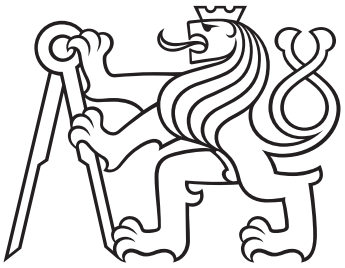
České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů
Program: Softwarové inženýrství a technologie



Backend mobilních aplikácii pro adiktologii

Backend for mobile apps in addictology area

BAKALÁRSKA PRÁCA

Vypracoval: Patrik Jankuv
Vedoucí práce: doc. Ing. Daniel Novák, Ph.D.
Rok: 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jankuv** Jméno: **Patrik** Osobní číslo: **483838**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Backend mobilních aplikácie pro adiktologii

Název bakalářské práce anglicky:

Backend for mobile apps in addictology area

Pokyny pro vypracování:

1) Zoznámte sa s problematikou kontrolovaného pitia a digitálních závislostí u děti
2) Realizujte analýzu požiadavkou na informačný systém a vyberte vhodnú technológiu.
3) Na základe analýzy požiadaviek navrhnite vhodnú architektúru, navrhnite vhodné databázové riešenie pre ukladanie dát.
4) Implementujte prototyp aplikácie. Riešenie riadne otestujte a zdokumentujte. Cieľom je navrhnúť a implementovať webovú aplikáciu, ktorá bude poskytovať backend pre mobilnú aplikáciu Kontrolované pitie. Aplikácia bude zbierať dáta z mobilnej aplikácie a následne umožní ich prezentovanie.

Seznam doporučené literatury:

1. Miovský,M., Čablová,L., & Jurystová,L. (2015).Časnádiagnostikaa krátkéintervencev adiktologii.In K. Kalina(Ed.),Klinickáadiktologie(286–293).Praha:GradaPublishing.
2. H. Brendryen, P. Kraft, and H. Schaalma, “Looking Inside the Black Box: Using Intervention Mapping to Describe the Development of the Automated Smoking Cessation Intervention ‘Happy Ending’,” The Journal of Smoking Cessation, vol. 5, no. 1, pp. 29–56, Jun. 2010.
3. H. Brendryen, F. Drozd, and P. Kraft, “A digital smoking cessation program delivered through internet and cell phone without nicotine replacement (happy ending): randomized controlled trial.,” Journal of medical Internet research, vol. 10, no. 5, p. e51, Jan. 2008.
4. Kulhánek A., Gabrhelík R., Novák D. & Brendren H. (2018). eHealth intervention for smoking cessation for Czech tobacco smokers: Pilot study of user acceptance. Adiktologie, 18(2).

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Daniel Novák, Ph.D., Analýza a interpretace biomedicínských dat FEL

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

Datum zadání bakalářské práce: **11.02.2021** Termín odevzdání bakalářské práce: **13.08.2021**

Platnost zadání bakalářské práce: **30.09.2022**

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

III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

Prehlásenie

Vyhlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác.

V Praze dne .....

.....
Patrik Jankuv

Poďakovanie

Ďakujem doc. Ing. Danielovi Novákovi, Ph.D. za vedenie mojej bakalárskej práce a za podnety, ktoré ju obohatili.

Poďakovanie patrí aj mojej rodine, ktorá ma počas štúdia podporovala.

Patrik Jankuv

Název práce:
Backend mobilních aplikácii pro adiktologii

Autor: Patrik Jankuv

Studijní program: Softwarové inženýrství a technologie
Obor:
Druh práce: Bakalárska práca

Vedoucí práce: doc. Ing. Daniel Novák, Ph.D.
Analýza a interpretace biomedicínských dat, FEL

Abstrakt: Bakalárska práca sa venuje nárvhu a vývoju backendu mobilnej aplikácie Kontrolované pitie. Backend poskytuje REST rozhranie na zber a zálohovanie dát, ktoré je možné použiť u Android i iOS verzie aplikácie. Údaje následne backend reprezentuje vo webovej aplikácii Back office. Myšlienka projektu je založená na metóde kontrolovaného pitia, ktorá sa v adiktológii využíva na zníženie množstva konzumovaného alkoholu pomocou dopredu nastavených denných plánov. Práca je súčasťou komplexnejšieho projektu, ktorý sa zaoberá problémom závislostí. Nosným technológiami backendu sú Spring Boot, Vaadin a PostreSql.
Klíčová slova: backend, kontolované pitie, REST API, Spring Boot, Spring Framework, Vaadin Framework, PostgreSQL

Title:
Backend for mobile apps in addictology area

Author: Patrik Jankuv

Abstract: The bachelor thesis deals with the design and development of the backend of the mobile application Controlled Drinking. The backend provides a REST interface for data collection and backup, which can be used with both Android and iOS versions of the app. The data is then represented by the backend in the Back office web application. The idea of the project is based on the controlled drinking method, which is used in addictionology to reducing alcohol consumption by using pre-set daily plans. The work is part of a more comprehensive project that deals with the problem of addiction. The supporting backend technologies are Spring Boot, Vaadin and PostreSql.
Key words: backend, controlled drinking, REST API, Spring Boot, Spring Framework, Vaadin Framework, PostgreSQL

Obsah

Zoznam použitých skratiek

WHO	Svetová zdravotnícka organizácia
KP	Kontrolované pitie
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
XML	Extensible Markup Language
POJO	Plain Old Java Object
AWS	Amazon Web Services
DI	Dependency Injection
IoC	Inversion of Control
SF	Spring Framework
CRUD	Create, Read, Update, Delete
HTTP	Hypertext Transfer Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
IDE	Vývojové prostredie
UI	User interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
AI	Artificial intelligence - umelá inteligencia
SOAP	Simple Object Access Protocol
API	Application programming interface - rozhranie pre programovanie aplikácií

Zoznam obrázkov

Úvod

Adiktológiu vnímame ako multidisciplinárny vedný odbor, ktorý sa zameriava na prevenciu, liečbu a výskum užívania návykových látok a iných foriem potenciálne návykového správania, ich vplyv na jednotlivca, spoločnosť a sociálnu integráciu osôb, ktoré trpia v dôsledku takýchto foriem správania. Táto oblasť vznikla v dôsledku postupného pretavovania nových poznatkov o príčinách a vývoji závislostného správania z odborov medicíny, psychológie, sociológie a sociálnej práce. Jej jadrom je biopsychosociálny model závislosti a liečby Svetovej zdravotníckej organizácie [1].

0.1 Motivácia

Konzumácia alkoholu je v Českej republike nadpriemerná aj vzhľadom na jeho dostupnosť. Podľa správy WHO[1] z roku 2018 sa Česká republika radí na tretie miesto na svete v ročnej spotrebe alkoholu na osobu. Priemerný Čech spotrebuje 14.4 litra alkoholu ročne [2]. Napriek škodlivosti a negatívnym zdravotným dôsledkom je konzumácia alkoholických nápojov tolerovaná. Počet závislých na alkohole každoročne narastá [3].

Z hľadiska pôsobenia na telo je alkohol jed. Medzi následky chronickej konzumácie alkoholu patrí poškodenie mozgu, pečene, pankreasu, dvanástnika a centrálneho nervového systému. Alkoholizmus spôsobuje metabolické poškodenie buniek v tele a potláča schopnosti imunitného systému. Prejavy nadmernej konzumácie alkoholu sa môžu prejaviť až po rokoch, ale ak závislá osoba pije aj naďalej, jeho život sa môže skrátiť o desať, pätnásť a viac rokov [4]. Mnoho ľudí si myslím, že jediným spôsobom liečby závislosti na alkohole je abstinencia [5] [6].

Táto úvaha však nie je správna a mnohé štúdie to dokazujú. Na základe tejto myšlienky v spolupráci s odborníkmi z kliniky adiktológie 1. lekárskej fakulty Univerzity Karlovej v Prahe vznikla mobilná aplikácia Kontrolované pitie. Jej hlavným cieľom je pomôcť znížiť konzumáciu objemu alkoholu na základe dopredu stanovených plánov.

0.2 Cieľ práce

Cieľom práce je vytvoriť backend, ktorý by zbieral údaje z mobilnej aplikácie Kontrolované pitie, ktorá nadväzuje na výsledky práce Davida Herela. Ten vo svojej bakalárskej práci *Mobilní aplikace pro kontrolované pití* vytvoril prvú verziu tejto mobilnej aplikácie[7] . Výstupom práce bude aplikácia, vytvorená pomocou Spring Boot frameworku, ktorej REST API rozhranie bude slúžiť na komunikáciu medzi mobilom a serverom. Primárnym cieľom servu bude ukladanie dát a ich následná

[1]Svetová zdravotnícka organizácia

prezentácia. Hlavným dôvodom, prečo vznikne backend je, aby mal adiktológ prístup k údajom užívateľov mobilnej aplikácie a dokázal s nimi pracovať. Preto súčasťou aplikácie bude aj webová aplikácia, ktorá prezentuje dáta od užívateľov v grafickej podobe.

0.3 Štruktúra práce

Práca na úvod pripomenie teoretický základ metódy kontrolovaného pitia. Ďalej je potrebné zanalyzovať a zozbierať funkčné, ale aj nefunkčné požiadavky na backend. Na trhu dnes existuje veľké množstvo technológii, ktoré sú schopné daný problém vyriešiť. Každá však má rôzne výhody a nevýhody a je preto potrebné dobré zváženie ich využitia. V teoretickej časti budú opísané nástroje, ktoré boli použité počas vývoja. Zároveň sa práca trocha detailnejšie pozrie na technológiu Spring Framework, ktorá je základným prvkom celej práce. Práca zároveň opíše proces implementácie a pokúsi sa naznačiť najzaujímavejšie problémy, ktoré sa počas vývoja objavili. Testovanie je veľmi dôležitý proces a opíše základne technológie využité pri testovaní.

Kapitola 1

Kontrolované pitie

Definícia kontrolovaného pitia (ďalej KP[1]) sa môže mierne odlišovať podľa typu služby. Zvyčajne však zahŕňa limitáciu množstva a frekvenciu konzumácie alkoholu, zároveň konzument by nemal vykazovať známky závislosti, ale ani sociálne alebo zdravotné problémy súvisiace s požívaním alkoholu [8]. Kľúčovým prvkom techniky KP je zníženie alebo obmedzenie celkovej konzumácie alkoholu a zníženie či obmedzenie rizikového spôsobu pitia vrátane redukcie ťažkých pijackých dní [9].

Technika KP nevyžaduje od konzumenta abstinenciu alebo absolvovanie liečebného programu. Vyžaduje od človeka, aby premýšľal o svojom správaní a našiel si vlastný systém konzumácie. KP okrem zníženia zdravotných rizík spojených s konzumáciou alkoholu vedie aj k liečbe závislosti na alkohole[9]. Je dôležité podotknúť, že technika KP nie je univerzálnym spôsobom liečby závislosti na alkohole a je vhodná len pre niektorých jedincov [9].

1.1 História

Závislosť na alkohole bola dlhodobo vnímaná ako nemoc, ktorej terapia si vyžaduje abstinenciu. Túto teóriu doživotnej abstinencie utvrdzovala štúdia, z ktorej výsledkov vyplývalo, že po dlhodobej pravidelnej konzumácii alkoholu dochádza v mozgu k neurálnej sentitizácii. Vďaka týmto zmenám v mozgu nie je preto možné kontrolovať konzumáciu alkoholu [10].

Zlom nastal v roku 1973, keď manželia Sobelloví vo svojej štúdii piršli k záveru, že KP môže byť pre niektoré osoby možným cieľom. Toto tvrdenie založili na štúdii na vzorke 70 mužov hospitalizovaných v Patton State Hospital. Títo muži boli klasifikovaní ako gamma pijaci [11] a dostali možnosť vybrať si medzi abstinenciou alebo KP. Ďalej tieto skupiny boli rozdelené na kontrolnú a experimentálnu skupinu. Experimentálnej skupine bola poskytnutá individuálna behaviorálna terapia. Kontrolná skupina absolvovala bežnú nemocničnú terapiu. Experimentálna skupina s cieľom dosiahnuť KP dosiahla po roku najlepšie výsledky [6]. To isté platilo aj po dvoch rokoch od ukončenia terapie [12]. Je dôležité dodať, že autori štúdie uvádzajú, že táto terapia je vhodná len pre niektoré osoby [5].

Táto publikácia manželov Sobellových rozdelila verejnú obec. Abstinencia však dodnes stále zostáva preferovaným liečebným cieľom. KP začína byť akceptovateľným cieľom a nástrojom prevencie aj v Českej republike.

[1] Kontrolované pitie

1.2 Realizácia

Počas metódy KP nejde iba o zníženie konzumácie dávok alkoholu. Ide o komplexný kognitívno-behaviorálny terapeutický program. Zameriava sa na získanie kontroly nad pitím, ale aj na nájdenie vnútornej motivácie [13].

Prístup aplikácie ku KP je založený na plánovaní dennej spotreby alkoholu. Užívateľ si v aplikácii vytvorí plán, ktorý sa následne snaží dodržať ako cieľ. Na konci každého dňa je veľmi dôležité sa zamyslieť nad svojimi úspechmi aj sklamaniami. Užívateľ môže tieto pocity, myšlienky a nálady zadať do aplikácie. Jedná sa o kľúčovú časť metódy KP [7].

Aplikácia sa snaží užívateľa udržať pri metóde KP aj motiváciu pomocou peňazí. Po dokončení plánu je možné vidieť aj finančné zhrnutie. Užívateľ tak okrem zdravotných benefitov môže byť motivovaný aj finančne.

Ďalším dôležitým nástrojom motivácie sú aj ocenenia. Tie umožňujú užívateľovi sledovať úspechy a slúžia ako odmena za ich dosiahnutie. Okrem toho môžu motivovať užívateľa k dosahovaniu ďalších cieľov.

Posledným ťahúňom je ukazovateľ promile. Ten informuje, ako si užívateľ vedie [7].

1.3 Pre koho je kontrolované pitie vhodné

Túto metódu môžu vyskúšať ľudia, ktorí netrpia fyzickou závislosťou na alkohole. Je dôležité si uvedomiť, že táto metóda nefunguje pre každého. Aj napriek jej efektivite existujú iné metódy, ktoré môžu byť účinnejšie.

KP môže prísť v úvahu hlavne u ľudí, ktorí sa nechcú z rôznych dôvodov rozhodnúť pre abstinenciu. Zároveň však technika KP môže pomôcť dosiahnuť abstinenciu. Postupným znižovaním dávok konzumácie alkoholu môže človek znížiť dávky alkoholu až na minimum. Kľúčovým prvkom KP je sebareflexia a uvedomenie si problému s alkoholom. Postupným dosahovaním čiastkových cieľov si vybudovať sebavedomie, ktoré pomôže obmedziť konzumáciu alkoholu na dlhodobo udržateľnú úroveň.

Kapitola 2

Analýza

Táto časť obsahuje opis mobilnej aplikácie pre kontrolované pitie, zber požiadavkov a výber konkrétnych technológií.

2.1 Kontrolované pitie

2.1.1 Súšastný stav

Kontrolované pitie je mobilná aplikácia vyvinutá pre platformu iOS a Android. Verzia pre iOS je vyvinuta natívne, zatiaľ čo Android aplikácia využíva hybridnú technológiu React native. Ide o druhú verziu aplikácie. Keďže prvá verzia aplikácie bola k dispozícii len pre systém Android, aktuálna verzia má aktualizovaný dizajn a vyžaduje sa odosielanie údajov z aplikácie na server, kde sa môžu využívať na analýzu. Koncepcia aplikácie je však podobná.

Aplikácia je založená na moduloch. Každý modul poskytuje určitú funkcionalitu. Pri prvom spustení aplikácie sa od používateľa vyžaduje, aby si vytvoril profil pomocou onboardingu. Profil obsahuje základné fyzické parametre o používateľovi, a to: hmotnosť, vek, postoj k fajčeniu.

Po vytvorení profilu používateľa obsahuje úvodná obrazovka karty s odkazmi na štyri základne moduly aplikácie:

Môj plán: Slúži na vytvorenie denného plánu. V tejto časti si používateľ vyberie deň a naplánuje presné množstvo alkoholu, ktoré plánuje skonzumovať.

Chystám sa piť: Ak používateľ konzumuje alkohol, je potrebné to zaznamenať. Práve to umožňuje tento modul. Keď skonzumovaný alkohol prekročí plán, aplikácia na to upozorní a spýta sa používateľa, či chce pokračovať v pití.

Ako sa cítim?: Dôležitá súčasť metódy kontrolovaného pitia. Používateľ vyplní pocity, ktoré prežíval v daný deň. Môže tiež vyplniť, ako dobre plnil plán a môže si pridať osobnú poznámku.

Môj denník: Je to pohľad späť na predchádzajúce dni. Obsahuje údaje vyplnené v module *Ako sa cítim*.

Okrem týchto štyroch hlavných modulov podporuje aplikácia motiváciu užívateľa modulom skrytým v časti profilu:

Achievements: Slúži ako motivačný prvok aplikácie. Odmeňuje používateľov za úspechy a zároveň ich motivuje k dosahovaniu ďalších cieľov.

Hlavná obrazovka obsahuje aj kalkulačku približného obsahu alkoholu v krvi. Výpočty sú založené na hmotnosti, pohlaví a nápojoch vyplnených v časti *Idem piť*. Hlavná obrazovka obsahuje taktiež rozdiel medzi sumou peňazí, ktorú používateľ plánoval minúť a sumou, ktorú skutočne minul. Nakoniec obsahuje hlavná obrazovka odkaz na nastavenia, kde môže používateľ zmeniť parametre profilu.

2.2 Špecifikácia požiadaviek

Špecifikácia požiadaviek vychádza z funkčnosti mobilnej aplikácie vyvinutej pre rôzne platformy. Hlavným cieľom backendu pre mobilnú aplikáciu je zhromažďovať údaje používateľov a využívať ich na výskum v oblasti adiktológie. Druhým cieľom je poskytnúť používateľom možnosť používania aplikácie na rôznych zariadeniach bez straty pokroku pomocou prihlasovania.

Údaje uložené v databáze sú pre človeka ťažko čitateľné. Preto je potrebné vytvoriť nástroj na vizualizáciu údajov pre adiktológov, takzvaný **Back office**. Tento nástroj sa musí jednoducho používať a nevyžadovať si žiadne inštalácie ani špeciálne vybavenie. Najlepšou možnosťou je preto vytvoriť webovú aplikáciu, ktorá môže bežať v prehliadači.

2.3 Funkčné požiadavky

Používateľ musí byť schopný:

1. vytvoriť nový profil
2. upraviť profil
3. vytvoriť účet pomocou registrácie založenej na e-mailu
4. prihlásiť sa do aplikácie pomocou e-mailu
5. načítať údaje účtu
6. odhlásiť sa zo zariadenia
7. používať aplikáciu bez registrácie
8. pridať deň
9. upraviť deň
10. zobraziť všetky dni používateľa
11. zobraziť zoznam spotrebovaných nápojov pre konkrétny deň
12. pridať spotrebovaný nápoj
13. odstrániť spotrebovaný nápoj
14. upraviť spotrebovaný nápoj
15. zobraziť spotrebovaný nápoj

16. pridať pocity pre deň
17. upraviť pocity pre deň
18. zobraziť pocity pre deň
19. odstrániť pocity pre deň
20. zobraziť úspechy.

Back office musí byť schopný poskytovať:

1. prihlásenie do aplikácie pomocou e-mailu
2. odhlásenie položky
3. zobrazenie zoznamu používateľov
4. zobrazenie detailu používateľa obsahujúci: pocity používateľov, časovú os spotrebovaného alkoholu za dni, zoznam dní, ak bol plán bohatý
5. zobrazenie štatistiky úspešných a neúspešných plánov, používateľ môže nastaviť špecifické parametre na základe veku, pohlavia, postoja k fajčeniu.

2.4 Nefunkčné požiadavky

1. Aplikácia poskytuje univerzálne API pre zariadenia s iOS a Android.
2. Aplikáciu je možné nasadiť na Amazon Web Services.
3. Back office je webová aplikácia.

2.5 Výber API

Hlavným zmyslom backendu bude poskytnúť API[1] mobilnej aplikácii, ktorá ho bude využívať na komunikáciu so serverom. GraphQl a REST sú dve riešenie, ktoré je možné použiť pri tvorbe API.

GraphQL

GraphQL je dotazovací jazyk. Bol vyvinutý Facebook ako interná technológia a následne uverejnený ako open-source. GraphQL funguje na úrovni aplikačnej vrsty a na rozdiel od REST API nekladie také veľké množstvo dotazov na rozhranie. Využíva iba jeden endpoint. Medzi jeho nevýhody patrí problém s cachovaním.

[1] Application programming interface - rozhranie pre programovanie aplikácií

REST API

REST API je programovacie rozhranie, ktoré je v súlade s REST architektúrou. Ide o API, ktoré je v porovnaní s GraphQL na trhu dlhšie a za ten čas sa stalo štandardom pre vývoj webových API. Medzi jeho neduhy patrí hlavne potreba veľkého množstva endpointov a nadmerne načítanie zdrojov, ktoré nie sú potrebné pre klienta. Na druhú stranu je toto riešenie považované za bezpečnejšie a kvôli dĺžke na trhu aj prepracovanejšie.

	REST API	**GraphQL API**
Architektúra	riadená serverom	riadená klientom
Organizované v zmysle	schéma a typový systém	endpointy
Načítanie dát	fixne určené	špecificky určené
Caching	áno	nie
Formát	JSON, XML, HTML a ďalšie	JSON
Učiaca krivka	ľahká	ťažká
Komunita	veľká	rastie

Tabuľka 2.1: Porozmanie základných vlastnosti REST API a GraphQL API

Ako API, ktoré bude poskytovať backend som sa rozhodol pre REST API. Ide o overenú technológiu, ktorá je na trhu už dlhšie a patrí medzi štandard. Limitácie, ktoré dané riešenie poskytne nepovažujem za tak zásadne vzhľadom na jednoduchosť mobilnej aplikácie.

2.6 Výber programovacieho jazyka

Na trhu dnes existuje veľké množstvo technológii, ktoré môžu implementovať backend mobilnej aplikácie. Každá technológia má svoje plusy, ale aj mínusy.

V nasledujúcej časti budú opísané programovacie jazyky, ich charakteristika a nástroje, ktoré je možné použiť pri vývoji backendu. S jazykmi, ktoré sú zahrnuté v tomto prehľade som sa nejakým spôsobom stretol počas štúdia.

C++

C++ je rozšírenou verziou C. Prináša triedy, čo umožňuje objektovo orientované programovanie na rozdiel od C. C++ je jedným z najstarších programovacích jazykov a zvyčajne sa využíva na programovanie operačných systémov, hier alebo databázových softvérov. C++ je low-level jazyk a môže interagovať priamo s hardwerom.

Kód napísaný v C++ je rýchly a je spustiteľný na rôznych platformách a zariadeniach. Umožňuje vývojárovi spravovať pamäť aplikácie, na druhu stranu to môže spôsobovať problémy u neskúsených programátorov a spôsobovať vážne zraniteľnosti systému. C++ frameworky na tvorbu backendu: Beast, Wt

Java

Java je jednou z najmocnejších backend technológii. Developeri ju používajú na vývoj prispôsobiteľných a na funkcie bohatých webových aplikácii už roky. Javu nájdeme v mobilných aplikáciách alebo v softvér mikrokontroléroch.

Silnou stránkou Javy sú škálovateľnosť, paralelizmus, open-source knižnice a veľmi silné zabezpečenie. Je preto vhodná pre aplikácie s vysokým zaťažením. Server-side programovanie je náročné na zdroje a čas. Java frameworky na tvorbu backendu: Spring, Struts, Hibernate

JavaScript

JavaScript jeden z najviac používaných programovacích jazykov súčasnosti. S Javou má spoločný len názov Java. Ide o skriptovací jazyk. Pomocou Node.js je možné vytvoriť serverové riešenie pomocou JavaScriptu. Jazyk nájdeme pri tvorbe webových, mobilných, single a multi-page aplikácii.

Hlavnou výhodou JavaScriptu je rýchlosť vývoja. Aplikácie sú vďaka Express.js, z ktorého Node.js berie väčšinu funkcionalít veľmi minimalistické a nenáročné na miesto. Pre developerov, ktorí programujú v inom programovacom jazyku môže byť náročné porozumieť eventmi riadenému backendu. JavaScript frameworky na tvorbu backendu: Express.js, Node.js, Meteor, Gatsby

PHP

PHP je open source programovací jazyk, ktorý sa využíva pri programovaní aplikácií na strane servera. Backend technológia sa využíva hlavne pre webové stránky a až okolo 79% stránok na internete využíva PHP [14].

Tento univerzálny skriptovací jazyk sa ľahko používa a je vhodný pre začiatočníkov. Má širokú komunitu a je relatívne lacný na vývoj, na druhú stranu PHP zaostáva za modernými backend technológiami svojou jednoduchosťou a jeho popularita klesá. PHP frameworky na tvorbu backendu: Laravel, Symfony, CakePHP, Slim R

Python

Python je popredný jazyk súčasnosti. Vďaka syntaxe jazyka podporuje programátorov k písaniu prehľadných skriptov. Podľa Stack Overflow ide o jazyk, ktorý sa chce naučiť najviac backend vývojárov [15]. Využíva sa v AI[2], pri spracovaní veľkého množstva dát, ale aj v backendoch.

Medzi hlavné prednosti patrí jednoducho pochopiteľný kód, veľké množstvo knižníc a dostupnosť. Akékoľvek prerušenie programu môže viesť k spomaleniu programu a v porovnaní s inými backend technológiami je vrstva pracujúca s dátami nie úplne vyvinutá. Python frameworky na tvorbu backendu: Django, Flask, CherryPy

Zo zoznamu vyššie spomenutých jazykov som sa rozhodol pre Javu. Jazyky C++ a PHP som vylúčil na začiatku, keďže C++ je relatívne náročné na programovanie a PHP pre jeho jednoduchosť. JavaScript som vylúčil kvôli veľmi malej

[2]Artificial intelligence - umelá inteligencia

skúsenosti s týmto jazykom. Pre Javu nakoniec rozhodla rýchlosť výslednej aplikácie, ale aj to, že s riešením podobného problému mám skúsenosť práve v Jave. Pokiaľ by som vyberal jazyk bez predošlej skúsenosti, tak by som zvolil Python.

Kapitola 3

Technológie

V nasledujúcej časti budú opísané technológie použité v tejto práci. Hlavným cieľom tejto práce je vytvoriť servrovú časť aplikácie *Kontrolované pitie*. Server musí poskytovať univerzálne API pre mobilné aplikácie pre Android a iOS.

3.1 Representational state transfer

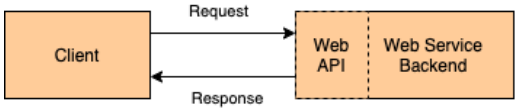
Representational state transfer (REST) je softvérový architektonický štýl pre distribuované prostredia, ktorý využíva podmnožinu protokolu HTTP [16]. Webová služba, ktorá sa riadi týmito usmerneniami, sa nazýva RESTful. REST navrhol a opísal v roku 2000 Roy Fielding v piatej kapitole svojej dizertačnej práce *Architectural Styles and the Design of Network-based Software Architectures* [16].

Rozhranie REST je použiteľné na jednotný a jednoduchý prístup k zdroju. Zdrojmi môžu byť údaje alebo stavy aplikácie (ak ich možno opísať špecifickými údajmi). Na rozdiel od známejšieho SOAP[1] je teda REST orientované na údaje, nie na procedúry [17]. Všetky zdroje majú svoje vlastné URI[2] a REST definuje štyri základné metódy (GET, POST, PUT a DELETE) na prístup k nim [18].

RESTful API vyžaduje nasledujúce kritériá:[20][21][16]

- architektúra klient-server 3.1,
- bezstavová komunikácia klient-server,
- cachovateľné údaje,
- jednotné rozhranie medzi komponentmi,
- vrstvený systém, ktorý organizuje každý typ servera zapojeného do získavania požadovaných informácií do hierarchií neviditeľných pre klienta,
- kód na požiadanie (voliteľný).

[1] Simple Object Access Protocol
[2] Uniform Resource Identifier



Obr. 3.1: Architektúra klient-server[19]

3.1.1 Štruktúra REST požiadavku

REST požiadavka je tvorená štyrmi základnými prvkami.

- HTTP metóda - definuje, čo sa má stať so zdrojom. Základne metódy sú POST na vytvorenie objektu, PUT na úpravu stávajúceho objekt, GET na vyvolanie objektu a DELETE na vymazanie objektu.
- endpoint - obsahuje jednotný identifikátor prostriedku (URI), ktorý slúži na identifikovanie zdroju na internete. Najčastejším typom URI je URL[3] adresa, ktorá slúži ako kompletná webová adresa.
- hlavička - obsahuje informácie potrebné pre klienta a server. Zvyčajne hlavička poskytuje autentifikačné údaje ako API kľúče, IP adresu servera a informáciu o formáte odpovede.
- telo - sprostredkuváva doplňujúce informácie. Napríklad môže sa jednať o dáta, ktoré chceme vytvoriť na servery. Dáta sa zvyčajne prenášajú v XML[4] alebo JSON[5] formáte.

```
POST /profile
HOST: controlled-drinking-app.herokuapp.com
Content-Type:application/json
Accept:application/json

{
"name": "Jan",
äge": 23,
"gender": "MALE",
"height": 181,
"smoker": "NO",
"weight": 75
}
```

Príklad REST requestu, hnedou označená HTTP metóda, červenou endpoint, zelonou headers a modrou telo requestu

3.1.2 Štruktúra REST odpovede

Server ako odpoveď neposiela samotný hľadaný zdroj, ale jeho reprezentáciu- strojovo čitateľný popis jeho aktuálneho stavu. Ten istý zdroj môže byť reprezentovaný v rôznych formátoch, ale najobľúbenejšie sú XML a JSON.

3.2 The Spring Framework

Je to open-source framework na vývoj aplikácií Java EE. Spring Framework je najpopulárnejší framework Java a môže pracovať s akoukoľvek aplikáciou Java.

[3]Uniform Resource Locator
[4]Extensible Markup Language
[5]JavaScript Object Notation

Rod Johnson predstavil jeho prvú verziu v októbri 2002 v knihe Expert One-on-One J2EE Design and Development. Spring Framework používa moduly, ktoré poskytujú rôzne funkcie. Medzi najpopulárnejšie patria Spring Boot, Security Spring alebo Spring MVC [22].

Medzi hlavné výhody Spring môžeme zaradiť použitie objektov POJO[6], ktoré umožňuje vývoj aplikácie bez webového servera. Vďaka tomu je aplikácia mimoriadne ľahká, čo je pri vývoji webových aplikácií veľmi významná výhoda. Objemnosť Spring umožňuje veľmi flexibilne konfigurovať aplikáciu podľa potreby. Navyše vďaka technológii Dependency injection je možné jednoduché testovanie [23].

Hlavnou nevýhodou tohto frameworku je jeho zložitosť. Má viac ako 2400 tried, čo z neho robí komplikovaný nástroj na používanie. Má vysokú krivku učenia, takže potrebuje vývojárov, ktorí už majú s ním určité skúsenosti pre rýchlu prácu. Spring používa mnoho paralelných mechanizmov, čo je tiež veľkou výhodou; zároveň to môže vývoj zmiasť [23][24].

Funkcie Spring umožňujú efektívny vývoj od jednoduchej webovej aplikácie až po komplexné podnikové aplikácie. Hlavné koncepty, na ktorých Spring Framework závisí, sú [25]:

- Inversion of Control
- Dependency Injection
- Aspektovo orientované programovanie
- Java Persistence API

3.2.1 Inversion of Control

Inversion of Control (ďalej IoC[7]) je princíp programovania postavený na inverznom toku riadenia ako tradičný tok riadenia. Namiesto programátora preberajú riadenie toku programu externé zdroje, ako sú rámce, služby a iné komponenty[25]. IoC možno použiť na zvýšenie modularity programu a jeho rozšíriteľnosti[26]. IoC umožňuje vývoj aplikácií v objektovo orientovanom programovaní a existuje niekoľko techník na jeho implementáciu:[27]

- Service locator pattern
- Dependency injection
- Contextualized lookup
- Template method design pattern
- Strategy design pattern

3.2.2 Dependency injection

Dependency injection (ďalej DI[8]) je technika v objektovo orientovanom programovaní na vkladanie závislostí medzi jednotlivé komponenty programu tak, aby

[6]Plain Old Java Object
[7]Inversion of Control
[8]Dependency Injection

jeden komponent mohol používať druhý bez odkazu naň v čase kompilácie programu. DI je formou širšej techniky acioc, ktorú využíva Spring Framework [28].

V Spring Frameworku má správu referencií na starosti Container. Container je zodpovedný za riadenie životných cyklov, inicializáciu a konfiguráciu objektov ich spájaním na základe vzťahov. Vytvorené komponenty sa nazývajú Bean. Kontajner môže byť nakonfigurovaný pomocou anotácie Java alebo na základe súborov XML [22].

3.2.3 Aspektovo orientované programovanie

Aspektovo orientované programovanie (AOP) je programovacia paradigma, ktorej cieľom je zvýšiť modularitu programu rozdelením na časti, ktoré sa neprekrývajú svojou funkcionalitou. AOP možno rozdeliť na dva typy, a to: statické AOP a dynamické AOP [29].

Statické AOP je rýchlejšie ako dynamické AOP, pretože weaving (vkladanie aspektov do aplikácie) prebieha počas zostavovania aplikácie. Ak je však potrebná zmena počas behu programu, aplikácia sa musí znovu skompilovať. Dynamické AOP je pomalšie ako statické AOP, ale zmeny kódu nevyžadujú kompiláciu celého programu, pretože weaving prebieha počas behu programu. Spring Framework využíva dynamické AOP [29] [22].

3.2.4 Java Persistence API

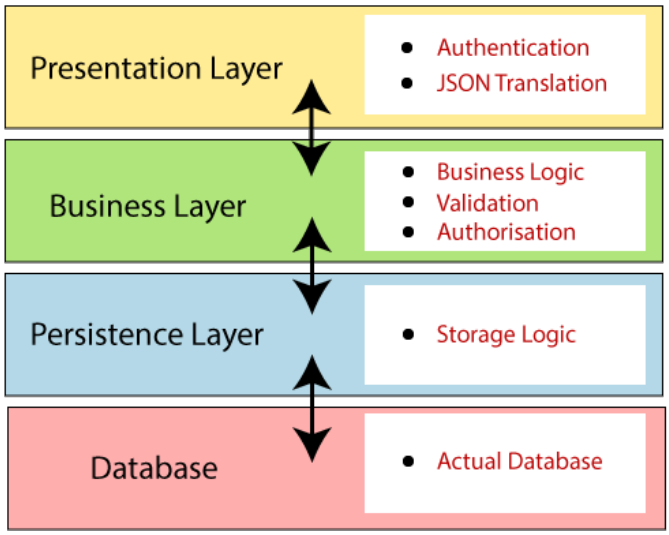
Java Persistence API (JPA) poskytuje vývojárom nástroj na objektovo-relačné mapovacie a na správu relačných údajov v Java aplikáciách [30]. JPA definuje súbor konceptov, ktoré možno implementovať pomocou frameworku alebo nástroja. Model JPA bol založený na základe Hibernate [31].

3.3 Spring Boot

Spring Boot je modul SF[9], čo znamená, že zdieľa všetky technologické koncepty SF. Umožňuje vytvárať samostatné aplikácie s minimálnou konfiguráciou. Vývojári zvyčajne používajú Spring Boot na vývoj rozhraní REST API. Hlavnou výhodou oproti SF je automatická konfigurácia. Spring Boot automaticky konfiguruje triedy na základe požiadaviek. Umožňuje písať kratšie kódy a znižuje opakovanie podobných častí kódu na vykonanie malých úloh. Spring Boot ponúka webové servery Jetty alebo Apache Tomcat bez potreby explicitných nastavení servera [32]. To všetko dohromady uľahčuje vývojárovi prácu a zároveň sa skracuje čas potrebný na vývoj aplikácie.

[9]Spring Framework

3.3.1 Spring Boot architektúra



Obr. 3.2: Spring Boot architecture [33]

Spring Boot má vrstevnú architektúru. Každá vrstva môže komunikovať len so susednou vrstvou. Existujú štyri vrstvy: [33]

- **Prezentačná vrstva:** Vrstva sa stará o komunikáciu pomocou HTTP požiadavkov a preklad JSON na objekty. V prípade potreby posunie požiadavku na business vrstvu, ktorá ju ďalej spracuje.
- **Business vrstva:** Má na starosti business logiku. Obsahuje triedy služieb a využíva služby vrstiev prístupu k údajom. V tejto vrstve sa vykonáva autorizácia a validácia na základe požiadavky z prezentačnej vrstvy.
- **Persistentna vrsta:** Obsahuje logiku uchovávania údajov. Prevádza objekty do tabuliek pomocou JPA.
- **Dátová vrstva:** Je zodpovedná za perzistenciu údajov, čo znamená, že údaje aplikácie budú uložené aj po ukončení behu programu.

3.4 Spring Security

Spring Security je vysoko prispôsobiteľný framework na overovanie a kontrolu prístupu. Je to de facto štandard na zabezpečenie aplikácií založených na frameworku Spring. Spring Security je rámec, ktorý sa zameriava na poskytovanie autentifikácie a autorizácie aplikácií Java [34]. Framework chráni aj pred útokmi ako sú cross-site request, session fixation alebo clickjacking. Šikovnou funkciou je podpora OAuth2, ktorá umožňuje overovanie používateľov pomocou sociálnych účtov ako sú Google, AppleID alebo Facebook [35]. Vďaka modularite SF možno aplikáciu Spring Boot ľahko rozšíriť o závislosti v súbore pom.xml a pridať do aplikácie Spring Security.

3.5 Vaadin Framework

Vaadin Framework je open-source framework na vytváranie webových aplikácií. Kód je napísaný v jazyku Java a potom preložený do jazyka JavaScript pomocou nástroja Google Web Toolkit. Preložený kód interpretuje webový prehliadač. Táto architektúra umožňuje vývoj aplikácií v jazyku JavaScript výlučne písaním kódu v jazyku Java [36].

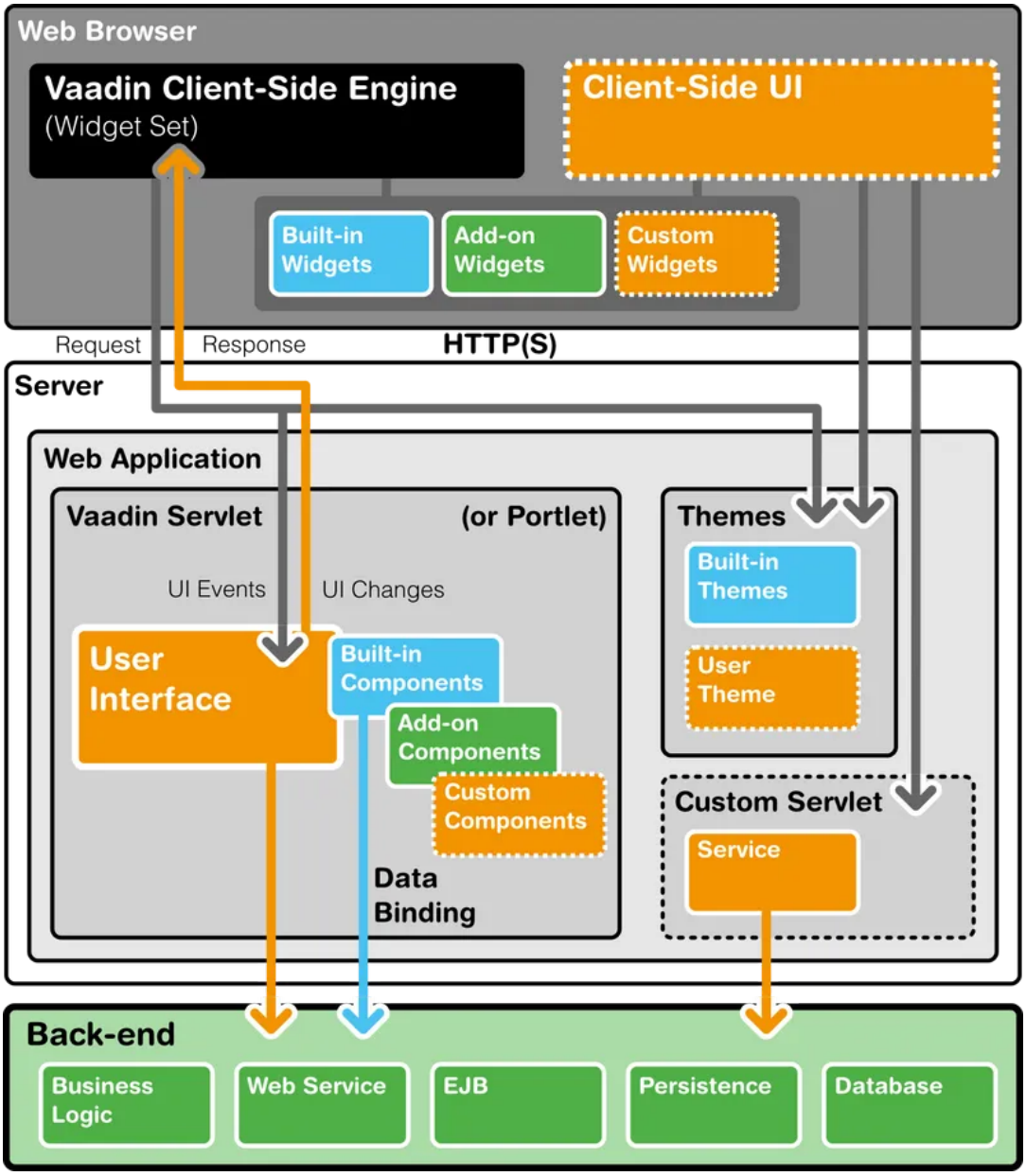
Vaadin je založený na serverside architektúre, čo znamená, že väčšina logiky beží na serveroch. Vaadin poskytuje knižnicu predpripravených komponentov, ktoré umožňujú vývojárom rýchlo nakódovať frontendové rozhrania. Samozrejmosťou je aj vytváranie nových komponentov [37] [36].

3.5.1 Vaadin architektúra

Architektúra Vaadin Runtime Architecture poskytuje základnú ilustráciu komunikácie na strane klienta a servera v situácii, keď bola stránka s kódom na strane klienta (engine alebo aplikácia) pôvodne načítaná do prehliadača.

- *užívateľské rozhranie* - Vaadin aplikácia poskytuje užívateľské rozhranie na interakciu užívateľa s business logikou a dátami aplikácie.
- *komponenty a widgety rozhrania* - užívaľské rozhranie je tvorené komponentami, ktoré sú vytvorené a umiestené aplikáciou. Widget je prvok zobrazený vo webovom prehliadači na strane klienta ako náprotivok komponentu na strane servera.
- *client-side engine* - má na starosti prekreslovanie komponentov na widgety v prehliadači. Komunikácie je realizovaná pomocou HTTP alebo HTTPS requestov.
- *vaadin servlet* - prijíma požiadavky od rôznych klientov, pomocou sledovania relácií pomocou súborov cookies určuje, ku ktorej relácii používateľa patria a deleguje požiadavky na príslušné relácie. Je možné ho prispôsobiť rozšírením.
- *motivy* -obsahuje CSS alebo Sass, ktoré definujú výzor komponentu. Vaadin poskytuje niekoľko prednastavených tém.
- *backend* - obsahuje všetku business logiku aplikácie oddelenú od prezentačnej vrstvy.
- *server push* - Vaadin podporuje server push, čo znamená, že zmeny v UI[10] sa propagujú priamo zo serveru bez potreby požiadavky od klienta.
- *data binding* - okrem modelu používateľského rozhrania poskytuje Vaadin aj dátový model na prepojenie údajov prezentovaných v políčkach komponentoch ako sú textové polia, zaškrtávacie políčka a výberové komponenty so zdrojom údajov. Pomocou dátového modelu môžu komponenty používateľského rozhrania priamo aktualizovať údaje aplikácie, často bez potreby akéhokoľvek riadiaceho kódu.

[10]User interface



Obr. 3.3: Vaadin architektúra[37]

- *klientské aplikace* - Vaadin podporuje aplikačné moduly na strane klienta, ktoré sa spúšťajú v prehliadači. Moduly na strane klienta môžu používať rovnaké widgety, témy a back-end služby ako aplikácie Vaadin na strane servera.

3.6 Ostatné nástroje

3.6.1 Hibernate ORM

Hibernate je framework napísaný v jazyku Java určený na objektovo-relačné mapovanie [38]. Používa sa na mapovanie java objektov na entity v relačnej databáze. Mapovacie súbory môžu konfigurovať mapovanie. Alternatívou je aj použitie anotácií na atribútoch objektov [39] [38]. Aplikácia *Kontrolované pitie* používa na mapovanie anotácie. Použitie Hibernate zjednodušuje prácu vývojára tým, že odpadá nutnosť ručne transformovať objekty do relačnej databázy.

3.6.2 PostgreSQL

PostgreSQL je objektovo-relačný databázový systém s otvoreným zdrojovým kódom [40]. V súčasnosti je najpoužívanejší bezplatný databázový systém. Podporuje MVCC, takže sprístupňuje snímku databázy každému používateľovi. Zmeny sa ostatným používateľom zobrazia až po potvrdení transakcie [41].

3.6.3 Git

Git je softvér na sledovanie zmien v ľubovoľnom súbore, ktorý sa zvyčajne používa na koordináciu programátorov, ktorí spoločne vyvíjajú zdrojový kód počas vývoja softvéru. Medzi jeho ciele patrí rýchlosť, integrita údajov a podpora distribuovaných nelineárnych pracovných postupov [42].

Pozitívnym vedľajším účinkom používania git je, že kód je zálohovaný pre prípad, že by lokálny počítač zlyhal.

3.6.4 Swagger

Swagger je open-source framework na navrhovanie, vytváranie, dokumentáciu a používanie RESTful webového rozhrania. Okrem editora na vytvorenie nového webového API obsahuje Swagger aj nástroje na automatizované dokumentovanie a testovanie existujúceho API (podľa URL API). Ďalej obsahuje nástroj na generovanie kódu podľa zadaného rozhrania. [43][44][45][46][47].

Swagger vyvíja spoločnosť SmartBear Software, ktorá sa aktívne podieľa na tvorbe iniciatívy OpenAPI. Zakladajúcimi členmi iniciatívy OpenAPI sú aj spoločnosti Microsoft, IBM a Google [48].

Swagger poskytuje podporu pre mnohé technológie vrátane Spring, Django, Node.js alebo Ruby on Rails. [43]

3.6.5 Javadoc

Javadoc je nástroj pre jazyk Java na generovanie dokumentácie API vo formáte HTML zo zdrojového kódu. V súčasnosti ho vlastní spoločnosť Oracle Corporation, ale pôvodne ho vytvorila spoločnosť Sun Microsystems. Formát HTML sa používa na zvýšenie pohodlia hypertextového prepojenia súvisiacich dokumentov [49].

Formát "doc comments"používaný spoločnosťou Javadocom je de facto priemyselným štandardom pre dokumentáciu tried jazyka Java. Niektoré IDE, napríklad IntelliJ IDEA, NetBeans a Eclipse, dokážu automaticky generovať Javadoc HTML [50][51][52]. Mnohé editory súborov pomáhajú používateľovi pri vytváraní zdrojov Javadoc a používajú informácie Javadoc ako interné odkazy pre programátora.

```
1   /**
2     * Count items in day and check if plan was accomplished
3     *
4     * @param day Day to validate plan
5     * @return true if plan is accomplished, false if not
6     */
7    private boolean countItemsAndCheckPlan(Day day) {
8        /** code */
9    }
```

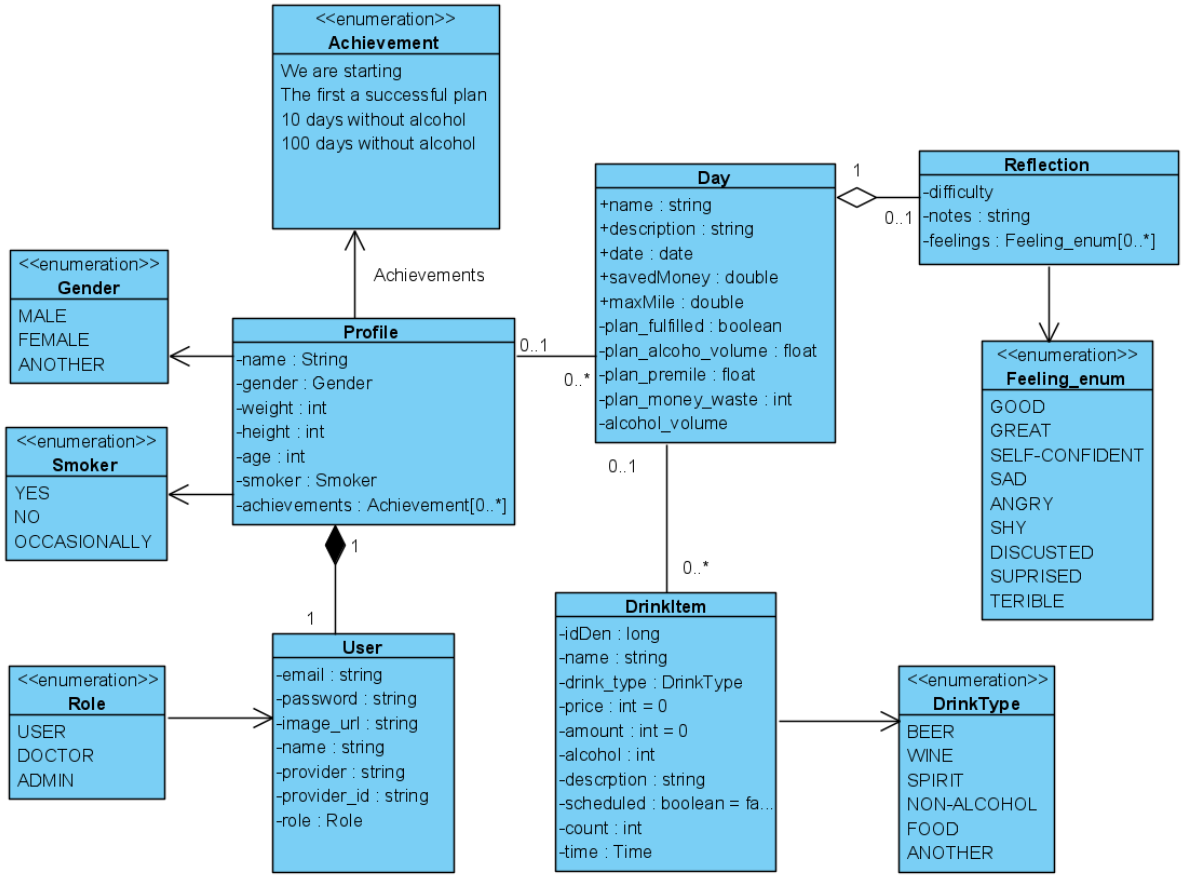
Listing 3.1: Javadoc príklad

Kapitola 4

Dizajn systému

4.1 Diagram tried

Diagram tried 4.1 predstavuje objektový model. Model je prenesený do dátovej vrstvy v aplikácii. Tento model je vďaka nástroju ORM Hibernate namapovaný na databázu PostgreSQL použitú v projekte.



Obr. 4.1: Diagram tried, vytvorené pomocou: Visual paradigm

4.1.1 User - Užívateľ

Objekt *User* predstavuje prihlasovacie údaje. Aplikácia poskytuje štandardnú autentifikáciu pomocou e-mailu a hesla. Ďalšou možnosťou prihlásenia je protokol oauth2 pre sociálne prihlásenie pomocou účtu Google alebo Facebook. Ak chce používateľ používať aplikáciu bez registrácie, služba vygeneruje e-mail a heslo.

4.1.2 Profil

Trieda *Profil* uchováva primárne terapeutické údaje o používateľovi. Tieto údaje sú kľúčové pre popis pacienta. Profil obsahuje nasledujúce atribúty:

- *name* - meno,
- *age* - vek,
- *gender* - používateľ má tri možnosti: muž, žena, iné,
- *height* - výška v centimetroch,
- *weight* - váha v kilogramoch,
- *smoker* - predstavuje postoj používateľa k fajčeniu pri výbere: áno, nie, príležitostne,
- *dosiahnuté výsledky* - opísané v ďalšej časti.

4.1.3 Achievements - Dosiahnuté výsledky

Mobilná aplikácia sa snaží motivovať používateľa cieľmi a odmeniť ho za ich splnenie odznakmi. Každý používateľ má rovnaké ciele. Z tohto dôvodu sa úspechy ukladajú ako enumerácie v backende. Atribút achievement používateľa je zoznam zozbieraných achievementov.

4.1.4 Day - Deň

Metóda kontrolovaného pitia je založená na vytvorení rozvrhu na deň. Základným atribútom je *date* poskytnutý kalendárom v mobilnej aplikácii. Keď používateľ vytvorí nový plán v aplikácii, je odoslaná POST požiadavka na server na vytvorenie novej inštancie dňa. Deň odkazuje na *reflection* a obsahuje zoznam nápojov.

4.1.5 Reflection - Reflexia

Účelom tohto objektu je zhromažďovať emócie používateľa pre konkrétny deň. Pole pocity obsahuje pocity používateľa počas dňa zo zoznamu *feeling enum*, ktorý obsahuje základné ľudské emócie. Okrem toho si používateľ môže urobiť poznámku a vybrať, ako ťažké bolo dodržiavať plán.

4.1.6 Drink item - Nápoj

Trieda má nasledujúce atribúty:

- *alcohol* - percento alkoholu v nápoji,
- *amount* - objem nápoja v mililitroch,
- *count* - počet nápojov,
- *time* - ak nápoj nie je naplánovaný, zaznamená sa čas jeho konzumácie,
- *description* - opis položky,
- *name* - názov značky,
- *scheduled* - príznak na zaznamenanie, ak je nápoj naplánovaný,
- *price* - cena jedného nápoja,
- *drink type* - typ nápoja ako napr. pivo, destilát a pod.

4.2 Nasadenie na server

Nasledujúci diagram opisuje infraštruktúru nasadenia aplikácie. Aplikácia bude bežať na linuxovom Debian počítači, ktorý poskytuje AWS[1]. Aplikácia mení dáta medzi databázou PostgresSQL prostredníctvom protokolu TCP/IP[2]. Komunikácia medzi multiplatformovými mobilnými aplikáciami sa realizuje pomocou rovnakého REST rozhrania. Používateľ môže spustiť Back office vo webovom prehliadači prostredníctvom HTTP[3].

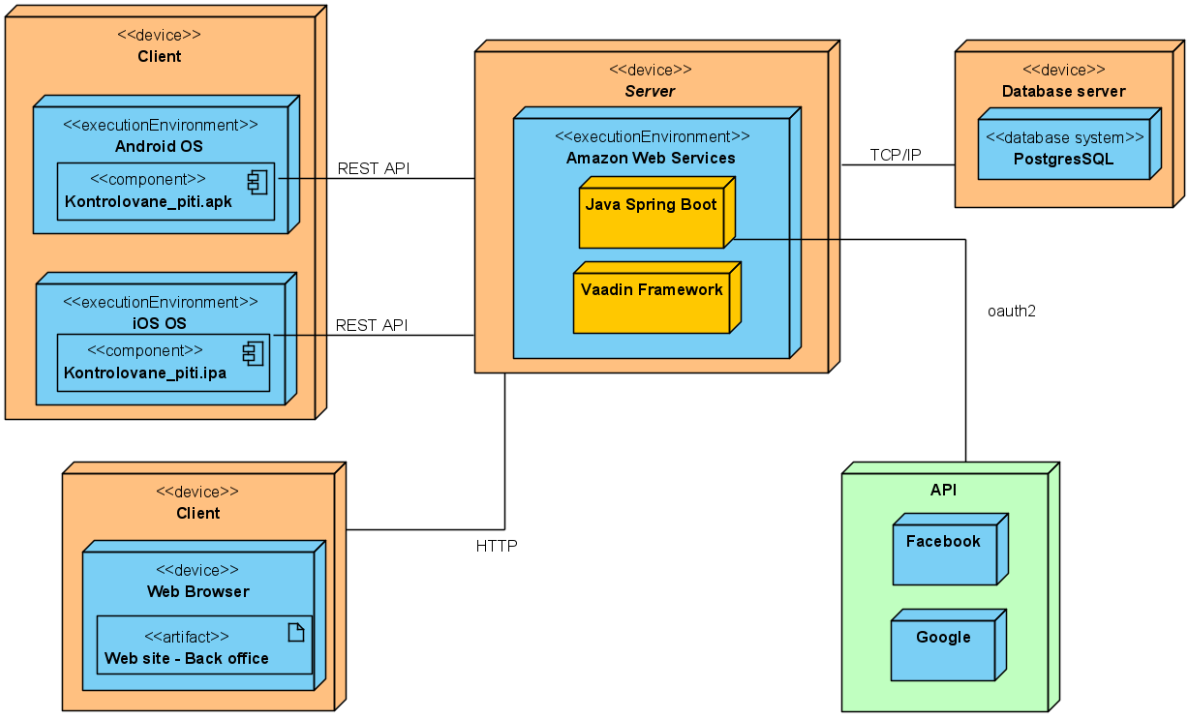
4.3 Dokumentácia

Dokumentácia je veľmi dôležitá v každom softvérovom projekte, keď sa na vývoji podieľa viac ľudí. Keďže táto práca je len súčasťou väčšieho projektu, je potrebné poskytnúť kvalitnú dokumentáciu pre ľudí, ktorí budú mobilnú aplikáciu pripájať k serveru. Na rýchle a prehľadné zdokumentovanie je vhodné použiť nástroje, ktoré dokážu vygenerovať dokumentáciu z kódu.

Swagger je užitočný nástroj na vytváranie dokumentácie RESTful API. Dokumentáciu budu využívať primárne vývojári mobilnej aplikácie. Tí nájdu na jednom URL všetky endpointy, ktoré môžu využiť pri prepájaní mobilnej aplikácie zo serverom.

Druhý typ dokumentácie je vygenerovaný pomocou Javadoc. Táto dokumentácia je určená pre vývojárov samotného backendu. Ide o statické HTML[4] stránky, ktoré je možné vygenerovať pomocou vývojového prostredia. Javadoc sa generuje z komentárov v kóde.

[1] Amazon Web Services
[2] Transmission Control Protocol/Internet Protocol
[3] Hypertext Transfer Protocol
[4] Hypertext Markup Language



Obr. 4.2: Model nasadenia

4.4 Zabezpečenie

Súčasťou aplikácie je aj zabezpečenie, ktoré má na starosti to, aby k dátam mali prístup iba oprávnení užívatelia. To je vytvorené pomocou štandardných nástrojov, poskytovanými SF. Jadro zabezpečenia je vytvorené pomocou SF modulu Spring Security, ten využíva JSON Web Token. Na uľahčenie používania je možné využiť aj OAuth2, ktoré umožňuje sa zaregistrovať a prihlásiť sa pomocou účtu od Googlu a Facebooku.

JSON Web Token (JWT)

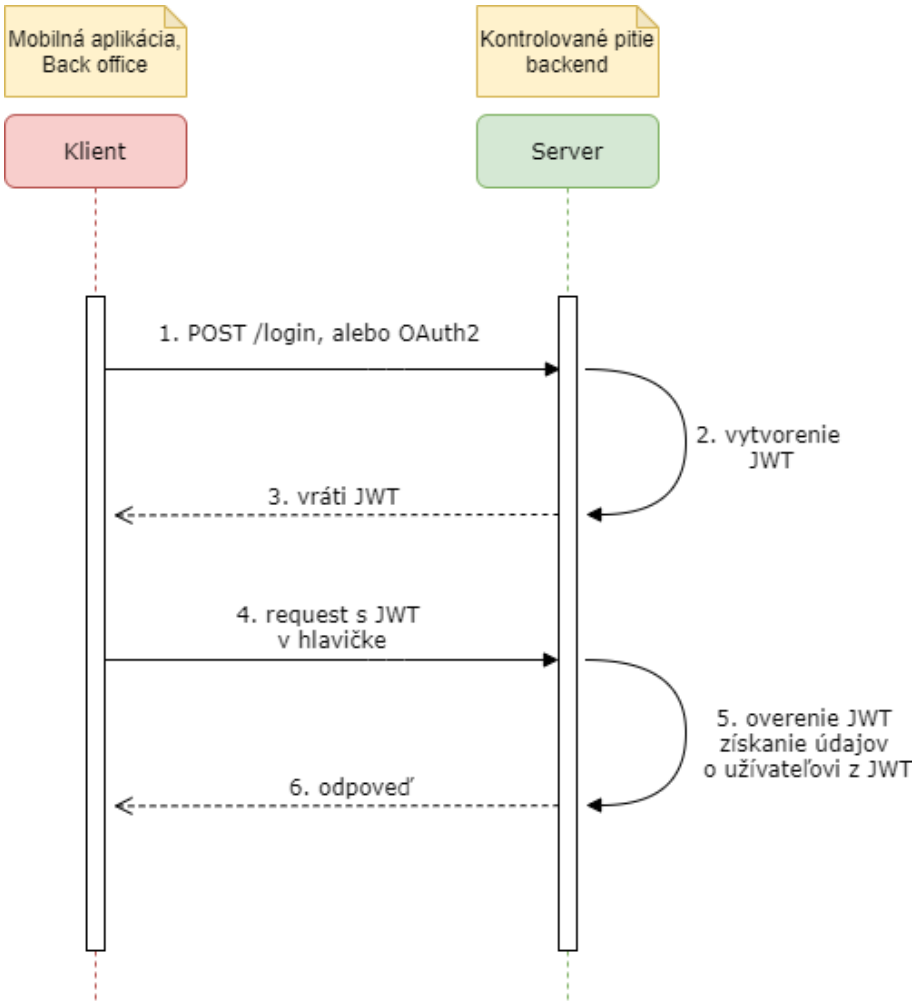
JWT je otvorený štandard, ktorý definuje kompaktný a samostatný spôsob bezpečného prenosu informácií medzi stranami vo forme JSON objektu. Tokeny sa podpisujú buď pomocou súkromného tajomstva alebo kľúča, ktorý je verejný alebo tajný [53]. Tokeny sú navrhnuté tak, aby boli kompaktné, bezpečné pre URL a použiteľné najmä v kontexte jednotného prihlásenia vo webovom prehliadači.

JWT sa typicky využíva na odvodenie roly overeného používateľa pre požadovanú službu, ale aj zároveň na identifikáciu identity, ktorá o službu žiada. Vďaka svojej relatívnej malej veľkosti sa JWT môže odosielať prostredníctvom adresy URL, parametra POST alebo v hlavičke HTTP. JWT obsahuje všetky potrebné informácie o entite, aby sa zabránilo viacnásobnému dopytovaniu databázy.

OAuth2

OAuth je otvorený štandard pre delegovanie prístupu, ktorý sa bežne používa ako spôsob, akým môžu používatelia internetu udeliť webovým stránkam alebo aplikáciám prístup k svojim informáciám na iných webových stránkach, ale bez toho,

aby im poskytli heslá. Tento mechanizmus používajú spoločnosti ako Google, Facebook, Microsoft a Twitter, aby umožnili používateľom zdieľať informácie o svojich účtoch s aplikáciami alebo webovými stránkami tretích strán.



Obr. 4.3: Diagram zobrazujúci autentifikciu a autorizáciu užívateľa pomocou JWT

Kapitola 5

Implementácia

5.1 Architektúra programu

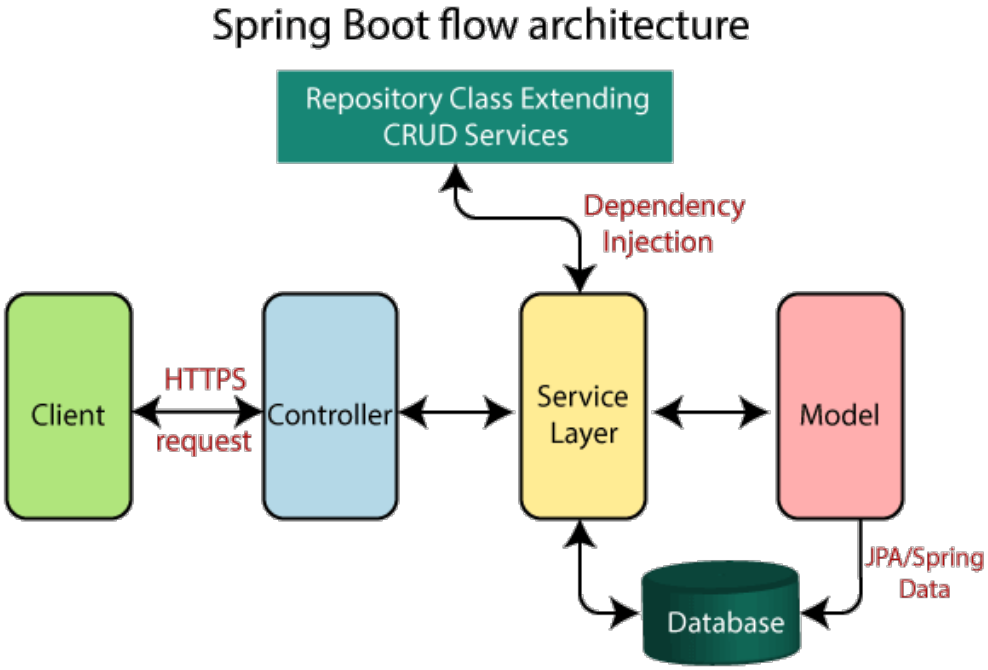
Architektúra toku Spring Boot využíva základy architektúry Spring Boot Framework podľa vrstiev, keď každá vrstva môže komunikovať len s vrstvou nad ňou alebo pod ňou v hierarchickej štruktúre. Program realizuje tieto vrstvy pomocou balíkov zložených z tried.

Model využíva princípy objektovo orientovaného programovania na vytvorenie modelu s anotáciou @Entity označujúcou triedu modelu. JPA transformuje označené triedy do databázových tabuliek aj so vzťahmi medzi objektmi. Aplikácia na kontrolované pitie používa Hibernate ako nástroj JPA. Rozhodujúce je nakonfigurovať spojenie medzi aplikáciou a databázou v súbore application.yml. Okrem prihlasovacích údajov je potrebné nastaviť typ databázy [22].

Service layer je v projekte implementovaná v balíkoch Dao a Service. Dao je zodpovedný za logiku ukladania, ako sú operácie CRUD alebo iné špecifické SQL požiadavky na databázu. Potom Dao tieto údaje prezentuje balíku Service, kde sa realizuje business logika programu. Táto vrstva aplikuje funkciu Dependency Injection.

Balík Controller predstavuje vrstvu s rovnakým názvom. Balík obsahuje všetky koncové body HTTP používané na komunikáciu s externými zariadeniami. Vrstva spracúva požiadavku a v prípade potreby volá logiku služby. Po spracovaní požiadavky vrstva controller vráti odpoveď späť klientovi.

Klientom je v tomto scenári mobilná aplikácia, ktorá vďaka HTTP požiadavkám komunikuje so serverom. Server odpovedá stavovými kódmi odpovede HTTP. Údaje sa prenášajú v tele požiadavky alebo odpovede vo formáte JSON.



Obr. 5.1: Diagram Spring Boot Flow Architectury[54]

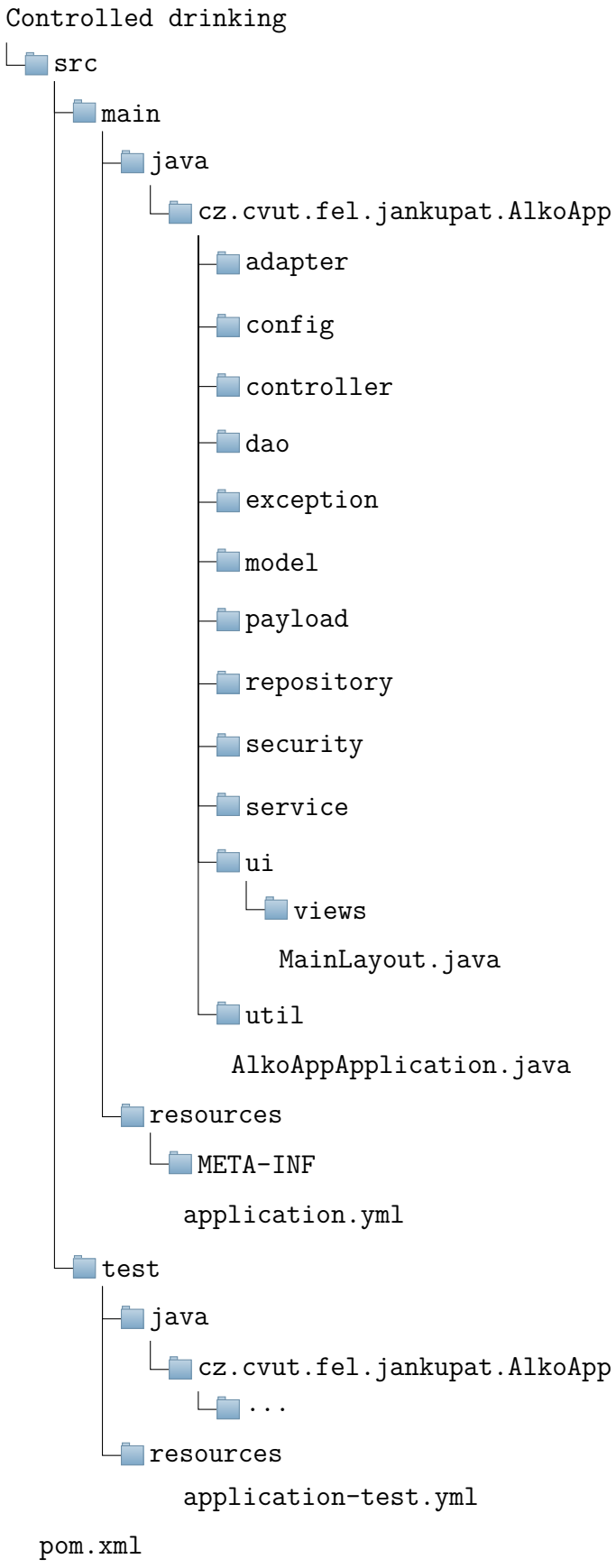
5.2 Štruktúra programu

Program je rozdelený do klasických javovských balíčkov. Koreňový *src* balíček obsahuje dve podriadené zložky *main* a *test*. *Main* obsahuje všetky súbory programu potrebné k úspešnému spusteniu aplikácie. Zložka *test* zahŕňa všetky testovacie súbory a konfiguračné súbory potrebné k spusteniu testov. To zahŕňa unit testy, implementačné testy a súbor *application-test.yml*, ktorý obsahuje konfiguráciu testovacej databázy. *Main* je tvorený tiež dvoma balíčkami *java* a *resources*. Resources obsahuje súbory, ktoré obsahujú prihlasovacie údaje do databázy a iné konfiguračné údaje. Balíček java je následne rozdelený do pod-balíčkov, ktoré rešpektujú flow architektúru Spring Bootu. Meno balíčka je zhodné s vrstvou, ktorú v architektúre predstavuje. Funkcia väčšiny balíkov bola opísaná v predošlom odstavci.

Okrem vyššie spomenutých balíkov sa v programe nachádza balík *config*, bez ktorého by bol beh programu nemožný. Obsahuje všetky konfiguračné súbory. Je zodpovedný za správne fungovanie zabezpečenia programu. Ďalej nastavuje fungovanie Swaggera a určuje aké metódy sú dovolené.

Balík *exception* obsahuje nastavenie výnimiek, ktoré sa pri nesprávnom behu programu vypíšu. Obsahom *security* sú servisy, ktoré sa využívajú pri zabezpečení programu. Od zvyšných servis sú oddelené kvôli prehľadnosti a odlišnom správaní ako ostatné. Zložka *payload* úzko súvisí so security, keďže tá využíva súbory v payload na správny beh zabezpečenia programu.

Balík *ui* sa odlišuje od iných tým, že ako jediný využíva funkcie frameworku Vaadin. Je tu definované UI rozhranie Back office. To sa spúšťa cez súbor *MainLayour.java* a ten následne využíva ďalšie obrazovky implementované v súboroch nachádzajúcich sa vo views. Hlavná trieda programu sa nachádza v súbore *ApkoApplication.java*. Na spustenie aplikácie je potrebné zavolať metódu *main()* v tejto triede.



Obr. 5.2: Stromový diagram predstavuje štruktúru programu rozdelenú do java balíkov. Funkcionalita balíkov sa riadi flow architektúrou Spring Boot, a preto má každý balík jedinečný účel v behu programu adekvátnej vrstve v diagrame. Okrem toho aplikácia obsahuje aj balíky zodpovedné za konfiguráciu, bezpečnosť alebo iné funkcie.

5.3 Dokumentácia

Na generovanie Swagger dokumentácie je potrebné do súboru pom.xml pridať dvojicu dvoch závislostí [55][56]. A potom zapnúť Swagger v hlavnej triede pomocou @EnableSwagger2. Výsledkom je HTML stránka na adrese /swagger-ui.html/. Stránka obsahuje zoznam adries URL poskytnutých backendom. Taktiež je poskytnutý model s atribútmi a požadovanými parametrami. Swagger umožňuje vykonávať požiadavky HTTP na server v prehliadači priamo zo stránky.

Dokumentáciu generovanú pomocou Javadoc nie je potrebné špeciálne nastavovať, keďže ide o vlastnosť jazyka Java. Avšak na rozdiel od dokumentácie vygenerovanej Swaggerom, ktorá sa automaticky upraví po skompilovaní kódu, je potrebné, manuálne vytvoriť príkaz na vygenerovanie Javadoc dokumentácie. Pri použití IDE[1] na generovanie je potrebné si dať pozor, aby verzia SDK použitá pri generovaní bola zhodná s SDK projektu.

```
1         <dependency>
2             <groupId>io.springfox</groupId>
3             <artifactId>springfox-swagger2</artifactId>
4             <version>2.9.2</version>
5         </dependency>
6         <dependency>
7             <groupId>io.springfox</groupId>
8             <artifactId>springfox-swagger-ui</artifactId>
9             <version>2.9.2</version>
10        </dependency>
```

Listing 5.1: Závislosti, ktoré je potrebné pridať do pom.xml na rýchle generovanie Swagger dokumentácie

5.4 Back office

Back office slúži na vizualizáciu údajov zozbieraných z mobilnej aplikácie. Ide o webovú aplikáciu a požiadavky na jej spustenie predstavuje iba potreba webového prehliadača. Vďaka tomu je zabezpečená multiplatformita. Tento nástroj je primárne určený pre adiktológov. Na správu databázy a úpravu údajov je potrebné využiť softvér tretích strán pre správu databáz napr. DBeaver. Jedná sa o frontendovú časť práce. Na vývoj som využil framework Vaadin, ktorého základne informácie som predstavil v predošlej časti. Keďže framework je postavený na Jave umožnil mi relatívny rýchly vývoj.

Profile details

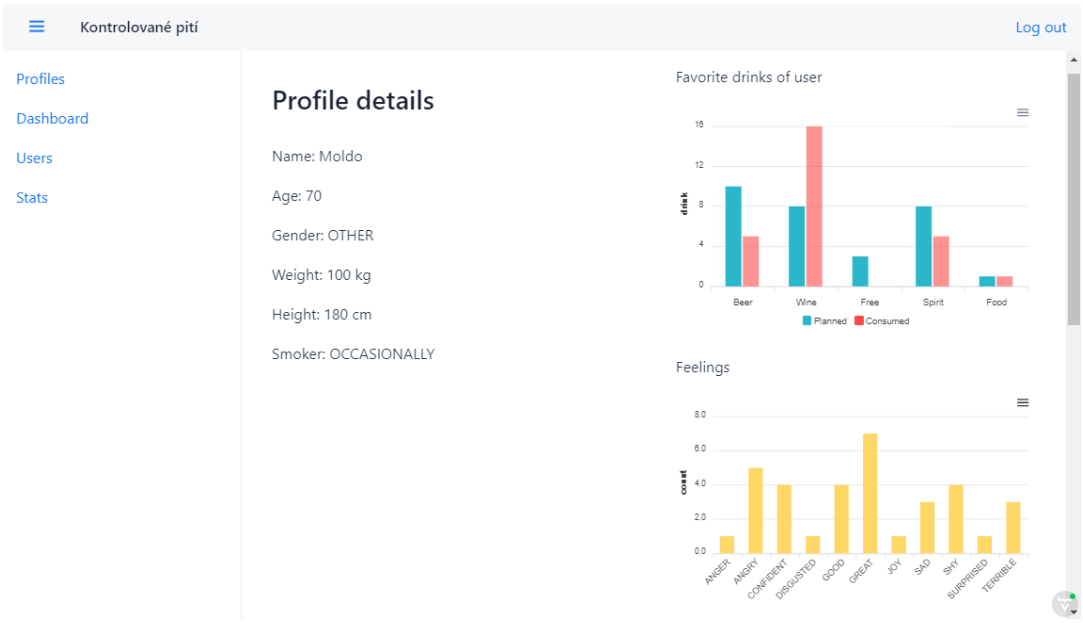
Táto stránka je zložená z viacerých komponentov. Úvod obsahuje údaje o užívateľovi, plus grafy s preferovanými nápojmi podľa druhu a náladami za celkové obdobie používania aplikácie.

Druhú časť stránky tvorí kalendár, ktorý je tvorený dvoma časťami. Prvá časť 5.4 je zložená z combox-u na výber mesiaca a integer fieldu na výber roku. Na základe údajov z týchto vstupných polí sa užívateľovi zobrazí mriežka so štvorčekmi rôznej farby. Farby vyjadrujú, či sa podarilo užívateľovi splniť plán pre daný deň. Štvorčeky

[1] Vývojové prostredie

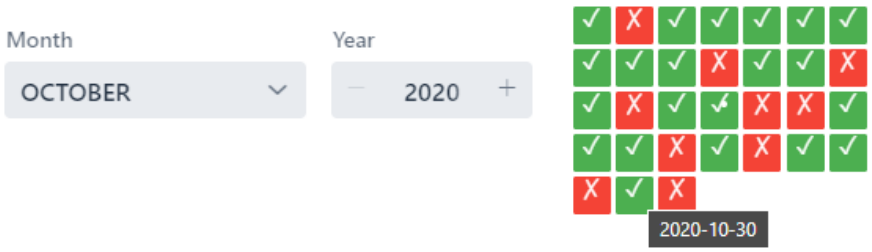
sú zelenej (splnený plán), červenej (nesplnený plán) alebo sivej (nie sú údaje o dni) farby. Po prejdení kurzorom na štvorček sa zobrazí dátum dňa, ktorý daný štvorček predstavuje. Štvorčeky sú realizované pomocou HTML elementu . V tejto časti sa nachádza potenciál na vylepšenie. Štvorčeky by mohli byť uložené do stĺpcov podľa dňa, tak ako v kalendári.

5.3

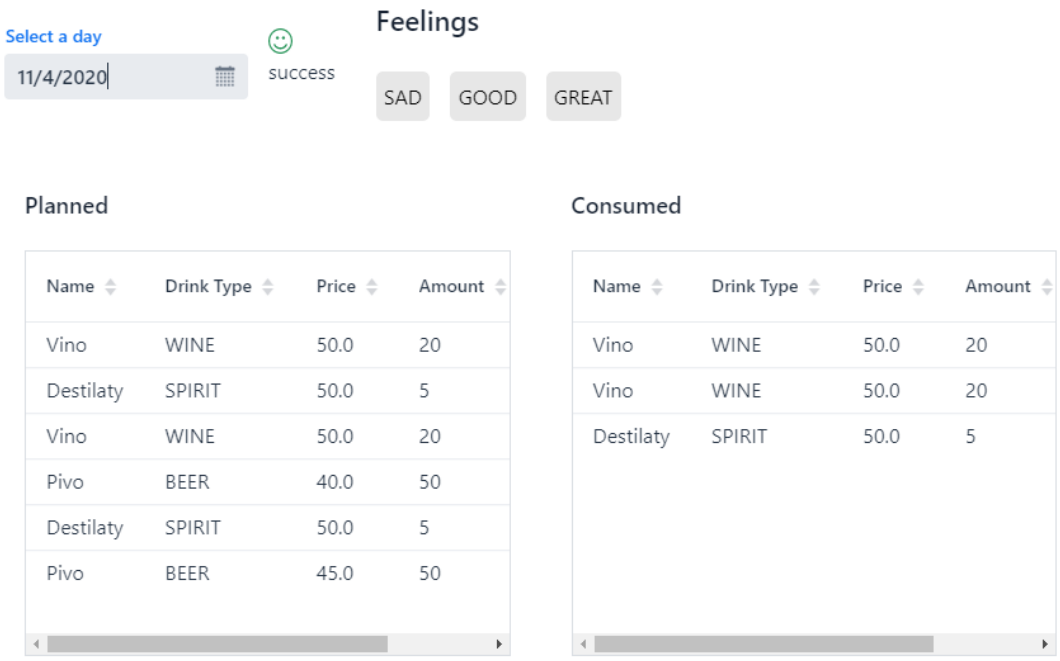


Obr. 5.3: Základné údaje o užívateľovi, grafy zobrazujúce početnosť druhov nápojov a nálad

Poslednú časť tvoria dva zoznamy 5.7. Ľavý zoznam obsahuje plánované nápoje pre daný deň, zoznam napravo obsahuje skonzumované nápoje. Adiktológ vyberie deň pomocou Vaadin komponentu date picker, ktorý sa nachádza hore nad zoznamami. Zároveň sa zobrazia aj nálady, ktoré mal užívateľ počas dňa.



Obr. 5.4: Kalendár zobrazujúci dni podľa plnenia plánov

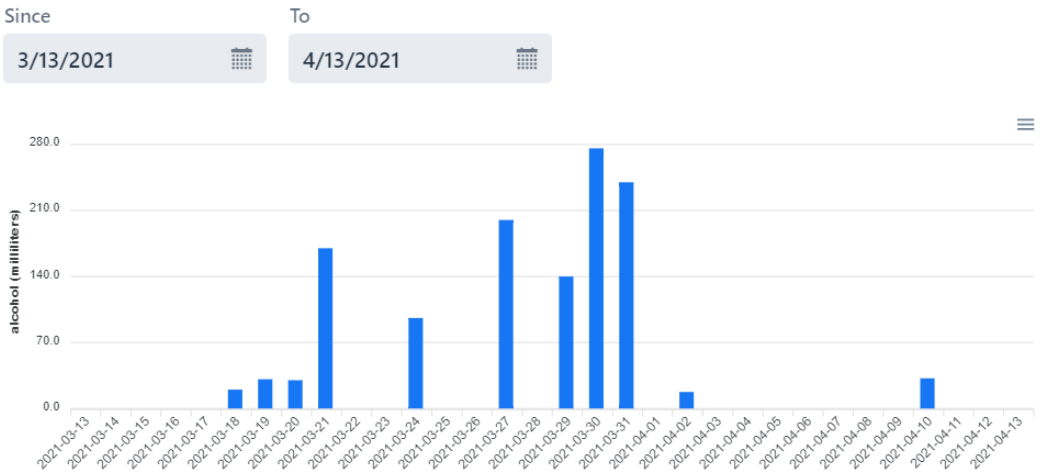


Obr. 5.5: Detailný zoznam nápojov a nálad podľa jednotlivých dní

Osi konzumácie alkoholu

Súčasťou obrazovky s detailom užívateľa sú aj dve časové osi, ktoré zobrazujú konzumáciu alkoholu užívateľa v čase. Objem alkoholu skonzumovaného predstavuje jediný objektívny údaj, ktorý je možné sledovať. Promile dopočítané na základe pohlavia a hmotnosti je veľmi nepresné, nakoľko každý jedinec reaguje na alkohol inak.

Prvá os je reprezentovaná stĺpcovým diagramom, kde výška stĺpca predstavuje objem alkoholu v mililitroch pre daný deň. Súčasťou grafu sú aj dva datepickre, ktoré slúžia na ohraničenie časového intervalu sledovaným grafom.



Obr. 5.6: Os zobrazujúca konzumáciu alkoholu počas dní

Druhá os slúži na vizualizáciu konzumácie počas dňa. Čiarový graf spája skonzumovaný alkohol počas jednotlivých hodín.

Obr. 5.7: Detailná os skonzumovaného alkoholu počas dňa

Kapitola 6

Testovanie

Táto kapitola obsahuje opis procesu testovania aplikácie. Zahrňa opis technológii a jednotlivých použitých testov.

6.1 Unit testovanie

Jedná sa o testovaciu metódu softvéru založenú na testovaných jodnotkách zdrojového kódu. V praxi to znamená, že sa testuje jeden alebo viacero modulov počítačového programu pomocou príslušných kontrolných údajov presne definovanými postupmi používania a prevádzkového nastavenia. Výstupy behu testu sú porovnávané s očakávanými výsledkami a následne sa rozhoduje o vhodnosti využitia modulov v aplikácii [57].

Spring svojou robusnosťou podporuje nástroje potrebné na realizáciu Unit testov v programovacom jazyku Java. Konkrétne ide o testovacie frameworky JUnit a TestNG, ktoré sú svojou popularitou najrozšírenejšie [58]. Tieto nástroje umožňujú programátorovi vytváranie mock objektov na testovanie častí kódu izolovane.

Ak sa vývoj riadi odporúčaniami pre architektúru Spring, výsledné prehľadné vrstvenie a komponentovosť veľmi uľahčí Unit testovanie. Objekty servisnej vrstvy je možné testovať pomocou stubbingu alebo mockingu DAO vrstvy a úložiska bez toho, aby ste museli pri spustení Unit testov pristupovať k perzistentným údajom v databáze. Správne Unit testy sa zvyčajne spúšťajú veľmi rýchlo, pretože nie je potrebné nastavovať infraštruktúru behu programu.

6.2 Implementácia Unit testov

Implementácia Unit testov v aplikácii je realizovaná pomocou JUnIT 4. Test pomocou JUnit vyzerá ako normálna java trieda a líši sa len anotáciou @Test nad deklaráciou metódy s obsahom testu. Všetky testy sa nachádzajú v balíčku test, ktorý je ďalej rozdelený do podbalíčkov na základe vrstiev, ktoré dané triedy testujú. Táto štruktúra programu pomáha sprehľadniť kontrolu a údržbu programu.

```
1     @Mock
2     DrinkItemDao itemDao;
3
4     @Mock
5     DayService dayService;
6
7     @InjectMocks
8     DrinkItemService itemService;
9
10    @BeforeEach
11    public void setUp() {
12        MockitoAnnotations.initMocks(this);
13        this.itemService = new DrinkItemService(itemDao, dayService
      );
14    }
15
16    @Test
17    void countItemsAndCheckPlanTest() {
18        Day day = new Day();
19        DrinkItem item1 = ObjectsGenerator.
      GenerateDrinkItemWithParameters(20.0, 50, 2, true, day, "beer");
20        DrinkItem item2 = ObjectsGenerator.
      GenerateDrinkItemWithParameters(20.0, 50, 2, true, day, "beer");
21
22        Set<DrinkItem> items = new HashSet<>();
23        items.add(item1);
24        items.add(item2);
25        day.setItems(items);
26
27        boolean result = itemService.countItemsAndCheckPlan(day);
28        assertEquals(true, result);
29    }
```

Listing 6.1: Ukážka testovacej metódy v DrinkItemServiceTest.java

6.3 Integračné testy

Integračné testy sú zamerané na integráciu rôznych vrstiev aplikácie. To znamená, že mockovanie nie je súčasťou integračného testovania. Optimálne je mať integračné testy oddelené od unit testov. Je to možné dosiahnuť vytvorením nového profilu špeciálne určeného pre integračné testy. Jedným z dôvodov je to, že integračné testy sú na rozdiel od unit testov časovo náročné a okrem toho často vyžadujú využitie databázy.

6.4 Testovacia databáza

Pri testovaní aplikácie je často potrebné overiť schopnosť ukladania dát do databázy a následnú možnosť spracovania CRUD[1] operácii. Na takéto testovanie je potrebné využiť databázu. Existujú niekoľko možnosti, ktoré je možné použiť. Jednou z nich je testovanie pomocou databázy, ktorá je využitá počas behu aplikácie. Tento prístup uľahčuje vývojárovi prácu, na druhej strane však dáta, ktoré sa počas testu v databáze vytvorili zostanú, ak ich vývojár na konci nevymaže. Testovacie

[1]Create, Read, Update, Delete

údaje môžu ovplyvniť beh aplikácie, čo môže mať negatívny vplyv na beh programu. Takýto prístup sa preto nevyužíva.

Ďalšia možnosť je využiť testovaciu databázu. Tá môže predstavovať reálny databázový server, ale vzhľadom k charakteru využitia môže byť vložená do aplikácie. H2 je open-source java databáza, ktorá môže byť vložená do java aplikácie, ako je tá napísaná pomocou Spring Boot.

KP využíva H2 iba na testovanie. Na využitie H2 je potrebné pridať závislosť do súboru pom.xml a vytvoriť nový konfiguračný súbor. Súbor test/resource/application-test.properties by mal obsahovať konfiguračné údaje H2 databázy. Dokument application-test.properties zadefinuje nový profil test, ktorý sa zaktivuje pomocou anotácie @ActiveProfiles("test"). Ak je nová konfigurácia aktívna, tak ta prepíše defaultne nastavenú konfiguráciu v main/resource/application.properties [59].

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

Listing 6.2: Závislosť, ktorá pridá H2 databázu do aplikácie

```
jdbc.driverClassName=org.h2.Driver
jdbc.url=jdbc:h2:mem:db;DB_CLOSE_DELAY=-1
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.
   H2Dialect
jdbc.username=sa
jdbc.password=sa
```

Listing 6.3: Časť súboru test/resource/application-test.properties, ktorej úlohou je nakonfigurovať H2 databázu použitú na testovanie

Záver

Cieľom práce bolo vytvoriť backend mobilnej aplikácie určenej pre znižovanie závislosti na alkohole. Tento cieľ sa podarilo splniť a výstupom práce je aplikácia *Kontrolované pitie*, ktorá poskytuje univerzálne REST rozhranie pre Android a iOS verzie aplikácie. Súčasťou riešenia je aj webová aplikácia Back office, ktorá slúži ako nástroj pre odborníkov z oblasti adiktológie.

V úvodnej časti práce som priblížil metódu kontrolovaného pitia, jej históriu a naznačil, pre koho je metóda vhodná. V analytickej časti som priblížil súčasný stav mobilnej aplikácie, opísal jej hlavné funkcie. Následne som zadefinoval funkčné a nefunkčné požiadavky, ktoré sú kladené na riešenie problému. Kapitola Technológie je obsahom teoretická a opisuje základne nástroje využité pri riešení problému. Snaží sa detailnejšie opísať Spring Framework a princípy, ktoré framework využíva. V nasledujúcej kapitole, tak ako aj názov napovedá, riešim celkový dizajn systému. To zahŕňa hlavne dátový model spolu s modelom nasadenia. V kapitole Implementácia opisujem praktické riešenie problému, štruktúru programu a spôsob akým sa jednotlivé vlastnosti programu nastavujú. Testovanie opisuje proces testovania. Ten sa primárne zameral na unit a integračné testy.

Mierným otáznikom je využitie Spring Bootu. Spring je primárne určený na tvorbu enterprise aplikácii. Spring svojou komplexnosťou predstavuje výzvu pre začiatočníkov, keďže má relatívne dlhú učiacu krivku. Tento faktor nehral v tomto prípade až tak dôležitú rolu, keďže s danou technológiou som mal už skúsenosť. Pokiaľ však má projekt fungovať aj v budúcnosti, bude potrebné zohnať ľudí, ktorí už danú technológiu ovládajú.

Do budúcna vidím hlavný potenciál hlavne v aktualizácii modelu a prispôsobeniu sa novej verzii aplikácie. Je predpokladané pridanie nových funkcionalít, ktoré bude potrebné zapracovať na backend. Zároveň je dôležité vyriešiť spôsob nasadzovania na server. Súčasné riešenie je nasadené iba na testovacom servery, ktorý sa využíval počas vývoja. Nové riešenie nasadzovania by malo byť jednoduché a čo najviac zautomatizované.

Bibliografia

1. KAMIL, Kalina et al. *Základy klinické adiktologie*. Grada Publishing as, 2008.
2. ORGANIZATION, World Health. *Global status report on alcohol and health 2018*. World Health Organization, 2019.
3. MANTHEY, Jakob; SHIELD, Kevin D; RYLETT, Margaret; HASAN, Omer SM; PROBST, Charlotte; REHM, Jürgen. Global alcohol exposure between 1990 and 2017 and forecasts until 2030: a modelling study. *The Lancet*. 2019, roč. 393, č. 10190, s. 2493–2502.
4. MRAVČÍK, Viktor; CHOMYNOVÁ, Pavla; NECHANSKÁ, Blanka; ČERNÍKOVÁ, Tereza; CSÉMY, Ladislav et al. Alcohol use and its consequences in the Czech Republic. *Central European journal of public health*. 2019, roč. 27, č. Supplement, S15–S28.
5. SOBELL, Mark B; SOBELL, Linda C. *Individualized behavior therapy for alcoholics: Rationale, procedures, preliminary results, and appendix*. State of California, Dept. of Mental Hygiene, 1972. Č. 13.
6. SOBELL, Mark B; SOBELL, Linda C. Alcoholics treated by individualized behavior therapy: One year treatment outcome. *Behaviour Research and Therapy*. 1973, roč. 11, č. 4, s. 599–618.
7. DAVID, Herel. *Mobilní aplikace pro kontrolované pití*. 2020. B.S. thesis. České vysoké učení technické v Praze.
8. ROSENBERG, Harold. Prediction of controlled drinking by alcoholics and problem drinkers. *Psychological Bulletin*. 1993, roč. 113, č. 1, s. 129.
9. SALADIN, Michael E; SANTA ANA, Elizabeth J. Controlled drinking: More than just a controversy. *Current Opinion in Psychiatry*. 2004, roč. 17, č. 3, s. 175–187.
10. VACEK, Jaroslav; VONDRÁČKOVÁ, Petra. Přístup harm reduction k užívání alkoholu. *Adiktologie,(12)*. 2012, roč. 2, s. 138–151.
11. JELLINEK, Elvin Morton. The disease concept of alcoholism. 1960.
12. SOBELL, Mark B; SOBELL, Linda C. Second year treatment outcome of alcoholics treated by individualized behavior therapy: Results. *Behaviour Research and Therapy*. 1976, roč. 14, č. 3, s. 195–215.
13. ADICARE. *ADICARE* [online]. [N.d.] [cit. 2020-12-30]. Dostupné z: `https://www.adicare.cz/zavislosti/terapeuticky-program-kontrolovana-konzumace-alkoholu/`.
14. *World wide web Technology Surveys*. 2009. Dostupné tiež z: `https://w3techs.com/`.

15. *Stack overflow developer SURVEY 2020*. 2020. Dostupné tiež z: https://insights.stackoverflow.com/survey/2020.

16. FIELDING, Roy T. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Irvine, 2000.

17. ALONSO, Gustavo; CASATI, Fabio; KUNO, Harumi; MACHIRAJU, Vijay. Web services. In: *Web Services*. Springer, 2004, s. 123–149.

18. FIELDING, Roy; RESCHKE, Julian. *Hypertext transfer protocol (http/1.1): Semantics and content*. rfc 7231, June, 2014.

19. MASSE, Mark. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. " O'Reilly Media, Inc.", 2011.

20. RICHARDSON, Leonard; RUBY, Sam. *RESTful web services*. " O'Reilly Media, Inc.", 2008.

21. ERL, Thomas; CARLYLE, Benjamin; PAUTASSO, Cesare; BALASUBRAMANIAN, Raj. *Soa with rest: Principles, patterns &constraints for building enterprise solutions with rest*. Prentice Hall Press, 2012.

22. JOHNSON, Rod; HOELLER, Juergen; DONALD, Keith; SAMPALEANU, Colin; HARROP, Rob; RISBERG, Thomas; ARENDSEN, Alef; DAVISON, Darren; KOPYLENKO, Dmitriy; POLLACK, Mark et al. The spring framework–reference documentation. *interface*. 2004, roč. 21, s. 27.

23. 01/11/2019, Tracy Watson; WATSON, Tracy. *Java Spring Framework – Pros, Cons, Common Mistakes*. 2019. Dostupné tiež z: https://skywell.software/blog/java-spring-framework-pros-cons-mistakes/.

24. *Advantages of Spring Framework With Limitations*. 2018. Dostupné tiež z: https://data-flair.training/blogs/advantages-of-spring/.

25. JOVANOVIĆ, Željko; JAGODIĆ, Dijana; VUJIČIĆ, Dejan; RANĐIĆ, Siniša. Java Spring Boot Rest WEB Service Integration with Java Artificial Intelligence Weka Framework. In: *International Scientific Conference "UNITECH 2017*. 2017, s. 323–327.

26. JOHNSON, Ralph E; FOOTE, Brian. Designing reusable classes. *Journal of object-oriented programming*. 1988, roč. 1, č. 2, s. 22–35.

27. FOWLER, Martin. Inversion of Control Containers and Dependency Injection pattern. *http://www. martinfowler. com/articles/injection. html*. 2006.

28. PRASANNA, Dhanji R. *Dependency injection*. Manning, 2009.

29. KICZALES, Gregor; LAMPING, John; MENDHEKAR, Anurag; MAEDA, Chris; LOPES, Cristina; LOINGTIER, Jean-Marc; IRWIN, John. Aspect-oriented programming. In: *European conference on object-oriented programming*. 1997, s. 220–242.

30. 2013. Dostupné tiež z: https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html.

31. TYSON, What is: Java By Matthew; TYSON, Matthew. *What is JPA? Introduction to the Java Persistence API*. JavaWorld, 2019. Dostupné tiež z: https://www.infoworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html.

32. *Spring vs Spring Boot vs Spring MVC - javatpoint*. 2019. Dostupné tiež z: https://www.javatpoint.com/spring-vs-spring-boot-vs-spring-mvc.
33. *Spring Boot Architecture - javatpoint*. 2018. Dostupné tiež z: https://www.javatpoint.com/spring-boot-architecture.
34. *Spring Security*. 2021. Dostupné tiež z: https://spring.io/projects/spring-security.
35. *Spring Security OAuth*. 2021. Dostupné tiež z: https://spring.io/projects/spring-security-oauth.
36. GRÖNROOS, Marko. *Book of vaadin*. Lulu. com, 2011.
37. *Vaadin 14 Docs*. 2021. Dostupné tiež z: https://vaadin.com/docs/v14/.
38. BAUER, Christian; KING, Gavin. *Hibernate in action*. Manning Greenwich CT, 2005.
39. FISHER, Paul Tepper; MURPHY, Brian D. *Spring persistence with Hibernate*. Springer, 2010.
40. STONEBRAKER, Michael; ROWE, Lawrence A. The design of Postgres. *ACM Sigmod Record*. 1986, roč. 15, č. 2, s. 340–355.
41. PORTS, Dan RK; GRITTNER, Kevin. Serializable snapshot isolation in PostgreSQL. *arXiv preprint arXiv:1208.4179*. 2012.
42. LOELIGER, Jon; MCCULLOUGH, Matthew. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.
43. *Documentation*. 2021. Dostupné tiež z: https://swagger.io/docs/.
44. *Swagger Inspector*. 2021. Dostupné tiež z: https://swagger.io/tools/swagger-inspector/.
45. *Swagger Codegen*. 2021. Dostupné tiež z: https://swagger.io/tools/swagger-codegen/.
46. *Swagger Editor*. 2021. Dostupné tiež z: https://swagger.io/tools/swagger-editor/.
47. *Swagger UI*. 2021. Dostupné tiež z: https://swagger.io/tools/swagger-ui/.
48. *New Collaborative Project to Extend Swagger Specification for Building Connected Applications and Services*. 2016. Dostupné tiež z: https://web.archive.org/web/20160427104213/http://www.linuxfoundation.org/news-media/announcements/2015/11/new-collaborative-project-extend-swagger-specification-building.
49. KRAMER, Douglas. API documentation from source code comments: a case study of Javadoc. In: *Proceedings of the 17th annual international conference on Computer documentation*. 1999, s. 147–153.
50. KROCHMALSKI, Jarosław. *IntelliJ IDEA Essentials*. Packt Publishing Ltd, 2014.
51. BOUDREAU, Tim; GLICK, Jesse; GREENE, Simeon; SPURLIN, Vaughn; WOEHR, Jack J. *NetBeans: the definitive guide: developing, debugging, and deploying Java code*. " O'Reilly Media, Inc.", 2002.

52. DEVILLARD, N. The eclipse software. *The messenger*. 1997, roč. 87, s. 19–20.

53. JONES, Michael; CAMPBELL, Brain; MORTIMORE, Chuck. JSON Web Token (JWT) profile for OAuth 2.0 client authentication and authorization Grants. *May-2015.{Online}. Available: https://tools. ietf. org/html/rfc7523*. 2015.

54. POINT, Java. *Spring Boot Flow Architecture*. 2018. Dostupné tiež z: `https://static.javatpoint.com/springboot/images/spring-boot-architecture2.png`.

55. RODRIQUEZ, F. *Maven Repository: io.springfox " springfox-swagger2*. 2021. Dostupné tiež z: `https://mvnrepository.com/artifact/io.springfox/springfox-swagger2`.

56. RODRIQUEZ, F. *Maven Repository: io.springfox " springfox-swagger-ui*. 2021. Dostupné tiež z: `https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui`.

57. HUIZINGA, Dorota; KOLAWA, Adam. *Automated defect prevention: best practices in software management*. John Wiley & Sons, 2007.

58. JOHNSON, Rod; HOELLER, Juergen; DONALD, Keith; SAMPALEANU, Colin; HARROP, Rob; RISBERG, Thomas; ARENDSEN, Alef; DAVISON, Darren; KOPYLENKO, Dmitriy; POLLACK, Mark; OTHERSD. 2021. Dostupné tiež z: `https://docs.spring.io/spring-framework/docs/current/reference/html/testing.html#unit-testing`.

59. 2021. Dostupné tiež z: `https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.external-config`.

Prílohy

A Obsah elektronickej prílohy

./kontrolované-pitie-src/ - zdrojový kód programu
./kontrolované-pitie-javadoc/ - dokumentácia kódu