

Gymnázium, Praha 6, Arabská 14

Programování

Teamový ročníkový projekt

SmartBox



Prohlášení

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna 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, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne 29. 4. 2020

Anotace

Zadáním této práce je vytvoření dočasné úschovny, pronajímanou pomocí mobilní aplikace. Úschovnu bude představovat prototypní box se zámkem. Zámek bude ovládaný mobilním telefonem a bude připojený k serveru, komunikujícím s mobilní aplikací.

Klíčová slova

Smart-Box; úschovna; API

Annotation

The assignment of this work is to create a temporary vault, rented using a mobile application. The depository will be a prototype box with a lock. The lock will be controlled by a mobile phone and will be connected to a server communicating with the mobile application.

Keywords

Smart-Box; depository; API

Obsah

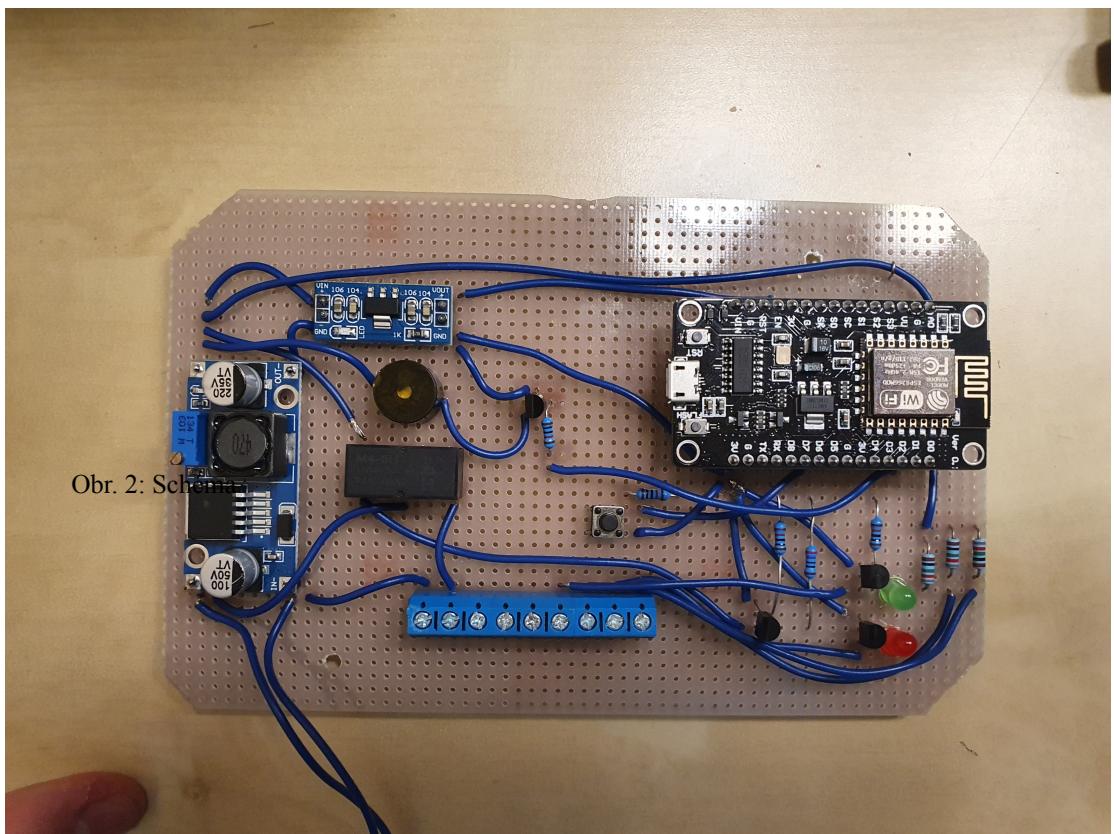
1.	Úvod	6
1.	Hardware.....	7
1.1.	Součástky	7
1.1.1.	Řídící deska.....	7
1.1.2.	Napájecí modul 5-45V na 3-40V/3A (step-down).....	8
1.1.3.	Stabilizátor napětí 4,5V-7V na 3,3V/0,5A.....	8
1.1.4.	Relé cívka M4-5H 5VDC	8
1.1.5.	Sirénka KXG1205C	8
1.1.6.	3MM zelené a červené LED diody	8
1.1.7.	Mikrospínač TD-03XA-T SMD	9
1.2.	Plošný spoj	9
2.	Box.....	11
2.1.	Výběr materiálu.....	11
2.2.	Tvorba prototypu.....	11
2.2.1.	První pokus	11
2.2.2.	Druhý pokus.....	11
3.	Server	12
3.1.	Výběr technologií.....	12
3.2.	API	12
3.3.	REST	12
3.4.	Python	12
3.5.	Django	12
3.6.	Django Rest framework	12
3.7.	Djoser	13
3.8.	Heroku	13
3.9.	PostgreSQL	13
3.10.	Postman	13
3.11.	Zabezpečení.....	13
3.11.1.	Django autentifikace	13
3.11.2.	Django REST framework autentifikace.....	13

3.11.3. Djoser	14
3.12. Popis funkce API	14
3.12.1. Komunikace s mobilní aplikací	14
3.12.2. Komunikace se zámkem	14
3.13. Modely	14
3.13.1. User	14
3.13.2. Box	14
4. SmartBox (aplikace)	15
4.1. Použité technologie	15
4.1.1. FloatingPanel	15
4.2. Uživatelské prostředí.....	15
4.3. Funkcionalita.....	15
4.4. Serverová komunikace.....	17
3. Závěr	18
4. Použitá literatura	19
5. Seznam obrázků a tabulek	19

1. ÚVOD

V dnešní době se ve světě objevuje trend sdílených služeb a na tomto konceptu jsme založili i náš projekt. Cílem našeho projektu je vytvořit službu sdílených prostor pod komerčním názvem SmartBox. Služba má nabízet uživateli úložný prostor, který je zabezpečený pomocí elektromagnetického zámku ovládaného přes iOS zařízení s aplikací SmartBox. Dále má vytvořit co nejpřehlednější prostředí s prvky jako je snadná registrace, hledání boxu v mapách a snadné ovládání boxů.

1. HARDWARE



Obr. 2: Schéma

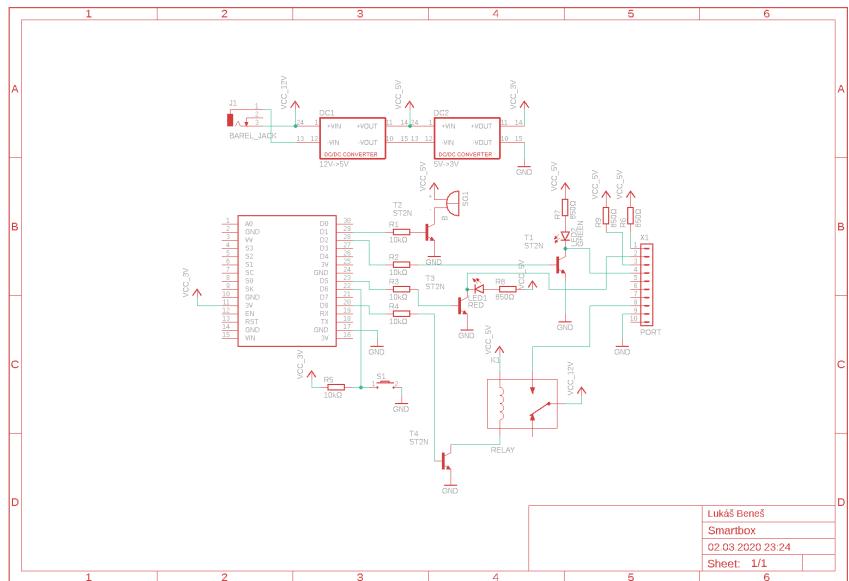
Obr. 1: Obvod

1.1. Součástky

1.1.1. Řídící deska

Celá hardwarová část je ovládána vývojovou deskou NodeMcu, která je založena na WiFi mikročipu ESP8266 od společnosti Espressif. Od stejné společnosti je i používaný mikrokontrolér ESP-12E. Taktovací frekvence mikrokontroléra je 80MHz s možností přetaktování až na 160MHz. Vývojová deska obsahuje Flash úložiště o velikosti 4MB a 11 univerzálních vstupních/výstupních pinů (GPIO). Mikrokontrolér je možné programovat mnoha programovacími jazyky. Pro tento projekt byl zvolen programovací jazyk Arduino.

Dalšími zvažovanými deskami byla například deska Arduino Uno, která nebyla vybrána z důvodu nutnosti dokoupení dalších součástek pro připojení k internetu. K prvotnímu testování projektu byla použita deska Arduino propojená s ESP-01 ve kterém se ESP-01 chovalo jako WiFi Shield a umožňovalo se Arduinu připojit k wifi síti. Toto zapojení bylo, ale zavrženo pro jeho technickou složitost. Samotné ESP-01 nebylo možno použít z důvodu nedostatku univerzálních pinu (GPIO), které má oproti ESP-12E pouze dva.



Obr. 2: Schéma

1.1.2. Napájecí modul 5-45V na 3-40V/3A (step-down)

Napájecí modul s řídícím čipem LM2596 je schopen snížit vstupní napětí v rozmezí od 5 až do 45V na napětí od 3 do 40V. V našem případě je modul použít na převedení 12V ze vstupního zdroje potřebných na napájení elektromagnetického zamku na 5V potřebných k napájení relé, pípáku a světelných ledek.

1.1.3. Stabilizátor napětí 4,5V-7V na 3,3V/0,5A

Stabilizátor s hlavním čipem AMS1117 je snižuje vstupní napětí v rozmezí od 4,5 až do 7V na stabilních 3,3V (přebytečných 1,7V stabilizátor tepelně vyzáří). Stabilních výstupních 3,3V je použito k napájení řídící desky.

1.1.4. Relé cívka M4-5H 5VDC

Relé se spíná puštěním proudu do pinu D8 přes tranzistor T4 (viz schéma). Sepnutím relé se 12V ze zdroje propojí se zámkem přes svorkovnici.

1.1.5. Sirénka KXG1205C

Sirénka KXG1205C je pěti voltová magnetická sirénka o hlasitosti 85 decibelů. Sirénka se spouští puštěním proudu do pinu D1 přes tranzistor T2 (viz schéma). Sirénka je vždy zapnuta při odemčení boxu aby upozornila uživatele.

1.1.6. 3MM zelené a červené LED diody

Box obsahuje dvě interní vývojové barevné ledky a dvě externí indikační barevné ledky pro uživatele. Externí ledky jsou zapojeny ve svorkovnici a jsou rozsvíceny vždy ve stejnou chvíli

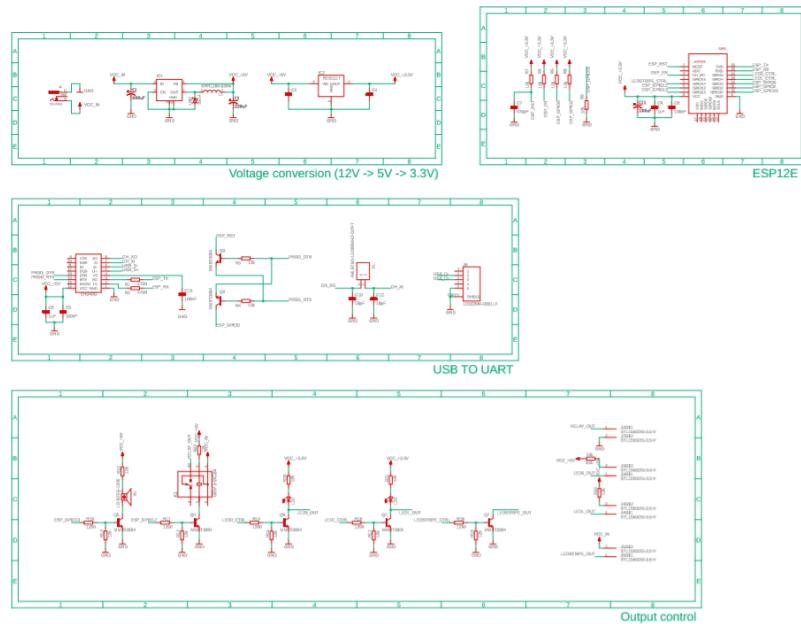
jako jejich interní barevné protějšky. Zelené ledky jsou rozsvíceny puštěním proudu do pinu D2 přes tranzistor T1 a červené puštěním proudu do pinu D5 přes tranzistor T3 (viz schéma).

1.1.7. Mikrospínač TD-03XA-T SMD

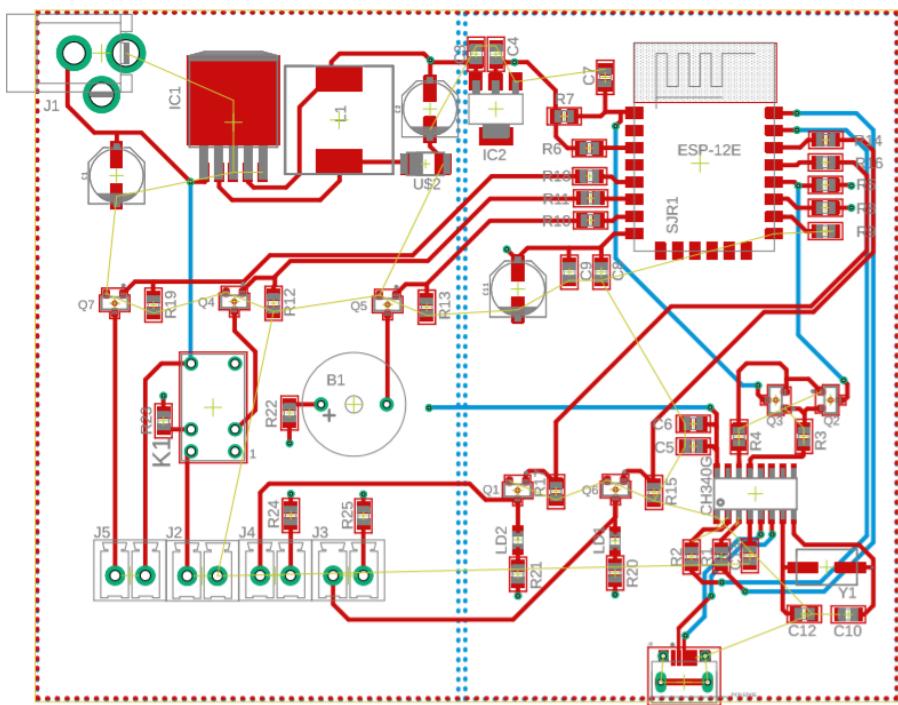
Jeho sepnutí je indikováno proudem na pinu D6(viz schéma). Spínač je použit na uvedení chipu do nastavitelného modu ve kterém chip vytvoří wifi access point. Po připojení k němu je možné na speciální IP adresu vybrat a vyplnit jméno a heslo wifi sítě ke které se má chip připojit. Toto umožňuje Arduino knihovna WifiManager.

1.2. Plošný spoj

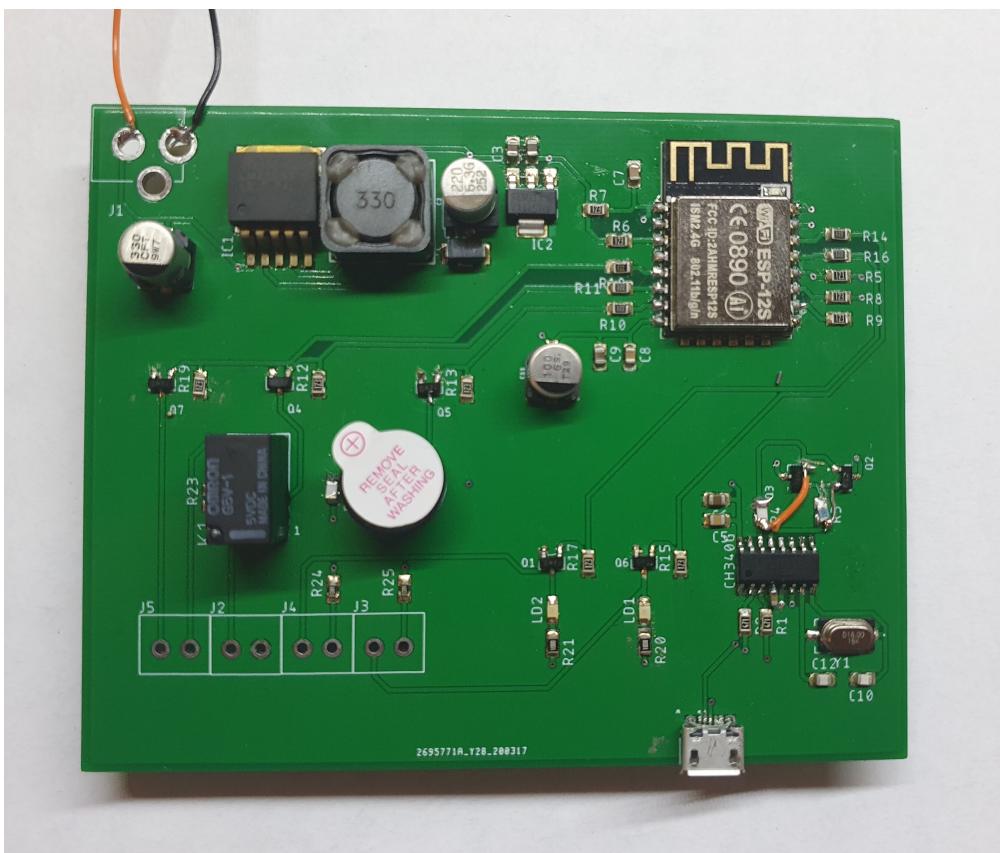
Pro realizaci projektu s větším množstvím úschoven je navrhnut, podle prototypu s menšími úpravami plošný spoj. Na osazeném plošném spoji se nachází identické zapojení USB TO UART převodníku s mikročipem CH340G jako na vývojové desce Nodemcu, která byla použita v prototypu. Dále je také použito stejné zapojení napájecího modulu a stabilizátoru jako u prototypu. Jediným rozdílem je, že napájecí modul je napevno nastaven na potřebných 5 voltů.(viz. schémaPCB) V první revizi došlo ke špatnému zapojení dvou odporů a zapomenutí jednoho kondenzátoru u sériového převodníku, ale i přesto byl po rychlé opravě obvod funkční.(viz. plošný spoj)



Obr. 3: Plošný spoj



Obr. 4: Deska PBC – schéma



Obr. 5: Deska PBC

2. Box

2.1. Výběr materiálu

První vizí jak vytvořit náš box bylo vymodelovat jej v nějakém 3D editoru a následně vytisknout ve 3D tiskárně, kterou naše gymnázium nabízí volně k využívání. Následně jsme ovšem našli spoustu nedostatků jako například maximální velikost dílu o rozměrech 25x25 centimetrů nebo obrovská spotřeba materiálu. Navíc jsme již při řešení tohoto problému měli obstaraný zámek a arduino desku pro které by velikost boxu byla absolutně nedostačující a pro uživatele by byl box naprosto nevyužitelný. Následovaly tedy návrhy materiálu typu železná či ocelová konstrukce dokonce i použití železobetonu, vše jsme následně z finančních důvodů a z důvodu složité konstrukce a následné obtížné manipulaci odmítli. Poté vzešel nápad, že nám umožní využít jeden ze spolupracovníků z týmu svůj starý počítačový stůl. Proto jsme se rozhodli vytvořit box z klasické a běžně dostupné dřevotřísky, jen jako prototyp pro snazší představu jak by náš finální produkt mohl vypadat. Po sestavení vypadal box naprosto neprofesionálně a po další skupinové poradě jsme se rozhodli postavit box od základů.

2.2. Tvorba prototypu

2.2.1. První pokus

Box jsme chtěli vytvořit v co nejvíce kompaktních rozměrech, aby byl box snadno přenosný na místo prezentace a zároveň, aby bylo možné demonstrovat jeho využití v praxi. Rozhodli jsme se tedy pro rozměry 30x30x30cm, přičemž zde nastala první komplikace. Nechali jsme nařezat šest naprosto shodných desek, které jsme měli v plánu následně upravit, což se s našimi základními pomůckami pro úpravu materiálu stalo nemožným. Nepodařil se nám ani jeden rovný řez, čímž byl materiál znehodnocen.

2.2.2. Druhý pokus

Na druhý pokus jsme si již připravili podrobný plán konstrukce. Na svrchní, spodní a zadní část jsme použili dřevotřískové desky o rozměrech 30x30cm a z bočních desek jsme odečetli tloušťku svrchní a spodní (2x18mm) desky pro zachování původně dohodnutých rozměrů 30x30x30 výsledné rozměry pro boční desky byly 30x26,4cm. Při návrhu dveří jsme využili 2mm rezervy z každé strany pro plynulé otevírání (rozměry 26x26cm) a zavírání dveří. Dveře drží na dvou pantech. Pro tuto desku je zvolena odlišná barva pro estetický efekt. Dále je na přední desce madlo pro snadné otevírání dveří. Vnitřek boxu je dále osazen dvěma ledkami, jelikož byl samotný box velice tmavý a některým potenciálním uživatelům by mohl připadat vnitřek boxu nepřehledný. Vývojovou desku jsme z technických důvodů umístili do venkovní části boxu a pro vývod kabelů musel být vyvrtán do spodní desky malý otvor. Desky drží po hromadě pomocí úhlových spojek a šroubů o délce 13mm.

3. SERVER

3.1. Výběr technologií

Důležitou součástí mojí části ročníkové práce byl výběr správných technologií pro tvorbu API. Nejprve jsem zvažoval použití frameworku Symfony pro jazyk PHP. Po krátkém odzkoušení Symfony jsem se ale rozhodl použít programovací jazyk Python a framework Django. K tomuto rozhodnutí mě vedlo to, že Django právě probíráme v hodinách programování a to, že pro Python existuje Django REST framework, který mi byl doporučen.

3.2. API

Application programming interface je rozhraní pro programování aplikací. Jeho hlavními výhodami je snadnější udržování a možnost měnit jednotlivé části aplikace nezávisle na sobě. Důvodem pro tvorbu API v případě našeho projektu byla možnost vytvořit dobře udržovatelnou aplikaci. Dalším důvodem byla možnost vytvořit aplikaci pro jiné platformy bez nutnosti přepisování již existujícího kódu pro jiné platformy.

3.3. REST

REST (Representational State Transfer) je architektura API, pomocí které lze jednoduše vytvářet, editovat, číst a mazat data z databáze serveru pomocí protokolu HTTP.

3.4. Python

Python je vysokoúrovňový skriptovací programovací jazyk, který v roce 1991 navrhl Guido van Rossum.

3.5. Django

Django je open source webový framework pro Python, který se volně drží MVC (Model-View-Controller) architektury.

3.6. Django Rest framework

Django Rest framework slouží k tvorbě webových API. Používají jej společnosti jako Mozilla, Red Hat a Heroku. Byl hlavním důvodem, proč jsem si pro tento projekt vybral programovací jazyk Python.

3.7. Djoser

Djoser je knihovna, která rozšiřuje funkčnost Django REST frameworku o autentifikační systém, který funguje na bázi REST. Jeho hlavní výhodou je, že podporuje, na rozdíl od autentifikace, která je součástí Django REST frameworku, autentifikaci pomocí takzvaného tokenu. Token je textový řetězec vygenerovaný po přihlášení uživatele pro bezpečnou komunikaci mezi uživatelem a API.

3.8. Heroku

Heroku je použito jako hosting pro API. Jeho výhodou je, že je jeho základní verze zdarma. Má oproti jiným hostingům jednoduché ovládání a nabízí veliké množství dalších užitečných funkcí.

3.9. PostgreSQL

PostgreSQL je open source objektově-relační databázový systém. Je dostupný v Heroku námísto SQLite, který slouží pro testování a vývoj.

3.10. Postman

Postman je API klient a je v něm možné vytvořit jakýkoliv typ HTTP požadavku a otestovat tak funkci API.

3.11. Zabezpečení

Autentifikace je důležitou částí projektu. Framework Django obsahuje velice dobré zabezpečení, a proto jsem se ho rozhodl použít.

3.11.1.Django autentifikace

Framework Django obsahuje celkem rozsáhlý autentifikační systém, který slouží převážně pro administraci webových aplikací a k přidávání obsahu, například fotek, na webové stránky. Tento systém je pro potřeby API příliš základní, je ale základem pro složitější typy autentifikace a Django REST framework a Djoser ho rozšiřují.

3.11.2.Django REST framework autentifikace

Django REST framework obsahuje několik způsobů autentifikace. Je v něm také možné vytvořit si svou vlastní autentifikaci podle vlastních požadavků. Tokenová autentifikace je jedním z předpřipravených typů autentifikace. Je však relativně jednoduchá a potýká se s několika problémy.

3.11.3.Djoser

Djoser je knihovna, která vychází z Django REST framework. Tato knihovna obsahuje set základních funkcí pro registraci, přihlášení, odhlášení, aktivaci účtu a resetování hesla. Dále již obsahuje funkční tokenovou autentifikaci, která funguje i s na míru vytvořeným uživatelem.

3.12. Popis funkce API

3.12.1.Komunikace s mobilní aplikací

Mobilní aplikace komunikuje s API přes HTTP protokol pomocí objektové notace JSON. API je dostupné na webu Heroku. Pokud chce uživatel (mobilní aplikace) vytvořit uživatelský profil pošle POST request na určenou URL s daty ve formátu JSON. API vytvoří pomocí knihovny Djoser uživatelský profil a uloží ho do databáze. Obdobně fungují další funkce související s autentifikací. Dále mobilní aplikace posílá požadavky (GET request) pro ovládání a vypůjčení boxu. Pro tyto akce musí být uživatel přihlášen.

3.12.2.Komunikace se zámkem

Komunikace se zámkem je pouze jednostranná zámek se v určitém intervalu dotazuje na svůj stav (odemčeno nebo zamčeno) a podle toho se buď odemkne nebo zamkne.

3.13. Modely

Django framework se stará automaticky o ukládání modelů do databáze a o jejich získání. Je to jedna z velkých výhod tohoto frameworku a šetří velké množství času.

3.13.1.User

Model uživatele dědí z třídy AbstractUser, která je součástí autentifikace Django frameworku. Obsahuje proměnné jméno, e-mail a boxy. O jeho ukládání se stará třída UserManager, která dědí z třídy BaseUserManager, která je také součástí autentifikace Django frameworku.

3.13.2.Box

Model boxu je základní model Django aplikace. Obsahuje proměnné pro určení polohy, stav (odemčeno nebo zamčeno), jméno a aktualního vlastníka.

4. SMARTBOX (APLIKACE)

Aplikace SmartBox je interaktivním prvkem projektu, který zprostředkovává rozhraní mezi uživatelem a zbytkem projektu. Program je napsán v programovacím jazyce Swift, který vyvíjí a spravuje společnost Apple. Aplikace je kompatibilní pouze na platformu iOS a to od verze 13 a výše.

4.1. Použité technologie

Celá aplikace je psána v programovacím jazyce Swift. Dále byly využity jak interní knihovny Swiftu, jako je základní framework Foundation, zajišťující základní fungování jazyka a datových typů. Tak UIKit pro práci s uživatelským prostředím a MapKit pro implementaci map společnosti Apple. Mimo interní knihovny Swiftu byl v aplikaci použit framework FloatingPanel [1] a JGProgressHUD pro vyobrazení indikátoru probíhajících procesů na pozadí aplikace [6].

4.1.1. FloatingPanel

FloatingPanel je externí framework, do aplikace implementovaný pomocí systému Podfiles, který se běžně v tomto jazyce využívá. Knihovna umožňuje přidat do programu základní grafický prvek, vyskytující se v mnoha aplikacích zaměřených na práci s mapami a to panel umístěný ve spod obrazovky. Díky tomuto prvku je možné jednoduše a elegantně zkombinovat v user interface (dále jen UI) mapu a ostatní důležité prvky, jako jsou tlačítka, texty a jiné. [1]

4.2. Uživatelské prostředí

Uživatelské prostředí pro SmartBox bylo navrhováno tak, aby bylo co nejjednodušší a zároveň se s ním uživatel nemusel příliš dlouho seznamovat. Po přihlášení v prvním okně je uživatel přesměrován na hlavní obrazovku. Zde nalezne mapu, na které je po odsouhlasení vyobrazena jeho současná poloha se všemi SmartBoxy v jeho okolí. Ve spodní části je umístěn interaktivní panel, jehož vzhled a funkcionalita byla přizpůsobena systémovým aplikacím iOS. Díky tomu je jednodušší se s prostředím seznámit a začít ho používat. Funkcionalita hlavního okna je rozvržena celkem do tří částí. A to do seznamu okolních SmartBoxů, detail úschovny a detail uživatele. Všechny tyto tři části byly implementovány díky knihovně FloatingPanel.

4.3. Funkcionalita

Kód je členěn celkem do čtyř základních okruhů. Na kterých závisí funkčnost programu.

Prvním jsou modely. Aplikace čerpá ze dvou modelů. Prvním je samotná struktura boxu, ve které jsou definovány základní informace o SmartBoxech, aby bylo pro uživatele možné s nimi pracovat a druhým samotný profil uživatele. V programu je model uživatele řazen tak, že

po získání dat ze serveru se databáze vyplní zároveň i daty podle modelu boxu. Dále se pracuje jen s daty uživatele a jedním polem pro poblíž nacházející se boxy.

Druhým okruhem je UserController. Ten řídí celý chod programu. Jako první řídí připojení a získávání dat ze serveru, tyto data zpracovává a ukládá do tří proměnných a to uživatele, pole s úschovnami poblíž a boxy uživatele. Jelikož aplikace využívá několik souběžně běžících view controllerů, které řídí UI, je v UserControlleru definován statický atribut shared. Ten odzakuje na samotný UserController, díky čemuž jsou všechny jeho funkce a data v něm uložená na jednom místě a přístupná odkudkoliv.

```
func defHidePanel(_ panel: FloatingPanelController) {
    hidePanel(panel)
    panelQuery.removeLast()
    if let lPanel = panelQuery.last { showPanel(lPanel) }
}

func defShowPanel(_ panel: FloatingPanelController) {
    showPanel(panel)
    if let lPanel = panelQuery.last { hidePanel(lPanel) }
    panelQuery.append(panel)
}
```

Obr. 6: Příklad kódu pro výměnu panelů

View controllery jsou dalším základním okruhem programu. Každé okno UI nebo view je řízeno samostatně pomocí view controlleru. Na hlavní stránce aplikace, kde je vyobrazená mapa, běží zároveň několik views a view controllerů. Z přihlašovacího okna je uživatel přesměrován na první view controller MapViewController, ten zajistí oprávnění k polohovým službám uživatele a vyobrazí mapu se vsemi SmartBoxy v okolí. Dále MapViewController vytvoří a spustí na pozadí všechny controllery a views, které jsou zprostředkovány pomocí FloatingPanelu. Tyto controllery jsou spuštěny celkem tři. Vyobrazen je však pouze BoxListViewController, jehož funkce je zobrazit list všech úschoven, které zajistí UserController od serveru. Dále po zvolení určitého SmartBoxu vyšle ListViewController pomocí NotificationCenter upozornění hlavnímu MapViewController, který může jako jediný řídit zobrazování a kooperaci ostatních viewcontrollerů. Ten změní zobrazení view na detail boxu. DetailViewController řídí toto okno. Jako první zkонтroluje index zvoleného boxu ve statické proměnné DetailViewController, která je nastavena před přesunem mezi controllery. Všechna data jsou dále podle tohoto indexu získána z UserControlleru a vyobrazena. Posledním viewcontrollerem je UserViewController. Ten není tak propojený s dvěma předchozími viewcontrollery. Pouze čte data z UserControlleru a případně s ním komunikuje v případě, že je data nutné aktualizovat.

Posledním důležitým okruhem programu jsou soubory definující samotné UI. Aplikace je postavena na klasickém storyboard. To znamená, že základní navigace programu je zaznamenána v jednom souboru, ve kterém jsou vyznačeny i vstupní body. Při navigaci je používán sou-

bor Main.storyboard (dále jen Main). Ten má v hlavním okně pouze nastavené view s mapou. Externí FloatingPanely jsou do UI přidány programově a to v MapViewController. Jejich obsah je však vytvořen ve čtyřech XIB souborech. Ty se chovají stejně jako Main, musejí být však zavolány jednotlivými viewcontrollery a připojeny na FloatingPanely jako contentview.

```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    BoxListViewController.boxChosen = indexPath.row
    tableView.deselectRow(at: indexPath, animated: false)
    NotificationCenter.default.post(name: BoxListViewController.tableCellAction, object: nil)
}
```

Obr. 7: Příklad kódu zavolání BoxDetailViewController

4.4. Serverová komunikace

SmartBox (aplikace) komunikuje se serverem pomocí url požadavků. A to tak, že všechny požadavky jdou přes UserController, který je zpřístupněn v celém programu přes statickou proměnnou shared.

Všechna data, která není nutno chránit jsou do url přidána jako url path. Citlivá data jako přihlašovací údaje nebo autentifikační token, který následně umožňuje serveru identifikaci uživatele jsou přidána do záhlaví zasílaného souboru. Jsou-li očekávána příchozí data od serveru, tak jsou formátována pomocí struktury json a dekódována podle protokolu Codable, kterému odpovídají modely, do příslušných proměnných v UserControlleru.

Při navazování spojení se serverem přes příslušné funkce jsou data získávána v UserControlleru, avšak ten je vrací volající funkci, která je dále ukládá do proměnných v UserControlleru. Tomu je tak, protože při navázání spojení je přesunut na jiné vlákno, které se stará pouze o navázání spojení se serverem. Po získání příslušných dat je opět přesunut na hlavní vlákno a s daty dále pracuje.

```
func loginUser(email: String, password: String, completion: @escaping (Bool) -> Void) {
    let orderURL = baseURL.appendingPathComponent("auth/token/login/") //vytvorení url

    //konfigurace urlRequest
    var request = URLRequest(url: orderURL)
    request.httpMethod = "POST"
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")

    let data: [String: String] = ["email": email,
                                  "password": password]

    let jsonEncoder = JSONEncoder()
    let jsonData = try? jsonEncoder.encode(data)
    request.httpBody = jsonData

    let task = URLSession.shared.dataTask(with: request) { (data, response, error) in
        let jsonDecoder = JSONDecoder()
        if let data = data, let token = try? jsonDecoder.decode(Token.self, from: data) {
            print("login checkpoint")
            UserController.token = token.auth_token
            self.loadAll {
                completion(true)
            }
        } else {
            completion(false)
        }
    }
    task.resume()
}
```

Obr. 8: Příklad serverové komunikace

3. ZÁVĚR

Cílem práce bylo vytvořit funkční model, který bude odpovídat konceptu chytré úschovny, ovládané skrze rozhraní mobilní aplikace. Naším úkolem bylo zpracovat jednotlivé části projektu, což se nám povedlo dle očekávání.

Zámek se však z bezpečnostního a funkčního hlediska nepodařilo vytisknout na 3D tiskárně.

4. POUŽITÁ LITERATURA

- [1] Copyright (c) 2018 Shin Yamamoto, <https://github.com/SCENEE/FloatingPanel>
- [2] Quickstart - Django REST framework. Home - Django REST framework [online]. Dostupné z: <https://www.djangoproject-rest-framework.org/tutorial/quickstart/>
- [3] Authentication - Django REST framework. Home - Django REST framework [online]. Dostupné z: <https://www.djangoproject-rest-framework.org/api-guide/authentication/>
- [4] How to Implement Token Authentication using Django REST Framework. Simple is Better Than Complex [online]. Copyright © 2015 [cit. 03.03.2020]. Dostupné z: <https://simpleisbetterthancomplex.com/tutorial/2018/11/22/how-to-implement-token-authentication-using-django-rest-framework.html>
- [5] User registration and authorization on a django API with djoser and JSON web tokens. - DEV Community. DEV Community [online]. Dostupné z: <https://dev.to/lewiskori/user-registration-and-authorization-on-a-django-api-with-djoser-and-json-web-tokens-4kc7>
- [6] Copyright (c) 2014-2018 Jonas Gessner, <https://github.com/JonasGessner/JGProgressHUD>

5. SEZNAM OBRÁZKŮ A TABULEK

Obrázek 1: Obvod

Obrázek 2: Schéma

Obrázek 3: Plošný spoj

Obrázek 4: Deska PBC

Obrázek 5: Deska PBC – schéma

Obrázek 6: Příklad kódu pro výměnu panelů

Obrázek 7: Příklad kódu zavolání BoxDetailViewController

Obrázek 8: Příklad serverové komunikace

