také užitečným serverovým jazykem, který byl tradičně předností jazyků jako PHP, Java, Python. Při vývoji webu je izomorfní aplikace ta, jejíž kód může běžet jak na serverové, tak na klientské straně a vývojáři tedy mohou zpracovávat jeden projekt jak pro server, tak pro klienta a přitom využívat stejných knihoven. Kombinuje tedy výhody single-page aplikací (SPA) a multi-page aplikací (MPA). V typické single-page aplikaci (SPA) je většina logiky aplikace, včetně routingu zapouzdřena v přiloženém souboru Javascriptu, který je poté odeslán klientovi. Tímto krokem se sice uleví serveru, protože nemusí zpracovávat tolik požadavků, ale naopak se zpomalí prvotní načtení a zatížení klienta, protože je potřeba mu poslat celou aplikaci. Výhodu vidím například v rychlosti vykreslení stránky pro klienta. Při žádosti klienta o zobrazení webu se náhled, který se vygeneruje na serveru odešle klientovi. Klient si vyrenderuje náhled okamžitě. Po prvním vykreslení se stáhne kompletní SPA na pozadí a následné akce se řeší na straně klienta.

1.2 Cíle práce

Cílem bakalářské práce bude představit a otestovat principy tvorby tzv. izomorfních webových aplikací s využitím nových frameworků Next.js a Meteor.js. Základní charakteristikou izomorfní webové aplikace je sdílení kódu mezi její serverovou a klientskou stranou, v tomto případě se jedná o použití JavaScriptu nejen pro webový prohlížeč, ale také pro server. Realizace tématu bude zamě-

řena na tvorbu izomorfních aplikací pomocí dvou nových frameworků Next.js a Meteor.js. Bude především zpracován jejich význam a porovnání s klasic- kou tvorbou webových aplikací a otestuji výhody a nevýhody jejich použití. V praktické části práce vytvořím dvě izomorfní Single page webové aplikace (SPA) s využitím server-side renderingu (SSR). Opět bude následovat porovnání obou použitých postupů, zjištěny rozdíly, výhody a nevýhody použití Nextu a Meteoru z praktického po- hledu vývojáře. Při tvorbě komplexní aplikace budou také využívány tradiční JavaScriptové frameworky React a Node.js.

1.3 Metody práce

V úvodu objasním pojmy SPA a MPA, zmíním jejich výhody a nevýhody, porovnám jejich rozdíly použití. Dále se zaměřím na izomorfní aplikace, principy tvorby a jejich význam. Následně představím frameworky Next.js a Meteor. Provedu jejich instalaci. Poznatky využiji při tvorbě vlastní izomorfní SPA. Při tvorbě aplikace budu sledovat jednoduchost, využití nástrojů a čitelnost kódu. Připravím dvě různé aplikace za pomoci výše zmíněných frameworků.

2 Node.js

NodeJS je open-source prostředí, které dovoluje použití JavaScriptu i na straně serveru. Díky tomu je možné psát i backendový kód za použití jednoho jazyka a snížit tím náročnost vývoje. Nejdříve byl vytvořen jako nástroj pro vývoj webových aplikací, později i aplikací na straně serveru. Využívá engine V8 JS, který také pohání prohlížeč Google Chrome a moduly, které rozšiřují původní nástroj o další funkce. Jedná se o jednovláknové řešení, což znamená, že vyžaduje použití asynchronního programování, kdy řešení jednoho požadavku není závislé na dokončení dalšího požadavku. [7][8]

2.1 Rozdíl oproti serverovému jazyku PHP

PHP oproti Node.js provádí všechny příkazy postupně a to přes webový server, který naslouchá na portu a když obdrží požadavek uživatele, zavolá kód PHP. To si převezme dotaz, zabere paměť a neuvolní ji, dokud nejsou provedeny všechny řádky kódu. Není tak škálovatelný jako Node.js, protože není schopen zpracovat tolik souběžných klientů současně. [7][8]

2.2 Použití Node.js

NodeJS je vhodný pro aplikace s vysokým počtem požadavků. Zejména se hodí pro aplikace se změnou v reálném čase, jako je chatovací aplikace nebo streamovací aplikace, jako například Netflix. Běžně se také používá pro vytváření jednostránkových aplikací (SPA) nebo v případě, že musí být server neustále aktivní. Není však vhodný pro projekty, které jsou silně závislé na výkonu procesoru. [7][8]

3 Tvorba webových aplikací

Webové aplikace jsou čím dále častější a nahrazují staré desktopové aplikace. Jejich snadná aktualizace a pohodlnější použití znamenají nejednu výhodu. Mezi dva hlavní vzory při návrhu webových aplikací nejčastěji patří vícestránková aplikace MPA a jednostránková aplikace SPA. Oba principy mají své výhody a nevýhody. [1][12]

3.1 Multi-page aplikace

Multi-page aplikace, zkráceně MPA jsou vícestránkové a fungují běžnějším způsobem. Jakákoliv změna vyžaduje opětovné vykreslení nové stránky ze serveru v prohlížeči. [2][12][13]

MPA jsou mnohem více rozsáhlé než SPA. Vzhledem k obrovskému množství obsahu fungují na různých uživatelských úrovních UI (user interface). Za pomoci AJAXu je zajištěno spolehlivé přenášení dat mezi prohlížečem a serverem. [12][13]

3.1.1 Výhody

Hlavní výhody tohoto principu tvorby ocení především uživatelé, kteří potřebují vizualizaci toho, kam v aplikaci jít. Klade se zde velký důraz na navigaci a víceúrovňové menu, které je základem při tvorbě MPA. Aplikace mají velice dobrou SEO optimalizaci, protože poskytují lepší šanci ohodnotit klíčová slova. [12][13]

3.1.2 Nevýhody

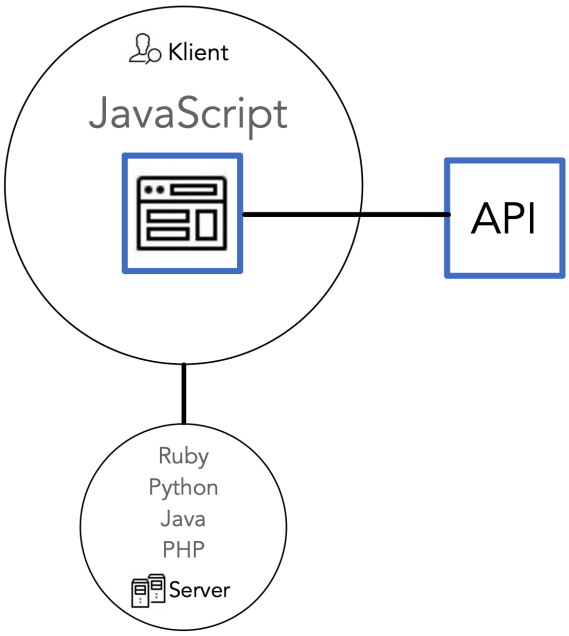
Při tvorbě apliace je nutné využít různé programovací jazyky a od toho se odvíjí i náročnost vývoje. Tvorba je tedy mnohem náročnější, delší a více složitější než u SPA.[12][13]

3.2 Single-page aplikace

Single-page aplikace, zkráceně SPA, je jednostránková aplikace fungující v prohlížeči a při používání nevyžaduje opětovné načítání stránky. S tímto typem aplikací se setkáváme každý den. Ze známějších např. (Gmail, GitHub). [2][11]

SPA jsou především o poskytování kvalitního a propracovaného UX (User expirience) a snaží se o co nejlepší „výkon“ ve smyslu žádného promarněného času a bez opětovného načítání stránky. Jedná se pouze o jednu webovou stánku, na kterou se pak veškerý content(obsah stránky) načte pomocí JavaScriptu, na kterém závisí. Jednostránkové weby pomáhají uživatele udržet ve webovém prostoru, kde se následně prezentuje obsah stránky a to jednoduchým a funkčním způsobem. [2][11]

SPA využívají nejčastěji client-side MVC(Model-View-Controller). [2]



Obrázek 1: Client-side MVC

3.2.1 Výhody

SPA jsou velice rychlé, a to hlavně z důvodu, že většina zdrojů (HTML + CSS + JS) se za celý životní cyklus aplikace načte pouze jednou a dále se přenášejí pouze data. Jejich vývoj je velice zjednodušen, protože pro vykreslování stránek není nutné psát kód na serveru. [12]

Dále dokáží efektivně ukládat do mezipaměti nebo na libovolné lokální úložiště. Pokud dojde k uložení dat, pak je může opětovně použít a dokonce i pracovat offline. [13]

3.2.2 Nevýhody

Dosáhnout SEO optimalizace jednostránkové aplikace je velice složité. Její obsah je načítán pomocí AJAXu (Asynchronous JavaScript a XML), což je metoda, která slouží k výměně dat bez obnovování webové stránky. [2]

Jsou pomalé z důvodu stahování rozsáhlých klientských frameworků, které jsou vyžadovány při načítání na straně klienta. Vyžadují povolení a přítomnost JavaScriptu v prohlížeči. Pokud ho tedy některý uživatel zakáže ve svém prohlížeči, nebude se SPA chovat ani fungovat správným způsobem. [12][13]

Další nevýhodu je nižší bezpečnost, protože díky CSS (Cross-Site Scripting) je umožněno vkládat skripty na straně klienta i ostatními uživately. [2][13]

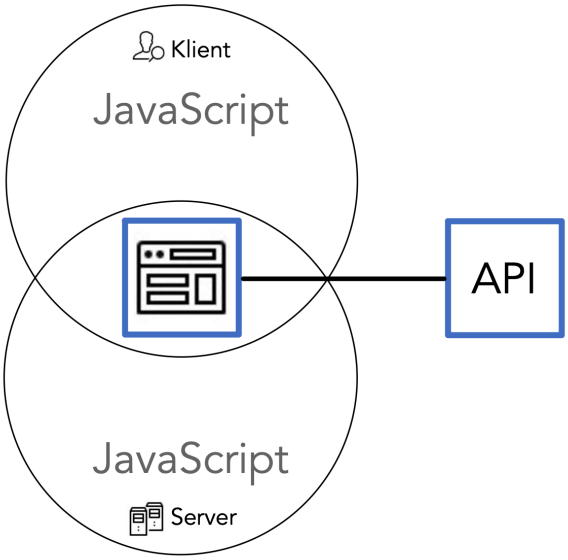
3.3 Izomorfní webové aplikace

Izomorfní webové aplikace sdílejí kód mezi serverem a klientem, to znamená, že obě strany jsou napsané v JavaScriptu. Tento princip vývoje kombinuje výhody SPA a MPA. Oproti single-page aplikacím umožňují rychlejší prvotní vykreslení. Aplikace tedy nevrátí ze serveru pouze čisté HTML, ale již celou vykreslenou úvodní stránku. [1][10]

V dřívější době vše fungovalo mnohem jednodušeji, webový prohlížeč si vyžádal určitou stránku (http://....) , následně server někde na internetu vygeneroval HTML stránku a odeslal ji zpět klientovi. Ačkoliv to fungovalo dobře,

a to hlavně z toho důvodu, že stránky HTML byly statické. [10][15]

V pozdější době se dostával více do hry JavaScript, který umožňoval stránkám, aby byly více dynamičtější a interaktivnější. Nyní je už web plně funkční platformou a JavaScript společně s HTML5 standardem umožňují developerům vyvíjet komplexní a zajímavé aplikace. [1][10]



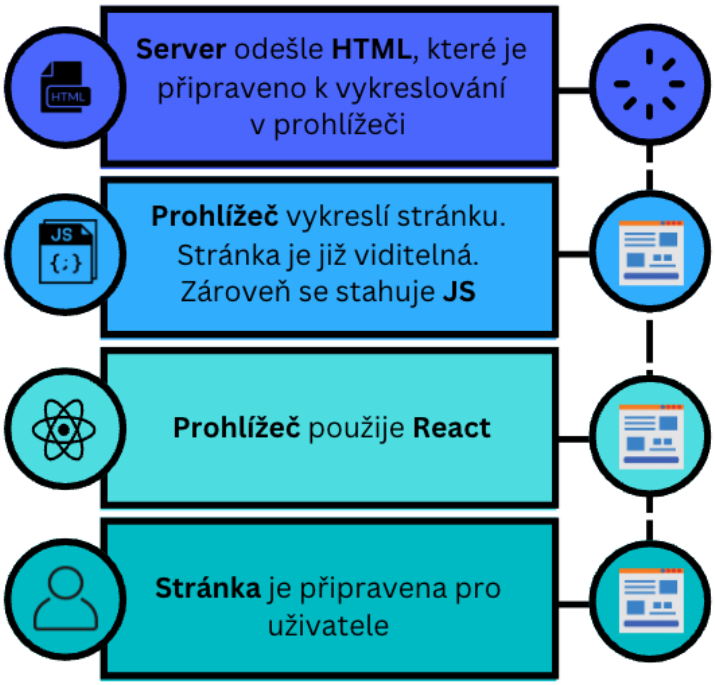
Obrázek 2: Client+server MVC

4 Způsob renderingu webových aplikací

Způsobem renderingu neboli vykreslování rozhodujeme, jak poskytnou prohlížeči webovou aplikaci. [3]

4.1 Server-side rendering (SSR)

Server-side rendering znamená zobrazení a vykreslení webové stránky na straně serveru místo vykreslování až v prohlížeči. Server tedy odešle plně vykreslenou stránku klientovi a nedochází tak k prodlevám při stahování JavaScriptu. Programovací jazyky jako PHP, Python, Ruby vykreslí HTML na serveru před odesláním klientovi do prohlížeče. [3][4]



Obrázek 3: Server-Side Rendering(SSR)

4.1.1 Výhody

SSR poskytuje vývojářům řadu výhod oproti standartnímu React client-side renderingu. Pokud uživatel používá starší webový prohlížeč nebo si zakázal JavaScript, může se stát, že se webové stránky nezobrazí stejně. Při použití metody SSR tomuto zabráníte a aplikace budou dostupné i v tomto případě. Mezi další výhody můžeme zahrnout i bezpečnost. Tím, že se veškeré vykreslování a činosti jako API, správa cookies nebo ověřování dat odehrává na serveru, nevystavíme klientova soukromá data. Dopomůže také ke zlepšení SEO vaší aplikace. Díky tomu, že klient obdrží HTML po vykreslení na serveru, nemusí vyhledávací boti čekat na vykreslení u klienta. [3][4]

4.1.2 Nevýhody

Tento způsob renderingu má i své nevýhody a jeho použití nemusí být vždy ideální volbou pro vaší aplikaci. Například pokud nasazujete aplikaci na vlastní server, tak musíte počítat s větším zatížením serveru a vyššími náklady na jeho údržbu. [4]

4.1.3 Ukázka SSR v Next.js

Ve výchozím nastavení frameworku Next.js se stránka generuje staticky. Pokud ji budeme chtít udělat dynamickou, jako třeba přidat API nebo databázi jako zdroj, budeme muset poté přidat konkrétní funkci.

```
1 function IndexPage(){
2    return<div>Toto je index stranky.</div>;
3 }
4 export default IndexPage;
```

Příklad 1: Ukázka SSR bez dynamiky

Vykreslil se pouze text uvnitř divu, tedy: "Toto je index stránky". Nepotřebovali jsme zavolat žádné externí API k fungovvání této metody. Pokud ale budeme chtít udělat metodu dynamickou například pozdravit uživatele, potřebujeme zavolat REST API na serveru, abychom získali o uživateli konkrétní informace.

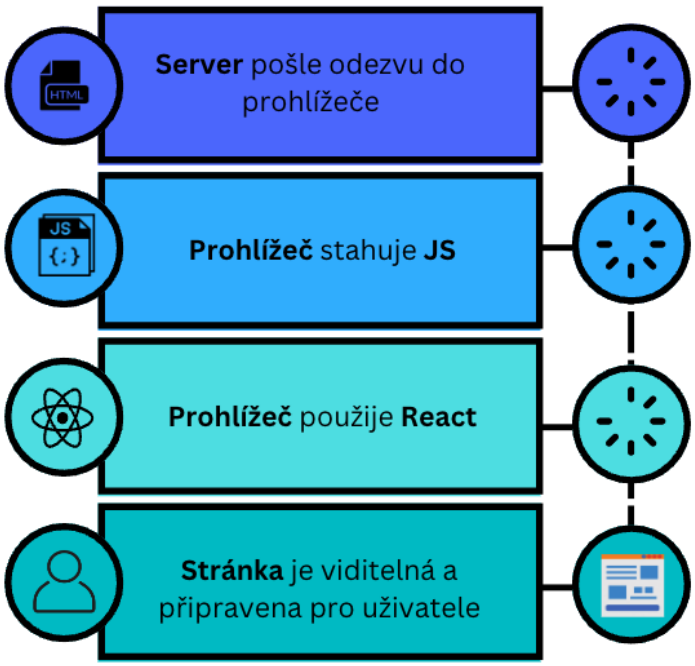
```
export async function getServerSideProps() {
const userRequest =
  await fetch('https://example.com/api/user');
const userData = await userRequest.json();
return {
    props: {
        user: userData
        }
    };
}
function IndexPage(props) {
    return <div>Vitej, {props.user.name}!</div>;
}
export default IndexPage;
```

Příklad 2: Ukázka SSR s dynamikou

Funkce getServerSideProps je vyhrazená v Next.js k volání REST API na serveru. Nejprve exportujeme funkci, která prohledá každou stránku a zajistí, aby se dynamicky vykreslovala pro každý požadavek. Uvnitř funkce vrátíme vlastnost (Props) a vložíme ji do naší komponenty stránky. Dále jen přepíšeme funkci IndexPage. Po odeslání kódu se vždy tato funkce vykreslí dynamicky na serveru.

4.2 Client-side rendering (CSR)

Client-side rendering spočívá v tom, že se JavaScript vykresluje v prohlížeči klienta, ne na straně serveru. To znamená, že je přijat pouze holý HTML dokument s JavaScriptovým souborem. Počáteční načítání stránky je pomalé, ale následně je každé další velice rychlé. Komunikace se serverem probíhá pouze za účelem získávání dat. Není potřeba přenačítat celé UI při každé komunikaci se serverem. CLR používají například knihovny jako React.js a Angular. [3][6]



Obrázek 4: Client-Side Rendering(CSR)

4.2.1 Výhody

Vykreslování na straně klienta (CSR) má pro webové aplikace několik výhod, jako je například lepší odezva. Snižuje také počet požadavků HTTP na server tím, že nevyžaduje opětovné stahování aktiv při každém načtení stránky. CSR navíc umožňuje jasné oddělení zájmů mezi klientem a serverem, což usnadňuje vytváření nových klientů pro různé platformy bez nutnosti úpravy kódu serveru a umožňuje nezávislé změny technologického zásobníku. [4][6]

4.2.2 Nevýhody

CSR má však i několik nevýhod, jako je vyšší počáteční načítání stránky, potřeba konfigurace serveru pro směrování požadavků na jeden vstupní bod. Kromě toho může také negativně ovlivnit SEO optimalizaci, protože vyhledávače mohou vidět prázdný obsah na stránce, když spouštějí JavaScript během procházení. [3][4][6]

5 Next.js

Next.js je open-source JavaScriptový webový framework, vytvořený firmou Vercel, který je navržen speciálně pro React. Dnes je používaný společnostmi jako Netflix, Twitch, TikTok a mnoho dalších.[20] V současné době Next.js nabízí širokou nabídku nových funkcí, včetně generování statického webu, přírůstkového statického generování, nativní podpory TypeScriptu, automatického dělení kódu, optimalizace obrázků nebo také hostingu a publikace zdarma. [20][5]

5.1 Historie

Vývoj webu prošel v posledních letech výraznými změnami. V minulosti bylo vytváření dynamických webových aplikací složité a vyžadovalo mnoho různých knihoven a konfigurací. Moderní rámce JavaScriptu, jako jsou Angular, React a Vue, však tento proces značně zjednodušily a zavedly nové a přinesly inovativní nápady do vývoje webových aplikací. [9][20]

React je open-source JavaScriptová knihovna, která byla vydána v roce 2013 a měla významný dopad na to, jak jsou vytvářeny webové a nativní aplikace. Výsledkem je, že se React stal jednou z nejpoužívanějších a nejoblíbenějších knihoven JavaScriptu s miliony webů, které jej používají pro různé účely. [20][5]

Hlavním problémem Reactu je, že ve výchozím nastavení běží na straně klienta, což může negativně ovlivnit optimalizaci pro vyhledávače (SEO) a počáteční výkon načítání. Prohlížeč si musí stáhnout celý balíček aplikací, analyzovat jeho obsah, spustit jej a vykreslit výsledek v prohlížeči, což u velkých aplikací může trvat několik sekund. [20]

Aby se zlepšil výkon webových aplikací vytvořených pomocí Reactu, začaly se hledat způsoby, jak předběžně vykreslit aplikaci na serveru. To prohlížeči umožňuje zobrazit vykreslenou aplikaci Reactu jako prostý HTML. To zna-

mená, že je interaktivní, jakmile je balíček JavaScript přenesen do klienta. [5][9]

5.2 Alternativy k frameworku Next.js

Framework Next.js není jedinou možností pro vykreslení JavaScriptu na straně serveru. V závislosti na konkrétních potřebách vašeho projektu můžete zvážit použití těchto alternativ.

5.2.1 Nuxt.js

Nuxt.js a Next.js nabízejí velice podobné funkce, jako je vykreslování na straně serveru, generování statických stránek a progresivní správa webových aplikací. Nuxt.js však vyžaduje větší konfiguraci. Nuxt.js je framevorkem jazyka Vue, takže pokud již máte knihovnu komponent Vue, může to být dobrá volba pro vykreslování na straně serveru. Pokud jde o výkon, SEO optimalizaci a rychlost vývoje, mezi nimi nejsou žádné významné rozdíly. Nakonec bude výběr mezi Nuxt.js a Next.js záviset na konkrétních potřebách a preferencích vývojáře. [14][5]

5.2.2 Gastby

Gatsby je populární framework pro vytváření statických webů. Vyniká v generování statických stránek a pro optimální výkon může být poskytován na Content Delivery Network. Nepodporuje však dynamické vykreslování na straně serveru, což může být omezení pro složitější vývoj webových stránek založených na datech. Naproti tomu Next.js tuto funkci nabízí, ale není tak vhodný pro generování statických stránek. [14]

5.2.3 Angular

Angular nabízí podporu pro generování statických stránek i vykreslování na straně serveru a má podporu velké společnosti Google. Pokud již znáte Angular

a máte knihovnu komponent postavenou v Angular, Angular Universal může být dobrou volbou pro vykreslování na straně serveru. Lze jej považovat za konkurenta jiných podobných frameworků jako Nuxt.js a Next.js. [14][5]

5.3 Instalace Next.js

Pokud si chceme nainstalovat framework Next.js je potřeba mít nainstalovaný Node.js. Konkrétně verzi 14.6.0 a vyšší. Podporované jsou všechny operační systémy (Windows, MacOS, Linux). Dále máme na výběr ze dvou možností instalace a to automatickou a manuální. [20]

5.3.1 Automatická instalace

Automatická instalace je mnohem snažší a máte jistotu, že si nainstalujete vše, co je potřeba. Není ani potřeba ani vytvářet prázdnou složku.

```
npx create-next-app@latest <nazev aplikace>
# nebo
yarn create next-app <nazev aplikace>
```

Příklad 3: Automatická instalace next.js [20]

Pokud chcete rovnou implementovat TypeScript stačí ho jen přidat za příkaz.

```
npx create-next-app@latest <nazev aplikace> --typescript
# nebo
yarn create next-app <nazev aplikace> --typescript
```

Příklad 4: Automatická instalace next.js s typescriptem

Poté, co se nám vše nainstaluje, stačí pomocí příkazu aplikaci spustit. Aplikace běží na lokálním serveru na našem zařízení. konkrétně na url: http://localhost:3000.

```
npm run dev
# nebo
```

```
yarn dev
```

Příklad 5: Příkaz pro spuštění aplikace

5.3.2 Manuální instalace

V terminálu přejdeme do složky projektu. Zde musíme nejprve naistalovat next, react a react-dom.

```
npm install next react react-dom
# nebo
yarn add next react react-dom
```

Příklad 6: Manuální instalace [20]

Poté otevřeme package.json a přidáme do něj skripty.

```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start",
  "lint": "next lint"
}
```

Příklad 7: Manuální instalace, přidání skriptů

6 Meteor.js

Meteor je full-stack JavaScriptový framework, který umožňuje tvorbu moderních webových, mobilních a desktopových aplikací. Pro vývoj lze využít vybranou řadou balíčků od Node.js nebo vlastní balíčkovací systém Atmosphere. [17]

6.1 Historie

Meteor se světu poprvé představil v prosinci 2011 pod jiným názvem. V dubnu 2012 se platforma oficiálně přejmenovala na Meteor. Nedlouho po svém vzniku se Meteor stal velice oblíbenou volbou vývojářů a s rostoucí komunitou rostla i jeho popularita. Od roku 2016 existuje Meteor v podobě, v jaké ho známe dnes. Samozžejmě se stále pracuje na jeho zlepšení a rozvoji. [17]

6.2 Principy fungování frameworku Meteor.js

Díky Meteoru můžete použít JavaScript pro vývoj nejen na straně klienta, ale i na straně serveru. Obsahuje vlastní balíčkovací systém nazvaný Atmosphere, kde najdete velké množství užitečných balíčků, které jsou vytvořené speciálně pro tento framework. Kdykoli jsou na serveru nebo klientovi provedeny změny, automaticky se vygeneruje sestavení, což eliminuje potřebu ručního obnovení prohlížeče nebo restartování serveru. [17][18] Meteor má několik integrovaných komponent:

- **Databáze:** Meteor v základu integruje MongoDB jako svou výchozí databázi a také obsahuje Minimongo, odlehčenou verzi MongoDB pro prohlížeč. Tyto databáze se aktualizují v reálném čase, což umožňuje okamžité změny uživatelského rozhraní předtím, než se data dostanou na server. [19]
- **Komunikace:** Klient a server komunikují prostřednictvím protokolu DDP

(Distributed Data Protocol), protokolu RPC (Remote Procedure Call) postaveného na WebSockets namísto HTTP, díky čemuž je Meteor ve výchozím nastavení platformou v reálném čase.[19]

- **Frontend UI frameworky:** Meteor podporuje Blaze, Angular a React jako primární šablonovací motory pro vytváření uživatelského rozhraní. [19]

6.3 Vývoj aplikace

Struktura aplikace v Meteoru je flexibilní a závisí na preferencích vývojáře, jak chtějí soubory organizovat. Jednoduchá aplikace může mít pouze tři soubory pro HTML, CSS a JavaScript. Meteor má specifickou adresářovou strukturu a soubory jsou jednoduše a přehledně uspořádány do složek klienta a serveru. Soubory HTML a CSS se používají pouze na straně klienta. Vzhledem k tomu, že JavaScript se používá na straně klienta i serveru, je nutné určit, které soubory se mají načítat na klienta a které na server. Soubory ve složce klienta jsou určeny pouze pro klienta, soubory ve složce serveru by se měly načítat pouze na server. [18][19]

Občas je potřeba aby nějaká komponenta běžela byla dostupná na obou stranách, ale běžela pouze na jedné. Díky opticky rozděleným adresářům do složek klienta a serveru má pro tento případ definované metody: Meteor.isClient a Meteor.isServer. [18][19]

6.4 Atmosphere nebo npm

Vývoj aplikace od začátku může být zdlouhavý a náročný úkol. Zde přichází jedna z výhod Meteoru, který vám umožní soustředit se na psaní jedinečného kódu pro vaši aplikaci, než ztrácet čas vytvářením základních funkcí, jako je ověřování uživatelů a synchronizace dat. Chcete-li zvýšit svou produktivitu, je dobré využít komunitní balíčky od NPM a Atmosphere. Do verze 1.3 nebylo

NPM vůbec podporované. Nyní ve verzi 2.1 je již plně funkční a můžeme tedy těžit z obou systémů. [17][19]

6.5 Instalace Meteoru.js

Meteor lze nainstalovat na Windows, Linux a také s Mac OS X 10.6 a novější verze. Pouze však 64-bitové verze těchto systémů. Při instalaci na Windows stačí stáhnout oficiální instalátor z hlavní stránky. Instalace Meteoru na systémy Mac a Linux se provádí pomocí příkazového řádku. [17]

Pro instalaci Meteoru na systémech Linux a OS X je použijte tento příkaz:

```
curl https://install.meteor.com/ | sh
```

Příklad 8: Instalace Meteor.js pro systémy Linux a OS X [19]

Pro instalaci Meteoru na systému Windows je použijte tento příkaz (je nutné mít naistalovaná Node.js):

```
npm install -g meteor
```

Příklad 9: Instalace Meteor.js pro systém Windows [17]

Po instalaci stačí vytvořit projekt:

```
meteor create myapp
```

Příklad 10: Vytvoření projektu [19]

Pro spuštění aplikace lokálně (Meteor server běží na http://localhost:3000/):

```
cd myapp
meteor npm install
meteor
```

Příklad 11: Spuštění aplikace lokálně [17]

6.5.1 Kompatibilita

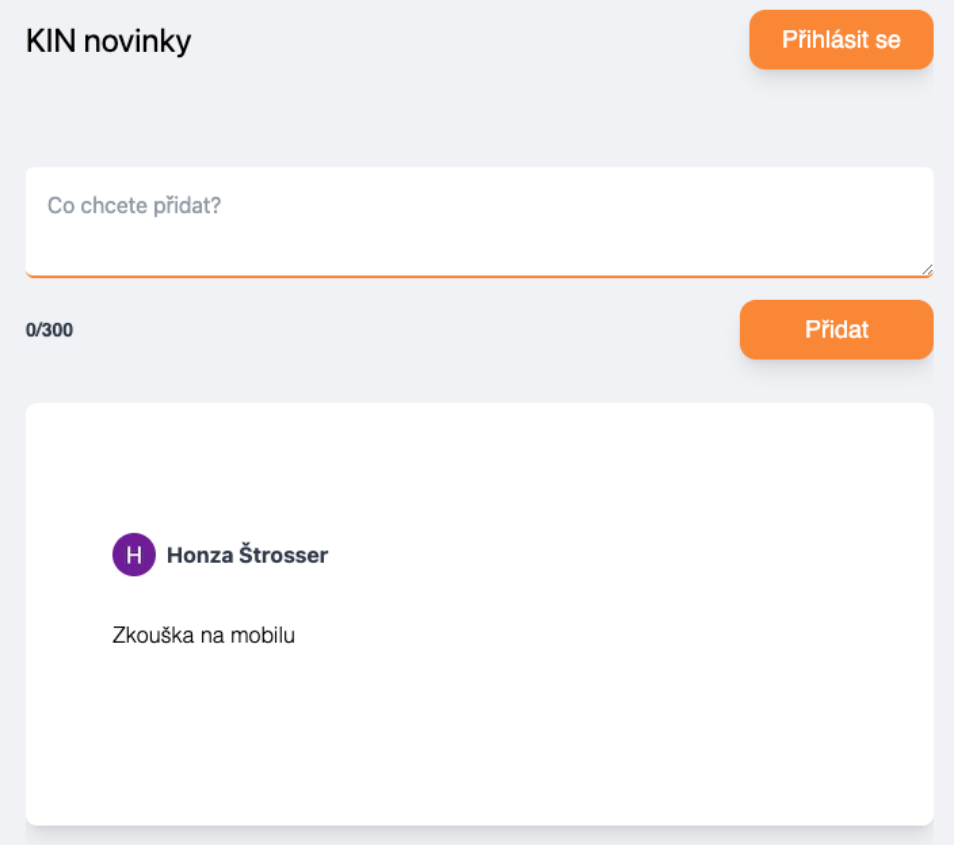
- Pokud používáte produkty od firmy Apple, konkrétně se jedná o zažízení s Mac M1(Arm64 verze) je nutné mít nainstalovanou Rosettu 2, jelikož Meteor využívá databázi MongoDB ke svému běhu.
- Meteor je kompatibilní s verzemi Node.js od 10.0.0. [17]
- Podporovány jsou verze Windows 7 a vyšší. [17]
- Pro vývoj aplikací na IOS je nutné mít nainstalované prostředí Xcode.

7 Praktická část

Nejprve, když jsem psal zadání, jsem myslel, že vytvořím úplně stejné aplikace, ale v průběhu tvorby jsem zjistil, že se oba hodí pro něco jiného. Proto mou praktickou částí této práce jsou dvě různé izomorfní Single-page webové aplikace. Sleduji na nich základní principy tvorby, rozdíly a jednoduchost z praktického pohledu vývojáře. Na konci práce oba frameworky porovnám z pohledu vývojáře.

7.1 Aplikace KIN-Novinky

První aplikaci své praktické části jsem vytvořil za použití frameworku Next.js. Aplikace slouží k přidávání příspěvků uživatelů, které jsou následně ukládány do databáze a vypisovány do uživatelského rozhraní. Všem uživatelům se i bez přihlášení vypisuje stejná kolekce příspěvků. Příspěvek je zobrazen s ikonou uživatele, který ho přidal. Uživatelé také mohou své příspěvky smazat. Přihlášení a ověřování je zabezpečeno Google účtem, který je pro uživatele podmínkou, aby mohl příspěvky přidávat. Zápisy se do databáze ukládají v reálném čase a nedochází tak k prodlevám při jejich vykreslování. Vytvořená aplikace je také plně responzivní. Kompletní zdrojový kód jsem nahrál jako veřejný repozitář na GitHub a je dostupný na adrese: `https://github.com/Strosser/BP_KIN-novinky`.



Obrázek 5: KIN-novinky landing page

Na úvodní stránce se nachází hned několik komponent. Je zde možnost přihlášení, přidání příspěvku, tlačítko přidat a dále se zde vypisují jednotlivé záznamy.

Hlavním účelem této aplikace je ukázat základní principy tvorby full-stack izomorfní aplikace v prostředí frameworku Next.js. Neslouží jako konkurence již vytvořeným aplikacím, ale má sloužit jako návod a studijní materiály pro vývojáře. Mnou vytvořená aplikace je dostuná na adrese: `https://bp-kin-novinky.vercel.app/`.

7.1.1 Použité technologie

Při tvorbě této aplikace byly využity i další služby a technologie, díky kterým se tvorba zjednodušila a zefektivnila. V této části práce zmiňuji všechny tech-

nologie a postupy, které jsem použil, abych mohl vytvořit full-stack izomorfní aplikaci KIN-novinky.

7.1.1.1 Next.js

Next.js je open-source JavaScriptový webový framework, vytvořený firmou Vercel, který je navržen speciálně pro React. [9]

7.1.1.2 React

React je JavaScriptová knihovna pro vytváření uživatelských rozhraní nebo webových aplikací. Umožňuje vývojářům vytvářet opakovaně použitelné komponenty. Protože framework Next.js je navržený pro react, nebyla nutná jeho instalace. Byl již nainstalován v základním projektu. [5]

7.1.1.3 Tailwind CSS

Pro stylizaci a úpravu vzhledu jsem se rozhodl použít framework Tailwind CSS. Nabízí kolekci již definovaných tříd CSS, které lze aplikovat na prvky a vytvářet responzivní a moderní návrhy. Jeho instalace a použizí je velice jednoduché. Stačilo pouze doinstalovat balíček NPM. [24]

```
1 npm install -D tailwindcss postcss autoprefixer
2 npx tailwindcss init -p
```

Příklad 12: Instalace Tailwind CSS

Pomocí těchto příkazů se automaticky vygenerují oba konfigurační soubory: "tailwind.config.js"a "postcss.config.js". [24]

Dále bylo nutné nakonfigurovat cesty k šablonám. Přidal jsem do konfiguračního souboru položky dle dokumentace. [24]

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

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

Příklad 13: Instalace Tailwind CSS

Následně jsem naimportoval Tailwind směrnice do souboru "global.css". [24]

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

Příklad 14: Instalace Tailwind CSS

V takto nakonfigurovaném projektu jsem mohl začít používat Tailwind CSS.

7.1.1.4 PostgreSQL

PostgreSQL je objektově-relační databázový systém. Tuto databázi jsem si vybral, protože jsem s ní již v minulosti pracoval. Databázi jsem si založil pomocí online služby Railway.

7.1.1.5 Railway

Vybral jsem si tuto službu, protože je doporučována firmou Vercel (tvůrce Next.js). Měla jednoduchou konfiguraci s Prismou a frameworkem Next.js. Její výhoda je v tom, že nabízí připravené šablony pro některé databáze. Nemusela

se nijak instalovat, pouze jsem si založil projekt na webu s vybranou databází. [23]

7.1.1.6 Prisma

Služba, která se dotazuje na data z databáze v aplikacích Next.js, je v mém případě PostgreSQL. Jedná se o objektově-relační model pro Typescript a Javascript. Vybral jsem si ji, protože umožňuje server-side rendering. Prismu bylo nutné si doinstalovat jako rozšiřující balíček. [21]

```
npm install prisma --save-dev
```

Příklad 15: Instalace Prisma

```
npx prisma init
```

Příklad 16: Inicializace Prismy v aplikaci

V projektu aplikace KIN-novinky mi Prisma automaticky naimportovala dva soubory:

- schema.prisma: Tvorba a správa databázového schématu.
- .env: dotnet soubor, který definuje URL databázové spojení. Zde jsem nakopíroval ULR adresu své vytvořené PostgreSQL databáze, kterou jsem si vyexportoval z Railway.

Po připojení bylo nutné nakonfigurovat schéma databáze. V dokumentaci Prismy je již základní kostra připravena. Já jsem si schéma pouze doupravil. Změny schématu jsem potvrdil namigrováním nových hodnot na server:

```
migrate dev
```

Příklad 17: Migrace schématu na server [21]

Pro přístup do databáze z Next.js za použití Prismy je nutná instalace prisma klienta.

```
npm install @prisma/client
npx prisma generate
```

Příklad 18: Instalace Prisma clienta [21]

Zkopíroval jsem kód z dokumentace a upravil takto:

```
import { PrismaClient } from "@prisma/client"

declare global {
    namespace NodeJS{
        interface Global {
    }
}}
interface CustomNodeJSGlobal extends NodeJS.Global {
    prisma: PrismaClient;
}
declare const global: CustomNodeJSGlobal

const prisma = global.prisma || new PrismaClient()

if (process.env.NODE_ENV !== "production") global.prisma = prisma
export default prisma
```

Příklad 19: Prisma client [21]

7.1.1.7 NextAuth.js

Protože Next.js nemá integrovanou funkci pro autentifikaci, doinstaloval jsem si balíček NextAuth.js. Jedná se o open-source alternativu přímo pro Next.js.

```
npm install next-auth
```

Příklad 20: Migrace schématu na server

Vytvořil jsem Api route "[...nextauth].js"ve složce /pages/api/auth.

```
import NextAuth from "next-auth"
import GithubProvider from "next-auth/providers/github"
import GoogleProvider from "next-auth/providers/google"
import { PrismaAdapter } from "@next-auth/prisma-adapter"
import prisma from "../../../prisma/client"

const adapter = PrismaAdapter(prisma)

export const authOptions = {
  adapter: adapter,
  secret: process.env.AUTH_SECRET,
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
    }),
    // moznost pridat dalsi
  ],
}

export default NextAuth(authOptions)
```

Příklad 21: Api k ověřování přes Google [22]

Dále jsem si vytvořil dotnet soubor ".env.local", kam jsem nahrál ověřovací klíče od Google API sloužící k přihlášení přes Goolge účet. Uložil jsem je do tohoto souboru z důvodu bezpečnosti. Pokud nahraji projekt, který bude veřejně přístupný nebo si někdo zobrazí zdrojový kód stránky, tento soubor se nikde nezobrazí a zůstane uložený pouze lokálně na mém zařízení.

7.1.1.8 React-hot-toast

Jedná se pouze o kosmetické rozšíření pro React. Díky tomuto balíčku jsem mohl nahradit standartní dialogová okna.

```
npm install react-hot-toast
```

Příklad 22: React-hot-toast

7.1.2 Úvodní stránka

Po otevření aplikace je možné si příspěvky pouze zobrazovat, nikoliv přidávat. Pro přidání příspěvku je nutné se přihlásit pomocí Google účtu.

```
import Link from "next/link"
import Prihlaseni from "./Prihlaseni"
import { getServerSession } from "next-auth/next"
import { authOptions } from "../../pages/api/auth/[...nextauth]"
import Prihlaseny from "./Prihlaseny"

export default async function Nav(){
  const session = await getServerSession(authOptions)
  return(
   <nav className="flex justify-between items-center py-8">
    <Link href={"/"}>
    <h1 className="text-2xl font-poppins">KIN novinky</h1>
    </Link>
    <ul className="flex items-center gap-6">
```

```
15      {!session?.user && <Prihlaseni />}
16      {session?.user && <Prihlaseny image={session.user?.image || ""}
            />}
17     </ul>
18    </nav>
19   )
20 }
```

Příklad 23: Ukázka kódu úvodní stránky

Tento kód importuje komponentu Link z frameworku Next.js a dvě mnou vytvořené komponenty. Dále importuje funkci getServerSession z balíčku next-auth/next a objekt authOptions ze souboru umístěného v pages/api/auth/[...nextauth], kde je zajištěno přihlášení pomocí Google účtu. Použil jsem komponentu Link, kterou Next.js používá jako alternativu k HTML <a> tagům. Tato komponenta obsahuje integrovaný prefetching. Dovoluje tedy Nextu přednačítat stránky, i když je uživatel ještě na té aktuální. Tím se zmenší doba načtení další stránky.

Funkci Nav jsem definoval jako asynchronní funkci, která vrací navigační lištu. Lišta obsahuje název a seznam položek.

Funkce getServerSession slouží k získání informací o uživatelské relaci ze serveru a používá se k tomu objekt authOptions. Získaný objekt session se poté používá k renderování obsahu navigační lišty.

Pokud objekt session neobsahuje vlastnost "user", zobrazí se komponenta Prihlaseni, která zobrazuje tlačítko pro přihlášení. Naopak pokud objekt session obsahuje vlastnost "user", místo toho se zobrazí komponenta Prihlaseny, která zobrazuje profilový obrázek, aby indikovala, že je uživatel přihlášen a mohl již příspěvky přidávat.

7.1.3 Přihlášení v aplikaci KIN-novinky

Abychom se mohli přihlásit, bylo nutné doinstalování balíčku NextAuth.js. Implementaci jsem popsal v kapitole použitých technologií.

```
1 import { signIn } from "next-auth/react"
2
3 export default function Prihlaseni(){
4   return (
5         <li className="list-none">
6           <button onClick={() => signIn()} className="text-lg
                bg-orange-400 shadow-lg hover:bg-orange-500 text-white
                py-2 px-6 rounded-xl">Prihlsit se</button>
7         </li>
8       )
9 }
```

Příklad 24: Ukázka kódu komponenty Přihlášení

Nejprve jsem si musel naimporovat funkci "signIn"z modulu "next-auth/react". Poté jsem definoval tlačítko, na kterém je "onClick"událost, která zajišťuje, že se po kliknutí na tlačítko zavolá funkce "signIn". Díky této fukci proběhne přesměrování na přihlášení.

Pokud proběhne přihlášení úspěšně, zavolá se komponenta "Prihlaseny", kde jsem změnil tlačítko na možnost odhlášení a implementovanou metodu na "signOut". Přidal jsem zde také funkci na zobrazení svých příspěvků. Na seznam svých příspěvků se dostaneme tak, že klikneme na svůj profilový obrázek.

7.1.4 Příspěvek

Nejprve jsem si vytvořil samotný příspěvek. To zahrnovalo vzhled a jaké parametry se uvnitř budou zobrazovat.

```
import Image from "next/image"
import Link from "next/link"

export default function Prispevek({ avatar, name, postTitle, id }:
    Prispevek){
  return (
      <div className="bg-white my-8 p-8 rounded-lg">
        <div className="flex items-center gap-2">
          <Image
            className="rounded-full" width={32} height={32}
                src={avatar} alt="avatar"
          />
          <h3 className="font-bold text-gray-700">{name}</h3>
        </div>
        <div className="my-8">
          <p className="break-all">{postTitle}</p>
        </div>
        <div className="flex gap-4 cursor-pointer items-center">
          <Link
            href={{
              pathname: `/post/${id}`,
            }}
          >
          </Link>
        </div>
      </div> )}
```

Příklad 25: Ukázka kódu komponenty Příspěvek - zkrácená verze

Nejdříve jsem si definoval očekávané parametry. Zde jsem také použil komponentu Image z frameworku Next.js, která slouží ke zobrazení profilového obrázku. Komponenta Link zabalí název příspěvku a nastaví trasu na straně klienta na stránku jednotlivých příspěvků pomocí funkce dynamického směrování Next.js.

7.1.5 Přidání příspěvku

Pro komponentu sloužící k přidání příspěvku bylo nutné si vytvořit vlastní Api, které bude komunikovat se serverem. Proto zde vytvořím dvě kategorie pro klientskou a serverovou stranu, aby ukázky byly přehlednější.

7.1.5.1 Přidání příspěvku front-end

Komponentu spravuji pomocí useState. Title obsahuje zadání uživatele pro název příspěvku, zatímco isDisabled se používá k deaktivaci tlačítka odeslání formuláře během vytváření příspěvku, aby nedocházelo k několikanásobnému klikání na tlačítko přidání. Tím jsem zabránil zahlcení serveru.

UseMutation hook jsem použil k vytvoření nového příspěvku. Funkce odešle požadavek POST na koncový bod /api/posts/pridatPrispevek s názvem příspěvku jako tělem požadavku. Zpětná volání onError a onSuccess jsem využil ke zpracování chyb a úspěšných odpovědí.

Funkce potvrditPrispevek je volána při odeslání formuláře. Zabrání výchozímu chování při odesílání formuláře, nastaví hodnotu isDisabled na hodnotu true a zavolá funkci mutate k vytvoření příspěvku s aktuálním stavem názvu jako názvem příspěvku.

```
export default function VytvoritPost() {
    const [title, setTitle] = useState('')
    const [isDisabled, setIsDisdabled] = useState(false)
    const queryClient = useQueryClient()

    const { mutate } = useMutation(
        async (title: string) =>
            await axios.post("/api/posts/pridatPrispevek", {
                title,
            }),
        {
            onError: (error) => {
                if (error instanceof AxiosError) {
                    toast.error(error?.response?.data.message)
                } setIsDisdabled(false)
            },
            onSuccess: (data) => {
                queryClient.invalidateQueries(["posts"]),
                toast.success('Pspvek byl pidn'), setTitle(""),
                setIsDisdabled(false) },
        })
    const potvrditPrispevek = async (e: React.FormEvent) => {
        e.preventDefault()
        setIsDisdabled(true)
        mutate(title)
    }
```

Příklad 26: Ukázka kódu komponenty pro přidání příspěvku - zkrácená verze

Následně jsem definoval uživatelské rozhraní formuláře, včetně textové oblasti pro název příspěvku, počítadla znaků a tlačítka pro odeslání. To je deaktivováno, když je stav isDisabled true. Po odeslání formuláře je zavolána funkce potvrditPrispevek. Počet znaků je omezený na 300, poté příspěvek nelze přidat. Přidal jsem i optickou změnu barvy, díky které upozorním uživatele na překročení počtu znaků.

```
return (
       <form onSubmit={potvrditPrispevek} className="my-8 rounded-md">
           <div className="flex flex-col gap-2 my-8rounded-lg">
               <textarea onChange={(e) => setTitle(e.target.value)}
                   name="tittle" value={title}
                   placeholder="Co chcete pridat?" className="p-4
                       text-gl rounded-md my-2 bg-white border-b-2
                       border-orange-400"></textarea>
               <div className="flex items-center justify-between">
                   <p className={`font-bold text-sm ${title.length >
                       300 ? "text-red-700" : "text-gray-700"
                       } `}>{`${title.length}/300`}</p>
                   <button disabled={isDisabled} className="text-lg
                       bg-orange-400 shadow-lg hover:bg-orange-500
                       text-white py-2 px-12 rounded-xl
                       disabled:opacity-25"
                       type="submit">Pridat</button>
               </div>
           </div>

       </form>
   )
```

Příklad 27: Ukázka kódu komponenty pro přidání příspěvku - UI

7.1.5.2 Přidání příspěvku back-end

Zde jsem vytvořil API pro komunikaci se serverem a ošetřil výjimky,d které mohou nastat.

Pracuji zde s požadavky POST do koncového bodu API. Funkce nejprve zkontroluje, zda je metoda požadavku POST, a poté použije funkci getServerSession ke kontrole, zda je uživatel ověřen. Pokud uživatel není ověřen, funkce vrátí odpověď s chybovou zprávou.

Pokud je uživatel ověřen, funkce využije Prisma ORM k vyhledání údajů o uživateli z databáze pomocí jeho e-mailové adresy. Pokud uživatel není v databázi nalezen dostanu zpět chybovou zprávou.

Následně ověřuji, zda není příspěvek moc dlouhý nebo naopak prázdný. Vše jsem ošetřil případným upozorněním.

Pokud je vše v pořádku, funkce vytvoří nový příspěvek pomocí metody vytvoření Prisma ORM a ID uživatele. Pokud je příspěvek úspěšně vytvořen, funkce vrátí odpověď s nově vytvořenými daty příspěvku a dialogové okno s úspěchem.

```
1  import type { NextApiRequest, NextApiResponse } from "next";
2  import { getServerSession } from "next-auth/next"
3  import { authOptions } from "../auth/[...nextauth]"
4  import prisma from "../../../prisma/client";
5
6  export default async function handler(
7      req: NextApiRequest,
8      res: NextApiResponse
9  ){
10     if (req.method === "POST") {
11         const session = await getServerSession(req, res, authOptions)
12         if (!session) return res.status(401).json({ message: "Pokud
               chcete pridat prispevek musite se prihlsit" })
13 //kontrola tittle
```

```
14        if (title.length > 300) return res.status(403).json({ message:
              "Prispevek je moc dlouhy" })
15        if (!title.length) return res.status(403).json({ message:
              "Prispevek je prazdny"})
16        try {
17            const vysledek = await prisma.post.create({data: {
18                title: title.toString(), userId: prismaUser.id,
19            }, })
20        console.log(vysledek)
21        res.status(200).json(vysledek)
22        }catch(err) {
23            res.status(403).json({ err: "Error"})
24        }}}
```

Příklad 28: Ukázka kódu komponenty pro přidání příspěvku - API

7.1.6 Zobrazení příspěvku

Pro zobrazení příspěvků jsem vytvořil asynchronní funkci s názvem fetchPrispevky, která vrací pole objektů s rozhraním PostsType. Funkce načte seznam příspěvků z koncového bodu API /api/posts/zobrazitPrispevek pomocí Axiosu a vrátí je jako pole objektů typu PostsType.

```
1  const fetchPrispevky = async (): Promise<PostsType[]> => {
2    const response = await axios.get("/api/posts/zobrazitPrispevek")
3    return response.data
4  }
5
6  interface PostsType {
7    title: string
8    id: string
9    createdAt?: string
10   user: User
```

```
}
```

Příklad 29: Ukázka kódu komponenty pro zobrazení příspěvku

Bylo nutné také vytvořit koncový bod API. Zde jsem použil dva argumenty (NextApiRequest, NextApiResponse) což jsou objekty představující příchozí požadavek a odchozí odpověď. Funkce vrací data z databáze jako odpověď na požadavek GET.

```

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse)
  {
  if (req.method === "GET") {
    try {
      const data = await prisma.post.findMany({
        include: {
          user: true
        },
        orderBy: {
          createdAt: "desc",
        },
      })
      return res.status(200).json(data)
    } catch (err) {
      res.status(403).json({ err: "Error" })
    }}}
```

Příklad 30: Ukázka kódu API pro zobrazení příspěvku

7.1.7 Smazání příspěvku

Pro odstranění příspěvku jsem napsal komponentu s názvem UpravaPrispevek. Uvnitř komponenty je stavová proměnná toggle, která je výchozím nastavením false. Tato proměnná slouží k přepínání zobrazení do potvrzovacího okna. K aktualizaci hodnoty toggle jsem použil funkci setToggle.

Pro zpracování mazání příspěvku jsem použil hook useMutation z knihovny react-query. Tento hook definuje funkci mutate, která pomocí knihovny Axios odesílá DELETE požadavek na API endpoint. Také jsem použil hook useQueryClient k vytvoření proměnné queryClient, která se používá k invalidaci dotazu, když je příspěvek úspěšně smazán.

Hook useMutation také definuje funkci onSuccess, která se zavolá po úspěšném provedení mutace. Použil jsem funkci toast z knihovny react-toastify ke zobrazení zprávy o úspěchu a invalidoval jsem dotaz auth-posts v queryClient.

Nakonec jsem definoval funkci smazatPrispevek pomocí funkce mutate vrácené hookem useMutation. Tato funkce bere jako vstupní hodnotu prop ID a volá funkci mutace s touto hodnotou k odeslání DELETE požadavku na API endpoint a smazání příspěvku se zadaným ID.

Přidal jsem také tlačítko "Smazat", které nás po kliknutí přepne do potvrzovacího okna. Po kliknutí na toto tlačítko se zavolá onClick event handler s dvěma akcemi. Nejprve se zastaví propagace události pomocí metody e.stopPropagation(). Následně se zavolá funkce potvrzeni s hodnotou true, aby se přepnulo zobrazení potvrzovacího okna.

Nakonec komponenta podmíněně zobrazuje komponentu PotvrzeniSmazani pomocí stavové proměnné toggle. Komponenta PotvrzeniSmazani bere dvě props, smazatPrispevek a potvrzeni, které jsou funkci předány z rodičovské komponenty a slouží k ovládání akcí pro smazání a potvrzení.

```
export default function UpravaPrispevek({ avatar, name, title, id }:
    EditProps) {
  const queryClient = useQueryClient()
  const [toggle, potvrzeni] = useState(false)
  let smazatToastID: string
  //smazat
  const { mutate } = useMutation(
    async (id: string) => await
        axios.delete('/api/posts/smazatPrispevek', { data: id }), {
    onSuccess: (data) => {
      toast.success("  Pspvek   byl    spn     smazn ", { id:
          smazatToastID })
      queryClient.invalidateQueries(["auth-posts"]) }})
  const smazatPrispevek = () => {mutate(id) }
  return ( <div className="flex items-center gap-2">
         <Image className="rounded-full" width={32} height={32}
             src={avatar} alt="avatar" />
         <h3 className="font-bold text-gray-700">{name}</h3>
       </div>
       <div className="my-8 "><p
           className="break-all">{title}</p></div>
       <div className="flex items-center gap-4 ">
         <button
           onClick={(e) => { e.stopPropagation() potvrzeni(true)}}
           className="text-sm font-bold text-red-500">
           Smazat</button></div>
     {toggle && <PotvrzeniSmazani smazatPrispevek={smazatPrispevek}
         potvrzeni={potvrzeni} />})}
```

Příklad 31: Ukázka kódu smazání příspěvku - zkrácená verze

```
export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse){

  if (req.method === "DELETE")
  {const session = await getServerSession(req, res, authOptions)
    if (!session) return res.status(401).json({ message: "Prosm,
        pihlate se." })

    try {
      const postID = req.body
      const vysledek = await prisma.post.delete({where:{id: postID}})
      res.status(200).json(vysledek)
    } catch (err) {
      res.status(403).json({ err: "Error" })
    }}}
```

Příklad 32: Ukázka kódu smazání příspěvku - API

Tento kód je Next.js API route handler, který ovládá DELETE request pro smazání příspěvku.

Nejprve se kontroluje, zda je požadavek DELETE. Pokud ano, tak se získá session pomocí getServerSession funkce z knihovny next-auth a nastaví se authOptions. Pokud session neexistuje, server vrátí chybový kód s hláškou "Prosím, přihlašte se."

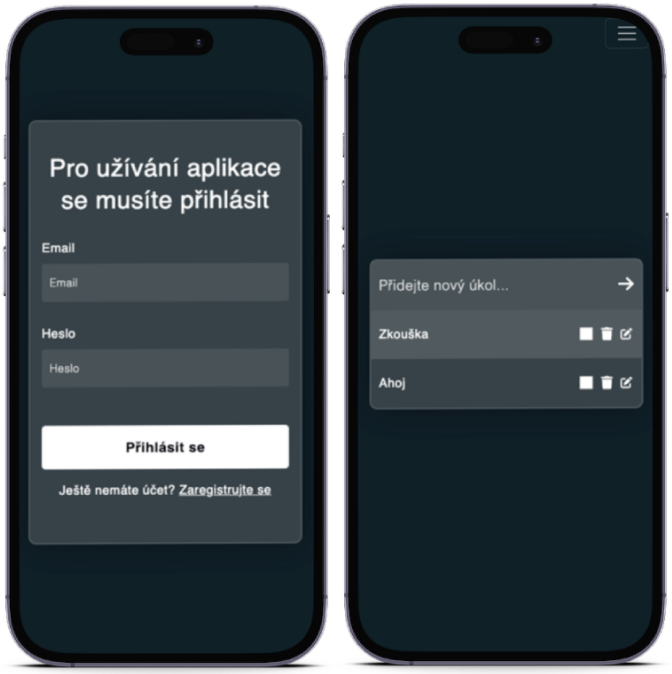
Pokud je session platná, tak se provede DELETE operace pro smazání příspěvku z databáze pomocí Prisma ORM. Poté se vrátí úspěšná odpověď s výsledkem smazání příspěvku. Pokud se operace nepodaří, server vrátí chybový kód s hláškou "Error".

7.1.8 Deployment aplikace KIN-novinky

Pro publikaci aplikace jsem se rozhodl využít hosting od firmy Vercel. Je zdarma a jelikož je od stejné firmy tak publikace byla velice jednoduchá. Nejprve jsem nahrál svůj projekt na GitHub. Po registraci na portálu Vercel, stačil projekt přidat a vybrat k publikaci. Pokud bylo vše bez chyb tak se projekt sestaví a je online. Pokud narazíte na nějaký problém, je nutné aplikaci opravit a znovu ji nahrát na GitHub. Poté je postup stejný.

7.2 Aplikace Úkolníček

Jako druhou aplikaci ze své praktické části jsem vytvořil za pomoci frameworku Meteor.js. Aplikace slouží ke vkládání úkolů přihlášených uživatelů, které jsou následně ukládány do databáze a vypisovány do uživatelského rozhraní. Uživatel se musí pro použití aplikace přihlásit. Uživatelé také mají možnost své úkoly dokončovat, mazat a upravovat. Přihlášení a ověřování je zabezpečeno pomocí funkcí Meteoru. Zápisy se do databáze ukládají v reálném čase a nedochází tak k prodlevě při vykreslování. Díky plné responzivitě je zajištěna stabilní mobilní verze aplikace. Kompletní zdrojový kód jsem nahrál jako veřejný repozitář na GitHub a je dostupný na adrese: `https://github.com/Strosser/BP-ukolnicek`.



Obrázek 6: Ukázka aplikace Úkolníček

Na úvodní stránce nás aplikace vyzve k přihlášení. Pokud účet založený nemáme, musíme se pro používání aplikace nejprve zaregistrovat. Při registraci vyplníme email, uživatelské jméno a heslo.

Po přihlášení se vypíší uložené úkoly z databáze přiřazené k našemu účtu. Zde jsem přidal možnosti přidat, dokončit, upravit nebo odstranit úkoly.

7.2.1 Použité technologie

Při tvorbě této aplikace byly především využity funkce frameworku Meteor.js. Pro tvorbu jsem se rozhodl využít JavaScriptovou knihovnu Blaze, kterou meteor nativně nabízí.

7.2.1.1 Blaze

Původně byl meteor navržený pro tento jazyk, proto jsem se rozhodl pro jeho použití. Jedná se o JavaScriptovou knihovnu, která využívá tvorbu HTML šablon (template) a kombinuje je s JavaScriptem. Blaze se nemusí nijak instalovat.

Je zahrnut při instalaci Meteoru a vytváření projektu.

7.2.1.2 MongoDB

Jako databázi jsem zvolil MongoDB, protože je také zahrnutá v instalaci meteoru. Její ovládání mi přišlo velice intuitivní a jednoduché. Data z této databáze jsem si zobrazoval pomocí aplikace MongoDB Compass.

7.2.1.3 Bootstrap

Pro stylizování aplikace jsem se rozhodl využít CSS framework Bootstrap. Zvolil jsem ho pro jeho použití, protože jsem s ním již v minulosti několikrát pracoval. Nabízí předpřipravené styly, které se přidávají jako třídy k jednotlivým elementům.

7.2.1.4 FlowRouter

FlowRouter je balíček pro klientovské routování v Meteoru, který umožňuje snadnou navigaci mezi stránkami nebo zobrazení ve webové aplikaci. Je navržen pro práci s knihovnou Blaze, která je šablonovacím enginem běžně používaným s Meteorem.

7.2.2 Inicializace databáze

Vytvořil jsem novou MongoDB kolekci s názvem "todo-list"pomocí zápisu "export const TodoList = new Mongo.Collection('todo-list')". Tato kolekce slouží k ukládání dat o úkolech v aplikaci. Každý dokument v této kolekci bude reprezentovat jeden konkrétní úkol a bude obsahovat informace jako text úkolu, stav úkolu (dokončený/nedokončený) a datum vytvoření.

```
1 export const TodoList = new Mongo.Collection('todo-list')
```

Příklad 33: Ukázka kódu inicializace databáze MongoDB

7.2.3 Přihlášení v aplikaci Úkolníček

Jako úvodní stránku jsem vytvořil přihlašovací formulář. Pro přihlášení je nutné zadat email a heslo, které je uloženo v databázi. Pokud nemáme vytvořený účet, vložil jsem zde odkaz na komponentu ‘register‘, pomocí které je možné si účet vytvořit.

```
<template name="login">
    <form id="loginForm" class="auth-form glass-container">
        <h3>Pro uzivani aplikace se musite prihlasit</h3>
        <label for="email">Email</label>
        <input type="email" placeholder="Email" id="email" required>
        <label for="password">Heslo</label>
        <input type="password" placeholder="Heslo" id="password"
            required>
        <button type="submit">Prihlasit se</button>
        <p class="text-center mt-3">Jeste nemate ucet? <a
            href="{{urlFor 'register'}}">Zaregistrujte se</a></p>
    </form>
</template>
```

Příklad 34: Ukázka kódu přihlášení v aplikaci Úkolníček

Zde jsem vytvořil komponentu, která slouží k ověření identity uživatele pomocí e-mailu a hesla a která umožňuje uživateli přístup k aplikaci v případě, že ověření proběhne úspěšně.

Poté se získají hodnoty e-mailu a hesla z příslušných polí formuláře. Funkce "toLowerCase()"převede e-mail na malá písmena a funkce "trim()"odstraní případné mezery na začátku a konci řetězce.

Poté se zobrazí načítací symbol pomocí funkce "showLoader()". Pomocí funkce "Meteor.loginWithPassword()"ověřuji uživatelské jméno a heslo. Pokud ověření selže, zobrazí se chybová hláška. Pokud ověření proběhne úspěšně, uživatel bude přesměrován na stránku s názvem "todoList"pomocí funkce

"FlowRouter.go('todoList')".

```
Template.login.events( {
    'submit #loginForm'(event, template) {
        event.preventDefault();
        const email = $('#email').val()?.toLowerCase()?.trim();
        const password = $('#password').val();
        showLoader();
        Meteor.loginWithPassword({email}, password, (err) => {
            hideLoader();
            if(err){
                showMessage('Zadali jste nespravne heslo');
            } else {
                showMessage('Prihlasen uspesne!', 'success');
                FlowRouter.go('todoList');
            }
        })
    }
} )
```

Příklad 35: Ukázka kódu funkcionality přihlášení

7.2.4 Registrace v aplikaci Úkolníček

Stejně jako u přihlášení je zde možnost registrace. Vyplníme zde email, uživatelské jméno a heslo. Následně se můžeme zaregistrovat.

```
<template name="register">
    <form id="registerForm" class="auth-form glass-container">
        <h3>Registrace</h3>

        <label for="username">Uzivatelsk jmno</label>
        <input type="text" placeholder="Uzivatelsk jmno" id="username"
            required>
```

```
7
8           <label for="email">Email</label>
9           <input type="email" placeholder="Email" id="email" required>
10
11          <label for="password">Heslo</label>
12          <input type="password" placeholder="Heslo" id="password"
                required>
13
14          <button type="submit">Zaregistrovat se</button>
15
16          <p class="text-center mt-3">Uz mate ucet? <a href="{{urlFor
                'login'}}">Prihlaste se</a></p>
17      </form>
18  </template>
```

Příklad 36: Ukázka kódu registrace v aplikaci Úkolníček

V následujícím příkladu jsem vytvořil komponentu, která slouží k registraci nového uživatele. Funkce "toLowerCase()"převede hodnoty na malá písmena a funkce "trim()"odstraní případné mezery na začátku a konci řetězce.

Poté se zobrazí načítací symbol pomocí funkce "showLoader()". Následuje zavolání metody "registerUser"pomocí funkce Meteor.call(). Tato metoda se nachází na serveru a vytvoří nového uživatele v databázi. Pokud registrace proběhne úspěšně, uživatel bude přesměrován na stránku přihlášení pomocí funkce "FlowRouter.go('login')".

```
1  Template.register.events( {
2      'submit #registerForm'(event, template) {
3          event.preventDefault();
4          const data = {
5              username: $('#username').val()?.trim(),
6              email: $('#email').val()?.toLowerCase()?.trim(),
7              password: $('#password').val()
```

```
8         }
9
10        showLoader();
11        Meteor.call('registerUser', data, (err, result) => {
12            hideLoader();
13            if(err) {
14                showMessage(err.reason, 'error')
15            } else {
16                showMessage(`Registrace probehla uspesne,\nNiny se
                      muzete prihlasit!`, 'success');
17                FlowRouter.go('login');
18            }
19        })
20    }
21 } )
```

Příklad 37: Ukázka kódu funkcionality registrace

7.2.5 Autentifikace uživatele

Pro ověření uživatele jsem vytvořil Meteor metodu nazvanou "registerUser", která bere jako argument objekt "data". Pro ověření toho, že objekt obsahuje požadované vlastnosti, jsem použil funkci "check"a "Match"z balíčku meteor/-check.

Jakmile se data ověří, vezmu vlastnosti "username", "email"a "password"z objektu "data"a použiji je k vytvoření nového uživatelského účtu pomocí metody "Accounts.createUser"poskytované balíčkem accounts-base. Poté vrátím ID nového uživatele klientovi.

Pokud během ověřování dat nebo vytváření uživatele dojde k chybě, použiji blok "try...catch"k odchycení chyby a vyhození nové chyby Meteor.Error s odpovídajícím kódem chyby a zprávou. Toto zajišťuje správné zpracování a

zobrazení chyb uživateli.

```
import { check, Match } from 'meteor/check'

Meteor.methods({
   registerUser( data ) {
       try {
           check(data, Match.ObjectIncluding({
               username: String,
               email: String,
               password: String
           }))
           const { username, email, password } = data;
           const userData = {
               email,
               password,
               profile: {
                   username
               } }
           const user = Accounts.createUser( userData );
           if(user) {
               return user;
           }
       } catch ( err ) {
           throw new Meteor.Error(err.error, err.reason ||
               err.message);
       }})
```

Příklad 38: Ukázka kódu metody pro ověřování uživatele

7.2.6 Přidání úkolu

Tato funkce se stará o přidání nové položky do seznamu úkolů. Pokud se v poli 'add-todo' nachází neprázdná hodnota, funkce zobrazí ikonu nahrávání pomocí funkce showLoader() a provede Meteor.call(), což je asynchronní metoda, která přidá novou úlohu.

Pokud se vyskytne chyba, funkce zobrazí chybové hlášení pomocí showMessage(). Pokud se nová úloha úspěšně přidá, funkce vymaže hodnotu v poli 'add-todo' a nastaví editační ID úlohy na null.

```
Template.todoList.events({
    'keypress #add-todo, click .save-todo'(event, template) {
        if((event.type === "keypress" && event.which === 13) ||
            (event.type === 'click')) {
            const value = $('#add-todo').val()?.trim();
            if(value) {
                showLoader();
                Meteor.call('addTodoMethod', value,
                    template.editTaskId.get(), (err, result) => {
                    hideLoader();
                    if(err) {
                        showMessage(err.reason, 'error');
                        return;
                    }
                    $('#add-todo').val('');
                    template.editTaskId.set(null);
                })
            }
        }
    }
```

Příklad 39: Ukázka kódu přidání položky do seznamu úkolů

7.2.7 Metody na práci s úkoly

Pro práci s úkoly jsem vytvořil celkem tři metody: dokončit, upravit a smazat. Postupně se budu věnovat jejich vytvoření.

7.2.7.1 Metoda dokončeno

Při změně stavu úkolu, tj. zaškrtnutí nebo odškrtnutí, zavolám Meteor metodu checkTask, kde předám ID úkolu a jeho nový stav. Pokud se při volání metody checkTask vyskytne chyba, zobrazím uživateli chybovou hlášku a vrátím původní stav úkolu (zaškrtnutý/odškrtnutý). Pokud vše proběhne v pořádku, zobrazím uživateli úspěšnou hlášku s novým stavem úkolu (dokončený/nezahájený).

```
'change .check-task'(event, template) {
       const isChecked = $(event.currentTarget).prop('checked');
       showLoader();
       Meteor.call('checkTask', this._id, isChecked, (err) => {
           hideLoader();
           if(err) {
               showMessage(err.reason, 'error');
               $(event.currentTarget).prop('checked', !isChecked);
           } else {
               showMessage(`Ukol oznacen jako dokonceny! ${isChecked
                   ? CONSTANTS.TODO_STATUS.COMPLETED :
                   CONSTANTS.TODO_STATUS.PENDING}!`, 'success');
           }
       })
   }
```

Příklad 40: Ukázka kódu metody dokončeno

7.2.7.2 Metoda upravit

Když uživatel klikne na tlačítko editace úkolu, zavolal jsem metodu template.editTaskId.set() s parametrem ID tohoto úkolu, abych později věděl, který úkol uživatel upravuje. Dále jsem nastavil hodnotu textového pole add-todo na text daného úkolu, aby uživatel viděl aktuální text, který upravuje.

```
'click .editTask'(event, template) {
      template.editTaskId.set(this._id);
      $('#add-todo').val(this.text);
  }
```

Příklad 41: Ukázka kódu metody upravit

7.2.7.3 Metoda smazat

Pokud uživatel klikne na tlačítko "removeTask", spustil jsem funkci, která odesílá Meteor metodu "removeTask"spolu s ID úkolu, který má být odstraněn. Pokud server vrátí chybu, zobrazil jsem uživateli chybovou zprávu. V opačném případě se zobrazí úspěšné hlášení o odstranění úkolu.

```
'click .removeTask'(event, template) {
      showLoader();
      Meteor.call('removeTask', this._id, (err, result) => {
          hideLoader();
          if(err) {
              showMessage(err.reason, 'error');
          } else {
              showMessage('Ukol odstrann uspesne!', 'success');
          }
      })
  }
```

Příklad 42: Ukázka kódu metody smazat

7.2.7.4 FlowRouter

Pro navigaci jsem vytvořil cestu pomocí FlowRouteru, který reaguje na URL '/todo-list'. Při načtení této URL zkontroluji, zda je uživatel přihlášen pomocí Meteor.userId(). Pokud ano, použiji metodu import() pro dynamické načtení modulu obsahujícího šablonu pro todo list. Poté vložím tuto šablonu do layoutu pomocí BlazeLayout.render(). Pokud uživatel není přihlášen, použiji metodu FlowRouter.go('login') k přesměrování na stránku přihlášení. Těchto cest jsem vytvořil několik. Pro příklad zde uvádím přesměrování do úkolníčku po přihlášení.

```
FlowRouter.route('/todo-list', {
  name: 'todoList',
  action() {
    if(Meteor.userId()) {
      import('../custom/todo-list/client/todo-list').then(() => {
        BlazeLayout.render('layout', {
            main: 'todoList'
        })
      })
    } else {
      FlowRouter.go('login')
    }
  }
});
```

Příklad 43: Ukázka kódu FlowRouteru - zkrácený verze

7.2.7.5 Deployment aplikace Úkolníček

Meteor také nabízí možnost publikace aplikace zdarma na vlastní servery. Postup byl stejný, jako při publikaci předchozí aplikace. Jen bylo nutné provést přihlášení na portálu Meteor up. Přišel jsem tím na velikou nevýhodu. Poté, co

jsem aplikaci nahrál, běžela bez problémů, ale při delší nečinnosti se zastavila, aby nezatěžovala servery. Její start opět proběhl až po pokusu o zobrazení a delší prodlevě. Musí se čekat na to, než se aplikace znovu na serveru nastartuje.

7.3 Porovnání frameworků Next.js a Meteor.js

Osobně jsem měl s oběma frameworky pozitivní zkušenosti. Při tvorbě menších projektů bych se spíše uchýlil k použití Next.js, protože mi poskytl rychlejší a jednodušší způsob tvorby webových aplikací. Naopak při větších projektech bych volil spíše Meteor, protože se více hodí pro tvorbu kompletních webových aplikací s mnoha funkcemi a integracemi bez nutnosti doinstalování rozšiřujících balíčků.

Next.js je React-based framework, který nabízí mnoho výhod, jako je například rychlost a efektivita, která zlepšuje celkovou výkonost aplikace. Jeho statické generování stránek může být velmi užitečné pro webové stránky, které se zřídka aktualizují, protože umožňuje jejich rychlejší načítání. Další výhodou je snadná integrace s Reactem a TypeScriptem, což usnadňuje tvorbu uživatelského rozhraní.

Na druhé straně Meteor.js je full-stack JavaScriptový framework, který poskytuje kompletní sadu nástrojů a knihoven pro vývoj webových aplikací v reálném čase. Nabízí integrovanou sadu technologií, včetně front-endového frameworku, back-endového frameworku a databáze. Meteor.js nabízí funkci synchronizace dat v reálném čase, která umožňuje bezproblémový přenos dat mezi serverem a klientem. Zaujal mne svou jednoduchostí a snadnou použitelností.

Pokud jde o zkušenost z vývoje, oba tyto frameworky nabízejí skvělou zkušenost pro vývojáře s jednoduše použitelnými API a rozsáhlou dokumentací. Next.js nabízí intuitivní systém routování na základě souborů a přehlednou adresářovou strukturu, což usnadňuje orientaci. Meteor.js nabízí jednotné API pro vývoj na straně klienta a serveru, což zjednodušuje vývojový proces a

snižuje úroveň náročnosti.

Na počátku své práce jsem si myslel, že Next.js bude o mnoho lepší pro vývoj izomorfních aplikací, ale postupem času jsem zjišťoval, že to není pravda. Dle mého názoru se pro vývoj full-stack izomorfních aplikací hodí více Meteor. Díky tomu, že není nutná instalace několika rozšíření k dosažení kompletní aplikace. Celkově mi tvorba aplikace ve frameworku Next.js trvala přibližně třikrát déle než za pomoci Meteoru. To bylo zapříčiněno především hledáním alternativních technologií, které mi pro tvorbu full-stack aplikace chyběly.

Co se týká nasazení, Next.js nabízí vestavěnou podporu pro generování statických stránek a volby pro serverless nasazení, což usnadňuje nasazení na platformy jako Vercel. Meteor.js nabízí vestavěný nástroj pro nasazení zvaný Meteor Up, který jsem já využil. Také zjednodušuje proces nasazení a poskytuje podporu pro různé poskytovatele hostingů.

Musím ale zmínit, že oba frameworky mají svá specifika a své výhody. Volba mezi nimi závisí na konkrétních potřebách projektu.

8 Závěr

Bakalářská práce byla zaměřena na tvorbu izomorfních webových aplikací ve dvou frameworcích a to Next.js a Meteor.js. Hlavním znakem izomorfní aplikace je tvorba serverové a klientské části za pomoci stejného jazyka. Tento přístup se mi osobně velice zamlouval, protože jsem se nemusel učit více technologí kvůli několika komponentám.

Naopak si myslím, že ne pokaždé je výhodnější psát aplikace tímto způsobem. V jazycích jako PHP, Java nebo Python, které jsou na serverový jazyk specializované, často zjistíte, že existuje mnohem jednodušší řešení.

V teoretické části své bakalářské práce jsem představil principy tvorby webových aplikací spolu s jejich výhodami a nevýhodami. Popsal jsem frameworky Next.js a Meteor.js. Zpracoval jsem jejich význam a otestoval výhody a nevýhody jejich použití.

Výsledkem praktické části práce jsou dvě izomorfní Single-page webové aplikace vytvořené za pomoci obou výše zmíněných frameworcích. Další částí je porovnání z praktického hlediska.

Celkově mě tvorba izomorfních webových aplikací velice zaujala a doufám, že se bude v budoucnu dočkáme více specializovaných frameworkům na serverovou část.

Seznam použité literatury a zdrojů

[1] *Isomorphic JavaScript: The Future of Web Apps | by AirbnbEng | The Airbnb Tech Blog | Medium. Medium – Where good ideas find you.* [online]. Dostupný z WWW:< <https://medium.com/airbnb-engineering/isomorphic-javascript-the-future-of-web-apps-10882b7a2ebc>>.

[2] *Single-page application vs. multiple-page application | by Neoteric | Medium. Medium – Where good ideas find you.* [online]. Dostupný z WWW:< < https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58>>.

[3] *Server-Side Rendering VS. Client-Side Rendering. Build Offshore Technology Team in India. In No Time* [online]. Dostupný z WWW:< < https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58>>.

[4] *Client-side Rendering. Patterns.dev - Modern Web App Design Patterns* [online]. Copyright © 2020 [cit. 07.02.2023]. Dostupný z WWW:< <https://www.patterns.dev/posts/client-side-rendering/>>.

[5] *Next js vs React: Comparing Two Popular Frontend Frameworks. Software Development Company | Business IT Solutions* [online]. Copyright © 2023 SPEC INDIA. All Rights Reserved. [cit. 07.02.2023]. Dostupný z WWW:< <https://www.spec-india.com/blog/nextjs-vs-react>>.

[6] *Client-side vs. server-side rendering: why it's not all black and white.* [online]. Dostupný z WWW: <https://www.freecodecamp.org/news/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d/>.

[7] *Proč k vývoji webových aplikací použít technologii NodeJS?. WEB MOBILE DEVELOPMENT AGENCY / Rascasone* [online]. Copyright © [cit. 07.02.2023]. Dostupný z WWW:< <https://www.rascasone.com/cs/blog/node-js-architektura-moduly-npm>>.

[8] *PHP Vs. Node.js, Explained. HubSpot Blog / Marketing, Sales, Agency, and Customer Success Content* [online]. [online]. Copyright © 2023 HubSpot, Inc. [cit. 11.04.2023]. Dostupný z WWW:< <https://blog.hubspot.com/website/php-vs-node-js>.

[9] *Next.js by Vercel - The React Framework. Next.js by Vercel - The React Framework* [online]. Copyright © [cit. 07.02.2023]. Dostupný z WWW:< <https://nextjs.org/>>.

[10] *Psát isomorfní webové aplikace? » phpFashion. phpFashion* [online]. Copyright © 2004, 2023 [cit. 07.02.2023]. Dostupný z WWW:< <https://phpfashion.com/psat-isomorfni-webove-aplikace>>.

[11] *Co je jednostránková webová aplikace (SPA) a kdy ji využít?. WEB MOBILE DEVELOPMENT AGENCY / Rascasone* [online].Copyright © [cit. 07.02.2023]. Dostupný z WWW:< <https://www.rascasone.com/cs/blog/jednostrankova-webova-aplikace-spa>>.

[12] *Vývoj webových aplikací: single-page vs. multi-page aplikace. WEB MOBILE DEVELOPMENT AGENCY / Rascasone* [online]. Copyright © [cit. 07.02.2023]. Dostupný z WWW:< <https://www.rascasone.com/cs/blog/jednostrankove-vicestrankove-web-aplikacespa-vs-mpa-kterou-si-vybrat>>.

[13] *Single-Page Application vs Multi-Page Application: Pros, Cons, and Which is Better?. Lvivity – Custom Software Development Company*

[online]. Copyright © 2013 [cit. 07.02.2023]. Dostupný z WWW:< <https://lvivity.com/single-page-app-vs-multi-page-app>>.

[14] *Next.js x Gatsby? Který framework zvolit pro React aplikaci. WEB MOBILE DEVELOPMENT AGENCY / Rascasone* [online]. Copyright © [cit. 07.02.2023]. Dostupný z WWW:< < https://www.rascasone.com/cs/blog/vyvoj-webovych-aplikaci-next-gatsby>>.

[15] *Co je IMA.js? Podívejme se na framework od Seznam.cz - Zdroják. Zdroják - o tvorbě webových stránek a aplikací* [online]. Dostupný z WWW:< <https://zdrojak.cz/clanky/co-je-ima-js-podivejme-se-na-framework-od-seznam-cz/>>.

[16] *React – A JavaScript library for building user interfaces. React – A JavaScript library for building user interfaces* [online].Copyright © 2023 Meta Platforms, Inc. [cit. 07.02.2023]. Dostupný z WWW:< < https://reactjs.org/>>.

[17] *Meteor.js* Introduction | Meteor Guide. Introduction | Meteor Guide [online]. Dostupný z WWW:< <https://guide.meteor.com/>>.

[18] Quick Guide to MeteorJS – What it Is, and Who Should Use it. [online]. Dostupný z WWW:< <https://www.freecodecamp.org/news/what-is-meteorjs-and-who-should-use-it/>>.

[19] Meteor Tutorial. [online]. Copyright © Copyright 2023. All Rights Reserved. [cit. 11.04.2023]. Dostupný z WWW:< <https://www.tutorialspoint.com/meteor/index.htm>>.

[20] *Next.js* Introduction | Learn Next.js. Next.js by Vercel - The React Framework[online]. Copyright © [cit. 11.02.2023]. Dostupný z WWW:< <https://nextjs.org/learn/foundations/about-nextjs>>.

[21] *Quickstart with TypeScript SQLite* Prisma | Next-generation ORM for Node.js TypeScript [online]. Copyright © [cit. 11.04.2023]. Dostupný z WWW:< <https://www.prisma.io/docs/getting-started/quickstart>>.

[22] *Getting Started | NextAuth.js* NextAuth.js [online]. Copyright © Iain Collins 2023 [cit. 11.04.2023]. Dostupný z WWW:< <https://next-auth.js.org/getting-started/example>>.

[23] *Railway Docs* Railway Docs [online]. Dostupný z WWW:< <https://docs.railway.app/>>.

[24] *Install Tailwind CSS with Next.js - Tailwind CSS* Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. [online]. Copyright © [cit. 11.04.2023]. Dostupný z WWW:< <https://tailwindcss.com/docs/guides/nextjs>>.

Seznam obrázků

Seznam příkladů

A Příloha

CD/DVD obsahuje plné znění mé bakalářské práce s názvem BP-Strosser.pdf a soubor Aplikace.pdf obsahující odkazy na repozitáře se zdrojovými kódy obou aplikací.

B Příloha

Aplikace KIN-Novinky: `https://bp-kin-novinky.vercel.app/`

Aplikace Úkolníček: `https://bp-ukolnicek.meteorapp.com/login`

GitHub repozitář se zdrojovými kódy aplikace KIN-Novinky: `https://github.com/Strosser/BP_KIN-novinky`

GitHub repozitář se zdrojovými kódy aplikace Úkolníček: `https://github.com/Strosser/BP-ukolnicek`