Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Využití front-end knihovny ReactJS pro vytvoření moderní webové aplikace

Artem Yanenko

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Artem Yanenko

Informatika

Název práce

Využití front-end knihovny ReactJS pro vytvoření moderní webové aplikace

Název anglicky

Usage of the ReactJS front-end library for a modern web application creation

---

Cíle práce

Hlavním cílem bakalářské práce je vytvoření moderní webové aplikace za použití front-end knihovny ReactJS. Cíle teoretické části práce zahrnují vypracování literární rešerše dostupných odborných zdrojů, podle které bude provedená charakteristika knihovny ReactJS a souvisejících s ní termínů. Mezi cíle praktické části patří návrh a vývoj webové aplikace a otestování její nasazení v praxi.

Metodika

Metodika teoretické části bakalářské práce je založena na studiu dostupných vědeckých informačních zdrojů. V praktické části práce budou provedeny návrh a implementace webové aplikace pro sdílené sledování videí platformy YouTube. Nejprve bude uskutečněna analýza základních charakteristik aplikace podle kterých bude realizován výběr nástrojů, technologií a knihoven vhodných pro vývoj moderního webu. Následně bude provedena samotná implementace webové aplikace za použití ReactJS. Na základě výsledků praktické části budou určeny závěry bakalářské práce.

Doporučený rozsah práce

30-40

Klíčová slova

ReactJS, JavaScript, Redux, TypeScript, Webová aplikace, Vývoj aplikace

---

Doporučené zdroje informací

Despoudis, Theo. TypeScript 4 Design Patterns and Best Practices : Discover Effective Techniques and Design Patterns for Every Programming Task, Packt Publishing, Limited, 2021. ISBN 9781800565418

Elrom, Elad. React and Libraries : Your Complete Guide to the React Ecosystem, Apress L. P., 2021. ISBN 1843733968

Ferguson, Russ. Beginning JavaScript : The Ultimate Guide to Modern JavaScript Development, Apress L. P., 2019. ISBN 9781484243954

Minnick, Chris. Beginning ReactJS Foundations Building User Interfaces with ReactJS : An Approachable Guide, John Wiley & Sons, Incorporated, 2022. ISBN 1479152594

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

Ing. Martin Pelikán, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 20. 2. 2023

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 2. 3. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 02. 03. 2023

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Využití front-end knihovny ReactJS pro vytvoření moderní webové aplikace“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury i dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne datum odevzdání _____

Poděkování

Rád bych touto cestou poděkoval Ing. Martinu Pelikánovi, Ph.D., za vedení své diplomové práce a konzultace.

Využití front-end knihovny ReactJS pro vytvoření moderní webové aplikace

Abstrakt

Tato bakalářská práce je věnována vývoji moderní webové aplikace s funkcionalitou umožňující synchronně sledovat videa z YouTube společně v prohlížeči. V teoretické části jsou charakterizovány technologie a vzory, které jsou široce využívány pro tvorbu škálovatelných moderních webových aplikací. Kromě toho jsou v teoretické části také zařazeny charakteristika a definice programovacích jazyků JavaScript a TypeScript i jejich knihoven a balíčků, které jsou užitečné pro vývoj realizovaný v praktické části práce. Hlavní část diplomové práce popisuje proces vývoje jednotlivých částí aplikace od signalizačního serveru až po byznysovou logiku a uživatelské rozhraní klientské aplikace. V závěru práce jsou analyzovány klady a zápory zvolených vývojových přístupů a návrhových voleb spolu s návrhem možných vylepšení pro dosažení lepší škálovatelnosti projektu.

Klíčová slova: ReactJS, JavaScript, Redux, TypeScript, webová aplikace, vývoj aplikace, WebSocket, WebRTC, monorepo, Lerna

Usage of the ReactJS front-end library for a modern web application creation

Abstract

This thesis is dedicated to the development of the modern web application with functionality allowing to synchronously watch YouTube videos together in the browser. In the theoretical part there are characterized the technologies and patterns that are widely used to build scalable modern web applications. The theoretical part also describes the characteristics and definition of the JavaScript and TypeScript programming languages as well as its libraries and packages that are useful for development in the practical part of the thesis. The main part of the diploma thesis describes the process of development for every part of the application from signalling server to business logic and user interface of the client application. Pros and cons of the chosen development approaches and design choices are analysed in the summary of the thesis together with suggestion of potential improvements to achieve better scalability of the project.

Keywords: ReactJS, JavaScript, Redux, TypeScript, Web Application, Application Development, WebSocket, WebRTC, Monorepo, Lerna

Obsah

Seznam obrázků

Seznam použitých zkratek

WebRTC – Web Real-Time Communication

ReactJS – React JavaScript

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

ECMA – European Computer Manufacturers Association

ES2015 – ECMA Script 2015

API – Application programming interface

CI – Continuous integration

CD – Continuous delivery

IDE – Integrated development environment

MVC – Model-View-Controller

DOM – Document Object Model

XML – Extensible Markup Language

JSX – JavaScript XML

URL – Uniform Resource Locator

HTTP – Hypertext Transfer Protocol

IoT – The Internet of Things

ANSI – American National Standards Institute

LRU – Least Recently Used

POSIX – Portable Operating System Interface

BSD – Berkeley Software Distribution

OS – Operating system

HTTPS – Hypertext Transfer Protocol Secure

UI – User interface

UX – User experience

NPM – Node Package Manager

CRUD – Create, read, update and delete

JSON – JavaScript Object Notation

JWT – JSON Web Token

ID – Identification

ICE – The Information and Content Exchange

1 Úvod

Rychlý pokrok v oblasti technologií umožnil vzájemné propojení lidem ze všech koutů světa. Jednou z nejoblíbenějších aktivit mezi přáteli je společné sledování videí, a se vznikem online platforem, jako je YouTube, se možnosti sdílení zážitků ze sledování ještě zvýšily. Tato bakalářská práce si klade za cíl prozkoumat vytvoření webové aplikace, která uživatelům umožní sledovat videa na YouTube společně s přáteli bez ohledu na jejich fyzickou polohu. Aplikace bude využívat bezserverový přístup a protokol WebRTC pro bezproblémové a bezpečné spojení mezi uživateli. Cílem této práce je vyvinout uživatelsky přívětivou a spolehlivou webovou aplikaci pro sdílené zážitky ze sledování YouTube. Tento projekt se snaží přispět tím, že poskytuje praktické řešení problematiky socializace na dálku. Použitím nejnovějších řešení ve webových technologiích tento projekt si klade za cíl vytvořit robustní aplikaci, která usnadní online skupinové sledování videostreamů. Kromě toho bude aplikace uživatelsky přívětivá a snadno ovladatelná, což uživatelům umožní vychutnat si sdílený zážitek ze sledování bez jakýchkoli technických potíží.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této bakalářské práce je vytvoření moderní webové aplikace za použití front-end knihovny ReactJS. Cíle teoretické části práce zahrnují vypracování literární rešerše dostupných odborných zdrojů, podle které bude provedena charakteristika knihovny ReactJS a s ní souvisejících termínů. Mezi cíle praktické části patří návrh a vývoj webové aplikace i otestování jejího nasazení v praxi.

2.2 Metodika

Metodika teoretické části této bakalářské práce je založena na studiu dostupných vědeckých informačních zdrojů. V praktické části práce budou provedeny návrh a implementace webové aplikace pro sdílené sledování videí platformy YouTube. Nejprve bude uskutečněna analýza základních charakteristik aplikace, podle kterých bude realizován výběr nástrojů, technologií a knihoven vhodných pro vývoj moderního webu. Následně bude provedena samotná implementace webové aplikace za použití ReactJS. Na základě výsledků praktické části budou vyvozeny závěry bakalářské práce.

3 Teoretická východiska

3.1 Webová aplikace

Webová aplikace je počítačový program, který je uložen na vzdáleném serveru, doručován přes internet prostřednictvím rozhraní prohlížeče a využívá webové technologie k provádění úkolů přes internet. Aplikace daného typu není nutné stahovat, protože jsou přístupné prostřednictvím sítě. Uživatelé k nim mohou přistupovat prostřednictvím webových prohlížečů, jako jsou Google Chrome, Mozilla Firefox nebo Safari. Tyto aplikace používají kombinaci skriptů na straně serveru ke zpracování, ukládání a získávání informací a skriptů na straně klienta k prezentaci informací uživatelům. Pro jejich fungování jsou vyžadovány webový server pro správu požadavků od klienta, aplikační server pro provádění požadovaných úkolů, a někdy i databáze pro ukládání informací [1] [2]

Webové aplikace jsou obvykle kódovány v jazyce podporovaném prohlížečem, jako jsou JavaScript a HTML, protože tyto jazyky spoléhají na spuštění programu prohlížečem. Některé aplikace jsou dynamické, a proto je pro jejich funkčnost nutné zpracování na straně serveru, jiné jsou naopak zcela statické, bez nutnosti zpracování na serveru. Takto vypadá typický tok webových aplikací:

1. Uživatel spustí požadavek na webový server přes internet, buď prostřednictvím webového prohlížeče, nebo uživatelského rozhraní aplikace.
2. Webový aplikační server provede požadovaný úkol a vygeneruje výsledky požadovaných dat.
3. Webový aplikační server odešle na webový server výsledky.
4. Webový server odpoví klientovi požadovanou informací, která se zobrazí na displeji uživatele [2].

3.2 JavaScript/TypeScript

3.2.1 JavaScript

JavaScript představuje multiplatformní, objektově orientovaný skriptovací jazyk, který byl v roce 1995 vyvinut Brendanem Eichem a následně představen jako LiveScript společností Netscape Communications Corp. JavaScript byl původně vytvořen pro zvýšení interaktivity webových stránek a kontrolu jejich chování. Programy JavaScript jsou běžně vkládány do souboru HTML, který představuje značkovací jazyk a neovlivňuje chování

stránky po jejím načtení. Použití JavaScriptu pak umožnuje nastavovat pravidla a ověřovat, zda byla dodržena. JavaScript se dnes nepoužívá pouze pro základní ověření vstupu, ale k přístupu k objektu *Document* v prohlížeči, k asynchronním voláním na webový server a k vývoji komplexních webových aplikací pomocí softwarových platforem, jako je například Node.JS. Webový prohlížeč je nejběžnějším prostředím, ve kterém se uplatňuje JavaScript, ale není jediný. Pomocí JavaScriptu lze vytvářet všechny druhy widgetů, rozšíření aplikací a další části softwaru. Zároveň se tento programovací jazyk dá využít nejen pro web, jeho programy mohou běžet i v hostitelském prostředí [3] [4].

JavaScript je považován za jeden ze tří stavebních kamenů, které jsou nutné k vytváření interaktivních webových stránek. Je to jediný programovací jazyk v trojici, kterou tvoří HTML, CSS a JavaScript. Jinými slovy, JavaScript je lepidlo, díky kterému vše funguje dohromady, a díky němu lze vytvářet bohaté webové aplikace. Jazyk JavaScript, pracující v tandemu se souvisejícími funkcemi prohlížeče, je technologie vylepšující web. Při použití na klientském počítači může jazyk pomoci přeměnit statickou stránku obsahu na poutavé, interaktivní a inteligentní prostředí [5].

3.2.2 TypeScript

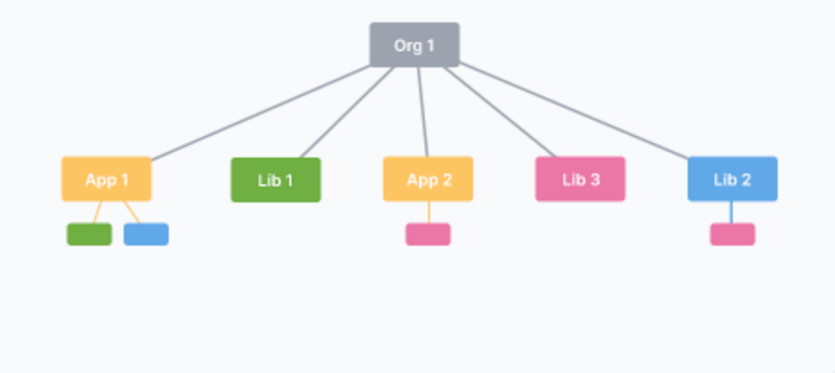
Přestože JavaScript představuje přirozené řešení pro multiplatformní modely a umí spustit téměř všechny moderní prohlížeče, je bohužel stále závislý na znalostech složitých vzorů a ladění kódu se musí provádět za jeho běhu. Aktuálně prohlížeče používají odlišné javascriptové enginy, každý z nich využívá jinou implementaci ES2015 a rychlost přidávání nových funkcí zároveň se může lišit. Proto vytváření velkých aplikací v JavaScriptu může být obtížné, stejně jako strukturování kódu do udržovatelné formy [6].

TypeScript představuje open-source jazykový projekt založený v roce 2012 společností Microsoft, který si klade za cíl posunout vývoj JavaScriptu na další úroveň. Je to silně typovaný kompilovaný programovací jazyk, který tvoří nadstavbu nad JavaScript a poskytuje lepší nástroje v jakémkoli měřítku tím, že přidává do JavaScriptu dodatečnou syntaxi pro podporu těsnější integrace s kódovým editorem. TypeScript má vysokou kompatibilitu se stávajícím kódem JavaScript a je nadmnožinou JavaScriptu, což znamená, že jakýkoli platný javascriptový program je také platným typescriptovým programem. TypeScript staticky identifikujte konstrukce JavaScriptu, které jsou pravděpodobně chybné, a umožňuje identifikovat potenciální problémy s běhovým prostředím a předcházet jim tím, že provádí statickou kontrolu typu v době kompilace. Kromě toho je vygenerovaný kód

vysoce kompatibilní s webovými prohlížeči, protože ve výchozím nastavení cílí na specifikaci ECMAScript 3, ale podporuje také ECMAScript 5 a ECMAScript 6. Většina dalších funkcí TypeScriptu je založena na budoucích návrzích ECMAScriptu, což znamená, že mnoho souborů TypeScriptu se nakonec stane platnými soubory JavaScriptu [7] [8] [9].

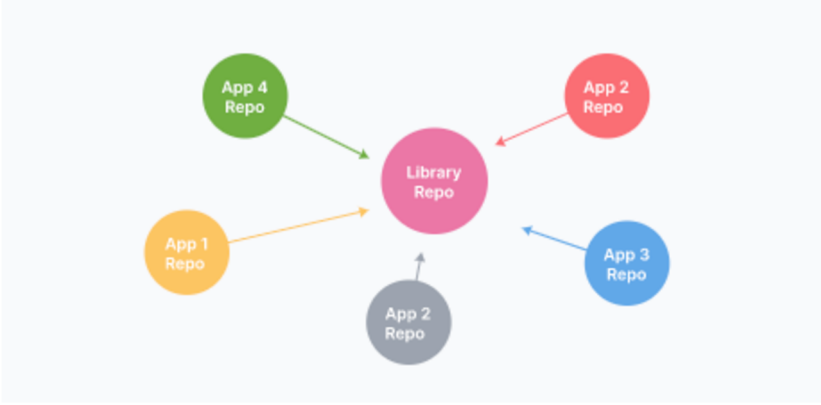
3.3 Monorepo

V systémech pro správu verzí je monorepo („*mono*“ znamená „jediné“ a „*repo*“ je zkratka pro „úložiště“) strategie vývoje softwaru, ve které je kód pro řadu projektů uchováván ve stejném úložišti. I když tyto projekty spolu mohou souviset, často jsou logicky nezávislé. Na obrázku č. 1 je znázorněn příklad struktury monorepo [10].



Obrázek 1: Grafické znázornění struktury monorepo. Zdroj: [11]

Opakem monorepa je polyrepo, kde je každý projekt uchováván na zcela samostatném úložišti s kontrolou verzí. Polyrepo je současný standardní způsob vývoje aplikací, ve kterém existuje separátní repozitář pro každý tým, aplikaci nebo projekt. Bývá běžné, že každý repozitář má jeden artefakt sestavení a jednoduchý postup sestavení. Příklad struktury multirepo je znázorněn na obrázku č. 2 [11].



Obrázek 2: Grafické znázornění struktury polyrepo. Zdroj: [11]

Mezi výhody použití monorepa patří:

1. **viditelnost** (každý uživatel vidí kód všech ostatních přispěvatelů),
2. **jednodušší správa závislostí** (sdílení závislostí je triviální, není potřeba správce balíčků, protože všechny moduly jsou umístěny ve stejném úložišti),
3. **jediný zdroj správné verze** (jedna verze každé závislosti znamená, že neexistují konflikty při verzování),
4. **konzistence** (prosazování standardů kvality kódu a jednotného stylu je snazší, když veškerá kódová základna je na jednom místě),
5. **sdílená časová osa** (přelomové změny v API nebo sdílených knihovnách jsou odhaleny okamžitě, což nutí různé týmy komunikovat dopředu),
6. **atomic commits** (ty usnadňují refaktorování ve velkém měřítku, a proto uživatel může aktualizovat několik balíčků nebo projektů v jednom potvrzení),
7. **implicitní CI** (nepřetržitá integrace je zaručena, protože veškerý kód je již sjednocen na jednom místě),
8. **sjednocené CI/CD** (pro každý projekt v repu lze použít stejný proces nasazení CI/CD),
9. **jednotný proces sestavení** (je možné použít sdílený proces sestavení pro každou aplikaci v úložišti) [11] [12].

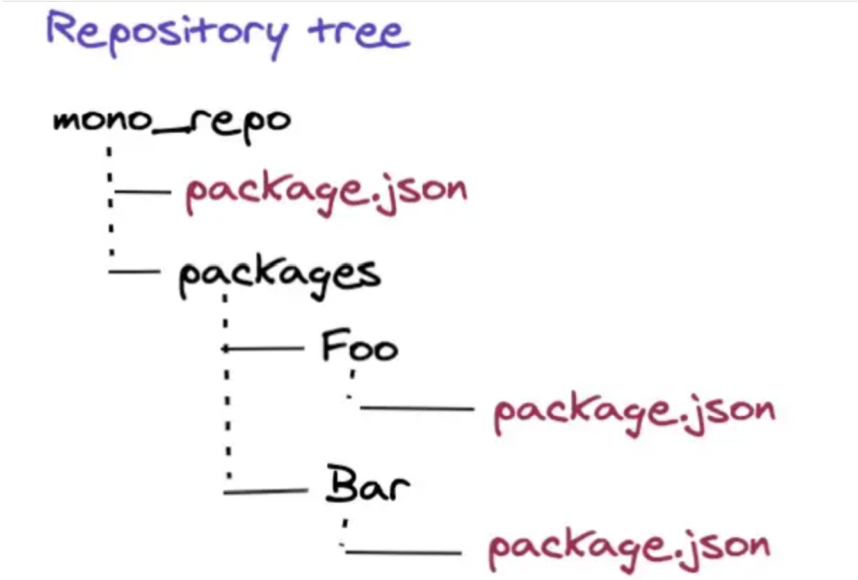
S růstem monorepa dochází k dosazení limitů návrhu v nástrojích pro správu verzí, sestavovacích systémech a kontinuálních integračních kanálech. Následující problémy mohou vést k použití systému multirepo:

1. **špatný výkon** (se zvětšováním monorepa mohou příkazy jako *git blame* trvat nepřiměřeně dlouho, může také dojít ke zpožďování IDE a následkem toho se zhoršuje produktivita; v takovém případě se testování celého repa při každém potvrzení stává neproveditelným),
2. **nefunkční main/master** (nefunkční master ovlivňuje každého, kdo pracuje v monorepu; nedodržení čistoty a aktuálnosti testů),
3. **křivka učení** (ta je pro nové uživatele strmější, pokud úložiště zahrnuje mnoho úzce propojených projektů),
4. **velké objemy dat** (monorepa mohou dosáhnout nepraktických objemů dat a potvrzení za den),

5. **vlastnictví** (udržování vlastnictví souborů je náročnější, protože systémy jako Git nebo Mercurial nemají vestavěná oprávnění k adresářům),
6. **recenze kódu** (oznámení mohou být velmi hlučná; například GitHub má omezená nastavení oznámení, která nejsou nejvhodnější pro sněhové skluzy žádostí o stažení a kontroly kódu) [12].

3.3.1 Lerna

Lerna je nástroj pro správu projektů JavaScript/TypeScript s více balíčky a je především přínosná pro projekty, které sdílejí společné závislosti. Jeho pomocí se řídí monorepa, která v sobě mohou obsahovat projekty zahrnující více balíčků. Vedení monorepa může být náročné, protože sekvenční sestavení a publikování jednotlivých balíčků trvá poměrně dlouho. Lerna poskytuje funkce, jako jsou *bootstrapping* balíčků, paralelizovaná sestavení a publikace artefaktů, které usnadňují správu monorepa. Lerna se většinou používá ve větších projektech, jejichž údržba může být časem náročná. Umožňuje modularizaci kódu do menších spravovatelných úložišť a abstrahování kódu ke sdílení, který lze použít v těchto dílčích úložištích. Vykonává těžkou práci při správě úloh, jako jsou verzování, správa sdílených závislostí, nasazování kódu, čímž snižuje zátěž, která přichází s architekturou monorepo. Na obrázku č. 3 je znázorněn příklad stromové struktury monorepo se dvěma „dílčími“ repozitáři *Foo* a *Bar* [13] [14] [15].



Obrázek 3: Příklad stromové struktury monorepo se dvěma „dílčími“ repozitáři Foo a Bar. Zdroj: [15]

3.4 Klient

Termín klient/server se používá k popisu výpočetního modelu pro vývoj počítačových systémů. Tento model je založen na rozdělení funkcí mezi dva typy nezávislých a autonomních procesů: server a klient. Klient je jakýkoli proces, který vyžaduje specifické služby od procesu serveru. Pojmu client-side lze rozumět tak, že akce probíhá na počítači uživatele (klienta). Vývojáři na straně klienta využívají své dovednosti v oblasti kódování k vytváření vizuálně přitažlivých, funkčních a užitečných webových aplikací a dynamických webových stránek. Vývoj na straně klienta se provádí téměř výhradně v JavaScriptu. To je samozřejmě kromě základního HTML a CSS-kódu [16] [17].

3.4.1 ReactJS

React je javascriptová knihovna pro vytváření interaktivních uživatelských rozhraní pomocí komponent, která byla vytvořena Facebookem v roce 2011 pro její použití na Facebooku a Instagramu. Roku 2013 byla první verze Reactu zveřejněna jako open-source software. React byl navržen z důvodu potřeby efektivně aktualizovat webové stránky v reakci na události. Jedná se především o události, které mohou spouštět aktualizace na webových stránkách, zahrnují vstup uživatele, nová data přicházející do aplikací z jiných webových stránek a zdrojů dat a data přicházející do aplikace ze senzorů jako údaje o poloze. Tradiční způsob webových aplikací vypořádávání se s daty měnícími se v průběhu času spočívá v tom, že se občas obnovují a v procesu kontrolují nová data. Technologie React představuje nový způsob pro snadnější vytváření aplikací reagující na nová data, místo jednoduchého obnovování stránky, nezávisle na tom, zda se základní data změnila, nebo nikoliv. Rozdíl v přístupech si lze představit jako *pull* (tradiční způsob aktualizace webových stránek) a *push* (reaktivní způsob vytváření webových stránek). Tato metoda aktualizace uživatelského rozhraní v reakci na změny dat se nazývá programování reakce [18].

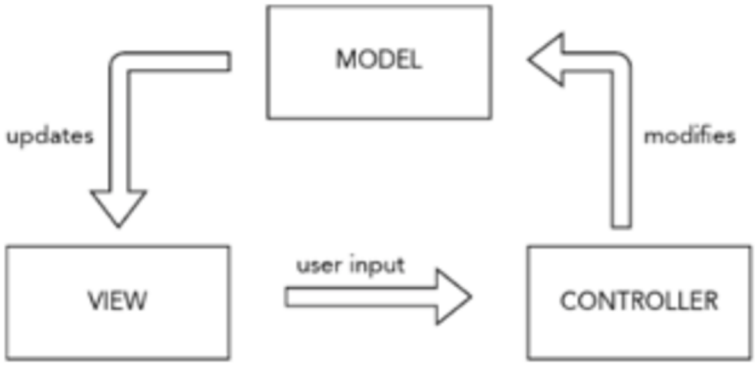
React představuje knihovnu pro vytváření a skládání komponent pro vytváření uživatelských rozhraní. Tyto komponenty reprezentují nezávislé části, které si mohou předávat data. Každá komponenta může představovat jednoduchý prvek, například jak tlačítko, tak i složitější elementy, jako je například navigační panel, který se skládá z kolekce tlačítek a rozevíracích seznamů. Velikost jednotlivých komponent v aplikaci je vhodné určovat pomocí konceptu jediné odpovědnosti, který představuje myšlenku, že komponenta by měla mít odpovědnost za jednu část funkčnosti programu [18].

MVC

Webové aplikace jsou obvykle sestavovány a popsány pomocí modelu MVC. Tento model zahrnuje následující prvky:

1. M (model představuje datovou vrstvu),
2. V (view [pohled] je to, co uživatel vidí a s čím interaguje),
3. C (controller [ovládač], který usnadňuje komunikaci s datovou vrstvou) [18].

V aplikaci MVC pohled posílá vstup do řadiče, který předává data mezi datovou vrstvou a pohledem. V modelu MVC technologie React se týká pouze pohledu (V), protože přebírá data jako vstup a v nějaké formě je prezentuje uživateli. Na obrázku č. 4 je zobrazen princip fungování modelu MVC [18].



Obrázek 4: Princip fungování modelu MVC. Zdroj: [18]

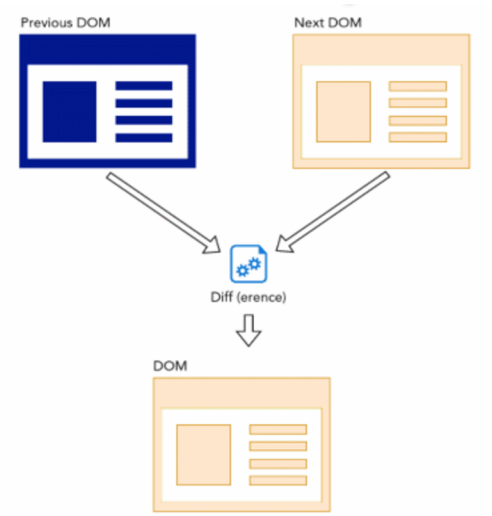
Jelikož React pouze vykresluje komponenty a nezáleží mu na zařízení nebo rozhraní používaném uživatelem, prezentování jednotlivých komponent určuje pak samostatná knihovna. Zpracováváním vykreslování komponent React ve webovém prohlížeči se zabývá knihovna ReactDOM. Pomocí virtuálního modelu dokumentu ReactDOM umožňuje rozhraním vytvořeným v Reactu efektivně zvládat množství změn na obrazovce, které vyžadují moderní webové aplikace. ReactDOM má řadu funkcí pro propojení mezi Reactem a webovými prohlížeči, ale ta, kterou využívá každá aplikace React, se nazývá ReactDOM.render. Obrázek č. 5 znázorňuje vztah mezi Reactem, ReactDOM a webovým prohlížečem [18].

Obrázek 5: Vztah mezi Reactem, ReactDOM a webovým prohlížečem. Zdroj: [18]

Virtual DOM

Objektový model dokumentu neboli DOM je programovací rozhraní pro dokumenty HTML (HyperText Markup Language) a XML (Extensible Markup Language), které definuje logickou strukturu dokumentů a způsob, jakým se k dokumentu přistupuje a jak se s ním manipuluje. Představuje interní reprezentaci webové stránky ve webovém prohlížeči, která převádí HTML, styly a obsah na skupiny, které lze provozovat pomocí JavaScriptu. Změny DOM způsobí změny toho, co uživatel pozoruje ve svém webovém prohlížeči, a naopak aktualizace provedené v něm ovlivňují DOM. S každou změnou DOM prohlížeč musí zkontrolovat, zda změna vyžaduje stránku, která se má překreslit, a až poté musí dojít k překreslení. Proto je ve srovnání s jinými druhy kódu JavaScript manipulace s DOM pomalá a neefektivní. K obtížnosti manipulace s DOM se přidává to, že funkce DOM nejsou vždy snadno použitelné a některé z nich mají příliš dlouhé názvy. Pro oba z těchto důvodů bylo vytvořeno mnoho různých knihoven pro manipulaci s JavaScript DOM [18] [19].

Virtuální DOM (VDOM) je koncept programování, kde ideální nebo „virtuální“ reprezentace uživatelského rozhraní je uchovávána v paměti a synchronizována se „skutečným“ DOM pomocí knihoven, jako je ReactDOM. Tento proces se nazývá sladění. Daný přístup umožňuje deklarativní API React: uživatel uvádí požadovaný stav uživatelského rozhraní a zajišťuje, aby DOM tomuto stavu odpovídal. Tím se abstrahuje manipulace s atributy, zpracování událostí a ruční aktualizace DOM, které by jinak musely být použity k sestavení aplikace. Na obrázku č. 6 je znázorněn princip fungování virtuálního objektového modelu dokumentu [18] [20].



Obrázek 6: Princip fungování virtuálního objektového modelu dokumentu. Zdroj: [18]

State management

Správa stavu je jednou z důležitých a nevyhnutelných funkcí každé dynamické aplikace. Stav aplikace určuje, co uživatel vidí, jak aplikace vypadá, jaká data jsou uložena apod. Současně existuje velké množství open-source knihoven navržených speciálně pro usnadnění a zpříjemnění správy stavu. Z velkého množství rámců uživatelského rozhraní JavaScript je React pravděpodobně tím, který má nejživější ekosystém a poskytuje jednoduché a flexibilní API pro podporu správy stavu v komponentě React. Stav představuje hodnotu dynamických vlastností komponenty v dané instanci a React poskytuje dynamické úložiště dat pro každou z nich. Interní data představují stav komponenty React a lze k nim přistupovat pomocí členské proměnné *this.state* komponenty. Kdykoli se změní její stav, komponenta se znovu vykreslí voláním metody *render( )* spolu s novým stavem [21].

JSX

JSX je zkratka pro JavaScript XML, které představuje rozšíření syntaxe JavaScript. Tato technologie umožňuje psát HTML kód k popisu toho, jak by mělo vypadat uživatelské rozhraní, přímo do projektu React, který podporuje skutečnost, že logika vykreslování je neodmyslitelně spojena s další logikou uživatelského rozhraní: jak se zachází s událostmi, jak se mění stav v průběhu času a jak jsou data připravena k zobrazení. Namísto umělého oddělování technologií vložením značek a logiky do samostatných souborů React odděluje věci pomocí volně spojených jednotek nazývaných „komponenty“, které obsahují obojí. React nevyžaduje uplatnění JSX, ale jeho aplikace jako vizuální pomůcky při práci s uživatelským rozhraním v kódu JavaScript se považuje za užitečnou. Toto rozšíření umožňuje také Reactu zobrazovat užitečnější chybové a varovné zprávy [22] [23] [24].

3.4.2 WebRTC

Webová komunikace v reálném čase (WebRTC) je projekt s otevřeným zdrojovým kódem, který je v současné době vyvíjen s cílem zajistit komunikaci typu peer-to-peer v reálném čase mezi webovými aplikacemi. Tato technologie poskytuje přidávání do aplikace možnosti komunikace v reálném čase, která funguje nad otevřeným standardem a podporuje video, hlas a obecná data, která se mají posílat mezi partnery, což umožňuje vývojářům vytvářet výkonná řešení pro hlasovou komunikaci a videokomunikaci. Technologie je dostupná na všech moderních prohlížečích i na nativních klientech pro všechny hlavní platformy. WebRTC je implementován jako otevřený webový standard a je dostupný jako běžný JavaScript API ve všech hlavních prohlížečích. Pro nativní klienty,

jako jsou aplikace pro Android a iOS, je k dispozici knihovna, která poskytuje stejné funkce. WebRTC má mnoho různých případů použití, od základních webových aplikací, které využívají kameru nebo mikrofon, až po pokročilejší aplikace pro videohovory a sdílení obrazovky. Přímá připojení peer-to-peer často poskytují nižší latenci, díky čemuž se hry, streamování videa, přenos dat ze senzorů atd. zdají být rychlejší a interaktivnější nebo vypadají tak, že skutečně probíhají v reálném čase, proto se používá tento termín. WebRTC umožňuje volat přímo z webové stránky bez jakéhokoli pluginu. To bylo umožněno pomocí mediálních rozhraní API prohlížeče pro načítání uživatelských médií, WebSocket pro přenos a HTML5 pro vykreslování médií na webové stránce. WebRTC je tedy vyvinutá forma komunikace WebSocket – protokolu Transport Layer, který přenáší data. WebSocket API je aplikační programovací rozhraní, které umožňuje webovým stránkám používat protokol WebSocket pro komunikaci se vzdáleným hostitelem [25] [26] [27] [28] [29].

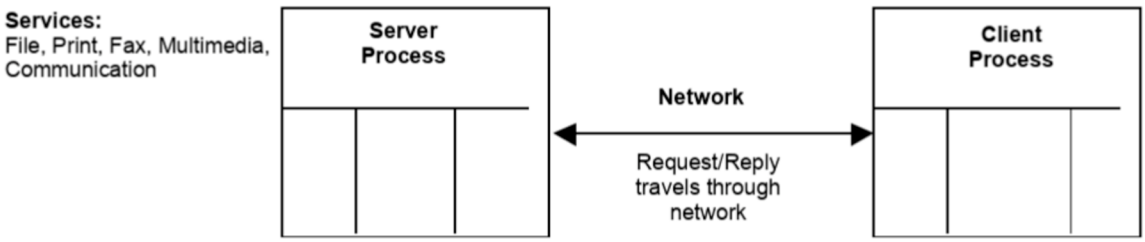
3.4.3 YouTube Player API

YouTube Player API (aplikační programovací rozhraní) je rozhraní přehrávače Iframe, které umožňuje vložit přehrávač videa YouTube na web a ovládat ho pomocí JavaScriptu. Pomocí funkcí JavaScriptu API lze zařadit videa do fronty, přehrát, pozastavit nebo zastavit tato videa, upravit hlasitost přehrávače nebo získat informace o přehrávaném videu. Je možné také přidat posluchače událostí, které se spustí v reakci na určité události přehrávače, jako je změna stavu přehrávače. Rozhraní API poskytuje možnost načíst zdroje související s videi, uživateli a seznamy videí, a zároveň umožňuje manipulovat s těmito kanály, jako jsou vytváření nových seznamů skladeb, přidávání videí jako oblíbených a odesílání zpráv [30] [31].

3.5 Server

Server je proces, který poskytuje požadované služby pro klienta. Procesy klienta a serveru mohou být umístěny ve stejném počítači nebo v různých počítačích propojených sítí. Když procesy klient a server sídlí na dvou nebo více nezávislých počítačích v síti, server může poskytovat služby pro více než jednoho klienta. Kromě toho může klient požadovat služby z několika serverů v síti bez ohledu na umístění nebo fyzické vlastnosti počítače, ve kterém se proces serveru nachází. Síť spojuje server a klienta dohromady a poskytuje

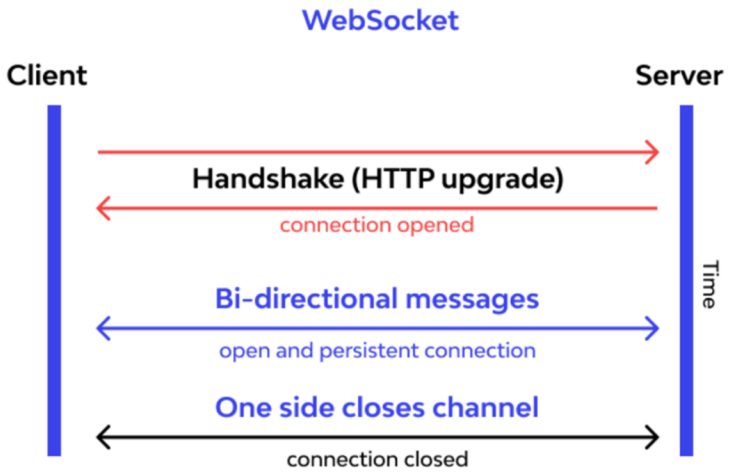
médium, přes které klienti a server komunikují. Obrázek č. 7 ukazuje základní výpočetní model klient/server [16] [17].



Obrázek 7: Základní výpočetní model klient/server. Zdroj: [16]

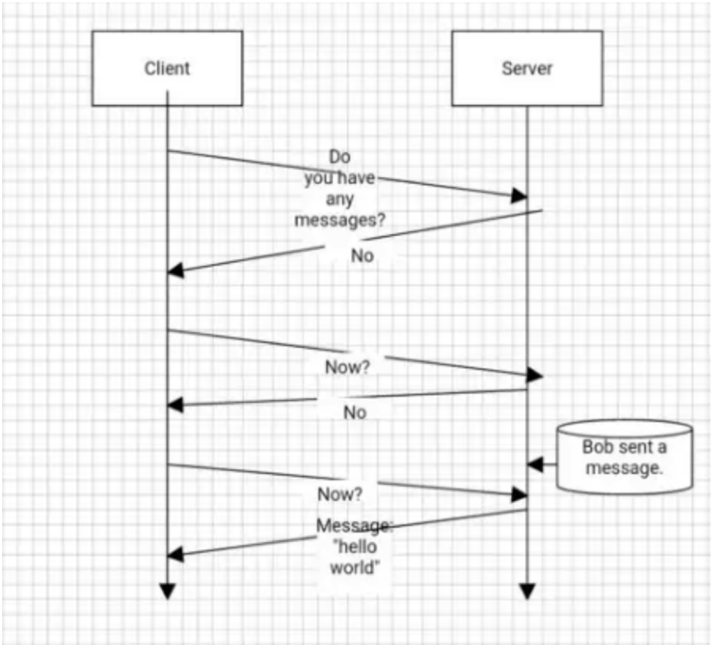
3.5.1 WebSocket

Podle konvenční definice WebSocket představuje duplexní protokol používaný především v komunikačním kanálu klient/server. Má obousměrnou povahu, což znamená, že komunikace probíhá mezi klientem a serverem navzájem. Když je vytvořen nový WebSocket (URL), začne se okamžitě připojovat. Během připojení se prohlížeč (pomocí hlaviček) zeptá serveru, jestli ten podporuje WebSocket. Pokud dostane kladnou odpověď, pak rozhovor pokračuje v protokolu WebSocket, který vůbec není HTTP. Spojení vyvinuté pomocí WebSocketu trvá tak dlouho, dokud jej kterákoli ze zúčastněných stran propustí. Jakmile jedna strana přeruší spojení, druhá strana nebude moci komunikovat, protože spojení se automaticky přeruší. Příklad takové komunikace je znázorněn na obrázku č. 8 [32] [33] [34].

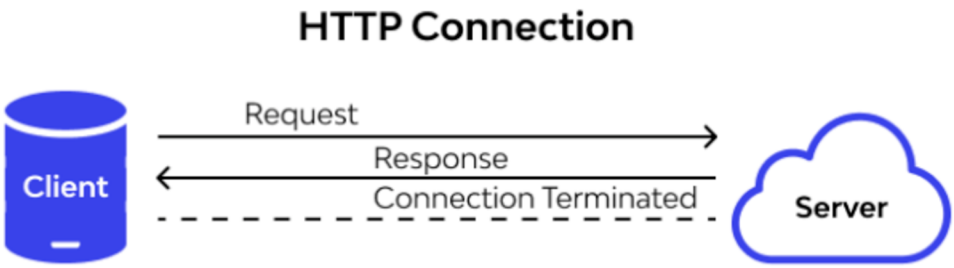


Obrázek 8: Komunikace mezi klientem a serverem u WebSocketu. Zdroj: [32]

V případě protokolu HTTP je k odpovědi potřeba žádost. Pro možnost přijetí zprávy je nutné se neustále dotazovat serveru, zda jsou nové zprávy (viz obrázky č. 9 a 10) [33].

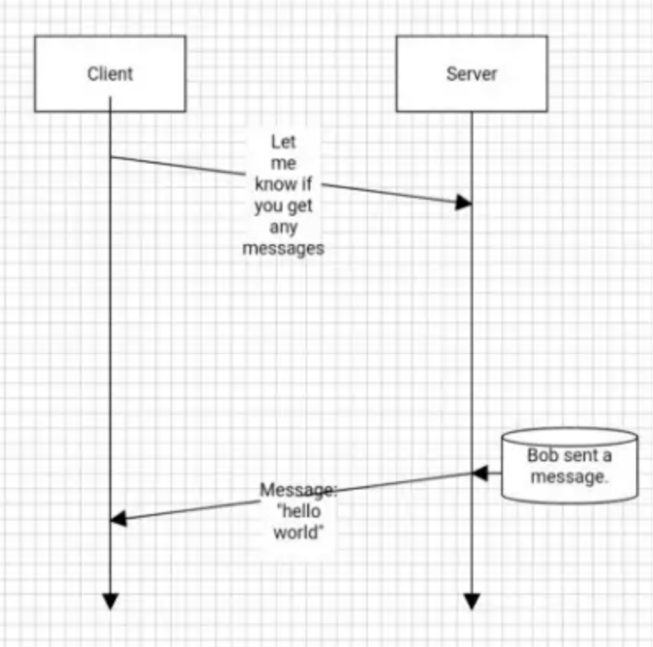


Obrázek 9: Znázornění komunikace mezi klientem a serverem v protokolu HTTP. Zdroj: [33]

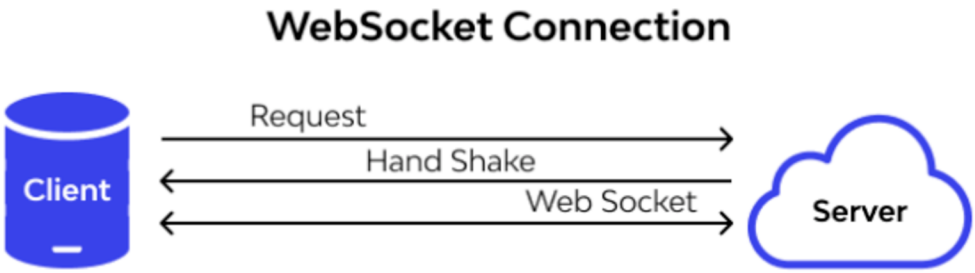


Obrázek 10: Životní cyklus připojení protokolu HTTP. Zdroj: [32]

U použití WebSocket, na jiné straně, pro poskytnutí odpovědi není potřeba posílat požadavek. Daný protokol umožňuje obousměrný tok dat, a proto stačí poslouchat jakákoli data (viz obrázky č. 11 a 12). Technologie WebSocket je velmi užitečná především při vývoji aplikací běžících v reálném čase, chatovacích aplikací, IoT (Internet věcí), online her pro více hráčů [32].



Obrázek 11: Znázornění komunikace mezi klientem a serverem u WebSocketu. Zdroj: [33]

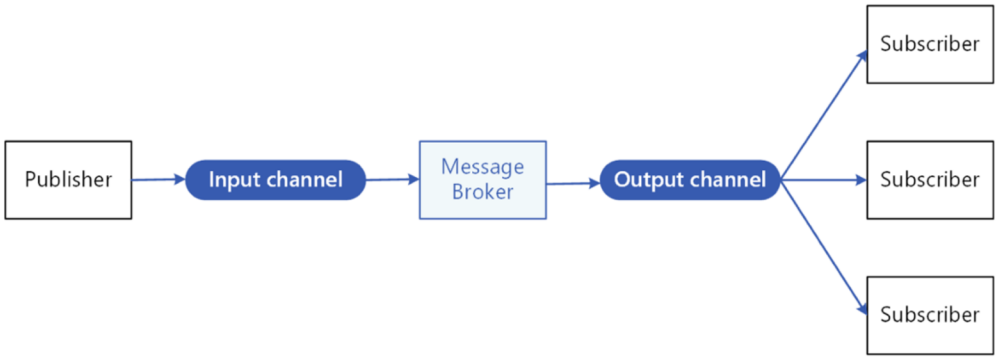


Obrázek 12: Životní cyklus připojení protokolu WebSocket. Zdroj: [32]

3.5.2 Publish–subscribe pattern

V cloudových a distribuovaných aplikacích při provedení události komponenty systému často potřebují poskytovat informace dalším komponentám. Asynchronní zasílání zpráv je účinný způsob, jak oddělit odesílatele od příjemce a vyhnout se blokování odesílatele, aby čekal na odpověď. Použití vyhrazené fronty zpráv pro každého příjemce však není efektivní pro jejich větší množství. Některé příjemce by také mohla zajímat pouze podmnožina informací. Publish/Subscribe je asynchronní podsystém zpráv, který umožňuje odesílateli oznamovat události všem zainteresovaným příjemcům, aniž by znal jejich totožnost. Jedná se o formu asynchronní komunikace mezi službami, která je používána v architekturách bez serverů a v architekturách mikroslužeb. Tato forma komunikace zahrnuje odesílatele a odběratele, kteří se spoléhají na zprostředkovatele zpráv, který předává jejich zprávy. Hostitel (odesílatel) publikuje zprávy (události) na kanálu, ke kterému

se pak mohou přihlásit odběratelé. Přestože Pub/Sub vychází z dřívějších návrhových vzorů, jako jsou řazení zpráv a zprostředkovatelé událostí, je flexibilnější a škálovatelnější. Klíčem k tomu je skutečnost, že Pub/Sub umožňuje pohyb zpráv mezi různými komponentami systému, aniž by si komponenty navzájem uvědomovaly svou identitu. V modelu Pub/Sub je každá zpráva publikovaná k tématu okamžitě přijata všemi odběrateli tématu. Daný model lze použít k aktivaci událostí řízených architektur nebo k oddělení aplikací za účelem zvýšení výkonu, spolehlivosti a škálovatelnosti. Obrázek č. 13 ukazuje logické součásti typu komunikace Publish/Subscribe [35] [36] [37].



Obrázek 13. Logické součásti typu komunikace Publish/Subscribe. Zdroj: [35]

Použití způsobu komunikace typu Pub/Sub má následující výhody:

1. odděluje subsystémy, které ještě potřebují komunikovat,
2. zvyšuje škálovatelnost a zlepšuje odezvu odesílatele,
3. zvyšuje spolehlivost,
4. umožňuje odložené nebo plánované zpracování,
5. umožňuje jednodušší integraci mezi systémy využívajícími různé platformy, programovací jazyky nebo komunikační protokoly, a také mezi místními systémy a aplikacemi běžícími v cloudu,
6. usnadňuje asynchronní pracovní postupy v rámci celého podniku,
7. zlepšuje testovatelnost,
8. poskytuje oddělení problémů pro aplikace [35].

3.5.3 Redis

Redis představuje open-source úložiště datových struktur v paměti používané jako databáze, cache, zprostředkovatel zpráv a streamovací stroj, které lze používat u většiny programovacích jazyků. Redis poskytuje datové struktury, jako jsou například řetězce, hash, seznamy, sady, seřazené sady s rozsahovými dotazy, bitmapy, hyperlogy, geoprostorové

indexy a streamy. Redis má vestavěnou replikaci, skriptování Lua, vyřazení LRU, transakce a různé úrovně perzistence na disku. Tato technologie zároveň poskytuje vysokou dostupnost prostřednictvím Redis Sentinelu a automatické dělení pomocí Redis Clusteru. Na těchto typech lze spouštět atomické operace, například:

- připojení k řetězci,
- zvýšení hodnoty v hashi,
- posunutí prvku do seznamu,
- výpočet průniku množin,
- sjednocení a rozdíly,
- získání prvku s nejvyšším hodnocením ve tříděné sadě [38; 39].

Pro dosažení špičkového výkonu pracuje Redis s datovou sadou v paměti. V závislosti na případu použití může uchovávat data buď pravidelným ukládáním datové sady na disk, nebo připojením každého příkazu do protokolu na disku. V případě, že je potřeba pouze síťová mezipaměť s bohatou funkcí, lze zakázat přetrvávání. Redis podporuje asynchronní replikaci s rychlou neblokující synchronizací a automatickým opětovným připojením s částečnou resynchronizací na rozdělení sítě. Redis je napsán v ANSI C a bez externích závislostí funguje na většině systémů POSIX, jako jsou Linux, *BSD a Mac OS X. Linux a OS X jsou dva operační systémy, kde je Redis vyvíjen a testován nejvíce, nicméně může fungovat i v systémech odvozených od Solarisu, jako je SmartOS, kde podpora je ale nejlepší snahou. Pro Redis neexistuje žádná oficiální podpora pro Windows [38] [39].

4 Vlastní práce

Praktická část bakalářské práce je věnována vytvoření webové aplikace pro společenské sledování videí z platformy YouTube. Pro vývoj aplikace je použit programovací jazyk TypeScript a knihovny ReactJS, Express atd. Jako vývojové prostředí je použit program Visual Studio Code. Nejprve budou provedeny analýza požadavků, výběr technologií a návrh uživatelského rozhraní, a následně se uskuteční vytvoření samotné webové aplikace i její otestování.

4.1 Analýza požadavků

Daná podkapitola se zabývá analýzou požadavků vyvíjené aplikace. V první fázi vytvoření aplikace je potřeba si definovat její základní funkcionality neboli nároky na užívání, kterým bude odpovídat.

Jedná se o dynamickou responzivní webovou aplikaci s názvem WeTube, která slouží pro společné sledování videí z platformy YouTube, je spustitelná na všech zařízeních, včetně mobilů, tabletů a desktopů, a proto není omezena konkrétním operačním systémem uživatele. V rámci použití aplikace je uživatel schopen vytvořit tzv. *lobby* – místnost pro sledování videí, kam může zasíláním odkazu na místnost přizvat její další účastníky. K vytvoření místnosti ani k připojení na ni není potřebná registrace do aplikace. Uživatelem aplikace může být jakákoliv osoba, která se dostane na webovou stránku, stejně jako účastníkem lobby bude jakákoliv osoba, která má přístup k odkazu na místnost. Účastníkům jednotlivých lobby je pak umožněno pomocí vkládání odkazu na video z YouTube přidávat videa do fronty videí a tato videa z fronty zároveň vymazávat. Uživatel lobby také může prohlédnout seznam účastníků a opustit lobby. Není možné vyřadit z lobby jiné uživatele aplikace. Podle charakteristiky obsahu vyvíjené aplikace a požadavků na ni lze odvodit následující základní funkcionality hotové aplikace:

1. Uživateli je umožněno se dostat na webovou stránku nezávisle na používaném zařízení nebo internetovém prohlížeči.
2. Uživateli je umožněno vytvořit novou místnost.
3. Uživateli je umožněno se připojit k existující místností pomocí jejího odkazu.
4. Uživateli je umožněno zadáním odkazu na video z platformy YouTube přidat do fronty nové video.
5. Uživateli je umožněno odstranit video z fronty.

6. Uživateli je umožněno zobrazit seznam účastníků místnosti.
7. Uživateli je umožněno opustit místnost.

Tyto funkcionality budou implementovány v rámci vývoje webové aplikace a budou zprostředkovány uživatelům hotové aplikace.

4.2 Výběr technologií

Účelem této kapitoly je představit přehled a analýzu technologií použitých při tvorbě webové aplikace. Implementace projektu této bakalářské práce má za cíl poskytnout příklad řešení na straně serveru i klienta a také předvést pracovní postup vývojového procesu. Server a klient jsou svou podstatou samostatné aplikace a komunikují spolu přes protokol WebSocket, což eliminuje potřebu použití stejného programovacího jazyka při vývoji. Takový přístup může být velmi užitečný, když je do vývoje zapojeno více týmů, kde tým A používá TypeScript k vývoji řešení na straně klienta a tým B používá svůj preferovaný jazyk k poskytování aplikací na straně serveru. Nevyhnutelnou nevýhodou takového přístupu je nutnost, aby si obě aplikace zachovaly veřejné rozhraní pro očekávanou komunikaci ve svém jazyce. Pro účely tohoto projektu bylo rozhodnuto použít jediný programovací jazyk – TypeScript. To by mělo umožnit sdílení společných modelů a utilit pro serverové i klientské implementace a zároveň eliminovat funkci veřejného komunikačního rozhraní mezi aplikacemi. Pro dosažení lepší znovupoužitelnosti kódu a verzování, a také automatizace mnoha opakujících se úloh, jako je sestavování/spouštění/publikování více aplikací současně, pro správu mono repozitářů bylo rozhodnuto použít technologii Lerna. Pokud jde o signalizační službu na straně serveru, ke zpracování příchozích připojení a upgradu požadavků HTTPS na komunikaci protokolu WebSocket se používá framework Express. Použití tohoto frameworku je velmi omezené a mohlo být nahrazeno vlastní implementací, ale vzhledem k jeho spolehlivosti a přizpůsobitelnosti bylo namísto ručního psaní nízkoúrovňového zpracování HTTP rozhodnuto použít toto řešení. Další open-source knihovna používaná na straně serveru je „ws“ – poskytuje obal pro komunikaci WebSocket a používá se pro manipulaci a operace WebSocketu, jako jsou otevírání/uzavírání připojení a také odesílání/přijímání zpráv. Na straně klienta je k dispozici mnoho knihoven a frameworků pro tvorbu dynamických webových aplikací. Mezi nejpopulárnější open-source řešení široce přijatá vývojáři patří ReactJS, Angular a Vue. Všechny tyto technologie nabízejí vývojářům odlišný přístup k budování aplikace a zároveň prezentují různé stupně flexibility při vývoji.

Pro tuto aplikaci bylo rozhodnuto použít knihovnu ReactJS, protože se má za to, že poskytuje nejlepší úroveň flexibility při volbě celkové architektury aplikace a zároveň umožňuje opětovně použít velkou část kódové základny a vyhnout se normám. Jako každá jiná uživatelsky orientovaná aplikace by tento klientský projekt měl poskytovat intuitivní design pro UI/UX komponentu aplikace. Aby se zabránilo vývoji vlastní knihovny komponent pro opakující se prvky, jako jsou tlačítka, vstupy, typografie a mnoho dalších, bylo rozhodnuto použít knihovnu „mui“. Tato knihovna poskytuje obrovské množství komponent připravených k výrobě, které se řídí systémem Material Design, vytvořeným a spravovaným společností Google. Takový přístup umožňuje uživateli aplikace používat známé uživatelské rozhraní/UX a pomáhá rychleji vytvářet nové funkce, a zároveň se starat o mnoho aspektů designu.

4.3 Návrh uživatelského rozhraní

Důležitou částí vytvoření aplikace je také určení jejího uživatelského rozhraní (UI), které v následující fázi vývoje bude sloužit k nasměrování zobrazení aplikace ve webovém prohlížeči. V této podkapitole budou definovány wireframy aplikace, které představují návrh umístění prvků aplikace na webové stránce. Při návrhu uživatelského rozhraní vyvíjené aplikace WeTube jsou vytvořeny wireframy pro zařízení desktop (1 440 px × 1 024 px) a mobil (IPhone 14 Pro, 309 px × 852 px) pro následující obrazovky:

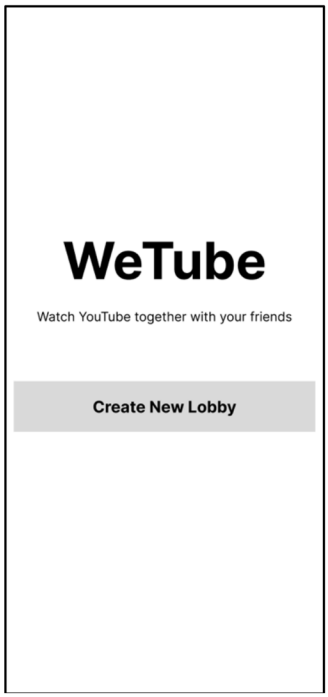
1. hlavní obrazovka aplikace,
2. obrazovka Lobby – znázornění seznamů videí,
3. obrazovka Lobby – znázornění seznamů účastníků místností.

V rámci této bakalářské práce je vytvoření wireframů provedeno v aplikaci pro design rozhraní Figma.

4.3.1 Hlavní obrazovka aplikace

V návrhu uživatelského rozhraní na hlavní obrazovce vyvíjené aplikace bylo rozhodnuto zobrazit logo aplikace – WeTube –, její slogan a tlačítko pro vytvoření místnosti. Wireframy pro hlavní obrazovku jsou ilustrovány na obrazcích č. 14 (pro desktopové rozhraní) a č. 15 (pro mobilní rozhraní).

Obrázek 14: Hlavní obrazovka vyvíjené aplikace WeTube pro desktopové rozhraní. Zdroj: vlastní zpracování



Obrázek 15: Hlavní obrazovka vyvíjené aplikace WeTube pro mobilní rozhraní. Zdroj: vlastní zpracování

4.3.2 Obrazovka Lobby – znázornění seznamů videí

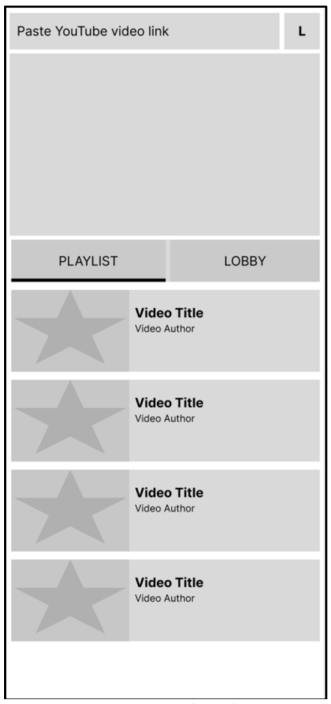
Obrazovka Lobby (místností) se znázorněním seznamu videí obsahuje následující prvky:

- logo aplikace v levém horním rohu,
- vyhledávací lištu pro vkládání odkazů na videa z YouTube uprostřed nahoře,

- tlačítko *Leave* pro opuštění místností,
- přehrávané video,
- tlačítko *Playlist,*
- tlačítko *Lobby* pro přesměrování na obrazovku *Lobby* se zobrazením seznamu účastníků,
- seznam videí v pořadí pod tlačítky *Playlist a Lobby*.

Wireframy pro obrazovku místností se zobrazením seznamu videí jsou ilustrovány na obrázcích č. 16 (pro desktopové rozhraní) a č. 17 (pro mobilní rozhraní).

Obrázek 16: Obrazovka Lobby – znázornění seznamů videí vyvíjené aplikace WeTube pro desktopové rozhraní. Zdroj: vlastní zpracování



Obrázek 17: Obrazovka Lobby – znázornění seznamů videí vyvíjené aplikace WeTube pro mobilní rozhraní. Zdroj: vlastní zpracování

4.3.3 Obrazovka Lobby – znázornění seznamů účastníků místností

Obrazovka Lobby (místností) se znázorněním seznamů účastníků místností obsahuje následující prvky:

- logo aplikace v levém horním rohu,
- vyhledávací lištu pro vkládání odkazů na videa z YouTube uprostřed nahoře,
- tlačítko *Leave* pro opuštění místností,
- přehrávané video,
- tlačítko *Lobby,*
- tlačítko *Playlist* pro přesměrování na obrazovku *Playlist* se zobrazením seznamu videí,
- seznam účastníků místností pod tlačítky *Playlist a Lobby.*

Wireframy pro obrazovku místností se zobrazením seznamu účastníků jsou ilustrovány na obrázcích č. 18 (pro desktopové rozhraní) a č. 19 (pro mobilní rozhraní).

Obrázek 18: Obrazovka Lobby – znázornění seznamů účastníků místností vyvíjené aplikace WeTube pro desktopové rozhraní. Zdroj: vlastní zpracování

Obrázek 19: Obrazovka Lobby – znázornění seznamů účastníků místností vyvíjené aplikace WeTube pro mobilní rozhraní. Zdroj: vlastní zpracování

4.4 Základní struktura projektu

Prvním a velmi důležitým krokem při zahájení tvorby aplikace je vybrat nejvhodnější strukturu projektu, která by umožnila bezproblémový vývoj a zajistila aplikaci do budoucna. Dobře strukturovaná kódová základna je důležitá z následujících důvodů. Těmi jsou:

1. **udržovatelnost** (dobře strukturovaná kódová základna usnadňuje pochopení a úpravu kódu, což zjednodušuje jeho údržbu v průběhu času; to je důležité zejména u velkých projektů s mnoha přispěvateli),
2. **škálovatelnost** (dobře strukturovaná kódová základna se snadno přizpůsobí novým vlastnostem a funkcím, aniž by se stala příliš složitou nebo obtížnou pro správu),
3. **spolupráce** (dobře strukturovaná kódová základna usnadňuje spolupráci více vývojářů na projektu; při dodržování zavedených standardů kódování a používání konzistentních návrhových vzorů mohou vývojáři efektivněji spolupracovat),
4. **znovupoužitelnost** (dobře strukturovaná kódová základna usnadňuje opětovné použití kódu v jiných projektech; rozdělením kódu na menší, opakovaně použitelné komponenty mohou vývojáři ušetřit čas a úsilí při vytváření nových projektů).

Pro účely tohoto projektu bylo rozhodnuto použít strukturu „monorepo“. Její využití usnadňuje správu závislostí a zajišťuje, aby veškerý kód byl v projektu aktuální. Pro dosažení nejlepší automatizace procesů kolem nastavení a správy monorepa využívá tento projekt nástroj Lerna. Pomáhá zjednodušit správu projektů monorepo tím, že poskytuje jasnou strukturu a sadu nástrojů pro správu více balíčků. To může ušetřit čas, úsilí a usnadnit udržení konzistence a kvality v rámci projektu. Je nutné globálně nainstalovat Lernu prostřednictvím správce balíčků NPM a spustit inicializační příkazy v kořenovém adresáři projektu. Jako další krok se rozhraní příkazového řádku Lerna použije k vytvoření tří následujících balíčků, které by definovaly základní strukturu projektu:

- Wetube-common (sdílená knihovna vytvořená pro lepší znovupoužitelnost kódu; především obsahuje definici rozhraní pro obousměrnou komunikaci mezi serverem a klientem),
- Wetube-signal (serverová strana projektu, který využívá protokol WebSocket k poskytování jednoduché funkce signalizační služby, jež pomáhá klientům

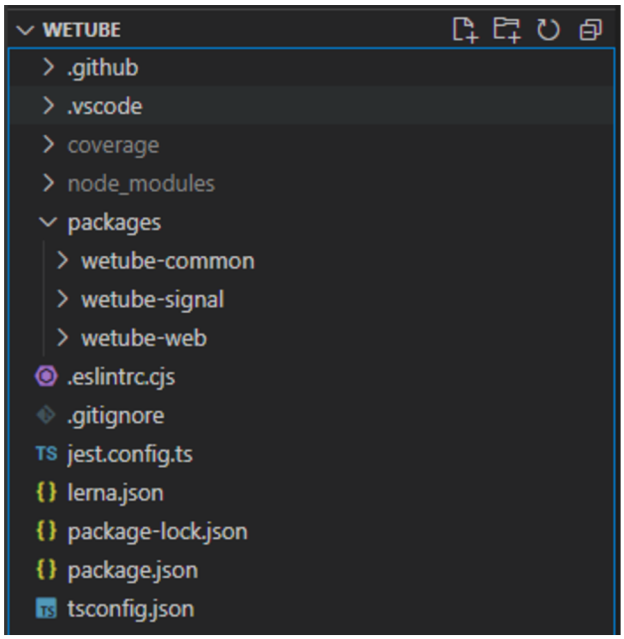
vyměňovat si nabídky dat a odpovědi za účelem navázání komunikace WebRTC),

- Wetube-web (klientská strana projektu, který využívá knihovnu ReactJS k vytvoření moderního dynamického webu, jenž odpovídá požadavkům projektu).

Než se začne s nastavením každého jednotlivého balíčku, je důležité definovat sdílené závislosti, které bude používat většina balíčků. Pro tento projekt budou tyto závislosti zahrnovat:

- **TypeScript**. Existuje mnoho důvodů, proč dát přednost TypeScriptu před JavaScriptem jako hlavnímu jazyku kódové báze. Tento projekt využívá TypeScript, aby odhalil chyby už v době kompilace, a také lepší nástroje, včetně vylepšené podpory editorů, lepšího automatického dokončování a vylepšených možností refaktorování kódu.
- **ESLint.** Nástroj pro analýzu statického kódu, který kontroluje běžné chyby v kódu a vynucuje soubor pravidel a pokynů, což může pomoci zlepšit celkovou kvalitu a udržovatelnost kódové základny. Použití ESLintu pro projekt TypeScript je zvláště užitečné, protože pomáhá identifikovat potenciální problémy a zachytit běžné chyby související s kontrolou typu a dalšími funkcemi specifickými pro TypeScript.

Znázornění struktury složek projektu otevřeného v kódu Visual Studio po dokončení fáze nastavení je prezentováno na obrázku č. 20.



Obrázek 20: Znázornění struktury složek projektu otevřeného v kódu Visual Studio. Zdroj: vlastní zpracování

Všechny balíčky lze sestavit pomocí příkazu „lerna run build“, který spustí sestavení skriptů všech balíčků v pořadí respektujícím řetězec závislostí. Balíčky jako wetube-signal a wetube-web lze spustit ve vývojovém režimu společně pomocí „lerna run start“ nebo samostatně pomocí příkazů „lerna run –scope=[package-name] start“. V důsledku výše uvedeného nastavení lze projekt spustit jediným příkazem, který vydává všechny ladicí informace potřebné k zahájení vývoje. Výsledek spuštění vyvíjené aplikace přes terminál je ilustrován na obrázku č. 21.

```
$ lerna run start
lerna notice cli v6.5.1

 >  Lerna (powered by Nx)   Running target start for 2 projects:

    - wetube-signal
    - wetube-web



> wetube-signal:start


> wetube-web:start

> wetube-signal@0.0.0 start
> node lib/wetube-signal-bundle.js
> wetube-web@0.0.0 start
> webpack serve --config webpack.dev.js
Server started
<i> [webpack-dev-server] Project is running at:
<i> [webpack-dev-server] Loopback: http://localhost:9000/
<i> [webpack-dev-server] On Your Network (IPv4): http://192.168.0.110:9000/
<i> [webpack-dev-server] Content not from webpack is served from 'C:\dev\wetube\packages\wetube-web\lib\public' directory
<i> [webpack-dev-server] 404s will fallback to '/index.html'
asset wetube-web.js 9.98 MiB [emitted] (name: main)
asset index.html 558 bytes [emitted]
977 modules
webpack 5.75.0 compiled successfully in 13556 ms
```

Obrázek 21: Výsledek spuštění vyvíjené aplikace přes terminál. Zdroj: vlastní zpracování

4.5 Implementace

Účelem této kapitoly je popsat architektonická řešení a možnosti návrhu, které byly důležité pro udržení vysoké flexibility kódové základny a podpory iterací v byznys logice nebo uživatelském rozhraní, aniž by byla ohrožena hlavní struktura projektu.

4.5.1 Wetube-common

Účelem balíčku wetube-common je sdílení společného kódu, který lze znovu použít v jiných částech aplikace. Z pohledu daného projektu je nejdůležitější částí tohoto balíčku komunikační rozhraní, které definuje obousměrnou komunikaci mezi signalizační službou a klientskou webovou aplikací. Vzhledem k tomu, že komunikace mezi serverem a klientem probíhá přes protokol WebSocket, který spoléhá na výměnu zpráv, je velmi důležité striktně definovat strukturu potenciálních zpráv. Dalším důležitým faktorem, který je třeba vzít v úvahu při navrhování tohoto rozhraní, je zajištění jeho nezávislosti na interní logice serveru a klienta. Účelem závazku by mělo být to, že jsou dohodnuty obě strany, což od nich

skrývá složitost konkrétní realizace. Aplikace většinou funguje na dvou entitách, jimiž jsou uživatel a lobby. Obě představují objekty nebo kolekce, se kterými se mají provádět akce. Zatímco množství informací známých o každé entitě se může pro server a klienta lišit, mělo by existovat společné rozhraní definující všechny potenciální akce, které lze provést, stejně jako definice struktury požadavku na provedení každé akce a odpovědi vrácené po jejím dokončení.

Pro splnění tohoto požadavku bylo rozhodnuto vytvořit sadu typů TypeScript, které lze umístit do balíčku wetube-common a zpřístupnit k nim klientské i serverové implementační balíčky. Obrázky č. 22 a 23 ukazují rozhraní pro lobby a uživatelské entity. Všechny akce a zprávy, které lze provést, jsou definovány jako objektové klíče a hodnoty jsou představeny jednoduchými objekty, jež obsahují typ požadavku a typ odpovědi pro příslušnou zprávu.

```
export type UserAPI = {
    [UserMessageType.CREATE]: WSEndpoint<EmptyObj, NewUserData>
}
```

Obrázek 22: Rozhraní pro lobby a uživatelské entity. Zdroj: vlastní zpracování

```
export type LobbyAPI = {
    [LobbyMessageType.CREATE]: WSEndpoint<
        EmptyObj,
        LobbyId
    >
    [LobbyMessageType.JOIN]: WSEndpoint<
        LobbyId,
        LobbyId & LobbyJoinerData & LobbyUsersData
    >
    [LobbyMessageType.LEAVE]: WSEndpoint<
        LobbyId,
        LobbyId & LobbyUsersData
    >
    [LobbyMessageType.DATA_OFFER]: WSEndpoint<
        LobbyId & LobbyRecipientData & LobbySessionDescriptionData,
        LobbyId & LobbyExchangeMessageData & LobbySessionDescriptionData
    >
    [LobbyMessageType.DATA_ANSWER]: WSEndpoint<
        LobbyId & LobbyRecipientData & LobbySessionDescriptionData,
        LobbyId & LobbyExchangeMessageData & LobbySessionDescriptionData
    >
    [LobbyMessageType.ICE_CANDIDATE]: WSEndpoint<
        LobbyId & LobbyRecipientData & LobbyIceCandidateData,
        LobbyId & LobbyExchangeMessageData & LobbyIceCandidateData
    >
}
```

Obrázek 23: Rozhraní pro lobby a uživatelské entity. Zdroj: vlastní zpracování

4.5.2 Wetube-signal

Klientská aplikace se snaží propojit uživatele pomocí protokolu WebRTC, který umožňuje komunikaci mezi webovými prohlížeči v reálném čase. Jednou z klíčových funkcí WebRTC je spojení peer-to-peer, což znamená, že zařízení komunikují přímo mezi sebou bez potřeby centrálního serveru. Aby však mohla dvě zařízení spolu komunikovat pomocí WebRTC, musí si vyměňovat informace o svých síťových adresách a možnostech. Zde přichází na řadu signalizační server. Jeho hlavním účelem je usnadnit výměnu informací mezi zařízeními, jako jsou síťové adresy, nabídky připojení a odpovědi na připojení. Jakmile si zařízení vymění tyto informace přes signalizační server, mohou navázat přímé spojení a začít spolu komunikovat. Účelem balíčku wetube-signal je poskytnutí odlehčené signalizační služby WebSocket s implementací pro rozhraní LobbyAPI a UserAPI, definovaná v balíčku wetube-common. Struktura tohoto balíčku je poměrně jednoduchá a člení se do tří hlavních vrstev:

- **Repository**. Tato vrstva je zodpovědná za interakci s úložištěm dat. Poskytuje aplikaci způsob, jak získávat i ukládat data a také provádět základní operace CRUD (vytváření, čtení, aktualizace, mazání) s daty. Vrstva repository abstrahuje podrobnosti o úložišti dat a poskytuje konzistentní rozhraní pro použití zbytku aplikace. V balíčku je zastoupena prostřednictvím rozhraní IUserRepository a ILobbyRepository a jejich odpovídajících implementací.
- **Service**. Tato vrstva je umístěna nad vrstvou repository a poskytuje funkci vyšší úrovně zbytku aplikace. Vrstva service je zodpovědná za implementaci byznys logiky a koordinaci akcí napříč více úložišti. Pokud například aplikace potřebuje provést složitou operaci, daná vrstva by tuto logiku zvládla. Logika této aplikace signalizační služby je řízena implementacemi UserService a LobbyService.
- **Controller**. Tato vrstva je zodpovědná za zpracování příchozích požadavků a vytváření příslušných odpovědí. Controller bývá obvykle umístěn mezi klientem (jako je webový prohlížeč) a byznys logikou aplikace řízenou vrstvou služeb. Přijímá požadavky, interpretuje je a posílá zpět odpovědi klientovi. Vrstva controller je reprezentována třídami UserController a LobbyController.

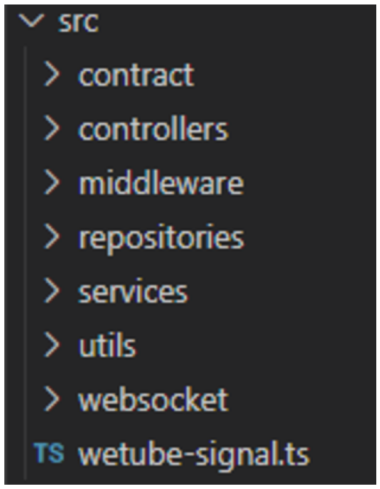
Kromě hlavních vrstev aplikace je důležité zmínit i volby designu provedené tak, aby byla aplikace flexibilnější vůči iteracím a příchozím změnám:

- **Utils**. Sada obalů závislostí třetích stran, která poskytuje zjednodušené nebo vlastní rozhraní pro externí knihovnu nebo službu. Použití obalu kolem závislosti třetí strany má několik výhod. Za prvé může usnadnit integraci a používání externí knihovny pro vývojáře, kteří by se jinak museli potýkat s výstřednostmi a zvláštnostmi knihovny. Za druhé může pomoci izolovat aplikaci od změn v knihovně třetí strany, protože obal lze aktualizovat tak, aby zvládl jakékoli změny nebo nové funkce, aniž by to ovlivnilo zbytek kódové základny. Za třetí obal může také poskytovat další funkce nebo vlastnosti, které nejsou přítomny v původní knihovně, jako jsou ukládání do mezipaměti, zpracování chyb nebo optimalizace výkonu. Příklady takových obalů v kódové základně jsou RandomNameGenerator a JWTAuth. Oba fungují jako jednoduchý obal pro externí knihovny „random-animal-name-generator“ a „jsonwebtoken“, což umožňuje kdykoli přepnout na jinou knihovnu nebo na vlastní implementaci beze změn ve zbytku kódové základny.
- **Middleware**. V kontextu této aplikace funkce middleware přijímá objekt požadavku a také funkci „next“ a může na nich provést určité zpracování před předáním řízení dalšímu middlewaru v řetězci. Middleware-funkce lze použít k provádění úloh, jako jsou ověřování, protokolování, analýza požadavků, zpracování chyb a další. Příkladem middlewaru pro tento projekt je ověřovací middleware, který by měl ověřit token připojený k požadavku a umožnit uživateli pokračovat v aplikaci, nebo odmítnout požadavek s neoprávněnou chybovou odpovědí. Implementace funkce getAuthMiddleware je znázorněna na obrázku č. 24.

```
export const getAuthMiddleware = (userService: UserService): WSEndpointMiddleware<AuthData> => {
    return (req, connection, next) => {
        const { auth } = req.body;

        if (auth) {
            connection.authToken = auth;
        }

        if (!connection.authToken) {
            next(ErrorResponse.UnauthorizedRequest);
            return;
        }

        try {
            const decoded = userService.validateUser(connection.authToken);
            req.user = decoded;
            next();
        } catch (_) {
            next(ErrorResponse.UnauthorizedRequest);
        }
    };
};
```

Obrázek 24: Implementace funkce getAuthMiddleware. Zdroj: vlastní zpracování

Na obrázku č. 25 je znázorněn příklad adresářové struktury služby wetube-signal otevřené ve Visual Studio Codu.



Obrázek 25: Příklad adresářové struktury služby wetube-signal otevřené ve Visual Studio Codu. Zdroj: vlastní zpracování

Na obrázku č. 26 je znázorněn hlavní bootstrap soubor pro službu wetube-signal. Tok aplikace provádí následující kroky:

1. vytvoření „express“ serveru a „ws“ serveru,
2. vyřešení vkládání závislostí pomocí definování instancí pro všechna úložiště, nástroje, služby a řadiče,
3. zaregistrování řadiče do vlastního řešení směrování WebSocket,
4. spuštění routeru a naslouchání na poskytnutém portu.

```
const app = express();
const server = http.createServer(app);
const wss = new WebSocketServer({ server: server });

// Define repositories
const userRepo = new UserInMemoryRepository();
const lobbyRepo = new LobbyInMemoryRepository();

// Define utils
const jwtAuth = new JWTAuth<AuthDataPayload>();
const randomNameGenerator = new RandomNameGenerator();

// Define services
const userService = new UserService(jwtAuth, userRepo, randomNameGenerator);
const lobbyService = new LobbyService(userRepo, lobbyRepo);

// Define controllers
const lobbyController = new LobbyController(userService, lobbyService);
const userController = new UserController(userService);

// Define router and register
const router = new WSRouter(wss);
router.register(lobbyController);
router.register(userController);

router.start();
server.listen(process.env.PORT || 3000, () => {
    console.log('Server started');
});
```

Obrázek 26: Hlavní bootstrap-soubor pro službu wetube-signal. Zdroj: vlastní zpracování

4.5.3 Wetube-web

Účelem této kapitoly je prozkoumat implementaci webové aplikace na straně klienta napsané v TypeScriptu a ReactJS, zejména prověřit, jak může dosáhnout komunikace se signalizační službou WebSocket, a také využít datové kanály WebRTC k umožnění komunikace peer-to-peer v reálném čase. Tato kapitola se zabývá architekturou a designem aplikace, a taktéž úvahami spojenými s vytvářením aplikace pro spolupráci v reálném čase a tím, jak je lze překonat. Pokud jde o architektonické volby kódové základny klientských aplikací, jedním z hlavních pilířů návrhu tohoto projektu je dosažení jasného oddělení logiky aplikace a strany prezentačního pohledu. Oddělení logiky od pohledové strany aplikace je důležité z několika důvodů. Jsou jimi:

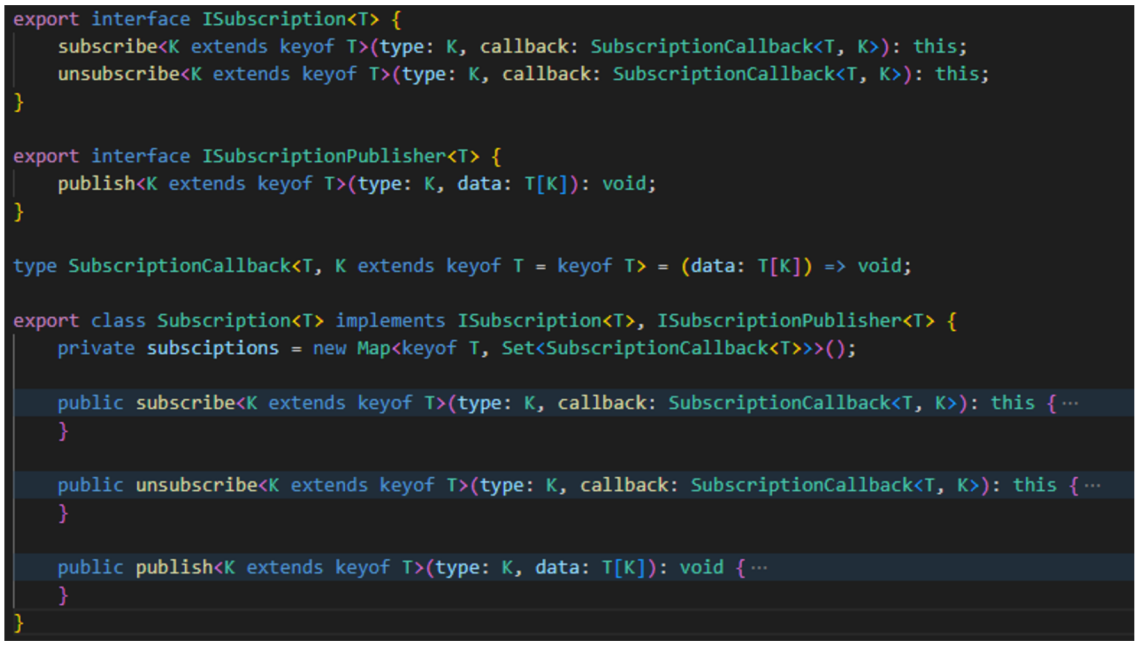
1. **Udržitelnost kódu**. Když jsou byznys logika a logika prezentace odděleny, změny jedné neovlivňují druhou, což usnadňuje aktualizaci nebo úpravu konkrétních částí aplikace, aniž by to ovlivnilo celou kódovou základnu.
2. **Znovupoužitelnost kódu**. Je snazší znovu použít stejnou logiku ve více pohledech nebo komponentách. To může vést ke zkrácení doby vývoje a může pomoci zajistit konzistenci napříč aplikací.
3. **Vylepšená škálovatelnost.** Oddělením logiky a pohledu můžeme izolovat úzká místa výkonu a optimalizovat konkrétní části aplikace, což umožňuje škálovat každou vrstvu aplikace nezávisle.
4. **Snadné testování**. Izolováním logiky a jejím testováním nezávisle na pohledu lze zajistit, že obchodní logika funguje tak, jak bylo zamýšleno, aniž by se muselo starat o prezentační vrstvu. To může vést ke komplexnějšímu testování a celkově robustnější aplikaci.

Celkově může oddělení logiky od pohledové strany aplikace vést k modulárnější, udržovatelnější a škálovatelnější kódové základně, což usnadňuje vývoj a údržbu aplikací s komplexní logikou v reálném čase. Pro dosažení požadovaného rozdělení mezi prezentací a vytížeností je logická struktura aplikace rozdělena na plynulé části:

- Utils,
- API,
- Services,
- Context,
- Pages.

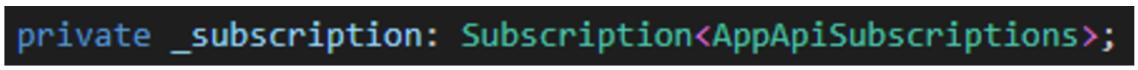
Utils

I když se třídy utilit nezdají být velkou součástí kódové základny aplikace, pro účely tohoto projektu je důležité zmínit nejprve některá jejich návrhová řešení, protože hrají určující roli v návrhu jiných aplikačních vrstev. Příkladem takového důležitého nástroje je třída Subscription. Poskytuje jednoduchý a opakovaně použitelný způsob správy předplatných a událostí v projektu TypeScript. Implementace třídy Subscription je znázorněna na obrázku č. 27.

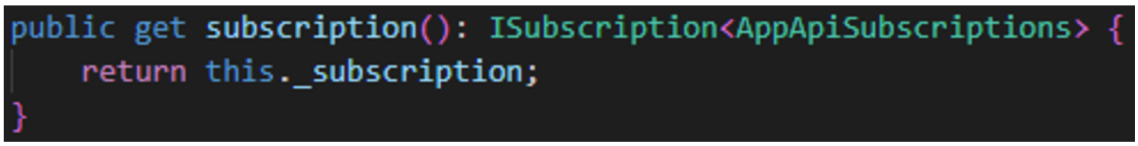


Obrázek 27: Implementace třídy Subscription. Zdroj: vlastní zpracování

Tato třída implementuje dvě rozhraní, ISubscription a ISubscriptionPublisher, která umožňují podrobnou kontrolu nad funkcemi, jež jsou přístupné koncovému uživateli třídy. Takový návrh například umožňuje publikovat události pro vlastníka instance soukromé třídy a současně zpřístupňovat veřejnosti pouze funkci přihlášení/odhlášení. Poskytuje také striktně typovaný obecný argument pro definování možných předplatných a problémů se zachycením při pokusu o přihlášení k odběru/publikování nesprávných událostí během kompilace. Na obrazcích č. 28 a 29 je znázorněn příklad toho, jak je třída Subscription definována jako soukromý člen třídy s plným přístupem k veřejnému rozhraní, ale veřejně přístupná pouze jako getter s rozhraním omezeným pouze na metody subscribe/unsubscribe.



Obrázek 28. Definování třídy Subscription jako soukromého člena třídy s plným přístupem k veřejnému rozhraní. Zdroj: vlastní zpracování



Obrázek 29: Definování třídy Subscription jako soukromého člena třídy s plným přístupem k veřejnému rozhraní. Zdroj: vlastní zpracování

Services

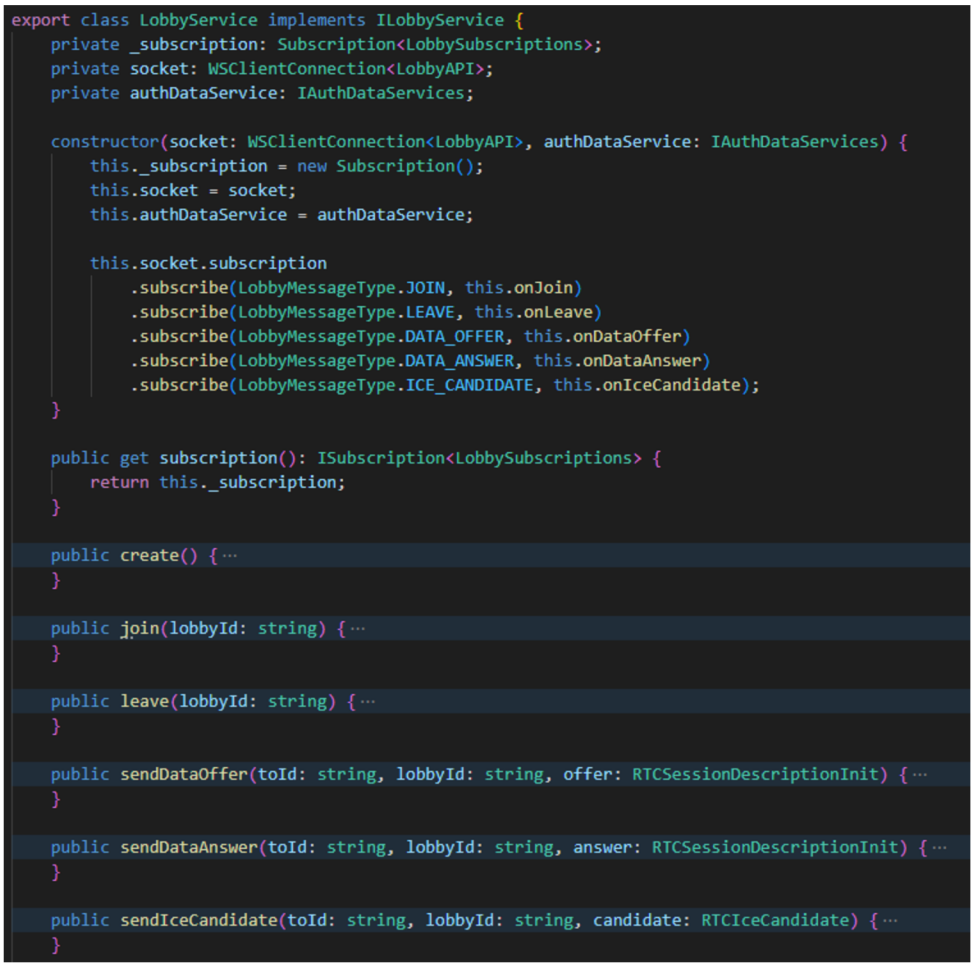
V kontextu front-endové aplikace je vrstva *services,* zodpovědná za zapouzdření veškeré byznys logiky a poskytování čistého rozhraní pro interakci. Každá služba je navržena tak, aby zvládla specifickou sadu úkolů, a je zodpovědná za správu veškerých nezbytných interakcí se serverem, a také za všechna místní data nebo stav, který je třeba

spravovat. Servisní vrstva může pomoci zjednodušit proces vývoje snížením množství duplicitního kódu a logiky v celé aplikaci. Konsolidací souvisejících funkcí do služeb se kódová základna stává modulárnější a snáze škálovatelná, protože jsou přidávány nové funkce, nebo upravovány stávající. To může vést k rychlejšímu vývoji a celkově spolehlivějšímu kódu. Tento projekt implementuje několik služeb navržených tak, aby zvládaly různé úkoly:

- IAuthDataService (poskytuje API pro použití úložišti ověřovacích dat, kde základní implementací může být řešení v paměti pro snadnější ladění nebo localStorage API pro produkční sestavení),
- IMediaMetadataService (umožňuje získat metadata o videích, jako jsou název, miniatura atd.; implementace využívá YouTube API pro přístup k těmto datům o médiích),
- ILobbyService (poskytuje přístup k byznys logice související s Lobby pomocí metod, jako jsou join, create, leave a další; implementace této služby využívá připojení WebSocket k signalizační službě wetube-signal pro provádění logiky a přístupových dat lobby),
- IMediaSyncService (poskytuje jednoduché rozhraní pro provádění akcí, jako jsou přehrávání a pauza, které synchronizují postup uživatelů při sledování videa; logika synchronizace médií implementovaná prostřednictvím komunikace protokolu WebRTC).

Hlavní logiku aplikace představují dvě služby: ILobbyService a IMediaSyncService. Pro hlubší pochopení základní logiky dané služby budou dále podrobně charakterizovány.

Třída LobbyService komunikuje s rozhraním WebSocket API na straně serveru prostřednictvím třídy programu utilit WSClientConnection, která je předána konstruktoru. Třída se přihlásí k odběru různých typů zpráv rozhraní WebSocket API a pro každý z nich definuje obslužné rutiny událostí. Když je přijata zpráva určitého typu, je vyvolán manipulátor události, který publikuje data odběratelům odpovídající události prostřednictvím objektu Subscription. Třída LobbyService také poskytuje metody pro vytvoření lobby, připojení k lobby, opuštění lobby a odesílání nabídek/odpovědí WebRTC dat jiným klientům ve stejné lobby. Přestože tato třída zvládá komunikaci se službou wetube-signal, jejím účelem je skrýt tuto skutečnost před svými spotřebiteli. Implementace třídy LobbyService je znázorněna na obrázku č. 30.



```
export class LobbyService implements ILobbyService {
    private _subscription: Subscription<LobbySubscriptions>;
    private socket: WSClientConnection<LobbyAPI>;
    private authDataService: IAuthDataServices;

    constructor(socket: WSClientConnection<LobbyAPI>, authDataService: IAuthDataServices) {
        this._subscription = new Subscription();
        this.socket = socket;
        this.authDataService = authDataService;

        this.socket.subscription
            .subscribe(LobbyMessageType.JOIN, this.onJoin)
            .subscribe(LobbyMessageType.LEAVE, this.onLeave)
            .subscribe(LobbyMessageType.DATA_OFFER, this.onDataOffer)
            .subscribe(LobbyMessageType.DATA_ANSWER, this.onDataAnswer)
            .subscribe(LobbyMessageType.ICE_CANDIDATE, this.onIceCandidate);
    }

    public get subscription(): ISubscription<LobbySubscriptions> {
        return this._subscription;
    }

    public create() {···
    }

    public join(lobbyId: string) {···
    }

    public leave(lobbyId: string) {···
    }

    public sendDataOffer(toId: string, lobbyId: string, offer: RTCSessionDescriptionInit) {···
    }

    public sendDataAnswer(toId: string, lobbyId: string, answer: RTCSessionDescriptionInit) {···
    }

    public sendIceCandidate(toId: string, lobbyId: string, candidate: RTCIceCandidate) {···
    }
```

Obrázek 30: Implementace třídy LobbyService. Zdroj: vlastní zpracování

Třída MediaSyncService umožňuje sdílet data o stavu přehrávání médií a bezproblémově informovat uživatele o jakýchkoli změnách. Implementace této třídy využívá WebRTC k synchronizaci přehrávání médií (např. videí) mezi více klienty připojenými ke společné lobby. Účelem třídy je poskytnout rozhraní pro inicializaci synchronizační služby a vysílání zpráv připojeným kolegům pro ovládání přehrávání médií (přehrávání, pauza atd.). Zatímco metody play a pause jednoduše vysílají zprávu play/pause každému připojenému peeru, metoda initialize bere jako parametry lobby ID, pole objektů LobbyUserObj představující uživatele v lobby a volitelné ID aktuálního uživatele. Provádění této funkce probíhá v následujícím pořadí:

- nastavení posluchačů událostí pro různé události související s lobby (např. join, data-offer, data-answer, ice-candidate),
- vytvoření objektu PeerConnectionClient z rozhraní WebRTC API pro každého uživatele v lobby (kromě aktuálního uživatele a uživatelů se stávajícím připojením),

- zaslání datové nabídky každému klientovi s otevřeným připojením.

Třída také definuje několik soukromých metod, včetně *onDataOffer*, *onDataAnswer* a *onIceCandidate*. Metoda *onDataOffer* je volána, když je přijata datová nabídka od připojeného peeru, a jako odezvu odešle datovou odpověď. Metoda *onDataAnswer* je volána, když je přijata datová odpověď od připojeného peeru, a nastavuje vzdálený popis odpovídajícího objektu *PeerConnectionClient*. Nakonec se metoda *onIceCandidate* zavolá, když je přijat kandidát ICE od připojeného peeru, a přidá kandidáta do odpovídajícího objektu *PeerConnectionClient*. Implementace metod *initialize* a *createClient* třídy MediaSyncService je znázorněna na obrázcích č. 31 a 32.

```
public initialize(lobbyId: string, users: LobbyUserObj[], currentUserId: string) {
    this.lobbyService.subscription
        .subscribe('join', this.onLobbyJoin)
        .subscribe('leave', this.onLobbyLeave)
        .subscribe('data-offer', this.onDataOffer)
        .subscribe('data-answer', this.onDataAnswer)
        .subscribe('ice-candidate', this.onIceCandidate);

    this.createClients(lobbyId, users, currentUserId);
}
```

Obrázek 31: Metoda initialize pro třídu MediaSyncService. Zdroj: vlastní zpracování

```
private createClients(lobbyId: string, users: LobbyUserObj[], currentUserId?: string): void {
    let shouldSendOffer = false;
    if (currentUserId) {
        this.userId = currentUserId;
        shouldSendOffer = true;
    }

    // filter out current user and users with existing connections
    users
        .filter(user => user.id != this.userId && !this.clients.has(user.id))
        .forEach(m => {
            const client = this.createClient(m.id);
            client.connection.addEventListener('icecandidate', (ev) => {
                if (ev.candidate) {
                    console.log(`[MediaSyncService] sending ice candidate for user ${client.id}`);
                    this.lobbyService.sendIceCandidate(m.id, lobbyId, ev.candidate);
                }
            });

            if (shouldSendOffer) {
                const offer = client.connection.createOffer();
                offer?.then((description) => {
                    client.connection.setLocalDescription(description);

                    console.log(`[MediaSyncService] sending offer for user ${client.id} (send offer)`);
                    this.lobbyService.sendDataOffer(m.id, lobbyId, description);
                });
            }

            this.clients.set(client.id, client);
        });
}
```

Obrázek 32: Metoda createClient pro třídu MediaSyncService. Zdroj: vlastní zpracování

API

Vrstva API funguje jako bod orchestrátoru pro celou aplikaci. Pokud by vrstva uživatelského rozhraní vyžadovala vyvolání jakékoli logiky, přístup k datům nebo přihlášení k odběru událostí obchodní logiky, mělo by se to udělat prostřednictvím vrstvy API. Potřeba třídy orchestrátoru vzniká, když má aplikace více tříd služeb, které zpracovávají specifické

aspekty obchodní logiky aplikace. V tomto scénáři aplikace potřebuje centrální řídicí bod, který může koordinovat různé služby a zajistit jejich bezproblémovou spolupráci. Třída orchestrátorů funguje jako prostředník mezi front-endovými komponentami a službami, řídí komunikaci mezi nimi a zajišťuje, že budou spolupracovat soudržně. Může také provádět dodatečné zpracování dat přijatých ze služeb, než je předá zpět front-endovým komponentám. Celkově může třída orchestrátoru pomoci učinit front-endovou aplikaci modulárnější a udržovatelnou tím, že poskytuje centrální bod řízení pro správu obchodní logiky aplikace. Jedním z příkladů takové orchestrace může být akce join lobby definovaná v metodě joinLobby třídy API. Kromě volání metody join ILobbyService také zkontroluje, zda operace proběhla úspěšně, a v takovém scénáři inicializuje IMediaSyncService. Implementace metody joinLobby třídy AppApi je znázorněna na obrázku č. 33.

```
public joinLobby(lobbyId: string) {
    return this.services.lobbyService.join(lobbyId)
        .then(res => {
            const { data } = res;
            if (!data || !data.user) {
                return Promise.reject();
            }

            this.services.mediaSyncService.initialize(data.lobbyId, data.users, data.user.id);

            return res;
        });
}
```

Obrázek 33: Metoda joinLobby pro třídu AppApi. Zdroj: vlastní zpracování

Další výhodou jediné vrstvy API pro projekt je jednoduché vkládání závislostí během vytváření instance třídy API. Takový přístup pomáhá zavést všechny závislosti na jednom místě a také podmíněně změnit základní závislost pro určitá rozhraní. Implementace metody createAppApi je znázorněna na obrázku č. 34.

```
export const createAppApi = () => {
    const authDataService = new InMemoryAuthDataServices();

    const socketUrl = DEBUG ? 'ws:localhost:3000' : 'wss:/wetube-signal.azurewebsites.net';
    const socketClient = new WSClientConnection<LobbyAPI>(socketUrl, {
        getAuthToken: () => authDataService.getData()?.token ?? null,
        authEndpoints: [LobbyMessageType.JOIN]
    });
    socketClient.connect();

    const lobbyService = new LobbyService(socketClient, authDataService);
    const mediaSyncService = new MediaSyncService(lobbyService);
    const mediaMetadataService = new MediaMetadataService();

    return new AppApi({
        authDataService,
        lobbyService,
        mediaSyncService,
        mediaMetadataService,
    });

};
```

Obrázek 34: Metoda createAppApi. Zdroj: vlastní zpracování

Context

React *context* je funkce, která umožňuje předávat data do stromu komponent bez nutnosti předávat je dolů na každou úroveň. Poskytuje způsob, jak sdílet data mezi komponentami, které spolu přímo nesouvisí, ale jsou potomky komponenty společného předka. Lze jej použít k tomu, aby se zabránilo zbytečnému předávání vlastností a zjednodušilo předávání dat v rámci aplikace React. Tato aplikace používá následující poskytovatele kontextu, kteří sdílejí data se zbytkem aplikace:

- *AppApiProvider* (sdílí instanci aplikačního API, aby poskytoval přístup k aplikační logice z libovolné komponenty uživatelského rozhraní),
- *ThemeProvider* (sdílí informace o tématu aplikace),
- *RouterProvider* (sdílí informace o aktuálním umístění aplikačního routeru a také o všech dostupných trasách aplikace [např. připojit se na stránku lobby, vytvořit stránku lobby]).

Pages

Tato část aplikace obsahuje implementaci uživatelského rozhraní prostřednictvím komponent ReactJS. Aplikace je rozdělena na dvě stránky: vytvořit a lobby, které obě ukládají komponenty specifické pro stránku a také vlastní háky reakcí pro přístup k vrstvě API z kontextu a správu místního stavu komponent. Účelem této vrstvy je být co nejjednodušší, neměla by si být vědoma žádné konkrétní implementace obchodní logiky, pouze podmíněně vykreslovat komponenty uživatelského rozhraní na základě stavu poskytovaného vrstvou API.

4.6 Vytvoření uživatelského rozhraní

V předchozí kapitole byla představena byznys logika API vyvíjené aplikace umožňující spravovat její základní funkcionalitu. Účelem této části je ponořit se do světa komponent uživatelského rozhraní React a zjistit, jak lze využít rozhraní API byznys logiky k vytvoření plně funkčního uživatelského rozhraní. Aplikace má dvě hlavní obrazovky: obrazovku vytvoření lobby a samotnou obrazovku lobby. Obě obrazovky jsou vytvořeny pomocí komponent React a jsou integrovány s rozhraním API byznys logiky pro vytvoření bezproblémového uživatelského zážitku.

4.6.1 Vytvoření stránky Create Lobby

Stránka vytvoření lobby (**Create Lobby**) by měla poskytovat bezproblémový a snadný způsob, jak uživatelům pomoci vytvořit lobby a sdílet je s přáteli. K dosažení tohoto cíle bylo rozhodnuto omezit počet interaktivních prvků na této stránce na minimum a nasměrovat pozornost uživatele k jediné akci, kterou hledá tlačítko „Vytvořit nové lobby“. Dalšími prvky na stránce jsou pouze logo a krátký popis podstaty aplikace, aby se uživatel přirozenou cestou poučil o její funkčnosti.

Z pohledu kódu uživatelského rozhraní je cílem této stránky být co nejjednodušší. Kód přidává na stránku několik statických prvků, jako jsou logo a krátký popis, stejně jako tlačítko s obslužným programem *onClick*, který při interakci s ním volá funkci *createLobby*. Na řadě je funkce *createLobby*, která čeká na výsledky vyvolání API byznys logiky, jež obsahuje objekt s informacemi o nově vytvořeném ID lobby. Poté, co je načteno vytvořené ID lobby, kód provede navigační akci ke změně adresy URL stránky na *„/join?lobbyId=[nové ID lobby]“*. Obrázek č. 35 znázorňuje vytvoření základní struktury React komponenty *CreateLobby*.

```
export const CreateLobby = () => {
    const { createLobby } = useCreateLobby();

    return (
        <StyledContainer maxWidth='sm'>
            <Stack spacing={2}>
                <LogoContainer>
                    <Logo
                        noWrap
                        variant="h1"
                        component="div"
                    >
                        {'WeTube'}
                    </Logo>
                    <ShortDescrition
                        variant="h5"
                        component="div"
                    >
                        {'Watch YouTube together with friends'}
                    </ShortDescrition>
                </LogoContainer>
                <CreateButton
                    onClick={createLobby}
                    variant='outlined'
                >
                    {'Create New Lobby'}
                </CreateButton>
            </Stack>
        </StyledContainer>
    );
};
```

Obrázek 35: Reactová komponenta CreateLobby. Zdroj: vlastní zpracování

Na obrázku č.36 je zobrazena logika vytvoření Lobby, která bude spuštěná po kliknutí na tlačítko „Create New Lobby“.

```
export const useCreateLobby = () => {
    const api = useAppApi();
    const navigate = useNavigate();

    const createLobby = useCallback(async () => {
        const lobby = await api.createLobby();
        if (!lobby.data) {
            return;
        }

        navigate({ to: '/join', search: { lobby: lobby.data.lobbyId } });
    }, []);

    return {
        createLobby,
    };
};
```

Obrázek 36: Funkce useCreateLobby. Zdroj: vlastní zpracování

Na příkladu této konkrétní stránky je již snadné vidět výhody vrstvené struktury aplikace. Vrstva uživatelského rozhraní si vůbec neuvědomuje žádnou složitost základních procesů uvnitř rozhraní API byznys logiky. Za oponou se navazuje komunikace WebSocket prostřednictvím zpráv handshake, pak signalizační služba provede vytvoření lobby a odešle odpověď klientovi, ale vrstva uživatelského rozhraní aplikace nic z toho nezná.

4.6.2 Stránka Lobby

Účelem stránky **Lobby** je propojit vzájemně více uživatelů, umožnit jim vybírat a sledovat videa YouTube společně bez nutnosti registrace účtu. Tohoto cíle lze dosáhnout mnoha různými způsoby, ale pro tento projekt bylo rozhodnuto zůstat pouze u základních prvků uživatelského rozhraní, které jsou pro uživatelskou zkušenost nejdůležitější. Mezi takové prvky patří:

- přehrávač s ovládáním,
- vyhledávací lišta,
- playlist,
- seznam lobby,
- tlačítko „Opustit lobby“.

Udržování počtu interaktivních prvků na minimu by mělo uživateli umožnit mnohem rychleji se orientovat v aplikaci a zajistit hladký zážitek z procesu adaptace. Obrázek č. 37 ilustruje JSX strukturu stránky *JoinLobby* na vysoké úrovni.

```
export const JoinLobby = () => {
    const theme = useTheme();
    const matches = useMediaQuery(`${theme.breakpoints.up('md')}`);
    const { playlist, player, currentVideo, lobbyUsers, leaveLobby } = useJoinLobby();

    return (
        <StyledContainer>
            <TopBarContainer>
                {matches ? <Logo
                    variant="h5"
                    component="div"
                >
                    {'WeTube'}
                </Logo> : null}
                <StyledVideoSearch
                    onSuggestionPick={playlist.enqueue}
                />
                <LeaveButton
                    onClick={leaveLobby}
                    variant='outlined'
                >
                    {'Leave'}
                </LeaveButton>
            </TopBarContainer>
            <StyledBox>
                <StyledPlayer
                    player={player}
                    videoId={currentVideo || ''}
                    onNextVideo={playlist.dequeue}
                />
                <StyledTabsPanel tabs={['Playlist', 'Lobby']}>
                    <StyledPlaylist
                        items={playlist.items}
                        onItemSelect={playlist.moveUp}
                        onItemDelete={playlist.remove}
                    />
                    <StyledLobbyList items={lobbyUsers} />
                </StyledTabsPanel>
            </StyledBox>
        </StyledContainer>
    );
};
```

Obrázek 37: Reactová komponenta JoinLobby. Zdroj: vlastní zpracování

Player

Bylo rozhodnuto použít vestavěný přehrávač YouTube, který uživatelům umožní přehrávat jakákoli videa YouTube uvnitř aplikace. YouTube Embedded Player API je sada rozhraní, metod a událostí poskytovaných službou YouTube, která vývojářům umožňuje ovládat a přizpůsobovat chování přehrávače videa YouTube vloženého do aplikace. Rozhraní API poskytuje mimo jiné programový přístup k ovládacím prvkům přehrávání, nastavení kvality videa, stavu přehrávače a oznámení události. S integrovaným přehrávačem YouTube mohou vývojáři vytvářet vlastní možnosti přehrávání videa, které se integrují s dalšími aspekty jejich webových aplikací. Je důležité zmínit také některé z mála nevýhod zabudovaného přehrávače YouTube:

1. omezené přizpůsobení (vzhled přehrávače lze v omezené míře přizpůsobit, což u některých aplikací nemusí stačit),
2. závislost na YouTube (přehrávač se spoléhá na servery a infrastrukturu YouTube, takže pokud YouTube nefunguje, přehrávač přestane fungovat),
3. omezená kontrola nad přehráváním (API poskytuje omezenou kontrolu nad přehráváním, takže nemusí být vhodné pro aplikace, které vyžadují pokročilejší funkce přehrávání nebo přizpůsobení).

Vestavěný přehrávač podporuje události jako „onReady“, „onError“ a „onStateChange“, ale žádná z nich neumožňuje sledovat, zda bylo přehrávání videa změněno kliknutím na časovou osu v nativních ovládacích prvcích přehrávače YouTube. Dalším specifikem přehrávače je, že události pauzy a přehrávání neobsahují žádné informace o tom, co je spustilo, například pokud to byla interakce uživatele s přehrávačem nebo programové vyvolání rozhraní API přehrávače. Společně tyto nevýhody přehrávače znemožňují použití nativních ovládacích prvků YouTube k dosažení požadované funkčnosti aplikace. Nativní ovládací prvky přehrávače lze deaktivovat nastavením proměnné přehrávače „controls“ na nulu. Přiklad nastavení proměnné „controls“ je znázorněn na obrázku č. 38.

```
playerVars: {
    controls: 0,
    autoplay: 1,
    mute: videoId ? 0 : 1
}
```

Obrázek 38: Nastavením proměnné přehrávače „controls“. Zdroj: vlastní zpracování

Vlastní ovládací prvky

Jak již bylo popsáno výše, projekt vyžaduje implementaci uživatelských ovládacích prvků přehrávače, které lze snadno dosáhnout pomocí několika interaktivních prvků, jež jsou úzce spjaty s rozhraním API integrovaného přehrávače YouTube. Pro ovládání stavu hráče jsou implementovány následující ovládací prvky:

- **Přehrát/Pozastavit**. Tato tlačítka umožňují provádět základní ovládání přehrávání a také se ujistit, že akce pochází z interakce uživatele a lze vyvolat logiku synchronizace.
- **Časová osa průběhu videa**. Ovládací prvek časové osy přehrávání se skládá z vodorovného pruhu, který představuje celou dobu trvání mediálního obsahu. Uživatel může kliknout nebo přetáhnout posuvník na liště a přeskočit

na konkrétní bod v médiích, jako je konkrétní okamžik ve videu. Ovládací prvek také zobrazuje aktuální dobu přehrávání a celkovou dobu trvání média. Celkově ovládání časové osy přehrávání poskytuje uživatelům pohodlný a intuitivní způsob navigace a ovládání přehrávání médií, což z něj činí základní funkci jakéhokoli přehrávače médií.

- **Hlasitost**. Interaktivní prvky pro základní ovládání hlasitosti zvuku. Toto ovládání není synchronizováno mezi všemi účastníky lobby, což umožňuje každému pohodlně sledovat video.

Obrázek č.39 ilustruje JSX strukturu komponenty PlayerControls.

```
return (
    <ControlsContainer>
        <StyledPlaybackSlider
            defaultValue={0}
            value={playbackSliderValue}
            onChange={onPlaybackSliderChange}
            aria-label="Default"
            valueLabelDisplay="off"
            size='small'
        />
        <SecondaryControlsContainer>
            <InternalControlsContainer>
                {
                    playing ?
                        <IconButton onClick={onPause}>
                            <Icon>pause</Icon>
                        </IconButton>
                        :
                        <IconButton onClick={onPlay}>
                            <Icon>play_arrow</Icon>
                        </IconButton>
                }
                <IconButton onClick={onPause}>
                    <Icon>volume_up</Icon>
                </IconButton>
                <StyledVolumeSlider
                    defaultValue={0}
                    value={volume}
                    onChange={onVolumeSliderChange}
                    aria-label="Default"
                    valueLabelDisplay="auto"
                    size='small'
                />
            </InternalControlsContainer>
            <Typography
                noWrap
                variant="body1"
                component="div"
            >
                {`${secondsToTimeFormat(elapsed)} / ${secondsToTimeFormat(duration)}`}
            </Typography>
        </SecondaryControlsContainer>
    </ControlsContainer>
);
```

Obrázek 39. JSX komponenty PlayerControls. Zdroj: vlastní zpracování.

Playlist

Playlist se skládá ze seznamu videí, z nichž každé představuje obrázek miniatury, název a jméno autora. Obrázek miniatury poskytuje vizuální reprezentaci obsahu videa, zatímco název a jméno autora poskytují další informace o videu. Všechny informace o videu jsou získávány prostřednictvím YouTube API a představují původní informace o videu YouTube. Funkcionalita seznamu videí umožňuje uživatelům přidávat nebo odebírat videa ze seznamu a přeskakovat videa podle potřeby. Tyto akce jsou synchronizovány se všemi ostatními uživateli v lobby, což zajišťuje, že stav seznamu skladeb zůstane pro všechny účastníky stejný. Pokud například uživatel přidá nové video do seznamu videí, všichni ostatní účastníci v lobby uvidí nové video přidané do seznamu. Podobně, pokud je video odstraněno nebo přeskočeno, všichni ostatní účastníci uvidí změnu v jejich zobrazení seznamu videí. Tato synchronizace zajišťuje, že všichni účastníci v lobby mají stejný pohled na seznam videí a mohou spolupracovat na vytváření sdíleného zážitku ze sledování videa. Umožňuje také uživatelům snadno sdílet svá oblíbená videa s ostatními. Obrázek č.40 zobrazuje příklad komponenty vysoké úrovně – *Playlist*.

```
export const Playlist = (props: PlaylistProps) => {
    const {
        className, items,
        onItemSelect, onItemDelete,
    } = props;

    return (
        <StyledList className={className}>
            <TransitionGroup>
                {items.map((item, i) => (
                    <Collapse key={i}>
                        <ListItem>
                            <StyledDeletableVideoItem
                                metadata={item}
                                onClick={onItemSelect}
                                onDelete={onItemDelete}
                            />
                        </ListItem>
                    </Collapse>
                ))}
            </TransitionGroup>
        </StyledList>
    );
};
```

Obrázek 40. Reactová komponenta Playlist. Zdroj: vlastní zpracování

Vyhledávací lišta

Vyhledávací lišta není samostatným prvkem, ale může být považována za další ovládací prvek pro komponentu seznamu videí. Přijímá URL videa YouTube, extrahuje ID

videa ze vstupu a provádí vyhledávání metadat pro toto video. Pomocí takového rozhraní mohou uživatelé přidávat nová videa do seznamu videí a společně objevovat nový obsah.

Seznam lobby

Jedná se o neinteraktivní seznam uživatelů, kteří jsou aktuálně v lobby. Seznam zobrazuje jména uživatelů, kteří byli náhodně vybráni signalizačním serverem, a je aktualizován, jakmile se kdokoli připojí / opustí lobby. Přestože tento prvek není interaktivní, je stále důležité jej mít, protože uživatelům poskytuje důležité informace, pokud se všichni připojili k lobby a byli připraveni začít sledovat video.

Tlačítko „Leave“

I když je možné znovu otevřít stránku „Create Lobby“ v případě, že by uživatel chtěl opustit lobby nebo vytvořit nové, je vždy lepší v uživatelském prostředí umožnit uživateli provést stejnou akci, aniž by opustil stránku, na které se nachází. Po kliknutí na tlačítko „Leave“ (opustit místnost) aplikace vymaže místní data o aktuální místnosti a přesměruje uživatele na stránku „Create Lobby“ (vytvořit místnost) jediným kliknutím.

4.7 Testování vyvinuté aplikace

V rámci kapitoly 4.1 byly stanoveny požadavky na funkčnost vyvíjené aplikace. Tato kapitola se zabývá otestováním, zda aplikace odpovídá výše stanoveným nárokům.

4.7.1 Požadavky na testování

Z požadované funkčnosti aplikace lze odvodit následující testovací scénáře:

1. Hlavní stránka aplikace:
    a. Při použití odkazu na hlavní webovou stránku aplikace je uživateli umožněno otevřít stránku nezávisle na používaném zařízení nebo internetovém prohlížeči.
    b. Hlavní stránka aplikace obsahuje tlačítko „CREATE LOBBY“.
    c. Při stisknutí tlačítka „CREATE LOBBY“ na hlavní stránce aplikace je uživateli umožněno vytvořit novou místnost.
    d. Při vytvoření nové místnosti je uživatel přesměrován na novou webovou stránku s vytvořenou místností.
2. Stránka vytvořené místnosti:
    a. Při použití odkazu na webovou stránku vytvořené místnosti aplikace je uživateli umožněno se k ní připojit.

b. Webová stránka vytvořené místnosti aplikace obsahuje následující prvky: vyhledávácí lištu, tlačítko „LEAVE“ v pravém horním rohu, obrazovku videa YouTube, záložky „PLAYLIST“ a „LOBBY“.
c. Při stisknutí tlačítka záložky „PLAYLIST“ se barva tlačítka změní na modrou a pod tlačítkem se objeví modrá čára. Barva tlačítka „LOBBY“ se změní na šedivou. Vpravo od videa se objeví seznam přidaných videí.
d. Při stisknutí tlačítka záložky „LOBBY“ se barva tlačítka změní na modrou a pod tlačítkem se objeví modrá čára. Barva tlačítka „PLAYLIST“ se změní na šedivou. Vpravo od videa se objeví seznam účastníků místnosti.
e. Při zadání odkazu na video z platformy YouTube do vyhledávácí lišty je umožněno přidat do fronty nové video, které se znázorní ve frontě pod záložkou „PLAYLIST“.
f. V desktopové verzi na záložce „PLAYLIST“ se při navedení kurzoru na konkrétní video objeví tlačítko s ikonou smazání. Při jeho stisknutí se video odstraní ze fronty.
g. V mobilní verzi na záložce „PLAYLIST“ se přetáhnutím doleva na konkrétním videu objeví tlačítko s ikonou smazání. Při jeho stisknutí se video odstraní ze fronty.
h. Při stisknutí tlačítka „LEAVE“ je uživateli umožněno opustit místnost a uskuteční se jeho přesměrování na hlavní stránku aplikace.

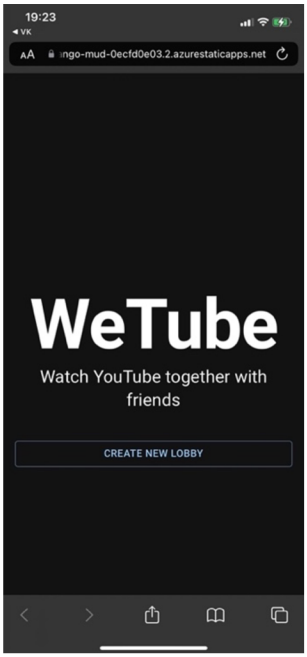
4.7.2 Postup a výsledky testování

Testování webové aplikace vytvořené v rámci této bakalářské práce je provedeno za použití scénáře definovaného v podkapitole 4.6.2. Testování bylo provedeno na následujících zařízeních:

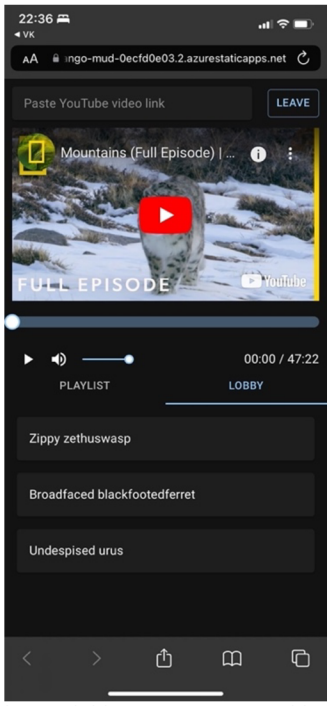
1. mobilní zařízení IPhone 11 (iOS 16.1.1),
2. mobilní zařízení Google Pixel 4a (Android 13),
3. notebook MacBook Pro (macOS Monterey 12.5.1).

V rámci testování vyvinuté aplikace na výše uvedených čtyřech zařízeních za použití internetových prohlížečů Google Chrome, Safari a Firefox byly replikovány všechny scénáře z kapitoly 4.6.1. Výsledky testování jsou následující:

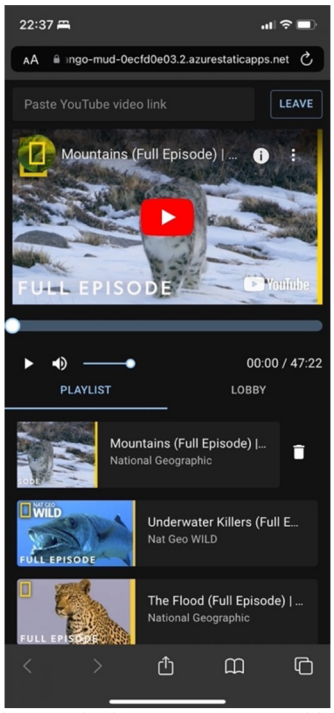
1. Mobilní zařízení IPhone 11 (iOS 16.1.1): otestování všech scénářů za použití prohlížečů Google Chrome, Safari a Firefox proběhlo úspěšně. Níže jsou přiloženy snímky obrazovky (č.41, č.42, č.43) při testování v prohlížeči Safari.



Obrázek 41: Hlavní stránka aplikace otevřená v prohlížeči Safari na zařízení IPhone 11. Zdroj: vlastní zpracování

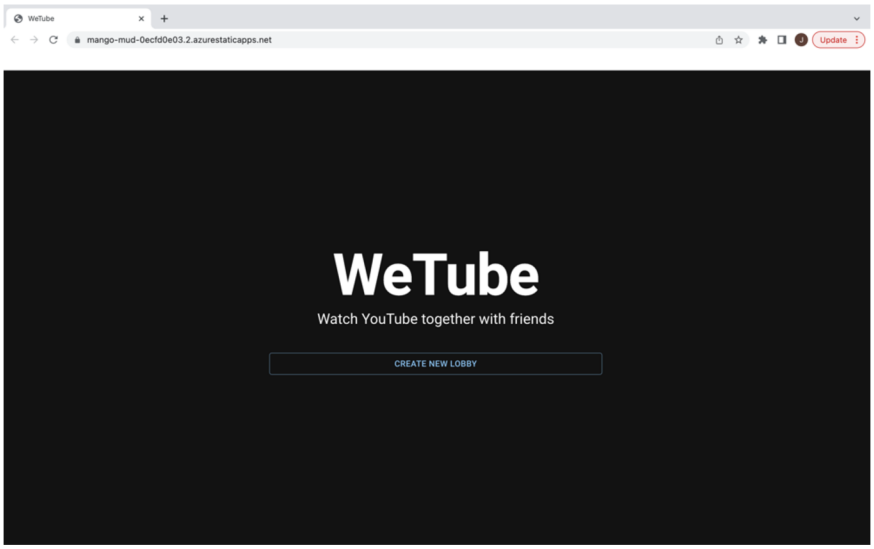


Obrázek 42: Stránka místností aplikace – lobby – otevřená v prohlížeči Safari na zařízení IPhone 11. Zdroj: vlastní zpracování

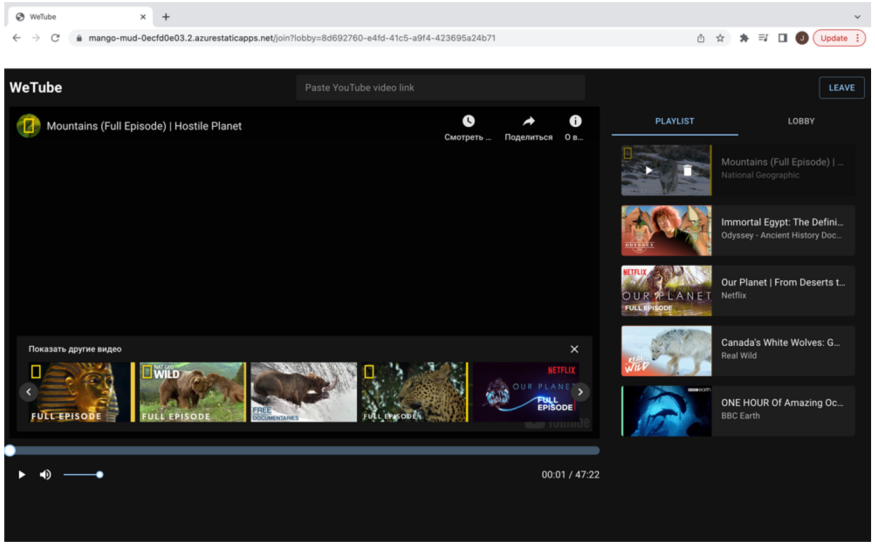


Obrázek 43: Stránka místností aplikace – playlist – otevřená v prohlížeči Safari na zařízení IPhone 11. Zdroj: vlastní zpracování

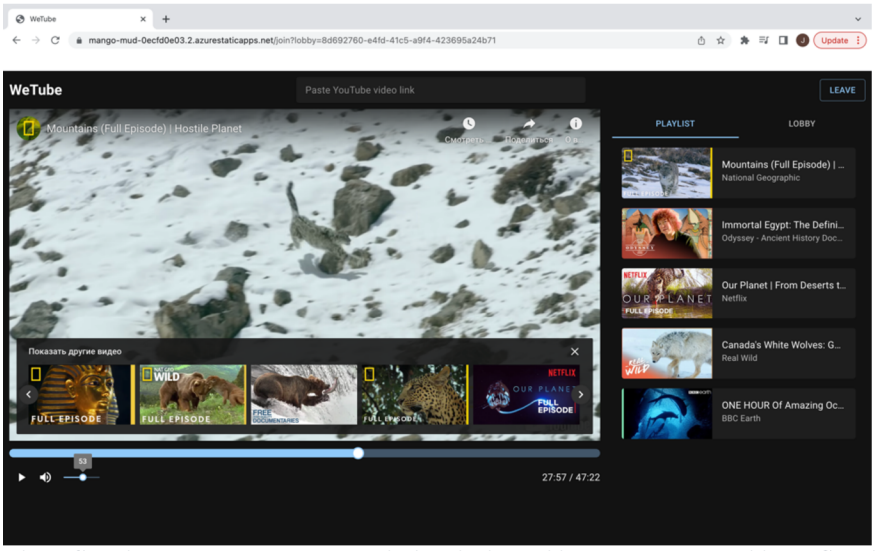
2. Mobilní zařízení Google Pixel: otestování všech scénářů za použití prohlížečů Google Chrome, Safari a Firefox proběhlo s úspěchem.
3. Notebook MacBook Pro (macOS Monterey 12.5.1): otestování veškerých scénářů s použitím prohlížečů Google Chrome, Safari a Firefox proběhlo úspěšně. Níže jsou přiloženy snímky obrazovky (č.44, č.45, č.46, č.47, č.48) při testování v prohlížeči Google Chrome.



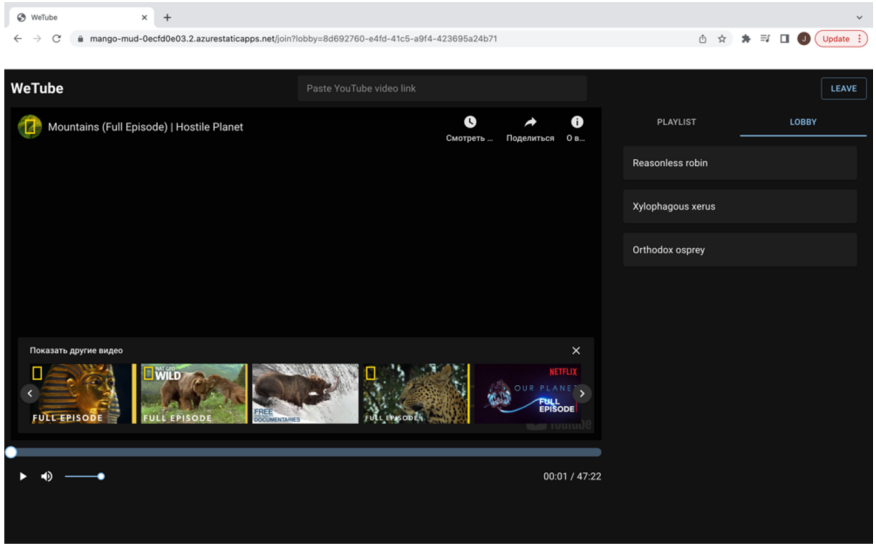
Obrázek 44: Hlavní stránka aplikace otevřená v prohlížeči Google Chrome na MacBook Pro. Zdroj: vlastní zpracování



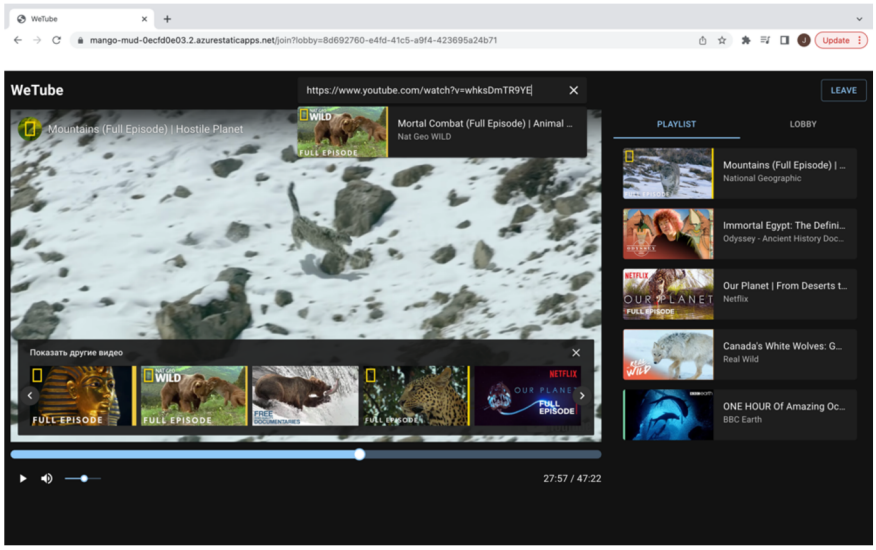
Obrázek 45. Stránka vytvořené místností – záložka Playlist aplikace otevřená v prohlížeči Google Chrome na MacBook Pro. Zdroj: vlastní zpracování



Obrázek 46. Stránka vytvořené místností – záložka Playlist aplikace otevřená v prohlížeči Google Chrome na MacBook Pro. Přehrávání videa. Zdroj: vlastní zpracování



Obrázek 47. Stránka vytvořené místností – záložka Lobby aplikace otevřená v prohlížeči Google Chrome na MacBook Pro. Zdroj: vlastní zpracování



Obrázek 48. Stránka vytvořené místností – záložka Playlist aplikace otevřená v prohlížeči Google Chrome na MacBook Pro. Přidání videa do fronty. Zdroj: vlastní zpracování

5 Výsledky a diskuse

Praktická část této bakalářské práce byla věnována vývoji responzivní webové aplikace pro společné sledování videí z platformy YouTube, která by byla spustitelná na různých typech zařízení a nebyla omezena konkrétním operačním systémem nebo webovým prohlížečem uživatele. V prvním kroku vytvoření aplikace byly definovány její základní funkcionality a nároky na užívání, jimž má odpovídat, stejně jako bylo určeno její uživatelské rozhraní pro desktopovou i mobilní verzi. Pro usnadnění správy závislostí a zajištění aktuálností kódu bylo rozhodnuto aplikovat strukturu projektu „monorepo“. Pro zajištění automatizace procesů kolem nastavení a správy monorepa byl použit nástroj Lerna. Vývoj aplikace byl uskutečněn za použití knihovny React a se začleněním technologií jako WebSocket a WebRTC, stejně jako nejlepších technických postupů.

Předkládaná práce demonstruje proveditelnost vytvoření webové aplikace, která využívá WebRTC a WebSocket k umožnění komunikace mezi uživateli v reálném čase. Tuto technologii lze využít k vytvoření široké škály aplikací, které vyžadují komunikaci v reálném čase, jako jsou videokonference nebo hraní her. Signalizační server WebSocket má však omezení, pokud jde o škálování nad jednou instancí. Při použití více instancí signalizačního serveru WebSocket je klíčové, aby byly relace uživatele a jeho připojení udržovány v rámci všech instancí serveru. WebSocket neposkytuje žádné prostředky komunikace mezi různými instancemi serveru, což může vést k nesrovnalostem v uživatelské zkušenosti. K překonání tohoto omezení se doporučuje použít vzor Pub/Sub pro signalizaci komunikace serveru mezi instancemi. Redis je oblíbené úložiště datových struktur v paměti, které poskytuje systém Pub/Sub, který lze použít k implementaci meziserverových komunikací. Redis poskytuje jediný komunikační bod pro signalizaci serverových instancí a umožňuje jim vyměňovat si informace v reálném čase. Využití systému Pub/Sub společnosti Redis může pomoci zajistit, aby byly všechny instance aktuální a byly si vědomy stavu uživatele.

Výsledkem této bakalářské práce je úspěšně vytvořená funkční aplikace, fungující na mobilních a desktopových zařízeních nezávisle na použitém operačním systému nebo webovém prohlížeči, kterou lze použít jako základ pro budoucí vývoj.

6 Závěr

Hlavní cílem této bakalářské práce bylo navrhnout a vyvinout webovou responzivní aplikaci pro společné sledování videí z platformy YouTube za použití knihovny React programovacího jazyka JavaScript. V teoretické části práce byly vymezeny pojmy webové aplikace, programovacích jazyků JavaScript a TypeScript, jejich knihovny ReactJS. Zároveň byla pozornost věnována charakteristice protokolů WebRTC a WebSocket i dalších technologií používaných při vývoji komunikací v reálném čase. V praktické části bakalářské práce byly provedeny analýza požadavků pro vyvíjenou aplikaci a výběr technologií používaných při vývoji. Kromě toho bylo také navrženo uživatelské rozhraní webové aplikace a následně byly vysvětleny její architektura a postup vývoje s příklady zdrojového kódu. V poslední části práce bylo uskutečněno testování vyvinuté aplikace, při němž bylo zjištěno, že výsledný produkt bezvadně funguje na mobilních i desktopových zařízeních v několika různých webových prohlížečích.

Vytvoření minimálního životaschopného produktu slouží jako pevný základ pro budoucí vývoj a zdokonalování aplikace. Tato bakalářská práce může být cenným informačním zdrojem pro zájemce o vytváření podobných aplikací nebo o proniknutí do světa komunikace v reálném čase a bezserverové architektury. Se získanými znalostmi je možné pokračovat v budování inovativnějších a sofistikovanějších aplikací, které využívají nejnovější webové technologie.

7 Seznam použitých zdrojů

[1] Web application (Web app). In: *TechTarget* [online]. [cit. 2023-01-14]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app

[2] WHAT IS A WEB APPLICATION?. In: *StackPath* [online]. [cit. 2023-01-14]. Dostupné z: https://www.stackpath.com/edge-academy/what-is-a-web-application/

[3] SRIPARASA, Sai Srinivas. *Javascript and Json Essentials* [online]. 1 edition. Packt Publishing, Limited, 2013 [cit. 2023-01-19]. ISBN 9781783286041. Dostupné z: https://ebookcentral.proquest.com/lib/czup/detail.action?docID=1481127

[4] STEFANOV, Stoyan a Kumar Chetan SHARMA. *Object-Oriented JavaScript* [online]. 2 edition. Packt Publishing, Limited, 2013 [cit. 2023-01-19]. ISBN 9781849693134. Dostupné z: https://ebookcentral.proquest.com/lib/czup/detail.action?docID=1192660

[5] GOODMAN, Danny, Michael MORRISON, Paul NOVITSKI a Tia Gustaff RAYL. *JavaScript Bible* [online]. 7 edition. John Wiley & Sons, Incorporated, 2010 [cit. 2023-01-19]. ISBN 9780470925607. Dostupné z: https://ebookcentral.proquest.com/lib/czup/detail.action?docID=624389

[6] NANCE, Christopher. *Typescript Essentials* [online]. 1 edition. Packt Publishing, Limited, 2014 [cit. 2023-01-21]. ISBN 9781783985777. Dostupné z: https://ebookcentral.proquest.com/lib/czup/detail.action?docID=1822800

[7] KVAPIL, Jiří. TypeScript: Úvod do TypeScriptu. In: *Itnetwork.cz* [online]. [cit. 2023-01-21]. Dostupné z: https://www.itnetwork.cz/javascript/typescript/uvod-do-typescriptu

[8] What is TypeScript?. In: *Typescriptlang.org* [online]. Microsoft [cit. 2023-01-21]. Dostupné z: https://www.typescriptlang.org/

[9] JANSEN, Remo H. *Learning TypeScript* [online]. 1 edition. Packt Publishing, Limited, 2015 [cit. 2023-01-21]. ISBN 9781783985555. Dostupné z: https://ebookcentral.proquest.com/lib/czup/detail.action?docID=4191237

[10] MORRIS, Kief. *Infrastructure as Code* [online]. 2 edition. O'Reilly Media, Inc., 2020 [cit. 2023-01-22]. ISBN 9781098114626. Dostupné z: https://books.google.cz/books?id=R24NEAAAQBAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[11] EAGLE, Alex, Kenneth CHAU, Jeff CROSS, Victor SAVKIN, Pete GONZALEZ, Justin REOCK a Benjy WEINBERGER. Understanding Monorepos. In: *Monorepo.tools* [online]. Narwhal Technologies Inc. [cit. 2023-01-22]. Dostupné z: https://monorepo.tools/#what-is-a-monorepo

[12] FERNANDEZ, Tomas. What is monorepo? (and should you use it?). In: *Semaphore* [online]. Rendered Text [cit. 2023-01-22]. Dostupné z: https://semaphoreci.com/blog/what-is-monorepo

[13] NIMISHBONGALE. How To Manage Monorepos With Lerna. In: *DigitalOcean* [online]. [cit. 2023-01-22]. Dostupné z: https://www.digitalocean.com/community/tutorials/how-to-manage-monorepos-with-lerna

[14] The Original Tool for JavaScript Monorepos. In: *Lerna.js* [online]. https://lerna.js.org/ [cit. 2023-01-22]. Dostupné z: https://lerna.js.org/

[15] BANSAL, Rachita. Introduction to Lerna. In: *Medium.com* [online]. [cit. 2023-01-22]. Dostupné z: https://rachitabansal.medium.com/introduction-to-lerna-3fb7382a4d4e

[16] YADAV, Subhash Chandra. *Introduction to Client Server Computing* [online]. 1 edition. New Age International Ltd, 2009 [cit. 2023-01-22]. ISBN 9788122428612. Dostupné z: https://ebookcentral.proquest.com/lib/czup/detail.action?docID=442141

[17] MENAA, Rima. Client-Side VS Server-Side. In: *Https://medium.com/* [online]. [cit. 2023-01-22]. Dostupné z: https://medium.com/@rimamenaa/client-side-vs-server-side-e52cd3d967af

[18] MINNICK, Chris. *Beginning ReactJS Foundations Building User Interfaces with ReactJS : An Approachable Guide: An Approachable Guide* [online]. John Wiley & Sons, Incorporated, 2022 [cit. 2023-02-02]. ISBN 9781119685586. Dostupné z: https://ebookcentral.proquest.com/lib/czup/detail.action?docID=6883629

[19] RAJPUT, Abhishek. DOM (Document Object Model). In: *Geeksforgeeks.org* [online]. [cit. 2023-02-02]. Dostupné z: https://www.geeksforgeeks.org/dom-document-object-model/

[20] Virtual DOM and Internals. In: *Reactjs.org* [online]. Meta Platforms, Inc. [cit. 2023-02-02]. Dostupné z: https://reactjs.org/docs/faq-internals.html

[21] NAWO, Arek. Top 6 React State Management Libraries For 2022. In: *Openreplay.com* [online]. [cit. 2023-01-29]. Dostupné z: https://blog.openreplay.com/top-6-react-state-management-libraries-for-2022/

[22] JSX. In: *Typescriptlang.org* [online]. Microsoft, 2012-2023 [cit. 2023-01-29]. Dostupné z: https://www.typescriptlang.org/docs/handbook/jsx.html

[23] Introducing JSX. In: *Reactjs.org* [online]. Meta Platforms, Inc. [cit. 2023-01-29]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html

[24] OGUNDIPE, Samuel. How to use TypeScript with React: A tutorial with examples. In: *Blog.logrocket.com* [online]. [cit. 2023-01-29]. Dostupné z: https://blog.logrocket.com/how-use-typescript-react-tutorial-examples/

[25] CHRISTOPHER, Anto. Understanding Web Real-Time Communication (WebRTC). In: *Https://medium.com/* [online]. [cit. 2023-01-22]. Dostupné z: https://medium.com/@anto.christo.20/understanding-web-real-time-communication-webrtc-d4cec5a43f2f

[26] Real-time communication for the web. In: *Webrtc.org* [online]. GoogleDevelopers [cit. 2023-01-22]. Dostupné z: https://webrtc.org/

[27] MANSON, Rob. *Getting Started with Webrtc* [online]. 1 edition. Packt Publishing, Limited, 2013 [cit. 2023-01-22]. ISBN 9781782166313. Dostupné z: https://ebookcentral.proquest.com/lib/czup/detail.action?docID=1389334

[28] ALTANAI. *Webrtc Integrator's Guide* [online]. 2 edition. Packt Publishing, Limited, 2014 [cit. 2023-01-22]. ISBN 9781783981274. Dostupné z: https://ebookcentral.proquest.com/lib/czup/detail.action?docID=1831905

[29] LEVENT-LEVI, Tsahi. What is WebRTC?. In: *BlogGeek.me* [online]. [cit. 2023-01-22]. Dostupné z: https://bloggeek.me/what-is-webrtc/

[30] What functionality is available through the YouTube API?. In: *Support.google.com* [online]. Google [cit. 2023-01-22]. Dostupné z: https://support.google.com/code/answer/74722?hl=en

[31] YouTube Player API Reference for iframe Embeds. In: *Developers.google.com* [online]. GoogleDevelopers [cit. 2023-01-22]. Dostupné z: https://developers.google.com/youtube/iframe_api_reference

[32] WebSocket. In: *Javascript.info* [online]. [cit. 2023-01-23]. Dostupné z: https://javascript.info/websocket

[33] TARNOWSKI, Dominik. What are Web Sockets?. In: *Medium.com* [online]. [cit. 2023-01-23]. Dostupné z: https://medium.com/@td0m/what-are-web-sockets-what-about-rest-apis-b9c15fd72aac

[34] The WebSocket API (WebSockets). In: *Developer.mozilla.org* [online]. [cit. 2023-01-23]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[35] Publisher-Subscriber pattern. In: *Learn.microsoft.com* [online]. [cit. 2023-01-28]. Dostupné z: https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber

[36] Pub/Sub Messaging: What is Pub/Sub Messaging?. In: *Aws.amazon.com* [online]. [cit. 2023-01-28]. Dostupné z: https://aws.amazon.com/pub-sub-messaging/

[37] O'RIORDAN, Matthew. Everything You Need To Know About Publish/Subscribe. In: *Ably.com* [online]. [cit. 2023-01-28]. Dostupné z: https://ably.com/topic/pub-sub

[38] Redis: A vibrant, open source database. In: *Redis.io* [online]. [cit. 2023-01-29]. Dostupné z: https://redis.io/

[39] Introduction to Redis: Learn about the Redis open source project. In: *Redis.io* [online]. [cit. 2023-01-29]. Dostupné z: https://redis.io/docs/about/