Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Návrh a vytvoření webové aplikace pro digitalizaci rybářských legitimací a pro převod papírové formy zápisu do elektronické

Tomáš Melichar

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Tomáš Melichar

Informatika

Název práce

Návrh a vytvoření webové aplikace pro digitalizaci rybářských legitimací a pro převod papírové formy zápisu do elektronické

Název anglicky

Design and creation of a web application for the digitization of fishing licenses and for the conversion of the paper form of registration into an electronic one

---

Cíle práce

Cílem této bakalářské práce je navrhnutí a následné vytvoření webové aplikace, která bude sloužit k digitalizaci rybářských legitimací a převodu funkce navazující na toto téma do elektronické podoby. Dílčími cíli je představit možnosti vývoje aplikací pomocí ASP.NET core a technologie REACT pro tvorbu front-end části aplikace.

Metodika

Práce se skládá ze dvou částí, teoretické a praktické. Metodika zpracování teoretické části vychází se studia odborných informačních zdrojů. Na základě poznatků z těchto zdrojů budou vytvořeny návrhy pro zpracování praktické části.

Praktická část se bude věnovat zpracování návrhu a implementaci webové aplikace pomocí ASP.NET core a REACT. Aplikace bude sloužit k digitalizaci rybářských licencí a převedení aktuálního papírového zápisu příchodu k vodě do formy elektronické.

Závěrem budou shrnuty poznatky, jež byly získány při tvorbě aplikace. Na základě testování a zpětné vazby budou navrženy možné rozšíření a úpravy této aplikace.

Doporučený rozsah práce

35-40 stran

Klíčová slova

C#, ASP.NET core, informační systém, webová aplikace, React, SQLite, digitalizace rybářských legitimací, Boostrap

---

Doporučené zdroje informací

BANKS, Alex a Eve PORCELLO. Learning React. New Derwent House, 69-73 Theobalds Road, London: O'Reilly UK, 2020. ISBN 1492051721.
https://docs.microsoft.com/cs-cz/dotnet/csharp/
https://getbootstrap.com/docs/5.0/

Předběžný termín obhajoby

2023/24 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 28. 11. 2023

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 9. 2. 2024

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 13. 03. 2024

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci *Návrh a vytvoření webové aplikace pro digitalizaci rybářských legitimací a pro převod papírové formy zápisu do elektronické* jsem vypracoval(a) 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 13.03.2024 _____

Poděkování

Rád bych poděkoval mému vedoucímu Ing. Jiřímu Brožkovi Ph. D za vedení mé práce, a také bych velice rád poděkoval mé přítelkyni Kátě za korekturu a cenné rady.

Návrh a vytvoření webové aplikace pro digitalizaci rybářských legitimací a pro převod papírové formy zápisu do elektronické

Abstrakt

Cílem této bakalářské práce je navrhnutí a následné vytvoření webové aplikace, která bude sloužit k digitalizaci rybářských legitimací a převodu funkce navazující na toto téma do elektronické podoby. Dílčími cíli je představit možnosti vývoje aplikací pomocí ASP.NET core a technologie REACT pro tvorbu front-end části aplikace.

Klíčová slova: C#, ASP.NET core, informační systém, webová aplikace, React, SQLite, digitalizace rybářských legitimací, Boostrap

Design and creation of a web application for the digitization of fishing licenses and for the conversion of the paper form of registration into an electronic one

Abstract

The goal of this bachelor's thesis is to design and subsequently create web application that will serve to digitize fishing licenses and transform the functionality related to this topic into and electronic form. The main objective is the demonstration of application development using ASP.NET Core and REACT technology for creating the front-end part of the application.

Keywords: C#, ASP.NET core, information system, web application, React, SQLite, digitalization of fishing licenses, Boostrap

Obsah

1 Úvod

Poslední roky se rozrůstá digitalizace v mnoha oborech a odvětvích. Elektronická podoba nám velice usnadňuje práci s osobními informacemi o držiteli a jsou díky tomu zároveň chráněny, jsou snadno čitelné, mohou se sdílet a zejména v administrativě jsou pak snadno dohledatelné, což ulehčuje práci z pohledu byrokracie. Má práce se zabývá vytvořením webové aplikace s funkcí ukládání osobních dat rybářů, jež byla doposud zapsána v papírové podobě. Údaje v rybářských organizacích jsou obsahem takzvaných rybářských licencí. Tyto licence jsou pro rybáře nezbytným dokladem, protože bez nich by neměli možnost vykonávat rybářské právo podmíněné rybářským řádem.

Sám se rybolovu věnuji několik let a mohu říci, že zmíněnou problematiku pozoruji již určitou dobu. Z toho důvodu jsem vytvořil návrh pro zjednodušení činnosti spojené se zápisem do rybářského lístku. Webové aplikace mají výhodu spuštění na jakékoliv platformě. Pro tvorbu aplikace použiji ASP.NET core a technologii REACT pro tvorbu front-end části aplikace. Práce se v teoretické části bude věnovat představení jednotlivých technologií, jež se pro tvorbu webových aplikací využívají. Bude vysvětlen princip a využití těchto technologií s možnými výhodami a nevýhodami jejich použití. Teoretická část pokryje většinu částí, které jsou důležité z pohledu tvorby webových aplikací a jejich částí. V praktické části bude popsán návrh a princip fungování aplikace. Nezbytnou součástí budou jednotlivé kroky, které vedly k zhotovení výsledné aplikace. Na závěr budou popsány kapitoly a výsledky, k nimž jsem v průběhu této práce dospěl společně s doloženými přílohami.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem této bakalářské práce je navrhnutí a následné vytvoření webové aplikace, která bude sloužit k digitalizaci rybářských legitimací a převodu funkce navazující na toto téma do elektronické podoby. Dílčími cíli je představit možnosti vývoje aplikací pomocí ASP.NET core a technologie REACT pro tvorbu front-end části aplikace.

2.2 Metodika

Práce se skládá ze dvou částí, teoretické a praktické. Metodika zpracování teoretické části vychází se studia odborných informačních zdrojů. Na základě analýzy těchto zdrojů budou vytvořeny návrhy pro zpracování praktické části.

Praktická část se bude věnovat zpracování návrhu a implementaci webové aplikace pomocí ASP.NET core a REACT. Aplikace bude sloužit k digitalizaci rybářských licencí a převedení aktuálního papírového zápisu příchodu k vodě do formy elektronické. Závěrem budou shrnuty poznatky, jež byly získány při tvorbě aplikace. Na základě testování a zpětné vazby budou navržena možná rozšíření a úpravy této aplikace.

3 Teoretická východiska

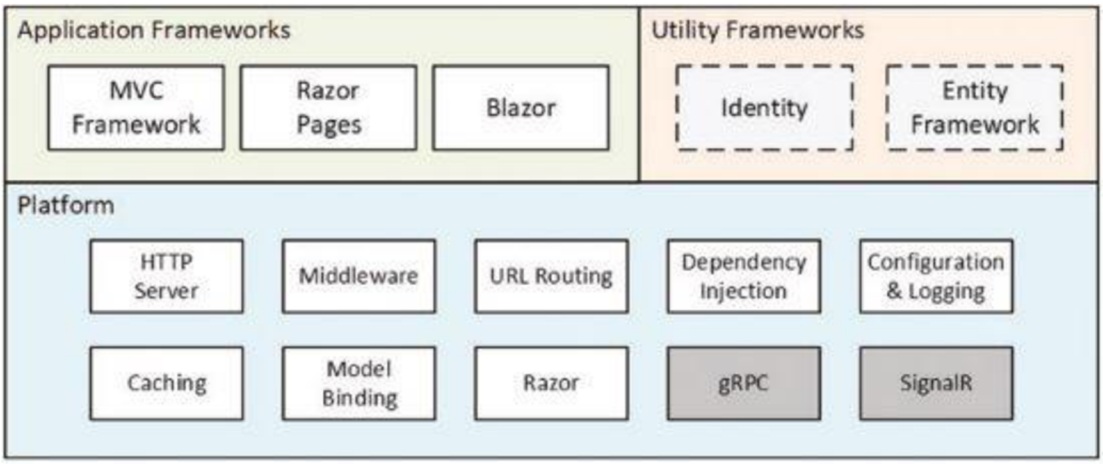
3.1 Technologie a jejich části pro vývoj webových aplikací

Tato kapitola se věnuje obecnému přehledu technologií pro tvorbu webových aplikací a obsahuje nejčastěji využívané technologie. Důležitým bodem jsou architektury vybraných technologií, a to konkrétně open-source frameworku ASP.NET Core. Dále jsou vysvětleny jednotlivé architektury, jejich výhody a následné porovnání těchto architektur mezi sebou. Další část je věnována technologiím a frameworkům pro tvorbu frontendové části aplikace, kde byly vybrány nejpoužívanější z nich. Obsahem této kapitoly jsou také nejznámější vývojová prostředí, přičemž podkapitola je soustředěna zejména na volně přístupná prostřední, která lze pro tvorbu použít. V závěru kapitoly jsou vysvětleny jednotlivé pojmy, jako například informační systém, webová stránka, responsive design.

3.1.1 ASP.NET Core

Společností Microsoft byla v roce 2002 představena platforma ASP.NET, která byla předchůdcem dnešního ASP.NET Core 6. ASP.NET Core je vytvořeno z platformy, v níž se nachází řada hlavních frameworků pro vytváření aplikací a také samotná platforma pro zpracování požadavků HTTP. (Freeman, 2022, s. 3)

Obr. č. 1 - Struktura ASP.NET Core



Zdroj: (Freeman, 2022)

3.1.2 Klíčové vlastnosti

ASP.NET Core je open-source framework, tudíž je přístupný komukoliv, kdo má zájem o tvorbu webové aplikace, nebo by rád vytvořil vlastní projekt s jinou související problematikou. V posledních letech je trendem zpřístupnění softwarů široké veřejnosti zdarma, ale pro vývojáře a tvůrce těchto open-source softwarů je důležité testování a zpětná vazba od uživatelů jejich produktů, díky čemuž dochází k neustálému zlepšování a vyvíjení softwarů. ASP.NET Core nabízí také možnost „cross-platform" umožňující použití různých operačních systémů, na nichž bude samotný framework spuštěn a využíván. Zásadním rozdílem oproti ASP.NET je vyšší výkonnost a škálovatelnost, což umožňuje efektivní vývoj webových aplikací, které jsou obsahově náročné a obsáhlé. (Microsoft, 2021)

3.1.3 Architektura MVC

Velice zásadní součástí ASP.NET Core je MVC architektura, jež byla využívána v původním ASP.NET. Původně byla konstruována pro webové stránky, to vedlo k potížím při přechodu na novější .NET Core z důvodu multiplatformních služeb a v důsledku přibývajících jednostránkových aplikací. Dřívější ASP.NET spoléhal na model zvaný Web Pages, který bohužel ukázal neefektivní a nepraktické využití pro webové projekty kvůli špatné škálovatelnosti těchto projektů. (Microsoft, 2021, s. 5)

Model-View-Controller (MVC) je prototyp zobrazující strukturu dané aplikace. Klade důraz na oddělení jednotlivých částí, díky čemuž je zajištěna správnost zobrazení ostatních méně vyzrálých architektur, na kterých byly webové stránky postaveny. Tento architektonický vzor je rozdělen do tří komponent, jež slouží zejména k rozšiřitelnosti aplikace z hlediska její složitosti. Díky rozdělení je možné jednotlivé části upravovat. Nemusíme tedy v případě, že upravujeme jen jednu část, zasahovat do celého objektu. Tyto tři komponenty nazýváme Model, View a Controller a mají mezi sebou určité vazby působící na funkčnost celkového objektu.

Model

Model zastává dvě hlavní funkce - správu datové struktury, vnitřních dat a obchodní logiku s interními operacemi. Model zásadně ovlivňuje také ostatní komponenty obsažené v architektuře. (Mozilla, 2022) Výše zmíněná návaznost těchto částí se může projevit několika způsoby. Jedním z nich je změna dat, kdy je prováděna změna obsahu, jež je

zobrazen v komponentu View. Při této změně komponent View reaguje zobrazením nových dat, ale pokud nedostačuje aktuální logika, odkazuje se na třetí komponentu Controller, ve které se žádaná logika upraví, nebo vytvoří nová. Samotný Model tvoří základní strukturu vytvořené aplikace a obsahuje informace o datech v ní. (Microsoft, 2022)

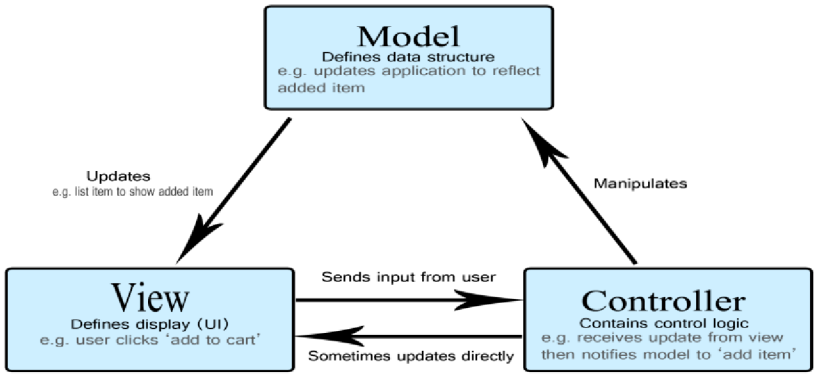
View

Tento prvek má za úkol zobrazit výsledný obsah stránky skrze uživatelské rozhraní. Pohledů existuje několik, záleží na určité funkcionalitě, pro kterou chceme View použít. (Microsoft, 2022) View můžeme nazvat také jako šablonu, jelikož v následné aplikaci máme těchto šablon několik, například šablona informace o uživateli, šablona článku. Nelze však říci, že je pouze šablonou, ale je také výsledným zobrazením výstupu, jež čerpá informace z ostatních prvků. (Itnetwork, 2016)

Controller

Controller řídí komunikaci mezi prezenční vrstvou View a datovou vrstvou Model a mezi samotným uživatelem, jenž k aplikaci přistupuje. (Itnetwork, 2016) Bez tohoto prvku by výsledná aplikace nebyla funkční, jelikož by nedošlo k propojení se zbylými dvěma prvky této architektury, a proto Controller můžeme považovat za pomyslný logický spoj.

Obr. č. 2 – Princip MVC Architektury



Zdroj:(Model View Controller example, 2023)

3.1.4 Architektura MVVM

S touto architekturou se v dnešním světě setkáváme poměrně často, aniž bychom o tom věděli. Se vzorem MVVM se setkáváme zejména při vývoji one-page aplikací založených na frameworku určeném pro uživatelské rozhraní. Nejznámějšími z těchto frameworků jsou React, Vue.js, Angular a další. Díky rozdělení aplikace na více částí je snazší aplikaci oprostit od častých chyb ve vývojovém prostředí a celkově usnadnit testování a vývoj samotné aplikace. Mluvíme zde o oddělení uživatelského rozhraní se samotnou logikou aplikace. Dále je díky tomuto rozdělení kód čitelnější, jak pro vývojáře, tak pro další spolupracující osoby. (Microsoft, 2023)

Stejně jako předešlý architektonický vzor je i MVVM rozděleno na tři části – Model, View a Model view. Každá z těchto částí je určena k odlišné činnosti, avšak tyto části jsou navzájem propojené a komunikují mezi sebou.

View

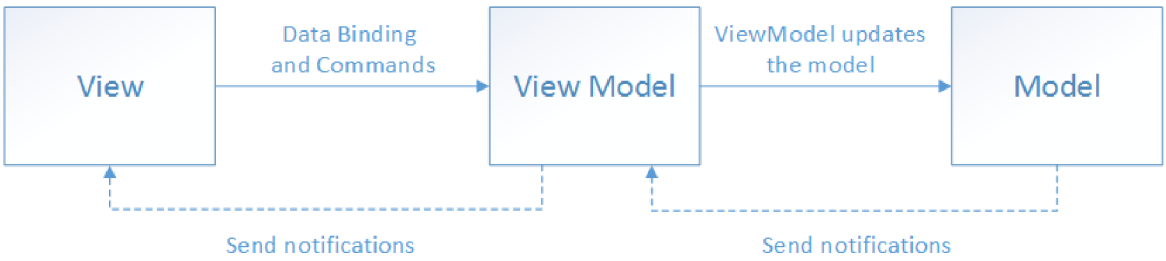
Lze říci, že se jedná o výsledné zobrazení, které jako uživatelé aplikace vidíme a interagujeme s ním. Prezentuje data poskytnutá Viewmodelu uživateli a díky separování logiky aplikace od UI zvyšuje skladebnost a testovatelnost celé aplikace. Neměl by být zatěžován náročnými logickými operacemi, ale naopak by měl být co nejjednodušší z důvodu správného zobrazení dat poskytnutých Viewmodelu a také správnou reakcí na interakce uživatele s aplikací. Obsahuje vizuální části aplikace, například labely, textová pole, tlačítka a jiné komponenty, jež slouží pro interakci s uživatelem. (Stonis, 2022, s. 10)

Viewmodel

Tento prvek je základním stavebním kamenem celé architektury. Slouží jako prostředník mezi zbylými dvěma prvky a obsahuje velmi důležitá data pro fungování celé aplikace. Na rozdíl od předchozího prvku neobsahuje vizuální část aplikace, ale obsahuje logiku a logické operace zakomponované v aplikaci. Zároveň odděluje logiku obsaženou v uživatelském prostředí a logiku samotných dat, přičemž zastupuje velice podstatnou funkci, kdy má za úkol poskytování logiky pro zobrazení dat v uživatelském rozhraní. Jedna z funkcí, kterou také vykonává, je dodržení responzibility aplikace pomocí asynchronních operací. (Stonis, 2022, s. 11) Mezi Viewmodelem a ostatními prvky je nejčastěji použita relace 1:N. Vysvětlujeme ji jako určitý typ vztahu mezi dvěma relacemi.

Také příkazy poskytované Viewmodelem nám určují funkce, jež má uživatelské prostředí nabízet. Díky těmto specifikacím je Viewmodel schopen kombinovat hodnoty dvou odlišných vlastností, kdy jsou tyto vlastnosti následně zobrazeny pohledem. (Microsoft, 2023)

Obr. č. 3 – Princip MVVM Architektury



Zdroj: (Model MVVM, 2023)

Model

Model je jedním z klíčových prvků, který představuje základ funkce celé architektury. Obsahem jsou podstatná pravidla pro manipulaci a práci s daty, ale také samotná data hierarchicky uspořádaná tak, aby udávala logickou část aplikace. Tento obsah je důležitý pro zobrazení dat v uživatelském rozhraní a pro práci s nimi. Základní logika pro načítání, ukládání, aktualizaci a mazání dat je také součástí obsahu Modelu. Díky této vrstvě dochází k lepší organizaci a orientaci v kódu. Na základě těchto vlastností dochází k použitelnosti kódu v dalších částech. Celková testovatelnost a vytváření samostatných testů je snazší díky odloučení logiky od uživatelského rozhraní. Právě odloučení logiky, kdy Model obsahuje pouze logickou část, nikoliv UI, umožňuje zajistit funkčnost a správnost logické části. Platí zde soběstačnost z pohledu architektury. Naopak rozhraní View je plně odkázáno na Modelu z důvodu zobrazení dat a logiky obsažené v rozhraní Model.

Rozdíly architektur MVVM a MVC

Tabulka č. 1 - Porovnání architektur MVC a MVVM

MVVM (Model View Viewmodel)	MVC (Model View Controller)
Je běžně využíván pro vývoj grafického uživatelského rozhraní, kdy usnadňuje vývoj díky oddělení front-end části od back-end.	Je využíván pro vývoj uživatelského rozhraní na základě rozdělení aplikace do tří navzájem propojených částí.
V MVVM reprezentuje Model entity či objekty domény. View je vrstva uživatelského rozhraní, kdežto View model udává spojitost mezi výše zmíněnými částmi Model a View.	V MVC představuje Model data a View představuje uživatelské rozhraní aplikace. Poslední část Controller zpracovává požadavky od zbylých částí aplikace.
Je využíván v nejpoužívanějších front-end frameworcích Vue.js, React a Angular.	Použit v ASP.NET, Java spring a Django.
Snadnější testování díky oddělení logické části od UI.	Různé druhy testů, jež umožňují testování jednotlivých vrstev odděleně bez nutného spuštění aplikace.
Nezávislá práce komponentů, z tohoto důvodu je možné provádět změny jednotlivých komponent bez narušení ostatních.	Podpora asynchronních požadavků, neovlivnění celé aplikace po použití některé modifikace.
Zásadní rozdíl tkví v rozdílnosti komponent. MVVM rozděluje aplikaci na Model, View a Viewmodel.	Zatímco MVC obsahuje komponenty Model, View a Controller.

Zdroj: Vlastní tvorba

3.2 SQLite

Pro vývoj aplikací lze použít několik databázových nástrojů. Vývojář dané aplikace volí nástroj, který použije v návaznosti na framework a použitý jazyk. Jedním z těchto nástrojů je také SQLite. Ten je využíván vývojáři, kteří potřebují nenáročnou a lehce pochopitelnou databázi.

3.2.1 Vlastnosti a výhody databázového nástroje SQLite

SQLite je open-source nástroj, tudíž je zdarma pro jakékoliv účely a lze také využít v projektech, jež jsou komerčně využívány bez následného postihu z právního hlediska. Lze jej použít na různých platformách (Windows, macOS, Linux a operační systémy na mobilních zařízení). Jednoduchý design a ukládání dat do jednoho souboru je po uživatele tohoto nástroje velice přínosné díky následné orientaci v databázi. Jeho nenáročné vlastnosti přispívají k rychlosti a efektivitě využití SQLite. Velkou výhodou je jednoduchá instalace, po které není potřeba složité nastavení nebo konfigurace serveru. (About SQLite, 2023)

SQLite je řazena mezi relační databáze, proto stejně jako ostatní relační databáze ukládá data do tabulek s návaznostmi mezi nimi a také podporuje většinu SQL dotazů. Data jsou ukládána do tabulek, ty následně vytváří schéma, jež má nastavenou určitou hierarchii a uspořádání tabulek. V každé tabulce jsou obsaženy sloupce, v nichž se nachází určité typy dat. (About SQLite, 2023)

3.2.2 Architektura a práce s daty v databázi

SQLite je jednosouborový systém. Tato vlastnost nám říká, že jsou veškeré tabulky a data v nich uložená v jednom souboru, což má značný vliv na přenos, zálohování a upload databáze, jelikož ulehčuje všechny tyto zmíněné úkony. Díky multiplatformnímu formátu výsledného souboru, do něhož jsou data uložena, má uživatel možnost kopírovat tento soubor mezi libovolnými 32 a 64bitovými systémy, nebo také mezi architekturami big a little-endian. Veškeré tyto přenosy jsou typu ACID, tento typ přenosu nám zajišťuje platnost dat i přes obsažené chyby. Pro práci s daty je využit SQL. Využívá základní operace, kterými jsou například *SELECT*, *INSERT*, *UDPATE* nebo *DELETE*. Podporuje také indexovací operace významně zrychlující vyhledávání výsledného indexu v tabulce a dat. (About SQLite, 2023)

3.3 Vývojové prostředí

Vývojové prostředí je místo, ve kterém vzniká aplikace či jiný produkt v softwarové podobě. Slouží zejména pro vývojáře k vývoji aplikací, přičemž vývojové prostředí zjednodušuje tuto práci a umožňuje například testování.

3.3.1 Užitečnost vývojových prostředí pro vývojáře

Vývojová prostředí přináší určité výhody. Díky automatické konfiguraci softwaru není vývojář nucen umět ovládat nástroje a konfigurace daného softwaru. To umožňuje přímou tvorbu aplikace bez nutnosti přípravy softwaru a prostoru pro danou aplikaci. Velmi přínosná je automatická úprava kódu, která udržuje určitá pravidla a strukturu dat, díky čemuž je kód konzistentní a nedochází k většímu množství chyb v syntaxi. V samotné syntaxi bývají určité části kódu zvýrazněny tučným písmem nebo kurzívou. To zejména slouží ke zlepšení orientace v kódu. V posledních letech se ve vývojových prostředí objevilo inteligentní doplnění kódu, které má vývojář možnost využívat, ale není to striktní a lze tedy tuto funkci vypnout. Je to užitečné z hlediska úspory času a zároveň je syntax díky této funkci bezchybná. (DevX, 2023)

Refaktoring neboli „restrukturalizace“ zdrojového kódu zefektivňuje a dělá kód čitelnější, což je z hlediska pohledu vytváření složitých aplikací například ve firmách velice užitečné, jelikož tento proces napomáhá k čitelnosti kódu pro ostatní vývojáře podílející se na daném projektu. (Techopedia, 2022)

3.3.2 Druhy vývojových prostředí

Vývojové prostředí je hlavním nástrojem pro vytvoření projektu. Možností, jak vývojové prostředí použít, je mnoho. Existuje však několik prostředí, jež vídáme nejčastěji, například ve školních počítačích. Každý uživatel má možnost výběru, což umožňuje vybrat prostředí, které splňuje jeho požadavky. Asi nejzásadnějším kritériem, skrz které bude prostřední vybráno, je podpora jazyka. Programovacích jazyků je v dnešní době mnoho, ale většina vývojových prostředí podporuje zejména ty, jež jsou používány nejčastěji.

Vývojové prostředí má samo o sobě určitou hierarchii, díky níž se uživatel v prostředí orientuje, avšak u většiny prostředí je tato hierarchie rozdílná, a proto si uživatel vybere to prostředí, které mu vyhovuje nejvíce. Díky těmto možnostem neexistuje ve vývojových prostředí monopol a využívá se jich hned několik. (DevX, 2023)

3.3.3 Nejznámější vývojová prostředí

Mezi nejznámější vývojová prostřední patří například Visual Studio Code, Visual Studio, IntelliJ IDEA, NetBeans, Eclipse. Tato vývojová prostředí jsem vybral z důvodu použití některého z nich v mé práci, ale také z důvodu využití v dnešní době.

Visual Studio Code

Základní specifikací vývojového prostředí je, zda podporuje platformu, na které chceme toto prostředí využívat. Visual Studio Code umožňuje multiplatformní využití, tudíž je možnost toto prostředí použít na macOS, Linux a samozřejmě na Windows. Mezi hlavní výhody VS Code patří funkce nesoucí název IntellliSense. Tato funkce napomáhá uživateli zejména zrychlit vývoj díky automatickému doplnění aktuálně psaného řádku. Samotné VS Code je jednoduchý editor zdrojového kódu s moderními a velice užitečnými nástroji pomáhající uživateli v mnoha ohledech. Lze jej tedy využívat ke každodenní práci na projektech. Funkce, jež je velice přínosná pro většinu uživatelů, je takzvaný Debugger. Umožňuje kód procházet po částech a kontrolovat jeho správnost. Další výhodou je podpora Gitu, není tedy potřeba při vývoji editor opouštět a vytvářet změny v samotném Gitu, ale právě díky zmíněné podpoře je možnost tyto změny provádět v samotném prostředí a poté je ukládat na Git. Velkou výhodou je, že VS Code je bezplatný a bez jakýchkoliv následných poplatků. (Microsoft, 2023)

NetBeans

Dalším OpenSource a multiplatformním vývojovým prostředním je NetBeans. NetBeans je rychlým a užitečným editorem obsahující několik nástrojů, jež umožňují snadné redaktorování kódu a samotnou opravu. Důležitými prvky jsou také editory a šablony, díky nimž se snadněji vytvářejí aplikace zejména v jazyku Java a PHP, ale samozřejmě i v jiných jazycích. (The Apache Software Foundation, 2023)

Eclipse

Vývojářská skupina Eclipse Foundation vytvořila projekt s názvem „Eclipse“ pod záštitou vytvoření multiplatformního vývojového prostředí s možností využití jakéhokoliv jazyka. Hlavní výhodou samotného Eclipse je možná rozšiřitelnost, kdy je možnost využít stovky tisíc modulů s novými funkcemi, které mohou být při tvorbě projektu užitečné a ušetří mnoho času. Eclipse disponuje zejména širokou komunitou a díky této vlastnosti lze v relativně krátkém čase zajistit opravu chyb. Nejvíce je využíván v oblasti vývoje Java aplikací určených pro ladění. Součástí je také nástroj, jež napomáhá organizaci práce a sledování změn v kódu. Tento nástroj se nazývá Mylyn. Umožňuje také sledování vaší práce a na základě dosavadního postupu je schopen uživatele odkazovat na informace, které by pro něho mohly být užitečné. (Eclipse, 2023)

3.4 Informační systém

Dá se říci, že informační systém je komplexní soubor obsahující určité prvky, jež ho tvoří. Tyto prvky jsou klíčové pro fungování systému jako celku. Mezi základní prvky patří hardware, na kterém je systém vytvořen, ale také hardware, na základě něhož mohou uživatelé k informačnímu systému přistupovat.

Serverový hardware má na starosti ukládání dat od uživatelů, přičemž tato data uchovává od všech programů a aplikací, jež jsou v systému obsaženy. Důležitý je také přístup, díky němuž má uživatel možnost přístupu odkudkoliv, ať už v samotné organizaci či mimo ni.

Další částí je software. Na softwaru závisí celý informační systém, jelikož pro každý informační systém je velice podstatný operační systém, kde jsou operace spuštěny. Softwarová část se nevěnuje pouze operačním systémům, ale také například souborům, do nichž se ukládají velmi cenná data, jež jsou získávána od uživatelů. Data uživatelů, jsou velmi zásadní pro informační systém, jelikož mohou vypovídat několik věcí, kupříkladu orientaci uživatele v systému, nebo s jakou částí měl problém při zpracování úlohy a podobně. Data tedy fungují rovněž jako zpětná vazba pro správce systému a díky nim lze opravit menší nedostatky v systému. (Techtarget, 2023)

3.4.1 Druhy informačních systémů

Informační systémy jsou využívány k mnoha účelům, a proto jsou rozděleny do určitých kategorií podle využití a vlastností, které mají splňovat. Na základě toho mohou podniky přetvářet funkce ve firmě pomocí výsledků z jednotlivých informačních systémů. Informační systém orientovaný na manažerskou činnost slouží zejména k získávání dat z ohodnocení jednotlivých služeb podniku. Jednotlivé služby či úkony lze tak zlepšit, nebo upravit pro dosažení lepších výsledků.

Další typ informačního systému souvisí s předchozím typem. Existují výkonné informační systémy, díky nimž mohou vysoce postavení pracovníci pracovat s analytickými a jinými nástroji obsaženými v systému. Ty pak slouží k vyhodnocení strategie obchodu či jiné bezprostřední analýze pro udržení produktivity a financí firmy. To se však neobejde bez podpory systému pro rozhodování. Systém pro podporu rozhodování napomáhá sledovat aktuální stav jednotlivého zboží. V aktuálním stavu systém rozlišuje jednotlivé faktory. Nejzásadnějším faktorem je prodej či neprodej daného produktu. Díky této informaci jsou majitelé či prodejci schopni správně určit velikost zásob na skladě, nebo vyřazení málo prodávaného produktu.

Dalším druhem je systém pro zpracování transakcí, který má za úkol podporu provozních procesů ve firmách. Je často využíván například v bankovnictví nebo při určitém zpracování transakcí ze strany zákazníka. (Techtarget, 2023)

3.4.2 Správa a procesy informačních systémů

Jednotná správa a údržba operačních systémů zajišťuje, aby byl systém spolehlivý a pro uživatele jednoduchý na práci a orientaci v něm. Správu a údržbu většinou vykonávají samotní správci dané sítě, kteří se v systému velice dobře orientují a jsou schopni systém vylepšovat podle nových technologií na trhu. Je také možnost přenechat správu systému externí firmě, například z důvodu složitosti. Tyto služby nabízí mnoho firem, většinou se starají o více systémů najednou a mají s technologiemi spojenými se systémem bohaté zkušenosti.

Nezbytnou součástí je správa dat, díky níž dochází k zajištění konzistence a integrity dat. Od samotného správce či firmy je nastavena také určitá záloha systému zajišťující, aby firmy o data v případě ztráty nepřišla.

Podstatná je také síťová infrastruktura. Zajišťuje správný chod celé sítě, po které data proudí a jsou přenášena mezi zařízeními v samotné síti. Správci mají na starost správnou konfiguraci jednotlivých zařízení, aby nedošlo k chybám, jež by mohly vést například ke ztrátě dat. (Techtarget, 2023)

3.4.3 Zabezpečení systému

Jednou z často opomíjených věcí je zabezpečení systému. K prevenci před možnými hrozbami a riziky, je nutné, aby správce učinil určité bezpečnostní kroky, díky nimž bude systém a data v něm bezpečně uložená. Častým řešením bývá udělení přístupu pouze osobám působícím v systému, a mají tak pravomoc k přístupu. Dalším krokem pro zabezpečení bývá antivirový software. Antivirových softwarů existuje mnoho a je tedy pouze na zvážení správce, který zvolí. Důležitá je průběžná kontrola zabezpečení za určité období. Kontrola dokáže odhalit takzvané „trhliny“ v zabezpečení vedoucí k ohnisku případného problému. I přes provedení všech těchto opatření není možné říci, že je systém stoprocentně chráněn proti všem hrozbám. Stejně tak jako se vyvíjí systémy, přibývají nové hrozby a viry. (Techtarget, 2023)

3.5 Webová stránka

Existuje mnoho webových stránek, avšak každá je v některých částech odlišná. Může se jednat tematicky o stejnou stránku, ale může obsahovat odlišné informace, či se lišit vzhledem. Každá webová stránka má určitou strukturu a určitý obsah. Obsahem nejčastěji bývají texty, obrázky, videa, formuláře, reklamy, ale také například interaktivní prvky k upoutání pozornosti.

3.5.1 Struktura a dostupnost webových stránek

Struktura je základním stavebním kamenem webové stránky. Udává, jak webová stránka bude vypadat, a jaký obsah se bude zobrazovat v její určité části. Nejčastěji se lze setkat s termíny jako header, body, main a footer. Je to základní rozdělení stránky, které předem říká, v jaké části stránky bude obsah zobrazen. Přístupnost webové stránky zajišťují další části jako navigace nebo menu, pomáhající uživatelům v orientaci na dané stránce. Dále jsou zde hypertextové odkazy, díky nimž je uživatel schopen přistupovat z jedné části stránky na další. (Mozilla, 2023)

3.5.2 Dostupnost

Přístup k webové stránce je ovlivněn dostupností internetu v zařízení uživatele. Rychlost zobrazení a vykreslení stránky je ovlivněno skrze rychlost a velikost přenosu dat ze strany uživatele. To, že se webové stránky vůbec zobrazí, je také způsobeno webovým hostingem. Webový hosting funguje na principu klient-server, kdy uživatel přistupuje k webové stránce uložené na serveru. Samotný hosting tedy umožňuje přístup ke stránce uložené na serveru. Velmi podstatnou částí je také protokol TCP/IP, jež řadíme mezi komunikační protokoly, jinými slovy zajišťuje přenos dat v síti. (Mozilla, 2023)

3.6 Webová aplikace

Na rozdíl od webové stránky, není obsah webové aplikace stejný, ale mění se v závislosti na interakci s uživatelem. Je to typický příklad dynamické stránky, kdy se po určitých krocích změní obsah a teprve ten se dá označit jako konečný. Webových aplikací lze v dnešní době najít spousty a denně s nimi pracuje velké množství osob. Mezi nejvíce zastoupené patří e-shopy a sociální sítě.

3.6.1 Výhody webových aplikací

Díky responzivnímu designu současných aplikací a webů je možné zobrazení na jakémkoliv zařízení (například i TV). Velmi zásadní je připojení k datové síti, jež nám umožňuje přístup téměř odkudkoliv. Velikou výhodou je vzdálený přístup bez nutnosti instalace dané aplikace, stejně je to i u aktualizací. Stačí vlastnit aktuální verzi webového prohlížeče. Aktualizace samotné aplikace se totiž řeší přes centrální servery, na nichž se nachází. Kapacitní nenáročnost aplikace je pro běžného uživatele výhodou, protože aplikace není uložena v paměti zařízení a nezabírá žádnou paměť, jak tomu bývá u běžných aplikací. (Adobe, 2021)

3.6.2 Možné nevýhody webových aplikací

Výhoda možnosti přístupu odkudkoliv díky internetovému připojení má i svá úskalí. Internetové připojení nemusí být vždy dostatečné a webovou aplikaci pak není možné načíst. Většina operací ve webové aplikaci vyžaduje neustálé internetové připojení, a proto je off-line přístup neefektivní a uživatel nemá možnost jakkoliv v práci postupovat. Problém pro zobrazení a užívání aplikace mohou představovat také hardwarové části zařízení využívané

při použití aplikace. Aplikace nemusí vždy podporovat zobrazení na starších zařízeních z důvodu nedostatečného výkonu či kvůli jiné softwarové vadě. Aplikace mohou mít rovněž problémy s některými z novějších části hardwaru uživatelského zařízení, s nimiž neumí spolupracovat, nebo nepodporuje toto zařízení. (Adobe, 2021)

3.7 Responzivní design

Termín responzivní design vznikl v roce 2010, jeho tvůrcem byl Ethan Marcotte. Vznikl z důvodu multiplatformního zobrazení webových stránek a aplikací. Napomáhá automaticky přizpůsobit obsah stránky velikosti okna, ve kterém danou stránku, aplikaci či jiné zobrazujeme. Cílem tohoto přístupu je tedy umožnit uživateli zobrazit daný obsah na jakémkoliv zařízení, ať už je to mobilní telefon, tablet, notebook nebo jiné zařízení. Nejčastěji je responzivní design spojován s CSS, ale zajištění responzivity je možné několika způsoby. (Mozilla, 2023)

3.7.1 Zajištění responzivity

Zajištění responzivity lze provést několika způsoby. Záleží na druhu konkrétního projektu. Při tvorbě komerčních aplikací je často požadováno vytvářet jedinečné komponenty a styly kvůli autorským právům a jedinečnosti dané práce a jejího vzhledu.

3.7.2 Media Queries

V CSS se můžeme setkat s definováním vlastního stylu pro určité zařízení, v němž je potřeba responzivitu zajistit. Zde se využívá CSS media queries, jež nám zajištuje možnost definovat styl pro daný komponent. (Mozilla, 2023)

3.7.3 CSS Frameworky

Pokud však není nutno vytvářet vlastní styl pro určitý komponent v projektu, je možné použít některý z CSS frameworků obsahující předem definované styly. Mezi nejznámější frameworky patří například Bootstrap nebo Marerial-UI. Definované třídy a komponenty, jež jsou obsahem těchto frameworků, jsou plně responzivní, a tudíž je lze využít pro multiplatformní aplikace. Lze je využít v projektech založených na React či jiných front-end programech. (BrowserStack, 2023)

3.7.4 Knihovny a komponenty

Každý tvůrce si vytváří své komponenty nejčastěji podle zvoleného obsahu. Lze však použít knihovny a balíčky pro určitý program, které nám velice usnadní práci díky možnosti výběru komponent odpovídajícím podmínkám daného projektu. Balíčky a knihovny jsou obsaženy v samotném programu, ale je nutné služby importovat, jinak nebude nastavení dané komponenty funkční kvůli nedostatečným informacím. (Mozilla, 2023) Programy nabízí pro kontrolu možnost otevření obsahu v předem definovaném zobrazení. Můžeme například zvolit určitý model mobilního zařízení, kde bude obsah zobrazen, a vývojář je schopný provést kontrolu, zda je obsah vygenerován správně. (BrowserStack, 2023)

3.8 Azure DevOps

Dalším z velmi užitečných nástrojů pro tvorbu větších projektů, zejména ve firmách, je služba Azure DevOps. Byla vyvinuta firmou Microsoft a neustále dochází k jejímu vylepšování a rozšiřování funkcí. Obrovská výhoda opět spočívá v neomezeném přístupu odkudkoliv, jelikož Azure DevOps je dostupný neomezeně z webového rozhraní. Díky možnosti přístupu více osob najednou, napomáhá zejména k rychlejšímu dosažení výsledků. Obsahuje několik velice zásadních služeb, které samotný DevOps tvoří. Nejvýznamnější z nich jsou Azure Boards, Azure Repos, Azure Pipelines. (Microsoft, 2024)

3.8.1 Azure Boards

Součástí DevOps je mimo jiné také Azure Boards. Je vedena jako samostatná webová služba. S její pomocí má uživatel možnost s již vytvořeným týmem plánovat a sledovat části daného projektu. Výhodou je rozdělení jednotlivých úkolu napříč celým projektem, díky čemuž je práce na projektu lépe organizovaná a usnadňuje také potřebné kontroly jednotlivých úkonů. Azure Boards funguje jako komunikační kanál pro tvorbu jednotlivých částí projektu. (Microsoft, 2024)

3.8.2 Azure Repos

Při tvorbě větších projektů dochází často k problémům v organizaci a postupu tvorby. Důležitým aspektem při tvorbě programu je verzování. Verzování rozděluje projekt na menší části. Azure Repos tyto kroky splňuje a umožňuje uživatelům zpětnou vzájemnou kontrolu napříč celým kódem. Základním principem je rozdělovat kód na menší části, verze, přičemž

každou část může spravovat jiný člen projektu. Tyto části jsou v průběhu času ukládány a následně spojeny do jednoho celku. Výhodou je, že rozdělení verzí zůstává stejné. I po sjednocení lze hledat samotné chyby v jednotlivých částech, nikoliv v celém projektu. (Microsoft, 2024)

3.8.3 Azure Pipelines

Další důležitou částí DevOps je Azure Pipelines. Tato část zajišťuje umístění kódu do systému správy verzí. DevOps podporuje dvě správy verzí, kam patří již zmínění Azure Repos, ale také Git. Mimo jiné automaticky zajišťuje a vytváří testování jednotlivých částí kódu a napomáhá v rozpoznání chyb v něm. Podporuje většinu programovacích jazyků, tudíž není tato funkce omezena jazykem. Azure Pipelines obsahuje dva procesy, jež pomáhají k opravám jednotlivých částí kódu a zvýšení kvality použitelnosti v konečné fázi projektu.

Prvním procesem je průběžná integrace neboli „CI“. Součástím tohoto procesu jsou automatizované testy, které týmy vytvářející daný program využívají ke kontrole a zachycení jednotlivých chyb po sjednocených samostatných částí projektu. Tento proces zlepšuje kvalitu samotného kódu a podílí se na jeho výsledné použitelnosti.

Dalším procesem je nepřetržité doručování, ve zkratce je název udáván jako „CD“. Nepřetržité doručování je taktéž využíváno pro testování, ale předchází krokem sestavení kódu do jednotné podoby. Výsledný kód je následně nasazován do více prostředí. V jednotlivých prostředích je testován za účelem zvýšení kvality a použitelnosti ve specifických prostředích a situacích. Díky zmíněnému procesu dochází k vytváření nových verzí a k provedení oprav verzí předešlých. Důsledkem je tedy nepřetržitá kontrola a oprava jednotlivých kroků, kdy ve výsledku vznikne finální „odladěný“ kód. (Microsoft, 2023)

4 Vlastní práce

Jako téma mé bakalářské práce jsem zvolil digitalizaci rybářských licencí. Pro vytvoření aplikace jsem zvolil vývojové prostředí Microsoft Visual Studio Code, ve kterém budu pro vývoj této aplikace využívat ASP.NET Core v aktuální verzi .NET 8.0 ve spojení s React. V tomto projektu bude tedy pro front-end část použit React a pro back-endovou část bude použit jazyk C#.

Na digitalizaci rybářských licencí se dívám z pohledu Rybářského svazu. Obsahem této aplikace je přihlášení do aplikace z pohledu administrátora nebo běžného uživatele. Administrátor má plnou kontrolu a zobrazují se mu všechny funkce uvnitř aplikace, mezi nimiž je možnost registrace nového uživatele do systému a registrování licence spojené s daným uživatelem. Dále je v aplikaci zakomponována editace nebo zneplatnění samotného uživatele či lístku. Uživatel by měl mít po přihlášení možnost náhledu pouze na samotnou licenci s jeho osobními údaji.

4.1 Návrh aplikace

Aplikace bude tvořena pomocí back-end části za využití programovacího jazyka C#, který je obsažen ve frameworku ASP.NET Core ve verzi 0.8. Architektura celého projektu bude postavena na architektuře MVC. Podstatné je samozřejmě také ukládání samotných dat do databáze, přičemž pro tento účel bude použit databázový nástroj SQLite, jehož výhodou je, že působí přímo v samotném Visual Studiu Code a tabulky obsažené v databázi společně s daty v nich jsou uložené na lokálním serveru. Lokální server je pak možno otevřít a zkontrolovat pomocí DB Browser pro SQLite.

4.2 Funkce aplikace

Uživatele rybářské licence zajímají zejména dva faktory. Prvním je zobrazení licence, aby mohla být použita při kontrole, a druhým faktorem je aktivní zápis licence neboli platnost. Pro běžného uživatele je podstatné zobrazení jeho licence obsahující pravdivé a přesné osobní údaje spolu s platností. První potřebnou funkcí je přihlášení. Po autorizaci bude uživateli umožněn vstup do aplikace, v níž bude zobrazen rybářský lístek s jeho osobními údaji.

Pro administrátora bude aplikace obsahovat další funkce – registrace nového uživatele, registrace rybářské licence a možnost zobrazení uživatelů a jejich licencí s možností úpravy nebo odstranění údajů.

4.2.1 Přihlášení uživatele

Pro vstup do samotné aplikace je nutné přihlášení se zaregistrovaným účtem. Druhy přihlášení jsou ale odlišné, protože jsou využity dvě různé role – administrátor a běžný uživatel. Pro přihlášení je využit e-mailový účet uživatele a jím zvolené heslo. Špatné zadání hesla či e-mailu by mělo spustit chybovou hlášku, která uživatele upozorní na opravu daného údaje. Pokud přihlášení proběhne bez chyby a údaje budou zadány správně, uživatel bude automaticky přesměrován na úvodní stránku aplikace.

4.2.2 Zobrazení licence

Na domovské stránce aplikace bude mimo jiné zobrazena samotná licence přihlášeného uživatele. Toto zobrazení bude platné pro všechny uživatele v systému. Licence by měla být zobrazena jako kontejner, v němž jsou zapsány údaje uživatele z původní papírové verze licence.

4.2.3 Registrace uživatele

Pro umožnění vstupu více uživatelům bude potřeba registrace. Funkci registrace může využívat pouze administrátor, díky tomu se předejde případné duplicitě a bude zajištěna správnost zadání. Pokud bude mít uživatel zájem o registraci, musí nejprve splnit podmínky pro možnost registrování jeho licence na příslušném oddělení. S nimi bude zaregistrován a bude je využívat ke vstupu do aplikace. Těmito údaji jsou e-mailová adresa a zvolené heslo. Údaje může administrátor po vyžádání a souhlasu uživatele kdykoliv změnit.

4.2.4 Registrace licence

Tato funkce bude spojena s registrováním uživatele. Administrátor bude při registrování samotného uživatele registrovat i uživatelovu licenci, kterou již uživatel vlastní, nebo si ji právě pořizuje. Registrace tedy mohou probíhat zpětně. Nový uživatel bude moci

využít rovnou registrování údajů digitálně a nebude muset vůbec využít papírovou formu licence.

4.2.5 Přehled o uživatelích a jejich licencích

Pro administrátora bude velice podstatná organizace zaregistrovaných licencí. Z toho důvodu je v aplikaci funkce zobrazující všechny zaregistrované uživatele a jejich licence. Administrátor může vyhledávat jednotlivé položky uvnitř přehledu v případě, že bude hledat údaje o konkrétním uživateli a jeho licenci. Přehled usnadní případné úpravy údajů, nebo změny platnosti licence. Funkce je oddělena na samostatné stránce, na kterou se uživatel dostane proklikem přes tlačítko na hlavní stránce aplikace.

4.2.6 Odhlášení z aplikace

Pokud bude chtít uživatel aplikaci opustit, využije funkce odhlášení z aplikace. Odhlásí uživatele aktuálně působícího v aplikaci a ho přesměruje zpět na přihlašovací stránku.

4.3 Implementace prostředí pro tvorbu projektu

Prvním krokem pro tvorbu projektu je instalace potřebných nástrojů pro jeho tvorbu. Pro tento projekt bylo instalováno vývojové prostředí VSCode. Do něhož bylo potřeba instalovat několik dalších rozšíření. Tato rozšíření napomáhají při tvorbě kódu a umožňují využít funkce, které nejsou obsahem „holého“ VSCode.

4.3.1 Rozšíření .NET

Rozšíření při tvorbě projektu na základě použití .NET technologií je užitečné, jelikož umožňují použít již existující funkce, jež jsou jeho obsahem. Pro tvorbu .NET projektů je potřeba několik nástrojů. Obsahem je balíček NuGet, který má uživatel možnost spravovat po instalaci tohoto rozšíření, a je řazen mezi zásadní nástroje pro tvorbu projektů založených na .NET. Obsahem jsou také funkce umožňující generaci kódu. Zlepšuje také kontrolu kódu díky integrovanému spouštění testů, přičemž výsledky těchto testů jsou následně zobrazeny v editoru. Další velmi užitečnou funkcí je podpora Azure umožňující uživateli využívat cloudových služeb. (Microsoft, 2023)

4.3.2 Rozšíření C#

Toto rozšíření patří mezi ty nejvíce důležité, vzhledem k tomu, že v projektu je využíván programovací jazyk C#. Podporuje přímo vývoj aplikací za využití jazyka C# společně s využitím ASP.NET frameworku. Užitečným nástrojem obsahující rozšíření je takzvaný refaktoring. Slouží k úpravě kódu, například k přejmenování již definovaných proměnných. Dalším usnadněním při práci je automatické doplňování kódu. (Microsoft, 2023) Tento nástroj nese název IntelliSense.

4.3.3 Rozšíření ES7 + React/...

Rošíření ES7 podporuje zejména usnadnění a zrychlení vývoje v Reactu pomocí zkratek. Neobsahuje pouze zkratky pro samotný React, ale také pro další technologie, například Redux, GraphQL a další. (Microsoft, 2023) Není nutností tento balíček instalovat ani využívat, ale zefektivní a zejména zrychlí kódování. Jako všechny předešlá rozšíření, je i ES7 zcela bezplatné a může jej tedy využít kdokoliv.

4.4 Adresářová struktura aplikace

Pro lepší orientaci při vývoji aplikace je podstatné uspořádání dílčích částí aplikace. Po vytvoření projektu s názvem LicenceApi byla instalována jednotlivá rozšíření, avšak tomu předcházelo vytvoření určité struktury projektu. Základním rozdělením bylo oddělení frontend části od části backendové. V části, kde se nachází pouze část frontendová, je po spuštění vidět samotný vzhled stránky, ale funkce bez backendové části a databáze k nim přiřazené nemohou být funkční. Z toho důvodu je vytvořena část, kde se nachází backend aplikace společně s propojenou databází, ve níž se nachází tabulky pracující s jednotlivými daty.

4.4.1 Struktura LicenceApi/frontend

V názvu podkapitoly je uvedené konkrétní pojmenování použité v projektu, ale nejčastěji se setkáme s pojmenováním client-app. V projektu bylo použito jiné pojmenování z důvodu vlastní orientace. Pojmenování client-app však odkazuje na fakt, že tato část je podstatná pro klienty, jelikož je to první věc, kterou po spuštění aplikace vidí.

Část adresářové struktury frontend se dělí na další části:

- **public,**

  Neměnné soubory, které se po založení aplikace vytvoří samy.
- **components,**

  Generované componenty, díky nimž je práce na této části aplikace rychlejší vzhledem k předem nadefinovanému vzhledu dané komponenty.
- **knihovna (lib),**

  Obsahem této složky je podsložka s názvem validations. Ve složce validations neboli prověření jsou nadefinována určitá pravidla, jež jsou využívána při zápisu dat uživatelem aplikace.
- **app,**

  V této složce se nachází nejčastěji upravované a nově vytvářené části vizuální části aplikace. Jsou zde například podsložky s názvy jednotlivých výše zmíněných funkcí, jako například register, login a další. V podsložkách jsou samotné pages neboli stránky. Každá z nich má odlišný obsah. Tento obsah s dalšími úpravami ve výsledku tvoří vzhled celé aplikace. Nachází se zde další složka – components. Složka components již v celkové struktuře je, avšak tato je odlišná od předchozí. Jednotlivé komponenty jsou nyní psány a není pro ně využíván předem definovaný vzhled.
- **README.md.**

  Jedná se o soubor určený pro zobrazení výsledků dosavadního postupu při tvorbě aplikace.

4.4.2 Struktura LicenceApi/api

Standardní pojmenování tohoto kořenového adresáře nese obvykle název server. V projektu je adresář pojmenován jako api. Pojmenování jednotlivých částí aplikace není nikterak omezeno, ale naopak každý tvůrce si volí názvy složek dle svého uvážení. Pro tvůrce aplikace to přináší určité výhody v podobě lepší orientace v celé části aplikace. Názvy jednotlivých částí jsou poté využívány například v samotném kódu a uživatel si použitá pojmenování ve většině případů pamatuje a není tedy nucen hledat v celém adresáři, jak byl daný soubor nebo složka pojmenován prostředím.

- **Controllers**

Jak již název napovídá, obsahem složky Controllers jsou jednotlivé komponenty aplikace zpracovávající HTTP požadavky klienta. Na základě daného požadavku vyvolají určitou metodu spojenou s tímto požadavkem.

- **Migrations**

  V této složce se nachází jednotlivé soubory, přičemž v každém z nich je zobrazena určitá změna, která ovlivní výslednou databázi. Složka je využívána zejména při vytváření nových tabulek v databázi a k jejich úpravám. Tyto migrace jsou využity díky využití Entity Frameworku.

- **Properties**

  V properties neboli vlastnostech najdeme soubor s koncovkou json. Tento soubor má na starosti konfiguraci spouštění backendové části projektu na konkrétní HTTP adrese.

- **Program.cs**

  Samotný program.cs je velmi zásadní pro správnost spuštění celé aplikace. Funguje jako vstupní bod celé aplikace a definuje spuštění určitých funkcí v aplikaci.

- **api.csproj**

  Spravuje sestavení celého projektu a je uvnitř něho definováno, jaké balíčky jsou používány.

4.5 Úprava vzhledu aplikace

Pro tvorbu frontend části aplikace byl využit React ve spojení s Next.js. Zmíněné spojení je v dnešní době velice rozšířené a je využíváno z několika důvodů. První zásadní výhodou je umožněné verzování pomocí Gitu, které napomáhá k ukládání různých částí projektu s možností návratu na předešlé kroky. Tento postup funguje jako určitá záloha projektu v případě nečekaných problémů a chyb. Další výhodou je, že statický obsah aplikací a stránek je takzvaně před generován. Před generování má za důsledek rychlejší načítání zbytku obsahu aplikace či stránky, a zkrátí tak celkovou dobu generování kompletní aplikace a jejího obsahu. Velice užitečné je automatické zobrazení změny v kódu v právě spuštěné aplikaci.

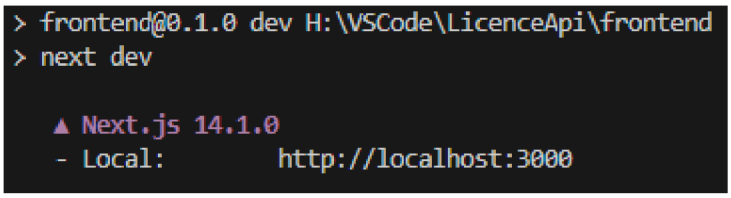
4.5.1 Spuštění pro kontrolu frontend části aplikace

Pomocí příkazu **pnpm dev**, který je zapsán do řádku terminálu, se následně aplikace spustí a pomocí vložení zobrazené http adresy do webového prohlížeče aplikace zobrazí.

Obr. č. 4 – Spuštění frontend části aplikace

Zdroj: Vlastní tvorba

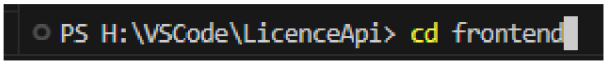
Obr. č. 5 – Vypsání adresy, na které se aplikace nachází



Zdroj: Vlastní tvorba

Správnému spuštění aplikace a zobrazení adresy předchází ještě jeden velmi důležitý krok, bez něhož by spuštění neproběhlo správně. Tím krokem je zadání příkazu **cd frontend.** Tento příkaz nás přesune z aktivního pracovního adresáře na adresář námi zvolený, v tomto případě adresář frontend.

Obr. č. 6 – Příkaz pro přesunutí do adresáře frontend



Zdroj: Vlastní tvorba

Pokud jsou kroky provedeny správně, zobrazí se na adrese v terminálu samotná aplikace a výsledek kódu. Proces může být stále spuštěn a během toho mohou probíhat změny v kódu, které se bezprostředně po uložení aktuálních změn promítnou na samotné aplikaci.

4.5.2 Vzhled přihlašovacího formuláře a validace prvků

Na první straně této aplikace je vytvořen přihlašovací formulář. Nejprve bylo potřeba vytvořit samotný přihlašovací formulář a upravit jeho vzhled. Dalším aspektem jsou jednotlivé údaje zadávané uživatelem. Pro ty bylo nutné vytvořit určité podmínky neboli validace. Tyto podmínky fungují na základě správnosti údajů zadaných uživatelem.

Podmínky působí po odeslání formuláře, po stisknutí tlačítka pro přihlášení. Na základě porovnání údajů v databázi bude vstup vyhodnocen jako chybný nebo správný.

Vzhled byl upraven pomocí již definovaných komponent, které je možno právě díky Next.js využívat. Tyto komponenty jsou stavěny na základu Reactu a mohou být využity v jakékoliv části tohoto projektu, ale nejprve se musí importovat. Není však nutné je importovat ručně. Při použití některé z komponent, jako například **FormLabel**, se po jeho naplnění obsahem tento řádek označí jako chybný a pomocí možnosti opravy **QuickFix** bude chyba odstraněna a daná komponenta se importuje automaticky. Není nutné je bezprostředně na začátku importovat všechny, protože nevíme, zda budou využity.

Zdrojový kód č. 1 - Vytvoření přihlašovacího formuláře

```
return (
    <>
      <div className="flex items-center justify-center min-h-screen">
        <Card className="w-96">
          <CardHeader className="text-center">
            <CardTitle>Přihlášení do evidence</CardTitle>
            <CardDescription>
              Přihlašte se pomocí svého emailu a hesla.
            </CardDescription>
          </CardHeader>
          <Form {...form}>
            <form onSubmit={form.handleSubmit(handleSubmit)}>
              <CardContent>
                <div className="grid w-full items-center gap-4">
                  <div className="flex flex-col space-y-1.5">
                    <FormField
                      control={form.control}
                      name="email"
                      render={({ field }) => (
                        <FormItem>
                          <FormLabel>E-mail</FormLabel>
                          <FormControl>
                            <Input placeholder="Váš e-mail" {...field} />
                          </FormControl>
                          <FormMessage />
                        </FormItem>
                      )}
                    />
                  </div>
                  <div className="flex flex-col space-y-1.5">
                    <FormField
                      control={form.control}
                      name="password"
```

```
                    render={({ field }) => (
                      <FormItem>
                        <FormLabel>Heslo</FormLabel>
                        <FormControl>
                          <Input
                            type="password"
                            placeholder="Vaše heslo"
                            {...field}
                          />
                        </FormControl>
                        <FormMessage />
                      </FormItem>
                    )}
                  />
                </div>
              </div>
            </CardContent>
            <CardFooter>
              <Button className="w-full mt-2" type="submit">
                Přihlásit se
              </Button>
            </CardFooter>
          </form>
        </Form>
      </Card>
    </div>
  </>
```

Zdroj: Vlastní tvorba

4.5.3 Vzhled registračního formuláře a validace prvků

Registrace nové licence do aplikace probíhá zároveň s registrací nového uživatele. Registrování nových uživatelů a jejich licencí má v kompetenci pouze osoba s administrátorskou rolí. Role jsou vytvořeny díky ASP.NET Identity Frameworku, který umožnuje tyto role vytvářet a na jejich základě zpřístupnit obsah aplikace určitým rolím. Samotný vzhled registračního formuláře je opět upraven pomocí importovaných komponent, na jejichž základě je vzhled formuláře postaven. Formulář obsahuje několik řádků, které je nutno vyplnit, poté jsou podrobeny validacím pro formát zápisu.

Například u čísla licence je validace nastavena tak, aby tento údaj obsahoval přesně šest znaků. Validace jsou vytvořeny pro všechny prvky tohoto formuláře, aby nedošlo k případné chybě zápisu. Přestože by tato skutečnost nastala, může osoba s pověřenou rolí chyby později upravit v přehledu licencí a jejich uživatelů.

Zdrojový kód č. 2 - Vytvoření validací pro registrační formulář

```
import { z } from 'zod';

export const registerSchema = z.object({
  email: z.string().email({ message: 'Špatný formát emailu' }),
  password: z.string().min(6, { message: 'Heslo musí mít alespoň 6 znaků'
}),
  confirmpassword: z.string().min(6, { message: 'Hesla se musí shodovat' }),
  birthnumber: z.string().length(11, {
    message: 'Rodné číslo musí obsahovat přesně 11 znaků včetně lomítka',
  }),
  licencenumber: z.string().refine((data) => /^\d{6}$/.test(data), {
    message: 'Evidenční číslo licence musí obsahovat právě 6 číslic',
  }),
  firstname: z
    .string()
    .refine((data) => data.trim() !== '', { message: 'Povinný údaj' }),
  lastname: z
    .string()
    .refine((data) => data.trim() !== '', { message: 'Povinný údaj' }),
  validity: z.string().refine(
    (value) => {
      const onlyDateFormat = /^\d{4}-\d{2}-\d{2}$/;
      return onlyDateFormat.test(value) || value.toLowerCase() ===
'neurčito';
    },
    {
      message:
        'Platnost licence musí být ve formátu "YYYY-MM-DD'
  ),
  birthplace: z
    .string()
    .refine((data) => data.trim() !== '', { message: 'Povinný údaj' }),
  placeofissue: z
    .string()
    .refine((data) => data.trim() !== '', { message: 'Povinný údaj' }),
});
```

Zdroj: Vlastní tvorba

4.5.4 Vzhled digitální legitimace

Nejpodstatnější částí pro uživatele je digitální rybářská licence. Uživatel je po úspěšném přihlášení převeden na hlavní stránku, na níž je zobrazena digitální verze rybářské licence s údaji zadanými při registraci. Tuto licenci pak uživatel bude mít stále při sobě, nahranou v elektronickém zařízení, bez možnosti ztráty. Pro vzhled byly použity

importované komponenty a kód byl následně upraven. Vycházel jsem z reálné papírové verze licence, kdy jsem se snažil tento model vytvořit co nejjednodušší pro uživatele.

4.5.5 Přehled uživatelů a jejich licencí

Přehled uživatelů a jejich licencí je samostatná stránka, na níž se dostaneme pomocí prokliku v menu hlavičky hlavní stránky. Zobrazí se kompletní výpis uživatelů systému a jejich licencí. Jednotlivé údaje jsou uspořádány za sebou, přičemž každý sloupec, nesoucí název druhu daného údaje, nám napomáhá k přehlednosti a lepší orientaci v přehledu.

Další velice podstatnou funkcí na této stránce je možnost editace jednotlivých údajů. Uživatel může zažádat o změnu údajů, jako například e-mailu, hesla a další. Druhou funkcí je ukončení platnosti dané legitimace nebo prodloužení o deset let. Jak je již zmíněno, licence každého uživatele má určitou platnost. Je potřeba ji prodlužovat, aby uživatel splňoval základní pravidlo pro umožnění lovu a tím měl aktivní rybářskou licenci. Po zaplacení prodloužení platnosti licence na stejné pobočce, kde byla licence vydána, bude uživateli platnost prodloužena uživatelem s rolí, jež tuto změnu umožňuje.

4.6 Databáze

Vzhledem k velikosti aplikace jsem zvolil použití databázového multiplatformního balíčku SQLite. Tento databázový nástroj pro správu relačních databází je využíván zejména na menší projekty. Umožňuje přístup k databázi bez nutnosti dalšího serveru skrze datový soubor, jenž vede přímo do samotné databáze. Pracuje tedy na lokálním zařízení, které slouží jako lokální server pro danou databázi, v níž jsou následně ukládána jednotlivá data aplikace. Samotné definování prvků obsažených v databázi a jejich následné aplikování lze uplatnit ve vývojovém prostředí. (About SQLite, 2023)

Díky databázovému nástroji DB Browser pro SQLite je možno do samotné databáze nahlédnou pomocí připojení na lokální server, kde je aplikace spuštěna. Umožní tím nahlédnutí do již vytvořené databáze a tabulek, což napomáhá ke zpětné kontrole a kontrole správného zápisu dat do databázových tabulek.

4.6.1 Použité třídy a jejich využití

Databáze obsahuje několik tabulek využitých k ukládání dat uživatelů a jejich licencí. K uživatelským účtům se pojí rozhraní ASP.NET Core Identity. Toto rozhraní se stará o účty uživatelů a jejich role. Samotné rozhraní automaticky vytvoří několik tabulek, konkrétně tři.

- **AspNetRoles**

První tabulka nese název AspNetRoles. Hlavním úkolem této tabulky je ukládání jednotlivých informací o rolích, které jsou následně přiřazovány k danému uživateli. Zásadními sloupci v této tabulce jsou sloupec s pojmenováním **Id**, sloužící jako unikátní identifikátor pro konkrétní roli, a sloupec **Name**. Konkrétně tento sloupec nese název role, přičemž název je dále používán v rámci celé aplikace. Názvy rolí jsem zvolil dle mých potřeb, ale nejčastěji použité pojmenování pro role v aplikaci nesou názvy jako User nebo Admin. Uvedené příklady názvů rolí jsem v práci použil také, a to z důvodu oddělení pravomocí jednotlivých uživatelů aplikace. Role jsou jedinečné, protože dokážeme oddělit přístup určitým uživatelům systému, což je užitečné zejména v aplikacích, kde určité funkce (v tomto případě například registraci uživatele) může provést pouze uživatel s rolí Admin.

- **AspNetUserRoles**

Druhou tabulkou ve zmíněném rozhraní je AspNetUserRoles. Obsahem této tabulky je UserId, to je rovněž obsahem tabulky AspNetUsers. V důsledku působení tohoto unikátního klíče v obou tabulkách dochází k propojení obou tabulek mezi sebou. To stejné platí v případě RoleId, taktéž působící v tabulce AspNetUserRoles, a zároveň v tabulce AspNetRoles. Účelem tabulky AspNetUserRoles je sledování a přiřazení uživatele k určité roli.

- **AspNetUser**

Poslední tabulkou je AspNetUser. Obsahem tabulky jsou informace o jednotlivých uživatelích systému. Mezi tyto informace patří zejména přihlašovací údaje, e-mailová adresa uživatele a jeho heslo, přičemž hesla jsou chráněna hashovací metodou, jež je obsažena v ASP.NET Identity frameworku a zajišťuje tím integritu a ochranu těchto hesel.

4.7 Tvorba jednotlivých částí databáze

Pro ukládání jednotlivých dat uživatelů je potřeba definovat třídy podle potřeb projektu. Těchto tříd jsem v projektu musel vytvořit více kvůli pokrytí všech jednotlivých úkonů v projektu, pro které budou data ukládána do databáze. Pro příklad jsem použil třídu Licence, naplnil jsem ji proměnnými reprezentující obsah tabulky licence.

Základem je nadefinování jednotlivých proměnných, kdy první z nich je platnost licence. Platnost licence bude zastupovat konkrétní datum platnosti licence uživatele. Další proměnnou je místo vzniku licence. Při registrování nové licence do systému bude vydavatel muset uvést, v jakém městě byla registrována. Další velice podstatnou proměnnou je číslo licence. Tím je myšleno unikátní šestimístné číslo každého uživatele. Jeho jedinečnost spočívá v tom, že se nemůže shodovat s číslem jiného uživatele.

Poslední definovanou proměnnou v této třídě je **Id**. Tato proměnná nese unikátní označení každého jednotlivého uživatele. Pomocí proměnných tohoto typu lze také jednotlivé tabulky následně propojovat mezi sebou, čehož jsem v tomto projektu musel využít také k propojení jednotlivé licence s uživatelem.

Ve spodní části příkladu je opětovně definována proměnná, která se již nachází v třídě ApplicationUser. Tedy v tomto případě za pomocí naplnění ForeignKey definovaným Users_Id dochází k spojení ve tvaru 1:1, což je další krok vedoucí k umožnění přidělování jednotlivých licencí daným uživatelům.

Zdrojový kód č. 3 - Vytvoření třídy Licence

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Microsoft.AspNetCore.Identity;

namespace Api.Data
{
    public class Licence
    {
        public int Id { get; set; }
        public DateOnly Validity { get; set; }
        public string PlaceOfIssue { get; set; }
        public int LicenceNumber { get; set; }

        [ForeignKey("ApplicationUser")]
        public int Users_Id { get; set; }
        public ApplicationUser ApplicationUser { get; set;}
```

Zdroj: Vlastní tvorba

4.7.1 Provedení migrace a její obsah

Pokud je třída vytvořena, je nutné pomocí zapsání příkazu do terminálu VSCodu vytvořit takzvanou migraci pro její zapsání. Příkaz bude vypadat takto: dotnet ef migrations add #zvolenynazevtetomigrace. Po úspěšném průběhu migrace je nutné aktualizovat samotnou databázi pomocí příkazu dotnet ef database update. Po úspěšném dokončení všech operací bude změna zapsána do databáze, a ve složce data bude automaticky vytvořen soubor, který představuje mnou vytvořenou migraci.

Zdrojový kód č. 4 - Migrace pro vytvořenou třídu licence

```
migrationBuilder.CreateTable(
                name: "Licence",
                columns: table => new
                {
                    Licence_Id = table.Column<int>(type: "INTEGER",
nullable: false)
                        .Annotation("Sqlite:Autoincrement", true),
                    Validity = table.Column<string>(type: "TEXT", nullable:
false),
                    PlaceOffIssue = table.Column<string>(type: "TEXT",
nullable: false),
                    LicenceNumber = table.Column<int>(type: "INTEGER",
nullable: false),
                    UserId = table.Column<int>(type: "INTEGER", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Licence", x => x.Licence_Id);
                    table.ForeignKey(
                        name: "FK_Licence_AspNetUsers_UserId",
                        column: x => x.UserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateIndex(
                name: "IX_Licence_UserId",
                table: "Licence",
                column: "UserId",
                unique: true);
```

Zdroj: Vlastní tvorba

V uvedeném příkladu je možné vidět, jak jsou pro jednotlivé proměnné vytvořeny sloupce následně reprezentující tyto vlastnosti. Také je možné vidět, jak je vytvořen cizí klíč, díky němuž je tato tabulka propojena s tabulkou ApplicationUser.

4.8 Endpointy a použité části pro jejich tvorbu

Další nezbytnou částí pro tvorbu aplikace bylo vytvoření jednotlivých endpointů, ve kterých jsou volány metody pro práci s daty, ale také samotná logika dané funkce. Pro to, aby vizuální část aplikace byla funkční, bylo potřeba vytvářet právě jednotlivé endpointy, na něž je vizuální část aplikace napojena. Pro vytvoření endpointu jsem vytvořil nejdříve servisu, v níž jsem vytvořil jednotlivé metody poté potřebné v controlleru. Bylo nutné vytvořit třídy s definovanými proměnnými, s kterými budu v endpointu pracovat. Pro přiblížení jsem jako příklad zvolil vytvoření endpointu pro získávání dat z databáze aktuálně přihlášeného uživatele aplikace.

Zdrojový kód č. 5 – Získávání dat uživatele pro zobrazení v digitální licenci

```
public async Task<InfoUserDTO> GetUserInfoAsync(int userId)
    {
        var user = await _userManager.FindByIdAsync(userId.ToString());

        if (user == null)
        {
            return null;
        }

        var licence = await
_licenceService.GetLicenceByUserIdAsync(user.Id);

        var userInfo = new InfoUserDTO
        {
            FirstName = user.FirstName,
            LastName = user.LastName,
            BirthPlace = user.BirthPlace,
            BirthNumber = user.BirthNumber,
            LicenceNumber = licence?.LicenceNumber,
            Validity = licence?.Validity,
            PlaceOfIssue = licence?.PlaceOfIssue
        };

        return userInfo;
```

Zdroj: Vlastní tvorba

Prvním krokem bylo vytvoření souboru s názvem UserService.cs. Cílem celého endpointu je získat jednotlivá data aktuálně přihlášeného uživatele a vložit je do UserInfoDTO. Po naplnění těchto proměnných budou data zobrazena na samotné licenci v jiných částech aplikace. Pro to, aby byl tento endpoint funkční, jsem vytvořil zmíněný UserInfoDTO, do kterého jsem definoval jednotlivé proměnné. Pro tuto funkci je potřeba naplnit proměnné FirstName až PlaceOfIssue.

Jednotlivé informace o uživateli jsou uloženy v tabulkách ApplicationUser a Licence. Jak je na samotném začátku vidět, na úvod je definována metoda s názvem GetUserInfoAsync. V této metodě využívám userId jakožto identifikátor aktuálně přihlášeného usera v aplikaci. Pomocí Id aktuálně přihlášeného uživatele a vnořené funkce GetUserByIdAsync jsem schopen získat data usera z databáze, avšak je potřeba získat další data z tabulky licence, což zajišťuje GerLicenceByUserIdAsync. Poté, co jsou veškerá potřebná data získána, jsou naplněna do již zmíněného UserInfoDTO. V neposlední řadě musí být služba registrována v hlavním programu.cs.

Po dokončení všech potřebných částí bylo potřeba endpoint vyzkoušet. K otestování endpointu jsem využil swagger. Swagger je velice užitečný a využívaný zejména pro dokumentaci a práci se samotnou API, je však nutné instalovat danou knihovnu, jež nám swagger dovolí použít. Umožňuje testovat již vytvořené části pomocí funkce execute. Funkce execute umožňuje naplnit vytvořený endpoint daty a vypsat průběh této funkcionality po naplnění mnou zvolenými daty. Průběh testu může skončit buď chybou nebo požadovaným výsledkem. Pro spuštění je potřeba do terminálu napsat příkaz dotnet run. Po správném průběhu je prohlížečem zobrazena adresa a připsáním dokumentace swagger za zmíněnou adresu jsem odkázán na samotný swagger, v němž testování provedu.

4.8.1 Metoda pro úpravu uživatelů

Tato metoda umožňuje upravit jednotlivé informace uživatele. Metoda získává data vybraného usera pomocí jeho Id a umožňuje provést změnu těchto dat s následným uložením do databáze.

4.8.2 Metoda pro úpravu, zneplatnění a prodloužení platnosti licence

Při výběru určité licence některého z uživatelů, tato metoda automaticky přepíše a uloží platnost uživatelské licence na hodnotu null nebo +10 let. Přijímá zvolený parametr validity, přepisuje jeho obsah a následně ho uloží zpět do databáze.

Zhodnocení výsledků

Hlavním cílem bakalářské práce bylo vytvoření aplikace za účelem digitalizace rybářských licencí. Mým cílem bylo také seznámení se s využívaným nástrojem pro tvorbu uživatelských rozhraní s názvem React, který jsem se rozhodl použít ve spojení s .NET. Práci v tomto nástroji jsem chtěl vyzkoušet, protože je dnes velice využívaný.

Výsledkem mé práce je aplikace umožňující zápis rybářských licencí do digitální formy, a to pouze za pomoci administrátorské role, společně se správou těchto licencí. Z pohledu běžného uživatele je tato aplikace koncipována co nejjednodušeji, aby se uživatel přihlásil a jeho licence byla zobrazena na hlavní stránce, k níž má přistup. Toto byl můj první úkol.

Druhým úkolem bylo samotné vytvoření nebo převedení aktuálních papírových licencí do digitální formy společně s možností jejich následné správy. Zejména pro správce těchto licencí v jednotlivých organizací je podstatná správa a kontrola licencí. Z toho důvodu jsem vytvořil v aplikaci několik funkcí.

První z nich je registrace uživatelů společně s jejich licencí. Dalšími funkcemi je zobrazení všech uživatelů a licencí, přičemž tyto dvě zobrazení jsou rozloženy na další dvě samostatné stránky aplikace. Na každé z těchto stran jsou tedy zobrazeny všechny záznamy z databáze. Aby správce mohl tyto licence a jejich vlastníky spravovat, bylo potřeba doplnit několik funkcí v těchto zobrazeních. Konkrétně zobrazení všech uživatelů obsahuje filtrování pomocí příjmení, což zjednodušuje hledaní uživatele v zobrazení všech licencí, kde je přidáno filtrování pomocí uživatelského Id.

Dalšími funkcemi je listování mezi stranami, přičemž každá strana je omezena pro zobrazení pouze deseti uživatelů najednou. Navazuje poslední funkce pro každý řádek tohoto zobrazení, a to možnost editace údajů daného uživatele, díky čemuž je zajištěno, že informace uživatelů v databázi budou aktuální.

V zobrazení licencí je obsaženo také několik funkcí. První z nich je již zmíněné filtrování pomocí uživatelského Id, což napomáhá k rychlému dohledání licence tohoto uživatele. Mimo stránkování jako u předešlého zobrazení jsou zde další tři funkce, které slouží zejména pro správu licencí. Editace licence je mezi nimi také společně s prodloužením platnosti licence o deset let. Poslední funkcí je zrušení platnosti licence. Tuto funkci jsem přidal v případě, kdyby uživatel spáchal přestupek, nebo neuhradil prodloužení platnosti jeho licence.

Vytvořená aplikace by z mého pohledu určitě uplatnění našla. Je zde ale několik možných funkcí a úprav, díky kterým by bylo možné převést do digitální formy celý rybářský systém, v němž jsou určité další části, jež mají návaznost právě na tuto práci:

- komunikace mezi uživatelem a správcem této aplikace (možnost dotazováním zasílání žádosti o změnu údajů),
- návaznost na rybářské povolenky (možnost zakoupení jednoho z vybraných druhu povolenek, platební metoda, přiřazení povolenky s platností pod účet uživatele),
- propojení s rybářskou organizací, kde je uživatel členem (přiřazení uživatele do organizace, evidence uživatelů jednotlivých organizací, vytvoření a správa organizací, přiřazení jednotlivých správců pod organizace, zakoupení platných členských známek a platba brigád),
- evidování uživatelských přestupků (deaktivace licence z důvodu přestupku, ukládání přestupku uživatele).

Zmíněná rozšíření aplikace jsou z mého pohledu určitě použitelná a lze s těmito myšlenkami pracovat. Avšak rybářský systém je jako celek velmi členité, složité a rozsáhlé téma. Tvorba tak rozsáhlého projektu by vyžadovala více osob a větší časovou dotaci, aby byl projekt dostatečně otestován pro splnění všech potenciálně nastalých situací.

Technologií pro tvorbu webových aplikací je několik a lze na základě této myšlenky vytvořit projekty, na jejichž základě jsme schopni vyřešit možné problémy a zejména usnadnit mnoho denně vykonávaných procesů. Ve spojitosti s praktickým využitím jsem zvolil právě toto téma.

5 Závěr

Cílem mé práce bylo vytvoření aplikace pro digitalizaci rybářských licencí. V návaznosti na toto téma ve své teoretické části práce představuji jednotlivé nástroje pro tvorbu zmíněných aplikací společně s dalšími částmi korespondujícími s tímto tématem.

Práce je rozdělena do dvou částí, přičemž první z nich je teoretická a druhá praktická. V první kapitole teoretické části jsem představil platformu ASP.NET Core, její vznik, využití a klíčové vlastnosti. (kapitoly 3.1.1 a 3.1.2). Následně je vysvětlen princip a obsah jejích dvou částí – architektury MVC a MVVM. (kapitoly 3.1.3 a 3.1.4). Každá z těchto architektur obsahuje určité prvky, jejichž účel a princip je vysvětlen v samostatných podkapitolách. V poslední podkapitole je pomocí vytvořené tabulky zobrazeno porovnání zmíněných architektur a výhody a nevýhody použití každé z nich. (kapitola 3.1.5).

Další kapitola je věnována databázovému nástroji SQLite. (kapitola 3.2). Jsou zde vysvětleny jednotlivé vlastnosti a výhody tohoto nástroje společně s architekturou a popsána práce v samotném nástroji za použití příkazů s názornými příklady. (kapitoly 3.2.1 a 3.2.2).

Nezbytným nástrojem pro vývoj aplikací je vývojové prostředí. Kapitola na toto téma obsahuje vysvětlení a přiblížení, čím vlastně samotné vývojové prostředí je, a k čemu je využíváno. (kapitola 3.3). Nezbytnou součástí jsou informace o užitečnosti těchto prostředí a specifikování jednotlivých druhů. (kapitoly 3.3.1 a 3.3.2). Jsou zde uvedeny příklady nejznámějších vývojových prostředí a každé z nich je popsáno z hlediska historie a výhod využití. (kapitola 3.3.3 a její části).

Dalším tématem je Informační systém. (kapitola 3.4). Vysvětleny jsou jednotlivé části tohoto systému a co je jejich obsahem. Též jsou uvedeny a popsány druhy informačních systémů a jejich části. (kapitola 3.4.1). Druhá polovina této kapitoly se věnuje správě jednotlivých procesů systémů a jeho zabezpečení. (kapitoly 3.4.2 a 3.4.3).

Navazují další dvě kapitoly. První z těchto kapitol je Webová stránka. (kapitola 3.5). Popisuje, co může být obsahem webové stránky a v dalších podkapitolách je uvedena dostupnost a struktura samotné webové stránky. (kapitoly 3.5.1 a 3.5.2).

Přichází na řadu druhá ze zmíněných kapitol. V této kapitole je na úvod popsáno, jak se webová aplikace odlišuje od webové stránky a co je jejím obsahem. (kapitola 3.6). Zbylé dvě podkapitoly uvádějí možné výhody a nevýhody webových aplikací. (kapitoly 3.6.1 a 3.6.2).

V závislosti na tvorbě webových aplikací a stránek bylo z mého pohledu vhodné uvést téma responzivní design. (kapitola 3.7). V této kapitole je na úvod uvedena historie vzniku tohoto přístupu a samotný cíl, proč byl tento přístup vytvořen. Obsahem této kapitoly jsou další podkapitoly, přičemž každá z nich přímo souvisí se samotným responzivním designem a jsou nedílnou součástí pro zajištění responzivity. (kapitoly 3.7.1 až 3.7.4).

Závěrečnou kapitolou v mé teoretické části je kapitola, která se týká nástroje pro usnadnění vývoje aplikací a jeho název je Azure DevOps. (kapitola 3.8). V mé práci jsem chtěl uvést alespoň jeden z nástrojů pro sdílenou tvorbu aplikací, z důvodu využívání těchto nástrojů při tvorbě rozsáhlejších projektů. Tyto nástroje zejména využívají firmy, které se danému vývoji aplikací věnují. Jednotlivé součásti tohoto nástroje jsou popsány v podkapitolách navazující přímo na toto téma. (kapitoly 3.8.1 až 3.8.3).

V praktické části mé práce se věnuji přiblížení řešené problematiky, a to konkrétně vytvoření webové aplikace. Uvádím nástroje, jež jsem při tvorbě této aplikace využil, a uvádím konkrétní příklady zpracování jednotlivých částí. Na úvod této praktické části představuji, jak si představuji digitalizaci rybářských licencí, a jaké základní nástroje budu využívat s tím, že zbylé nástroje, jež jsem využil, jsou uvedené v dalších kapitolách a podkapitolách této praktické části. (kapitola 4).

Nezbytný byl návrh aplikace, jako takové. Popisuje při návrhu aplikace používané nástroje a k čemu každá z nich bude užitečná. (kapitola 4.1). Dále popisuje samostatně každou funkci, která bude v aplikaci obsažena, přičemž uvádím u každé z nich význam, a jakou úlohu bude v této aplikaci mít. (kapitoly 4.2 a 4.2.1 až 4.2.6).

Následuje samotná implementace prostředí pro tvorbu dané aplikace. (kapitola 4.3). Základní instalace vývojového prostředí je zmíněna v předchozí kapitole. Bylo však potřeba instalovat určitá rozšíření, která byla pro tvorbu této aplikace klíčová. Název, důvod a výhody instalace každého z těchto rozšíření je uvedeno v samostatných kapitolách. (kapitoly 4.3.1 až 4.3.3).

V pořadí čtvrté kapitole se snažím přiblížit, jak vypadá adresářová struktura mé aplikace. (kapitola 4.4). Pro přehlednost, jak již uvádím v zmíněné kapitole, jsou rozděleny části aplikace na frontend a backend celé aplikace. Toto rozdělení jsem využil také při tvorbě podkapitol, kdy se každá z nich věnuje struktuře obou částí tohoto rozdělení. Je také uveden obsah a princip objektů obsažených v celkové struktuře aplikace. (kapitoly 4.4.1 a 4.4.2).

Zásadní je vizuální stránka aplikace. Ta je tvořena pomocí kombinace knihovny React a frameworku Next.js. Díky tomuto frameworku je umožněno importovat již konstruované

componenty, jež využívám k tvorbě jednotlivých stránek aplikace. Toto spojení má několik výhod, které popisuji v úvodní části kapitoly s názvem „Úprava vzhledu aplikace“. (kapitola 4.5). Nutná je kontrola vytváření jednotlivých vizuálních částí aplikace, proto se v této části nachází podkapitola, v které vysvětluji, jak spustit samotnou aplikaci a jaké výhody má, když mohu mít aplikaci spuštěnou a zároveň provádět úpravy ve vzhledu dané stránky pouze pomocí ukládání úprav a automatickému propsání těchto změn. (kapitola 4.5.1). Navazují podkapitoly týkající se samotného vzhledu jednotlivých formulářů. Jsou zde popsány postupy tvorby těchto vzhledů a použité komponenty. Uvedl jsem také příklad jednoho z formulářů a vytvoření validace pro další z formulářů. (kapitoly 4.5.2 až 4.5.5).

Nezbytnou součástí pro tvorbu této, ale i jakékoliv jiné aplikace, jsou databáze. Databáze, jak je již zmíněno v další z kapitol, je velice nezbytná z důvodu ukládání uživatelských dat a k práci s nimi. (kapitola 4.6). Jelikož se tato aplikace zabývá digitalizací a přenosem papírové formy zápisu do elektrické formy, musí být každá z licencí, a tudíž i data v ní uložena právě v databázi, z důvodu uchování těchto dat k další práci s nimi. Díky použití rozhraní ASP.NET Core Identity, jsou již v databázi vytvořeny určité tabulky, které se zejména starají o role uživatelů v aplikaci a o uživatele samotné. Tyto tabulky jsou popsány v podkapitole, jež se vztahuje právě na použité rozhraní. (kapitola 4.6.1).

K tématu navazuje další kapitola, která se zabývá samotným vytváření tabulek uvnitř databáze, jež budou využity pro ukládání rozdílných dat aplikace. (kapitola 4.7). Je zde uveden příklad konkrétní tvorby uvedené třídy a následné provedení migrace a dalších kroků pro zapsání této třídy do celkové databáze. (kapitola 4.7.1). K jednotlivým krokům tvorby jsou uvedeny příklady v podobě vložených obrázků.

V jedné z posledních kapitol mé praktické části se věnuji tvorbě endpointů a jednotlivých metod pro funkcionalitu jednotlivých částí aplikace. (kapitola 4.8). Opět je vysvětlen samotný princip, čím endpoint je, a k čemu slouží společně s metodami. Pro uvedení konkrétního příkladu zhotovení endpointu jsem vybral získáváni dat od uživatele. Na toto téma navazují podkapitoly, přičemž v každé z nich je popsán princip metod, které jsem zvolil.

Jako většina aplikací i tato je rozšiřitelná. V kapitole zhodnocení výsledků jsem uvedl výsledek své práce a několik možných rozšíření, která by z praktického hlediska byly velice užitečné. Samotný program je odevzdán do UIS a jeho otevření je možné pomocí stejné či novější verze vývojového prostředí Visual Studio Code. Pro nahlédnutí do aplikace je vytvořen uživatel s rolí admin, jenž má přístup do všech částí aplikace. Pro nahlédnutí do databáze jsem osobně využíval DB Browser pro SQLite.

6 Seznam použitých zdrojů

1. About SQLite, 2023. SQLITE. *Www.sqlite.org* [online]. [cit. 2023-11-24]. Dostupné z: https://www.sqlite.org/about.htm
2. ADOBE, 2021. O webových aplikacích. *Helpx.adobe.com* [online]. [cit. 2023-11-29]. Dostupné z: https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html
3. BROWSERSTACK, 2023. How to make React App Responsive using react-responsive? *Browserstack* [online]. [cit. 2023-11-27]. Dostupné z: https://www.browserstack.com/guide/how-to-make-react-app-responsive
4. DEVX, 2023. Development Environment. DEVX. *Devx.com* [online]. [cit. 2024-01-31]. Dostupné z: https://www.devx.com/terms/development-environment/
5. ECLIPSE, 2023. Eclipse Platform Overview. ECLIPSE. *Eclipse.dev* [online]. [cit. 2024-01-17]. Dostupné z: https://eclipse.dev/eclipse/eclipse-charter.php
6. FREEMAN, Adam, 2022. *Pro ASP.NET Core 6*. Ninth Edition. One New York Plaza, Suite 4600 New York, NY 10004-1562: APress. ISBN 9781484279564.
7. FREEMAN, Adam, 2022. *Pro ASP.NET Core 6*. Ninth Edition. One New York Plaza, Suite 4600 New York, NY 10004-1562: APress. ISBN 9781484279564.
8. ITNETWORK, 2016. MVC architektura. ITNETWORK. *Www.itnetwork.cz* [online]. [cit. 2023-11-06]. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor
9. MICROSOFT, 2021. What is ASP.NET Core? MICROSOFT. *Microsoft.com* [online]. [cit. 2023-11-01]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core
10. MICROSOFT, 2022. Overview of ASP.NET Core MVC. MICROSOFT. *Microsoft.com* [online]. [cit. 2023-11-06]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-6.0
11. MICROSOFT, 2023. .NET Extension Pack. MICROSOFT. *Microsoft.com* [online]. [cit. 2024-02-05]. Dostupné z: https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.vscode-dotnet-pack

12. MICROSOFT, 2023. C# for Visual Studio Code. MICROSOFT. *Microsoft.com* [online]. [cit. 2024-02-05]. Dostupné z: https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp
13. MICROSOFT, 2023. ES7+ React/Redux/React-Native snippets. MICROSOFT. *Microsoft.com* [online]. [cit. 2024-02-05]. Dostupné z: https://marketplace.visualstudio.com/items?itemName=dsznajder.es7-react-js-snippets
14. MICROSOFT, 2023. Model-View-ViewModel (MVVM). MICROSOFT. *Microsoft.com* [online]. [cit. 2023-11-17]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/architecture/maui/mvvm
15. MICROSOFT, 2023. What is Azure Pipelines? MICROSOFT. *Microsoft.com* [online]. [cit. 2024-01-19]. Dostupné z: https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops
16. MICROSOFT, 2023. Why did we build Visual Studio Code? MICROSOFT. *Code.visualstudio.com* [online]. [cit. 2024-01-17]. Dostupné z: https://code.visualstudio.com/docs/editor/whyvscode
17. MICROSOFT, 2024. What is Azure Boards? MICROSOFT. *Microsoft.com* [online]. [cit. 2024-01-19]. Dostupné z: https://learn.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops
18. MICROSOFT, 2024. What is Azure DevOps? MICROSOFT. *Microsoft.com* [online]. [cit. 2024-01-19]. Dostupné z: https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops
19. MICROSOFT, 2024. What is Azure Repos? MICROSOFT. *Microsoft.com* [online]. [cit. 2024-01-19]. Dostupné z: https://learn.microsoft.com/en-us/azure/devops/repos/get-started/what-is-repos?view=azure-devops
20. Model MVVM, 2023. In: *Www.microsoft.com* [online]. [cit. 2024-03-01]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/architecture/maui/media/mvvm-pattern.png
21. Model View Controller example, 2023. In: *Developer.mozilla.org/* [online]. [cit. 2024-03-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/MVC/model-view-controller-light-blue.png
22. MOZILLA, 2022. The Model. MOZILLA. *Developer.mozilla.org* [online]. [cit. 2023-11-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/MVC

23. MOZILLA, 2023. How the web works. *Developer.mozilla.org* [online]. [cit. 2023-11-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works
24. MOZILLA, 2023. Responsive design. *Developer.mozilla.org* [online]. [cit. 2023-11-27]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design
25. STONIS, Michael, 2022. *Enterprise Application Patterns Using .NET MAUI* [online]. EDITION v1.0. Redmond, Washington 98052-6399: Microsoft Corporation [cit. 2023-11-17]. Dostupné z: https://dotnet.microsoft.com/en-us/download/e-book/maui/pdf
26. TECHOPEDIA, 2022. Refactoring. ROUSE, Margaret. TECHOPEDIA. *Techopedia.com* [online]. [cit. 2024-01-31]. Dostupné z: https://www.techopedia.com/definition/3865/refactoring
27. TECHTARGET, 2023. DEFINITION information systems (IS). *Techtarget.com* [online]. [cit. 2023-12-07]. Dostupné z: https://www.techtarget.com/whatis/definition/IS-information-system-or-information-services
28. THE APACHE SOFTWARE FOUNDATION, 2023. Apache NetBeans. THE APACHE SOFTWARE FOUNDATION. *Netbeans.apache.org* [online]. [cit. 2024-01-17]. Dostupné z: https://netbeans.apache.org/front/main/

7 Seznam obrázků, tabulek, grafů a zkratek

7.1 Seznam obrázků

7.2 Seznam zdrojových kódů

7.3 Seznam tabulek

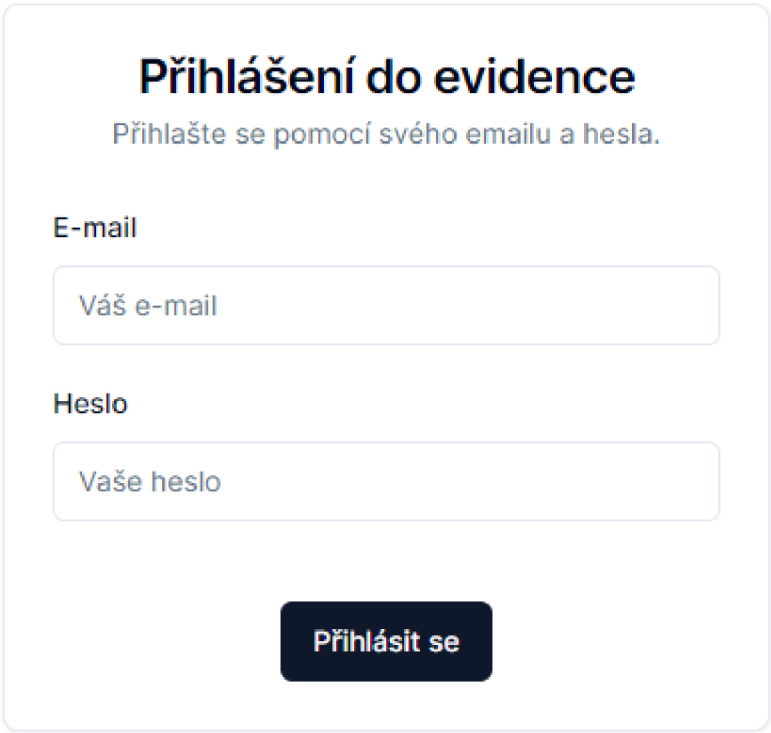
7.4 Seznam použitých zkratek

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
ASP	Active Server Pages
CSS	Card and Socket Services
MVC	Model-view-controller
MVVM	Model-view-viewmodel
PHP	Hypertext Preprocessor
SQL	Structured query language
TCP	Transmission Control Protocol
TV	television
UI	Umělá inteligence
VSCode	Visual Studio Code

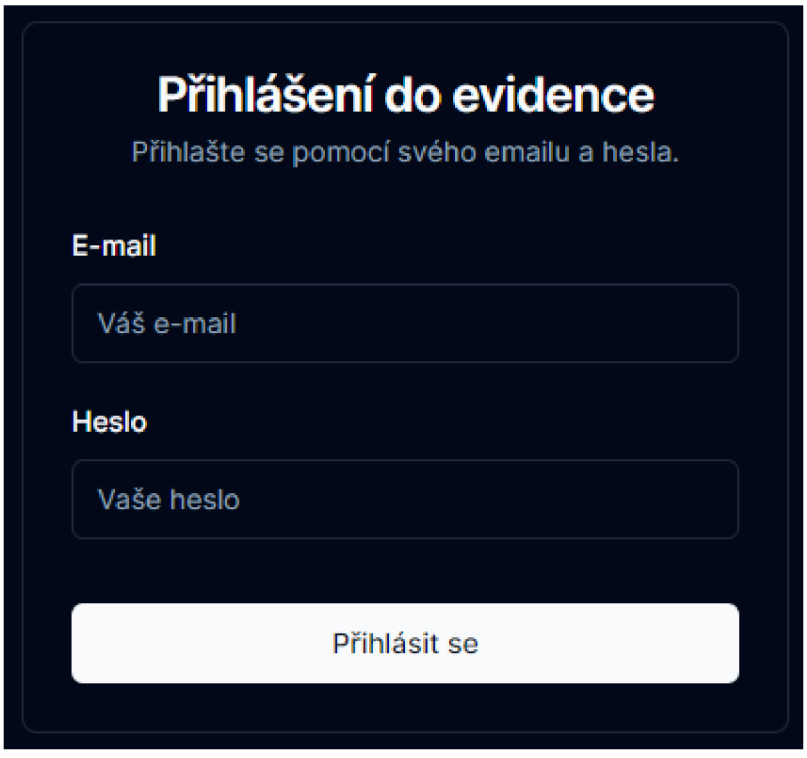
Přílohy

Příloha č. 1- Vytvořený zdrojový kód je dostupný v přiloženém .zip souboru

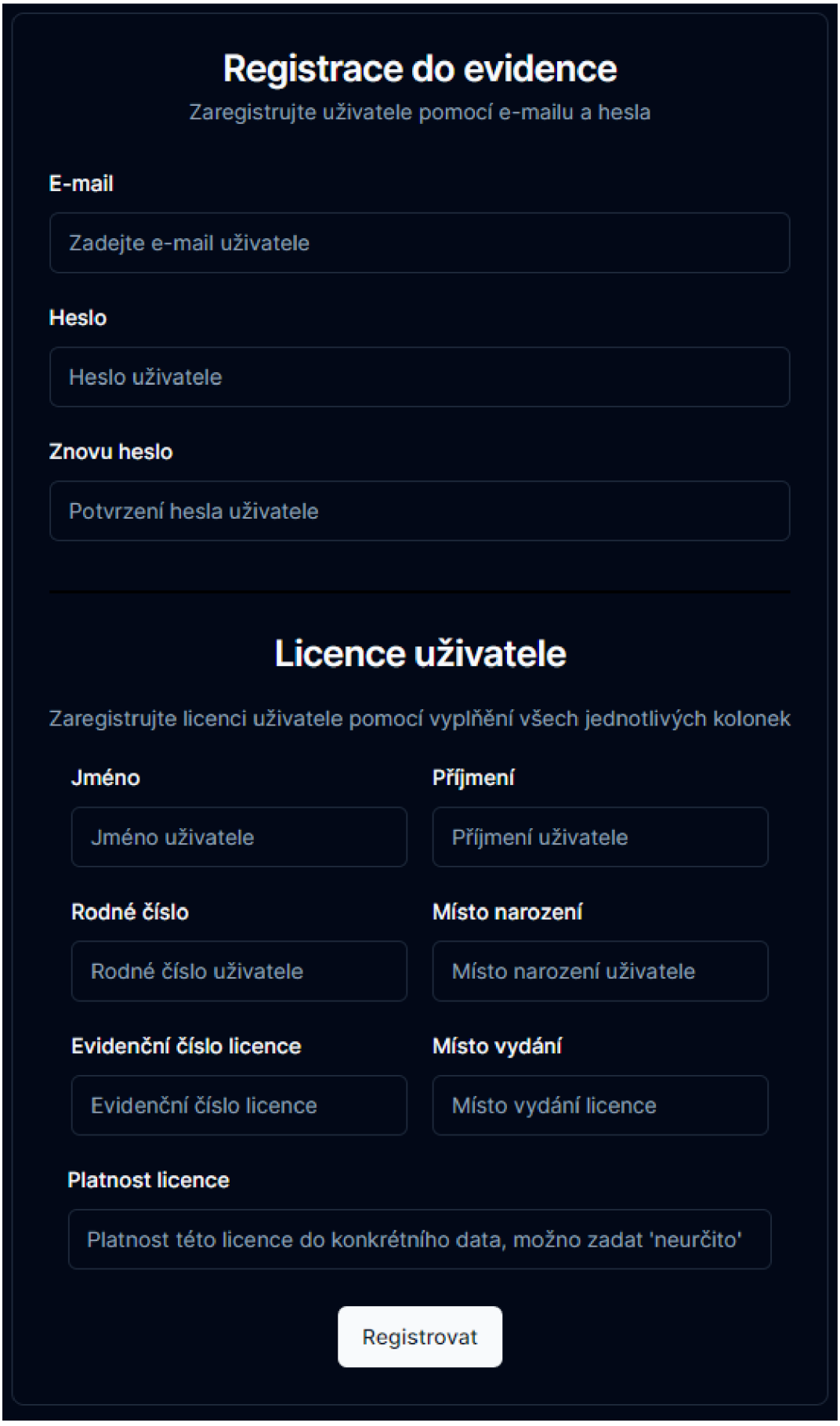
Příloha č. 2 – Formulář přihlášení do evidence (světlý vzhled)



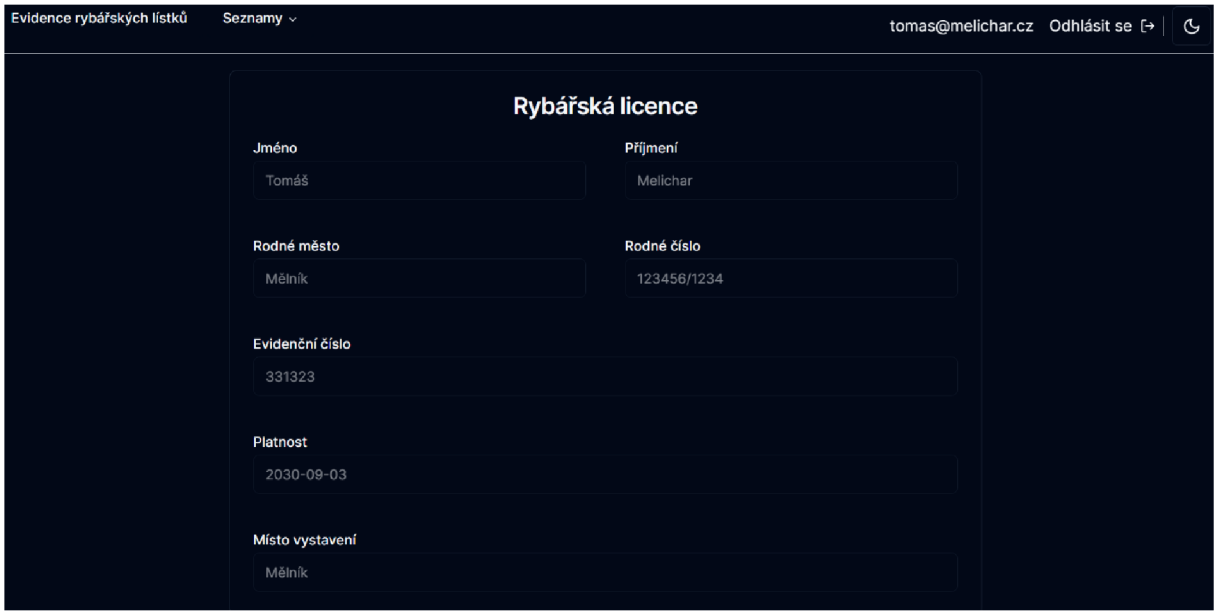
Příloha č. 3 - Formulář přihlášení do evidence (tmavý vzhled)



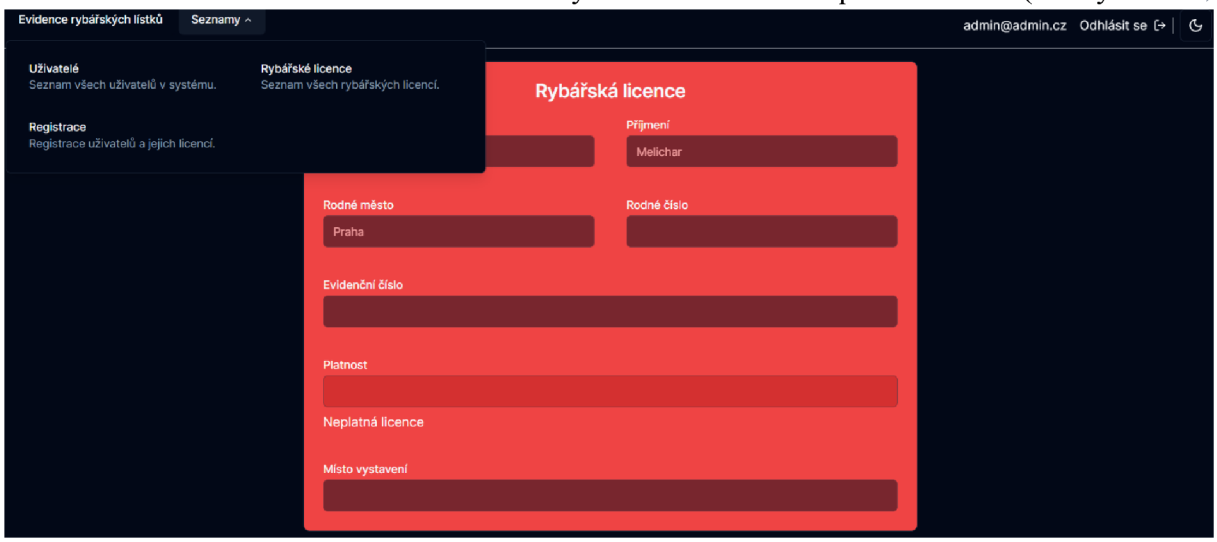
Příloha č. 4 - Formulář registrace uživatele a jeho licence do evidence (tmavý vzhled)



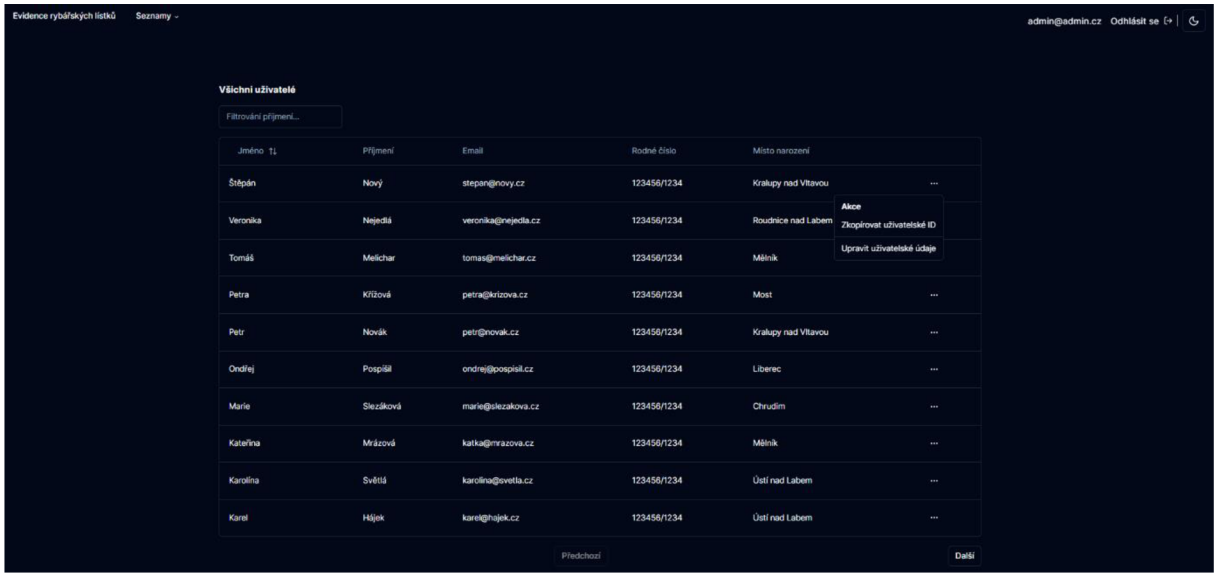
Příloha č. 5 - Domovská stránka s platnou licencí příkladového uživatele (tmavý vzhled)



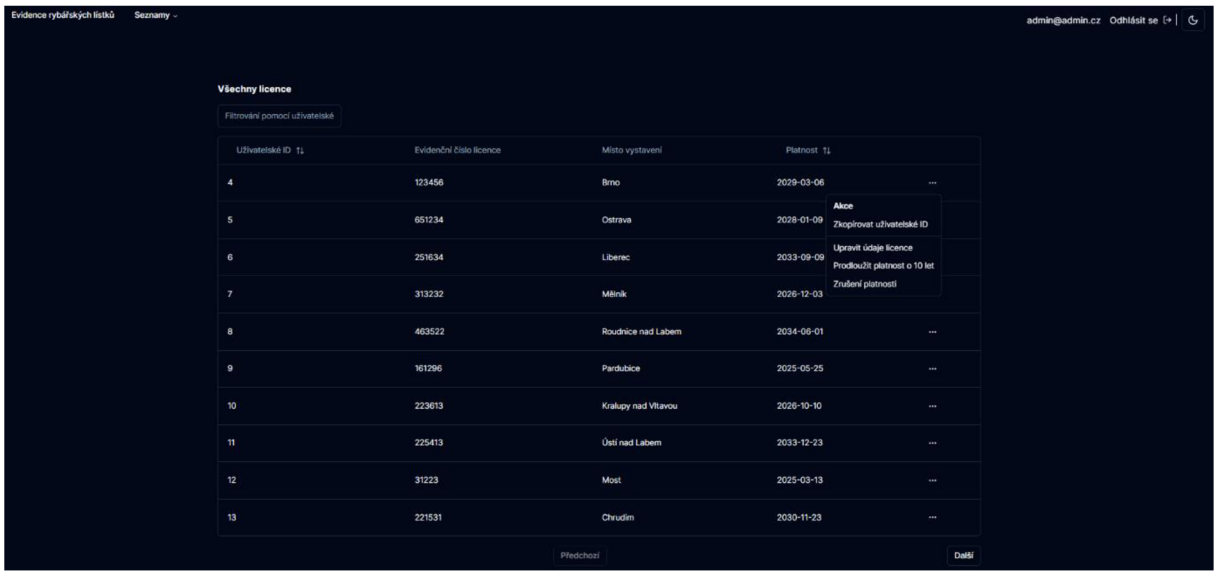
Příloha č. 6 - Příklad zobrazení domovské stránky administrátora bez platné licence (tmavý vzhled)



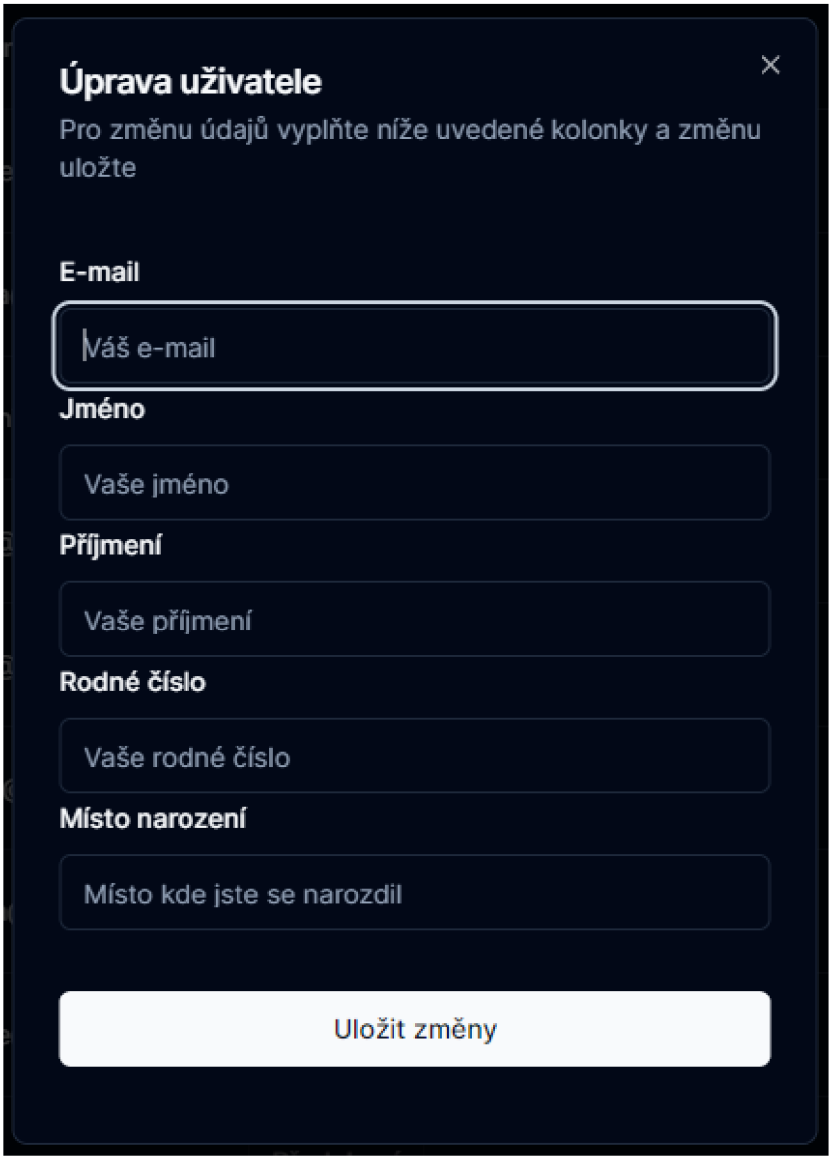
Příloha č. 7 - Data table všech uživatelů a zobrazení možných akcí (tmavý režim)



Příloha č. 8 - Data table všech licencí a zobrazení možných akcí (tmavý režim)



Příloha č. 9 - Akce pro úpravu dat uživatele (tmavý režim)



Příloha č. 10 - Akce pro úpravu dat licence (tmavý režim)