UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

REAL-TIME WEBOVÉ INTERAKTIVNÍ ANKETY

Bc. Josef Böhm

Diplomová práce

2022

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Josef Böhm**
Osobní číslo:	**I20200**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Real-time webové interaktivní ankety**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem diplomové práce je vytvořit webovou aplikaci pro realizaci interaktivních real-time hlasovacích anket určených pro dynamické prezentace, přednášky či výuku.
Teoretická část práce bude obsahovat rešerši možných způsobů pro realizaci real-time webových aplikací zahrnující seznam aktuálních a případně i historických metod a jejich vlastnosti. Dále bude v teoretické části proveden popis zvolených technologií pro realizaci webové aplikace a proveden její návrh.
V praktické části bude implementována webová aplikace, která umožňuje realizaci real-time hlasovacích anket. Každá anketa se skládá z předem definovaných anketních otázek (s možností výběru jedné nebo více odpovědí), které si nadefinuje prezentující. Prezentující má dále možnosti spuštění ankety, přechody mezi otázkami a zobrazení výsledků. Hlasující mají k dispozici URL odkaz do aplikace, kde vždy vidí pouze aktuální anketní otázku na kterou mohou odpovědět. Prezentující má k dispozici veřejnou výsledkovou obrazovku ankety, kde bude zobrazena aktuální anketní otázka a zvolené odpovědi. Po odeslání odpovědí hlasujícím dojde k real-time aktualizaci zobrazení výsledků, rozhraní hlasujících je rovněž real-time aktualizováno, pokud dojde ke změně otázky.

Rozsah pracovní zprávy: **50 – 60 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

WANG, Vanessa, Frank SALIM a Peter MOSKOVITS. The definitive guide to HTML5 WebSocket. [Berkeley, Calif.]: Apress, [2013]. Expert's voice in Web development. ISBN 9781430247401.
ZAKAS, Nicholas C. Understanding ECMAScript 6: the definitive guide for JavaScript developers. San Francisco: No Starch Press, 2016. ISBN 9781593277987.
The WebSocket API (WebSockets). MDN Web Docs [online]. © 2005-2021 [cit. 2021-9-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

Vedoucí diplomové práce: **Ing. Roman Diviš**
Katedra softwarových technologií

Datum zadání diplomové práce: **8. listopadu 2021**
Termín odevzdání diplomové práce: **20. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2021

Prohlášení

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 16. 08. 2022

Bc. Josef Böhm

Poděkování

Rád bych tímto poděkoval vedoucímu diplomové práce Ing. Romanu Divišovi, Ph.D. za konzultace a rady týkající se návrhu a realizace práce.

V Pardubicích dne 16. 08. 2022

Bc. Josef Böhm

ANOTACE

Práce se zabývá vývojem komplexní realtimové webové aplikace primárně sloužící pro vytváření a spouštění anket. Podrobně popisuje zvolené technologie včetně jejich alternativ a věnuje se rozboru architektury dílčích částí jednotlivých programů. Demonstrační část bude poukazovat na možnosti využití jak v soukromé, tak v komerční sféře. Výsledkem této práce je vyřešení všech dílčích problematik související se správnou implementací a zvolením vhodných technologií včetně dodržení a splnění předem definovaných cílů.

KLÍČOVÁ SLOVA

reálný čas, webová aplikace, websockety

TITLE

REAL-TIME WEB-BASED INTERACTIVE POLLS.

ANNOTATION

The thesis deals with the development of a complex real-time web application primarily used for creating and running surveys. It describes in detail the selected technologies, including their alternatives, and deals with the analysis of the architecture of partial parts of individual programs. The demonstration part will point out the possibilities of use in both the private and commercial spheres. The result of this work is the solution of all partial issues related to the correct implementation and selection of appropriate technologies, including compliance and fulfilment of predefined goals.

KEYWORDS

Real-time, Web application, Websockets

OBSAH

SEZNAM ZKRATEK

API	Application Programming Interface
CD	Continuous Deployment
CI	Continuous Integration
CRC	Cyclic Redundancy Check
DB	Database
DI	Dependency Injection
EDA	Event-Driven Architecture
HTTP	Hypertext Transfer Protocol
HW	Hardware
IOC	Inversion of Control
ISO	International Organization for Standardization
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVC	Model-View-Controller
ODBC	Open Database Connectivity
ORM	Object relational mapping
OS	Operational system
REST	Representational state transfer
RFC	Request for Comments
RT	Real time
RTOS	Real time operating system
SQL	Structured Query Language
SW	Software
TCP	Transmission Control Protocol
USB	Universal Serial Bus
UUID	Universal Unique Identifier
WIMS	WWW Interactive Multipurpose Serve
WS	WebSocket
WSS	WebSocket Secure

SEZNAM ILUSTRACÍ

SEZNAM TABULEK

ÚVOD

Během několika posledních let se využití realtimových technologií ve společnostech dramaticky zvýšilo díky poskytování nejvyššího možného komfortu pro návštěvníka webových stránek, a tím zajištění vyššího zisku a podílu na trhu oproti konkurenci, která v tomto odvětví zaostává. Realtimové technologie existovaly již dříve, ale jejich rozsah nebyl ani zdaleka tak markantní, jako je tomu v dnešní době. Dnes je zažitým standardem, že webová stránka automaticky aktualizuje své části bez vnějšího zásahu uživatele. Příkladem může být webová stránka, která poskytuje informace o fotbalovém utkání, kde se aktualizuje aktuální skóre nebo sledování polohy automobilů na mapě. Jejich působnost nicméně sahá mnohem dál. V dnešní době převažují webové služby, které tuto vlastnost využívají zejména proto, že je to dominující prvek hojně využívaný v herních designech a sociálních sítích. Velmi častým výskytem jsou dnes také chatboti, kteří se aktivují po návštěvě webové stránky, kde se snaží se zákazníkem udržet kontakt a vyřešit jeho problém či případné dotazy za pomocí implementované umělé inteligence, která na základě textové analýzy usiluje o poskytnutí relevantní zpětné vazby. V případě sociálních sítí panují obrovské nároky na různorodou komunikaci a dokáží uživatele téměř v reálném čase udržet na webové platformě, pokud možno co nejdéle a vyvíjet neustálý psychologický tlak, který nutí konzumenta se neustále vracet a vytvářet umělý dojem, že nevyužíváním dané služby přichází o něco důležitého. Z marketingového hlediska je nutné brát tyto koncepty jako momentálně nejlepší možné řešení pro získání potenciálních uživatelů, kteří budou realtimově orientované webové stránky využívat jak v každodenních činnostech, tak i třeba v profesionálním, či zábavním sektoru.

Samotné realtimové webové služby mají ale mnohem širší využití. Stále častěji se objevují služby, které zastupují prostředky pro komunikaci v úzkém kruhu uživatelem definovaných skupin ve formě různých interaktivních anket či dotazníků hojně využívaných například v různých zábavných soutěžích, nebo dokonce i ve školní výuce. Mezi tyto služby je možné také zahrnout prostředky poskytující podporu pro telefonní hovory, video chatovací služby, streamovací platformy nebo například peer-to-peer sdílení souborů.

Výsledná forma této práce demonstruje využití těchto služeb v jednoduchém konceptu realtimové webové ankety nabízející maximální možnou formu interakce uživatele a autorem vytvořené ankety formou obrazových a textových úloh, na které uživatel odpovídá výběrem z nabízených možností. Celá práce je designována tak, aby bylo možné využívat, pokud možno

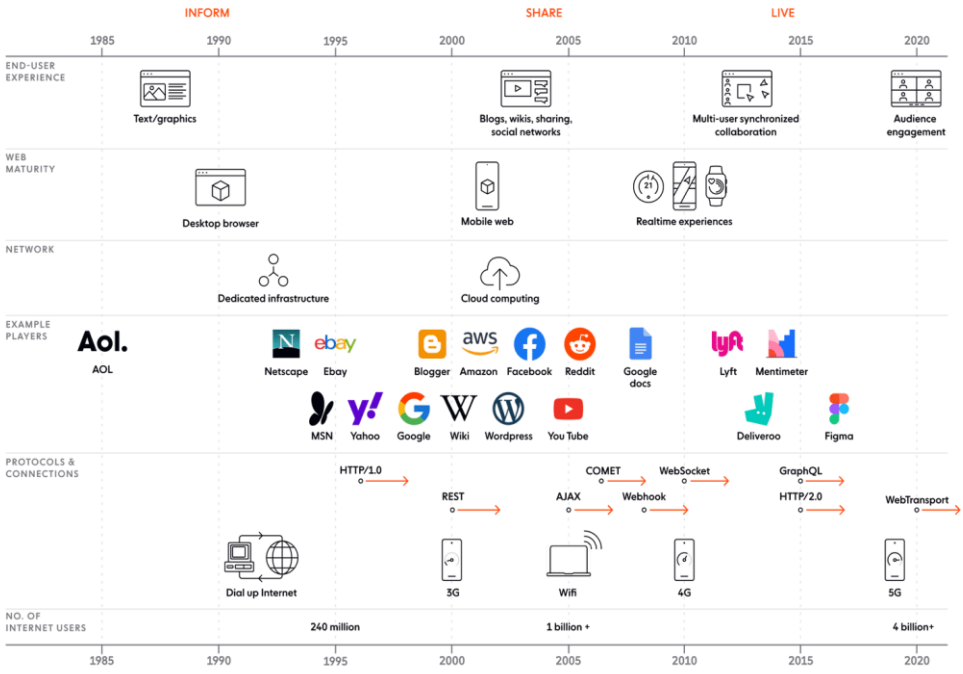
jakékoliv zařízení, které je schopné zobrazovat webové stránky a bylo připojené do sítě, kde je daná služba spuštěna.

Základní myšlenka pro použití této služby je především interakce přednášejícího a přítomných studentů, kde přednášející připraví anketu, která reflektuje již probrané téma za účelem získání zpětné vazby, zda byla probíraná látka studenty řádně pochopena nebo například pro získání statistických dat formou dotazníku. Bude zde podrobně popsána problematika realtimové komunikace, včetně celé cesty vývoje aplikace jak z teoretické, tak praktické oblasti.

1 HISTORIE REALTIMOVÝCH WEBOVÝCH APLIKACÍ

Odvětví informační technologie zabývající se problematikou webových stránek se v posledních několika dekádách výrazně změnilo. Hlavním důvodem těchto radikálních změn je několik. Dramatické navýšení výkonu počítačů a neustálý tlak vývojářů zdokonalovat prostředky pro tvorbu aplikací, které umožňují produkovat kvalitní a spolehlivý software použitelný pro koncové zákazníky, jsou jednou z těchto hlavních změn. Dráhu těchto vylepšení z velké části koriguje tlak trhu, který určuje směr preferencí odrážející se ve vytváření konkurence. Směr preferencí dále vybízí k vývoji prostředků zjednodušující existující procesy či programy nebo vylepšení těch, které trpí určitými nedostatky jak z pohledu architektury, tak výkonu a jejich použití je z tohoto aspektu omezující.

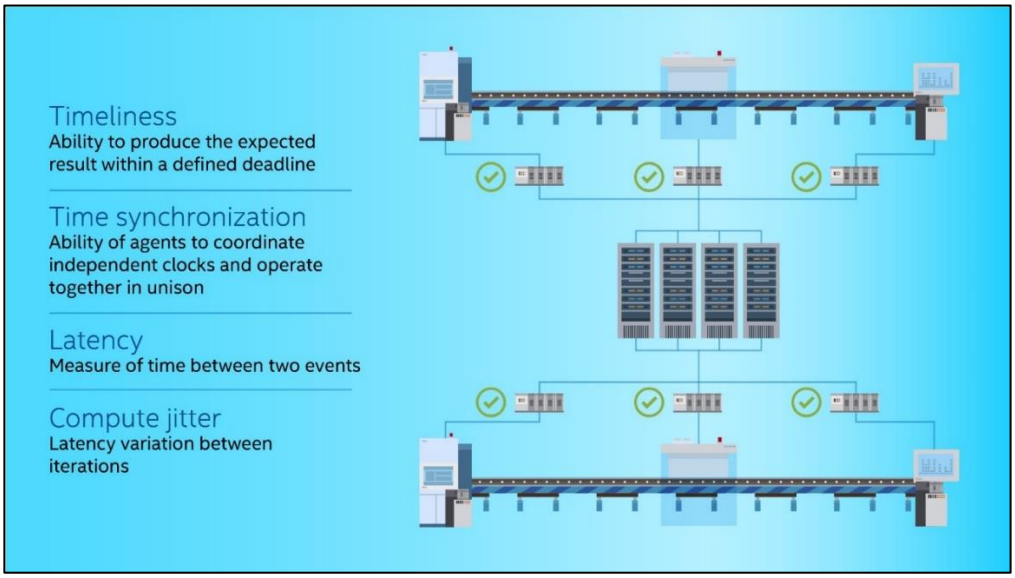
Příkladem takové technologie je realtimová webová služba, která se za dvě poslední dekády těší velké oblibě a je prakticky používaná v téměř každé webové aplikaci, která zpracovává uživatelská data v reálnem čase či poskytuje souhrn dat ostatních subjektů a dokáže jej zobrazit v momentě jejich dostupnosti. Tato kapitola se bude soustředit na představení konceptu realtimové webové technologie hojně používané v moderních aplikacích. Dále se zde bude probírat zrod této technologie a bude zde popsán minulý a současný model fungování. Na obrázku níže je možno spatřit vývoj realtimových technologií zobrazen na časové ose.



Obrázek 1.1 – Časová osa zobrazující vývoj realtimových technologií (Stichbury, 2022)

1.1 DEFINICE REALTIMOVÝCH SYSTÉMŮ

Pojem real-time označuje libovolný systém zpracovávající informace hardwarovými a softwarovými komponenty, které provádějí aplikační funkce v reálném čase a mohou reagovat na události v rámci předvídatelných a specifických časových omezeních. Mezi systémy v reálném čase řadíme například systémy řízení letového provozu, systémy řízení procesů, systémy autonomního řízení, telekomunikační systémy, řízení laboratorních experimentů a řízení chemických reakcí. Aby mohl existovat realtimový systém, musí splňovat některá kritéria. Mezi tato kritéria se řadí schopnost produkovat očekávané výsledky do určitého termínu, krátká doba odezvy a vysoká spolehlivost. Více informací na obrázku níže. (Hudec, 2016; Liu, 2000; Real-Time Systems Overview, 2022).



Obrázek 1.2 – Vlastnosti realtimového systému (Real-Time Systems Overview, 2022).

1.1.1 Soft realtimové systémy

Časové limity těchto systémů jsou přibližné. Fungují i přesto, pokud zmeškají svůj termín na dokončení úlohy. Pokud systém zmešká čas na provedení požadavku, nevede ke kritickému selhání, ale výsledkem bude určitá degradace kvality výstupních dat, čímž se automaticky snižuje užitečnost systému. Tyto systémy se nejčastěji vyskytují na webových stránkách nebo například herních online serverech. Obecně se jedná o takové systémy, kde nedodržení lhůt nevede ke katastrofickým scénářům (Hudec, 2016; Real-Time Systems Overview, 2022).

1.1.2 Hard realtimové systémy

Jedná se o systémy, kde existují absolutní časové limity. Pokud dojde k nedodržení termínu, může to mít katastrofální následky. Například pozdní příkaz k zastavení vlaku může způsobit kolizi a příliš pozdě shozená bomba může zasáhnout civilní obyvatelstvo místo zamýšleného vojenského cíle. Takové systémy se používají ve zdravotnictví, letectví nebo třeba v jaderném průmyslu (Hudec, 2016; Liu, 2000).

1.1.3 Firm realtimové systémy

Pokud systém zareaguje po uplynutí časového limitu, stává se odezva bezcenná. V takovém systému několik zmeškaných termínů nepovede k úplnému selhání, ale pokud by takových zmeškání bylo více, mohlo by to mít fatální důsledky. Tyto systémy jsou použity při streamování videí nebo v geolokačním odvětví (Hudec, 2016).

1.1.4 Odezva

Jako jeden z možných příkladů, který dokazuje existenci odezvy, je možné uvést sběr obrazových informací očima člověka a následné zpracování okcipitálním lalokem[1] a následnou reakcí prefrontálního kortexu[2]. Prvním zpožděním zde hraje roli rychlost světla. Pokud pozorovaný předmět změní svoji viditelnou vlastnost, bude chvíli trvat, než se tato informace dostane ke zpracování okem, ovšem záleží na vzdálenosti pozorovatele a pozorovaného předmětu. Následně trvá, než příslušné procesy v mozku onu změnu zaregistrují a následně poslední část mozku korektně zareaguje. Nejrychlejší reakce člověka na vnější události dosahují v nejlepších případech kolem 100 milisekund (Yuhas, 2012).

Realtimové systémy s nulovou odezvou neexistují. Vždy je zde prodleva způsobena transferem informací a jejich výpočtovém zpracováním. Časové měřítko je však relativní a při výběru vhodného systému je vždy nutné zvážit, o jaký druh problému se jedná, aby bylo možné stanovit časové termíny. V každé situaci se vyplatí využívat jiné časové nároky na navrhovaný systém. Je žádoucí cílit na komfort uživatele a efektivně využívat přidělené zdroje klienta, stejně tak zdroje poskytovatele patřičné služby (Hudec, 2016; Leggetter, 2013).

[1] Okcipitální lalok je část mozkové kůry zodpovědná za zrakové vnímání (Brabenec, 2018).
[2] Prefrontální kortex je oblast lidského mozku integrující informace do různých zdrojů (Beneš, 2010).

1.2 DŮVODY VZNIKU

Pro demonstraci fungování použijeme existující model e-mailové služby, kterou většina uživatelů dodnes používá jako primární komunikační prostředek. Jakmile se uživatel přihlásí pomocí svého účtu, na hlavní stránce vidí příchozí zprávy, které mu byly zaslány. Požadavek na fungování je takový, že uživatel by chtěl pasivně sledovat nově příchozí zprávy, aniž by musel sám žádat server o aktualizování svého obsahu. Pro zajištění této funkcionality by klient musel v pravidelných intervalech posílat požadavky na server, zda mu nepřišla nějaká nová zpráva a v případě, že ano, tak aby mu byla zaslána a zobrazena. Jedná se o jednu z technik realizující realtimovou komunikaci, které zde bude věnována samostatná kapitola.

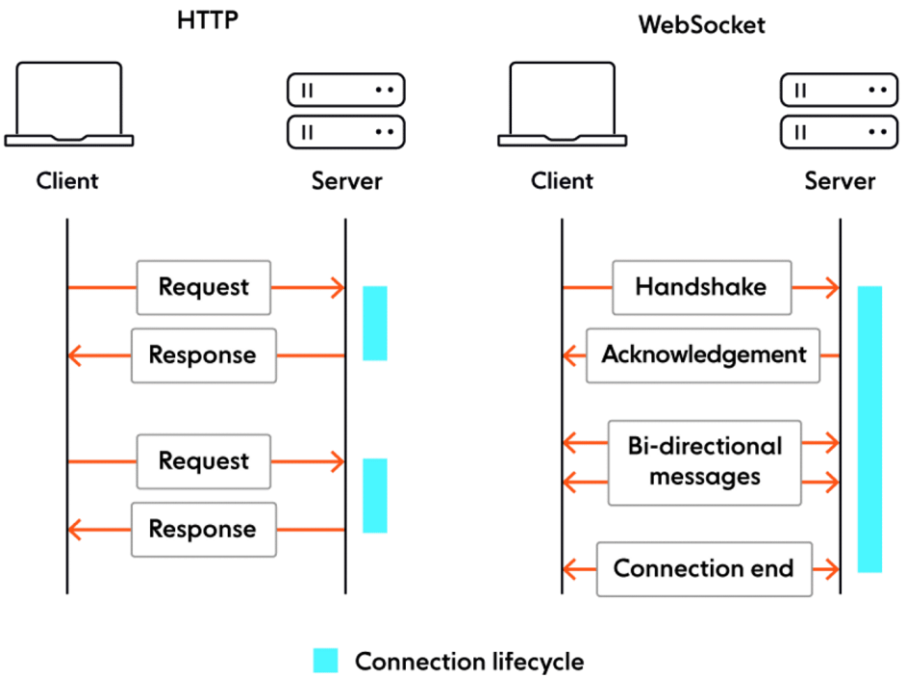
Pro několik desítek uživatelů by to fungovalo poměrně efektivně, a pokud by server nebyl nijak zvláště vytížen jinými požadavky, bylo by toto řešení použitelné. Problém tohoto konceptu nastává v momentě, kdy uživatelů využívající tuto službu je mnohonásobně více, což se v případě využívání emailového klienta dá i očekávat. Výsledkem bude neustálé zbytečné zasílání požadavků, které nedělají prakticky nic jiného, než že nutí server individuálně kontrolovat příchozí zprávy daných klientů a informovat je o skutečnosti ve formě nově příchozích zpráv. Takové řešení je velmi neefektivní a stojí server spoustu výpočetních zdrojů a velmi zatěžuje provoz na síti. Dlouhou dobu ovšem efektivní řešení této problematiky neexistovalo, a tak byli vývojáři nuceni využít existující techniky k dosažení žádoucího výsledku za cenu, že razantně stoupne provoz na síti včetně výpočetního výkonu, nebo alternativně nechá uživatele samotného danou aktualizační akci vyvolat v případě potřeby, což už ovšem opouštíme realtimový efekt jako takový.

V důsledku rostoucí poptávky na stále více interaktivní webové stránky, ať už ze strany marketingu, či od samotných vývojářů, se tento problém začal počátkem druhého milénia řešit, což byl prvopočátek realtimových webových služeb, které eliminovaly většinu popsaných problémů a nabízely možnost realizace i nad velkým počtem klientů. Samotná myšlenka tohoto návrhu je prostá, protože pouze stačí, aby existoval mechanismus, kdy bude protistrana, se kterou je navázána komunikace schopna zaslat zprávu v momentě, kdy je připravena (Leggetter, 2013; Nyklíček, 2013).

1.3 HISTORIE REAL-TIME

První oficiální architektonický model pravých realtimových webových služeb se objevuje v roce 2000 v Itálii v nezávislé výzkumné organizaci nazývané M.C.2. Rok poté je spuštěn WIMS server, jakožto první realtimově orientovaný výpočetní server na světě, který byl navržen pro intenzivní matematické výpočty dostupný skrze internet či lokální síť sloužící pro studenty jako validační prostředek pro již vytvořené matematické úlohy a je schopen najednou zpracovávat vícero uživatelských požadavků a zpětně je informovat o výsledcích. Tento koncept již zůstal prakticky doposud nezměněn a jediné úpravy, kterými si toto řešení prošlo, spočívá v celkové optimalizaci ve formě hardwarových zařízení a sofistikovanými algoritmy jak na straně klienta, tak i serveru.

V roce 2008 byl pro specifikaci HTML5 představen protokol websocket. V prosinci 2009 byl Google Chrome prvním webovým prohlížečem, který tento protokol začal plně podporovat. Jeho vznik byl reakcí na již zastaralé techniky známé jako short a long polling používaných převážně v 90. letech. Jeho oblíbenost byla značná, a proto se také tento protokol dočkal v prosinci 2011 standardizace. Na obrázku níže je porovnání websocketů a HTTP protokolu.



Obrázek 1.3 – Porovnání konektivity tradičního HTTP a Websocketů (Stichbury, 2022)

Další důležitý milník v historii realtimových webových služeb nastal v roce 2011, kdy firma Google zveřejnila zdrojové kódy technologie poskytujících API pod označením WebRTC, které umožňovaly zprostředkovat video přenosové služby skrze webový prohlížeč a umožnilo vzniku moderních streamovacích platforem známých dnes jako Twitch, YouTube nebo například Netflix či Disney+. Tato technologie se stala brzy standardem na základě schválení organizací IETF a jeho API pro webový prohlížeč bylo stanoveno organizací W3C (Leggetter, 2013).

2 WEBOVÉ REAL-TIME TECHNOLOGIE

Při výběru určité technologie vždy zvážit důležité aspekty, které budou definovat náš cíl. Existující technologie nabízející širokou škálu možností je nutné nejdříve pečlivě analyzovat a teprve poté můžeme zvolit vhodnou variantu pro svůj aktuální problém. Předchozí kapitoly se soustředily především na obecné řešení problematiky, které popisovaly konstrukty naturálního řešení problému. Jelikož je vývoj veškerých technologií velmi dynamicky měnící se prostředí, nelze očekávat, že uvedené informace budou v době čtení této kapitoly aktuální, a tudíž zde budou představeny pouze ty nejrozšířenější a nejpoužívanější frameworky.

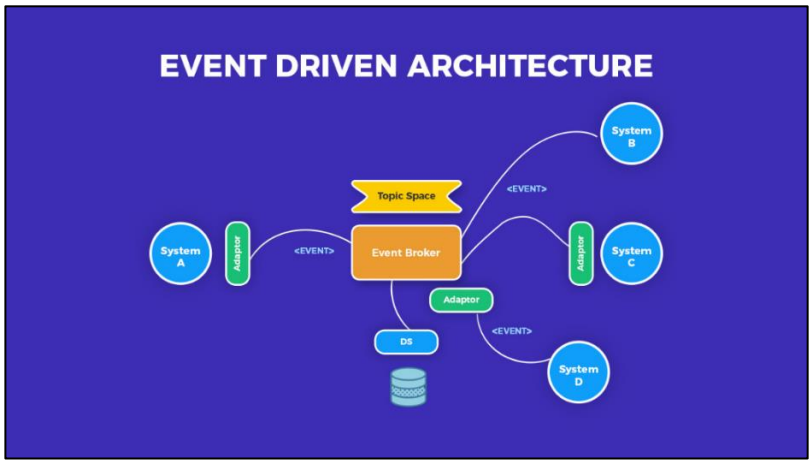
Téměř každý programovací jazyk používaný pro vývoj webových aplikací obsahuje základní prostředky zprostředkovávající realtimovou komunikaci. Tento prostředek se nazývá websocket. Websockety jsou považovány za ideální mechanismus pro realtimovou komunikaci, protože na rozdíl od http protokolu snižují režijní čas, který je nutný ke zpracování metadat serverem, aby mohl odpovědět na požadavek, a hlavně také websockety významně snižují latenci přenosového kanálu. Přesně tyto výhody jsou primární důvod jeho použití při implementaci knihoven orientované výše uvedeným směrem.

Je nutné podotknout, že v dnešní době se nativní implementace websocketů v podstatě nevyužívá, kde hlavním důvodem je zde jejich relativní složitost. Samotné použití nativní knihovny není náročné, nicméně programátor bude většinou očekávat mnohem více funkcionalit, které mu výrazně usnadní implementaci jeho aktuálního problému, nebo chybějí části, které řídí zpracování velké množství požadavků a má schopnost určité adaptace na využití síťových prostředků.

Websockety nejsou ovšem jedinou variantou pro dosažení realtimové komunikace. Existuje poměrně velké množství protokolů, které jsou orientovány vždy unikátním směrem a nabízejí řešení pro různou škálu problematik. Celá idea používání realtimových struktur v programu vychází z takzvané událostmi řízenou architekturou, která obecně popisuje problematiku celého spektra popisované problematiky. Pro pochopení celé problematiky se jedná o klíčovou znalost, na kterou se bude v následujících kapitolách navazovat (Kapoor, 2021; Toman, 2012).

2.1 UDÁLOSTMI ŘÍZENÁ ARCHITEKTURA (EDA)

Událostmi řízená architektura, dále jen „EDA“ z anglického „Event-Driven-Architecture“ je paradigma softwarové architektury podporující produkci, detekci, zpracování a reakci na události. Definice události v tomto kontextu považována jako významná změna stavu. Základní myšlenka této architektury je vztah mezi producenty a konzumenty. Důležitá je včasná reakce konzumenta na událost vytvořená producentem, avšak reakce ze strany producenta je čistě volitelná. Vznik této architektury byl podnícen normalizací této problematiky pro snadnější orientaci v aplikacích využívající asynchronní komunikaci. Struktura této architektury je na obrázku níže.



Obrázek 2.1 – Ukázka EDA architektury (Lakhwinder, 2019)

Základním pilířem, na kterém je celá architektura postavena je takzvaná slabá vazba, což je architektonický směr určující sílu závislosti jednotlivými částmi systému, a další významnou vlastností je snadná rozšiřitelnost ve smyslu přidání a odebrání dalších struktur. Existují zde čtyři logické vrstvy, které určují tok události. Každá z těchto vrstev poskytuje soubor operací definující pravidla toku události. První vrstva představuje producenta, který vytvoří událost na základě určité změny v systému, kterou kontroluje. Tyto změny mohou být vyvolány například uživatelem nebo připojeným čidlem snímající fyzikální jevy. Tato vrstva je zodpovědná za vytvoření konkrétní datové struktury, která je použita v následujících vrstvách.

Druhá vrstva zodpovídá za přenos datové struktury události směrem ke konzumentovi skrze transportní kanál, kterým může být například TCP/IP nebo třeba soubor či prosté volání metody. Veškeré události, které jsou asynchronně odesílány, se většinou řadí do front, které čekají na zpracování v téměř reálném čase. Třetí vrstva zde představuje zpracovatele, který

získá strukturu dat od producenta, která obsahují informace o události, na základě, kterých se rozpozná činnost, která má být vykonána.

Čtvrtá, finální vrstva má na starost vytvoření a vykonání činnosti včetně příslušných dat ve formátu, které jsou očekávána již samotným producentem. Zakončení tohoto cyklu je zpracování dat na straně odesílatele požadavku, který událost vytvořil a vykoná příslušné akce definované programátorem.

Důležitá část je také správné rozdělení různých způsobů zpracování událostí. Existuje členění na celkem tři různé typy, kde se typicky ve složitějších systémech používá jejich kombinace, nicméně opět se jedná o specifické případy využití pro konkrétní problematiku. Jednoduchý a zároveň první způsob je provádění kontroly změny stavu měřitelných veličin. Jakákoliv změna těchto příslušných hodnot vede ke vzniku události, na kterou je příslušným systémem následně reagováno. Typickým případem je například prosté měření teploty či událost orientovaná na změnu konkrétní proměnné.

U komplikovanějších problematik je třeba více sofistikovaného rozhodování. Takový způsob se nazývá proudové zpracování. Jedná se o kategorizace události normální a významné. Využívá se v momentě, kdy je třeba ze souboru informací vyhodnotit vznik události splňující definovaná kritéria. Příkladem může být detekce významných anomálií. Třetím typem je zde kombinace výše uvedených způsobů, kde se aplikuje výrazně složitější logika, která vychází z analýzy vstupních dat, ze kterých se vytvářejí závěry vedoucí ke vzniku události či nikoliv.

Datová struktura popisující událost je většinou v předem definovaném formátu. Její definice popisuje základní stavbu dělící se na hlavičku a tělo. Její přesná implementace je ovšem již v rukou samotného vývojáře, avšak pro dodržení probírané architektury se jedná o velmi zásadní podmínku, kterou je nutno dodržet (Hazelcast, 2014; Lakhwinder, 2019).

2.2 ARCHITEKTONICKÉ VZORY

2.2.1 Publish-Subscribe (PUB-SUB)

Základní myšlenka využití stojí na takzvaných odběratelích a vydavatelích. Podobně funguje služba odběru tvůrců na platformě YouTube, kde nově vydané video tvůrcem vytvoří událost, která odešle oznámení všem odebírajícím uživatelům o této skutečnosti ve formě

notifikace. Řešení PUB-SUB funguje s použitím middleware (prostředníka), který obdržené zprávy zasílá klientům pouze a jenom těm, kterých se daná událost týká.

Další příklad může zastávat klasické rádiové vysílání, kde rádiová stanice (publisher) odesílá svoje vysílání do příslušné antény (middleware) a poté je signál zachycen tranzistorovými přijímači na straně klienta (subscriber). Důležitý poznatek a hlavní rozdíl mezi předchozím způsobem a tímto je ten, že vydavatel nemá žádné informace o jeho odběratelích.

Signifikantní výhodou tohoto řešení je volná vazba, čímž je otevřena cesta škálovatelnosti a flexibilitě. Zde je výhoda zároveň nevýhodou, která představuje určitá omezení. Existence prostředníka nedokáže efektivně kontaktovat odesílatele s informací o doručení zprávy, čímž si nemůže být jistý, že daný obsah, který měl obdržet, skutečně dostal (Google Cloud, 2022).

2.2.2 WebHooks

Jedná se o architektonický vzor, který nemá jasně specifikovanou standardizaci. To, jakým způsobem je implementován, je čistě na zvolené knihovně, která implementuje očekávané chování. Přestože se nedá obecně determinovat, které funkce a technologie bude využívat, bude zde uveden příklad jedné z možné variant implementace.

Webhook je uživatelem definované zpětné volání typicky realizované skrze HTTP. Webhooky poskytují odlehčený způsob získání dat mezi dvěma různými API (server-server). Jsou využívány ve webových aplikacích, které typicky přijímají malé množství dat z jiných aplikací. Standardně jsou informace ve formátu JSON nebo XML, což jsou snadno zpracovatelné datové struktury. Téměř vždy se jedná o HTTP-POST požadavek.

Aby bylo možné Webhooky použít, je nutné, aby klient poskytl serveru unikátní URL a specifikoval událost. Tato URL musí být veřejná, aby byla viditelná ostatními servery. V tomto momentě se již klient nemusí opakovaně dotazovat serveru, zda událost nastala, čímž se sníží režie na síti. Klient obdrží notifikaci v momentě, kdy událost na serveru nastane a na klientské URL zpětně zašle informace o aktivované události. Webhooky by měly sloužit pouze jako oznámení o určité události, která nastala. Technicky však mohou data být obsažena v zasílané struktuře, nicméně se to obecně nedoporučuje (Red Hat, 2022).

Pokud se klientská URL dostane do nepravých rukou, existuje bezpečnostní riziko, kdy může klient dostávat falešné zprávy. Aby se tomuto zabránilo, tak nejčastější obranou je využití

TLS (HTTPS) spojení většinou s kombinací bezpečnostních tokenů obsažené přímo v URL (SendGrid Team, 2014).

Typickým příkladem je využití běžného GitHub repositáře. V momentě, kdy klient odešle změny ve svém zdrojovém kódu do svého úložiště, GitHub vyvolá událost, která kontaktuje jiný server, například CircleCI, který na událost reaguje spuštěním připravených testů (Red Hat, 2022).

2.2.3 REST Hooks

Oficiální popis této služby sám sebe nepovažuje za určitou specifikaci, ale za kolekci existujících prostředků, které ve své podstatě vytvářejí stejné chování jako již zmíněné Webhooky. Tento prostředek vznikl na základě iniciativy společnosti Zapier.

Princip fungování je prostý. Na základě sestavené URL, která obsahuje všechna důležitá data, slouží k odeslání žádosti směrem k serveru, který jej zpracuje. Důvodem tvorby tohoto řešení je reakcí na neustálé dotazování klienta na změnu zdroje. Namísto toho však klient čeká na změnu, takže strana serveru není zatížena nadbytečnými požadavky. V podstatě se jedná o Webhooky využité v REST. První z výhod je zde již opakovaně popsána, a to výrazné snížení režie jak na straně klienta, tak serveru, protože veškeré změny jsou odesílány v momentě, kdy data jsou připravena.

Nejsilnějším argumentem použití REST Hooků je jejich snadné a intuitivní využití. Opět tím, že je zde využit protokol HTTP, není nutná změna existující architektury, avšak zároveň jsou limitovány pro použití v jiných protokolech. Prvotní nastavení může být ovšem již složitější. Díky tomu je velmi jednoduché využití i v komplexních systémech, protože samotná myšlenka nepředstavuje zátěž pro budování architektury.

Jako každé řešení má ve své podstatě určitá negativa a ani zde tomu nebude jinak. Na rozdíl od způsobu, kde je klient nucen opakovaně zasílat požadavky směrem na server, se zde role obrátí. Tím je vytvářeno konzistentní dotazování směrem na klienta, a tudíž zvýšená zátěž na serveru. Toto řešení je velmi oblíbené napříč webovými vývojáři, jednak proto, že je poměrně jednoduché na použití a zejména proto, že jeho využití zdrojů je minimalizováno na co nejmenší možné hodnoty a nabízí tak efektivní řešení problému s obsluhou velké množiny klientů (Zapier, 2013).

2.3 STANDARDIZOVANÉ TECHNOLOGIE

2.3.1 WebSockets

Websockety jsou zajímavým řešením EDA hlavně proto, že jsou již v základu zabudované ve webových prohlížečích. Z technického hlediska se jedná o knihovnu vycházející ze standardních socketů. V podstatě se jedná o protokol poskytující plně duplexní komunikaci na jediném TCP spojení. V současné době podporován pouze protokol HTTP/1.1. Pro toto řešení existuje standardizace organizací IETF pod označením RFC 6455 v roce 2011 a později také samotné API bylo standardizováno konsorciem W3C (Rai, 2013).

Nevychází ze standardu HTTP, avšak implementuje funkci handshake a servery je tak zpracován jako požadavek na upgrade. Primární vlastností tohoto protokolu je poskytování interakce mezi webovým prohlížečem a webovým serverem. Jeho nespornou výhodou je nižší režie a velmi výrazně usnadňuje realtimové přenosy. Komunikace je realizována skrze TCP na portu 80 či 443, což poskytuje další výhodu, a to vyhýbání se blokaci pravidel ve firewallu, protože tyto porty jsou standardně otevřené a nejsou blokované jako ostatní newebové připojení (Wang, 2013).

Další významnou výhodou je jeho podpora širokým spektrem technologií, téměř napříč všemi dostupnými webovými prohlížeči od softwarových gigantů, jako je Google, Microsoft či Apple. Ovšem podpora samotným klientem není dostatečná a je třeba, aby server umožňoval použití této technologie také. Jeho definice v URI je definováno jako „ws“ představující nešifrované spojení a „wss“ jakožto spojení šifrované, a zbytek pravidel vychází z obecné URI syntaxe.

Jako každé řešení mají websockety určité nevýhody. Webový prohlížeč musí být plně kompatibilní s HTML5. Websockety se po ztrátě připojení automaticky neobnoví a je třeba tento problém vyřešit buď vlastní implementací, nebo využitím knihovny třetí strany (Fette, 2011; Melnikov, 2011).

2.3.2 Server-Sents-Events (SSE)

Velmi jednoduchá myšlenka, která naturálně poskytuje řešení problému, pokud není vyžadována obousměrná komunikace. Pokud je potřeba na straně klienta aktualizovat informace nejrůznějšího typu, zašle se ze serveru zpráva, která je následně zpracována

příslušnými metodami na straně klienta. Výrazně se tím redukuje složitost architektury systému, jelikož není třeba definovat žádné speciální protokoly a další související nastavení. Většina systému lpí právě na použití obousměrné komunikace, tudíž použití tohoto řešení má smysl pouze ve speciálních případech, kdy je to skutečně potřeba a nepředpokládá se, že by se měl původní návrh měnit.

Z hlediska bezpečnosti se tohoto řešení nejeví jako příliš vhodné. Obousměrné systémy pracují na bázi autentizační metodiky, kdežto zde tato absence může znamenat završení použití i přes některé zmíněné výhody. Tím, že zde není zpětná vazba od klientů, zda obdrželi zprávu, může server zbytečně využívat své zdroje snahou aktualizovat klienty, kteří se již dávno odpojily, nebo zprávu nedostali z důvodu ztráty konektivity celou, což vede ke ztrátě informací (Babel, 2017).

2.3.3 WebRTC

Na základě rostoucí popularity používání webových prohlížečů bylo nutné vyvinout JavaScriptové API, které by umožňovalo implementaci webových služeb pro realtimový oboustranný přenos multimediálního obsahu. Dříve toto bylo možné jen za pomocí speciálního softwaru, který se instaloval do počítačů. Dnes je dostačující mít nainstalovaný pouze webový prohlížeč bez dalších přídavných modulů.

Rozhraní API WebRTC plní několik klíčových funkcí, včetně přístupu k multimediálním datům ze zařízení, jejich záznamu, zahájení, sledování a ukončení spojení P2P mezi zařízeními prostřednictvím prohlížečů a usnadnění obousměrného přenosu dat prostřednictvím více datových kanálů. WebRTC pracuje na třech primárních komponentách, aby bylo možné efektivně iniciovat interakci P2P.

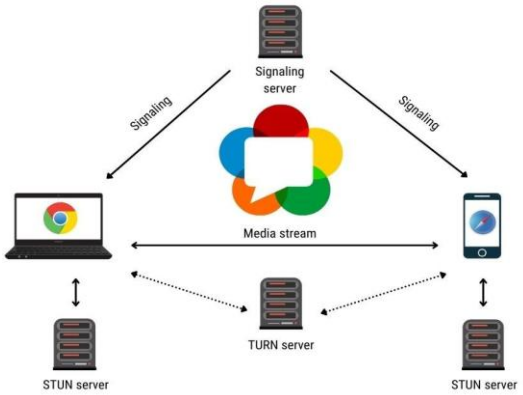
Jedná se vlastně o signalizační/komunikační protokoly:

- SIP (Session Initiation Protocol),
- SDP (The Session Description Protocol),
- ICE (The Interactive Connectivity Establishment protocol).

Ve většině případů WebRTC spojuje uživatele realtimově ze zařízení do zařízení pomocí komunikace P2P. V situacích, kdy jsou uživatelé v různých sítích, které mají firewally s překladem síťových adres (NAT), které brání RTC, lze WebRTC používat ve spojení se servery Session Traversal Utilities for NAT (STUN). To umožňuje přeložit danou IP adresu na

veřejnou, aby bylo možné navázat vzájemná spojení. Celá konektivita je zobrazena na obrázku níže.

Existují však i sítě, které jsou natolik omezené, že k překladu IP adres nelze použít ani server STUN. V těchto případech se WebRTC používá se serverem Traversal Using Relays around NAT (TURN), který předává provoz mezi uživateli a umožňuje jim připojení. K nalezení nejlepšího spojení se používá protokol ICE (Brando, 2022; Demir, 2021; Dutton, 2020; Zlatkov, 2018).



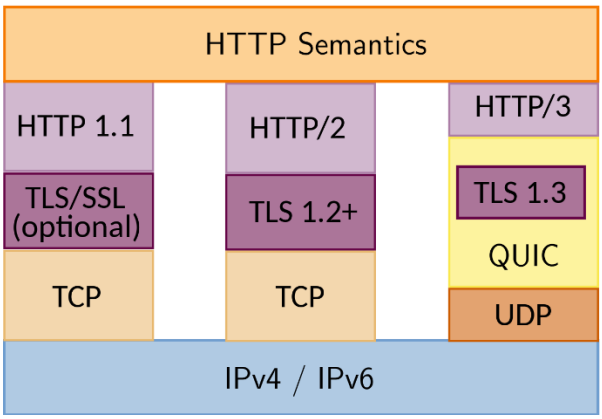
Obrázek 2.2 – Schéma komunikace pomocí WebRTC protokolu (Brando, 2022)

2.3.4 WebTransport

Jedná se kompletně o novou specifikaci, která je alternativou k websocketům. Tento nový protokol nabízí zabezpečený multiplexovaný realtimový přenos a definuje API pro spolehlivé a nespolehlivé odesílání dat. Při spolehlivém přenosu dat je odesílatel informován o úspěchu či neúspěchu přenosu, přičemž neúspěšné přenosy jsou obvykle odesílány znovu. Při nespolehlivém přenosu neexistuje žádné potvrzení o úspěšnosti přenosu a tyto pakety jsou nenávratně ztracené. Nespolehlivý přenos se často používá například pro streamování videa, kde jde především o rychlost a kde je menší ztráta dat přijatelná.

WebTransport umožňuje vytvářet více streamů prostřednictvím jediného připojení a je méně náročný na zdroje při vytváření spojení. Zatímco websockety používají protokol HTTP/1.1, WebTransport pracuje s HTTP/3, což je nadcházející verze transportního protokolu používaného na webových stránkách. HTTP/3 používá pro výměnu dat na transportní vrstvě protokol QUIC. Jedná je experimentální síťový protokol od společnosti Google založený na UDP, který poskytuje kryptografickou ochranu podobně jako TLS a lze jej otevírat a zavírat

bez větší režie. Existující verze protokolu HTTP je na obrázku níže. (Posnick, 2022; Thomson, 2022).



Obrázek 2.3 – Porovnání jednotlivých verzí HTTP protokolu (Sedrubal, 2022)

2.4 SPECIÁLNÍ PŘÍPADY – POLLING

Jedná se o techniky, které nemají definované standardy. Na rozdíl například od websocketů, kde existují funkce umožňující sofistikovaně nakládat s přidělenými zdroji, pokud možno co nejefektivněji, se tyto techniky využívají jednoduše jako cyklické dotazování bez hlubšího rozhodovacího mechanismu (Ganesan, 2018; Aggarwal, 2021).

2.4.1 Short polling

Jedná se o techniku, kdy se klient cyklicky většinou v pevně daných časových intervalech dotazuje serveru. Typicky se využívá způsobem, kdy v případě kladného vyřízení jsou v odesílané odpovědi serverem přítomna data, a pokud nejsou, je odesláno prázdné tělo odpovědi.

Tato technika je velmi jednoduchá, ale spotřebovává většinu zdrojů na straně klienta než na serveru. Také je nutné počítat s určitou odezvou. Příkladem použití může být dotazování serveru na data, která produkuje v nepravidelných intervalech. V momentě, kdy jsou data připravena, jsou v následujícím dotazu klientovi odeslána a v opačném případě nezíská nic. Klient vyčká několik sekund a dotaz opakuje. Nevýhoda tohoto řešení je, že pokud jsou data na serveru připravena, klient se do nedozví a musí se čekat, než se opět dotáže. Obecně platí, že toto je technika z 90. let a přelomu tisíciletí, kdy ještě nebylo nic lepšího. Dnes je to již překonané a nedoporučuje se používat.

2.4.2 Long polling

Tato technika využívá podobného scénáře jako předchozí způsob. Klient odešle žádost o data směrem na server. Server nicméně nějakou dobu vyčkává. Pokud je přítomen čas, za který je nutné žádost vyřídit, čeká definovaný časový interval. V opačném případě čeká do doby, než jsou data připravena. Jeho majoritní nevýhoda pramení v mnohonásobně vyšší spotřebě přidělených zdrojů na serveru.

2.5 ZÁVĚR

Z výše uvedených existujících řešení je již na první pohled patrné, že nebude existovat jeden univerzální způsob, který by vyřešil všechny problémy. Hodně záleží, co se od daných systémů očekává a zda se orientují směrem po výkonnostní stránce, bezpečnostní politikou či spolehlivosti zasílání informací. Porovnání jednotlivých probraných technik jsou v tabulce níže.

Pro účely tohoto projektu bylo nutné, aby požadavky mezi klientem (webový prohlížeč uživatele) a serverem (webová aplikace) byly garantovány. To zajistí protokoly postavené na TCP. Protože prezentující řídí průběh kvízu, musí klienty prostřednictvím serveru notifikovat o určitých událostech, z čehož automaticky vyplývá potřeba využití obousměrného připojení. Z analyzovaných možností toto zajistí websockety a webtransport. Protože jsou websockety vyspělou technologií, která je široce podporována na desktopu i na mobilních zařízeních, byly použity v této práci jako adekvátní řešení pro zajištění realtimové komunikace. Webtransport je aktuálně rozpracovaná technologie, která má potenciál v budoucnu nahradit websockety.

Tabulka 2.1 – Porovnání probíraných realtimových protokolů a technologií

Protokol / Technika	Využívá / Realizovatelné	Standard	Poznámka
WebSockets	TCP, HTTP-like	IETF, W3C	Obousměrná komunikace skrze TCP
			Navržen pro webové prohlížeče a servery
			Výhodné pro nízký overhead
			Podporován většinou prohlížečů
Webhooks	URI, HTTP	–	Uživatelsky definované zpětné volání skrze HTTP

Protokol / Technika	Využívá / Realizovatelné	Standard	Poznámka
			Reaguje na událost
			Požadavky se zasílají na speciální URI
			Umožňuje spouštění událostí v reálném čase
REST Hooks	HTTP	Zapier	Vylepšené strategie webhooku
			Využívá REST API
			V podstatě webhook v REST
Pub-Sub	–	–	Model producent a odběratel
			Obousměrný
			Vrstva prostředníka mezi klientem a serverem
			Důraz na volnou vazbu
SSE	HTTP, HTML5, DOM	WHATWG, W3C	Server neustále odesílá aktualizace klientovi
			Jednosměrný
WebRTC	UDP	IETF, W3C	P2P komunikace
			Využívaný převážně pro multimediální data
			Obousměrný
			Podporován většinou prohlížečů
Webtransport	HTTP, QUIC	IETF, W3C	Vylepšená alternativa websocketu
			Výhodné pro nízký overhead
			Obousměrný
			Pouze klient-server
Short polling	HTTP, TCP	–	Opakované dotazování
			Pevně dané časové intervaly
Long polling	HTTP, TCP	–	Opakované dotazování
			Odpověď po uplynutí času nebo přítomnosti dat

3 ROZBOR FUNKCIONALIT PROJEKTU

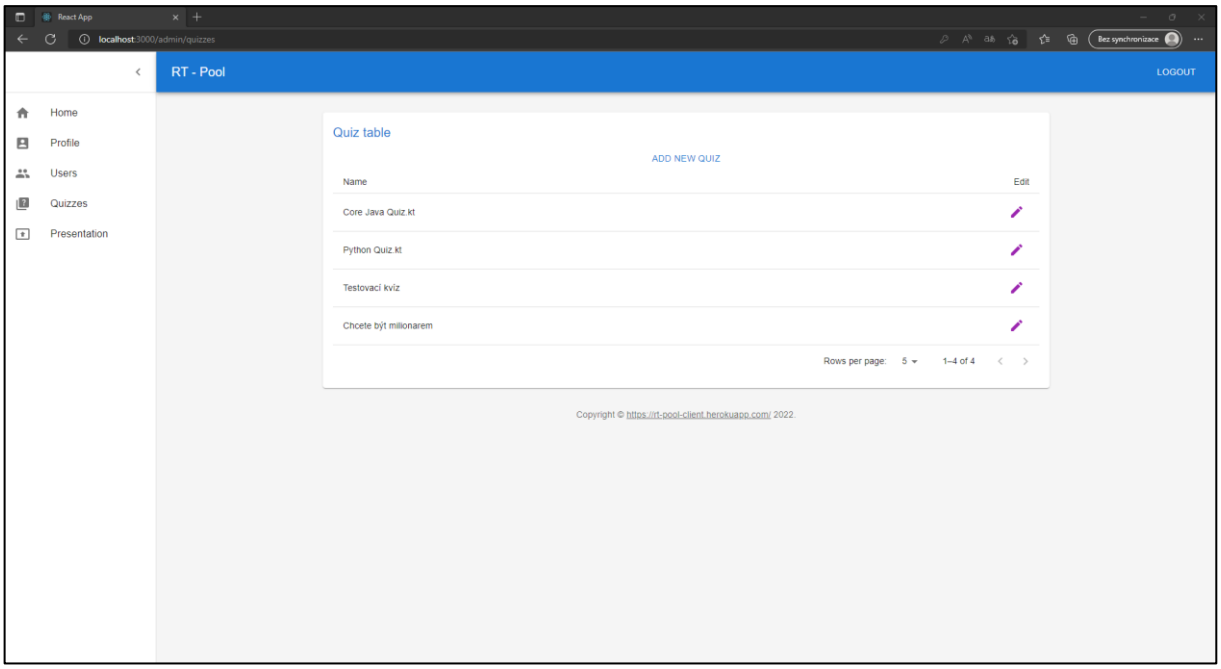
Celý projekt je postaven na jednoduché myšlence vytvořit interaktivní kvíz použitelný v libovolném odvětví. Konkurence podobné služby nabízí již roky, nicméně ne všechny jsou vhodné pro použití v konkrétních případech. Většina komerčně dostupných služeb své fungující části často zpoplatňuje nebo je dokonce neimplementují vůbec. Tento projekt byl speciálně navržen pro použití na akademické půdě, kde každý vyučující může využít jeho unikátnosti a nemusí se obávat, že by narazil na problémy jak z hlediska bezpečnosti, tak třeba právních přestupků většinou definovaných v EULA. Tato část se bude zaměřovat představení všech dostupných komponent, které se v projektu nacházejí. Každá z těchto částí bude podrobně probrána a následně budou vysvětleny její možnosti a způsoby manipulace.

Zde je uveden krátký výčet funkcionalit, které budou podrobně vysvětleny:

- operace s kvízy, otázkami a s odpověďmi,
- obsluha spuštěné prezentace,
- administrace uživatelských profilů.

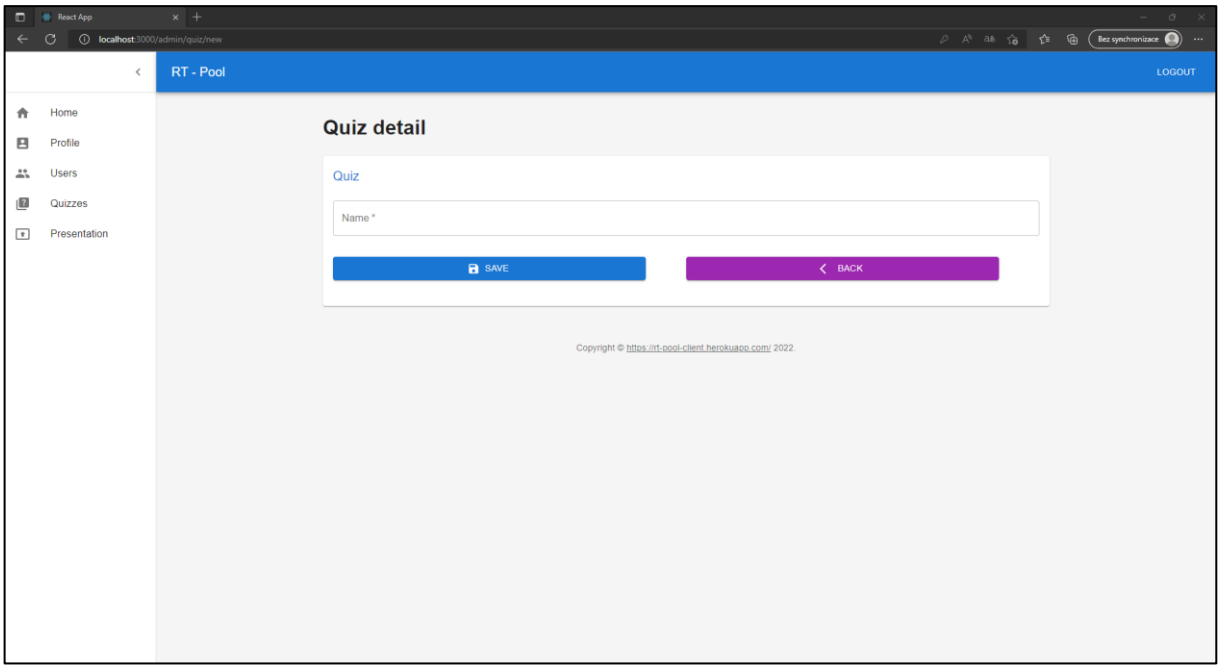
3.1 MANIPULACE S KVÍZY

Jedná se o jednu z nejdůležitějších záložek v aplikaci. Uživatel zde vidí existující kvízy, které v minulosti vytvořil, včetně jejich možností nového založení či modifikace těch, které jsou zobrazeny v tabulce. Vytvořené kvízy se zobrazují pouze uživatelům, kteří jej vytvořili, vyjma administrátora, který má možnost vidět všechny. Tím je zajištěna plná izolace od ostatních uživatelů. Budoucí možnost práce s těmito vytvořenými strukturami by mohla být implementována funkcionalita umožňující sdílení napříč jednotlivými uživateli, či dokonce skupinami. Nicméně pro tyto případy by to bylo zcela nadbytečné, protože již zde existuje možnost exportu kvízu do souboru, který je možné sdílet.



Obrázek 3.1 – Seznam aktuálně vytvořených kvízů

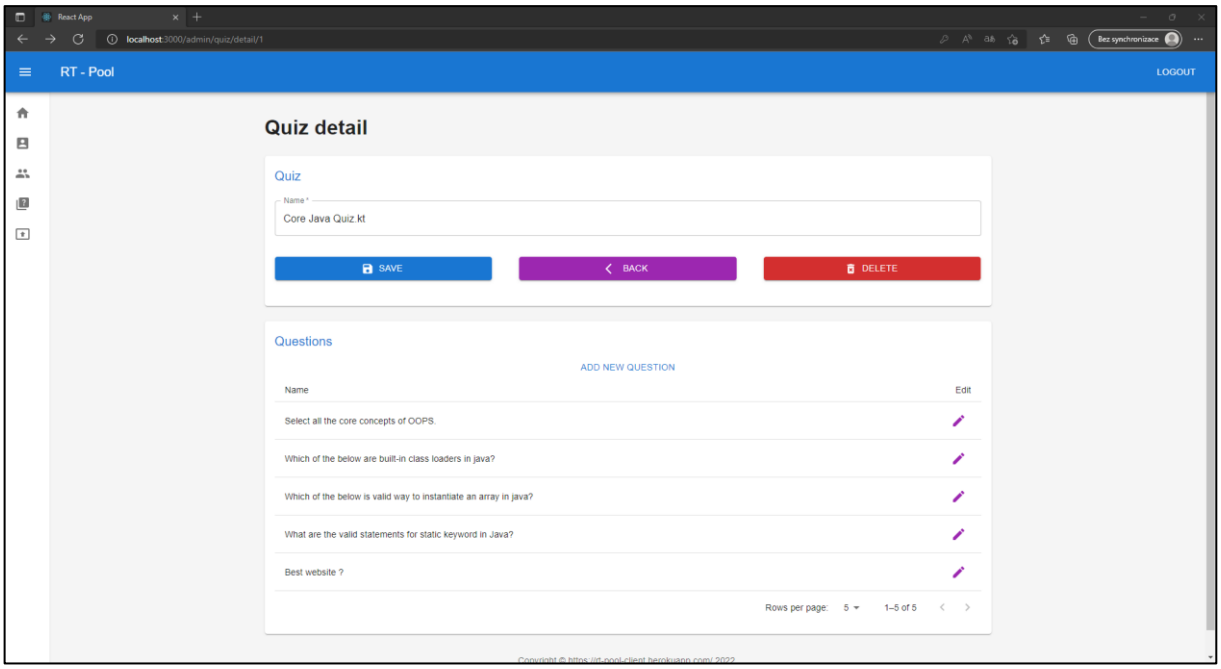
Pokud se uživatel rozhodne vytvořit nový kvíz, musí mu přidělit nové, unikátní jméno a přidat jej do seznamu. Po odeslání požadavku o vytvoření bude přesměrován na patřičný formulář. Je již patrné, že dalším krokem bude vytvoření jednotlivých otázek, proto bude uživatel po vytvoření přesměrován na stránku, která umožňuje editaci celého kvízu. Na obrázku výše je pro demonstraci zobrazen seznam aktuálně vytvořených kvízů.



Obrázek 3.2 – Formulář pro vytvoření nového kvízu

Na obrázku výše je zobrazen prázdný formulář. Jako demonstrační ukázku budeme pracovat s již existujícím kvízem, který má již vytvořené některé otázky. Pokud bychom založili kvíz úplně nový, spodní tabulka by byla jenom prázdná. Při vývoji byl kladen důraz na jednoduchost a přehlednost. Většina komerčních řešení poskytuje tvorbu kvízu většinou jenom na jedné obrazovce a při výskytu poměrně velkého množství různých elementů se stránka může jevit složitě a uživatele dokáže odradit. Klasická cesta vytvoření kvízu je v této aplikaci velmi prostá.

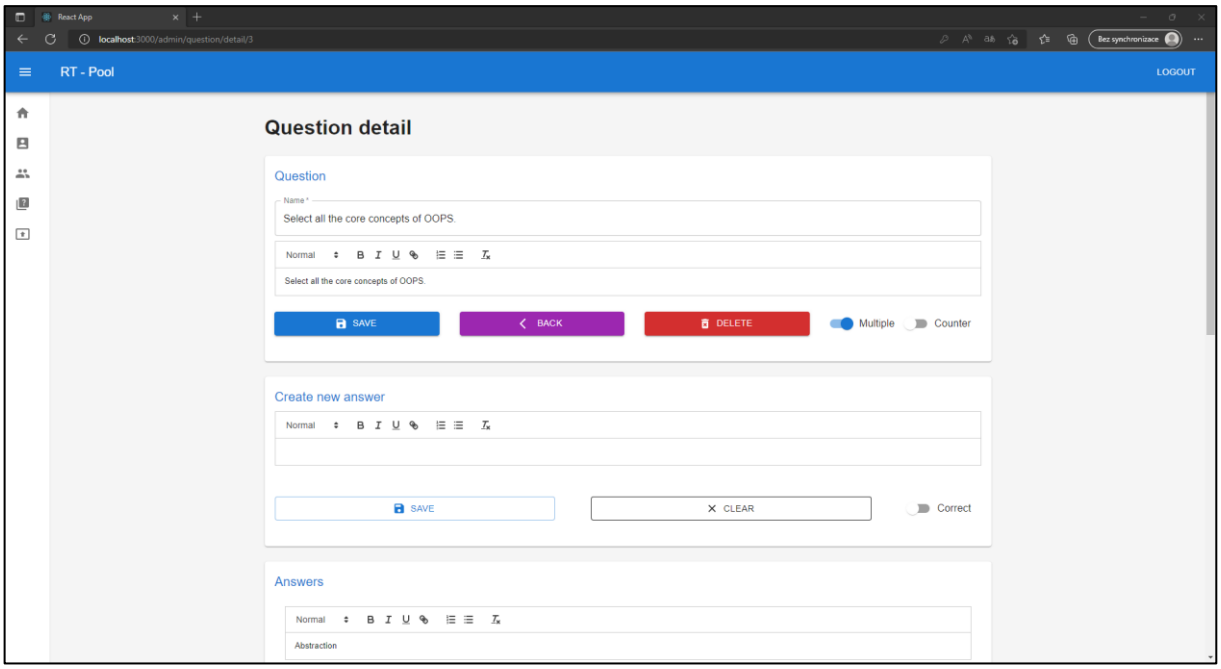
Editace kvízů má separované rozhraní, které nabízí editaci vždy jedné jediné struktury, kde je následně provedeno přesměrování na editaci jednotlivých otázek. Finální přesměrování by vedlo směrem na definování odpovědí na konkrétní vybranou otázku. Panel umožňuje celý vytvořený kvíz smazat, avšak za dodatečného varování, pokud již existují otázky. Tabulka zobrazující otázky opět umožní přesměrování do editačního okna. Pro demonstraci je na obrázku níže již vytvořený kvíz.



Obrázek 3.3 – Editační panel kvízu

Poslední částí vytvoření kvízu je editační panel pro vytvoření otázky a jednotlivých odpovědí. V horní části můžeme spatřit dvě textová pole obsahující text související s názvem otázky. Důvodem existence těchto dvou polí je vyřešení problému s identifikací otázek. Sekundární textové pole slouží jako hlavní text, který je zobrazován hlasujícím uživatelům, kdežto vrchní část slouží jako primární identifikátor otázky, který je viditelný v seznamu v momentě, kdy je spuštěna prezentace na straně prezentujícího. Tento text je libovolný, textová pole v tomto případě na sobě nemají žádnou provázanost.

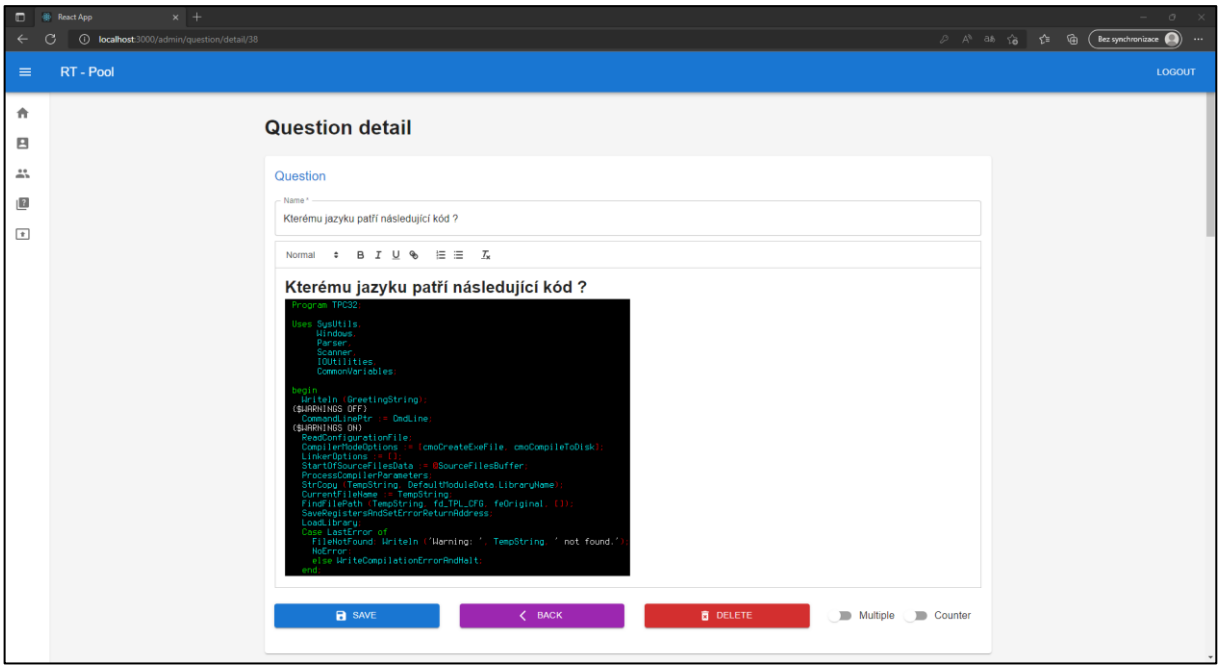
Skutečnost, která tento kvíz dělá unikátním na rozdíl od konkurence, je možnost vložení různých netextových elementů do otázky. Většina již hotových řešení se soustředí na grafické rozhraní, kde jsou veškeré elementy pevně dány a uživatel má velmi málo prostoru pro dlouhé popisy otázek a téměř většina z nich vůbec nepodporuje přidání obrazových elementů, či odkazů, nebo podporuje třeba jen základní formátovaní textu. Tyto problémy, které bylo třeba vyřešit, což byl jeden z klíčových požadavků na vývoj, byly elegantně vyřešeny prostřednictvím volně dostupné komponenty, který funguje jako prostý HTML editor, kde jsou tyto elementy zobrazovány jako separované webové komponenty a vykresleny dle standardu webového prohlížeče. Na obrázku níže je zobrazen editační panel.



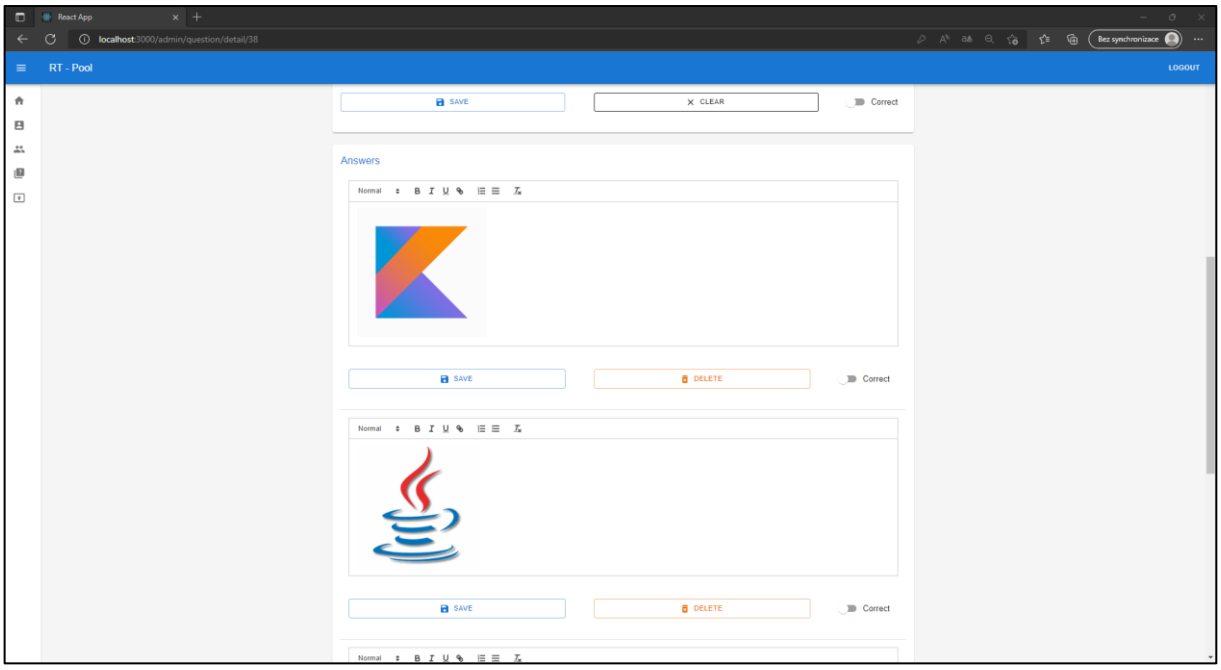
Obrázek 3.4 – Editační panel vybrané otázky

Existuje zde prakticky podpora všech grafických prvků, jako je například použití animací, zkopírovaného, předem naformátovaného textu, nebo například odkazů, které mohou ukazovat na obrázky uložené na internetu a nechybí zde ani podpora videí, též za předpokladu využití odkazu. Tím je kreativita uživatele ve své podstatě neomezena, nicméně je nutné zohlednit základní fakt, a tím je podpora mobilních zařízení, kde se rozložení stránky nemusí vždy úplně dobře zobrazit, přesto že tato podpora je zde již zabudována jak pro telefony Android, tak i pro uživatele iOS zařízení, včetně tabletů a osobních počítačů.

Formátovaný text je zejména důležitý, pokud je nutné zobrazovat například části zdrojového kódu nebo jiných textových pasáží, které by byly zapotřebí kopírovat a dočasně s nimi pracovat v jiném editoru. Další prvek, který je důležitý, co se týče nastavení otázky jako takové, je možnost volby, zda se jedná o otázku, kde odpovědí může být více a zda budou odpovědi uživatelům vůbec zobrazeny. Spodní část formuláře již jen replikuje zmíněné mnohotvárné textové pole ve formě HTML editoru, kde jsou funkcionality jak u otázky totožné s možností označení správně otázky. Na obrázcích níže jsou již využity grafické elementy a formátovaný text.



Obrázek 3.5 – Ukázka grafického zobrazení vytvářené otázky

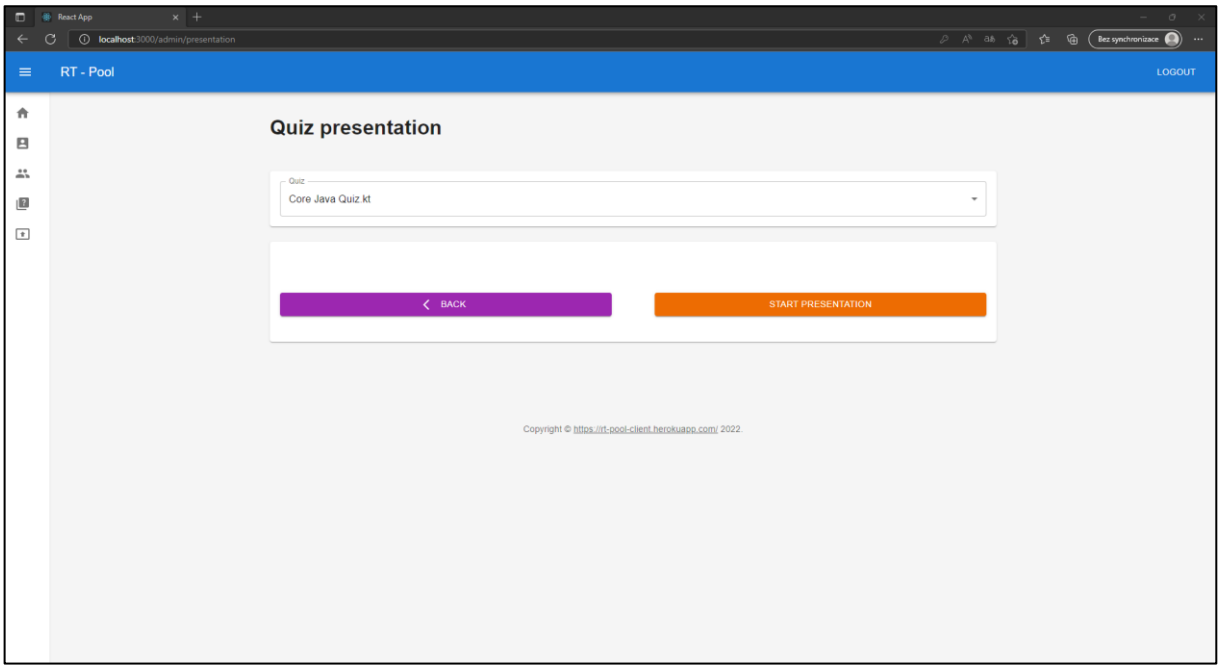


Obrázek 3.6 – Ukázka vložení obrázku do odpovědi

3.2 PREZENTAČNÍ MODUL

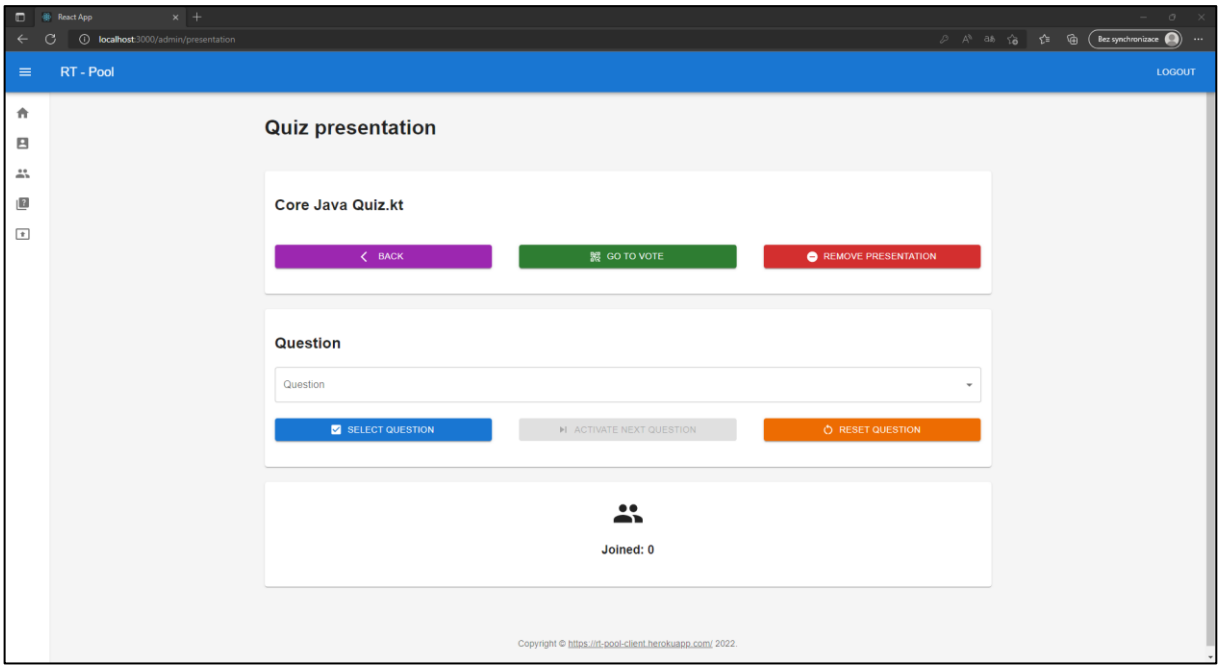
Tato komponenta zodpovídá za průběh prezentovaného kvízu. V úvodní části se nachází selektor s kvízem, kde si uživatel vybere kvíz, který chce spustit, což je zobrazeno na obrázku níže. Každý uživatel smí mít spuštěn pouze jeden kvíz. Každá taková instance se v dynamické

paměti udržuje, dokud je aktivní. Tím, že se na straně serveru nahraje zrcadlo dané struktury prezentace, nic nebrání tomu, aby uživatel modifikoval i právě aktivní kvíz, nicméně změny se projeví až po jeho znovuspuštění. Idea spuštění právě jednoho kvízu na uživatele je zakořeněna ve specifikaci projektu. Situací, kdy je potřeba mít spuštěno více instancí není, tudíž není ani důvod takové k takové vlastnosti.



Obrázek 3.7 – Selektor pro výběr kvízu

Další obrazovkou, na kterou je uživatel přesměrován v momentě, kdy spustí kvíz, je hlavní ovládací panel, který poskytuje veškerou kontrolu nad touto vytvořenou instancí. Tato hlavní obrazovka je zobrazena na obrázku níže.

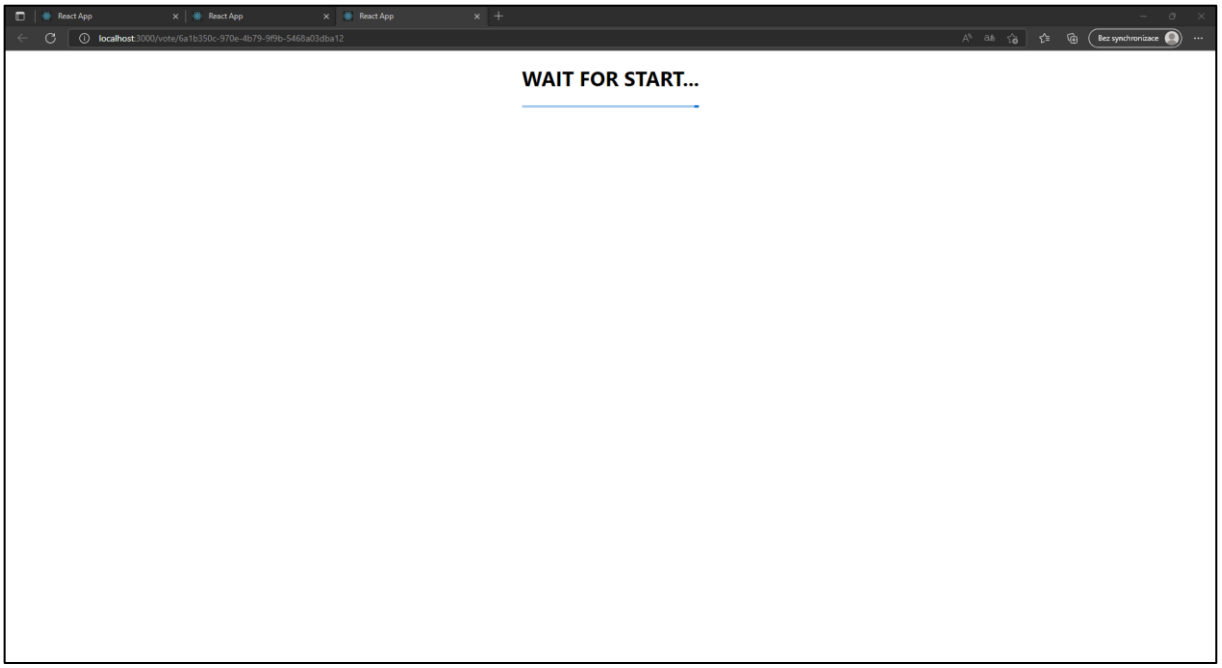


Obrázek 3.8 – Hlavní ovládací panel spuštěného kvízu

První část ovládá životní cyklus prezentace, kde je možné se buď pouze vrátit na editační panel kvízu, nebo jednoduše současnou prezentaci ukončit. Aby byli uživatelé schopni se do kvízu připojit, je nutné jim poskytnout příslušnou URL, která povede na již aktivní část prezentace, viz. obrázek níže. Každé nově vytvořené prezentaci je vygenerován speciální identifikátor, který odlišuje všechny běžící instance prezentací od jednotlivých prezentujících.

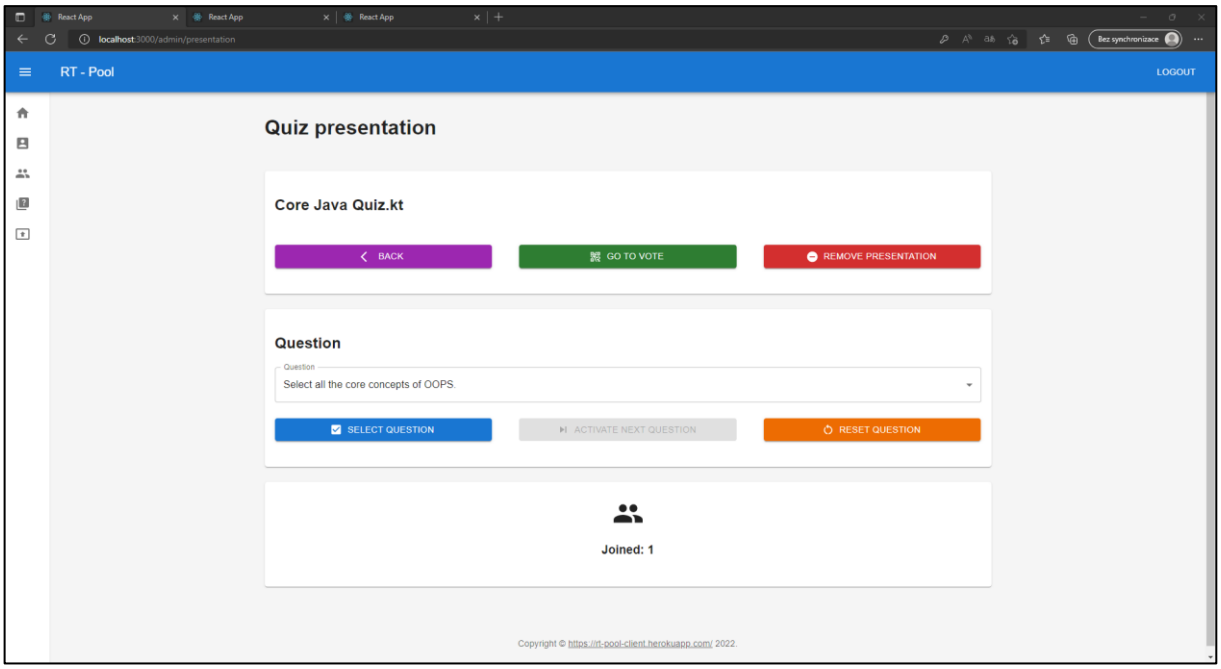
Obrázek 3.9 – Ukázka vygenerovaného QR kódu pro přihlášení do aktivního kvízu

Pokud bude prezentace spuštěna v rámci přednášky libovolného předmětu ve velkém sále s přítomným dataprojektorem, je předpokládáno, že většina studentů bude používat mobilní zařízení pro připojení do prezentace, a pro tyto účely je vždy vygenerován QR kód, který si mohou načíst pomocí integrované kamery v chytrém telefonu a jsou přesměrovány do prohlížeče, kde budou ve frontě čekat na další kroky prezentujícího. V případě, že student bude chtít využít počítač, bude mu poskytnut odkaz, který mu připojení umožní. Úvodní obrazovka zobrazena na obrázku níže.



Obrázek 3.10 – Úvodní obrazovka čekatele na spuštění prezentace

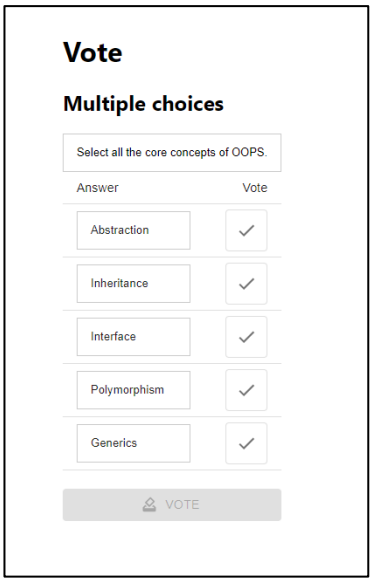
Po otevření vygenerovaného odkazu se uživatel dostane do úvodní části, kde vyčkává, dokud prezentující neprovede další kroky. Každému uživateli se vygeneruje jedinečný identifikátor, který se uloží do jeho webového prohlížeče. Toto UUID je použito následně při každém pokusu o připojení do prezentace. Tím je zaručeno, že uživatel po restartování zařízení nebo prostého zavření a znovuotevření webového prohlížeče neztratí svůj dosavadní postup. Pokud si otevře ve svém přihlížeči novou kartu, ve které se bude snažit do kvízu připojit, bude použito jeho vygenerované UUID a zobrazí se mu již stejný obsah.



Obrázek 3.11 – Prezentace s jedním připojeným účastníkem

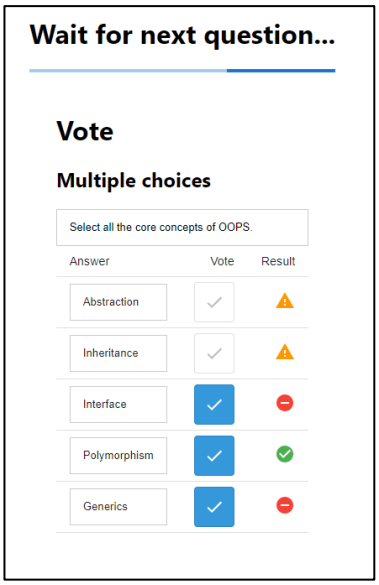
Zde se pomocí selektoru vybere otázka, která bude aktivována. Tím, že existuje možnost vložení pouze obrazového elementu místo textu, je zde vidět využití již zmíněného identifikátoru otázky, která je viditelná jen v rámci prezentace. Prezentující může pomocí tlačítka vyvolat další otázku v pořadí nebo si může jednoduše zvolit kteroukoliv otázku požaduje. Obrázek výše demonstruje notifikaci o nově připojených uživatelých.

Dále je zde funkce, kdy je možné vyresetovat aktivní otázku. Tímto krokem jsou veškeré odpovědi anulovány a uživatelé mají další šanci na otázku odpovědět. Toto chování je výhodné zejména u otázek, které uživatelům po zodpovězení neukazují správně odpovědi. V dolní liště je vidět počet aktivních uživatelů, kteří jsou připojeni na prezentaci a buď čekají ve frontě, odpovídají na otázku, nebo čekají na další.



Obrázek 3.12 – Zobrazení aktivní otázky na mobilním zařízení

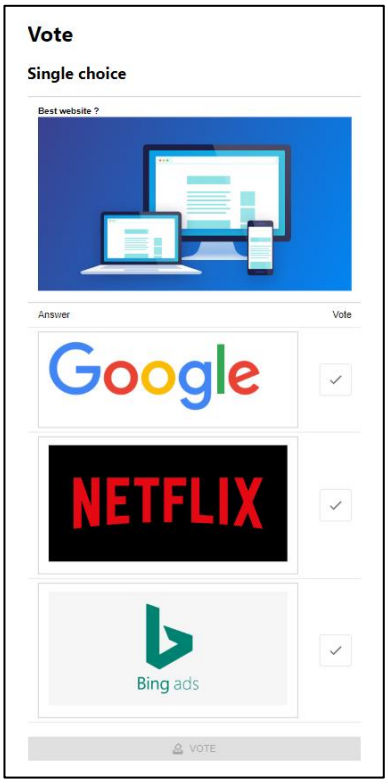
Uživatel uvidí aktivovanou otázku, viz. obrázek výše, kde má možnost vidět její zadání včetně odpovědí a také zda zadává jednu nebo více odpovědí. Na obrázku je patrné rozložení pro mobilní telefon, a proto byl kladen důraz na ovládání na dotykových obrazovkách a použití komponent, které nevyžadují použití myši či klávesnice. V dolní části má možnost odeslat své odpovědi.



Obrázek 3.13 – Vyhodnocení otázky

Pokud otázka byla nastavena tak, aby po odeslání odpovědí zobrazila výsledky, uvidí uživatel následné vyhodnocení. Zelený symbol značí správně zodpovězenou otázku, červený

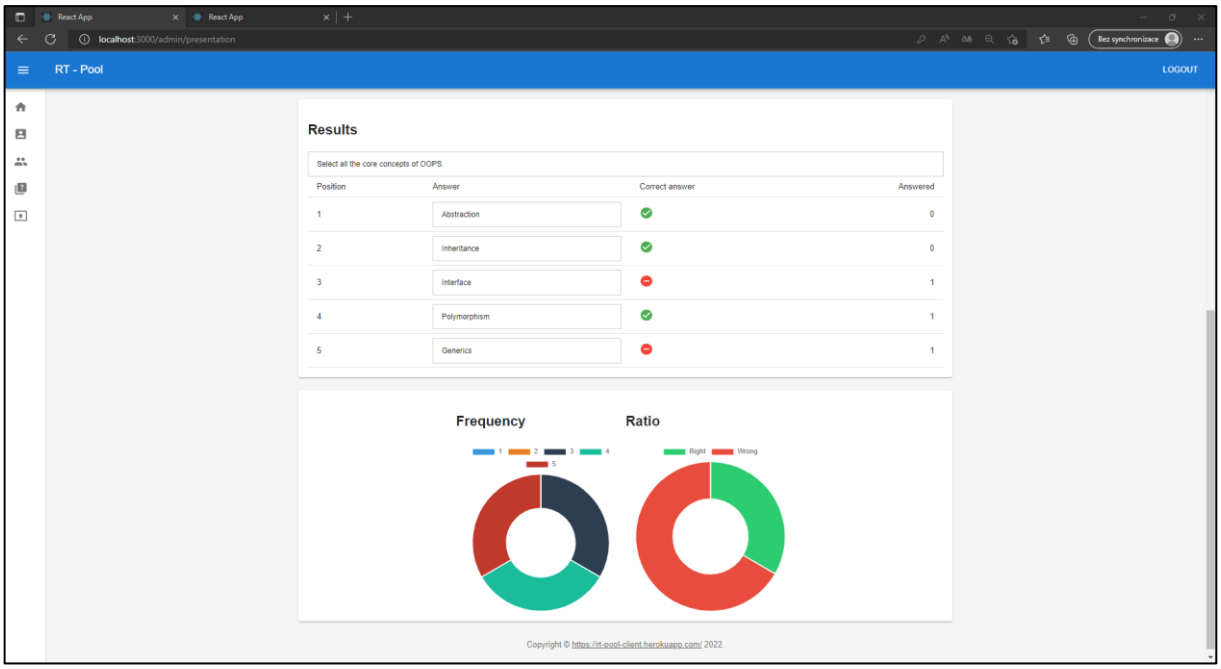
označuje chybě zvolenou odpověď a žlutý poukazuje na odpověď, která měla být zvolena. V horní části obrazovky je vidět pokyn pro čekání na další kroky prezentujícího, viz obrázek výše. Obrázek níže demonstruje zobrazení grafických elementů na mobilním zařízení,



Obrázek 3.14 – Ukázka zobrazení otázky s grafickým obsahem

Prezentujícímu se na stejné obrazovce, ve které ovládá prezentaci, ve spodní části objeví dva nové panely. První zobrazuje otázku, kterou vidí i uživatelé s dodatečnými informacemi, jako je správnost a četnost odpovědí.

V dolní liště jsou zobrazeny grafy s četností jednotlivých odpovědí a procentuální správnost přítomných hlasujících. V momentě, kdy otázka nezobrazuje po zodpovězení správné odpovědi, jsou zde tyto statistiky přesto vyobrazeny, a prezentující může s dodatečným komentářem otázku resetovat a umožnit studentům druhou šanci, nebo je informuje o skutečnosti sám a pokračuje buď další otázkou, nebo prezentaci ukončí. Demonstrační obrazovka je na obrázku níže.



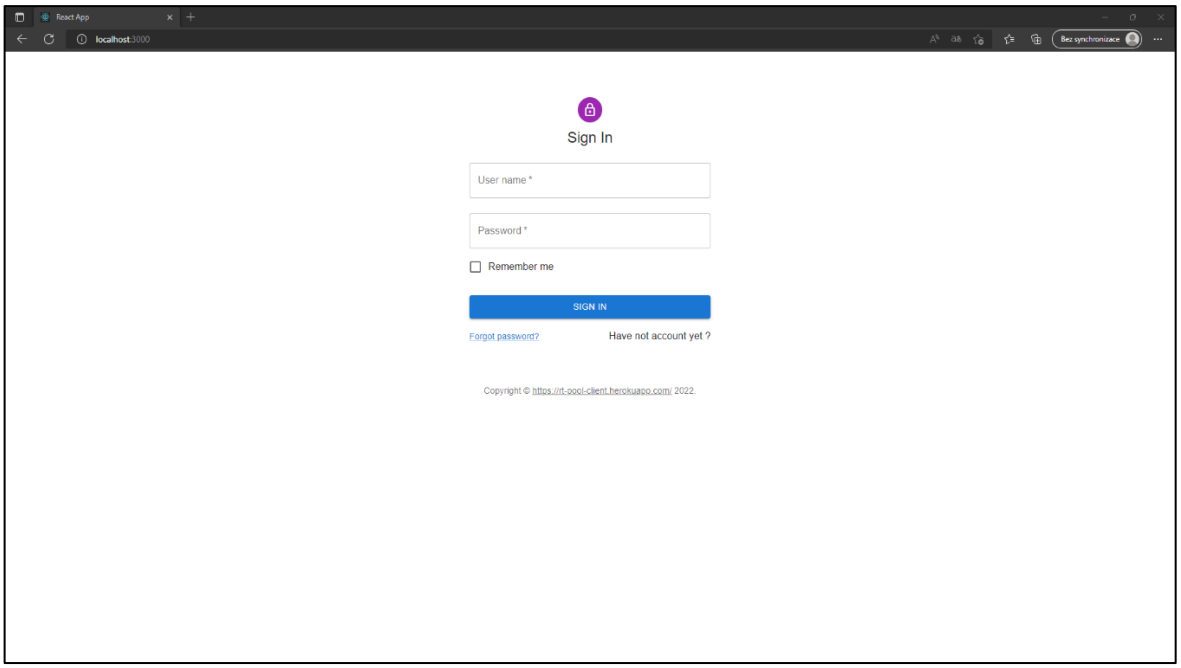
Obrázek 3.15 – Přehled aktuální otázky včetně hlasovacích statistik

3.3 PŘIHLÁŠENÍ A REGISTRACE UŽIVATELŮ

Pro zobrazení aplikace je potřeba použít libovolný webový prohlížeč. Odkaz na stránku pak bude modifikován dle konkrétního nastavení provozovatele. Hned v úvodu je zobrazena výzva k přihlášení uživatele. Aplikace sama o sobě nenabízí možnost průchodu aplikací bez přihlášení, protože návrh počítá s tím, že každé další akce jsou vázány na specifického uživatele, což bylo požadavkem při tvorbě.

Po prvotním spuštění existuje pouze jeden administrační účet, který je po celou dobu existence aplikace jenom jeden. Pokud je zapotřebí více uživatelů, kteří budou mít své nezávislé nastavení na ostatních, je možná registrace, kde ovšem aktivace účtu řídí hlavní administrátor, který buď požadavek o tvorbu účtu zamítne, nebo aktivuje.

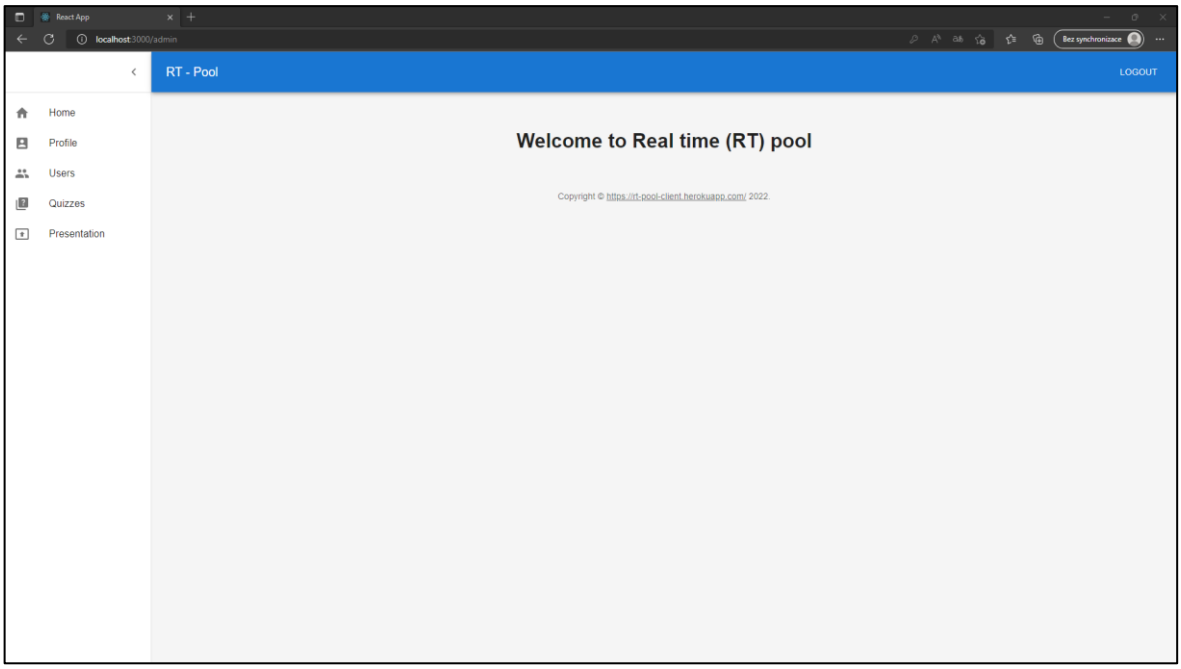
Pokud uživatel zapomene heslo, musí požádat administrátora o jeho změnu způsobem, že administrátor původní heslo odstraní a uživatele vyzve k vytvoření nového hesla. Uživatelská jména musí být vždy unikátní, stejně tak e-mailová adresa, protože možnost přihlášení je možná za použití obojího. Validace hesla probíhá standardně jako u jiných aplikací, kde existují určité limity, kterými musí uživatel projít, jinak jeho účet nebude vytvořen, viz. obrázek níže.



Obrázek 3.16 – Přihlašovací stránka

3.4 HLAVNÍ OBRAZOVKA

Po úspěšném přihlášení se stránka přesměruje na hlavní obrazovku sloužící primárně jako rozcestník na ostatní komponenty. Kromě uvítací obrazovky se na této stránce nenacházejí žádné důležité informace. Panel pro odhlášení, včetně nabídky umístěné vlevo, zůstávají vždy na svém místě a jsou tak statickými komponenty až do odhlášení uživatele. Tlačítkem „Home“ vybraným v levém panelu se vždy dostaneme na tuto hlavní obrazovku, viz. obrázek níže.

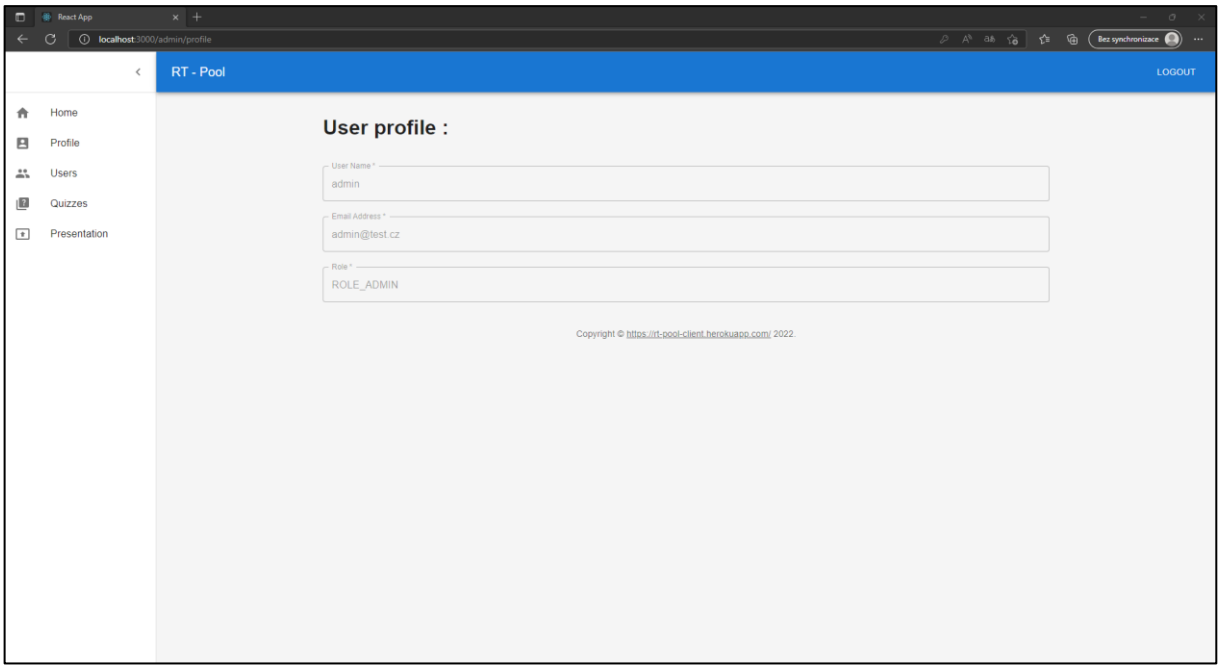


Obrázek 3.17 – Hlavní obrazovka aplikace po přihlášení

3.5 PROFIL UŽIVATELE

Další komponent poskytuje informace o přihlášeném uživateli, kde jsou zobrazeny údaje vyplněné během registrace a následně přidělená příslušná role. Každému nově vytvořenému uživateli je přidělena role moderátor, která nabízí téměř všechny funkce, vyjma možnosti schvalovat nově registrované uživatele. Tato role přísluší pouze a jenom administrátorovi. V programu, co se designu uživatelových rolí týče, je připravena ještě jedna role, která je ovšem použita v momentě, kdy je uživatel v roli hlasujícího. Tato role ovšem přístup do této části programu nemá, a tudíž by zde nebyla ani zobrazena.

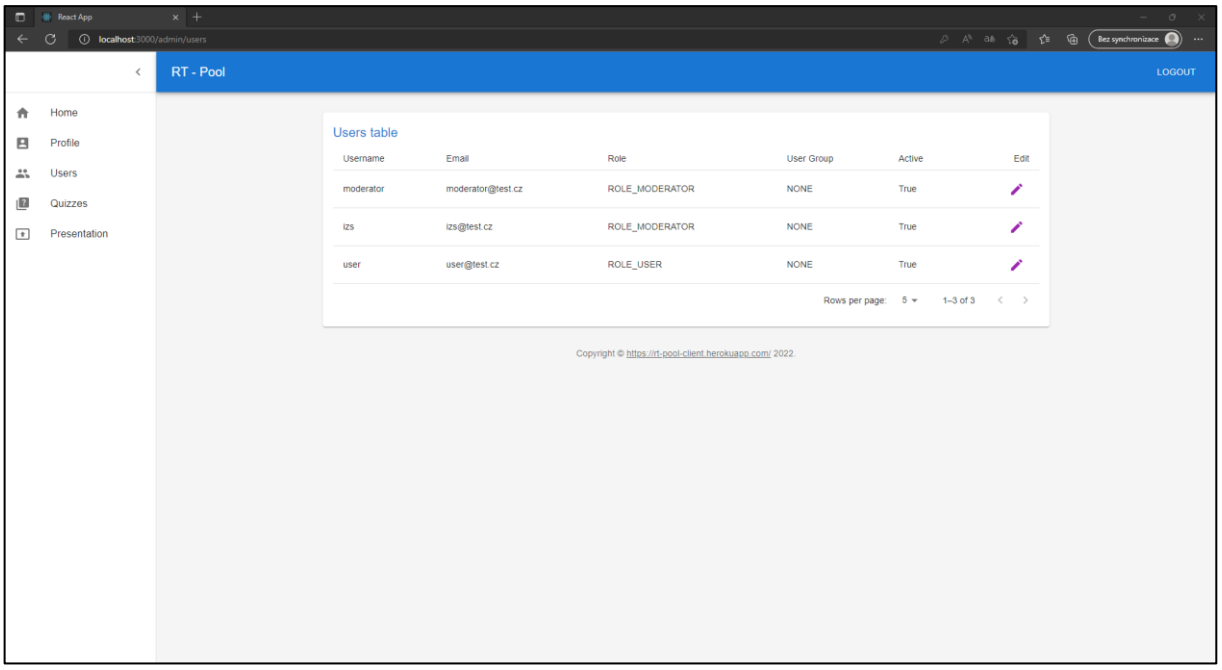
Důvod schvalování registrací je opodstatněn faktem, že obyčejný uživatel by neměl do této části aplikace samovolně vstupovat, protože by mohl celkem snadno zneužít jeho funkcionality k nekalým účelům, i přesto, že již zde existují určité restrikce. Profil uživatele zobrazen v aplikaci na obrázku níže.



Obrázek 3.18 – Profil uživatele

3.6 SEZNAM REGISTROVANÝCH UŽIVATELŮ

Tato sekce je přístupná pouze administrátorovi, viz. obrázek níže. V tabulce jsou vidět účty, které jsou registrované a ke kterým má administrátor plnou moc. Zde se také aktivují profily, které žádají o povolení vstupu do aplikace. Administrátor může změnit uživatelovu roli, skupinu, nebo dokonce odstavit účet úplně. Změna uživatelského jména ani e-mailu není povolena, protože je vázána na existující data, pokud nějaké uživatel již stačil vytvořit. Tato funkcionalita je nicméně poměrně náročnější na implementaci, a jelikož k tomu nebyl vytvořen speciální požadavek, je k dispozici tento základ umožňující základní manipulaci s uživateli obecně.



Obrázek 3.19 – Seznam aktuálních registrovaných uživatelů

3.7 SHRNUTÍ

Veškeré nutné části pro ovládání prezentace zde byly podrobně popsány, včetně průvodu vytvoření kvízu a jeho aktuálních možnostech, které může prezentující využít. Stěžejní částí této aplikace je vygenerování výsledných statistik do souboru, kde jsou jednotlivé četnosti odpovědí vypsány.

Další rozšíření této aplikace by se zaměřovala právě na statistiky jednotlivých uživatelů, kde by se mohly jednotlivé otázky bodovat a vyhodnocovat jednotlivé uživatele včetně zobrazení finálního žebříčku s těmi, kteří si vedli v kvízu nejlépe a vytvořit tak soutěžní prostředí, které by mohlo být v rámci prezentace zábavnější formou, ve kterém výherci mohou získat různé benefity, které pomohou úspěšně zdolat zkoumaný předmět.

Stále se však jedná o plnohodnotnou aplikaci, která splňuje všechna zadaná kritéria a tím se stává silným prostředkem pro použití v téměř jakékoliv sféře.

4 POUŽITÉ TECHNOLOGIE

K vývoji aplikací existuje nepřeberné množství prostředků umožňující její implementaci. Některé jsou orientované čistě pro webové aplikace, jiné plní účel výhradně na desktopových systémech. V dnešní době již velká část cílí na chytré telefony. Pokud jde výhradně o kategorizaci webových prostředků, téměř v každém z nich je možno docílit kýženého výsledku, avšak s odlišnou formou implementace a přístupu. Jejich volba je ovšem otázkou daleko více parametrů, které je třeba zohlednit. Většina firem, které působí v roli softwarových vývojářů, klade především důraz na vývoj v technologiích, ve kterých jsou jejich současní programátoři zdatní, přesto volba jiné technologie by byla pro konkrétní řešení vhodnější. Tento trend je obecně využíván zejména u projektů, u kterých je kladen důraz na včasné doručení zákazníkovi. Obecně jakákoliv další znalost v oblasti vývoje v různých technologiích vyžaduje poměrně dost času, který by firma musela svým zaměstnancům proplatit. Následně ani poté nemá jistotu, že by zvolené nové technologie fungovaly o poznání lépe než ty, které používá. Ovšem je dobré vědět, že tyto technologie existují, a je dobré znát alespoň základní rozdíly mezi nimi, které by v budoucnu mohly být cenou informací, která by mohla potencionální technologii uvést na list priorit o poznání výše. Stejně jako u většiny projektů, tak i zde byl při zvolení vhodných technologií kladen důraz na současné schopnosti vývojáře, tak i vhodnost jednotlivých prostředků. Tato kapitola se bude zaměřovat především na porovnání těch největších současně používaných prostředků pro vývoj webových aplikací společně se serverem a bude zde podrobně zvolené technologie popisovat tak, aby byla pochopena alespoň základní idea těchto použitých prvků.

4.1 BACKEND WEBOVÉ APLIKACE

4.1.1 Spring-boot

Vychází z frameworku Spring, což je velmi populární volně dostupný aplikační rámec pro vývoj J2EE aplikací. Jeho historie sahá až do roku 2002, kdy byl dokončen jeho vývoj, a o rok později uveden na trh. Argumentů pro vznik tohoto frameworku bylo hned několik. Bylo potřeba vyřešit problémy a zjednodušit vývojářům aplikace jejich vývoj. Použití návrhových vzorů pomáhá udržovat strukturu programu velmi rychle nabývající na objemu.

Obrázek 4.1 – Logo frameworku Spring-boot (Logo-Logos.com, 2022)

Na obrázku výše je logo Spring-boot frameworku. První část problematiky se soustředila na vyřešení problému, který řeší odstranění těsných programových vazeb pomocí návrhového vzoru Inversion of Control, který zjednodušeně přesouvá odpovědnost vytváření objektů přímo na framework. Dále je nutná možnost volby implementace businnes vrstvy pro aplikační architekturu, protože je velmi nevhodné, pokud by tomu bylo naopak. Stěžejní částí je také přístup k datům, který je možný díky JDBC nebo přístup k ORM technologiím a prostředkům, mezi které řadíme například Hibernate, MyBatis nebo JDO. Další oceňovanou částí je možnost konfigurace pouze z určitých míst, které jasně a pevně definují svůj význam. Nelze také opomenout správu a konfigurační management business komponent. A v neposlední řadě usnadnění a psaní unit testů. Těchto výhod je samozřejmě mnohonásobně více a jejich popis by patrně pokryl celou práci samostatně, proto jsou zde vyjmenovány pouze ty nejdůležitější, které jsou již tak dost silným argumentem pro potencionální použití v novém projektu orientovaném ve výše uvedeném směru podobně jako je implementována aplikace popisovaná touto prací (IBM Cloud Education, 2020).

4.1.2 Kotlin

Popsaný framework je možné programovat v jakémkoliv podporovaném jazyce postaveném na JVM. V tomto projektu byl zvolen jazyk Kotlin. Kotlin je o poznání novější jazyk. Jeho vznik je datován rokem 2011, ale stabilní verze se dočkává až o rok později a je vyvíjen společnostní JetBrains. Hlavní použití tohoto jazyka bylo v prvotních fázích své existence pro vývoj Android aplikací, protože tehdy primární používaný programovací jazyk Java byl v některých ohledech velmi nevhodný a nepřehledný, byť nabízel stejné funkcionality. Popularita jazyku Kotlinu raketově rostla až do té míry, kdy jej společnost Google v roce 2019 zařadila mezi preferované jazyky pro vývoj aplikací pro Android systémy a Java se tak stala sekundářem. Aktuální logo jazyka Kotlin je na obrázku níže.

Obrázek 4.2 – Logo programovacího jazyka Kotlin (Logo-Logos.com, 2022)

Dalším důvodem popularity jazyka je fakt, že je postavený nad JVM, stejně tak jako Java, takže je tím zaručena kompatibilita. Zajímavostí také může být, že zdrojový kód Kotlinu je převeditelný do Javy a opačně. Další nespornou výhodou je možnost využití stávajících knihoven určených pro Javu, což je jeden velmi důležitých vlastností, které popularitě Kotlinu zajisté pomohli.

V dnešních dobách není Kotlin již doménou pouze Android aplikací, ale pomalu a jistě se stává velmi dominantním hráčem na poli v oblasti softwarového vývoje. Čím dál více firem, které začínají nové projekty hledají softwarové vývojáře, kteří tento jazyk ovládají i valná většina vývojářů Javy na tento jazyk přechází, protože jeho nesporné výhody jsou primárními důvody, proč je Java přesunuta na druhou kolej.

Základní idea, která určila vznik Kotlinu, je totiž jednoduchá, a to vylepšení koncepčních a syntaktických vlastností, které Java používá. V následující části bude představeno pět hlavních důvodů proč použít Kotlin namísto Javy.

4.1.3 Kotlin vs Java

Kotlin je staticky typovaný jazyk. Oproti Javě má kratší zápisy kódu řešící tentýž problém. Jeho stručnost může naopak snižovat čitelnost. Jeho zápis nevyžaduje používání středníků, čímž se stává kód čistší. Dále je zde daleko lepší podpora pro práci s textovými řetězci, kde jsou již zabudované elegantní šablony. Kotlin poskytuje jednoduchý způsob, jak používat proměnlivé a neměnné deklarace pro různé datové struktury.

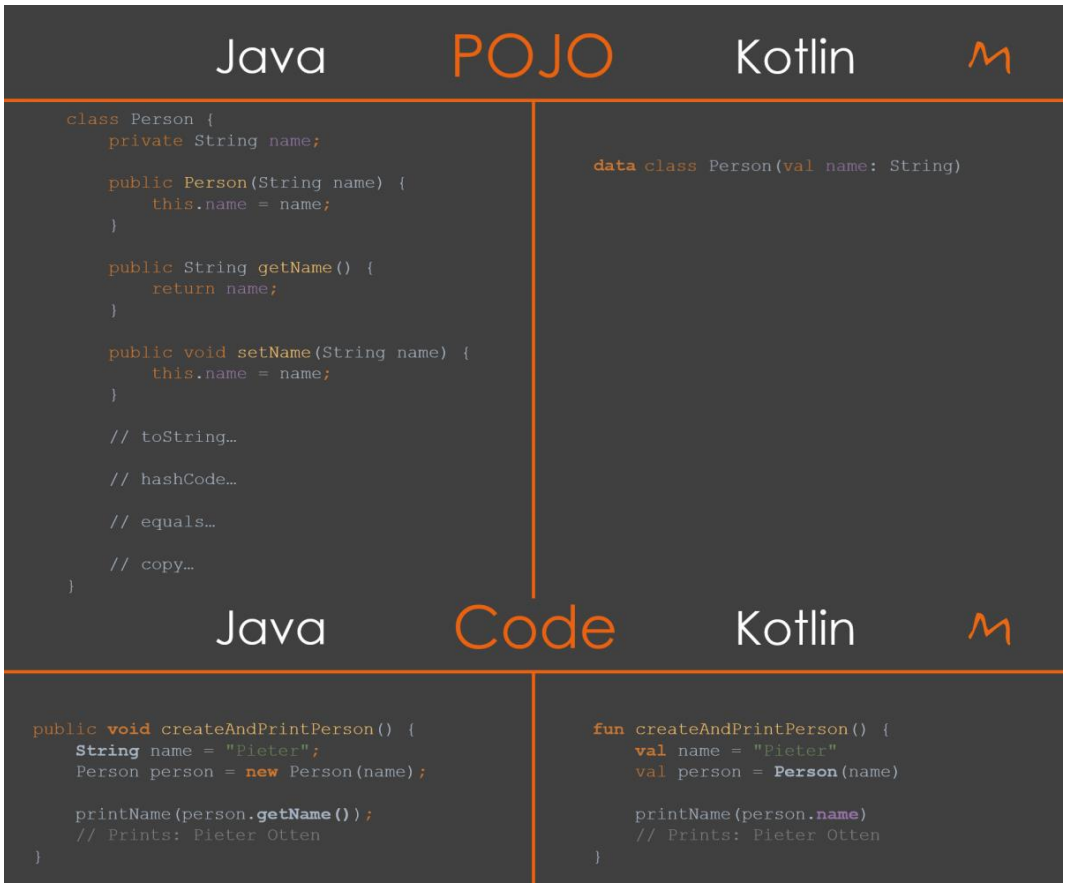
Kotlin si může snadno vyměňovat a využívat informace z Javy mnoha způsoby. To je jedna z nejsilnějších výhod Kotlinu. Java a kód Kotlin mohou koexistovat ve stejném projektu. Kotlin dobře spolupracuje s programovacím jazykem Java. Kromě toho lze v projektech Kotlinu použít řadu knihoven Javy, díky čemuž je ještě více kompatibilní.

Nejen knihovny, ale spousta frameworků z Javy je kompatibilní s Kotlinem, včetně některých pokročilých frameworků. Projekt v Javě můžete snadno převést bez velkých

zásadních úprav, jako je převod celého projektu na Kotlin. To je obrovská výhoda pro vývojáře, protože se ve skutečnosti nemusí učit nový jazyk. Každý, kdo zná Javu, bude znát a umět programovat v Kotlinu.

Jednou z největších výhod Kotlinu oproti Javě jsou nulové reference. Přístup k členu nulové reference má za následek vyvolání výjimky. Tím je celkový kód kotlinu bezpečný, co se týče reakcí na proměnné, které mohou být nulové, protože je v rukou programátora, aby ošetřil dané stavy vhodnou reakcí na ně, jelikož editor bude na tyto problémy v kódu aktivně vyzývat programátora k nápravě.

Dále pak má Kotlin integrované knihovny pracující s vlákny, které se nazývají coroutines. Jedná se o velmi mocný prostředek, který dokáže na jednom vlákně vytvořit asynchronní zpracování požadavků a efektivně tak zpracovat paralelně značné množství dat. Určitě je třeba zmínit existence datových tříd, kterými svým jednoduchým zápisem dramaticky zjednodušuje čitelnost kódu. Datová třída je speciální třída, která má deklarované proměnné, na které se dostaneme přímo, aniž bychom potřebovali metody typu getter a setter, nicméně stejné výhody lze aplikovat i zde. Jsou určité problematiky, které jsou v Kotlinu lépe řešené, ale faktem zůstává, že většina z nich je spíše subjektivním hodnocením daných jazyků. Porovnání zápisu zdrojových kódu je na obrázku níže. (Mediaan, 2018).



Obrázek 4.3 – Porovnání programovacího jazyka Kotlin a Java (Mediaan, 2018)

4.2 FRONTEND WEBOVÉ APLIKACE

Jedná se o aplikaci obsahující vnitřní logiku, kterou doprovází grafické rozhraní, se kterým pak uživatel pracuje. Představuje kolekci různých prvků, které jsou využity pro komunikaci se serverem a s uživatelem samotným a výstupem toho je webová stránka dostupná skrze webový prohlížeč. Implementace takové stránky nutně nevyžaduje serverovou část, která pracuje s daty, protože existují případy, kdy stránka plní účel pouze informačního charakteru, a tudíž ke svému fungování nepotřebuje žádnou hlubší logiku.

Použití v této práci se právě využívá koordinace klienta a serveru, protože realtimová část aplikace je provozována právě na serveru a klientská část již zde figuruje pouze jako reakce na výstup informací ze serveru. Je také považována za vstupní bránu uživatelských dat, které jsou následně generovány do příslušných předem připravených struktur, které jsou dané servisní funkce přípravné přijmout a zpracovat.

Existence frameworků a knihoven dostupných pro implementaci této části existuje velmi mnoho. Pokud je framework navržen tak, aby pracoval s klasickým REST a dalšími službami včetně například websocketů, můžeme jej libovolně použít a brát jej jako univerzálně použitelný prostředek.

Primárním komunikačním prostředkem mezi klientem a serverem jsou objekty, které se nazývají JSON. Jedná se o strukturu dat podobné třídě, která je interpretována v textové podobě a příslušné převodníky, jak na straně serveru, tak na straně klienta, dokážou s těmito strukturami pracovat a následně mají možnost je transformovat do tříd se stejnou signaturou, a tím tak vytvořit nové objekty, se kterými je nadále pracováno dle definovaných funkcí.

Pokud jsou tyto majoritní požadavky splněny, již nic nebrání tomu vybrat si framework, který již toto splňuje. Na trhu opět existuje mnoho konkurencí, které jsou všechny orientovány podobným směrem, a to mít možnost vyvíjet webové stránky, pokud možno rychle, efektivně a nejlépe bez ztráty kvality ve smyslu existující struktury. Vždy je výhodnější využít takové frameworky, které jsou již delší dobu v působení, a to primárně proto, že již pravděpodobně existují knihovny řešící nejrůznější problémy a jsou již k dispozici statistiky, co se stability a bezpečnosti týče.

Tyto informace jsou vždy celkem snadno dohledatelné, a tudíž není těžké vyhledat právě ty nejpopulárnější a následně porovnat mezi nimi zásadní rozdíly, které pak pomohou zvolit právě ten, který bude nejlépe řešit existující problém. V této práci byl zvolen framework React, nicméně stejnou práci by odvedl například stejně populární Angular, takže se nejedná o ultimátní řešení, ale jde pouze o subjektivní preferenci této volby a roli zde hrála určitá míra experimentace.

4.2.1 React

Jedná se o JavaScriptovou knihovnu pro vytváření uživatelského rozhraní. React je svobodný software pod licencí MIT a je vyvíjen společností Meta Platforms. Využívá se například pro tvorbu webových stránek nebo také mobilních aplikací. Aktuální logo frameworku je na obrázku níže.

Obrázek 4.4 – Logo frameworku (Logos Download, 2022)

Každá vzniklá knihovna má svou historii ohledně důvodu svého vzniku. Ať už něco zásadního chybělo, nebo často vznikaly problémy, které neměly jednoduchá řešení, protože neexistovaly mechanismy, které by jim předcházely, což se dost často projevuje u projektů, které nekontrolovatelně rostou a jsou vyvíjeny různými vývojáři uskupených nejčastěji do týmů.

Vývojáři podílející se na vývoji webové stránky Facebook se dostali do situace, kdy začala být jejich kódová báze s postupem času hůře udržovatelná. Jelikož se do aplikace stále přináší nové funkce, bylo potřeba zakročit dříve, než se stane hotový kód nerozšiřitelný. Tato nepříjemná situace gradovala kolem roku 2011, kdy se Facebooku velmi dařilo a společnost začala přijímat stále více vývojových týmů, které se podílely na vývoji nových funkcí.

Společnost brzy pochopila, že stávající řešení není schopné pokrýt jejich potřeby, a tak začala vyvíjet svůj vlastní framework, který by jim vývoj dramaticky ulehčil. Ještě tentýž rok svou prvotní verzi využil v takzvaných Facebook News Feed a o rok později na platformě Instagram.

V roce 2013 se společnost rozhodla, že jejich stávající řešení poskytne zadarmo, a tak byly zveřejněny její zdrojové kódy volně ke stažení. Samozřejmě tento akt nebyl nic jiného než prostý marketing, ze kterého nakonec profitují všichni. Při vývoji právě takto komplexní aplikace je zapotřebí důkladného testování, což stojí dost času, a ne všechny chyby lze na první pohled odhalit. Zveřejnění do světa přimělo vývojáře tento framework využívat a poukazovat právě na nově objevené problémy, které by se v případě proprietárního řešení hledaly mnohem hůře, a tak se začala budovat komunita, která tento framework využívá prakticky dodnes.

Dalším důležitým milníkem v oblasti vývoje se považuje rok 2015, kdy se na trh uvedl React Native, což je framework, který dokáže fungovat bez nutnosti použití webového prohlížeče a je možné jej instalovat jako samostatnou aplikaci jak na systémy Windows, Linux, či macOS,

tak i na iOS a Android. Tento framework je dnes viditelný například v programech Discord či klientu online hry League of Legends, která opustila své stávající řešení Adobe Air.

Základní struktura spočívá v prosté myšlence komponent, kde každá zobrazovaná komponenta je brána jako funkce. Tyto komponenty mají možnost vstupu argumentů, ze kterých se následně vnitřní logikou vrací objekt, který je poté využit jako grafická komponenta, která je vnořená v jiné. Tyto komponenty neboli funkce jsou sestavovány v jazyce podobném HTML, kde jednotlivé komponenty představují celkový obsah a jsou součástí kořene, který je iterován až směrem k nejnižší části, která je postupně zobrazována. Příklad funkcionální komponenty na je na obrázku níže,

Obrázek 4.5 – Ukázka deklarace funkcionální komponenty

Dalším argumentem, proč se stal React tak populárním, je především jeho výkon. JavaScript je velmi rychlý jazyk, nicméně aktualizace DOM, což je objektově orientovaná reprezentace XML nebo HTML dokumentu, je zde již výrazně pomalejší. React řeší uvedený problém tím, že vytváří svůj takzvaný „Virtual DOM“ a tím má plnou kontrolu nad tím, co se potřebuje zrovna vykreslit a neplýtvá časem překreslováním statického obsahu, u kterého neproběhla žádná změna (Ilyukha, 2022).

4.2.2 Typescript

Podobně jako tomu je mezi programovacími jazyky Kotlin a Java, existuje velmi podobný vztah mezi jazyky JavaScript a TypeScript. Ten byl vytvořen společností Microsoft v roce 2012

a byl vydán jako open source. Z hlediska stavby se jedná o nadstavbu jazyka JavaScript, kde je mimo jiné rozšířen o statické typování, třídy a rozhraní, včetně podpory paradigmat objektově orientovaného programování.

TypeScript je opět reakcí na zastaralé koncepty JavaScriptu. Rozhodně se nejedná o špatný jazyk, nicméně tím, že dnes jsou vyvíjeny daleko rozsáhlejší systémy, než tomu bylo dříve, tyto jazyky se začaly stávat poněkud nevhodnými pro udržení kvality kódu, a tak TypeScript přišel s řešením, které odstraňuje některé neduhy a zlepšuje komfort samotného vývojáře. Největší problém JavaScriptu je, že neumožňuje uvádět datové typy, což poté znamená, že nelze efektivně kontrolovat, jaký typ objektu může přijít a často to vede k dlouhým debuggováním a hledáním triviálního problému, kterému se dalo elegantně předejít již při psaní, protože editor by včas na onen problém upozornil.

Hlavním argumentem, proč použít JavaScript i v dnešní době je rychlost, ale to se týká spíše softwarových gigantů poskytující multimodální obsah, kde je důraz kladen na extrémní rychlosti, nicméně v běžných webových stránkách jsou tyto rozdíly spíše zanedbatelné. Jeho oblíbenost a užitečnost byla natolik velká, že dokonce společnost Google integrovala TypeScript do svého frameworku Angular. V dnešní době již existuje i podpora pro React, což bylo využito při vývoji klientské části aplikace v tomto projektu. Aby TypeScript mohl fungovat, potřebuje pro svůj běh speciální druh kompilátoru Transpiler. Současné logo jazyka je na obrázku níže. (Intro to Typescript, 2021).

Obrázek 4.6 – Logo programovacího jazyka TypeScript (Towards Dev, 2021)

4.3 VYUŽITÉ REALTIMOVÉ TECHNOLOGIE

Pro zajištění realtimové komunikace mezi klientem a serverem byly využity knihovny, které implementují STOMP protokol (STOMP Protocol Specification, 2012).

4.3.1 Definice protokolu STOMP

Jedná se jednoduchý, textově orientovaný protokol. Původně byl navržen pro použití ve skriptovacích jazycích, jako je například Ruby. Jeho design rámců vychází z protokolu http a je široce podporován různými frameworky, a dobře se využije k použití skrze websockety.

Protokol je podobný http a funguje skrze TCP využívající následující příkazy:

- CONNECT,
- SEND,
- SUBSCRIBE,
- UNSUBSCRIBE,
- BEGIN,
- COMMIT,
- ABORT,
- ACK,
- NACK,
- DISCONNECT.

Jeho konkrétní implementace je využita v knihovnách Apache ActiveMQ, HornetQ, OpenMQ, RabbitMQ, STOMP.js a další.

4.3.2 Implementace

V klientské části projektu byla použita knihovna STOMP.js. Jejím původním autorem je Jeff Mesnil, nicméně později se k vývoji připojilo mnoho dalších vývojářů. Tato knihovna umožňuje připojit se k brokerovi STOMP přes websockety. Podporuje kompletní specifikace STOMP včetně všech aktuálních variant protokolů. Nejvyužívanější brokeři využívají STOMP přes websockety buď nativně, nebo pomocí pluginů. Funguje prakticky v každém JavaScriptem založeném prostředí. Spring boot má ve svém jádře již obsaženou implementaci protokolu STOMP, a ta také byla v projektu použita. Jeho fungování je založeno na architektonickém vzoru publish-subscriber.

4.3.3 Použití

Uživatel při inicializaci funkce definuje adresu serveru, parametr cíle a návratovou funkci. Parametr cíle určuje, co bude výsledná funkce odebírat a návratová funkce bude volána vždy v momentě, kdy server odešle odpověď směrem ke klientům, avšak pouze těm, jichž se to týká.

5 SPECIFIKACE PROJEKTU

Cílem této práce bylo vytvořit webovou aplikaci pro realizaci interaktivních realtimových hlasovacích anket určených pro dynamické prezentace, přednášky či výuku. Každá anketa se skládá z předem definovaných anketních otázek (s možností výběru jedné nebo více odpovědí), které si nadefinuje prezentující. Prezentující má dále možnosti spuštění ankety, přechody mezi otázkami a zobrazení výsledků. Hlasující mají k dispozici URL odkaz do aplikace, kde vždy vidí pouze aktuální anketní otázku, na kterou mohou odpovědět. Prezentující má k dispozici veřejnou výsledkovou obrazovku ankety, kde bude zobrazena aktuální anketní otázka a zvolené odpovědi. Po odeslání odpovědí hlasujícím dojde k realtimové aktualizaci zobrazení výsledků. Rozhraní hlasujících je rovněž realtimově aktualizováno, pokud dojde ke změně otázky.

5.1 ROZBOR ZADÁNÍ

Výše uvedená specifika konkrétně definuje několik základních stavebních kamenů, které již předem zužují možnosti ve výběru konkrétních technologií. Předem je patrné, že aplikace bude typu klient-server a bude využívat realtimové prvky. Bude třeba určitě počítat s možností, že uživatel nebude využívat pouze desktop, nýbrž své mobilní zařízení, a tak je třeba v průběhu tvorby uživatelského prostředí s tímto faktem počítat. Z hlediska bezpečnosti bude třeba využít určité bezpečnostní prvky, které budou uživatele bezchybně autentizovat i autorizovat.

S využitím již existujících znalostí s vývojem aplikací na vysoké škole se jedná o zadání, které bude ve své podstatě kombinovat již naučené a předem vyzkoušené techniky, které bude nutné karetně svázat do jednotlivých celků a při návrhu celého projektu bude nutný apel na správný strategický směr. Zvolené technologie jsou již specifikované a byly zde i popsány, tudíž se této části nebudou následující kapitoly věnovat a bude zde logické členění dle zvoleného postupu při implementaci tohoto projektu až do jeho finální fáze.

5.2 FUKČNÍ POŽADAVKY

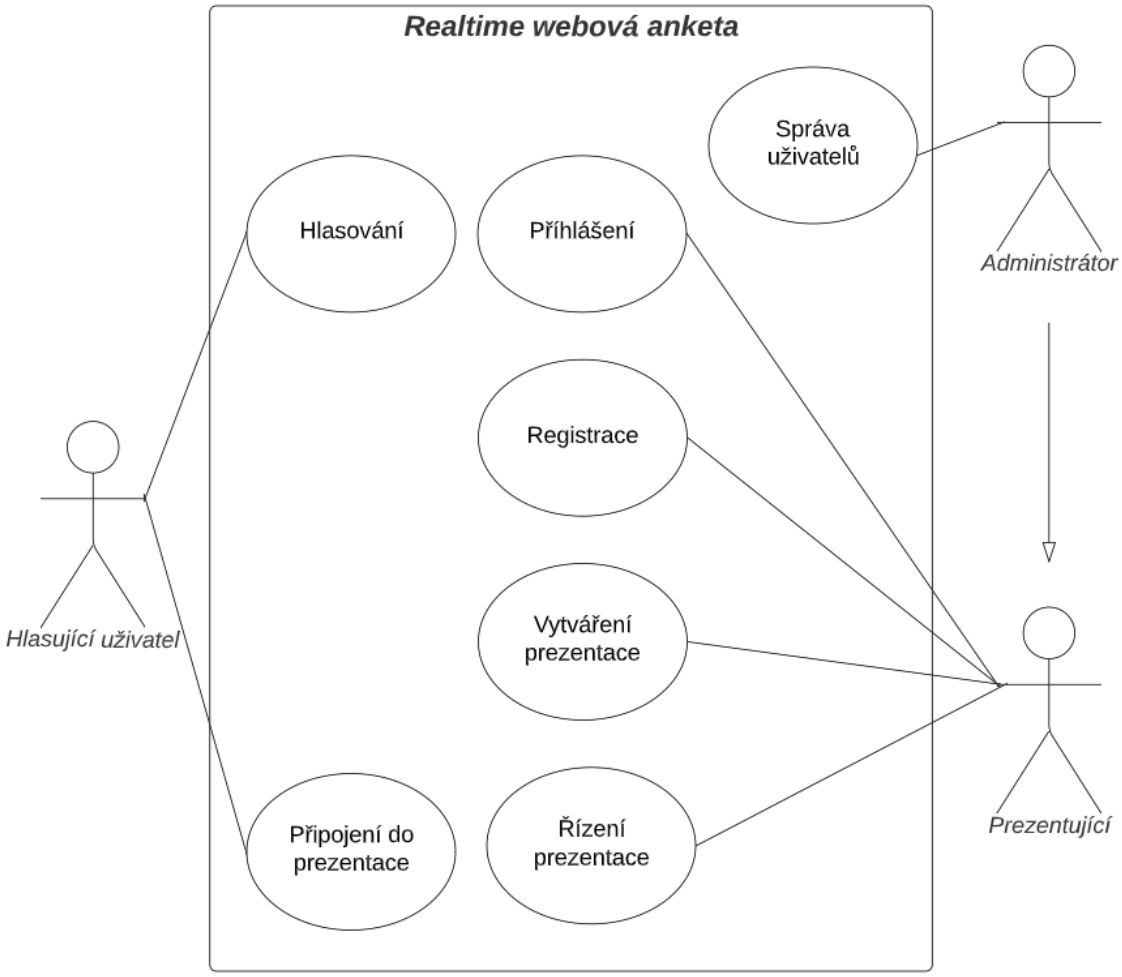
- *FP 1* – Interaktivní anketa bude umožnovat registraci uživatelů.
  - *FP 1.1* – Anonymní uživatelé pouze v roli hlasujících.
  - *FP 1.2* – Evidence uživatelů.
- *FP 2* – Interaktivní anketa bude umožnovat vytvoření kvízu.
  - *FP 2.1* – Vytvoření otázky.
  - *FP 2.2* – Vytvoření odpovědí.
  - *FP 2.3* – Možnost uspořádat pořadí (otázek / odpovědí).
  - *FP 2.4* – Persistence vytvořeného kvízu, otázek a odpovědí.
- *FP 3* – Interaktivní anketa bude umožnovat spuštění a správu spuštěného kvízu.
  - *FP 3.1* – Odhalení výsledků otázky.
  - *FP 3.2* – Posun na další otázku.
  - *FP 3.3* – Posun na libovolnou otázku.
  - *FP 3.4* – Restart otázky a vynulování výsledků u dané otázky.
  - *FP 3.5* – Export výsledků z kvízu (volitelné).
  - *FP 3.6* – Import / export kvízu do čitelného souboru (volitelné).

5.3 NEFUKČNÍ POŽADAVKY

- *NP 1* – Aplikace bude typu klient-server.
- *NP 2* – Aplikace bude využívat SQL databázi PostgreSQL (Hibernate).
- *NP 3* – Server aplikace bude vytvořen pomocí Spring frameworku (Kotlin).
- *NP 4* – Klient aplikace bude realizován pomocí React knihovny (Typescript).
- *NP 5* – Grafické rozložení aplikace bude podporované chytrými telefony, tablety.
- *NP 6* – Aplikace bude podporovat bezpečnostní prvky (CORS, CSRF, XSS).
- *NP 7* – Aplikace bude využívat standardizované realtimové technologie.
- *NP 8* – Server aplikace bude poskytovat REST API (HTTP).

5.4 USE-CASE DIAGRAM APLIKACE

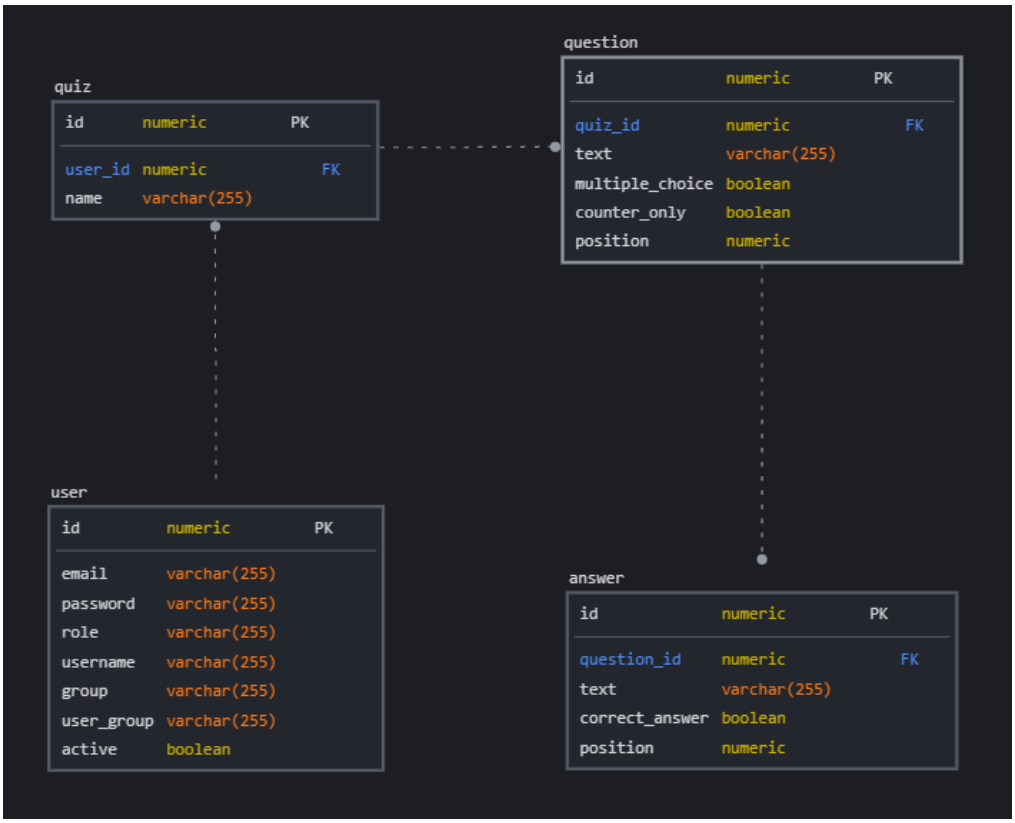
Dle specifikace je zde vytvořen use-case diagram, který je viditelný na obrázku níže, ve kterém figurují celkem tři aktéři. Hlasující uživatel má možnost připojení se do aktivní prezentace a může hlasovat. Prezentující se může do aplikace zaregistrovat, a po schválení administrátorem, i přihlašovat. Po přihlášení může vytvářet a řídit prezentace. Administrátor může provádět stejné činnosti jako prezentující s výjimkou správy uživatelů, které může spravovat pouze on.



Obrázek 5.1 – Use-case diagram prezentované aplikace

5.5 ANALÝZA A MODELOVÁNÍ DATOVÉHO MODELU

Prvním problémem, který zde vniká, je přesná specifikace dat, která budou využívána. Ze zadání jsou patrné čtyři hlavní datové entity. Na obrázku níže je zobrazen aktuální databázový model použitý v aplikaci. Rozbor jednotlivých entit bude popsán v podkapitolách.



Obrázek 5.2 – Databázový model využitý v projektu

5.5.1 Entita uživatele

Uživatele bude nutné evidovat zejména proto, aby bylo možné zajistit pouze výlučný přístup k aplikaci. Přestože zadání přímo nespecifikuje atributy, které budou muset být implementovány, opět jde většinu těchto dat získat přímo ze zadání. Je třeba umožnit uživatelovu autentizaci, takže bude třeba permanentně udržovat kombinaci uživatelského jména vůči heslu. V tomto projektu zde byl využit i e-mail, se kterým je možné provádět autentizaci též. Tyto atributy, kromě hesla, musí být unikátní a vstupem bude textový řetězec, který obdrží také omezené díly na standardních 255 znaků. Dalším důležitým atributem zde bude role, která určuje uživatelova práva vykonávat akce. V budoucích iteracích vývoje se zde počítá s využitím skupin, tudíž je tento atribut zde rezervován, i přesto, že současná implementace ho vůbec nevyužívá. Posledním atributem je booleovská hodnota určující

platnost účtu. Datový model uživatele je nyní definován a je možné pokračovat v návrhu datového modelu.

5.5.2 Entita kvíz

Tato entita bude hlavní branou vytváření kvízů. V tomto bodě je již nutné brát v potaz jednotlivé návaznosti existujících entit. V základu je předpokládáno, že jednotlivé kvízy budou odděleny jednotlivým uživatelům, tudíž bude nutné této entitě přiřadit vždy uživatelův unikátní identifikátor, deklarován jako primární klíč, který musí být vždy vyplněn, protože kvíz nesmí existovat, aniž by někomu patřil. Tento vztah bude identifikován jako typ 1:N, což pro tento případ znamená, že jeden uživatel může mít libovolný počet kvízů. Kvíz nebude kromě jeho názvu a klíče uživatele obsahovat další atributy, čímž je tato entita v rámci zadání dostatečně definována.

5.5.3 Entita otázka

Zadání určuje další důležité pravidlo, kde jednotlivé kvízy budou obsahovat sadu otázek. Počet zde není omezen a tím pádem bude využit stejný vztah 1: N jako v případě uživatele a kvízu. Atributy otázky jsou opět vyčteny ze zadání. Základní nutné atributy zde zastupují primární klíč v roli hlavního identifikátoru a cizí klíč značícího rodičovskou entitu určitého kvízu. Aby bylo možné otázku používat, musí obsahovat atribut, který bude obsahovat znění otázky. Jak již bylo v předešlé kapitole řečeno, otázka bude obsahovat dva různé atributy pro definici otázky. První bude základní textové popsání neboli identifikátor, který bude využíván primárně prezentujícím, a nebude viditelný pro hlasující uživatele.

Dle zadání bylo nutné splnit požadavek na definici otázky tak, aby kromě textového popisu existovala možnost formátovaného textu a například ještě zkombinovanou s obrázky. Tento způsob musí být řešen universálně, a proto bude druhý atribut sloužit jako úložiště HTML kódu, který zajistí splnění podmínky možnosti vytváření komplexní struktury. Dalším požadavkem je možnost volby, zda otázka bude mít pouze jednu či více možností, a proto zde bude rezervován booleovský atribut. Posledním požadavkem bude zajistit, zda se výsledné odpovědi budou po zodpovězení uživatelem zobrazovat či nikoliv. První varianta uvažovala o nastavení této možnosti nad celým kvízem, nicméně kvůli vyšší variabilitě zde tato možnost bude fixována na jednotlivé otázky, a pro tyto účely bude opět existovat již zmíněný booleovský atribut.

5.5.4 Entita odpověď

Primární klíč a cizí klíč již považujeme za standard a ani zde tyto atributy nebudou výjimkou. Zadání nedefinuje pravidlo, že budou existovat otázky, které by neměly vytvořené odpovědi, a proto musí existovat entita odpověď, kde bude rodičem právě vytvořená otázka, stejně jako v předešlém příkladě v roli 1: N. Početní limit opět stanoven nebyl, takže přednášející může zvolit libovolný počet odpovědí na otázku. Znění otázky již není nutné označovat speciálním identifikátorem a stačí pouze definovat atribut umožňující uložit HTML kód jako v předchozím případě. Poslední atribut bude určovat, zda se jedná o správnou otázku. Tímto krokem byl celý datový model navržen a připraven k implementaci.

5.6 VYUŽITÁ DATABÁZE

Realizace bude vyžadovat existenci databázového serveru. Charakter dat nejlépe odpovídá typu relační databáze. Tím, že Spring framework implementuje ORM, nebude nutné využívat jazyk SQL a pouze stačí využít programové konstrukty tohoto frameworku. Tím je také zaručena kompatibilita napříč různými relačními databázemi, pro které jsou již implementovány operace pro jejich používání. Jednoduše řečeno, existuje jedno univerzální API, které zařídí již potřebné kroky tak, že vytvoří SQL kód přímo na míru pro konkrétní databázi, a proto je její změna v projektu daleko jednodušší a nevyžaduje příliš mnoho zásahů do již vytvořeného kódu.

Existují realtimové databáze, které jsou orientovány na velmi často měnící se data. Vzhledem k tomu, že v tomto projektu těchto dat není mnoho a nepředpokládá se významné vytěžování síťového provozu ani výpočetního výkonu, nebyl tento typ databáze v projektu využit a namísto toho stávající logika drží data přímo v RAM paměti, čímž je výsledný efekt rychlostně srovnatelný, avšak mnohem hůře škálovatelný.

Pro tento projekt byla zvolena databáze PostgreSQL. Jedná se o databázi, která je volně dostupná pro komerční využití a plně zdarma. Pro testování aplikace se využívá nativní databáze ve Spring frameworku s názvem H2. V teoretické rovině by tato databáze mohla fungovat i v produkční verzi aplikace, nicméně i ti samotní vývojáři apelují na to, že by se měla používat čistě při vývoji aplikace a jejím debuggování. Tento typ databáze není totiž prakticky vůbec škálovatelný a samotné uložení databázové struktury včetně metadat do jediného nešifrovaného textového souboru není v žádném případě validním způsobem, jak nakládat s daty uživatele.

5.7 BACKEND WEBOVÉ APLIKACE

Pokud vývojáři pracují ve větších týmech, je jim práce rozdělena tak, aby mohli pracovat nezávisle na sobě a nemuseli tak čekat na implementaci nějaké části. Pokud je vývojář sám, má možnost si zvolit kroky, které nejrychleji povedou k výsledku. Existuje mnoho technik, které popisují, jakým způsobem by se mělo postupovat při vývoji, avšak jejich aplikace často vyžaduje mnoho zkušeností, a tudíž se někdy musí improvizovat a často spoléhat pouze na intuici. Systematicky to v tomto bodě po definici datové struktury znamená, že dalším logickým krokem bude implementace serveru.

Server zde bude plnit roli výpočetního a řídicího modulu, který bude webové aplikaci poskytovat prostředky skrze REST API pro práci s veškerými daty, které již zde byly popsány. Dále bude implementovat mechanismy, které budou vstupní data transformovat do struktur, se kterými bude opět nakládáno dle potřeb.

Dalším krokem by mělo být vytvoření plánu vývoje. Obecné problémy ze zadání převést na konkrétní problémy a rozhodnout priority a návaznosti. Není na škodu vytvořit modelové příklady různých situací, které pomohou odhalit nejasnosti, které na první pohled nemusí být úplně zřejmé. Pokud nastane situace, že ze zadání není možné některé kroky rozhodnout a jedná se o problémy, které by mohly zásadním způsobem změnit charakter vyvíjené aplikace, je nutné vždy kontaktovat zadavatele a ujistit se, jakým způsobem se bude daná situace řešit. Po tomto kroku je již možné uvažovat o reálné implementaci, kdy další nezbytnou částí vývoje bude třeba rozdělit jednotlivé úkoly tak, aby existoval plán pro každý modul, včetně pořadí, ve kterém by měly být implementovány.

5.7.1 Design a rozdělení modulů

Každá třída by měla implementovat potřebné metody, avšak jen do té míry, aby plnily jen takové operace, kde jejich charakter je popisován významem třídy. Bude kladen důraz i na zachování jednoduchosti a na čitelnost jednotlivých metod, aby je bylo možné efektivně testovat.

Již existující datový model bude sloužit jako odrazový můstek pro implementaci serveru. Každá definovaná entita bude mít svou vlastní servisní třídu, která bude nad uloženými a nově příchozími datovými strukturami implementovat základní operace umožňující jejich vytváření, mazání, čtení a modifikaci. V první části bude nutné vybudovat již konkrétní datový model,

který bude využit pro vytvoření tabulek v databázi. K tomu slouží entitní třída, ze které se bude tabulka vytvářet. Jedná se čistě o datovou třídu obsahující proměnné a základní převodní funkce, viditelno na obrázku níže.

```
package cz.upce.diplomathesis.rtpool.server.entity

import cz.upce.diplomathesis.rtpool.server.dto.QuizDto
import cz.upce.diplomathesis.rtpool.server.model.QuizModel
import javax.persistence.*
import javax.validation.constraints.NotBlank

@Entity
@Table(name = "quiz")
class Quiz(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long = Long.MIN_VALUE,
    var name: @NotBlank String = "",
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    var user: User? = null
) {
    fun toDto(): QuizDto {
        return QuizDto(id, name)
    }

    fun toModel(): QuizModel {
        return QuizModel(
            id, name, emptyList()
        )
    }
}


snappify.io
```

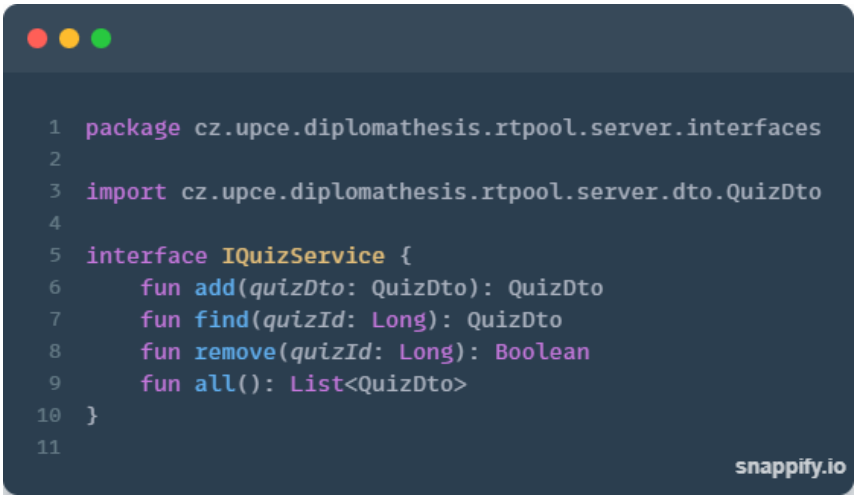
Obrázek 5.3 – Entita kvízu

Pomocí speciálních anotací je možné nastavit a specifikovat různá nastavení a omezení. Z takovéto třídy se následně vygeneruje v připojené databázi tabulka, kde jsou jednotlivá pravidla použita a plně reflektuje vytvořenou třídu.[3]

Aby bylo možné nad touto entitou vykonávat potřebné operace, musí existovat servisní třída, která tyto operace bude implementovat. Bude se jednat pouze o základní ovládání umožňující

[3] V případě zájmu popis fungování těchto a jiných funkcí je nutné nastudovat ve vhodné literatuře volně dostupné na internetu. V této práci bude popsán pouze základní charakter a nezbytné části, aby bylo možné chápat návaznosti a zvolené postupy.

vkládat nově přidané kvízy, možnost hledání dle identifikátoru, změna již existujících, jejich odstranění a vrácení všech přítomných. Demonstrace rozhraní je na obrázku níže.



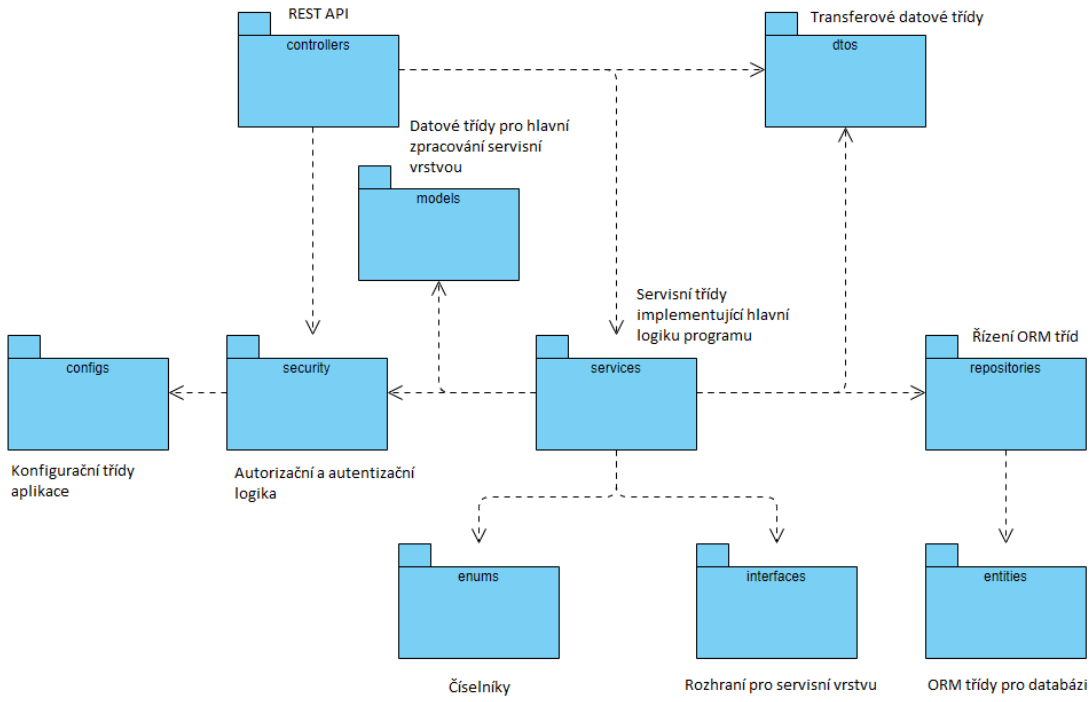
Obrázek 5.4 – Ukázka rozhraní pro entitu kvíz

Zobrazené rozhraní neobsahuje funkci modifikace, protože pro přidávání nového kvízu, v případě existujícího identifikátoru ve vstupním modelu, sama provede změnu nad existujícím záznamem, a v případě absence vytvoří nový. Implementace tohoto rozhraní bude obsahovat základní logiku pro práci s těmito vstupními daty, která jsou ale v jiné třídě, než je již zmíněná entita. Je obecně známo, že servisní třídy by měly pracovat s takzvanými modelovými třídami. Tyto třídy jsou odděleny od těch entitních a musí existovat převodník na obě strany. V momentě, kdy servisní třídu opouštějí data směrem ke klientovi, je zde přítomna další transformace na takzvanou DTO, což je třída čistě designována pro přenos skrze REST.

Někomu se ovšem může zdát, že jde o duplicitní vytváření tříd, které ve finále fungují stejně a jenom se jmenují jinak. Je nutné si uvědomit, že entitní třída obsahuje veškeré parametry, které nemusejí být vždy potřeba a mohou být uloženy ve formátech, se kterými se těžko pracuje a je nutné přetypovat datové typy nebo reagovat na prázdné hodnoty určitou logikou. V případě DTO tříd tomu není jinak. Například entita uživatele může obsahovat choulostivé informace, které je nutné vhodným převodem vyseparovat tak, aby byly odeslány jen takové data, která jsou nutná. Výsledný důvod může tedy být jak z hlediska bezpečnosti, tak výkonu, kdy odeslání většiny, byť neškodných dat může být zbytečné.

Obdobná struktura bude implementována pro zbývající entity, pro které budou jak servisní, tak koncové REST třídy nazývané kontroléry. V tomto bodě existuje již celkem robustní a logická programová struktura, která poskytuje základní operace pro veškeré entity. Obrázek

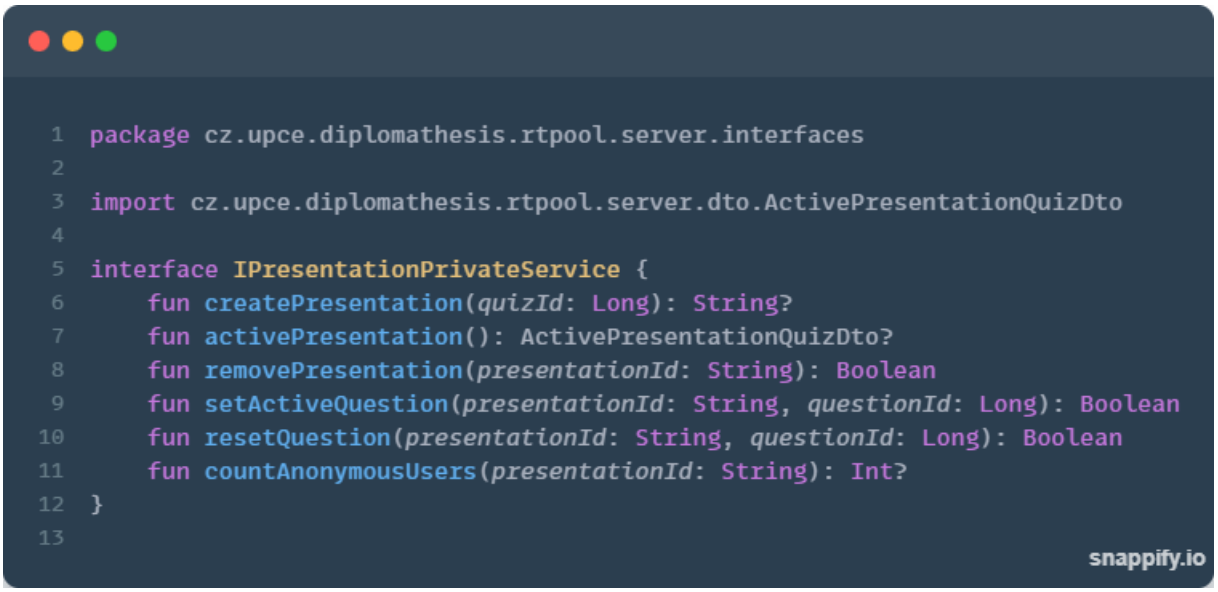
níže představuje výslednou strukturu balíčků aplikace. Nyní bude třeba vytvořit jádro implementující funkce realtimové ankety.



Obrázek 5.5 – Diagram balíčků serverové části

5.7.2 Jádro realtimové prezentační vrstvy serveru

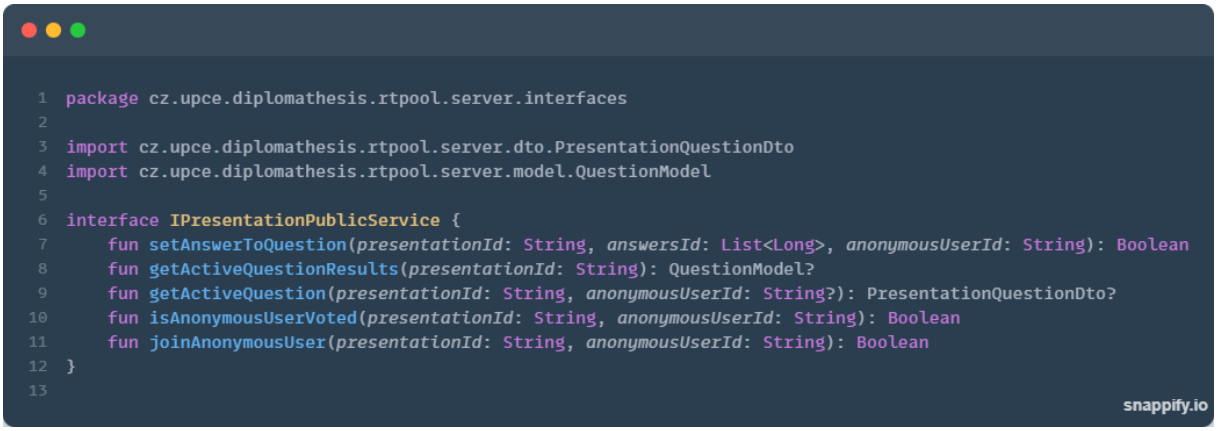
Aby se daly jednotlivé funkcionality vůbec definovat, bylo nutné je rozdělit na dva logické celky. Prvním je část modulu, jehož rozhraní je zobrazeno na obrázku níže, která se bude starat o operace, které jsou řízeny prezentujícím. Rozdělení je implementováno hlavně z hlediska bezpečnosti, protože všechna tato volání jsou možná pouze pod přihlášeným uživatelem, kdežto v případě veřejné druhé vrstvy nebude žádný validační mechanismus aplikován. Prezentujícímu musí být umožněno kvíz vytvořit a odstranit. Následně dle zadání bylo nutné implementovat mechanismus umožňující resetovat aktuální otázku a nastavit libovolnou otázku jako aktivní. Dodatečně bylo nutné implementovat funkci, která vrací aktuální prezentaci a počet aktivních uživatelů připojených na konkrétní prezentaci.



```kotlin
package cz.upce.diplomathesis.rtpool.server.interfaces

import cz.upce.diplomathesis.rtpool.server.dto.ActivePresentationQuizDto

interface IPresentationPrivateService {
    fun createPresentation(quizId: Long): String?
    fun activePresentation(): ActivePresentationQuizDto?
    fun removePresentation(presentationId: String): Boolean
    fun setActiveQuestion(presentationId: String, questionId: Long): Boolean
    fun resetQuestion(presentationId: String, questionId: Long): Boolean
    fun countAnonymousUsers(presentationId: String): Int?
}

```


Obrázek 5.6 – Rozhraní privátní prezentační vrstvy

Druhý modul, jehož rozhraní je zobrazeno na obrázku níže, je čistě pro uživatele interagující s otázkami kvízu. Tato volání nejsou vázána na konkrétní uživatele autorizovaných v aplikaci, ale pomocí UUID vygenerované na straně klienta. Uživateli v roli hlasujícího musí být umožněno odpovědět na aktivní otázku. Dále má možnost vrácení správných odpovědí. Musí být schopen aktivní otázku obdržet. Dodatečné funkce nutné pro korektní fungování je funkce, která informuje klientskou část, zda uživatel již hlasoval, což je výhodné, zejména pokud uživatel restartoval prohlížeč a funkce pro registraci do aktivního kvízu.



```kotlin
package cz.upce.diplomathesis.rtpool.server.interfaces

import cz.upce.diplomathesis.rtpool.server.dto.PresentationQuestionDto
import cz.upce.diplomathesis.rtpool.server.model.QuestionModel

interface IPresentationPublicService {
    fun setAnswerToQuestion(presentationId: String, answersId: List<Long>, anonymousUserId: String): Boolean
    fun getActiveQuestionResults(presentationId: String): QuestionModel?
    fun getActiveQuestion(presentationId: String, anonymousUserId: String?): PresentationQuestionDto?
    fun isAnonymousUserVoted(presentationId: String, anonymousUserId: String): Boolean
    fun joinAnonymousUser(presentationId: String, anonymousUserId: String): Boolean
}

```


Obrázek 5.7 – Rozhraní prezentační veřejné vrstvy

Jednotlivé implementace metod zde uvedených rozhraní nejsou nikterak zajímavou částí, protože jde ve své podstatě o kombinaci technik pracujících s kolekcemi a validacemi nad

nejrůznějšími strukturami. Uvedeny zde budou pouze vybrané ty části, které jsou z pohledu řešení nejzajímavější.

```
@Synchronized
override fun createPresentation(quizId: Long): String? {
    val userId = userService.getLoggedUserId()
    val presentationId = UUID.randomUUID().toString()

    if (presentationService.isQuizForUserAlreadyActive(userId)) {
        presentationService.setNewPresentationModel(
            presentationId, presentationService.buildPresentationModel(quizId, userId)
        )
        return presentationId
    }

    throw IllegalStateException("User already have active quiz")
}
```


Obrázek 5.8 – Příklad funkce vytvářející prezentační model

Výše zobrazená funkce vytváří prezentační model. Argumentem funkce je identifikátor kvízu, který má být spuštěn. Poté se vygeneruje speciální dočasný identifikátor, kterým bude nově vytvořená instance prezentace označena. Následuje kontrola, zda uživatel nemá již aktivní prezentaci. V případě, že ano, program vrátí výjimku o chybě a informuje jej o skutečnosti. V opačném případě se vytvoří nový model prezentace do kolekce typu hodnota a klíč, a návratovým typem je UUID prezentace.

Nepopsaným zbývajícím krokem, který byl záměrně zamlčen, je vrácení uživatele. V argumentu funkce není, tudíž se musí získat někde jinde. V tomto případě se jedná o takzvané JWT tokeny, které uživatel využívá jako svůj podpis. Tento token mimo jiné obsahuje roli uživatele včetně přihlašovacího jména. Tento token získá v momentě, kdy se úspěšně přihlásí. Aby nemusel pokaždé, když zasílá dotaz na server, vyplňovat uživatelské jméno a heslo, využívá tento token, který je na straně serveru pomocí privátního klíče rozluštěn, následně je ověřena jeho platnost a teprve poté je pokračováno v běhu zaslaného dotazu. Framework sám implementuje mechanismus, který umožňuje v momentě vykonávání dotazu identifikovat zdroj původu dle tokenu a následně možnost vypsat uživatele.

Obrázek 5.9 – Prezentační model aktivního kvízu

Tato datová struktura, která je zobrazena na obrázku výše, je základní stavební kámen celé aplikace. V momentě, kdy uživatel zadá příkaz o vytvoření prezentace, je právě tato struktura vytvořena. První částí při generování modelu je celý kvíz vytažen z databáze a transformován do vhodných struktur rozšiřující jej o atributy, které nemá jinak smysl trvale ukládat a následně vložen do tohoto modelu. Dokud je prezentace aktivní, je udržována v paměti RAM. Parametry třídy tvoří identifikátor UUID, identifikátor aktivní otázky, identifikátor prezentujícího, jeho aktuální stav, kolekce s uživateli, kteří zrovna odpověděli a kolekce všech aktivních uživatelů.

Tato celkem komplexní struktura je poté využívána pokaždé, kdy nastane nějaká změna, a to jak ze strany prezentujícího, tak hlasujících. Jelikož jsou data celkem často aktualizována, zejména v momentě hlasování, není vhodné tyto data ukládat do databáze. Struktura je uložena ve sdílené kolekci, kde jsou prezentace přístupné na základě klíčů, vygenerovaných SSID. Je umožněno několika různým prezentujícím spouštět najednou své prezentace ve stejnou dobu. Aby toto bylo vůbec možné, bylo by třeba vyřešit pár základních problémů.

Samotné vytváření prezentací problém není, každá prezentace obdrží své vlastní SSID a je striktně vázána na přihlášeného uživatele. Pokud by přihlášený uživatel chtěl sabotovat existující prezentaci jiného uživatele, nemá v podstatě žádnou možnost, jak to provést, protože v každé metodě v privátním modulu probíhá kontrola vlastníka. Pokud uživateli daná prezentace nepatří, nebude ze sdílené kolekce vymazána. Toto platí pro veškeré funkce v programu určené výhradně přihlášeným uživatelům.

Pro hlasující také existují mechanismy, které zamezí cílené i necílené nežádoucí změně stavů. Pokud má prezentace již aktivní otázku, nelze se jí již zúčastnit, což je obdoba whitelistu. Pokud uživatel již jednou odpovídal, další možnost mít nebude, dokud prezentující neresetuje aktivní otázku. Odpovědi uživatelů jsou drženy v paměti i poté, co se přechází na jinou otázku. V momentě, kdy prezentující přejde na již všemi odhlasovanou otázku, všem hlasujícím se ukážou jejich odpovědi včetně validace, pokud byla aktivní. Což znamená, že v době existence kvízu je tato historie ukládána. Z této kolekce odpovědí se následně generují statistiky dokončeného kvízu.

5.8 FRONTEND WEBOVÉ APLIKACE

Za předpokladu, že již spuštěný implementovaný server, který je napojen na databázi, má veškeré potřebné konfigurace a existuje vystavené REST API obsahující všechny nezbytné funkce, nastává čas implementace klientské části. Oproti serveru zde budou řešeny problémy jiného charakteru. A to především z oblasti zaměřující se na grafické prostředí a interní logiku pracující s daty generované v samotném klientu a data, která přicházejí ze serveru. V této části budou některé zajímavé ukázky jednotlivých komponent, avšak je třeba brát zřetel na fakt, že většina funkcí, které jsou implementovány, zastávají spíše repetitivní roli, bude podobně jako v předchozích podkapitolách rozebrána dílčí část.

5.8.1 Design a rozdělení modulů

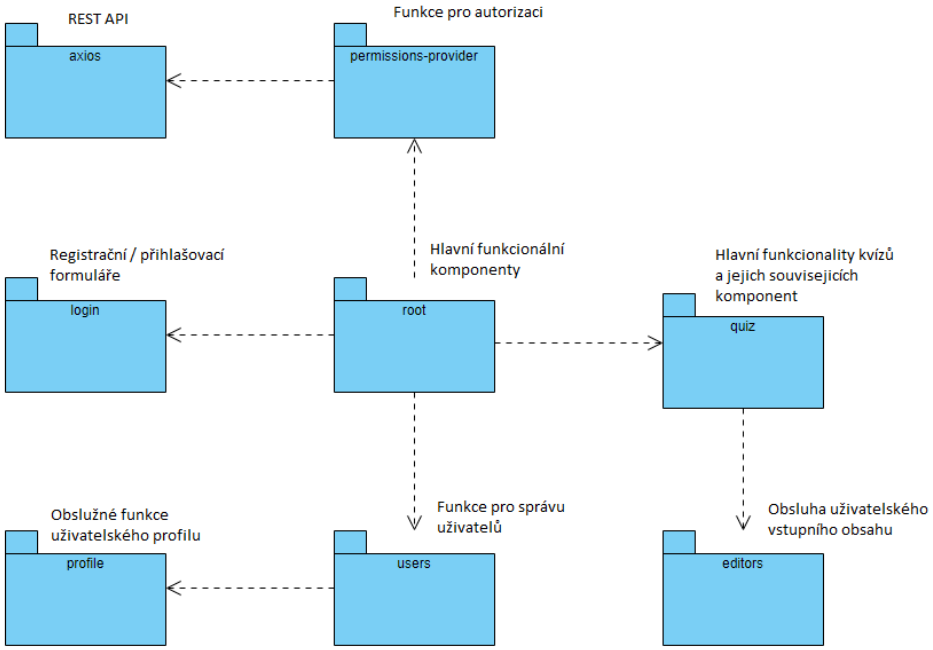
Podobně jako tomu bylo u serveru, dochází opět k logické separaci dílčích částí, avšak s trochu odlišným přístupem. Velmi podobně budou implementovány kontrolery, které budou zrcadlově odpovídat existujícímu REST API na serveru.

Obrázek 5.10 – Příklad některých funkcí volající kontrolér na serveru

Na příkladu je vidět o poznání rozdílná syntaxe, kde se již neutváří třída obsahující funkce, ale jedná se o samostatné asynchronní funkce, které jsou importovány do příslušných modulů jako statické metody. Grafický interface, který je určen pro přednášejícího a hlasujícího uživatele, je podrobně rozebrán v jedné z předchozích kapitol, takže se zde již tyto ukázky vyskytovat nebudou.

Hlavní součástí je kořen, který obsahuje další funkcionální komponenty. Ty zastávají funkci buď funkcionální komponenty, která obsahuje vnitřní logiku pracující s interními daty, kde výstupem bude grafické rozložení, které je možné vykreslit, nebo se jedná o třídu podobné té servisní, která zastává práci, zejména pokud jde o získání či transformaci dat.

Vývojář by měl svou snahu do vytváření jednotlivých komponent klást důraz na jejich potencionální opětovné použití. Každá funkcionální komponenta by měla obsahovat pouze vstupní data, pouze za předpokladu, že je potřebuje a měla by být využitelná v kterémkoliv místě projektu. Většinou se jedná o programově uzavřené mikro ekosystémy, které žijí svůj vlastní život a s ostatními komponentami prakticky nesdílí nic jiného než vstupní data. Celá aplikace je řízena pomocí událostí, které jsou vyvolány samotnými uživateli, nebo v případě využití websocketů serverovou částí.



Obrázek 5.11 – Diagram komponent klientské části

Existující struktura projektu je zobrazena na obrázku výše. Ke každé viditelné stránce je definována sada, která je ve většině případů brána jako samostatný celek, která obsahuje elementy obsažené na konkrétní stránce. Sdílené třídy budou zejména ty, které implementují chování využitelné universálně po celém projektu. Jedná se o třídy zastávající funkci kontrolerů, různých převodníků datových objektů nebo o globální sdílený datový prostor. Dále se to také týká vlastních komponent, kde například textové pole poskytující možnost vytváření formátovaného textu v otázkách a odpovědích je definováno knihovní funkcí, která poskytuje pouze základní komponentu.

Obrázek 5.12 – Obalená existující komponenta ReactQuill

Jelikož je předpokládáno, že tato komponenta bude univerzálně využívána na více místech, byla obalena funkcí, která již specifikuje konkrétní nastavení knihovní komponenty, a je rozšířena o funkce nutné pro korektní fungování aplikované logiky. Příklad takové komponenty je zobrazen na obrázku výše. Posledním typem jsou komponenty, které jsou většinou složeninou jiných vytvořených komponent, kde zastává roli rodiče a je vstupní branou pro jednotlivé potomky.

Jelikož je kód těchto komponent velmi dlouhý, není možné jej zde rozumně vyobrazit, a proto se jim nebude věnovat větší pozornost. Navíc se jedná o stále stejné opakující se vzory, které pro čtenáře nemají již další informační hodnotu. Rámci návrhu problematiky klientské části zde byly řečeny ty nejdůležitější body, na základě, kterých se postupovalo při jeho návrhu a následném vývoji.

5.9 DESIGN REALTIMOVÉ KOMUNIKACE

V momentě, kdy se uživatel připojí do aktivního kvízu, je automaticky spuštěna funkce, kde za pomocí knihovny STOMP proběhne konfigurace nutná pro vytvoření instance. Nejdůležitější vstupní parametry funkce tvoří URL odkazující na adresu serveru a inicializaci odběratelské funkce obsahující dva nezbytné parametry. Prvním parametrem je textový řetězec

identifikující konkrétního producenta a název funkce, která bude zavolána vždy, kdy server odešle data.

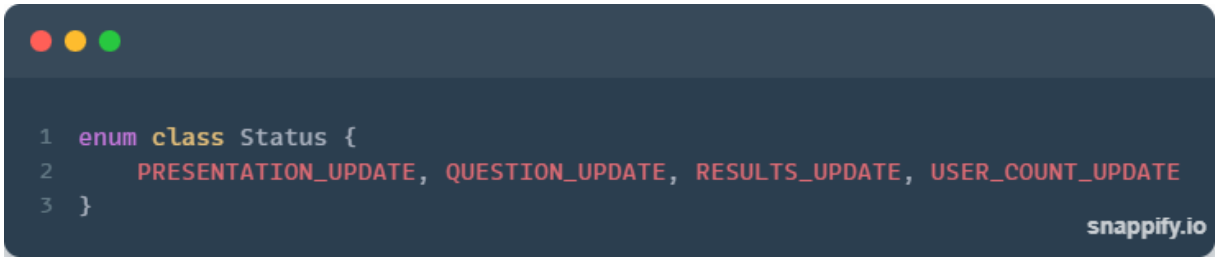
Obrázek 5.13 – Funkce realizující inicializaci STOMP klienta

Jedním z hlavních požadavků na aplikaci je, aby ve stejnou dobu mohli různí prezentující spustit své prezentace. Aby toto bylo realizovatelné, je nutná určitá koordinace serveru a klienta. Prezentacím v době jejich vytváření je vygenerováno UUID, které je platné pouze po dobu existující instance. Aby klient obdržel pouze zprávy, které se ho týkají, registruje svůj odběr právě na UUID prezentace. Tím je zaručeno separování od ostatních běžících prezentací a problém je tímto vyřešen. Implementace této funkce je zobrazena na obrázku výše.

5.9.1 Systém zpráv

Zpráva, kterou klient prostřednictvím notifikace obdrží, je ve formátu JSON a je nutné provést konverzi, ze které je možné získat již odeslanou informaci. Systém těchto zpráv je

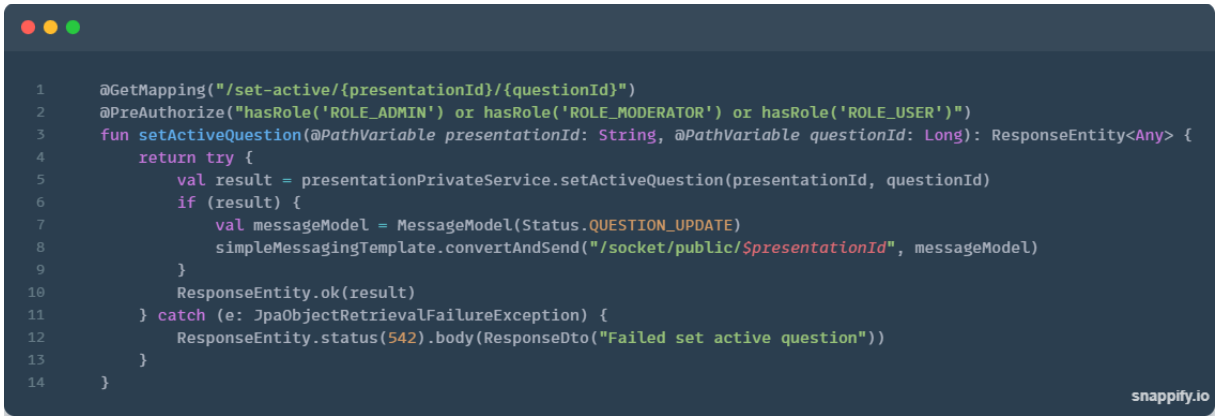
postaven na enumeračních hodnotách, kterými prezentující řídí aktuální chování hlasujících uživatelů, viz. obrázek níže.



```
enum class Status {
    PRESENTATION_UPDATE, QUESTION_UPDATE, RESULTS_UPDATE, USER_COUNT_UPDATE
}
```


Obrázek 5.14 – Ukázka třídy řídících stavů

Pokaždé, kdy prezentující provede změnu v prezentaci, tuto změnu musí všichni klienti, kterých se to týká, obdržet.



```
@GetMapping("/set-active/{presentationId}/{questionId}")
@PreAuthorize("hasRole('ROLE_ADMIN') or hasRole('ROLE_MODERATOR') or hasRole('ROLE_USER')")
fun setActiveQuestion(@PathVariable presentationId: String, @PathVariable questionId: Long): ResponseEntity<Any> {
    return try {
        val result = presentationPrivateService.setActiveQuestion(presentationId, questionId)
        if (result) {
            val messageModel = MessageModel(Status.QUESTION_UPDATE)
            simpleMessagingTemplate.convertAndSend("/socket/public/$presentationId", messageModel)
        }
        ResponseEntity.ok(result)
    } catch (e: JpaObjectRetrievalFailureException) {
        ResponseEntity.status(542).body(ResponseDto("Failed set active question"))
    }
}
```


Obrázek 5.15 – Ukázka funkce nastavující změnu aktivní otázky prezentujícími

Na obrázku výše je implementace funkce, která kombinuje dvě metody síťové komunikace. První je registrace klasické REST funkce, kterou volá prezentující v momentě, kdy žádá o změnu aktivní otázky na jím zvolenou otázku. Druhá odesílá odpověď všem odebírajícím za pomoci websocketů.

Nejprve proběhne pokus o změnu otázky. Pokud toto selže, nejčastěji z důvodu, že prezentace již není aktivní, vrátí zpátky informaci o selhání požadavku. V případě, že se aktivní otázku nastavit podařilo, je vygenerována instance obsahující požadavek, který je odeslán všem klientům, kteří mají svou registrovanou funkci nastavenou na příslušné UUID prezentace.

Obrázek 5.16 – Reakce klienta na obdrženou zprávu

Na obrázku výše je reakce klienta v roli prezentujícího na obdrženou zprávu.

5.10 SHRNUTÍ

Celá kapitola popsala postup při vytváření aplikace s dodatečnými vysvětleními některých vybraných částí kódu včetně podrobného rozboru některých řešených problémů. Ze strany serveru je nutné dodat, že problematika, kterou se tato kapitola zabývala, je více než obecná, a ve své podstatě aplikovatelná na širokou škálu různých podobných frameworků, z čehož by se daly jednotlivé myšlenkové pochody naprosto bez problému aplikovat jinde. To samé platí u knihoven poskytující možnost budování realtimových aplikací. Využitá knihovna splnila dokonale svůj účel, stejně tak by to dokázaly jiné knihovny. Komplexita projektu byla díky těmto prostředkům dramaticky redukována, přestože bylo nutné i tak vyřešit některé ne úplně triviální problémy, jako byla například aktivní izolace prezentací nebo například udržování aktivní instance prezentace v paměti.

ZÁVĚR

Zadání projektu bylo úspěšně splněno. Výsledkem této práce je plně fungující řešení poskytující software, který je možné využít prakticky v kterémkoliv odvětví. Uživatel má možnost vytvářet interaktivní kvízy, s možností vytvářet otázky a odpovědi za pomocí editoru umožňující formátování textu a vkládání různých grafických elementů. Dále má možnost správy registrovaných uživatelů a správy spuštěného kvízu. V textové části byla provedena rešerše realtimových technologií na webu a na základě jejího výsledku byla navržena a implementována aplikace.

Základní myšlenka tohoto projektu byla poskytnout univerzitám software, který mohou využívat bez jakýchkoliv licenčních omezení, poskytující již zmíněné funkcionality s cílem obohatit stávající výuku o zábavnou formu interakce se studenty. Pro prezentujícího je následně vytvářena velmi důležitá zpětná vazba, která je důležitým měřítkem určující kvalitu jeho přednesu a schopností jednotlivých studentů. Tyto informace může využít k tomu, aby upravil výuku problematických částí kurzu, u kterých třeba ani nepředpokládá, že by studentům mohly způsobovat problémy.

Jedná se o práci, která bude mít reálné využití a nebude se jednat o další aplikaci, která pouze plní účel potvrzení navrženého abstraktního konceptu. V budoucnu je žádoucí, aby se toto řešení dále udržovalo a rozšiřovalo o další funkce. Software bude zcela jistě poskytovat silnou konkurenční výhodu napříč trhem, kde ve své podstatě nemá přímou konkurenci v poskytovaných funkcích.

Práce poskytla velmi cenné zkušenosti v oblasti fungování realtimových technologií a dalších strukturálních konceptů vývoje, včetně návrhu webových aplikací. Nutno podotknout, že návrh uživatelského rozhraní, hlavně po grafické stránce, nebylo vůbec snadné vymyslet. Díky dostupnosti kvalitních materiálů a předpřipravených šablon lze poměrně snadno nalézt inspiraci a bez výrazného grafického cítění vyprodukovat přívětivé prostředí pro uživatele.

POUŽITÁ LITERATURA

AGGARWAL, Anuradha. *Short Polling vs Long Polling vs Web Sockets* [online]. 14 March 2021 [cit. 2022-08-12]. Dostupné z: https://anuradha.hashnode.dev/short-polling-vs-long-polling-vs-web-sockets

BABEL, Nicolas. *What is Server-Sent Events?. Axway* [online]. 27 September 2017 [cit. 2022-08-12]. Dostupné z: https://blog.axway.com/learning-center/apis/api-streaming/server-sent-events

BENEŠ, Jiří. Neurobiologie. WikiSkripta [online]. 16. 02 2010 [cit. 2022-08-12]. Dostupné z: https://www.wikiskripta.eu/w/Neurobiologie

BRABENEC, Luboš. OPAKOVÁNÍ NEUROVĚD. *Informační systém Masarykovy univerzity* [online]. 2018 [cit. 2022-08-12]. Dostupné z: https://is.muni.cz/el/phil/podzim2018/PS_BA026/um/opakovanineuroved.pdf

BRANDO, Carolina. How is WebRTC transforming the communication landscape. *ApiRTC* [online]. 27 September 2017 [cit. 2022-08-15]. Dostupné z: https://apirtc.com/how-is-webrtc-transforming-the-communication-landscape/

Event-driven architecture with Pub/Sub. *Google Cloud* [online]. 2022-08-04 [cit. 2022-08-12]. Dostupné z: https://cloud.google.com/solutions/event-driven-architecture-pubsub

FETTE, Ian; MELNIKOV, Alexey. RFC 6455 The WebSocket Protocol [online]. IETF, [online]. December 2011 [cit. 2022-08-12]. Dostupné z: https://www.rfc-editor.org/rfc/rfc6455 prosinec 2011. Kapitola 1.7.

GANESAN, Bharathvaj. *Polling vs SSE vs WebSocket– How to choose the right one* [online]. 31 July 2018 [cit. 2022-08-12]. Dostupné z: https://codeburst.io/polling-vs-sse-vs-websocket-how-to-choose-the-right-one-1859e4e13bd9

DEMIR, Hamit. What is WebRTC and how does it work? Real-time communication with WebRTC. *And Media* [online]. 19 January 2021 [cit. 2022-08-15]. Dostupné z: https://antmedia.io/webrtc-transforms-live-streaming/

DUTTON, Sam. Get started with WebRTC. *Web.dev* [online]. 24 November 2020 [cit. 2022-08-15]. Dostupné z: https://web.dev/webrtc-basics/

HUDEC, Tomáš. KIT/NNOS1 2021–2022 – Operační systémy 1: 12 Víceprocesorové systémy, systémy reálného času a vestavěné systémy. *Moodle* [online]. 2016 [cit. 2022-08-12]. Dostupné z: https://moodle.upce.cz/moodle/mod/book/view.php?id=244186&chapterid=3025#OS-12-2--RT

ILYUKHA, Vitaliy. Is React.js Fast?. *Jelvix* [online]. 2022 [cit. 2022-08-15]. Dostupné z: https://jelvix.com/blog/is-react-js-fast

Intro to Typescript: Typescript. In: *Https://towardsdev.com/intro-to-typescript-c81593ae1147?gi=e97cb9604b7d: Towards Dev* [online]. 28 September 2021 [cit. 2022-08-12]. Dostupné z: https://logos-download.com/9747-react-logo-download.html

Java Spring Boot: What is Java Spring Boot? [online]. IBM. IBM Cloud Education, 25 March 2020 [cit. 2022-08-12]. Dostupné z: https://www.ibm.com/cloud/learn/java-spring-boot

KAPOOR, Ajay. Types of Real-Time Web Applications You Can Build With Node.js Technology. *LEGGETTER* [online]. 28 October 2013 [cit. 2022-08-12]. Dostupné z: https://javascript.plainenglish.io/types-of-real-time-web-applications-you-can-build-with-node-js-technology-ba2af2af888a

Kotlin logo download for free Source: https://logo-logos.com/. In: *Logo-Logos.com* [online]. 2022 [cit. 2022-08-12]. Dostupné z: https://logo-logos.com/kotlin-logo-5.html

Kotlin vs java. In: *Mediaan* [online]. 28 June 2018 [cit. 2022-08-12]. Dostupné z: https://mediaan.com/mediaan-blog/kotlin-vs-java

LAKHWINDER, Kumar. *Understanding the Event-driven Architecture* [online]. In: . 3 September 2019 [cit. 2022-08-12]. Dostupné z: https://softobiz.com/understanding-the-event-driven-architecture/

LEGGETTER, Phil. History, Background, Benefits & Use Cases of Realtime. *LEGGETTER* [online]. 28 October 2013 [cit. 2022-08-12]. Dostupné z: https://www.leggetter.co.uk/2013/10/28/history-background-benefits-use-cases-realtime.html

LIU, Jane W. S. *Real-time systems*. Upper Saddle River: Prentice Hall, c2000. ISBN 0130996513. What is a webhook?. Red Hat - We make open source technologies for the enterprise [online]. 1 June 2022 [cit. 2022-08-09]. Dostupné z: https://www.redhat.com/en/topics/automation/what-is-a-webhook

What is a webhook?. Red Hat - We make open source technologies for the enterprise [online]. 1 June 2022 [cit. 2022-08-09]. Dostupné z: https://www.redhat.com/en/topics/automation/what-is-a-webhook

NYKLÍČEK, Jaromír. Webové aplikace pracující v reálném čase. Brno, 2013. Diplomová práce. Masarykova univerzita. Vedoucí práce RNDr. Radek Ošlejšek, Ph.D.

POSNICK, Jeff. Using WebTransport. *Web.dev* [online]. 18 July 2022 [cit. 2022-08-15]. Dostupné z: https://web.dev/webtransport/

RAI, Rohit. Socket. IO Real-Time Web Application Development. Birmingham: Packt Publishing, 2013. ISBN 978-1-78216-078-6

React. In: *Logos Download* [online]. 2022 [cit. 2022-08-12]. Dostupné z: https://logos-download.com/9747-react-logo-download.html

Real-Time Systems Overview: The Need for Real-Time Systems. *Intel* [online]. (nedatováno)2022 [cit. 2022-08-12]. Dostupné z: https://www.intel.com/content/www/us/en/robotics/real-time-systems.html

SEDRUBAL, -. Protocol Stack of HTTP/3 compared to HTTP/1.1 and HTTP/2. In: *Wikipedia* [online]. 16 February 2022 [cit. 2022-08-15]. Dostupné z: https://en.wikipedia.org/wiki/HTTP/3#/media/File:HTTP-1.1_vs._HTTP-2_vs._HTTP-3_Protocol_Stack.svg

STICHBURY, Jo. The realtime web: evolution of the user experience. Ably [online]. 28 July 2022 [cit. 2022-08-14]. Dostupné z: https://ably.com/blog/the-realtime-web-evolution-of-the-user-experience

STOMP Protocol Specification, Version 1.2. The Simple Text Oriented Messaging Protocol [online]. 22. 10 2012 [cit. 2022-08-11]. Dostupné z: https://stomp.github.io/stomp-specification-1.2.html

THOMSON, Martin. WebTransport Explainer: Problem and Motivation. *GitHub* [online]. 25 May 2022 [cit. 2022-08-15]. Dostupné z: https://github.com/w3c/webtransport/blob/main/explainer.md

TOMAN, František. Real-time webové aplikace [online]. Praha, 2012 [cit. 2022-08-10]. Dostupné z: https://insis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=33191. Bakalářská práce. Vysoká škola ekonomická v Praze.

WANG, Vanessa, Frank SALIM a Peter MOSKOVITS. The definitive guide to HTML5 WebSocket. [Berkeley, Calif.]: Apress, [2013]. Expert's voice in Web development. ISBN 9781430247401.

What are REST Hooks? What are they not?. *REST Hooks* [online]. ZAPIER, 27 August 2013 [cit. 2022-08-12]. Dostupné z: https://resthooks.org/docs/

What is a webhook?. Red Hat - We make open source technologies for the enterprise [online]. 1 June 2022 [cit. 2022-08-09]. Dostupné z: https://www.redhat.com/en/topics/automation/what-is-a-webhook

What Is an Event-Driven Architecture?. *HAZELCAST* [online]. 2014 [cit. 2022-08-12]. Dostupné z: https://hazelcast.com/glossary/event-driven-architecture/

What's a Webhook?. TWILIO SendGrid [online]. 24 June 2014 [cit. 2022-08-09]. Dostupné z: https://sendgrid.com/blog/whats-webhook/

YUHAS, Daisy, ed. Speedy Science: How Fast Can You React? [online]. 24 May 2012 [cit. 2022-08-10]. Dostupné z: https://www.scientificamerican.com/article/bring-science-home-reaction-time/

ZLATKOV, Alexander. How JavaScript works: WebRTC and the mechanics of peer to peer networking. *SessionStack Blog* [online]. 5 July 2018 [cit. 2022-08-14]. Dostupné z: https://blog.sessionstack.com/how-javascript-works-webrtc-and-the-mechanics-of-peer-to-peer-connectivity-87cc56c1d0ab