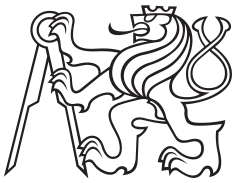
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Interaktivní tabule pro vchod do FELu v Dejvicích

David Bubeník

Vedoucí: Ing. Jiří Šebek
Studijní program: Softwarové inženýrství a technologie
Leden 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bubeník** Jméno: **David** Osobní číslo: **474635**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Interaktivní tabule pro vchod do FELu v Dejvicích

Název bakalářské práce anglicky:

Interactive board for entrance to FEL in Dejvice

Pokyny pro vypracování:

Cílem práce je navrhnout a realizovat funkční interaktivní tabuli pro vchod do budovy fakulty elektrotechnické ČVUT v Dejvicích. Jedná se o podobný projekt jako Interaktivní fasáda FEL ČVUT „LINKY“.
Cíle pro tuto práci jsou:
- Realizujte hardware tabuli (terminál) a zanalyzujte způsob, jak tabuli vzdáleně ovládat a nahrávat na ní obsah. Tabule bude obsahovat ovládací prvky (Nahoru, dolu, doprava, doleva, enter), připojení k internetu, webové rozhraní. Z funkčních požadavků se zaměřte na systém fronty kde každý uživatel má limitovaný čas,nahrávání a tvorba statického/dynamického obsahu, code snippety konzumující poskytnuté API k ovládání HW tabule a možnost pro uživatelé napsat jednoduchou hru, nebo interaktivní obsah
- Navrhněte a implementujte software pro tabuli.
- Implementujte klientskou část (aplikace tabule) i část serverovou (administrační rozhraní).
- Vytvořte prototyp, který otestujete (výsledky zhodnoťte). Testy proveďte jak uživatelské tak integrační.
- Vytvořte dokumentaci (uživatelskou i programátorskou)

Seznam doporučené literatury:

1) Masse, Mark. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. " O'Reilly Media, Inc.", 2011.
2) Upton, Eben, and Gareth Halfacree. Raspberry Pi user guide. John Wiley & Sons, 2014.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek, kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2020** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Děkuji panu Ing. Jiřímu Šebkovi za odborné konzultace, vedení bakalářské práce i jí předcházejícího semestrálního projektu. Děkuji také své rodině, přítelkyni a přátelům za obrovskou morální podporu při psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 5. ledna 2021

Abstrakt

Cílem práce je analyzovat a navrhnout projekt funkční interaktivní tabule pro vchod do budovy Fakulty elektrotechnické ČVUT v Dejvicích. Myšlenkou se jedná o podobný projekt jako je Interaktivní fasáda Fakulty elektrotechnické ČVUT „Linky“, ale s odlišným cílem a využitím. Nemalou součástí práce je také popis implementace návrhu vypracovaného s důrazem na budoucí rozšiřitelnost projektu a uživatelské testování této implementace.

Klíčová slova: nodejs, javascript, raspberrypi, mvc, typescript, rest, api, http, terminal, displej, zábava, web, server, vps, linux, debian, nestjs, typeorm

Vedoucí: Ing. Jiří Šebek

Abstract

The goal of this thesis is to analyze and design a project of a functional interactive whiteboard for the entrance to the building of the Faculty of Electrical Engineering CTU in Dejvice. The idea is similar to project of Interactive Facade of the Faculty of Electrical Engineering CTU "Linky", but with a different goal and use. An important part of the work is also a description of the implementation of the described design developed with emphasis on the future extensibility of the project and user testing of this implementation.

Keywords: nodejs, javascript, raspberrypi, mvc, typescript, rest, api, http, terminal, display, entertainment, web, server, vps, linux, debian, nestjs, typeorm

Title translation: Interactive board for entrance to FEL in Dejvice

Obsah

Obrázky

Výpisy

Kapitola 1
Úvod

1.1 Předmluva

Asi každý dlouhodobější návštěvník dejvického kampusu v Praze, přicházející od Vítězného náměstí, si musel všimnout pěti svislých světelných pruhů na boku budovy Fakulty elektrotechnické ČVUT. Tento projekt Institutu Intermédií FEL ČVUT nazvaný "Linky", který vznikl v roce 2015, je úžasným oživením přilehlého Parku Vítězství. Na této instalaci se velice často objevují různé barevné animace a tvůrci dokonce v roce 2019 umožnili registrovaným a potvrzeným uživatelům zobrazovat animace vlastní. Právě tato světelná instalace mi byla inspirací pro téma této bakalářské práce.

Cílem této práce je popsat způsob realizace interaktivní tabule (dále jen terminál) do vchodu budovy Fakulty elektrotechnické ČVUT v Dejvicích, na které je možné zobrazit složitější grafické prvky. Na samém začátku práce popíši omezení projektu "Linky" a provedu analýzu využití instalace terminálu do vchodu budovy. V dalších kapitolách detailně představím návrh jednoho možného řešení a předvedu praktickou implementaci tohoto návrhu. Na závěr analyzuji výsledky provedeného uživatelského testování, zhodnotím výsledek implementace a předložím, jakým způsobem by bylo možné finální implementaci ještě vylepšit.

1.2 Popis omezení projektu "Linky"

Projekt Institutu Intermédií FEL ČVUT "Linky", který se poprvé rozsvítil v únoru roku 2015, je světelný projekt, který pomocí pěti svislých vysoce svítivých segmentovaných sloupů dělá z fasády budovy FEL impozantní projekci. Na této projekci lze zobrazit jednoduché tvary jako například vlajku České republiky, binární hodiny, odjezdy MHD a dokonce i zjednodušenou verzi hry Space Invader. Přestože mají jednotlivé svislé sloupy několik desítek zobrazovacích bodů, horizontálně je jich pouhých 5 a navíc s obrovskými mezerami, které tvoří okna budovy. To značně omezuje možnosti zobrazování složitějšího obsahu, protože by nebyl rozeznatelný.

Tato práce se proto zabývá sice podobnou myšlenkou jako jsou "Linky", ale se snahou ukázat výhodu další menší instalace za použití zobrazovacího

zařízení s vyšším rozlišením a standardním rozvržením zobrazovacích bodů, která by mohla být umístěna přímo ve vchodu do budovy.

Část I

Teoretická část

Kapitola 2
Analýza

V této kapitole bude provedena analýza možných využití terminálu při instalaci ve vchodu do budovy FEL. Dále bude nastíněno několik variant, které jsou dostupné pro realizaci a instalaci terminálu. Nakonec bude vybrána jedna varianta, která bude detailněji rozebrána v následujících kapitolách.

2.1 Možná využití

Přemýšlíme-li nad terminálem jako nad dobře viditelným zobrazovacím zařízením, nabízí se nám hned několik možností, co lze prezentovat za obsah, aby přitáhl pozornost kolemjdoucího a poskytl mu nějaké informace.

2.1.1 Využití bez vstupního rozhraní

Pokud by se terminál skládal pouze ze zobrazovacího zařízení, bez možnosti uživatelského vstupu, můžeme definovat tyto možná využití:

Zobrazování užitečných informací. Na terminálu bude možné zobrazovat v závislosti na čase nebo jiných okolnostech novinky, počasí, stav školních zařízení, technické údaje, apod. Terminál by tudíž mohl přispět k informovanosti příchozích ohledně dění kolem nich.

Prezentace a zviditelnění. Na terminálu bude také možné zobrazovat prezentace školních projektů nebo spolků. Pokud by byla zajištěna dostatečná viditelnost terminálu, je možné, že vznikne poptávka po zobrazování reklamních sdělení či bannerů podobně jako na kioscích na veřejných místech.

2.1.2 Využití se vstupním rozhraním

Pokud bude součástí terminálu i nějaký typ vstupního zařízení (klávesnice, gamepad, apod.), pomocí kterého uživatel bude moci ovládat dění na zobrazovacím zařízení, lze definovat i následující případy užití:

Interaktivní průvodce. V rámci jiné bakalářské práce vznikla interaktivní mapa budovy FELu, která by mohla pomoci novým studentům s orientací po budově. Terminál by tudíž mohl usnadnit přístup k této mapě. Noví studenti

by tak získali okamžitý přehled a informace o tom, že takový projekt existuje a je dostupný.

Dotazníky a ankety. Terminál by mohl zvýšit účast ve školních dotaznících a anketách.

Zábava. Studenti by měli možnost prezentovat na terminálu svůj vlastní obsah, který by nejdříve předali online ke kontrole moderátorem, ten by se poté automaticky zařadil do fronty k zobrazení. Systém by poté střídal uživatelské obsahy v časových intervalech. Tvůrci obsahu by měli k dispozici i vstupní zařízení terminálu, díky kterému by bylo možné vytvářet například interaktivní prezentace či hry. Tato interakce by se mohla stát mezi studenty a ostatními návštěvníky oblíbenou aktivitou a zpříjemněním dne při příchodu do budovy.

2.1.3 Shrnutí

Všechny výše zmíněné body by se na terminálu mohly v závislosti na potřebě, času a jiných okolnostech střídat. Tím by vznikl interaktivní zážitek, který by mohl mít pro své okolí zábavný, informativní i naučný charakter.

Pokud bychom se vrátili k bodu 1.2 , kde jsou popsána omezení instalace “Linky”, je jasně vidět, že na této instalaci je možné realizovat pouze první bod z 2.1.1 a poslední bod z 2.1.2. Navíc pouze ve velmi omezeném měřítku kvůli limitaci zobrazovacího zařízení.

Pro tuto práci byl vybrán poslední bod z 2.1.2, jehož využití bude dále rozvinuto a nakonec i implementováno, protože se jedná o nejzajímavější typ interakce s terminálem. Navíc díky myšlence fronty, nahrávání vlastního obsahu a jeho střídání lze tento systém využít k zobrazování obsahu ostatních bodů tohoto seznamu možných využití.

2.2 Dostupné metody pro realizaci

Před detailním rozebráním jednoho konkrétního návrhu implementace by bylo vhodné předvést seznam některých možností, které se nám nabízejí při rozvaze o realizaci terminálu.

2.2.1 Hardware

Je vhodné se v našich představách omezit na malý terminál velikostně tak, aby se vešel do vchodu budovy a příliš nezabíral prostor chodby. Samozřejmě lze přemýšlet i ve větším měřítku, ale to není cílem ani zadáním této práce. V této kapitole nebudou popsány možnosti fyzické konstrukce pouzdra terminálu, protože závisí na velkém množství faktorů jako například prostředí kolem prostoru, velikost tohoto prostoru, apod.

Zobrazovací zařízení

Vzhledem k tomu, že hlavním požadavkem pro realizaci terminálu je zobrazovací zařízení s vysokým rozlišením, naskytnou se nám při výběru tohoto zařízení tyto možnosti:

Konvenční spotřební zařízení. Příkladem může být počítačový monitor nebo televize jakéhokoliv typu. Dnes standardně prodávané modely dosahují vysokého rozlišení HD i v nejnižších cenových kategoriích. Výhodou použití těchto zařízení je jejich vybavenost standardními konektory pro přenos obrazu. Při výběru pro naše účely je ale důležité dbát na fyzickou velikost (hlavně tloušťku), možnosti mechanické instalace a životnost obrazových bodů. Chceme-li, aby byl terminál uzavřený systém v určitém pouzdru či bedně, jsou tyto faktory důležité proto, aby byla dodržena bezpečnostní nařízení místa instalace a zároveň se nezkracovala životnost terminálu.

Spotřební zařízení vyrobená pro daný účel. Další variantou jsou zařízení přímo určená a vyrobená pro použití v podobných masově vyráběných terminálech či kioscích. Tyto zobrazovací zařízení jsou vyrobena s důrazem na bezpečnost, stabilitu a životnost. Bohužel jejich nevýhodami může být vyšší cena, nízká dostupnost nebo i propertiální konektory a protokoly přenosu obrazu.

Projekce. Je zde i možnost využití metody projekce obrazu například konvenčním projektorem. Výhodou může být fakt, že i velice malý projektor dokáže vytvořit velkoplošný obraz s vysokým rozlišením. Mezi nevýhody patří například nutnost nízkého okolního osvětlení.

Vlastní řešení. Samozřejmě je zde i možnost vyrobit si (nebo si nechat vyrobit) vlastní zobrazovací zařízení na míru. Vhodně se může jevit například pravidelná mřížka LED diod (matrix) s vlastní řídící jednotkou. Tento způsob, ačkoli díky němu můžeme dosáhnout jakéhokoliv výsledku, je velice komplikovaný, a pokud bychom chtěli splnit náš požadavek velkého množství obrazových bodů, i velice nákladný.

Vstupní zařízení

Ve výbavě terminálu nesmí chybět ani nějaké vstupní zařízení, které nám umožní zobrazovat na terminálu obsah, který vyžaduje od sledujících zpětnou vazbu. Existuje několik variant, jak od uživatele lze interakční data získat:

Tlačítka. Jedním z nejzákladnějších vstupních zařízení je mechanické tlačítko. Díky němu lze terminálu předávat údaj stisknuto/nestisknuto. Vhodným množstvím a umístěním lze dosáhnout dobrého pokrytí možností, jak terminál ovládat. Mechanická tlačítka mívají velkou životnost a odolnost vůči špatnému zacházení. Jejich nevýhodou je ale přílišná jednoduchost a pouhé dva logické stavy.

Lineární vstupy. Mezi analogové lineární vstupy patří vše od jednoosých posuvníků až po tříosé joysticky. Mezi nejvíce rozšířené patří joystick dvouosý s tlačítkem při stisku. Díky lineárnímu průběhu těchto vstupních prvků lze terminálu předávat složitější informace, které se mohou hodit například při hraní her nebo transformacích virtuálních objektů. Nevýhodou je ale nižší odolnost a vyšší poruchovost kvůli složitější konstrukci.

Dotykové plochy. Dotykové plochy jsou v poslední době velice rozšířenou technologií díky jednoduchosti a praktičnosti. Průhledná dotyková plocha umístěná nad zobrazovacím zařízením poskytuje výhodu rozšířené možnosti interakce, protože se klikatelný obsah může měnit. Tato technologie má při využití v terminálech a kioscích značné výhody, a proto je komerčně velice rozšířená.

Ostatní vstupy. Při výběru vstupního zařízení v dnešní pokročilé době již není nutné omezovat se pouze na senzory vyžadující mechanický pohyb. Existuje řada jiných vstupních zařízení, jako jsou například světelné senzory, infračervené senzory, kamery s rozpoznáním gest, senzory zrychlení, apod. Při výběru těchto typů senzorů je ale třeba dbát na to, že vstupní prvky terminálu musí být snadno použitelné a odolné.

Tyto varianty lze i kombinovat. Příkladem této kombinace je například gamepad, který je určený pro hraní her.

Řídící jednotka

Vstupní a výstupní zařízení nemohou sama od sebe pracovat, pokud nejsou řízena nějakou mikroprocesorovou jednotkou. Zvláště v našem případě, kdy se snažíme zobrazovat dynamický obsah ve vysokém rozlišení, je třeba, aby byla jednotka dostatečně výkonná. Hodí se také možnost komunikovat s řídící jednotkou vzdáleně. Naskytují se nám tyto možnosti:

Konvenční osobní počítač. Použití standardně prodávaného osobního počítače (stolní PC) je pro naše účely zcela dostačující. Nevýhodou ale může být vysoká cena, velikost a malé množství variant. Konvenční zařízení také většinou svojí konstrukcí zamezují možnostem instalace a úpravám. Během posledního desetiletí však přišel rozmach mobilních zařízení pro osobní použití. Patří mezi ně tablety, mobilní telefony, laptopy apod. Jedná se většinou o jednodeskový počítač, jehož součástí je i vstupní a výstupní zařízení. To vše ve velmi malé konstrukci. Nevýhodou, při využití pro terminál, může být například málo možností konektivity a fakt, že vstupní a výstupní zařízení jsou již součástí.

Jednodeskové osobní počítače. Za posledních pár let vzniklo několik zajímavých projektů malých osobních počítačů. Všechny komponenty (kromě vstupních a výstupních zařízení) jsou umístěné na jedné desce. Tyto počítače mají díky pokroku v miniaturizaci malé rozměry a jsou dostatečně výkonné. Příkladem může být jednodeskový počítač Raspberry Pi 4.

Jednodeskové mikrokontrolérové počítače. Další možností může být jiný typ jednodeskových počítačů založený na pomalejších mikrokontrolérech. Tyto řídící jednotky jsou ale většinou určené pro automatizaci a až na výjimky jsou nedostatečně výkonné pro použití v terminálu. Výhodou je stejně jako u osobních jednodeskových počítačů jejich velikost.

Vlastní řešení. Je zde i možnost vlastního řešení, ale z našich požadavků plyne, že vyrobit řídící jednotku s dostatečným výkonem by bylo v rámci této práce zcela nemožné.

Konektivita

V případě, že se rozhodneme s terminálem vzdáleně komunikovat (o čemž jsme se v rámci této práce rozhodli), je třeba zajistit nějaký druh konektivity. Nejjednodušším a nejrozšířenějším způsobem dnešní doby je samozřejmě komunikace přes internetový protokol. Není-li vyžadována přístupnost k terminálu z neomezené fyzické vzdálenosti, může být další variantou například proximitní komunikace pomocí Bluetooth nebo jiných bezdrátových či drátových technologií.

2.2.2 Software

Při realizaci softwarové části projektu je možné využít obrovské množství programovacích jazyků, technik a softwarových architektur, které by mohly vývoj usnadnit. Cílem této kapitoly však není všechny je zmínit a detailně analyzovat pro naše účely, protože jejich výhody či nevýhody se například u výběru programovacího jazyka nedají zcela objektivně rozlišit. Zaměříme se proto raději na silné stránky každé skupiny a v závislosti na nich, společně s osobní preferencí, provedeme výběr.

Programovací jazyk

Před samotnou rozvahou, jakou architekturu pro realizaci projektu použít, je třeba se rozhodnout o tom, jaký programovací jazyk chceme použít. K tomu nám mohou pomoci různé články na internetu, ve kterých jsou zkoumány výhody a nevýhody jednotlivých jazyků a jejich popularita. Vyřadíme-li programovací jazyky nižší a vyšší - neimperativní [1] kvůli jejich vysoké specializaci, složitosti a celkové nevhodnosti pro náš projekt, ve kterém není naším primárním cílem velmi vysoká výkonnost, získáme již méně rozsáhlý seznam. Nechceme-li z tohoto seznamu nyní vybírat podle naší osobní preference nebo zkušenosti, můžeme se rozhodnout podle dostupných informací o popularitě a podpoře komunity. Je samozřejmé, že nedostaneme všude stejnou odpověď, ale budeme-li čerpat informace z velkých portálů jako je například GitHub, zjistíme, že nejpopulárnějšími programovacími jazyky jsou vesměs JavaScript (a TypeScript), Python, Java a Go [2]. Přestože všechny tyto jazyky mohou být pro naše účely vhodné, jeden z nich by mohl zaujmout naší pozornost silněji díky vysoké oblíbenosti ve webových technologiích - JavaScript (a jeho

mladšího, staticky typovaného bratra TypeScript). Nic nám samozřejmě nebrání (a nejspíše je to i nevyhnutelné) použít při realizaci více programovacích jazyků.

Architektura

Rozvaha nad tím, jakou architekturu a technologie použít, se samozřejmě odvíjí a je velice závislá na vybraném programovacím jazyce. Proto nemá význam před samotným výběrem jazyka detailněji zkoumat tyto možnosti. Je vhodné si udělat pouze abstraktní plán (pro většinu programovacích jazyků společný), kterým se chceme řídit.

Při hledání na českém internetu můžeme narazit na zajímavý článek z webu itnetwork.cz nazvaný "Dependency injection a softwarové architektury" [3], díky kterému si můžeme udělat obrázek o tom, jaké jsou nejčastější techniky při psaní větších softwarových projektů. Hned v prvních částech článku zjistíme, že jde o tyto architektury:

Monolitická architektura. Celá aplikace funguje jako jedna vrstva, bez dělení na menší části. Tato architektura je většinou přezdívána "Spaghetti code" právě kvůli vysoké koncentraci různých skupin kódu. Architektura se hodí spíše pro prototypování, malé projekty a začátečníky.

Architektura rour a filtrů. Aplikace se skládá z velkého množství malých částí. Každá část se snaží provádět pouze jednu konkrétní operaci. Tyto části se následně spojují, aby vznikl ucelený program. Tato architektura je používána spíše v operačních systémech UNIX nebo v projektech, kde je třeba transformovat velké množství dat do jiné podoby.

Dvouvrstvá architektura. Tato architektura se vyznačuje tím, že není primárně určena k tomu, aby prezentovala data uživateli. Může se chovat buď jako prostředník mezi dvěma aplikacemi, nebo jako poskytovatel obsahu jiné aplikaci. Obě tyto varianty mohou být pro realizace projektu výhodné.

Tří a více vrstvá architektura. Architektura třívrstvá je v jádru totožná s architekturou dvouvrstvou, ale navíc do sebe balí prezenční vrstvu (obsah viditelný uživatelem). Vznikne tím ucelená aplikace, která dokáže data zpracovávat a formátovat uživateli bez jiných závislostí nebo prostředníků. Tato ucelenost se ale může stát i nevýhodou, protože vícevrstvé architektury zakončené prezenční vrstvou lze velice špatně kombinovat s jinými aplikacemi. Více než třívrstvé aplikace už pouze rozdělují nebo rozšiřují tři zmíněné vrstvy na menší části. Tato architektura se také jeví velice vhodně pro realizaci projektu, protože zajišťuje všechny potřebné komponenty, které vyžadujeme.

Technologie komunikace

V dalším bodě analýzy bude rozvaha o tom, jak by mohla vypadat komunikace mezi naším terminálem a jeho administrací. Protože jsme v rozvaze o dostupných hardwarových možnostech konektivity neuvedli jiné příklady

konektivity než tu přes internetový protokol, budeme dále zvažovat pouze technologie odvíjející se od tohoto protokolu. Nabízí se nám tedy technologie založené na protokolech UDP a TCP/IP, které jsou bezprostředními sousedy protokolu IP.

Protokol UDP a všechny na něm stavěné technologie nepočítají při odesílání s odezvou od příjemce. Je zde proto velká pravděpodobnost ztráty nebo nekompletnosti dat. Výhodou může ale být z toho plynoucí vyšší rychlost průtoku dat a nižší latence. Proto se tento protokol hojně využívá při hlasové komunikaci, online hrách nebo při streamování videí. Pro účely této práce je ale tento protokol spíše nevhodný, protože data, která budeme odesílat, musí být přijata ve stejném tvaru, v kterém byla odeslána, aby se obsah mohl správně zobrazovat.

Protokol TCP/IP má na rozdíl od předešlého protokolu značnou výhodu. Je u něj zaručeno, že data dorazí do cíle ve stejném tvaru a pořadí, jako byla odeslána. Je proto používán všude, kde se nehodí protokol UDP, a i v samotné implementaci adresace v síti.

Na protokolu TCP/IP je založeno i několik velice používaných a k našemu tématu relevantních protokolů jako například:

HTTP(S) – komunikační protokol primárně určený pro webové stránky

SSH – komunikační protokol pro zabezpečenou komunikaci po síti

WebSocket – komunikační protokol, který dovoluje navázat stálé oboustranné spojení mezi dvěma počítači v téměř reálném čase

Knihovny a frameworky

Ve chvíli, kdy se rozhodneme jaký programovací jazyk použít, jakým architektonickým návrhem se chceme řídit (v praxi to znamená na kolik částí aplikaci rozdělit), a jak bude probíhat komunikace mezi součástmi našeho projektu, je třeba se rozhodnout jaké konkrétní nástroje chceme využít.

Samozřejmě je možné řádně si nastudovat informace o námi zvolených technikách a realizovat vše bez pomoci již hotových nástrojů někoho jiného. To se ale v praxi jeví jako zbytečný a časově náročný krok, který by mohl vést ke stejnému nebo horšímu výsledku než při použití již hotových nástrojů.

Mnohdy účinnějším způsobem je využití některých dostupných knihoven a frameworků, které nám okolí nabízí. Díky tomu je možné věnovat se návrhu našeho nápadu a neztrácet čas a úsilí na něčem, co bylo již vytvořeno a téměř dovedeno k dokonalosti. V závislosti na rozšířenosti námi vybraného programovacího jazyka a architektury lze najít několik různých celosvětově používaných nástrojů pro usnadnění práce.

Příklady z několika nejpoužívanějších programovacích jazyků světa [2]:

- Java
  - Frameworky - Spring, Hibernate, Struts a další.
  - Nástroje pro správu závislostí - Maven, Gradle, Ant a další.

- JavaScript
  - Frameworky
    - Klientské - Angular, React, Vue a další.
    - Serverové (Node.js) - Nest, Express, Meteor a další.
  - Nástroje pro správu závislostí - npm, yarn
- Python
  - Frameworky - Django, Dash a další.
  - Nástroje pro správu závislostí - Pip, Poetry a další.

Ukládání dat

Posledním důležitým bodem je výběr technologie pro ukládání dat. Ze zadání a seznamu možných využití terminálu (2.1) plyne, že bude třeba ukládat nějaká data, která mají být zachována mezi instancemi aplikace. Využít pro ukládání dat volatilní médium, jako je například paměť RAM (ve které je po dobu chodu aplikace uložený její stav a proměnné), není vhodné, protože po zrušení procesu aplikace je obsah odstraněn.

Data je tedy třeba ukládat do nevolatilního média, jako je například pevný disk počítače. Variantou je také cloudové úložiště (úložiště dostupné po síti), ale z důvodu pevné závislosti na rychlém internetovém připojení, diskutabilní bezpečnosti a perzistenci dat tuto variantu vyřadíme. Data tedy budou vždy uložena na pevném disku v nějaké formě souboru.

V našem projektu bude třeba ukládat množství dat o uživatelích a obsahu, který se bude prezentovat na terminále. Tato data by měla být dostupná kdykoliv a okamžitě. Nejrozšířenější formou pro ukládání malých až středně velkých dat jsou databázové systémy. Na konci roku 2020 patřily mezi ty nejznámější (podle webové stránky *db-engines.com* [4]) tyto:

- Relační (tabulky a vazby mezi nimi)
  - Oracle
  - MySQL (MariaDB)
  - Microsoft SQL
  - PostgreSQL
- NoSQL (JSON dokumenty, key-value, apod.)
  - MongoDB
  - Redis (přestože ji nelze považovat za zcela perzistentní)

Kromě dat o uživatelích a obsahu je třeba také ukládat obsah samotný (nahraný uživatelem). Tento obsah se velikostně může velice lišit a může dosahovat až několika stovek MB. Databázové systémy (alespoň ty výše zmíněné) nejsou na tento typ souborů zcela určeny, a proto by nemusely být zcela vhodné. Druhou variantou je tedy neukládat tento typ dat do databázových systémů, ale přímo na úložiště.

Kapitola 3
Návrh

V této kapitole bude zpracován konkrétní návrh jedné možné implementace projektu. V první fázi bude definován seznam všech interakcí člověka s projektem. Dále bude proveden výběr praktik, technologií a metod popsaných v předchozí kapitole. Na závěr budou vyřešeny otázky a problémy, které z našeho výběru plynou. Díky tomuto zcela konkrétnímu návrhu bude možné provést ukázkovou implementaci samotného projektu a provést na něm uživatelské testování.

V následujícím textu se nachází tyto pojmy:

Backend – Program, který je jádrem projektu, jehož úkolem je zpracovávat příchozí data a produkovat nové.

Frontend – Program, který je prezenční vrstvou, jehož úkolem je být rozhraním mezi uživatelem a Backendem.

3.1 Základní rozvržení

Před zcela konkrétním návrhem a jeho obhajobou je třeba si nyní předvést, jak vypadá základní rozvržení projektu, aby bylo možné si ho dobře představit. Hlavní myšlenkou je rozdělit návrh do dvou oddělených, na sobě nezávislých a samostatně pracujících jednotek:

3.1.1 Terminál

Interaktivní tabule - fyzický hardware zobrazující obsah a dovolující interakci uživatelů s obsahem s těmito vlastnostmi:

- Je připojen k internetové (lokální, či veřejné) síti, aby byla možná vzdálená komunikace
- Hardware
  - Malého počítače
  - Zobrazovacího zařízení (Počítačový monitor)
  - Veřejně dostupného vstupního zařízení (gamepad)

- Veřejnosti skrytého vstupního zařízení (klávesnice)

- Software
  - Minimální instalace operačního systému s grafickým rozhraním
  - Program pro vzdálenou administrační správu
  - Program pro prezentaci obsahu (dále prezentován jako Frontend terminálu)

3.1.2 Portál

Systém pro správu obsahu a vzdálenou administraci rozdělený na tři části:

Frontend - Webová aplikace běžící na počítači uživatele.

Backend - Aplikace běžící na vzdáleném cloudovém serveru (mimo hardware terminálu) starající se o následující:

- Naslouchá příchozím požadavkům (od Frontendu, nebo jiných odesílatelů).
- Spravuje frontu obsahu pro terminál.

Databázový server - Server poskytující backendu ukládat perzistentně data běžící nejlépe na stejném serveru jako Backend.

3.2 Případy užití

Pro vhodný přehled o funkcích celého systému je třeba definovat, jaké interakce mohou jednotlivé skupiny uživatelů (aktéři) se systémem dělat. Interakce níže jsou rozděleny podle rolí, kterých může daný aktér nabývat. Je samozřejmé, že pokud má aktér vyšší roli než předešlý, má oprávnění k provádění jeho rolí.

Z obrázku výše je zřejmé, že jednotlivé případy jsou ještě rozděleny do dvou pomyslných kategorií podle toho, k jaké části projektu se více vztahují (terminál samotný nebo jeho administrační portál).

V následujícím bodě bude proveden podrobný rozbor všech definovaných případů užití.

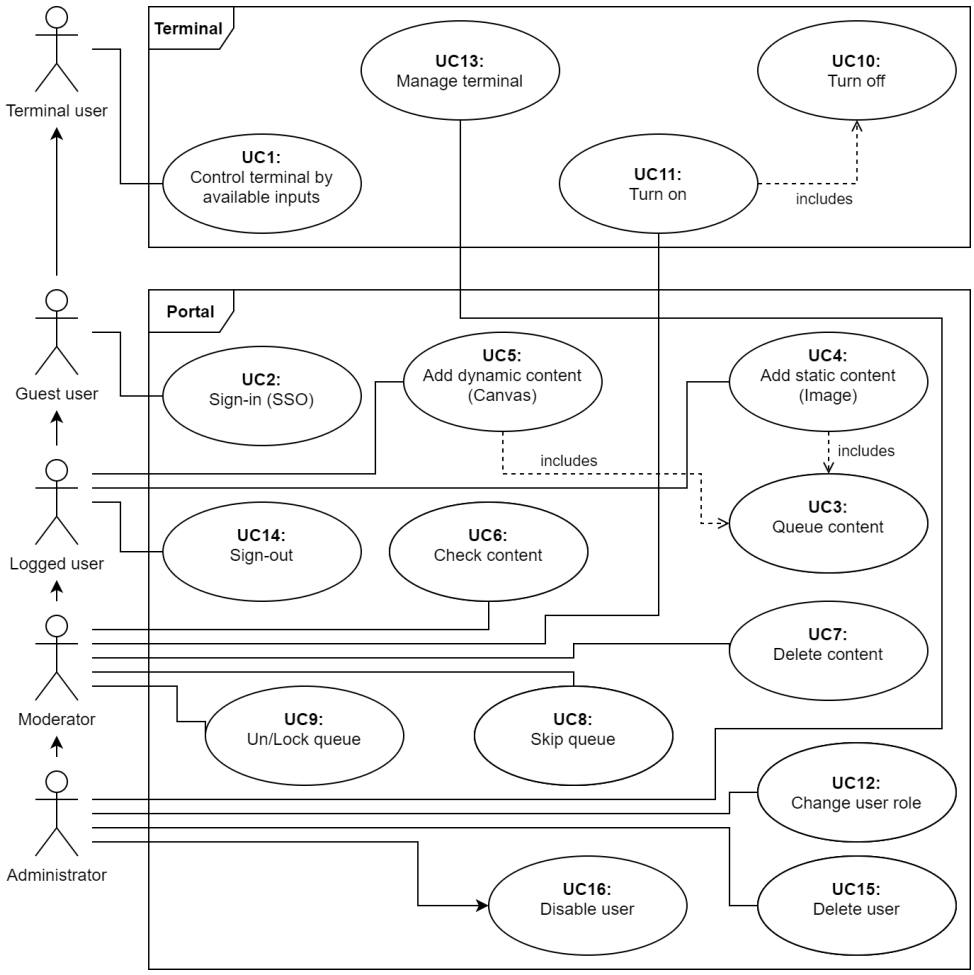
3.2.1 Aktéři

Aktéři (role) jsou seřazeni sestupně od nejnižšího oprávnění po nejvyšší oprávnění. Jak již bylo zmíněno, aktér je potomkem aktéra nad ním a dědí jeho případy užití:

Uživatel terminálu Kdokoliv, kdo má přístup k terminálu.

Případy užití:

- UC1 - Ovládat dostupnými vstupy



Obrázek 3.1: Grafické znázornění případů užití UML diagramem

Nepřihlášený uživatel Kdokoliv, kdo otevře administraci terminálu a není přihlášený.

Případy užití:

- UC2 - Přihlásit se přes SSO
- UC3 - Odhlásit se

Přihlášený uživatel Jakýkoliv uživatel přihlášený do administrace terminálu.

Případy užití:

- UC4 - Přidat statický obsah (obsahuje UC3)
- UC5 - Přidat dynamický obsah (obsahuje UC3)

Moderátor Přihlášený uživatel s rolí moderátora. Může spravovat obsah.

Případy užití:

- UC6 - Zkontrolovat obsah
- UC7 - Smazat obsah
- UC8 - Přeskakovat frontu
- UC9 - (De)aktivovat frontu
- UC10 - Zapnout terminál (obsahuje UC10)

Administrátor Přihlášený uživatel s rolí administrátora. Práva na všechny akce.

Případy užití:

- UC12 - Nastavit roli
- UC13 - Ovládat terminál
- UC15 - Smazat uživatele
- UC16 - Deaktivovat uživatele

3.2.2 Akce

UC1 Ovládat dostupnými vstupy

- Aktér ovládá terminál vstupy (tlačítky), které má k dispozici před sebou na panelu.
- Hlavní scénář:
  1. Aktér podle obsahu na obrazovce naviguje pomocí vstupních tlačítek.
  2. Systém reaguje na uživatelské vstupy

UC2 Přihlásit se (přes OAuth)

- Aktér se přihlásí do systému pomocí externího přihlášení službou OAuth2.
- Hlavní scénář:
  1. Aktér klikne na tlačítko „Přihlásit se“
  2. Aktér bude přesměrován na stránku, kde provede přihlášení svými přihlašovacími údaji
  3. Aktér bude přesměrován zpět do systému
  4. Systém zpracuje OAuth přihlášení
  5. Systém přihlásí uživatele

UC3 Zařadit se do fronty

- Pod případ užití, který zařadí vložený obsah do fronty, aby se mohl zobrazit na obrazovce terminálu.
- Hlavní scénář:
  1. *Předpoklad: UC4, nebo UC5 + UC6*

2. Systém zařadí obsah do fronty
3. Systém pošle obsah do terminálu (v případě, že je na obsah řada, jinak vyčká na správný čas)

UC4 Přidat statický obsah

- Aktér nahraje obrázek nebo video do systému, aby se v budoucnu zobrazil na terminálu.
- Hlavní scénář:
  1. Aktér klikne na tlačítko „Nahrát statický obsah“
  2. Aktér vyplní formulář
  3. Aktér přidá statický obsah k nahrání (obrázek, nebo video)
  4. Aktér odešle formulář
  5. Systém obsah uloží a čeká na kontrolu obsahu (*UC6*)

UC5 Přidat dynamický obsah

- Aktér vytvoří a nahraje dynamický obsah (webová stránka) do systému, aby se v budoucnu zobrazil na terminálu.
- Hlavní scénář:
  1. Aktér klikne na tlačítko „Nahrát dynamický obsah“
  2. Aktér vyplní formulář
  3. Aktér vytvoří v editoru, nebo nahraje dynamický obsah
  4. Aktér odešle formulář
  5. Systém obsah uloží a čeká na kontrolu obsahu (*UC6*)

UC6 Zkontrolovat obsah

- Aktér zkontroluje obsah, který je ve frontě, a povolí jeho zobrazení na terminálu.
- Hlavní scénář:
  1. *Předpoklad: UC4, nebo UC5*
  2. Aktér klikne na tlačítko „Fronta“
  3. Aktér klikne na řádek představující nezkontrolovaný obsah
  4. Aktér zkontroluje obsah
  5. Aktér povolí (tlačítkem) obsah
  6. Aktér potvrdí svou akci
  7. Systém provede změnu
  8. *Systém pokračuje s UC3*

UC7 Smazat obsah

- Aktér má možnost odstranit obsah z fronty, a to i pokud je zrovna aktivní (zobrazený) na terminálu.
- Hlavní scénář:

1. Aktér klikne na tlačítko „Fronta“
2. Aktér klikne na řádek představující obsah
3. Aktér klikne na tlačítko „Smazat obsah“
4. Aktér potvrdí svou akci
5. Systém odstraní obsah z fronty
6. Systém, v případě že obsah je na terminálu, odstraní obsah z terminálu

UC8 Přeskočit frontu

- Aktér má možnost přeskočit obsah ve frontě.
- Hlavní scénář:
  1. Aktér klikne na tlačítko „Správa terminálu“
  2. Aktér klikne na tlačítko „Posunout frontu“
  3. Aktér potvrdí svou akci
  4. Systém se posune na další obsah ve frontě

UC9 (De)aktivovat frontu

- Aktér může aktivovat/deaktivovat funkci fronty, a tím zamknout/odemknout obsah, který je v tuto chvíli na terminálu.
- Hlavní scénář:
  1. Aktér klikne na tlačítko „Správa terminálu“
  2. Aktér klikne na tlačítko „Aktivovat/Deaktivovat frontu“
  3. Systém aktivuje/deaktivuje frontu

UC10 Vypnout terminál

- Aktér tímto „vzdáleným“ příkazem vypne terminál *(tzn. na terminálu nebude žádný obsah, fronta se deaktivuje a vstupy nebudou reagovat).*
- Hlavní scénář:
  1. *Požadavek: Terminál musí být v zapnutém stavu*
  2. Aktér klikne na tlačítko „Správa terminálu“
  3. Aktér klikne na tlačítko „Vypnout terminál“
  4. Systém pošle terminálu příkaz k „vypnutí“ terminálu

UC11 Zapnout terminál

- Aktér tímto „vzdáleným“ příkazem zapne terminál *(tzn. pokud byl terminál vypnutý, bude znovu zobrazovat obsah a reagovat na vstupy).*
- Hlavní scénář:
  1. *Požadavek: Terminál musí být ve vypnutém stavu*
  2. Aktér klikne na tlačítko „Správa terminálu“

3. Aktér klikne na tlačítko „Zapnout terminál“
4. Systém pošle terminálu příkaz k „zapnutí“ terminálu

UC12 Nastavit roli

- Aktér může jinému aktérovi nastavit jakoukoliv stejnou nebo nižší roli, než má on sám.
- Hlavní scénář:
  1. Aktér klikne na tlačítko „Uživatelé“
  2. Aktér klikne na řádek představující uživatele
  3. *Požadavek: Aktér musí mít oprávnění ke změně na konkrétní roli. To závisí na jeho osobní roli v systému.*
  4. Aktér vybere uživateli jinou roli
  5. Aktér klikne na „Uložit“
  6. Systém provede změnu role

UC13 Ovládat terminál

- Aktér má vzdálený (nebo lokální) přístup do Linuxové konzole našeho terminálu.
- Hlavní scénář:
  1. Aktér se připojí k SSH serveru terminálu
  2. Aktér zadá své přihlašovací údaje
  3. Systém přihlásí aktéra do konzole

UC14 Odhlásit se

- Aktér se odhlásí ze systému.
- Hlavní scénář:
  1. *Požadavek: Aktér je přihlášený do systému (UC2)*
  2. Aktér klikne na tlačítko „Odhlásit se“
  3. Systém odhlásí uživatele
  4. Aktér bude přesměrován na úvodní stránku

UC15 Smazat uživatele

- Aktér může smazat uživatelský účet ze systému *(v případě přihlašování přes OAuth, jakožto jediným způsobem, nelze ale zamezit v opětovné registraci uživatele).*
- Hlavní scénář:
  1. Aktér klikne na tlačítko „Uživatelé“
  2. Aktér klikne na řádek představující uživatele
  3. Omezení: Aktér nemůže odstranit svůj účet
  4. Aktér klikne na „Smazat“
  5. Aktér svou akci potvrdí

6. Systém provede odstranění účtu

UC16 Deaktivovat uživatele

- Aktér může deaktivovat uživatelský účet v systému.
- Hlavní scénář:
  1. Aktér klikne na tlačítko „Uživatelé“
  2. Aktér klikne na řádek představující uživatele
  3. Omezení: Aktér nemůže deaktivovat svůj účet
  4. Aktér klikne na „Deaktivovat účet“
  5. Aktér svou akci potvrdí
  6. Systém provede deaktivaci účtu a neumožní majiteli účtu provést přihlášení

3.3 Hardware terminálu

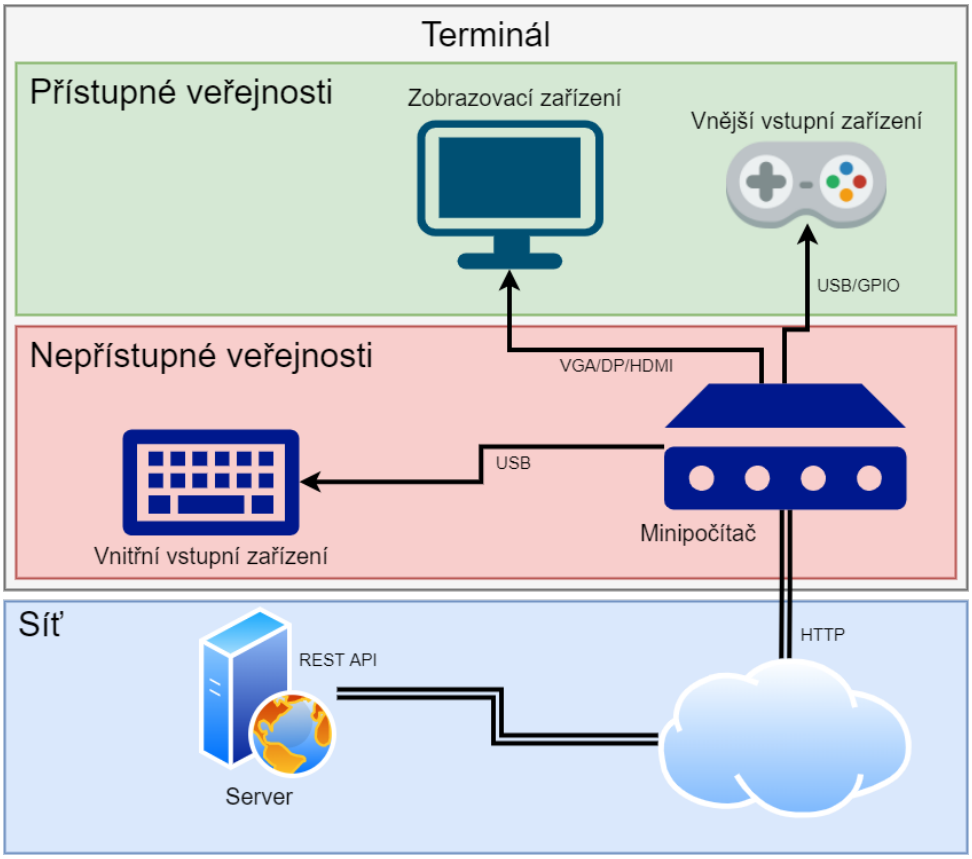
Součástí projektu je návrh a implementace hardwarové stránky terminálu. Jak bylo již zmíněno, terminál se skládá ze čtyř hlavních komponent. Tyto komponenty dohromady tvoří ucelený systém (Obrázek 3.2), který by nemělo být složité umístit do pouzdra podobné velikosti a rozložení, jako je například kiosek, který se v posledních letech stal součástí většiny prodejen McDonald's (viz Obrázek 3.3).

Na rozdíl od tohoto kiosku nebude ale v tomto návrhu terminálu popsána dotyková plocha nýbrž vestavěný gamepad. Přestože se dotyková plocha může jevit jako výhodnější, cílem této práce není navrhnout a implementovat finální verzi hardwarové stránky terminálu, ale pouze funkční demo, na kterém lze představit a otestovat funkcionalitu implementace systému. Níže bude popsán hardware vybraný pro naše funkční demo.

3.3.1 Řídící jednotka

Jako řídící jednotka pro terminál byl vybrán celosvětově oblíbený jednodeskový počítač Raspberry Pi v nejnovější čtvrté verzi. Jedná se o kompletní počítač (bez periferií) se čtyřjádrovým procesorem ARM a 4GB RAM (v nejvyšší variantě). S jeho možností duálního video výstupu až ve 4k rozlišení, velkými možnostmi konektivity (WiFi, Ethernet, Bluetooth, USB, atd.) a rozměrem přibližně velikosti kreditní karty (tloušťka asi 2 cm včetně chladiče), se jedná o velice vhodný výběr pro tento projekt.

Tento malý počítač, kromě již zmíněných výhod, obsahuje i několik GPIO pinů, díky kterým se rozšiřují možnosti při výběru hardwaru vstupních a výstupních zařízení, protože není nutné omezovat se pouze na komerčně vyráběná řešení.



Obrázek 3.2: Diagram zobrazující uspořádání hardware terminálu

Obrázek 3.3: Samoobslužný kiosek v prodejnách McDonalds (zdroj [5])

3.3.2 Zobrazovací zařízení

V případě zobrazovacího zařízení jsou naše možnosti výběru o něco rozsáhlejší, ale rychlou dedukcí nad výhodami a nevýhodami jednotlivých řešení lze dojít k závěru, že většina variant nejsou buďto finančně výhodné (speciální panely pro použití v kioscích), nebo jsou časově a znalostně náročné (vlastní řešení displeje jako je například LED matrix).

V případě zobrazovacího zařízení jsou naše možnosti výběru o něco rozsáhlejší, ale rychlou dedukcí nad výhodami a nevýhodami jednotlivých řešení lze dojít k závěru, že většina variant nejsou buď finančně výhodné (speciální panely pro použití v kioscích), nebo jsou časově a znalostně náročné (vlastní řešení displeje jako je například LED matrix).

Byla tedy vybrána jednoduchá cesta a to v podobě použití standardního počítačového monitoru či televize se vstupem standardu HDMI. Výběr konkrétního modelu velice závisí na finálním návrhu produkční verze terminálu, a proto v tomto dokumentu nebude zmíněn[1].

Lze ale zmínit několik doporučení, která by mohla být relevantní při výběru takovéto zobrazovací jednotky. Při výběru monitoru nebo televize pro použití v terminálu, který bude v provozu nonstop, je třeba dbát na správnou volbu. Vhodný je model s panelem, který nebude trpět na takzvaný burn-in. Touto vadou v podobě vypalování některých barev jednotlivých pixelů po nějakém čase trpí například panely technologie OLED, a proto nejsou vhodné. Vhodně se také nejeví modely bez systému VESA mount, který nám rozšiřuje možnosti instalace zařízení, a modely s velkou snímkovou frekvencí, které mají vyšší nároky na chlazení (herní monitory a televize). Panel by také měl mít minimální rozlišení 1280x720px, i když se tento požadavek v dnešní době zdá poněkud zbytečný (v prodejnách již nenajdete nový monitor či televizi s nižším rozlišením).

3.3.3 Vstupní zařízení

Jak bylo již zmíněno, při návrhu hardware terminálu se počítá s tím, že se nejedná o finální produkční variantu terminálu. Proto ani při výběru vstupního zařízení nemá význam jít cestou vlastního řešení nebo nějakých specializovaných řešení.

Jako vstupní zařízení terminálu byl proto vybrán ovladač herní konzole Xbox One S, který obsahuje dva analogové dvouosé joysticky a několik dalších jednoduchých tlačítek. Dalším důvodem výběru byla kompatibilita s počítačem Raspberry Pi a s většinou webových prohlížečů pomocí Gamepad API (více o tomto tématu později). Nevýhodou při produkčním použití může být ale nízká odolnost a chybějící možnost bezpečného uchycení k zamezení krádeže. Proto není pro produkční verzi terminálu tento ovladač zcela vhodný a bude třeba se uchýlit k vlastnímu řešení v podobě panelového gamepadu (jako tomu bylo u starých arkádových her).

[1] Při implementaci návrhu a testování byla použita standardní 30” televize Philips (rok výroby 2014) s rozlišením sníženým z 1080p na rozlišení 720p.

3.4 Software

V této části bude zpracován návrh samotného softwaru celého systému. Jedná se o nejdůležitější část, protože na jejím výsledku závisí směr, kterým se implementace povede. Vhodným výběrem programovacího jazyka, architektury, technologií a způsobu komunikace jednotlivých komponent lze dosáhnout efektivního a čistého kódu. Nevhodný výběr může celý proces implementace velice protáhnout, omezit funkcionalitu nebo dokonce zamezit dokončení. Při výběru bude tedy třeba zvážit všechny naskytnuté možnosti a porovnat jejich výhodnost pro tento projekt.

3.4.1 Programovací jazyky

Přestože požadavky tohoto projektu značně neomezoval žádný z deseti nejpoužívanějších programovacích jazyků [2], byl vybrán programovací jazyk TypeScript, který je nadstavbou nad programovacím[2] jazykem JavaScript.

Tento jazyk byl vybrán z několika důvodů:

1. Osobní preference z důvodu několikaletých zkušeností s jazykem JavaScript[3].
2. TypeScript rozšiřuje jazyk JavaScript o statické typování, což nese výhody v přehlednosti a funkčnosti a zdrojového kódu.
3. Lze ho použít jako hlavní jazyk pro frontend a backend.
4. Má menší nároky na výkon.

JavaScript je použitelný nejen pro psaní klientského prohlížečového kódu (pro který původně vznikl), ale také pro psaní serverových aplikací. Tyto programy se pak spouštějí v systémovém prostředí Node.js, které na rozdíl od webových prohlížečů podporuje i například práci se souborovým systémem. Toto prostředí obsahuje velice optimalizovaný JavaScriptový engine V8, který dnes používá většina webových prohlížečů.

Přestože JavaScript je velice univerzální jazyk, v projektu je třeba ještě několika jiných pomocných jazyků. Těmi jsou:

HTML – Značkovací jazyk pro tvorbu webových stránek

CSS – Standard pro úpravu stylů HTML stránek

Bash – Skriptovací jazyk pro operační systémy GNU

SQL – Dotazovací jazyk pro databázové systémy

[2] JavaScript je sice stále popsán (například v otevřené encyklopedii Wikipedia) jako "skriptovací", ale díky jeho rychlému vývoji si dnes již dovolím nazývat ho jazykem programovacím.

[3] V tomto dokumentu nebudeme zcela rozlišovat tyto dva jazyky, protože jejich syntaxe si je velice podobná a TypeScript se musí nejdříve do JavaScriptu převést, aby byl spustitelný. To znamená, že kde není žádný rozdíl mezi jazyky JavaScript a TypeScript, budeme používat název JavaScript.

3.4.2 Správce závislostí

V případě vývoje v JavaScriptu se nám nabízí pouze jeden druh správce závislostí, a tím je služba Node Package Manager (zkráceně NPM). Tento správce balíčků je jednoduchým systémem pro organizaci závislostí projektu. Jeho primární funkcí je stahovat automaticky závislosti z hlavního repozitáře npmjs.com a poskytnout nám nástroj pro organizaci spustitelných skriptů (sestavování, spouštění, testy, apod.).

Díky tomuto správci se nám otevřou dveře k více než milionu open-source knihoven a projektů, díky kterým není zapotřebí vyvíjet veškerý kód našeho projektu od nuly. Velké množství těchto balíků klade i důraz na dodržování architektonických vzorů a praktik (např. frameworky).

V posledních letech vzniká nový projekt nazvaný Yarn, který je rozšířením stávajících nástrojů NPM. Jeho výhodou je podpora takzvaných Workspaces (více v části o implementaci) a vyšší rychlost při stahování a kompilaci balíků. V tomto projektu je použití této nadstavby výhodné.

3.4.3 Klientské aplikace

V tomto projektu je třeba dvou klientských aplikací s odlišným využitím. Obě klientské aplikace potřebují komunikovat s hlavním serverem pomocí internetového protokolu, a proto se jako nejvýhodnější typ jeví klientská webová aplikace. Tyto klientské webové aplikace (nazývané též Single Page Application - SPA) jsou samostatnou aplikací běžící zcela na klientském zařízení ve webovém prohlížeči.

Největší výhodou tohoto typů klientské aplikace je fakt, že po prvním stažení souborů aplikace ze serveru není třeba již obnovovat webovou stránku ani stahovat další data. Pokud je zapotřebí komunikovat se serverem, děje se tak asynchronně na pozadí.

Pro všechny klientské aplikace v tomto projektu je proto použit známý framework React. Tato knihovna je velice rozšířeným nástrojem pro tvorbu uživatelského rozhraní a podporuje metodu tvorby jednostránkových klientských aplikací - SPA. Projekt je vyvíjen a využíván společností Facebook a za posledních několik let se stal velice zralým. Jeho použití v projektu velice urychlí vývoj a zaručí dodržování správných postupů při psaní zdrojového kódu.

Terminál

Z výhod popsaných v úvodu této podkategorie plyne i možnost provozování aplikace bez připojení k internetu (offline), pokud máme lokální webový server, který dokáže prohlížeči předat soubory aplikace. Přesně tato výhoda bude využita v návrhu aplikace pro terminál.

Vzhledem k tomu, že terminál nevyžaduje žádné připojení k databázi obsahu (protože obsah mu je zasílán z hlavního serveru), není zapotřebí ani žádných prvků, které by měly být veřejnosti skryté. Software terminálu

proto může být klientská aplikace založená na frameworku React s možností obousměrné komunikace se serverem.

Tuto obousměrnou komunikaci se serverem realizujeme pomocí takzvaných WebSocketů. Tento internetový protokol zajistí naší aplikaci komunikaci se serverem v reálném čase a s téměř instantní odezvou. Klientská aplikace se pouze připojí k serveru, autorizuje se a komunikační kanál je navázán. V případě dočasného výpadku internetového připojení nebo restartu aplikace terminálu se spojení opět naváže.

Protože terminál obsahuje i vstupní zařízení, je třeba tyto vstupní data nějak zpracovávat a předávat aplikaci. K tomu nám poslouží rozhraní Gamepad API, které je součástí většiny moderních prohlížečů. Díky tomuto rozhraní máme přístup k událostem, které signalizují změnu stavu tlačítek, nebo joysticků na připojeném ovladači. Aplikace tedy může na tyto akce prováděné uživatelem reagovat v reálném čase.

Nakonec je třeba dodat, že popsaná klientská aplikace musí běžet v prohlížeči s podporou výše zmíněných technologií (Websocket a Gamepad API) a také režimu kiosku. Tento režim zaručí, že uživatel bez přístupu k vnitřnímu vstupnímu zařízení nebude mít možnost okno prohlížeče shodit a ovládat systém samotný. Detailnější popis nastavení operačního systému pro provoz terminálu bude popsán v části implementace.

Portál

Pro uživatelské rozhraní portálu (administrace terminálu) bude také použita metoda SPA. Na rozdíl od terminálu, kde bude prostředí nakonfigurováno podle jednoho operačního systému a hardwaru, u veřejného portálu, ke kterému má přístup veřejnost, je třeba definovat jeden důležitý požadavek.

Aplikace musí být plně funkční alespoň na prvních třech nejpoužívanějších webových prohlížečích. Podle webové stránky *netmarketshare.com* [6] to jsou prohlížeče:

- Chrome (~65.4%)
- Safari (~17.9%)
- Firefox (~3.3%)

U každého prohlížeče je třeba ještě zajistit funkčnost pro aktuální a alespoň jednu předešlou verzi:

- Chrome verze 86.0+
- Safari verze 13.0+
- Firefox verze 83+

(Všechna výše uvedená data se vztahují k prosinci 2020)

3.4.4 Databázový a souborový systém

V projektu je třeba ukládat pouze tři skupiny dat. Těmi jsou:

- Uživatelská data
- Metadata nahraného obsahu a fronta
- Nahraný obsah samotný

Tyto data je třeba ukládat ve strukturované a indexovatelné formě na pevném disku, aby nedošlo ke ztrátě mezi instancemi. Proto využijeme známý relační databázový systém MariaDB. Do této databáze budeme ukládat všechny údaje o uživatelích, frontě a obsahu, který byl na server nahrán.

Přestože je možné do relačního databázového systému ukládat i velké soubory, budeme do něj ukládat pouze metadata nahraného obsahu (index, velikost, datum nahrání, apod.). Nahraný obsah samotný (obrázky, videa, programy, apod.) se uloží do souborového systému a v databázi se vytvoří vazba, která jednoznačně ukáže na tento obsah. Zamezíme tím případným budoucím problémům s výkonem databáze.

Protože data v databázi mají být strukturovaná, je třeba provést návrh této struktury. V relační databázi se jedná o seznam entit (tabulek) s nějakými vlastnostmi. Tyto entity jsou dále mezi sebou propojeny vazbami.

3.4.5 Backend

Na závěr návrhu projektu budou definovány všechny důležité prvky aplikačního serveru běžícího mimo hardware terminálu. Účelem tohoto serveru bude řídit správu obsahu terminálu a příchozích uživatelských dotazů. V následujících podkapitolách bude představena architektura a jednotlivé části tohoto backendu.

Komunikace s portálem

Nejjednodušší a také nejpoužívanější způsob komunikace mezi klientskou aplikací (SPA) a webovým serverem je architektura rozhraní REST. Tato architektura je standardem pro komunikaci ve webovém prostředí a definuje, jak lze jednoduše vytvořit, číst, editovat a smazat data ze serveru pomocí HTTP dotazů [7]. Pro REST API je nejčastější standardizovaný formát dokumentů JSON.

Definicí takzvaných CRUD (Create-Read-Update-Delete) operací na našem webovém serveru zjednodušíme a zpřehledníme komunikaci s portálem. Navíc můžeme vytvořit přehlednou dokumentaci (například standardu OpenAPI), díky které bude mít kdokoliv přehled o funkcích našeho API a může s ním provádět operace s dobrou predikcí chování a odpovědí.

Komunikace s terminálem

Při komunikaci s terminálem by bylo také možné využít webový server a HTTP dotazy. Výhodnější metodou komunikace ale bude, jak již bylo zmíněno v návrhu klientské aplikace terminálu, použití WebSocketů. Bohužel je třeba, aby náš backend naslouchal na dalším portu pro tento protokol, ale to by v případě vlastního cloudového serveru neměl být problém.

Výhoda WebSocketu je v obousměrné komunikaci v reálném čase a garanci pořadí příjmu zpráv, tak jak byly odeslané [8]. Navíc k jednomu serveru může být ve stejnou chvíli připojeno více klientů, a tím se neomezujeme pouze na jeden terminál.

Architektura

Vzhledem k tomu, že obě klientské aplikace jsou vlastními programy fungujícími na klientské straně, není zapotřebí, aby náš aplikační server obsahoval prezenční vrstvu. Vyřadíme-li začátečnickou a špatně škálovatelnou monolitickou architekturu (spaghetti-code), značně si ulehčíme budoucí trápení. Hlavní architekturou pro aplikační server bude tedy vícevrstvá architektura.

V ekosystému programovacího jazyka JavaScript se v době vzniku tohoto dokumentu nevyskytuje velké množství ucelených frameworků pro tvorbu serverových aplikací v prostředí Node.js. Ucelenými frameworky myslíme ty, které jsou schopny nařídit striktní design pattern a architekturu.

V tomto malém seznamu ucelených frameworků se ale vyskytuje jeden, který je speciální svým přístupem. Jedná se o framework NestJS. Tento framework je kombinací několika jiných menších frameworků (express.js, socket.io, dotenv, typeorm, class-validator, atd.) a několika architektonických patternů, snaží se tak vytvořit vlastní architektonický vzor. Tento vzor se dá nazvat jako Module-Service-Controller(-View) a byl inspirovaný klientským frameworkem AngularJS.

Díky tomuto frameworku je možné okamžitě pracovat na vývoji aplikačního serveru a není nutné vytvářet velké množství boilerplate kódu (kostry). V dokumentaci NestJS [9] je popsáno vše potřebné pro implementaci všech komponent našeho aplikačního serveru (Webový server s REST, Websocket server, Databázový model, atd.).

Přihlašování přes OAuth

Jako poslední je třeba zmínit, jak by mělo vypadat přihlašování uživatelů portálu do systému. Systém přihlašování pomocí protokolu OAuth 2 dovoluje využít třetí stranu jako přihlašovací autoritu a přidělovat jí přístup API službám [10]. Díky tomu není třeba v projektu řešit vlastní registrace uživatelů a jejich udržování v databázi. To se jeví pro tento projekt velice výhodně, protože ČVUT nabízí hned několik takových přihlašovacích autorit.

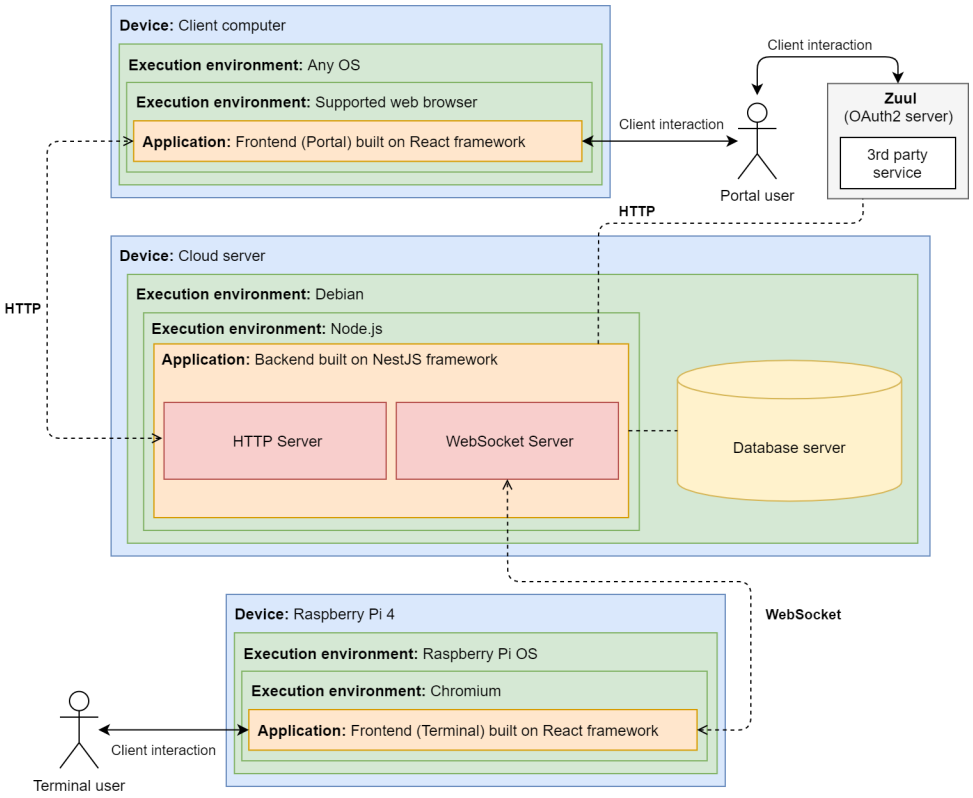
Jednou z nich je autorizační server Zuul, který vznikl na fakultě Informačních technologií ČVUT [11]. Tento autorizační server povoluje studentům

ČVUT přihlašování školními údaji a přístup k některým APi rozhraním školy (např. KosAPI, Usermap, apod.).

Využitím tohoto autorizačního serveru bude zaručeno, že všichni uživatelé portálu budou nějakým způsobem spjatí s ČVUT. Zároveň budeme moci získat základní data, která by jinak uživatel musel při registraci vyplňovat sám. Široké veřejnosti (mimo ČVUT) nebude umožněn přístup k nahrávání a správě obsahu terminálu.

3.5 Souhrn

V této části dokumentu byl proveden kompletní návrh celého projektu, včetně detailního popisu konkrétních softwarových technologií a praktik, které je vhodné využít. Před následující části, ve které bude popsána implementace tohoto návrhu, bude proveden souhrn této části pomocí ilustrace (Obrázek 3.4) znázorňující souhrn návrhu projektu včetně doporučených exekučních prostředí pro backend.



Obrázek 3.4: Souhrn návrhu celého projektu

Část II

Praktická část

Kapitola 4
Implementace

V této kapitole bude popsána struktura implementovaného řešení včetně toho, jak byly vyřešeny problémy, které se objevily během vývoje. Zdrojový kód, který je v této kapitole popisován, a návod na jeho instalaci je přiložen v příloze A tohoto dokumentu.

4.1 Obecný přehled

Před popisem jednotlivých částí projektu bude popsána obecná struktura a výčet použitých technologií.

4.1.1 Charakteristika implementace

Cílem této implementace bylo vytvořit zcela funkční řešení terminálu a jeho administrace. Přestože výsledná implementace koresponduje s návrhem a je zcela funkční, není připravena pro produkční chod. Důvodem je nedokončenost některých funkcionalit a nízká úroveň otestovanosti kódu. Je také nutné vyřešení otázky týkající se povolení chodu projektu od vedení školy a finální konstrukce terminálu, včetně jeho umístění ve škole.

4.1.2 Použité technologie

Nemá význam poukazovat na všechny technologie, knihovny a jiné závislosti, které byly během implementace využity, a také dělit tento výpis podle klientů (terminálu a portálu) a serveru. Každá část projektu využívá buď všechny, nebo pouze část níže vypsaných technologií.

Následující výpis není kompletní a obsahuje pouze ty nejzajímavější a nejdůležitější technologie:

REST – *„Architektonický styl pro poskytování standardů mezi počítačovými systémy na webu, což usnadňuje vzájemnou komunikaci systémů.“* [7] (přeloženo z Angličtiny)

Node.js – Vývojová platforma pro spouštění JavaScriptového kódu na straně serveru. [12]

TypeScript – Jazyk rozšiřující JavaScript o statické typování. [13]

React – *"JavaScriptová knihovna pro vytváření reaktivních uživatelských rozhraní"* [14] (přeloženo z Angličtiny)

Redux – *"Předvídatelný stavový kontejner pro JavaScriptové aplikace"* [15] (přeloženo z Angličtiny)

socket.io(-client) – Implementace WebSocket serveru (klienta) pro obousměrnou komunikaci v reálném čase. [16]

NPM – Největší registr knihoven a jiného zdrojového kódu pro JavaScript s více než milionem open-source balíčků. [17]

Yarn – Nadstavba NPM zrychlující správu závislostí s podporou Workspaces [18]

Nest.js – Framework pro vytváření škálovatelných (nejen) webových aplikací v Node.js, podporující vývojáře v psaní čistého a přehledného kódu. [9]

Bootstrap – Populární HTML, CSS a JS knihovna stylů

SASS – Preprocesorový skriptovací jazyk ulehčující psaní stylů

FontAwesome – Knihovna vektorových ikon

Axios – Implementace HTTP klienta

Yup/Joi – Validátor JSON schémat

class-validator/class-transformer – Nástroj pro validaci/transformaci Javascriptových tříd

TypeORM – Implementace techniky ORM [19] v TypeScriptu

Swagger – Nástroj pro standardizaci a zjednodušení dokumentace API

Passport – Framework pro ulehčení autentifikace

4.1.3 Obecná struktura zdrojového kódu

Všechny části projektu byly implementovány s použitím programovacího jazyka TypeScript. Dalšími minoritními jazyky použitými při implementaci jsou HTML, (S)CSS a Linux Bash. O těchto jazycích ale není třeba detailněji hovořit, protože jejich zastoupení v projektu nehraje žádnou významnou roli. Níže bude detailněji představen již zmíněný majoritní jazyk TypeScript a struktura samotného projektu.

TypeScript

Jazyk TypeScript se snaží přiblížit JavaScript ke svému striktně typovanému "otci z druhého kolene" - Javě. Použitím nové verze JavaScriptu nazvané ES6, která přináší mimo jiné novou syntaxi tříd obohacenou v jazyce TypeScript o statické typování, vznikl jazyk, který se již blíží spíše plnohodnotnému programovacímu jazyku než jazyku skriptovacímu. Také díky jeho univerzálnosti přes klientské i serverové prostředí byl vybrán pro tento projekt, protože usnadňuje strukturování kódu a sdílení kódu mezi těmito prostředími.

Struktura

Zdrojový kód je díky využití principu monorepozitáře [20], který je implementován pomocí Yarn Workspaces, strukturován do jednoho velkého repozitáře, který obsahuje všechny části projektu. Tyto níže vypsané balíčky (packages) dohromady tvoří kompletní kódovou základnu projektu:

@terminal/frontend-terminal. Obsahuje veškerý zdrojový kód (klientské) aplikace terminálu. Dále obsahuje nástroje pro sestavení produkční verze programu (build), která se nahrává na terminál samotný.

@terminal/frontend-portal. Obsahuje veškerý zdrojový kód klientské aplikace portálu (administračního rozhraní). Dále obsahuje nástroje pro sestavení produkční verze programu (build), která je klientům předávána aplikačním serverem.

@terminal/backend. Obsahuje veškerý zdrojový kód serverové aplikace. Dále obsahuje nástroje pro sestavení produkční verze programu (build), která je nahrána a provozována na cloudovém aplikačním řešení.

@terminal/shared. Obsahuje zdrojový kód, který je sdílený mezi několika předchozími balíčky. Dovoluje znovupoužití kódu bez jeho duplikace.

Přestože balíky sdílí společný adresář *node_modules*, který obsahuje všechny závislosti, fungují jako oddělené a na sobě nezávislé projekty. Každý balík tedy potřebuje vlastní manifest *package.json* obsahující informace o projektu (verze, autor, skripty, apod.) a seznam závislostí, které projekt vyžaduje[1].

Díky této struktuře projektu byla usnadněna práce se závislostmi jednotlivých částí. Celý projekt je možné spravovat z jednoho Git repozitáře a taktéž je možné sdílet kód mezi jednotlivými částmi, čímž snižujeme množství duplikovaného kódu.

4.1.4 Sestavování kódu

Protože je celý projekt napsaný v jazyce TypeScript, engine prohlížeče ani Node.js prostředí nedokáží tento kód zpracovávat. Je proto třeba provádět

[1] Soubor package.json lze přirovnat k souboru pom.xml v nástroji pro správu projektů Java - Maven.

takzvaný transcribing[2] do JavaScriptu. Tímto převodem dojde k odstranění veškerého TypeScriptu specifického kódu, a tím zůstane čistý a spustitelný JavaScript. Sestavování je třeba provádět i při vývoji. Při sestavování kódu v našem projektu se ještě kromě transcribingu provádějí jiné akce, které jsou závislé na cílovém prostředí.

Klientské prostředí

Veškerý zdrojový kód klientských aplikací (portál a terminál) ještě prochází procesem, který řídí knihovna webpack a babel. Konfiguraci těchto knihoven v naší aplikaci řídí nástroj *create-react-app*, díky kterému se nemusíme starat o konfiguraci. Vše je nastaveno a připraveno k sestavení už od vývojářů. V případě, že chceme provést vlastní konfiguraci, lze provést takzvanou eject operaci, která nám dá volnost konfigurace, ale připraví nás o možnost držet se vývojáři definovaných standardů.

Tento proces, který se spouští při sestavování, obsahuje kroky, které zaručí, že výsledný zdrojový kód nebude obsahovat žádné externí závislosti a bude spustitelný ve webovém prohlížeči:

- Převede všechen prohlížeči nepodporovaný JS kód (babel)
- Zpracuje všechny závislosti a přidá je do kódu
- Provede kompilaci SCSS do CSS
- Provede minifikaci JS a CSS
- Provede zabalení veškerého JS a CSS do balíků (bundles)
- Assety které jsme importovali v kódu, přejmenuje a přesune tam, kde budou přístupné.

Serverové prostředí

V případě serverového kódu (backend) je proveden už pouze převod nepodporovaného JS kódu a minifikace. V případě naší implementace jsou tyto procesy také z části řízeny nástroji frameworku NestJS.

4.2 Program terminálu

Tento program zajišťuje zobrazování obsahu, který přijal od serverové aplikace, svému okolí. Dále zajišťuje, aby mohl uživatel interagovat pomocí vstupního zařízení s dynamickým obsahem.

[2] V tomto případě jde o princip podobný kompilaci, ale výsledným kódem není binární spustitelný kód (jako je tomu např. v Javě), nýbrž kód v jiném jazyce (v tomto případě JavaScript).

4.2.1 Obecná struktura

Přestože je celý program terminálu psaný stylem klientské aplikace, z obecného pohledu se o klientskou aplikaci nejedná. Aplikace není spuštěna na zařízení uživatele a uživatel nemá přístup ke zdrojovému kódu této aplikace. S tím je při implementaci zabezpečení terminálu počítáno.

Program je po sestavení spustitelný ve webovém prohlížeči. V této podkapitole o programu terminálu bude definován adresář */packages/frontend-terminal*, v přiloženém zdrojovém kódu (Příloha A), jako kořenový. Adresářová struktura projektu je poté následující:

- src/ – Zdrojový kód aplikace pro vývoj
  - assets/ – Assety (obrázky, fonty, apod.), které se vkládají do kódu
  - components/ – React UI komponenty
  - services/ – Služby (například pro příjem/odesílání dat)
  - store/ – Definice stavového kontejneru Redux
  - types/ – Definice TS typů
  - index.tsx – Vstupní soubor aplikace
- build/ – Sestavený zdrojový kód pro produkční běh
- public/ – "Veřejné" soubory aplikace
- tsconfig.json – Konfigurace TypeScriptu
- package.json – NPM manifest projektu
- Soubory .env – Konfigurační soubory (podle názvu se konfigurace aplikuje na dané prostředí - vývoj, produkce, testy)

4.2.2 Komunikace s backendem

Komunikace se serverovou aplikací se navazuje přes protokol WebSocketů. Pro usnadnění práce byla použita knihovna socket.io-client, která implementuje funkce klienta.

Díky této implementaci je možné využívat ke komunikaci se serverem standardní JavaScriptové API metod a eventů, místo nutnosti serializovat/-deserializovat výstupní/vstupní data a spravovat spojení ručně. Po navázání spojení se serverem (pomocí `io('http://example.com');`)"je možné odesílat serveru zprávy (např. `socket.emit('foobar', data);`), nebo pomocí eventu data očekávat (např. `socket.on('foobar', callback)`). To vše navíc s podporou formátu JavaScriptových objektů, které jsou de/serializovány z/do JSONu. V případě, že se spojení přeruší, se klient bude snažit navázat spojení znovu po nekonečně dlouhou dobu (lze nakonfigurovat).

Komunikace mezi serverem a terminálem tedy probíhá pouze po tomto živém kanálu. Terminál nemá implementovány žádné další nástroje pro komunikaci přes HTTP nebo jiné protokoly. Pokud je třeba, aby si terminál stáhnul

nějaký externí obsah, nebo volal externí API, je třeba implementovat tuto funkcionalitu přímo v obsahu, který terminál přijal. Je samozřejmě možné implementovat rozhraní, které může přijatý obsah využívat, ale to v tomto případě není nezbytné a jedná se spíše o nice-to-have[3].

4.2.3 Stavový kontejner Redux

Aplikace obsahuje centrální stavový kontejner, který je implementován pomocí knihovny redux. Díky této knihovně bylo možné v aplikaci vytvořit jediný kontejner, který obsahuje veškeré potřebné data o stavu terminálu. K tomuto stavu má přístup celá aplikace. Aktualizace tohoto stavu jsou předvídatelné a nemutovatelné (nelze změnit jen část stavu). Tímto lze dosáhnout přehlednějšího kódu, protože většina komponent aplikace nepotřebuje řešit svým vlastním způsobem správu stavu. Navíc aktualizace stavu probíhají asynchronně, tudíž neblokují uživatelské rozhraní.

Stav má v aplikaci následující tvar:

```
{
  webSocketState: WebSocketState,
  content: {
   type: EContentType,
   id: number,
   show: boolean,
   data: string,
   dataSize: number,
   progress: number,
   info: {
     author: string,
   },
  }
}
```

Výpis 4.1: Formát stavového kontejneru v aplikaci terminálu

V tomto objektu lze tedy uložit veškerá důležitá data o aktuálním stavu aplikace. V závislosti na tomto stavu se uživateli zobrazuje obsah, spořič (obsah typu NOTHING) nebo načítací obrazovka.

4.2.4 Životní cyklus

Životní cyklus terminálu není komplikovaný a skládá se ze tří hlavních částí.

Inicializace

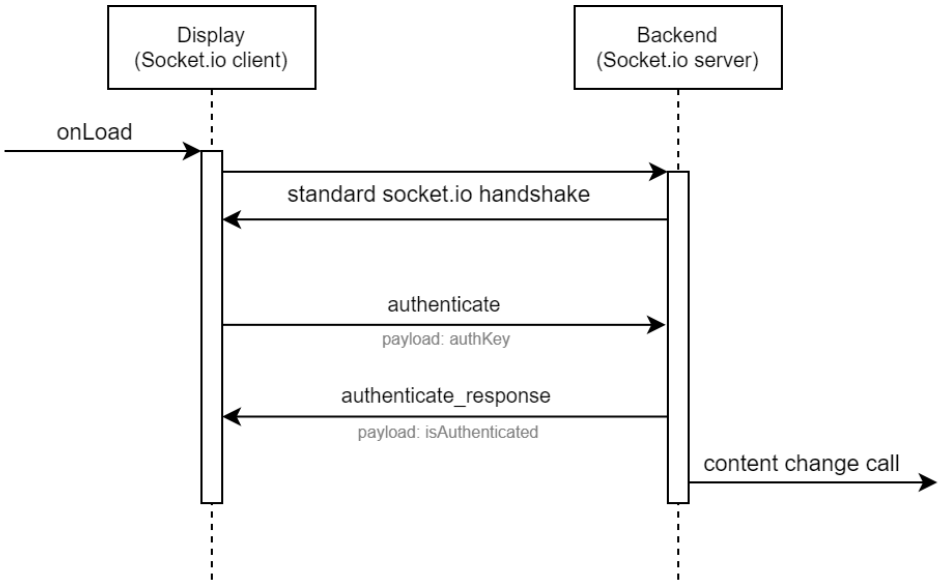
Během inicializace aplikace terminálu budou provedeny následující kroky:

1. Inicializace stavového kontejneru (*redux-store*).
2. Zobrazení načítacího UI (komponenta *LoadingScreen*).

[3] Funkcionalita nad rámec která je sice vítaná, ale není nezbytná.

3. Připojení k aplikačnímu serveru přes protokol *WebSocket*
4. Odeslání autentifikačního klíče.
5. Inicializace posluchačů naslouchajících zprávám od serveru.

Body 2. a 3. lze ještě přiblížit následující ilustrací (Obrázek 4.1):



Obrázek 4.1: Sekvenční diagram inicializace komunikace se serverem

Příjem a zobrazování obsahu

Po finálním kroku inicializace následuje čekání na odpověď ohledně úspěšnosti autentifikace. Pokud byla autentifikace úspěšná (tzn. byla přijata zpráva "`authenticate_response`"), aplikace je připravena zobrazovat obsah, který jí bude zaslán.

Aplikace čeká na následující zprávy od serveru:

`data_start` – Informace o začátku přenosu nového obsahu obsahující metadata tohoto obsahu.

`data_chunk` – Část přijímaného obsahu a informace o tom, zda je tato část poslední.

`server_ping` – Příznak dotazu na stav. Terminál následně odešle svůj stav server ve zprávě "`state`".

V případě, že byla přijata zpráva "*data_chunk*" s informací, že je tato část poslední, přejde se k zobrazení obsahu pomocí komponenty *<typ>Screen*, kde *<typ>* je závislý na typu právě přijatého obsahu (pro obrázek je to komponenta *ImageScreen*). Toto rozdělení může vést k duplikaci kódu, ale dává nám volnost při přidávání nebo úpravě jednotlivých typů obsahu.

Smrt

Po úspěšném zobrazení obsahu se čeká na další obsah k zobrazení. Terminál se nachází v tomto stavu, dokud nenastane finální životní cyklus. Tímto životním cyklem je smrt, která může nastat z jediného důvodu, a tím je zavření okna prohlížeče (ať už standardně nebo pádem). Prohlížeč se sám postará o odstranění dat v mezipaměti.

4.2.5 Gamepad API

Součástí implementace serveru je jednoduchý systém, který zpracovává a odesílá dynamický obsah do terminálu. Ve chvíli, kdy terminál provede zobrazení obsahu, který je označen jako interaktivní, program terminálu inicializuje rozhraní Gamepad API.

Toto rozhraní dovoluje uživatelům vstupního zařízení terminálu (v našem případě Xbox ovladače) předávat obsah běžícímu dynamickému obsahu. Po inicializaci se přejde k přiřazení objektu ovladače, který obsahuje aktuální stav tlačítek a jiných vstupních prvků.

Problém při použití Gamepad API je ten, že nedokáže odesílat eventy o stisknutí tlačítka svým posluchačům. Proto je zapotřebí, aby dynamický obsah, který využívá vstupní zařízení terminálu, zobrazoval obsah v jednotlivých oknech cyklu (obsah zobrazujeme v oknech - ticích). Pak je možné se v každém okně daného cyklu (tiku) zeptat na stav tlačítek vstupního zařízení.

4.2.6 Další komponenty aplikace

Aplikace ještě obsahuje několik dalších komponent, které stojí jen za stručný popis:

Soubor *`/socket/socket.service.ts`* – Obsahuje singleton (inicializuje se pouze jednou) třídu, při jejíž inicializaci se aplikace pokusí navázat spojení se serverem, zajistí autentifikaci a nabídne rozhraní knihovny socket.io-client zbytku aplikace.

Soubory *`*.scss`* – Definice stylů, které se používají v UI napříč aplikací. Soubory formátu SCSS se při sestavování aplikace zkompilují do jednoho velkého CSS souboru (tzn. CSS bundlu).

Soubory *`*.sh`* – Soubory určené ke snadnému vzdálenému spouštění a vypínání terminálu přes rozhraní portálu, nebo SSH připojení.

4.3 Program portálu

Tento program běžící na počítači uživatele je rozhraním mezi uživatelem a aplikačním serverem. Jeho účelem je poskytovat možnost snadného nahrávání a správy obsahu.

4.3.1 Obecná struktura

V případě této webové aplikace, která se řídí konceptem SPA, se dá říct, že je pouze úhlednějším rozhraním (prostředníkem) pro komunikaci s rozhraním aplikačního serveru. Ostatně to je většina klientských aplikací v posledních letech. Díky tomu není struktura aplikace z architektonického pohledu nijak zajímavá a není třeba detailně popisovat každou komponentu, ze které se portál skládá. Důvodem je fakt, že většina těchto komponent neobsahuje žádnou složitou logiku, pouze definice UI a metody pro volání API.

Program je po sestavení spustitelný ve webovém prohlížeči. V této podkapitole o programu portálu bude definován adresář /packages/frontend-portal, v přiloženém zdrojovém kódu (Příloha A), jako kořenový. Adresářová struktura projektu je poté následující:

- src/ - Zdrojový kód aplikace pro vývoj
  - assets/ - Assety (obrázky, fonty, apod.), které se vkládají do kódu
  - components/ - React UI komponenty
  - pages/ - React UI komponenty, které se chovají jako jednotlivé stránky aplikace
  - services/ - Služby (například pro příjem/odesílání dat)
  - store/ - Definice stavového kontejneru Redux
  - types/ – Definice TS typů
  - utils/ - Pomocné nástroje a rozhraní
  - index.tsx – Vstupní soubor aplikace
- build/ – Sestavený zdrojový kód pro produkční běh
- public/ – “Veřejné” soubory aplikace
- tsconfig.json – Konfigurace TypeScriptu
- package.json – NPM manifest projektu
- Soubory .env – Konfigurační soubory (podle názvu se konfigurace aplikuje na dané prostředí - vývoj, produkce, testy)

Adresářová struktura je téměř totožná jako v předchozí kapitole 4.2. To platí i pro samotnou architekturu aplikace, protože je také koncipována jako SPA, která je napsaná s pomocí knihovny React.

4.3.2 Komunikace s backendem

Na rozdíl od aplikace terminálu komunikuje aplikace portálu se serverem pomocí HTTP dotazů. Navíc se snaží dodržovat standardy stylu REST a následuje formát dotazů HTTP definovaný frameworkem serveru - NestJS. Příklady formátu dotazů budou tedy představeny až v popisu aplikačního serveru.

Je vhodné zmínit, že pro komunikaci byla použita knihovna *axios*, která je implementací HTTP klienta. V souborech "*/utils/Api.ts*" a "*/services/api*" se nachází vlastní třída, která obaluje implementaci *axios* pro účely této aplikace. Vzniklá třída má tyto výhody oproti výchozí třídě implementace *axios*[4]:

- Vzniklá třída je singleton (inicializována pouze jednou).
- Implementuje do sebe konfiguraci aplikace.
- Formátuje chybové hlášky dotazů do formátu, který je pro tuto aplikaci vhodnější.
- Přidání typování výsledků dotazu (viz následující příklad - Výpis).

```
// Example of an HTTP query without the Api class, where the result of the
    query is not typed:
const response: any = axios.get(url, getData)

// Example of an HTTP query with the Api class, where the result of the
    query will be typed:
const response: IResponse = Api.get<IResponse>(url, getData)
```

Výpis 4.2: Rozdíl mezi použitím upravené a neupravené třídy axios

4.3.3 Přihlašování

O tom, zda je uživatel přihlášen, rozhoduje přítomnost JWT. Tento token se do klientské aplikace dostane následujícím způsobem:

- Při požadavku na přihlášení proběhne přesměrování na stránku autentifikačního serveru Zuul (dále jen Zuul).
- Tato stránka nás po úspěšném přihlášení přesměruje na předem definovanou callback adresu.
- Na této stránce se provede kontrola přihlášení vůči našemu serveru.
- Pokud byla kontrola úspěšná, je nám aplikačním serverem přidělen JWT obsahující základní data o uživateli.

V seznamu výše byl zjednodušeně popsán princip autentifikace protokolu OAuth 2 z pohledu klienta. Detailnější popis tohoto protokolu se nachází v kapitole, která popisuje implementaci aplikačního serveru.

Klientská aplikace portálu se pouze ptá, zda JWT existuje v úložišti prohlížeče (*LocalStorage*) a zda je validní (sedí checksum). [21]

[4]Tato třída byla vytvořena přibližně před dvěma lety pro jiný projekt, na kterém jsem pracoval. V té době implementace axios ještě nepodporovala typování v TypeScriptu. Dnes toto typování ale podporuje, a proto je vlastní implementace zbytečná.

4.3.4 UI a router

UI aplikace je rozděleno na dvě skupiny stránek, mezi kterými se dynamicky (bez obnovení stránky) přepíná pomocí knihovny react-router:

- Veřejné
  - Přihlášení
    - Výchozí stránka pro nepřihlášené uživatele.
    - Odkaz na přihlašovací stránku Zuulu.
    - Provede odhlášení uživatele (odstraní JWT).
  - Callback
    - Na tuto stránku je přesměrován uživatel, který úspěšně provedl přihlášení na Zuulu.
    - Přihlásí uživatele pomocí JWT a přesměruje ho na výchozí stránku pro přihlášené uživatele.
- Pouze pro přihlášené
  - Hlavní stránka
    - Výchozí stránka pro přihlášené uživatele
    - Obsahuje tabulku fonty a historie fronty
  - Vytvořit statický obsah
    - Stránka, kde uživatel může odeslat statický obsah (obrázek) ke kontrole a následnému zobrazení na terminálu.
  - Vytvořit dynamický obsah
    - Stránka, kde uživatel může odeslat statický obsah (canvas) ke kontrole a následnému zobrazení na terminálu.
  - Spravovat obsah
    - Přístup pouze pro role MODERÁTOR a vyšší.
    - Správa obsahu (potvrzování obsahu, mazání obsahu, apod.).
  - Spravovat terminál
    - Přístup pouze pro role MODERÁTOR a vyšší
    - Správa stavu terminálu (vypnutí, zamknutí fronty, přeskočení na další obsah, atd.)
  - Spravovat uživatele
    - Přístup pouze pro role ADMINISTRÁTOR a vyšší
    - Správa uživatelských účtů
- Stránka chyby 404 (stránka nenalezena)

Je nutné mít na paměti, že i když je obsah stránek z kategorie “pouze pro přihlášené” viditelně nedostupný (pokud není uživatel přihlášený), je možné

nahlédnutím do zdrojového kódu tento obsah přečíst. Samotná ochrana proti provádění akcí neprávem je implementována v aplikaci serveru.

Při vytváření UI byly použity tyto knihovny, které značně urychlují tvorbu vzhledu a dynamických funkcí stránky díky velkému množství předprogramovaných a nastylovaných React komponent:

react-bootstrap – Implementace knihovny *bootstrap* pro použití v Reactu

mdbreact – Implementace knihovny *mdb-ui-kit* pro použití v Reactu

Na závěr stojí za zmínku vlastní řešení zobrazování bootstrap modálních oken, které nevyžaduje externího řízení stavu a je rozšířeno o možnost definovat dynamicky obsah okna. Bylo toho dosaženo použitím React referencí na komponenty [22]. Tím se zvyšuje univerzálnost těchto vyskakovacích oken. Příklad této implementace se nachází například v souboru */src/components/-ContentDetail.modal.tsx*.

4.3.5 Stavový kontejner Redux

Aplikace obsahuje centrální stavový kontejner, který používá stejnou knihovnu jako aplikace terminálu. V případě implementace aplikace portálu je ale stavový objekt podstatně menší

```
{
 isAuthenticated: boolean;
 userFetched: boolean;
 bearer?: string;
 user?: User;
}
```

Výpis 4.3: Formát stavového kontejneru v aplikaci portálu

V tomto objektu se ukládá pouze aktuální stav přihlášení. Ve zbytku aplikace buď není třeba žádného stavu, nebo je stav spravován přímo v konkrétní komponentě. Je tomu tak například v případě komponent, ve kterých se nacházejí tabulky nebo formuláře. Stav tabulek nebo formulářů je sice uložen jako JavaScriptový stavový objekt, ale nemá význam ho mít dostupný v celé aplikaci, protože žádná jiná komponenta přístup k těmto datům nevyžaduje.

4.3.6 Další komponenty aplikace

Aplikace ještě obsahuje několik dalších komponent, které stojí jen za stručný popis:

Soubor *./src/setupProxy.js* – Obsahuje konfiguraci proxy, která umožňuje přístup k API backendu z vývojářského serveru klientské aplikace.

Soubory **.scss* – Definice stylů, které se používají v UI napříč aplikací. Soubory formátu SCSS se při sestavování aplikace zkompilují do jednoho velkého CSS souboru (tzn. CSS bundlu).

Soubor */config.ts* – Prostředník, který vytváří vhodnější rozhraní konfigurace, aby při získávání konfigurační hodnoty nebylo nutné pokaždé používat globální proměnnou "*process.env*"

4.4 Program serveru

Finální a také nejdůležitější program v projektu je aplikační server. Účelem tohoto programu je umožnit nahrávání obsahu, spravovat frontu obsahu a dle potřeby odesílat aktuální obsah terminálu.

4.4.1 Obecná struktura

Struktura celé aplikace je řízena architektonickým vzorem frameworku NestJS [9]. Díky tomuto frameworku je struktura celé aplikace přehledná. Zároveň je třeba psát méně kódu. Framework NestJS také využívá experimentální funkce dekorátorů [23], které známe například z frameworku Spring pro jazyk Java.

Struktura architektury NestJS

Základem architektury frameworku NestJS je složení celé aplikace ze stromu modulů (Module). Tento strom modulů si instance frameworku načte po spuštění aplikace. Díky modulům lze lépe organizovat strukturu aplikace. Moduly lze do sebe vnořovat. V naší aplikaci se moduly dále skládají z těchto komponent:

Provideři (Providers) – Třídy sdílené v daném modulu pomocí Dependency Injection [3]. Providery jsou v naší aplikaci například:

- **Služby (Services)** – Business logika nebo jiné třídy s nějakou logikou aplikace.
- **Brány (Gateways)** – Brány pro komunikaci pomocí WebSocketu.
- **Úlohy (Tasks)** – Třídy, které v závislosti na čase (např. periodicky) volají akce.

Kontrolery (Controllers) – Rozhraní pro příjem a odesílání HTTP požadavků.

Výjimky (Exceptions) – Seznam výjimek (chyb), které mohou v modulu nastat. Tato komponenta není přímo definována v dokumentaci frameworku, ale přispívá k větší přehlednosti kódu.

Entity (Entities) – Databázové entity (ORM)

NestJS obsahuje ještě několik dalších komponent, které mohou být součástí modulů, ale ty nemá význam zmiňovat, protože buď nejsou v naší aplikaci použity, nebo je jejich význam velice malý.

Adresářová struktura

V této podkapitole bude popsána adresářová struktura této aplikace. Aplikace je po sestavení spustitelná v prostředí Node.js. Dále v této podkapitole bude definován adresář */packages/backend*, v přiloženém zdrojovém kódu (Příloha A), jako kořenový. Adresářová struktura projektu je pak následující[5]:

- src/ - Zdrojový kód aplikace pro vývoj
  - ApiModule/ – Modul obshaující implementaci HTTP REST API
  - config/ – Konfigurační soubory a rozhraní
  - DatabaseModule/ – Modul obsahující definice databázové struktury (ORM)
  - decorators/ – Vlastní dekorátory, které znepřehledňují kód
  - EventsModule/ – Modul rozhraní WebSocket serveru
  - guards/ – Definice "hlídačů" HTTP požadavků
  - interceptors/ – Vlastní interceptory
  - MvcModule/ – Modul pro odeslání klientské aplikace terminálu klientovi
  - QueueModule/ – Modul pro správu fronty terminálu
  - TasksModule/ – Modul obsahující asynchronní úlohy
  - TerminalModule/ – Modul pro ovládání stavu terminálu
  - app.module.ts – Kořenový modul, který obsahuje všechny moduly aplikace
  - main.ts – Vstupní soubor aplikace (inicializace frameworku)
  - swagger.module.ts – Modul pro inicializaci integrace s OpenAPI (Swagger)
- dist/ – Sestavený zdrojový kód pro produkční běh
- data/ – Data uložená aplikací (Obsah pro terminál a stav fronty)
- tsconfig.json – Konfigurace TypeScriptu
- package.json – NPM manifest projektu
- Soubory .env – Konfigurační soubory (podle názvu se konfigurace aplikuje na dané prostředí - vývoj, produkce, testy)

[5] V tomto stručném pohledu na adresářovou strukturu se nachází krátký popis, který nemusí zcela vysvětlit, jaký je účel tohoto adresáře/struktury. Pokud se jedná o důležitou součást aplikace, kterou je třeba detailněji vysvětlit, bude popsána v tomto dokumentu později.

4.4.2 HTTP API a REST

Hlavním komunikačním prostředkem mezi uživateli portálu a logikou aplikačního serveru je HTTP server. Framework NestJS nám automaticky tento HTTP server nastaví a spustí. Nám poté stačí pouze vytvořit v modulu kontroler a v něm definovat jednotlivé cesty (dále jen routy) pomocí dekorátorů. Veškeré dotazy a odpovědi jsou poté formátovány podle architektury REST.

Routy

Aplikace obsahuje tyto HTTP routy:

/ (GET) – V produkčním sestavení nám tato routa vrátí klientskou aplikaci portálu

/api/v1 – Kořenová routa API (Obsahuje dokumentaci API - Swagger)

- **/auth** – Routy pro řízení autentifikace
  - **/zuul (GET)** – Přesměruje uživatele na přihlašovací stránku Zuulu
    - **/callback (GET)** – Zpracuje odpověď od Zuulu a provede přihlášení
  - **/profile (GET)** – Vrací data přihlášeného uživatele
- **/user** – CRUD pro správu uživatelů
  - **/ (GET)** – Vrátí seznam všech uživatelů
  - **/:id (CREATE)** – Vytvoření uživatele. Není implementováno.
  - **/:id (GET)** – Vrátí jednoho konkrétního uživatele
  - **/:id (PUT)** – Upraví jednoho konkrétního uživatele
  - **/:id (DELETE)** – Odstraní jednoho konkrétního uživatele
- **/content** – CRUD pro správu obsahu a terminálu
  - **/ (GET)** – Vrátí frontu (seznam všech nahraných obsahů)
  - **/image (POST)** – Routa pro nahrání obrázkového obsahu
  - **/canvas (POST)** – Routa pro nahrání dynamického obsahu
  - **/:id (PUT)** – Upraví jeden konkrétní obsah
  - **/:id (DELETE)** – Odstraní jeden konkrétní obsah
  - **/:id (GET)** – Vrátí jeden konkrétní obsah
  - **/data (GET)** – Vrátí/Zobrazí data konkrétního obsahu
  - **/terminal (GET)** – Vrátí aktuální stav terminálu
  - **/terminal (PUT)** – Upraví stav terminálu

Detailnější popis (dokumentace) jednotlivých rout se nachází na adrese "*<URL>:<PORT>/api/v1*" po spuštění aplikace. Specifikace je vygenerována automaticky podle specifikace OpenAPI (Swagger)[6]. Specifikace obsahuje

[6]Generace je sice automatická, ale vyžaduje, aby každá routa, třída a autentifikační metoda obsahovala dekorátor, který jí popisuje.

dokumentaci všech rout včetně detailu o tom, jaká vstupní data očekávají. V dokumentaci lze také dohledat, jaká data či chybové odpovědi můžeme od serveru čekat nazpět.

HTTP odpovědi

NestJS definuje vlastní specifikaci a nástroje pro formátování HTTP odpovědí ve formátu JSON. Formát takové odpovědi závisí na HTTP stavovém kódu. Dělí se ale primárně na dvě skupiny.

V případě HTTP odpovědí s kódy 2xx je vráceno JSON schéma, které definoval programátor. Můžeme vracet jakékoliv schéma nebo nic (jako je tomu zvykem například u kódu 204 - No Content). Framework v sobě také integruje nástroj class-transformer, díky kterému můžeme provádět bezpečnou transformaci databázových entit do JSON schémat. Databázové entity mají ale v dokumentu vlastní kapitolu, a proto jejich popis nyní přeskočíme.

V případě HTTP chyby (HTTP odpovědi, která nemá kód 2xx), kromě chyby s kódem 400 (chyba validace formuláře), vypadá struktura odpovědi v JSONu následovně:

```
{
  statusCode: number, // Error in numeric format
  error: string, // Error in text format
  message: string // Custom error text
}
```

Výpis 4.4: Struktura JSON odpovědi HTTP chyby (kromě chyby 400)

Chyba s kódem 400 je rezervována pro chyby, které se týkají chybně vyplněného formuláře (nebo obsahují více než jednu chybu). Framework nám v tomto případě vrátí stejnou strukturu jako pro všechny ostatní chyby, ale v poli "*message*" je obsaženo schéma, které je výstupem validační knihovny *class-validator*, kterou NestJS interně používá pro validaci formulářů.

Tato struktura je ale značně nepřehledná a navíc obsahuje pole, která jsou pro naší aplikaci zcela zbytečná. Proto byla definována vlastní výjimka (*CustomBadRequestException*), která provede transformaci výstupu knihovny do námi vyžadovaného formátu. Struktura odpovědi v JSONu s chybou 400 poté vypadá následovně:

```
{
  statusCode: 400,
  error: 'Bad Request',
  message: 'Vstupn data nejsou validn',
  fieldErrors: { // List of errors of individual fields
    [key: string]: string[] // One field can have several text errors
  }
}
```

Výpis 4.5: Struktura JSON odpovědi HTTP chyby s kódem 400

Tato struktura je jednoduchá a zcela dostačující pro účely této aplikace. Výjimka "*CustomBadRequestException*" je také využívána v kontrolerech pro

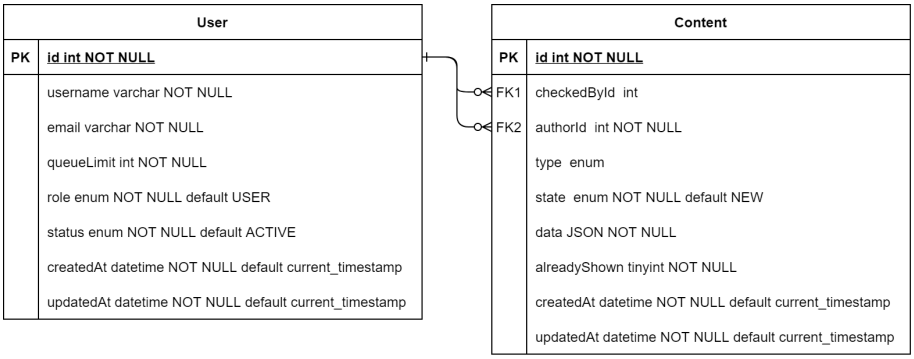
transformaci business výjimek[7], které vznikly až v logice aplikace (většinou ve službě - servise) za kontrolerem a po validaci vstupních dat.

4.4.3 Databáze

Nedílnou součástí celé aplikace je systém pro komunikaci s databázovým serverem. Server do databáze ukládá data o uživatelích a nahraném obsahu. I zde nám byl nápomocen framework NestJS, díky kterému je integrace s databázovým serverem velice snadná. Framework totiž v sobě integruje nástroj *type-orm*, díky kterému lze komunikovat s databází technikou objektově relačního zobrazení (ORM) [19].

Díky této technice je možné v kódu definovat několik entit, se kterými lze pracovat jako z JavaScriptovými objekty. Tyto entity jsou definovány v modulu "*DatabaseModule*". O správu (persistenci) obsahu se pak stará framework samotný. My v kódu pouze voláme metody, které nám poskytuje API nástroje a provádíme operace s JavaScriptovým objektem.

Následující ilustrace (Obrázek 4.2) znázorňuje strukturu databázového modelu, která byla automaticky vygenerována po definici entit.



Obrázek 4.2: Výsledný vygenerovaný databázový model jako UML diagram

Databázový model obsahuje několik vlastností typu enum (výčtový typ):

- User
  - role – [USER, MOD, ADMIN]
  - status – [ACTIVE, INACTIVE]
- Content
  - type – [NOTHING, IMAGE, VIDEO, STATIC_CANVAS, DYNAMIC_CANVAS]
  - state – [NEW, ALLOWED, DENIED]

[7]Například výjimka, kterou aplikace vyhodí v případě, že se uživatel snaží přidat nový obsah, ale limitace jeho účtu to zakazuje.

Je vhodné si povšimnout vlastnosti "*data*" v tabulce *Content*. Důvod pro toto zdánlivě nestrukturované pole je ten, že v závislosti na měnícím se typu obsahu se mění i struktura metadat obsahu. Zde je uvedeno několik příkladů:

IMAGE (Obrázek) – rozlišení, mime type, umístění na pevném disku

VIDEO (Video) – rozlišení, délka, opakování, umístění na pevném disku

NOTHING (Žádný obsah) – žádné

DYNAMIC_CANVAS (Dynamický obsah) – umístění jednotlivých částí a assetů na pevném disku

Struktura tohoto pole proto není definována přímo v databázovém modelu, protože by se značně zvýšila jeho složitost, ale definice a pravidla pro práci s tímto polem se nachází přímo ve zdrojovém kódu aplikace. Přestože tato funkce není implementována, bylo by možné ukládat do tohoto pole přímo binární data obsahu místo odkazu na pevný disk.

4.4.4 Řízení přístupu k API

Všechny funkce této aplikace jsou podmíněny tím, že je uživatel přihlášený. Přihlášení uživatelé mohou nahrávat obsah a sledovat frontu. Aplikace ale slouží také k administraci nahraného obsahu, uživatelů a terminálu, a proto bylo třeba nějakým způsobem řídit přístup k jednotlivým API prostředkům. V aplikaci jsou definovány tyto uživatelské role (přihlášených uživatelů):

USER – uživatel

MOD – moderátor

ADMIN – administrátor

Detailní definice toho, jakou mají tyto role pravomoc, se nachází v sekci 3.2 tohoto dokumentu. Návrh je v tomto případě totožný s implementací.

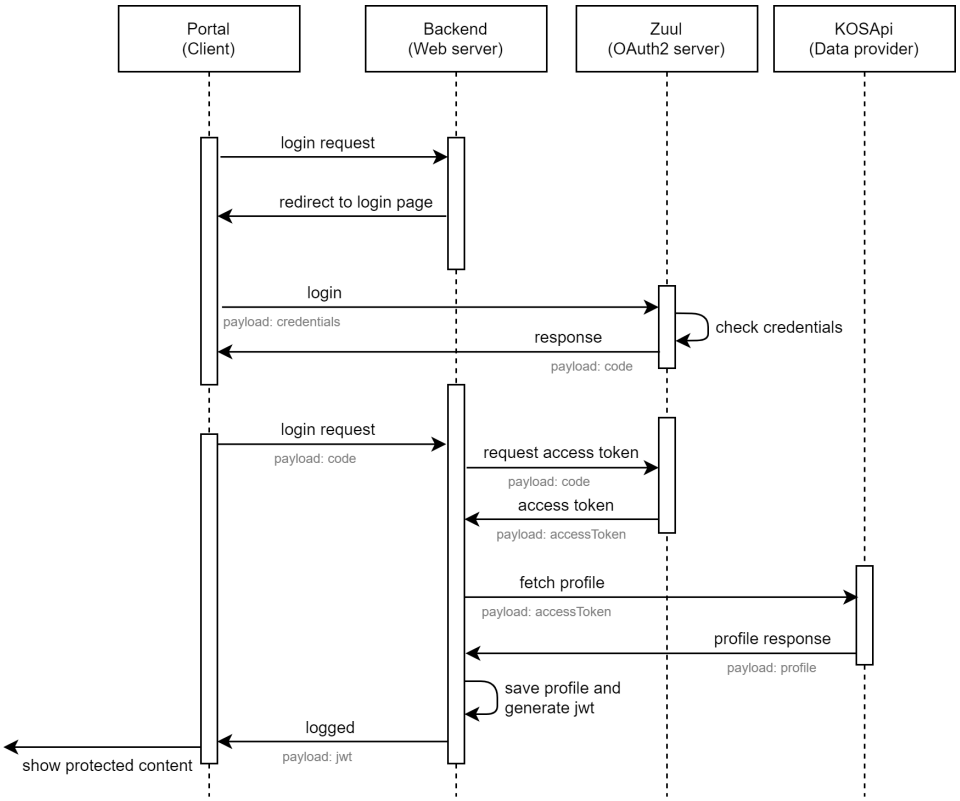
Přihlášení

Přihlášení do aplikace probíhá pomocí protokolu OAuth 2 (dále jen OAuth) [10]. Díky tomuto přístupu není nutná registrace a všechna přihlašovací logika včetně zabezpečení uživatelských hesel je spravována OAuth serverem.

Jako OAuth server byl pro tento projekt vybrán autorizační server ČVUT "Zuul". Tento server nám dovoluje přihlásit pomocí školních údajů. Po přihlášení dostane naše aplikace informaci o tom, že jsou údaje správné a je nám přidělen přístup k několika API rozhraním školy.

Proto, aby bylo možné využít tento OAuth server, bylo třeba provést registraci v app manageru, který je součástí implementace Zuulu [23]. V tomto rozhraní bylo také nastaveno, kam má Zuul přesměrovat uživatele po úspěšném přihlášení.

Ve chvíli, kdy je uživatel přesměrován zpět do našeho projektu, započne proces kontroly, získání uživatelských dat a generace JWT, který je následně předán klientské aplikaci. Spíše než popisovat tento proces slovy, je nejlepší grafické znázornění pomocí sekvenčního diagramu:



Obrázek 4.3: Sekvenční diagram procesu přihlášení přes Zuul

K implementaci tohoto procesu přihlašování a následného řízení přístupu (které bude detailněji popsáno v následujícím bodě) nám znovu pomohl framework NestJS, který v sobě integruje knihovnu passport. Pro následnou generaci JWT, který se používá k řízení přístupu ve zbytku aplikace, byla použita knihovna *jsonwebtoken*.

Ve chvíli, kdy má klient svůj JWT, získává přístup ke všem ostatním zdrojům API. Díky frameworku NestJS je pak kontrola práv zase o něco snazší. V implementaci jsou použity některé techniky frameworku, které nám zaručí, že uživatel nezíská přístup k obsahu, ke kterému by přístup mít neměl.

Hlídač (Guard)

Middleware který provede nějaký typ kontroly ještě před spuštěním kódu v definici routy. Aby nebylo nutné kód této kontroly duplikovat, poskytuje nám framework dekorátor @UseGuard. V implementaci jsou použity dva druhy hlídačů:

AuthGuard(“jwt”) – Hlídač který kontroluje, zda požadavek obsahuje

JWT a zda je tento token validní. Provede také stažení kompletního uživatelského profilu z databáze.

RolesGuard – Vlastní hlídač, který kontroluje, zda má přihlášený uživatel alespoň jednu z rolí, které jsou definovány u routy dekorátorem *@Role*.

Interceptor

Middleware provede transformaci obsahu před odesláním odpovědi klientovi. Aby nebylo nutné kód této kontroly duplikovat, poskytuje nám framework dekorátor *@UseInterceptors*.

Framework nám poskytuje interceptor *ClassSerializerInterceptor*, který dokáže převádět JavaScriptové objekty na JSON schémata. Zásadním nedostatkem této třídy je ale nemožnost filtrovat jednotlivá pole podle role.

Příkladem tohoto nedostatku by mohla být situace, kdy si uživatel načte frontu obsahu. Z databáze stáhneme všechny entity obsahu, které se uživateli smí zobrazit. Tato entita ale obsahuje i informaci o fyzickém umístění obsahu na disku nebo informaci o tom, kdo provedl kontrolu tohoto obsahu. Tyto údaje by neměl uživatel vidět a výchozí interceptor (*ClassSerializerInterceptor*) nám nedovolí tyto údaje skrýt (transformuje slepě celý objekt).

Z tohoto důvodu bylo provedeno vlastní rozšíření tohoto interceptoru[8], které tento nedostatek odstranilo a dovoluje nám definovat, která pole se mají zobrazit konkrétním rolím.

4.4.5 Validace

Další funkcí frameworku NestJS, která byla použita pro validaci vstupních dat, je technika rour (pipes). Díky této technice je možné zvalidovat a transformovat vstupní data do formátu, který je vyžadován ještě před zavoláním metody routy.

Jako šablonu, jak validovat a transformovat, používáme DTO (Objekty přenosu dat). Knihovna *class-validator* obsahuje velké množství dekorátorů, které můžeme v DTO použít. Roura *ValidationPipe* se potom postará o deserializaci do objektu nebo vyhození výjimek při chybě.

DTO má v projektu ještě jeden účel. Vytváří definici toho, v jakém formátu si má klient se serverem předávat obsah.

4.4.6 Websocket komunikace

Komunikace s terminálem je navázána pomocí obousměrného spojení protokolem Websocket. Díky integraci frameworku s knihovnou socket.io bylo možné snadno implementovat Websocket server, který naslouchá na stejném portu jako HTTP server. V modulu eventů (*EventsModule*) je poté vidět, že

[8] Ve zdrojovém kódu frameworku jsem během implementace vlastního interceptoru narazil na nedostatek, který komplikoval rozšíření. Tento nedostatek jsem nahlásil a byl komunitou následně opraven. [24]

pokud chceme naslouchat nějaké příchozí zprávě, stačí implementovat metodu a připojit k ní dekorátor *@SubscribeMessage*.

Server po inicializaci Websocket serveru očekává spojení s klientem (v našem případě aplikace terminálu). V případě, že klient naváže spojení, se očekává, že okamžitě po vytvoření spojení klient zašle autentifikační kód. Pokud ho nezašle do pár sekund[9], je odpojen. Systém dokonce zaručuje, že klientovi není zaslán žádný obsah i pokud zůstane připojen. Stav klienta uložený v mezipaměti serveru totiž obsahuje příznak, zda byl klient v minulosti úspěšně autentifikován.

4.4.7 Fronta

Jednou z nejdůležitějších částí aplikačního serveru je implementace modulu fronty (*QueueModule*). . Účelem tohoto modulu je udržovat a měnit stav aktuálního obsahu.

Stav fronty

Stav fronty a aktuální obsah se udržují v mezipaměti serveru. Zároveň se ale při každé aktualizaci stavu provede asynchronní operace, která stav (obsah ne[10]) uloží do JSON souboru na pevný disk. Tím zaručíme, že se při restartu aplikačního serveru neztratí tento stav.

```
{
 currentId: number | null;
 pushDate: number | null;
 locked: boolean;
}
```

Výpis 4.6: JSON objekt představující stav fronty

Úlohy

S modulem fronty úzce souvisí jednoduchý modul úloh (TaskModule). Účelem tohoto modulu je spouštět každých 15 sekund tyto dvě akce modulu fronty:

1. Povel k aktualizaci lokálního stavu fronty.
2. Inicializace procesu, který odesílá lokální obsah terminálu přes Websocket spojení.

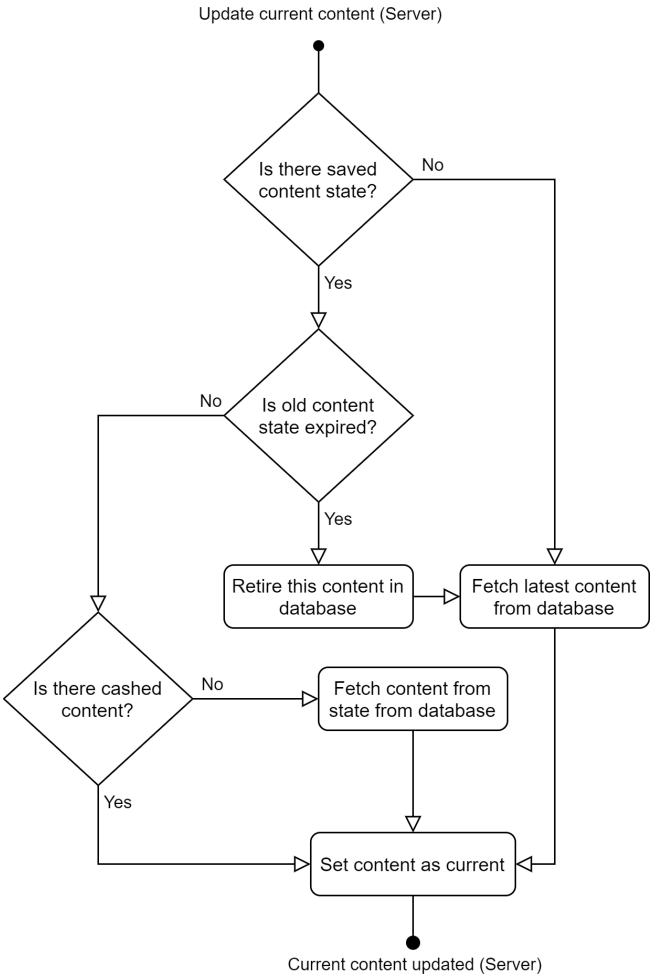
Aktualizace lokálního stavu

Aktuální obsah terminálu se může změnit pouze v případě, že fronta není zamčená a minulý obsah byl aktuálním dostatečně dlouho (výchozím nastavením je jedna hodina). Následující ilustrace znázorňuje hlavní proces modulu

[9] V naší implementaci je to přesně 1 sekunda, ale je možné, že tato hodnota bude muset být v produkčním prostředí navýšena.

[10] Obsah je zbytečné ukládat, protože si ho lze jednoduše stáhnout znovu z databáze.

fronty, který zaručí aktualizaci na aktuální obsah v případě, že jsou podmínky splněny.



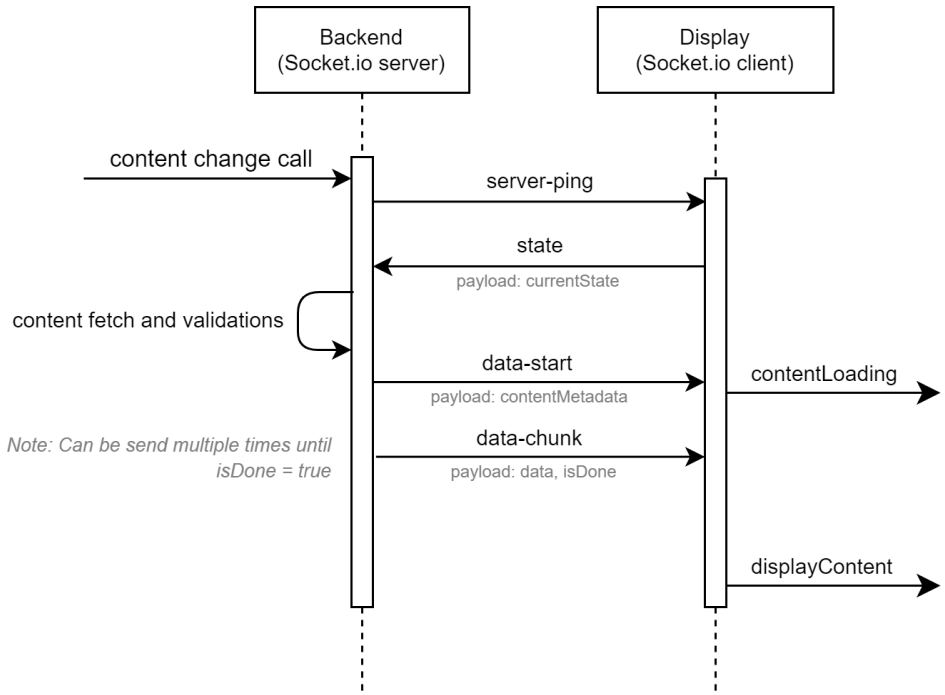
Obrázek 4.4: Diagram znázorňující hlavní proces aktualizace stavu aktuálního obsahu

Tento proces může být spuštěn třemi způsoby:

- Povelem od modulu úloh (periodická kontrola)
- Povelem od metody, která přeskakuje frontu
- Povelem od terminálu (po navázání spojení)

Aktualizace obsahu terminálu

Proces aktualizace obsahu na terminálu je vždy inicializován ve chvíli, kdy skončí proces aktualizace lokálního stavu. Následující ilustrace znázorňuje, jakým způsobem probíhá komunikace mezi serverem a terminálem, než se změní obsah na terminálu.



Obrázek 4.5: Sekvenční diagram procesu změny obsahu terminálu

4.4.8 Ovládání terminálu

Stav terminálu závisí na tom, jaký je lokální stav fronty na aplikačním serveru. Příkazy na změnu stavu jsou ale terminálu předávány pomocí Websocket spojení. To je nám bohužel k ničemu v případě, že je terminál vypnutý nebo zaseklý, protože žádné spojení neexistuje.

V adresáři implementace terminálu (*/packages/frontend-terminal*) se nachází dva skripty *start.sh* a *stop.sh*, které dokáží zapnout a vypnout prohlížeč, ve kterém je spuštěna aplikace terminálu. Tyto skripty jsou vzdáleně volány v případě, že jsme zavolali metodu "*toggleTerminal*" v modulu terminálu (*TerminalModule*). Tato metoda nám dovolí předat terminálu příkaz ke spuštění skriptu pomocí protokolu SSH. Pro tento typ komunikace byla využita implementace SSH klienta *ssh2-promise*.

4.5 Konfigurace řídící jednotky terminálu

Jednodeskový počítač Raspberry Pi 4 který byl použit jako řídící jednotka terminálu, vyžadoval konfiguraci, aby mohl pracovat jako samostatná jednotka, ke které není nutné fyzicky přistupovat. Balík obsahující tuto konfiguraci a návod na instalaci se nachází v příloze B.

4.5.1 Operační systém

Jako operační systém byl vybrán "Raspberry Pi OS with desktop" vydaný 2. prosince 2020 s kernelem verze 5.4 [26]. Tento operační systém je vhodný

z toho důvodu, že je distribuován přímo výrobci našeho jednodeskového počítače. Obsahuje všechny komponenty, které jsou vyžadovány:

- Jednoduché a nenáročné grafické rozhraní (LXDE)
- Nástroje pro vzdálenou správu (VNC a SSH server)
- Webový prohlížeč (Chromium)
- Webový server, který prohlížeči "naservíruje" aplikaci terminálu
- Podporu našeho vstupního zařízení (Xbox kontroleru)

4.5.2 Nastavení

V operačním systému počítače bylo provedeno několik úprav, které zajišťují chod počítače jako kiosku:

- Byl přidán uživatel, který má minimální požadovaná oprávnění pro chod aplikace terminálu
- Přechod do režimu spánku byl vypnut
- Spořič obrazovky byl vypnut
- Po vypnutí se počítač sám zapne
- Pokud na obrazovce existuje kurzor myši, automaticky se po jedné sekundě nečinnosti skryje

4.5.3 Úlohy po spuštění

Po načtení operačního systému počítače jsou provedeny následující akce, které nevyžadují žádný uživatelský vstup:

1. Automatické přihlášení účtu s minimálními oprávněními "terminal"
2. Spuštění VNC a SSH serveru
3. Spuštění webového serveru
4. Spuštění webového prohlížeče v režimu kiosku

Kapitola 5

Uživatelské testování

Cílem uživatelského testování bylo získat odezvu od uživatelů k designu a použitelnosti terminálu a administračního portálu.

5.1 Nasazení

Testerům byl přístupný terminál a notebook s otevřenou aplikací portálu. Jejich nasazení vypadalo následovně:

Terminál – Počítač s konfigurací terminálu dle bodu 4.5 umístěný v obývacím pokoji.

- **Výstupní zobrazovací zařízení** – 30” televize, rozlišení nastaven na 1280x720px.
- **Vstupní zařízení** – Xbox ovladač připojený přes USB rozhraní.
- **Síťové připojení** – Pevné připojení přes Ethernetový port do lokální sítě.

Aplikační server – Běžící na notebooku ve stejné místnosti jako terminál.

- **Síťové připojení** – Připojení přes bezdrátovou síť do lokální sítě.

Portál – Otevřený v prohlížeči na notebooku, na kterém běží aplikační server.

Všechny součásti projektu byly tedy umístěny v jedné místnosti a na jedné lokální síti (která měla přístup k internetu). Testeři tak měli možnost pozorovat a interagovat s terminálem a portálem ve stejnou chvíli. Toto prostředí mělo simulovat uživatele, který stojí před terminálem ve vchodu do budovy a provádí akce na portálu přes své mobilní zařízení.

5.2 Průběh testování a zpětná vazba

Testování se zúčastnilo 5 testerů. Všichni testeři již měli předešlou zkušenost s používáním webových aplikací a byli seznámeni se smyslem projektu. Testeři prováděli testy nezávisle na sobě.

Před zahájením testu bylo každému uživateli zadáno několik úkolů, které se dělily na etapy podle role, která byla uživateli nastavena:

- Uživatel
  - Přihlásit se
  - Přidat statický obsah
  - Přidat dynamický obsah
  - Zjistit jaký typ obsahu a od jakého autora, se zobrazí na terminálu po aktuálně zobrazovaném obsahu.
  - Ovládat dynamický obsah, který se zobrazuje na terminálu
- Moderátor
  - Povolit uživatelem nahraný obsah
  - Zamítnout uživatelem nahraný obsah
  - Zjistit aktuální stav terminálu bez očního kontaktu s jeho zobrazovacím zařízením (z portálu)
  - Přeskočit z aktuálního obsahu terminálu na následující
  - Zamknout a odemknout frontu
  - Vypnout/Zapnout terminál
- Administrátor
  - Upravit roli daného uživatele
  - Odstranit uživatele
  - Deaktivovat uživatele

Testerům bylo vždy nastaveno prostředí tak, aby byly všechny úkoly splnitelné.

Grafické rozhraní portálu

Na konci testů byl testerům zadán úkol, aby zhodnotili, jak se jim líbil vzhled portálu, se kterým pracovali. Všichni testeři uvedli, že přestože vzhled portálu nepůsobí nijak výrazně, je zcela dostačující pro tento účel. Dva testeři pochválili grafickou zpracovanost tabulek a modálních oken.

Funkcionální řešení portálu

Dalším úkolem pro testery po dokončení všech testů bylo, aby provedli hodnocení funkcionálního řešení a jednoduchosti použití portálu. Testerům přišlo řešení jednoduché a přehledné pro všechny zadané úkoly, až na dvě výjimky:

- Tři testeři uvedli, že se jim nepodařilo nalézt tlačítko, které by zamítlo obsah při provádění kontroly obsahu.

- Všichni testeři uvedli, že nemohli nahrát dynamický obsah, protože nevěděli, jak ho vytvořit.

Díky těmto datům bylo zjištěno, že uživatelé jsou zmateni z toho, že v portálu nelze obsah zamítnout, ale pouze odstranit. Důvodem, proč uživatelé nevěděli, jak přidat dynamický obsah, bylo to, že nemají znalost programování, a proto se s tímto výsledkem počítalo.

Funkcionální řešení terminálu

Posledním úkolem testerů bylo, aby zhodnotili funkcionální řešení a jednoduchost použití terminálu. Všichni uživatelé uvedli, že se obsah zobrazuje zcela přehledně. Při ovládání dynamického obsahu nezaznamenali žádný problém. Čtyři testeři zmínili, že nerozeznali dynamický obsah od statického, a proto nevěděli, kdy mohou obsah začít ovládat.

Zhodnocení

Výsledky testů nám ukázaly, že je možné provést na projektu několik úprav, které by mohly vést k větší uživatelské přívětivosti. Zde je několik kroků, díky kterým by se nemusely při dalších testech vyskytovat stejné nebo podobné problémy:

1. Přejmenovat tlačítko, které odstraňuje obsah na “Zamítnout / Odstranit”. Zajistíme tím, že uživatelům bude jasné, že se jedná o akci s totožným výsledkem.
2. Na stránce pro přidání dynamického obsahu uvést, že požadavkem pro tento krok je dobrá znalost webových programovacích technik.
3. Na obrazovce terminálu graficky znázornit obsah, který je možné ovládat vstupním zařízením.

Část III

Závěr

Kapitola 6
Zhodnocení

Cílem této bakalářské práce bylo provést detailní analýzu a návrh nápadu, který jsem dostal již v prvním semestru bakalářského studia, a nakonec realizovat příklad implementace tohoto návrhu. Přestože byl v tomto dokumentu popisován projekt interaktivní tabule ve vchodu do budovy FEL ČVUT, cílem práce nebylo provést instalaci terminálu na toto umístění. Debaty o této instalaci nebyly bohužel kvůli situaci se světovou pandemií zcela možné a ani vhodné.

I když jsem se potýkal při psaní této bakalářské práce s programátorskými problémy (nekompletní dokumentace a nutnost vlastních řešení), časovými problémy (stěhování) a komunikačními problémy (situace ohledně pandemie), cíl práce se podařilo s určitými kompromisy splnit a jsem s ním spokojen.

Největším kompromisem bylo omezení typů obsahů, které lze na terminálu zobrazovat. Z kapitoly návrhu je vidět, že v plánu bylo provést implementaci systému, který dovolí přihlášeným uživatelům vkládat vlastní dynamický obsah. Systém, který by uživatelům takové vývojové prostředí poskytl a dokázal provádět kontroly vytvořeného dynamického obsahu, implementován nebyl. Tato funkcionalita by vyžadovala rozsáhlou uživatelskou dokumentaci a také více času, který jsem bohužel na práci neměl. Proto bylo povoleno přidávat tento dynamický obsah pouze moderátorům a administrátorům, kteří ví jak takový obsah vytvářet.

Při práci na implementaci jsem si osvojil několik nových způsobů a technik realizace softwaru v jazyce TypeScript. Myslím si, že díky výběru frameworků React a NestJS byl postup implementace mnohem rychlejší a zábavnější, než kdybych si nevybral žádný framework a realizoval implementaci od nuly. Získal jsem také několik nových zkušeností s jednodeskovým počítačem Raspberry Pi a jeho pokročilými funkcemi.

Největší překážkou při realizaci byla časová náročnost projektu a fakt, že jazyk TypeScript ještě není zcela zralý jazyk (je relativně nový), a proto nemá kolem sebe velkou komunitu pokročilých uživatelů, se kterými bych mohl svůj postup konzultovat. Bylo třeba několika kompromisů v kódu, abych dosáhl žádaného výsledku.

V rámci testování výsledné implementace jsem mohl provést uživatelské testy pouze v kruhu rodiny a přátel. Tyto testy společně se zpětnými samokontrolami se ale ukázaly zcela dostačujícími pro cíl této práce, kterým

bylo provést návrh a příkladnou implementaci tohoto návrhu.

Rád bych se projektu v budoucnu nadále věnoval, prováděl různá vylepšení a další testy. Chtěl bych také dosáhnout toho, aby byl projekt škole přínosný a terminál byl opravdu nainstalován do vchodu do budovy. Nadále si myslím, že se jedná o zajímavý, zábavný a přínosný projekt.

Literatura

[1] Programovací jazyky. *Programovací jazyky* [online]. Fakulta dopravní ČVUT: ČVUT, 2020 [cit. 2020-12-29]. Dostupné z: https://www.fd.cvut.cz/personal/xfabera/Y1PJ/cviceni1/UPG_progr_jazyky.pdf

[2] Github Language Stats. *Github Language Stats* [online]. Madnight: GitHub, 2020, 3. čtvrtletí 2020 [cit. 2020-12-29]. Dostupné z: https://madnight.github.io/githut/#/pull_requests/2020/3

[3] ČÁPKA, David. Dependency injection a softwarové architektury: Online kurz. *Dependency injection a softwarové architektury* [online]. itnetwork.cz: itnetwork.cz, 2017 [cit. 2020-12-29]. Dostupné z: https://www.itnetwork.cz/navrh/architektury-a-dependency-injection

[4] DB-Engines Ranking: Trend Popularity. *DB-Engines Ranking: Trend Popularity* [online]. db-engines.com: solid IT, 2020 [cit. 2020-12-29]. Dostupné z: https://db-engines.com/en/ranking_trend

[5] DB-Engines Ranking: Samoobslužné kiosky u McDonald's zpříjemní lidem výběr z menu. *Samoobslužné kiosky u McDonald's zpříjemní lidem výběr z menu* [online]. mcdonalds.cz: McDonald's, 2015, 4.12.2015 [cit. 2020-12-29]. Dostupné z: https://www.mcdonalds.cz/o-nas/pro-media/samoobsluzne-kiosky-u-mcdonalds-zprijemni-lidem-vyber-z-menu/

[6] Market share for mobile, browsers, operating systems and search engines. *Market share for mobile, browsers, operating systems and search engines* [online]. netmarketshare.com: NetApplications.com, 2020 [cit. 2020-12-29]. Dostupné z: https://netmarketshare.com/?options=%7B%22filter%22%3A%7B%22%24and%22%3A%5B%7B%22deviceType%22%3A%7B%22%24in%22%3A%5B%22Desktop%2Flaptop%22%2C%22Mobile%22%2C%22Tablet%22%5D%7D%7D%5D%7D%2C%22dateLabel%22%3A%22Custom%22%2C%22attributes%22%3A%22share%22%2C%22group%22%3A%22browser%22%2C%22sort%22%3A%7B%22share%22%3A-1%7D%2C%22id%22%3A%22browsersDesktop%22%2C%22dateInterval%22%3A%22Monthly%22%2C%22dateStart%22%3A%222020-01%22%2C%22dateEnd%22%3A%222020-11%22%2C%22segments%22%3A%22-1000%22%7D

[7] What is REST? *What is REST?* [online]. codecademy.com: Codecademy, 2020 [cit. 2020-12-29]. Dostupné z: `https://www.codecademy.com/articles/what-is-rest`

[8] Uživatel oberstet. Can websocket messages arrive out-of-order? In: *Can websocket messages arrive out-of-order?* [online]. stackoverflow.com: Stack Exchange, 2012 [cit. 2020-12-29]. Dostupné z: `https://stackoverflow.com/a/11809503`

[9] Documentation | NestJS - A progressive Node.js framework. *Documentation | NestJS - A progressive Node.js framework* [online]. nestjs.com: NestJS, 2020 [cit. 2020-12-29]. Dostupné z: `https://docs.nestjs.com/`

[10] KUBA, Martin. OAuth 2. *OAuth 2* [online]. Ústav výpočetní techniky: Masarykova univerzita, 2018 [cit. 2020-12-29]. Dostupné z: `https://is.muni.cz/el/1433/jaro2018/PA160/um/PA160-OAuth-2018.pdf`

[11] Autorizační server ČVUT v Praze. *Autorizační server ČVUT v Praze* [online]. Praha: ČVUT, 2013 [cit. 2020-12-29]. Dostupné z: `https://auth.fit.cvut.cz/`

[12] ROUSE, Margaret. What is Node.js?: Definition from WhatIs.com. *What is Node.js?: Definition from WhatIs.com* [online]. techtarget.com: WhatIs.com, 2012 [cit. 2020-12-29]. Dostupné z: `https://whatis.techtarget.com/definition/Nodejs`

[13] TypeScript: Typed JavaScript at Any Scale. *TypeScript: Typed JavaScript at Any Scale.* [online]. typescriptlang.org: Microsoft, 2020 [cit. 2020-12-29]. Dostupné z: `https://www.typescriptlang.org/`

[14] React - A JavaScript library for building user interfaces. *React - A JavaScript library for building user interfaces* [online]. reactjs.org: Facebook, 2020 [cit. 2020-12-29]. Dostupné z: `https://reactjs.org/`

[15] Redux - A Predictable State Container for JS Apps. *Redux - A Predictable State Container for JS Apps* [online]. redux.js.org: Dan Abramov, 2020 [cit. 2020-12-29]. Dostupné z: `https://redux.js.org/`

[16] Socket.IO. *Socket.IO* [online]. socket.io: socket.io, 2020 [cit. 2020-12-29]. Dostupné z: `https://socket.io/`

[17] Npm | build amazing things. *Npm / build amazing things* [online]. npmjs.com: npm, 2014 [cit. 2020-12-29]. Dostupné z: `https://www.npmjs.com/`

[18] Workspaces | Yarn. *Workspaces / Yarn* [online]. yarnpkg.com: Yarn, 2020 [cit. 2020-12-29]. Dostupné z: `https://classic.yarnpkg.com/en/docs/workspaces`

[19] HOYOS, Mario. What is an ORM and Why You Should Use it. *What is an ORM and Why You Should Use it* [online]. bitsrc.io: medium.com, 2018, 24.12.2018 [cit. 2020-12-29]. Dostupné z: `https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a`

[20] GEHMAN, Chuck. What Is a Monorepo? *What Is a Monorepo?* [online]. perforce.com: Perforce Software, 2020, 5.3.2020 [cit. 2020-12-29]. Dostupné z: `https://www.perforce.com/blog/vcs/what-monorepo`

[21] Introduction to JSON Web Tokens. *Introduction to JSON Web Tokens* [online]. jwt.io: Auth0, 2020 [cit. 2020-12-29]. Dostupné z: `https://jwt.io/introduction/`

[22] Refs and the DOM. *Refs and the DOM* [online]. reactjs.org: Facebook, 2020 [cit. 2020-12-29]. Dostupné z: `https://reactjs.org/docs/refs-and-the-dom.html`

[23] TypeScript: Documentation - Decorators. *TypeScript: Documentation - Decorators* [online]. typescriptlang.org: Microsoft, 2020 [cit. 2020-12-29]. Dostupné z: `https://www.typescriptlang.org/docs/handbook/decorators.html`

[24] APPS MANAGER BETA. *APPS MANAGER BETA* [online]. fit.cvut.cz: ČVUT, 2014 [cit. 2020-12-29]. Dostupné z: `https://auth.fit.cvut.cz/manager/app-types.xhtml`

[25] BUBENÍK, David. Private methods in ClassSerializerInterceptor make impossible to extend from it. *Private methods in ClassSerializerInterceptor make impossible to extend from it* [online]. github.com: GitHub, 2020 [cit. 2020-12-29]. Dostupné z: `https://github.com/nestjs/nest/issues/5957`

[26] Operating system images – Raspberry Pi. *Operating system images – Raspberry Pi* [online]. raspberrypi.org: Raspberry Pi (Trading) Limited., 2020 [cit. 2020-12-29]. Dostupné z: `https://www.raspberrypi.org/software/operating-systems/`

Použité zkratky

API Application Programming Interface

apod. A podobně

atd. A tak dále

CRUD Create, Read, Update, Delete

CSS Cascading Style Sheets

ČVUT České vysoké učení technické

DTO Data transfer object

FEL Fakulta elektrotechnická

GNU GNU's Not Unix!

GPIO General-purpose input/output

HD Vysoké rozlišení

HDMI High-Definition Multimedia Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IP Internet Protocol

JSON JavaScript Object Notation

JWT Json web token

LED Light-Emitting Diode

LXDE Lightweight X11 Desktop Environment

MHD Městská hromadná doprava

ORM Object Relational Mapping

PC Osobní počítač

RAM Random Access Memory

REST Representational State Transfer

SSH Secure Shell

SSO Single Sign-On

TCP Transmission Control Protocol

tzn. Tak zvaný

UDP User Datagram Protocol

UI Uživatelské rozhraní

USB Universal Serial Bus

VESA Video Electronics Standards Association

VNC Virtual Network Computing

Přílohy

Příloha A

Zdrojový kód projektu

V příloze *source.zip* se nachází klon Git repozitáře umístěného na adrese `https://gitlab.fel.cvut.cz/bubenda1/Terminal` (commit: *a76d8355*), který obsahuje zdrojový kód implementace rozšířený o dokumentační komentáře a poznámky.

V případě, že Git repozitář obsahuje novější commit, doporučuji ho použít místo přiloženého archivu. Může obsahovat opravy chyb, které nebyly nalezeny během této práce nebo nové funkce.

Soubor *readme.md* (formát *Markdown*), který se nachází v kořenové složce této přílohy, obsahuje základní pokyny k instalaci a sestavení projektu.

Příloha B

Zdrojový kód konfigurace terminálu

V příloze *terminal.zip*[1] se nachází obraz nakonfigurovaného operačního systému terminálu a pokyny na jeho instalaci do jednodeskového počítače Raspberry Pi 4 (soubor *readme.md* ve formátu *Markdown*).

[1] Je možné, že tato příloha není součástí elektronické verze tohoto dokumentu, protože se její velikost pohybuje kolem 2.7GB. Součástí tištěné verze dokumentu je DVD, které tuto přílohu obsahuje.