Informačný systém pre reklamnú agentúru

Bc. Natália Hyrliková

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Natália Hyrliková**
Osobní číslo:	**A22492**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Informační systém pro reklamní agenturu**
Téma práce anglicky:	**Information System for an Advertising Agency**

Zásady pro vypracování

1. Popište požadavky zákazníka na jednotlivé funkce informačního systému.
2. Navrhněte databázový model systému.
3. Implementujte serverovou část aplikace pomocí Java Spring Boot a vytvořte její uživatelské rozhraní.
4. Aplikujte zabezpečení komunikace mezi front-endem a backendem.
5. Vytvořte dokumentaci systému s využitím Open API.

Forma zpracování diplomové práce: **tištěná/elektronická**
Jazyk zpracování: **Slovenština**

Seznam doporučené literatury:

1. SONI, Namrata. Spring Boot with React and AWS. APress, 2021. ISBN 1484273915.
2. SHARMA, Sourabh. Modern API Development with Spring 6 and Spring Boot 3 – Second Edition: Design scalable, viable, and reactive APIs with REST, gRPC, and GraphQL using. PACKT PUB, 2021. ISBN 1804613274.
3. LARSSON, Magnus. Microservices with Spring Boot and Spring Cloud. 2nd Edition. Packt Publishing Limited, 2021. ISBN 1801072973.
4. HECKLER, Mark. Spring Boot: Up and Running. USA: O'Reilly Media, 2021. ISBN 1492076988.
5. RICHTER, Justin a SANSO, Antonio. OAuth 2 in Action. Manning Publications, 2017. ISBN 1492076988.

Vedoucí diplomové práce: **Ing. Petr Žáček, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne

Natália Hyrliková v.r.
podpis studenta

ABSTRAKT

Diplomová práca je zameraná na návrh informačného systému pre účely reklamnej agentúry. Predpokladaným výsledkom je implementovaná serverová časť aplikácie, používateľské rozhranie a nasadenie na aplikačný server.

Práca sa skladá z teoretickej a praktickej časti. V teoretickej časti sa venujem analýze funkcionálnych požiadaviek na systém, architektúre a výberu technológií. Praktická časť je zameraná na návrh databázového modelu a implementáciu backend a frontend časti.

Kľúčové slova: reklamná agentúra, informačný systém, PostgreSQL, Java Spring Boot, Angular, WildFly, REST API

ABSTRACT

The diploma thesis is focused on designing an information system for the purposes of an advertising agency. The expected outcome is the implemented server-side application, user interface, and deployment to the application server. The thesis consists of theoretical and practical parts. In the theoretical part, I analyze the functional requirements of the system, architecture, and technology selection. The practical part focuses on designing the database model and implementing the backend and frontend components.

Keywords: advertising agency, information system, PostgreSQL, Java Spring Boot, Angular, WildFly, REST API

V prvom rade by som rada vyjadrila vďaku svojej rodine, za ich podporu počas celého môjho štúdia.

Poďakovanie tiež patrí mojim kolegom, ktorí mi poskytli cenné rady pri procese tvorby tejto práce.

Osobitné poďakovanie patrí môjmu partnerovi Branislavovi Pikovi. Tvoja láska a trpezlivosť mi dodali silu aj v tých najnáročnejších chvíľach.

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

OBSAH

ÚVOD

Reklamná agentúra po rokoch svojho pôsobenia na slovenskom trhu dospela k poznatku, že pre ich potreby je vhodné zmeniť pracovné postupy a zaviesť digitalizáciu procesov na jednotlivých oddeleniach. Každé oddelenie malo svoje zabehnuté postupy, mnoho krát riešené na úrovni Excel tabuliek. V prípade ak jednotlivé oddelenia neposkytli informácie načas alebo ich poskytli v nekvalitnej forme vznikali finančné straty.

Reklamná agentúra ma oslovila so žiadosťou, aby som pre nich vytvorila informačný systém na mieru a teda proces vývoja sa stal témou tejto diplomovej práce. Vývoj bol rozdelený do niekoľkých krokov. Ako prvé som absolvovala niekoľko kôl konzultácií s klientom, na ktorých sme prebrali detaily požiadaviek na systém a samotné činnosti zamestnancov. Taktiež sme sa dohodli na časovom pláne, ktorý zachytáva jednotlivé implementačné fázy. V druhom kroku som navrhla technológie, ktoré by mali byť použité počas vývoja. Rozhodla som sa použiť databázu PostgreSQL v kombinácii s backend riešením v Java Spring Boot frameworku, ktorého súčasťou je aj publikovanie REST API. Používateľské rozhranie bolo navrhnuté implementovať v technológii Angular 16 a zabezpečenie komunikácie som docielila pomocou Basic autentifikácie. Kompletný informačný systém je nasadený na aplikačnom serveri JBoss WildFly.

Implementačné fázy sú rozdelené do troch základných iterácií. Súčasťou každej iterácie je vykonanie analýzy potrieb, návrh riešenia, implementácia riešenia, kontrola rozpracovanosti a akceptácia klientom. V tejto práci sa venujem prvej implementačnej fáze teda základnej implementácii.

Informačný systém výrazne prispel k zlepšeniu efektivity fungovania a optimalizácii interných procesov v reklamnej agentúre, nakoľko odteraz môžu zhromažďovať všetky informácie na jednom mieste. Okrem toho tiež uľahčuje sledovanie stavu reklamných plôch, kampaní a práce zamestnancov vo firme.

I. TEORETICKÁ ČASŤ

1 INFORMAČNÝ SYSTÉM

Informačný systém je prepojená súprava komponentov používaná na zber, ukladanie, spracovanie a prenos dát a digitálnych informácií. Je to zbierka hardvéru a softvéru, dát, ľudí a procesov, ktoré spoločne pracujú na premenení surových dát na užitočné informácie[1].

Informačný systém môže mať niekoľko podôb:

- Webová aplikácia – prístup k takémuto informačnému systému je umožnený zariadeniam, ktoré sú pripojené k internetu. Typickým príkladom je webová stránka, cez ktorú sa vedia zamestnanci prihlasovať a pracovať s dátami [1].
- Počítačová aplikácia – tento druh informačného systému musí byť inštalovaný do svojho počítača vo forme programu [1].
- Mobilná aplikácia – pre používanie tohto typu informačného systému je potreba stiahnutia aplikácie do mobilného zariadenia [1].

1.1 Výhody informačného systému

Medzi hlavné výhody zavedenia informačného systému do spoločnosti patrí:

- Úspora času a finančných prostriedkov – prínos v podobe zavedenia automatizácie procesov a pracovných postupov poskytne skrátenie doby strávenej počas vykonávania a tým ušetrí náklady na pracovnú silu
- Zvýšenie komfortu – namiesto prácneho vyhľadávania informácií v papierovej forme, pomocou pár klikov máme dostupné presné a požadované informácie
- Dostupnejšie analytické dáta – digitalizovaná podoba dát je ľahko spracovateľná do ucelenej štatistiky tak, aby ju bolo možné vyhodnocovať a tým zlepšovať výkonnosť spoločnosti
- Zvýšenie kvality služieb – jeden zdroj informácií poskytuje v čase vždy pravdivé údaje a tým pomáha zamedziť chybovosti a strate dát

1.2 Životný cyklus vývoja informačného systému

Vývoj informačného systému prechádza niekoľkými vývojovými fázami, ktoré sú reprezentované v tzv. životnom cykle. Základné fázy vývoja informačného systému sú:

- Identifikácia potrieb a plánovanie – identifikácia a získavanie požiadaviek na systém od používateľov zahŕňa analýzu súčasných procesov a zber požiadaviek od

zákazníka. Výstupom je taktiež stanovanie veľkosti projektu, naplánovanie kapacít a odhad časového plánu.

- Analýza – spočíva v hĺbkovom rozbore získaných poznatkov, vytváraní dokumentov v podobe používateľských scenárov alebo testovacích scenárov podľa typu požiadavky na implementáciu.
- Návrh – obsahuje proces vytvorenia architektúry informačného systému od návrhu databázy, používateľského rozhrania až po použité technológie na samotný vývoj a správu riešenia.
- Implementácia – predstavuje samotný vývoj, teda realizáciu riešenia v podobe úkonov ako sú programovanie, vytvorenie prostredí na nasadenie a beh aplikácie.
- Testovanie – overuje kvalitu implementácie prostredníctvom rôznych typov testov ako napríklad jednotkové alebo akceptačné testy.
- Nasadenie a podpora – táto fáza zahŕňa zavedenie informačného systému do prevádzkového režimu. Taktiež je súčasťou vykonávanie podpory pre používateľov, ladenie chýb ktoré neboli zistené vo fáze testovania.

Jednotlivé vývojové fázy sa môžu opakovať v závislosti od požiadaviek na nové funkcionality a potreby aktualizácie.

2 DOSTUPNÉ RIEŠENIA

Výsledkom hľadania aktuálne dostupných riešení bolo množstvo rôznych aplikácii/informačných systémov, ktoré poskytovali možnosti rezervácie termínov, ale neposkytovali rozlišovanie entity reklamná plocha, resp. kombináciu entity reklamná plocha, reklamné zariadenie. Ďalšie riešenia poskytovali možnosti vykonávania objednávok bez možností rezervácie reklamnej plochy resp. jej objednanie na presný dátum. Veľkou nevýhodou väčšiny informačných systémov bola platformová závislosť.

Preto som pristúpila k riešeniu, ktoré bude plne spĺňať predstavy klienta a v budúcnosti bude možné ho rozširovať o nové funkcionality.

Pre ukážku som vybrala dvoch zástupcov od rôznych dodávateľov a v každom identifikujem časť, ktorá by sa dala využiť a zároveň identifikujem chýbajúce podstatné požiadavky na systém.

2.1 KODAS – Informačný systém KIS

Informačný systém KIS je balík programov vyvinutý vo firme KODAS určený na spracovanie ekonomických informácií malých a stredných firiem v sústave podvojného, alebo jednoduchého účtovníctva. Systém je vyvíjaný pre prevádzku na platforme Microsoft Windows a pre ukladanie dát je nutný Microsoft SQL Server [2].

Aktuálne podporované klientske operačné systémy [2]:

- Windows XP
- Windows Vista
- Windows 7

Informačný systém KIS, by sa dal využiť z pohľadu modulov pre registráciu a správu zákazníkov, odosielanie a evidenciu objednávok.

Diskvalifikáciou pre využitie tohto informačného systému je najmä absencia základných entít ako je reklamná plocha a kampaň. Druhou podstatnou nevýhodou je závislosť na platforme Windows a zároveň zastaranosť riešenia. Z pohľadu implementácie potrebnej funkcionality by vznikli vyššie náklady na implementáciu chýbajúcich modulov ako na implementáciu vlastného informačného systému.

2.2 KROS – Informačný systém ONIX

ONIX je inteligentným riešením pre firmy, ktoré potrebujú efektívnejšie riadiť firemné procesy. Program ocenia najmä obchodné firmy, ktoré sa venujú zákazkovému a veľkoobchodnému predaju. Informačný systém Onix poskytuje základné moduly pre správu zákaziek, riadenie veľkoobchodu, riadenie služieb, informácie o výrobe, riadenie eshopu [3].

Výhodou tohto informačného systému je jeho rozmanitosť v podobe už existujúceho modulu správa zákaziek a evidencia služieb. Taktiež je výhodou existencia fakturačného modulu, ktorý dokáže exportovať faktúry priamo do účtovného softvéru. Informačný systém je platformovo závislý na operačnom systéme Windows.

Po zvážení výhod som dospela k záveru, že závislosť na operačnom systéme je nevhodná nakoľko zamestnanci reklamnej agentúry pracujú v teréne a tento fakt by viedol k dodatočnej implementácii novej platformovo nezávislej aplikácie dostupnej z internetu. Ďalšou nevýhodou informačného systému Onix je nie úplne dodržaná terminológia používaná v praxi reklamnou agentúrou, a preto by používatelia museli meniť svoje návyky na pracovné návyky.

3 TECHNOLÓGIE

V tejto časti sa budem venovať jednotlivým technológiám, ktoré sú často používané v procese vývoja informačného systému zároveň som sa rozhodla tieto technológie použiť aj pri vývoji informačného systému pre reklamnú agentúru.

3.1.1 Ubuntu

Ubuntu je kompletná distribúcia operačného systému Linux pre pracovné stanice a servery, založená na Linux distribúcii Debian. Ubuntu server je plne bezplatný a funguje na takmer každom hardvéri alebo virtualizovanej platforme. Ubuntu server dokáže vykonávať všetko, čo by server vykonávať mal, ako napríklad web-hosting, file-sharing, cloud služby. Ubuntu server ma minimálne požiadavky na hardware. Je to jedna z najlacnejších a najefektívnejších volieb pri výbere operačného systému pre server [4].

3.1.2 Nginx

Nginx je open-source webový server a reverzný proxy server. Je to jedno z najpopulárnejších riešení na poskytovanie webového obsahu a je známe svojou vysokou výkonnosťou, stabilitou, bohatými funkciami, jednoduchou konfiguráciou a nízkou spotrebou zdrojov. Nginx sa používa na webových serveroch, cloudových službách a ďalších online platformách, ktoré potrebujú efektívne a spoľahlivo poskytovať webový obsah. Môže byť tiež použitý ako loadbalancer, HTTP cache alebo mail proxy [5].

3.1.3 WildFly

WildFly tiež známy ako JBoss je multiplatformový open source aplikačný server napísaný v jazyku Java, aktuálne udržiavaný spoločnosťou Red Hat [6]. Poskytuje všetky potrebné funkcie na spustenie webovej aplikácie, ako je napríklad:

- vytvorenie databázového pripojenia (regulácia počtu otvorených pripojení) [7]
- nasadenie aplikácie vo forme WAR [7]
- základná autentifikácia [7]
- udržiavanie logov aplikácie [7]

3.1.4 Basic autentifikácia

Predstavuje základnú autentifikáciu klienta voči aplikačnému serveru. Aplikačný server kontroluje hlavičku HTTP na obsah atribútu *Authorization*, ktorý sa skladá z kľúčového

slova Basic a textového reťazca vo formáte base64, ktorý obsahuje zakódovanú informáciu v tvare meno_používateľa:heslo [8]. V prípade úspešného overenia je požiadavka spracovaná v samotnej aplikácii. V opačnom prípade je odpoveď na požiadavku HTTP kód 401 Unauthorized.

Obrázok 1: Príklad priebehu Basic autentifikácie

3.1.5 PostgreSQL

PostgreSQL je voľne šíriteľný objektovo-relačný databázový systém, ktorý na prácu využíva jazyk SQL [9]. Dáta sú reprezentované ako množina tabuliek, ktoré spájajú cudzie kľúče [10].

Funkcie, ktoré podporuje PostgreSQL:

- Užívateľom definované typy [10]
- Užívateľom definované operátory [10]
- Referenčná integrita vrátane cudzích kľúčov [10]
- Podpora indexovania [10]
- Triggery [10]
- Transakcie [10]
- Funkcie [10]

Tieto funkcie prispievajú k skutočnosti, že PostgreSQL je bezpochyby najvyspelejší databázový systém z programovacieho hľadiska. Používanie PostgreSQL môže dramaticky skrátiť programovací čas pri mnohých projektoch, pričom jeho výhody stúpajú, so zložitosťou projektu [10].

3.1.6 Spring boot

Spring Boot je open source framework určený pre vývoj webových aplikácií v jazyku Java [11]. Toto sú hlavné výhody používania Spring Boot frameworku:

- Uľahčuje konfiguráciu prostredia [11]
- Nie je potrebná konfigurácia pomocou XML [11]
- Súčasťou je web server (nie je potrebné konfigurovať externý web server) [11]
- Automaticky rozoznáva bežne používané knižnice [11]

3.1.7 JUnit

JUnit je testovací framework pre jazyk Java, ktorý uľahčuje vývojárom testovať svoje aplikácie [12]. Pred samotným testovaním je potrebné vytvoriť testovací prípad, to znamená vytvoriť scenár, ktorý zabezpečí, že logika programu funguje podľa očakávania[13]. JUnit testovanie vo všeobecnosti pomáha pri vývoji čistého kódu, zisťuje chyby v kóde a robí kód spoľahlivejším.

JUnit poskytuje niekoľko funkcií, ktoré uľahčujú vytváranie a spúšťanie testov:

- Overenie - používa sa na overenie očakávaného správania systému. JUnit poskytuje sadu metód na kontrolu výsledkov testu [12].
- Testové spúšťače - sa používajú na vykonávanie testov a hlásenie výsledkov. JUnit poskytuje grafický testovací spúšťač, ktorý môže spúšťať testy a zobrazovať výsledky [12].
- Testové sady - sa používajú na zoskupenie súvisiacich testov. JUnit poskytuje spôsob vytvárania testových skupín, ktoré možno spúšťať spolu [12].
- Hlásenia - Pri spúšťaní testov JUnit pomáha analyzovať výsledky. Poskytuje zabudovaného reportéra, ktorý vypíše informácie o vykonaných testoch [12].

3.1.8 REST API

REST je architektúra API, ktorá nám umožňuje pristupovať k dátam a vykonávať nad nimi CRUD operácie. REST je bez-stavový, čím jednak značne zjednodušuje komunikáciu s API a umožňuje paralelné spracovanie obsahu. Zároveň ho možno dosť ľahko použiť s HTTP, čo je veľmi rozšírený protokol [14]. REST API by malo akceptovať len JSON formát v požiadavkách a tiež v odpovediach preto je menej flexibilné [15].

Medzi najpoužívanejšie HTTP metódy na získavanie a odosielanie dát patria tieto:

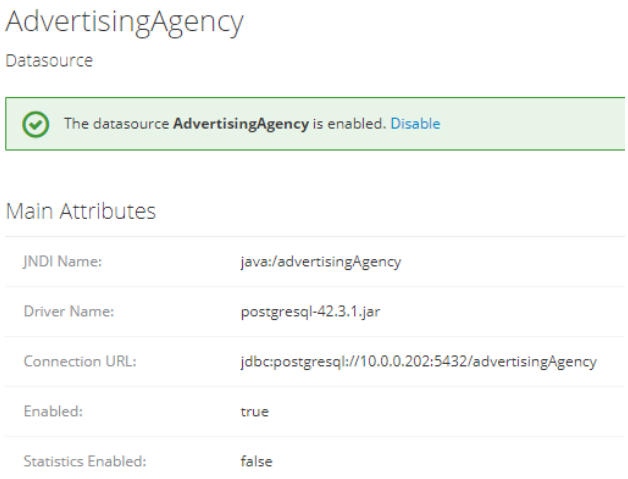
- GET - metóda sa používa na zhromažďovanie informácií zo servera prostredníctvom URI. Ide o metódu určenú len na čítanie a nemala by nijako ovplyvňovať údaje [16].
- POST - metóda sa používa na vytváranie nových entít, ako aj na odosielanie údajov na server, napríklad informácií o zákazníkoch, nahrávanie súborov atď. prostredníctvom formulárov HTML [17].
- PUT - metóda sa používa na aktualizáciu entity alebo vytvorenie novej entity [18].
- DELETE - metóda sa používa na odstránenie existujúcich reprezentácií cieľového prostriedku poskytnutého prostredníctvom URI [18].

3.1.9 Algoritmus SHA-256

SHA-256 je skratka názvu Secure Hash Algorithm 256-bit. Ide o 256-bitový bezpečný hašovací algoritmus, ktorý sa používa na kryptografické zabezpečenie [19]. V podstate sa jedná o matematickú funkciu, ktorá dokáže vstupným dátam priradiť unikátny číselný reťazec s určitou pevne danou dĺžkou [20]. Najčastejšie sa táto hašovacia funkcia používa na overenie integrity dát, ukladanie hesiel alebo pri vytváraní digitálnych podpisov [21].

3.1.10 JDBC

Java Database Connectivity (JDBC) je aplikačné programovacie rozhranie (API), ktoré umožňuje programátorovi pripojenie a interakciu s databázami. Aby bolo možné použiť JDBC na pripojenie Java aplikácií k špecifickému databázovému serveru, je potrebný ovládač JDBC [22]. Na identifikáciu databázy slúži URI. Formát URI je závislý na type databázy. V tomto prípade je to postgresql.



Obrázok 2: Príklad JDBC URI

3.1.11 Postman

Aplikácia Postman ja nástroj, ktorý slúži vývojárom na testovanie a správu API. Vďaka grafickému rozhraniu môžeme jednoducho vytvárať, posielať alebo testovať HTTP/HTTPS požiadavky. Požiadavky môžu byť napríklad typu GET, POST, PUT, DELETE. Vytvorené požiadavky je možné zatriediť do oddelených pracovných prostredí alebo kolekcií priamo v aplikácii. Postman taktiež podporuje import alebo export takýchto kolekcií, čo uľahčuje zdieľanie API medzi vývojármi v tíme [23].

3.1.12 Angular

Angular je frontendový framework pre tvorbu webových aplikácií. Stavia na komponentovej architektúre so službami a používa jazyk TypeScript namiesto čistého JavaScriptu. Angular je postavený na tzv. Komponentoch. Základom každej komponenty je programová trieda a tým pádom stavia na základoch objektovo orientovaného programovania. K tejto triede potom Angular nadviaže vlastnú HTML šablónu a dokonca aj CSS pomocou tzv. direktív. Týmto spôsobom môžeme potom rozdeliť webovú stránku na samostatné celky a poskladať ju práve pomocou princípov objektovo orientovaného programovania [24].

3.1.13 Bootstrap

Bootstrap je široko používaný open-source front-end framework, ktorý umožňuje vývojárom rýchlo a efektívne vytvárať responzívne a mobile-first webové stránky a aplikácie. Tento nástroj využíva kombináciu HTML, CSS a JavaScript technológií na poskytnutie konzistentného a príťažlivého vzhľadu pre rôzne webové projekty [25]. Bootstrap je populárny vďaka responzivite. Webové stránky vytvorené pomocou tohto frameworku sa dokážu automaticky prispôsobiť rôznym veľkostiam obrazoviek zariadení a tým používateľovi zobrazia webovú aplikáciu na rôznych zariadeniach s optimalizovaným dizajnom [25].

3.1.14 GitLab

GitLab je systém na správu verzií, ukladanie a spoločný vývoj programového kódu. Git repozitár môže obsahovať históriu vývojových verzii a vetiev, na ktoré sa vývojári môžu obrátiť, keď sa vyskytnú problémy, aby skontrolovali kód, opravili ho a vrátili sa späť k stabilnej verzii softvéru [26]. Medzi výhody GitLabu patrí napríklad to, že je open source a tiež podporuje CICD proces nasadzovania [27].

3.1.15 IntelliJIdea

IntelliJ IDEA je integrované vývojové prostredie Cross-platform (IDE) pre Javu a Kotlin. Je vyvinutý spoločnosťou JetBrains a je k dispozícii v dvoch vydaniach: komunitné vydanie a komerčné vydanie. Komunitné vydanie je k dispozícii na základe licencie open source [28]. Ide o multiplatformové IDE to znamená, že je vhodné pre vývoj v systémoch Windows, MacOS aj Linux. IDE je možné rozšíriť o množstvo doplnkov, vďaka ktorým bude program ešte kompletnejší. Poskytuje automatické dokončovanie, analýzu kódu a možnosť inteligentného refaktorovania [28].

3.1.16 Navicat

Navicat je aplikácia používaná pre správu a vývoj databázových systémov. Podporuje lokálne aj vzdialené pripojenie do databázy, pri čom pripojenia môžu byť napríklad typu Oracle, PostgreSQL, MySQL, MongoDB a mnohé iné. Výhodou pre používateľa je intuitívne grafické rozhranie, ktoré umožňuje navrhnutie databázovej schémy, migráciu dát, vykonávanie SQL príkazov, import a export databázového modelu [29].

II. PRAKTICKÁ ČASŤ

4 POTREBY A POŽIADAVKY ZÁKAZNÍKA

Získavanie predstáv a požiadaviek klienta prebiehalo formou konzultácií v priestoroch reklamnej agentúry. Konzultácie boli v intervaloch 2x za týždeň, pričom na každej konzultácii sa vytvárali zápisy, ktoré som neskôr použila pri návrhu samotného informačného systému.

Klient požaduje aby bola aplikácia dostupná mimo jeho priestorov napr. pre obchodné oddelenie vo forme webovej stránky, ktorá bude využívaná na notebookoch klienta v prehliadači Chrome alebo tabletoch.

4.1 Navrhnuté riešenie novej aplikácie

Novú aplikáciu navrhujem implementovať v niekoľkých fázach tak, aby klient mohol priebežne využívať nové funkcionality a videl aktuálne rozpracované zmenové požiadavky/rozšírenia systému.

Navrhnuté implementačné fázy:

A) Fáza 1- Základná implementácia
- Inicializácia vývojového prostredia pre vývoj
- Inicializácia testovacieho prostredia pre klienta
- Inicializácia produkčného prostredia
- Inicializácia databázového servera
- Migrácia dát / Inicializácia databázy existujúcimi plochami
- Implementácia prihlásenia
- Implementácia zoznamu plôch, vytvorenie a editácia reklamnej plochy
- Implementácia zoznamu kampaní na ploche, vytvorenie a editácia
- Implementácia vytvorenia úkonu lepiča plagátov
- Implementácia prehľadu histórie úkonov na reklamnej ploche

B) Fáza 2 – Rozšírenie funkcionalít
- Implementácia evidencie ponúk
- Implementácia evidencie objednávok
- Implementácia evidencie klientov

C) Fáza 3 – Rozšírenie funkcionalít
- Implementácia objednávok tlače
- Implementácia objednávok odvozu/zvozu vytlačených plagátov

- Implementácia fakturačného modulu
- Implementácia štatistík

Táto diplomová práca sa venuje riešeniu požiadaviek klienta špecifikovaných pre odsek A) t.j. Fáza 1 – Základná implementácia.

4.2 Funkcionálne požiadavky na systém

Systém vyžaduje rozlišovanie používateľského konta tak, aby bolo možné rozlíšiť v čase, kto kedy vykonal akú činnosť v systéme. Systém bude logovať dané úkony do súborov na serveri a v prípade potreby sa tieto logy dajú prehľadávať.

Každý používateľ bude vystupovať s priradenou rolou a na tomto základe bude ošetrený prístup k dátam na obrazovkách informačného systému.

Dáta sa budú uchovávať perzistentne v databáze na databázovom serveri, ktorý bude spustený na virtuálnom serveri.

4.2.1 Základný popis požiadaviek

Informačný systém bude dostupný cez webové rozhranie prehliadača. Systém overí používateľa pomocou prihlasovacích údajov, zobrazí základnú obrazovku a nastaví položky v menu aplikácie podľa pridelenej roly.

4.2.2 Proces prihlásenia

Používateľ informačného systému po zadaní URL adresy do webového prehliadača uvidí možnosť vloženia prihlasovacích údajov.

Prihlasovacie údaje budú tvorené pomocou dvoch základných parametrov a to prideleného používateľského mena a hesla. V prípade používateľského mena sa jedná o skratku, vytvorenú z prvých dvoch písmen priezviska a prvých dvoch písmen mena používateľa. Pre príklad ak by sa prihlasoval Janko Hraško, tak jeho prihlasovacie meno by bolo *jahr*. Každý používateľ bude mať prihlasovacie meno jedinečné. Po piatich za sebou idúcich neplatných pokusoch o prihlásenie z dôvodu nesprávneho hesla, systém zablokuje používateľské konto a zobrazí hlášku o zablokovaní konta. Odblokovanie používateľského konta bude môcť vykonať iba správca aplikácie.

4.2.3 Záhlavie aplikácie

Bude obsahovať informácie v akej časti aplikácie sa nachádza používateľ a informácie o prihlásenom používateľovi.

4.2.4 Menu aplikácie

Po prihlásení používateľa do aplikácie sa nastavia položky menu podľa priradenej roly.

Menu aplikácie v prvej implementačnej fáze bude nasledovné:

- Prehľad reklamných plôch
- Prehľad kampaní
- História úkonov
- Odhlásiť

4.2.4.1 Položka menu Prehľad reklamných plôch

Základnú obrazovku po prihlásení pre rolu administrátor, manažér lepenia a distribúcie, riaditeľ a obchodný zástupca predstavuje prehľad reklamných plôch.

Obrazovka bude obsahovať zoznam reklamných plôch evidovaných v systéme. Zoznam plôch prestavujú stĺpce: *Číslo plochy, Úsek/smer, Ulica/cesta, Umiestnenie, Mestská časť, Stav Plochy*.

Počet záznamov v tabuľke reklamných plôch bude 10 záznamov. Stránkovanie sa bude riešiť formou posúvania medzi stranami stlačením šípky v spodnej časti tabuľky.

V prípade ak používateľ klikne na tlačidlo editácie, zobrazí sa mu obrazovka, ktorá okrem editácie bude slúžiť na zobrazenie detailu plochy a bude obsahovať predom vyplnené komponenty s informáciami o ploche spolu s fotkou plochy a Google mapou. Komponenty budú umožňovať editáciu, teda vykonanie zmien a následne tieto zmeny môžu byť odoslané po stlačení tlačidla *Potvrdiť* na spracovanie v backend časti.

Tabuľka bude ďalej obsahovať tlačidlo pre nahratie fotky plochy. Po stlačení tlačidla sa zobrazí dialógové okno s tlačidlom pre výber fotografie *Nahrať*. Po potvrdení vloženia fotky používateľ stlačí tlačidlo *Potvrdiť*. Po úspešnom nahraní fotografie sa zobrazí potvrdzujúca hláška.

Súčasťou obrazovky bude tiež možnosť vytvorenia novej plochy, ktorá predstavuje základnú entitu celého informačného systému, spolu s reklamným zariadením, na ktorom bude

reklamná plocha umiestnená. Existovať bude niekoľko typov reklamných zariadení podľa veľkosti:

- Billboard
- Bigboard
- Megaboard
- Stena

Ďalej budem evidovať pri reklamnej ploche tieto atribúty:

- Lokalita
- Katastrálne územie
- Mesto
- Mestská časť
- Vlastníctvo
- Mimo obce
- Dátum postavenia
- Dátum odstránenia
- Status
- GPS pozícia
- Ceny
- Umiestnenie
- iné

Atribúty Lokalita, Katastrálne územie, Obec budú reprezentované ako číselníkové hodnoty.

4.2.4.2 Položka menu História úkonov

Bude zobrazená ako základná obrazovka pre rolu *lepič plagátov* a bude zobrazovať zoznam úkonov, ktoré lepič plagátov vykonal v minulosti.

Tabuľka s prehľadom histórie bude obsahovať stĺpce: *Číslo plochy, Typ úkonu, Dátum vytvorenia, Veľkosť súboru*.

Po kliknutí na tlačidlo detailu sa zobrazí detail úkonu v novom dialógovom okne.

Stránkovanie bude riešené formou šípok na spodnej strane tabuľky a zobrazením čísla aktuálnej stránky. Počet zobrazených položiek v prehľade úkonov bude 10.

Súčasťou obrazovky pre históriu úkonov bude aj možnosť vytvorenia nového úkonu pomocou tlačidla *Vytvoriť úkon*.

Po kliknutí na tlačidlo, systém zobrazí používateľovi okno, ktoré bude obsahovať komponenty:

- Výber fotografie
- Výber plochy (filtrovanie nad zoznamom plôch priradených lepičovi plagátov)
- Typ fotografie
    - Detail
    - Pozičná
- Typ úkonu
    - Štandardný úkon
    - Údržba
- Typ akcie
    - Štandardný úkon
        - Výlep
        - Prelep
        - Fotodokumentácia
        - Dolepka
        - Montáž baner
        - Demontáž baner
        - Prelep na modro
        - Montáž baner na bielo
        - Montáž baner na prenájom
        - Zaslepenie
    - Údržba
        - Montáž dosiek
        - Demontáž dosiek
        - Výmena dosiek
        - Kosenie
        - Oprava lavičky
        - Výmena lavičky
        - Poškodená plocha – nutná oprava

Obrazovka bude obsahovať tlačidlo *Potvrdiť*, ktoré po stlačení odošle požiadavku na vytvorenie úkonu a presmeruje používateľ späť na obrazovku prehľadu. Taktiež sa zobrazí hlásenie o úspešnom vytvorení nového úkonu.

Lepič plagátov absolvuje rôzne úkony v rámci svojich povinností. Typy akcií, ktoré má na výber reprezentujú jeho povinnosti a v budúcnosti budú ďalej využívané na rozšírenia aplikácie o evidenciu opráv.

4.2.4.3 Položka menu Prehľad Kampaní

Kampane vytvorené v systéme budú zobrazené v tabuľke v prehľade Kampaní. Ku každej kampani budem zobrazovať názov, stav, počet reklamných plôch zahrnutých do kampane a čas trvania. Tabuľka tiež bude obsahovať tlačidlo pre editáciu kampane.

Konkrétne reklamné plochy zahrnuté do kampane sa zobrazia v detaile kampane. Aj v tomto prípade využijem logiku zdieľania komponenty a teda obrazovka pre editáciu bude slúžiť aj ako obrazovka na zobrazenie detailu o kampani. V editácii kampane bude možné priradzovať plochy do kampane pomocou tlačidla *Pridať plochu.* Po pridaní sa plochy zobrazí v zozname plôch zaradených do kampane.

Pre uľahčenie práce obchodníka, bude aplikácia umožňovať pridanie plochy do kampane cez Prehľad kampaní, kde sa v každom riadku zobrazí tlačidlo *Plus*. Po kliknutí sa používateľovi zobrazí modálne okno, v ktorom vyplní číslo plochy a dátum obsadenosti plochy v danej kampani. Následne úspešné priradenie reklamnej plochy do kampane prebehne po kliknutí na tlačidlo *Potvrdiť*. Počet plôch uvedený v kampani predstavuje hodnotu, ktorá popisuje aký počet plôch si klient praje v dopyte. To znamená, že počet plôch zaradených do kampane je rozdielny od počtu plôch, ktoré klient na konci vyberie do objednávky.

Vytvorenie novej kampane bude možné prostredníctvom tlačidla *Vytvoriť kampaň*. Po kliknutí na tlačidlo sa zobrazí okno, v ktorom používateľ zadá potrebné hodnoty atribútov pre vytvorenie a svoju voľbu potvrdí tlačidlom *Potvrdiť*.

4.2.5 Používateľské roly

Reklamná spoločnosť má niekoľko oddelení, preto v aplikácii potrebujem priradzovať jednotlivým používateľom roly, ktoré budú naviazané na oprávnenia. Tieto oprávnenia budú možnosť používateľa vidieť a interagovať s jednotlivými funkcionalitami.

Roly vystupujúce v aplikácii:

- Administrátor
- Riaditeľ
- Obchodník
- Lepič plagátov
- Manažér lepenia a distribúcie

Nie je vylúčené, že do aplikácie počas vývoja nasledujúcich fáz nepridám ďalšie roly. Pre príklad uvádzam napr. rolu Účtovník, ktorá bude mať prístup k objednávkam a evidencii vystavených faktúr.

Spravovanie používateľov je taktiež predmetom budúceho vývoja, ktorý nie je zahrnutý v prvej fáze implementácie.

4.2.5.1 Rola Administrátor

Predstavuje správcovské konto, ktoré bude mať k dispozícii všetky funkcionality informačného systému dostupné v čase prihlásenia.

Administrátor nebude vytvárať v systéme nové záznamy, nebude sa využívať na aktívne používateľské procesy. Jeho použitie bude mienené primárne pre revíziu.

4.2.5.2 Rola Riaditeľ

Bude najsilnejšiu rolou v informačnom systéme. Jej priradenie bude primárne pre dve osoby v spoločnosti a to pre CEO a CBO reklamnej agentúry.

Rola bude mať prístup ku všetkým záložkám aplikácie. Pre vysvetlenie bude mať dohľad nad všetkými úkonmi na záložke *História úkonov*, bude vedieť zobraziť zoznam plôch a interagovať so záznamami, bude môcť vidieť prehľad kampaní, vytvárať a editovať kampane.

4.2.5.3 Rola Obchodník

Obchodník v prvej fáze uvidí prehľad reklamných plôch, bude môcť vykonávať všetky úkony na tejto obrazovke opísané v **Chyba! Nenašiel sa žiaden zdroj odkazov.**.

Obchodník bude schopný vytvoriť novú plochu, editovať jej atribúty a nastaviť jej status.

Taktiež bude mať možnosť interagovať so záložkou *Prehľad kampaní* a využívať všetky dostupné funkcionality bez obmedzenia.

Táto rola bude slúžiť primárne pre zamestnancov obchodného oddelenia.

4.2.5.4 Lepič plagátov

Ide o používateľa, ktorý sa nachádza v teréne. To znamená potrebuje mať prístup k informačnému systému mimo kancelárie a tak aj musím pristupovať k návrhu jeho používateľského rozhrania.

Lepič plagátov bude vedieť zobraziť *Históriu úkonov*, ktoré boli ním vytvorené a zároveň bude vedieť vytvoriť nový záznam o úkone. Nebude mať možnosť zobraziť prehľad plôch a ani prehľad kampaní.

4.2.5.5 Rola Manažér lepenia a distribúcie

Je rola, ktorá sa nachádza na pomedzí medzi rolou *Obchodník* a *Lepič plagátov*. Bude mať prístup k prehľadu plôch, nebude môcť vytvárať nové plochy a ani editovať informácie o ploche. Zobrazenie detailu bude povolené.

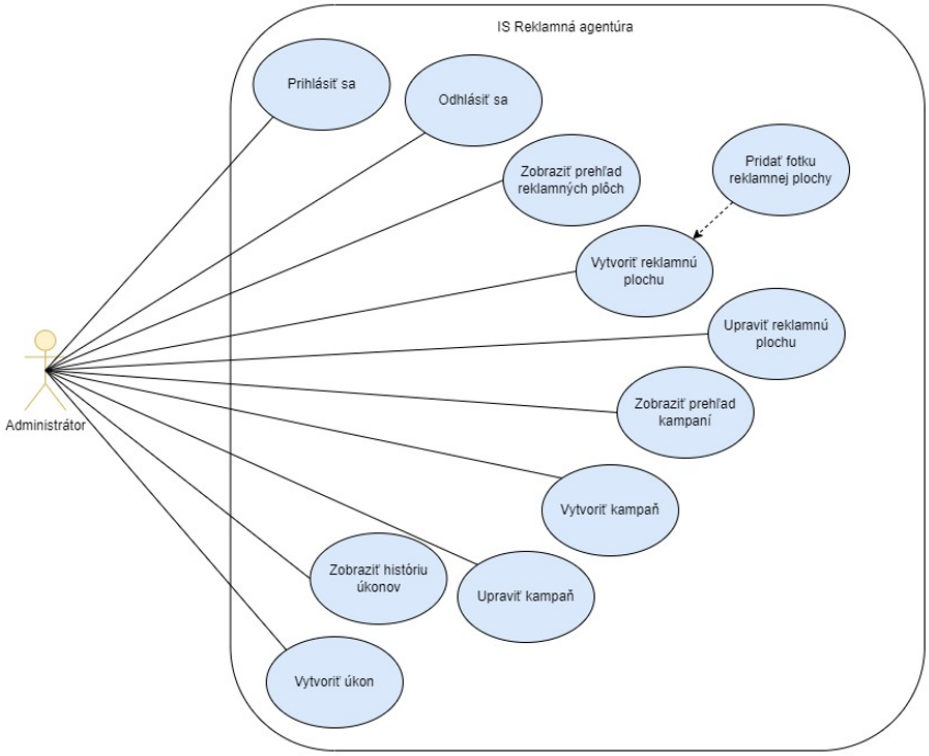
Taktiež bude mať k dispozícii zobrazenie *História úkonov*.

Manažér bude mať možnosť vidieť prehľad kampaní, nebude ju môcť vytvoriť ani meniť jej informácie.

4.3 Diagram prípadov použitia

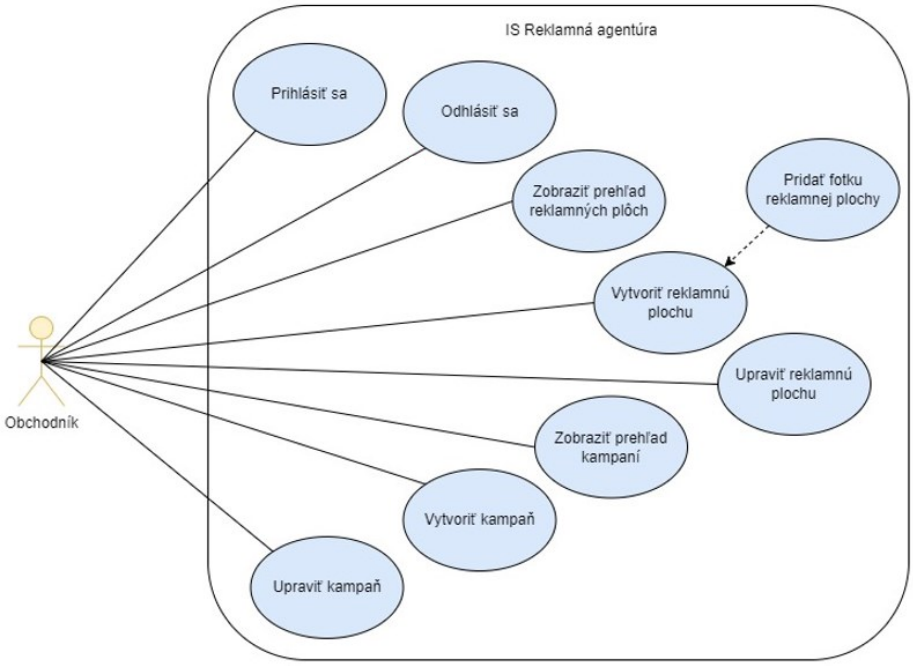
Pre grafické znázornenie spôsobu použitia systému vzhľadom na používateľa použijem diagram prípadov použitia.

4.3.1 Diagram prípadov použitia pre rolu Administrátor



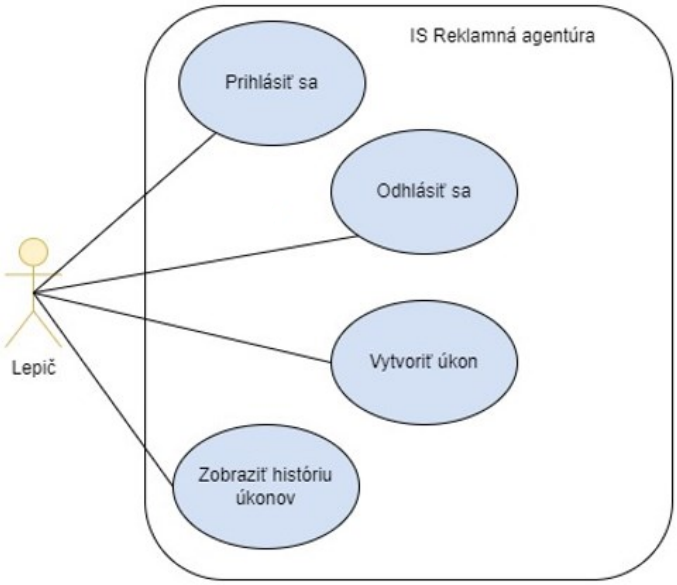
Obrázok 3: Prípad použitia rola Administrátor

4.3.2 Diagram prípadov použitia pre rolu Obchodník



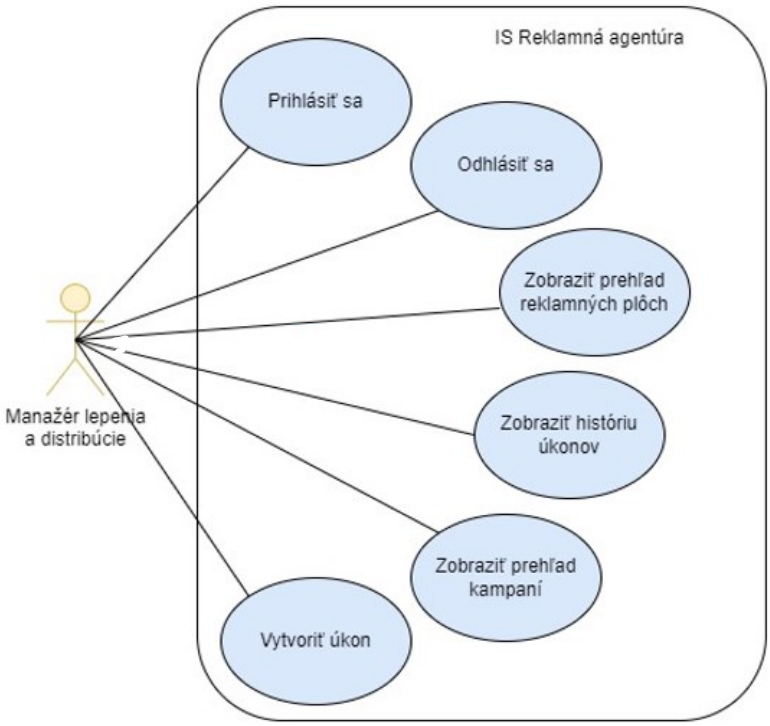
Obrázok 4: Prípad použitia rola Obchodník

4.3.3 Diagram prípadov použitia pre rolu Lepič



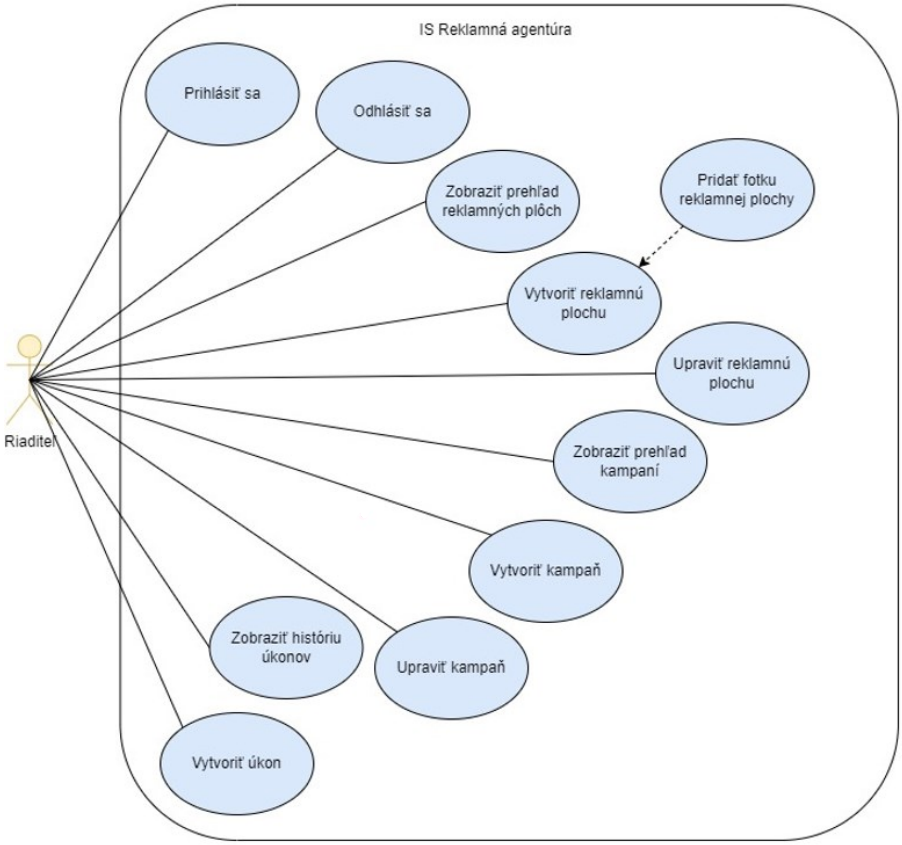
Obrázok 5: Prípad použitia rola Lepič

4.3.4 Diagram prípadov použitia pre rolu Manažér lepenia a distribúcie



Obrázok 6: Prípad použitia rola Manažér lepenia a distribúcie

4.3.5 Diagram prípadov použitia pre rolu Riaditeľ



Obrázok 7: Prípad použitia rola Riaditeľ

4.3.6 Diagram prípadov použitia pre Systém

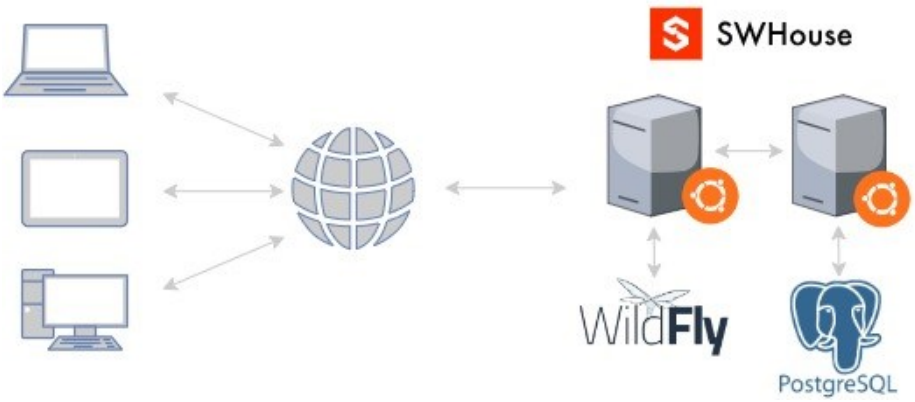
Obrázok 8: Prípad použitia Systém

5 ARCHITEKTÚRA

Na základe analýzy systému som sa pre beh aplikácie rozhodla zvoliť nasledovnú architektúru.

Webová aplikácie bude nasadená na virtuálnom serveri v spoločnosti SWHouse s.r.o.. Spoločnosť vytvorí dva virtuálne servery s operačným systémom Ubuntu, ktoré budú umiestnené v cloude na ich serverovej farme.

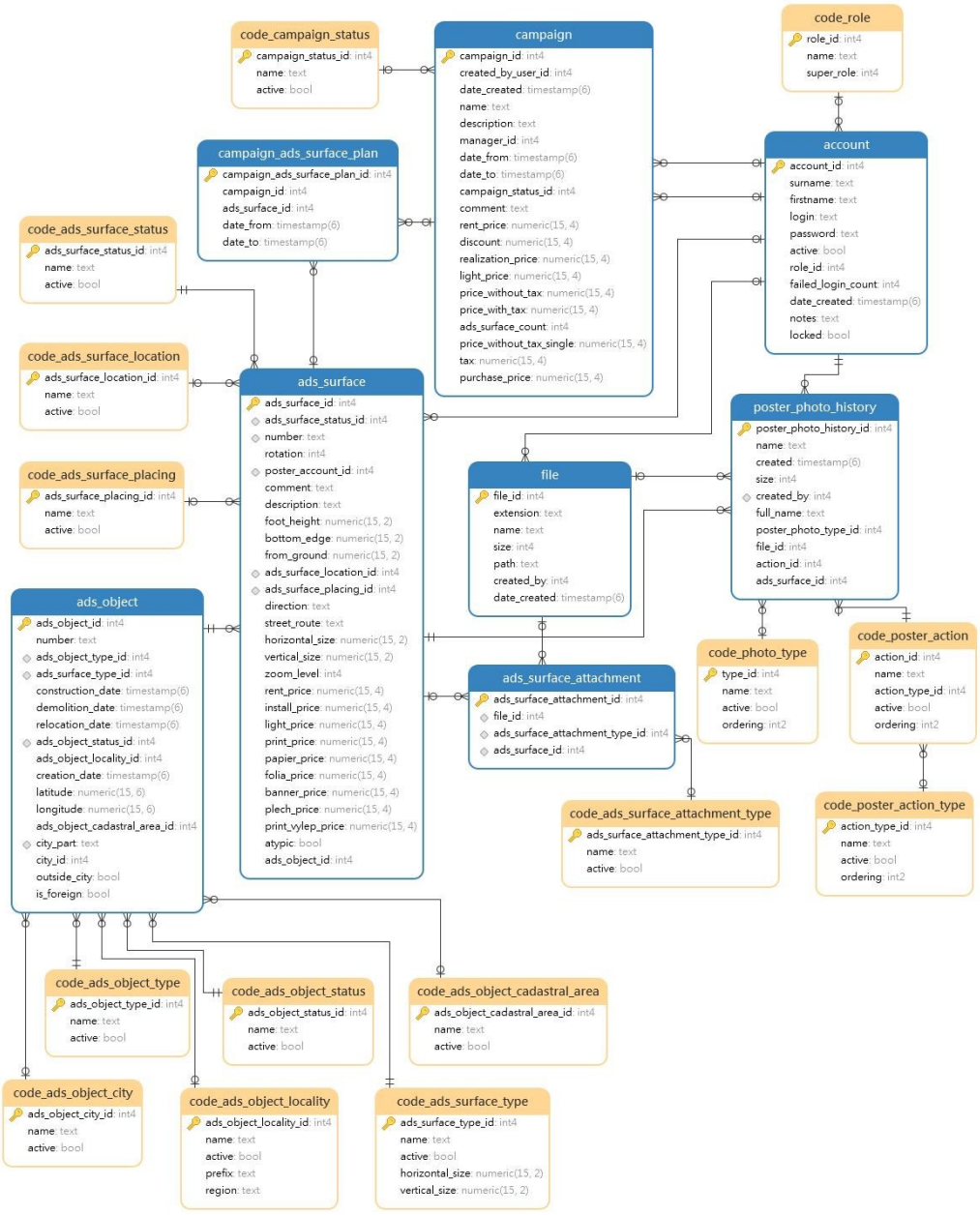
Klientske zariadenie, ktoré bude obsahovať softvér na zobrazovanie webových stránok, vykreslí používateľské rozhranie po zadaní URL adresy v prehliadači. Prehliadač odošle požiadavku cez internet smerujúcu na serverový priestor spoločnosti SWHouse s.r.o.. V spoločnosti bude nastavený reverzný webový server Nginx, ktorý funguje ako proxy a smeruje požiadavky na vopred preddefinovaný virtuálny server s operačným systémom Ubuntu. Na virtuálnom serveri bude spustená služba aplikačného servera WildFly, ktorá bude komunikovať na porte 8080 a prijímať požiadavky. Tieto požiadavky následne spracuje v podobe obslúženia nasadeným zdrojovým kódom v podobe súboru WAR. Aplikačný server bude mať zadefinované databázové pripojenie na databázový server PostgreSQL. Požiadavky obsahujúce CRUD operácie budú toto databázové spojenie využívať na získavanie dát a ich ukladanie, resp. manipuláciu s nimi. Virtuálny server, na ktorom bude spustená služba aplikačného servera, bude nezávislý od virtuálneho servera, kde bude spustený databázový server. Pre zložitosť architektúry neuvádzam všetky prvky v schéme architektúry.



Obrázok 9: Schéma architektúry

6 DATABÁZOVÝ MODEL

Po analýze požiadaviek na systém som vytvorila databázový model, ktorý tvoria potrebné databázové entity slúžiace na perzistenté ukladanie dát. Tabuľky v databázovom modely sú farebne rozlíšené s ohľadom na typ dát, ktoré sa v nich nachádzajú. Modrou farbou sú označené tabuľky pre základné entity, oranžovou farbou sú označené tzv. číselníkové tabuľky. Každá tabuľka má svoj unikátny primárny kľúč a prepojenie medzi tabuľkami je zabezpečené cudzím kľúčom. Na vytvorenie modelu bol použitý nástroj Navicat. Tabuľky boli vytvorené pomocou DDL syntaxe.



Obrázok 10: Databázový model

6.1 Číselníkové tabuľky

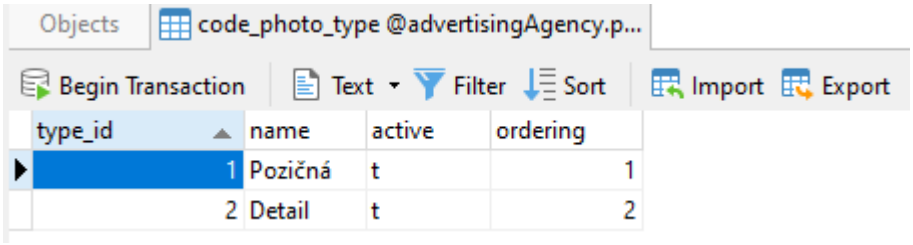
Číselníkové tabuľky obsahujú dáta, ktoré si môžeme predstaviť ako zoznam povolených hodnôt, ktoré môžu nadobúdať atribúty v iných tabuľkách, ktoré sú s nimi prepojené. Okrem farebného rozlíšenia v schéme, každý názov číselníkovej tabuľky začína kľúčovým slovom *code*, pre jednoduchšiu orientáciu.

Každú tabuľku tvoria tieto základné atribúty:

- Primárny kľúč, ktorého názov sa skladá z názvu tabuľky a príponou *id*
- Názov *(name)*, ktorý uchováva textový reťazec
- Príznak (active), slúžiaci na rozlíšenie aktívnosti záznamu

Okrem základných atribútov môžu tabuľky obsahovať dodatočné atribúty ako napríklad *ordering*, ktorý používam na zoradenie záznamov, v prípade ak nie je možné použiť abecedné radenie. Výnimočne sa tu nachádzajú číselníkové tabuľky, ktoré sú cudzím kľúčom prepojené s inými číselníkmi.

Ako príklad uvediem číselníkovú tabuľku *code_photo_type*, ktorá uchováva akého typu môže byť fotografia. Táto tabuľka obsahuje dva záznamy. Prvý záznam pozostáva z primárneho kľúča *type_id* s hodnotou 1, názvu *name* s hodnotou „Pozičná“, príznaku *active* s hodnotou true a atribútu *ordering* s hodnotou 1. Hodnota primárneho kľúča v druhom zázname je 2, názov je „Detail“, príznak *active* je taktiež nastavený na hodnotu true a *ordering* má číslo 2. Z toho vyplýva, že dáta z tejto tabuľky budem poskytovať v poradí „Pozičná“, „Detail“.



type_id	name	active	ordering
1	Pozičná	t	1
2	Detail	t	2

Obrázok 11: Záznamy v tabuľke code_photo_type

6.2 Základné tabuľky

Medzi základné databázové tabuľky v schéme patria: *account*, *ads_object*, *ads_surface*, *ads_surface_attachment*, *campaign*, *campaign_ads_surafce_plan*, *file* a *poster_photo_history*. Tabuľky sa nachádzajú v schéme *public*.

6.2.1 Databázová tabuľka account

Tabuľka slúži na uchovávanie informácií o používateľoch v systéme. Primárnym kľúčom je atribút *account_id* , ktorý slúži ako jedinečný identifikátor záznamu. Ďalšie atribúty sú: meno (*firstname*) a prievisko (*surname*), prihlasovacie meno (*login*), heslo (*password*) uložené vo formáte hash, príznak (*active*), ktorý hovorí či má používateľ prístup do systému, rola (*role_id*), počet po sebe idúcich neúspešných pokusov o prihlásenie (*failed_login_count*) pričom hodnota sa vynuluje po úspešnom prihlásení, dátum vytvorenia záznamu respektíve používateľa (*date_created*), poznámky (*notes*), príznak (*locked*), ktorý je preddefinovaný na hodnotu false. V prípade, že sa používateľ neprihlási úspešne päťkrát po sebe, príznak sa nastaví na hodnotu true, čo znamená, že bude zablokovaný a musí ho manuálne odblokovať správca aplikácie.

Tabuľka 1: Databázová tabuľka account

account			
Atribút	**Dátový typ**	**Kľúč**	
acount_id	int4	PK	NN
surname	text		NN
firstname	text		NN
login	text		NN, UNIQUE
password	text		NN
active	bool		NN
role_id	int4	FK	NN
failed_login_count	int4		
date_created	timestamp(6)		
notes	text		
locked	bool		

6.2.2 Databázová tabuľka ads_object

Všetky informácie o reklamných zariadeniach sú uložené v tabuľke *ads_object*, pričom atribút *ads_object_id* slúži ako primárny kľúč. Ďalšie atribúty sú: označenie reklamného zariadenia (*number*), typ reklamného zariadenia (*ads_object_type_id*), typ reklamnej plochy na zariadení (*ads_surface_type_id*), dátum výstavby (*construction_date*), dátum zbúrania (*demolition_date*), dátum premiestnenia (*relocation_date*), stav reklamného zariadenia (*ads_object_status_id*), lokalita reklamného zariadenia (*ads_object_locality_id*), dátum vytvorenia (*creation_date*), zemepisná šírka (*latitude*), zemepisná dĺžka (*longitude*), katastrálne územie (*ads_ ads_object_cadastral_area_id*), mestská časť (*city_part*), mesto (*city_id*), mimo mesta (*outside_city*), cudzia plocha (*is_foreign*).

Tabuľka 2: Databázová tabuľka ads_object

ads_object			
Atribút	**Dátový typ**	**Kľúč**	
ads_object_id	int4	PK	NN
number	text		NN
ads_object_type_id	int4	FK	NN
ads_surface_type_id	int4	FK	NN
construction_date	timestamp(6)		
demolition_date	timestamp(6)		
relocation_date	timestamp(6)		
ads_object_status_id	int4	FK	NN
ads_object_locality_id	int4	FK	
creation_date	timestamp(6)		
latitude	numeric(15,6)		
longitude	numeric(15,6)		
ads_object_cadastral_area_id	int4	FK	
city_part	text		
city_id	int4	FK	
outside_city	bool		
is_foreign	bool		

6.2.3 Databázová tabuľka ads_surface

Zoznam reklamných plôch uchovávam v tabuľke *ads_surface*. Atribút *ads_surface_id* je primárnym kľúčom v tabuľke. Ďalšie atribúty sú: stav reklamnej plochy (*ads_surface_status_id*), číslo plochy (*number*), rotácia (*rotation*), id lepiča, ktorému plocha patrí (*poster_account_id),* komentár (*comment*), popis (*description*), výška stĺpu na ktorom je plocha umiestnená (*foot_height*), vzdialenosť pravého dolného rohu plochy od zeme (*bottom_edge*), celková výška od zeme (*from_ground*), lokalita (*ads_surface_location_id*), umiestnenie (*ads_surface_placing_id*), smer (*direction*), ulica (*street_route*), šírka plochy (*horizontal_size*), výška plochy (*vertical_size*), veľkosť priblíženia na mape (*zoom_level*), cena za prenájom (*rent_price*), cena za inštalačné práce (*install_price*), cena za osvetlenie plochy (*light_price*), cena za vytlačenie reklamy (*print_price*), cena papiera (*paper_price*), cena za plachtu (*folia_price*), cena za banner (*banner_price*), cena za plech (*plech_price*), cena vylepenia (*print_vylep_price*), príznak atypických rozmerov plochy (*atypic*), id reklamného zariadenia (*ads_object_id*).

Tabuľka 3: Databázová tabuľka ads_surface

ads_surface			
Atribút	**Dátový typ**	**Kľúč**	
ads_surface_id	int4	PK	NN
ads_surface_status_id	int4	FK	NN
number	text		NN
rotation	int4		
poster_account_id	int4	FK	
comment	text		
description	text		
foot_height	numeric(15,2)		
bottom_edge	numeric(15,2)		
from_ground	numeric(15,2)		
ads_surface_location_id	int4	FK	
ads_surface_placing_id	int4	FK	
direction	text		
street_route	text		
horizontal_size	numeric(15,2)		
vertical_size	numeric(15,2)		
zoom_level	int4		
rent_price	numeric(15,4)		
install_price	numeric(15,4)		
light_price	numeric(15,4)		
print_price	numeric(15,4)		
papier_price	numeric(15,4)		
folia_price	numeric(15,4)		
banner_price	numeric(15,4)		
plech_price	numeric(15,4)		
print_vylep_price	numeric(15,4)		
atypic	bool		
ads_object_id	int4	FK	NN

6.2.4 Databázová tabuľka ads_surface_attachment

Zoznam príloh ku reklamným plochám, uchovávam v tabuľke *ads_surface_attachment*. Primárnym kľúčom je *ads_surface_attachment_id*. Ďalej sa tu nachádzajú cudzie kľúče: id súboru (*file_id*), typ prílohy (*ads_surface_attachment_type_id*) a id plochy (*ads_surface_id*).

Tabuľka 4: Databázová tabuľka ads_surface_attachment

ads_surface_attachment			
Atribút	**Dátový typ**	**Kľúč**	
ads_surface_attachment_id	int4	PK	NN
file_id	int4	FK	NN
ads_surface_attachment_type_id	int4	FK	NN
ads_surface_id	int4	FK	NN

6.2.5 Databázová tabuľka campaign

Tabuľka *campaign* uchováva informácie o kampaniach. Každá kampaň má svoj jedinečný primárny kľúč *campaign_id*. Ďalšie atribúty sú: dátum vytvorenia (*date_created*), názov kampane (*name*), popis ku kampani (*description*), id manažéra kampane (*manager_id*), obdobie trvania kampane *date_from* a *date_to*, stav kampane (*campaign_status_id*), komentár (*comment*), cena za prenájom plôch (*rent_price*), zľava (*discount*), cena realizácie (*realization_price*), cena osvetlenia (*light_price*), cena bez DPH (*price_without_tax*), cena s DPH (*price_with_tax*), počet reklamných plôch zaradených do kampane (*ads_surface_count*), priemerná cena plochy v kamapni bez DPH (*price_without_tax_single*), výška DPH (*tax*), zaplatená cena (*purchase_price*).

Tabuľka 5: Databázová tabuľka campaign

campaign			
Atribút	**Dátový typ**	**Kľúč**	
campaign_id	int4	PK	NN
created_by_user_id	int4	FK	
date_created	timestamp(6)		
name	text		
description	text		
manager_id	int4	FK	
date_from	timestamp(6)		
date_to	timestamp(6)		
campaign_status_id	int4	FK	
comment	text		
rent_price	numeric(15,4)		
discount	numeric(15,4)		
realization_price	numeric(15,4)		
light_price	numeric(15,4)		
price_without_tax	numeric(15,4)		
price_with_tax	numeric(15,4)		
ads_surface_count	int4		
price_without_tax_single	numeric(15,4)		

tax	numeric(15,4)		
purchase_price	numeric(15,4)		

6.2.6 Databázová tabuľka campaign_ads_surface_plan

Zoznam reklamných plôch, ktoré sú priradené do kampane je uložený v tabuľke *campaign_ads_surface_plan* s primárnym kľúčom *campaign_ads_surface_plan_id*. Na základe atribútov *camaign_id* a *ads_surface_id* viem určiť, ktorá plocha patrí do ktorej kampane. Atribúty *date_from* a *date_to* určujú odkedy do kedy bude daná reklamná plocha zaradená do kampane.

Tabuľka 6: Databázová tabuľka campaign_ads_surface_plan

campaign_ads_surface_plan			
Atribút	**Dátový typ**	**Kľúč**	
campaign_ads_surface_plan_id	int4	PK	NN
campaign_id	int4	FK	NN
ads_surface_id	int4	FK	NN
date_from	timestamp(6)		
date_to	timestamp(6)		

6.2.7 Databázová tabuľka file

Databázová tabuľka *file* slúži na uchovávanie informácii o súboroch. Primárnym kľúčom je *file_id*. Ďalej potrebujem uchovávať príponu súboru (*extension*), názov súboru (*name*), veľkosť súboru (*size*), cestu na disku kde je súbor uložený (*path*), id používateľa, ktorý súbor vytvoril (*created_by*) a dátum vytvorenia súboru (*date_created*).

Tabuľka 7: Databázová tabuľka file

file			
Atribút	**Dátový typ**	**Kľúč**	
file_id	int4	PK	NN
extension	text		
name	text		
size	int4		
path	text		
created_by	int4	FK	
date_created	timestamp(6)		

6.2.8 Databázová tabuľka poster_photo_history

Všetky úkony realizované na reklamnej ploche uchovávam v tabuľke *poster_photo_history* s primárnym kľúčom *poster_photo_history_id*. Jeden záznam v tabuľke pozostáva z týchto atribútov: názov úkonu (*name*), dátum realizovania (*created*), veľkosť fotky patriacej k úkonu (*size*), id používateľa, ktorý realizoval úkon (*created_by*), celý názov úkonu (*full_name*), ktorý sa skladá z čísla plochy, id reklamnej plochy a názvu typu úkonu, typ fotografie vytvorenej pri realizovaní úkonu (*poster_photo_type_id*), id súboru (*file_id*), v ktorom budú uložené informácie o fotografii, typ úkonu, ktorý bol realizovaný (*action_id*), id plochy (ads_surface_id).

Tabuľka 8: Databázová tabuľka poster_photo_history

poster_photo_history			
Atribút	**Dátový typ**	**Kľúč**	
poster_photo_history_id	int4	PK	NN
name	text		
created	timestamp(6)		
size	int4		
created_by	int4	FK	NN
full_name	text		
poster_photo_type_id	int4	FK	
file_id	int4	FK	
action_id	int4	FK	NN
ads_surface_id	int4	FK	NN

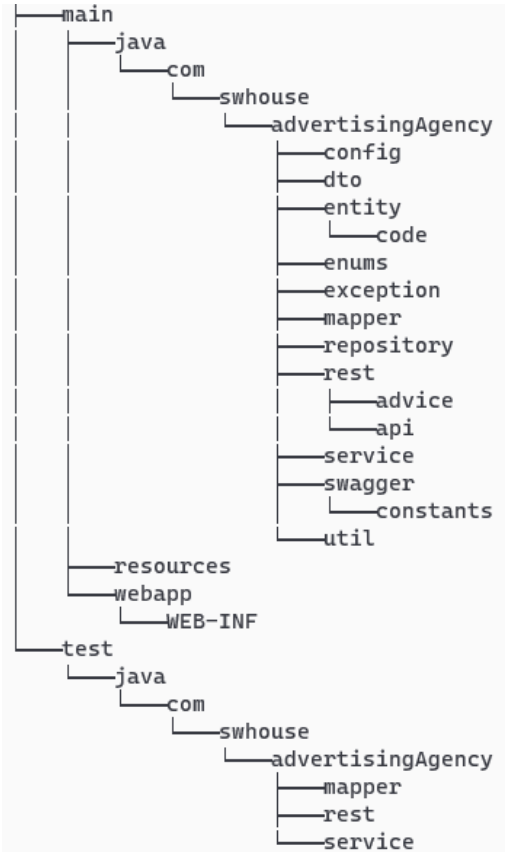
7 IMPLEMENTÁCIA BACKEND

7.1 Štruktúra kódu backend

Vzhľadom na fakt, že správna štruktúra projektu uľahčuje čitateľnosť, rozšírenie ale aj modifikovanie rozhodla som sa kód z pohľadu backendu rozdeliť do niekoľkých priečinkov. Medzi základné priečinky, v ktorých je implementovaná logika aplikácie patria:

- controller
- service
- mapper
- repositroy

Ďalším priečinkom je napríklad priečinok entity, v ktorom sa nachádzajú java triedy, predstavujúce jednotlivé databázové entity, priečinok config, v ktorom je okrem iného nakonfigurovaná autorizácia na aplikačnej vrstve pomocou JWT tokenu.



Obrázok 12: Štruktúra kódu backend

7.2 Controller

Controller je označený anotáciou @RestController a stará sa o spracovanie HTTP požiadaviek a riadenie prístupu k službám. V tomto riešení sa nachádza niekoľko controllerov a každý z nich zabezpečuje spracovanie požiadaviek pre odpovedajúci logický celok.

- *LoginController*
- *PosterController*
- *CodeController*
- *CampaignController*
- *CampaignAdsSurfacePlanController*
- *AdsSurfaceController*

7.2.1 Login Controller

Ako príklad funkcie controlleru uvediem *LoginController*. Controller na základnej ceste */login* publikuje niekoľko endpointov ako napríklad endpoint pre prihlásenie a ednpoint pre získanie údajov a prihlásenom používateľovi. LoginController implementuje metódy LoginControllerApi, ktoré definuje rozhranie API a podporuje generovanie Open API dokumentácie.

```
@Tag(name = "Login controller", description = "Provides endpoints for login")
public interface LoginControllerApi {

    @Operation(
            summary = "Login",
            description = "Login.",
            responses = {
                    @ApiResponse(
                            responseCode = SUCCESS_CODE,
                            description = SUCCESS_DESCRIPTION,
                            content = @Content(schema = @Schema(implementation = LoginResponseDto.class))),
                    @ApiResponse(
                            responseCode = NOT_FOUND_CODE,
                            description = NOT_FOUND_DESCRIPTION,
                            content = @Content(
                                    schema = @Schema(implementation = ErrorResponse.class))),
                    @ApiResponse(
                            responseCode = SERVER_ERROR_CODE,
                            description = SERVER_ERROR_DESCRIPTION,
                            content = @Content(
                                    schema = @Schema(implementation = ErrorResponse.class)))})
    LoginResponseDto login(@NotNull @Valid @RequestBody LoginRequestDto request);
```

Obrázok 13: LoginControllerApi.java

7.2.1.1 Endpoint login

Je publikovaný na ceste */login/authenticate* a zabezpečuje prihlásenie používateľa do systému, respektíve získanie JWT tokenu. Endpoint je typu POST a teda na vstupe očakáva platné *RequestBody* v podobe prihlasovacieho mena a hesla. Požiadavku ďalej smeruje do *LoginService*, ktorá na základe prihlasovacieho mena získa pomocou volania do repozitára informácie a používateľskom účte. V ďalšom kroku overí či bolo zadané správne používateľské heslo a či nie je daný účet zablokovaný. V prípade úspešného overenia, service vygeneruje JWT token pomocou metódy *jwtUtil.generateToken(user)* a do controlleru vracia odpoveď s informáciami o úspešnom prihlásení. V tomto prípade controller vracia túto odpoveď na požiadavku so stavovým kódom 200. V prípade neúspešného overenia hesla, service pomocou repozitára aktualizuje počet neplatných pokusov o prihlásenie a do controllera vracia odpoveď s informáciou o neúspešnom prihlásení. Zároveň v prípade, ak zostávajúci počet pokusov na prihlásenie je menší respektíve rovný nule, postará sa o zablokovanie účtu.

7.2.1.2 Endpoint getLoggedUser

Pre úspešné získanie detailu používateľského účtu predpokladám overenie JWT tokenom, ktorý je úspešne rozpoznaný medzi aktuálne platnými tokenmi. V prípade ak by vypršala platnosť tokenu alebo by išlo o nekorektný token, vrátim chybový HTTP kód.

V prípadne úspešného rozpoznania tokenu sa vracia odpoveď s detailnými informáciami o účte.

```
@SecurityRequirement(name = "jwt-token")
@GetMapping(value = "/logged-account-detail")
public AccountDto getLoggedUser(@Parameter(hidden = true) @RequestHeader("Token") String auth) {

    log.info("Get logged user detail. Authorization {}, JWT-token {}", auth, auth.substring(7));

    if (jwtUtil.extractExpiration(auth.substring(7)).after(new Date(System.currentTimeMillis()))) {
        return loginService.getLoggedAccount(jwtUtil.extractUsername(auth.substring(7)));
    }
    return null;
}
```

Obrázok 14: Endpoint getLoggedUser

7.3 Service

Service alebo služba slúži ako prepojenie medzi controllerom a repozitárom. Hlavným cieľom je oddeliť prezenčnú vrstvu od vrstvy, ktorá ma prístup k dátam. V service je

implementovaná kompletná funkcionalita, čo zabezpečuje prehľadnosť kódu, znižuje duplicitu a chybovosť. Služby, ktoré zabezpečujú funkcionalitu:

- *AdsObjectService*
- *AdsSurfaceService*
- *CampaignAdsSurfacePlanService*
- *CampaignService*
- *CodeService*
- *FileService*
- *JwtUserDetailService*
- *LoginService*
- *PosterService*

7.3.1 Metóda getCampaign

Logika pre získanie zoznamu kampaní je implementovaná v *CampaignService*. Metóda na vstupe očakáva dva parametre, *pageSize*, ktorý určuje veľkosť stránky a *pageIndex*, ktorý určuje koľká strana v zozname sa má vrátiť. Tieto parametre ďalej vstupujú do volania metódy repozitára pre získanie kampaní z databázy. Získané dáta sú potom mapované na návratový typ metódy a vrátené do controlleru, z ktorého prišla požiadavka.

```
public List<CampaignDto> getCampaign(Integer pageSize, Integer pageIndex) {

    return campaignRepository.getCampaign(pageSize, pageIndex) List<Campaign>
                             .stream() Stream<Campaign>
                             .map(campaignMapper::toDto) Stream<CampaignDto>
                             .collect(Collectors.toList());
}
```

Obrázok 15: Metóda getCampaign

7.4 Mapper

Mapper je komponent v aplikácií, ktorý sa stará o mapovanie dát medzi objektami v rôznych vrstvách aplikácie. V mojom riešení existuje mapper, pre každú databázovú entitu. Každý mapper pritom disponuje dvomi metódami. Metóda, *toDto,* stará o mapovanie entity na tzv. DTO objekt, ktorý používam na úrovni controlleru, Opačná metóda, *toEntity,* sa stará o mapovanie z objektu DTO na entitu, ktorá sa používa na úrovni repozitára. Mapovanie prebieha

vždy na úrovni service. Jedna z výhod mapovania je, že odoslané dáta v odpovediach na požiadavky obsahujú len nevyhnutné informácie a predchádzam tak úniku citlivých dát.

```
@Component
public class CampaignAdsSurfacePlanMapper {

    public CampaignAdsSurfacePlanDto toDto(CampaignAdsSurfacePlan entity) {

        CampaignAdsSurfacePlanDto dto = new CampaignAdsSurfacePlanDto();

        dto.setCampaignAdsSurfacePlanId(entity.getCampaignAdsSurfacePlanId());
        dto.setCampaignId(entity.getCampaignId());
        dto.setAdsSurfaceId(entity.getAdsSurfaceId());
        dto.setDateFrom(DateTimeConverter.toZonedDateTime(entity.getDateFrom()));
        dto.setDateTo(DateTimeConverter.toZonedDateTime(entity.getDateTo()));

        return dto;
    }

    public CampaignAdsSurfacePlan toEntity(CampaignAdsSurfacePlanDto dto) {

        CampaignAdsSurfacePlan entity = new CampaignAdsSurfacePlan();

        entity.setCampaignAdsSurfacePlanId(dto.getCampaignAdsSurfacePlanId());
        entity.setCampaignId(dto.getCampaignId());
        entity.setAdsSurfaceId(dto.getAdsSurfaceId());
        entity.setDateFrom(DateTimeConverter.toSqlTimestamp(dto.getDateFrom()));
        entity.setDateTo(DateTimeConverter.toSqlTimestamp(dto.getDateTo()));

        return entity;
    }

}
```

Obrázok 16: Príklad metód mapovania

7.5 Repository

Repozitár je zodpovedný za manipuláciu a prístup k dátam v databáze. Je označený anotáciou @Repository a poskytuje CRUD operácie. Sú to operácie pre vytvorenie, mazanie, editovanie alebo čítanie záznamov z databázy. V tomto prípade pre manipuláciu s dátami v repozitároch využívam DML a DQL príkazy.

```
@Query("SELECT * FROM " + AdsSurface.TABLE_NAME + " " +
       "ORDER BY number DESC " +
       "LIMIT (:inPageSize + 1) " +
       "OFFSET (:inPageIndex * :inPageSize)")
List<AdsSurface> getAdsSurface(Integer inPageSize, Integer inPageIndex);

@Query("INSERT INTO " + AdsSurfaceAttachment.TABLE_NAME + " " +
       "(file_id, ads_surface_attachment_type_id, ads_surface_id ) " +
       "VALUES (:inFileId, :inAdsSurfaceAttachmentTypeId, :inAdsSurfaceId) " +
       "RETURNING ads_surface_attachment_id")
Integer createAdsSurfaceAttachment(Integer inFileId, Integer inAdsSurfaceAttachmentTypeId, Integer inAdsSurfaceId);
```

Obrázok 17: Príklad metód repozitára

8 IMPLEMENTÁCIA FRONTEND

8.1 Štruktúra kódu frontend

Frontend riešenie je rozdelené do niekoľkých logických celkov, ktoré predstavujú základnú kostru. Na tomto rozdelení stavia celá aplikácia a predurčuje ju do budúcnosti ľahšie udržiavateľnú. Základné logické celky predstavujú:

- app.components
- app.shared.backend
- app.shared.entity
- app.shared.guards
- app.shared.modals
- app.shared.services

Okrem toho sa tu nachádzajú aj pomocné triedy ako napríklad triedy v priečinku enum, resolvers alebo preloads.

Obrázok 18: Štruktúra kódu frontend

8.2 Components

V riešení app.components predstavujú jednotlivé časti užívateľského rozhrania, teda obrazovky, s ktorými interaguje používateľ. Každý komponent sa skladá minimálne z dvoch logických celkov a to zo triedy TS a štruktúry HTML. Naviac k týmto dvom môže byť zadefinovaný súbor SCSS, ktorý umožňuje definovať špecifické štýly aplikované na komponent.

8.2.1 Komponent create-history

Obrazovka slúži na vytvorenie záznamu o novom úkone lepičom. Jej implementácia je rozdelená do troch logických celkov a to:

- create-history.component.html
- create-history.component.ts
- create-history.component.scss

V HTML štruktúre sa nachádza kód, ktorý definuje rozloženie jednotlivých komponentov HTML a teda ako bude používateľ interagovať s danou obrazovkou.

create-history.component.html

```
<div class="silk-card silk-card__modal">


  <div class="silk-card-content widget-body">

    <form [formGroup]="form">

      <div class="row">

        <div class="col-12">

          <div class="silk-card-footer">

            <div class="form-group margin-0">

              <div class="row justify-content-left  col-md-12">

                <div class="justify-content-center">
                  <img class="image-size" [src]="url">
                </div>

              </div>

            </div>

          </div>
```

Obrázok 19: Ukážka štruktúry HTML

TypeScript súbor slúži na zadefinovanie triedy a samotnej logiky komponentu. Obsahuje anotáciu @Component, v ktorej definujem prepojenie na HTML a SCSS súbory. Ďalej sa v triede nachádzajú metódy ako napríklad *fillFilteredListOfSurfaces*, v ktorej

implementujem filtrovanie po zadaní textu nad komponentom zobrazujúcim zoznam plôch v procese vypĺňania informácií o novom úkone.

Ďalšou dôležitou metódou je metóda *submit* slúžiaca na spracovanie vložených respektíve vybraných hodnôt na obrazovke a ich odoslanie pomocou rest volania v metóde *createPosterHistory(posterHistory: HistoryDto)*.

```
fillFilteredListOfSurfaces() {

  console.log("fillFilteredListOfSurfaces")

  this.filteredAdsSurfaces = Array.from(this.adsSurfaceData.values()).filter((item : SelectType<number> ) => {
    return item.value.toLowerCase().startsWith(this.filterOfSurface.value.toLowerCase());
  });

  if (this.filteredAdsSurfaces.length === 0 && this.adsSurfaceData.length > 0) {
    const firstItem = this.adsSurfaceData.values().next().value;
    this.filteredAdsSurfaces.push(firstItem);
  }
}
```

Obrázok 20: Metóda fillFilteredListOfSurfaces

Poslednou súčasťou komponentu je súbor, v ktorom definujem štýly. Nie je povinný, ale je možné ho využiť v prípade, ak máme potrebu zadefinovať špecifický štýl danému komponentu mimo globálneho SCSS súboru. SCSS samotné je nadstavbou CSS s niekoľkými doplňujúcimi funkciami, ktoré ho pomáhajú udržiavať jednoduchší na správu.

create-history.component.scss

```
.image-size{
    width: 675px;
    padding-left: 15px;
}
```

Obrázok 21: Ukážka SCSS súboru

8.3 Backend

Táto zložka predstavuje logický celok, ktorý poskytuje integrácie na backend služby. Využívam HTTP rest volania na získanie dát pomocou triedy *api-backend.service.ts*, v ktorej je zadefinovaný HttpClient v konštruktore. Ako príklad uvediem volanie metódy na vytvorenie záznamu o úkone lepiča. Vstupom do metódy *createPosterHistory(posterHistory: HistoryDto)* je objekt DTO s potrebnými atribútmi, ktorý sa posiela v tele požiadavky. Metóda

vykonáva volanie typu POST, na zadefinovanej URL, vyskladanej zo základnej URL a cesty k samotnému endpointu. Návratovou hodnotou je id vytvoreného záznamu.

```
createPosterHistory(posterHistory: HistoryDto): Observable<number> {
  return this.http.post<number>(
    url: `${this.baseUrl}/poster`,
    posterHistory
  );
}
```

Obrázok 22: Volanie createPosterHistory

8.4 Entity

Predstavuje objekty, ktoré sú využívané interne pre logické operácie prebiehajúce počas pracovania s aplikáciou alebo sa jedná o objekty, ktoré využívam pri volaniach backend služieb.

Delím ich na dva typy a to číselníkové objekty, s prefixom Code, využívané primárne pre komponenty typu select a objekty štandardné reprezentujúce reálne objekty v systéme. Ako príklad uvediem číselníkový objekt CodeActionType, ktorý obsahuje atribúty vďaka ktorým viem používateľovi zobraziť možnosť výberu typu úkonu.

code-action-type.ts

```
1  export interface CodeActionType {
2    actionTypeId: number;
3    name: string;
4    active: boolean;
5    ordering: number;
6  }
```

Obrázok 23: Príklad číselníkového objektu

Druhým menovaným typom je objekt DTO, ktorý predstavuje objekt reálneho sveta v tomto prípade úkon lepiča.

history-dto.ts

```
1  export interface HistoryDto {
2    posterPhotoHistoryId: number;
3    name: string;
4    created: string;
5    size: number;
6    createdBy: number;
7    fullName: string;
8    posterPhotoTypeId: number;
9    fileId: number;
10   actionId: number;
11   adsSurfaceId: number;
12   data: string;
13   number:string;
14 }
```

Obrázok 24: Príklad DTO objektu

8.5 Guards

Využíva sa na kontrolu prístupu k samotnej aplikácii, teda rozhraniu po prihlásení sa do aplikácie a kontrolu zobrazených komponentov používateľovi na základe priradenej roly.

Moje riešenie obsahuje dve triedy, ktoré riešia logické celky:

- Kontrola prihlásenia v triede *auth.guard.service.ts*
- Kontrola oprávnení/rola prihláseného používateľa v triede *role-guard.service.ts*

V module smerovania okrem definovania komponentov pre rôzne URL, následne určujem aj potrebu kontroly prihlásenia sa a kontrolu oprávnení, respektíve kontrolu prístupu danej roly na obrazovku.

Ako príklad môžem uviezť smerovanie na obrazovku prehľadu plôch, kde je potrebné, aby mal používateľ priradenú jednu z rolí administrátor, riaditeľ, manažér lepenia a distribúcie alebo obchodný manažér.

```
{
  path: 'ads',
  component:AdssurfaceComponent,
  canActivate: [
    AuthGuardService,
    RoleGuardService,
  ],
  resolve: {
    data: AdsSurfaceResolver,
  },
  data: {
    requiredRoles: [ ClientRoleEnum.ADMINISTRATOR, ClientRoleEnum.RIADITEL, ClientRoleEnum.MANAZER_PRE_VYLEP_A_DISTRIBUCIU, ClientRoleEnum.OBCHODNY_MANAZER],
    path: EPath.ADS,
  },
},
```

Obrázok 25: Smerovanie na obrazovku prehľad plôch

8.6 Modals

Modálne okno je užitočné pre ušetrenie interakcií používateľa a poskytuje účinný nástroj na zjednodušenie používania aplikácie. Vytvorila som tri komponenty modálnych okien a to:

- Pridanie plochy do kampane: campaign-plan-modal.component
- Nahratie fotky: photo-modal.component
- Zobrazenie detailu úkonu lepiča: poster-history-detail-modal.component.ts

V prípade pridávania plochy do kampane je otvorenie modálneho okna volané z obrazovky prehľadu kampaní ale aj z obrazovky na vytvorenie novej kampane. Po stlačení tlačidla na pridanie plochy ku kampani.

```
campaignPlan(row: CampaignOverview): void {

  const matDialogConfig: MatDialogConfig = {
    panelClass: 'small-modal',
    autoFocus: false,
  };
  const matDialogRef: MatDialogRef<CampaignPlanModalComponent> = this.modalService.open(CampaignPlanModalComponent, matDialogConfig);
  matDialogRef.componentInstance.type = TypeOperationEnum.NEW;
  matDialogRef.componentInstance.adsSurfaceDto = this.codeService.adsSurface;
  matDialogRef.componentInstance.campaignId = row.campaignId;

  matDialogRef.afterClosed().subscribe( observerOrNext: (result: boolean) => {
    if (result) {
      this.setPage(0);
    }
  });
}
```

Obrázok 26: Príklad volania otvorenia modálneho okna

8.7 Services

V Angular aplikácii objekt service predstavuje základnú jednotku, ktorá implementuje logiku aplikácie a zdieľa dáta medzi komponentami. Vytvára sa ako singleton, to znamená, že existuje v pamäti iba jeden krát naprieč všetkými časťami aplikácie. Využitím services dochádza k jednoduchšiemu organizovaniu kódu.

Riešenie obsahuje nasledujúce services:

- ads-surface.service.ts: logické operácie s reklamnými plochami a zariadeniami
- app.service.ts: logické operácie na správu základných operácií
- campaign.service.ts.: logické operácie s kampaňami
- client.service.ts: logické operácie o používateľoch
- code.service.ts: logické operácie pre prácu s číselníkmi
- logout.service.ts: manažment odhlásenia používateľa
- poster.service.ts: logické operácie na prácu s históriou lepiča
- token.service.ts: správa prístupových tokenov k BE službám

Dobrým príkladom je CodeService, ktorá v sebe implementuje logiku získania dát CodeActionType na základe actionTypeId. Metóda sa volá napríklad v prípade ak chcem zobraziť detailné informácie o úkone. Objekt, ktorý sa vracia v odpovedi na požiadavku detailu obsahuje len id typu akcie, čo môže byť pri zobrazení ťažko čitateľné. Preto je potrebné toto id preložiť na textovú hodnotu. V tele metódy prehľadávam zoznam typov akcií a na základe vstupného id porovnávam či sa v zozname nachádza. V prípade nájdenia vraciam objekt CodeActionType inak vraciam hodnotu null.

Inými slovami metóda slúži ako prekladač číselných hodnôt na textové reťazce.

```
getActionTypeById(actionTypeId: number | undefined): CodeActionType | null {
  if (actionTypeId) {
    const codeActionTypeId: CodeActionType | undefined = this.codeActionType
      ?.find(actionType => actionType.actionTypeId === actionTypeId);

    if (codeActionTypeId) {
      return codeActionTypeId;
    }
  }
  return null;
}
```

Obrázok 27: Metóda getActionTypeById

9 HAŠOVANIE HESLA

Nakoľko je heslo citlivý údaj a nemalo by sa odosielať vo forme jednoduchého textu, je potrebné zabezpečiť jeho prenos medzi frontendom a backendom.

Pre zabezpečenie som zvolila hašovací algoritmus SHA-256, pretože heslo je týmto algoritmom chránené vďaka tomu, že nie je možné získať jeho pôvodnú podobu.

V procese prihlásenia, po tom ako používateľ zadá heslo do formulára, sa toto heslo spracováva v kóde. V prvom kroku je k heslu pridaný tzv. salt, pomocou ktorého zvýšim bezpečnosť a zamedzím tak prípadnému slovníkovému útoku. Ide o textový reťazec, tvorený veľkými písmenami, malými písmenami a číslami. V druhom kroku nový, vzniknutý, reťazec hašujem algoritmom SHA-256 pomocou JavaScript knižnice *shajs*.

Tento haš je odoslaný v tele požiadavky na prihlásenie. Telo požiadavky je odoslané použitím HTTPS, teda s využitím SSL certifikátu, ktorý znemožní prípadné odchytenie po sieti.

```
login(): void {
  if (this.form?.valid) {
    const loginRequest: LoginRequest = {
      username: this.username?.value,
      password: shajs( algorithm: 'sha256').update( data: this.pwdSalt + this.password?.value).digest( encoding: 'hex'),
    };
```

Obrázok 28: Použitie metódy shajs

10 ZABEZPEČENIE KOMUNIKÁCIE MEZDI FRONTEDOM A BACKENDOM

Na zabezpečenie komunikácie medzi backendom a frontendom som využila výhodu poskytujúcu aplikačným serverom a to Basic autentifikáciu manažovanú priamo na WildFly.

Aplikačný server odchytáva každé volanie a v prípade ak neobsahuje dané volanie v HTTP hlavičke parameter *Authorization* automaticky zamietne toto volanie. K zamietnutiu volania dochádza taktiež v prípade ak sa nezhodujú prístupové údaje v podobe zakódovaného base64 textu predstavujúceho meno aplikačného používateľa a heslo. Na to, aby som vedela použiť Basic autentifikáciu, som potrebovala nakonfigurovať aplikačný server a zároveň urobiť také zmeny v kóde, ktoré budú reflektovať potreby WildFly.

10.1 Pridanie aplikačného používateľa WildFly

Aplikačný server WildFly rozlišuje dva rozdielne používateľské typy a to:

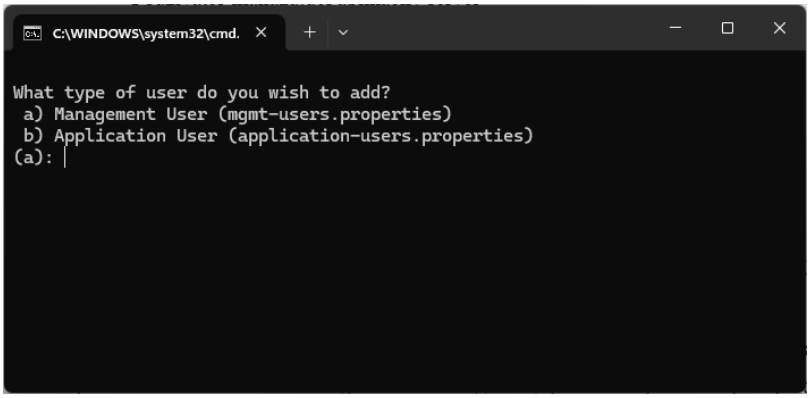
- Používateľ manažujúci aplikačný server
- Aplikačný používateľ (Využívam pre Basic autentifikáciu)

Pridanie aplikačného používateľa vyžaduje nasledovné atribúty:

- Používateľské meno
- Používateľské heslo
- Priradenie do používateľskej skupiny

Takto vytvoreného používateľa následne využijem pre nakonfigurovanie ďalšej časti priamo v kóde aplikácie.

Samotný používateľ sa pridáva dvoma spôsobmi. Prvým je využitie nástroja aplikačného servera tzv. Management konzoly, ktorý je ale komplikovanejší a preto som zvolila pridanie pomocou druhého spôsobu a to cez príkazový riadok a spustenie skriptu *add-user.bat*.



Obrázok 29: Pridanie aplikačného používateľa WildFly

10.2 Konfigurácia backend

Konfigurovanie aplikácie a zabezpečenia je zadefinované pomocou konfiguračného súboru web.xml. V tomto súbore sa nachádza niekoľko definičných značiek a to:

- <security-constraint>: definuje obmedzenie pre všetky publikované endpointy, ktoré v adrese URL obsahujú „/rest/*“. Taktiež je možné zadefinovať typy volaní, ktoré sa kontrolujú. V tomto prípade som zadefinovala mnou používané typy volaní POST, PUT, DELETE, GET a iné
- <auth-constraint>: Špecifikuje roly, ktoré majú prístup k zabezpečeným endpointom na ceste /rest/*. V mojom prípade to je iba rola *advertising*.
- <login-constraint>: Konfigurovanie prístupového mechanizmu voči aplikačnému serveru. Pre toto riešenie som zvolila typ Basic, teda očakávam v HTTP hlavičke parameter Authorization v kódovom formáte base64.
- <security-role>: definuje role, ktoré majú prístup k bezpečnostným obmedzeniam. Pre moje potreby to je taktiež rola *advertising*.

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>All resources</web-resource-name>
        <description>application security constraints
        </description>
        <url-pattern>/rest/*</url-pattern>
        <http-method>DELETE</http-method>
        <http-method>PUT</http-method>
        <http-method>HEAD</http-method>
        <http-method>OPTIONS</http-method>
        <http-method>TRACE</http-method>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>advertising</role-name>
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>ApplicationRealm</realm-name>
</login-config>
<security-role>
    <role-name>advertising</role-name>
</security-role>
```

Obrázok 30: Konfiguračný súbor web.xml

10.3 Konfigurácia frontend

Konfigurácia na strane frontendu prebieha v tzv. interceptore. Pomocou interceptora definujem, čo sa má stať pred odoslaním požiadavky na backend. Vytvorila som vlastný *CustomHttpInterceptor,* ktorý implementuje Angular *HttpInterceptor*, konkrétne povinnú metódu *intercept*. V metóde pridávam autentifikačné HTTP hlavičky ku každej odoslanej požiadavke na backend. Okrem hlavičky s názvom *token*, ktorá obsahuje hodnotu JWT tokenu, tu pridávam taktiež hlavičku s názvom *Authorization*, ktorá zabezpečuje overenie pomocou Basic autentifikácie. Hodnota tejto hlavičky je vyskladaná z kľúčového slova Basic a medzerou oddeleného reťazca vo formáte base64, ktorý v sebe obsahuje informácie o používateľskom mene a hesle.

```
intercept(req: HttpRequest<any>, next: HttpHandler):
  Observable<HttpEvent<any>> {
  if (this.tokenService.isLoggedIn()) {
    req = req.clone( update: {
      headers: new HttpHeaders()
        .append( name: 'token',  value: 'Bearer ' + this.tokenService.token())
        .append( name: 'Authorization',  value: 'Basic ' + btoa( data: 'meno:heslo'))
    });
  } else {
    req = req.clone( update: {
      headers: new HttpHeaders()
        .append( name: 'Authorization',  value: 'Basic ' + btoa( data: 'meno:heslo'))
    });
  }
```

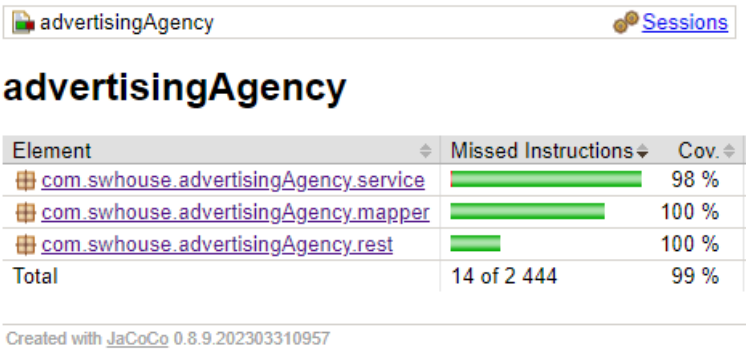
Obrázok 31: Ukážka metódy intercept

11 TESTOVANIE

Testovanie aplikácie prebiehalo v niekoľkých krokoch, pričom v každom kroku som testovala inú časť aplikácie.

11.1 Testovanie backend

V prvom kroku som testovala funkcionalitu kódu na strane backendu pomocou JUnit testov. Testy sú napísané pre úroveň controlleru, service a mappera, kde minimálne požadované pokrytie kódu testami bolo 75 percent. Pre získanie prehľadu o pokrytí som nechala vygenerovať report pomocou pluginu JaCoCo.



Obrázok 32: JaCoCo report

11.1.1 Controller test

Na tejto úrovni testujem schopnosť aplikácie prijímať a odpovedať na požiadavky od klienta. Testy sú napísané pre každý endpoint v controlleri. Ako príklad uvediem test pre endpoint na získavanie dát o ploche na základe id. V tomto teste robím požiadavky typu GET na adresu */rest/adssurface/{adsSurfaceId}*, kde adsSurfaceId nahrádzam náhodne vygenerovaným číslom. Ako odpoveď očakávam objekt AdsSurfaceDto, ktorého hodnota parametra adsSurfaceId bude zhodná s číslom, ktoré som poslala v požiadavke. Okrem toho očakávam návratový status kód 200.

```java
@Test
void getAdsSurfaceById() throws Exception {

    Integer id = Instancio.create(Integer.class);
    AdsSurfaceDto adsSurfaceDto = Instancio.create(AdsSurfaceDto.class);
    adsSurfaceDto.setAdsSurfaceId(id);

    when(adsSurfaceService.getAdsSurfaceById(id)).thenReturn(adsSurfaceDto);

    mockMvc.perform(get( urlTemplate: "/rest/adssurface/{adsSurfaceId}", id)
                            .header( name: "Token",  ...values: "Bearer " + jwtToken))
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$.adsSurfaceId", IsEqual.equalTo(id)));
}
```

Obrázok 33: JUnit test na úrovni controlleru

11.1.2 Service test

Správnosť jednotlivých funkcionalít, logických operácií a manipuláciu s dátami overujem na úrovni service testov. V prípade testu pre funkcionalitu na vytvorenie kampane overujem, či sa práve jedenkrát vykonala metóda v repozitári na vloženie záznamu o kampani do databázy. Taktiež overujem že návratová hodnota z metódy v tomto prípade Id novo vytvoreného záznamu nie je prázdna.

```java
@Test
void createCampaign() {

    CampaignDto campaignDto = Instancio.create(CampaignDto.class);
    AccountDto accountDto = Instancio.create(AccountDto.class);
    Campaign campaign = Instancio.create(Campaign.class);

    when(mapper.toEntity(campaignDto)).thenReturn(campaign);

    Integer result = service.createCampaign(campaignDto, accountDto);

    verify(repository, times( wantedNumberOfInvocations: 1))
            .createCampaign(accountDto.getAccountId(), campaign.getDateCreated(), campaign.getName(),
                    campaign.getDescription(), campaign.getManagerId(), campaign.getDateFrom(),
                    campaign.getDateTo(), campaign.getCampaignStatusId(), campaign.getComment(),
                    campaign.getRentPrice(), campaign.getDiscount(), campaign.getRealizationPrice(),
                    campaign.getLightPrice(), campaign.getPriceWithoutTax(), campaign.getPriceWithTax(),
                    campaign.getAdsSurfaceCount(), campaign.getPriceWithoutTaxSingle(), campaign.getTax(),
                    campaign.getPurchasePrice());

    assertNotNull(result);

}
```

Obrázok 34: JUnit test na úrovni service

11.1.3 Mapper test

JUnit testy pre úroveň mapper overujú správne fungovanie mapovania medzi objektami entity a DTO. Pri takýchto testoch porovnávam, či je hodnota atribútu v pôvodnom objekte,

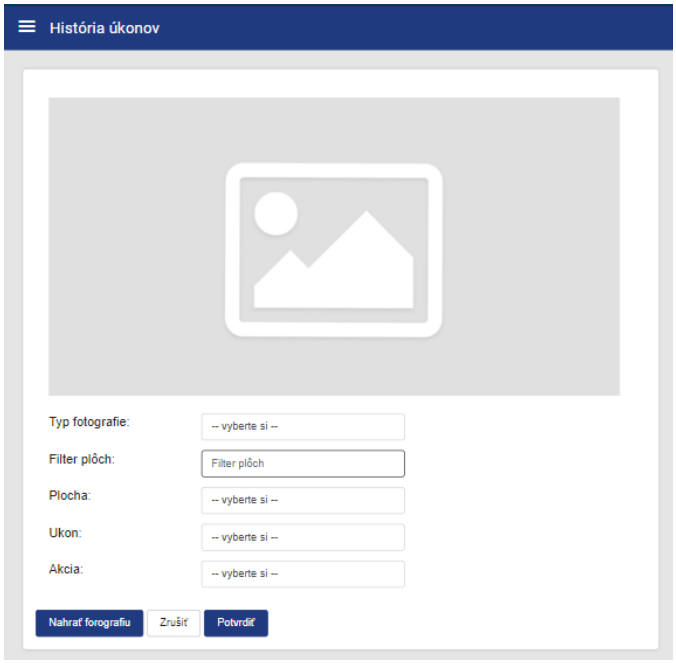
rovná hodnote atribútu v objekte vytvorenom mapperom. Respektíve či sa nová hodnota rovná očakávanej hodnote.

```java
@Test
void toEntity() {

    AccountDto dto = Instancio.create(AccountDto.class);

    Account entity = mapper.toEntity(dto);

    assertEquals(entity.getAccountId(), dto.getAccountId());
    assertEquals(entity.getSurname(), dto.getSurname());
    assertEquals(entity.getFirstname(), dto.getFirstname());
    assertEquals(entity.getLogin(), dto.getLogin());
    assertEquals(entity.getPassword(), dto.getPassword());
    assertEquals(entity.getActive(), dto.getActive());
    assertEquals(entity.getRoleId(), dto.getRoleId());
    assertEquals(entity.getFailedLoginCount(), dto.getFailedLoginCount());
    assertEquals(entity.getNotes(), dto.getNotes());
    assertEquals(entity.getLocked(), dto.getLocked());

}
```

Obrázok 35: JUnit test na úrovni mapper

11.2 Testovanie frontend

V druhom kroku som testovala aplikáciu z pohľadu frontendu. Pre toto testovanie som zvolila tzv. manuálne testovanie. Okrem vizuálnej stránky som otestovala správnu navigáciu medzi obrazovkami a taktiež funkčnosť jednotlivých komponentov ako sú napríklad tlačidlá. Pri testovaní som sa do aplikácie prihlásila pod používateľmi s rôznymi rolami, aby som overila správnosť nastavenia oprávnení pre jednotlivé zobrazenia a úkony na obrazovkách.



Obrázok 36: Testovanie zobrazenia na tablete

12 VÝSLEDKY PRÁCE

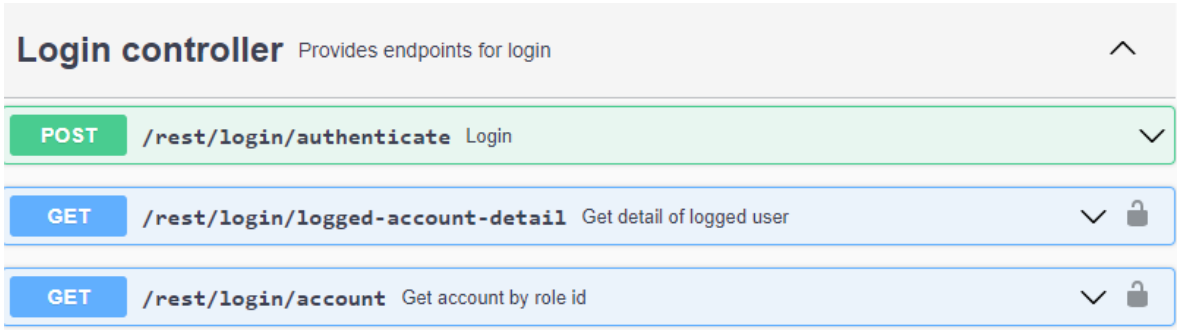
Výsledok práce odprezentujem na obrázkoch, ktoré znázorňujú implementované funkcionality a vykonané činnosti pre dosiahnutie cieľa.

12.1 Backend

Z pohľadu backendu je výsledkom práce REST API, ktoré je rozdelené do niekoľkých controllerov na základe logických celkov. Každý controller obsahuje implementované endpointy s funkcionalitami, ktoré spĺňajú požiadavky klienta. Neoddeliteľnou súčasťou je prezentácia vytvorených databázových tabuliek s testovacími dátami. Taktiež som vytvorila kolekciu v programe Postman, ktorú je možné zdieľať medzi developermi.

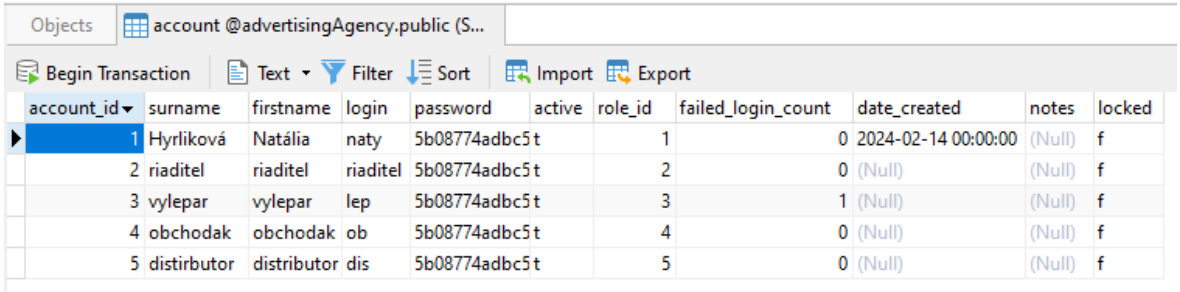
12.1.1 Login controller

Endpointy, ktoré sú publikované Login controllerom sú zobrazené na obrázku nižšie.



Obrázok 37: Swagger: LoginController

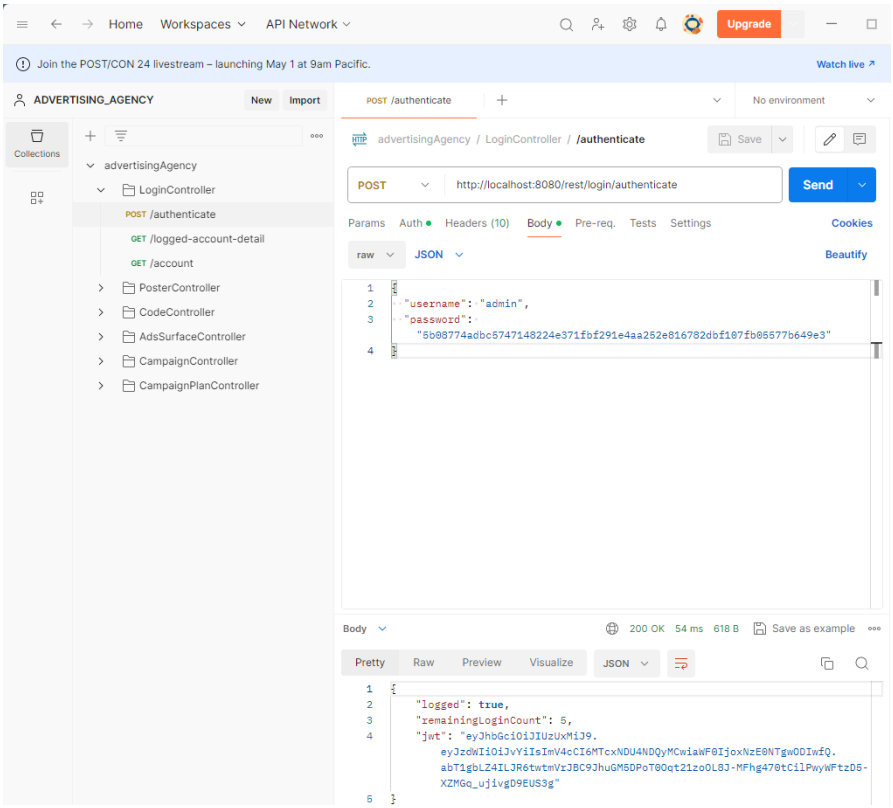
V databázovej tabuľke account, sú vytvorení používatelia, pričom každý používateľ má priradenú inú používateľskú rolu.



account_id	surname	firstname	login	password	active	role_id	failed_login_count	date_created	notes	locked
1	Hyrliková	Natália	naty	5b08774adbc5	t	1	0	2024-02-14 00:00:00	(Null)	f
2	riaditel	riaditel	riaditel	5b08774adbc5	t	2	0	(Null)	(Null)	f
3	vylepar	vylepar	lep	5b08774adbc5	t	3	1	(Null)	(Null)	f
4	obchodak	obchodak	ob	5b08774adbc5	t	4	0	(Null)	(Null)	f
5	distirbutor	distributor	dis	5b08774adbc5	t	5	0	(Null)	(Null)	f

Obrázok 38: Dáta v tabuľke account

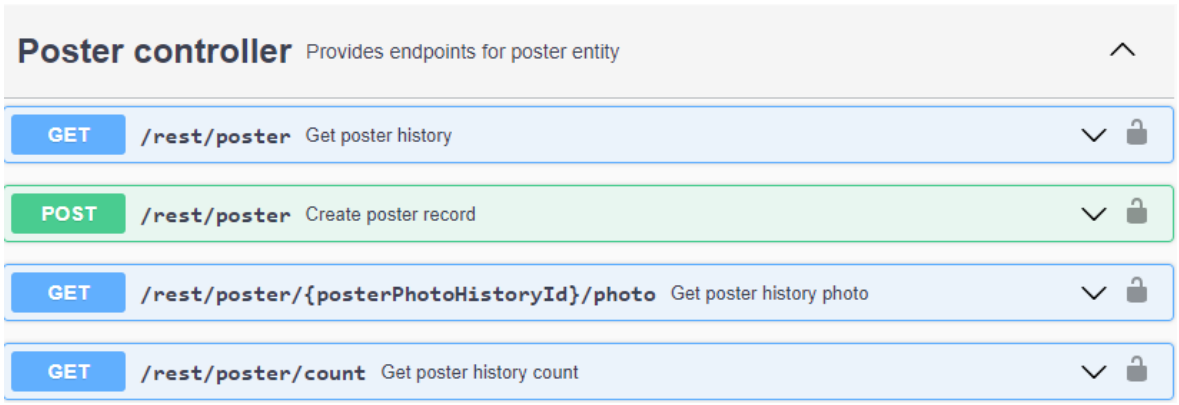
Volanie endpointu pre prihlásenie v aplikácii Postman obsahuje v tele požiadavky prihlasovacie údaje. Ako odpoveď je vrátený okrem iného aj JWT token.



Obrázok 39: Postman: endpoint POST /authenticate

12.1.2 Poster controller

Poster controller sa stará o požiadavky, ktoré sa týkajú úkonov lepenia. Na obrázku nižšie je zobrazený kompletný zoznam endpointov poskytovaných týmto controllerom.



Obrázok 40: Swagger: PosterController

Dáta, ktoré sú odpoveďou na volanie z controlleru sú uložené v databázovej tabuľke poster_photo_history.

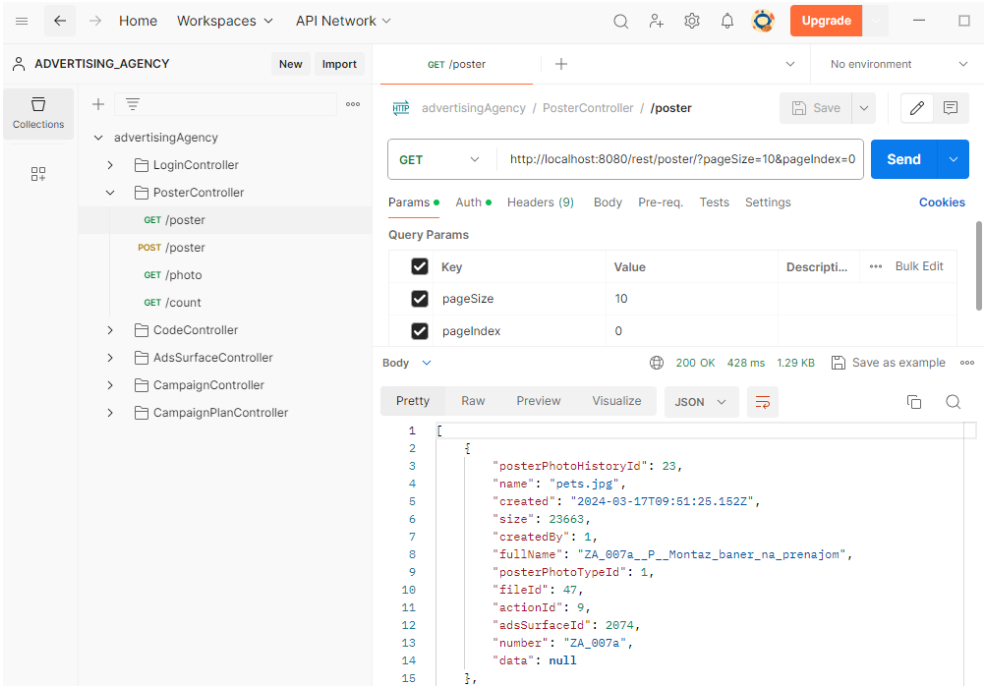
Objects | poster_photo_history @advertisingAgen...

Begin Transaction | Text | Filter | Sort | Import | Export

poster_photo_history_id	name	created	size	created_by	full_name	poster_photo_type_id	file_id	action_id	ads_surface_id
20	wallpaper.jpg	2024-03-13 14:05:48.087	275193	1	KE_004a__P__Vy	1	42	16	1345
21	pets.jpg	2024-03-13 14:06:29.728	23663	1	KE_002a__D__Kc	2	43	14	4349
22	wallpaper.jpg	2024-03-14 16:55:16.663	275193	3	BA_107a__P__Fc	1	46	6	2640
23	pets.jpg	2024-03-17 09:51:25.152	23663	1	ZA_007a__P__M	1	47	9	2074

Obrázok 41: Dáta v tabuľke poster_photo_history

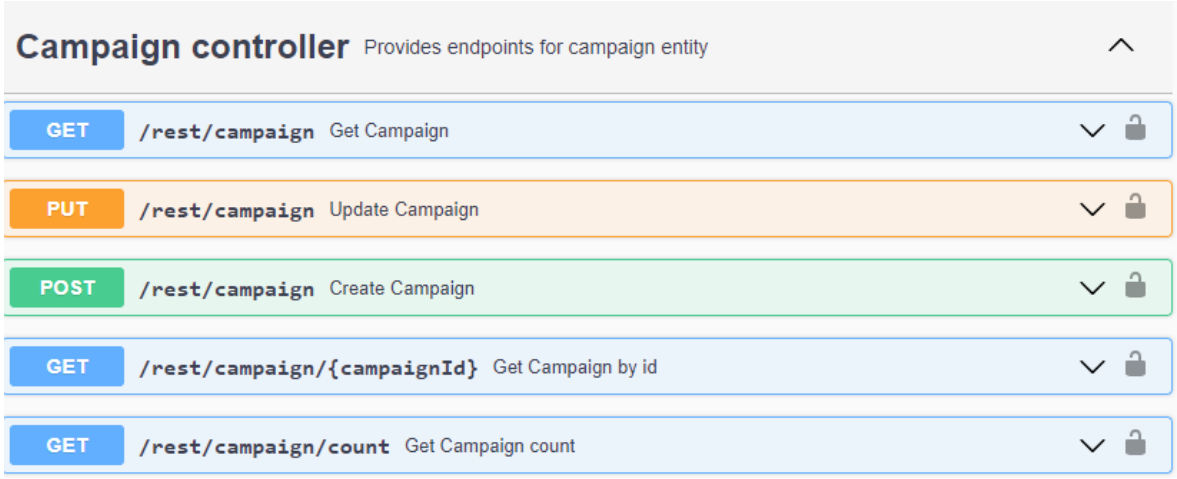
V kolekcii aplikácie Postman je vytvorený priečinok s názvom PosterController, kde sú pripravené všetky volania, patriace do tejto kategórie.



Obrázok 42: Postman: endpoint GET /poster

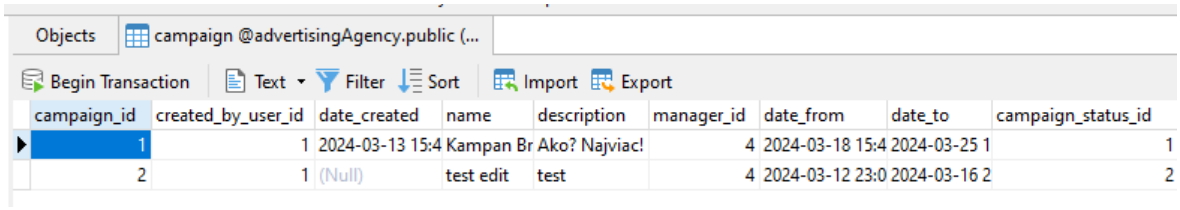
12.1.3 Campaign controller

O správu kampaní sa stará Campaign Controller, v ktorom sa nachádzajú endpointy napríklad pre vytvorenie kampane, úpravu kampane a získanie informácií o danej kampani na základe id.



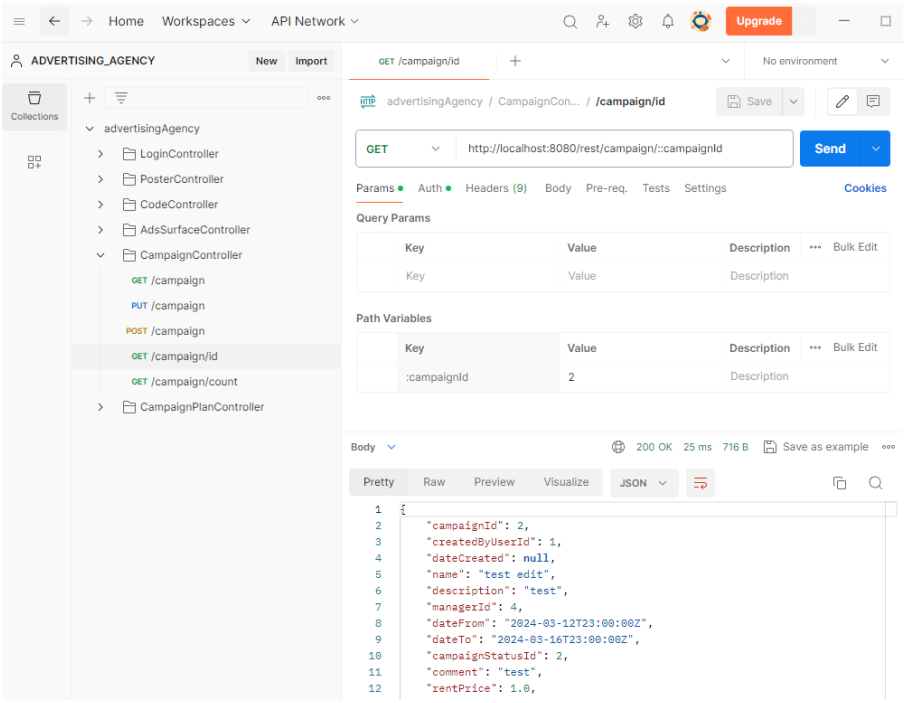
Obrázok 43: Swagger: CampaignController

Tabuľka s názvom campaign ukladá dáta o kampaniach zobrazených v systéme.



Obrázok 44: Dáta v tabuľke campaign

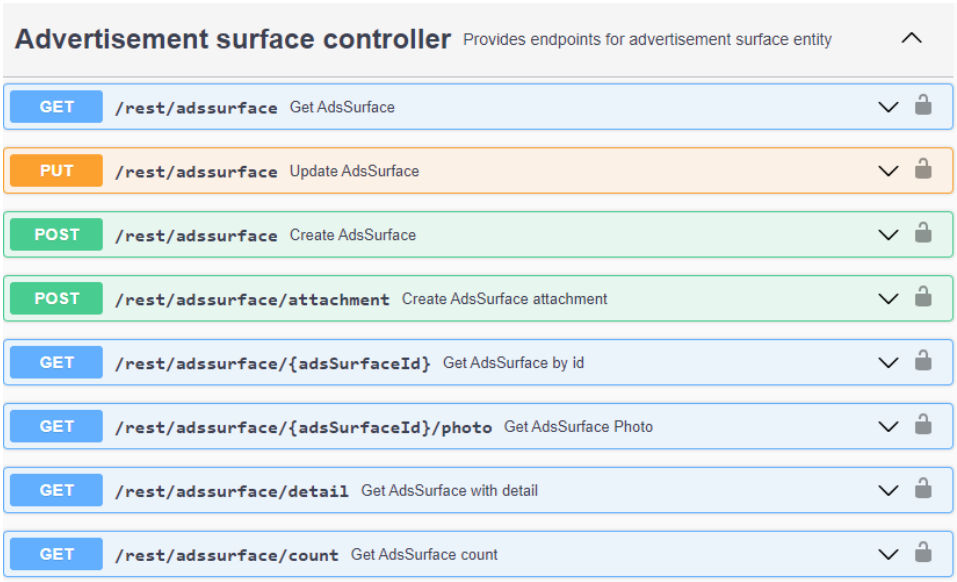
Dáta o konkrétnej kampani je možné získať pomocou volania typu GET. Príklad tohto volania v aplikácii Postman je zobrazený na obrázku nižšie.



Obrázok 45: Postman: endpoint GET /campaign/id

12.1.4 AdsSurface controller

Tento controller slúži na získanie informácií evidovaných pre reklamnú plochu. Základnými operáciami sú získanie prehľadu plôch, vytvorenie novej reklamnej plochy a editácia informácií o už existujúcej ploche.



Obrázok 46: Swagger: AdsSurfaceController

Databázová tabuľka ads_surface je vytvorená pre potreby uchovávania informácií o reklamnej ploche. Reprezentuje základnú entitu v aplikácii.

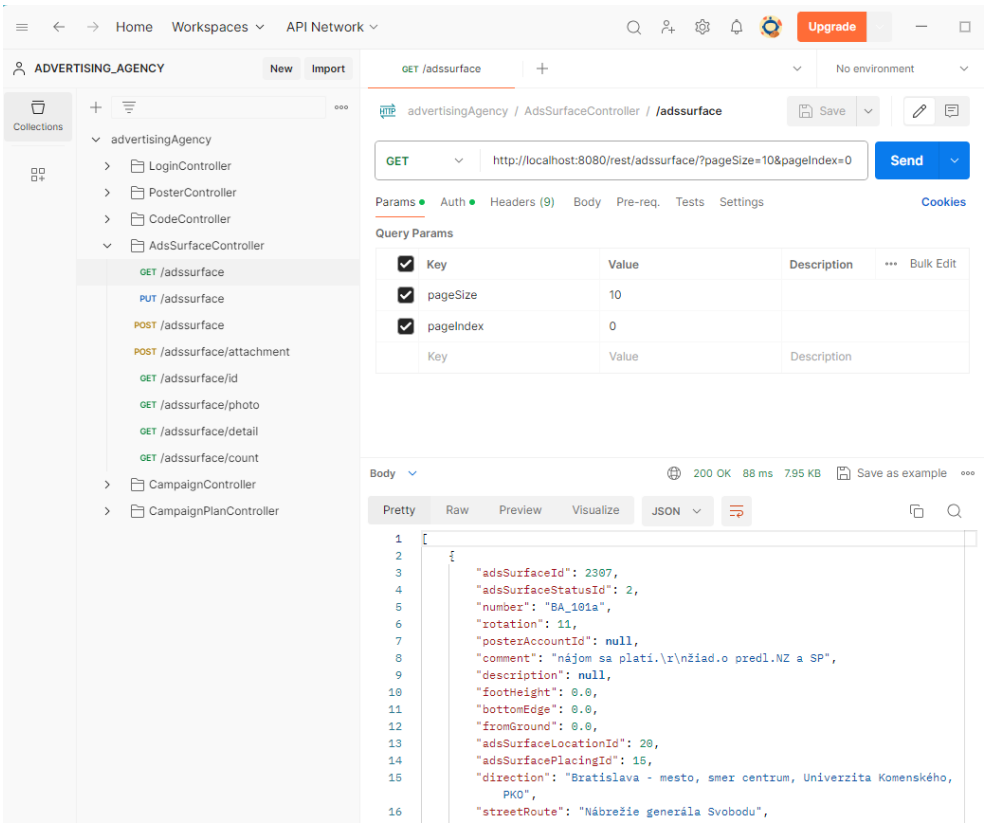
Objects | ads_surface @advertisingAgency.public ...

Begin Transaction | Text | Filter | Sort | Import | Export

ads_surface_id	ads_surface_status_id	number	rotation	poster_account_id	comment
1072	2	ZA_001a	4	3	nájom sa platínová NZ oc
1292	2	KE_003a	11	3	nová NZ od 15.3.2006náj
1322	2	ZA_002a	4	3	nájom sa platínová NZ oc
1323	3	ZA_003a	1	3	nová NZ od 1.1.2011 na d
1336	2	ZA_005a	1	3	nová NZ od 1.1.2011 na d

Obrázok 47: Dáta v tabuľke ads_surface

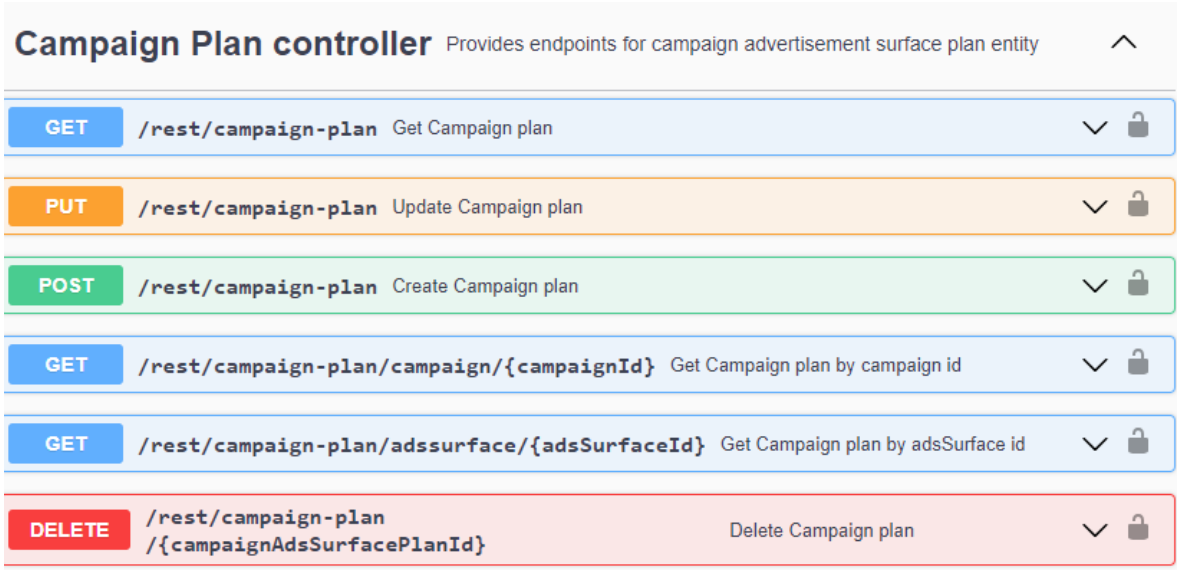
Pripravené volania na prácu s entitou ads_surface sa nachádzajú v kolekcii ADVETISING_AGENCY a sú typu GET, POST, PUT.



Obrázok 48: Swagger: endpoint GET /adssurface

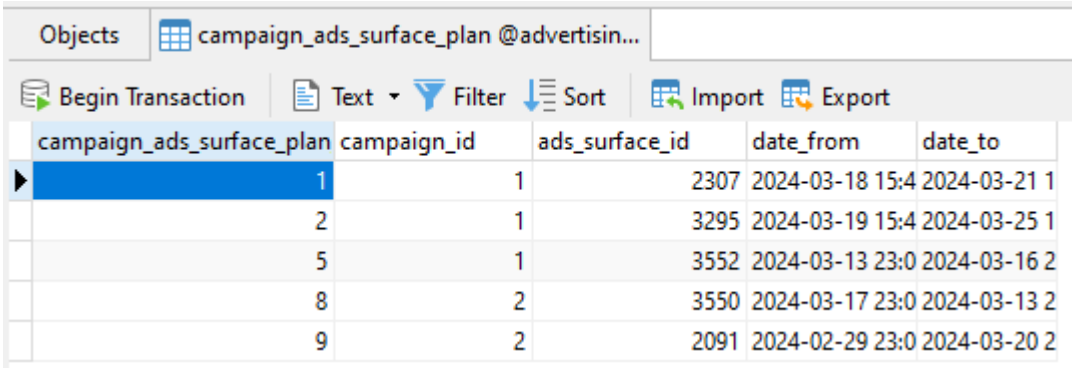
12.1.5 CampaignPlan controller

Pre každú plochu je možné evidovať informáciu o zaradení do kampane. Tieto informácie získavam pomocou publikovaných endpointov v rámci Campaign Plan Controlleru.



Obrázok 49: Swagger: CampaignPlanController

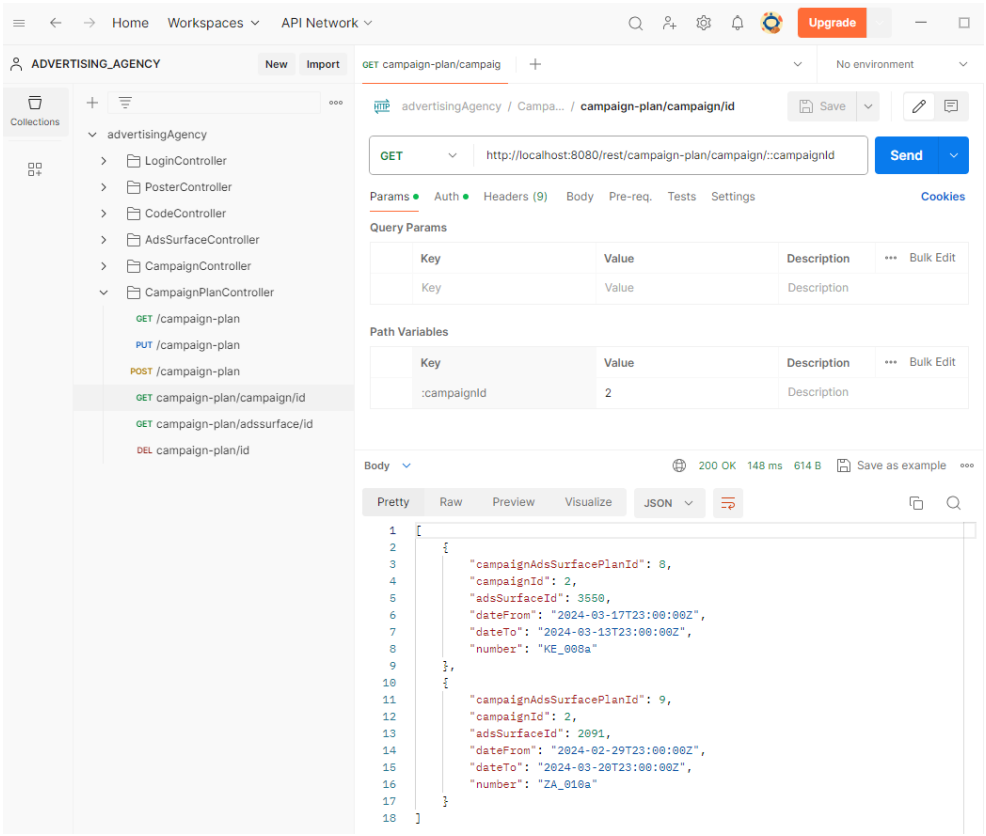
Evidencia informácií o zaradení plochy do kampane je realizovaná formou databázovej tabuľky campaign_ads_surface_plan.



campaign_ads_surface_plan	campaign_id	ads_surface_id	date_from	date_to
1	1	2307	2024-03-18 15:4	2024-03-21 1
2	1	3295	2024-03-19 15:4	2024-03-25 1
5	1	3552	2024-03-13 23:0	2024-03-16 2
8	2	3550	2024-03-17 23:0	2024-03-13 2
9	2	2091	2024-02-29 23:0	2024-03-20 2

Obrázok 50: Dáta v tabuľke campaign_ads_surface_plan

Pre prácu s volaniami v controlleri Campaign Plan som vytvorila ukážkové požiadavky v aplikácii Postman.



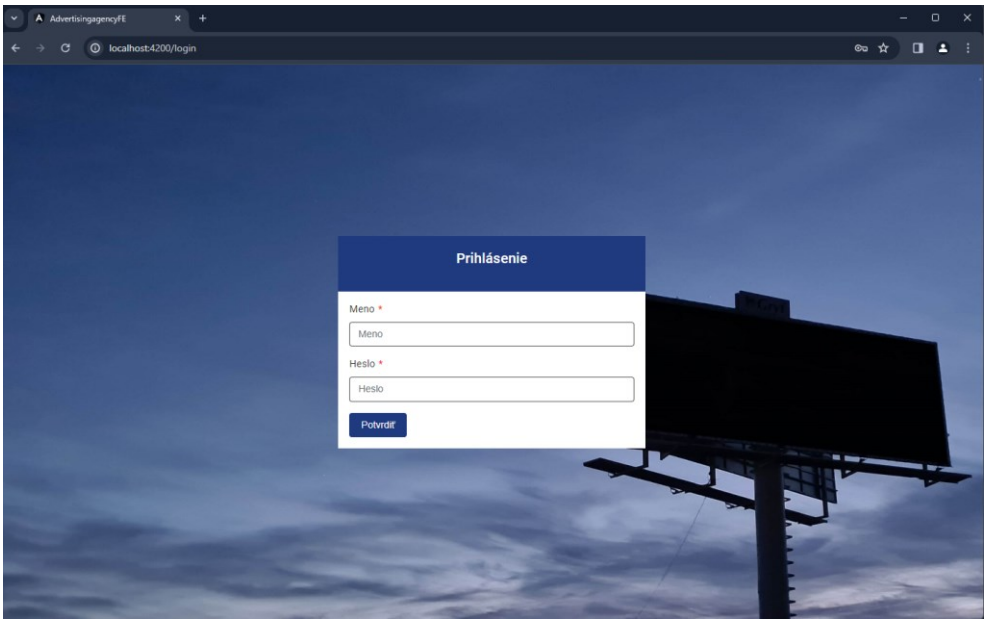
Obrázok 51: Postman: endpoint GET /campaign-plan/campaign/id

12.2 Frontend

Frontend reprezentujú jednotlivé obrazovky implementované podľa odpovedajúcich logických celkov. Kontrola prístupu je riadená formou prihlásenia do aplikácie a dostupná funkcionalita je riadená na základe používateľskej roly prihláseného používateľa.

12.2.1 Prihlásenie

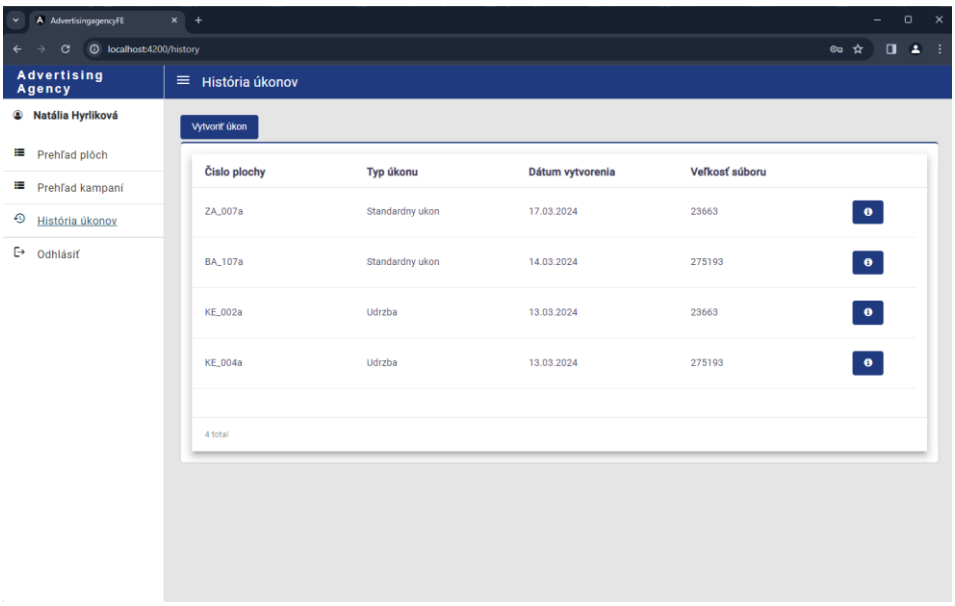
Obrazovka pre prihlásenie pozostáva z jedného hlavného komponentu, do ktorého používateľ zadáva svoje prihlasovacie údaje. Následne sú prihlasovacie údaje odoslané prostredníctvom kliknutia na tlačidlo potvrdiť.



Obrázok 52: Obrazovka prihlásenie

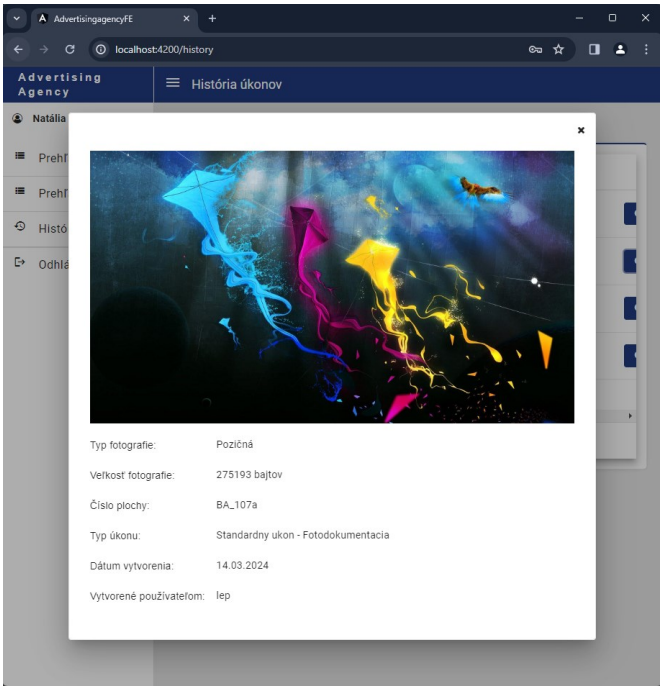
12.2.2 História úkonov

Dominantným prvkom tejto obrazovky je zoznám vykonaných úkonov s informáciami ako je číslo plochy, typ úkonu, dátum vytvorenia a veľkosť súboru. Taktiež sa na obrazovke nachádza tlačidlo pre zobrazenie detailu o úkone a tlačidlo pre vytvorenie nového úkonu.



Po kliknutí na tlačidlo pre zobrazenie detailu úkonu, sa zobrazí modálne okno, v ktorom je zobrazená aj fotografia úkonu.

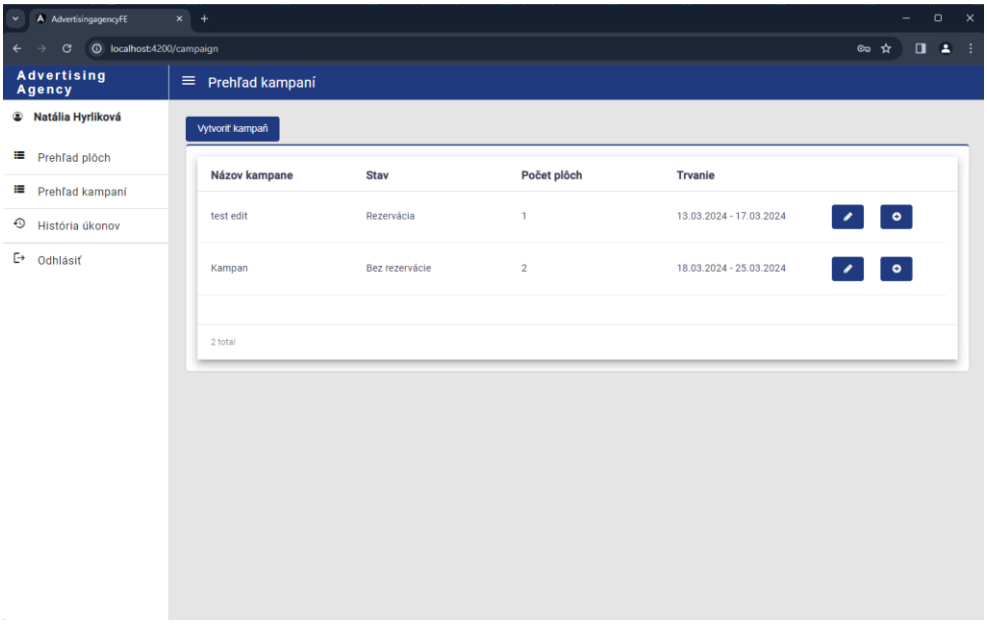
Obrázok 53: Obrazovka História úkonov



Obrázok 54: Modálne okno detail úkonu

12.2.3 Prehľad kampaní

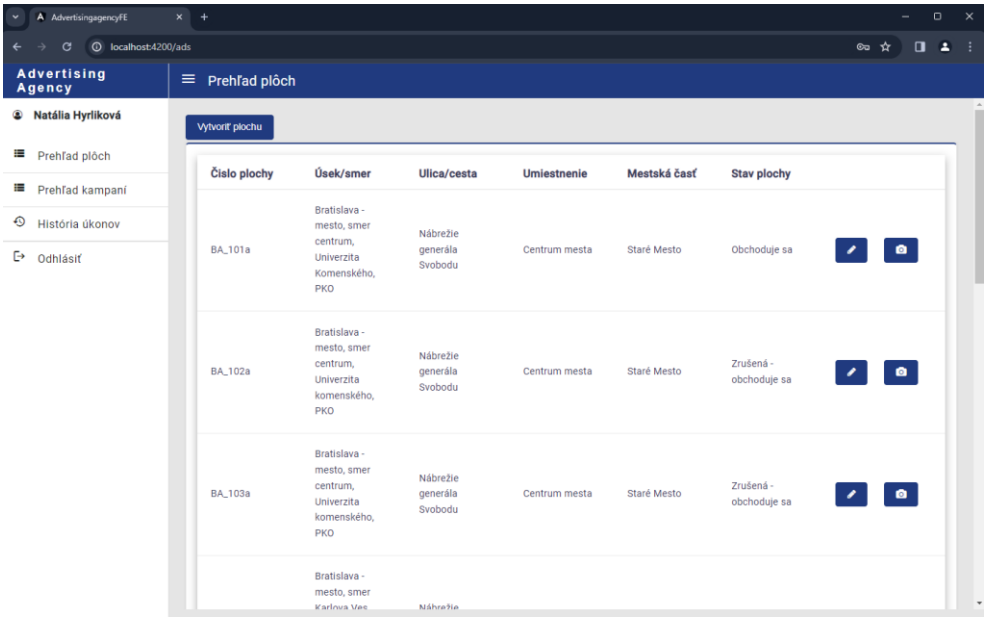
Obrazovka je tvorená zoznamom kampaní a tlačidlami pre vytvorenie novej kampane, editáciu kampane a pridanie reklamnej plochy ku kampani.



Obrázok 55: Obrazovka prehľad kampaní

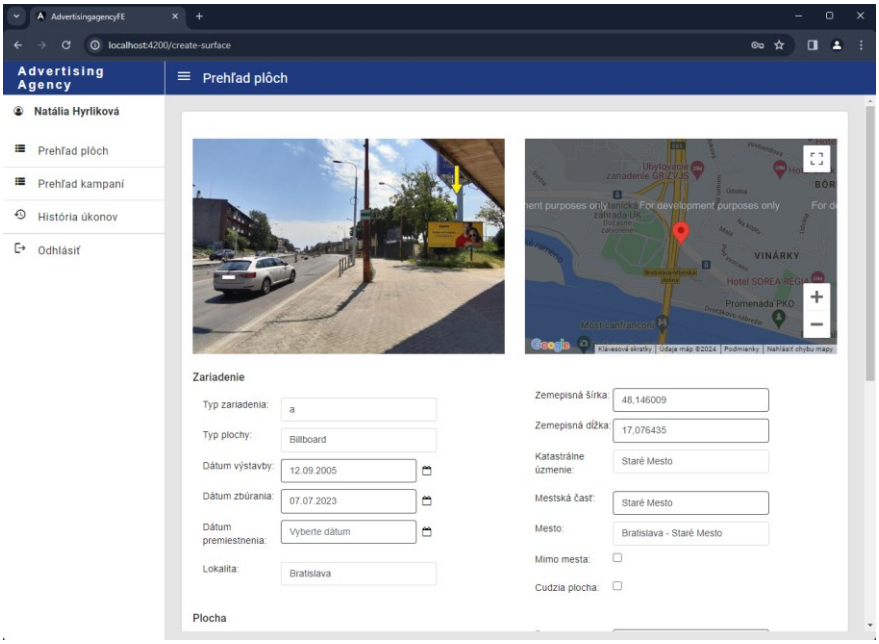
12.2.4 Prehľad plôch

Okrem zoznamu plôch sa na obrazovke nachádzajú aj tlačidla pre vytvorenie plochy, editáciu, respektíve zobrazenie detailu a tlačidlo pre nahratie fotografie k reklamnej ploche.



Obrázok 56: Obrazovka prehľad plôch

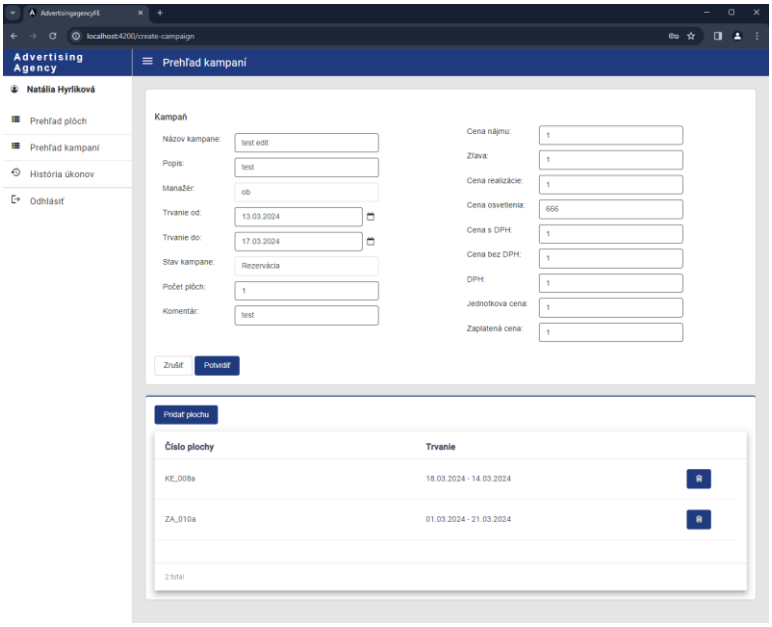
Po kliknutí na tlačidlo editácie sa zobrazí obrazovka s detailom reklamnej plochy, s možnosťou editácie jednotlivých atribútov. Taktiež je tu zobrazená fotografia plochy a mapa s vyznačeným miestom, na ktorom sa plocha nachádza.



Obrázok 57: Obrazovka úprava a detail plochy

12.2.5 Plán kampane

Obrazovka poskytuje pohľad na detailné informácie o kampani spolu so zoznamom reklamných plôch zaradených do tejto kampane. Jednotlivé plochy môžu byť odstránene z kampane pomocou tlačidla vedľa každej plochy. Na zaradenie plochy ku kampani je možné použi tlačidlo *Pridať plochu*.

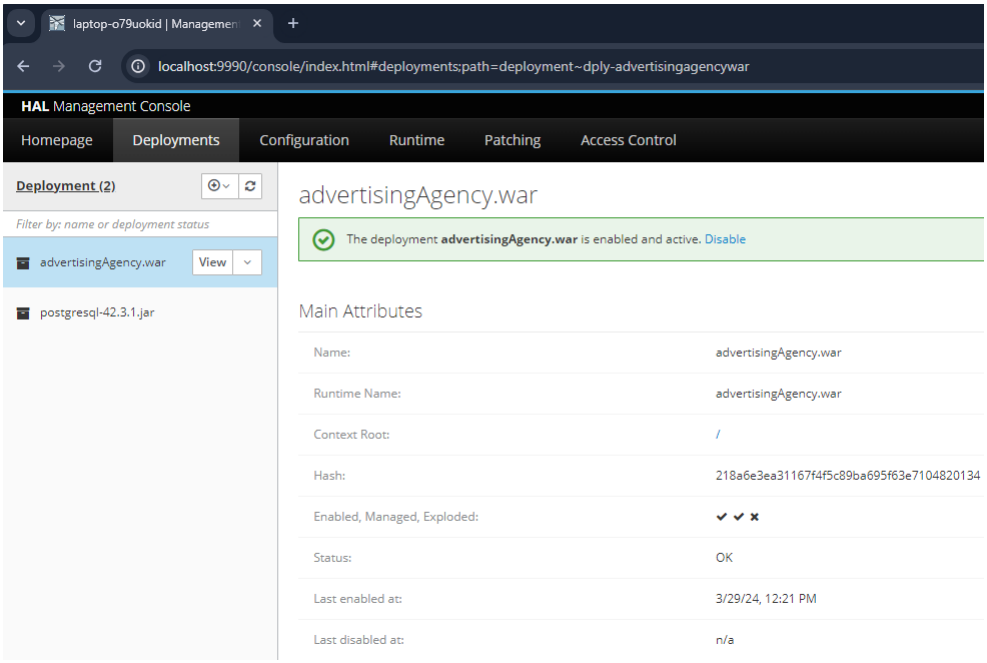


Obrázok 58: Obrazovka plán kampane

12.3 Prevádzka na aplikačnom serveri

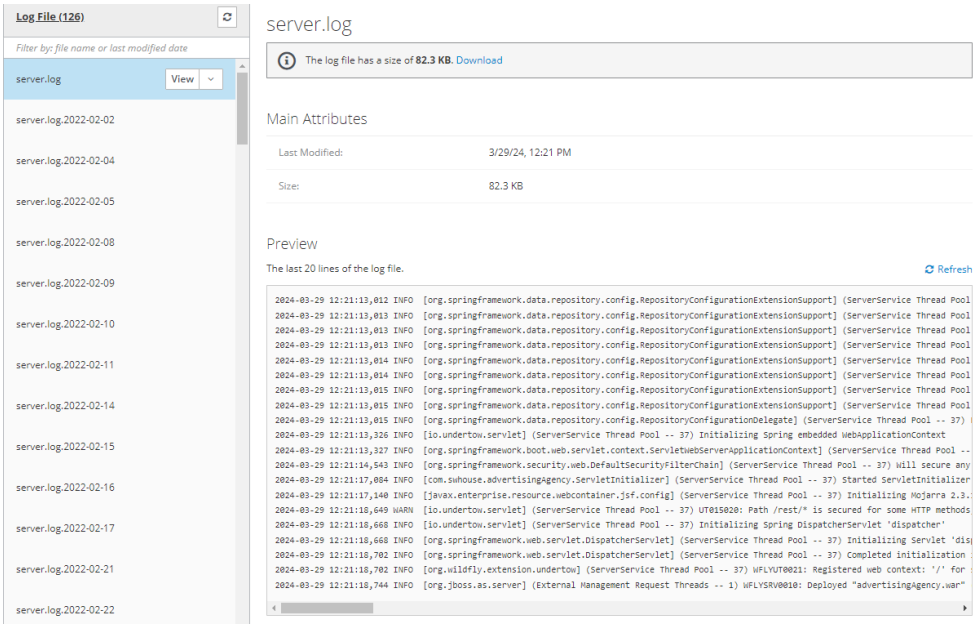
Zvolený aplikačný server WildFly JBoss hosťuje skompilovaný kód, frontend a backend časti, pomocou jedného súboru typu WAR, ktorý je použitý v procese nasadzovania.

12.3.1 Nasadený WAR súbor



Obrázok 59: WildFly deployments

12.3.2 Serverové logy



Obrázok 60: Wildfly logy

ZÁVER

Mojou úlohou bolo vytvoriť informačný systém pre reklamnú agentúru, ktorý bude jednoduchý a intuitívny pre používateľov.

Na začiatok som vykonala detailnú analýzu požiadaviek klienta a upresnila si predstavu čo by mal informačný systém ponúkať v prvej fáze implementácie. Tieto poznatky som postupne využívala v jednotlivých krokoch implementácie a navrhla som technológie, ktoré používam naprieč projektom. Navrhla som databázový model v technológii PostgreSQL, ktorý som aplikovala v mnou vytvorenej databáze. Následne som si urobila inicializáciu backend projektu v Java Spring Boot frameworku a rozdelila si logické objekty do controllerov. V priebehu implementácie backend endpoitov som sa napojila na databázu, z ktorej získavam dáta. Výstupom týchto korkov je ucelený backend, ktorý slúžil ako predpoklad pre začatie vývoja frontend časti.

Súčasťou práce bol aj návrh obrazoviek, ktorý spĺňa UX a UI trendy. Dôraz je kladený na praktické rozmiestnenie komponentov na jednotlivých obrazovkách vďaka čomu je vytvorený informačný systém prehľadný a jednoduchý na používanie najmä na zariadeniach akými sú tablet alebo stolný počítač resp. notebook. Obrazovky boli akceptované klientom na spoločných sedeniach pri kontrole rozpracovanosti.

Počas celého procesu implementácie som udržiavanie verzií kódu vykonávala pomocou nástroja GitLab, vďaka ktorému som bola schopná udržiavať prehľadnosť vykonaných zmien a implementovaných funkcionalít.

V minulosti som už prišla do praktickej skúsenosti s technológiou Java Spring Boot a PostgreSQL, avšak vývoj frontend časti som doteraz nepraktizovala. Samotná implementácia v technológii Angular, navrhnutie štruktúry projekt a ozrejmenie si štandardov používaných počas vývoja pre mňa predstavovalo tú najväčšiu výzvu. Zároveň je to aj časť, na ktorú som najviac pyšná pretože som sa naučila používať nový framework. Nadobudla som veľa nových poznatkov, ktoré v budúcnosti zúročím na ďalších projektoch.

Do budúcna predpokladám, po dohode s klientom, pokračovanie v ďalších implementačných fázach a v rozvoji funkcionalít, ako napríklad evidencia objednávok, spracovanie štatistík a evidencia klientov.

Aktuálny stav spĺňa všetky požiadavky klienta a taktiež prebehla akceptácia diela na základe akceptačných testov vykonaných klientom.

ZOZNAM POUŽITEJ LITERATÚRY

[1] OBS. Informačný systém – Čo to je a aké výhody vám priniesie. Online. OBS. 2024. Dostupné z: https://obs.sk/blog/co-je-to-informacny-system-a-ake-vyhody-vam-dokaze-priniest. [cit. 2024-04-30].

[2] KODAS. KIS - Informačný systém KODAS. Online. KODAS. 2024. Dostupné z: http://kis.kodas.sk/KIS/oKise.aspx. [cit. 2024-04-30].

[3] DREVEŇÁK, MIROSLAV. SPOZNAJTE PODNIKOVÝ INFORMAČNÝ SYSTÉM ONIX. Online. KROS. 2016. Dostupné z: https://www.kros.sk/blog/spoznajte-podnikovy-informacny-system-onix/. [cit. 2024-04-30].

[4] QUANTUMS TECHNOLOGIES S.R.O. Čo je Ubuntu server? Online. SYSMART. 2024. Dostupné z: https://sysmart.sk/slovnik-pojmov/ubuntu-server.html. [cit. 2024-03-29].

[5] DAMI DEVELOPMENT S.R.O. Nginx. Online. DAMI. 2024. Dostupné z: https://www.damidev.com/slovnik/nginx. [cit. 2024-03-29].

[6] WILDFLY. About the WildFly Project. Online. WildFly. 2024. Dostupné z: https://www.wildfly.org/about/. [cit. 2024-03-29].

[7] BOS, Spencer. Java Basics: What Is Wildfly? Online. JRebel. 2020. Dostupné z: https://www.jrebel.com/blog/wildfly. [cit. 2024-03-29].

[8] MARCHIONI, F. Configuring HTTP Basic Authentication with WildFly. Online. Mastertheboss. 2019. Dostupné z: https://www.mastertheboss.com/jbossas/jboss-security/configuring-http-basic-authentication-with-wildfly/?utm_content=cmp-true. [cit. 2024-03-29].

[9] POSTGRESQL. About. Online. PostgreSQL. 2024. Dostupné z: https://www.postgresql.org/about/. [cit. 2024-03-29].

[10] WIKIPÉDIA. PostgreSQL. Online. Wikipédia. 2024. Dostupné z: https://sk.wikipedia.org/wiki/PostgreSQL. [cit. 2024-03-29].

[11] MICROSOFT. What is Java Spring Boot? Online. Microsoft – cloud, počítače, aplikácie a hry. ©2024. Dostupné z: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-java-spring-boot. [cit. 2024-03-29].

[12] KANAI, Shinji. JUnit: A Complete Guide. Online. Headspin. 2022. Dostupné z: https://www.headspin.io/blog/junit-a-complete-guide. [cit. 2024-03-29].

[13] REDDY, Seeni Lathasree. What is JUnit and How we use JUnit Testing. Online. Medium. 2019. Dostupné z: https://medium.com/@lathasreeseeni/junit-2d9857773e8. [cit. 2024-03-29].

[14] HANÁK, Drahomír. Stopárov sprievodca REST API. Online. Itnetwork.sk. ©2024. Dostupné z: https://www.itnetwork.sk/programovani/nezaradene/stoparov-sprievodca-rest-api. [cit. 2024-03-29].

[15] AU-YEUNG, John a DONOVAN, Ryan. Best practices for REST API design. Online. Stack Overflow. 2020. Dostupné z: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/. [cit. 2024-03-29].

[16] KOJNOKOVÁ, Michaela. REST API: Kľúč k efektívnej komunikácii pri vývoji softvéru. Online. MsgLife. 2023. Dostupné z: https://msgtester.sk/rest-api/. [cit. 2024-03-29].

[17] RED HAT. What is a REST API? Online. RedHat. 2020. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api. [cit. 2024-03-29].

[18] MEDIUM. What are GET, POST, PUT, PATCH, DELETE? A walkthrough with JavaScript's Fetch API. Online. Medium. 2019. Dostupné z: https://medium.com/@9cv9official/what-are-get-post-put-patch-delete-a-walkthrough-with-javascripts-fetch-api-17be31755d28. [cit. 2024-03-29].

[19] GOOGLE. Definícia algoritmu SHA 256. Online. Google Ads Pomocník. ©2024. Dostupné z: https://support.google.com/google-ads/answer/9004655?hl=sk. [cit. 2024-03-29].

[20] ZECHMEISTER, Jindřich. Co znamená SHA otisk a šifrování o 256 bitech? Online. SSL Market. 2018. Dostupné z: https://www.sslmarket.sk/blog/co-znamena-otisk-o-256-bitech. [cit. 2024-03-29].

[21] WIKIPÉDIA. Secure Hash Algorithm. Online. Wikipédia. 2022. Dostupné z: https://cs.wikipedia.org/wiki/Secure_Hash_Algorithm. [cit. 2024-03-29].

[22] THEASTROLOGYPAGE. Čo je pripojenie k databáze Java (jdbc)? - definícia z technológie. Online. Theastrologypage. 2024. Dostupné z: https://sk.theastrologypage.com/java-database-connectivity. [cit. 2024-03-29].

[23] POSTMAN. About Postman. Online. Postman. 2024. Dostupné z: https://www.postman.com/company/about-postman/. [cit. 2024-03-29].

[24] MÁCA, Jindřich. Úvod do Angular. Online. Itnetwork.sk. 2024. Dostupné z: https://www.itnetwork.sk/javascript/angular/zaklady/uvod-do-angular-frameworku. [cit. 2024-03-29].

[25] VISIBILITY. Bootstrap. Online. Visibility. 2024. Dostupné z: https://visibility.sk/blog/slovnik/bootstrap/. [cit. 2024-03-29].

[26] TECHFOCUS. CO JE TO GITLAB A K ČEMU SLOUŽÍ? Online. TechFocus. 2022. Dostupné z: https://techfocus.cz/komercni-sdeleni/4655-co-je-to-gitlab-a-k-cemu-slouzi.html. [cit. 2024-03-29].

[27] GITLAB. 10 reasons why enterprises choose GitLab. Online. GitLab. ©2024. Dostupné z: https://about.gitlab.com/why-gitlab/. [cit. 2024-03-29].

[28] A., Damián. Intellij IDEA, nainštalujte toto IDE na vývoj s Java z PPA. Online. Ubunlog. 2024. Dostupné z: https://ubunlog.com/sk/intellij-idea-idea-java/. [cit. 2024-03-29].

[29] NAVICAT. Navicat. Online. Navicat. 2024. Dostupné z: https://navicat.com/en/products. [cit. 2024-03-29].

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

API	Application Programming Interface
CBO	Chief Business Officer
CEO	Chief Executive Officer.
CICD	Continuous Integration Continuous Deployment
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DML	Data Manipulation Language
DPH	Daň z Pridanej Hodnoty
DQL	Data Query Language
DTO	Data Transfer Object
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
JWT	JSON Web Token
REST	Representational State Transfer
SCSS	Sassy CSS
SHA	Secure Hash Algorithm
SQL	Structured Query Language
TS	TypeScript
UI	User Interface

URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User Experience
WAR	Web Application Archive
XML	eXtensible Markup Language

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM PRÍLOH

PRÍLOHA P I: OBSAH CD

Štruktúra obsahu CD:

- fulltext.pdf – text diplomovej práce vo formáte PDF
- prilohy.zip – prílohy k diplomovej práci
    - zdrojoveKody.zip – komprimované zdrojové kódy aplikácie
    - advertisingAgency.postman_collection.json – súbor pre importovanie kolekcie pre Postman aplikáciu
    - dbinit.sql – súbor pre inicializáciu databázy