KLOZET - VEŘEJNÉ ZÁCHODY

Maturitní práce

AUTOR

Marek Fořt

VEDOUCÍ PRÁCE

Karel Jílek

OBOR VZDĚLÁNÍ

Gymnázium 79-41-K/81

MÍSTO A ROK VYDÁNÍ

Praha, 2018

ANOTACE

Maturitní práce se zabývá rozšířením a zlepšením aplikace na veřejné záchody, Klozet. Aplikace je napsána ve Swiftu, backend poté v Pythonu. Během vývoje jsem nejdříve musel vyřešit, jak bude vypadat UI, poté ho implementovat do aplikace tak, aby zároveň odpovídala zásádám UX a HIG Applu. Kromě toho jsem musel vymyslet komunikaci se serverem - především tedy čtení, přidávání a editování záchodů v databázi.

ANNOTATION

This seminar thesis concerns itself with expanding and imporving of application for public toilets, Klozet. Application is written in Swift, backend is written in Python. During the development I had to first figure out how the UI would look like, then I had to implement it into application, so it is according to UX and Apple's HIG. Besides that I had to think through the communication with the server - that is reading, adding and editing of toilets in database.

PROHLÁŠENÍ

Prohlašuji, že jsem jediným autorem této maturitní práce a všechny citace, použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium Jana Keplera, Praha 6, Parléřova 2 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

…….

podpis autora

SEZNAM ZKRATEK

HIG	Human Interface Guidelines
IDE	Intergrated Development Environment
JSON	JavaScript Object Notation
MVC	Model-View-Controller
MVVM	Model-View-ViewModel
REST	Representational State Transfer
UI	User Interface
UX	User Experience

OBSAH

ÚVOD

Snad každý z nás se již potkal s opravdu palčivým problémem - člověk potřebuje na záchod, ale nemůže najít žádný veřejně přístupné toalety. Tento problém jsem se snažil vyřešit pomocí mobilní aplikace, jelikož mobil má dnes snad už téměř každý neustále při sobě.

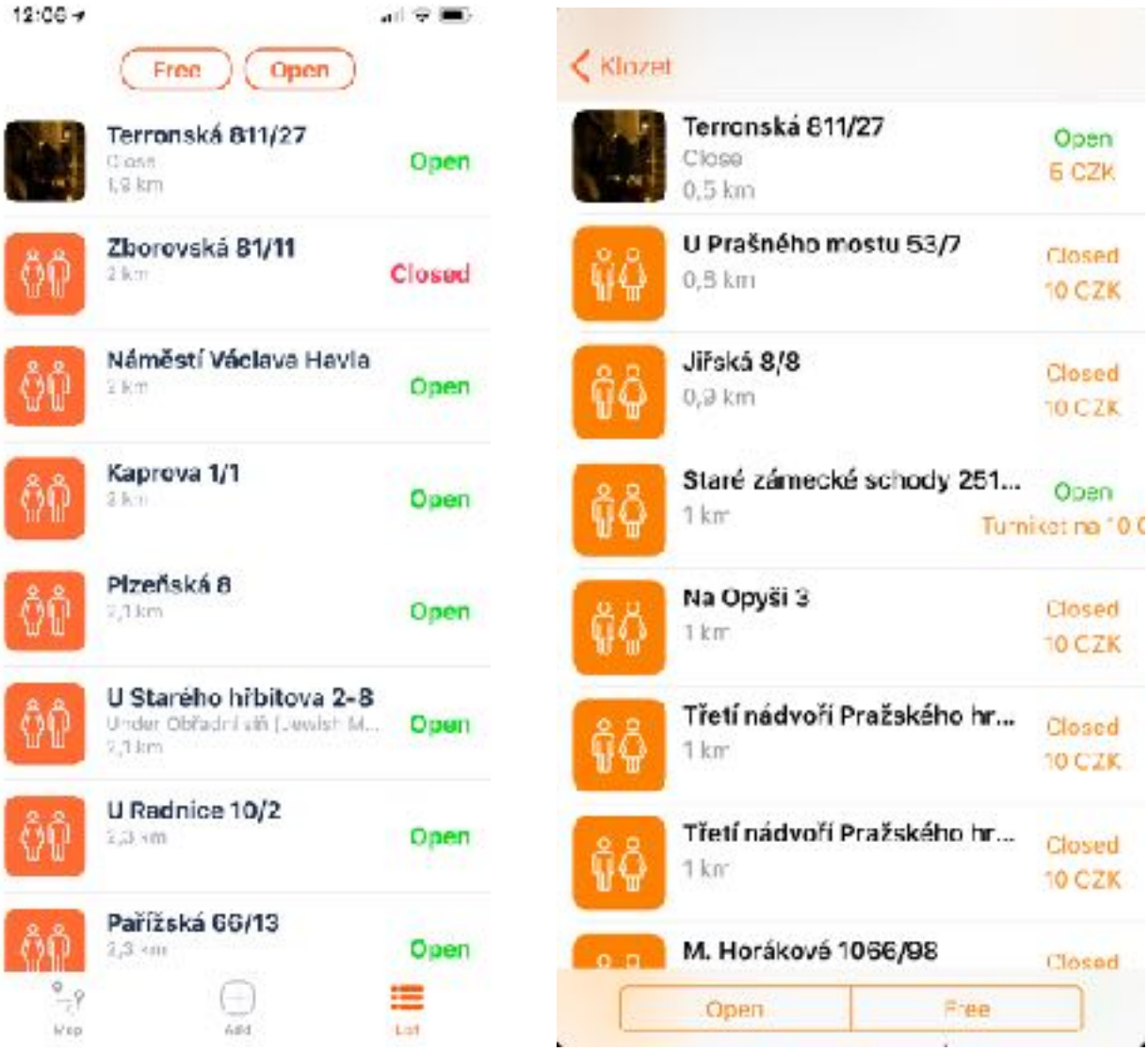
Ze začátku jsem začal se statickou databází z portálu geoportalpraha.cz, v rámci maturitního projektu jsem si dal za úkol aplikaci rozšířit a zakomponovat do ní přidávání záchodů, dávající možnost uživatelům databázi veřejných záchodů rozšiřovat. Kromě toho jsem do aplikace přidal i možnost úpravy záchodů.

To pro mě znamenalo změnit design, aby se daly zmíněné funkce co nejsmysluplněji přidat. Při vytváření nového designu jsem se rozhodl předělat i předcházející design, aby odpovídal té nejlepší současné praxi, kterou udává Apple v již zmíněných HIG.

Pár aplikací už na tento způsob funguje, ale má byla první, která fungovala v ČR. Nedávno se na Appstore objevila ještě aplikace WC kompas, která ale momentálně vypadá pouze jako port Android verze.

DESIGN

Před tím, než jsem vůbec začal vytvářet něco na počítači, jsem musel vymyslet základní rozvržení aplikace, co bude jaké tlačítko dělat, atd. První rozvržení jsem tedy udělal na papíře, poté už jsem se přesunul do aplikace Sketch. Největší změnu oproti původnímu designu bylo přidání komponenty UITabBarController. V předchozím designu se k seznamu přistupovalo pomocí UINavigationBarItem, který byl umístěný v pravém horním rohu, čímž ale už nezbývalo místo pro přidání záchodu. UITabBarController tedy představoval jednoduché a elegantní řešení tohoto problému. Dále jsem pro většinu textu zvolil tučnější font, zlepšujíc tak čitelnost a navazujíc na iOS 11 design. S tím souvisí i zvolení sytějších barev. Tato změna je například vidět na seznamu záchodů na obrázku níže:



Design aplikace před (vlevo) a poté

Co se týče mého workflow, tak kromě Sketche a několika skvělých pluginů bych ještě rád zmínil Zeplin. Zeplin mi umožnil rychle exportovat assets ve všech potřebných rozlišeních do Xcode projektu. Zároveň automaticky exportuje i barvy, které najde ve Sketch dokumentu. Díky tomu jsem mohl k barvám jednoduše přistupovat přes UIColor. Co se týče UIImage, tak jsem se chtěl vyvarovat definování pomocí String. Proto jsem se rozhodl pro knihovnu Swiftgen, která mi umožnila zapisovat obrázky pomocí enum Asset, takže jsem se nemusel bát žádných překlepů a navíc jsem k tomu i dostal automatické Xcode napovídání.

FRONTEND

Volba programovacího jazyku byla poměrně jednoduchá. Jelikož jsem chtěl vytvořit iOS native aplikaci, zbývaly mi pouze dvě možnosti - Swift a Objective-C. Vzhledem k tomu, že dnes je komunita Swiftu živější a i já sám Swift umím líp, představoval pro mě Swift jasnou volbu. Jako IDE je pro iOS native vývoj téměř jedinou a zároveň nejrozšířenější možností Xcode, který má na starosti samotný Apple.

První, do čeho jsem se pustil, bylo předělání toho, jak aplikace komunikuje se serverem, respektive jak pracuje s REST. Předchozím řešením bylo použití knihovny Alamofire, která v době, kdy já jsem začal pracovat na projektu, nebyla konvertovaná do nejnovější verze Swiftu, navíc s příchodem Swift protokolu Codable se mi zdála zbytečně robustní. Proto jsem vytvořil class APIService, která s Codable pomocí generics posílá REST požadavky na můj server.

Poté společně s APIService jsem implementoval novou architekturu aplikace - přešel jsem od MVC k MVVM, jelikož už před rozšířením byly některé z UIViewControllers značně velké, takže MVVM rozdělením ViewControlleru na View a ViewModel se pomohlo v kódu lépe orientovat. Kromě přechodu k přehlednější architektuře jsem se rozhodl ještě pro jeden krok - async kód se velmi často stane složitým a člověku se obrazovka rychle zaplní closures. Proto jsem se rozhodl, že je načase využít reactive programování, které je pro práci s async funkcemi a reagovaním na ně ideální. Jako reactive knihovnu jsem zvolil ReactiveSwift, jelikož podle mě lépe odpovídá Swift syntaxi než druhá velmi populární varianta RxSwft. Obě knihovny ale fungují velmi podobně a záleží spíše na osobních preferencích. Každopádně jsem se zbavil poměrně složité práce s queues, místo nich jsem ve ViewModelu jednoduše stáhnul data a poté pomocí SignalProducer upozorňoval View o nových datech. Ve funkci setupBindings() ve View jsem poté definoval, jak se má na nová data reagovat.

Dalším krokem bylo převedení mého designu, který jsem měl v aplikaci Sketch, do kódu. Někteří preferují práci s UIStoryboard, mně osobně se lépe ale pracuje, když mám celý design v kódu. Kromě převedení nového designu jsem tedy musel předělat i ten, co jsem před tím měl pouze ve vizuální podobě.

Kromě převádění designu jsem musel i domyslet práci se serverem, což však díky již zmíněným Codable a mnou vytvořeným APIService bylo poměrně jednoduché.

BACKEND

Rozšíření aplikace se samozřejmě neobešlo bez velkých změn na backendu. Většinu backend kódu jsem napsal v Pythonu. Zvolil jsem jednoduché řešení v podobě Flask, čímž jsem se robustním systémům jako Django, jelikož Flask pro účel aplikace bohatě stačil a nijak jsem této volby později nelitoval. Čeho jsem ale litoval, byl způsob, kterým jsem před tím ukládal záchody. Jelikož jsem původně v aplikaci zobrazoval data jenom ze statické databáze, bylo tehdy nejjednodušším řešení posílat statický JSON soubor. Toto řešení ale naprosto nedostačovalo novým funkcím přidávání a upravování záchodů.

Tudíž jsem sáhl po MySQL pro ukládání dat a pymysql knihovnu pro práci s nimi. Nejdůležitější table představovala table toilets (viz obrázek diagram toilets níže)

Myslím, že u většiny polí je poměrně jasné, jak fungují, již z předešlého obrázku. Jediná položka, která nemusí být jasná, je unchecked_hours, ke které se dostanu až dále, až budu popisovat ukládání obrázků. Další table jsou open_times, které jsou pomocí foreign key navázány na toilet_id z toilets. K tomuto kroku mě vedlo to, že popisování, kdy jsou záchody otevřené, je poměrně složité a tato struktura mi připadala nejpřehlednější a nejsnadnější na budoucí údržbu a zároveň udržela table toilets dostatečně malou, aby byla ještě přehledná. Poslední table je edit_toilets - je velice podobná table toilets. Zde se zaznaménavají všechny poslané úpravy záchodů. Já, potažmo i někdo jiný s přístupem k databázi, poté tyto dotazy projede, vyhodnotí a změní hlavní table toilets na základě poslaných dat. Samozřejmě, že by bylo lepší, kdyby celý tento proces byl automatizovaný, ale v tomto případě člověk musí počítat s

tím, že ne všechny úpravy zaslané uživateli budou správné, a proto zde musí přijít na řadu lidská kontrola.

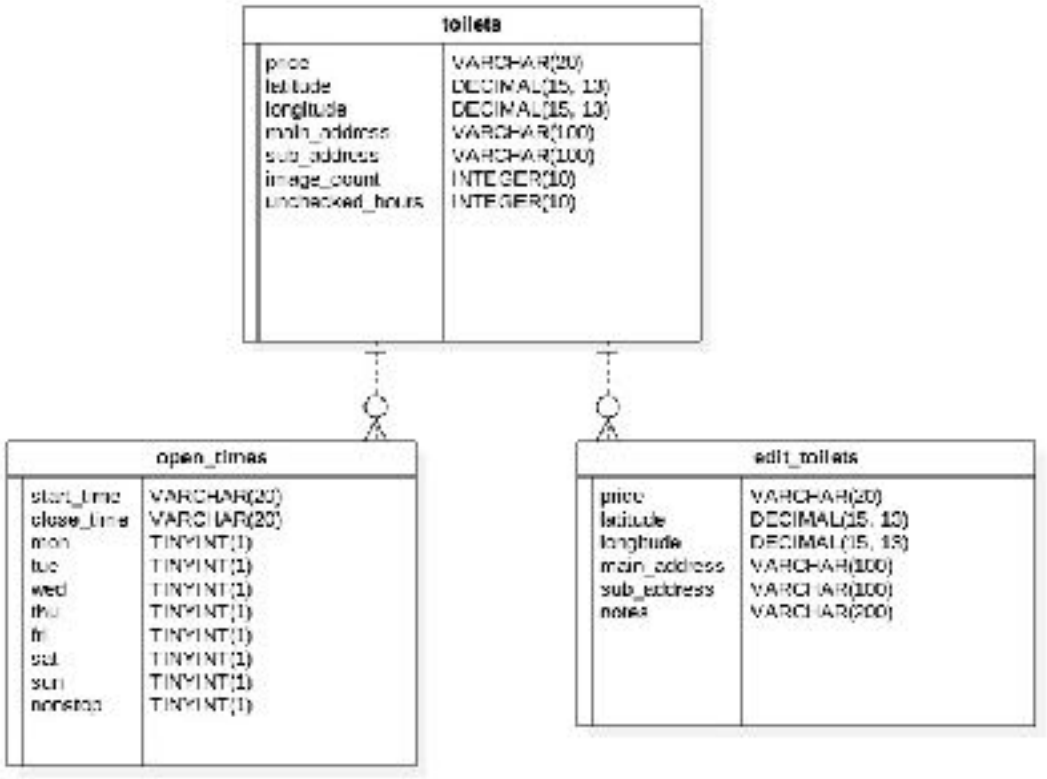


Diagram toilets

OBRÁZKY

Ukládat obrázky do MySQL by byl dost nestandardní postup, místo toho jsem tedy pro každý vytvořil složku, ze které potom mohla aplikace obrázky stahovat. Systém ukládání a stahování jsem nastavil takto: při vytvoření každého nového záchodu se také ve složce toilet_img a hours_img vytvoří nová složka pojmenovaná podle id záchodu. Přičemž toilet_img je složka pro ukládání fotek záchodů, hours_img poté pro fotky, kde uživatel vyfotil otevírací hodiny. Když se pošle nová fotka záchodu, získáme z table toilets pole image_count, aby se poté image dal přidat příslušny index, podle kterého půjde stáhnout. Např. tedy pokud je image_count 3 a toilet_id 99, přidá se do dané složky obrázek s adresou /toilets_img/389/2.jpg. Zároveň se vytvoří

i minified verze 2_min.jpg, která slouží pro zobrazování obrázků v listu, kde bylo potřeba se vypořádat s tím, že obrázky se musí zobrazit rychleji a tvoří se větším počtu, na rozdíl od detailu záchodu. Zároveň je uživatelsky přívětivější, když budou mít obrázky menší velikost a aplikace tak nebude tak datově náročná.

Co se týče obrázků otevřených hodin, funguje proces zpracování víceméně stejně. V table se ukládají do pole unchecked_hours - tudíž tyto fotky čekají na to, až je stáhnu a překonvertuji data do MySQL. Samozřejmě pokud by se počet přidávaných fotek nějak razantně zvětšil, nemohl bych vše řešit osobně a pravděpodobně bych musel vymyslet robustnější a efektivnější řešení. Každopádně myslím, že nedává smysl přemýšlet příliš, jak bude fungovat systém s tisíci uživateli, když takových čísel zatím zdaleka nedosahuje.

PLÁNY DO BUDOUCNA

Momentálně největším problémem, na který bych mohl narazit je údržba aplikace. Ta je bohužel v momentální podobě docela náročná. Pravděpodobně nejnáročnější proces pro administraci je pravděpodobně přepisování časů, kdy jsou záchody otevřené. Teď sice jde v systému poměrně snadno najít, které záchody mají neověřené fotky časů, ale poté si už musím tyto fotky stáhnout manuálně a manuálně je přepsat. Kdybych se měl připravit na to, že aplikaci bude využívat mnohem více uživatelů, potřeboval bych tento proces zautomatizovat. Zároveň by se i nabízela možnost, že časy rozepíše sám uživatel, což by bylo pro administraci úplně jednodušší (ale zase by se možná snížil počet poslaných časů, jelikož by to mohlo uživatele odradit).

Kromě zjednodušení administrace bych také časem rád implementoval hodnocení záchodů a tipy (např. čeho se vyvarovat, atd.). Pokud by se aplikaci hodně dařilo, tak by se asi daly přidat účty, komentáře k fotkám, notifikace…ale to už se možná prozatím dostávám příliš daleko.

KDE PROJEKT NAJÍT

Aplikace je dostupná na Appstore pod jménem Klozet - Prague Toilets (https://itunes.apple.com/US/app/id1170530956?mt=8).
Projekt je také volně dostupný na Githubu (https://github.com/fortmarek/Klozet)

POUŽITÉ TECHNOLOGIE

Sketch - design editor; https://www.sketchapp.com
Apple Developer Documentation - dokumentace frameworků jako UIKit, atd.; https://developer.apple.com/documentation
Swift - Apple opensource programovací jazyk; https://github.com/apple/swift
MySQL - jednoduchá práce s databázemi; https://www.mysql.com
Python - programovací jazyk; https://www.python.org
ReactiveSwift - reaktivní framework pro Swift; https://github.com/ReactiveCocoa/ReactiveSwift
Flask - web development; http://flask.pocoo.org