České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů
Obor: Softwarové inženýrství a technologie

Webová aplikace pro zkvalitnění péče o pacienty

BAKALÁŘSKÁ PRÁCE

Vypracoval: David Hájek
Vedoucí práce: Ing. Petr Aubrecht, Ph.D.
Rok: 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hájek** Jméno: **David** Osobní číslo: **474556**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Webová aplikace pro zkvalitnění péče o pacienty

Název bakalářské práce anglicky:

Web application for better patient care

Pokyny pro vypracování:

Vytvořte webovou aplikaci, která by zkvalitnila péči o pacienty. Aplikace bude stát na třech základních pilířích. Provede pacienta nemocí, interaktivním způsobem pacienta o nemoci vzdělá a umožní mu monitorovat základní fyziologické údaje. Aplikace bude vyvinutá podle principů enterprise aplikací za použití technologií Java EE.
Postupujte následovně:
1) Prostudujte problematiku dostupné softwarové podpory zdravotní péče o pacienty.
2) Analyzujte konkurenci a vytipujte lékařské odvětví s nejvyšší pravděpodobností uplatnění.
3) Pro dané odvětví vytvořte návrh aplikace, která bude brát ohled na jejich skutečné požadavky a potřeby.
4) Proveďte implementaci webové aplikace za využití technologií Java EE a dodržte zásady enterprise aplikací.
5) Použijte RESTové rozhraní pro budoucí rozšíření i na jiné platformy (mobilní aplikace).
6) Navrhněte postup pro nasazení v realném prostředí.

Seznam doporučené literatury:

GONCALVES, Antonio. Beginning Java EE 7. Berkeley, CA: Apress, [2013]. Expert's voice in Java. ISBN 9781430246268
DASCHNER, Sebastian. Architecting modern Java EE applications. Packt, [2017]. Designing lightweight business-oriented enterprise applications in the age of cloud, containers, and Java EE 8. ISBN 9781788393850

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Aubrecht, Ph.D., Infor, s.r.o.

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

Datum zadání bakalářské práce: **12.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Petr Aubrecht, 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

Prohlášení

Prohlašuji, že jsem svůj semestrální projekt vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne ....

David Hájek

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Petru Aubrechtovi, Ph.D. za neocenitelné rady a pomoc při tvorbě bakalářské práce. Bez jeho cenných rad by práce rozhodně nedosáhla výsledné míry propracovanosti a odbornosti. Také bych chtěl poděkovat těm, kteří mi věnovali svůj čas při odborných konzultacích zejména proděkanovi 3. LF UK MUDr. Davidu Marxovi, MUDr. Evě Koblihové, vedoucí lékařce JIP v Ústřední vojenské nemocnici v Praze, a kolegovi Jiřímu Drahošovi, studentovi 3. LF UK.

David Hájek

Název práce:
Webová aplikace pro zkvalitnění péče o pacienty

Autor: David Hájek

Studijní program: Softwarové inženýrství a technologie
Obor: Softwarové inženýrství a technologie
Druh práce: Bakalářská práce

Vedoucí práce: Ing. Petr Aubrecht, Ph.D.
Katedra počítačů, Fakulta elektrotechnická, České vysoké učení technické v Praze
Konzultant: –

Abstrakt: Bakalářská práce se zabývá aplikací, která má za cíl zkvalitnit péči o pacienty. Chce toho dosáhnout tím, že minimalizuje čas, jenž pacient stráví v nemocnici. Postup práce spočívá v identifikování problémů v lékařství a najití jejich řešení v podobě aplikace. Výsledkem práce je webová aplikace, která zvyšuje kvalitu léčby skrze průvodce nemoci, personalizovaného vzdělávání a monitoringu. Aplikace je tvořena podle principů enterprise aplikací. Soustředí se na bariatrické zákroky, na nichž ověřuje koncept systému.
Klíčová slova: kvalitní zdravotní péče, enterprise aplikace, webová aplikace, software, bariatrie

Title:
Web application for better patient care

Author: David Hájek

Abstract: This bachelor's thesis deals with an application that aims to improve health care for patients. In order to achieve that, it minimizes the time that the patient spends in a hospital. The thesis procedure consists of identifying problems in medicine and finding their solutions in the form of application. The result of this thesis is a web application that increases the quality of patient treatment through disease guides, personalized education, and monitoring. The application is created according to the principles of enterprise applications and it concentrates on bariatric surgery on which it verifies the concept of the system.
Key words: quality health care, enterprise application, web application, software, bariatrics

Obsah

Kapitola 1

Úvod

1.1 Motivace

Stav českého zdravotnictví není v dobré kondici. Problémem číslo jedna je personální krize. V roce 2018 až 40 % nemocnic nebylo schopno poskytnout plnou lůžkovou kapacitu. [6] Podle serveru Zdravotnictví volá o pomoc chybí v nemocnicích aktuálně 967 doktorů a průměrný věk lékaře je bezmála 50 let. [39] V roce 2013 jsme měli čtvrtý nejvyšší počet návštěv lékaře na osobu na světě. V průměru člověk v Česku navštíví lékaře jedenáctkrát do roka. [38] Jedním z důvodů je i nedostatečná důvěra pacienta ve zdravotnictví. Ale nejedná se jen o české zdravotnictví. Celosvětově populace stárne a je potřeba se začít přizpůsobovat.

Jedním z řešení je minimalizovat čas, který pacient stráví v nemocnici. A to je hlavní cíl mé bakalářské práce.

1.2 Nápad

Základní princip by spočíval v tom, že by byla pacientovi v nemocnici nabídnuta služba, která by mu pomohla se skrz naši aplikaci zorientovat ve své nemoci. Aplikace provede pacienta léčbou, dohlédne na dodržení plánu a připomene všechno potřebné. Dále pacienta vzdělá interaktivním způsobem o jeho nemoci a popřípadě umožní jeho příbuzným lépe pochopit jeho zdravotní stav. Na závěr by aplikace umožnila monitorovat stav pacienta na dálku. Pacient by mohl ukládat cenné informace o sobě, jako je například teplota či stupeň bolesti, a zaznamenával by tak vývoj svého stavu. Lékař by mohl vše vidět a optimalizovat způsob léčby. Aplikace by byla vhodná hlavně pro nemoci se složitějším průběhem, typicky s plánovanou operací. Cílem by bylo minimalizovat čas, který pacient stráví v nemocnici, zvýšit informovanost pacienta a zlepšit léčbu ze strany pacienta.

1.3 Rozhovor s MUDr. Davidem Marxem

S proděkanem třetí lékařské fakulty Univerzity Karlovy, panem doktorem Marxem, jsme se sešli 14. 11. 2019, abychom mu já a můj kolega Jiří Drahoš představili náš nápad a vyslechli si jeho reakci. Jiří Drahoš je studentem třetí lékařské fakulty a pomáhal mi při kontaktu se světem medicíny. Zajímalo nás, co si o nápadu myslí a předpokládali jsme, že nás nasměruje na správnou cestu a doporučí odborníka na další konzultace.

Po delší úvodní prezentaci projektu, kdy bylo zapotřebí podrobně vysvětlit a popsat celou jeho podstatu, shledal doktor Marx jeho myšlenku zajímavou, pouze si nebyl jist, kde by se aplikace v celém rozsahu dala využít. Od začátku se přikláněl hlavně k jedné části, a to k personalizovanému vzdělávání (personalized education).

Jako příklad nemoci jsme při prezentaci uvedli zánět žlučníku. Ten byl ale vnímán jako ne zcela vhodný, neboť by se u toho nedal využít plný potenciál aplikace. Kde byla spatřena možnost reálného využití, byla naopak plastická chirurgie, bariatrická chirurgie nebo dlouhodobé nemoci, kde sice není potřeba operativní zákrok, ale určitě by se využil modul vzdělávání a ocenila by se i nasbíraná data. Mezi takové dlouhodobé nemoci patří Alzheimerova choroba nebo roztroušená skleróza. Další oblastí využití mohou být případně rehabilitace.

Jako místo pro další konzultace nám byla doporučena OB Klinika a.s. v Praze, kde se zabývají léčením obezity, případně Endokrinologický ústav Fakultní nemocnice v Motole. Bohužel se mi ani s jednou klinikou nepodařila sjednat konzultace. Ta nakonec proběhla v Ústřední vojenské nemocnici viz. kapitola 3.2.1.

1.4 Cíl bakalářské práce

Současná moderní medicína sestávající z mnoha oborů disponuje širokou škálou léčebných postupů, které se pochopitelně liší. Aby se hlavní téma bakalářské práce dalo uchopit, bude nejlepší, když si vytipujeme jedno odvětví, ve kterém základní princip aplikace ověříme. Po konzultaci s panem doktorem Marxem jsem se rozhodl, že pilotní projekt postavím pro bariatrickou chirurgii. Při bariatrickém výkonu se ocení minimalizace času, který pacient stráví v nemocnici. Je potřeba dodržovat přísné předoperační a zejména pooperační režimy, jakými jsou například dieta a různá vyšetření. [4] A určitě se využije i měření základních fyziologických dat, jako je hmotnost, BMI[1] nebo krevní tlak.

Cílem tedy je vytvořit webovou aplikaci, která by zkvalitnila péči o pacienty. Aplikace bude stát na třech základních pilířích. Provede pacienta nemocí, interaktivním způsobem pacienta o nemoci vzdělá a umožní mu monitorovat základní fyziologické údaje. Aplikace bude vyvinutá podle principů enterprise aplikací za použití technologií Java EE.

Nejdříve identifikujeme, co by pacientům nejvíce pomohlo při jejich léčbě. Na

[1] BMI představuje index tělesné hmotnosti

základě identifikovaných potřeb vytvoříme návrh. Podle návrhu vytvoříme rozhraní pro pacienty s nejdůležitějšími funkcemi. Vytvoříme i jednoduché administrátorské rozhraní pro doktory. Pro případné propojení s informačním systémem nemocnice neboli NIS a popřípadě s jinými platformami (mobil) se na závěr připraví RESTové rozhraní.

Kapitola 2

Bariatrická chirurgie

V dnešním moderním světě už málokdy strádáme hladem. Jídlo bereme vlastně jako samozřejmost a už se nepídíme za tím, kde se vzalo nebo co se za ním skrývá. Ztratili jsme s ním kontakt. Žijeme v rychlém světě, kde se práce a jiné aktivity upřednostňují před zdravým životním stylem. Takový přístup může u jedinců vyústit až v extrémní stav. Bariatrická chirurgie pak přichází na řadu jako poslední řešení jednoho takového stavu.

Extrémní stav, který řeší bariatrická chirurgie, je obezita. V současnosti se obezita považuje za jednu z nejrozšířenějších chorob speciálně ve vyspělých zemích. Prevalence neboli výskyt nadváhy se za posledních 100 let dostal z mizivých procent na více než polovinu evropské dospělé populace. Od 80. let se její výskyt v evropských zemích ztrojnásobil [4], což je alarmující. V obezitě hrají roli životní podmínky, životní styl, i prostředí, ve kterém se jedinec nachází. Většinou takový životní styl doprovází kladná energetická bilance, kdy je energetický příjem vyšší než výdej, a nízké sportovní aktivity. Obezitu definujeme jako chronické zánětlivé onemocnění, jejímž hlavním znakem je množení tuku. Za člověka s nadváhou se považuje jedinec, který má BMI (body mass index) větší než 25. [4]

Léčba obézních lidí je velmi komplexní a obsahuje různé postupy. Vždy je dobré začít konzervativními způsoby. Nicméně při selhání konzervativních metod s velkým množstvím relapsů a u morbidně obézních lidí s BMI nad 40 je jasným favoritem bariatrická chirurgie. Tato chirurgická disciplína má velmi efektivní vliv na dlouhodobou léčbu těžké obezity.[4]

S rozvojem miniinvazivních operačních metod neboli laparoskopie na konci minulého století zaznamenala bariatrická chirurgie velký nárůst. Zatímco v roce 1999 se provedlo ve světě 50 tisíc bariatrických zákroků, v roce 2017 to bylo víc než 690 tisíc zákroků. [4]

Abychom pochopili kroky, ke kterým přistoupíme v dalších kapitolách, je potřeba objasnit problematiku obezity a bariatrie jako takové. Proto v této kapitole představíme základní příčiny stavů, které bariatrie řeší. Identifikujeme věci, které je vhodné sledovat. Pochopíme, jak probíhá bariatrický zákrok a co je potřeba pro úspěšné vyléčení z obezity. To všechno si vysvětlíme, abychom pak byli schopni vybrat správnou technologii a postup při tvorbě aplikace.

2.1 Obezita jako nemoc

Obezitu tedy definujeme jako chronické onemocnění, při kterém dochází ke shromažďování tuku. U žen se považuje standardní obsah tuku v rozmezí 25–30 % a u mužů mezi 20–25 %. Obsah tuku ale není snadné změřit. Proto se pro identifikaci stupně obezity používá BMI index neboli index tělesné hmotnosti. Hodnota BMI se vypočítá podílem tělesné hmotnosti a druhé mocniny výšky. U dospělých je obezita považována s BMI indexem nad 30 a nadváha pak s BMI indexem vyšším než 25. [4]

Aby byl pacientovi doporučen bariatrický výkon, musí trpět závažnou obezitou. K indikaci takové obezity se používá opět index tělesné hmotnosti. Pro bariatrické výkony se doporučují pacienti s BMI nad 40. Taková obezita je životu nebezpečná a pacient musí jednat rychle. [4] Navíc zde selhávají standardní metody hubnutí.

Classification	BMI (kg/m^2)	Risk of comorbidities
Underweight	< 18.5	Low (but risk of other clinical problems increased)
Normal range	18.5 to 24.9	Average
Overweight	≥ 25	
Pre-obese	25.0 to 29.9	Increased
Obese class 1	30.0 to 34.9	Moderate
Obese class 2	35.0 to 39.9	Severe
Obese class 3	≥ 40.0	Very severe

	Waist circumference (cm)	
Comorbidity risk	**Women**	**Men**
Above action level 1	≥80	≥94
Above action level 2	≥88	≥102

Obrázek 2.1: Klasifikace obezity podle Světové zdravotnické organizace [7]

2.2 Rizika obezity

Abychom pochopili vážnost těchto nemocí, je potřeba si uvědomit, že s obezitou přicházejí různé komplikace. Obezita nejenom zhoršuje kvalitu života, ale také zvyšuje mortalitu. [4] Způsoby jakými může mít obezita negativní vliv na člověk je několik. Komplikace rozlišujeme na mechanické, kam patří například kloubní onemocnění, dušnost nebo spánková apnoe, a metabolické, kam patří diabetes, sterilita nebo deprese. [4]

2.3 Léčba

Při léčbě obezity lze využít konzervativní a operativní metody. Mezi konzervativní patří zkoumání stravovacích návyků a jejich změna. Operativní jinak řečeno bariat-

rické metody přicházejí na řadu, když selhávají konzervativní. Považují se za nejúčinnější a mají velmi pozitivní vliv na snížení hmotnosti, zlepšení krevního tlaku a léčbu cukrovky 2. typu.

2.3.1 Vyšetření

Při anamnéze se zjišťuje vznik obezity, poruchy příjmu potravy, deprese, návyky. V neposlední řadě se zkoumá také motivace pacienta ke změně. Pacient je vyšetřen a pokud si to jeho zdravotní stav žádá, je mu nabídnuta bariatrická chirurgie. Než ale podstoupí samotný zákrok, musí nejdříve projít sérii vyšetření, aby se ověřilo, jestli netrpí např. poruchou přijmu potravy.

Kontradikcí k bariatrickému zákroku může být několik. Hlavní podmínkou pro udělení povolení k zákroku je projití veškerých vyšetřeních a projevení vůle zhubnout. K častým kontradikcím patří právě psychický a mentální stav pacienta nebo jeho neúnostnost k provedení operace v celkové anestezii. [4]

Důležité je zdůraznit, že pacient musí prokázat hlavně vůli zhubnout. Pacient si musí být vědom, že bariatrie není kouzelná hůlka, která ho bez jeho přičinění a aktivního přístupu zbaví obezity. Pokud si těchto faktů není vědom, nemůže bariatrický zákrok podstoupit.

2.3.2 Zákrok

Bariatrická chirurgie funguje na principu chirurgického zmenšení obsahu žaludku. Využívá několika metod a postupů, mezi kterými se vybírá na základě konkrétních okolností. Samotný zákrok probíhá laparoskopicky a pacient může být propuštěn už druhý den domů.

2.3.3 Pooperační stav

Pacient může začít přijímat tekutou potravu už prvních pár hodin po operaci. Jelikož není schopen pozřít větší obsah jídla, váha jde drasticky dolů. Pacient dochází na kontroly.

2.3.4 Dieta

Základem je omezit objem jídla na jednotlivou porci zhruba 150 ml a jíst víckrát denně. Strava je nejdříve kašovitá. Kombinují se šetřící a redukční diety. Při jídle nepijeme, abychom nezaplnili zmenšený objem žaludku. Při tendenci k zvracení zmenšujeme porce a jí se až osmkrát denně. [4]

2.3.5 Metody bariatrické chirurgie

K nejjednodušším a nejčastějším metodám v bariatrické chirurgii se řadí bandáž žaludku. Jedná se o zákrok, kdy se žaludek obváže svorkou, která omezí jeho obsah. V posledních letech se neprokazuje jako nejvhodnější. Jednotlivé metody bariatrické chirurgie jsou: [4]

- Tubulizace žaludku (Sleeve gastrectomy) – je zákrok, při kterém je odstraněna velká část žaludku.
- Žaludeční bypass – je zákrok, při kterém je tenké střevo napojeno na začátek žaludku a žaludek je tím pádem přeskočen. Na rozdíl od tubulizace je žaludek ponechám na svém místě.
- Bandáž žaludku – je zákrok, při kterém svorka aplikovaná na žaludek zmenší jeho obsah. Jako jediná metoda je plně reverzibilní. Bohužel není moc účinná a často selhává.

2.4 Tubulizace žaludku

Tubulizace žaludku je postavena na principu chirurgického odstranění velké části žaludku. Jedná se o část, kde se objevují tzv. „hladové hormony“, které vytvářejí zažívací hormony (ghrelin) a způsobují pocit hladu. Žaludek dostane podobu trubice a vejde se do něj 120-150ml stravy. Odstraněná část žaludku se vyndá z břicha. Výsledným efektem této operace je, že pacient nepociťuje tolik hlad. [4]

Kapitola 3

Specifikace požadavků

V minulé kapitole jsme si vysvětlili, co to bariatrie je. Nyní je potřeba nasbírat a namodelovat konkrétní požadavky, které by aplikace měla splňovat. Díky kvalitní specifikaci požadavků budeme schopni přijít s relevantním návrhem, který nebudeme muset v pozdější fázi implementace tolik měnit a ušetříme tím čas a zdroje. Pokud nebudeme mít správné požadavky, je těžké vytvořit správné výsledné řešení.

3.1 Byznys cíle

Hlavní byznys cíle projektu máme definované už v zadání. Určují nám směr, jakým se má celý projekt odvíjet a čeho má dosáhnout. Byznys cíl popisuje, co má organizace očekávat, že dosáhne za určitý čas.[36] Hlavním byznys cílem bakalářské práce je zkvalitnit péči o pacienty a dosáhnout tak minimalizaci času, který pacient stráví v nemocnici. K nezanedbatelným benefitům projektu patří úspora časových kapacit lékařů a ostatního zdravotnického personálu, což by mimo jiné mohla být jedna z cest, jak řešit současnou personální krizi v českém zdravotnictví 1.1.

3.2 Konzultace s odborníky

Byznys cíle a požadavky je potřeba konzultovat se zainteresovanými osobami. Konzultace nám zvýší šanci, že aplikace pokryje co nejvíce skutečných problémů a stane se tak opravdu přínosnou. Rozhodl jsem se tedy oslovit některé bariatrické kliniky. S klinikou v Ústřední vojenské nemocnici se mi podařilo domluvit konzultaci. Sešel jsem se s MUDr. Evou Koblihovou, která je vedoucí lékařka na JIP a provádí bariatrické zákroky. Cílem konzultace bylo představit projekt, vyslechnout si dojmy paní doktorky, zjistit, jak probíhá celý proces léčby, a další informace. Konzultaci jsem vedl formou rozhovoru.

3.2.1 Ústřední vojenská nemocnice

Co si o projektu myslíte?

Líbilo by se to pacientům, jelikož by ocenili jednodušší zpracování celé tématiky na jednom místě. Interaktivní aplikace by v tomto směru mohla být užitečná. Pacienti mají v oblibě i jiné aplikace.

Nejčastěji prováděné zákroky u vás na klinice?

V Ústřední vojenské nemocnici se provádí hlavně tubulizace žaludku (sleeve gastrectomy) a baypass žaludku. Mezi jednotlivými zákroky se vybírá na základě konkretní situace a stavu pacienta (věk, ezofageální reflux[1], diabetes). Často se stává, že pacient má sám vlastní představu nad tím, co je správné a preferuje konkrétní variantu. Dále se doktoři často dostávají do styku s pacienty, kteří prodělali neúspěšnou bandáž žaludku a musejí ji opravovat.

Jaké fáze při bariatrickém zákroku můžeme identifikovat?

I. V první fázi si musí pacient obstarat veškerá potřebná vyšetření, která jsou k provedení takového zákroku potřeba a umožní vůbec zahájit jeho léčbu. Spadá sem interní, psychiatrické, klinické a nutriční vyšetření. Psychiatrické vyšetření se provádí, aby se prokázalo, že je pacient ochoten zhubnout a že si je plně vědom, co takový zákrok obnáší. Klinické vyšetření obsahuje rentgen, ultrasonografii a gastroskopii. Obstarat všechna vyšetření může zabrat až několik týdnů.

II. Do druhé fáze pacient postoupí, pokud projde všemi vyšetřeními z první fáze. Je stanoven termín operace a její metoda. Dále je pacientovi určena podmínka, aby zhubl cca 5 kilo. Tato fáze trvá dva měsíce.

III. V těsné předoperační době, kdy pacient splnil podmínku z druhé fáze, si musí obstarat předoperační vyšetření u svého praktického lékaře.

IV. Nástup na operaci je v den a nebo den před samotnou operací, přičemž vždy záleží na okolnostech. Následně proběhne operace. Z nemocnice je pacient propuštěn po dvou až třech dnech.

V. Pátá fáze trvá od propuštění z nemocnice do 30. dne od operace. V této fázi mohou nastat komplikace a musí se vyndat stehy (po 10–14 dnech od operace).

VI. V poslední fázi se chodí už pouze na kontroly. Kontroly jsou po 2, 6 a 12 měsících. Klienti, kterým se nedaří hubnout, na kontroly většinou přestávají chodit.

Jaké jsou milníky při léčbě?

První rok dochází k intenzivnímu hubnutí. Váha jde dolů a pacient se cítí dobře. Po dvou až třech letech dojde ke stagnaci i mírnému přibrání. Pacienti si zvyknout na nový jídelníček a „prasej“. Musí se najít příčina nárůstu váhy. Po dvou letech, pokud dojde k úspěšnému zhubnutí, dojde k povadnutí kůže a je potřeba abdominální plastika, která není hrazená pojišťovnou.

Je bariatrie hrazená v České republice pojišťovnou?

[1]Ezofageální reflux je mechanismus, který způsobuje pálení žáhy.

Ano, je.

Jaké komplikace mohou nastat?

Mezi klasické komplikace spojené s bariatrickým zákrokem patří pálení žáhy a zvracení. K vážným komplikacím řadíme krvácení, otevření ran a proniknutí střevního obsahu do dutiny břišní. Vzniká zánět a následně sepse[2], na kterou hrozí úmrtí.

Kdo všechno se podílí na léčbě?

O pacienta se stará bariatrický tým. Ten se skládá z psychologa, chirurga, nutričního terapeuta a diabetologa / internistu. Při kontrolách pacient navštěvuje všechny členy týmu.

O co se zajímáte a co měříte u svých pacientů při kontrole?

Ptáme se jich, jestli netrpí pálením žáhy, nezvrací, kolik kilogramů zhubli, kolik tekutin pijí denně a jaký je BMI. Dále se zajímáme, na jaká jídla se projevuje intolerance. Určitě se hodí měřit krevní tlak a obvod pasu.

Kdo může vidět data o pacientech?

Doktor má přistup do všech údajů v rámci NIS[3] krom psychiatrického vyšetření. Psycholog nebo nutriční terapeut ale nemají přístup skoro k žádným datům, jelikož nejsou doktoři.

Můžou mezi sebou sdílet doktoři informace shromažďované v aplikaci?

Nedokážu odpovědět.

Prostor na připomínky

Velké oblibě mezi pacienty se v poslední době těší kalorické tabulky. Stálo by za úvahu, jestli do aplikace neimplementovat deníček, kam by si pacienti zapisovali svůj jídelníček. V monitorovacím modulu by pacienti mohli také shromažďovat postřehy, které na sobě zaznamenávají.

Jaký je socioekonomický stav pacientů?

Většina pacientů vlastní chytré zařízení, na kterém by mohla být aplikace využívána.

Děkuji za Váš čas.

3.3 Požadavky

Byznys potřeby a cíle se transformují na byznys požadavky. Byznys požadavek nám říká, co se má stát a proč. Po studování bariatrie a konzultaci s odborníkem dávají smysl následující byznys požadavky.

Byznys požadavky

[2]Sepse je orgánová dysfunkce způsobená infekcí.

[3]NIS je zkratka pro nemocniční informační systém.

BR01 Jako doktor budu moct přidat nového pacienta.
BR02 Jako pacient budu moct vidět, v jaké jsem fázi nemoci.
BR03 Jako pacient uvidím, které nadcházející události mě čekají.
BR04 Jako pacient můžu zkontrolovat, že jsem splnil všechno.
BR05 Jako pacient potřebuji získat informace o své nemoci a její léčbě.
BR06 Jako pacient potřebuji vidět svůj vývoj, abych zůstal motivován v léčbě.
BR07 Jako pacient potřebuji napsat zprávu doktorovi.
BR08 Jako doktor budu moct odpovědět pacientovi.
BR09 Jako pacient potřebuji editovat svůj profil, abych mohl zaznamenat změny.
BR10 Jako pacient můžu zkontrolovat, že jsem si obstaral všechny vyšetření.

Systémové požadavky

SR01 Systém umožňuje nahrát pacientovi údaje jím zaznamenané.
SR02 Systém umožňuje komunikovat s externí službou, aby mohl ušetřit práci lékařům.
SR03 Systém bude pravidelně notifikovat pacienty o tom, co mají dělat, aby na nic nezapomněli.
SR04 Systém bude zabezpečený, aby ochránil citlivé údaje pacientů.

3.4 Základní dělení systému do modulů

Jak už bylo nastíněno v úvodu, aplikace má za cíl provést pacienta nemocí, dostatečně a hlavně kvalitně pacienta vzdělat a zaznamenávat údaje o pacientovi. Na základě definovaných požadavků jsem se rozhodl rozdělit aplikaci na tři základní části.

- Průvodce nemocí
- Edukační modul (Personalizovaná edukace)
- Monitoring

3.4.1 Webová aplikace pro pacienty

Hlavním bodem, se kterým budou pacienti interagovat, bude webová aplikace. Bude obsahovat hlavní funkcionality, které dají dohromady tři pilíře zmíněné výše.

Průvodce nemocí

Tento modul doslova provede pacienta nemocí. Sdělí mu, v jaké fázi nemoci se momentálně nachází. Připomene mu, v kolik si má vzít prášky nebo kdy má přestat jíst. Ujistí se, že pacient dodržuje stanovený režim. Nebo pomůže pacientovi správně

odcvičit. K výše zmíněnému pomůžou kontrolní to-do seznamy a propojení s edukačním modulem.

Pacient bude díky tomu lépe připraven. Minimalizují se tak případy, že by si pacient před operací zapomněl vzít např. prášek, a nebyl by schopen operaci podstoupit. Naopak se maximalizuje efekt léčby samotného pacienta. V první fázi bariatrické léčby viz. kapitola 3.2.1 se často stává, že pacient zapomene obstarat všechna vyšetření. Přijde poté zbytečně na kontrolu, kde tím pádem nemůže dostat termín operace. To je další příklad, kdy by aplikace mohla pomoct.

Edukační modul

Představte si scénář, kdy pacient přijde na vyšetření a tam mu je sdělen jeho zdravotní stav. Zjistí, že bude muset jít za dva měsíce na operaci a je mu vyčten seznam věcí, které bude muset udělat. Když přijde domů, půlku věcí zapomene a začne shánět informace po nedůvěryhodných zdrojích (internet, fóra atd.).

S tím chce bojovat edukační modul. Zde pacientovi přehledně a interaktivně zprostředkujeme informace o nemoci, kterou trpí. Informace budou poskytnuté přímo nemocnicí a pacient tak nenarazí na nepravdivé informace, díky kterým by mohl učinit rozhodnutí, která by měla špatný dopad na jeho léčbu. Navíc budou informace personalizované přímo pro dotyčnou osobu.

Monitoring

Monitoring bude umožňovat jednoduché zaznamenávání základních dat o pacientově stavu. Důležité je, že data bude nahrávat do aplikace pacient sám. Bude si tedy moct měřit stupeň bolesti, váhu nebo např. nahrávat fotky hojící se rány. Pacient bude mít díky tomu pocit, že je o něho lépe postaráno a lékař data využije při další kontrole pacienta. Pacient bude mít také k dispozici interaktivně zprostředkovaná data, kde uvidí svůj vývoj, a bude motivován pokračovat v léčbě.

3.4.2 Administrace

Pro správu pacientů a nemocí je potřeba udělat administrátorské rozhraní. Budou do něj mít přístup pouze zdravotníci a administrátoři aplikace. Cílem administrace je také odpovídat pacientům na dotazy.

3.4.3 Rozšíření na další platformy

Aplikace by měla umožňovat komunikaci i s jinými aplikacemi. Tento scénář by nastal v případě, kdyby byla možnost zjednodušit a zautomatizovat proces registrace

pacienta a propojit aplikaci s NIS[4]. Je třeba také počítat s rozšířením na jiné platformy. Největší smysl dává rozšíření na mobilní platformy. Oba případy by mělo pokrýt RESTové rozhraní viz. kapitola 6.12.

[4]Nemocniční informační systém

Kapitola 4

Analýza

Cílem této kapitoly je rozvést do většího detailu nasbírané požadavky a namodelovat je.

4.1 Konkurence

Pro specifické nemoci už některé softwary existují. Ku příkladu diabetici už své aplikace mají a jsou na vysoké úrovni. Lze uvést MySugr [13] nebo Vitadio [12]. Je tedy potřeba vybírat taková onemocnění, kde pomoc v podobě aplikace ještě chybí. Nicméně aplikace se stejným konceptem v České republice ještě není, ač můžeme najít jiné podobného charakteru.

Ulekare.cz

U lékaře je největší česká medicínská online poradna a průkopník v oblasti telemedicíny si klade za cíl zefektivnit a zkvalitnit české zdravotnictví [10].

Jejich byznys model je postaven na principu, kdy pacient, kterého něco trápí, položí dotaz online a zaplatí za něj poplatek. Cena se odvíjí podle toho, jak rychle chcete, aby vám doktor odpověděl. Na dotaz vám odpoví jeden ze 300 doktorů, kteří s U lékaře spolupracují. [10]

Online Ambulance

Online ambulance je unikátní internetový portál umožňující přehledným a snadným způsobem určit na základě popsané nabídky symptomů pravděpodobnou příčinu vašich zdravotních obtíží [9].

Zde popíšete vaše symptomy a aplikace vám ukáže diagnózu. Dále pak můžete oslovit jejich lékaře v online poradně. [9]

Ordinace.cz

Ordinace.cz je online poradna, kde se spojíte s lékařem poradíte se s ním ohledně vašeho zdravotního problému. [11]

Všechny předchozí projekty fungují na podobném principu. Položíte dotaz a

doktor vám poradí. Naše aplikace se liší především v tom, že je vázaná na konkrétní kliniku, kde se pacient léčí. Pacient se tedy nedostane na tuto platformu náhodou, ale je mu nabídnuta skrze kliniku. Cíl tedy není substituovat přímo lékaře ale zkvalitnit léčbu na konkrétní klinice. Naše aplikace v tomto směru vybočuje a nedává smysl ji více porovnávat.

V zahraničí podobné projekty existují. Nejpodobnější aplikace je projekt SeamlessMD ve Spojených státech amerických. SeamlessMD se také snaží minimalizovat čas, který pacient stráví v nemocnici [14]. Zdravotnictví se samozřejmě v zahraničí liší, proto nelze počítat s tím, že by se SeamlessMD dal využít v České republice.

4.2 Případy užití

Případ užití (use case) popisuje, jak člověk dosáhne cíle procesu. [15] Na základě specifikovaných požadavků v kapitole 3 jsem namodeloval následující případy užití. Dávají nám funkční představu o aplikaci.

Uživatel

UUC-01	Přihlásí se do aplikace
UUC-02	Edituje svůj profil

Doktor

DUC-01	Vypíše všechny nemoci
DUC-02	Vytvoří novou nemoc
DUC-03	Edituje svoje nemoci
DUC-04	Otevře detail nemoci
DUC-05	Vypíše seznam svých pacientů
DUC-06	Otevře detail pacienta –include–> DUC-05
DUC-07	Registruje pacienta nového/existujícího
DUC-08	Sleduje vývoj stavu pacienta
DUC-09	Odepíše pacientovi –include–> PUC-10
DUC-10	Vypíše všechny kurzy edukačního modulu
DUC-11	Edituje jednotlivé kapitoly kurzu –include–> DUC-10

Pacient

PUC-01	Dokončí registraci –include–> DUC-07
PUC-02	Vypíše své nemoci
PUC-03	Otevře detail nemoci –include–> PUC-02
PUC-04	Projde si vzdělávací program
PUC-05	Zobrazí si plán akcí
PUC-06	Zobrazí si detail akce –include–> PUC-05
PUC-07	Uloží do systému naměřenou teplotu
PUC-08	Uloží do systému fotodokumentaci
PUC-09	Zobrazí fázi nemoci, ve které se nachází
PUC-10	Položí dotaz doktorovi
PUC-11	Vyplní dotazník

Administrátor
AUC-01 Přidá lékaře

4.3 Studie proveditelnosti

Studie proveditelnosti (Feasibility study) je analýza, která vezme v potaz všechny relevantní faktory zahrnující ekonomický pohled, varianty, možnosti a pravděpodobnost úspěšné realizace. Projektoví manažeři využívají studii k tomu, aby mohli porovnat pozitiva a negativa projektu předtím, než do něj investují čas a peníze. [16] V tomto projektu je velmi náročné studii proveditelnosti určit. Špatně se vyčísluje opravdový přínos. Záleží na mnoha faktorech mimo jiné i na klinice, kde by se aplikace používala. Můžeme pouze nekonkrétně říct, jaké přínosy by aplikace měla.

Hlavním benefitem je samozřejmě zkvalitnění péče o pacienty. Pacienti budou lépe připraveni a léčba bude lépe probíhat viz. 3.3. Může se tak například zmenšit počet návštěv nebo zbytečných kontrol, kdy pacient není správně připraven. Pacient stráví v nemocnici méně času.

Nevýhodou je počáteční investice ze strany kliniky, a to hlavně časová. Doktoři operující na klinice by museli věnovat čas a energii při založení nové nemoci v systému. Jelikož jsou vytížení, je potřeba tomuto riziku věnovat pozornost a mitigovat ho tím, že přidanou práci doktorům co nejvíce zjednodušíme a z automatizujeme. Proto je vhodné počítat s variantou na napojení na NIS a jiné služby. Jak už bylo zmíněno v benefitech, tato počáteční investice by se měla vrátit na lépe připravených pacientech a tím pádem například na menším počtu návštěv.

Počáteční investice by v dlouhodobém měřítku měla být splacena. Na závěr této kapitoly sdílím některá data ze dvou studií, které měřily přínos u podobných projektů v zahraničí. Přináší věrohodnost do celého konceptu.

4.3.1 Navazující léčba na bariatrické zákroky

Studie proběhla v roce 2011 v Ontariu, Kanada formou telefonátů s pacienty, kteří podstoupili bariatrickou operaci. Pacientů bylo 46. Zaobírala se navazující léčbou se zaměřením na pacienta. 36 pacientů pokračovalo v léčbě v rámci specializovaného programu. Dotázání pacienti se shodli v tom, že pro navazující léčbu je klíčové, aby jim byla poskytnuta podpora a speciální personalizovaná péče, která jim pomůže držet správnou dietu a zůstat v dobré kondici. Dále pacienti ocenili monitorování a nabyli pocit, že jejich praktický a jiný jejich lékař, by neposkytli podporu a informace v takové kvalitě. [40]

4.3.2 Efekt vzdělávání na návštěvy lékaře v pooperační fázi

Studie se zajímala o vztah mezi vzděláním pacienta a rizikem nutného vyhledání lékařské pomoci v pooperační fázi. Studie dotazovala pacienty, kteří podstoupili bariatrickou operaci, mezi lety 2015 a 2016. Měřila úroveň všeobecného vzdělání

a základního povědomí o medicíně. Medicínské znalosti měřila formou lékařského testu, konkrétně testem REALM-SF[1]. Tuto úroveň vzdělávání porovnávala s množstvím komplikacích a náhlých výjezdů do nemocnic. Studie proběhla na 95 pacientech s výsledkem, že pacienti s nízkým stupněm vzdělávání a s nízkým dosaženým skóre v lékařském testu mají až třikrát větší riziko komplikací a náhlých výjezdů do nemocnice. Studie tak prokazuje, že by takovým pacientům prospěla nějaká forma dovzdělání, míněno v základním povědomí o lékařství, a snížila by riziko komplikací v pooperační fázi. [41]

[1]Rapid Estimate of Adult Literacy in Medicine – Short Form

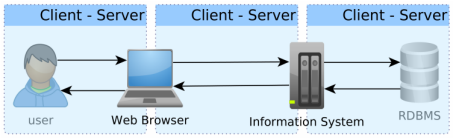
Kapitola 5

Návrh a struktura kódu

Na základě analýzy a namodelovaných požadavků jsem vypracoval následující návrh. Jeho cílem je ukázat základní rozdělení aplikace a jakých praktik vývoje se budeme držet. Nebude zde ale zacházeno do technických detailů implementační části.

5.1 Klient-server paradigma

S vývojem komunikačních technologií a osobních počítačů máme v dnešní době dva hlavní způsoby, jak mít aplikaci uloženou. Buď můžeme mít program uložen na osobním počítači a spouštět ho lokálně, nebo ho můžeme mít na centrálním místě na síti a klient slouží pouze jako prezentační rozhraní. [3] Rozhodl jsem se, že systém, který vytvářím, bude webová aplikace. To znamená druhá varianta. Dosáhneme tím centrálního uložení dat, vysoké kompatibility a dostupnosti napříč různými zařízeními a jednoduché aktualizovatelnosti. Aplikace, kterou popisuji, je postavena na architektuře klient-server pomocí protokolu HTTP. Architektura, jak můžete vidět na obrázku 5.1, funguje na principu, kdy klient (webový prohlížeč) posílá přes počítačovou síť požadavky na server, který je zpracovává a vrací odpověď. Tento model se také nazývá dvouvrstvý [18]. Klient je aktivní a posílá žádosti na server. Server je pasivní a vyčkává na síti až bude dotázán. Serverů máme několik druhů. Většinou to však bývají webové servery, databázové servery nebo e-mailové servery.



Obrázek 5.1: Klient server paradigma [17]

Za hlavní výhody této architektury se považují:

- Aniž by byl klient ovlivněn, dokážeme aplikaci snadno udržovat. Příkladem může být nová aktualizace.

- Ukládání dat centralizovaně zaručuje, že jsou data aktuální a dostupná.
- Servery bývají bezpečnější než většina klientských aplikací. [18]

Jako nevýhody můžeme zmínit následující:

- Jakmile přestane server fungovat, klienti nemohou být obslouženi.
- Architektura klient-server může být přetížena.

Po odeslaní požadavku klientem začne server se zpracováním. Jako odpověď vrací HTML stránku. Jelikož chceme stránky měnit na základě různých parametrů a ne jen vracet statickou stránku, budeme ještě potřebovat technologii, která nám to umožní a vytvoří tak dynamickou aplikaci. Technologií pro vytváření dynamického obsahu je hodně a mezi ty nejpoužívanější zařazujeme např. PHP, ASP.NET nebo Java EE.

5.2 Back-end a front-end

Front-end je část systému, která prezentuje uživateli finální výsledek. Běží na straně klienta. Front-end je zapojen ve všem, co klient vidí. Spadají do něj technologie HTML, CSS nebo JavaScript. [19] Back-end je na rozdíl od front-endu část aplikace, která se stará o vše, co klient nevidí. Přijme dotazy, zpracuje data, spojí se s databází a na závěr všechno pošle klientovi. Stará se o chod aplikace, ale uživateli je skryta.

5.2.1 Komunikace mezi back-endem a front-endem

Tradičně fungovala komunikace mezi back-endem a front-endem relativně přímočaře. Back-endová část zpracovala požadavek zaslaný klientem, vygenerovala html stránku a tu poslala klientovi. Ten ji pouze zobrazil. Se sílícími front-endovými technologiemi, jakými jsou hlavně JavaScriptové frameworky, se od toho principu oddalujeme a dáváme více zodpovědnosti klientské stráně. [3] Webovou aplikaci tedy můžeme rozdělit do dvou skupin podle toho, jakým způsobem aplikaci můžeme obsluhovat.

REST a SOAP služby

Aplikace s RESTovým rozhraním vrací jako odpověď datovou strukturu. Klientská aplikace, která přes RESTové rozhraní komunikuje s back-endem, je pak nezávislá a oddělená. Ke komunikaci využívají metody HTTP protokolu POST, GET, UPDATE a DELETE. Formát komunikace se dělí na JSON, XML apod. RESTové rozhraní nám umožňuje větší volnost při tvorbě klientské části, ale jeho nevýhodou je větší pracnost při zabezpečení a i samotné tvorbě klientské aplikace.

Generování GUI na serveru

Do této skupiny spadají aplikace, které generují HTML stránky dynamicky na serveru a přímo je vracejí jako odpověď. Klasickým příkladem je model MVC, kdy kontroler z modelu vytáhne potřebná data a do předdefinovaných šablon (view) vloží konkrétní obsah. Příkladem takové technologie jsou JSP, JSF a další.

Přestože moderní trendy směřují k spíše aplikacím využívající API přístup, které se více zaměřují na klientskou stranu (single-page aplikace apod.), je potřeba si dobře rozmyslet, jestli je takový přístup opravdu ten správný pro váš projekt. Jelikož moje aplikace nebude potřebovat časté dotazování na server, bude obsluhovat jen jednu klientskou aplikaci a bude potřeba mít kvalitní zabezpečení, rozhodl jsem se zvolit druhou možnost, a to konkrétně Java Server Faces. Když píši kvalitní zabezpečení, tak tím nemyslím to, že by se RESTové rozhraní nedalo kvalitně zabezpečit, ale je to náročnější. Pokud to projekt vyloženě nevyžaduje, je lepší se toho vyvarovat. Nicméně se v našem projektu připraví i RESTové rozhraní, kdyby mělo dojít k rozšíření na další platformy nebo ke komunikaci s jinou službou.

5.3 Enterprise aplikace

Systém pro zkvalitnění péče o pacienty má být aplikace pro nemocnice. Musí tak splňovat určité vlastnosti. Aplikace musí být výkonná, aby vždy zvládla vysokou zátěž. Musí být robustní, aby dokázala správně reagovat na všechny možné scénáře a nespadla při prvním špatně zadaném vstupu. A v neposlední řadě by měla být dobře škálovatelná a modifikovatelná. Aplikacím splňujícím zmíněné vlastností říkáme Enterprise Aplikace (podniková aplikace, EA, enterprise software) dále jen EA. Ty sice nemají formální definici, ale mají společné charakteristiky.[20] EA je softwarová platforma, která je navrhnuta tak, aby zvládla operovat v korporátním prostředí firem nebo státu. EA jsou komplexní, škálovatelné a distribuované. Většinou se skládají z komponent a jsou založeny na enterprise architektuře. [22] Enterprise aplikace řeší problémy enterprise organizací. [21] Musejí být spolehlivé, bezpečné a robustní. Často jsou EA navrženy tak, aby byly integrovány do jiných EA nebo s nimi komunikovaly, a to napříč různými síťovými technologiemi (internet, intranet apod.). Zároveň musí splňovat náročné bezpečnostní nároky. EA také operují nad velkými datovými sety.

„Enterprise applications are about the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data.“ – Martin Flower [5]

Softwarové řešení ve stylu EA se využívá pro velké organizace, kam přináší nástroje pro zefektivnění byznys procesů, zvýšení produktivity a zlepšení chodu spíše celé organizace než jedince. Příkladem takových organizací mohou být banky, školy, státní správy anebo dříve zmíněné nemocnice. Vývoj takových aplikací znamená tvořit robustní komponenty, které spolu spolupracují a jsou lehce upravitelné. Dojde-li tak např. ke změně databáze, přepojení na novou databázi by se mělo vyhnout větším úpravám aplikace. V EA často aplikujeme vrstvení. Kód rozdělujeme do nezávislých

vrstev. Důvod je pořád stejný. Snažíme se zachovat rozšiřitelnost a modifikovatelnost.

Je vhodné si uvědomit, že v dnešní době by se měly i malé firmy řídit principy enterprise aplikací. Podle Martina Flowera může být EA i menší aplikace.[20] EA tedy nemusí být jen pro velké korporace. Svět je propojen a i malá organizace bude muset zvládnout věci, které jsou specifické pro EA. Například budou muset zvládnout komunikaci s jinými enterprise aplikacemi nebo mít vysokou míru zabezpečení.

5.4 Byznys potřeby

Více než kdy dříve je právě v dnešní době potřeba se ptát, proč vyvíjíme, co vyvíjíme. Rychlost vývoje se zrychluje a až příliš často se stává, že programátor tráví moc času nad implementací funkcionality, která má malý přínos. [2] Dále je potřeba určovat vývoj aplikace podle byznys potřeb. Vývoj se nesmí řídit technologiemi. To, že využijeme bleeding edge[1] technologie, nemusí ve finále přinést žádné výhody. Snahou číslo jedna pro každého programátora by mělo být naplnění všech byznys požadavků, které klient vyžaduje.

5.5 Enterprise architektura

Enterprise architektura (EA) se skládá z praktik analýzy, návrhu, plánování a implementace za použití holistického principu[2] s cílem dosáhnout úspěšného vývoje a vytyčené strategie. [23] Enterprise architektura tedy zobrazuje, manipuluje a ukládá velké množství komplexních dat a podporuje a automatizuje byznysové procesy s daty. Musí splňovat následující požadavky:

- **Persistentní data** – relační databáze, grafové databáze, NoSQL
- **Komplexní datová integrace**
- **Souběžný přístup k datům** – od různých uživatelů a s různými scénáři
- **Různé vstupní rozhraní** – komplexní uživatelské rozhraní (formuláře, několik stránek), měřící data
- **Automatizace procesů** – propojení i s jinými službami
- **Výkon a robustnost**

Jelikož bude aplikace používána v lékařském prostředí, musí dostát výše zmíněným bodům. Především musí být robustní a bezpečná. Proto jsem se rozhodl řídit principy enterprise aplikací a enterprise architektury.

[1] Bleeding edge technologie jsou technologie, které jsou natolik nové, že jejich využití není ověřeno a je riskantní.

[2] Holistický princip znamená celistvý postup, kdy všechny části pracují dohromady jako celek.

5.6 Vícevrstvé aplikace

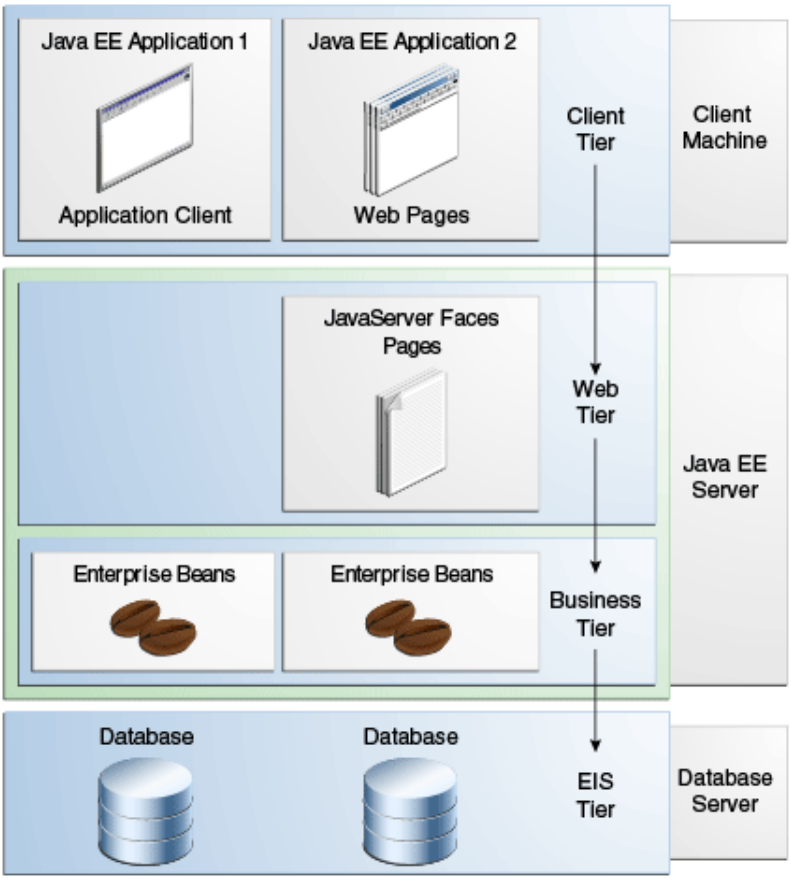
Dříve než se pustím do vysvětlení vícevrstvé architektury, kterou ve své aplikaci využívám, je potřeba rozlišit terminologicky logické vrstvy od fyzických vrstev. Je mezi nimi podstatný rozdíl a často se zaměňují. Logické vrstvy (layers) představují způsob, jakým strukturalizujeme kód. Fyzické vrstvy (tiers) nám zato rozdělují systém podle toho, kde je kód spouštěn [25].

Vícevrstvá architektura (**multilayered architecture**) tedy rozděluje kód do jednotlivých logických vrstev podle zodpovědnosti. Každá vrstva může přistupovat pouze k vrstvě právě pod ní (striktně), nebo ke všem nižším vrstvám (uvolněně). Vrstvy kódu (tzv. layers) dělíme do dvou kategorií – s horizontálním vrstvením a vertikálním. Vertikální dělení se řídí doménovým modelem. [2] V tomto projektu jsem se rozhodl pro architekturu horizontálního vrstvení. Na high-level úrovni dělím aplikaci na 3 základní logické vrstvy:

- **Prezentační vrstva** – prezentuje uživateli data, která získává z vrstvy byznys logiky.
- **Byznys logika** – obsahuje byznys logiku aplikace.
- **Persistentní vrstva** – stará se uložení dat, komunikuje s databází.

Aplikaci dělím i do fyzických vrstev. Řídím se tedy vícevrstvou architekturou (**multitier architecture**), která rozlišuje aplikační logiku do komponent podle funkčnosti [24]. V kapitole 5.1 jsem mluvil o dvouvrstvé architektuře v rámci klient-server paradigmatu. Dvě vrstvy nám většinou nestačí a musíme přidat třetí vrstvu. Třetí vrstva slouží jako vícevláknový aplikační server, který spojuje klienta a back-end úložiště. Aplikace se tedy dělí do tří fyzických vrstev (lokací): klientský počítač, aplikační server a databáze. [24] To už popisuji třívrstvý model, který využívá Java EE platforma. Přestože jsou vícevrstvé Java EE aplikace považovány za třívrstvé, ve skutečnosti se vrstva s aplikačním serverem rozděluje na webovou vrstvu a vrstvu byznys logiky. Java EE platforma tedy využívá čtyřvrstvý model, který můžete vidět na obrázku 5.2, a jednotlivé vrstvy jsou popsány následovně [24]:

- **Klientská vrstva** – běží na klientském počítači. V mém případě v prohlížeči.
- **Webová vrstva** – běží v aplikačním serveru.
- **Vrstva byznys logiky** – běží v aplikačním serveru.
- **Enterprise informační server EIS** – běží na databázovém serveru.



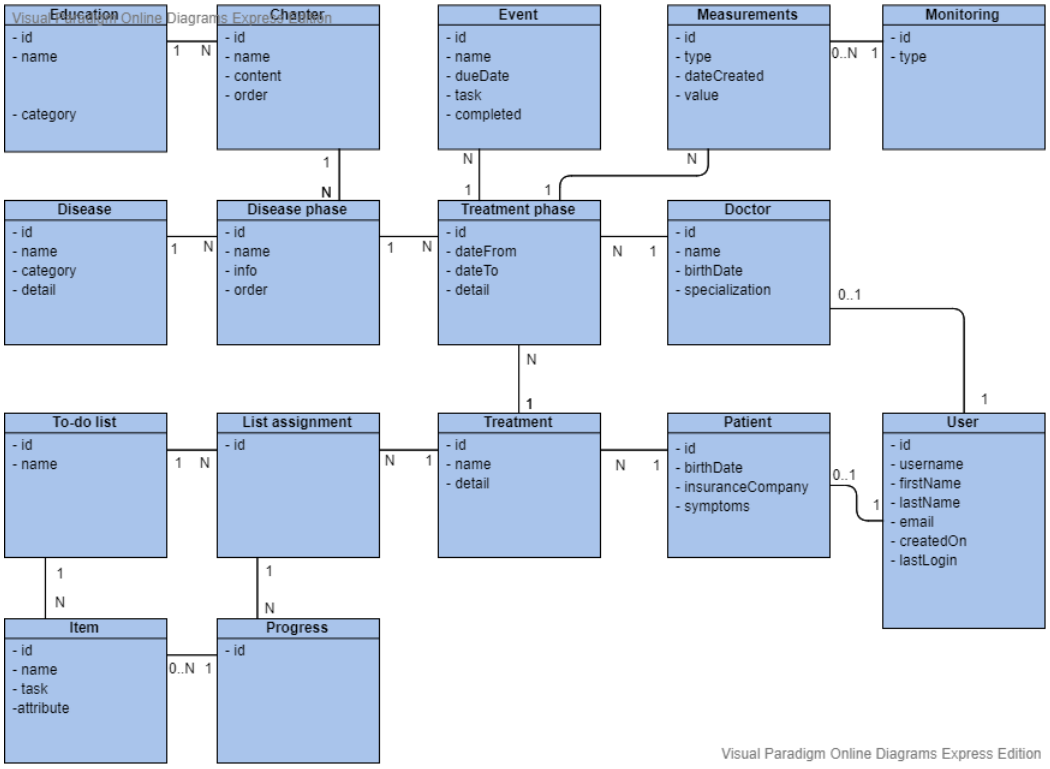
Obrázek 5.2: Vícevrstvá Java EE aplikace [24]

5.7 MVC

Softwarová architektura MVC (Model, View, Controller) se snaží oddělit aplikační logiku od prezentace. Tohoto principu se snažím držet napříč celou aplikaci. Jako model využívám JPA. K ovládání celé aplikace používám backing beany a servicy. Pro vykreslování pak používám Java Server Faces. Dosáhl jsem tak jasného oddělení.

5.8 Diagram tříd

V této sekci popisuji návrh tříd a vztahy mezi nimi. Vybral jsem k tomu diagram tříd, který je vyobrazen na obrázku 5.3. Popsal jsem ho pomocí UML jazyka. Jsou v něm uvedeny jednotlivé třídy, jejich atributy a kardinality. Ve fázi návrhu jsem mu věnoval více času, jelikož nám přímo definuje hranice aplikace a co s ní budeme moct dělat.



Obrázek 5.3: Diagram tříd (class diagram)

5.9 Databáze

Pro datovou vrstvu jsem se rozhodl použít databázi. V dnešním světě máme dva hlavní typy databází: **relační** (relation, SQL) a **nerelační** (NoSQL). Relační databáze mají pevně danou strukturu v tabulkovém schématu. No-sql je databázový koncept, který se snaží nahradit tabulkové schéma pomocí jiných prostředků. Za cíl má většinou vytvořit optimalizované úložiště, které se hodí například na velkou datovou zátěž.

Nevýhodou NoSQL je například to, že nemůžou dosáhnout plnohodnotného transakčního ACID[3] modelu.

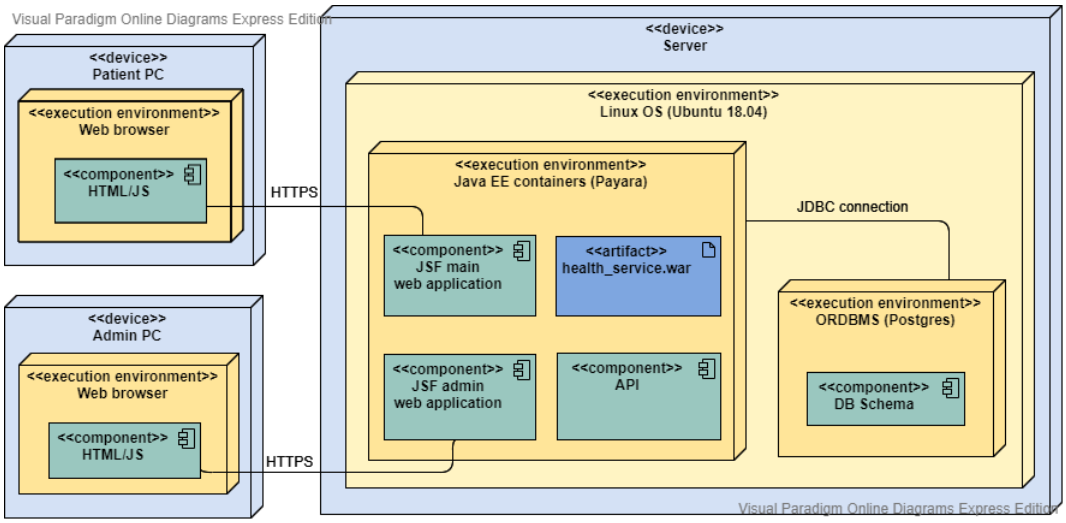
ACID vlastnosti jsou pro enterprise aplikace důležité a neexistuje žádný pádný důvod pro nerelační databázi. Relační databáze patří stále k nejčastějším databázím. Z těchto důvodů jsem se rozhodl pro relační databázi.

5.10 Nasazení

V této bakalářské práci simuluji případ, kdy je aplikace využívána pro bariatrickou chirurgii. V reálném chodu má projekt za cíl postarat se o mnohem více nemocí. Fungoval by následující model. Služby naší aplikace by využívala klinika, která by

[3]ACID je zkratka, která vystihuje 4 databázové vlastnosti – atomicita (atomicity), konzistence (consistency), izolovanost (isolation), trvalost (durability)

je poskytovala pacientům. Počet nemocí záleží na tom, kolik jich bude klinika požadovat. Při přidání nové nemoci bude akorát potřeba součinnost ze strany kliniky, aby se identifikovaly správné věci do jednotlivých pilířů viz. 3.4. Klinice by aplikace přinesla dva výrazné benefity. Zaprvé by šetřila čas lékařů a zadruhé by umožňovala klinice poskytovat pacientům nadstandardní služby, které by zvedly její prestiž.



Obrázek 5.4: Diagram nasazení

5.11 Cloud vs. monolitický server

Před nasazením je potřeba si rozmyslet, jestli aplikace poběží v cloudovém prostředí nebo v rámci monolitického Java EE serveru. Při položení takové otázky si musíme opět říct, co je pro nás opravdu potřeba. Moderní trendy opět ukazují na jeden směr, ale to neznamená, že je to ten správný pro náš projekt. Tím moderním trendem je cloudové řešení. V zásadě to představuje mikroservisní architekturu (microservise architecture). [3] Před rozhodnutím je dobré vzít v potaz následující body:

- Cloudová řešení se stále řídí klient-server paradigmatem. Mikroservisy jsou od sebe izolované a nezávislé. Komunikují mezi sebou strohým způsobem (např. pomocí JSON formátu).
- Java EE servery se stávají lehčími a vyžadují méně operační paměti. Jedna režijní instance zabírá zhruba 100 MB paměti. To na běžném serveru stačí na stovky instancí [3].
- Cloudová řešení jsou lépe škálovatelná než monolitická Java EE aplikace.
- Při cloudovém řešení často vsázíme na třetí stranu, která nám poskytne infrastrukturu. Znamená to ale, že naše data jsou zpracována někým jiným a vyžaduje si to velkou důvěru [3].

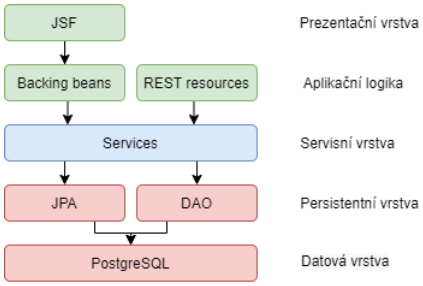
- Když vložíme cloudové řešení do rukou třetí strany, outsourcujeme jejich know-how a šetříme naše zdroje. Musíme ale počítat s rizikem, které souvisí s faktem, že jsme na třetí straně závislí [3].
- Mikroservisy jsou složitější na implementaci a případnou údržbu než single aplikace na monolitickém serveru. Musí se například zajistit distribuované transakce, tj. atomické operace nad databází.

Pro tento projekt by cloudové řešení přineslo největší benefit v podobě snadné škálovatelnosti. Mikroservisy se dají vícenásobně instancovat, paralelizovat. Zátěž se pak dá lépe rozložit na serverech. Jelikož aplikace nemá v dohledné budoucnosti za cíl obsloužit enormně velkou uživatelskou základnu a náhlé výkyvy nejsou očekávány, jelikož byznys model nepočítá například s auto-registrací, benefit v podobě škálovatelnosti není tak relevantní. Nevýhody oproti monolitické aplikaci představují složitá implementace a údržba. Dekompozice systému není snadná a když se provede špatně, může vyústit například v to, že při změně nějaké funkcionality musíme modifikovat všechny servisy, které jsou na ni vázané. Další nevýhodou mikroservis je, že mezi sebou musí komunikovat pomocí HTTP protokolu, a jsou tak pomalejší. V našem případě převažují u cloudového řešení nevýhody nad výhodami, a proto jsem se rozhodl pro monolitickou aplikaci. Monolitická aplikace se samozřejmě musí sledovat a kdyby se blížila výkonnostnímu limitu serveru, je potřeba začít uvažovat o migraci na cloud. V tom případě bude potřeba provést komplexní analýzu, aby se systém dekomponoval na správné komponenty a nevznikaly problémy při údržbě a dalším vývoji.

Kapitola 6

Implementace

V této kapitole popisuji, jak jsem přistoupil k samotné implementaci. Je tu rozebráno, jaké jsem použil technologie a některé implementační detaily. Zabývám se zde i zabezpečením a testováním. Použití vybraných technologií se snažím odůvodnit. Pracuji zde s rozdělením do podrobnějších logických vrstev, které v aplikaci využívám. Vrstvy jdou, jak říká obrázek 6.1, v následujícím pořadí: prezentační vrstva, vrstva aplikační logiky, servisní vrstva, persistentní vrstva a datová vrstva.



Obrázek 6.1: Podrobné logické vrstvy

6.1 Java

Hlavní technologie, která je v tomto projektu použita, je programovací jazyk Java, konkrétně verze 8. Byla to pro mne vcelku snadná volba. Java stále patří mezi jeden z nejpoužívanějších jazyků na světě, podle TIOBE indexu dokonce nejpoužívanějším [26], speciálně pak v odvětví podnikových aplikací (enterprise aplikací). Má velkou komunitu a díky tomu je dobře zdokumentovaná. Sice je její syntaxe rozsáhlá, ale s vhodným editorem je vývoj rychlý. Více kódu přináší na oplátku pořádek a kvalitní typovou kontrolu. Když se něco rozbije, víme většinou kde. Java obsahuje různé API, které dávají programátorovi více možností a šetří čas. Povíme si o nich později. Java je vysokoúrovňový jazyk, který nám umožňuje se soustředit na implementaci řešení byznys problémů.

Programovací jazyk Java je všestranný objektově orientovaný programovací jazyk. Je silně typovaný. Byl vytvořen tak, aby po zkompilování mohl být spuštěn

na jakékoliv platformě, kde je podporovaná Java. [27] Java aplikacím se přezdívá WORA (write onece run anywhere). Java soubor je kompilován do bytecodu, kterému se říká Java code. Má příponu .class a nese stejné jméno jako je název třídy v původním Java souboru. Java Virtaul Machine (JVM) bytecode interpretuje. Java Runtime Environment (JRE) vytváří minimální prostředí, ve kterém může být java aplikace spuštěna. Obsahuje JVM, základní třídy a podpůrné soubory. Základní vývojářské prostředí pro Javu se nazývá Java Development Kit (JDK). Obsahuje JRE, interpretr (Java), kompilátor (javac), archivátor (jar), generátor dokumentace (javadoc) a další. [29]

6.1.1 Platforma Java

Java je multiplatformní software nazývaný Platforma Java. Skládá se z několika softwarových produktů a specifikací (API). Na Javu můžeme narazit na mobilních zařízeních, vestavěných systémech, osobních počítačích, i na podnikových serverech. Platforma Java se skládá z následujících platforem:

- Java SE – obsahuje základní funkcionality.
- Java ME – jedná se o subset Javy SE pro mobilní nebo vestavěné systémy. Obsahuje malý JVM a hodí se pro IoT.
- Java FX – specializuje se na obsáhlý desktopové aplikace.
- Java EE – je standard pro vývoj Enterprise aplikací.
- Web profile – je subset javy EE pro webové aplikace.
- Micro profile – je subset javy EE pro mikroservisy. Obsahuje navíc Fault tolerance, metrics, health checks a JWT.

Jednotlivé platformy představují API knihoven. Využíváme je, abychom nemuseli ztrácet čas implementací často používaných funkcionalit a mohli se zaměřit na programování byznys logiky. Například Java SE nám poskytuje Collections API. Díky němu nemusím implementovat ArrayList a rovnou využít ten poskytovaný Java SE. Platformy Java SE, ME a FX častou slouží jako klient k Java EE aplikaci.

6.2 Java EE

Ze stejného důvodu, proč používáme Java SE, používáme i Java EE. Jak napovídá její název Java Enterprise Edition, poskytuje nám soubor specifikací k vývoji enterprise aplikací. Je postavena na Java SE a rozšiřuje ji. Znamená to, že Java EE je komplementární k Jave SE. Nabízí nám API k transakcím, messagingu, persistenci a dalším. Java EE poskytuje pouze rozhraní, samotná implementace záleží na kontejneru, který představuje runtime prostředí pro naší aplikaci nebo-li aplikační server. Stejně jako ostatní Java platformy skládá se z JVM a různých API. Obsahuje

všechny výhody, které má Java viz. v kapitole 6.1. Javu EE můžeme chápat jako standard.

Java EE byla v roce 2017 přesunuta pod Eclipse Foundation a její název byl změněn z Java EE na Jakarta EE. [3]

6.2.1 Spring

Spring je konkurenční technologie, která se zaměřuje na enterprise aplikace. Byla vytvořena, aby usnadnila jejich vývoj. V počátcích měla Java EE dříve známá jako J2EE několik problémů a spring přišel s řadou vylepšení a nových konceptů. Například zavedl Inversion of Control (IoC). Spring je modulární a na rozdíl od Java EE je to aplikační framework. [30] Mezi nejčastější moduly patří Core container, Spring security, data access, spring MVC a další.

Spring boot je též framework pro vývoj enterprise aplikací. Na rozdíl od Springu se soustředí na redukci kódu a zjednodušuje vývoj. Je tvořen jako stand-alone aplikace. Nemusíme nasazovat WAR soubor a aplikace se dokáže spustit sama. [31] Jeho součástí je totiž servletový server Tomcat.

6.2.2 Proč Java EE?

Z důvodů popsaných v kapitole 5.5 se řídím principy enterprise aplikací. Mezi populární EA technologie patří Python (Django), Java (Java EE, Spring) a PHP (Laravel). Jelikož jsem se rozhodl pro jazyk Java, zbyly mi dvě možnosti – Java EE a Spring.

Přestože byl Spring framework vyvinut, aby usnadnil vývoj java ee aplikací, Java EE nezahálela a Spring dohnala. Dnes jsou na podobné úrovni a záleží na konkrétním případě. Pro Java EE jsem se rozhodl v tomto projektu z následujících důvodů:

- Velmi snadná konfigurace
- Snazší rozšiřitelnost
- Rychlejší build-time

Největší výhodou Java EE je, že základní logika aplikace může být implementována s minimem kódu. [2]

6.2.3 Základ

Celý koncept Javy EE je založen na tzv. managed beanách. Rozlišujeme dva druhy: buď můžeme použít **Enterprise Java Beany (EJB)** nebo **Context and Dependency Injection for Java (CDI)** managed beany. Jejich rozdíl je popsán v kapitole 6.3. Záleží na konkrétním případě, který z nich použijeme. Čistá java v kombinaci

s jednou z typu bean nám tvoří moderní Java EE aplikaci. [2] Implementují se tak, že vytvoříme java třídu a přidáme jí anotaci.

6.2.4 Technologie

Celá Java EE je tvořena dílčími specifikacemi. Těmi hlavními jsou:

- Servlety – fungují na principu zpracování HTTP požadavku a vygenerování HTML stánky. Používají se při vývoji webových aplikací.
- Java Server Faces (JSF) – je modernější technologie k JSP. Umožňuje nám vkládat do XHTML šablon data pomocí speciálních znaků.
- Context a Dependency Injection (CDI) – umožňuje nám vkládat závislosti a udržuje kontext.
- Java Persitence API (JPA) – je objektově relační přístup k databázím.
- Enterprise Java Beans (EJB) – komponenty byznys logiky.

6.2.5 Aplikační servery

Aby mohla být Java EE aplikace spuštěna, nahráváme ji na takzvané aplikační servery. Ty běží na serveru a zpracovávají HTTP/HTTPS požadavky, spravují spojení s databází pomocí connection poolu, odesílají emaily atd. Vytvářejí naší aplikaci běhové prostředí (runtime environment). Aplikační servery jsou dodávány různými firmami, ale Java EE aplikace by měla být spustitelná na každém, který implementoval příslušné specifikace, je „Java EE kompatibilní“.

Open Source aplikační servery

- GlassFish
- JBoss
- TomEE

Komerční aplikační servery

- IBM WebSphere
- Oracle WebLogic Server

Já jsem se rozhodl pro aplikační server Payara, která vychází z aplikačního serveru GlassFish. Má dobré uživatelské prostředí a plnou implementaci, která představuje tzv. uvreference implementation. To znamená, že implementace na tomto aplikačním serveru ukazuje ostatním, jak má být nějaký standard, v našem případě Java EE, implementovaná. Payaru jsem upřednostnil před GlassFishem, jelikož je schopna rychle řešit problémy a poskytovat zpětnou vazbu.

6.3 CDI beany vs. EJB beany

Nejdříve je potřeba vysvětli, že nevybíráme explicitně jednu nebo druhou variantu. CDI je komplementární s EJB a pracují dohromady. Záleží tedy vždy na konkrétním příkladu.

CDI (Context and Dependency Injection) zajišťuje vkládání závislostí a udržení kontextu. Dovoluje nám tedy vkládat entity, aniž bychom museli vědět, co potřebují k jejímu vytvoření. CDI tedy aplikujeme, pokud potřebujeme injection, události, interceptory, decoratory, životní cyklus a další. CDI beana je definovaná jednou z následujících anotací: @ApplicationScoped, @SessionScoped, @RequestScoped, @ConversationScoped nebo @Dependent.

EJB beana může využívat všech výhody CDI bean, akorát přidává navíc kontejner, který se umí postarat o transakce, security, conccurency a pooling. Definujeme ji jednou z následujících anotací: @Stateful, @Stateless nebo @Singleton. Pro EJB se rozhodneme například při vytváření RESTového rozhraní (JAX-RS), nebo při práci s EntityManagerem. Umožní nám to používat transakce.

Abych to shrnul, CDI představuje základní beanu a EJB přináší více funkcionalit. Většinou začneme s čistým POJO, ke kterému přidáme CDI funkcionality, a když potřebujeme EJB funkce, definujeme beany jako EJB.

6.4 Scope

Každý objekt obsluhován CDI má svůj scope a životní cyklus. Beana je přiřazena konkrétnímu kontextu a zůstává tak dokud není zničena CDI kontejnerem. Neexistuje, aby se beana odstranila z kontextu manuálně. Máme následující scopy: [1]

- @ApplicationScoped
- @SessionScoped
- @RequestScoped
- @ConversationScoped
- @Dependent

EJB má následující scopy:

- @Stateless
- @Singleton
- @RequestScoped

6.5 Návrhové vzory v Java EE

6.5.1 Inversion of control

Při běžném programování používáme například jednu vybranou knihovnu a náš kód rozhoduje o tom, co je z té knihovny a jak použito. U Inversion of control (IoC) je to obráceně. Framework je ten, co volá náš kód. Většinou se tento návrhový vzor používá u frameworku. Framework rozhoduje o tom, kdy je co instancováno. IoC přesouvá odpovědnost za vznik vazeb na někoho jiného. DI je konkrétní implementace IoC.

Hollywood princip – „Don't call us. We will call you."

Snažíme se o následující body:

- Přehledný a rozšiřitelný kód
- Decoupling
- Moduly nemusí řešit, jak jsou jiné moduly vytvářeny
- Při výměně jedné části nevzniká problém

6.5.2 CDI

CDI zajišťuje vkládání závislostí (DI) a hlídá kontext. Beany vkládáme tzv. typesafe způsobem. To znamená, že nemusíme nic konfigurovat v XML souborech a stačí pouze anotace. [1] Když potřebujeme instanci beany, stačí použít anotaci @Inject.

6.5.3 Producer

CDI producer je funkcionalita Javy EE pro vytváření objektů. Nejčastěji se využívá při návrhovém vzoru továrny. Producer je implementován pomocí metody, která má anotaci @Produces. Návratový typ pak představuje objekt, který můžeme injectovat. CDI se o ostatní postará.

```
@Named
@Singleton
public class LoggerProducer
{
    @Produces
    public Logger produceLogger(InjectionPoint
       injectionPoint) {
        return Logger.getLogger(
                injectionPoint.getMember()
                .getDeclaringClass().getName());
    }
}
```

6.6 Build systemy pro Java EE

Hlavní úloha build systému je zkompilovat zdrojový kód a vytvořit artefakty. [2] Automatizuje celý proces sestavování aplikace a zajišťuje, že se vše sestaví ve správném pořadí. Správný build systém je spolehlivý a vytváří reprodukovatelné sestavení aplikace. Tato vlastnost je klíčová při zavádění Continues Delivery[1]. CD je pak využíváno u Continues Integration[2] serverech jako je Jenkins. [2] Pro Javu máme dva hlavní build systémy: Maven a Gradl. Já jsem se rozhodl pro Maven, jelikož je populárnější a má velkou komunitu.

6.7 Konfigurace

Často při vývoji narazíme na věci, které je nutno konfigurovat dynamicky. Kupříkladu potřebujeme znát cestu, kam ukládat obrázky, a ta se mění podle prostředí, ve kterém je aplikace spuštěna. Jedná se tedy o data, která se nemění sestavením a kompilací aplikace. Java EE nám k tomu umožňuje vytvořit ConfigExposer, který za pomoci návrhového vzoru producer nám dovolí číst z konfiguračního souboru. Data jsou uložena pomocí klíče a hodnoty.

6.8 EJB Timer

Ve světě UNIXu jsme všichni zvyklí spouštět periodické scripty pomocí CRONu. EJB třída nám umožňuje spouštět podobné scripty pomocí EJB timer.[amj] Nejčastěji se používá se Startupem, nebo se vytvoří persistentní, který je udržován synchronizovaný pomocí databáze.

```
@Singleton
@Startup
public class PeriodicJob {
    @Schedule(minute = "*/10", hour = "*", persistent =
       false)
    public void executeJob() {
        //code
    }
}
```

6.9 Datová vrstva

V kapitole 5.9 jsme si vysvětlili, proč jsem se rozhodl v **datové vrstvě** pro relační databázi. Dále bylo potřeba vybrat správný RDBMS (relational database manage-

[1] CD je metodika, která usnadňuje proces nasazení aplikace.
[2] Průběžná integrace je souhrn metod a nástrojů k urychlení vývoje.

ment system). RDBMS je software, který nám umožňuje nad fyzicky uloženými daty provádět operace. Známé databázové systémy jsou:

- **MySQL** je relační databáze pro menší projekty. Je součástí oblíbené kombinace technologií LAMP[3].
- **Oracle** je rychlý multiplatformní databázový systém.
- **PostgreSQL** je silný objektově-relační open-source systém pro řízení databází.

Oracle není open-source software. Umožňuje sice studentům stáhnout zdarma verzi, ale pro komerční užití se musí platit. Z tohoto důvodu jsem vybíral pouze mezi MySQL a PostrgeSQL. MySQL je jeden z nejpoužívanějších databázových systémů. Jeho hlavní výhodou je, že pro zpracování základních operací je rychlý. Děje se tak na úkor robustnosti. Je vhodný tedy především na malé projekty. Pro enterprise aplikace se více hodí PostgreSQL. Jedná se ve srovnání s MySQL o robustnější a kvalitnější řešení. Sice jsou základní operace o trochu pomalejší, ale jakmile dojde na navýšení zátěže, Postgres předčí MySQL. Je to dáno tím, že Postgres je postaven kvalitněji. Výhodou není jen lepší výkon při zatížení a škálovatelnost, ale i funkcionality a jejich kvalita.

	MySQL	PostgreSQL
Výkon	Rychlejší při menší zátěži	Při větší zátěži je rychlejší.
ACID	Vyhovuje ACID pouze na InnoDB a NDB	Vyhovuje ACID
Materialized view	Nepodporuje	Podporuje

Tabulka 6.1: Tabulka porovnávající MySQL a PostgreSQL [35]

Z výhod popsaných výše jsem se rozhodl pro PostgreSQL. Pro EA je vhodnější.

Zajímavostí Postgresu jsou sekvence. Pokud chceme vytvořit automaticky inkrementovaný atribut např. ID, musíme vytvořit sekvenci, kterou ke sloupečku přidáme. Sekvence si pamatuje posledně přidané číslo a při INSERTu vrátí nové.

6.10 Persistentní vrstva

Persistentní vrstvu jsem rozdělil mezi **JPA** (Java Persistent API) a dao vrstvu. Napojení na databázi je specifikováno v persistence.xml souboru, který je umístěn v META-INF adresáři. V moderní Java EE je to jeden z mála XML konfiguračních souborů, který je používán. Na konfiguraci databázového připojení nastavíme pouze odkaz pomocí JNDI(poznámka) name. Celá konfigurace je pak nastavená na aplikačním serveru. Oddělujeme tak konfiguraci databázového spojení od aplikace.

[3]LAMP je zkratka pro technologie Linux, Apache, MySQL a PHP

6.10.1 JPA

Jako model jsem se rozhodl použít Java Persistence API, který mi usnadňuje práci s daty nad databází. JPA totiž používá objektově relační mapování. Vytvářím tak entity, které se shodují s návrhem databáze. Jedna instance entity kopíruje jeden záznam (řádek) v tabulce.

Takové entity musí splňovat určité vlastnosti. Musí být anotována javax.persistence., musí mít public konstruktor bez parametrů, nesmí být final a další. K přenosu jiných dat mezi vrstvami pak používám DTO (Data Transfer Object).

6.10.2 DAO

Data Access Object (DAO) je strukturální návrhový vzor, který nám umožňuje izolovat vrstvu aplikační logiky od operací nad daty za pomocí abstraktního rozhraní. Cílem tohoto rozhraní je skrýt od zbytku aplikace komplexitu, kterou představuje provádění CRUD (poznamka pod carou) operací. Velkou nevýhodou je počet tříd, které vzniknou při použití tohoto návrhové vzoru. Každá entita musí mít vlastní DAO třídu.

6.10.3 Entity Manager

Entity a jejich životní cyklus jsou spravovány v persistence contextu. Entity manager představuje rozhraní pro komunikaci s persistence contextem. Mezi základní metody patří [2]:

- **persist** – Registruje entitu do persistence contextu a uloží ji do databáze.
- **flush** – Synchronizuje persistence context s databází.
- **merge** – Synchronizuje stav entity s contextem.
- **remove** – Odstraní entitu.

6.11 Java Server Faces

Pro prezentační vrstvu jsem se rozhodl využít technologii Java Serve Faces (JSF). JSF jsou soubory se speciálními XML značkami, které umožňují odkazovat na Java beany. Abych mohl stránkovat a provádět další potřebné úkony na základě URL, využívám značku <f:viewParam/>, která zpracuje GET parametry.

6.11.1 Composite components

Pro zvýšení přehlednosti kódu prezentační vrstvy jsem se rozhodl použít Composite components, které nám umožňují rozdělovat části šablon do znovupoužitelných komponent. Komponenta se skládá z rozhraní a implementace.

Musíme dát pozor při aktualizování komponent. Komponenty jsou tvořeny v UINamingContaineru, který každé komponentě přikládá vlastní unikátní id. [37] Důvodem je, aby se zabránilo konfliktům id, které by vznikly při vícenásobném použití. Proto je potřeba při odkazování v rámci jedné komponenty použít #{cc.clientId}, který nám unikátní identifikátor daný komponenty řekne.

6.11.2 Validátory

Validátor umožňuje validovat beany a jejich atributy. Validátor specifikujeme u atributu nebo metody pomocí anotace. Java EE nám poskytuje řadu předem definovaných validátorů. Pokud narazíme na případ, kdy potřebujeme vytvořit vlastní validátor, můžeme ho implementovat následujícím způsobem. V tomto konkrétním případě jsem potřeboval kontrolovat krevní tlak, který musí být ve formátu systolický/diastolický např. 110/80.

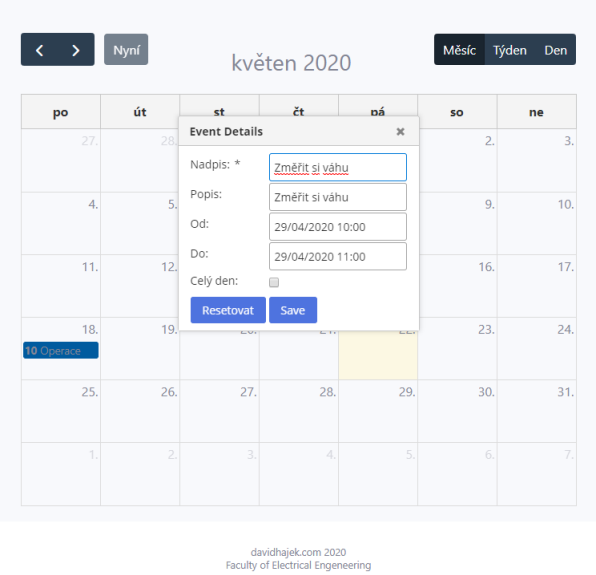
```
@Target({ FIELD })
@Retention(RUNTIME)
@Documented
@Constraint(validatedBy = { BloodPressureValidator.class
   })
public @interface BloodPressure {
    String message() default "Error message";
    Class<?>[] groups() default { };
    Class<? extends Payload>[] payload() default { };
}
```

6.11.3 PrimeFaces

PrimeFaces představují framework pro JSF. Poskytují nám hotové komponenty, které usnadňují vývoj a šetří čas. Příkladem může být komponenta kalendář, kterou si můžete prohlédnout na obrázku 6.2. Problém nastává při implementaci vlastního stylování. PrimeFaces obsahují různé designové šablony, ale ty nejsou k dispozici zdarma.

6.12 REST aplikace

RESTové rozhraní je postaveno na endpointech. Ty zpracovávají požadavky na základě kombinace url adresy a HTTP metody. V Javě EE používáme **Java API for**



Obrázek 6.2: Komponenta vytvořená frameworkem PrimeFaces

RESTful web services (JAX-RS). Endpointy jsou implementovány pomocí klasického POJO[4] a anotací @Path. Anotací @Produces specifikuje návratový MIME type[5] a anotací @Consumes definujeme MIME type, který dokážeme zpracovat. K zabezpečení používám JWT token, který ověřuji pomocí filtru. Při každém požadavku se podívá do hlavičky požadavku, jestli obsahuje validní token. Implementoval jsem ho následovně.

```
@NameBinding
@Retention(RUNTIME)
@Target({TYPE, METHOD})
public @interface JWTTokenNeeded {
}


@Provider
@JWTTokenNeeded
@Priority(Priorities.AUTHENTICATION)
@Stateless
public class JWTTokenNeededFilter implements
   ContainerRequestFilter {
    @Override
    public void filter(ContainerRequestContext
       requestContext) throws IOException {
                //validate request
        }
}
```

[4]POJO je zkratka pro plain old Java object
[5]MIME type je typ internetového média.

6.13 Zabezpečení

Zabezpečení je nesmírně důležitý krok, obzvlášť mluvíme-li o citlivých datech pacientů. Při vytváření enterprise aplikace je důležité při zabezpečování dodržet několik principů. Jako nejlepší řešení se jeví neimplementovat zabezpečení vlastními silami, ale využít už fungující řešení. Bez znalostí expertů by bylo riskantní implementovat vlastní řešení. Moderní enterprise frameworky proto většinou zabezpečení už mají.

6.13.1 Moderní principy zabezpečení

Moderní webová aplikace musí být dostatečně zabezpečená, aby odolala moderním hackerským útokům. Je zapotřebí dodržovat řadu principů, které zabrání jejímu prolomení. Mezi klasické útoky, na něž musíme být připraveni, považujeme SQL injection, cross site request forgery (CSRF), cross-site scripting (XSS), man in the middle a odhalení přihlašovacích údajů uživatelů. Abychom předešli těmto a dalším útokům, je vhodné se držet těchto pravidel:

- Zašifrovaná komunikace – Standardní způsob šifrování dat u webových aplikací při komunikování pomocí protokolu HTTP je TLS nebo SSL. Nikdo tak nemůže číst komunikaci mezi serverem a klientem.
- Správně se starat o přihlašovací údaje – Při ukládání hesel je potřeba dbát na to, aby hesla nebyla v plaintextu a ideálně použit při šifrování sůl.
- Správně se chovat k citlivým datům – Citlivá data bychom neměli ukládat v repozitáři.
- Ošetření dat – Vstupní data bychom měli ošetřit, aby nezpůsobily XSS nebo SQL injection.
- Zabezpečit REST – RESTové rozhraní je potřeba řádně zabezpečit, jelikož je náchylné na prolomení. Ideální způsob zabezpečení je pomocí JWT a CSRF tokenu.

6.13.2 Payara file realm

Realm představuje pravidla zabezpečení. Obsahuje uživatele, kteří mohou a nemusí být přiřazeni k nějaké skupině. Jelikož jsem se rozhodl používat aplikační serveru Payara, můžeme přímo na serveru definovat file realm. V něm definujeme a spravujeme uživatele a skupiny. Přihlašovací údaje jsou schovány na serveru a s aplikací nemají nic společného. Nemusíme zajišťovat hashování hesel v databázi a vytvářet vlastní autentizační mechanismus.

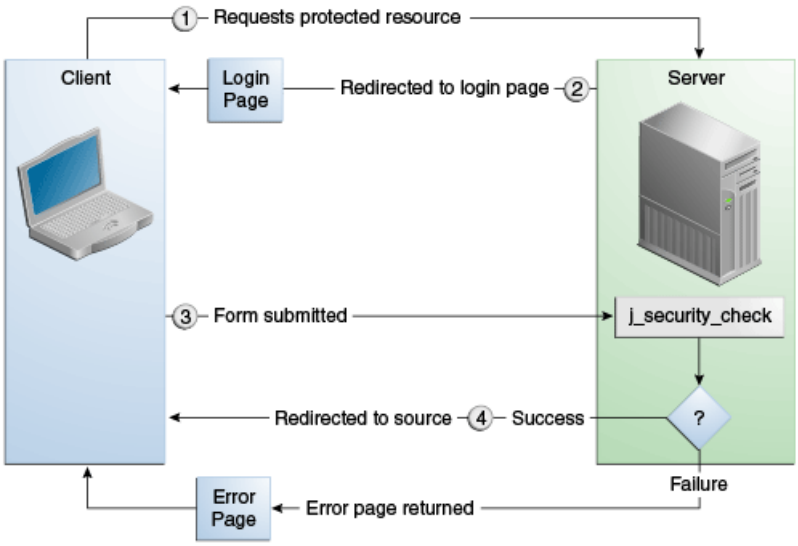
V aplikaci nastavíme zabezpečení v souboru web.xml, kde přidáme security-constrain. Role-name v auth-constraint nám říká, která role má pod danou url přístup.

```
<security-constraint>
        <web-resource-collection>
                <web-resource-name>Patient
                   pages</web-resource-name>
                <url-pattern>/patient/</url-pattern>
        </web-resource-collection>
        <auth-constraint>
                <role-name>patient</role-name>
        </auth-constraint>
</security-constraint>
```

Ještě musíme nastavit, jakým způsobem a kde se budeme autentizovat, tz. autentizační mechanismus. Java EE podporuje následující autentizační mechanismy:

- Basic authentication
- Form-based authentication
- Digest authentication
- Client authentication
- Mutual authentication

Já jsem se rozhodl pro Form-based autentizační mechanismus. Jeho princip popisuje obrázek 6.3.



Obrázek 6.3: Form-based autentizace [42]

```
<login-config>
        <auth-method>FORM</auth-method>
        <realm-name>file</realm-name>
```

```
    <form-login-config>
        <form-login-page>/login.xhtml</form-login-page>
                <form-error-page>/loginerror.xhtml</form-error-page>
        </form-login-config>
</login-config>
```

Ještě je zapotřebí přidat glassfish-web.xml soubor do adresáře WEB-INF. Tento soubor slouží k tomu, abychom propojili file realm na aplikačním serveru s aplikací. Nastavíme v něm, které role patří do jaké skupiny.

```
<security-role-mapping>
    <role-name>patient</role-name>
    <group-name>patient</group-name>
</security-role-mapping>
```

Tento druh zabezpečení se hodí pro vývoj a pro domácí projekty. Pro komplikovanější aplikace s větší uživatelskou základnou bude potřeba vyměnit file realm za JDBC realm nebo za LDAP autentizace[6]. Tuto operaci můžeme přesunout na administrátora serveru.

6.14 Monitorování a loggování

Nasazením aplikace práce nekončí. Musíme mít možnost kontrolovat, že chod aplikace funguje podle našich představ. Snažíme se hlavně zachytit chyby a monitorovat výkon naší aplikace.

6.14.1 Monitorování

Aplikace je potřeba monitorovat, abychom dokázali odhalit výkonnostní problémy. Můžeme tak kontrolovat, jestli naše aplikace není přetěžovaná, popřípadě lze na přetížení reagovat. Payara server v nejnovějších verzích umožňuje monitorování.

6.14.2 Logování

Logování ukládá informace o běhu aplikace na pozadí. Nejčastěji se snažíme odchytit chyby. Při vytvoření chyby není vhodné uživateli ukazovat jakékoliv detaily, jelikož by to představovalo bezpečnostní riziko. Zároveň je dobré informaci o chybě uložit, aby se programátor mohl dozvědět podrobnosti. Při vytvoření jakékoliv chyby

[6]LDAP je odlehčený protokol, který umožňuje autentizaci.

je uživatelovi vygenerována pouze univerzální zpráva o chybě, její detail je uložen v logu. Pro logování vlastních zpráv využívám třídu Java.util.logging.Logger, kterou vkládám do CDI spravovaných bean. Všechny zalogované informace můžeme najít v logu serveru. Výhodou Payary je možnost měnit úroveň logování, aniž bychom byli nuceni znovu nasadit aplikaci.

6.15 Testování

Každá aplikace by měla mít na několika úrovních testy. Při vývoji slouží především k tomu, abychom mohli rychle ověřit, zda je celá aplikace funkční a zda jsme při implementaci nové funkcionality nerozbili omylem jinou část. Máme k dispozici několik typů testů, které se dělí podle rozsahu testu. Častou praxí bývá zapojení testů do Continous Delivery (CD). Při každém nasazení tak automaticky kontrolujeme, jestli jednotlivé části aplikace fungují. Samozřejmě čím větší pokrytí, tím lepší.

6.15.1 Jednotkové testování

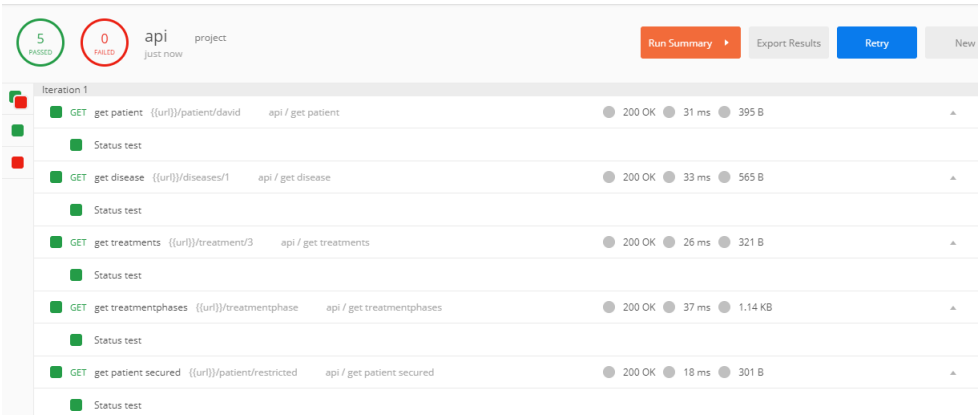
Jednotkové testování se provádí na úrovni kódu. Snaží se otestovat co nejmenší kus kódu, většinou testujeme pouze jednotlivé metody. Bohužel v enterprise aplikaci, v niž používáme vrstvenou architekturu, je velmi obtížné testovat jednotlivé metody izolovaně. Kód je propojen na několika úrovních a často potřebuje k fungování kontakt s databází. Z těchto důvodů je lepší se soustředit na jiné typy testů. V Javě používám pro jednotkové testování Junit framework, jelikož je to nejrozšířenější testovací framework pro Javu. Jednotkové testy se pouštějí většinou přímo v IDE[7].

Pro komplexnější testování Java EE aplikace se osvědčuje praktika, kdy vytvoříme separátní aplikaci, která pomocí Junit testů otestuje hlavní Java EE aplikaci.

6.15.2 Testování skrze postman

Pro testování RESTového rozhraní jsem použil aplikaci Postman, ve které je možné vytvořit sady testů. Toto řešení není vhodně z dlouhodobého hlediska, jelikož se nedá efektivně použít při CD. Ovšem při vývoji se jedná o rychlý a efektivní způsob, jak testovat endpointy automaticky. Testy v Postmanovi nám provolávají jednotlivé endpointy a je možné tak dosáhnout dobrého pokrytí. Aplikace musí být při tomto testování zapnuta. Obrázek 6.4 ukazuje výsledek jedné sady testů.

[7]IDE je software, který vytváří vývojové prostředí.



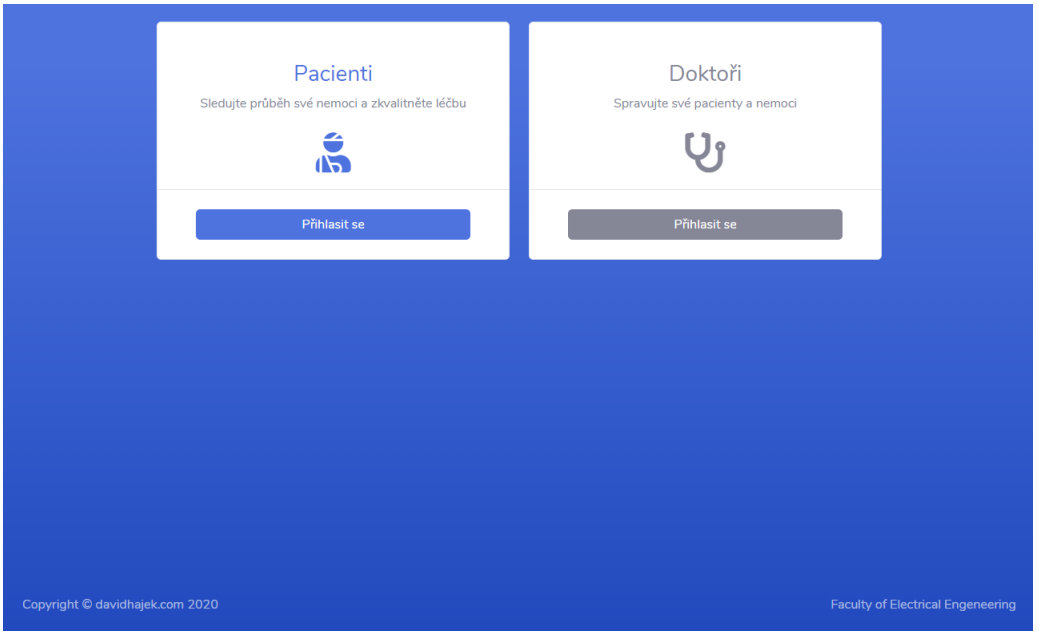
Obrázek 6.4: Výsledky testů v aplikaci Postman

6.16 Ukázka aplikace

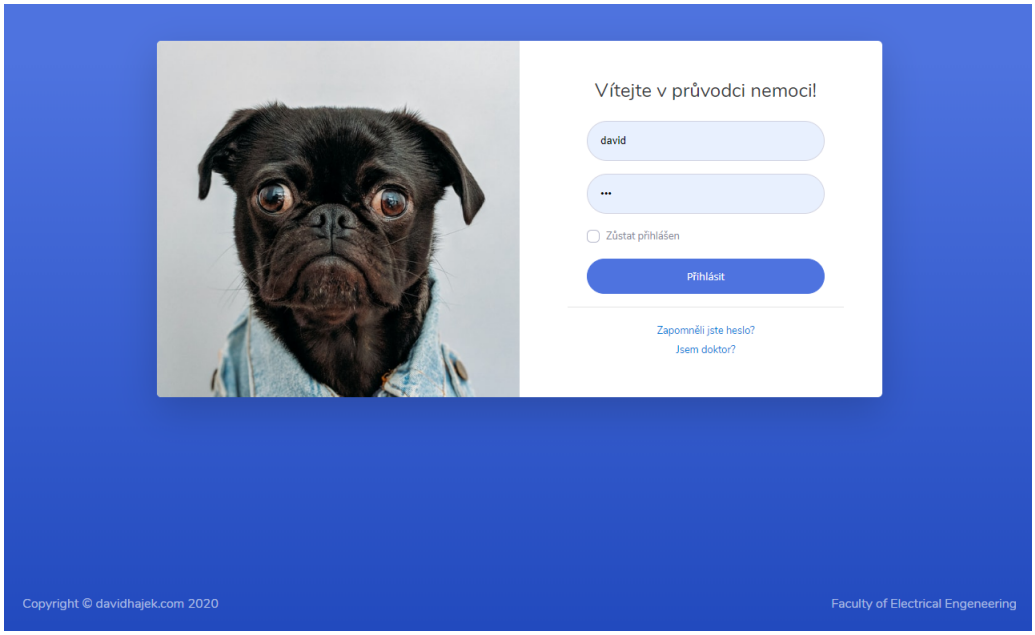
V této sekci přikládám několik ukázek vytvořené aplikace. Při vývoji aplikace jsem dbal na dobrý uživatelský zážitek (UX), jelikož ho vnímám jako klíčový element při podpoře pacienta v užívání aplikace a zlepšení jeho léčby.

Pacientovi bylo na vstupním vyšetření sděleno, že podstoupí bariatrickou operaci a že si musí předtím obstarat různá vyšetření. Po příchodu z nemocnice domů si pacient otevře naši aplikaci na stránce se vstupní bránou viz. 6.5 a přesměruje se na stránku s přihlašovacím formulářem pro pacienty viz. 6.6, kde se přihlásí. Otevře si průvodce nemoci viz. 6.8, v němž se dozví, jaké nejbližší události ho čekají, v jaké je fázi nemoci a co má dělat. Například zde zjistí, jaké vyšetření si musí ještě obstarat, je mu tak připomenuto, aby na žádné nezapomněl. Jelikož pacient nemusel při vstupním vyšetření pochopit veškeré podrobnosti zákroku, na stránce edukace viz. 6.9 si vybírá kurz a čte si jeho kapitoly viz. 6.10. Dosahuje tak vyšší informovanosti o svém stavu a stává se klidnějším. Jak dokazuje studie v kapitole 4.3.2, se zvýšenou informovaností o léčbě dochází ke snížení rizika komplikací, zlepšuje se tak kvalita léčby a v neposlední řadě se šetří i čas lékařů. V průběhu nemoci sleduje na stránce s přehledem viz. 6.7 základní údaje. Pacient také zaznamenává základní udaje na stránce monitoring viz. 6.11. Vidí zde různé vývojové grafy, které ho motivují.

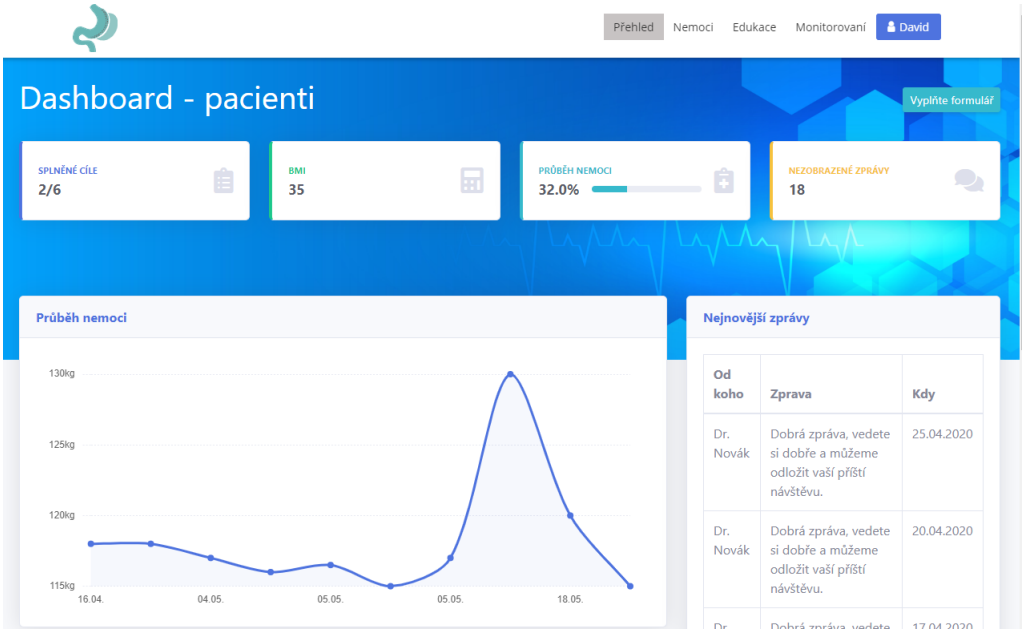
Lékaři, kteří chtějí kontrolovat své pacienty, si otevřou lékařský portál viz. 6.12. Mohou zde upravovat detaily nemocí, vzdělávací kurzy nebo pacienty.



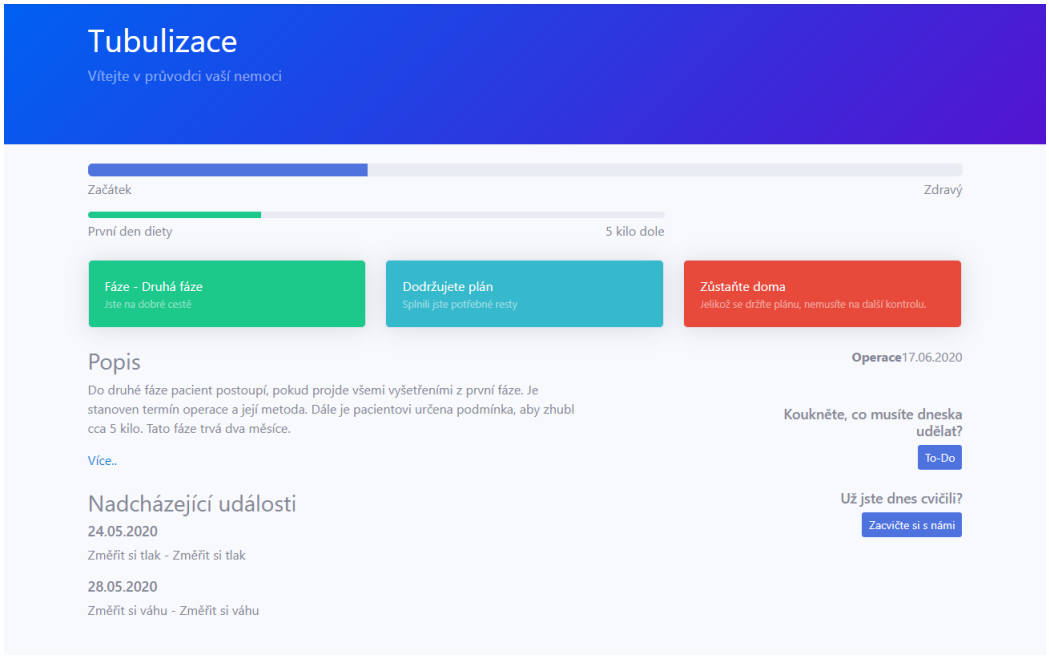
Obrázek 6.5: Vstup do aplikace



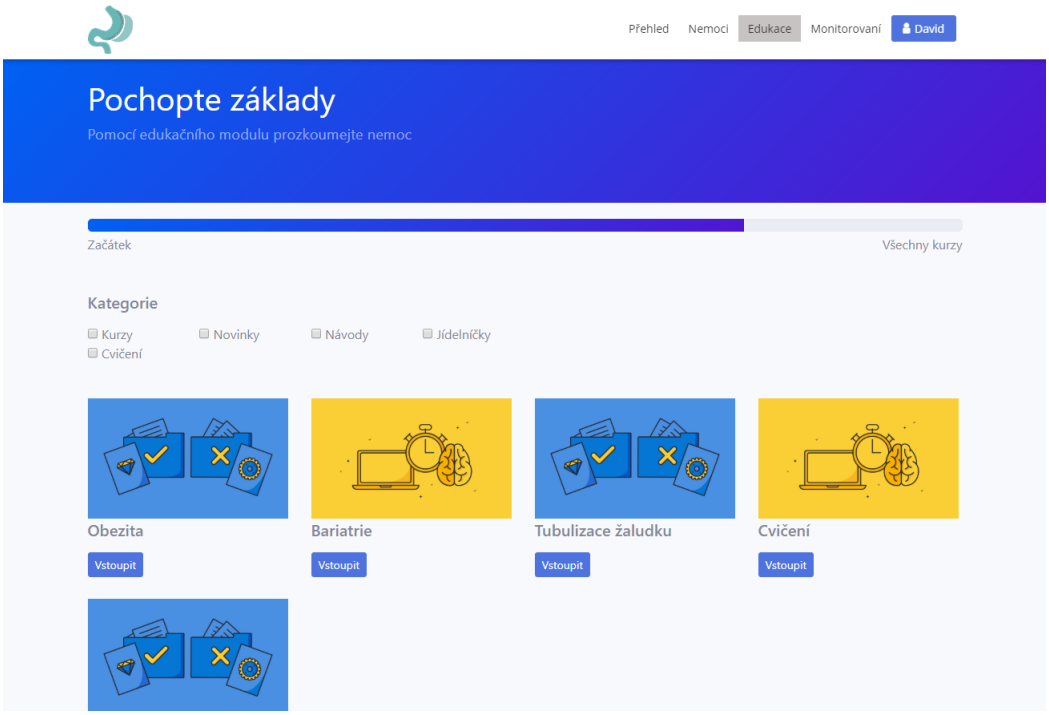
Obrázek 6.6: Formulář na přihlášení



Obrázek 6.7: Přehled

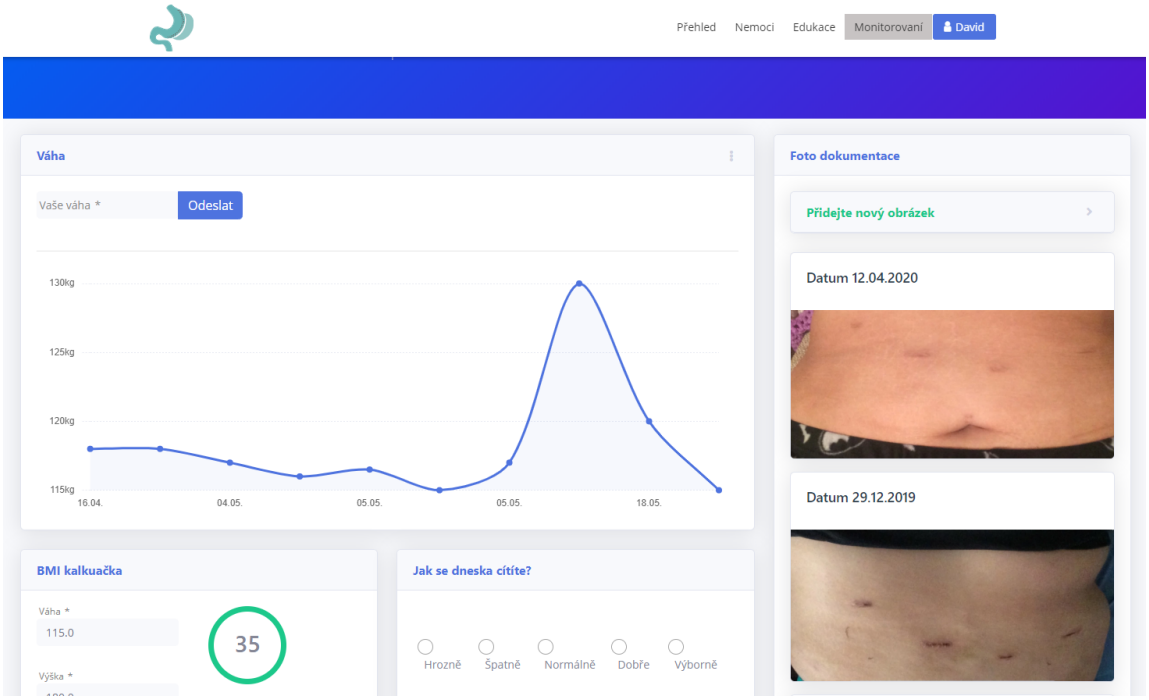


Obrázek 6.8: Detail nemoci v průvodci

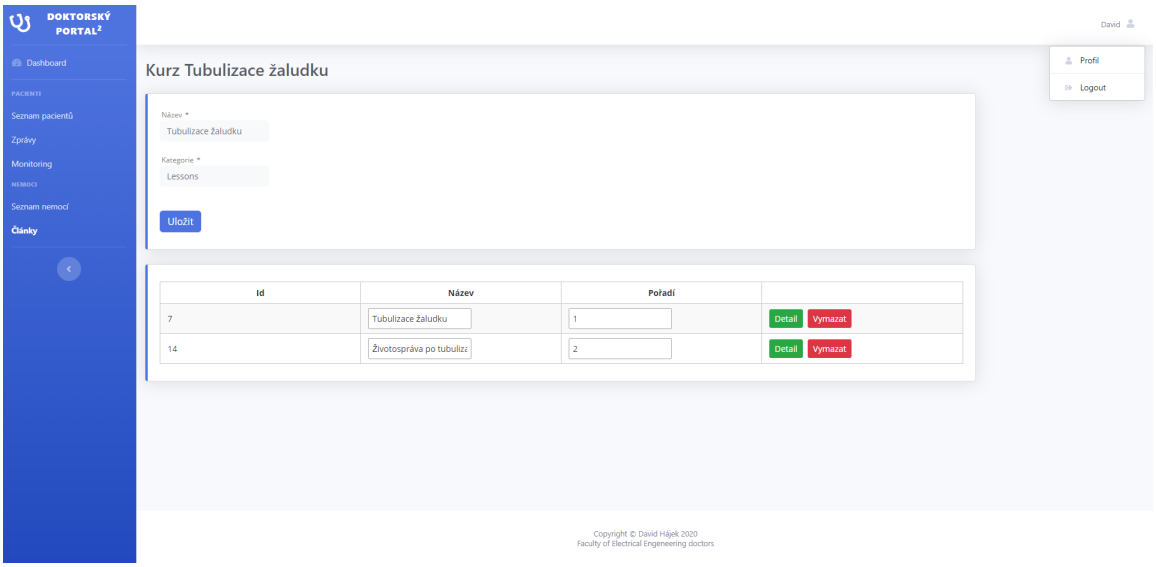


Obrázek 6.9: Edukační modul

Obrázek 6.10: Detail kurzu



Obrázek 6.11: Monitoring



Obrázek 6.12: Doktoři

Závěr

Cílem bakalářské práce bylo vytvořit systém, který zkvalitní péči o pacienty. Podařilo se mi na základě odborných konzultací vymyslet a vytvořit návrh aplikace, která využívá tři pilíře rozebrané v kapitole 3.4. Následně se mi podařilo implementovat jádro celé aplikace, jež umožňuje prokázat funkčnost konceptu. Do konce se bohužel nepovedly dotáhnout některé funkcionality. Ty jsou připraveny vizuálně a na úrovni datové vrstvy. Budu rád, pokud se vývoji aplikace a samotné problematice budu i nadále věnovat.

Na závěr bych rád dodal, že programování je z mého pohledu prvotně o řešení problémů a až druhotně o technologiích. Těší mě, že jsem přišel s konceptem, který se o řešení jednoho takového problému pokusil. Cením si hlavně nabytých zkušeností nasbíraných při hledání řešení a kombinování světa programování a medicíny. Pozitivně vnímám příležitost vyzkoušet si dle mého názoru zajímavé technologie, jež jsou pro tento projekt vhodné. Mým cílem je nadále se v nich zlepšovat.

Literatura

[1] Goncalves Antonio. *Beginning Java EE 7*. Berkeley, CA. Apress. 2013

[2] Daschner Sebastian. *Architecting Modern Java EE Applications*. Birmingham. Packt. 2017.

[3] Späth Peter. *Beginning Jakarta EE Enterprise Edition for Java: From Novice to Professiona*. Leipzig, Sachsen, Germany. Apress. 2019.

[4] KASALICKÝ Mojmír. *Bariatrie: chirurgická léčba obezity a cukrovky*. Praha. Maxdorf. 2018.

[5] FOWLER, Martin. Patterns of enterprise application architecture. Boston: Addison-Wesley, c2003. The Addison-Wesley Signature Series. ISBN 0-321-12742-0.

[6] VESELKA, Josef. Zdravotnictví volá o pomoc: Pět problémů českého zdravotnictví [online]. In: . 2018 [cit. 2020-05-20]. Dostupné z: https://archiv.ihned.cz/c1-66333100-pet-problemu-ceskeho-zdravotnictvi

[7] James, William & Jackson Leach, Rachel & Kalamara, Eleni & Shayeghi, Maryam. (2001). The Worldwide Obesity Epidemic. Obesity research. [online] 9 Suppl 4. 228S-233S. 10.1038/oby.2001.123.

[8] *Bariatrické chirurgické výkony*. WikiScripta, MediaWiki, 2019. [Online; 9.1. 2017]

[9] Jak to funguje? [online]., 1 [cit. 2020-05-08]. Dostupné z: https://www.onlineambulance.cz/o-online-ambulanci

[10] *Online lékařská poradna* [online]., 1 [cit. 2020-05-08]. Dostupné z: https://www.ulekare.cz/jak-funguje-poradna

[11] *O nás* [online]., 1 [cit. 2020-05-08]. Dostupné z: https://www.ordinace.cz/clanek/o-nas/

[12] *Jak to funguje* [online]., 1 [cit. 2020-05-08]. Dostupné z: https://vitadio.cz/jak-to-funguje

[13] *See what you can do with mySugr App* [online]., 1 [cit. 2020-05-08]. Dostupné z: https://www.mysugr.com/en/diabetes-app

[14] SeamlessMD Overview [online]. , 1 [cit. 2020-05-08]. Dostupné z: https://seamless.md/product-overview/

[15] What is a Use Case? - Definition & Examples [online]. 2019, , 1 [cit. 2020-05-08]. Dostupné z: https://study.com/academy/lesson/what-is-a-use-case-definition-examples.html

[16] KENTON, WILL. Feasibility Study [online]. 11. 2019, , 1 [cit. 2020-05-08]. Dostupné z: https://www.investopedia.com/terms/f/feasibility-study.asp

[17] KŘEMEN, Petr. Enterprise architecture [online]. 2018, , 1 [cit. 2020-05-08]. Dostupné z: https://cw.fel.cvut.cz/b181/_media/courses/b6b33ear/lectures/lecture-01-intro.pdf

[18] Klient-server [online]. 2019 [cit. 2020-05-09]. Dostupné z: https://cs.wikipedia.org/wiki/Klient-server

[19] What's the Difference Between the Front-End and Back-End? [online]. 2015, , 1 [cit. 2020-05-09]. Dostupné z: https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end

[20] FOWLER, Martin. EnterpriseApplication [online]. 2014, , 1 [cit. 2020-05-09]. Dostupné z: https://martinfowler.com/bliki/EnterpriseApplication.html

[21] BEAL, Vangie. Enterprise application [online]. 2016,, 1 [cit. 2020-05-09]. Dostupné z: https://www.webopedia.com/TERM/E/enterprise_application.html

[22] Enterprise Application (EA) [online]. 2017, , 1 [cit. 2020-05-09]. Dostupné z: https://www.techopedia.com/definition/24804/enterprise-application-ea

[23] FEAPO. A Common Perspective on Enterprise Architecture [online]. 2013, , 1-2 [cit. 2020-05-10]. Dostupné z: https://web.archive.org/web/20161220084017/http://feapo.org/wp-content/uploads/2013/11/Common-Perspectives-on-Enterprise-Architecture-v15.pdf

[24] Distributed Multitiered Applications [online]. 2017, , 1 [cit. 2020-05-10]. Dostupné z: https://javaee.github.io/tutorial/overview004.html

[25] Should all apps be n-tier? [online]. 2005, , 1 [cit. 2020-05-10]. Dostupné z: http://www.lhotka.net/weblog/ShouldAllAppsBeNtier.aspx

[26] TIOBE Index for May 2020 [online]. 2020, , 1 [cit. 2020-05-10]. Dostupné z: https://www.tiobe.com/tiobe-index/

[27] The Java Language Environment [online]. 1997, , 1 [cit. 2020-05-10]. Dostupné z: https://www.oracle.com/technetwork/java/intro-141325.html

[28] RAI, Akanksha. How JVM Works – JVM Architecture? [online]. , 1 [cit. 2020-05-10]. Dostupné z: https://www.geeksforgeeks.org/jvm-works-jvm-architecture/

[29] Differences between JDK, JRE and JVM [online]. , 1 [cit. 2020-05-10]. Dostupné z: https://www.geeksforgeeks.org/differences-jdk-jre-jvm/

[30] Spring Framework [online]. 2019, , 1 [cit. 2020-05-10]. Dostupné z: https://cs.wikipedia.org/wiki/Spring_Framework

[31] KUMAR SRIVASTAVA, Pramod. Understanding the Basics of Spring vs. Spring Boot [online]. 2018, , 1 [cit. 2020-05-10]. Dostupné z: https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot

[32] Difference Between Java EE and Spring [online]. , 1 [cit. 2020-05-10]. Dostupné z: https://www.educba.com/java-ee-vs-spring/

[33] Spring vs. Java EE [online]. 2019, , 1 [cit. 2020-05-10]. Dostupné z: https://javatutorial.net/spring-vs-java-ee

[34] WODEHOUSE, Carey. SQL vs. NoSQL Databases: What's the Difference? [online]. 2019, , 1 [cit. 2020-05-10]. Dostupné z: https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/

[35] PostgreSQL vs MySQL: What's the Difference? [online]. , 1 [cit. 2020-05-10]. Dostupné z: https://www.guru99.com/postgresql-vs-mysql-difference.html

[36] What Are Business Goals? - Definition & Examples [online]. , 1 [cit. 2020-05-10]. Dostupné z: https://study.com/academy/lesson/what-are-business-goals-definition-examples-quiz.html

[37] Class UINamingContainer [online]. 2011, , 1 [cit. 2020-05-10]. Dostupné z: https://docs.oracle.com/javaee/6/api/javax/faces/component/UINamingContainer.html

[38] How Often Do You See A Doctor? [online]. In: . 2016 [cit. 2020-05-20]. Dostupné z: https://www.helgilibrary.com/charts/how-often-do-you-see-a-doctor/

[39] Zdravotnictví volá o pomoc [online]. In: . 2020 [cit. 2020-05-20]. Dostupné z: https://zdravotnictvivolaopomoc.cz/

[40] Aarts, M., Sivapalan, N., Nikzad, S. et al. Optimizing Bariatric Surgery Multidisciplinary Follow-up: a Focus on Patient-Centered Care. OBES SURG 27, 730–736 (2017). https://doi.org/10.1007/s11695-016-2354-2

[41] Stephen T. Mahoney, Dahlia Tawfik-Sexton, Paula D. Strassle, Timothy M. Farrell, and Meredith C. Duke.Journal of Laparoendoscopic & Advanced Surgical Techniques.Sep 2018.1100-1104.http://doi.org/10.1089/lap.2018.0093

[42] Securing Web Applications [online]. , 2017 [cit. 2020-05-21]. Dostupné z: https://javaee.github.io/tutorial/security-webtier002.html