**Česká zemědělská univerzita v Praze**

**Provozně ekonomická fakulta**

**Katedra informačního inženýrství**


**Diplomová práce**

# Informační systém pro vedení obchodních procesů stomatologického centra

**Maksym Omelianenko**

# ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

# ZADÁNÍ DIPLOMOVÉ PRÁCE

Maksym Omelianenko

Systémové inženýrství a informatika
Informatika

Název práce

**Informační systém pro vedení obchodních procesů stomatologického centra**

Název anglicky

**Information system for business process management of dental center**

---

**Cíle práce**

Primárním cílem této práce je vyvinout a vytvořit systém administrativy a kontroly práce s klienty stomatologického centra ve formě webové aplikace. Administrační systém bude zahrnovat zaznamenávání pacienta na recepci, editaci a mazání záznamů, historii záznamů pro konkrétní čas, vytvoření pracovního plánu pro stomatologické centrum a jeho zaměstnance, schopnost upravit plán. Kontrolní systém bude zahrnovat kontrolu záznamů, aby se předešlo konfliktům, oznámení přítomnosti nebo nepřítomnosti potřebného personálu pro nadcházející léčbu, oznámení (připomenutí) pacienta o nadcházející léčbě. V případě změny plánu by systém měl informovat správce o potřebě zpracovat (zrušit, přeplánovat nebo přepsat jiného lékaře) pacienty zaznamenané v tomto okamžiku. Systém bude poskytovat různé úrovně přístupu pro ředitele, administrátory a lékaře a také analytiku pro ředitele centra.

**Metodika**

Práce se skládá z teoretické a praktické části. Teoretická část zahrnuje studium a analýzu zdrojů popisujících, co je informační systém, jeho typy, modely pro vývoj informačních systémů, příznačností UX a UI designu a typy architektury webových aplikací.

V praktické části bude na základě studované teorie provedena analýza, návrh a vytvoření webového informačního systému, který provádí úkoly popsané v pracovním účelu.

**Doporučený rozsah práce**

50-60 stran

**Klíčová slova**

webová aplikace, vedení obchodních procesů, informační system, UX a UI design, JavaScript, NodeJS

---

**Doporučené zdroje informací**

ALAN COOPER About Face 3: The Essentials of Interaction Design. Wiley; 3rd edition (May 7, 2007) ISBN-13: 978-0470084113

ANTHONY MOLINARO SQL Cookbook: Query Solutions and Techniques for Database Developers. O'Reilly Media; February 9, 2009 ISBN-13: 978-0596009762

FLANAGAN D. JavaScript: The Definitive Guide. O'Reilly Media; 6th edition (May 13, 2011) ISBN-13: 978-0596805524

THOMAS M. CONNOLLY Database Systems. A Practical Approach to Design, Implementation, and Management. Pearson; 6 edition (January 18, 2014) ISBN-13: 978-0132943260

---

**Předběžný termín obhajoby**

2019/20 LS – PEF

**Vedoucí práce**

Ing. Marek Pícka, Ph.D.

**Garantující pracoviště**

Katedra informačního inženýrství

Elektronicky schváleno dne 9. 3. 2020

**Ing. Martin Pelikán, Ph.D.**

Vedoucí katedry

Elektronicky schváleno dne 9. 3. 2020

**Ing. Martin Pelikán, Ph.D.**

Děkan

V Praze dne 30. 11. 2020

**Čestné prohlášení**

Prohlašuji, že svou diplomovou práci "Informační systém pro vedení obchodních procesů stomatologického centra" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.11.2020 _____

**Poděkování**

Rád bych touto cestou poděkoval panu Ing. Marku Pícková Ph.D. za odborné vedení, rady, připomínky a trpělivost při zpracovaní diplomové práce.

# Informační systém pro vedení obchodních procesů stomatologického centra


## Abstrakt

Tato práce se skládá ze dvou hlavních částí – teoretické a praktické.

V teoretické části je čtenáři naznačeno, co informační systém ve skutečnosti je, jsou popsány jeho vlastnosti a jeho hlavní typy. Následuje popis modelovacího jazyka UML, je uvedeno, k čemu se používá, a demonstrují se také hlavní typy diagramů používaných při modelování informačních systémů. Dále se čtenář seznámí s pojmem webová aplikace, s jejími výhodami, architekturou a technologiemi používanými k její implementaci.

Praktickou část tvoří analýza, návrh systému a implementace webové aplikace s využitím technologií popsaných v teoretické části. Analytická část obsahuje popis aktuální situace webové aplikace a popis její očekávané funkčnosti, dále jsou prezentovány diagramy případů jejího použití, scénáře použití a diagramy aktivit. Ve fázi návrhu je představen datový slovník, diagram tříd a drátěný model uživatelského rozhraní. Implementační část popisuje vytváření grafického návrhu rozhraní webové aplikace i jeho využití pomocí frameworku React a hlavního kódu serverové části zapsaného v Node.js.



**Klíčová slova:** informační systém, webová aplikace, UML, framework, JavaScript, React, Node.js, Express, PostgreSQL.

# Information system for business process management of dental centre

**Abstract**

This work consists of two main parts - theoretical and practical. The theoretical part describes what the information system actually is, describes its properties and the main types of information systems. The following is a description of the UML, what it is used for, and also demonstrates the main types of diagrams that are used during the modelling of information systems. Further, the reader will get acquainted with the term web application, its advantages, architecture, and technologies used to implement the application in the practical part of the thesis.

The practical part consists of analysis, system modelling and implementation of a web application using the technologies described in the theoretical part. The analytical part contains a description of the current situation and a description of the expected functionality of the web application, then a diagram of use, use cases, activity diagrams are presented. The modelling phase introduces a data dictionary, class diagram, and wireframe. The implementation part describes the creation of a web application design and its implementation using the React framework and the main server-side code written in the Node.js.

**Keywords**: information system, web application, UML, framework, JavaScript, React, Node.js, Express, PostgreSQL.

# Obsah

# Seznam obrázků

# Seznam tabulek

# 1 Úvod

V dnešní době je nemožné představit si podnikání bez použití informačních systémů. Vzhledem k zrychlujícímu se životnímu tempu a obrovskému toku informací, které každý den dostáváme, musí být tyto systémy rychlé, spolehlivé a v neposlední řadě snadno použitelné. Pohodlí znamená nejen jednoduchý a přímočarý design, ale také přístupnost z jakéhokoli zařízení, a to kdykoli. Proto je tato práce věnována vývoji informačního systému pro řízení obchodních procesů stomatologického centra ve formátu webové aplikace.

Aplikace se vytváří pro stávající stomatologické centrum v Kyjevě. Dříve se zde používal systém „Clinic2004“, který však jeho vývojář v roce 2014 přestal podporovat. Poté přešli na systém „ZubnayaFeya“, ale kvůli jeho vysokým nákladům a nepohodlnému rozhraní se vedení centra rozhodlo objednat systém určený speciálně pro jejich potřeby. Především byl vyžadován pohodlný a jednoduchý postup pro objednání pacienta u lékaře a zobrazení pracovního rozvrhu zaměstnanců. V této práci je popsána tvorba tohoto systému.

Teoretická část práce seznamuje čtenáře se základními pojmy informačních systémů. Dále popisuje výhody webových aplikací, jejich architekturu a technologie použité k jejich vytvoření. Praktická část je věnována analýze, návrhu a implementaci webové aplikace.

# 2 Cíl práce a metodika

## 2.1 Cíl práce

Primárním cílem této práce je vyvinout a vytvořit systém administrativy a kontroly práce s klienty stomatologického centra, a to ve formě webové aplikace. Administrační systém bude zahrnovat zaznamenávání údajů o pacientovi na recepci, editaci a mazání záznamů, vytvoření pracovního plánu pro stomatologické centrum a jeho zaměstnance, schopnost, aby ředitel mohl upravit plán. Kontrolní systém bude zahrnovat kontrolu objednání, aby se předešlo konfliktům, kontrolu přítomnosti nebo nepřítomnosti personálu potřebného pro nadcházející léčbu. Aplikace také musí připomenout pracovníkům nutnost informování pacienta o nadcházející léčbě. Systém bude poskytovat různé úrovně přístupu, a to pro ředitele, recepční a lékaře.

## 2.2 Metodika

Práce se skládá z teoretické a praktické části. Teoretická část bude zahrnovat studium a analýzu zdrojů popisujících, co informační systém je a jeho typy, dále modely pro vývoj informačních systémů, jejich modelování pomocí UML, architekturu webových aplikací, systém řízení báze dat a bezpečnosti informačních systémů na webu.

V praktické části bude na základě studované teorie provedena analýza a vytvořen návrh webového informačního systému, který provádí úkoly popsané v cíli práce. Z popisu aktuální situace a očekávané funkčnosti aplikace bude v analytické části provedeno stanovení aktérů, systémových požadavků a úrovně přístupu uživatelů. Dále bude vypracován UseCase diagram a popsány scénáře použití, které dal následně zpracovávají do diagramů aktivit. V závěru analýzy bude představen abstraktní diagram tříd. V návrhové části se sestaví datový slovník a uvede diagram tříd, na jehož základě bude představen databázový diagram. Také zde prezentován model uživatelského rozhraní a jeho grafické znázornění. Poslední část popíše implementaci webové aplikace. Kvůli dostupnosti a popularitě JavaScriptu byl pro serverovou část vybrán Node.js. Pro práci byl vybrán framework Express, protože je docela minimalistický, ale kvůli jeho popularitě existuje spousta knihoven pro řešení mnoha typických úkolů. Pro databázi byl zvolen relační databázový systém

PostgreSQL, který se na trhu velmi dobře etabloval jako spolehlivé, svobodné a otevřené řešení. Klientská část je navržena pomocí JavaScriptu na základě jednoho z nejpopulárnějších frameworků – React.

# 3 Teoretická východiska

## 3.1 Informační systém

### 3.1.1 Definice systému

Systém – formuje jediné celé kombinace hmotných a nehmotných objektů, spojené několika společnými znaky, vlastnostmi, účely, podobnými podmínkami existence, života, fungování atd.

Fungování systému – proces postupný v čase, zpracovávání vstupní informace do výstupní.

Subsystém – součást systému, sestávající z jednoduchých vzájemně propojených prvků majících specifický funkční účel (Titorenko, 2003).

### 3.1.2 Definice pojmu Informační systém (IS)

Informační systém – vzájemně propojená sada informačních, technických, softwarových, matematických, organizačních, právních a jiných prostředků s pracovníky, jejichž cílem je shromažďovat, zpracovávat, ukládat a vydávat informace a přijímat rozhodnutí nezbytná k dosažení cíle.

Informační tok – sada dat cirkulujících v systému, jakož i mezi systémem a vnějším světem, nezbytná pro řízení, analýzu a řízení operací. Přenos a příjem informačních toků se provádí pomocí různých druhů nosičů informace (Titorenko, 2003).

### 3.1.3 Vlastnosti IS

Lze rozlišit následující základní vlastnosti, které jsou společné pro všechny informační systémy (Titorenko, 2003):

Složitost – systém závisí na komponentech, které jsou v něm obsaženy, na jejich strukturální interakci, jakož i na složitosti vnitřních a vnějších vztahů.

Dělitelnost – systém se skládá z několika subsystémů nebo z prvků identifikovaných podle určitých kritérií a odpovídajících konkrétním cílům a úkolům.

Strukturovanost – určuje přítomnost navázaných spojení a vztahů mezi prvky uvnitř systému, rozdělení elementů systému podle úrovní a hierarchií.

Celistvost systému – znamená, že všechny prvky systému jsou podřízeny jednomu cíli a fungují jako celek.

Přizpůsobivost systému – znamená přizpůsobivost systému podmínkám konkrétní oblasti předmětu.

Integrovatelnost – znamená možnost interakce systému s nově připojenými komponenty nebo subsystémy.

### 3.1.4 Procesy v IS

Procesy zajišťující provoz informačního systému lze rozdělit na (Azad, 2007):

- shromažďování a vkládání informací z externích a interních zdrojů;
- zpracování příchozích informací;
- ukládání informací pro jejich následné použití;
- výstup informací v uživatelsky přívětivé formě;
- zpětná vazba, tj. prezentace informací zpracovaných v tomto systému pro úpravu příchozích informací.

## 3.2 Klasifikace IS

Informační systémy lze rozdělit podle jejich použití do dvou skupin:

- průzkumové – systémy pro vyhledávání;
- řídící – systémy pro organizaci, plánování a rozhodování.

Řídící informační systémy lze rozdělit do několika hlavních kategorií, z nichž každá obsadí (podle potřeby) určité místo v životním (výrobním) cyklu podniku a provede nezbytná opatření s jeho informační podporou. Existuje větší množství kategorizací, ale zde bude použito rozdělení na ERP systém, CRM systém a EDMS systém (Buenova, 2017).

### 3.2.1 ERP systém

ERP – je metodika pro efektivní plánování a řízení všech podnikových zdrojů, které jsou nezbytné pro realizaci prodeje, výroby, nákupu a účetnictví při provádění zákaznických objednávek v oblasti výroby, distribuce a služeb.

Plánování podnikových zdrojů (ERP) je proces používaný společnostmi ke správě a integraci důležitých částí jejich podnikání. Množství softwarových aplikací ERP je pro společnost důležité, protože jí pomáhá implementovat plánování zdrojů integrací všech procesů potřebných pro její provoz do jediného systému. Softwarový systém ERP může také integrovat plánování, nákup zásob, prodej, marketing, finance, lidské zdroje a další.

Téměř každý ERP systém obsahuje následující sadu subsystémů (Labarre, 2020):

- výroba;
- nabídka a prodej;
- úložný prostor;
- údržba zařízení a vyráběných výrobků;
- finance;
- logistika.

Úlohy, které řeší ERP-systém (Labarre, 2020):

- standardizace formulářů hlášení a informačních systémů;
- zlepšování spolupráce mezi odděleními;
- řízení a synchronizace procesů;
- integrace s dodavateli.

### 3.2.2 CRM systém

CRM-systém (Řízení vztahů se zákazníky nebo Customer Relationship Management) je aplikační software pro organizaci, určený k automatizaci strategií interakce se zákazníky (klienty), zejména ke zvýšení prodeje, k optimalizaci marketingu a zlepšení služeb zákazníkům tím, že ukládá informace o zákaznících a historii vztahů s nimi, zavádí a zlepšuje obchodní procesy a následnou analýzu výsledků.

Ve skutečnosti lze za CRM systém považovat jakoukoli možnost kontroly a účetnictví, která může pomoci zlepšit interakci se zákazníkem. I když uchováváte historii hovorů a kontaktů na papíře nebo v aplikaci Excel, lze to považovat za CRM systém. Proto lze definici pojmu CRM omezit na obecnější a stručnější podobu.

CRM systém je jakýkoli software, který pomůže úspěšně sledovat, usměrňovat a plánovat práci s klienty (Kinzyabulatov, 2018).

Úkoly, které může podnik vyřešit implementací systémů CRM (Buenova, 2017):

- zachování a zvýšení loajality zákazníků poskytováním zákaznického servisu té nejvyšší kvality;
- zvýšení prodeje prostřednictvím lepšího porozumění potřebám klientů;
- zvýšení efektivity marketingových aktivit a díky tomu přesnější plánování a následné hodnocení;
- snížení rizika ztráty zákazníka společně s propuštěním zaměstnance;

- eliminace situace, kdy začíná několik zaměstnanců současně pracovat pro jednoho klienta;
- uchovávání všech informací o klientovi na jednom místě;
- možnost komparativní analýzy práce manažerů;
- hledání slabých míst v práci s klienty.

## 3.3 Unified Modeling Language (UML)

Unified Modeling Language (UML) je rodina grafických notací založených na jednom metamodelu. Pomáhá při popisu a návrhu softwarových systémů, zejména systémů postavených pomocí objektově orientovaných technologií (OO). Tato definice je poněkud zjednodušená. Ve skutečnosti mohou různí lidé v UML spatřovat různé věci. Je to důsledek jak vlastní historie vývoje jazyka, tak různých pohledů odborníků na to, co zefektivňuje proces vývoje softwaru.

UML je objektově orientovaný modelovací jazyk s následujícími základními charakteristikami:

- Jde o vizuální modelovací jazyk, který poskytuje vývoj reprezentativních modelů pro organizaci interakce zákazníka a vývojáře IS.

- Obsahuje mechanismy pro rozšiřování a specializaci základních jazykových konceptů.

UML obsahuje interní sadu nástrojů, které jsou akceptovány v mnoha metodách a nástrojích pro modelování. Tyto koncepce jsou nezbytné ve většině aplikací.

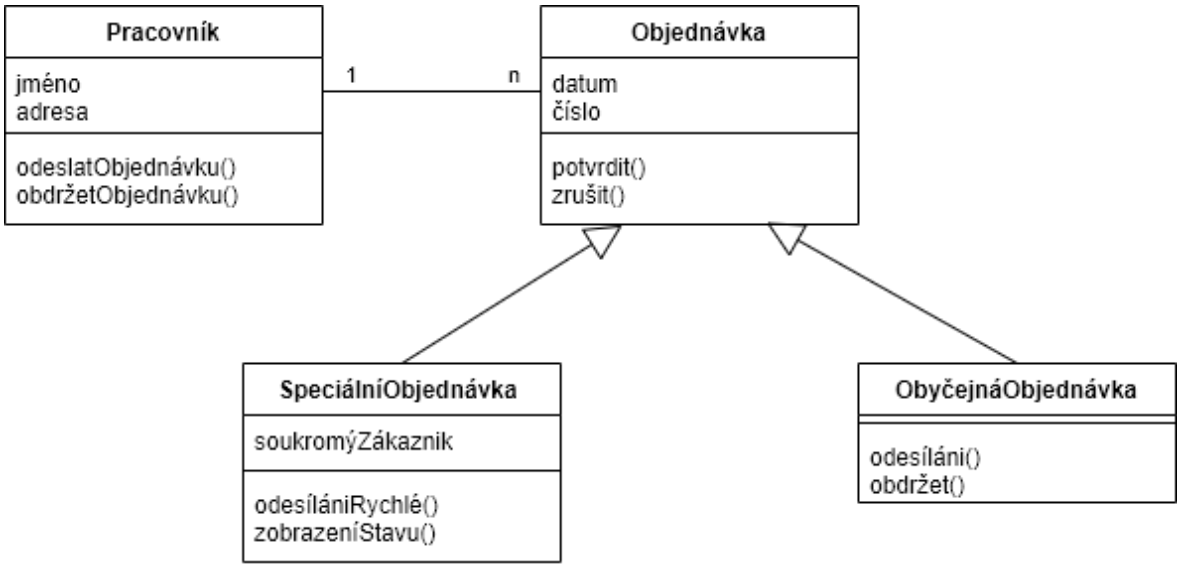
Spojení UML tvoří první písmena Unified Modeling Language. Základem spojení je slovo Unified (sjednocený), což znamená, že našim obrázkům rozumíme nejen my, ale také ostatní, kteří UML znají. Jedná se tedy o mezinárodní jazyk pro kreslení obvodů (Miles, 2006).

### 3.3.1 Diagramy tříd

**Diagram tříd** – je strukturální diagram modelovacího jazyka UML, který demonstruje obecnou strukturu hierarchie tříd systému, jejich interakce, atributy, metody, rozhraní a vztahy mezi nimi. Je široce používán nejen pro dokumentaci a vizualizaci, ale také pro návrh pomocí dopředného nebo zpětného inženýrství.

Účelem vytvoření diagramu tříd je grafické znázornění statické struktury deklarativních prvků systému (třídy, typy atd.). Obsahuje také některé prvky chování (například operace), ale jejich dynamika by se měla projevit v diagramech jiných typů (sekvenční diagramy, stavové diagramy). Pro usnadnění může být diagram tříd ještě doplněn o zobrazení balíčků, včetně vnořených (Arlow, 2005).

**Obrázek 1 - Diagram tříd**




Zdroj: vlastní zpracování

**Třídy** jsou základními prvky jakéhokoli objektově orientovaného systému. Jde o popisy sbírek homogenních objektů s jejich skutečnými vlastnostmi – atributy, operace, vztahy a sémantika.

V rámci modelu je každé třídě přiřazeno jedinečné jméno, které ji odlišuje od ostatních tříd. Pokud je použit složený název (název balíčku, do kterého je třída zařazena, je přidán na začátek názvu), musí být název třídy v balíčku jedinečný.

**Atribut** (vlastnost) je pojmenovaná vlastnost třídy, která popisuje rozsah hodnot, které může exemplář atributu nabývat. Třída může mít libovolný počet atributů nebo žádný. V druhém případě je blok atributů ponechán prázdný.

Atribut představuje určitou vlastnost modelované entity, kterou mají všechny objekty této třídy. Názvem atributu, stejně jako názvem třídy, může být text. V praxi se k pojmenování atributu používá jedno nebo několik krátkých podstatných jmen, která vyjadřují určitou vlastnost třídy, do které atribut patří.

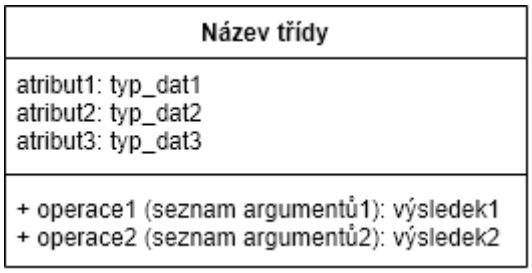
Specifikaci atributu můžeme upřesnit zadáním jeho typu, multiplicity (pokud je atributem pole některých hodnot) a počáteční výchozí hodnoty.

**Operace** (metoda) je realizace třídní metody. Třída může mít libovolný počet operací nebo žádnou. Operace jsou graficky prezentovány ve spodním bloku popisu třídy.

Povoleny jsou pouze názvy operací. Název operace, stejně jako název třídy, musí být text. V praxi se k pojmenování operace používají krátké slovesné konstrukce, které popisují určité chování třídy, do které operace patří. Každé slovo v názvu operace je obvykle psáno velkými písmeny, s výjimkou prvního, například move nebo isEmpty.

Můžete určit operaci nastavením jejího popisu, který zahrnuje název, typ a výchozí hodnotu všech parametrů (Arlow, 2005).

**Obrázek 2 - Zobrazení třídy v UML**




Zdroj: vlastní zpracování

## Vztahy mezi třídami

V UML existují čtyři typy vztahů:

- vztah (závislost);
- asociace;
- zobecnění;
- implementace.

Tyto vztahy jsou základními prvky pro popis vztahů v UML, které se používají k vývoji dobře konzistentních modelů.

**Vztah** – sémanticky představuje vztah mezi dvěma prvky modelu, ve kterém může změna jednoho prvku (nezávislého) vést ke změně sémantiky jiného prvku (závislého). Je to graficky znázorněno tečkovanou čarou, někdy se šipkou směřující k entitě, na které závisí další (Obrázek 2).

**Asociace** – je strukturální vztah mezi prvky modelu, jenž popisuje sadu vztahů, které existují mezi objekty. Asociace ukazuje, že objekty jedné entity (třídy) jsou propojeny s objekty jiné entity takovým způsobem, že je možno z objektů jedné třídy přecházet do druhé (Miles, 2006).

*Vícenásobná asociace* je rozsah celých čísel označujících možný počet souvisejících objektů. Je zapsán jako výraz s minimální a maximální hodnotou; k jejich oddělení se používají dva body. Nastavením multiplicity vzdáleného konce přidružení určíme, kolik objektů může existovat na vzdáleném konci přidružení pro každý objekt třídy na blízkém konci. Počet objektů musí být ve stanoveném rozsahu. Množinu lze definovat jako jednu 1, nulu nebo jednu 0 .. 1, libovolnou hodnotu 0 .. * nebo *, jednu nebo více 1 .. *. Můžeme také určit rozsah celočíselných hodnot, například 2 .. 5, nebo zadat přesné číslo, například 3 (Obrázek 2).

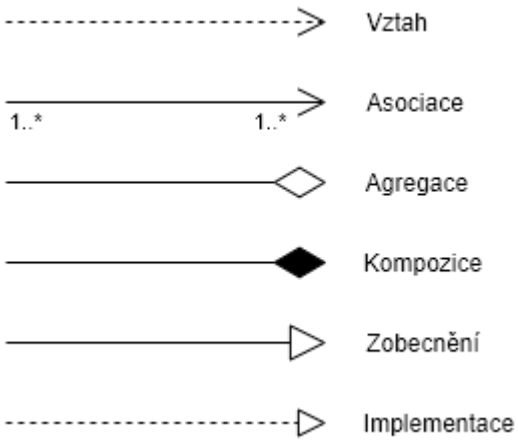
*Agregace* – je zvláštní druh asociace, která představuje strukturální vztah celku s jeho částmi. Jako typ přidružení lze agregaci pojmenovat. Jeden agregační vztah nemůže zahrnovat více než dvě třídy (kontejner a obsah). K agregaci dochází, když je jedna třída kolekcí nebo kontejnerem jiných. Graficky je agregace reprezentována prázdným diamantem na straně bloku třídy „celá“ a přímkou vedoucí z diamantu do třídy „část“ (Obrázek 2).

*Kompozice* – je přísnější verzí agregace, neboli agregace podle hodnoty. Kompozice je formou agregace s jasně definovanými vlastnickými vztahy částí a celku. Má pevnou závislost na životnosti instancí třídy kontejneru a obsažených instancí třídy. Pokud je kontejner zničen, zničí se také veškerý jeho obsah. Je to graficky znázorněno jako agregace, ale s vyplněným diamantem (Obrázek 2) (Arlow, 2005).

**Zobecnění** – vyjadřuje specializaci nebo dědičnost, ve které je podle specifikací zobecněného prvku (rodiče) vytvořen specializovaný prvek (dítě). Dítě zdědí strukturu a chování rodiče. Zobecnění je graficky znázorněno jako plná čára s prázdnou šipkou směřující k rodiči (Obrázek 2) (Miles, 2006).

**Implementace** – je sémantický vztah mezi třídami, kde jedna z nich (poskytovatel) definuje dohodu, kterou musí druhá (klient) dodržovat. Jedná se o vztahy mezi rozhraními a třídami, které tato rozhraní implementují. Poskytovatel je obvykle reprezentován abstraktní třídou. Graficky je vztah implementace hybridem vztahů generalizace a závislostí: trojúhelník označuje dodavatele a druhý konec tečkované čáry označuje zákazníka (Obrázek 2) (Miles, 2006).

**Obrázek 3 - Vztahy v UML**



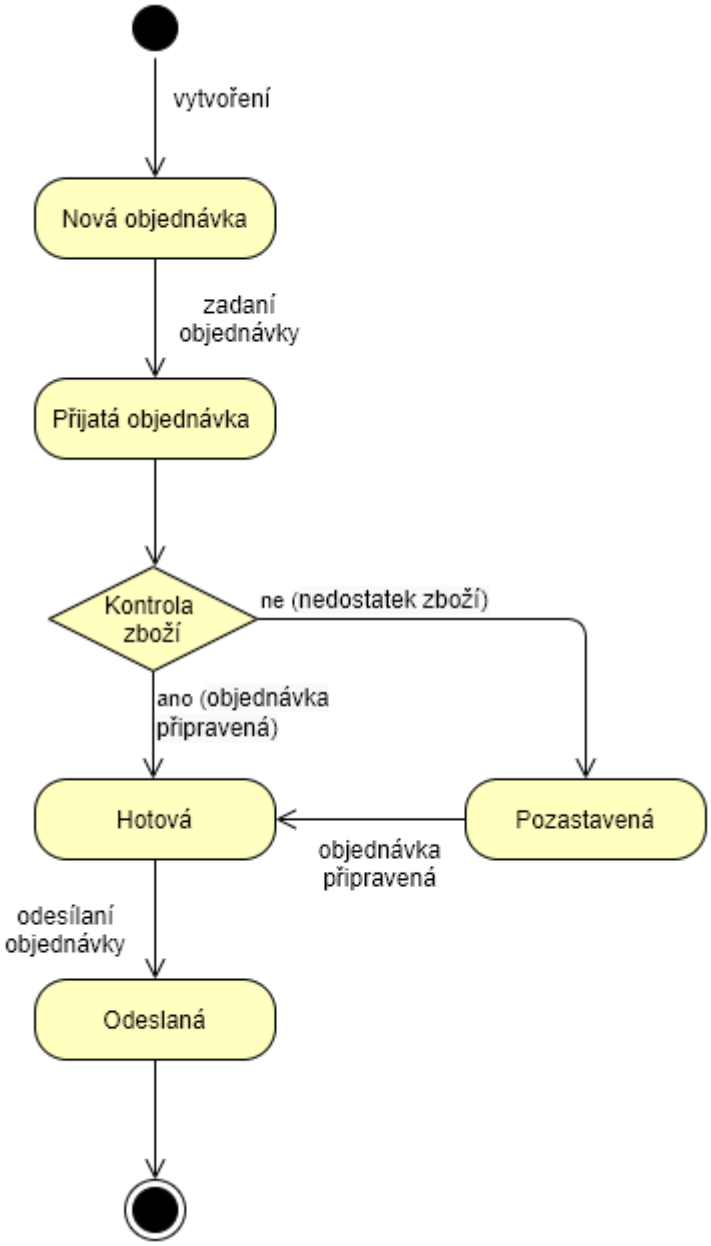

Zdroj: Miles, 2006, vlastní zpracování

### 3.3.2 Stavové diagramy

Stavové diagramy se používají k popisu chování komplexních systémů. Určují všechny možné stavy, ve kterých může být objekt, a proces změny stavu objektu v důsledku určitých událostí. Tyto diagramy se obvykle používají k popisu chování jednoho objektu v několika případech použití.

Obdélníky představují stavy, kterými objekt během svého chování prochází. Stavům odpovídají určité hodnoty atributů objektů. Šipky představují přechody z jednoho stavu do druhého, které jsou způsobeny provedením některých funkcí objektu (Obrázek 4). Existují také dva typy pseudostavů: počáteční stav, ve kterém je nově vytvořený objekt umístěn, a konečný stav, který objekt neopustí, jakmile se tam přesune (Arlow, 2005)

**Obrázek 4 - Stavový diagram**




Zdroj: vlastní zpracování

### 3.3.3 Diagramy použití

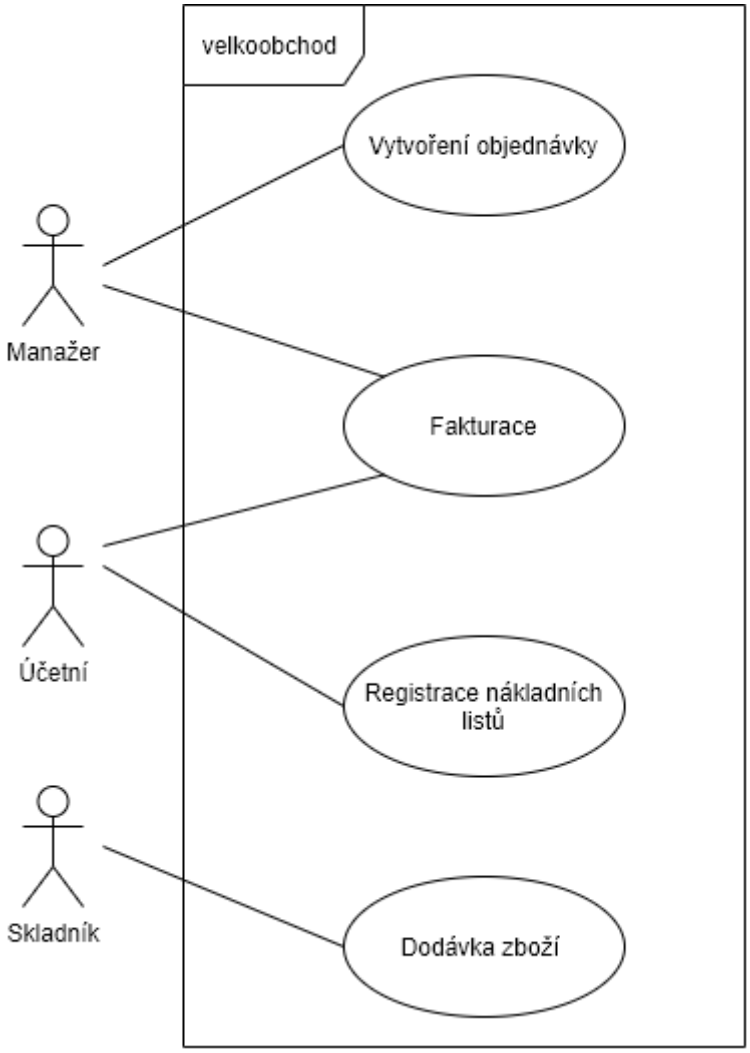
Diagramy použití nebo Use Case diagramy popisují funkčnost IS, která bude viditelná pro uživatele systému. „Každá funkčnost“ je zobrazena jako „případy použití“ (use case) nebo jednoduše precedent. Případ použití je typická interakce uživatele se systémem. Tento typ diagramu je určen pro:

- modelování vzhledu systému z hlediska jeho případů použití;

- stanovení funkčních požadavků na systém. (Odpověď na otázku: „Jaké funkce budou v systému?“);
- definici hranice funkčnosti navrženého systému.

Precedent je na diagramu označen oválem spojeným s uživateli, kteří se obvykle nazývají Aktory (herci, actors) (Obrázek 5). Aktory v tomto případě používají systém. Aktor je sada logicky souvisejících rolí prováděných při interakci s případy použití nebo entitami (systém, subsystém nebo třída). Účastníkem může být osoba nebo jiný systém, subsystém nebo třída, která představuje něco mimo podstatu. Graficky je účastník zobrazen jako „člověk“ (Arlow, 2005).

**Obrázek 5 - Diagram použití**



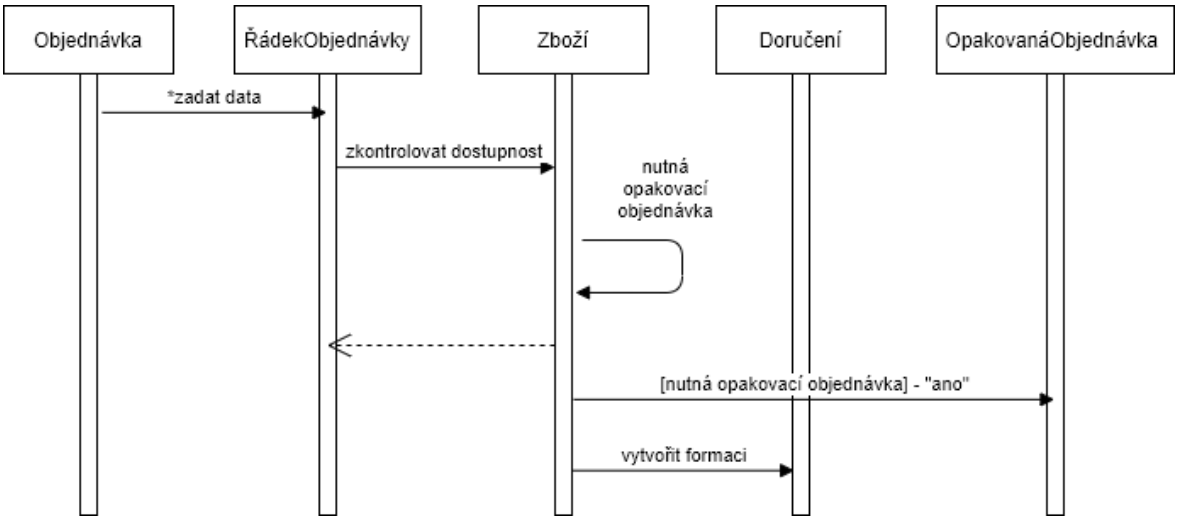

Zdroj: vlastní zpracování

### 3.3.4 Sekvenční diagram

Tento typ diagramu se používá k přesnému určení logiky scénáře použití. Sekvenční diagramy zobrazují typy objektů, které spolu komunikují, když provádějí precedenty, zprávy, které si navzájem posílají, a jakékoli návratové hodnoty spojené s těmito zprávami. Obdélníky na svislých čarách zobrazují „životnost“ objektu. Čáry se šipkami a nápisy názvů metod označují volání metody na objektu (Obrázek 6).

1. jsou zadány řádky objednávky;
2. v každém řádku se kontroluje dostupnost zboží;
3. jsou-li zásoby dostatečné, zahájí se dodávka;
4. pokud je zásoba nedostatečná, je zahájena změna pořadí.

**Obrázek 6 - Sekvenční diagram**




Zdroj: vlastní zpracování

Zprávy se zobrazují v pořadí, jak jsou znázorněny na obrázku, tj. shora dolů. Pokud má objekt poslat zprávu sám sobě (samo-delegování), šipka začíná a končí na jedné „životní linii“.

K diagramům lze přidat kontrolní informace: popis podmínek, za kterých je zpráva odeslána, znamení opakovaného odeslání zprávy (iterační značka), znak návratu zprávy (Mindalev, 2015).

## 3.4 WEB aplikace

### 3.4.1 Co je to webová aplikace

Webová aplikace je počítačový program, který používá k provádění úkolů přes internet webové prohlížeče a webové technologie.

Webové aplikace používají ke zpracování, ukládání a načítání informací kombinaci scénářů na straně serveru a scénářů na straně klienta k předávání informací uživatelům. To umožňuje uživatelům (například zákazníkům nebo zaměstnancům) komunikovat se společností pomocí online formulářů, systémů pro správu obsahu, nákupních košíků online obchodů a dalších. Kromě toho umožňují aplikace zaměstnancům vytvářet dokumenty, sdílet informace, spolupracovat na projektech a pracovat na sdílených dokumentech bez ohledu na umístění nebo zařízení (Shklar, 2009).

### 3.4.2 Výhody webových aplikací

Z principu fungování webových aplikací vyplývají jejich výhody oproti desktopovým protějškům (Gibb, 2015):

*Přístup z jakéhokoli zařízení.* S webovou aplikací můžete pracovat kdekoli na světě z počítače, tabletu nebo smartphonu připojeného k internetu.

*Úspora zdrojů.* Webové aplikace fungují na všech platformách a eliminují potřebu vyvíjet aplikace samostatně pro Android a iOS. Navíc není ve webových aplikacích vyžadováno schválení mobilního obchodu.

*Přizpůsobivost.* Pokud nativní aplikace potřebují určité operační systémy, je pro práci s webovou aplikací vhodný jakýkoli operační systém (Windows, MAC, Linux atd.) a jakýkoli prohlížeč (Internet Explorer, Opera, FireFox, Google Chrome atd.).

*Absence klientského softwaru.* Levnější a snadnější instalace, údržba a upgrade klientského rozhraní. Není třeba uživatelům připomínat, aby aktualizovali své aplikace. Aktualizace na nejnovější verzi nastane při příštím načtení stránky.

*Zabezpečení sítě.* Webový systém má jediný vstupní bod, který lze centrálně chránit a konfigurovat.

*Škálovatelnost.* Se zvyšujícím se zatížením systému není nutné zvyšovat kapacitu klientských míst. Webová aplikace umožňuje zpracovávat více dat, a to obvykle pouze pomocí hardwarových zdrojů, bez přepisování kódu nebo změny architektury.

*Ochrana před ztrátou dat.* Uživatelská data jsou uložena v „cloudu“, za integritu, které jsou poskytovatelé hostingu odpovědní, a jsou chráněna před ztrátou v případě poškození pevného disku počítače.

### 3.4.3 Architektura webových aplikací

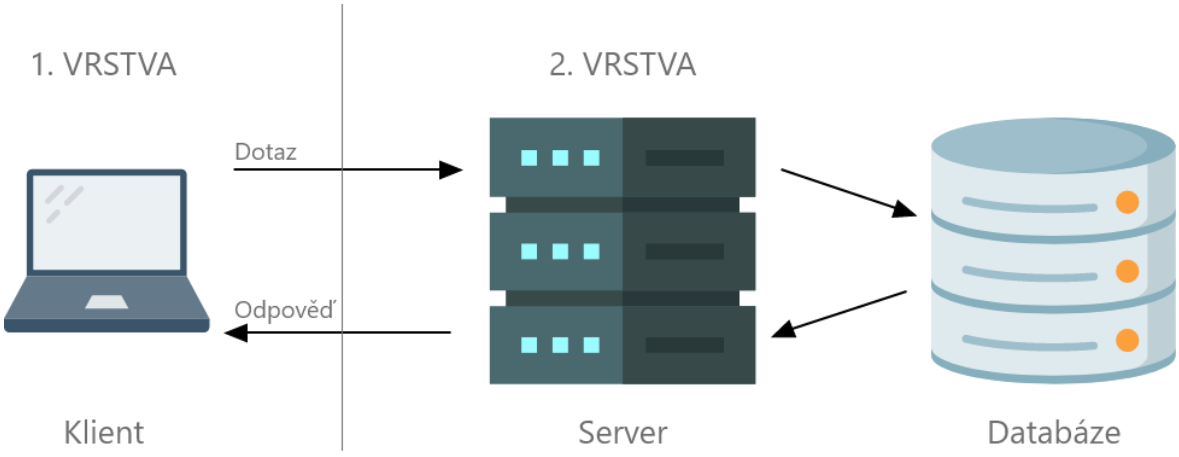
Webová aplikace je aplikace postavená na architektuře klient-server, ve které je klientem prohlížeč a serverem webový server (v širším slova smyslu).

Hlavní část aplikace je zpravidla umístěna na straně webového serveru, který zpracovává přijaté požadavky v souladu s obchodní logikou produktu a generuje odpověď zaslanou uživateli. Prohlížeč převede přijatou odpověď ze serveru na grafické uživatelské rozhraní.

Architektura klient-server definuje obecné principy organizace interakce v síti, kde jsou servery, poskytovatelé uzlů některých specifických funkcí (služeb) a klienti (spotřebitelé těchto funkcí). Praktické implementace této architektury se nazývají technologie klient-server (Kamajdanov, 2014).

**Dvouvrstvá architektura** – distribuce tří základních komponent mezi dva uzly (klient a server). Dvouvrstvá architektura se používá v systémech klient-server, kde server reaguje na požadavky klientů přímo a v plném rozsahu.

**Obrázek 7 Dvouvrstvá architektura**




Zdroj: Kamajdanov, 2014. Vlastní zpracování

Umístění komponent na straně klienta nebo serveru určuje následující základní modely jejich interakce v rámci dvoustupňové architektury:

- Terminálový server – distribuovaná prezentace dat.
- Souborový server – přístup ke vzdálené databázi a souborovým prostředkům.
- Databázový server – vzdálená prezentace dat.
- Aplikační server – vzdálená aplikace.

*Klientem* je prohlížeč, ale existují i výjimky (v případech, kdy jeden webový server (BC1) zadá požadavek jinému (BC2), roli klienta hraje webový server BC1). V klasické situaci (kdy prohlížeč funguje jako klient), aby uživatel viděl grafické rozhraní aplikace v okně prohlížeče, musí prohlížeč zpracovat přijatou odpověď z webového serveru, který bude obsahovat informace implementované pomocí HTML, CSS, JS (nejpoužívanější technologie). Právě tyto technologie „objasňují" prohlížeči, jak přesně je nutné „vykreslit" vše, co obdržel jako odpověď (Kamajdanov, 2014).

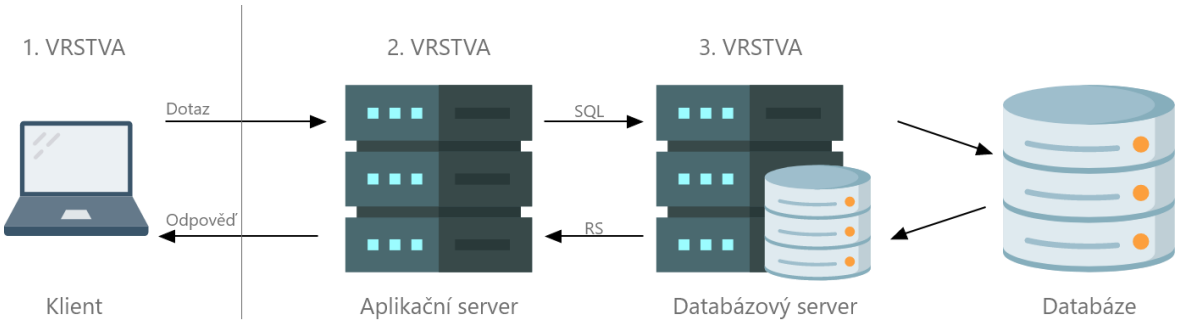
*Webový server* je server, který přijímá požadavky HTTP od klientů a vydává na ně odpovědi HTTP. Označuje jak software, který funguje jako webový server, tak samotný počítač, na kterém software běží. Nejběžnějšími typy softwaru webového serveru jsou Apache, IIS a NGINX. Testovaná aplikace běží na webovém serveru, který lze implementovat pomocí široké škály programovacích jazyků: PHP, Python, Ruby, Java, Perl atd. (Kamajdanov, 2014).

*Databáze* ve skutečnosti není součástí webového serveru, ale většina aplikací bez ní jednoduše nemůže vykonávat všechny funkce, které jí byly přiřazeny, protože v databázi jsou uloženy všechny dynamické informace aplikace (pověření, uživatelská data atd.).

Databáze je informační model, který umožňuje uspořádané ukládání dat o objektu nebo skupině objektů, které mají sadu vlastností, které lze kategorizovat. Databáze fungují pod kontrolou takzvaných systémů správy databází (dále jen DBMS). Nejoblíbenější DBMS jsou MySQL, MS SQL Server, PostgreSQL, Oracle (všechny jsou klient-server).

**Třívrstvá architektura** – síťová aplikace je rozdělena na dvě nebo více částí, z nichž každá může běžet na samostatném počítači. Specializované části aplikace na sebe vzájemně působí výměnou zpráv v předem dohodnutém formátu.

**Obrázek 8 Třívrstvá architektura**




Zdroj: Kamajdanov, 2014. Vlastní zpracování

Třetím odkazem ve třívrstvé architektuře je aplikační server, tj. komponenty jsou distribuovány následovně:

- Prezentace dat je na straně klienta.
- Aplikační komponenta je na vyhrazeném aplikačním serveru (volitelně sloužící jako middleware).
- Správa zdrojů je na databázovém serveru, který představuje požadovaná data.

Třívrstvá architektura může být rozšířena na vícevrstvou (N-tier, Multi-tier) přidělením dalších serverů, z nichž každý bude poskytovat své vlastní služby a využívat služby jiných serverů různých úrovní.

Dvouvrstvá architektura je jednodušší, protože všechny požadavky obsluhuje jeden server, ale právě proto je méně spolehlivá a klade vyšší nároky na výkon serveru.

Třívrstvá architektura je složitější, ale vzhledem k tomu, že jsou funkce distribuovány mezi servery druhé a třetí úrovně, poskytuje tato architektura (Kamajdanov, 2014):

- vysoký stupeň flexibility a škálovatelnosti;
- vysoké zabezpečení (protože zabezpečení lze definovat pro každou službu nebo vrstvu);
- vysoký výkon (protože úkoly jsou distribuovány mezi servery).

### 3.4.4 Komunikace klient-server

Webové prohlížeče komunikují s webovými servery pomocí protokolu Hypertext Transfer Protocol (HTTP). Pokud klikneme na odkaz na stránce, vyplníme formulář nebo spustíme vyhledávání, prohlížeč odešle požadavek HTTP na server.

Tento požadavek zahrnuje (Ludin, 2017):

- Cestu, která identifikuje cílový server a prostředek (například soubor, konkrétní datový bod na serveru, spuštěná služba atd.).
- Metodu, která definuje požadovanou akci (například získání souboru, uložení nebo aktualizace některých dat). Níže jsou uvedeny různé metody/příkazy a jejich přidružené akce:
    - GET – získejte konkrétní zdroj (například soubor HTML obsahující informace o produktu nebo seznam produktů).
    - POST – vytvořte nový zdroj (například nový článek, přidejte nový kontakt do databáze).
    - HEAD – získejte metadata o konkrétním zdroji, aniž byste dostali obsah jako GET. Můžete například použít požadavek HEAD k zjištění, kdy byl zdroj naposledy aktualizován, a až poté použít (dražší) požadavek GET k načtení zdroje, který byl změněn.
    - PUT – aktualizujte existující zdroj (nebo vytvořte nový, pokud neexistuje).
    - DELETE – odstraní zadaný zdroj.
    - TRACE, OPTIONS, CONNECT, PATCH – a další specifické příkazy pro složitější dotazy.

Webové servery čekají na zprávy s požadavky klientů a zpracovávají je. Poté odpoví do prohlížeče zprávou s odpovědí HTTP. Ta obsahuje stavový kód odpovědi HTTP, který označuje, zda byl požadavek úspěšný (například „200 OK“ znamená úspěch, „404 Not Found“, pokud nelze zdroj najít, „403 Forbidden“, pokud uživatel nemá povolení zdroj zobrazit atd.). Tělo úspěšné odpovědi na požadavek GET bude obsahovat požadovaný zdroj.

Po vrácení HTML stránky je prohlížeč zpracovává. Poté může prohlížeč prozkoumat odkazy na jiné zdroje (například stránka HTML obvykle používá soubory JavaScriptu a CSS) a odeslat samostatný požadavek HTTP ke stažení těchto souborů (Ludin, 2017).

### 3.4.5 Klientská část

Klientská část nebo jednoduše klient je „tváří“ aplikace, tedy tím, co uživatel vidí. Za prvé je zodpovědný za rozhraní a přímou interakci s uživatelem. Grafické rozhraní se zobrazí v prohlížeči. Uživatel interaguje s webovou aplikací přesně prostřednictvím

prohlížeče, kliknutím na odkazy a tlačítka. K provádění složitých operací generuje klient požadavky na server a zpracovává z něj odpovědi (Purewal, 2014).

#### 3.4.5.1 HTML, CSS, JavaScript

Hlavním jazykem používaným k popisu grafického rozhraní webové aplikace je HTML (Hyper Text Markup Language). HTML je nedílnou součástí a základem téměř jakékoli webové stránky. Jazyk HTML slouží především pro popis struktury webové stránky. Další technologií použitou k vytvoření klientské části je: *Cascading Style Sheets* – kaskádová tabule stylů. Jednoduše řečeno, jazyk CSS je navržen tak, aby dokumentům HTML poskytoval potřebný vzhled. Cílem vytvoření CSS bylo tedy oddělit popis logické struktury webové stránky od jejího vzhledu. Jak již víme, HTML se používá k popisu struktury, CSS se používá pouze k popisu toho, jak bude tato logická struktura vypadat (Duckett, 2011).

K „animaci“ grafického rozhraní se používají další technologie: skripty JavaScriptu a komponenty zabudované do webové stránky vytvořené v prostředí Flash, Java nebo Silverlight. Všechny tyto prvky webové stránky mohou vzájemně interagovat: program napsaný v JavaScriptu a spuštěný na webové stránce může manipulovat s komponentami vloženými do stránky, čímž implementuje bohaté uživatelské rozhraní.

Pro urychlení vývoje a strukturování projektu se často používají JS frameworky. Klientské frameworky nemají nic společného s logikou aplikace. Tento typ frameworku funguje v prohlížeči. S jejich pomocí můžeme vylepšit a implementovat nová uživatelská rozhraní. Klientské frameworky umožňují vytvářet různé animace a jednostránkové aplikace. Jednotlivé klientské frameworky se liší funkčností a použitím (Robbins, 2012).

Frameworky nejsou knihovny.

*Knihovna* je jednodušší součástí softwarové architektury. Softwarovou knihovnu lze použít jednoduše jako sadu subsystémů s úzkou funkčností, aniž by to ovlivnilo architekturu hlavního softwarového produktu a aniž by na něj byla uložena jakákoli omezení.

*Framework* neposkytuje vývojáři pouze potřebnou funkčnost, ale také diktuje pravidla pro budování aplikační architektury, nastavení výchozího chování v počáteční fázi vývoje, vytváření rámce, který bude třeba rozšířit a změnit podle stanovených požadavků. Framework může také zahrnovat podpůrné programy, knihovny kódů, skriptovací jazyk

a další software, který usnadňuje vývoj a integraci různých komponent velkého softwarového projektu (Robbins, 2012).

#### 3.4.5.2 React

React je nástroj pro vytváření uživatelských rozhraní. Jeho hlavním úkolem je poskytovat zobrazení toho, co lze vidět na webových stránkách. Díky funkci React je mnohem jednodušší vytvářet rozhraní rozdělením každé stránky na malé bloky zvané komponenty.

Komponenta React je jednoduše řečeno část kódu, která představuje část webové stránky. Každá komponenta je funkcí JavaScriptu, která vrací část kódu, která představuje část stránky.

Ve světě JavaScriptu je React rozhodujícím lídrem. Aby bylo možné pružně používat React při vývoji webu, existuje mnoho dalších nástrojů. Například zde není zdaleka vyčerpán seznam takových nástrojů poskytovaných knihovnami, které lze použít ve spojení s Reactem. Jedná se o Redux, MobX, Fluxy, Fluxible, RefluxJS. Ve vývoji React můžeme také použít jQuery AJAX, Superagent, Axios a Fetch API (Martin, 2019).

### 3.4.6 Strana serveru

Serverová část nebo server je „mozkem“ aplikace, kde se provádějí všechny složité výpočty, ukládají se hromadná data a koordinuje se práce jako celek. Miliony klientských systémů mohou komunikovat s jedním serverem současně.

Kód na straně serveru lze zapsat v kterémkoli z programovacích jazyků. Má plný přístup k operačnímu systému serveru a vývojář si může vybrat, který programovací jazyk (a jakou verzi) by chtěl použít. Chceme-li implementovat aplikační server, musíme vybrat jeden z jazyků (Node.js, Ruby, PHP, Scala, Java, C #, .NET atd.) a MVC framework pro tento jazyk (Express pro Node.js, Ruby on Rails, Play pro Scala, Laravel pro PHP atd.) (Purewal, 2014).

Vývojáři obvykle píšou svůj kód pomocí webových frameworků. Klientská a serverová část jsou velmi odlišné, a proto se frameworky také liší. Klientské frameworky zjednodušují rozložení a prezentaci dat, zatímco frameworky serverové části poskytují mnoho „základních“ funkcí webového serveru, které bychom jinak museli provádět sami, jako je podpora relací, podpora a ověřování uživatelů, snadný přístup k databázi, šablony knihoven atd.

Serverová část umožňuje řešit následující úkoly (Purewal, 2014):

- efektivní ukládání a doručování informací;
- přizpůsobitelné uživatelské prostředí;
- řízený přístup k obsahu;
- ukládání informací o relaci/stavu;
- oznámení a komunikaci;
- analýzu dat.

#### 3.4.6.1 Node.js

Node nebo více formálně Node.js je runtime platforma s otevřeným zdrojovým kódem, která umožňuje vývojářům vytvářet pomocí JavaScriptu všechny druhy serverových nástrojů a aplikací. Běhové prostředí je určeno k použití mimo kontext prohlížeče (tj. běží přímo na počítači nebo na serverovém OS). Prostředí tedy vylučuje API-rozhraní jazyka JavaScript pro prohlížeč a přidává podporu pro tradiční API-rozhraní OS, a to včetně knihoven HTTP a souborových systémů (Lim, 2019).

Při pohledu na vývoj webového serveru má Node.js několik výhod (Herron, 2020):

- Skvělý výkon. Node.js byl navržen pro optimalizace šířky pásma a škálovatelnosti ve webových aplikacích a velmi dobře řeší mnoho běžných problémů s vývojem webu (například webové aplikace v reálném čase).
- Kód je zapsán jako „obyčejný starý JavaScript“, což znamená, že je nutno strávit méně času psaním kódu pro prohlížeč a webový server spojený s „přepínacími technologiemi“ mezi jazyky.
- JavaScript je relativně nový programovací jazyk a má výhody ve zlepšování designu jazyka ve srovnání s jinými tradičními jazyky pro webové servery (například Python, PHP atd.). Mnoho dalších nových a populárních jazyků je kompilováno/převedeno do JavaScriptu, takže můžeme také použít CoffeeScript, ClosureScript, Scala, LiveScript atd.
- Node.js Package Manager (NPM) poskytuje přístup ke stovkám tisíc opakovaně použitelných balíčků. Má také nejlepší rozlišení závislostí ve své třídě a lze jej také aplikovat k automatizaci většiny sestavovacích nástrojů.
- Je přenosný, má verze pro Microsoft Windows, OS X, Linux, Solaris, FreeBSD, OpenBSD, WebOS a NonStop OS. Kromě toho má dobrou podporu mezi mnoha poskytovateli hostingu, kteří často poskytují

specifickou infrastrukturu a dokumentaci pro hostování webů provozovaných v uzlu.

- Má velmi aktivní ekosystém a komunitu vývojářů, kteří jsou vždy připraveni pomoci.

Další běžné úlohy programování webu nejsou Node.js přímo podporovány, pokud chceme přidat konkrétní podporu pro různé metody HTTP (například GET, POST, DELETE atd.) Pro různé cesty URL („trasy“), vracení statických souborů nebo použití šablon k vytvoření dynamických odpovědí musíme však napsat kód ručně, nebo použít framework Express (Herron, 2020).

#### 3.4.6.2 Express

Express je pro Node.js asi jedním z nejoblíbenějších webových frameworků. Poskytuje následující mechanismy (Lim, 2019):

- Psaní zpracovávače pro požadavky s různými metodami HTTP v různých URL-adresách (trasách);.
- Integrace s renderovacími mechanismy „view“ pro generování odpovědí vložením dat do šablon.
- Nastavení obecných parametrů webové aplikace, jako je port pro připojení, a umístění šablon, které se používají k zobrazení odpovědi.
- „middleware“ pro další zpracování požadavků kdykoli v kanálu zpracování požadavků.

Zatímco samotný Expres je docela minimalistický, vytvořili vývojáři kompatibilní middlewarové balíčky, aby bylo možno vyřešit téměř jakýkoli problém s vývojem webu. Existují knihovny pro práci s cookies, relacemi, přihlašovacími údaji, parametry URL, POST data, hlavičky zabezpečení a mnoho dalších (Lim, 2019).

## 3.5 Systém řízení báze dat

DBMS (podle anglického database management system) je obecný pojem, který odkazuje na všechny druhy zcela odlišných nástrojů, a to od počítačových programů po vestavěné knihovny. Tyto aplikace spravují nebo pomáhají spravovat datové sady. Protože tato data mohou mít různé formáty a velikosti, byly vytvořeny různé typy DBMS.

DBMS jsou založeny na databázových modelech – definovaných strukturách pro zpracování dat. Každý DBMS je navržen tak, aby pracoval s jedním z nich, s přihlédnutím ke zvláštnostem operací s informacemi (Purewal, 2014).

### 3.5.1 SQL a NoSQL

SQL znamená „Structured Query Language“. *Relační databáze* je datový soubor s předdefinovanými vztahy mezi nimi. Tato data jsou organizována jako sada tabulek sestávajících ze sloupců a řádků. Tabulky ukládají informace o objektech reprezentovaných v databázi. Každý sloupec tabulky ukládá konkrétní datový typ, v každé buňce nalezneme hodnotu atributu. Každý řádek tabulky je sada hodnot souvisejících s jedním objektem nebo entitou. Každý řádek v tabulce lze označit jedinečným identifikátorem nazývaným primární klíč a řádky z několika tabulek lze propojit pomocí cizích klíčů. K těmto datům lze přistupovat mnoha způsoby, přičemž není nutná reorganizace databázových tabulek.

SQL databáze ukládají strukturovaná data, která obvykle představují objekty reálného světa. Řekněme, že se jedná o informace o osobě nebo o obsahu košíku v obchodě, které jsou seskupené do tabulek, jejichž formát je nastaven ve fázi navrhování skladu.

Struktura NoSQL je odlišná. Například databáze zaměřené na dokumenty ukládají informace ve formě hierarchických datových struktur. Můžeme mluvit o objektech s libovolnou sadou atributů. To, co bude v relační databázi rozděleno do několika vzájemně propojených tabulek, může být uloženo v nerelační databázi jako integrální entita. Nerelační databáze jsou však lépe škálovatelné (Hills, 2016).

Je těžké jednoznačně říci, že ten či onen model je lepší. Každý z nich je navržen tak, aby řešil své vlastní problémy. Relační databáze mají řadu výhod, které jsou pro tuto práci relevantní:

- Jednou z důležitých výhod relačního přístupu je jeho jednoduchost a snadné pochopení koncovým uživatelem. Jedinou informační strukturou je tabulka.
- Při navrhování relačních databází platí přísná pravidla založená na matematickém aparátu.
- Relační model poskytuje úplnou nezávislost dat. Změníme-li strukturu relační databáze, úpravy, které je třeba provést v aplikačních programech, jsou obvykle minimální.

- Manipulace s daty na úrovni jazyka DBMS se provádí ne-navigačně, proto není pro vytváření dotazů a psaní aplikačních programů nutné znát konkrétní organizaci databáze v externí paměti. Samozřejmě se při provádění dotazů na fyzické úrovni realizuje navigace v záznamech tabulky, ale tyto akce se provádějí postupy samotného DBMS.

### 3.5.2 PostgreSQL

PostgreSQL je populární bezplatný systém pro správu objektově-relačních databází. PostgreSQL je založen na SQL a má mnoho funkcí. Jednou ze silných stránek PostgreSQL je jeho architektura. Stejně jako mnoho komerčních databází DBMS lze PostgreSQL použít v prostředí klient-server, což poskytuje mnoho výhod pro uživatele i vývojáře.

Jádrem PostgreSQL je back-end proces databáze. Běží na jednom serveru. Přístup z aplikací k databázovým datům se provádí prostřednictvím databázového procesu. Klientské programy nemohou samy přistupovat k datům, i když jsou spuštěny ve stejném počítači jako proces serveru (Schonig, 2018).

Toto oddělení klientů a serveru umožňuje vybudovat distribuovaný systém. Je možno oddělit klienty od serveru prostřednictvím sítě a vyvíjet klientské aplikace v uživatelsky přívětivém prostředí.

Výhody PostgreSQL (Schonig, 2018):

- *Open source:* PostgreSQL je bezplatný a otevřený systém správy objektově-relačních dat, při absenci konvenčních RDBMS. Je vhodný pro použití jak objektově orientovaných databází, tak databází založených na datech.
- *Pokročilé nástroje:* Můžeme spouštět vlastní programy a přizpůsobovat PostgreSQL podle svých potřeb.
- *Zodpovědná komunita:* Drtivá komunita vývojářů je vždy připravena pomoci. Kromě toho je možno využít placenou podporu centrálních společností. Komunita podporuje PostgreSQL a aktualizuje platformu prostřednictvím globálního týmu PostgreSQL.
- *Vysoké hodnocení uživatelů:* PostgreSQL má 4,4 hvězdičky (z pěti), a to na základě 452 recenzí na G2 Crowd.

## 3.6 Bezpečnost

Dostupnost webové aplikace odkudkoli a z jakéhokoli zařízení je nepochybně její obrovskou výhodou, ale zároveň je důvodem vzniku řady hrozeb ze strany útočníků. Komunita OWASP (Open Web Application Security Project) zkoumá chyby zabezpečení ve webových aplikacích a na webových stránkách. Jednou za 3–4 roky publikuje seznam 10 nejběžnějších ohrožení těchto systémů. Z nejnovější zprávy jsou běžné 3 (OWASP, 2020):

*Injekce* – zranitelnost injekcí SQL, NoSQL, OS a LDAP nastane, když jsou neověřená data odeslána tlumočníkovi jako součást příkazu nebo dotazu. Škodlivá data mohou způsobit, že tlumočník provede neočekávané příkazy nebo přistupuje k datům bez řádného povolení.

*Cross-Site Scripting* – XSS nastane, jestliže aplikace přidá neověřená data na novou webovou stránku, aniž by ji ověřila nebo transformovala, nebo když aktualizuje otevřenou stránku prostřednictvím rozhraní API prohlížeče pomocí dat dodaných uživatelem, která obsahují kód HTML nebo JavaScript. Je zvlášť nebezpečný pro uživatele, protože útok XSS je zaměřen přesně na ně. Útočník obecně vloží do webové aplikace skript, který se spustí u každého uživatele, který navštíví škodlivou stránku. Pomocí XSS mohou útočníci v prohlížeči oběti spouštět skripty, které jim umožňují zachytit relace uživatelů, zfalšovat stránky webu nebo přesměrovat uživatele na škodlivé weby.

*Nesprávná konfigurace nastavení zabezpečení* je běžnou chybou. Dochází k ní v důsledku použití standardního nastavení zabezpečení, neúplné nebo konkrétní konfigurace, otevřeného cloudového úložiště, nesprávných záhlaví HTTP a podrobných chybových zpráv obsahujících citlivá data. Všechny operační systémy, rámce, knihovny a aplikace musejí být nejen správně nakonfigurovány, ale také včas opraveny a aktualizovány.

Existuje mnoho způsobů, jak tyto chyby zabezpečení zneužít. Proto již obsahuje většina frameworků ochranu před nejběžnějšími typy útoků. Kromě toho je pro základní ochranu uživatelských dat nutno použít systém ověřování uživatelů webové aplikace (Ludin, 2017):

- Identita – je prohlášení o tom, kdo jste. V závislosti na situaci to může být získání účtu (identity) pomocí uživatelského jména nebo e-mailu.

- Ověření – poskytnutí důkazu, s kým jste ve skutečnosti totožní – ověření, že znáte heslo pro tento účet.
- Autorizace – kontrola vaší role v systému a rozhodnutí, zda udělit přístup k požadované stránce nebo zdroji.

Další základní metodou zabezpečení je použití protokolu HTTPS (Hypertext Transport Protocol Secure), který lze na webovém serveru použít k zašifrování veškerého provozu mezi serverem a uživatelem, včetně přihlašovacích údajů, které by jinak byly odeslány jako prostý text (který kdokoli, kdo zachytí požadovat osobu). Důrazně doporučujeme používat HTTPS (Ludin, 2017).

Webové stránky využívající HTTPS jsou chráněny protokolem TLS (Transport Layer Security), který poskytuje tři hlavní vrstvy ochrany (Ludin, 2017):

- Šifrování přenášených dat, aby se zabránilo jejich zachycení. Díky tomu nebudou útočníci schopni zjistit, jaké informace si návštěvníci webových stránek vyměňují, ani sledovat jejich akce na jiných stránkách nebo získat přístup k jejich údajům.
- Bezpečnost údajů. Jakákoli změna nebo zkreslení přenášených dat bude zaznamenána bez ohledu na to, zda k tomu došlo úmyslně či nikoli.
- Ověřování zajišťuje, že se návštěvníci dostanou na přesné místo, které chtějí, a chrání je před útokem prostředníka. Uživatelé těmto webům důvěřují více.

# 4 Vlastní práce

Tato část práce popisuje proces vytváření IS s ohledem na technologie popsané v teoretické části.

Zpočátku používal zákazník systém „Clinic2004“, který plně vyhovoval jeho potřebám, až na to, že program pracoval pouze v operačním systému Windows a měl omezený počet licencí. Zaměstnanci s ním tedy mohli pracovat pouze mimo své pracoviště. Poté, co vývojář přestal systém v roce 2014 podporovat, bylo nutno hledat alternativní řešení, kterým se stal systém „ZubnayaFeya”, který umožňoval vzdálený přístup, ale kvůli vysokým nákladům na jeho provoz a nepohodlnému rozhraní se vedení centra rozhodlo jej opustit. Bylo rozhodnuto objednat systém přímo pro potřeby centra.

Cílem této diplomové práce je proto poskytnout jednoduchý a pohodlný systém, který umožní rychle a snadno spravovat pracovní plán a objednat pacienta na návštěvu lékaře.

## 4.1 Popis systému a seznam požadavků

Podle požadavků zákazníka byl sestaven následující popis IS: Do systému mají přístup pouze zaměstnanci centra – ředitel, lékaři a administrátoři. Po přihlášení do systému může ředitel upravit otevírací dobu centra, přidat nebo odebrat ordinace, přidat nebo odebrat zaměstnance. Během přidání zaměstnance do systému přiřadí ředitel jeho jméno, pozici, specializaci a přihlašovací údaje pro vstup do systému. Objednání schůzky pacienta s lékařem provádí recepční, která předem zadá pacienta do databáze. Zaznamená základní osobní údaje pacienta, jako je jméno, datum narození, rodné číslo, číslo pojištění, adresa, telefon, e-mail, preferovaný způsob oznámení – telefonicky, e-mailem nebo SMS – a také alergie pacienta, pokud existují. Dále se vyplní pokyn k objednání schůzky, kde je uvedeno jméno lékaře, který bude provádět léčbu, datum a čas schůzky, ordinace a také popis potíží klienta, diagnóza a komentář pro lékaře. Recepční může také upravit nebo odstranit již existující objednávku. Po uložení se objednávka promítne do pracovního plánu. V minimalizovaném stavu se zde zobrazí pouze jméno ošetřujícího lékaře, jméno pacienta a indikátor ukazující, zda byla pacientovi schůzka připomenuta. Harmonogram lze zobrazit pro aktuální den a pro všechny ordinace s možností filtrování schůzek pro konkrétního lékaře, nebo pro jednotlivé lékaře na celý týden. Pracovní rozvrh je viditelný pro všechny uživatele. Záznamy by se neměly navzájem překrývat a mezi schůzkami s pacienty by mělo být 30minutové okno, aby se ordinace připravila na dalšího pacienta. Lékař a ředitel

mohou pacienta rovněž objednat. Lékař může pro nemocného objednat schůzku s jiným lékařem, ale nemůže upravovat záznamy. V případě změn v objednávce je nutno o tom pacienta informovat, což provádí pouze administrátor. Lékař může zanechat žádost o změnu svého pracovního harmonogramu, která vstoupí v platnost až po schválení ředitelem. Systém by měl také zabránit kolizím, jako jsou dvě schůzky s jedním lékařem současně, pracovní plán lékaře by měl odpovídat pracovnímu plánu centra atd. Rozhraní webové aplikace by mělo být jednoduché, intuitivní, bez použití jasných, příliš kontrastních barev a nemělo by být vizuálně přetížené.

### 4.1.1 Seznam aktérů

1. *Ředitel* má nejvyšší úroveň přístupu. Má schopnost přidávat a odebírat zaměstnance do systému, spravovat pracovní plán a záznamy o objednaných návštěvách.
2. *Recepční* odpovídá za práci s pacienty. Může zaregistrovat pacienta při první návštěvě a objednat pro pacienta schůzky s lékařem.
3. *Lékař* může pro pacienta naplánovat další schůzku a upravit svůj pracovní rozvrh.

### 4.1.2 Seznam požadavků

Systém musí poskytovat uživateli následující možnosti:

- přihlásit se do systému;
- přidat zaměstnance do systému;
- smazat zaměstnance;
- přidat ordinace do systému;
- smazat ordinace;
- mít různé úrovně přístupu pro zaměstnance;
- nastavit provozní dobu centra;
- upravit vlastní pracovní rozvrh;
- schválit změny v pracovním rozvrhu;
- zobrazit pracovní plán;
- zaregistrovat pacienta při první návštěvě;
- objednat pacienta na návštěvu k lékaři;
- upravit/smazat objednávku

### 4.1.3 Přístupová práva

**Tabulka 1 Přístupová práva zaměstnanců**

| | Ředitel | Administrátor | Lékař |
|---|---|---|---|
| Přihlásit se do systému | + | + | + |
| Přidat zaměstnance do systému | + | - | - |
| Smazat zaměstnance | + | - | - |
| Přidat ordinace do systému | + | - | - |
| Smazat ordinace | + | - | - |
| Nastavit provozní dobu centra | + | - | - |
| Upravit vlastní pracovní rozvrh | + | - | + |
| Schválit změny v pracovním rozvrhu | + | - | - |
| Zobrazit pracovní rozvrh | + | + | + |
| Zaregistrovat pacienta při první návštěvě | + | + | - |
| Objednat pacienta na návštěvu k lékaři | + | + | + |
| Upravit objednávku | + | + | - |
| Smazat objednávku | + | + | - |

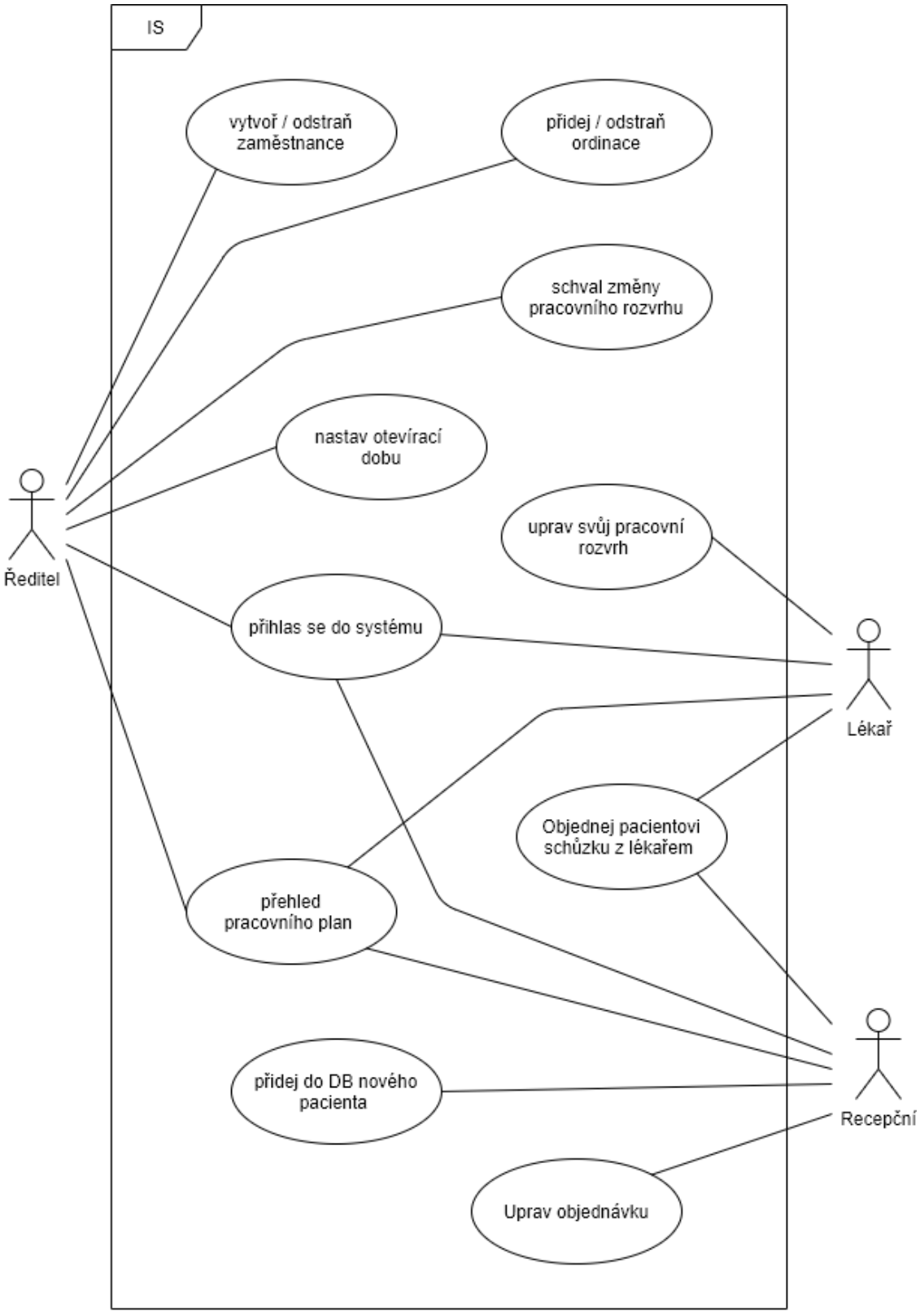
Zdroj: Vlastní zpracování

## 4.2 Analýza systému

### 4.2.1 Use Case diagram

Tento diagram formálně zobrazuje strukturu funkčnosti systému s ohledem na komunikaci s dalšími aktéry. Use Case diagram byl vytvořen na základě seznamu požadavků.

**Obrázek 9 Use Case diagram systému**




Zdroj: Vlastní zpracování

### 4.2.2 Scénář použiti

Scénáře použití popisují postupné kroky, kterými uživatel a systém procházejí při vzájemné interakci. Následují tři hlavní případy použití:

1. Registrace pacienta na schůzku s lékařem a jeho přidání do databáze.
2. Přihlášení ředitele do systému a vytvoření nového zaměstnance.
3. Lékař vytvoří požadavek na změnu pracovního rozvrhu.

První z nich popisuje situaci, kdy pacient přijde do centra a chce si domluvit schůzku s lékařem, v tomto případě je obsluhován recepční.

**Tabulka 2 případ použití 1**

| | | |
|---|---|---|
| **Název** | Registrace pacienta na schůzku s lékařem a jeho přidání do databáze | |
| **Hlavní aktér** | Recepční | |
| **Trigger** | Pacient přišel poprvé se stížnostmi na bolest zubů a chce si domluvit schůzku k léčbě | |
| **Výsledek** | Vytvořena objednávka na schůzku s lékařem. | |
| **Základní tok** | | |
| | **Role** | **Akce** |
| 1 | Recepční | Přihlášení |
| 2 | Systém | Přihlášení bylo úspěšným |
| 3 | Recepční | Přidat nový záznam |
| 4 | Systém | Zadejte jméno a rodné číslo |
| 5 | Recepční | Zadá jméno a příjmení |
| 6 | Recepční | Zadá rodné číslo |
| 7 | Systém | Zkontrolujte, zda pacient existuje v databázi |
| 8 | Systém | Pacient neexistuje, vyplňte registrační formulář |
| 9 | Recepční | Zadávání údajů o pacientovi |
| 10 | Systém | Vyberte lékaře |

| | | |
|---|---|---|
| 11 | Recepční | Výběre lékaře ze seznamu |
| 12 | Systém | Nastavte datum plánované návštěvy |
| 13 | Recepční | Výběre data |
| 14 | Systém | Kontrola, zda lékař v ten den pracuje |
| 15 | Systém | Vyberte ordinace pro příjem |
| 16 | Recepční | Vybere ordinace ze seznamu |
| 17 | Systém | Zadejte čas zahájení a odhadovaný čas ukončení léčby |
| 18 | Recepční | Zadá času zahájení a ukončení příjmu |
| 19 | Systém | Kontrola, zda čas odpovídá pracovnímu plánu lékaře |
| 20 | Systém | Zkontrolujte, zda je lékař v tuto chvíli volný |
| 21 | Systém | Zkontrolujte, zda je kancelář v tuto chvíli volná |
| 22 | Systém | Vyplňte formulář stížnosti a diagnózy |
| 23 | Recepční | Vyplní formulář |
| 24 | Recepční | Ulož pacienta do databáze a vytvoř objednání |
| 25 | Systém | Přidání pacienta do databáze |
| 26 | Systém | Automatická rezervace 30 minut po ukončení léčby |
| 27 | Systém | Vytvoření plánované schůzky v pracovním plánu |

Zdroj: Vlastní zpracování

Druhý případ použití popisuje přihlášení ředitele do systému a přidání nového zaměstnance, a to s uvedením jeho jména, role, specializace, přihlašovacího jména a hesla.

**Tabulka 3 případ použití 2**

| **Název** | | Ředitel vytvoří nového zaměstnance |
|---|---|---|
| **Hlavní aktér** | | Ředitel |
| **Trigger** | | Je třeba do systému přidat nového lékaře |
| **Výsledek** | | Zaměstnanec úspěšně přidán |
| **Základní tok** | | |
| | **Role** | **Akce** |
| 1 | Ředitel | Přihlášení |
| 2 | Systém | Přihlášení bylo úspěšným |
| 3 | Ředitel | Přidat nového zaměstnance |
| 4 | Systém | Zadejte jméno |
| 5 | Ředitel | Zadá jména a příjmení |
| 6 | Systém | Vyberte roli tohoto uživatele |
| 7 | Ředitel | Zadá roli uživatele |
| 8 | Systém | Vyberte specializaci tohoto uživatele |
| 9 | Ředitel | Vybírá specializaci |
| 10 | Systém | Zadejte přihlašovací jméno a heslo pro tohoto uživatele |
| 11 | Ředitel | Vytvoří pro tohoto uživatele přihlašovací jméno a heslo |
| 12 | Ředitel | Vytvoř nového uživatele |
| 13 | Systém | Uživatel XXX byl úspěšně vytvořen |

Zdroj: Vlastní zpracování

T Třetí scénář popisuje situaci, kdy lékař chce změnit svůj pracovní plán, pro který provede změny ve stávajícím rozvrhu a ponechá požadavek na ředitele.

**Tabulka 4případ použití 3**

| | | |
|---|---|---|
| **Název** | | Lékař vytvoří požadavek na změnu svého pracovního harmonogramu |
| **Hlavní aktér** | | Lékař |
| **Trigger** | | Lékař chce vytvořit požadavek na změnu pracovního plánu |
| **Výsledek** | | Požadavek byl úspěšně vytvořen |
| **Základní tok** | | |
| | **Role** | **Akce** |
| 1 | Lékař | Přihlášení |
| 2 | Systém | Přihlášení bylo úspěšným |
| 3 | Lékař | Změnit pracovní plán |
| 4 | Systém | Zadejte požadovaný čas začátku a konce pro každý den týdnu |
| 5 | Lékař | Zavedení nových časů zahájení a ukončení |
| 6 | Lékař | Poslat žádost |
| 7 | Systém | Kontrola souladu harmonogramu s harmonogramem centra |
| 8 | Systém | Váš požadavek byl úspěšně vytvořen |

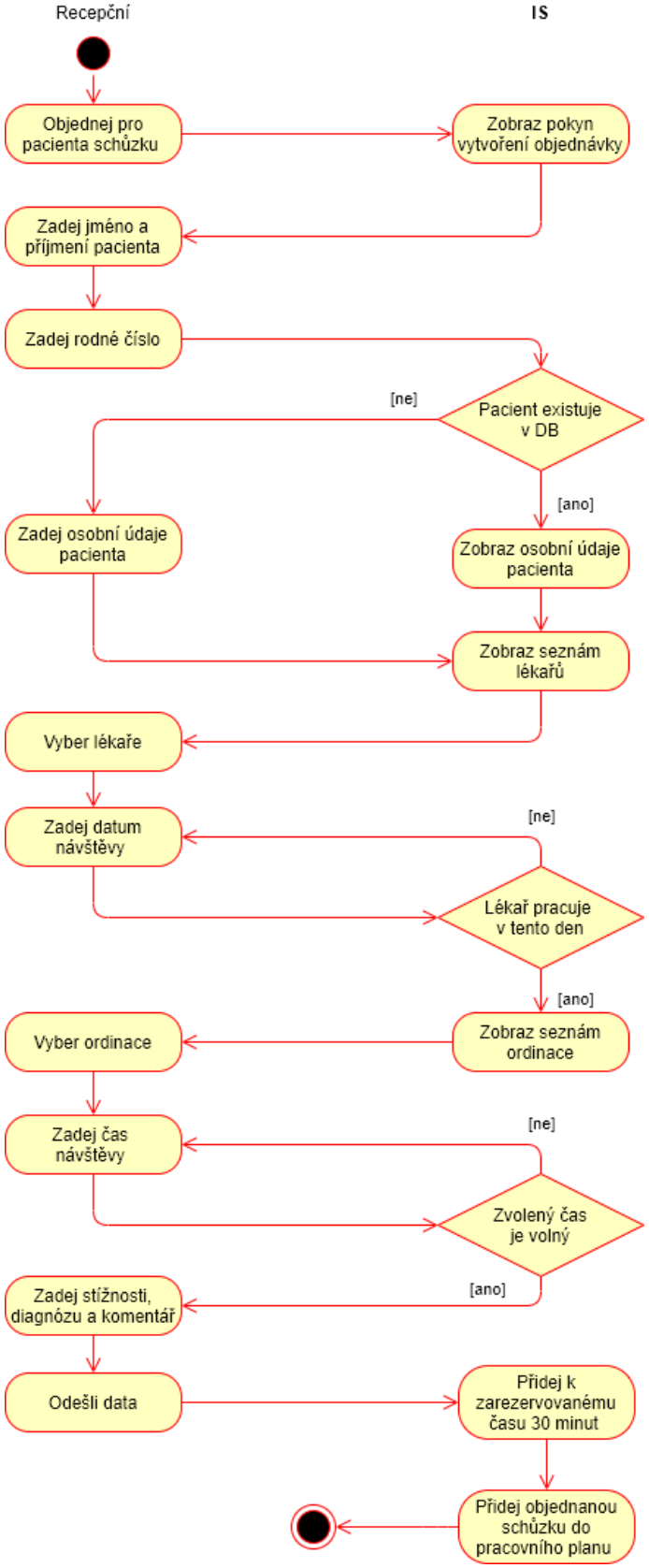
Zdroj: Vlastní zpracování

### 4.2.3 Podrobnější definice Use Casu

Dalšími kroky v analýze jsou podrobnější zpracování existujících případů použití v diagramech aktivit. Níže jsou uvedeny dva diagramy, které popisují proces registrace pacienta na schůzku s lékařem a přidání pacienta do databáze.

Nejprve zadá recepční jméno a identifikační číslo pacienta. Systém ověří, zda pacient v databázi již existuje. Pokud je pacient v databázi uveden, začne administrátor objednávání schůzky s lékařem, pokud zde není, vyplní o pacientovi všechna potřebná data. Dále vybere recepční lékaře a ordinaci, ve které se bude schůzka konat. Poté vybere datum a čas a systém zkontroluje, zda je tento čas volný. Pokud tomu tak není, zvolí jiný čas a zadá do objednávky stížnosti pacienta, případně diagnózu a komentář pro lékaře. Nakonec systém uloží data a vytvoří záznam.

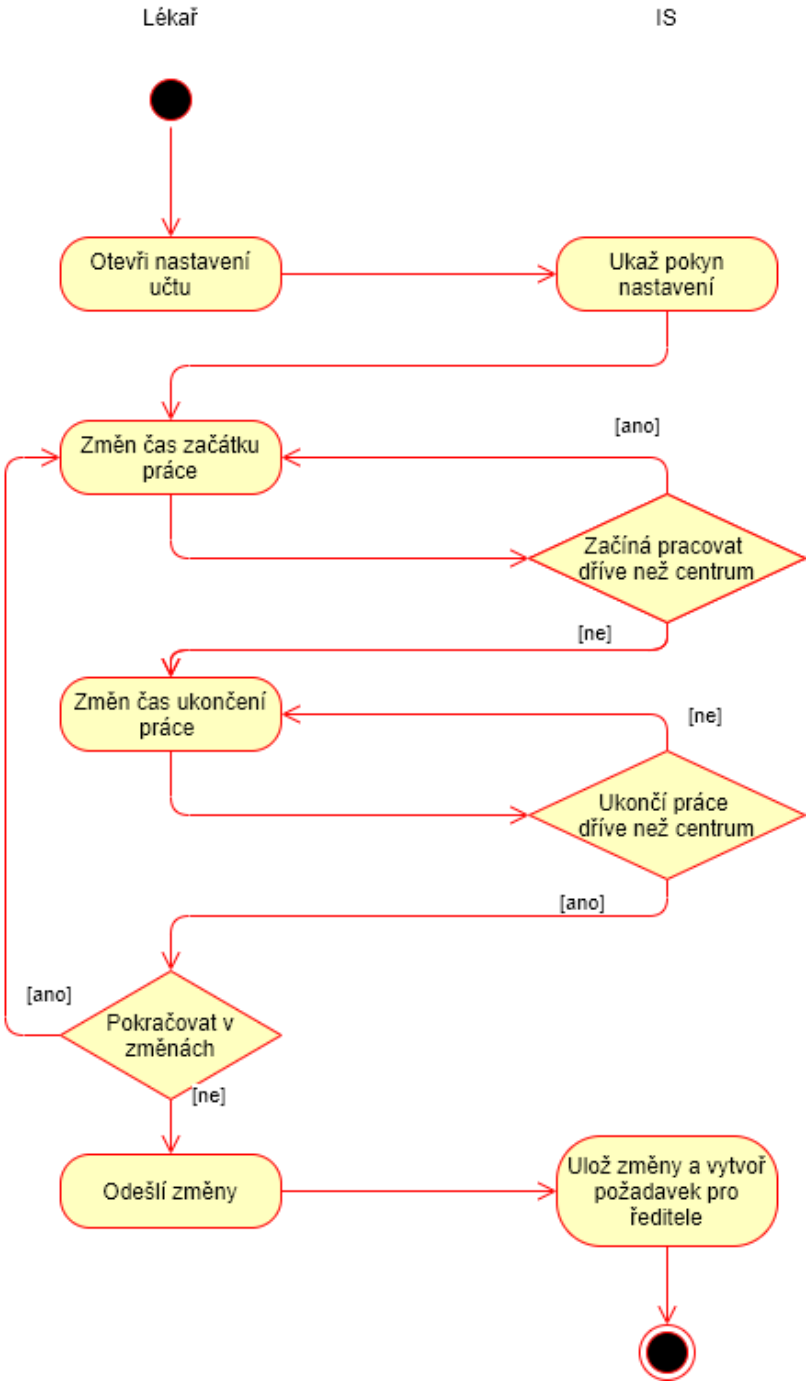
**Obrázek 10 diagram aktivit 1**




Zdroj: Vlastní zpracování

Vytvoření dotazu, zda je možno změnit pracovní plán lékaře. Na stránce nastavení může lékař změnit svůj pracovní plán zadáním nových časů začátku a ukončení práce pro každý pracovní den v týdnu do systému. Systém musí tyto údaje zkontrolovat, aby nebyl překročen čas provozní doby centra, a pak vytvořit dotaz na změnu pracovního plánu, který bude očekávat schválení ředitele.

**Obrázek 11 diagram aktivit 2**

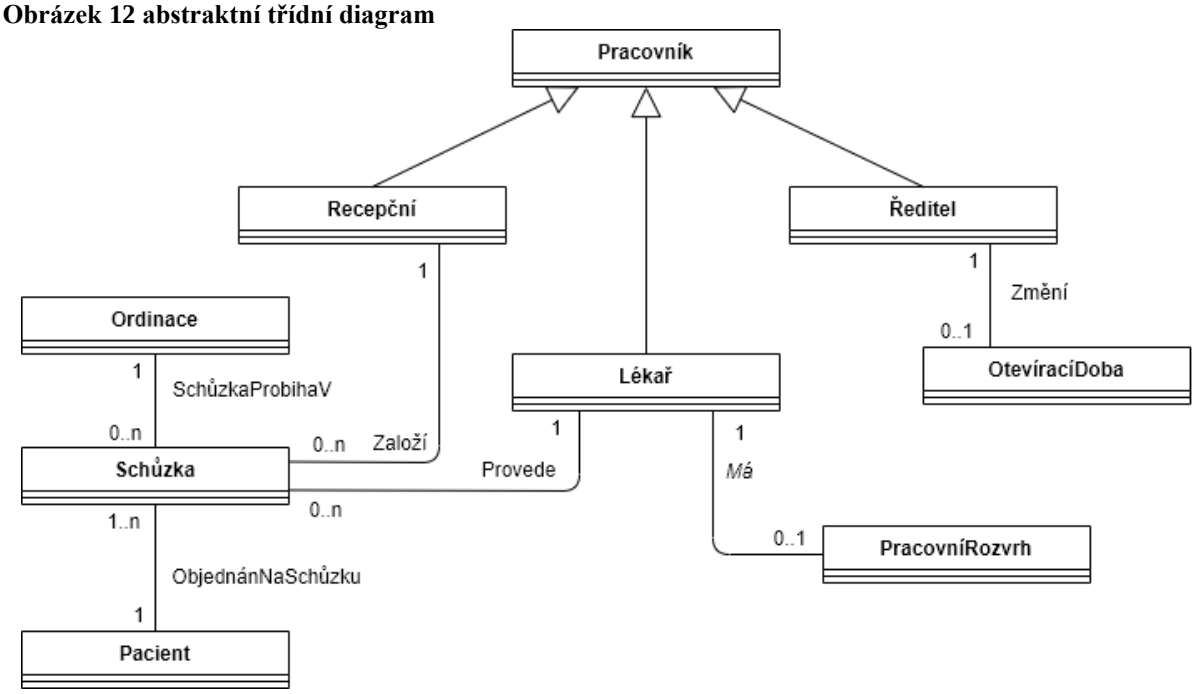



Zdroj: Vlastní zpracování

### 4.2.4 Abstraktní třídní diagram

Na základě předchozí analýzy byl sestaven abstraktní diagram uvádějící základní třídy v systému a vztahy mezi nimi.

**Obrázek 12 abstraktní třídní diagram**




Zdroj: Vlastní zpracování

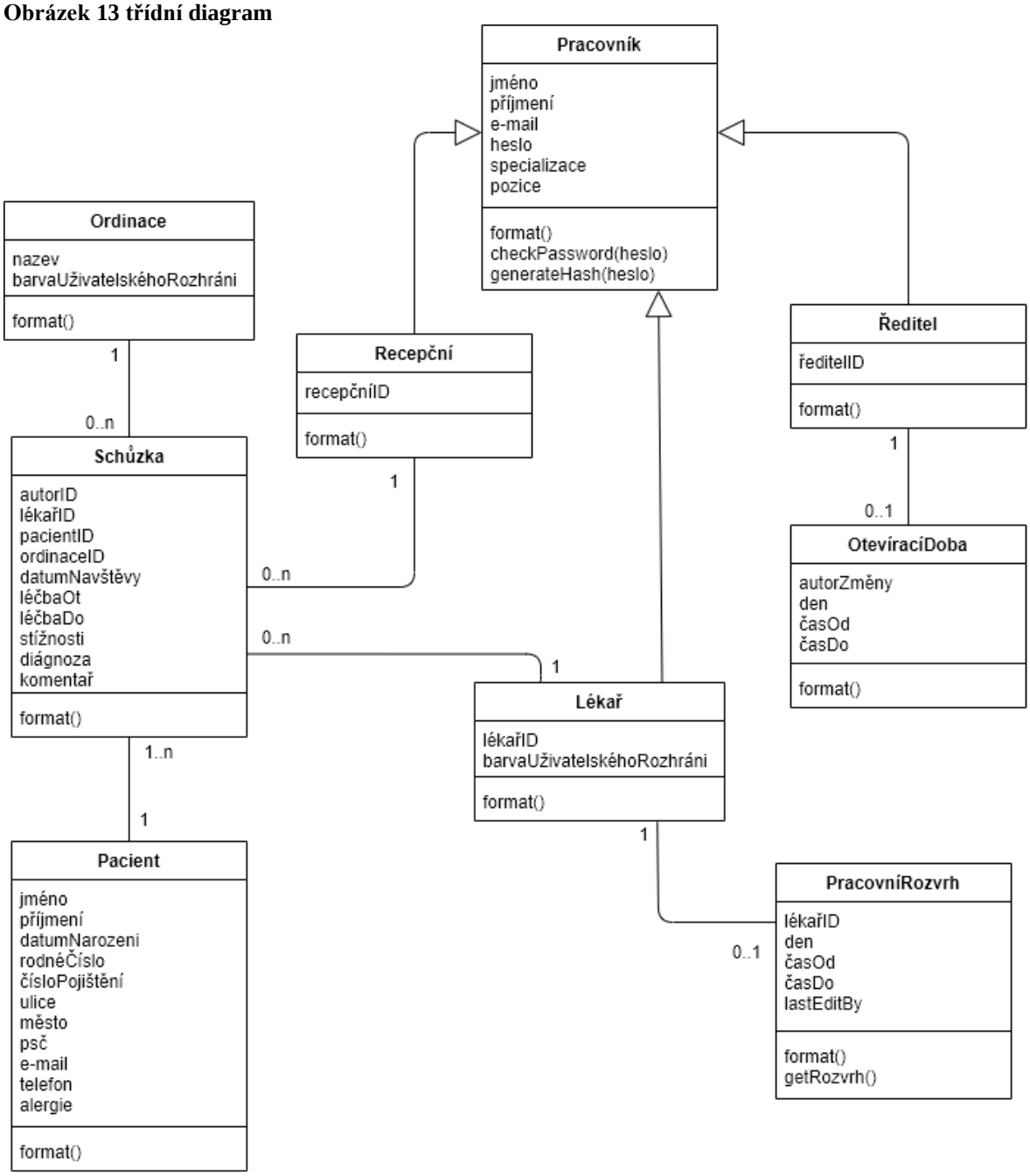
Dále na základě těchto tříd byl sestaven návrhový třídní diagram a datový slovník, který popisuje účel tříd a datové typy použité v nich.

## 4.3 Návrh systému

### 4.3.1 Třídní diagram a relační datový model

Následující diagram tříd zobrazuje objektový návrh, popisující systémová data a vztahy mezi nimi. Dále na základě třídního diagramu se sestaví uživatelské rozhraní a vytvoří se databáze.

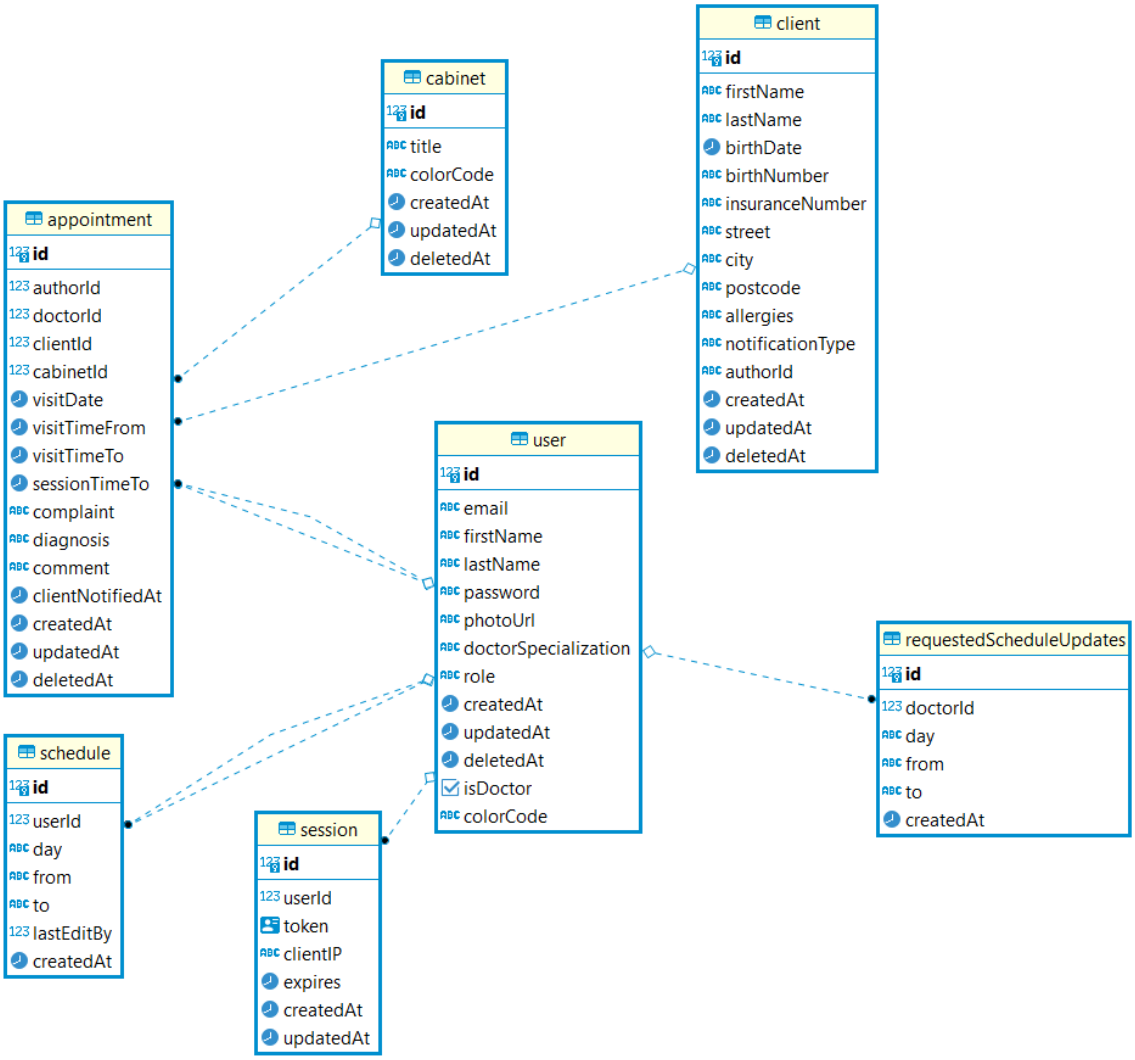
**Obrázek 13 třídní diagram**




Zdroj: Vlastní zpracování

Dalším krokem je vytvoření relačního databázového modelu. Relační datový model popisuje databázi, její strukturu a vlastnosti i manipulaci s daty. Představuje informace o objektu, představované jako dvourozměrné pole. Název pole – určuje, jak má být k datům tohoto pole přistupováno během automatických operací s databází (ve výchozím nastavení se názvy polí používají jako záhlaví sloupců tabulky). Datový typ je určen hodnotami, které budou zadány do tohoto pole.

**Obrázek 14 relační databázový model**




Zdroj: Vlastní zpracování

### 4.3.2 Datový slovník

***User*** – tato tabulka popisuje pracovníka, obsahuje přihlašovací údaje e-mail a heslo, foto, jméno a příjmení pracovníka. Pole *doctorSpecialization* specifikuje lékaře. Kvůli tomu že ředitel taky může být lékařem, přidáno pole *isDoctor* logického typu. Každý lékař má pole *colorCode* které obsahuje barvu využívánu pro jeho zobrazení v uživatelském rozhráni. Pole *role* typu enum_user_role obsahuje seznám možných roli (ředitel, recepční a lékař). Role – umožňuje systému rozlišovat mezi zaměstnanci s

různými úrovněmi přístupu. Navíc pamatuje čas přidaní pracovníka, posledních změn a kdy byl odstraněn ze systému.

**Tabulka 5 uživatel**

| **User** | |
|---|---|
| **Pole** | **Datový typ** |
| id | integer |
| firstName | character varying |
| lastName | character varying |
| email | character varying |
| password | character varying |
| photoUrl | character varying |
| doctorSpecialization | character varying |
| role | enum_user_role |
| createdAt | timestamp with time zone |
| updatedAt | timestamp with time zone |
| deletedAt | timestamp with time zone |
| isDoctor | boolean |
| colorCode | character varying |

Zdroj: Vlastní zpracování

***Session*** **–** Tabulka relací uživatelů, vygeneruje se při přihlášení pracovníka do systému. Každá relace je vytvořena samostatně pro každý prohlížeč (webový klient) a je uložena jako cookies v prohlížeči. Její vztah s uživatelem jeden k mnoha. Jeden uživatel může mít mnoho relací.

**Tabulka 6 relací**

| Session | |
|---|---|
| **Pole** | **Datový typ** |
| id | integer |
| userId | integer |
| token | uuid |
| clientApi | character varying |
| expires | timestamp with time zone |
| createdAt | timestamp with time zone |
| updatedAt | timestamp with time zone |

Zdroj: Vlastní zpracování

***Cabinet*** **–** tabulka popisuje ordinace, její název a kód barvy pro zobrazeni v uživatelském rozhraní, a taky čas vytváření, změny a odstranění ze systému.

**Tabulka 7 ordinace**

| Cabinet | |
|---|---|
| **Pole** | **Datový typ** |
| id | integer |
| title | character varying |
| colorCode | character varying |
| createdAt | timestamp with time zone |
| updatedAt | timestamp with time zone |
| deletedAt | timestamp with time zone |

Zdroj: Vlastní zpracování

***Schedule*** **–** tabulka, kde jsou uloženy dny s časem. Používá se pro uložení pracovního rozvrhu lékaře a otevírací dobu centra. Pole *userId* – cizí klíč pro pracovníka

jeden k mnoha. Uživatel má několik řádků pracovního rozvrhu. Jeden řádek na jeden den. *Day* číselník typu řádek - „pondělí“, „úterý“…Pole *from* a *to* zobrazuji pracovní časovou mezeru od-do a ukládá pouze čas. Pole *lastEditBy* - cizí klíč pro uživatele, který si pamatuje, kdo naposledy upravil tuto položku. Může to být lékař nebo ředitel.

**Tabulka 8 rozvrh**

| **Schedule** | |
|---|---|
| **Pole** | **Datový typ** |
| Id | integer |
| userId | integer |
| day | enum_schedule_day |
| from | timestamp with time zone |
| to | timestamp with time zone |
| lastEditBy | integer |
| createdAt | timestamp with time zone |

Zdroj: Vlastní zpracování

***RequestedScheduleUpdates*** **–** tabulka, vygeneruje se při změně lékařem svého pracovního rozvrhu. Pole *doctorId* cizí klíč pro lékaře, který je autorem dotazu. Dále stejně jako schedule má pole *Day* (číselník typu řádek) - „pondělí“, „úterý“…Pole *from* a *to* zobrazuji pracovní časovou mezeru od-do a ukládá pouze čas. Pole *createdAt* – uloží čas vytvoření dotazu.

**Tabulka 9 dotaz na změnu rozvrhu**

| **RequestedScheduleUpdates** | |
|---|---|
| **Pole** | **Datový typ** |
| Id | integer |
| doctorId | integer |

| | |
|---|---|
| day | enum_schedule_day |
| from | timestamp with time zone |
| to | timestamp with time zone |
| createdAt | timestamp with time zone |

Zdroj: Vlastní zpracování

***Client*** **–** tabulka do které se ukládá údaje o pacientovi. Jméno, příjmení, datum narozeni, rodné číslo, adresa atd. Navíc se ukládá Id pracovníka, který přidal pacienta do databáze. A taky čas přidaní, čas poslední změny a čas odstranění pacienta z databázi.

**Tabulka 10 pacient**

| **Client** | |
|---|---|
| **Pole** | **Datový typ** |
| id | integer |
| firstName | character varying |
| lastName | character varying |
| birthDate | date |
| birthNumber | character varying |
| insuranceNumber | character varying |
| street | character varying |
| city | character varying |
| postcode | character varying |
| allergies | character varying |
| notificationType | character varying |
| autorId | integer |
| createdAt | timestamp with time zone |

| | |
|---|---|
| updatedAt | timestamp with time zone |
| deletedAt | timestamp with time zone |

Zdroj: Vlastní zpracování

***Appointment* –** tabulka do které se ukládá údaje o objednané schůzce s lékařem, a proto má pole *authorId, doctorId, clientId, cabinetId* – jsou to cizí klíče které ukazuji kdo objednával schůzku, lékaře, který provede léčbu, objednaný pacient a ordinace kde léčba bude probíhat. Appointment ukládá datum schůzky, předpokládaný časový interval, diagnóza, stížnosti pacienta a komentář pro lékaře který bude provádět léčbu.

**Tabulka 11 schůzka**

| **Appointment** | |
|---|---|
| **Atribut** | **Datový typ** |
| id | integer |
| authorId | integer |
| doctorId | integer |
| clientId | integer |
| cabinetId | integer |
| visitDate | date |
| visitTimeFrom | timestamp with time zone |
| visitTimeTo | timestamp with time zone |
| sessionTimeTo | timestamp with time zone |
| complaint | character varying |
| diagnosis | character varying |
| comment | character varying |
| clientNotifiedAt | timestamp with time zone |

| | |
|---|---|
| createdAt | timestamp with time zone |
| updatedAt | timestamp with time zone |
| deletedAt | timestamp with time zone |

Zdroj: Vlastní zpracování

### 4.3.3 Prototypování a grafický návrh systému

Výše uvedené požadavky na systém a UML diagramy, a hlavně Use Case diagram jsou nutné pro vytvořeni prototypu uživatelského rozhraní. Prototyp obsahuje vzhled produktu, jeho logiku a základní funkčnost.

První fází prototypování je vytváření modelu aplikace. Jednou z možností rozvržení je wireframe nebo drátový model. Pro jeho zpracovaní byl použitý nástroj Adobe XD. Dále jsou uvedeny základní obrazovky systému.

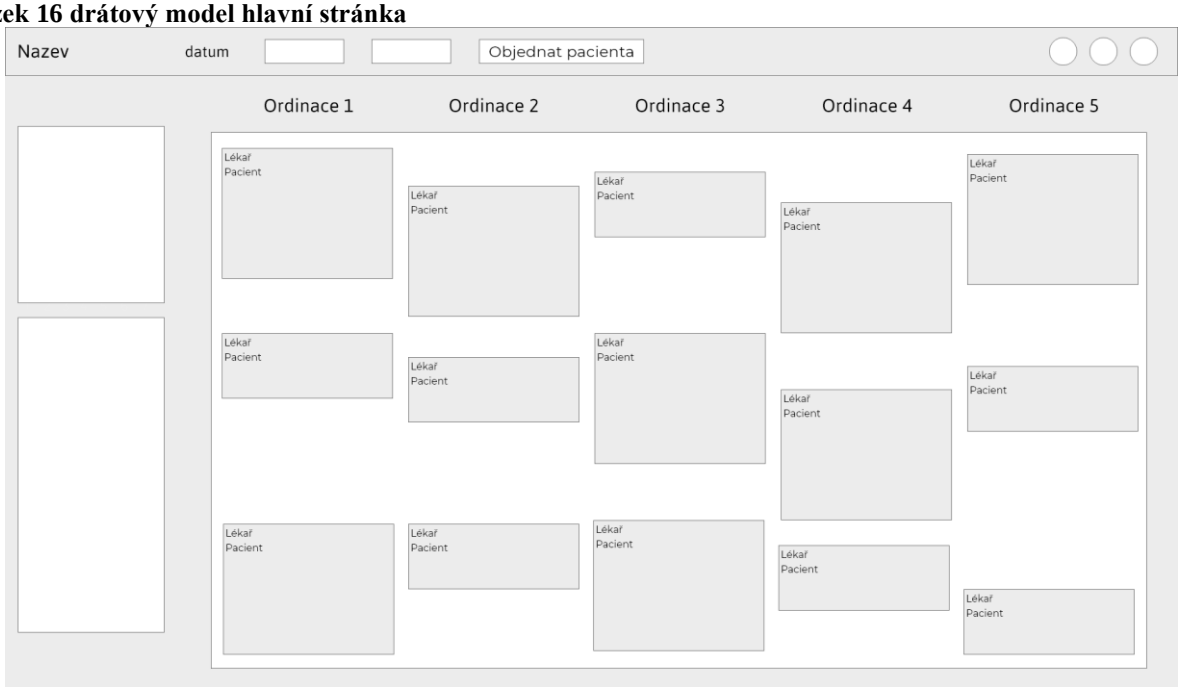
První obrazovka zobrazuje přihlašovací okno. Uživatel musí zadat své uživatelské jméno a heslo.

**Obrázek 15 drátový model přihlášení do systému**


Zdroj: Vlastní zpracování

Po přihlášení se uživateli zobrazí hlavní stránka s pracovním plánem centra pro aktuální den. Vizuálně to bude provedeno ve formě deníkové stránky. Logo a hlavní ovládací prvky budou umístěny v horní části.

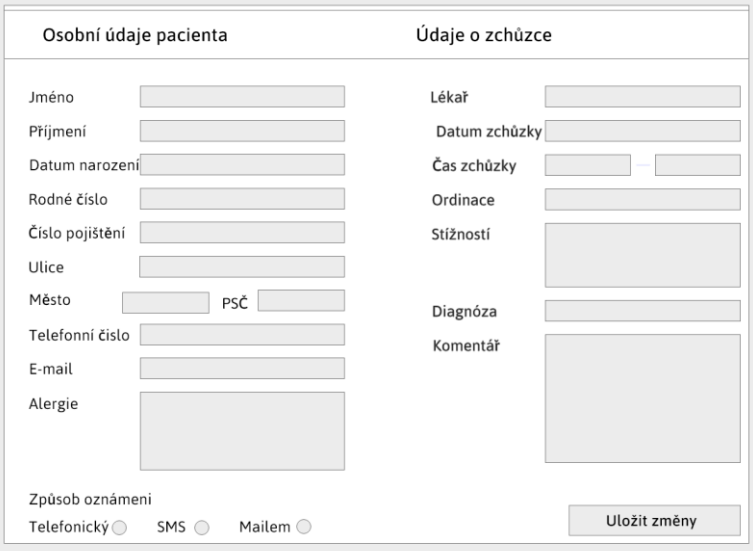
**Obrázek 16 drátový model hlavní stránka**




Zdroj: Vlastní zpracování

V dalším okně se zobrazuje formulář pro přidání pacienta do systému a jeho objednaní na schůzku. Bylo rozhodnuto spojit tyto dva kroky, aby se zjednodušil proces a snížil počet akcí prováděných správcem. Levá část formuláře odpovídá za osobní údaje pacienta. V pravém jsou vyplněny všechny údaje potřebné k registraci u lékaře.

**Obrázek 17 drátový model formulář registrace**




Zdroj: Vlastní zpracování

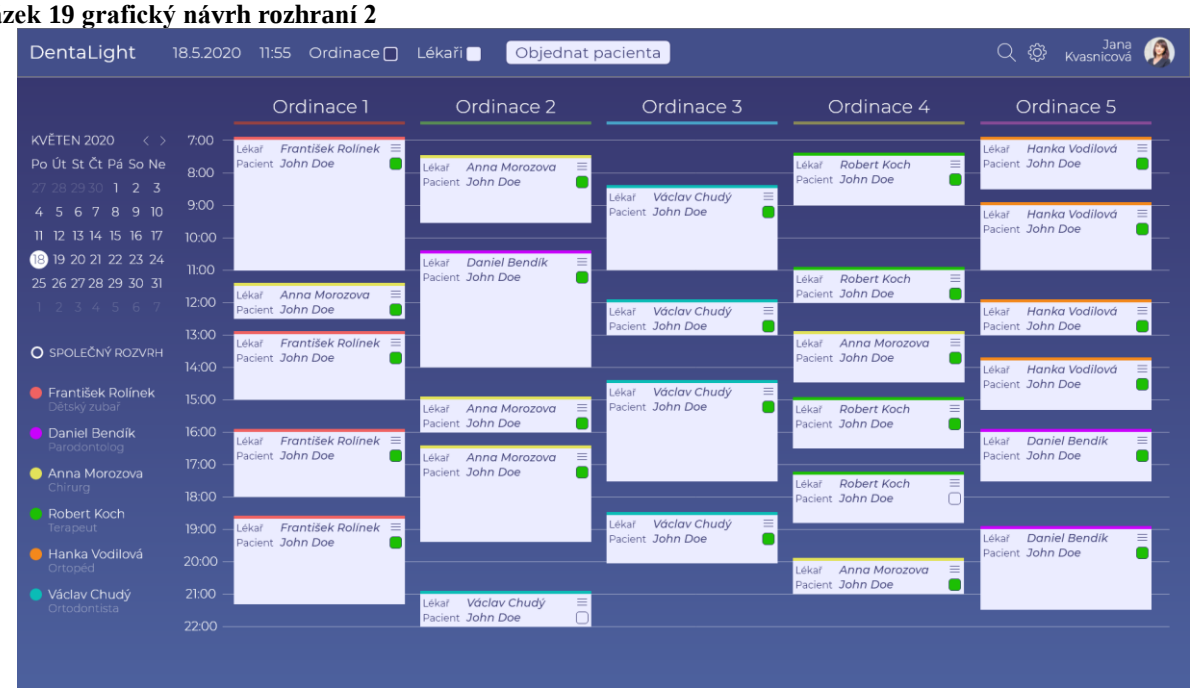
Dalším krokem je přidání stylu a detailů do drátového modelu. S pomocí Adobe XD bylo vytvořeno vzorkové grafické rozhraní aplikace. Jeho účelem bylo podrobněji popsat ovládací prvky a zobrazit požadovaný design uživatelského rozhraní.

**Obrázek 18 grafický návrh rozhráni 1**


Zdroj: Vlastní zpracování

S podrobnějším rozpracováním rozhraní bylo rozhodnuto představit hlavní část ve formě stránky deníku, kde řádky jsou hodiny a sloupce ordinace. Ve sloupci ordinace se zobrazuji jednotlivé schůzky, které budou probíhat v této ordinace. Na štítku schůzky se zobrazuje jméno lékaře, který provede léčbu a jméno pacienta. Navíc se zobrazuje ukazatel, jestli pacient byl oznámen o nadcházející schůzce. Vlevo je zobrazen kalendář aktuálního měsíce a pod ním je seznam lékařů pro snazší filtrování záznamů.

**Obrázek 19 grafický návrh rozhraní 2**




Zdroj: Vlastní zpracování

Po kliknutí na tlačítko „Objednat pacienta“ se otevře vyskakovací okno s formulářem pro přidání pacienta do databáze a objednaní schůzky s lékařem. Zaměstnanec musí vyplnit formulář a po jeho odeslání se v rozvrhu objeví štítek s novou schůzkou.

**Obrázek 20 grafický návrh rozhraní 3**


Zdroj: Vlastní zpracování

## 4.4 Implementace systému

Implementační část obsahuje popis přímé práce s kódem. S využitím vybraných programovacích jazyků JavaScript a Node.JS byla na základě sestaveného návrhu implementována webová aplikace. Níže jsou uvedeny části zdrojového kódu a popis jeho fungování. Veškerý zdrojový kód aplikace je obsažen v archivu opraveném v dalších materiálech.

### 4.4.1 Databáze

Popis databáze byl proveden pomocí migrací, z nichž každá obsahuje určitou změnu ve struktuře databáze: vytvoření tabulky, vytvoření indexu atd. Například, následující část kódu popisuje vytvoření tabulky User, které je přiřazeno ID, které je automaticky vyplněno a je primárním klíčem. Pole role používá číselník tří hodnot. A je uvedeno, že pole je povinné.

```
module.exports = {
  up: (queryInterface, Sequelize) => queryInterface.createTable('user', {
    id: {
      type: Sequelize.INTEGER,
      allowNull: false,
      primaryKey: true,
      autoIncrement: true,
    }, …
    role: {
      type: Sequelize.ENUM({
        values: ['director', 'doctor', 'administrator'],
      }),
      allowNull: false,
    }, … }),
  down: queryInterface => queryInterface.dropTable('user'),
};
```

Chcete-li ovládat a ukládat logickou strukturu databáze, zkontrolujte jedinečnost klíčů atd. používají se unikátní indexy. Dále se pro tabulku User, pro primární klíč a email vytvoří unikátní indexy user_pkey a user_email_key, které kontroluji, zda klič a email jedineční.

```
CREATE UNIQUE INDEX user_pkey ON "user"(id int4_ops);
CREATE UNIQUE INDEX user_email_key ON "user"(email text_ops);
```

### 4.4.2 Serverová část

Aby byla zajištěna bezpečnost, musí být heslo zašifrováno a musí být zkontrolována jeho platnost. To je zajištěno následujícími metodami.

*generateHash(password)* - před zapsáním nového uživatele nebo aktualizací hesla uživatele heslo hashujeme pomocí metody bcrypt, která nám umožňuje zašifrovat heslo v jednom směru, ale díky algoritmům bcrypt neztratíme možnost zkontrolovat platnost přenášeného řetězce (porovnání řetězců). Chrání také uživatele v případě krádeže databáze. Všechna hesla jsou šifrována.

*comparePassword(str)* je funkce, která kontroluje platnost hesla předaného řetězce.

Při vytváření záznamu o naplánované schůzce, aplikace odesíláme data z klientského rozhraní o klientovi (může to být nový klient) a o samotné schůzce. V rámci jednoho kroku v obchodním procesu musíme provést změny ve dvou tabulkách najednou. A pokud v jedné z nich byly chyby zápisu, musíte vrátit všechny změny zpět. K tomu využíváme výhod relačních databází, jako je transakce. Vytvoříme transakci (získáme štítek ve formě dlouhého řetězce), předáme ji jako parametr a na konci transakci potvrdíme nebo zrušíme.

Ukázkový kód:

```
const transaction = await db.sequelize.transaction();
try {
  const newClient = await db.client.create(body, transaction);
  const newAppointment = await db.appointment.create(data, transaction);
  // fix
  await transaction.commit();
} catch (err) {
  // cancel
  await. transaction.rollback();}
```

Při vytváření záznamu o naplánované schůzce musíme zkontrolovat, zda je datum a čas aplikace v rozvrhu lékaře.

```
// is date time in range of chosen doctor's schedule
    const schedule = await db.schedule.getSchedule({ db, userId: doctor.id });
    if (!schedule) {
      throw new SoftError('Schedule is not set for doctor'); }
    const weekDay = db.schedule.days[from.weekday()];
    const limitsFrom = moment(`${visitDate} ${schedule[weekDay].from}`, 'DD-MM-
YYYY HH:mm');
```

```
const limitsTo = moment(`${visitDate} ${schedule[weekDay].to}`, 'DD-MM-YYYY HH:mm');
if (from.isBefore(limitsFrom) || to.isAfter(limitsTo)) {
 throw new InvalidRequestError(`Date time not in limits [${weekDay} ${schedule[weekDay].from}-${schedule[weekDay].to}]`); }
```

Na tomto řádku odkazujeme na naši metodu získání rozvrhu lékaře:

```
const schedule = await db.schedule.getSchedule({ db, userId: doctor.id });
```

Pokud není stanoven plán lékaře, generujeme chybu:

```
if (!schedule) {
 throw new SoftError('Schedule is not set for doctor'); }
```

Určíme, jaký den v týdnu to je, potřebujeme to, abychom pochopili, na jakou dobu v rozvrhu je třeba věnovat pozornost:

```
const weekDay = db.schedule.days[from.weekday()];
```

Poté vytvoříme dvě instance data s časem z plánu:

```
const limitsFrom = moment(`${visitDate} ${schedule[weekDay].from}`, 'DD-MM-YYYY HH:mm');
const limitsTo = moment(`${visitDate} ${schedule[weekDay].to}`, 'DD-MM-YYYY HH:mm');
```

A nakonec samotná kontrola, zda je čas aplikace v požadovaných mezích

Pokud je datum „od“ dřívější než datum „limitsFrom“ nebo datum „do“ pozdější než „limitsTo“, způsobíme chybu:

```
if (from.isBefore(limitsFrom) || to.isAfter(limitsTo)) {
 throw new InvalidRequestError(`Date time not in limits [${weekDay} ${schedule[weekDay].from}-${schedule[weekDay].to}]`);
```

Pokud je datum záznamu o naplánované schůzce v rozvrhu, zbývá pouze zkontrolovat, zda je ordinace k tomuto datu volna.

Uděláme požadavek v databázi při hledání schůzky, která je umístěna ve stejné ordinace (kabinetId), a časy začátku a konce relace se protínají se zadaným časem vytvořené objednávky:

```
// is cabinet free in this time
```

```
const conflictedAppointment = await db.appointment.findOne({
 attributes: ['id'],
 where: {
  cabinetId: cabinet.id,
  [Op.or]: [
   { visitTimeFrom: { [Op.between]: [from.toDate(), to.toDate()] } },
   { visitTimeTo: { [Op.between]: [from.toDate(), to.toDate()] } },
  ],
 },
});
if (conflictedAppointment) {
 throw new AlreadyAcceptedError(`Cabinet is busy by appointment #${conflictedAppointment.id}`);
}
```

Při registraci nového uživatele zkontrolujeme, zda je e-mail jedinečný. Na úrovni databáze již v tomto poli máme jedinečný index, ale musíme vygenerovat uživatelsky přívětivější chybu.

```
// check for duplicates by email
{
 const user = await db.user.findOne({
  attributes: ['id', 'email'],
  where: { email }
 });
 if (user) {
  throw new AlreadyRegisteredError('Email is busy');
 }
```

Koncové body api / auth jsou popsány níže.

Jak vidíme, je označena metoda požadavku (POST, GET),

pak adresa / přihlášení (tj. úplná adresa přístupu k tomuto prostředku bude vytvořena jako http: // host / v1 / auth / signup)

Univerzální funkce, která kontroluje autorizaci uživatele.

Ale protože jsme předali také parametr (objekt s polem role = director), produkujeme. dodatečné ověření, že oprávněný uživatel je ředitel.

```
authenticate ({role: 'director'})
```

A na konci registrace se jedná o odkaz na samotnou funkci, která je zodpovědná za zpracování požadavku na daný koncový bod. V tomto případě vytvoření nového uživatele.

```
router.post('/signup', authenticate({ role: 'director' }), signup);
router.delete('/:id', authenticate({ role: 'director' }), archive);
router.post('/signin', signin);
router.post('/logout', authenticate(), logout);
router.get('/ping', authenticate(), ping);
```

### 4.4.3 Uživatelské rozhraní

Tento skript odpovídá za zpracování přihlašovacího formuláře. Stisknutím klávesy Enter nebo kliknutím na tlačítko se tato funkce vyvolá s parametry e-mail a heslo, které odešlou pole e-mailu a hesla k ověření na server.

```
const LoginPage = () => {
 const dispatch = useDispatch();
 const classes = useStyles();
 const onSubmit = (email, password) => {
  request({
   url: 'http://localhost:3031/v1/auth/signin',
   method: 'POST',
   body: {
    email,
    password,
   },})
```

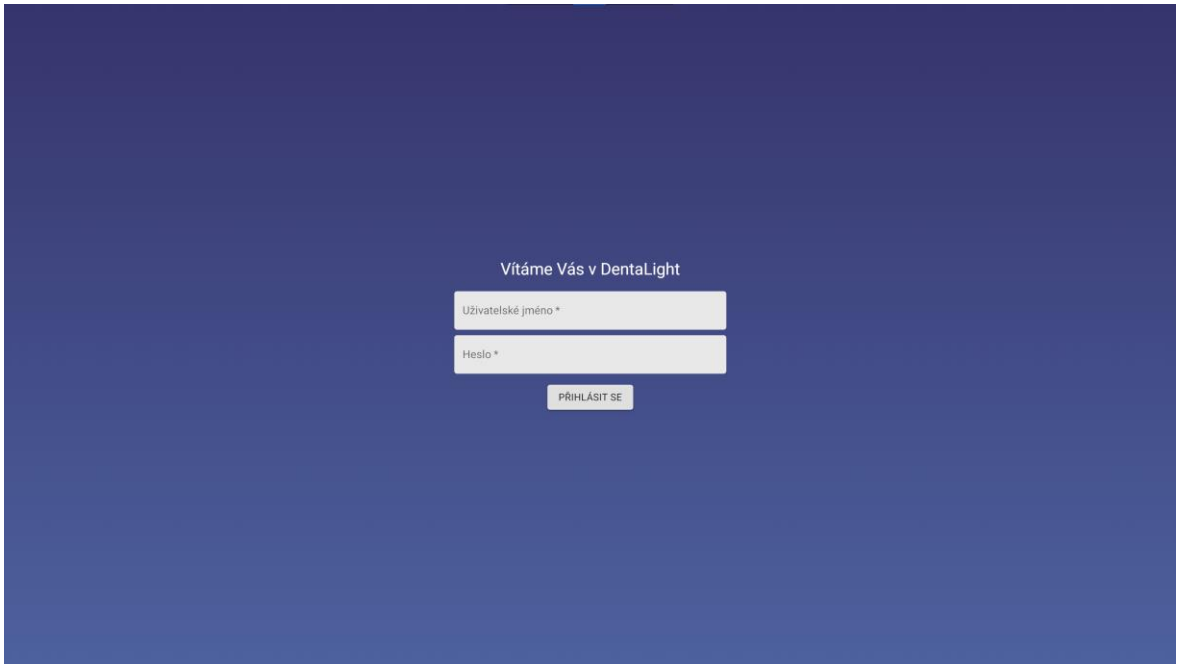
Po úspěšném požadavku na přihlášení server vrátí data aktuálního uživatele a jeho tokenu (který se poté použije pro další požadavky). Uživatelská data jsou uložena v místním úložišti (redux store)

```
   .then(({ token, user }) => {
    setCookie('token', token);
    dispatch({ type: 'login', payload: { user } });
   })
   .catch((e) => {
```

V případě neúspěšného požadavku je token odstraněn z cookies (pouze pro případ). deleteCookie('token');

```
    console.log(e);
   }); };
 return (
  <Container className={classes.container}>
   <LoginForm onSubmit={onSubmit} />
  </Container>  );};
```

**Obrázek 21 screenshot realizovaného rozhraní**




Zdroj: Vlastní zpracování

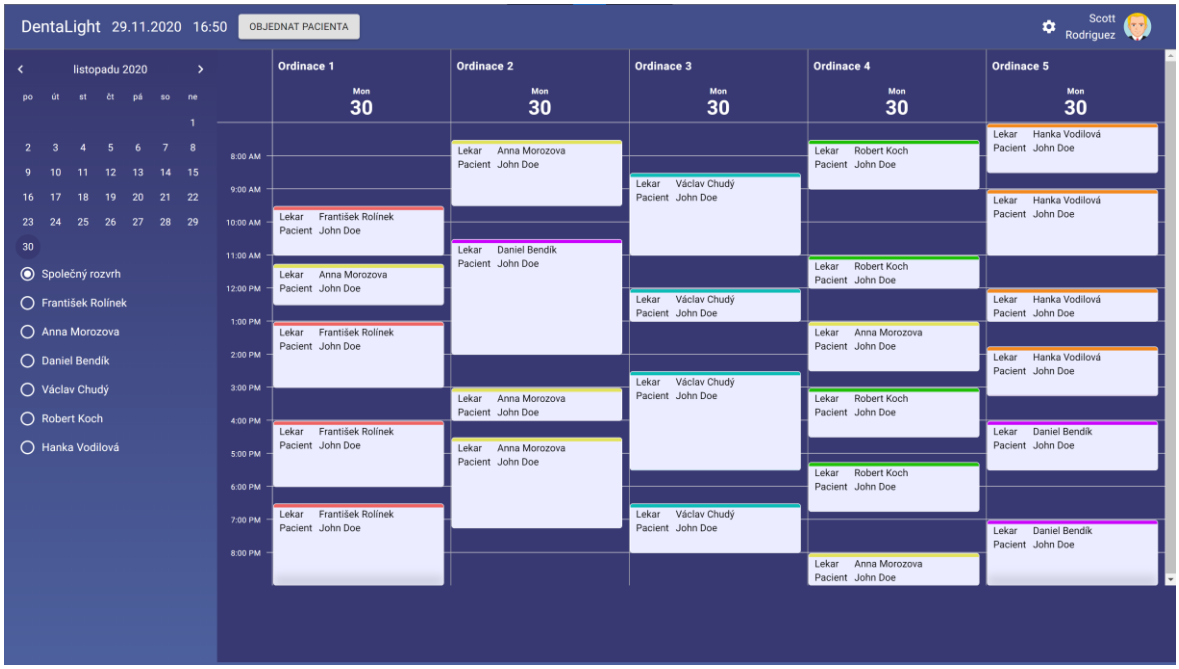
Script vykreslení plánu. Tato komponenta používá knihovnu dx-react-scheduler, která přebírá parametry pro zobrazení plánu pro konkrétní den.

```
 return (
  <Grid className={classes.container}>
   <Scheduler
    data={
     selectedPerson !== 'all'
      ? // vykreslit pouze vybrané schůzky lékaře
       data.filter((app) => `${app.doctorId}` === `${selectedPerson}`)
      : // nebo pokud je filtrování vypnuto, zobrazí se všichni lékaři
       data}
   >
    <ViewState
     defaultCurrentDate="2020-11-29"
     // zobrazí rozvrh vybraného dne
     currentDate={date.toISOString().substring(0, 10)}
    />
    {/** seskupení podle ordinace */}
    <GroupingState grouping={grouping} />

    {/** definice otevírací doby vybraného data stanovená ředitelem */}
    <DayView startDayHour={dayTime.from} endDayHour={dayTime.to} cellDuration={60}
/>
```

```
    {/** Vykreslení štítku*/}
    <Appointments appointmentContentComponent={Appointment} />
    {/** Vykreslení skupin podle ordinace */}
    <Resources data={resources} mainResourceName="cabinetId" />
    <IntegratedGrouping />
    <GroupingPanel />
  </Scheduler>
</Grid>);
```

**Obrázek 22 screenshot realizovaného rozhraní**



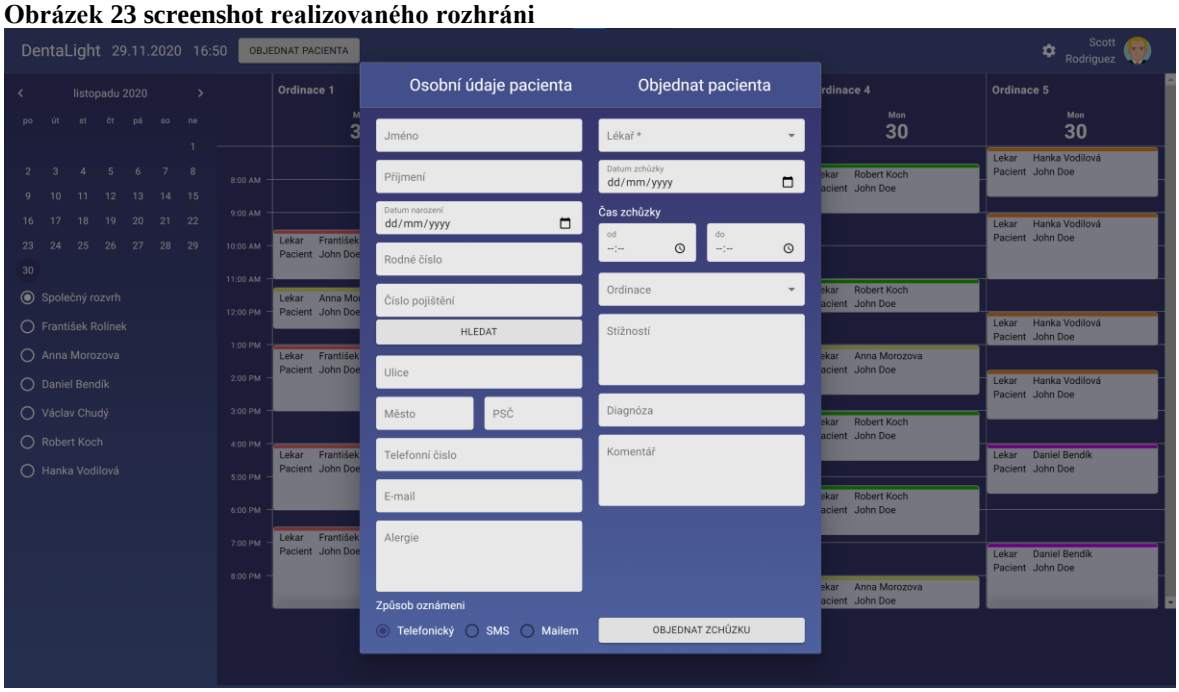

Zdroj: Vlastní zpracování

Další skript odpovídá za zpracovaní události kliknutí na tlačítko objednat pacienta. Tato funkce provádí požadavek POST na server s informacemi o pacientovi a schůzce.

```
const createAppointment = () => {
  const visitDate = appointment.visitDate?.split('-').reverse().join('-');
  const birthDate = client.birthDate?.split('-').reverse().join('-');
  const formattedAppointment = { ...appointment, visitDate };
  const formattedClient = { ...client, birthDate };
  if (!isExistClient) {
    delete formattedClient.id;}
  request({
    url: 'http:// localhost:3031/v1/appointment',
    method: 'POST',
    body: { appointment: formattedAppointment, client: formattedClient },
    headers: { Authorization: `Bearer ${token}` },
```

```
}).then(() => setOpen(false));};
```

**Obrázek 23 screenshot realizovaného rozhráni**




Zdroj: Vlastní zpracování

# 5 Výsledky a diskuse

Cílem diplomové práce bylo vytvořit webový informační systém pro pracovníky dentálního centra. Tento cíl byl splněn a informační systém funguje dle požadavků, které byly specifikované na začátku v části analýzy. Do tohoto informačního systému mají přístup pouze registrovaní pracovníky. Pro každého zaměstnance se v závislosti na jeho úrovni přístupu vytvoří uživatelské rozhraní s funkcemi nezbytnými pro práci.

Role uživatele, což teď je číselník, v budoucnu je vhodné ji nahradit tabulkou s atributy přístupu k jednotlivým systémům nebo funkcím, což by poskytlo flexibilitu s přístupem k datům.

Přestože aplikace splňuje požadavky, je zde příležitost pokračovat v jejím vývoji a přidávat nové funkce a moduly. Jedním z takových modulů může být poskytovaní hlášení o výkonu centra na základě existujících tabulek, pro které budou dostatečné schopnosti SQL. Kromě toho můžete také vytvořit mobilní aplikaci.

Při tvorbě této práce autor vycházel ze znalostí získaných v předmětech absolvovaných během studia a čerpáním informací ze zdrojů, uvedených na konci práce.

# 6 Závěr

V teoretické části této diplomové práce se čtenář seznámí s tím, co informační systém ve skutečnosti je. Dále se zde dočte o jeho vlastnostech a jeho hlavních typy. Následuje popis modelovacího jazyka UML, a demonstrace hlavních typy diagramů používaných při návrhu informačních systémů. Dále se čtenář seznámí s pojmem webová aplikace, s jejími výhodami, architekturou a technologiemi používanými k její implementaci.

V praktické části byl navržen a vytvořen webový informační systém. Tato tvorba byla rozložena na tří základní částí. S použitím metodologie a technologie uvedených v teoretické časti.

V první části se čtenář seznámil s analýzou přijatých systémových požadavků taky zde uvedený scénáře, které ukazují, jak by se měl systém chovat v interakci s uživatelem a jejich podrobnější zpracování pomocí diagramů aktivit. Výsledkem stal třídní diagram ukazující základní třídy a vztahy mezi nimi.

Ve druhé části – návrhu systému, po představení diagramu tříd a schématu relační databáze, byl sestaven datový slovník s podrobnějším popisem tabulek, za co jsou odpovědné a jaké datové typy byly použity. Na základě případů použití a diagramu tříd byl taky vytvořen návrh drátového modelu uživatelského rozhraní. A poté jeho graficky návrh.

Posledním krokem je již sestavován samotný systém. Čtenář seznámil s hlavními částmi zdrojového kódu popisujícími proces vytváření databáze, serverovou stránku aplikace a uživatelské rozhraní. Na konci práce uvedeny screenshoty webové aplikace.

Celá diplomová práce byla vypracována s využitím znalostí nabytých na České zemědělské univerzitě v Praze, v navazujícím oboru Systémové inženýrství a informatika a čerpáním informací ze zdrojů, uvedených na konci práce.

# 7 Seznam použitých zdrojů


ARLOW Jim, NEUSTADT Ila, 2005. *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design.* Addison-Wesley Professional; 2nd Edition. ISBN-13 : 978-0321321275

AZAD Adam, 2007. *Implementing Electronic Document and Record Management Systems.* Auerbach Publications, ISBN-13 : 978-0849380594

BUNOVA E.V., 2017. *Automatizace obchodních procesů společností v souladu s konceptem CRM. kolektivní monografie*. Moskva: Pero -- 134 s. [Elektronické vydání]. ISBN 978-5-00122-028-2

DUCKETT Jon, 2011. HTML and CSS: Design and Build Websites. John Wiley & Sons; 1st edition. ISBN-13: 978-1118008188

GIBB Robert, 2016. *What is a Web Application* [online]. USA Dostupné z: https://blog.stackpath.com/web-application

HERRON David, 2020. *Node.js Web Development: Server-side web development made easy with Node 14 using practical examples, 5th Edition.* Packt Publishing. ISBN-13: 978-1838987572

HILLS Ted, 2016. *NoSQL and SQL Data Modeling: Bringing Together Data, Semantics, and Software*. Technics Publications; First edition. ISBN-13: 978-1634621090

KAMAJDANOV Vladislav, 2014. *Architektura informačních systémů* [online]. Dostupné z: https://sites.google.com/site/arhitekturainformacionnyhsist/home

KINZYABULATOV Ramil, 2018. *CRM. Podrobně a na případu: Vydání 1.*: Publishing solutions. ISBN 978-5-4483-3293-7

LABARRE Olivia, 2020. *Enterprise Resource Planning (ERP)* [online]. USA [cit. 2020-06-07] Dostupné z: https://www.investopedia.com/terms/e/erp.asp

LIM Greg, 2019. *Beginning Node.js, Express & MongoDB Development.* Independently published. ISBN-13 : 978-1078379557

LUDIN Stephen, GARZA Javier, 2017. *Learning HTTP/2: A Practical Guide for Beginners.* O'Reilly Media; 1st edition. ISBN-13: 978-1491962442

MARTIN Shifa, 2019. *The Top JavaScript Frameworks For Front-End Development in 2020* [online]. Mumbai. Dostupné z: https://www.freecodecamp.org/news/author/shifa

MILES Russ, 2006. *Learning UML 2.0: A Pragmatic Introduction to UML*. O'Reilly Media; 1st Edition. ISBN-13: 978-0596009823

MINDALEV Igor, 2015. *Elektronický vzdělávací a metodický komplex pro obor Business Informatics*. Dostupné z: http://enisey.name/umk/mbp/index.html

OWASP, 2020. *OWASP Top Ten* [online]. Dostupné z: https://owasp.org/www-project-top-ten

PUREWAL Semmy, 2014. *Learning Web App Development: Build Quickly with Proven JavaScript Techniques.* O'Reilly Media; 1st edition. ISBN-13: 978-1449370190

ROBBINS Jennifer, 2012. *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics.* O'Reilly Media; Fourth edition. ISBN-13: 978-1449319274

SCHONIG Hans-Jurgen, 2018. *Mastering PostgreSQL 11: Expert techniques to build scalable, reliable, and fault-tolerant database applications, 2nd Edition.* Packt Publishing. ISBN-13: 978-1789537819

SHKLAR Leon, 2009. *Web Application Architecture: Principles, Protocols and Practices 2nd Edition*. Wiley; 2nd edition. ISBN-13: 978-0470518601

TITORENKO G.A., 2003. *Automatizované informační technologie v ekonomii: učebnice. pro univerzity*. M: UNITI, -- 399 s. - ISBN5-238-00040-5.