GYMNASIUM JANA KEPLERA

Parléřova 2/118, 169 00 Praha 6

Online burza učebnic

Maturitní práce

Autor: Fedor Stryapunin

Třída: 4.C

Školní rok: 2021/2022

Předmět: Informatika

Vedoucí práce: Ing. Šimon Schierreich

Praha, 25.3.2022

GYMNASIUM JANA KEPLERA
Kabinet informatiky

ZADÁNÍ MATURITNÍ PRÁCE

Student:	**Fedor Stryapunin**
Třída:	**4.C**
Školní rok:	**2021/2022**
Platnost zadání:	**30. 9. 2022**
Vedoucí práce:	**Šimon Schierreich**
Název práce:	**Online burza učebnic**

Pokyny pro vypracování:

Cílem práce je vytvořit webovou aplikaci, která bude sloužit jako burza pro studenty GJK, kteří se tam budou moci přihlásit pomocí google účtu a poté přidávat nabídky učebnic a jiných věcí. Nabídky budou mít kategorii, cenu, popis a status(aktivní, prodáno). Studenti své nabídky budou moci spravovat. Nabídky bude možné si zobrazit dle kategorie, ceny a data přidání.

Doporučená literatura:

[1] MARTIN, Robert C. *Design Principles and Design Patterns*. **www.objectmen- tor.com, 2000. Dostupné z: https://fi.ort.edu.uy/innovaportal/file/2032/1/design_principles.pdf.**

[2] FOWLER, Martin. *Patterns of Enterprise Application Architecture*. **Boston, Massachusetts, USA: Addison-Wesley Professional, 2003. The Addison-Wesley Signature Series. ISBN 978-0-321-12742-6.**

[3] EVANS, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. **Boston, Massachusetts, USA: Addison-Wesley Professional, 2003. ISBN 978-0-32-112521-7.**

[4] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. **2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.**

URL repozitáře:

https://github.com/saltyfedor/bookexchange

student *vedoucí práce*

V Praze dne 21. 12. 2021

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů. Nemám žádné námitky proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Praze dne 25. března 2022 Fedor Stryapunin

Poděkování

Chtěl bych poděkovat svému vedoucímu práce, Šimonu Schierreichovi, za jeho rady a návrhy v oblasti nutných funkcí pro opravdové nasazení aplikace.

Abstrakt

Práce se věnuje vytvoření webové aplikace, která slouží jako školní burza. Výsledkem práce je aplikace, vytvořená za pomocí SPA architektury, která umožňuje přidávat a spravovat nabídky a komunikaci mezi uživateli.

Klíčová slova

webová aplikace, burza, školství

Abstract

The subject of this work is the creation of a web application with the goal of replacing existing physical markeplace with a digital one. The result of the work is a creation of a web application based on SPA architecture that allows users to upload their listings, manage them and communicate with each other.

Keywords

web development, education, online marketplace

Obsah

1. Úvod

1.1 Problém

Studenti na gymnáziu Jana Keplera potřebujou k výuce učebnice. Ty si koupí a následně je chtejí prodát. V současnostní se to řeší tím způsobem, že se jednou ročně pořádá fyzická burza učebnic, kde se studenti po dobu několika vyučovacích hodin scházejí a prodávají mladším studentům své staré učebnice.

To má určité nedostatky. Fyzická burza je chaotická, nepřehledná a omezená pouze na učebnice. Také se koná pouze jednou ročně a pokud chce student koupit učebnice v jiném čase, musí je obtížné shánět po lidech ve škole nebo si koupit nové, drahé kopie. Pokud je naopak chce prodat, je to ještě náročnější. Osobně nabízet učebnice, mimo dobu konáni burzy, je pro většinu studentů nepříjemné. Může je prodat na intenetu, to ale neplatí pro materiály specifické pro naší školu.

Fyzická burza je též časově náročná, studenti tam dokážou strávit i několik hodin a musí se kvůli ní rušit výuka.

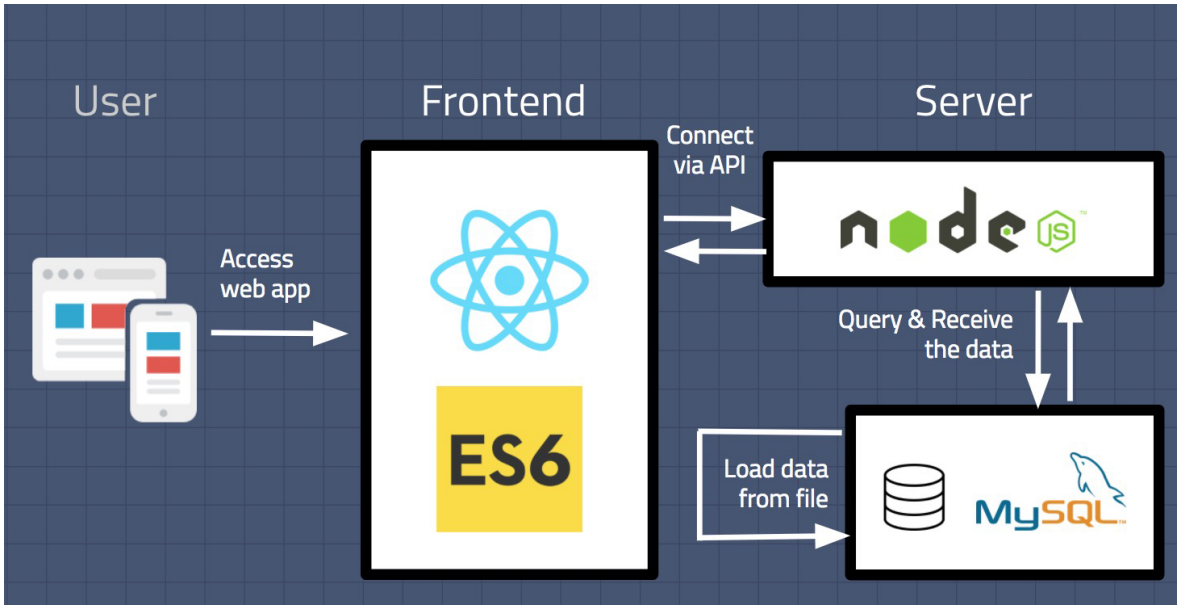
1.2 Řešení

Řešením je digitalizace studentské burzy. Ve své práci jsem se zaměřil na vytvoření webové aplikace, která tyto problémy řeší. Umožňuje prodej a nakupování učebnic v jakýkoliv čas, je přehledná, nabídky jdou snadno prohledat a řadit dle ceny. Studenti na ní můžou nabízet i jiné věci.

2. Implementace

2.1 Architektura

Pro účel této webové aplikace jsem zvolil architekturu SPA-API, nebo-li tzv. "jednostránku". Vypadá takto :



Obrázek 2.1: Schéma SPA architektury
https://www.byperth.com/2018/04/19/guide-building-data-science-web-application-with-react-nodejs-and-mysql/stock-analyzer-project-architecture-2/

Uživatel si na začítku jednou načte front-end aplikace, který komunikuje s API za pomocí HTTP requestů. Front-end běží v prohlížeči na HTML, CSS a Javascriptu, má vlastní routing a žádné další html stránky se už nenačítají.

API komunikuje s databázi a file-systémem a vrací pouze data v JSON formátu nebo obrázky, nikoliv html views.

2.1.1 Výhody SPA-API

Použitá architektura má hned několik výhod. Umožňuje větší interaktivitu, stránky se po prvotním načtení načítají rychleji a, díky tomu, že se nenačítaji další stránky, výsledná aplikace působí velmi hladce.

2.1.2 Nevýhody SPA-API

Mezi nevýhody SPA-API patří to, že potřebuje pro běh více systemových zdrojů a není nejlepší pro projekty, které spolíhají na SEO, což v tomto případě neplatí. Nepředpokládám, že by se nabídky z

aplikace někdy objevili na vrcholu výsledku v googlu.

2.2 Technologie

2.2.1 Front-end

Front-end aplikace je napsaný v javascritpu, což je dáno architekturou.

Jako základ front-endu jsem zvolil framework React. React je deklarativní framework, založený na rozdělení aplikace na UI komponenty, které spravují svůj vlastní stav, což umožňuje velmi rychlý a bezbolestní vývoj. Výsledek je pak přehledný a snadno se debuguje.

React-router

React-router se stará o routing, vrácení správných UI komponentů na základě url. Umožňuje snadnou tvorbu tzv. protected routes, které jsou uživateli nepřístupne pokud není přihlášen. Při pokusu o načtení takového url, přesměruje aplikace uživatele na domovskou stránku aplikace.

Redux

Redux je knihovna pro spravování stavu aplikace. Umožňuje snadnou aktualizaci UI, použití middlewaru, který se hodí pro práci s websocket zprávami. Načítání dat je též zařízeno přes Redux. V Reduxu ukládám pouze stav, který musí být přístupen ve více částech aplikace, např. informace o uživateli.

Styled-components

Styled-components, mi umožňuje definovat css styly přímo v souborech jednotlivých komponentů, využívát inheritance a tím výrazně zmenšit opakování zdrojového kódu.

Eliminuje taky problém nepřehledných, tisíce řádkových css souboru, kde není jasné jaká trída patří čemu a kolik z toho css kódu je vlastně už nevyužito.

Google - gsi

Klientská knihovna pro zjednodušení procesu komunikace s google api.

2.2.2 Back-end / API

Back-end v této aplikaci je napsaný v node.js. Node.js jsem vybral kvůli tomu, že zpracovává requesty asynchronně, což znamená, že i v případě, když aplikaci využívá hondě lidí najednou,

bude mít rozumný průměrný čas vyřešení requestů.

Samotný back-end je poměrně komplexní, a tak uvedu pouze ty zajimávé použité balíčky.

Express.js

Express je minimalistická knihovna pro vyřizováni http requestu. Má výborný deklarativní systém middlewaru a snadno poskytuje statické soubory.

DOTENV

Jediná volba pro přehledné spravování velkého množství proměn, které slouží pro prvotní nastavení aplikace.

WS

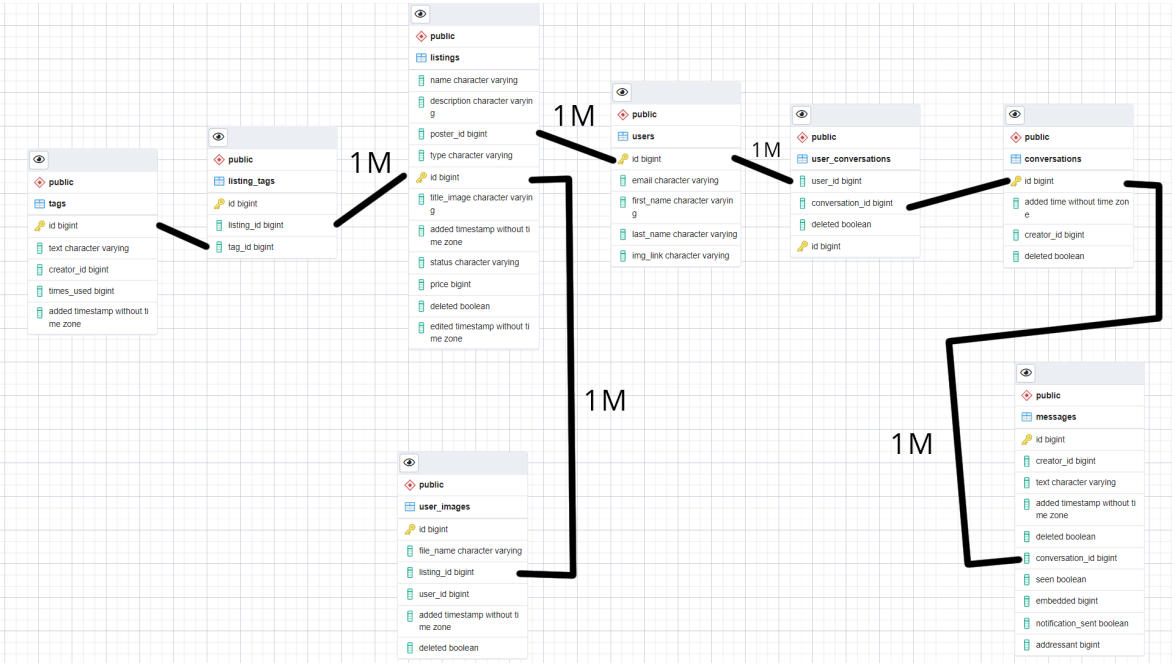
Minimalistická knihovna pro vytvoření wesocket serveru.

node-cron

Cron spravuje procesy. Používám ho na pravidelný spuštění tasku, který se stará o posílání emailových notifikací.

2.2.3 SQL databáze

Databáze je napsaná v PostgreSql. Její stavba je vidět na přiloženém schématu.



Databáze obsahuje tabulky users, listings, tags, conversations, user-images, messages a dvě many-to-many spojovácí tabulky listing-tags a user-conversations pro propojení uživatelů a konverzací a nabídek a jejích tagů.

2.3 Problémy

2.3.1 Přihlášení pomocí google účtu a perzistování

Klasická registrace, kdy uživatel vyplní formulář se svými údají, je v dnešní době přežitek. Trvá dlouho a opakovat tento proces pro každou službu, kterou uživatel chce použít je zkrátka otravné. Každý student na GJK má přidělený google účet. Proto jsem se rozhodl implentovat registraci a přihlášení pomocí google účtu.

To ovšem není tak jednoduché, jak se na první pohled může zdát.

Nejdříve bylo potřeba založit projekt v google konzoli a získat google client id a api key tokeny, aby google api moje requesty vůbec zvážilo. Pak jsem naimplementoval tlačítko přihlásit se s googlem, které uživateli nabídne výběr účtu, pomocí kterých se může přihlásit. Po zpracování requestu vrátí google api token, který lze vyměnit za uživatelská data. Ten se pošle na back-end, back-end se zeptá na data a tyto data se uloží do tabulky users.

Po úspěšné implementaci přihlášení nastává další problém. Žádný uživatel se nechce přihlašovat pokaždé, kdy navštíví nějakou stránku. Pro persistenci jsem se však rozhodl vynechat klasického cookie a session přístupu a využít refresh token - access token schéma.

Při prvotní autorizaci s googlem vráti back-end http read-only cookie s refresh tokenem, která je uložena v prohlížeči. Tuto cookie pošle aplikace společně s autorizačním requestem při každém načtení stránky (v SPA architektúře tím pádem jednou za sessi) na back-end a dostane zpátky access token, který pak posílá s každým citlivým requestem. Pokud' token chybí, nebo je neplatný,

back-end request zamítne.

2.3.2 Reagování na nabídky a instantní zprávy

V aplikaci jsem chtěl udělat systém reakcí na nabídky, který uživatelům umožní spolu komunikovat. Přirozeně mě napadl systém zpráv a reakcí. To v sobě skrývá ale poněkud' složitý problém.

Prvotně načíst zprávy při otevření záložky zprávy je jednoduché. Pokud aplikace ale už běží a v ten moment pošle uživateli někdo zpávu, jak se má pak aplikace dozvědět, že mu nějaká zpráva vlastně přišla?

Jako nejelegantnější řešení mi přišlo užití websocketu.

Websocket je oboustranný komunikační protokol pro komunikaci se serverem. Je to nejméně náročná možnost z hlediska systémových zdrojů na back-endu a z hlediska objemu posílaných dat.

K provedení této implementace jsem výužil knihovny ws, pomocí které vytvoří back-end vedle http serveru websocket server. Front-end při načtení otevře websocket spojení s back-endem. Pokud uživatel zprávu pošle, podívá se websocket server, zda adresát patří mezi právě připojené uživatele a pokud ano, pošle mu přes websocket zprávu.

Na frontendu se pak o příchozí zprávy stará redux middleware, který po přechodu zprávy aktualizuje UI.

2.4 Chyby

2.4.1 Nevyužití TypeScriptu

TypeScript je nadmnožina javascriptu, která se stará o vynucení dodržování typu proměn, jelikož to javascript jako jazyk sám o sobě nedělá. Není to problém u projektu s nízkou komplexitou, ale můj projekt během vývoje na ní rapidně nabíral. Kvůli tomu jsem musel strávit hodně času opravováním bugů, které by při vynucení typů nevznikly.

3. Technická dokumentace

Instalace tohoto projektu je přes všechny moje snahy poněkuď složitá.

3.1 Prerekvizity

Pro instalaci a spuštění projektu je potřeba mít nainstalovaný NPM nebo-li node packet manager a PostgreSql server. Vřelé doporučuju také pgAdmin 4, pro snadný import databáze.

3.2 Instalace

3.2.1 Import databáze

Nejdříve je potřeba vytvořit databázi. To je možné udělat dvěma způsoby. Buď exekucí sql scriptu ze souboru keplerdb-plain, který se nachází v repozitáři ve složce book-exchange-db nebo vytvořením prázdné databáze v pgAdminu a provedením příkazu restore ze souboru keplerdb-sql.

3.2.2 Instalace dependecies

Ve složkách book-exchange-fronted a book-exchange-backend je třeba provést příkazy npm install, pro stažení všesch potřebných balíčku.

3.2.3 Vytvoření .env souboru

Pro správný běh aplikace je potřeba nastavit některé proměnné.

V obou částech aplikace se v root složce nachází soubor example.env, kde jsou všechny potřebné proměnné popsané.

Aplikace používá k některým funkcím google konzoli. Je možné vytvořit nový projekt nebo použít tyto tokeny:

CLIENT-ID = '1028693232863-ahvojojs4rkgl1qv10p7q5010mk6hilk.apps.googleusercontent.com'

API-KEY = 'AIzaSyDZisDYd6quGbSGqez16HOOmMPtVYODmxA'

3.2.4 Lokální spuštění aplikace

Po všech předchozích krocích stačí v obou složkách provést příkaz:

npm start

Závěr

Výsledekem práce je aplikace, která řeší problémy fyzické burzy. Obsahuje všechny k tomu potřebné náležitosti, jako tvorba a spravování nabídek, jejích třídění dle tagů a systém komunikace mezi uživateli.

Dovolím si tvrdit, že jsem zadání splnil. Za skutečný cíl jsem si pokládal vytvoření nejenom aplikace, která naplňuje text zadání, ale aplikace, která se opravdu dá používat.

Stojí za zmínku chybějící systém moderace, která se momentálně dá provádět pouze úpravou dat v databázi. Na vytvoření takového systému mi bohužel nezbyl čas.

Během práce jsem prohloubil své znalosti node.js, naučil se pracovat s mnoha knihovnami a poprvé úspěšně implementoval websockety.

Nakonec chci říct, že se obávám pouze náročnosti instalace řešení. Nejradši bych to udělal sám, ale snad budou uvedené instrukce stačit. Dále vím, že zmíňovat tokeny v dokumentaci není nejlepší, ovšem nenapadá mě jak to jinak udělat.

Seznam obrázků