Gymnázium, Praha 6, Arabská 14

Programování

Maturitní práce

Meet ME

Duben 2021 Vladimír Ekart 4.E

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. dubna 2021

Anotace

Cílem této maturitní práce je vytvoření služby, umožňující dvěma a více registrovaným uživatelům jednoduše nalézt společný čas v jejich kalendářích, za pomoci mobilní aplikace. V této dokumentaci naleznete způsob řešení tohoto problému, popis práce s databází, serverové komunikace, práci s kalendářem, kontakty a lokální pamětí. Dále je zde popsána struktura celého projektu a jeho dílčích částí, ze kterých se skládá.

Abstract

The goal of this graduation work is to create a service that allows two or more registered users to easily find a common time in their calendars, using a mobile application. In this documentation you will find how was this problem solved, an example of working with the database, server communication, working with the calendar, contacts and local memory. Furthermore, the structure of the whole project and its partial parts of which it consists are described.

Obsah

1. Úvod

1.1. Zadání

Cílem mé maturitní práce je vytvořit mobilní aplikaci pro iOS využívající kalendář uživatele, který si s aplikací synchronizuje. Služba umožní dvěma nebo více lidem nalézt společný čas v jejich dni. Na začátku uživatel zvolí období, ve které se chce potkat, dále pošle pozvánku druhému. Ten když ji přijme, tak se lokálně na obou zařízeních zpracuje dané období a na serveru se anonymně najdou úseky, které oběma vyhovují. Tato výsledná data se zobrazí každému uživateli v jeho rozvrhu a ten si událost bude moct přidat do svého kalendáře, který běžně využívá. Aplikace samotná bude napsána v programovacím jazyce Swift. Serverová část bude dále běžet na Javascript s databází na Mongo DB.

1.2. Struktura projektu

Projekt je tvořen dvěma částmi serverem a aplikací. Serverová část umožňuje samotnou komunikaci mezi uživateli. Dále jsou zde uchovávána data za pomoci databáze, tvořené dvěma databázovými modely. V této části projektu také dochází ke zpracování kalendářových dat a vytváření samotných událostí. Aplikace slouží jako prostředek pro komunikaci s uživatelem a pro získávání dat, se kterými se v projektu dále pracuje. Kalendářová data jsou zde předzpracována a posílána na serverovou část.

2. Server

2.1. Použité technologie

Serverová část je napsána v programovacím jazyce JavaScript. Samotný kód není napsán v čisté verzi tohoto jazyka, ale byl použit softwarový systém Node.js [12] a framework Express [1], který tvoří základní strukturu a asynchronní přenos dat mezi serverem i uživatelem a přesměrování požadavků přicházejících na API (Application Programming Interface). Pro databázi bylo použito Mongo DB [2], se kterým v programu přímo pracuje knihovna Mongoose [3]. Autentifikace je řešena za pomoci passport [4] a passport-jwt [5] (JSON Web Token).

2.2. Struktura programu

Program serverové části se rozděluje do čtyř základních okruhů, které dohromady zajišťují chod aplikace. Požadavky na API jsou nejdříve zachyceny v routeru, který je dále přesměrovává do kontrolerů. Data jsou zde přeposlána na kontrolu do validátoru a zpracována v kontroleru. Ten v

případě, že jsou v pořádku, pracuje se samotnou databází. Ta využívá posledního ze základních okruhů modelů. V modelech je definována struktura databáze.

2.2.1.Router

O přesměrovávání požadavků na API se stará knihovna Express. Ta je primárně definována v souboru index.js, který slouží jako vstupní bod aplikace. Zde se požadavek směruje do jedné ze tří základních cest definované path v url do souborů routeru v routes. Část routeru vyžaduje před tímto přesměrováním autentifikaci uživatele, za pomoci tokenu v hlavičce požadavku pod klíčem Authorization. Autentifikace probíhá v konfiguračním souboru pro passport. Samotný router po identifikaci uživatele požadavek přesměrovává do daných kontrolerů.

```
User.findOne({ phone: body.phone }).then(user => {
    if (user) {
        return res.status(400).json({ phone: "phone already exists" })
    } else {
        const newUser = new User(body)

        if (!newUser) {
            return res.status(400).json({ error: "New user not created" })
        }

        bcrypt.genSalt(10, (_, salt) => {
            bcrypt.hash(newUser.password, salt, (error, hash) => {
                if (error) throw error
                newUser.password = hash
                newUser
                    .save()
                    .then(usr => res.status(201).json(usr))
                    .catch(err => res.status(400).json(err))
            })
        })
    }
})
```

Obr. 1: Vytvoření uživatele

2.2.2.Kontrolery

Logické jádro serveru je definováno v konrolerech. Cílem tohoto okruhu je zpracování příchozích dat a následná práce s databází, kde jsou data uložena. Konrolery jsou v projektu dva. Jeden slouží pro práci s modelem uživatele a druhý pro model schůzky. Všechny požadavky nesoucí informa-

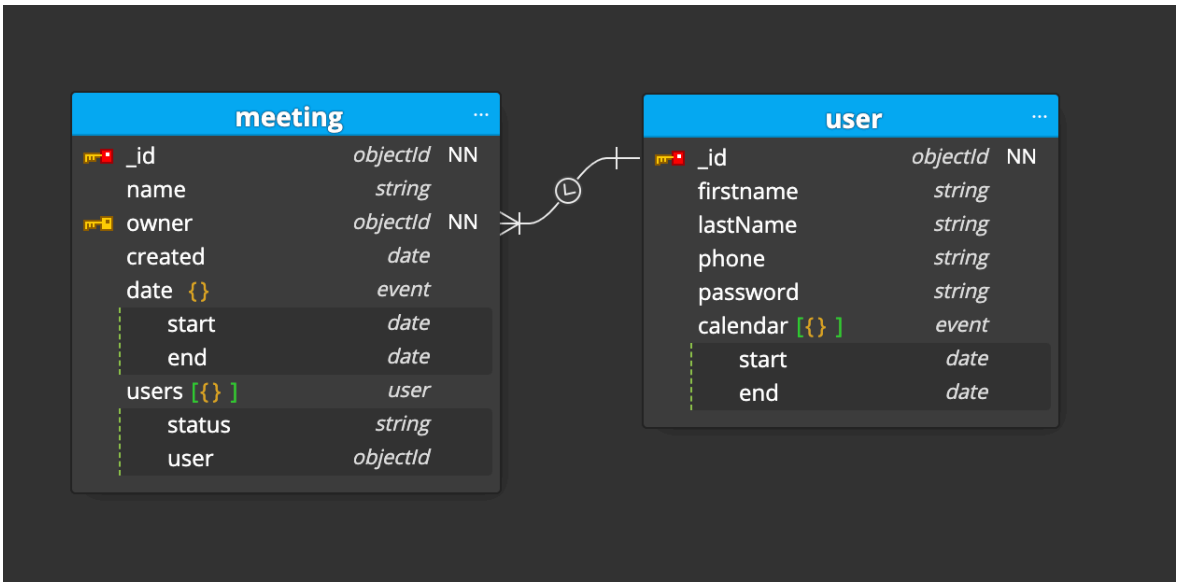
ce ve svém těle jsou před prací s databází nejdříve odeslány na kontrolu do validátorů. Když jsou v pořádku, je dále prováděna samotná operace s nimi.

Nejdůležitějšími operacemi v kontrolerech je správa uživatele a následná práce s kalendářem. Při registraci je požadavek zkontrolován a data jsou hledána v databázi pro nalezení případných duplicit. Model uživatele je zpracován a za pomoci knihovny bcrypt [6] je zašifrováno jeho přístupové heslo. V případě přihlášení se porovnávají přihlašovací údaje a kryptovaný hash. Tato data jsou přeměněna na payload, ten je použit pro vygenerování tokenu uživatele v jwt a následně odeslán zpět v odpovědi.

Kalendářová data jsou v databázi uchovávána trvale a neustále se udržují aktuální. V případě požadavku o volný kalendář více uživatelů jsou jednotlivé rozvrhy vytaženy z databáze, zpracovány s ohledem na kritéria definovaná v požadavku a odeslána zpět uživateli. Kalendář je vždy zpracováván celý, jelikož se uchovávají pouze volné úseky, nedojde tak ke zpracovávání neaktuálních dat

2.2.3.Validátory

Validátory jsou volány z kontrolerů. Jejich účelem je zkontrolovat správný formát a kompletnost přijatých dat. Pro každý požadavek, který vyžaduje validaci je implementován samostatný validátor. Ve validátorech jsou použity dvě základní knihovny is-empty [7], ta zjistí zda-li jsou všechna kontrolovaná pole kompletní před vstupem do metod knihovny validator [8], který následně validuje jednotlivá kritéria, kladená na vstup. V případě nalezení nesprávností pošle validátor zprávu v poli errors, proměnná isValid je poté zjišťována, jako velikost pole errors. Jesliže je daný validátor určený pro kontrolu vstupu k úpravě dat v databázi, je do výstupu validátoru přidáno navíc pole changes, zde jsou zaznamenány změny, o které se uživatel pokouší.



Obr. 2: Databázový model

2.2.4.Modely

Databáze je tvořena dvěma modely userSchema a meetingSchema. O integraci modelů se zasluhuje knihovna mongoose, která je navržena pro práci s Mongo DB za pomoci Node.js. Model uživatele obsahuje osobní údaje uživatele, telefonní číslo, které bylo použito jako identifikační údaj a musí být pro každého uživatele unikátní. Dále je v tomto modelu pole pro kalendář, kde jsou data s volnými místy v uživateli kalendáři.

Model schůzky nese základní informace o ní, majitele, datum vytvoření, pole s uživateli a statusy, zda-li danou schůzku přijali či nikoliv. K těmto schůzkám mají přístup všichni uživatelé na listu jednotlivé schůzky včetně majitele, který je také do tohoto listu přidán.

2.3.Praktická implementace

Pro komunikaci aplikace se serverem byla serverová část nahrána a běží na dvou službách Heroku a Mongo DB Atlas. Tyto dvě služby spolu komunikují a API je tak přístupné na veřejné URL.

2.3.1.Heroku

Pro implementaci programu na Heroku v něm bylo provedeno několik úprav. První do kódu byla přidána knihovna dotenv [13]. Tato knihovna umožňuje dynamicky nastavovat proměnné, jako je například cesta k databázi, port na kterém API poběží nebo přihlašovací údaje. Pro samotné Heroku je přidán soubor Procfile, ten obsahuje informace ke spuštění aplikace pro samotnou službu. Na službě Heroku jsou již jen definované hodnoty pro env proměnnou MONGODB_URI, díky čemuž je schopná se spojit s databází.

2.3.2.Mongo DB Atlas

Databáze samotná je inicializována a nastavena na Mongo DB Atlas. Na této službě je vytvořený cluster s přihlašovacími údaji zakomponovanými do URI, pomocí které se ke clusteru připojuje služba Heroku.

3. Aplikace

Aplikace je interaktivním prvkem projektu a zprostředkovává rozhraní mezi uživatelem a zbytkem projektu. Aplikace je instalovatelná na iOS zařízení s verzí systému iOS 14 a vyšší.

3.1. Použité technologie

Aplikace byla napsána v programovacím jazyce Swift, spravovaným společností Apple a určeným k vývoji na jejich zařízení. Jsou zde primárně využívány interní knihovny samotného jazyka. Dále do projektu byly importovány dvě externí knihovny JGProgressHUD [10] pro zobrazení indikátoru probíhajících procesů na pozadí a SPPermissions [11], díky které je možné graficky vyžádat od uživatele potřebná oprávnění k fungování aplikace.

3.2. Uživatelské prostředí

Cílem vytvořeného UI (uživatelské prostředí) bylo zachování jednoduchosti a intuitivnosti aplikace. UI je tvořeno třemi základními částmi: přihlášení, správa a vytváření událostí. Po přihlášení je uživatel přesměrován na hlavní stranu. Zde se nachází roztříděné listy se schůzkami podle statusu, zaznamenaného u přihlášeného uživatele. Ten má dále možnost otevření detailu a následné úpravy jednotlivých událostí, nebo vytvoření nové. Zde si zvolí název, délku trvání a uživatele, které může vybrat ze svého listu kontaktů na zařízení nebo manuálně zadat hledané číslo. Z důvodu diskrétnosti se list se zadanými účastníky ověří až po pokusu o vytvoření schůzky, bez detailů o nenalezených kontaktech. Po zadání těchto údajů je uživatel přesměrován na list s prázdnými a plnými úseky. Pro toto zobrazení bylo použité speciálně upravené TableView s možností expandovat jednotlivé buňky. Uživatel má díky tomu možnost se rychle a jednoduše orientovat v rozvrhu. Po zvolení plné buňky je zobrazen list s událostmi v tomto čase z kalendáře extrahovaného ze zařízení nebo hláška upozorňující na blokaci jiným uživatelem. Při zvolení volného času si uživatel nastaví časový interval, ve kterém chce aby se schůzka konala. Poté se vytvoří zadaná schůzka, ke které má dále přístup z hlavní strany.

3.3. Struktura programu

Samotný program je členěn do čtyř základních okruhů. Osoba operující s aplikací interaguje s Main.storyboard, do kterého jsou vkládány navíc soubory .xib a definují samotné uživatelské prostředí. To dále řídí sada ViewControllerů, které má každé okno UI unikátní. ViewControllery se na to připojují k UserControlleru, kde jsou uchovávána samotná data a je zde definována komunikace se serverem a pamětí zařízení. Poslední okruh tvoří modely, ty reprezentují strukturu přijímaných a odesílaných dat ze serveru.

3.3.1.Modely

Frontendová část má definované dva soubory s modely. Ty jsou děleny na uživatele a schůzky. V těchto souborech se nachází více než jedna struktura a to z toho důvodu aby bylo možné se strukturami dále pracovat při síťové komunikaci, musí všechny být definovány jednoduše a odpovídat protokolu Codable, který je stěžejní pro následnou výměnu dat se serverem.

3.3.2.Controller

Controller v kódu definovaný jako UserController a reprezentuje v této aplikaci jádro programu. Pro jeho vytvoření zde byl použit návrhový vzor Singleton. Je zde využito jeho výhody jedné globální, statické instance v celém projektu. Jako přístupový bod zde slouží proměnná shared a je do ní přistupováno v celém projektu. Data jsou uložena do proměnné user, která odpovídá modelu definovanému v modelu pro uživatele a v jedné své instanci obsahuje pole se schůzkami odpovídajícími modelu definovanému v meeting modelu. Controller dále řídí veškerou serverovou komunikaci, práci s lokální pamětí, do které ukládá aktuální přihlašovací token a získávání kalendářových dat ze zařízení.

3.3.3.User Interface

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 soubor Main.storyboard (dále jen Main). UI na několika místech využívá TableView, díky tomu je možné zobrazovat data v listech. Tato TableView musela být přizpůsobena obohacením o složitější funkcionalitu, jako je získávání dat od uživatele nebo zakomponování dalších TableView do jednotlivých buněk v případě výběru konkrétního volného úseku. Buňky jsou přizpůsobeny v .xib souborech.

3.3.4.View Controllery

Každé okno UI nebo view je řízeno samostatným ViewControllerem, který se stará a sleduje interakci s uživatelem skrze uživatelské prostředí. ViewControllery se v aplikaci rozdělují na dva typy. Prvním typem jsou ty, které řídí okna součástí Main. Zde jsou získávána data skrze UserController, řízena navigace mezi okny a interakce s uživatelem. Tyto soubory spolu komunikují skrze Main příslušnými funkcemi, které jsou součástí těchto tříd. Druhý typ ViewControllerů se stará o chod menších komponent, jako jsou buňky TableView. Tyto soubory jsou navíc párovány s .xib soubory. Jelikož jsou buňky řízené těmito controllery, tak mezi sebou přímo nikdy nekomu-

nikují, ale spojují se s nadřazenými Main ViewControllery, ty řeší následnou komunikaci. Tyto dva typy spolu nemohou komunikovat přímo, je zde proto využito NotificationCenter, které je schopné přenášet data a spouštět příslušné funkce mezi třídami v celé aplikaci.

3.4.Serverová komunikace

Meet ME komunikuje se serverem pomocí url požadavků. A to tak, že všechny požadavky jdou přes UserContoller, který je zpřístupněn v celém programu přes statickou proměnnou shared. Pro komunikaci je v aplikaci implementována integrovaná knihovna Swiftu Foundation, která mimo jiné dokáže formátovat posílaná data do požadovaného JSON formátu. Samotná komunikace probíhá na vedlejších vláknech, ze kterých jsou data ukládána v proměnných UserControlleru, ze kterých jsou funkce volány. Po přesunutí zpět na hlavní vlákno je volající funkce upozorněna pomocí argumentu completion.

```
private func getMeetings(completion: @escaping ([Meeting]?) -> Void) {
    let userURL = baseURL.appendingPathComponent("sec/meeting/")

    var request = URLRequest(url: userURL)
    request.httpMethod = "GET"
    request.setValue(self.token ?? "", forHTTPHeaderField: "Authorization")

    let task = URLSession.shared.dataTask(with: request) { (data, response, error) in
        let jsonDecoder = JSONDecoder()
        jsonDecoder.dateDecodingStrategy = .formatted(DateFormatter.iso8601Full)
        if let data = data, let meetings = try? jsonDecoder.decode([Meeting].self, from: data) {
            print("meetings: OK")
            completion(meetings)
        } else {
            print("meetings: FAILED")
            completion(nil)
        }
    }
    task.resume()
}
```

Obr. 3: Serverová komunikace

3.5.Práce s kalendářem

Kalendář je v aplikaci získáván na několika místech. Samotný kód pro získání dat z kalendáře je definovaný v UserControlleru volaný z funkce pro příslušnou serverovou komunikaci. Při získávání dat se kalendář získává v rozmezí od současnosti na měsíc dopředu, v ideálním stavu jsou tak na serveru zaznamenána data v měsíčním období a uživatel v něm může vytvářet schůzky. Před odesláním jsou kalendářová data zpracována a jsou v nich nalezeny volné úseky modelem CalendarModel. Tento zpracovaný kalendář se dále odesílá na server, kde se data uchovávají. Tento proces probíhá ze dvou míst. Pokaždé, když se uživatel přihlásí do aplikace, díky čemuž je kalendář při vytváření aktuální a poté když je Meet ME puštěno

systémem na pozadí, odkud se tento proces provede a aplikace je znovu uspána.

4. Zpracování kalendářových dat

Kalendářová data jsou v projektu zpracovávána na několika místech jak na serverové části, tak v samotné aplikaci. Všechna tato zpracování pracují na podobném principu, algoritmus je všude mírně modifikován, tak aby data odpovídala požadovaným kritériím.

```
var sorted: [StartEnd] = []
var all: [(date: Date, start: Bool)] = []
for event in calendar {
    all.append((date: event.start, start: true))
    all.append((date: event.end, start: false))
}
all.append((date: Date(), start: false))
all.append((date: Calendar.current.date(byAdding: .month, value: 1, to: Date())!, start: true))
var active = 1
var start: Date = Date()
for item in all.sorted(by: { (a, b) -> Bool in
    return a.date < b.date
}) {
    if !item.start {
        active -= 1
        if active == 0 {
            start = item.date
        }
    } else {
        if active == 0 {
            sorted.append(StartEnd(start: start, end: item.date))
        }
        active += 1
    }
}
self.calendar = sorted
```

Obr. 4: Zpracování kalendářových dat

Na začátku se vezmou události ze všech porovnávaných kalendářů a každá událost se rozdělí na dva objekty, jeden nese informaci o začátku a druhý o konci dané události, s přesným datem a informací o začátku nebo konci. Tímto způsobem se naplní pole A, to má dvojnásobnou velikost oproti počtu událostí. Pole A se setřídí pomocí knihovních funkcí vzestupně podle data. Následně je vytvořeno pole B a počítadlo otevřených událostí, pole A se začne postupně procházet. Při načtení jednotlivých objektů z pole A se upraví počítadlo otevřených událostí podle toho, jestli jde o objekt nesoucí začátek nebo konec události. V případě vyhledávání společného volného času na serveru se toto počítadlo porovnává s počtem uživatelů zahrnutých v události, když je počet uživatelů roven počtu otevřených událostí, tak se zaznamená začátek nalezeného úseku, který končí dalším objektem v poli A, který označuje konec jedné z otevřených událostí a volný úsek pro všechny zúčastněné uživatele tak končí. Tento nalezený úsek končí a je zazna-

menán jako nová událost do finálního pole B. V případě vyhledávání volných úseků v kalendáři uživatele na zařízení se hledají naopak úseky, kde je počítadlo otevřených událostí rovno nule.

4.1.Modifikace

V programu se objevují dvě větší modifikace tohoto algoritmu. Na serveru je jako další kritérium pro vybrání úseku přidán minimální časový interval. Ten je porovnáván s časovými rozdíly identifikovaných úseků. V samotné aplikaci při získávání a zpracování dat z kalendáře je tento algoritmus navíc obohacen o přidání umělé události, která končí na začátku prohledávaného období a začíná na konci procházeného úseku. Počet otevřených událostí, tak začíná na hodnotě 1 a ne 0. Díky tomu dojde k vytvoření hranic, kde se budou data zpracovávat.

5. Závěr

Maturitní práci se podařilo vytvořit dle očekávání. Projekt řeší jeho hlavní problém, kterým je zjednodušení procesu vyhledávání společného volného času mezi dvěma a více lidmi. Úspěšně se mi podařilo v projektu práce s databází, serverová komunikace, získávání a zpracování kalendářových dat, práce s lokální pamětí a práce s kontakty.

V průběhu vývoje byl nalezen nedostatek v zadání práce a to navržený postup a systém řešení při domlouvání schůzek mezi více uživateli. Původní návrh počítal více s diskrétností vůči uživateli, ale zároveň značně zpomaloval tento proces tak, že aplikace již neplnila svůj původní účel. Změnou bylo přesunutí kalendářových dat uživatele na server, což umožnilo urychlit proces, jelikož má uživatel možnost rovnou pracovat a vybírat v aktuálních datech a nemusí hádat a následně v několika krocích domlouvat volné úseky.

V budoucnu plánuji tuto aplikaci obohatit o list přátel, tak se služba stane opět diskrétnější, ale bude zároveň využije elegantního a jednoduchého řešení, které bylo nakonec použito.

Seznam obrázků

Seznam literatury

[1] Copyright (c) 2009-2014 TJ Holowaychuk <tj@vision-media.ca>
Copyright (c) 2013-2014 Roman Shtylman <shtylman+expressjs@gmail.com>
Copyright (c) 2014-2015 Douglas Christopher Wilson <doug@somethingdoug.com>
https://github.com/expressjs/express

[2] Copyright © 2018 MongoDB, Inc,
https://www.mongodb.com

[3] Copyright (c) 2004-2013 Sergey Lyubka
Copyright (c) 2013-2020 Cesanta Software Limited,
https://github.com/cesanta/mongoose

[4] Copyright (c) 2011 Jared Hanson,
https://github.com/TelegramPassport/passport-js

[5] Copyright (c) 2014 themikenicholson,
https://github.com/mikenicholson/passport-jwt

[6] Copyright (c) 2012 Nevins Bartolomeo <nevins.bartolomeo@gmail.com>
Copyright (c) 2012 Shane Girish <shaneGirish@gmail.com>
Copyright (c) 2014 Daniel Wirtz <dcode@dcode.io>
https://github.com/dcodeIO/bcrypt.js

[7] Ianstormtaylor. (n.d.). Ianstormtaylor/is-empty. Retrieved March 31, 2021, https://github.com/ianstormtaylor/is-empty

[9] Copyright (c) 2018 Chris O'Hara <cohara87@gmail.com>,
https://github.com/validatorjs/validator.js

[10] Copyright (c) 2014-2018 Jonas Gessner,
https://github.com/JonasGessner/JGProgress- HUD

[11] Copyright (c) 2017 Ivan Vorobei,
https://github.com/sunxcode/SPPermission

[12] Nodejs. (n.d.). Nodejs/node. Retrieved March 31, 2021,
https://github.com/nodejs/node

[13] Copyright (c) 2015, Scott Motte, https://github.com/motdotla/dotenv