VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MOBILNÍ APLIKACE PRO EVIDENCI STAVEBNÍCH KONTROL

MOBILE APPLICATION FOR BUILDING INSPECTIONS RECORDING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **STANISLAV GABRIŠ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148352

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Gabriš Stanislav**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Mobilní aplikace pro evidenci stavebních kontrol**
Kategorie:	Mobilní aplikace
Akademický rok:	2022/23

Zadání:

1. Prostudujte principy tvorby mobilních aplikací, seznamte se s různými vývojovými prostředími.
2. Analyzujte požadavky na mobilní aplikaci pro Ředitelství silnic a dálnic, která bude umožňovat pracovníkům zadávat dokumentaci o denních kontrolách na jednotlivých částech staveb včetně nahrávání fotografií apod. Zaměřte se také na ošetření situací, kdy na místě bude slabý signál nebo výpadky spojení.
3. Navrhněte mobilní aplikaci dle požadavků. Návrh konzultujte s vedoucím.
4. Navrženou aplikaci implementujte a otestujte na vhodném vzorku dat.
5. Zhodnoťte dosažené výsledky a další možnosti pokračování tohoto projektu.

Literatura:

- McWherter, J., Gowell, S.: Professional Mobile Application Development. John Wiley & Sons Inc, 2012. ISBN 9781118203903.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	24.10.2022

Abstrakt

Cieľom tejto bakalárskej práce je navrhnúť a vytvoriť mobilnú aplikáciu pre ŘSD ČR, ktorá používateľom umožní dokumentovať postup na stavbách a to aj v prípade, že na mieste stavby nie je prístup na internet. Aplikácia je implementovaná pre operačný systém Android v jazyku C# s využitím frameworku .NET MAUI, v ktorom je napísaná samotná aplikácia komunikujúca s databázou pomocou REST API a užívateľským rozhraním napísaným v jazyku XAML. Prínosom tejto práce je digitalizácia dokumentácie na stavbách, ktorá sľubuje zlepšenie efektívnosti a dostupnosti tohto úkonu.

Abstract

The aim of this bachelor thesis is to design and create a mobile application for the ŘSD ČR, that will allow users to document progress on construction sites even in cases where there is no internet connection. The application is implemented for the Android operating system in the C# language using the .NET MAUI framework. The application communicates with the database using a REST API and has a user interface written in XAML language. The benefit of this work is in digitization of documentation on construction sites which aims to improve efficiency and availability.

Kľúčové slová

Mobilná aplikácia, Android, .NET MAUI, C#, Stavebný denník

Keywords

Mobile application, Android, .NET MAUI, C#, Construction site diary

Citácia

GABRIŠ, Stanislav. *MOBILNÍ APLIKACE PRO EVIDENCI STAVEBNÍCH KONTROL.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

MOBILNÍ APLIKACE PRO EVIDENCI STAVEBNÍCH KONTROL

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Vladimíra Bartíka, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Stanislav Gabriš
4. mája 2023

Poďakovanie

Rád by som sa poďakoval vedúcemu práce Ing. Vladimírovi Bartíkovi, Ph.D. za jeho ochotu, usmernenie a odbornú pomoc pri vypracovávaní bakalárskej prace.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

V dnešnej dobe môžeme sledovať snahu digitalizácie v takmer každom jestvujúcom odvetví. Zámerom je väčšinou zjednodušenie a zefektívnenie práce. Populárnymi spôsobmi digitalizácie sú hlavne webové a mobilné technológie, ktoré už v dnešnej dobe umožňujú vytvoriť kvalitné softvérové riešenie väčšiny existujúcich problémov. Jedným z odvetví, v ktorom je taktiež dopyt po digitalizácii je aj stavebné odvetvie. Pri prácach na stavbe existuje množstvo úkonov, ktoré boli dlhodobo vykonávané napríklad papierovou formou a dalo by sa k nim vytvoriť pohodlnejšie softvérové riešenie. Jedným z takýchto úkonov je samotná dokumentácia prác na stavbe riešená v tejto práci.

Cieľom tejto bakalárskej práce je vytvoriť mobilnú aplikáciu pre Ředitelství silnic a dálnic ČR na zariadenia Android, ktorá by používateľom umožnila zefektívniť vytváranie dokumentácie, vo forme stavebného denníka definovaného zákonom, priamo na stavenisku. Aplikácia teda umožní používateľom vyhotoviť záznam dokumentácie jednoducho na ich mobilnom zariadení. Záznam bude okrem textovej časti možné obohatiť aj o fotodokumentáciu použitím galérie alebo fotoaparátu priamo z aplikácie. Tieto záznamy sú odosielané do databázy a používateľ s nimi môže pracovať aj v už existujúcej webovej aplikácii. Niektoré staveniská, ako napríklad mosty, sa ale môžu nachádzať v odľahlých lokáciách, kde nemusí byť prístup k internetovej sieti. Z tohto dôvodu je nutné aby aplikácia používateľov neobmedzovala a dala sa využiť aj v offline móde.

V prvej kapitole tejto práce sú uvedené rôzne existujúce prístupy k vývoju mobilných aplikácií ako aj ich výhody a nevýhody. Druhá kapitola pokračuje v teoretickom základe práce a približuje rôzne softvérové architektonické prístupy súvisiace s prácou, ako aj popis komunikácie mobilného klienta so serverom. V tretej kapitole je detailnejšie rozobraný celkový problém riešený touto prácou, ako aj motivácia za ňou. Sú tu taktiež priblížené pojmy týkajúce sa stavebného odvetvia a mobilnej aplikácie. V neposlednom rade sú tu uvedené existujúce riešenia daného problému. Nasledujúca kapitola bližšie uvádza návrh celkového riešenia a čo by mala aplikácia umožňovať. Najrozsiahlejšia kapitola pojednáva už o samotnej implementácii práce a je rozdelená do logických sekcií podľa architektúry z návrhu. Pred záverom posledná kapitola popisuje priebeh a dôsledky testovania aplikácie na používateľoch počas jej vývoja.

Kapitola 2

Tvorba mobilných aplikácií

Táto kapitola popisuje existujúce prístupy k vývoju mobilných aplikácií pre systémy Android a iOS, ktoré tvoria približne 72% a 27% podielu trhu mobilných operačných systémov k decembru roku 2022 (dáta prevzaté z [13]). Je tu opísaný rozdiel medzi vývojom priamo na jednu platformu (tzv. natívny vývoj) a vývojom multiplatformným. Ukazuje výhody a nevýhody jednotlivých prístupov a konkrétne technológie, ktoré tieto prístupy využívajú. Medzi týmito technológiami je opísaná aj technológia *.NET MAUI*, ktorá bola nakoniec zvolená aj pri implementácii práce.

2.1 Natívny vývoj

Tento pojem sa vzťahuje k vývoju produktu priamo na jeden určitý systém. Na vývoj sa využíva systémovo špecifický programovací jazyk. V prípade Androidu je to Java alebo Kotlin, v prípade systému iOS jazyky Objective-C a Swift. Natívne aplikácie sú známe svojou rýchlosťou a responzivitou čo je spôsobené priamim prísupom k natívnym funkciám systému, ktoré môžu naplno využiť. [12]

Jeho hlavnou nevýhodou, keďže sa kód nedá recyklovať, je potreba vývoja viacerých produktov pre každú konkrétnu platformu zvlášť, čo zvyšuje čas potrebný na vývoj a taktiež cenu. Tento prístup je preto najvhodnejší pre aplikácie, ktoré vyžadujú veľkú rýchlosť, minimálnu odozvu a majú komplexnú funkcionalitu. [12]

2.1.1 Android

Android je open-source operačný systém, postavený na jadre Linuxu, využívaný hlavne pre mobilné zariadenia a tablety. Bol vyvinutý firmou Google a prvýkrát vydaný v roku 2008 pod názvom Android 1.0. Android poskytuje prostredie a nástroje pre developerov, kde môžu pomocou jazyka Java vytvárať mobilné aplikácie a prispôsobovať ich pre rôzne zariadenia a rôzne veľkosti obrazoviek. Tieto aplikácie sa následne dajú stiahnuť z Obchodu Google Play alebo od iných poskytovateľov. Tieto aplikácie môžu poskytovať rôznu funkcionalitu od zábavy a hier, cez sociálne aplikácie, aplikácie zobrazujúce počasie a využívajúce GPS, a pod. [5]

Android využíva virtuálny systém *Dalvik*, optimalizovaný pre mobilné zariadenia, ktoré majú obvykle menšie technologické možnosti, pre beh aplikácií, každá vo svojom procese s vlastnou instanciou virtuálneho systému. [5]

Ku koncu roku 2022 bol Android lídrom v trhu mobilných operačných systémov s podielom viac ako 70% a je teda najpoužívanejším systémom v tejto oblasti na svete. [13]

Kotlin je programovací jazyk vyvíjaný firmou JetBrains ako alternatíva jazyku Java pre vývoj aplikácií na Android. Predstavený bol v roku 2011 a v roku 2012 bol uvedený ako open-source. Oficiálna verzia 1.0 bola oznámená v roku 2016 zatiaľ čo v roku 2017 dostal oficiálnu podporu od firmy Google v ich vývojovom prostredí Android Studio. [5]

Java virtual machine (skratka JVM) je zodpovedná za interpretáciu kompilovaného Java kódu. JVM je schopné interpretovať súbory aj z iných jazykov, ak je dodržaný požadovaný formát. Práve toto využíva Kotlin a produkuje súbory, ktoré JVM dokáže taktiež interpretovať. [5]

Úspech jazyka Kotlin môže byť pripísaný elegantnejšiemu syntaxu a prvkami moderných programovacích jazykov. Taktiež je braný ohľad na bezpečnosť a Kotlin dokáže takmer eliminovať napríklad *NullPointerException*, ktorá je v jazyku Java najrozšírenejšia. Kotlin je taktiež 100% kompatibilný s jazykom Java, čo dovoľuje využívať aplikačné rámce a rozhrania aj z tohto jazyka. [5]

2.1.2 iOS

Operačný systém iOS bol vyvinutý firmou Apple a to hlavne pre mobilné zariadenia ako iPhone a iPad. Bol navrhnutý s užívateľsky prívetivým rozhraním s ohľadom aj na bezbečnosť a ochranu uživateľských súkromných informácii. [7]

Okrem vstavaných aplikácií a funkcií ako *Siri*, *iMessage* alebo *FaceTime*, poskytuje iOS širokú selekciu nástrojov pre vývojárov na vytváranie rôznorodých aplikácií. Výsledkom je rozmanitý ekosystém aplikácií tretích strán od hier po biznis aplikácie.

iOS sa stal populárnym a všestranným operačným systémom, ktorý zmenil spôsob akým interagujeme s mobilnými zariadeniami a ku koncu roku 2022 zaberá okolo 27% podielu trhu. [13]

Swift je programovací jazyk od firmy Apple, navrhnutý pre rýchlosť, bezpečnosť a jednoduchosť. Primárne je využívaný pre vývoj softvéru na platformy od firmy Apple ako macOS, iOS a pod. Ako open-source bol na trh uvedený v roku 2015 a odvtedy získaval na popularite. Jednou z jeho veľkých výhod je možnosť využívať prvky jazyka Objective-C, ktorý bol predchodcom jazyka Swift. [3]

Celkovo je Swift moderným a všestranným programovacím jazykom určeným na vývoj pre platformy Apple, ktorý figuruje ako atraktívna možnosť pre vývojárov tvoriacich kvalitné a spoľahlivé aplikácie napríklad pre zariadenia iPhone. [3]

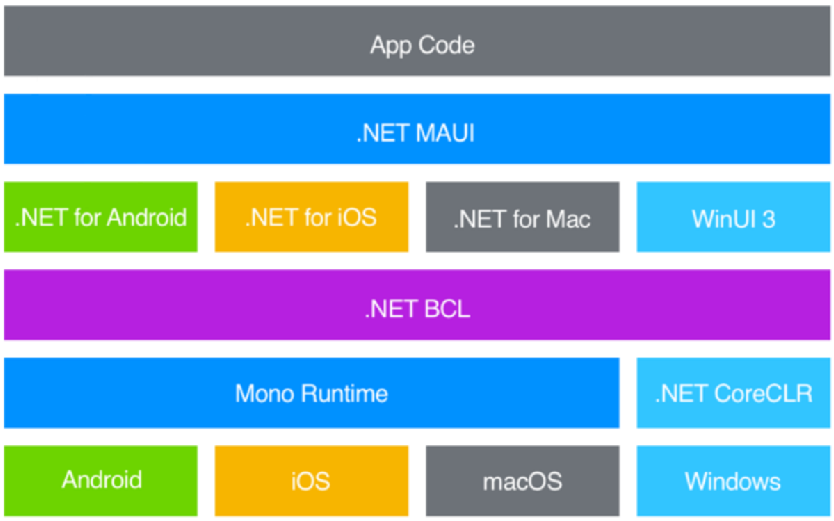
2.2 Multiplatformný vývoj

Pri multiplatformnom prístupe je vývojárom umonžnené vyvíjať aplikácie, ktoré budú fungovať na viacero platformách no so zdieľaným kódom. Využité sú pri tom multiplatformné frameworky, ktoré abstrahujú špecifické knižnice. Priklady populárnych frameworkov sú *Xamarin* (.NET MAUI) od Microsoftu s jazykom C# a XAML, *Flutter*[1] od Googlu s jazykom Dart alebo *React Native*[2] od Mety s JavaScriptom, ktorých kódy sa prekladajú do natívneho kódu pre konkrétnu platformu, a teda využívajú natívne prvky. [12]

Výhody tohto prístupu sú napríklad menej potrebného času na vývoj pretože kód sa dá recyklovať, jednoduchšia údržba, keďže existuje len jeden zdrojový kód, čo taktiež umožňuje aktualizovať aplikáciu zároveň na všetkých platformách. Nevýhodou je väčšinou väčšia a o niečo pomalšia aplikácia, nemožnosť využiť všetky systémovo špecifické funkcie pri vývoji, atp. [12]

2.2.1 Xamarin a .NET MAUI

Tieto technológie, vyvinuté spoločnosťou Microsoft, sú určené na vývoj multiplatformných aplikácií v jazyku C# a XAML. MAUI (skratka pre *Multi-platform App User Interface*) oficiálne vydaná v roku 2022 nahradila Xamarin ako jeho nástupca. Okrem vývoja aplikácií pre zariadenia Android a iOS taktiež podporuje vývoj pre systémy Windows a macOS, všetko zo zdieľaného kódu. Ako nástupca Xamarin s ním mnohé funkcie zdieľa, stavia na nich a vylepšuje. Okrem spoločného kódu umožňuje aj prístup k systémovo špecifickému kódu. [4]



Obr. 2.1: Architektúra .NET MAUI aplikácie (prevzaté z [4])

[1] https://flutter.dev/
[2] https://reactnative.dev/

Na obrázku 2.1 je znázornená architektúra .NET MAUI. V dolnom riadku obrázka sa nachádzajú jednotlivé platformy, ktoré MAUI podporuje. Nad tým sa nachádzajú *Behové prostredia*, na ktorých sú jednotlivé programy, vyvinuté v .NET, spustené na konkrétnych platformách. .NET *Base Class Library* (BCL), ako kolekcia základných tried, rozhraní a typov zahrnutých v .NET aplikačnom rámci, poskytuje základ pre aplikácie a umožňuje zdielaný kód (logiku) medzi platformami tým, že abstrahuje detaily jednotlivých platforiem. Každá platforma ale definuje užívateľské rozhranie rôzne a .NET umožňuje využívať aj tieto platformovo špecifické rámce (.NET pre Android, .NET pre iOS, atď.), no ako bolo už opísané v úvode podkapitoly 2.2, tento prístup vyžaduje udržiavanie zdrojového kódu pre každú platformu zvlášť. Tu nastupuje .NET MAUI, ktoré poskytuje spoločný aplikačný rámec pre vytváranie užívateľských prostredí na rôzne platformy. Na vrchole obrázka sa už nachádza samotný kód využívajúci predošlé stupňe, primárne teda stupeň .NET MAUI a .NET BCL. [4]

Aplikácie .NET MAUI sa kompilujú rôzne. Pre Android sa z jazyka C# vytvorí medzi jazyk, z ktorého sa potom pomocou *just in time* kompilácie vytvorí natívny kód. Pre zariadenia iOS sa zas využíva *ahead of time* kompilácia z jazyka C# do natívneho jazyka symbolických inštrukcií. Pre ostatné podporované platformy sa využívajú ďalšie metódy. [4]

.NET MAUI poskytuje niekoľko prvkov na zobrazovanie dát, indikovanie aktivity, podporuje viacero typov stránok a spôsobov navigácie. Umožňuje využívať dátové väzby (*data binding*) a upravovať správanie prvkov grafického rozhrania. Taktiež ponúka prístup k natívnym funkciám akými sú rôzne senzory, úložisko alebo aj sieťová konektivita. Funkcia *Hot reload* poskytuje možnosť editácie zdrojoveho kódu alebo súborov XAML a aplikácie týchto zmien bez podtreby rekompilácie. Toto všetko je poskytnuté abstrakciou platformovo špecifických implementácií do jednoho multiplatformového projektu. [4]

C# je moderný, *objektovo orientovaný* a silne typovaný programovací jazyk, ktorý vývojárom umožňuje vytvárať bezpečné a robustné aplikácie nad .NET. Svoje základy má v C jazykoch a je teda podobný jazykom ako C, C++ alebo Java. [15]

V jadre je C# objektovo orientovaný jazyk, kde si vývojár definuje typy a ich správanie, no je vyvíjaný s podporou aktuálnych praktík vývoja softvéru. [15]

Medzi prvky tohto jazyka patrí napríklad *Garbage collection*, alebo automatické spravovanie pamäti, kde sa automaticky uvoľnuje pamäť nevyužívaných objektov. Spravovanie výnimiek umožňuje detekciu potencionálnych chýb a reágovanie na ne. *Language Integrated Query* (LINQ) syntax poskytuje jednotný formát pre prácu s dátami. Medzi ďalšie prvky patria napríklad *Lambda* výrazy, asynchrónne operácie alebo nulovateľné typy. [15]

Architektúra .NET sa skladá z knižníc a *Common Language Runtime* (CLR), pričom CLR je implementáciou medzinárodného štandardu *Common Language Infrastructure* (CLI). Zdrojový kód napísaný v jazyku C# je zkompilovaný do medzi jazyku (IL), ktorý spĺňa špecifikáciu CLI a je uložený v súboroch s príponou dll. Pri spustení programu sa tento jazyk načíta do CLR, ktoré ho prevedie pomocou JIT do natívneho strojového kódu. CLR taktiež poskytuje garbage collection, správu zdrojov a správu výnimiek a interoperabilitu iných .NET jazykov ako napríklad F# alebo Visual Basic. [15]

Kapitola 3

Architektúra

Architektúra softvérových systémov sa zameriava na dizajn a organizáciu softvéru. Určuje ako medzi sebou jednotlivé časti softvéru interagujú a aké sú ich povinnosti. Cieľom architektonických vzorov je zlepšiť udržiavateľnosť aplikácií, spraviť ich prehladnejšie a jednoducho rozširovateľné. Aplikácia navrhnutá pomocou určitej overenej architektúry má väčšiu pravdepodobnosť byť zrozumiteľnejšia a analyzovateľnejšia. V súčasnosti existuje niekoľko architektúr, každá so svojimi výhodami a nevýhodami. [11]

Táto kapitola opisuje princípy a architektúry, ktoré súvisia s implementáciou projektu. Sú tu opísané pojmy ako *architektúra Klient-Server*, *Viacvrstvová architektúra* (anglicky Layered architecture alebo aj N-Layer architecture), návrhový vzor *Model-View-ViewModel* (skratka MVVM). Priblížené sú aj pojmy z oblasti komunikácie ako *REST API* a *OpenAPI*.

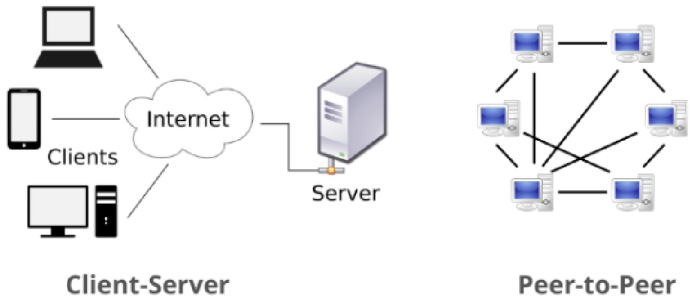
3.1 Klient-Server

Architektúra klient-server je populárny spôsob organizácie softvérových aplikácií, ktoré využívajú sieť. Ako je možné vidieť na obrázku 3.1, v tejto architektúre jeden alebo viac klientov, ako sú webové prehliadače alebo mobilné zariadenia a aplikácie na nich, komunikuje s centrálnym serverom cez sieť. Server poskytuje služby, údaje alebo iné zdroje klientom, zatiaľ čo klienti posielajú serveru požiadavky na prístup k týmto službám a zdrojom. [11]

Architektúra klient-server umožňuje oddelenie úloh medzi klientom a serverom, čo uľahčuje vývoj a údržbu veľkých softvérových systémov. Klient je zodpovedný za prezentáciu užívateľského rozhrania a spracovanie užívateľských vstupov, zatiaľ čo server je zodpovedný za spracovanie dát a vykonávanie zložitejších výpočtov. Komunikácia medzi klientom a serverom je typicky sprostredkovaná protokolom, ako je *HTTP*, ktorý umožňuje efektívny a bezpečný prenos dát cez sieť. [11]

Pre porovnanie alternatívna architektúra *Peer-to-Peer* (obrázok 3.1) má decentralizovaný model kde sú si všetky zariadenia rovné a môžu slúžiť ako klient a aj ako server.

Medzi výhody tejto architektúry patrí napríklad bezpečnosť, keďže server spravuje a kontroluje prístup k zdrojom a zaisťuje tak, že k nim môže pristúpiť len autorizovaný užívateľ. Medzi nevýhody naopak patrí riziko chyby na strane serveru čo môže vyústiť do znefunkčnenia celého systému, keďže klienti nemôžu pristúpiť k údajom.

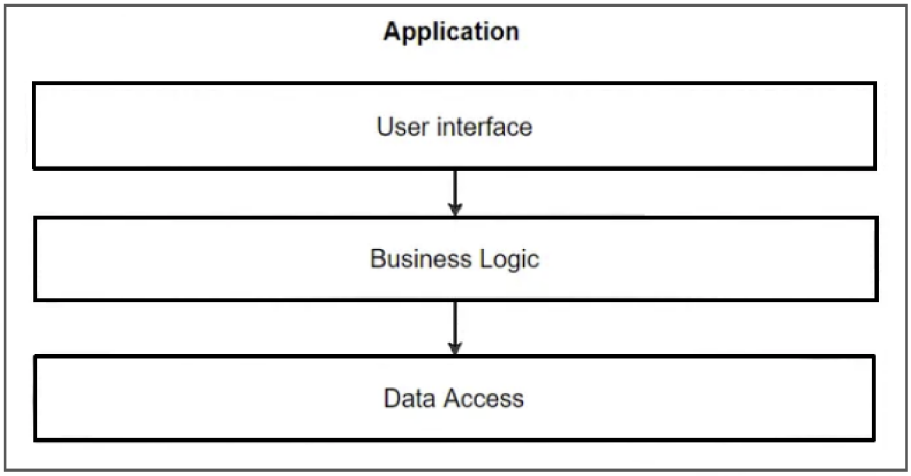


Obr. 3.1: Porovnanie princípu architektúry Klient-Server a Peer-to-Peer (zdroj https://www.networkstraining.com/peer-to-peer-vs-client-server-network/)

3.2 Viacvrstvová architektúra

Tento typ architektúry využívaný pri návrhu softvérových systémov sa vyznačuje oddelením funkčnosti systému do samostatných vrstiev, pričom každá vrstva vykonáva špecifickú funkciu a komunikuje so susednými vrstvami vopred definovaným spôsobom. Každá vrstva je zodpovedná za špecifickú časť aplikácie a vrstvy sú usporiadané v logickom poradí zhora nadol, pričom každá vrstva nadväzuje na vrstvu pod ňou. Takéto delenie robí systém modulárnejším, čo zľahčuje jeho pochopiteľnosť a údržbu. [9]

Vo viacvrstvovej architektúre, ako je vidieť na obrázku 3.2, je najvyššou vrstvou zvyčajne prezentačná vrstva, ktorá je zodpovedná za prezentáciu informácií užívateľovi a taktiež prijímanie užívateľských vstupov. Prezentačná vrstva komunikuje s vrstvou pod ňou, čo je zvyčajne vrstva *obchodnej logiky* (anglicky business layer). Vrstva obchodnej logiky je zodpovedná za implementáciu logiky, výpočtov a procesov aplikácie. Komunikuje s vrstvou pod ňou, čo je zvyčajne vrstva prístupu k údajom (anglicky data access layer), ktorá je zodpovedná za prístup a manipuláciu s údajmi z dátových úložísk aplikácie. [9]



Obr. 3.2: Reprezentácia viacvrstvovej architektúry (prevzaté z [9])

Vrstvená architektúra je flexibilný vzor architektúry, ktorý umožňuje rôzne implementácie, ako je pridávanie alebo odstraňovanie vrstiev alebo pridávanie ďalších funkcií do vrstiev. [9]

Celkovo je viacvrstvová architektúra vzor softvérovej architektúry využívaný pri návrhu webových, podnikových a mobilných aplikácií, ktorý oddeľuje funkčnosť systému do odlišných vrstiev s rôznymi funkciami, nazývané aj Separation of Concerns (skratka SoC), čo má za následok zrozumiteľnejšiu štruktúru, jednoduchšiu údržbu a flexibilitu systému. [9]

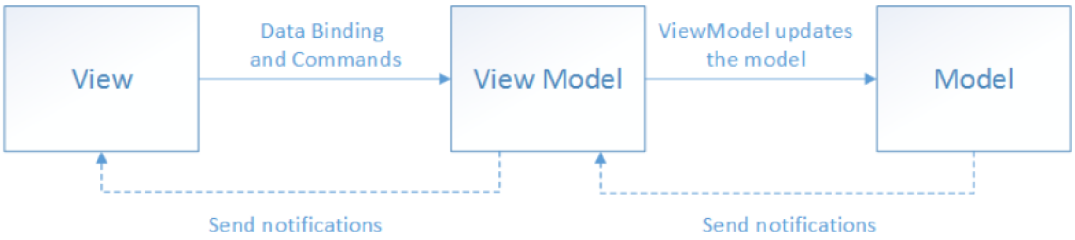
3.3 Model-View-ViewModel

Model-View-ViewModel (skratka MVVM) je návrhový vzor, využívaný pre oddelenie grafického rozhrania (skratka *GUI* – z anglického Graphical User Interface) od logiky aplikácie. Ako je znázornené na obrázku 3.3 Skladá sa z troch komponent, *Model*, *View* a *ViewModel*, kde každá má svoju úlohu. [14]

- *Model* je časť, v ktorej sú obsiahnuté dáta aplikácie a nevie o existencii iných komponent. Niekedy je tu zahrnutá aj obchodná (anglicky business) logika aplikácie. [14]
- *View* predstavuje prezentačnú vrstvu aplikácie, teda grafické rozhranie a prvky, ktoré užívateľ vidí, ako napríklad tlačidlá, textové polia alebo iné grafické prvky. Je zodpovedný za grafickú reprezentáciu dát Modelu a prijímanie vtupov od používateľa no neobsahuje logiku. View vie o existencii ViewModelu. [14]
- *ViewModel* je medzivrstvou, ktorá sprostredkúva interakciu medzi komponentami View a Model bez toho aby malo vedomosť o existencii časti View. View sa naviaže pomocou ViewModelu na Model a pri zmenách Modelu sa môže automaticky aktualizovať. ViewModel okrem naväzovania obsahuje aj logiku prezentačnej vrstvy, ako je napríklad validácia vstupov alebo formátovanie. [14]

Jednou z kľúčových výhod návrhového vzoru MVVM je, že rozdeľuje úlohy do vrstiev. Oddelením GUI od dát a obchodnej logiky môžu vývojári vytvárať opakovane použiteľné a testovateľné komponenty, ktoré sa ľahšie udržiavajú a aktualizujú. Vzor tiež zlepšuje škálovateľnosť a výkon tým, že umožňuje vývojárom optimalizovať každý komponent nezávisle. [14]

V praxi sa tento vzor typicky implementuje pomocou dátovej väzby (anglicky *data binding*, viď obrázok 3.3), čo je technika, ktorá umožňuje automatickú synchronizáciu dát medzi komponentami View a Model. Pri zmene údajov na strane Modelu, ViewModel automaticky aktualizuje View bez potreby manuálneho zásahu vývojára. [14]



Obr. 3.3: Grafická reprezentácia modelu MVVM (prevzaté z [14])

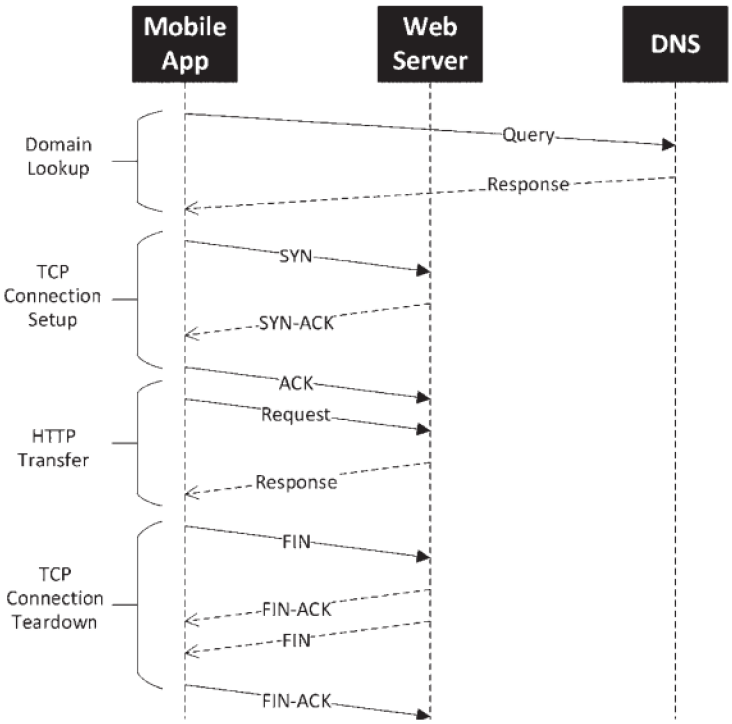
3.4 Komunikácia

Typicky má komunikácia klienta so serverom podobu požiadavku a odpovedi, kde klient posiela požiadavok vo formáte, ktorému server rozumie a čaká na odpoveď, zatiaľ čo server spracuje požiadavok a odosiela odpoveď. Na túto komunikáciu sa využíva niekoľko protokolov a server *DNS*[1]. Sú to napríklad protokoly *UDP*[2], *TCP*[3] a v neposlednom rade protokol *HTTP*[4]. Každý z týchto protokolov má pri komunikácii inú funkciu. [11]

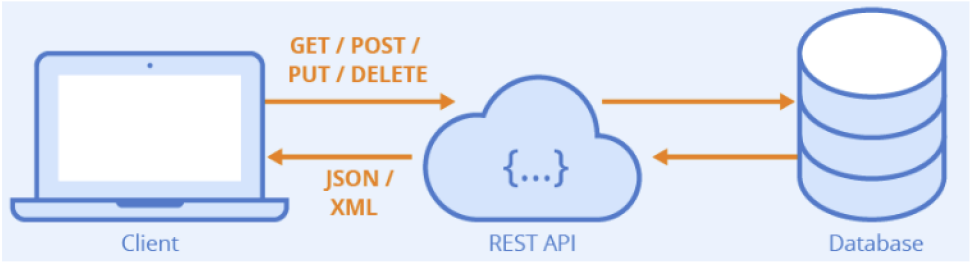
- *DNS* (skratka z anglického Domain Name Server) je využívaný pre preklad domén serverov na IP adresy v prípade, že ich klient nepozná čo klientovi umožní nadviazať spojenie so severom. [11]
- *UDP* (skratka z anglického User Datagram Protocol) je protokol, ktorým klient komunikuje so serverom DNS bez toho aby bolo potrebné nadviazať vzájomné spojenie. Server na klientovu požiadavku odpovedá zaslaním IP adresy priradenej k serveru. [11]
- *TCP* (skratka z anglického Transmission Control Protocol) je protokol, ktorý slúži na zaistenie spojenia medzi konkrétnym serverom a klientom za pomoci tzv. *three-way handshake*. Po nadviazaní spojenia si môžu klient a server navzájom posielať dáta. TCP spojenia sú *plne duplexné* (anglicky full-duplex), čo umožňuje vysielať aj prijímať dáta ako klientovi tak serveru súčasne. [11]
- *HTTP* (skratka z anglického Hypertext Transfer Protocol) sa už používa na prenos konkrétnych dát (požiadaviek a odpovedí) medzi klientom a serverom. Existuje niekoľko špecifikácií (napríklad HTTP/1.1) s podporou niekoľkých metód požiadaviek, ako napríklad GET, POST, PUT a DELETE, a rovnako existuje štandard pre typy odpovedí serveru, označené číselným kódom, napríklad 200 (OK), 403 (Forbidden) alebo 500 (Internal Server Error). [11]

Na obrázku 3.4 je znázornený priebeh spojenia a komunikácie klienta so serverom. Na začiatku, pokiaľ klient nepozná IP adresu serveru, kontaktuje server DNS pomocou protokolu UDP a názvu domény (krok Domain Lookup). Po získaní IP adresy je pomocou protokolu TCP naviazané spojenie s konkrétnym serverom (krok TCP Connection Setup). Po naviazaní TCP spojenia, si klient a server môžu navzájom pomocou protokolu HTTP vymieňať dáta (požiadavky a odpovede) (ktrok HTTP Transfer). Na ukončenie tohto TCP spojenia sa využíva paket s príznakom *FIN* (krok TCP Connection Breakdown). [11]

[1] RFC 1035
[2] RFC 768
[3] RFC 793
[4] RFC 2616 (doplnený novšími špecifikáciami)



Obr. 3.4: Priebeh komunikácie medzi klientom, serverom a DNS serverom (prevzaté z [11])



Obr. 3.5: Znázornenie využitia REST API ako komunikačného rozhrania (zdroj https://www.seobility.net/en/wiki/REST_API)

3.4.1 Representational state transfer

S vývojom webových technológií rástla aj potreba vytvoriť štandard, ktorý by popisoval chovanie a komunikáciu medzi jednotlivými časťami webu. Z tejto potreby vzišiel architektonický štýl s názvom Representational state transfer (skratka REST). Cieľom REST je poskytnúť jednotné rozhranie pre prístup a manipulovanie s prostriedkami (anglicky resources), zväčša za použitia protokolu HTTP (taktiež popísaný v podkapitole 3.4). [6]

REST je vo svojej podstate súbor pravidiel, ktoré definujú, ako môžu klienti interagovať so serverom, napríklad prostredníctvom HTTP požiadaviek a odpovedí. Pri HTTP sú využívané metódy ako GET, POST, PUT alebo DELETE (viď obrázok 3.5). Klient môže napríklad použiť metódu GET na získanie informácií o zdroji, metódu POST na vytvorenie nového zdroja alebo metódu PUT na aktualizáciu existujúceho zdroja. [6]

Jednou z kľúčových vlastností rozhraní REST je, že sú bezstavové, čo znamená, že server neuchováva žiadne predchádzajúce informácie z komunikácie s klientom. Namiesto toho každá požiadavka obsahuje všetky potrebné informácie, aby server porozumel a spracoval požiadavku, a každá odpoveď obsahuje všetky potrebné informácie, aby klient odpovedi porozumel a spracoval ju. [6]

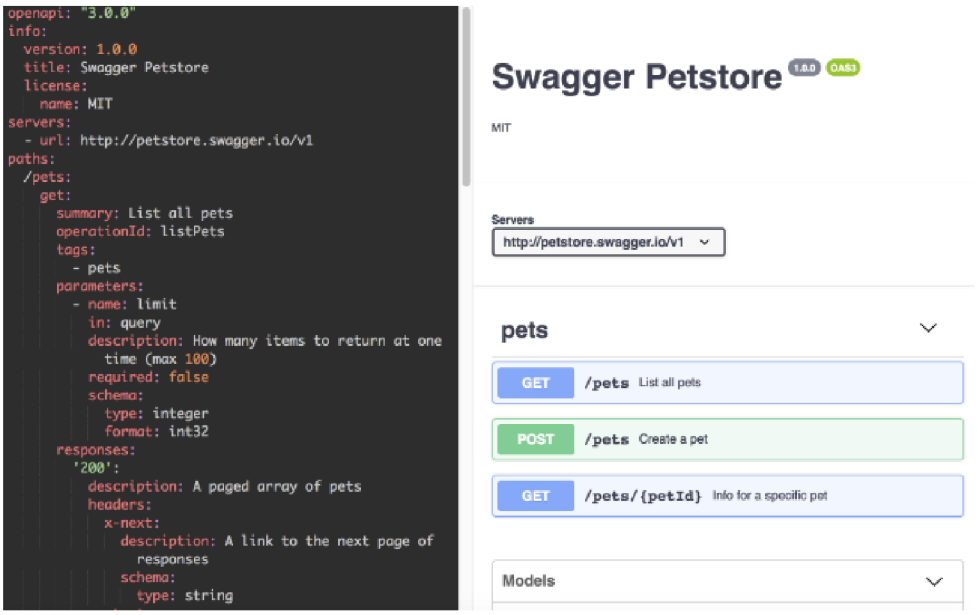
Okrem štandardizácie rozhrania pre prístup a manipuláciu so zdrojmi používajú REST API aj štandardizovaný dátový formát na reprezentáciu zdrojov, kde sa stali najpopulárnejšími *JSON* alebo *XML*. To umožňuje klientom jednoducho analyzovať a manipulovať s údajmi zo servera a zabezpečuje interoperabilitu medzi rôznymi systémami a programovacími jazykmi. [6]

3.4.2 OpenAPI

OpenAPI, predtým tiež známi ako *Swagger*, je štandard pre definíciu, dokumentovanie a testovanie rozhraní REST API. Poskytuje ľudsky a strojovo čitateľný formát na popis koncových bodov (anglicky *endpoint*), parametrov a odpovedí rozhrania, ako aj ďalšie metadáta (podporované typy médií, požiadavky na autentifikáciu, atď.). [10]

Dokumenty typu OpenAPI sú písané v jazyku *JSON* alebo *YAML*. Tieto dokumenty môžu využiť vývojári alebo programi na to určené na vygenerovanie knižníc, grafického rozhrania alebo klientov REST API. Využitie špecifikácie OpenAPI zvyšuje interoperabilitu a použiteľnosť rozhraní, keďže sa dajú využiť pri vývoji v rôznych programovacích jazykoch. [10]

Jednou z kľúčových funkcií OpenAPI je schopnosť generovať z neho interaktívnu dokumentáciu (viď obrázok 3.6) pre REST API, ktorú môžu použiť vývojári na skúmanie a testovanie jej koncových bodov. Dokumentáciu OpenAPI možno použiť aj na generovanie testovacích prikladov rozhrania [10], čo môže pomôcť zabezpečiť správnosť a spoľahlivosť REST API.



Obr. 3.6: Príklad dokumentácie REST API podľa špecifikácie OpenAPI (ľavá strana obrázku) a grafická reprezentácia tohto dokumentu (pravá strana obrázku).
Snímka obrazovky zo stránky https://editor.swagger.io/

Základnú štruktúru dokumentu OpenAPI tvoria metadáta, kde patrí napríklad verzia špecifikácie, popis daného rozhrania alebo jeho názov. Ďalej sa tu nachádzajú servery (adresa na ne) a popis k nim. Konkrétne koncové body sú pod záložkou `paths` a k nim sú vypísané jednotlivé opéracie, ktoré podporujú, a taktiež možné odpovede zo strany serveru vo formáte kód odpovedi a schéma dát. V prípade, že operácia používa parametre, pridá sa sekcia `parameters`. Popis parafrázovaný z a bližšie informácie ku všetkým prvkom na [10].

Kapitola 4

Problém a existujúce riešenia

Vývoju softvéru často predchádza konkrétny problém, ktorý je treba vyriešiť. V tejto kapitole bude vysvetlený problém, ktorý motivoval vytvorenie tohto softvéru ako aj existujúce riešenia vedenia stavebného denníku, ktoré inšpirovali vývoj. Nachádza sa tu aj vysvetlenie pojmov, súvisiacich s oborom, ktorým sa riešenie zaoberá, ako napríklad čo je to *stavba* alebo *stavebný denník*.

V prvom rade je načrtnutý riešený problém, jeho rozsah a dopad. Ďalej je zhrnutá motivácia, teda prečo je táto aplikácia vyvíjaná, aký je jej hlavný cieľ a zámer. Uvedené sú pojmy využívané v týchto opisoch pre ich lepšie pochopenie. Preskúmané sú niektoré existujúce riešenia, ich silné a slabé stránky a nakoniec je z týchto informácií vyvedený návrh riešenia v kapitole 5. Od tejto kapitoly budú v značnej miere používané výrazy *zápis* a *záznam*, ktoré ale majú rovnaký význam a teda sú to synonymá.

4.1 Motivácia

Stavebný priemysel sa do značnej miery spolieha na presnú a včasnú evidenciu pri postupe stavieb, ktorá je nevyhnutná na sledovanie míľnikov projektu, hodnotenie kvality a alókaciu zdrojov. Mimo tejto internej potreby stavebých firiem, je táto potreba dokumentovať stavby a práce na stavbách v Českej republike daná zákonom[1]. Tradičné metódy vedenia evidencie, ako napríklad papierové formuláre, však môžu byť časovo náročnejšie, náchylné na chyby a ťahšie aktualizovateľné. Keďže sa mnohé staveniská nachádzajú v odľahlých oblastiach s obmedzeným prístupom k internetovému pripojeniu, webové riešenia sa nedajú plne využiť v teréne.

Po zvážení týchto obmedzení a výziev, vzišla motivácia vytvoriť mobilnú aplikáciu pre Ředitelství silnic a dálnic ČR, ktorá používateľom umožní vykonávať evidenciu na stavbách, vo forme *stavebného denníka* priamo na mieste z mobilného telefónu a to aj za neprítomnosti internetového pripojenia. Aplikácia má tak eliminovať niektoré nedostatky existujúcich riešení, a tým uľahčiť a zefektívniť prácu pre používateľov zodpovedných za dokumentáciu.

[1] Zákon č. 183/2006 Sb. (Stavební zákon) a Vyhláška o dokumentaci staveb č. 499/2006 Sb.

4.2 Stavebný denník a stavba

Výsledná aplikácia spĺňa pri dokumentácii a evidencii na stavbách funckiu stavebného denníka, preto je vhodné uviesť tento pojem a taktiež čo to vôbec je stavba. Oba tieto pojmy sú zároveň stanovené v stavebnom zákone Českej republiky.

> „*Stavbou se rozumí veškerá stavební díla, která vznikají stavební nebo montážní technologií, bez zřetele na jejich stavebně technické provedení, použité stavební výrobky, materiály a konstrukce, na účel využití a dobu trvání. Dočasná stavba je stavba, u které stavební úřad předem omezí dobu jejího trvání. Za stavbu se považuje také výrobek plnící funkci stavby. Stavba, která slouží reklamním účelům, je stavba pro reklamu.*“ (§2 odst. 3 zákona č. 183/2006 Sb.)

Stavebný denník je základný dokument o tom, čo sa na stavbe deje v jej priebehu. Zapisujú sa do neho prehliadky, kontroly alebo aj zmeny oproti projektovej dokumentácii. Je dôležitý ako pre stavebný úrad, tak aj v prípade súdneho sporu. [8]

Nie je definovaný v zákone ako pojem. Miesto toho je v ňom opísané čo musí obsahovať, kto s ním môže narábať a pravidlá k vedeniu tohto denníku. Informácie o vedení deníku sú obsiahnuté zákonom č. 183/2006 Sb. a vyhláškou č. 499/2006 Sb.

> „*Při provádění stavby vyžadující stavební povolení nebo ohlášení stavebnímu úřadu musí být veden stavební deník, do něhož se pravidelně zaznamenávají údaje týkající se provádění stavby; u ohlašovaných staveb uvedených v §104 odst. 1 písm. e) až k) postačí jednoduchý záznam o stavbě.*“ (§157 odst. 1 zákona č. 183/2006 Sb.)

4.3 Existujúce riešenia

K problému, ktorým sa táto práca zaoberá existuje v súčasnosti niekoľko riešení. Dlhodobo využívaným riešením bolo písať stavebný denník papierovou formou. Tento prístup má ale veľa nedostatkov, čo spôsobilo dopyt po digitalizácii v tomto priestore. V tejto sekcii sú opísané viaceré druhy riešení. Ku každej je uvedený opis ako aj zhodnotenie výhod a nedostatkov. V podsekcii 4.3.2 je uvedené súčasné riešenie využívané Ředitelstvím silnic a dálnic ČR v podobe webovej aplikácie. V ďalšej podsekcii 4.3.3 je uvedený príklad komerčne dostupnej mobilnej aplikácie, a zároveň prečo je nedostačujúca.

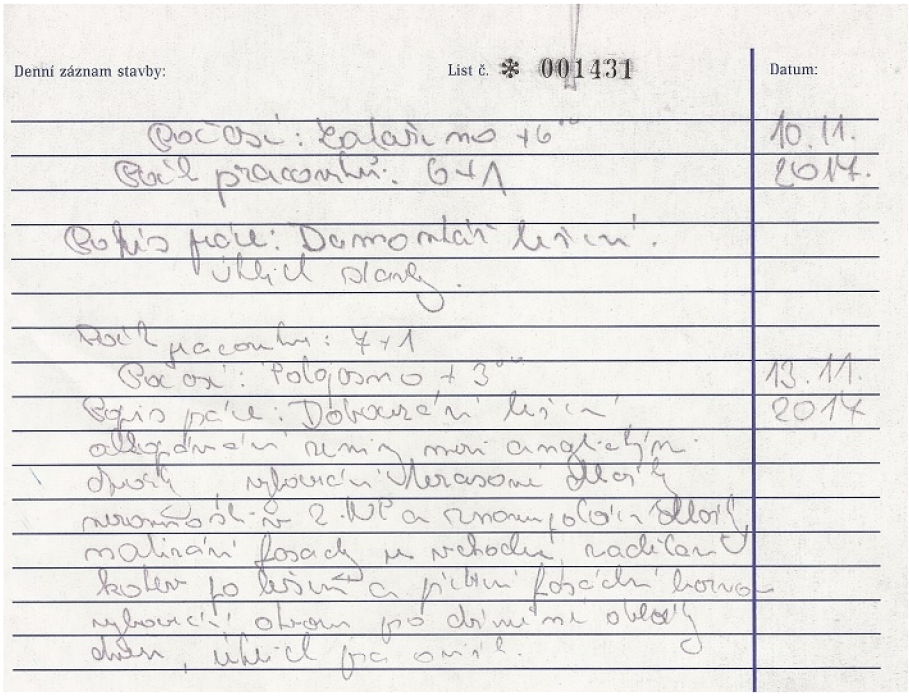
4.3.1 Papierová forma

Táto forma je základnou formou ako sa dá stavebný denník viesť. Sú jednoduché, hmatateľné a ľahko sa rozdeľujú medzi pracovníkov na pracovisku. Papierové formuláre majú však niekoľko nedostatkov, ktoré obmedzujú ich efektivitu v dnešnom stavebnom prostredí.

Po prvé, papierové formuláre sa môžu ľahko stratiť alebo poškodiť, čo vedie k neúplným alebo nepresným záznamom. Papierové formuláre navyše vyžadujú manuálne zadávanie údajov, čo je časovo náročné, náchylné na chyby a často vyžaduje prepis z jedného formátu do druhého. To predstavuje riziko nezrovnalostí v údajoch, ktoré môže byť ťažké identifikovať a opraviť. Taktiež po vytlačení a distribúcii formulára ho nemožno jednoducho upravovať alebo aktualizovať. To môže spôsobiť problémy, keď je potrebné formulár revidovať alebo keď je potrebné pridať nové dátové polia. V takýchto prípadoch môžu pracovníci potrebovať použiť nový formulár, čo vedie k zmätku a možným chybám.

Jednou z ďalších nevýhod je napríklad dodržiavanie zákona v niektorých bodoch. Napríklad „*Záznamy o postupu prací a jejich souvislostech se zapisují tentýž den, nejpozději následující den, ve kterém se na stavbě pracuje. U technicky jednoduchých staveb se mohou záznamy se souhlasem stavebního úřadu provádět nejdéle za období jednoho pracovního týdne.*“ (příloha č. 16 písm. C odst. 3 vyhlášky 466/2006 Sb.) je všeobecne v písomnej forme ťažké odsledovať.

Z týchto nedostatkov je jasne viditeľné, že je potreba nájsť modernejšie riešenia s flexibilnejším prístupom. Tu nastupujú elektronické riešenia ako webové a mobilné aplikácie.



Obr. 4.1: Príklad časti papierového zápisu do stavebného denníku z praxe (prevzaté z [8])

Na obrázku 4.1 je možné vidieť, že v papierovej forme sa dá stavebný denník viesť jednoduchou fomou zošitu aj keď existujú predformátované verzie [8]. Je vhodné podotknúť komentár autora článku [8] k obrázku „*I přes nedostatky, je to dobrý příklad. Málokdo v praxi vyplňuje deník takto poctivě.*“ (Radek Motzke), čo len svedčí o nedostatkoch papierovej formy.

Elektronická forma formuláru

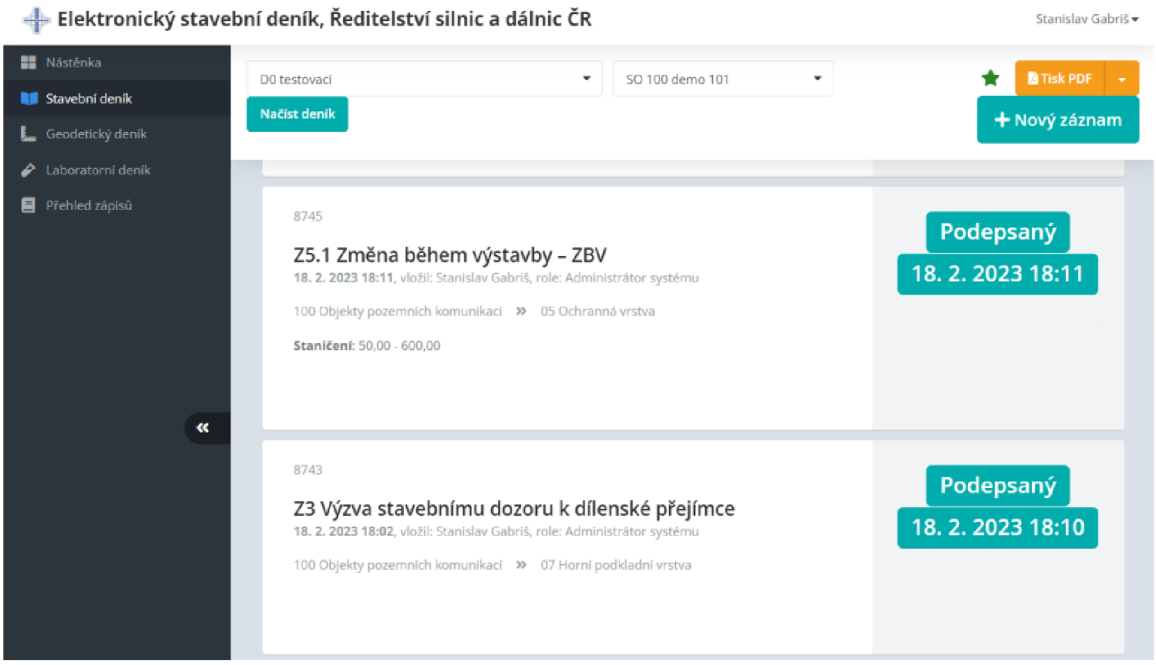
Ďalším logickým krokom po papierovej forme, je digitalizácia stavebného deníka, a teda elektronický stavebný deník. Presunutie do elektronickej formy už samo o sebe rieši niektoré problémy. Eliminuje sa potreba manuálneho zadávania niektorých údajov ako aj ručné formátovanie, redukujú sa chyby a zjednodušuje sa úprava, ukladanie a vyhľadávanie záznamov. Elektronické verzie formulárov sú však často obmedzené na stolné počítače, čo môže obmedziť ich dostupnosť pre stavebných robotníkov na stavenisku.

4.3.2 Webová aplikácia Elektronický stavební deník ŘSD

Webové aplikácie sú už v súčasnosti bežné a populárne riešenie digitalizácie v mnohých sektroch. Oproti predtým spomenutým riešeniam ponúkajú niekoľko výhod, vrátane výhod elektronického formulára, ako je jednoduchý prístup, spolupráca, synchronizácia údajov v reálnom čase a je v nich jednoduchšie vynútenie dodržiavania zákona. Takáto aplikácia poskytuje centralizovanú platformu na správu a zdieľanie dokumentov medzi zainteresovanými stranami, či už sú to stavby vedúci, zhotovitel alebo iný.

Súčasné prevedenie tejto formy stavebného denníka v Ředitelství silnic a dálnic ČR má podobu aplikácie *Elektronický stavební deník - ESD* (ukážka obrazovky na obrázku 4.2) určenej hlavne pre stolné počítače alebo tablety.

Aplikácia používateľom umožňuje pracovať so stavebným denníkom podobne, ako pri práci s papierovou formou. Najdôležitejšiou časťou je na obrázku 4.2 v bočnom panely zvýraznená položka Stavební deník. Na tejto obrazovke si vedia používatelia zvoliť stavbu, na ktorej pracujú alebo na ktorej by si chceli prezrieť dostupné záznamy. Záznamy sa zobrazujú ako prehľad pod sebou formou položiek a obsahujú informácie z formulára dáného záznamu. Keďže je aplikácia určená primárne pre zariadenia s väčšou obrazovkou, do tohto prehľadu sa zväčša zmestia všetky informácie o konkrétnom zápise. Používateľ, podľa svojich práv, môže pridávať a upravovať jednotlivé zápisy vyplnením detailného formuláru. Zápis musí byť ďalej podpísaný do dvoch dní od založenia. Všetky zápisy na danej stavbe sa dajú tlačidlom `Tisk PDF` preformátovať do dokumentu PDF. Hviezdičkou pri tomto tlačidle sa dá daná stavba uložiť do „obľúbených“ pre jednoduchšiu navigáciu v budúcnosti. Časť Nástenka obsahuje prehľad obľúbených stavieb a výzvy pre daného uživateľa. Výzva je zápis na ktorý sa dá odpovedať.

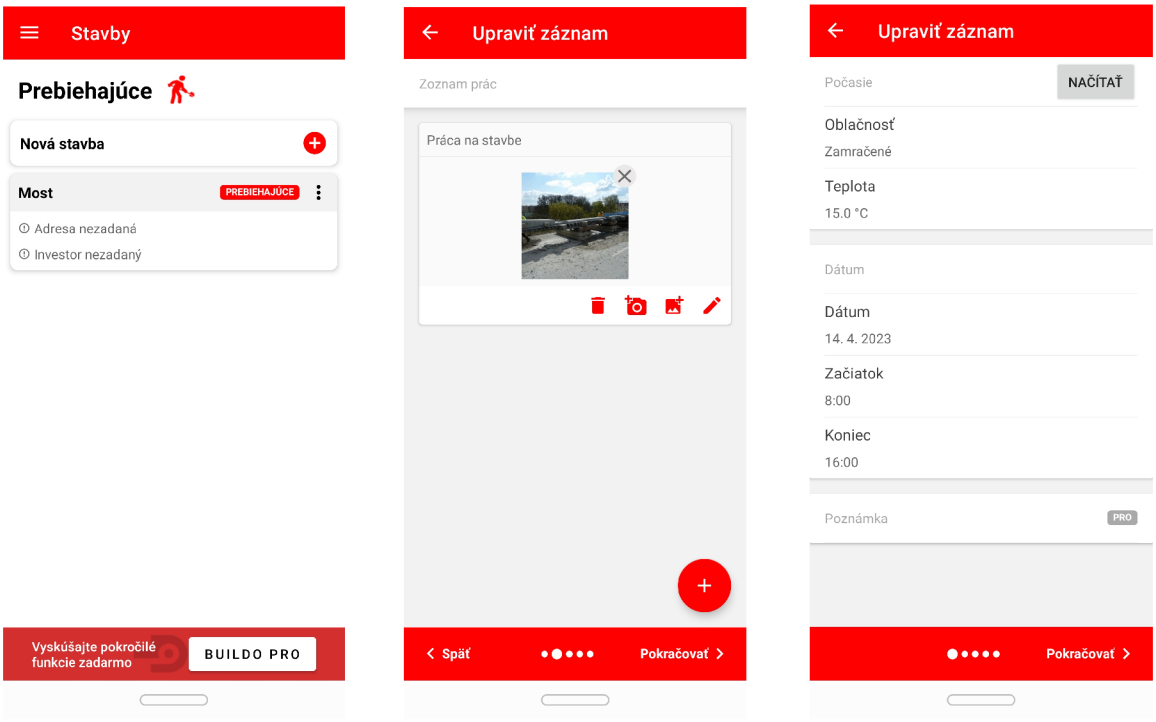


Obr. 4.2: Výpis zápisov vo webovej aplikácii ESD

Forma webovej aplikácie má v porovnaní s mobilnou verziou niekoľko výhod. Je to najmä možnosť prístupu k tejto aplikácii zo všetkých zariadení s webovým prehliadačom. Tento fakt má za následok aj to, že aplikáciu netreba sťahovať a následne aktualizovať pri každej novšej verzii. Medzi nevýhody patrí menšia využiteľnosť hardvérových a softvérových funckií konkrétnej platformy a hlavne nutnosť pripojenia k internetovej sieti.

4.3.3 Komerčná aplikácia Buildo

Na českom a slovenskom trhu sa nachádza hneď niekoľko mobilných aplikácií plniacich funkciu stavebného denníka, medzi nimi je aj aplikácia *Buildo*[2], ktorá je dostupná v limitovanej verzii zdarma z obchodu Google Play. Táto aplikácia je určená skôr pre menšie stavby.



(a) Výpis stavieb (b) Výpis záznamov (c) Pridanie nového záznamu

Obr. 4.3: Snímky obrazovky z aplikácie Buildo

V aplikácii sa dá najskôr vyplniť údaje o firme. Po ich vyplnení je možné na hlavnej obrazovke, nazvanej Stavby (obrázok 4.3a) pridávať stavby, ktoré sa dajú nazvať, pridať k nim lokáciu a investora. Po vytvorení budú vo výpise taktiež na tejto stránke, rovnako ako ich status, či bola stavba dokončená alebo prebieha. Ku stavbe sa dajú nasledovne pridávať denné záznamy (obrázok 4.3c) do stavebného denníka. Tento záznam má formu formuláru s políčkami na vyplnenie. K záznamu sa taktiež dajú pridávať alebo priamo vyfotiť fotky ako je znázornené na obrázku 4.3b. Je možné ku záznamu pridať aj pracovníkov a čo každý z nich robil, podobne sa dajú pridať aj prítomne stroje, poprípade materiál. Zaujímavou funkciou je aj možnosť po pridaní záznamu odoslať tento záznam vo formáte PDF priamo na e-mailové adresy investorov. Aplikácia podporuje priamo aj exportovanie stavebného denníku, t.j. všetky denné záznamy za určité obdobie, vo formáte PDF do mobilu.

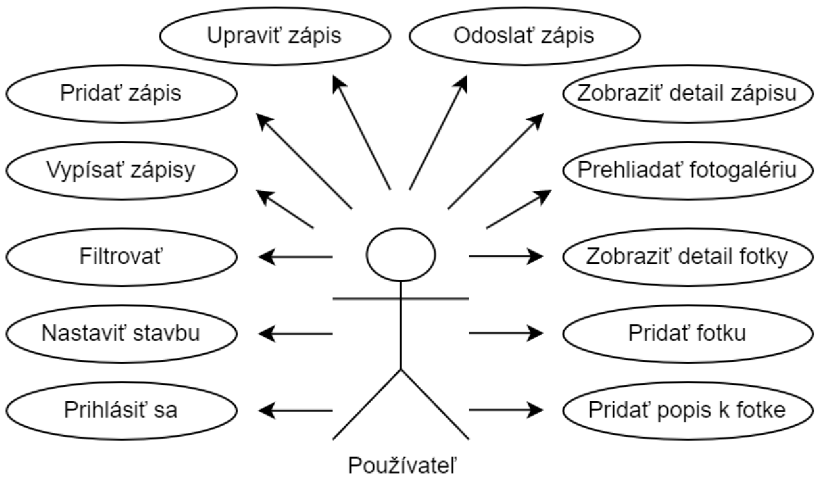
[2]Aplikácia Buildo: https://buildoapp.com/CZ/

Výhodou tejto mobilnej aplikácie je možnosť začať s ňou pracovať hneď po stiahnutí bez toho aby bolo vyžadované registrovať sa. Aplikácia taktiež spĺňa kritérium funkčnosti aj bez internetového pripojenia. Keďže do aplikácie sa nie je nutné registrovať, všetky údaje sú len lokálne na užívateľovom zariadení, čo môže byť brané ako nevýhoda. Pre väčšie stavby, s ktorými Ředitelství silnic a dálnic pracuje, je táto aplikácia taktiež nedostačujúca. V neposlednom rade aplikácia nenapĺňa kritéria stanovené Ředitelstvím silnic a dálnic ČR.

Kapitola 5

Návrh riešenia

Pred samostatným implementovaním riešenia je nutné si najskôr ujasniť, čo má vlastne aplikácia ponúkať. V tejto kapitole je opísané, aké boli na začiatku ciele. V prvom rade je zhrnuté akú funkcionalitu by mala aplikácia spĺnať, uvedený je aj diagram prípadov použitia. Popísaná je aj zvolená architektúra celej implementácie. Ďalej je rozobraný prvotný návrh vo forme prototypu a uvedené jeho nedostatky.



Obr. 5.1: Diagram prípadov použitia

Cieľom tejto bakalárskej práce je vytvoriť mobilnú aplikáciu pre zariadenia Android, s možnosťou rozšírenia o platfomu iOS v budúcnosti. Aplikácia by mala zastávať funkciu elektronického stavebného denníka, ktorá poskytuje časť funkcionality webovej aplikácie popísanej v sekcii 4.3.2 a vyhovuje požiadavkám Ředitelstva silnic a dálnic ČR. Aplikácia by mala umožnovať funkcionalitu a interakcie znázornené diagramom 5.1. Táto funkcionalita teda zahŕňa najmä prácu so zápismi. Aplikácia musí byť schopná funkcionality aj v prípade, že na mieste nie je možnosť pripojiť sa do internetovej siete a aj v prípade slabého signálu. Zároveň, aplikácia musí vedieť komunikovať s rovnakou databázou, využívanou pre webovú aplikáciu. Aplikácia bude komunikovať s touto databázou využitím REST API (opísané sekciou 3.4.1) špecifikovanou štandardom OpenAPI (opísané sekciou 3.4.2). Užívateľ bude môcť svoje lokálne zmeny, ktoré pridal počas toho ako nemal prístup na internet, manuálne odoslať cez REST API a rovnako si bude môcť synchronizovať dáta s webovou aplikáciou.

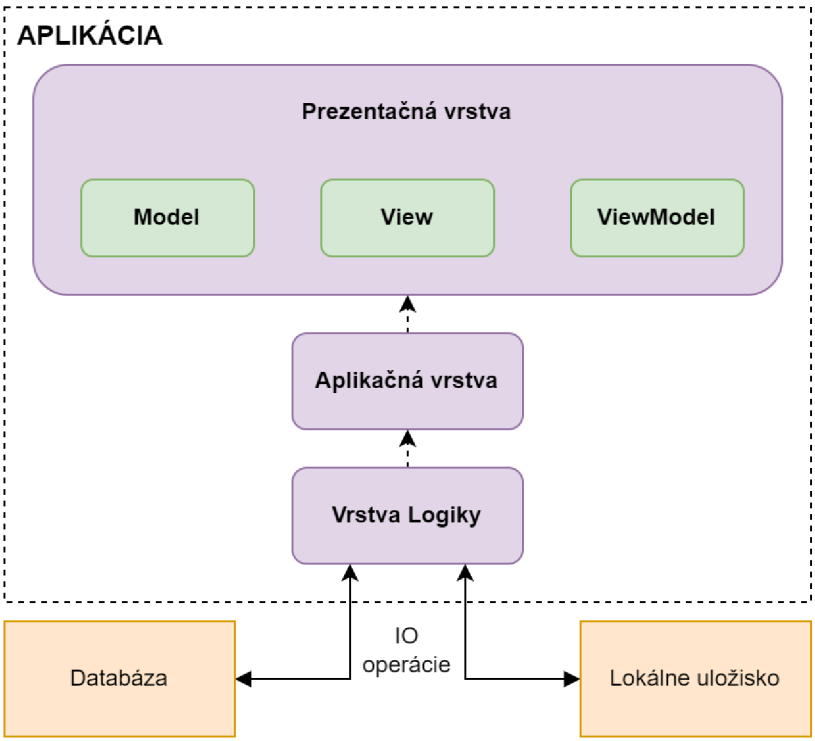
Aplikácia automaticky znemožní používateľovi upravovať a odosielať zápis v prípade, že ubehly od začiatku dňa jej založenia 2 dni. Do aplikácie sa bude dať prihlásiť rovnakými prihlasovacími údajmi, využívanými vo webovej aplikácii a aplikácia bude mať prívetivé používateľské rozhranie. Na implementáciu bude využité prostredie Visual Studio a aplikačný rámec .NET MAUI. Detailnejší popis diagramu 5.1 je v bodoch vypísaný nižšie.

- **Prihlásiť sa**, pre prihlásenie budú používatelia využívať rovnaké údaje ako pri prihlasovaní do webovej aplikácie. V prípade nesprávnych prihlasovacích údajov sa nebude možné do aplikácie dostať.
- **Nastaviť stavbu** sa skladá z dvoch možností, a to vybranie stavby a vybranie konkrétnej časti stavby. Táto voľba bude slúžiť ako celoaplikačný filter a bude ďalej ovplyvňovať celú aplikáciu. Po vybraní stavby sa budú uživateľovi zobrazovať zápisy práve na vybranej stavbe a bude môcť pridávať a upravovať zápisy na tejto stavbe. Vybraná stavba by mala byť vždy vidieť na relevantných obrazovkách (napríklad vo výpise zápisov).
- **Filtrovať**, aplikácia bude podporovať filtrovanie zápisov aj podľa dátumu.
- **Vypísať zápisy**, v aplikácii sa budú dať stiahnuť zápisy ku konkrétnemu nastavenému filtru a zobrazia sa ako výpis položiek. Tieto zápisy by mali byť po stiahnutí stále prítomné na zariadení a uživateľ s nimi bude môcť pracovať.
- **Pridať zápis**, pri zvolenej stavbe bude možné, aj bez prístupu k internetovej sieti, pridávať nové zápisy, pričom zápisy budú mať viacero druhov a formulárovú formu. To aké druhy zápisov bude uživateľ môcť pridávať bude závisieť na jeho právach, ovplyvnené jeho rolou.
- **Upraviť zápis** bude, podobne ako pridať zápis možné aj bez pripojenia na internet, aplikácia bude rozlišovať zápisy, ktoré sa dajú a nedajú upravovať. Zápisy, ktoré sú upravovateľné no staršie dvoch dní budú zneprístupnené.
- **Odoslať zápis** znamená, že keď uzná používateľ za vhodné, môže svoje lokálne zmeny na zápise zo zariadenia odoslať pomocou REST API do databázy. Uživateľ bude o úspešnosti tejto akcie notifikovaný.
- **Zobraziť detail zápisu** je funkcia, ktorá užívateľovi poskytne prehľad všetkých infomácií o jednom konkrétnom zápise, ako autor zápisu, dátum vytvorenia alebo podpisu, popis, poprípade špeciálne informácie pri špeciálnych zápisoch.
- **Prehliadať fotogalériu**, zápisy budú mať okrem formuláru aj galériu, v ktorej budú zobrazené fotky tohto zápisu.
- **Zobraziť detail fotky**, v galérii sa bude dať kliknúť na konkrétnu fotku, čo presunie používateľa na detailnejší pohľad.
- **Pridať fotku**, do zápisu, ktorý je upravovateľný, sa v galérii bude dať pridať fotka. Fotka sa bude dať pridať buď funkciu fotenia priamo z aplikácie alebo vybraním fotky z gálerie mobilného zariadenia. Podporované budú obe možnosti.
- **Pridať popis k fotke** bude možné z detailu fotky, pri ktorej bude možnosť vyplniť políčko s popisom.

- Špeciálny typ zápisu – Dení zápis – bude zložený z ďalších zápisov. Tým pádom bude sám fungovať ako prehľad zápisov a budú sa k nemu taktiež dať pridávať zápisy.
- Okrem popísaných funkcií musí mať mobilná aplikácia obrazovku *Nástěnka*. Na tejto obrazovke sa budú zobrazovať špeciálne zápisy, ku ktorým sa dá pridať odpoveď (fungujú na báze výzva – odpoveď)
- Užívateľ bude môcť pri niektorých akciách, vyžadujúcich komunikáciu po sieti, prerušiť po určitom čase komunikáciu (napríklad ak bude operácia trvať príliš dlho pri slabom internete). Pri implementácii sa k tomu využije tzv. *CancellationToken*.

Architektúra

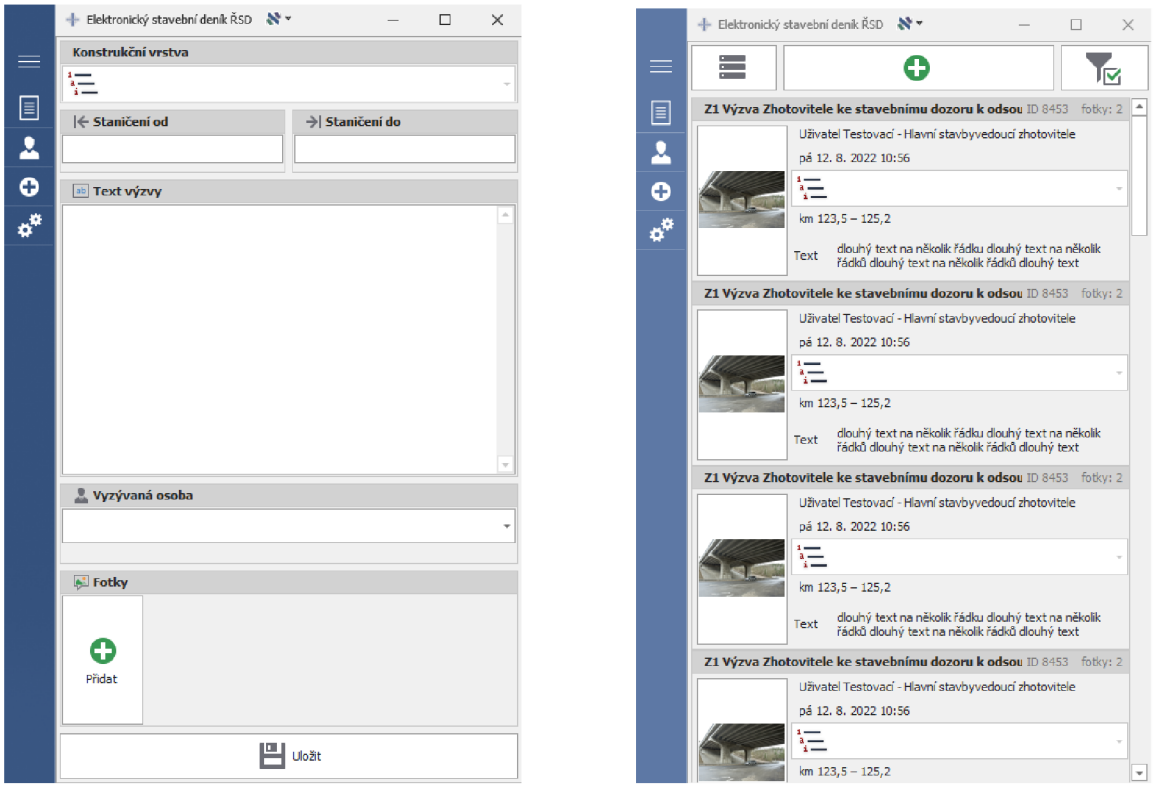
Čo sa architektúry využitej pri implementácii týka, bude aplikácia rozdelená podľa viacvrstvovej architektúry (popísanej v sekcii 3.2). Aplikácia bude mať tri hlavné vrstvy. Prvou vrstvou bude *prezentačná vrstva*, v ktorej bude zahrnutá logika zobrazovania dát a ich naviazanie, ako aj samotné grafické rozhranie. Táto vrstva bude ďalej využívať model MVVM (popis v sekcii 3.3) pre oddelenie samotnej implementácie grafického rozhrania od zobrazovaných dát. Ďalšou vrstvou bude *aplikačná vrstva*, slúžiaca ako rozhranie, zodpovedná za sprostredkovanie funkcionality logickej vrstvy. Ako posledná bude figurovať *vrstva logiky*, ktorá bude spracovávať dáta získané z prezentačnej vrstvy, databázy alebo lokálneho uložiska. Táto architektúra je znázornená na obrázku 5.2.



Obr. 5.2: Návrh architektúry aplikácie

Prototyp

Pred samotnou implementáciou bol vypracovaný návrh funkcionality, popísaný v predchodzích častiach, a prototyp aplikácie. Tento prototyp vyobrazený na obrázku 5.3 bol vypracovaný nástrojom *Windows Presentation Foundation*[1], ktorý sa dá využiť aj pre vývoj plnohodnotných aplikácií na platformu Windows, v ktorom sa dá relatívne jednoducho navrhnúť grafické rozhranie aj bez funkcionality. Od tohto návrhu sa odvíjal ďalší vývoj aplikácie.



(a) Formulár zápisu

(b) Výpis zápisov

Obr. 5.3: Prototyp aplikácie

Aj napriek tomu, že bol tento prototyp vytvorený hneď na začiatku vývoja, v období kedy ešte všetky špecifikácie aplikácie neboli definované, ukázal sa ako celkom presný a poslúžil na ujasnenie niektorých nejasností. Prototyp sa riadil prvotnou špecifikáciou Ředitelstva silnic a dálnic ČR a slúžil ako inšpirácia pri implementácii. Čo sa ukázalo ako nedostatočné bolo, že z prototypu nebolo jasné, ktoré zápisy je možné upravovať a ktoré nie. Rovnako nebolo hneď jasné, na ktorej stavbe sa aktuálne užívateľ nachádza. Prototyp nerozlišoval medzi uložením a odoslaním zápisu. Ukázalo sa, že prototyp tiež prehodnotil veľkosť a viditelnosť niektorých prvkov. Čo sa naopak osvedčilo a bolo prevzaté v implementácii bol formát *Menu* popísaný v sekcií 6.5, formulár alebo obsah dát vo výpise k jednotlivému zápisu.

[1] https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-7.0

Kapitola 6

Implementácia

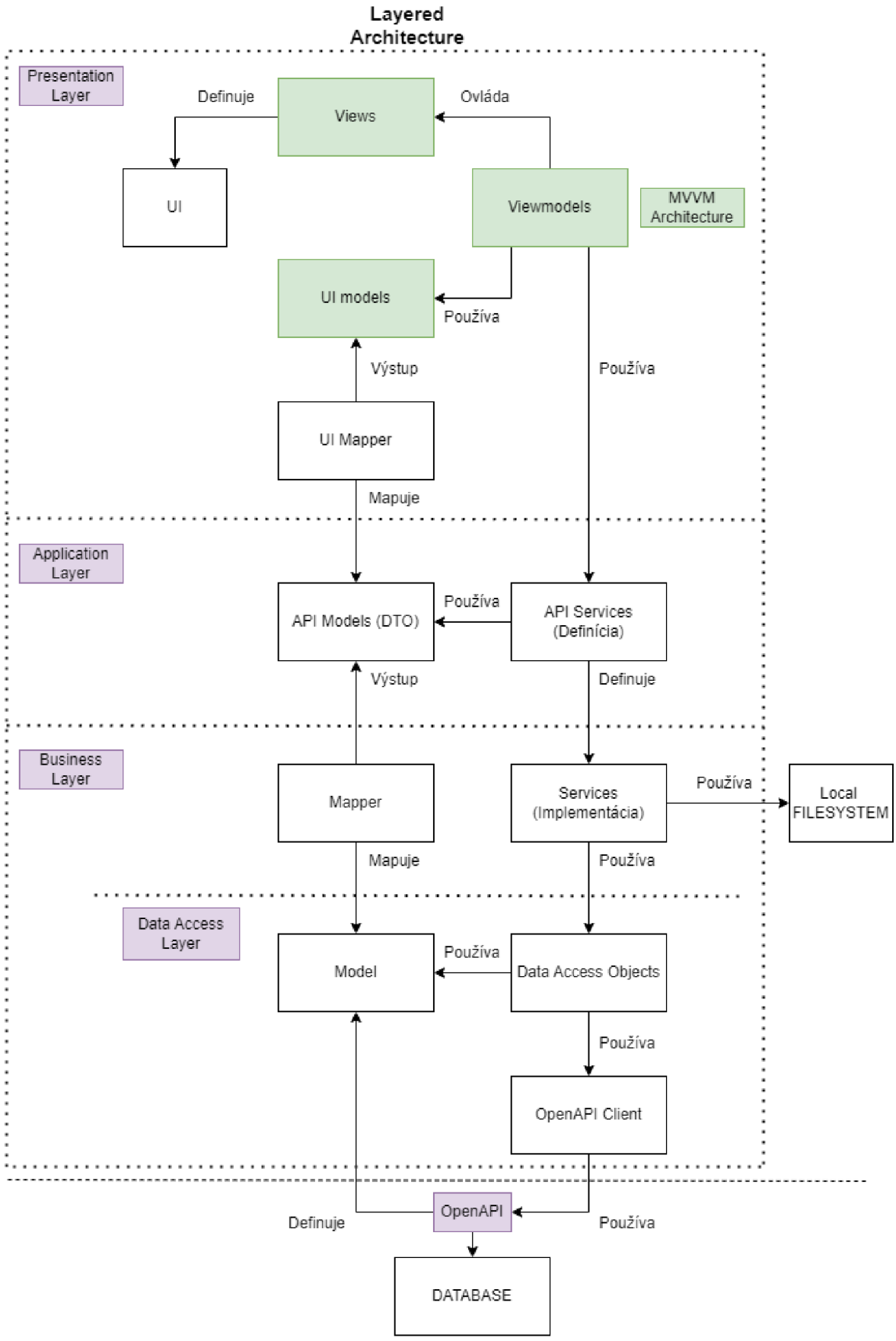
Implementácia, ako proces premeny teoretického konceptu na praktickú realitu, je v kontexte tejto práce o preklade návrhu riešenia (kapitola 5) s využitím popísaných technológií a konceptov v predchádzajúcich kapitolách na funkčnú a použiteľnú realizáciu mobilnej aplikácie. Implementácia zahŕňa skutočné kódovanie a nasadenie aplikácie pre využitie používateľmi.

Mobilná aplikácia nesie rovnaký názov ako webová aplikácia (sekcia 4.3.2) a to *Elektronický stavební deník* (skratka *ESD*). V tejto kapitole je v prvom rade popis využitých nástrojov a technológií, a proces implementácie. Štruktúra projektu je popísaná architektúrou na obrázku 6.1, ktorá sa riadi viacvrstvovou architektúrou popísanou v sekcii 3.2. Táto štruktúra je využitá pre rozdelenie do sekcií, opisujúcich jednotlivé časti implementácie. Popis začína od spodnej časti obrázku a postupne opisuje vrstvy smerom hore. V sekcii 6.5 je popísané finálne grafické rozhranie aplikácie a nakoniec je uvedená krátka pasáž o vydávaní aplikácie pre zariadenia Android.

6.1 Použité technológie

Prvá fáza aplikácie, t.j. fáza opísaná touto prácou, je určená pre zariadenia so systémom Android (platforma Android je opísaná v sekcii 2.1.1). V budúcnosti je možné a požadované aplikáciu rozšíriť aj pre zariadenia iOS (platforma iOS je opísaná v sekcii 2.1.2). Z tohto dôvodu bol zvolený multiplatformný vývoj v .NET MAUI (opísaný sekciou 2.2.1), umožňujúci využívať natívne prvky platforiem, ako napríklad senzory (napríklad kamera) alebo grafické elementy, ich abstrakciou. Implementácia prebiehala v grafickom prostredí *Visual Studio*[1], ktoré sa špecializuje na projekty v .NET. Čo sa týka konkrétneho programovacieho jazyka, tak projekty .NET MAUI sú písané v C# (popis taktiež v sekcii 2.2.1) s tým, že definícia grafického rozrania využíva značkovací jazyk *XAML* založený na jazyku *XML* a vyvinutý spoločnosťou Microsoft. Pri implementácii bol v menšej miere využitý aj platený softvér *DevExpress*[2], ktorý obsahuje nápomocné prvky grafického rozhrania pre .NET MAUI. Využitý vo väčšej miere bol aj *.NET MAUI Community Toolkit*, ktorý rozširuje možnosti pri práci s .NET MAUI, či už pridaním funkcionality alebo prvkov grafického rozhrania. Implementácia sledovala koncepty opísané v kapitole 3, komunikácia využíva princípi REST v špecifikácii OpenAPI a samotná architektúra aplikácie je stavaná na viacvrstvovom vzore, s prezentačnou vrstvou využívajúcou model MVVM.

[1] https://visualstudio.microsoft.com/
[2] https://www.devexpress.com/



Obr. 6.1: Architektúra projektu

Pred začatím

Ešte pred začatím implementácie bolo nutné stiahnuť Visual Studio a uistiť sa, že sú k nemu stiahnuté aj všetky potrebné časti, aby bolo možné pracovať s .NET MAUI. Po otvorení editoru sa dá zvoliť z niekoľko typov projektov, medzi nimi nový .NET MAUI projekt. Visual Studio odlišuje pojmy *projekt* a riešenie (z anglického *solution* – tento výraz bude používaný ďalej), kde solution obsahuje všetky projekty, ktoré spolu súvisia a ďalšie informácie. Projekt sa na druhú stranú v solution nachádza a obsahuje súbory, z ktorých je pri prekladaní vyrobený spustitelný súbor. Po zvolení typu je pripravený solution so základnou štruktúrou a jedným projektom .NET MAUI pripravený na preloženie.

6.2 Úroveň logiky

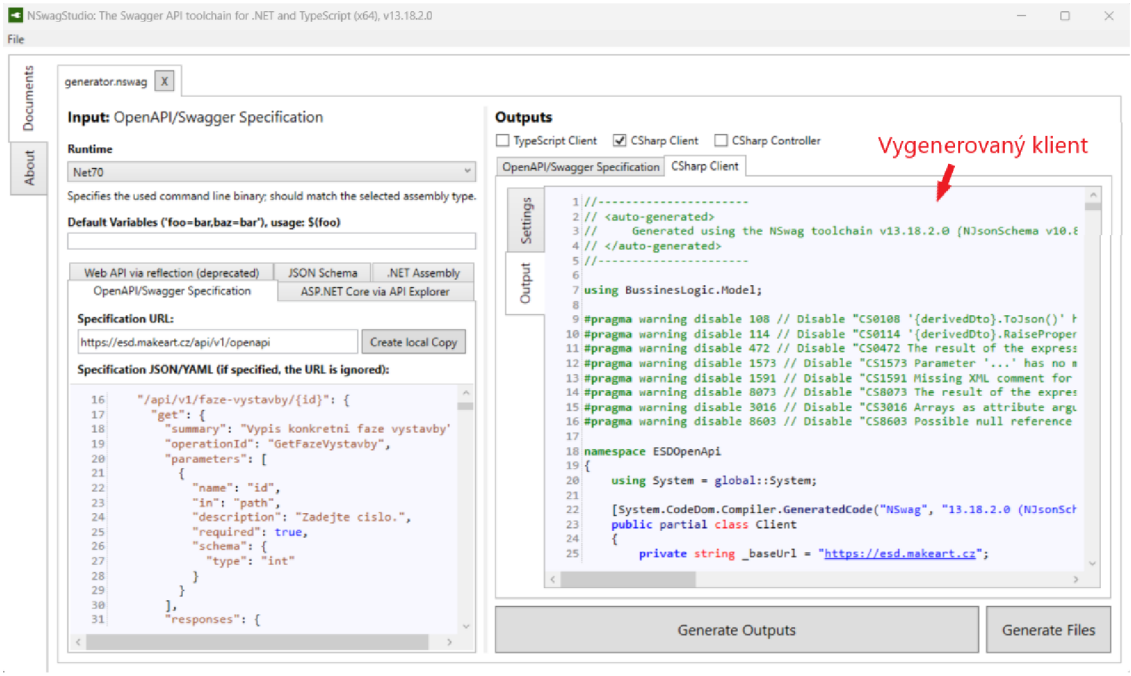
Ako prvá časť implementácie je opísaná na obrázku 6.1 časť *Business Layer*. Predstavuje časť viacvrstvovej architektúry, ktorá má za úlohu postarať sa o logiku aplikácie. Je to teda výpočetne najnáročnejšia časť, ktorá má prístup k dátam. Popísané tu sú aj časti v obrázku zakreslené mimo túto vrstvu, ako *Local Filesystem* a komunikácia s databázou, pretože s ňou úzko súvisia. V solution sa nachádza pod súborom `BusinessLogic`.

Data Access Layer je vrstva, ktorá nie je priamo definovaná v štruktúre celého solution, ale pre lepšie pochopenie a rozdelenie bola v obrázku vykreslená ako časť Business Layer. Ako už názov naznačuje, jedná sa o vrstvu, ktorá slúži ako vstupná brána aplikácie pre dáta. V prípade tohto projektu sa jedná o dáta z vonkajšej databázy. Je využitá rovnaká databáza akú používa webová aplikácia (sekcia 4.3.2). Komunikácia je zabezpečená rozhraním definovaním pomocou špecifikácie OpenAPI. Spoločne s autorom databázy bola táto špecifikácia navrhnutá. Pre návrh a testovanie bolo využité Visual Studio, kde sa za pomoci anotácií dájú relatívne jednoducho definovať schémy, ktoré predstavujú jednotlivé triedy s anotovanými vlastnosťami (anglicky properties – tento výraz bude používaný ďalej), cesty, predstavujúce adresu zdrojov a všetky podobné položky definované v štandarde OpenAPI. Nakoniec je vygenerovaný súbor s touto špecifikáciou vo formáte *JSON*. Špecifikácia bola počas vývoja postupne upravovaná a doplňaná podľa potreby.

Z pripravenej špecifikácie bol pomocou nástroja *NSwag*[3] v grafickom prevedení *NSwagStudio* vygenerovaný klient v jazyku C# pre konzumáciu rozhrania. Obrázok 6.2 ukazuje jeho využitie. Na ľavú stranu je skopírovaná JSON špecifikácia a na pravej strane je vygenerovaný klient k tejto špecifikácii, ktorý vyhovuje rôznym upraviteľným nastaveniam.

Nakoniec je teda tento vygenerovaný klient v obrázku 6.1 položka `OpenAPI Client`, zatiaľ čo položka `Model` predstavuje schémy dát, ktoré aplikácia prijíma cez protokol HTTP (sekcia 3.4), definované v špecifikácii OpenAPI. Poslednou časťou tejto „podvrstvy“ sú v obrázku znázornené `Data Access Objects` (skratka DAO), slúžiace ako sprostredkovateľ metód v OpenAPI klientovi. V solution sa nachádzajú pod priečinkom `DAOs` a sú to jednoduché triedy s metódami predstavujúcimi *CRUD* – z anglického Create, Read, Update a Delete – operácie nad dátami.

[3] https://github.com/RicoSuter/NSwag



Obr. 6.2: Náhľad na nástroj NSwagStudio pre prácu so špecifikáciou OpenAPI

Zvyšok logickej vrstvy tvoria na orázku 6.1 položky `Mapper` a `Services`.

`Mapper` predstavuje časť logickej vrstvy, ktorá slúži na transformáciu modelu (trieda obsahujúca dáta), ktorý aplikácia prijíma z REST API, na iný model, a to taký, ktorý je skutočne využívaný aplikáciou (Označený na obrázku ako `API Models` popísaný o sekciu nižšie 6.3).

`Services` predstavuje implementáciu metód obsahujúcich drvivú väčšinu logiky aplikácie. Je to niekoľko tried spracovávajúcich dáta, či už z aplikácie, lokálneho úložiska alebo databázy. Tieto triedy sú ďalej rozdelené do priečinkov `Cache` a `Offline` podľa toho, či si dáta, ktoré spracovávajú udržiavajú vo vyrovnávacej pamäti. `Services` využívajú `Data Access Objects` pre prístup k údajom z databázy a `Mapper` na mapovanie týchto údajov do použiteľnej formy, a taktiež využívajú aj samé seba. Servisy, ktoré sú pod priečinkom `Offline` majú na starosti prácu s lokálnym uložiskom (na obrázku `Local FILESYSTEM`). Servisy s vyrovnávacou pamäťou majú dáta nachystané v prípade potreby. Najzaújímavejšou z týchto tried je trieda `CacheZapisyService`, v ktorej je obsiahnutá logika manipulácie so zápismi. Keďže existuje viacero druhov zápisov, každý rozdielny typ vyžaduje vlastný model, no práca s týmito modelmi je v prípade servisu takmer vždy rovnaká. Jedná sa hlavne o uloženie dát, odoslanie dát, vyčítanie dát alebo úpravu dát zápisu. Pre to aby sa kód nemusel zbytočne opakovať bol využitý, v jazyku C# dostupný, koncept *polymorfizmu* a *dedičnosti*. Tieto koncepty dovoľujú vytvoriť generickejší kód. Všetky druhy zápisu dedia od určitej „základnej“ triedy zápisu. Takto napríklad nie je potrebné pre ukladanie zápisu definovať n metód pre n typov zápisov, miesto toho je implementovaná jedna metóda `Save`, ktorá očakáva ako vstup určitý zápis, no v „základnej“ forme, zatiaľ čo na základe iného vstupného parametru dokáže určiť o aký typ sa jedná a ďalej s týmto „základným“ typom pracuje ako s konkrétnym. Tieto koncepty sú v aplikácii do veľkej miery využité pri predávaní informácií medzi vrstvami.

Ukladanie dát do lokálneho úložiska

Do lokálneho úložiska sú ukladané všetky informácie z aplikácie, ktoré je vhodné a potrebné uchovávať, pre možnosť funkcionality aj bez internetového pripojenia. Na platforme Android je možné využiť tri spôsoby ukladania. Ako prvé je možné využiť tzv. *Preferences*, určené pre ukladanie jednoduchých dvojíc typu kľúč – hodnota. Tento spôsob je využítý pre ukladanie e-mailu a hesla užívateľa v prípade, že si zvolí zapamätanie prihlasovacích údajov. Keďže tieto údaje predstavujú citlivé informácie, je využité rozhranie `SecureStorage`[4], ktoré pre platformu Android zašifruje dáta. Ďalší spôsob je využitie *systému súborov*, a teda ukladanie informácií do jednotlivých súborov. Týmto prístupom sú ukladané zvyšné dáta v aplikácii, ako zápisy, fotky alebo iné dáta, ktoré sú prístupné len pre aplikáciu. Dáta sú ukladané vo formáte JSON. Posledným a nevyužitým spôsobom akým by sa dali údaje ukladať je *lokálna databáza SQLite*.

6.3 Úroveň aplikačného rozhrania

Ďalšou časťou na obrázku 6.1 je časť *Application Layer*, ktorá predstavuje rozhranie medzi vrstvou logiky a prezentačnou vrstvou (popísanou v sekcii 6.4).

V tejto vrstve sa nachádza definícia servisov (časť `API Services`) vrstvy logiky a slúži ako jej abstrakcia. To znamená, že sa tu nenachádza logika. Táto definícia je sprístupnená prezentačnej vrstve, ktorá ju môže využiť bez toho aby „vedela“ o vrstve logiky. Týmto spôsobom je možné kedykoľvek zmeniť skutočnú implementáciu servisov bez toho aby boli nutné zmeny v prezentačnej vrstve.

Taktiež sa v tejto vrstve nachádza časť `API Models`, ktorá predstavuje model dát s ktorými servisy pracujú. Skratka *DTO*[5] (z anglického Data transfer object) len znázorňuje, že pre prenos dát medzi vrstvami je využitý iný než pôvodný model, čo znamená, že triedy v tomto modely presne nezodpovedajú modelu definovanému v OpenAPI.

6.4 Prezentačná vrstva

Poslednou časťou na obrázku 6.1 je *Presentation Layer*. Zahŕňa samotné grafické rozhranie a logiku zobrazovania. V tejto vrstve je ďalej využitý MVVM model (popísaný sekciou 3.3), ktorý oddeľuje grafické rozhranie od dát, ktoré sa zobrazujú.

Model

Časť `UI Models` predstavuje znova upravený model dát rozdielny od modelu API. Táto forma modelu obsahuje takú štruktúru, ktorá vyhovuje konkrétnym častiam grafického rozhrania. Pre prevod medzi modelmi API a modelmi UI je využitý komponent `UI Mapper`.

ViewModel

Každá obrazovka má priradený vlastný `ViewModel`. Táto časť využíva UI Models a určuje čo sa má na obrazovke zobraziť.

[4] https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/storage/secure-storage?view=net-maui-7.0&tabs=android

[5] https://stackoverflow.com/questions/1051182/what-is-a-data-transfer-object-dto

Keďže viewmodel nevie o view, tak len poskytuje pomocou `ObservableProperty` (jedna položka) a `ObservableCollection` (kolekcia položiek) anotácií možnosť sledovať zmeny premenných. Pri každej zmene položky, či už zo strany kódu alebo zo strany grafického rozhrania, je vyslaná notifikácia z tejto premennej, ktorú je možné odchytiť a spracovať (táto skutočnosť je využitá a opísaná v odseku o views). Triedy viewmodel rovnako poskytujú možnosť naviazania *udalostí* (z anglického event) na konkrétne metódy v týchto triedach za pomoci anótácie `RelayCommand`. Viewmodel je taktiež zodpovedný za využívanie vrstvy logiky, na čo využíva úroveň aplikačného rozhrania (6.3).

Viewmodel je rovnako zodpovedný za spracovanie výnimiek (anglicky *exceptions*). Jeden z najdôležitejších riešených prípadov je zlyhanie a výnimka z klienta pre rozhranie REST. V tomto prípade ide najmä o výnimky pri neočakávaných návratových kódoch vrátených serverom alebo očakávaných ale chybných kódoch. Táto výnimka je v kóde označená ako `ApiException`. Ďalšia dôležitá výnimka `WebException` môže nastať v prípade chyby spojenia so serverom, napríklad pri zariadení bez pripojenia na internet. Aplikácia sa taktiež snaží odchytiť aj výnimky iných druhov a to v prípadoch kde existuje riziko vzniku takejto výnimky. Pri narazení na nejakú výnimku aplikácia s touto chybou oboznámi uživateľa vyskakovacím oknom s relevantnou hláškou.

View

Nakoniec je samotné grafické rozhranie (`UI`) definované časťou `Views`. Každé view (slovensky zobrazenie) predstavuje určitú stránku. To znamená, že ak je používateľ napríklad na stránke s prihlásením jedná sa o loginview alebo loginpage. Ako už bolo spomenuté v použitých technológiách (6.1), tak jednotlivé view sú napísané v jazyku XAML. Ku každému view ale existuje aj tzv. „code-behind“ v jazyku C#, ktorý obsahuje minimum kódu a slúži len na inicializáciu zobrazenia a naviazanie s viewmodelom, poprípade s manipuláciou samotného rozhrania bez poznania dát (napríklad v prípade zobrazenia notifikácie alebo skrytia prvku v grafickom rozhraní). Po naviazaní s viewmodelom sa dajú využiť viewmodelom poskytnuté udalosti pri zmenách premenných.

Pre naviazanie s premennými je využitý koncept *data binding*. V princípe je funkčnosť taká, že v prípade napríklad textového poľa ako vizuálneho prvku, je k jeho vlastnosti `Text` naviazaná premenná z viewmodelu s anotáciou `ObservableProperty`. Takýmto spôsobom dokáže aj view a aj viewmodel reagovať na zmeny skutočných dát v tejto premennej alebo v grafickom rozhraní, ako dôsledku užívateľského vstupu.

Pri interakcii s grafickým rozhraním sú taktiež vysielané rôzne udalosti. Takáto udalosť môže predstavovať napríklad navigovanie na inú stránku alebo aj zmenu v grafickej komponente. Na tieto udalosti sa dá rovnako reagovať naviazaním metód z viewmodelu. Pre príklad pri navigácii zo stránky `Nastavení stavebních deníků` je vo viewmodely uložený stav zvolených nastavení do pamäte. Pre naväzovanie na niektoré udalosti bol využitý *.NET MAUI Community Toolkit*[6]. Jedná sa o kolekciu nástrojov a vizuálnych elementov, ktorý je voľne dostupný a zjednodušuje prácu v .NET MAUI. Príklad využitia v projekte:

```
<ContentPage.Behaviors>
        <toolkit:EventToCommandBehavior
            EventName="NavigatingFrom"
            Command="{Binding NavigatingFromCommand}"/>
</ContentPage.Behaviors>
```

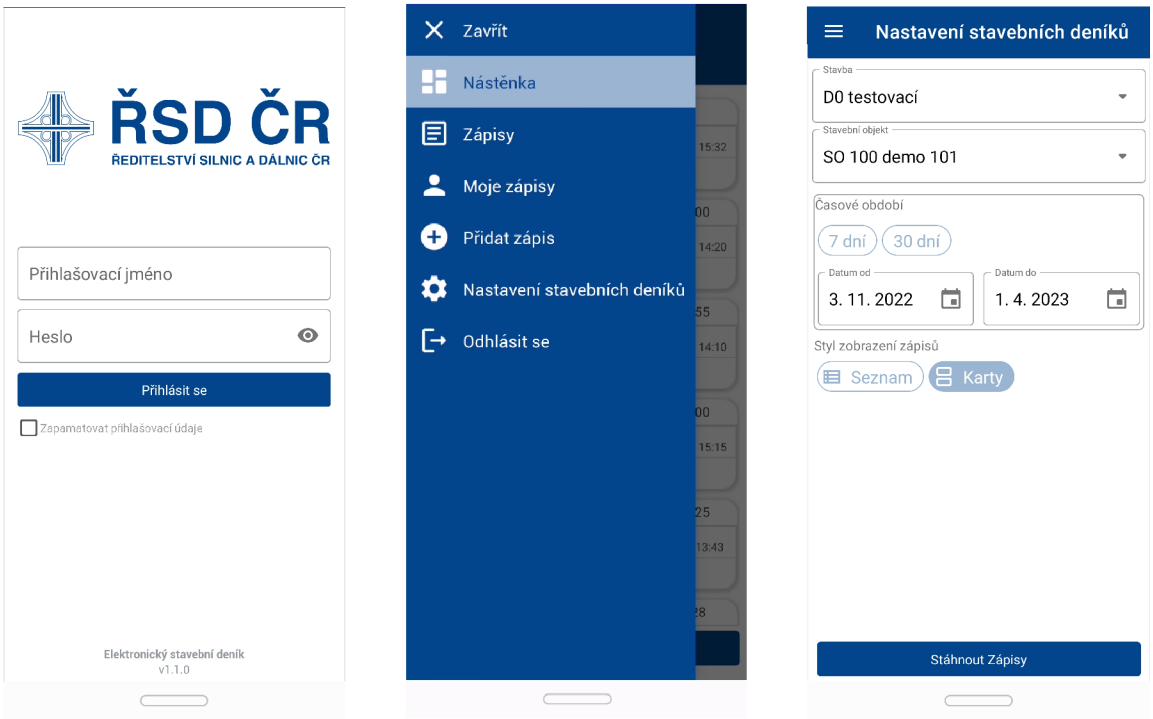
[6] https://github.com/CommunityToolkit/Maui

Aj keď .NET MAUI poskytuje niekoľko elementov grafického rozhrania, nestačí to na pokrytie potrieb a boli vytvorené aj vlastné. Tieto vlastné elementy sa potom dajú využiť na stránke ako prvok rozhrania. Takýto vlastný prvok je definovaný ako `ContentView` obsahujúci už existujúce elementy a definujúci pomocou nich nový. V prípade vlastného elementu je code-behind využitý vo väčšej miere a to najmä z dôvodu vytvorenia vlastností tohto prvku, na ktoré by sa dalo napojiť pomocou data bindingu. Vlastné elementy sa nachádzajú v priečinku `CustomControls`.

Každá stránka má svoju cestu registrovanú cez `AppShell`, ktorý definuje navigáciu. Toto je nutné aby bola navigácia medzi stránkami možná.

6.5 Finálne užívateľské rozhranie

Užívateľské rozhranie je tá časť aplikácie, s ktorou používateľ interaguje a predstavuje grafické zobrazenie dát. V tejto sekcii je popísané finálne grafické užívateľské rozhranie implementovanej aplikácie, ako aj ukážka niektorých obrazoviek z aplikácie.



(a) Prihlasovacia stránka (b) Menu (c) Nastavenia

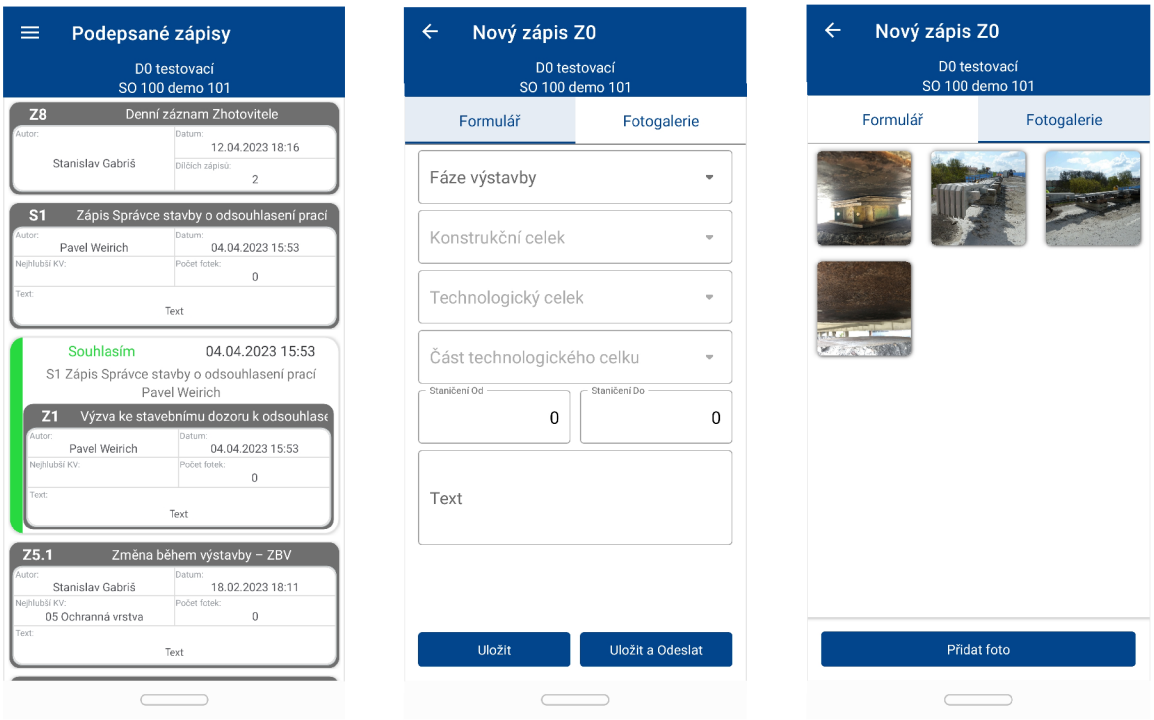
Obr. 6.3: Snímky obrazovky z aplikácie

Po spustení aplikácie je nutné prihlásiť sa. Na obrázku 6.3a je vyobrazené ako vyzerá táto prvá stránka s ktorou sa používateľ stretne. Stránka má jednoduchý dizajn s políčkami na zadávanie údajov umiestnenými tesne nad stredom obrazovky v zmysle *zlatého rezu*.

V prípade hlavného menu (obrázok 6.3b) bolo možné zvoliť niekoľko prístupov. Najpoužívanejšie verzie v existujúcich aplikáciách sú riešenia typu horný alebo dolný panel, alebo bočný panel. Ako hlavné menu bola v aplikácii zvolená verzia vysúvací bočný panel (v angličtine *flyout menu*). Tento štýl navigácie je využívaný v mnohých aplikáciách a veľa

používateľov je s ním oboznámená. Mimo toho boli využité aj prístupy horný a spodný panel ako podradené spôsoby navigácie.

Na obrázku 6.3c je obrazovka s nastaveniami stavebného denníku. Tu si uživateľ nastaví nastavenia, ktoré majú vpliv na celú aplikáciu. Vie si tu nastaviť konkrétnu stavbu, časové obdobie a štýl zobrazenia zápisov. Nastavená stavba je viditeľná vo všetkých relevantných stránkach.



(a) Výpis zápisov (b) Formulár zápisu (c) Galéria zápisu

Obr. 6.4: Snímky obrazovky so zápismi

Výpis zápisov, na obrázku 6.4a, sa nachádza vo viacerých obrazovkách. Sú to obrazovky `Podepsaná zápisy`, `Moje zápisy` a `Nástěnka`. Na týchto stránkach je možné aktualizovať zápisy potiahnutím prstom smerom dolu. Špeciálne typy zápisov majú odlišný štýl od obyčajných zápisov a na stránke nástenka je taktiež využitý rozličný štýl. Na obrazovke podpísané zápisy je po kliknutí na konkrétny zápis používateľ presmerovaný na detail tohto zápisu. V prípade stránky moje zápisy sa dá so zápismi prstom posunuť smerom doľava, čo umožní zápis buď upraviť alebo vymazať.

Obrázok 6.4b už vyobrazuje konkrétny formulár k typu zápisu. Tento formulár môže užívateľ vypisovať podľa jeho uváženia. Tlačítkom `Uložit` si používateľ zápis alebo zmeny na zápise uloží len do lokálneho uložiska. V prípade tlačítka `Uložit a Odeslat` sa aplikácia pokúsi odoslať zápis pomocou REST API do databázy.

Na poslednom obrázku 6.4c je zobrazená galéria ku konkrétnemu zápisu. Medzi formulárom a galériou sa dá navigovať posunutím prsta alebo prekliknutím vrchného menu. V galérii je možné pridať fotky pomocou tlačítka, kde bude po zakliknutí uživateľ vyzvaný aby si zvolil metódu vybrania fotografie a to buď vyfotením alebo vybraním z galérie. Fotka sa po vybraní pridá do galérie. Po zakliknutí konkrétnej fotografie je používateľ presmerovaný do detailu fotografie, kde je fotografia zobrazená vo väčšom rozlíšení a dá sa k nej pridať popis.

Tu popísané obrazovky zobrazujú hlavné stránky a grafické prvky aplikácie, no nevyobrazujú ich úplne všetky. Medzi zvyšné prvky patria rozdielne formuláre pri iných typoch zápisov, nástenka, vyskakovacie okná alebo už spomenutý detail fotky.

6.6 Inštalačný súbor

Jedným z posledných krokov pred nasadením je vytvorenie inštalačného súboru. Pre zariadenia Android je formát tohto súboru nazývaný *APK* (skratka z anglického Android Application Package) alebo *AAB* (skratka z anglického Android App Bundle), využívaný na distribúciu a inštaláciu mobilných aplikácií. APK obsahuje všetky zdroje potrebné na spustenie aplikácie na zariadení so systémom Android, vrátane kódu a obrázkov. Formát AAB je formát pre oficiálne vydanie v aplikácií Google Play, pomocou ktorého vie táto aplikácia sťiahnuť optimalizovanú verziu APK do každého zariadenia [1].

Pri vytváraní súboru APK a AAB je potrebné podpísať tento súbor podpisovacím kľúčom (anglicky tzv. app signing key alebo release key). Tento kľúč by mal byť utajený a bezpečný, pretože sa po vydaní aplikácie nedá zmeniť. Tento podpis stačí na distribúciu súboru APK inak než cez Google Play. Naopak ak je aplikácia vydávaná pomocou Google Play[7] je nutné súbor podpísať ešte tzv. *upload key* kľúčom. Google vie vygenerovať release key a v takom prípade bude upload key ten kľúč, ktorým bol podpísaný súbor AAB. [2]

V prvom rade je teda potrebné vygenerovať súbor *keystore*, v ktorom sa kľúč nachádza. V editore Visual Studio sa zatiaľ dá využiť pre celý proces vydávania len terminál. Na vygenerovanie súboru sa používa nástroj `keytool` a príslušné argumenty, ako typ kryptografie alebo obdobie validity kľúča v dňoch. Dodatočne je potrebné pridať heslo, ktoré súbor chráni, a aj osobné údaje, ako meno a priezvisko. Tento súbor je potrebné uložiť na bezpečné miesto a nestratiť (v prípade, že aplikácia bude vydaná cez Google Play). V hlavnom súbore `.csproj` projektu .NET MAUI je pridaný kód špecifikújúci cestu k súboru `keystore`. V prostredí Visual Studio je nastavený `Release` a nejaké zariadenie Android. Pre vytvorenie a podpísanie súborov AAB a APK je taktiež z príkazového riadku využitý príkaz `dotnet publish` s argumentami špecifikujú verziu .NET, heslo k súboru `keystore` a poprípade ďalšie argumenty. Výsledkom týchto krokov je súbor `publish` pod `bin/release` v štruktúre projektu, kde sa nachádzajú podpísané súbory APK a AAB pripravené na distribúciu a vydanie alebo testovanie.

Dodatočné informácie

Pre dokumentáciu zadania a postupu bola využitá platforma *Trello*[8]. Na verzovanie pri implementácii bol využitý nástroj *Git* s webovou aplikáciou *GitLab*[9].

V súbore `MauiProgram.cs` je inicializácia aplikácie aj s použitými knižnicami. Aplikácia vužíva koncept *Dependency injection* (skratka DI). Tento koncept znamená, že komponenty, ktoré vyžadujú určité závislosti, si tieto závislosti neinicializujú samy a miesto toho sú im poskytnuté na vyžiadanie. V tomto projekte je využitý najmä typ injekcie do konštruktoru (z anglického *Constructor Injection*), kde sú v už spomenutom súbore v metóde `ConfigureServices` inicializované jednotlivé závislosti a ďalej môžu byť využívané deklaráciou v konštruktore bez toho aby si ich museli jednotlivé komponenty spravovať.

[7] Google Play Console – vydávanie aplikácií na Android
[8] https://trello.com/
[9] https://about.gitlab.com/

Kapitola 7

Testovanie

Počas celého vývoja aplikácie boli v pravidelných intervaloch konzultované a testované postupy s Ředitelstvím silnic a dálnic ČR. V tejto kapitole bude popísané ako testovanie prebiehalo a aké boli jeho výsledky.

Testovanie bolo zložené z testovania funkčnosti a testovania použiteľnosti. Testovanie funkčnosti malo za úlohu ukázať, či aplikácia spĺňa vyžadovanú funkcionalitu, zatiaľ čo použiteľnosť sledovala či je grafické užívateľské rozhranie prívetivé a kde by sa dalo vylepšiť.

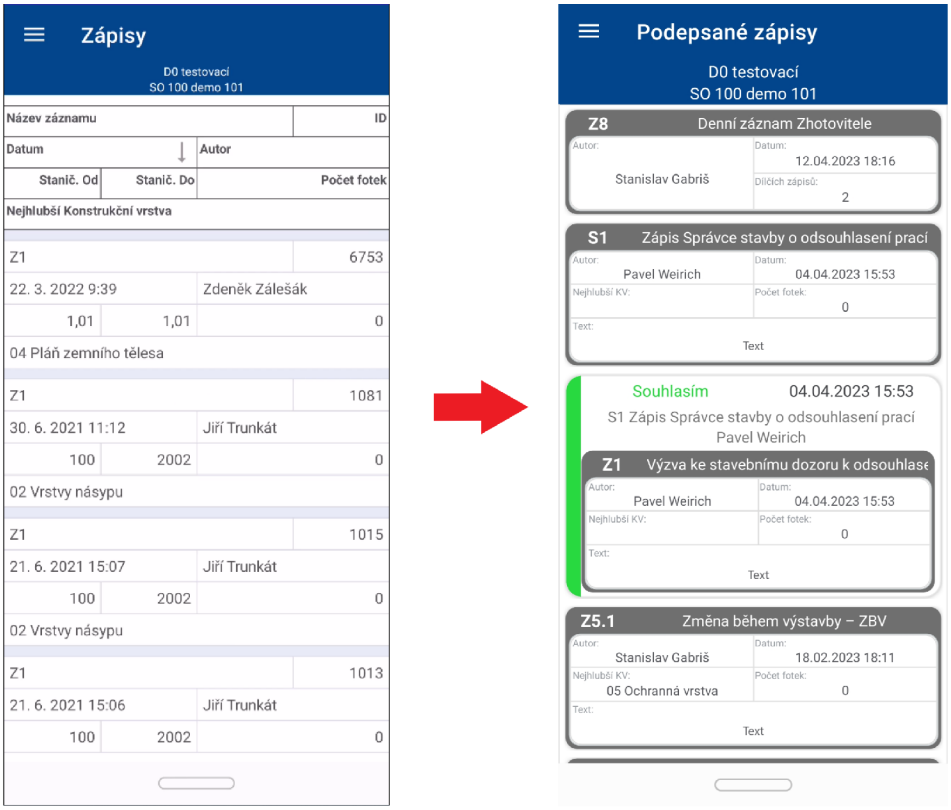
V začiatkoch vývoja, teda v zime 2022, bolo testovanie prevádzané na diaľku pomocou komunikačného prostriedku *Microsoft Teams*. Išlo hlavne o určenie či sa aplikácia vyvíja správnym smerom. Intenzívnejšie testovanie taktiež pomocou platformy Teams ale už aj distribúciou inštalačného súboru a osobným stretnutím bolo prevádzané začiatkom roku 2023.

Úlohy pri testovaní by sa dali zhrnúť do niekoľko bodov. Úlohy mali za cieľ otestovať aj grafické rozhranie a jeho intuitívnosť.

1. Prihláste sa do aplikácie pomocou rovnakých prihlasovacích údajov, ktoré využívate pri webovej aplikácii.
2. Nastavte si stavbu a časový filter a stiahnite zápisy.
3. Skúste pomocou navigácie pridať nový zápis
4. Vyplnte formulár a pridajte fotografie z galérie a následne vyfotením.
5. Navigujte sa do detailu fotografie a pridajte popis, zmeny uložte
6. Uložte zápis
7. Nájdite uložený zápis pomocou bočného menu a znovu ho upravte, následne ho odošlite.
8. Opäť sa navigujte k tomuto odoslanému zápisu.
9. Navigujte sa a otvorte podpísaný zápis.
10. Odhláste sa z aplikácie.

Niektorý používatelia mali ako prvý problém hneď pri inštalácii aplikácie. Tento problém však nebol spôsobený aplikáciou ale len povolením inštalácie z neznámich zdrojov do zariadenia. Pri testovaní bolo objavených niekoľko funkcionálnych chýb.

Ako problém sa ukázalo, že niektorím používateľom sa nedalo vyfotiť fotku priamo z aplikácie. Z testovania vyplynulo, že problém je na zariadeniach s novšou verziou systému Android, t.j. od Android 11 vyššie. Dôvod bol v použitej knižnici `MediaPicker` a riešením bolo upravenie súboru `AndroidManifest`. Problém nastal v neskorších testovaniach aj v galérii fotiek, kde nebolo možné po odstránení fotiek pridať novú fotku. Problém bol zatiaľ vyriešený prechodom na staršiu verziu .NET MAUI kým nebude opravený. Jedným z ďalších problémov bola aj duplikácia fotiek vo fotogalérii pri odosielaní zápisu.



Obr. 7.1: Evolúcia dizajnu na základe poznámok používateľov

Čo sa grafického rozhrania týka, aj to si prešlo pri testovaní svojimi úpravami podľa poznámok používateľov. Ukázalo sa napríklad, že používatelia nevedeli aký je rozdiel medzi záložkou `Zápisy` a záložkou `Moje zápisy` v menu. Konkrétnejšie nevedeli kde nájdu podpísané zápisy, preto bola stránka `Zápisy` premenovaná na `Podepsané zápisy`. Taktiež sa im zdalo, že text s vybranou stavbou má príliš malý font a je ťahšie čitateľný. Veľmi dôležitým bola poznámka, že pôvodný dizajn výpisu zápisov sa zdal neprehladný. Niektorý používatelia nechápali čo je na stránke vyobrazené a čo predstavuje zápis. Z tohto dôvodu bol dizajn výpisu prerobený aby bolo jasne viditeľné čo patrí ku konkrétnemu zápisu, a aby mali špeciálne zápisy odpovedajúci dizajn. Tieto zmeny sú vyobrazené na obrázku 7.1.

Kapitola 8

Záver

V rámci tejto bakalárskej práce bola vytvorená mobilná aplikácia na zariadenia Android vo frameworku .NET MAUI určená pre Ředitelství silnic a dálnic ČR. Aplikácia zastáva funkciu stavebného denníka a umožňuje používateľom pridávanie, úpravu a prehliadanie záznamov zo stavby. Ďalej je umožnené pridávanie fotografií k záznamom pomocou galérie alebo priamo fotoaparátu z aplikácie. Aplikáciu je možné používať aj v režime bez pripojenia na internet a rovnako je možné aplikáciu synchronizovať s už existujúcou webovou aplikáciou.

Na začiatku boli uvedené a porovnané rôzne prístupy k vývoju mobilných aplikácií. K týmto prístupom boli uvedené technológie a programovacie jazyky nimi využívané. V teórii boli taktiež opísané architektonické prístupy využité pri návrhu a implementácii riešenia. Následne bol približený problém a motivácia za celou bakalárskou prácou. Vysvetlené sú pojmy dôležité pre pochopenie problému ako aj existujúce riešenia tohto problému. V tejto časti sa nachádza aj popis webovej aplikácie s ktorou táto práca úzko súvisí, keďže komunikuje s rovnakou databázou. V návrhu riešenia sú približené informácie využité pri implementácii, ako aj diagram prípadov použitia, ktoré aplikácia umožňuje. Je tu zadefinované akou architektúrou sa aplikácia riadi a nakoniec prototyp, na základe ktorého bola aplikácia implementovaná ďalej. Pri implementácii sú už myšlienky z návrhu pretransformované do konečného riešenia a popis je rozdelený do logických častí podľa architektúry z návrhu. Uvedený je aj formát výstupného súboru vo forme inštalačného súboru. Nakoniec je popísaný proces testovania aplikácie na užívateľoch ako aj jeho výsledok. Ukázané je ako toto testovanie reálne ovplyvnilo finálny dizajn a aké nedostatky boli odhalené.

Výsledná aplikácia teda slúži na prácu so zápismi a komunikuje aj s databázou pomocou internetu, zatiaľ čo umožňuje aj prácu v režime offline. Aplikácia sa taktiež dá do budúcnosti rozšíriť. Medzi väčšie takéto rozšírenia patrí uvedenie aplikácie pre platformu iOS. Funkcionalita aplikácie sa dá rozšíriť napríklad o možnosť približovania a odďalovania fotiek, poprípade kreslenie do fotiek, pridanie filtrovania podľa viacerých kategórii priamo vo výpise záznamov alebo umožnenie podpisovať zápisy priamo z mobilného zariadenia.

Literatúra

[1] Android Open Source Project. *About Android App Bundles* [online], 8. marca 2023 [cit. 2023-4-20]. Dostupné z: https://developer.android.com/guide/app-bundle.

[2] Android Open Source Project. *Sign your app* [online], 12. apríla 2023 [cit. 2023-4-20]. Dostupné z: https://developer.android.com/studio/publish/app-signing#generate-key.

[3] Bohon, C. *Apple's Swift programming language: Cheat sheet* [online], 25. októbra 2021 [cit. 2023-3-16]. Dostupné z: https://www.techrepublic.com/article/apples-swift-programming-language-the-smart-persons-guide/.

[4] Britch, D. a Gechev, I. *What is .NET MAUI?* [online], 30. januára 2023 [cit. 2023-3-16]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0.

[5] Guo, L. *The First Line of Code: Android Programming with Kotlin.* Singapore: Springer, 2022. ISBN 981191799X.

[6] Gupta, L. *What is REST* [online], 7. apríla 2022 [cit. 2023-3-22]. Dostupné z: https://restfulapi.net/.

[7] McWherter, J. a Gowell, S. *Professional Mobile Application Development.* John Wiley & Sons, Inc., 2012. ISBN 978-1-118-20390-3.

[8] Motzke, R. *Stavební deník: Jak ho vést? Je nutný i pro stavbu svépomocí? Příklad z praxe* [online], 13. februára 2018 [cit. 2023-3-25]. Dostupné z: https://www.estav.cz/cz/5688.stavebni-denik-jak-ho-vest-je-nutny-i-pro-stavbu-svepomoci.

[9] Ozkaya, M. *Layered (N-Layer) Architecture* [online], 6. septembra 2021 [cit. 2023-3-20]. Dostupné z: https://medium.com/design-microservices-architecture-with-patterns/layered-n-layer-architecture-e15ffdb7fa42.

[10] Ramírez, S., Epstein, T., Ellis, R. et al. *OpenAPI Specification* [online], 21. februára 2020 [cit. 2023-3-23]. Dostupné z: https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.3.md.

[11] Randhawa, T. S. *Mobile Applications: Design, Development and Optimization.* Cham: Springer International Publishing AG, 2022. ISBN 978-3-030-02389-8.

[12] Schmitt, J. *Native vs cross-platform mobile app development* [online], 24. augusta 2022 [cit. 2023-3-16]. Dostupné z: https://circleci.com/blog/native-vs-cross-platform-mobile-dev/.

[13] StatCounter. *Mobile Operating System Market Share Worldwide* [online]. 2023 [cit. 2023-2-20]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[14] Stonis, M., Jain, T. a Pine, D. *Model-View-ViewModel (MVVM)* [online], 4. novembra 2022 [cit. 2023-3-20]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm.

[15] Wagner, B., Dykstra, T., Sharkey, K. et al. *A tour of the C# language* [online], 13. februára 2023 [cit. 2023-3-17]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.