Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Webová aplikace pro administraci zdravotní dokumentace

Richard Tichý

© 2024 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Richard Tichý

Informatika

Název práce

Webová aplikace pro administraci zdravotní dokumentace

Název anglicky

Web application for the administration of health records

Cíle práce

Cílem teoretické části práce bude analyzovat existující aplikaci pro administraci pacientů a následně vysvětlit jednotlivé výhody a nevýhody existující aplikace. Dílčím cílem bude analyzovat tvorbu webových aplikací a zhodnotit jejich výhody a nevýhody. Dále bude cílem analyzovat vývojové technologie ASP.NET a PHP Symfony.

Cílem praktické části bakalářské práce bude navrhnout a implementovat webovou aplikaci pro správu a dokumentaci potřebných údajů pacientů do online databáze. Aplikace bude primárně určená pro praktické lékaře, bude co nejpřehlednější a současně bude poskytovat veškeré možné informace, které praktický lékař potřebuje ke své práci. V aplikaci bude umožněno přidání dat, editování pacientů a jejich filtrace. Aplikace bude následně testována praktickým lékařem.

Metodika

V úvodu bakalářské práce bude zanalyzována původní aplikace pro administraci pacientů a následně bude porovnána s možnostmi nových funkcí webové aplikace. Aplikace bude tvořena podle vodopádového modelu a pomocí technologie ASP.NET a propojena s online databází. Následně bude webová aplikace testována praktickým lékařem. Závěrem bude vytvořena stručná dokumentace aplikace.

Doporučený rozsah práce

30–60 stran

Klíčová slova

webová aplikace, vývoj aplikace, zdravotní dokumentace, praktičtí lékaři, ASP.NET

Doporučené zdroje informací

BELLINASO, Marco. Webové programování v ASP.NET 2.0: problém, návrh, řešení. Brno: Computer Press, 2007. ISBN 978-80-251-1893-1.

ČERNÝ, Jaroslav. ASP.NET a ADO.NET: tvorba dynamických webových stránek. Praha: Grada, 2003. Moderní programování. ISBN 80-247-0474-9.

LAURENČÍK, Marek. SQL: Podrobný průvodce uživatele. Praha: Grada, 2018. ISBN 978-80-271-2154-0.

Předběžný termín obhajoby

2023/24 LS – PEF

Vedoucí práce

Ing. Dana Vynikarová, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 4. 9. 2023

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 3. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 07. 02. 2024

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Webová aplikace pro administraci zdravotní dokumentace“ jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne datum odevzdání _____

Poděkování

Rád bych touto cestou poděkoval Ing. Daně Vynikarové, Ph.D. za konzultace, které mi velmi pomohly k napsání této práce. Dále bych rád poděkoval Tomáši Stejskalovi za velmi cenné konzultace ohledně zjednodušení a optimalizace kódu webové aplikace. Nakonec bych rád poděkoval paní doktorce MUDr. Ivě Tiché za pomoc při pochopení lékařských aplikací, které se používají při výkonu práce.

Webová aplikace pro administraci zdravotní dokumentace

Abstrakt

Tato bakalářská práce se zabývá vývojem prototypu webové aplikace pro lékaře. Prototyp je vyvíjen za účelem dalšího vývoje a možnosti uvést celkový program k vyzkoušení u více lékařů. Teoretická část ukazuje podrobný popis a zanalyzování vybrané lékaři předně používané pevné aplikace. Dále je v teoretické části řešen popis a analýza dostupných technologií pro vývoj webových aplikací. Praktická část se nejdříve zabývá přípravou a plánováním, jakým způsobem bude vyvíjena webová aplikace s databází, co vše bude obsahovat databáze a jaké budou vazby mezi tabulkami. Dále praktická část popisuje propojení databáze s vývojovým prostředím a následný vývoj webové aplikace v jazyce C#. V závěru se bakalářská práce zabývá hodnocením lékaře po vyzkoušení webové aplikace a následnému napsání dokumentace webové aplikace. Vyvíjený prototyp webové aplikace je ve fázi, kdy je možné se přihlašovat do aplikace přes uložené přihlašovací údaje. Po přihlášení do aplikace webová aplikace zobrazí seznam karet pacientů. Karty pacientů uživatel může vytvořit či editovat. Součástí karet pacientů jsou také funkční stránky pro napsání dekursu, anamnézy a očkování. Očkování má také možnost zobrazování, editace, historie očkování vakcín pacienta.

Klíčová slova: webová aplikace, vývoj aplikace, zdravotní dokumentace, praktičtí lékaři, ASP.NET

Web application for the administration of health records

Abstract

This bachelor's thesis deals with the development of a web application prototype for doctors. The prototype is being developed for the purpose of further development and the possibility of introducing the entire program for testing by more doctors. The theoretical part shows a detailed description and analysis of selected applications primarily used by doctors. Furthermore, the theoretical part addresses the description and analysis of available technologies for web application development. The practical part first deals with the preparation and planning of how the web application with a database will be developed, what the database will contain, and what the relationships between the tables will be. The practical part also describes the connection of the database with the development environment and the subsequent development of the web application in the C# language. In conclusion, the bachelor's thesis deals with the evaluation of the doctor after testing the web application and subsequent writing of the web application documentation. The developed prototype of the web application is at the stage where it is possible to log in to the application through stored login details. After logging into the application, the web application displays a list of patient cards. The user can create or edit patient cards. The patient cards also include functional pages for writing progress notes, anamnesis, and vaccinations. Vaccinations also have the option of displaying, editing, and the history of the patient's vaccine vaccinations.

Keywords: web application, application development, medical records, general practitioners, ASP.NET

Obsah

1 Úvod

Praktičtí lékaři využívají k výkonu práce pevné aplikace, které používají lokální databázi ve formě textového souboru. Po několika letech používání se textový soubor rozroste do několika gigabytů. Dodnes téměř nikdo nenabízí webovou aplikaci praktickým lékařům. Většina vývojáři se drží pevných aplikací. Na konci každého pracovního dne si praktičtí lékaři kvůli pevným aplikacím a textovému souboru s databází na disku musí sami zálohovat data na flash disky. V dnešní době musí každý praktický lékař střídat minimálně dva flash disky, na kterých jsou uloženy zálohy. Tyto pevné aplikace jsou také licencované pouze na jeden rok. Každá licence je uplatnitelná jen na jedno zařízení, což je většinou stolní počítač v ordinaci. Pokud si lékař neplatí více licencí, tak má-li výjezd do terénu, např. k pacientovi domů, nemá možnost zapsat si veškeré potřebné informace do programu a teprve po návratu do ordinace musí tyto informace přepisovat z poznámek na papíře nebo v mobilním telefonu. Stejně tak u pacienta na výjezdu lékař nemůže vytvořit eRecept nebo vystavit pracovní neschopnost.

Webová aplikace umožňuje jak přístup do programu přes internet, ale má i podporu pro telefonní zařízení. Díky tomu je přístupná odkudkoliv, čímž se lékařům zjednodušuje práce. Lékaři by tak mohli na výjezdech rovnou zapisovat informace do karet pacientů, ověřit on-line stav pojištění, zkontrolovat užívané léky, vydávat recepty nebo neschopenky, podívat se do předchorobí, které nemoci a jaká vyšetření již pacient absolvoval. Dále by se lékaři nemuseli starat o zálohu databáze na konci každého pracovního dne, jelikož zálohy by probíhali automaticky na serverech, kde by webová aplikace byla zprovozněná a nastavená na automatické zálohy. V této bakalářské práci je vytvořen prototyp této webové aplikace.

V bakalářské práci se teoretická část zabývá základním popisem dostupných pevných aplikací a následné zanalyzováním jedné z nejpoužívanějších pevných aplikací mezi praktickými lékaři. Praktická část se zabývá přípravou a návrhem webové aplikace a databáze. Po návrhu se praktická část zaobírá vývojem databáze a přípravou vývojového prostředí pro webovou aplikaci. Po praktické části následuje kapitola výsledků, která se věnuje vyjádření praktického lékaře a sepsání dokumentace pro správné použití webové aplikace.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem teoretické části práce je analyzovat existující aplikaci pro administraci pacientů a následné vysvětlení jednotlivých výhod a nevýhod existující aplikace. Dílčím cílem je analyzovat tvorbu webových aplikací a zhodnotit jejich výhody a nevýhody. Dále je cílem analyzovat vývojové technologie ASP.NET a PHP Symfony.

Cílem praktické části bakalářské práce je navržení a implementace webové aplikace pro správu a dokumentaci potřebných údajů pacientů do online databáze. Aplikace je primárně určená pro praktické lékaře, tímto je program co nejpřehlednější a současně poskytuje veškeré možné informace, které praktický lékař potřebuje ke své práci. V aplikaci je umožněno přidání dat, editování pacientů a jejich filtrace. Aplikace je následně testována praktickým lékařem.

2.2 Metodika

V úvodu bakalářské práce bude zanalyzována původní aplikace pro administraci pacientů a následně bude porovnána s možnostmi nových funkcí webové aplikace. Aplikace bude tvořena podle vodopádového modelu a pomocí technologie ASP.NET a propojena s online databází. Následně bude webová aplikace testována praktickým lékařem. Závěrem bude vytvořena stručná dokumentace aplikace.

3 Teoretická východiska

3.1 Existující aplikace

V této práci je vycházeno z aktuálně nejpoužívanějších programů pro praktické lékaře, které jsou velmi obsáhlé a komplikované. Každý program designově odlišný, má jiné funkce a vzájemně spolu nejsou kompatibilní. Ne všechny funkce jsou praktickými lékaři využívány a často se v nich obtížně orientují. K nejznámějším a nejpoužívanějším programům patří Smart Medix, Medicus a mnou analyzovaný Duas Altera. Všechny tyto programy pracují na bázi pevných aplikací instalovaných na pracovních počítačích.

3.1.1 SmartMedix

SmartMedix je jedna z nejpoužívanějších aplikací pro praktické lékaře. Tato aplikace je hlavním produktem společnosti Medax Systems. Aplikace je vyvíjena od roku 2008 až do dnešního dne. Tato aplikace je jako jedna z mála vyvinuta pro jednotlivé lékařské obory. Je založena na modulárním informačním systému, každý uživatel si zaplatí a využívá jen potřebné moduly. Díky tomu má praktický lékař přehledné grafické rozhraní. SmartMEDIX zahrnuje všechny běžné funkcionality zdravotnického systému jako je vedení zdravotnické dokumentace, vystavování žádanek, receptů, posudků, vykazování péče na zdravotní pojišťovny, objednávání apod. SmartMEDIX navíc disponuje pokročilými funkcemi pro pořizování dokumentace prostřednictvím uživatelsky nebo dodavatelsky připravených formulářů. (1)

Tabulka 1 Ceník SmartMedix aplikace (2)

	Lékař	Sestra	Celkem
Licence	16 500 Kč	6 600 Kč	23 100 Kč
Podpora	8 575 Kč	4 287 Kč	12 862 Kč
Celkem	25 075 Kč	10 887 Kč	35 962 Kč
1. rok	35 962 Kč		
Další léta	12 862 Kč		

3.1.2 **PC Doktor**

PC DOKTOR vyvinutý společnosti CGM CompuGroup Medical je komplexní informační software určený pro všechny typy ambulancí a menší zdravotnická zařízení. Obsahuje veškeré manažerské přehledy související s chodem ordinace, včetně správy financí.
Je využíván při vedení zdravotnické dokumentace, pomáhá poskytovat kvalitní lékařskou péči o pacienty a zajišťuje bezpečnou elektronickou komunikaci. (3)

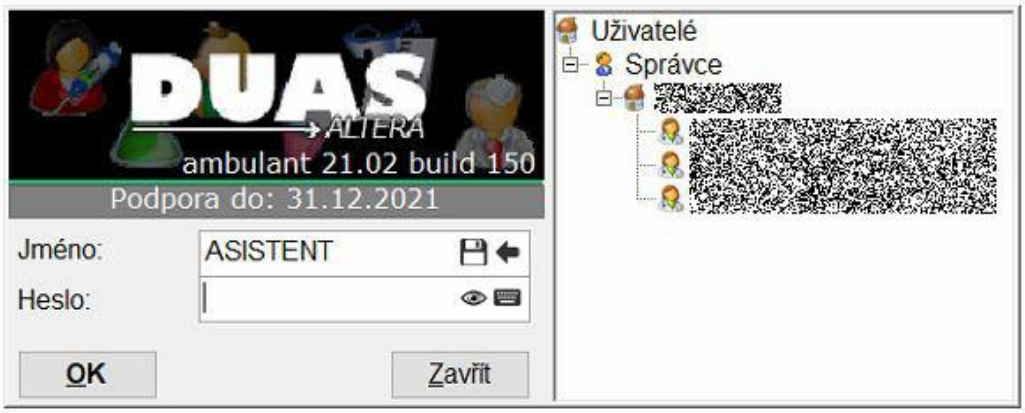
3.2 **Duas Altera – Ambulant**

DuasAltera – Ambulant je jedním z novějších programů, které praktičtí lékaři používají pro administrativu pacientů. Zatím není mezi lékaři tolik rozšířen. Program byl vyvíjen firmou SWS Kolín jako pevný klient ve spolupráci s lékaři tak, aby co nejlépe vyhovoval jejich potřebám, je také modulární. Do programu byly implementovány zkušenosti s používáním již rozšířených programů. Konzultace vývojářů s lékaři je pro vývoj programu klíčová, každý lékař může navrhnout zlepšení či úpravu za účelem zefektivnění rychlosti či opravení malých chyb. Program lze použít pro všechna lékařská odvětví a všechny pozice. Lékař si za licenci pro používání programu musí zaplatit jednou ročně. (4)

3.2.1 **Přihlašování do programu**

Při spuštění programu je vyžadováno přihlášení. Na pravé straně přihlašovacího okna je zobrazena stromová hierarchie uživatelů, kteří jsou pro danou licenci na daném zařízení registrováni do systému programu. Při přihlášení se musí zadat uživatelské jméno, které si lékař při registraci zvolí a potvrdí. Heslo se zadává nepovinně. Pokud lékař pracuje na daném počítači sám a nepřeje si zadávat heslo, tak při registraci heslo nevyplní. Lékař může do programu zaregistrovat i zdravotní sestru, která také vyplní registrační formulář. (5)

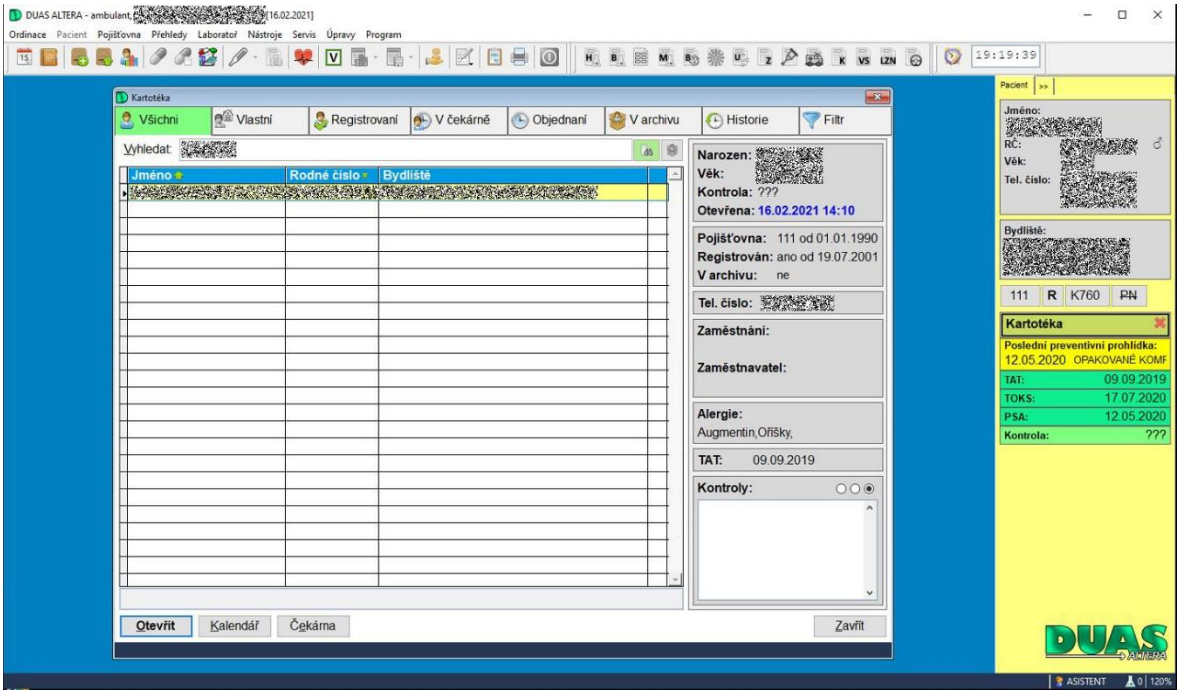
Po přihlášení do programu má každý uživatel stejné možnosti. Zdravotní sestra má přístup k celému programu a do všech nástrojů, do některých může jen nahlížet. V programu není žádný filtrový či přístupový rozdíl. Sestra však sama nemůže odesílat např. eRecept, protože nemá kvalifikovaný certifikát. (5)



Obrázek 1 Přihlašování do aplikace (6)

3.2.2 **Seznam pacientů**

Seznam pacientů se zobrazí po úspěšném přihlášení uživatele do programu. Seznam pacientů je zasazen v podokně hlavního okna programu. Tímto způsobem se vytváří v programu dojem internetového prohlížeče. Ve výpisovém seznamu je možnost filtrace podle rodného čísla nebo podle jména. Zobrazovací podokno programu, kde se zobrazují veškeré informace o pacientech je navrženo tak, aby byla zobrazovaná data pro uživatele graficky co nejpřehlednější. V hlavním výpisovém seznamu je zobrazeno příjmení, jméno, rodné číslo, zdravotní pojišťovna, poznámka o registraci pacienta. Při výběru konkrétního pacienta v seznamu jsou v pravé části obrazovky zobrazeny podrobnější informace o pacientovi. Nejdůležitější informací pro lékaře v podrobném rychlém výpisu v pravé části je kolonka, v níž jsou zapsány informace o aktuálním zdravotním stavu pacienta při poslední provedené prohlídce. V pravé části se zobrazuje také rychlý souhrn informací jako je krevní tlak, očkování proti tetanu, poslední preventivní prohlídka včetně screeningových onkologických vyšetření jako je kolonoskopie, mamografie a preventivní vyšetření stolice na okultní krvácení. (5)

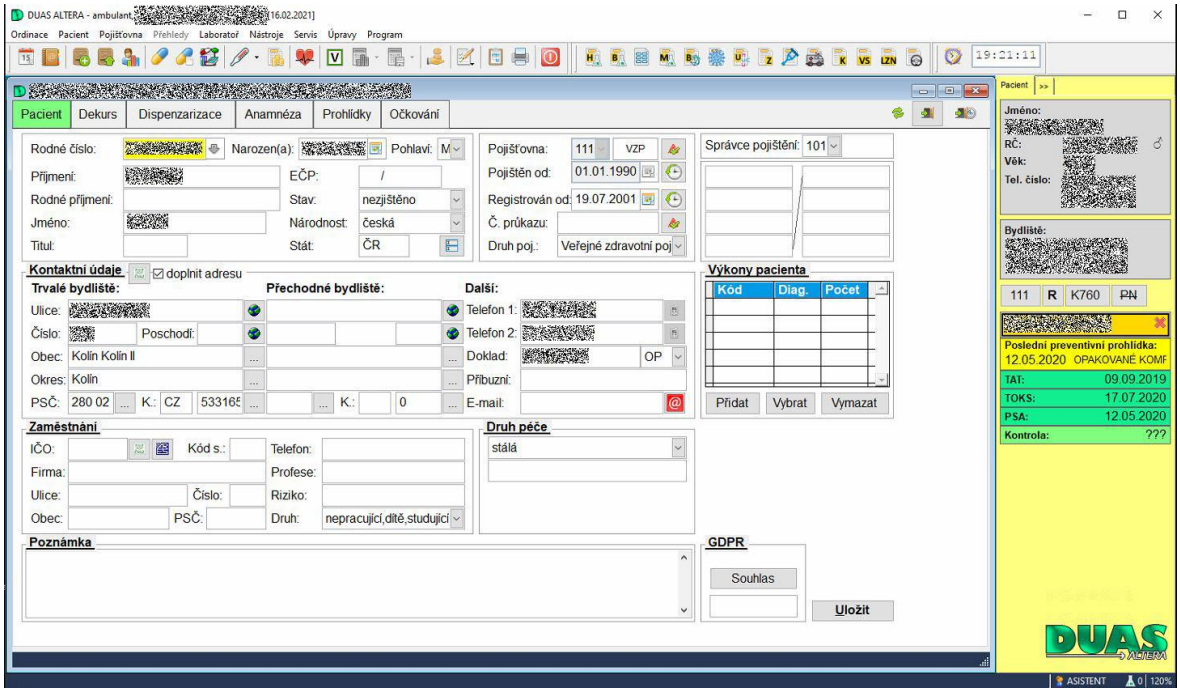


Obrázek 2 Seznam pacientů (7)

3.2.3 **Karta pacienta**

Kartu pacienta uživatel otevře skrze seznam pacientů. V seznamu pacientů si uživatel vybere konkrétního pacienta a dvojklikem se mu otevře nové okno, kde budou zobrazeny veškeré informace o pacientovi. V okně pacienta jsou různé záložky. Do záložky pacienta se zapisují osobní informace o pacientovi, které jsou rozděleny do bloků základních informací. V prvním bloku jsou k vyplnění základní data pacienta – jméno, příjmení, trvalé bydliště, kontaktní adresa, telefonní číslo. Lze zadat i rodné příjmení, které je důležité pro udržení kontinuity po změně jména např. po sňatku. Dalším blokem je druh péče a nyní nově, zda je pacient například uprchlíkem či nikoli. Druh péče udává, zda je pacient u lékaře registrován na stálo nebo jde jen na jednorázové akutní ošetření. Dalším velkým a důležitým blokem jsou kontaktní údaje. V bloku kontaktních údajů jsou pod bloky trvalé bydliště a kontaktní adresa. Zadává se ulice a číslo popisné, obec, okres a poštovní směrovací číslo. Kontaktní adresa je důležitá pro případ, že pacient nebydlí v místě trvalého bydliště a je třeba se s ním korespondenčně spojit (například zaslat nálezy, laboratorní výsledky apod.) Posledním podoknem v této sekci je podokno další. Lékař do této sekce zapisuje jedno nebo více telefonních čísel pacienta, jeho email a také příbuzné osoby. Další sekcí je registrující lékař. Tento údaj je pro praktické lékaře důležitý, musí být k dispozici. Je-li registrováno v rámci jedné licence na stejném zařízení více uživatelů, všichni vidí

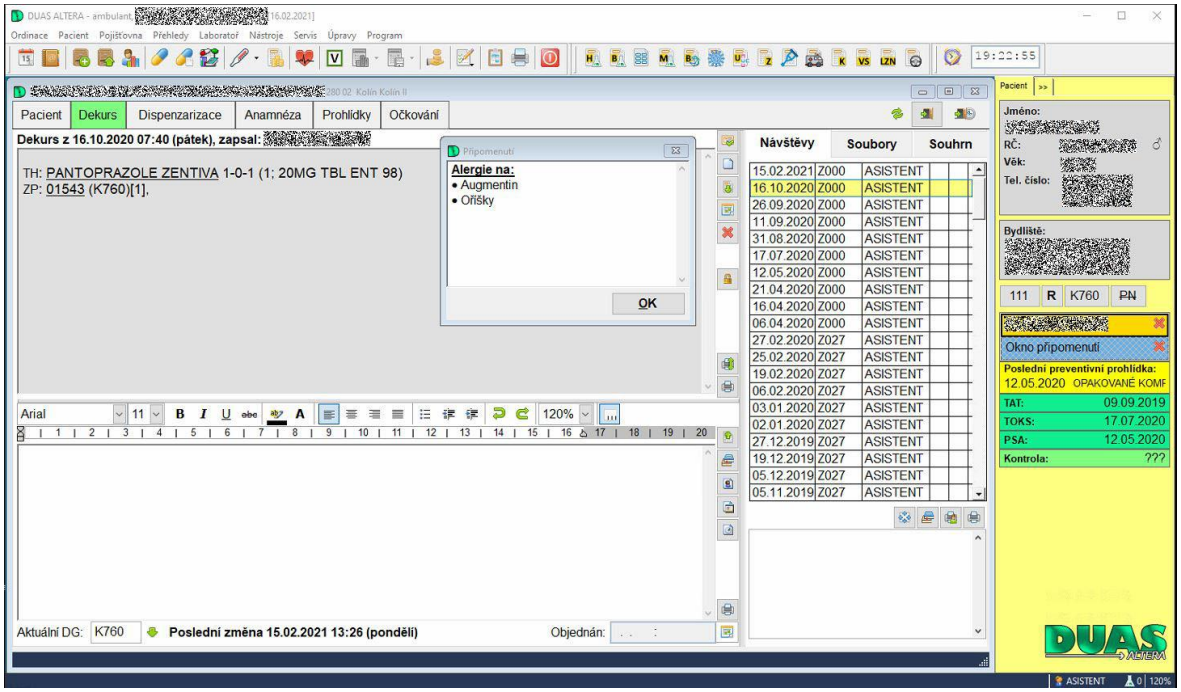
do záznamů všech a je snazší mít přehled, kdo a ve kterém časovém období pacienta ošetřoval. V této sekci se vyplňují i údaje o lékaři – lékařské osobní číslo, adresa, telefon a email. V dolní části karty pacienta jsou další navigační listy. První z nich je list pro poznámky. V tomto listu lze napsat dva druhy poznámek. První je možnost sdílené poznámky, kterou vidí všichni lékaři, co jsou registrování na stejné licenci anebo je také možnost vepsání do soukromé poznámky, kde poznámku vidí jen lékař, který ji napsal. Další důležitým listem je informovaný souhlas, který se vyplňuje v případě, pokud pacient není plnoletý nebo byl zbaven svéprávnosti. List zaměstnání je pro lékaře důležitý při vystavení pracovní neschopnosti anebo kvůli posouzení zdravotního stavu v souvislosti se zaměstnáním. List škola je také důležitý z již zmíněných důvodů. (5)



Obrázek 3 Karta pacienta (8)

3.2.4 **Dekurz v pacientské kartě**

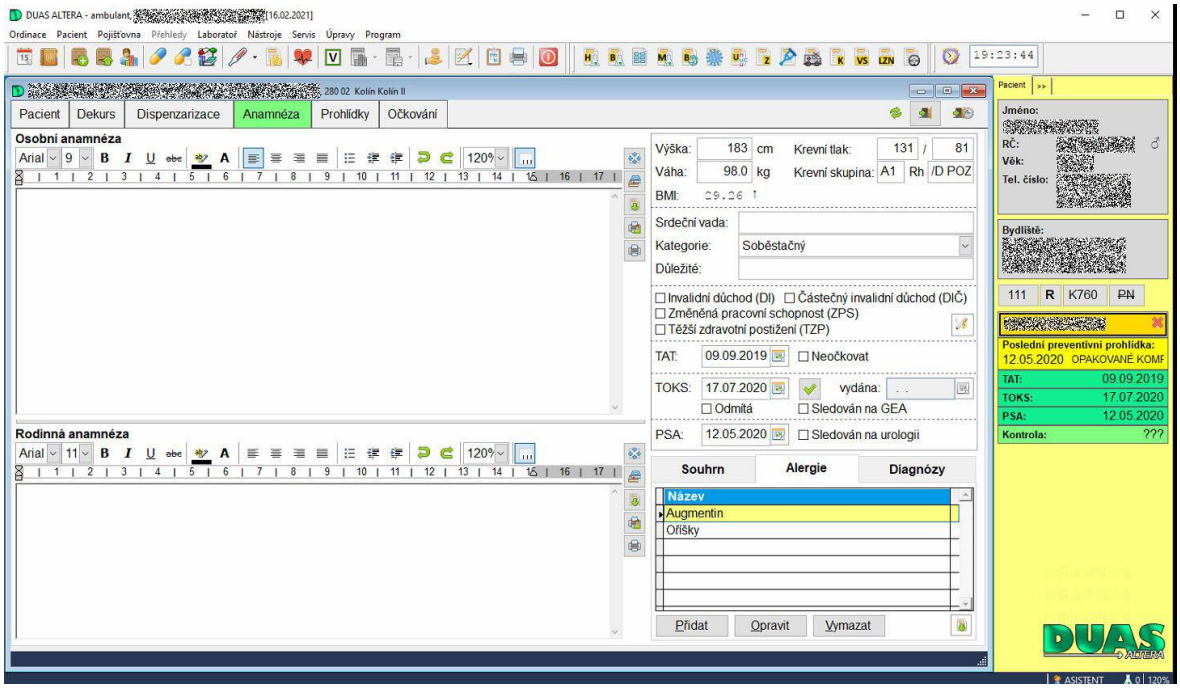
Dekurz lékaři používají na popsání aktuálního zdravotního stavu pacienta v den lékařské prohlídky. V dekurzu jsou viditelná dvě okna. V horní části má lékař zapsán stav pacienta z minulého lékařského vyšetření. V dolní části dekurzu lékař zapisuje nový aktuální stav pacienta z aktuálního lékařského vyšetření. V pravé dolní části je možnost zobrazení data příštího vyšetření. V pravé části obrazovky je viditelný souhrn všech minulých návštěv pacienta v ordinaci, které je možné detailně rozkliknout. (5)



Obrázek 4 Dekurs v pacientské kartě (9)

3.2.5 **Anamnéza v pacientské kartě**

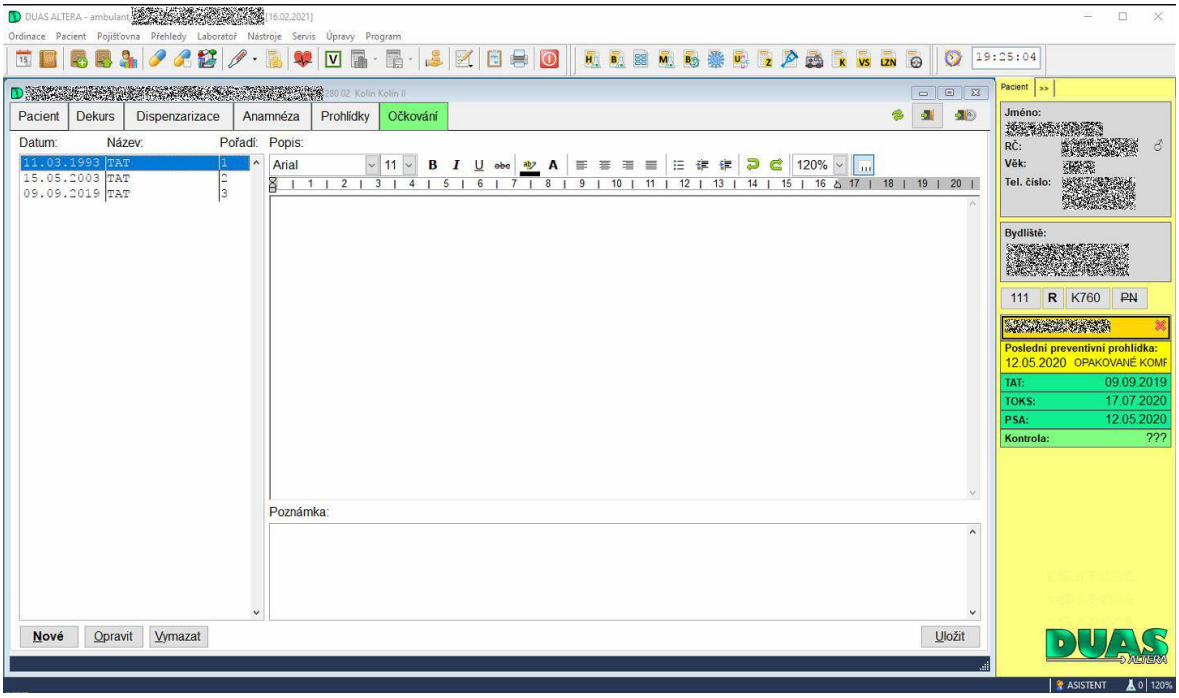
Anamnéza v kartě pacienta popisuje veškerá onemocnění, která pacient za svůj život prodělal. Anamnéza se dělí na rodinnou a osobní. Do horního okna se zapisuje anamnéza osobní, kde lékař zapisuje nebo má zapsány veškeré pacientovy nemoci, úrazy nebo operace. Do dolního okna se zapisuje rodinná anamnéza. Tento údaj má lékař k dispozici, aby znal detaily ohledně těžkých nemocí či vážných zdravotních problémů v rodině. Tento údaj pomáhá lékaři ohlídat projevy těžkých či vážných rodinných onemocnění, včetně srdečních vad. Zadává se zde i kategorie pacienta, která se určuje podle soběstačnosti či stupně závislosti pacienta na druhé osobě. Posledním údajem je, zda pacient pobírá invalidní důchod či má vystavenou pracovní neschopnost. (5)



Obrázek 5 Anamnéza v pacientské kartě (10)

3.2.6 **Očkování v kartě pacienta**

Očkování v kartě pacienta slouží pro evidenci všech aplikovaných vakcín, které byly pacientovi za celou historii aplikovány. Pokud chce lékař založit nový záznam, zadá popis vakcíny. Do popisu se napíše název vakcíny, číslo šarže (výrobní číslo), podané množství, datum expirace (datum použitelnosti) a množství látky aplikované pacientovi. Dalším údajem je pořadí dávky, zda se jedná o novou vakcinaci nebo přeočkování Při přeočkování se eviduje pořadí dávky. Do cesty podání se vyplní, jakým způsobem byla vakcína podána, zda do svalu nebo podkožně. Posledním údajem je místo podání a do tohoto údaje se zapisuje, kam byla injekce aplikována. (5)

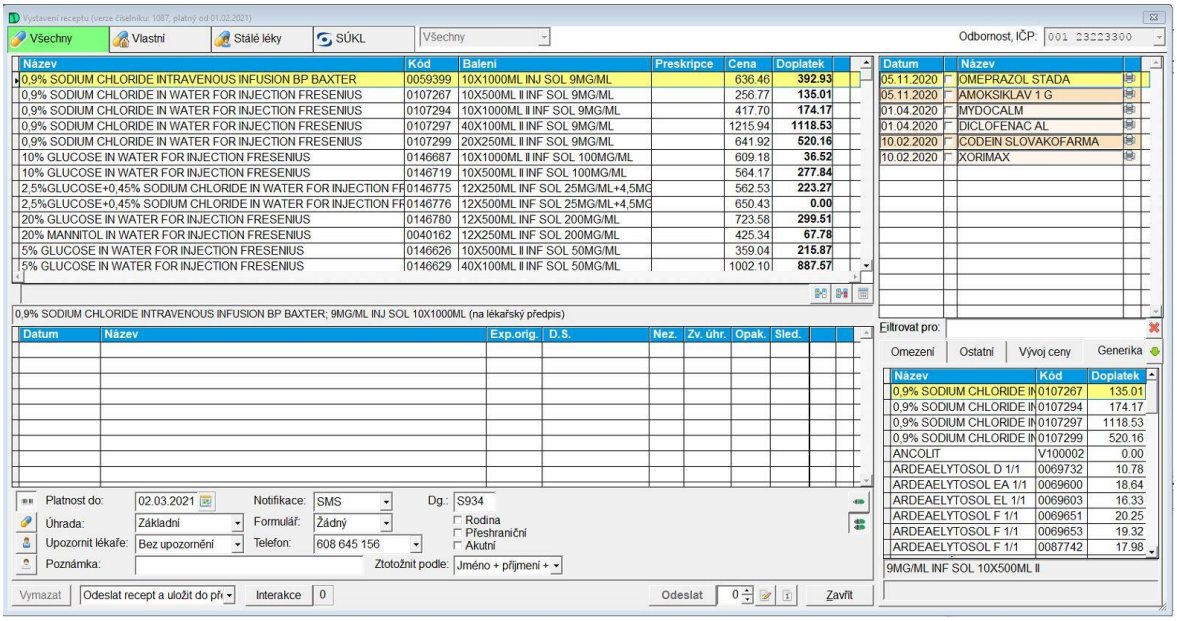


Obrázek 6 Očkování v pacientké kartě (11)

3.2.7 **Vystavení receptu**

Vystavení receptu (předpisu na léky) se dá zobrazit prostřednictvím modro–žluté ikonky tablety. Při otevření vystavení receptu se zobrazí tři tabulky. V horní tabulce se z databáze všech léků v systému označí vybraný lék, který lékař chce vystavit pacientovi na předpis. Ten se přepíše do spodní tabulky, kde lékař doplní počet balení a dávkování. Nakonec lékař vybere, jakou cestou pacientovi recept doručí, zda emailem, SMS či v papírové formě do ruky a případně upraví dobu platnosti receptu. V třetí tabulce se ukazuje historie všech předepsaných léků od lékaře pacientovi – název léku a datum předpisu. V tomto přehledu se zobrazuje informace, kdy byl lék vystaven pacientovi a také název léku. Rozkliknutím tlačítka s tabletou vpravo dole se zobrazí lékový záznam.

V lékovém záznamu se zobrazují všechny léky předepsané pacientovi u všech lékařů za posledních 12 měsíců. Tím je možné ověřit, zda pacient neužívá některé léky duplicitně, popřípadě identifikovat léky, jejichž název si pacient nepamatuje. (5)



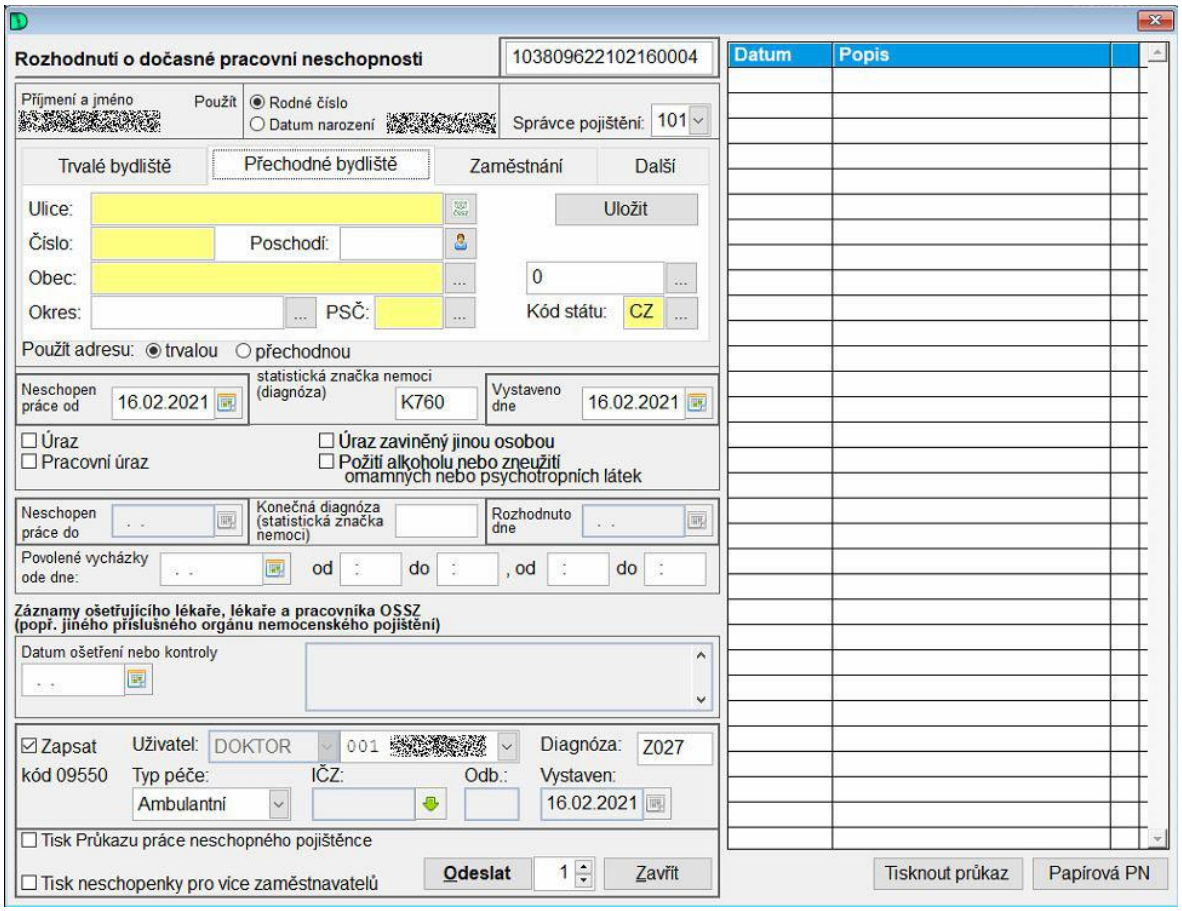
Obrázek 7 Vystavení receptu (12)

3.2.8 **Rozhodnutí o dočastné pracovní neschopnosti**

Okno dočasné pracovní neschopnosti se otevírá, chce-li lékař vystavit pracovní neschopnost. Toto okno se skládá ze dvou bloků informací. V první části se zadává místo pobytu nemocného. Z předem zadaných údajů se vybere buď trvalé bydliště nebo kontaktní adresa. V druhé části se zadávají informace o zaměstnavateli. Práci dnes usnadňuje

propojení s Českou správou sociálního zabezpečení, z jejíž databáze má lékař možnost informace o zaměstnavateli stáhnout. Pokud se informace neobjeví, lze je zadat i manuálně.

Nakonec je nutné zadat datum počátku pracovní neschopnosti, číselný kód diagnózy a datum příští kontroly. Pracovní neschopnost je pak odeslána elektronicky jak zaměstnavateli, tak na Správu sociálního zabezpečení. (5)



Obrázek 8 Vystavení neschopenky (13)

3.3 Závěr analýzy programu

Program je vyvíjen zkušenými programátory. Velkou výhodou programu je zjednodušení administrace pacientů. Pro lékaře je digitální forma administrace pacientů mnohem rychlejší než vyplňovat zdravotní karty ručně nebo na psacím stroji. Tento program je velmi nápomocný také pro zjednodušení práce zdravotních sester, které nemusí většinu své pracovní doby hledat pro lékaře papírové zdravotní karty pacientů, vyplňovat osobní údaje do žádanek na vyšetření a na odběry krve.

Hlavní nevýhodou programu je, že se za program platí roční licenční poplatek, který je dost drahý. Po jeho koupi se licence aktivuje na jednom zařízení, kde se potom může stáhnout program. U praktických lékařů se tato licence uplatní obvykle na stolním počítači, který mají stabilně v ordinaci. Pokud musí lékař opustit ordinaci a odjet za pacientem na výjezd do jeho bydliště, tak si s sebou nemůže vzít notebook nebo jiné zařízení. Licence je jednorázová a uplatnitelná jen na jedno zařízení. Za každé další připojené zařízení se musí připlácet. Toto praktické lékaře velmi omezuje. Z výše uvedených důvodů je výhodnější vyvinout webovou aplikaci, která je flexibilnější a mnohé nevýhody pevných aplikací eliminuje.

3.4 Dostupné technologie pro vývoj webové aplikace

Vybrané technologie od Microsoftu např. ASP.NET mají pro vývoj webových aplikací řadu výhod. Tyto frameworky jsou open-source, díky tomu si vývojáři mohou modifikovat frameworky podle své potřeby. Frameworky mají dlouhodobou podporu a hlavní výhodou je možnost vývoje aplikací pro různé operační systémy. Frameworky podporují také vývoj a funkčnost webových aplikací na různých prohlížečích. Díky této výhodě uživatelé mohou používat operační systémy s prohlížeči dle jejich vlastního výběru, aniž by se museli starat o nekompatibilitu webových aplikací. Dalším vybraným frameworkem je PHP Symfony, který je vybrán především kvůli optimalizaci pro vysoký výkon, díky čemuž jsou aplikace efektivně rychlé a spolehlivé. Tento framework má také podporu pro integraci jiných knihoven a frameworků, což zvyšuje rozšiřitelnost webových aplikací.

3.4.1 **ASP.NET**

„*ASP.NET je součástí tradičního .NET Frameworku. Framework ASP.NET přinesl několik konceptů tvorby webových aplikací. Mezi nejznámější koncepty tvorby webových stránek patří ASP.NET WebForms, ASP.NET MVC a ASP.NET Web Pages. Protože je ASP.NET součástí .NET Frameworku, jeho budoucnost je s .NET Frameworkem spjatá a odkázaná na pouhé bezpečnostní záplaty*“. (14) ASP.NET podporuje vázání dat. *„Vázání dat je proces načítání dat z datového zdroje a jejich dynamické přiřazování určité vlastnosti vizuálního prvku uživatelského rozhraní“*. (15) *„ASP.NET neběží na počítači uživatele, ale dynamicky generuje prvky, které používá prohlížeč pro vykreslování webových stránek“*. (16)

3.4.2 **.NET Framework**

NET Framework je spravované prostředí pro spouštění operačního systému Windows, který poskytuje služby pro běžící aplikace. Framework se skládá ze dvou hlavních součástí. První součástí je spouštěcí modul CLR, který zpracovává veškeré spuštěné aplikace. (14) Druhou hlavní součástí je .NET framework, který poskytuje knihovny otestovaných a znovu použitých kódů, které mohou vývojáři vyvolat ve svých vlastních aplikacích.
Framework poskytuje pro spuštění aplikací více služeb. Hlavní službou je například správa paměti, kterou poskytuje CLR modul.

Vývojové architektury a technologie jsou obsaženy ve Frameworku ve formě různých tříd pro oblasti vývoje webových aplikací nebo také pro správu dat.

Vzájemné funkční spolupráce jazyka, což jsou kompilátory jazyka, které se zaměřují na .NET Framework a následně vydají mezikód neboli CIL, který je kompilován CLR modulem. Po kompilaci modulem jsou kódy dostupné v ostatních jazycích, ve kterých programátoři pracují na vytváření aplikací. (17)

„*Poslední hlavní službou je souběžné spuštění. Framework tímto řeší konflikty verzí vytvořených aplikací tím, že v jednom počítači existuje více verzí modulu CLR. Toto řešení jistí, že vytvořené aplikace mohou běžet na verzích Frameworku, na kterých byly aplikace vytvořeny“*. (17)

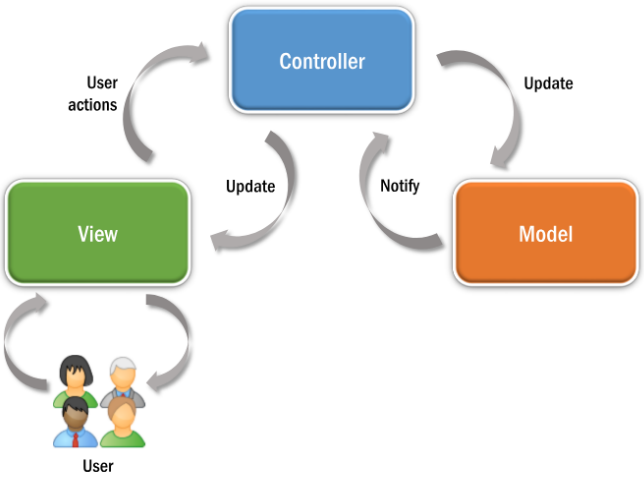
3.4.3 **Model MVC**

MVC architektura odděluje aplikaci do tří základních skupin komponent, jsou jím model, zobrazení a controllery. Hlavním principem Modelu MVC je oddělení programu, podle účelu používání a také podle druhů práce. Tímto principem se požadavky uživatelů přesměrovávají na controller, který zprostředkovává práci s modelem, se kterým provádí uživatelské akce a načítání výsledků dotazů. (18)

3.4.4 **ASP.NET Core MVC**

Architektura je odlehčená opensourcová prezentační architektura. Poskytuje způsob pro vytváření dynamických webových aplikací a webů. Controller propojuje model s pohledem a poskytuje uživateli data. Model obsahuje veškerou logiku kódu. Funkce modelu je především příjem parametrů z venku a vysílání dat ven, což znamená, že neví, odkud data přicházejí a jak budou formátována a následně vypsána. Pohled má na starosti zobrazení výstupu uživateli. Tato architektura je založena především na směřování. Směřování porovnává příchozí požadavky http protokolu a odesílá tyto požadavky do spustitelných koncových bodů aplikací. Směřování může být také založené na konvencích. Konvenční směřování definuje globální formáty URL adres, které webové aplikace přijímají a poskytuje způsob pro mapování formátů na konkrétní metodu akce na daném controlleru. Pro převod dat požadavků klienta na objekty, které controller může zpracovat, slouží vazby modelu. ASP.NET Core MVC má také možnost ověřování. Ověřování funguje tím, že objekt modelu dekóduje atributy ověření datových poznámek. Tyto atributy jsou kontrolovány na straně klienta před odesláním hodnot na server. Architektura má také možnost ověřování dat požadavků na serveru. (19)

Architektura má podporu injektáže závislostí. Tato závislost dovoluje, aby objekt závisel na jiném objektu. Injektáž se realizuje pomocí controllerů, které vyžadují služby prostřednictvím konstruktorů, díky kterým mohou dodržovat explicitní závislosti. Pro testování je možné použití rozhraní a injektáže. Architektura obsahuje funkci „testHost“, díky kterému se dají usnadnit integrační testy, které slouží pro zajištění správné kompatibility komponent aplikace. (20)



Obrázek 9 Architektura MVC (21)

3.5 **PHP Symfony**

PHP Symfony je opensourcový Framework licencovaný MIT pro PHP. Tento framework je tvořený komponenty PHP, které zjednodušují vývoj webových aplikací. Tato technologie je objektově navržená na principech MVC architektury. Webová aplikace se dělí na controllery, modely a pohledy, jako u ASP.NET Core MVC. (22) PHP Symfony nemá v základu instalované controllery. Pro instalaci controlleru se musí doinstalovat „Maker Bundle“, který slouží pro generování různých tříd. Tato technologie nemá možnost předávání závislostí přes konstruktor. Místo konstruktoru existuje vlastnost „ConstainerAwareTrait“, která umožňuje předat controlleru celý kontejner. (23)

4 Vlastní práce

4.1 Funkční a nefunkční požadavky

Při návrhu webové aplikace jsou důležité funkční a nefunkční požadavky. Za tímto účelem jsou ve funkčních požadavcích vypsány požadavky, jaké funkcionality mají fungovat v prototypu. V nefunkčních požadavcích jsou požadavky, které daný prototyp musí splňovat z hlediska podpory.

4.1.1 Funkční požadavky

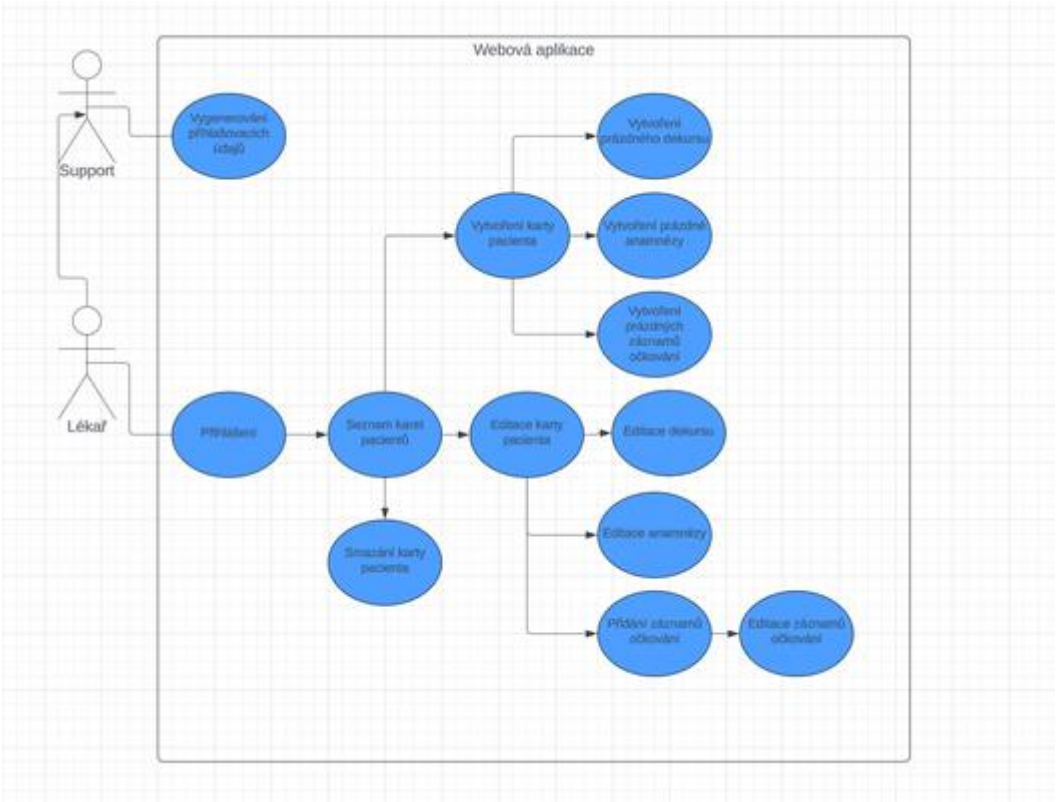
- Přihlášení – Webová aplikace umožní lékaři přihlášení do webové aplikace
- Vytvoření karty pacienta – Webová aplikace umožní lékařům vytvoření karty pacienta, následnou editaci karty pacienta a smazání karty pacienta.
- Vyhledávání karty pacienta – Webová aplikace umožní lékaři, aby bylo možné vyhledávat karty pacientů podle jména, příjmení anebo rodného čísla.
- Vytvoření záznamů – V kartě pacienta systém umožní lékaři vytvořit záznamy dekursu, anamnézy a historie očkování pacienta. Při vytváření záznamů pacientova očkování webová aplikace umožní lékaři založit záznam a následně jakýkoliv dřívější záznam o naočkování pacienta editovat.

4.1.2 Nefunkční požadavky

- Uživatelsky přívětivé prostředí – Webová aplikace musí být přehledná, jelikož se klade důraz na rychlost zadávání dat do webové aplikace.
- Responzivní design – Webová aplikace umožňuje používání na počítačích, tabletech i telefonních zařízeních.
- Udržitelnost a rozšiřitelnost – Webová aplikace je přizpůsobená k používání na různých druzích webových prohlížečů.
- Autorizace přihlášení – Webová aplikace ověřuje, zda jsou přihlašovací údaje správné.
- Snadná navigace – Webová aplikace umožňuje lékaři rychle a snadně se pohybovat na webové aplikaci

4.2 Use case diagram

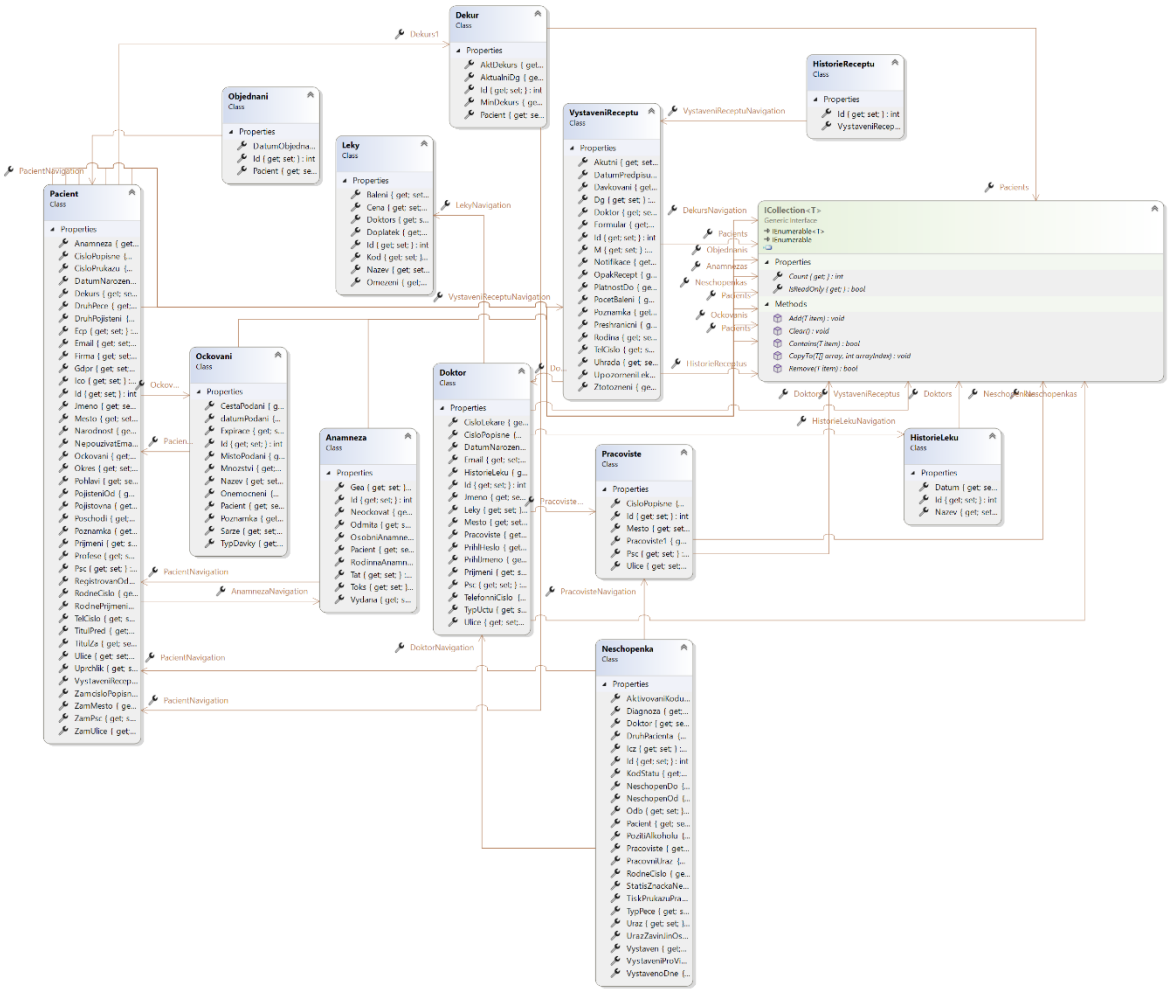
Při návrhu webové aplikace je vytvořen Use case diagram, který popisuje interakce mezi Support týmem a lékařem a webovou aplikací pro správu pacientských dat. Diagram ukazuje různé akce, které praktický lékař může provádět, jako je přihlášení Support týmem vygenerovanými přihlašovacími údaji, vytvoření, editace, smazání pacientských karet, které obsahují záznamy z dekursu, anamnézy a historie očkovaných vakcín daným pacientům.



Obrázek 10 UseCase diagram

4.3 Návrh aplikace – UML diagram

Před vývojem webové aplikace je vytvořen UML diagram. Tento diagram je vytvořen aby byla možná představa, jaké modely jsou potřeba pro správný chod programu a jaké vlastnosti mají jednotlivé modely. Po návrhu modelů a jejich vlastností je potřeba doplnit pro jednotlivé modely také jejich vazby. Při návrhu vazeb je důležité znázornit, jaká vazba propojuje jednotlivé modely. Po dokončení návrhu UML diagramu je znázorněná kostra, jak daná aplikace na konci vývoje vypadá.



Obrázek 11 UML Diagram webové aplikace

4.4 Příprava praktické části

Na začátku vývoje aplikace je potřeba projít veškeré dostupné funkce aktuálně používaných aplikací. Aktuální aplikace mají velké množství funkcí, které pro praktické lékaře nejsou důležité a nejsou používané. Je tedy potřeba vyhodnotit, které funkce praktický lékař používá, a které funkce naopak nepoužívá. Po vybrání důležitých funkcí, které praktičtí lékaři používají, je v dalším kroku navrhnuta databáze. Při navrhování databáze musí být promyšleno, jaké tabulky mají mít vazby mezi sebou a také co je v jednotlivých tabulkách obsaženo.

4.5 Vytvoření databáze

Prvním krokem je nainstalování virtuálního SQL serveru pro počítač. „*SQL server je výkonný databázový systém, využívaný zejména ve středních a větších institucích*“. (24) SQL server se v možnostech instalace nakonfiguruje a dokončí se instalace. Po nainstalování SQL serveru se stáhne prostředí pro vývoj databází Microsoft Management studio. Ve vývojovém prostředí se musí připojit na server a založit dotaz (Querry). V dotazu se založila databáze jménem PracMedic.

4.5.1 Založení tabulek v databázi

Všechny tabulky jsou vytvořeny dotazy, které se psaly do querry textového souboru, který má koncovku .sql. Tabulky byly vytvořeny dotazy create table jmenoTabulky();. V závorkách těchto příkazů jsou napsány dotazy na vytvoření jednotlivých sloupců s jejich datovými typy. První řádek je určen pro vytvoření hlavního sloupce, kde se generují a následně ukládají ID jednotlivých záznamů. Použitá syntaxe pro vytvoření prvního sloupce je id int IDENTITY(1,1) primary key. Tento dotaz zakládá sloupce s číselnou proměnnou a číslování od jedné do začátku zapisování. Další záznamy se zvyšují o jednu, poslední je primary key, který určuje, že tento sloupec je primárním klíčem tabulky. Při ukládání jsou jednotlivé záznamy v řádcích. Další sloupce jsou vytvořeny určením jména sloupce a datovým typem. Nejpoužívanějším datovým typem v databázi je varchar(početBitů). Dalšími použitými datovými typy jsou boolean, integer a date. Poslední důležitou hodnotou bylo definovat hodnotu null, která dovoluje ukládat prázdné hodnoty do databáze.

4.5.2 **Definování tabulek v databázi**

První vytvořenou tabulkou je tabulka Doktor. Tato tabulka slouží pro evidenci lékařů, kteří jsou zaregistrováni do databáze a mohou tak používat vytvářený program. Zaregistrování lékařů nutné k tomu, aby mohli používat program je zamýšleno tak, že napíšou a zažádají podporu o přístup. Lékaři vyplní dotazník s osobními údaji a následně jim bude podporou vygenerováno přihlašovací jméno, které je ve formátu e-mailu a přihlašovacího hesla, které bylo automaticky vygenerováno. Následně podpora pošle potvrzení o zaregistrování s přihlašovacími údaji.

V tabulce Doktor jsou definovány osobní údaje s adresou ordinace lékaře. Po základní definici údajů se do tabulky ukládají kontakty a osobní číslo lékaře, které je důležitou součástí pro ověření, zda registrovaný je opravdu lékař.

Po hlavní tabulce Doktor je vytvořena tabulka Pacient, kam se ukládá seznam pacientů všech lékařů, kteří používají program. Tabulka Pacient je definována jak pro ukládání základních informací, tak pro ukládání rozšířených informací. Mezi základní informace patří osobní údaje, trvalé a přechodné bydliště pacienta a telefonické a emailové kontakty pacienta. Mezi rozšířené informace o pacientovi patří informace o pojištění pacienta, od kdy a u které zdravotní pojišťovny je pojištěn, od kdy je u lékaře zaregistrován, číslo zdravotního pojištění, druh poskytované péče – zda pravidelná nebo nepravidelná. Také mezi tyto informace patří rodné číslo. Pokud je pacient z jiné země v rámci EU, rodné číslo nahrazuje EČP, takže se vyplňuje sloupec pro EČP. Dále se zapisují informace o zaměstnání pacienta, kdy se ukládá IČO, název a adresa firmy, ve které pacient pracuje, profese, kterou vykonává. Po vytvoření a definování tabulky Pacient je nutné vytvořit tabulky Dekurz, Anamnéza, Očkování.

Tabulka Dekurz je velmi důležitá. Lékař do ní zapisuje aktuální zdravotní stav pacienta při vyšetření, jeho subjektivní potíže. Také se do dekurzu zapisují výsledky laboratorních vyšetření, předepsané léky, pracovní neschopnosti, případně odborné nálezy z propojených zařízení.

Další tabulkou je tabulka Anamnéza, která slouží jako archiv, který se dělí na rodinnou a osobní anamnézu. V rodinné anamnéze se shromažďují údaje o nemocech, které se vyskytly v rodině. V osobní anamnéze se zapisují nemoci, které pacient za život prodělal. Vedle těchto údajů se v anamnéze ukládá, kdy pacient byl naposledy očkován proti tetanu. U tohoto údaje má lékař k dispozici možnost zaškrtnou políčko neočkovat, pokud si pacient

nepřeje být naočkován. Dalším údajem je údaj TOKS, kde se ukládá datum, kdy byl pacient naposledy testován při prevenci rakoviny tlustého střeva. Pod tímto údajem je kolonka odmítá, kterou lékař zaškrtne, pokud pacient nepřeje být testován na prevenci rakoviny tlustého střeva. Vedle tohoto údaje je možnost zaškrtnou políčko sledován na GEA (gastroenterologická ambulance).

Po tabulce Anamnéza je vytvořena poslední tabulka spojená s pacientem, která se jmenuje Očkování. V této tabulce se ukládá název vakcíny, podané množství, v jakém pořadí očkování byla vakcína naočkována pacientovi, typ dávky, šarže dávky, expirace dávky, proti jaké nemoci je vakcína podána, cesta a místo podání vakcíny. Pokud by lékař potřeboval, tak do této tabulky je možnost ukládání poznámky, kde si lékař může uložit, cokoliv uzná za vhodné ohledně očkování pacienta.

Další tabulkou je tabulka Pracoviště, která slouží pro ukládání informací o pracovišti lékaře. V této tabulce se ukládá, zda lékař je v samostatné ordinaci či nemocnici a také hlavní adresu pracoviště lékaře.

Na tabulku Pracoviště navazuje tabulka Vystaveni receptu. Do receptu je potřeba zadat základní informace o lékaři, který daný recept vydal. Po těchto základních informacích je do tabulky ukládáno datum, do kdy je recept platný, základní informace o vypsaných lécích, včetně dávkování a počtu předepsaných balení. Zde se také ukládají informace, zda pacient je přeshraniční, telefonní číslo či email potřebný pro zaslání elektronického receptu. Dále zda lékař chce být upozorněn při vydání léků z receptu a zda chce definovat opakované vydávání léku na receptu.

Po tabulce receptů je založena tabulka jménem Neschopenka, kde se ukládají základní údaje o pacientovi, včetně národnosti a statistické značky nemoci. V této tabulce se ukládá také z jakého důvodu je pacientovi vydána neschopenka, zda jde o úraz, pracovní úraz a zda je daná neschopenka použita pro více zaměstnavatelů. Posledním hlavním údajem je datum vydání neschopenky a do kdy je daná neschopenka platná.

Pro budoucí možnost použití API pro léky je vytvořena tabulka Léky, kde se ukládá název léku, identifikační kód léku, počet balení, omezení používání, cena léku a možný doplatek v lékárně.

Poslední tabulkou pro budoucí rozšíření aplikace je založení tabulky pro objednání pacientů, kde se ukládá datum objednání pacienta.

```
create table Doktor(
id int IDENTITY(1,1) primary key not null,
jmeno varchar(30) null,
prijmeni varchar(30) null,
email varchar(50) null,
ulice varchar(60) null,
cisloPopisne varchar(10) null,
mesto varchar(40) null,
psc varchar(10) null,
datumNarozeni date null,
telefonniCislo varchar(25) null,
cisloLekare varchar(20) null,
prihlJmeno varchar(50) null,
prihlHeslo varchar(50) null,
typUctu varchar(20) null,
pracoviste int null,
leky int null,
historieLeku int null);
```

Obrázek 12 Kód dotazu vytvoření tabulky Doktor

4.5.3 **Propojení tabulek v databázi**

Tabulka Doktor má definované propojení s tabulkami HistorieLeku, Leky, Pracoviste. Tabulka Doktor využívá vazby skrz cizí klíče. Vytvoření cizího klíče funguje přes vytvoření proměnné typu integer, která by se nejlépe měla jmenovat podle tabulky, do které se díky dotazu uloží primární klíč cílové tabulky. Ve všech aktuálně vytvořených tabulkách v databázi je primární klíč připojen na jednotlivé id. V tabulce Doktor jsou vytvořeny číselné proměnné, které se jmenují historieLeku, leky a pracoviste. Do všech těchto proměnných se uloží cizí klíče, které budou mít hodnotu null, aby se založil pacient, aniž by lékař musel vyplňovat i propojené tabulky. Proměnné s cizími klíči budou odkazovat na primární klíče tabulek, které jsou definovány podle jmen proměnných. Cizí klíč je vytvořen pomocí dotazu alter table Doktor add foreign key (pracoviste) references Pracoviste(id). V tomto dotazu je příkaz alter table, který slouží pro dodatečnou editaci, smazání, přidání definovaných proměnných do tabulky po založení tabulek. Po příkazu definujeme tabulku, do které přidáme cizí klíč. Příkaz add foreign key (pracoviste) založí cizí klíč do proměnné, která je definována v závorkách. Po definování proměnné je references Pracoviste(id), kde

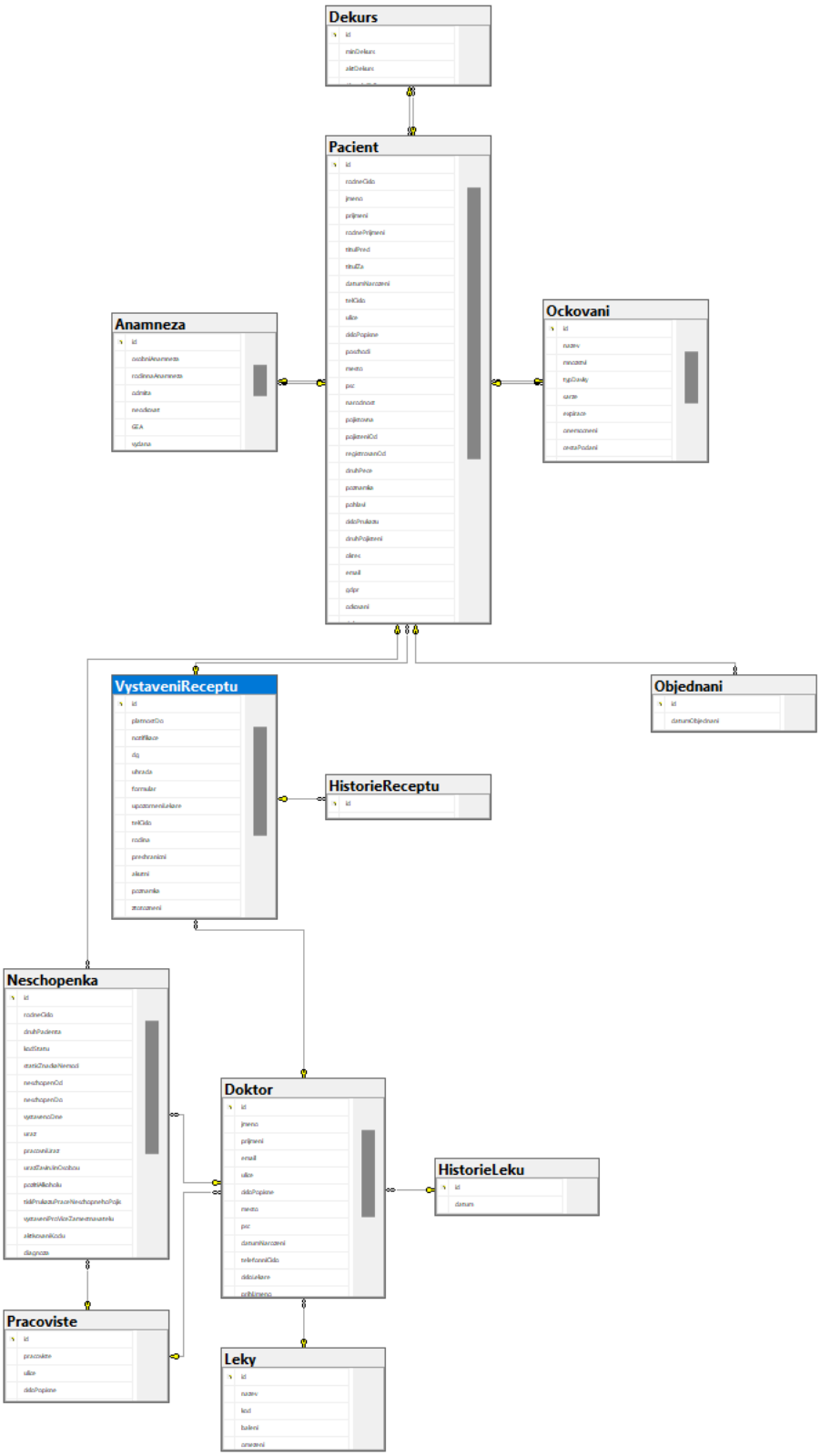
je odkazování na jméno tabulky a v závorkách je definována proměnná, ve které je uložený primární klíč. Tabulka Doktor je propojená s tabulkou Pracoviště z důvodu dodatečných informací o lékaři. Pokud lékař musí vytvořit recept na lék, tak se do receptu automaticky uloží informace o lékaři a jeho pracovišti, aby se mohla ověřit pravost receptu a tím lékárníci mohli pokračovat ve vydávání předepsaných léků. Tabulka Doktor je tímto také propojena s tabulkou HistorieLeku, která slouží k ukládání historie vypsaných receptů pacientům.

Tabulka Pacient je propojena s tabulkami Dekurs, Anamnéza, Očkování, VystaveniReceptu. Tabulka pacient má definované číselné proměnné pro ukládání cizích klíčů, které se jmenují dekurs, anamneza, ockovani, vystaveniReceptu. Tabulka Pacient je propojena s tabulkami proto, že každý pacient má pouze jeden dekurs, který se přepisuje na nejaktuálnější uložení, jednu anamnézu, která se upravuje podle změn historie onemocnění a historii očkování, kde se upravují očkovací látky, které pacient měl v historii aplikované. Tabulky jsou ale definovány cizími klíči k tabulce Pacient z důvodu ukládání jednotlivých id každého záznamu, pro správné načítání a ukládání záznamů do tabulek mezi sebou.

Tabulka Objednani je propojena s tabulkou Pacient, aby při objednávání pacientů byly rovnou automaticky definovány základní údaje pacienta.

Tabulka VystaveniReceptu je propojena s tabulkou Doktor, aby v receptu byly automaticky definovány informace o lékaři, který daný recept vystavil pacientovi. Tabulka HistorieReceptu je propojena s tabulkou VystaveniReceptu, z důvodu načítání, znovu použití již dříve vystavených receptů pacientovi.

Poslední tabulkou je tabulka Neschopenka, která je propojena s tabulkami Pracoviste, Pacient, Doktor. Tabulka je propojena s tabulkami proto, aby se při vystavování neschopenek automaticky vyplnily informace o lékaři, který neschopenku vydává a také informace o jeho pracovišti. Aby se neschopenka přiřadila k danému pacientovi, kterému lékař neschopenku vydává.



Obrázek 13 Databázový diagram

4.6 Příprava prostředí vývoje aplikace

Nejprve je nutné vytvořit projekt, který je pro vývoj webových aplikací. Při zakládání projektu je vybrána možnost ASP.Net Core Web App (Model-View-Contorller) v jazyce C#. Po zvolení této možnosti je zvoleno jméno projektu PracMedic, který představuje název vyvíjené aplikace. Dalším krokem při založení projektu je výběr nejnovějšího frameworku 8.0, který má v budoucnu delší časovou podporu. Dále je v konfiguraci provedeno základní nakonfigurování projektu pro https protokol. V konfiguraci projektu není nakonec definováno žádné ověřování.

Po založení projektu je potřeba nainstalovat nugget balíčky, které obsahují již zkompilovaný kód, který je podporovaný Microsoftem. Díky balíčkům jsou k dispozici nástroje, které jsou potřebné pro vytvoření webové aplikace a její funkčnosti. Prvním nainstalovaným balíčkem je balíček EntityFramework, který slouží pro jednodušší práci s databází a s manipulováním dat, migrace databáze, slouží pro úpravu databáze podle potřeby, LINQ dotazů, pro možnost psát databázové dotazy přímo v kódu C#. Dalším balíčkem je balíček Microsoft.AspNetCore.Components.QuickGrid.EntityFrameworkAdapter, který je využíván pro lepší integraci balíčku EntityFramework se standartou QuickGrid, která slouží pro zobrazení dat v tabulkové formě a je obsažena v nejnovějším základním frameworku Blazor. Dalším balíčkem je Microsoft.AspNetCore.Identity.UI. Tento balíček obsahuje uživatelské rozhraní, které je používáno pro ověřování při přihlašování. Dalším balíčkem je Microsoft.EntityFrameworkCore.Design, který je jeden z nejvíce užitečných balíčků. Tento balíček má v sobě možnost Scaffoldování DbsContextu, to znamená, že z existující databáze automaticky vygeneruje veškeré modely a následně i DbContext, který je prostředníkem mezi databází a webovou aplikací. Dalším balíčkem je Microsoft.EntityFrameworkCore.SqlServer, který umožňuje webové aplikaci používat SQL server, díky kterému webová aplikace může ukládat, upravovat a mazat data prostřednictvím objektově orientovaného přístupu, bez nutnosti použití SQL dotazů. Balíček Microsoft.EntityFrameworkCore.Tools slouží pro další manipulaci s daty a doplňuje možnosti ostatních balíčků EntityFrameworkCore. Posledním balíčkem je Microsoft.VisualStudio.Web.CodeGeneration.Design, který slouží jako doplnění automatizace scaffoldingu při generování. Dále usnadňuje modelování a práci s modely.

4.6.1 Připojení databáze

Pro připojení databáze je potřebné otevřít connected services, kde se vybere možnost přidání databáze z SQL serveru. Při výběru možností je vybrána možnost SQL Server Express LocalDB. Po zvolení možnosti se otevře založení ConnectionString, který je použit pro propojení databáze s webovou aplikací. U založení definujeme, jak se ConnectionString má nazývat a rozklikneme jeho hodnoty. Pro definování hodnot vybereme název databázového serveru, který má v sobě databázi. Po zvolení serveru se aktualizují možnosti vybrání databáze. Zvolíme databázi, která má v sobě již definované tabulky pro webovou aplikaci. Po vybrání názvu databáze připojíme databázi k webové aplikaci. Po připojení databáze je potřeba vygenerovat modely, které budou představovat tabulky v databázi. K vygenerování modelů je nutné do konzole nugget balíčků napsat příkaz Scaffold-DbContext "Server=LAPTOP-8HFTBRPN;Database=PracMedic;Integrated Security=True;TrustServerCertificate=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Force. V tomto příkazu je definován ConnectionString a jaký nugget balíček má provést generaci modelů. Následně je definováno, že do složky Models se mají vytvořit modely. Force se v tomto případě používá pro znovu vygenerování neboli aktualizaci či přepsání se změnami, které je potřeba pokud Models již mají modely. Modely jsou definovány ve formě tříd.

4.7 Vývoj aplikace

4.7.1 Stránka pro přihlášení

Prvním krokem pro přihlašování je založení složek všech stránek. Složky jsou založeny ve složce Views. Složka pro přihlašovací stránku se jmenuje Account. Ve složce je založena razor stránka, která se jmenuje Login. Po založení view Login je třeba pokračovat dalším krokem ve složce controllers a založit controller AccountController, který funguje jako třída, která řídí view Login, který je ve složce Account. Aby se view a controller propojily správně, musí se controller jmenovat podle jména složky, ve které je daný view. V razor stránkach se k stylování používají předdefinované hodnoty, které se volají v tagu jako parametr atributu class. Veškerý základní kód pro správný chod stránky je obalen v tagu form, kde je důležitý atribut asp-controller, ve kterém se definuje název controlleru. Atribut asp-action slouží pro definování metody v controlleru, kde se nachází hlavní logika

pro funkčnost kódu. Posledním atributem je method, kde je parametr post, který slouží pro odesílání dat na server. Hlavním tagem pro stylování je tag div, který používá předdefinované parametry v atributu class. Díky těmto parametrům je možné přihlašovací stránku používat i na mobilních zařízeních. Pro přihlašovaní je na stránce vytvořený tag label, který slouží pro zobrazování textu a tag input, kam uživatel píše data, která se mají odeslat na server na ověření. Tag input má atribut type, který je definovaný skrze parametr email a password, že se jedná právě o ověřování emailem a heslem. U emailu je díky tomuto parametru aplikovaný automatický regex pro validaci zadávání vyplněných údajů, aby byl údaj podobný emailové adrese. Pod inputy, kde uživatel zadává data je tag button, který slouží pro zobrazení tlačítka. Button má definovaný atribut type s parametrem submit, který je používán pro potvrzení a odeslání dat na ověření, zda se shodují data s daty již uloženými v databázi.

```
<form asp-controller="Account" asp-action="Login" method="post">
    <img class="mb-4" src="/docs/5.0/assets/brand/bootstrap-logo.svg" alt="" width="72" height="57">
    <h1 class="h3 mb-3 fw-normal">Přihlášení</h1>
    <div class="form-floating">
        <input type="email" class="form-control" placeholder="name@example.com">
        <label>Emailová adresa</label>
    </div>
    <div class="form-floating">
        <input type="password" class="form-control" placeholder="Password">
        <label>Heslo</label>
    </div>
    <button class="w-100 btn btn-lg btn-primary" type="submit">Přihlásit se</button>
    <p class="mt-5 mb-3 text-muted">©2024</p>
</form>
```

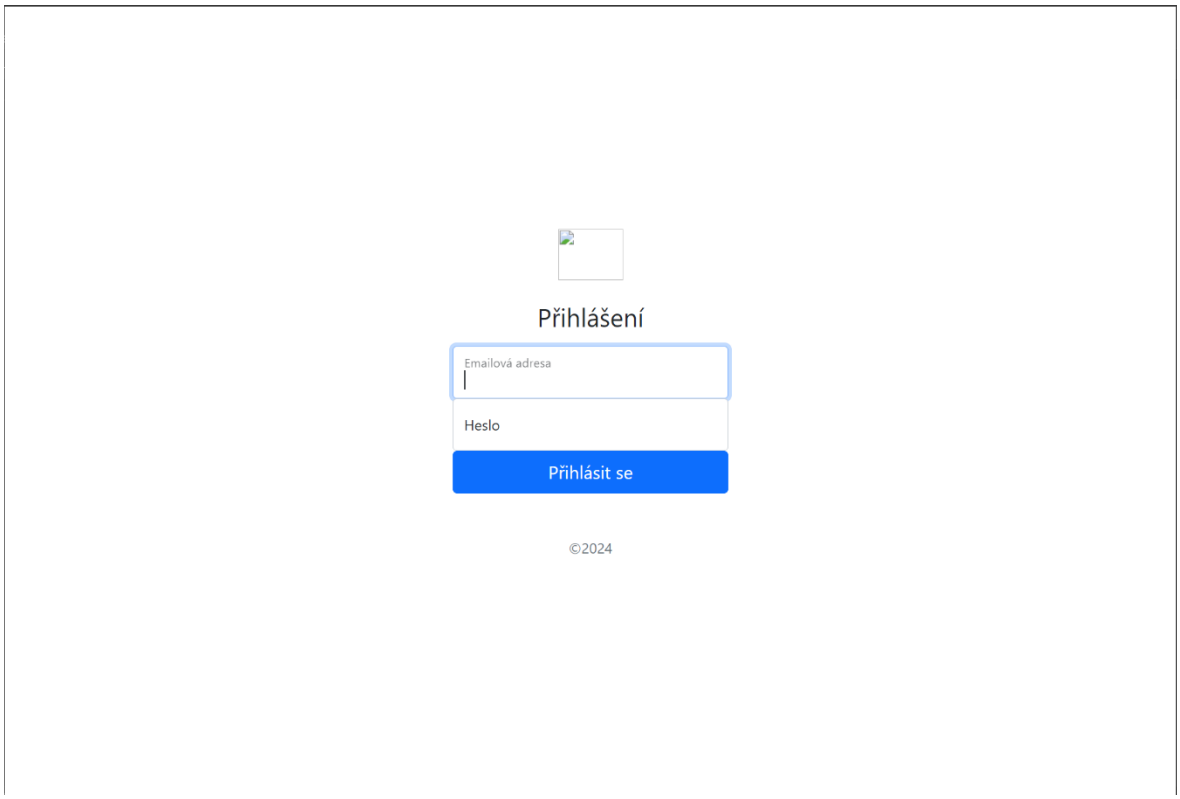
Obrázek 14 Login View

V controlleru Account je vytvořena logika, která má za účel zprovoznit již definovaný View Login. Do controlleru je založena proměnná jménem context, která má v sobě obsáhlá data ze třídy PracMedicContext. Pod proměnnou context je metoda, která se jmenuje Login. Jako parametry této metody jsou založeny proměnné typu string neboli textového řetězce pro email a heslo. V metodě Login založíme proměnnou doktor, ve které jsou uloženy data ze třídy Doktor, která se nachází ve složce Models. Do proměnné doktor uložíme LINQ dotaz, kde z proměnné context přistupujeme ke DbSet Doktors, který má v sobě obsaženou veškerou kolekci všech záznamů z tabulky Doktors. V metodě where se hledá doktor, jehož zadané PrihlJmeno odpovídá hodnotě v proměnné email v databázi. FirstOrDefault vrací nalezený záznam v databázi a pokud záznam nebyl nalezen, tak vrací hodnotu null. Dále se vytvoří podmínka, ve které se ověřuje, zda proměnná doktor není null a zároveň se zadané heslo shoduje s již uloženým heslem v databázi. Pokud všechno souhlasí tak uživatel

je přesměrován na stránku Seznam, která je ve složce SeznamPacientu. Pokud přihlašovací jméno nebo heslo nesouhlasí, tak se stránka obnoví a Login View uživatel může znovu zkusit zadat přihlašovací údaje.

```
using Microsoft.AspNetCore.Mvc;
using PracMedic.Models;

namespace PracMedic.Controllers
{
    0 references
    public class AccountController : Controller
    {
        PracMedicContext context = new PracMedicContext();

        0 references
        public async Task<ActionResult> Login(string email, string heslo)
        {
            Doktor doktor = new Doktor();
            doktor = context.Doktors.Where(x => x.PrihlJmeno == email).FirstOrDefault();
            if (doktor != null && doktor.PrihlHeslo == heslo)
            {
                return RedirectToAction("Seznam", "SeznamPacientu");
            }
            return View();
        }
    }
}
```

Obrázek 15 Login controller



Obrázek 16 Stránka přihlašování

4.7.2 **Seznam pacientů**

Seznam pacientů je vytvořen ve složce Views v podsložce SeznamPacientů. Stránka je pojmenována Seznam. Tento View používá dynamický model, ve kterém jsou automaticky dostupné veškeré modely. Jako první je založena možnost pro filtraci karet pacientů. Tato filtrace je složena z tagu form, který odkazuje na controller SeznamPacientů a v controlleru na metodu Seznam. Form má také definovaný atribut method s parametrem post, který posílá data na server. Ve formu je založen input, který má v sobě definovaný atribut type s parametrem search, který aktivuje možnost vyhledávání. Dále je ve formu button, ve kterém je definován atribut type s parametrem submit. Dále je pod filtrací založen odkazovací tag, který je nastylován, jako tlačítko. Tento odkazový tag je definovaný na controller Pacient s vybranou metodou AddPacient. Slouží k přesměrování na View AddPacient, kde se zakládá nový pacient. Posledním tagem je tag table, kde je vytvořena tabulka, která ukazuje vybrané informace při zobrazování karet pacientů. Nejprve je v tabulce založen první řádek, kde jsou založené sloupce s textem, pod kterým budou vypisované informace o pacientech. Další založený řádek je dynamický, protože je v cyklu foreach, který porovnává každý definovaný sloupec mezi každým zadaným sloupcem. Jako parametr cyklu je definovaná proměnná typu model Pacient. Dále se určí zdroj dat, ze kterého se má vypisovat přes cykl. Nakonec je v cyklu založen odkaz, který funguje na editaci karty pacienta. Tento odkaz je nadefinován v controlleru Pacient, ve kterém je metoda EditPacient. V odkazu je také přesměrování na kartu podle pacientova id v databázi.

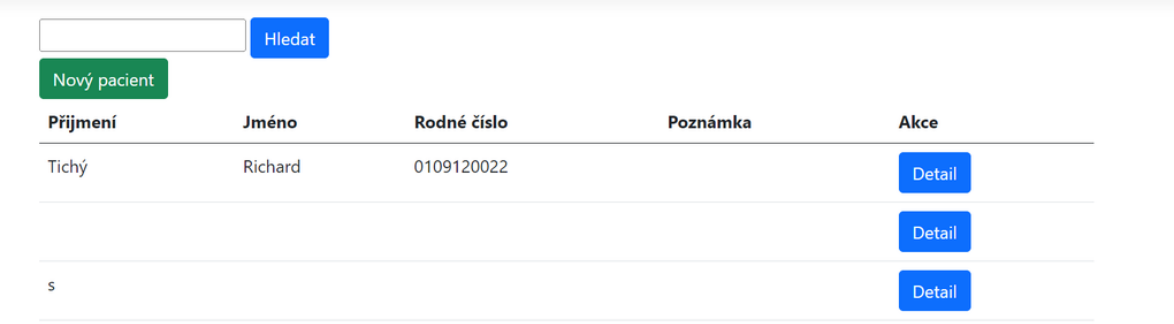
```
<div class="row">
    <div class="col-md-9">
        <form asp-controller="SeznamPacientu" asp-action="Seznam" method="post">
            <input type="search" name="searchString"></input>
            <button type="submit" class="btn btn-primary">Hledat</button>
        </form>

        <a asp-controller="Pacient" asp-action="AddPacient" class="btn btn-success">Nový pacient</a>
        <table class="table">
            <thead class="thead-light">
                <tr>
                    <th>Příjmení</th>
                    <th>Jméno</th>
                    <th>Rodné číslo</th>
                    <th>Poznámka</th>
                    <th>Akce</th>
                </tr>
            </thead>
            <tbody>
                @foreach (Pacient vypis in Model)
                {
                    <tr>
                        <td type="text" id="Prijmeni" name="Prijmeni">@vypis.Prijmeni</td>
                        <td type="text" id="Jmeno" name="Jmeno">@vypis.Jmeno</td>
                        <td type="text" id="RodneCislo" name="RodneCislo">@vypis.RodneCislo</td>
                        <td type="text" id="Poznamka" name="Poznamka">@vypis.Poznamka</td>
                        <td>
                            <a asp-controller="Pacient" asp-action="EditPacient" asp-route-id="@vypis.Id" class="btn btn-primary">Detail</a>
                        </td>
                    </tr>
                }
            </tbody>
```

Obrázek 17 View seznamu pacientů

V controlleru SeznamPacientu je založena proměnná context, která obsahuje základní struktury modelů. Pak je založen list, který se jmenuje pacients a obsahuje model Pacient. Po proměnné je vytvořen konstruktor, ve kterém je do proměnné pacients typu list vloženo nastavení dat z databáze. V základní metodě Seznam je podmínka pro filtraci pacientů.
V podmínce do proměnné pacient jsou uložena data, která se rovnají záznamu v databázi podle příjmení, jména nebo rodného čísla. Do vracející se hodnoty je vložena jako parametr proměnná, která vypíše nalezené návrhy karet pacientů.

```
public class SeznamPacientuController : Controller
{
    PracMedicContext context = new PracMedicContext();
    private List<Pacient> pacients;

    0 references
    public SeznamPacientuController()
    {
        this.pacients = this.context.Pacients.ToList();
    }

    0 references
    public IActionResult Seznam(string searchString)
    {
        if (!String.IsNullOrEmpty(searchString)) // pro hledani
        {
            IEnumerable<Pacient> pacient = this.pacients.Where(s => (s.Prijmeni != null && s.Prijmeni.Contains(searchString))
            || (s.Jmeno != null && s.Jmeno.Contains(searchString)) || (s.RodneCislo != null
            && s.RodneCislo.Contains(searchString))).ToList();
            return View(pacient.ToList());
        }
        return View(context.Pacients.ToList());
    }
}
```

Obrázek 18 Cotnroller Seznamu



Obrázek 19 Stránka seznamu karet pacientů

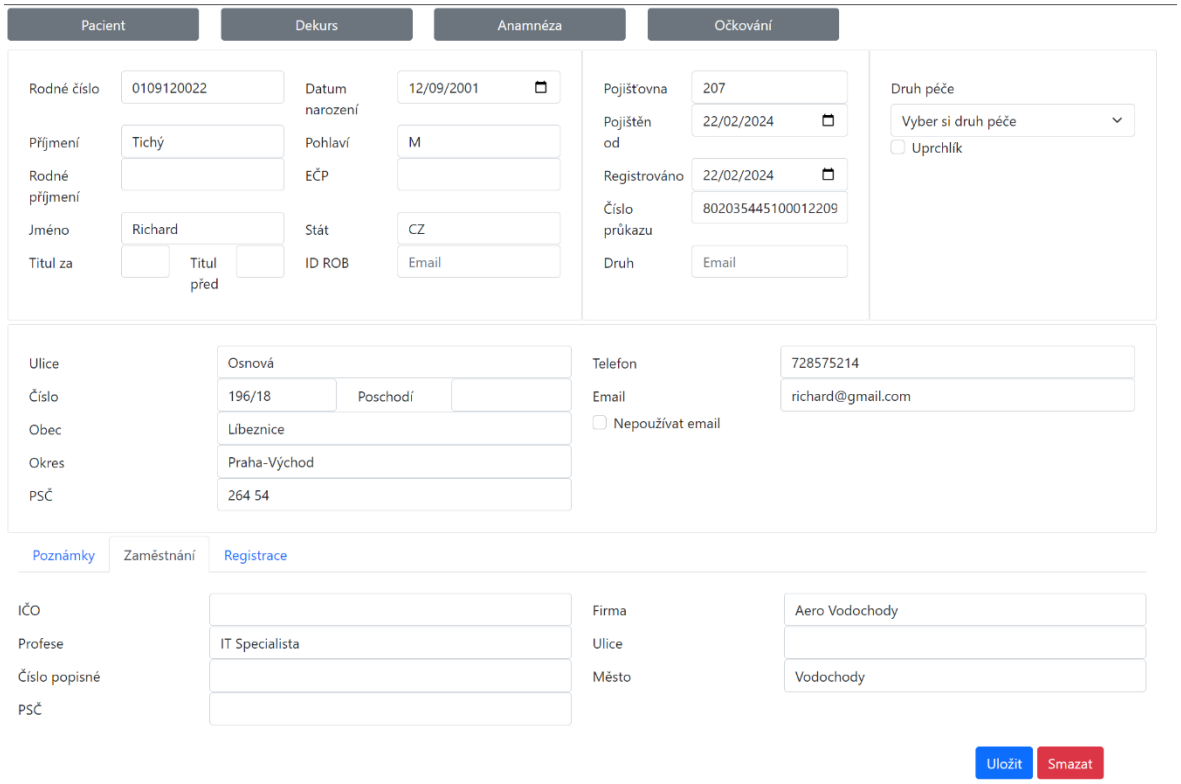
4.7.3 Stránka karta pacienta

Karta pacienta se dělí na dva Views. Prvním z nich je AddPacient, který slouží pro založení nové karty pacienta. Nová karta pacienta je prázdná, bez možnosti překlikávání mezi pacientem, dekursem, anamnézou a očkováním. Tato navigace je skryta z důvodu zakládání záznamů na jednotlivých stránkách. Do prázdné karty pacienta lékař zapíše veškeré potřebné informace, které o pacientovi potřebuje znát a následně zmáčkne tlačítko uložit. Při ukládání se vygeneruje do databáze nový záznam s unikátním id a do záznamu se uloží všechny vyplněné informace. Zároveň s novým záznamem se vytvoří i nové prázdné záznamy anamnézy, dekursu, očkování. Po založení záznamů webová aplikace přesměruje lékaře zpět na stránku Seznam. Pro editaci záznamů se použije EditPacient, kde jsou jednotlivé inputy automaticky vyplněny a mohou být upravovány. Pro navigaci mezi stránkou Pacient a podstránkami jsou založeny odkazy, ve kterých jsou definovány jednotlivé akce na přesměrování, kde je uložen ve ViewBagu id záznamů. Dalšími akcemi jsou definovány controllery a metody, ve kterých je kód pro správné zprovoznění funkčního proklikávání mezi stránkou a podstránkami. Každý input a label ve View má akci asp-for, kde je definován záznam, který bude použit na ukládání, či vypsání dat.

V controlleru Pacient je založena proměnná, která obsahuje context modelů. Pod proměnnou context jsou založeny soukromé listy, které obsahují data z modelů anaméza, dekurs, očkování a pacient. Pod proměnnými je konstruktor, který inicializuje tyto proměnné a připraví k použití. V controlleru jsou dvě stejnojmenné metody AddPacient. První metoda je pro akci HTTPGET, která vygeneruje prázdný formulář neboli stránku, kde

lékař může založit kartu nového pacienta. Druhá metoda je pro akci HTTPGET, kde jsou posílána zadaná data z inputů na server a případně provedeny změny v databázi. V této metodě je založena podmínka, ve které se ověřuje, zda zadaná data jsou v souladu s podmínkami v databázi. V podmínce se pak do cizích klíčů neboli vazeb ukládají nově vygenerované id záznamů anamnézy, dekursu, očkování. Dále se pomocí LINQ dotazu vyhledá pacient podle id dekursu a následně se nastaví pacientovo id do dekursu. LINQ dotaz je implementován i na anamnézu a očkování, kdy dotaz vyhledá pacienta podle id anamnézy, očkování a následně se do pacienta uloží jejich id. Pokud je podmínka splněna, tak uživatel je přesměrován na EditPacient s daty, které jsou vyhledány podle id.

Metoda EditPacient má také dvě metody. První slouží pro definování id za účelem překlikávání mezi stránkou pacienta a podstránkami dekurs, anamnéza a očkování. Pro tento účel se založí ViewBag, který se použije pro přesměrování na jednotlivé stránky v záznamu pacienta. Druhá metoda slouží pro odesílaní dat na server. V této metodě se propojují inputy s proměnnými v databázi. Díky tomuto propojení se rovnou pošlou data do databáze a pokud se něco změnilo od posledního záznamu, tak to databáze v záznamu přepíše a uloží. Při stisku tlačítka uložit se provede kontrola, zda zadaná data odpovídají podmínkám dat v databázi a následně je lékař přesměrován zpět na stránku Seznam. Poslední metodou v controlleru je metoda DeletePacient. Tato metoda slouží pro smazání všech záznamů v databázi k vybranému pacientovi. V metodě se nejprve načte z databáze id pacienta. Dále se načte id dekursu, anamnézy a očkování podle již zjištěného id pacienta. Aby se záznamy mohly smazat, tak je potřeba nastavit do vazeb hodnotu null. Do proměnných cizích klíčů se staticky uloží hodnota null. Po uložení se musí použít metoda SaveChanges z proměnné context, která uloží do databáze změny. Dále se smažou všechny záznamy, které byly součástí id daného pacienta. Po smazání je uživatel přesměrován na stránku Seznam.



Obrázek 21 Stránka karty pacienta

```
namespace PracMedic.Controllers
{
    1 reference
    public class SeznamPacientuController : Controller
    {
        PracMedicContext context = new PracMedicContext();
        private List<Pacient> pacients;

        0 references
        public SeznamPacientuController()
        {
            this.pacients = this.context.Pacients.ToList();
        }

        0 references
        public IActionResult Seznam(string searchString)
        {
            if (!String.IsNullOrEmpty(searchString)) // pro hledani
            {
                IEnumerable<Pacient> pacient = this.pacients.Where(s => (s.Prijmeni != null && s.Prijmeni.Contains(searchString))
                || (s.Jmeno != null && s.Jmeno.Contains(searchString)) || (s.RodneCislo != null && s.RodneCislo.Contains(searchString))).ToList();
                return View(pacient.ToList());
            }
            return View(context.Pacients.ToList());
        }
    }
}
```

Obrázek 20 Seznam karet pacientu controller

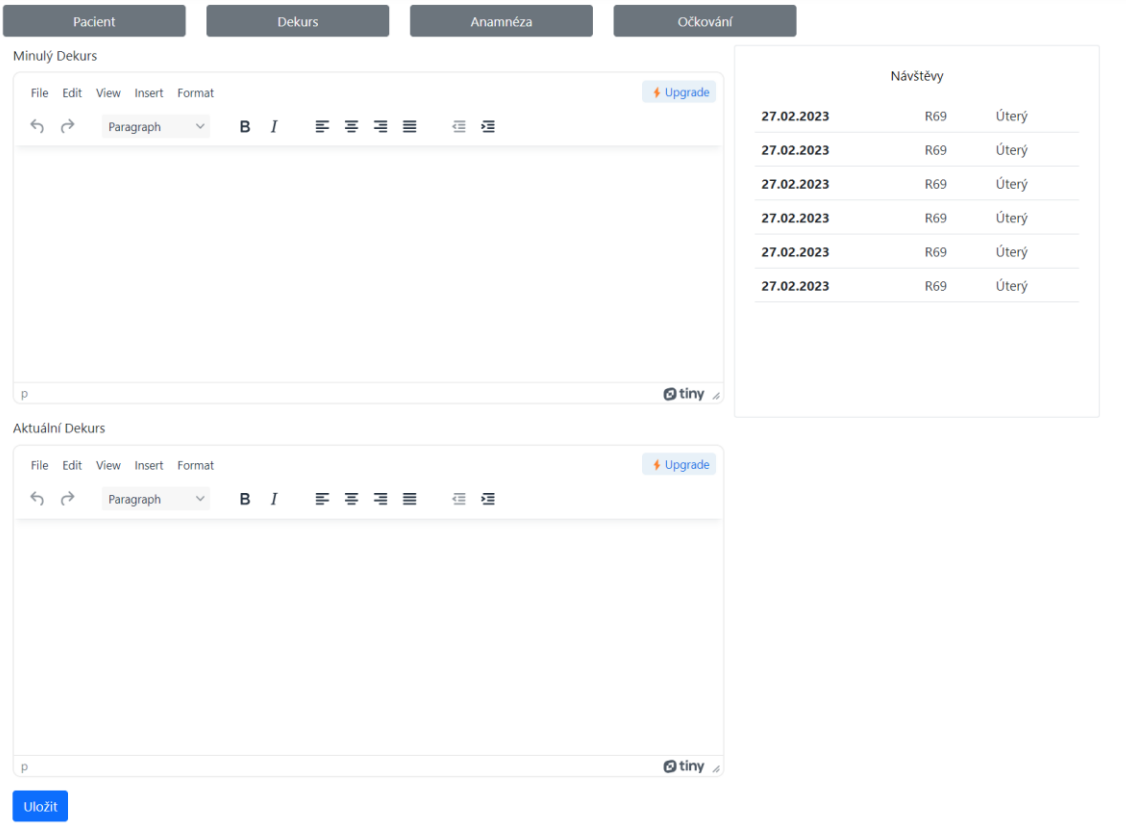
4.7.4 **Stránka dekurs**

Nejprve jsou ve View EditDekurs založeny odkazy na navigaci mezi stránkou Pacient a podstránkami Anamnéza a Očkování. Pod odkazy je pak založený form, který má definovanou metodu EditPacient. Ve formu jsou pak labely a pod nimi jsou inputy ve formě tagu textarea. Díky tomuto tagu je možnost základní editace textu v okně inputu. Nejprve je založená textarea, která odkazuje na textovou proměnnou MinDekurs, který slouží pro zobrazení napsaného dekursu při minulé kontrole. Další textarea je odkázána na proměnnou AktDekurs, kde lékař píše aktuální postup, co se během kontroly událo. Při uložení se AktDekurs uloží do MinDekursu, aby při příští kontrole lékař měl nejaktuálnější záznam z minulé kontroly. Při opětovném otevření dekursu bude AktDekurs prázdný a připravený na vytvoření nového záznamu, který se po uložení znovu uloží do minDekursu. Nakonec je v dekursu vidět historie objednání pacienta, kdy byl na prohlídkách. Tato tabulka je staticky nastavená, jelikož zatím slouží jen pro rozvrhnutí grafického rozhraní.

V dekurs controlleru je nejprve založena proměnná context a privátní listy pro pacienta a dekurs. Následně je založen controller pro tyto proměnné. Následně má controller dvě metody. První metoda je pro akci vygenerování stránky. V této metodě je nejprve zjištěno id dekursu, pak je potřeba vrátit list pacienta. K vrácení listu je použita funkce where, ve které filtrujeme, zda daný dekurs má shodné id s pacientem, podle kterého hledáme. Pod filtrací uložíme do ViewBagu minDekurs, díky kterému můžeme zprovoznit nahrazování minDekursu AktDekursem. Dále se založí ViewBagy, díky kterým se bude moci přesměrovávat mezi stránkou a podstránkami. Následně se vrátí v metodě id dekursu. Druhá metoda slouží pro posílání dat na server. V této metodě je podmínka pro ověřování podmínek, zda zadaná data souhlasí s podmínkami dat databáze. V podmínce zjistíme id dekursu a následně nastavíme, aby z inputu akDekursu se ukládala data do inputu minDekurs. Následně se při splnění podmínky a uložení lékař přesměruje na stránku Seznam. Pokud podmínka nebude splněná, tak se obnoví stránka.

```
namespace PracMedic.Controllers
{
    public class DekursController : Controller
    {
        PracMedicContext context = new PracMedicContext();
        private List<Dekur> dekurs;
        private List<Pacient> pacients;

        0 references
        public DekursController()
        {
            this.dekurs = this.context.Dekurs.ToList();
            this.pacients = this.context.Pacients.ToList();
        }

        [HttpGet]
        0 references
        public IActionResult EditDekurs(int id)
        {
            Dekur dekurs = this.context.Dekurs.Find(id);
            Pacient pacient = this.pacients.Find(x => x.Id == dekurs.Pacient);
            this.ViewBag.dekurs = dekurs.MinDekurs;

            this.ViewBag.pacientId = dekurs.Pacient; // pacient
            this.ViewBag.anamnezaId = pacient.Anamneza;
            this.ViewBag.ockovaniId = pacient.Ockovani;
            return View(dekurs);
        }

        0 references
        public IActionResult EditDekurs(Dekur dekurs1)
        {
            if (this.ModelState.IsValid)
            {
                Dekur db = this.context.Dekurs.Find(dekurs1.Id);
                db.MinDekurs = dekurs1.AktDekurs;
                db.AktualniDg = null;

                this.context.SaveChanges();

                return RedirectToAction("Seznam", "SeznamPacientu");
            }
            return View();
        }
    }
}
```

Obrázek 22 Dekurs controller



Obrázek 23 Stránka dekursu pacienta

4.7.5 Stránka anamnézy

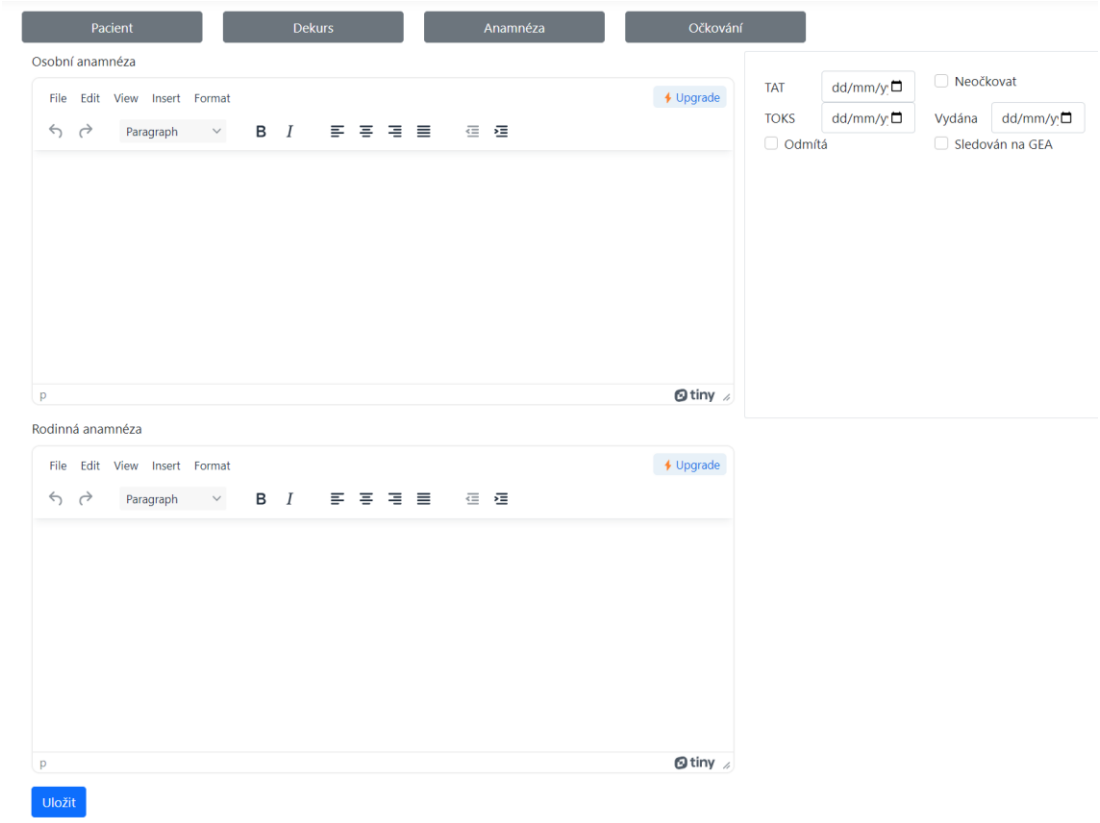
Ve View anamnézy jsou nejdříve založeny odkazy pro navigaci mezi stránkami. Pod odkazy je založen form, který je definován na metodu EditAnamneza. V anamnéze jsou vytvořeny dva inputy přes tagy textarea. Prvním inputem je osobní anamnéza pacienta. Do tohoto inputu lékař zapisuje veškeré dosud nemoci, které daný pacient za život prodělal. Druhým inputem je rodinná anamnéza, do které lékař zapisuje veškeré nemoci, jak dědičné tak prodělané v rodině pacienta. Dále lze na stránce anamnéza uložit datum očkování pacienta proti tetanu. Pokud by pacient nechtěl být očkován, tak lékař má možnost zaškrtnou checkbox, který má datový typ bool, který posílá hodnoty true nebo false. Další možností je možnost zaškrtnutí data, kdy pacient byl vyšetřen na krvácení do stolice. Pokud pacient nechce být vyšetřen, tak lékař má možnost zaškrtnou políčko o odmítnutí vyšetření. Posledním údajem je datum vyšetření a zda je sledován u lékaře s onemocněním zažívajícího traktu (GEA).

V controlleru anamnézy je založená proměnná context. Po této proměnné jsou založeny dva privátní listy pacient a anamnéza. Pod proměnnými je založen konstruktor

pro přípravu proměnných. Pod konstruktorem jsou dvě metody. První metoda slouží pro vygenerování stránky ze záznamu. V této metodě se nejdříve zjistí id záznamu anamnézy a po zjištění se id anamnézy porovná s id pacienta. Dále se založí ViewBagy, do kterých se uloží id jednotlivých záznamů na všech stránkách, aby byla možnost navigace mezi stránkami se záznamy, které souvisí s pacientem. V druhé metodě se odesílají data na server. V metodě je vytvořena podmínka pro ověření, zda data odpovídají podmínkám databáze. V podmínce se pak ke každému inputu přiřadí sloupec z databáze, díky kterým se bude moci ukládat a zapisovat do databáze.

```
namespace PracMedic.Controllers
{
    1 reference
    public class AnamnezaController : Controller
    {
        PracMedicContext context = new PracMedicContext();
        private List<Anamneza> anamneza;
        private List<Pacient> pacients;

        0 references
        public AnamnezaController()
        {
            this.anamneza = this.context.Anamnezas.ToList();
            this.pacients = this.context.Pacients.ToList();
        }

        [HttpGet]
        0 references
        public IActionResult EditAnamneza(int id)
        {
            Anamneza anamneza = this.context.Anamnezas.Find(id);
            Pacient pacient = this.pacients.Find(x => x.Id == anamneza.Pacient);
            this.ViewBag.dekursId = pacient.Dekurs;
            this.ViewBag.ockovaniId = pacient.Ockovani;

            this.ViewBag.pacientId = anamneza.Pacient;
            return View(anamneza);
        }

        0 references
        public IActionResult EditAnamneza(Anamneza anamneza1)
        {
            if (this.ModelState.IsValid)
            {
                Anamneza db = this.context.Anamnezas.Find(anamneza1.Id);
                db.OsobniAnamneza = anamneza1.OsobniAnamneza;
                db.RodinnaAnamneza = anamneza1.RodinnaAnamneza;
                db.Odmita = anamneza1.Odmita;
                db.Neockovat = anamneza1.Neockovat;
                db.Gea = anamneza1.Gea;
                db.Vydana = anamneza1.Vydana;
                db.Tat = anamneza1.Tat;
                db.Toks = anamneza1.Toks;
                this.context.SaveChanges();

                return RedirectToAction("Seznam", "SeznamPacientu");
            }
            return View();
        }
```

Obrázek 24 Anamnéza controller



Obrázek 25 Stránka anamnézy pacienta

4.7.6 **Stránka očkování**

Tato stránka má má vytvořené dvě View. První se jmenuje AddOckovani a druhá se jmenuje EditOckvani. AddOckovani je založena za účelem vytvoření záznamu o očkování pacienta. Také je založena pro možnost ukládat a zobrazovat historii očkování pacienta v minulosti. Při vytvoření nového záznamu očkování se vygeneruje nové id záznamu, díky kterému se v tabulce na stránce očkování zobrazuje historie, proti jakým nemocem byl jakou látkou pacient naočkován. Veškeré záznamy se na stránce očkování následně mohou v případě potřeby upravovat. EditOckovani je použit u editace záznamů, které obsahují historii vakcín, které byly pacientovi v minulosti naočkovány. Tabulka historie očkování obsahuje záznamy, kdy byla vakcína aplikována, její název a id záznamu.

V controlleru očkování je opět založena nejdříve proměnná context a pod ní jsou založeny soukromé proměnné s obsahem listu pacient a ockovani. Pod proměnnými je opět založený konstruktor, kde se proměnné připravují na použití. Pod konstruktorem jsou čtyři

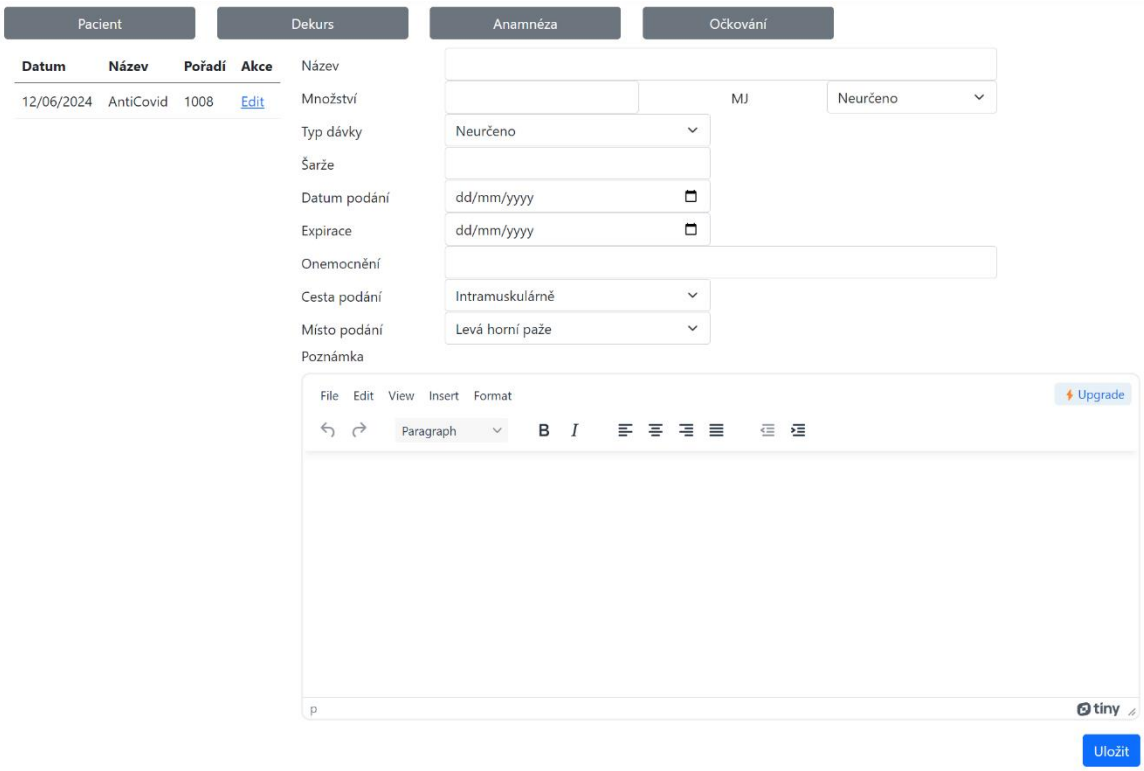
metody. Dvě metody jsou použity na založení záznamů historie očkování. První z nich slouží pro vygenerování stránky. V první metodě se nejdříve vyhledá id pacienta z databáze a následně se založí ViewBagy pro navigování mezi stránkami, které obsahují jednotlivé id. Nakonec se v metodě vrací vytvoření nového objektu očkování. Druhá metoda slouží pro samotné založení záznamu. V této metodě je podmínka pro ověření dat a v podmínce je definováno přidání nového záznamu do databáze. Nakonec se v metodě vrací stránka očkování s nově vygenerovaným parametrem id. Poslední dvě metody jsou pro editaci záznamů očkování. V první metodě se nejdříve načítá id záznamu očkování. U tohoto načítání se použije metoda Find, která vrací pouze jeden záznam podle parametru id. Dále se porovná id pacienta z databáze s id pacienta z očkování. Dále se v metodě vytvořily Viewbagy pro navigaci. Nakonec se v metodě získávají data z tabulky HistorieOckovani. V druhé editační metodě se vytvoří podmínka pro ověření dat a do podmínky se definují inputy se sloupci z databáze, díky kterým je možnost ukládat změny do databáze. Dále se uloží změny a pokud se podmínka splní, tak bude lékař přesměrován na stránku AddOckovani, kde bude moci založit další záznam o očkování. Pokud se podmínka nesplní, tak se stránka znovu načte.

```
1 reference
public class OckovaniController : Controller
{
    PracMedicContext context = new PracMedicContext();
    private List<Ockovani> ockovani;
    private List<Pacient> pacients;

    0 references
    public OckovaniController()
    {
        this.ockovani = this.context.Ockovanis.ToList();
        this.pacients = this.context.Pacients.ToList();
    }

    [HttpGet]
    0 references
    public IActionResult AddOckovani(int pacientId)
    {
        Pacient pacient = this.context.Pacients.Find(pacientId);

        this.ViewBag.anamnezaId = pacient.Anamneza;
        this.ViewBag.dekursId = pacient.Dekurs;
        this.ViewBag.pacientId = pacient.Id;
        this.ViewBag.ockovanis = this.ockovani.Where(x => x.Pacient == pacient.Id).ToList();
        return View(new Ockovani());
    }

    [HttpPost]
    0 references
    public IActionResult AddOckovani(Ockovani ockovani)
    {

        if (this.ModelState.IsValid)
        {

            this.context.Add(ockovani);
            this.context.SaveChanges();


            return RedirectToAction("AddOckovani", "Ockovani", new { pacientId = ockovani.Pacient });
        }
        return View();
    }
```

Obrázek 26 Očkováná controller #1

```
[HttpGet]
0 references
public IActionResult EditOckovani(int id)
{
    Ockovani ockovani = this.context.Ockovanis.Find(id);
    Pacient pacient = this.pacients.Find(x => x.Id == ockovani.Pacient);

    this.ViewBag.pacientId = ockovani.Pacient;
    this.ViewBag.dekursId = pacient.Dekurs;
    this.ViewBag.anamnezaId = pacient.Anamneza;


    this.ViewBag.ockovanis = this.ockovani.Where(x => x.Pacient == pacient.Id).ToList();


    return View(ockovani);
}


0 references
public IActionResult EditOckovani(Ockovani ockovani1)
{

    if (this.ModelState.IsValid)
    {
        Ockovani db = this.context.Ockovanis.Find(ockovani1.Id);

        db.Nazev = ockovani1.Nazev;
        db.Mnozstvi = ockovani1.Mnozstvi;
        db.TypDavky = ockovani1.TypDavky;
        db.Sarze = ockovani1.Sarze;
        db.Expirace = ockovani1.Expirace;
        db.Onemocneni = ockovani1.Onemocneni;
        db.CestaPodani = ockovani1.CestaPodani;
        db.MistoPodani = ockovani1.MistoPodani;
        db.Poznamka = ockovani1.Poznamka;
        db.datumPodani = ockovani1.datumPodani;

        this.context.SaveChanges();


        return RedirectToAction("AddOckovani", "Ockovani", new { pacientId = ockovani1.Pacient });
    }
    return View();
}
```

Obrázek 28 Očkování controller #2



Obrázek 27 Stránka očkování pacienta

5 Výsledky

5.1 Hodnocení lékaře po vyzkoušení webové aplikace

„*Pracuji jako praktická lékařka a dostala jsem možnost vyzkoušet webovou aplikaci PracMedic. V současnosti využívám lékařský software Duas altera, který je samozřejmě komplexnější a jednotlivé funkce přesahují možnosti této práce. Jeho hlavní nevýhodou je však nutnost licencovaného placeného přístupu ke každému počítači zvlášť. To v praxi znamená, že pokud potřebuji vyšetřit pacienta mimo ordinaci, např. v návštěvní službě, nemám k softwaru přístup. Mám placenou licenci pouze na stolní počítač v ordinaci. Z výše uvedeného vyplývá, že při práci v terénu nemám možnost zkontrolovat stav pojištění pacienta, užívané léky dle lékového záznamu, prodělané choroby, alergie, nemůžu vypsat elektronický recept, vydat žádanku na další vyšetření, vydat poukaz na sanitu, neschopenku atd., aniž bych musela vše vypisovat ručně. Záznam z vyšetření si musím poznamenat na papír a po návratu do ordinace vše přepisovat do počítače. Všechny tyto nedostatky odstraňuje webová aplikace, ke které bych se mohla přihlásit z jakéhokoliv mobilního zařízení. Aplikaci jsem vyzkoušela, zadávání informací o pacientovi mi nedělalo problém a systém je funkční*“. (25)

5.2 Dokumentace webové aplikace

Lékař se musí zaregistrovat u podpory, aby získal vygenerované přihlašovací údaje. Na přihlašovací stránce se po obdržení přihlašovacích údajů může lékař přihlásit přes e-mailové přihlašovací jméno a heslo. Po úspěšném přihlášení je přesměrován na stránku seznamu karet pacientů. Na této stránce může lékař vyhledávat pacienty podle příjmení, jména anebo rodného čísla. Také tu může vytvářet kartu nového pacienta, editovat kartu pacienta. Pro smazání karty pacienta lékař musí zvolit editaci karty pacienta a zmáčknout červené tlačítko smazat, pro automatické smazání karty pacienta i s dekursem, anamnézou a historií očkování. Při vytváření karty nového pacienta lékař zadá pacientovy osobní údaje, údaje o pojišťovně, kontakt a zaměstnání. Při vytváření pacientské karty je navigace k dekursu, anamnéze a očkování neviditelná. Po vytvoření pacientské karty se navigace sama objeví. Pokud pacient není z České republiky, nemusí mít rodné číslo. V tomto případě se místo rodného čísla vyplní EČP neboli evidenční číslo pojištěnce nebo datum narození. Pokud je cizinec uprchlík, lékař zaškrtne políčko uprchlík, díky kterému bude vědět, že pacient může být registrován jen na krátkou dobu. Po vyplnění všech údajů lékař zmáčkne tlačítko uložit. Po uložení se přesměruje na editaci pacientské karty. V editaci lékař vidí navigaci k ostatním záznamům pro daného pacienta. V dekursu vidí v horní tabulce, co vše při vyšetření zjistil a co bylo využito k vyšetření. V dolní tabulce vyplní vše, co při aktuální kontrole vyšetřoval, co využil a zda pacient dostal vakcínu, či předepsané léky. V anamnéze pacientské karty v horní tabulce lékař vyplní celou historii, co pacient za život prodělal za nemoci. Do této tabulky při každém novém onemocnění pacienta přidá nakonec seznamu nový záznam o nemoci. V dolní tabulce lékař vyplní veškeré nemoci, které byly v rodině, či stále jsou. V očkování pacienta lékař vyplní veškeré hlavní kolonky o očkování a uloží. Uložený záznam se zobrazí jako historie očkování v levém sloupci. Jednotlivé záznamy historie očkování se dají editovat.

6 Závěr

Cílem této práce bylo vyvinout prototyp webové aplikace pro praktické lékaře s propojenou databází. V prototypu webové aplikace je zprovozněno přihlašování do webové aplikace. Ve webové aplikaci je funkční responzivní design stránek. V seznamu karet pacientů je možno filtrovat pacienty, zakládat, editovat a mazat. Ke každému pacientovi je možnost napsat dekurs, anamnézu a očkování. U očkování je možno vidět, editovat historii aplikovaných vakcín.

Prototyp webové aplikace obsahuje pouze základní funkce, které by v budoucnu mohly být rozšířeny o možnost vypsání receptů léků, vypsání neschopenek a možnost zobrazení jejich historie u jednotlivých pacientů. Po doplnění výše uvedených funkcí by bylo možno zprovoznit prototyp do stavu použitelnosti pro praxi.

Při vývoji webové aplikace a databáze jsem řešil problémy s ukládáním nových záznamů, které nešly uložit, protože cizí klíče byly nastaveny na nenulové hodnoty. Tento problém se řešil vytvořením nové databáze, kde cizí klíče byly vytvořeny s podporou nulových hodnot.

7 Seznam použitých zdrojů

(1) PERSSON, Viktor. 2008. *Úvod, SmartMEDIX, MEDAX Systems, program pro vedení ordinace lékaře* [online]. [cit. 2024-2-28]. Dostupné na internete: https://www.medax.cz/index.php

(2) PERSSON, Viktor, 2008. *Úvod, SmartMEDIX, MEDAX Systems, program pro vedení ordinace lékaře: Cena*. Online. In: Medax.cz. Dostupné z: https://www.medax.cz/. [cit. 2024-02-28].

(3) JIRKŮ, Michaela a Cyril MUCHA. 2013. PC Doktor. *Practicus*. **12**(9-10): 48-51. ISSN 1213-8711.

(4) KOCOUREK, Tomáš. 1999. *DUAS ALTERA* [online]. [cit. 2024-2-28]. Dostupné na internete: https://www.softwareservis.eu/duas-altera/

(5) REINISCH, Radim (2021). Duas Altera [Software]. SoftwareServis. Dostupné z: https://www.softwareservis.eu/duas-altera/

(6) REINISCH, Radim, 2021. *Da-prihlaseni*. Online. In: Softwareservis. Dostupné z: https://www.softwareservis.eu/wp-content/uploads/2021/02/da-prihlaseni.jpg. [cit. 2024-02-28].

(7) REINISCH, Radim, 2021. *Da-kartoteka*. Online. In: Softwareservis. Dostupné z: https://www.softwareservis.eu/wp-content/uploads/2021/02/da-kartoteka.jpg. [cit. 2024-02-28].

(8) REINISCH, Radim, 2021. *Da-zakladni-udaje*. Online. In: Softwareservis. Dostupné z: https://www.softwareservis.eu/wp-content/uploads/2021/02/da-zakladni-udaje.jpg. [cit. 2024-02-28].

(9) REINISCH, Radim, 2021. *Da-dekurz*. Online. In: Softwareservis. Dostupné z: https://www.softwareservis.eu/wp-content/uploads/2021/02/da-dekurz.jpg. [cit. 2024-02-28].

(10) REINISCH, Radim, 2021. *Da-anamneza*. Online. In: Softwareservis. Dostupné z: https://www.softwareservis.eu/wp-content/uploads/2021/02/da-anamneza.jpg. [cit. 2024-02-28].

(11) REINISCH, Radim, 2021. *Da-ockovani*. Online. In: Softwareservis. Dostupné z: https://www.softwareservis.eu/wp-content/uploads/2021/02/da-ockovani.jpg. [cit. 2024-02-28].

(12) REINISCH, Radim, 2021. *Da-erp*. Online. In: Softwareservis. Dostupné z: https://www.softwareservis.eu/wp-content/uploads/2021/02/da-erp.jpg. [cit. 2024-02-28].

(13) REINISCH, Radim, 2021. *Da-epn*. Online. Softwareservis. Dostupné z: https://www.softwareservis.eu/wp-content/uploads/2021/02/da-epn.jpg. [cit. 2024-02-28].

(14) HOLEC, Miroslav. 2021. .NET, .NET Core, .NET Framework, ASP.NET Core aneb jak se v tom všem vyznat. *Miroslavholec* [online]. [cit. 2024-2-28]. Dostupné na internete: https://www.miroslavholec.cz/blog/asp-net-core-mvc-rest-api-net-framework-signalr-odata-blazor

(15) ESPOSITO, Dino a Jaroslav ČERNÝ. 2003. *ASP.NET a ADO.NET: tvorba dynamických webových stránek*. Praha: Grada. Moderní programování, 1. ed. ISBN 80-247-0474-9.

(16) BELLINASO, Marco. 2007. *Webové programování v ASP.NET 2.0: problém, návrh, řešení*. Brno: Computer Press. 1. ed. ISBN 978-80-251-1893-1.

(17) WARREN, Genevieve. 2022. Začínáme s .NET Framework. *Learn.microsoft* [online]. [cit. 2024-2-28]. Dostupné na internete: https://learn.microsoft.com/cs-cz/dotnet/framework/get-started/

(18) SMITH, Steve. 2023. Přehled ASP.NET Core MVC. *Learn.microsoft* [online]. [cit. 2024-1-28]. Dostupné na internete: https://learn.microsoft.com/cs-cz/aspnet/core/mvc/overview?view=aspnetcore-8.0

(19) PETROVAJ, Martin. 2023. Lekce 3 - Úvod do MVC architektury v ASP.NET Core. *Itnetwork* [online]. [cit. 2024-2-29]. Dostupné na internete: https://www.itnetwork.cz/csharp/asp-net-core/zaklady/uvod-do-mvc-architektury-v-aspnet-core

(20) LARKIN, Kirk, Steve SMITH a Brandon DAHLER. 2023. Injektáž závislostí v ASP.NET Core. *Learn.microsoft* [online]. [cit. 2024-2-28]. Dostupné na internete: https://learn.microsoft.com/cs-cz/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-8.0

(21) МИРОШНИЧЕНКО, Евгений, 2019. *Model-View-Controller architectural pattern.svg*. Online. In: Wikimedia. 31.7.2019. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/7/7a/Model-View-Controller_architectural_pattern.svg. [cit. 2024-02-28].

(22) MÁCA, Jindřich. 2018. Lekce 1 - Úvod do Symfony frameworku pro PHP. *Itnetwork* [online]. [cit. 2024-2-28]. Dostupné na internete: https://www.itnetwork.cz/php/symfony/zaklady/uvod-do-symfony-frameworku-pro-php

(23) OLIŠAR, Petr. 2017. Jak funguje Dependency Injection v Symfony a v Nette. *Archiv.pehapkari* [online]. [cit. 2024-2-28]. Dostupné na internete: https://archiv.pehapkari.cz/blog/2017/01/15/jak-funguje-dependency-injection-v-symfony-a-v-nette/

(24) LAURENČÍK, Marek. 2018. *SQL - Podrobný průvodce uživatele: podrobný průvodce uživatele*. Praha: Grada Publishing. Průvodce (Grada). 1.ed. ISBN 978-80-271-0774-2.

(25) TICHÁ, Iva. Praktická lékařka [E-mailové sdělení]. Praha, 21.1.2024

(26) SMITH, Steve. 2023. Zásady architektury. *Microsoft* [online]. [cit. 2024-1-28]. Dostupné na internete: https://learn.microsoft.com/cs-cz/dotnet/architecture/modern-web-apps-azure/architectural-principles

(27) PETROVAJ, Martin. 2023. Lekce 5 - Pohled, middleware a routování v ASP.NET Core MVC. *Itnetwork* [online]. [cit. 2024-2-28]. Dostupné na internete: https://www.itnetwork.cz/csharp/asp-net-core/zaklady/pohled-middleware-a-routovani-v-aspnet-core-mvc

8 Seznam obrázků, tabulek a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam použitých zkratek

Framework je softwarová struktura pro podporu programování, vývoje a organizaci jiných softwarových projektů

CLR (Common Language Runtime)

CIL (Common Intermediate Language)

MVC (Model View Controller)

URL (Uniform Resource Locator)

Injektáž závislostí se používá pro dosažení inverze řízení mezi třídami a jejich závislostmi.

Kontejner je objekt, který zapouzdřuje a sleduje komponenty.

ID – unikátní identifikační číslo

Varchar() – datový typ pro ukládání jakýchkoliv znaků

Boolean – datový typ, který definuje hodnoty true/false

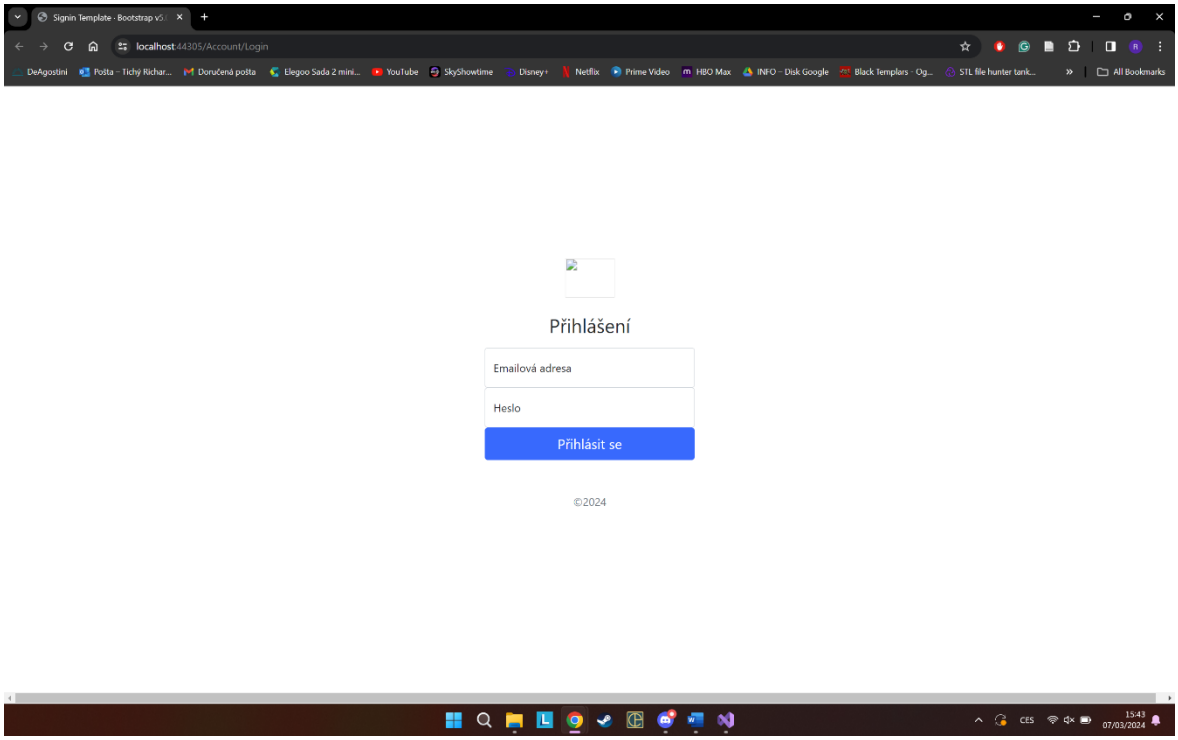
Date – datový typ, který definuje datum

EČP – evidenční číslo pacienta, které je v ČR správním orgánem přiděleno cizincům s dlouhodobým pobytem

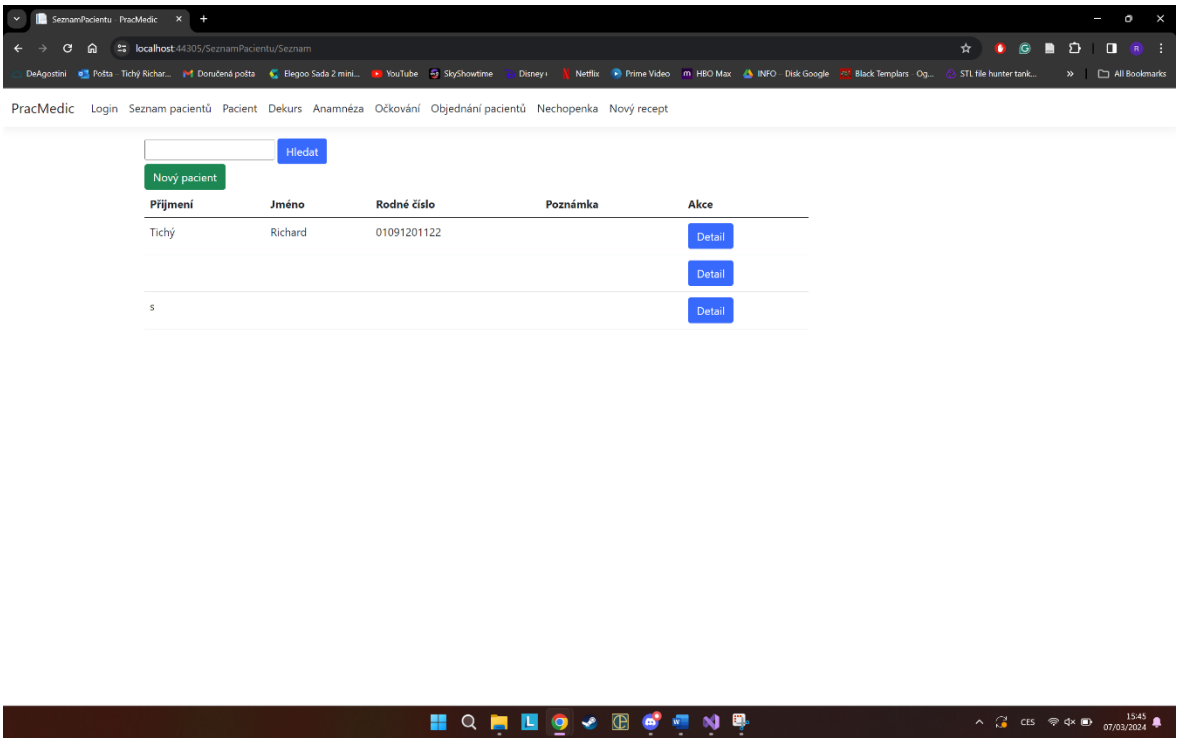
GEA – gastro enterologické ambulance, kde se sleduje stav tlustého střeva u rizikových pacientů

API – aplikační programové rozhraní, které slouží pro předání dat mezi aplikacemi.

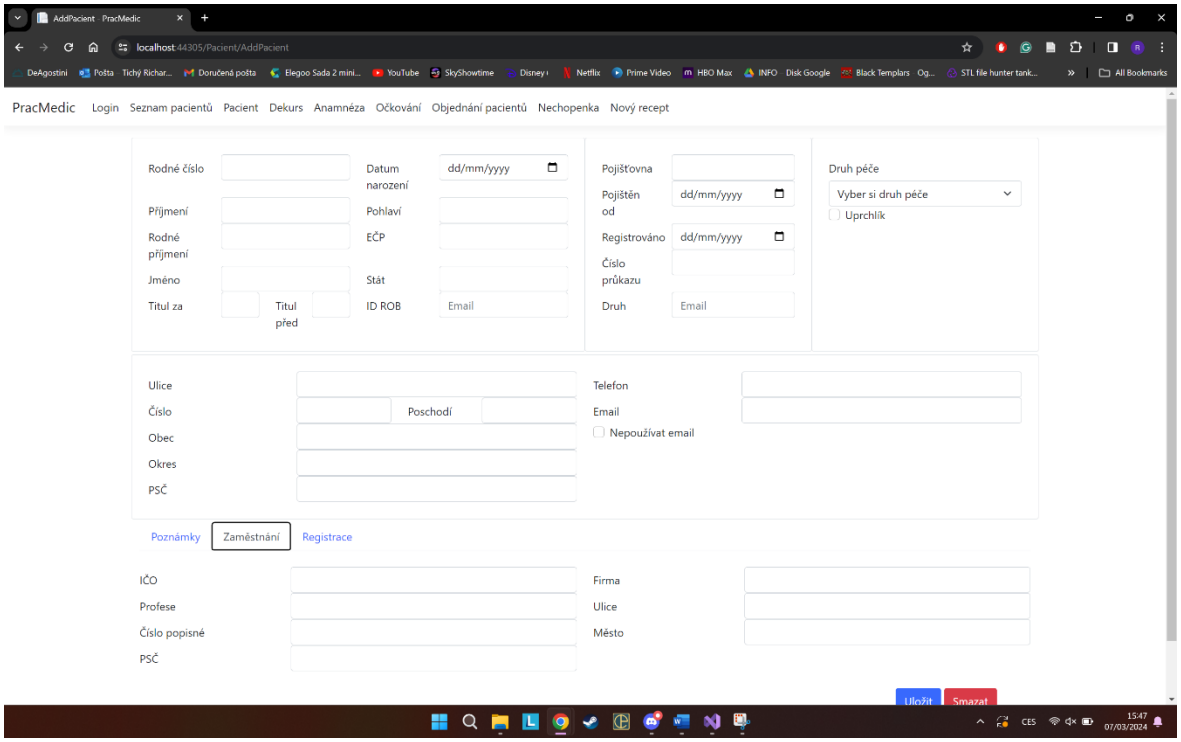
9 Seznam příloh



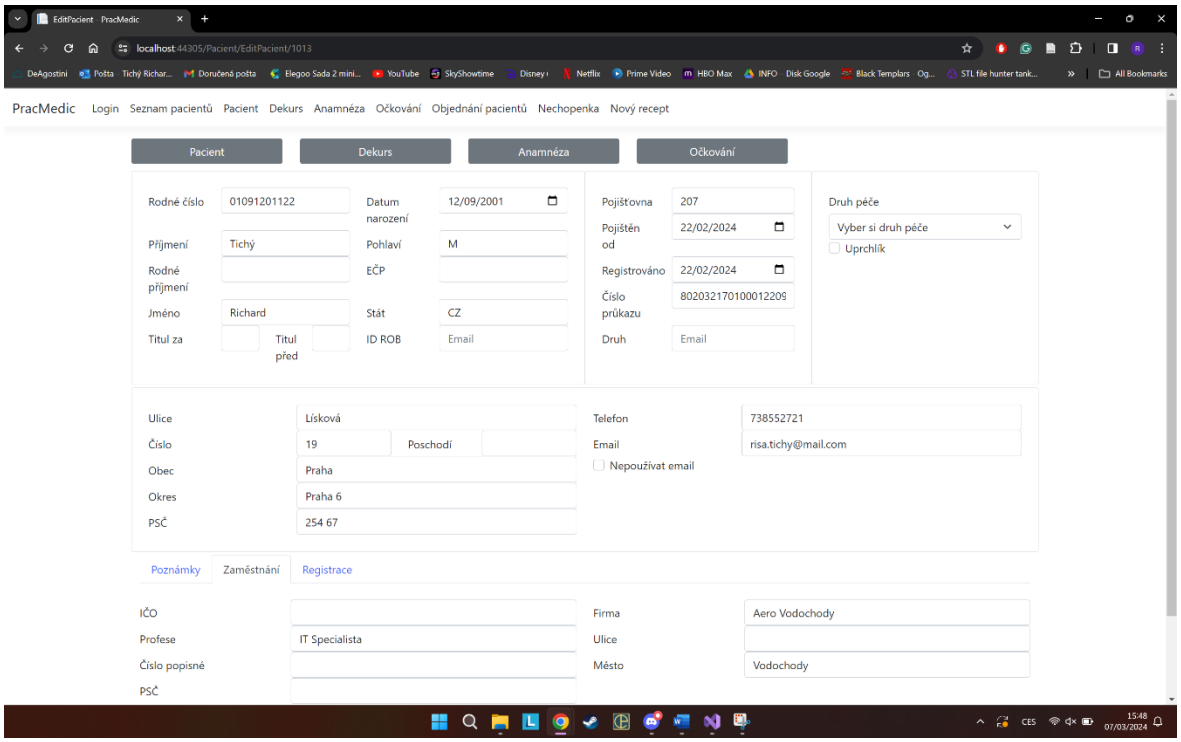
Příloha 2 Přihlašovací okno



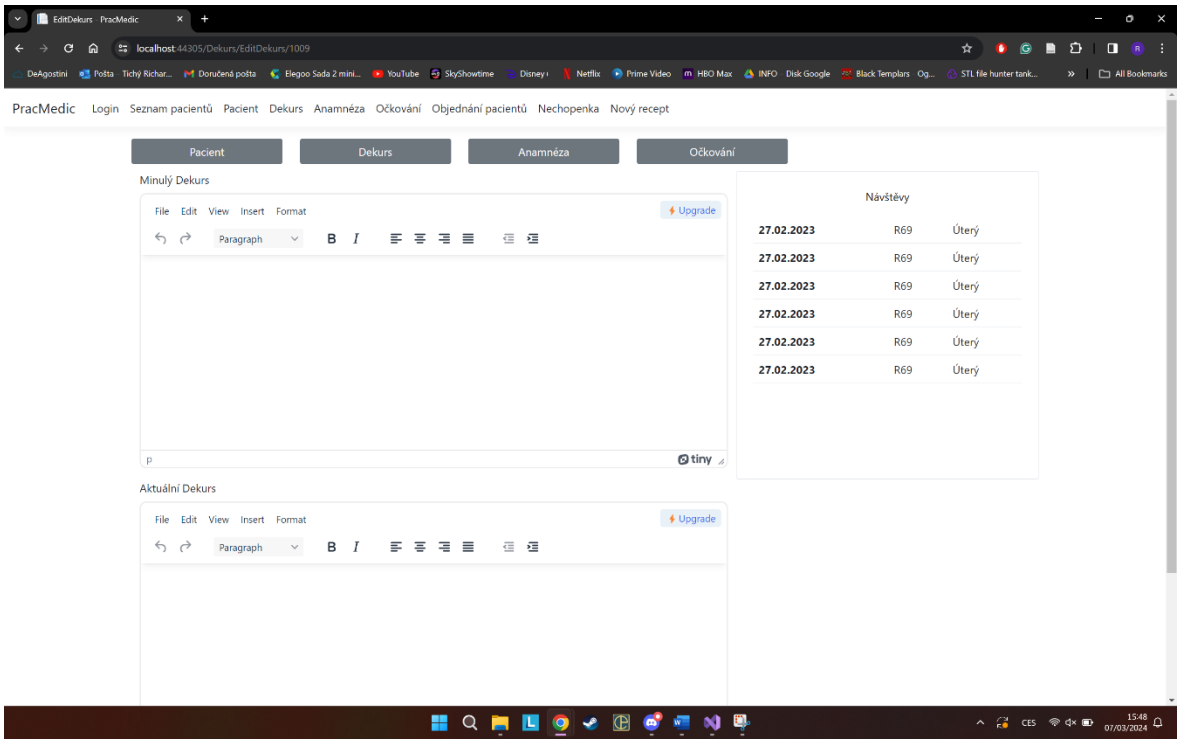
Příloha 1 Seznam karet pacientů



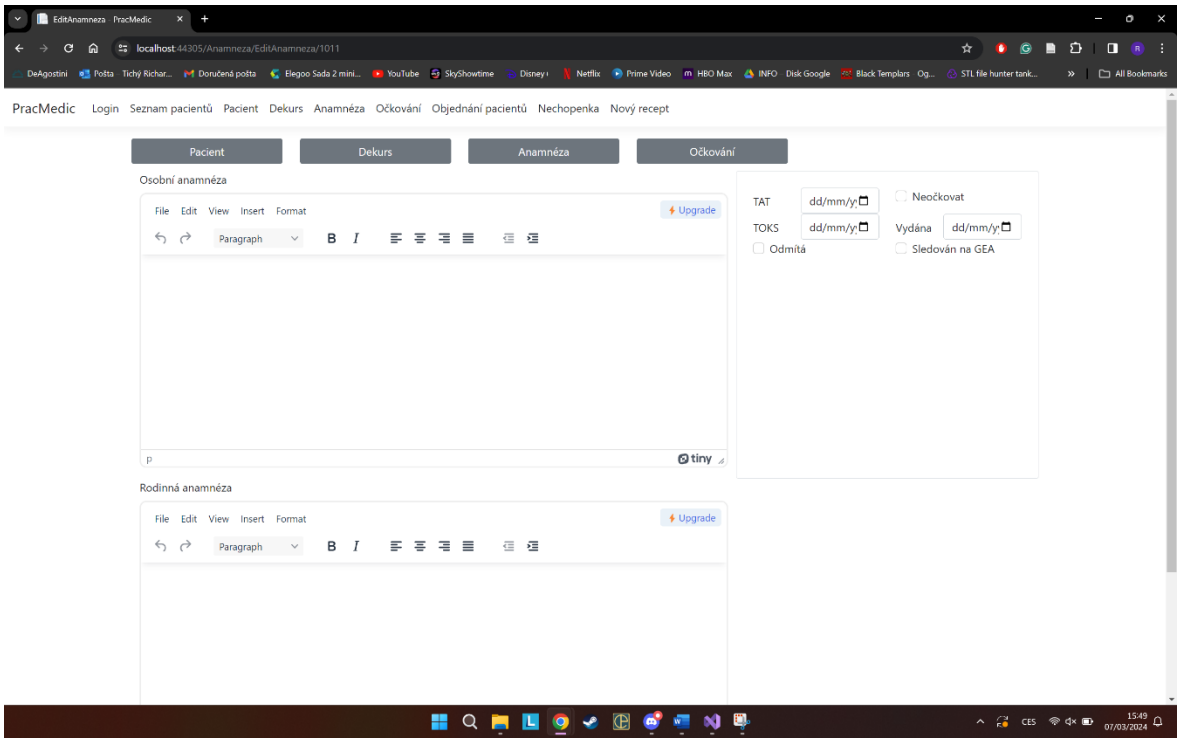
Příloha 3 Přidání pacientské karty



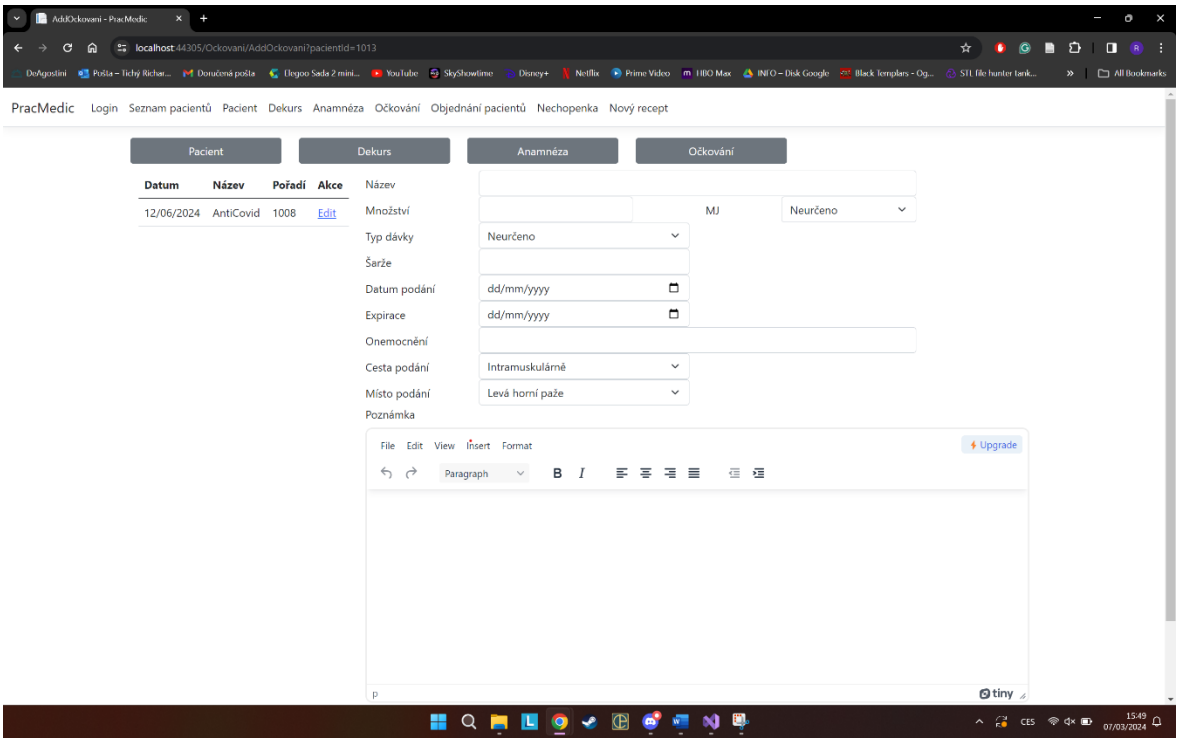
Příloha 4 Editace pacientské karty



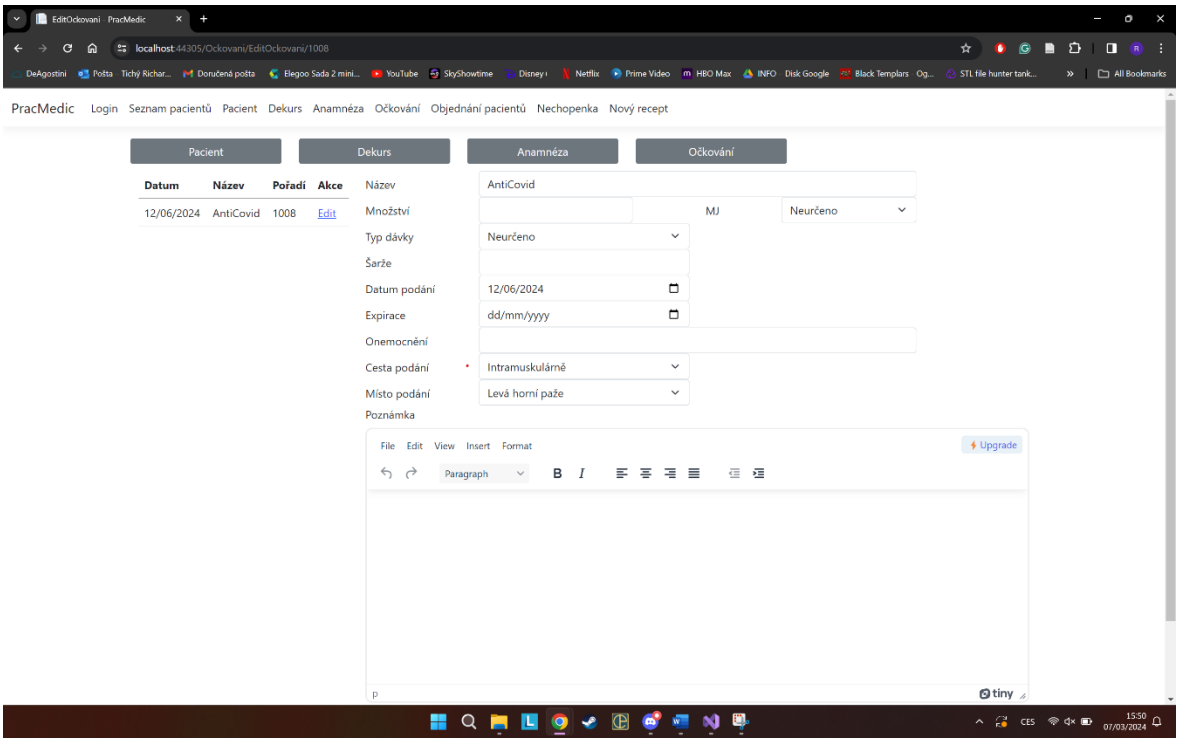
Příloha 5 Záznam dekursu



Příloha 6 Záznam anamnézy



Příloha 8 Přidání záznamu očkování



Příloha 7 Editace záznamů historie očkování