

GYMNÁZIUM, PRAHA 6, ARABSKÁ 14

PROGRAMOVÁNÍ



ROČNÍKOVÝ PROJEKT

BURZA UČEBNIC

Vypracoval:
Vyučující:

Adam Lisner, 4.E
Ing. Daniel Kahoun

Únor 2022

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 2. března 2022

Poděkování

Rád bych na tomto místě poděkoval panu Ing. Danielu Kahounovi za celé čtyři roky, během kterých mě vedl k programování, neboť bez něj by tento ročníkový projekt nemohl nikdy vzniknout.

ANOTACE

Cílem tohoto ročníkového projektu bylo vytvořit webovou aplikaci pro burzu učebnic. Uživatelé mohou přidávat své knihy k prodeji a prohlížet si inzeráty jiných uživatelů. V případě zájmu si také mohou knihy i kupovat. Aplikace bylo naprogramována v jazyce JavaScript s použitím Node.js a frameworku Next.js. Data jsou ukládána v databázi PostgreSQL a na cloudovém uložišti Firebase Storage

KLÍČOVÁ SLOVA

Node.js, JavaScript, Next.js, React, PostgreSQL, učebnice

ABSTRACT

The aim of this year's project was to create a web application for the textbook exchange. Users can add their books for sale and view other users' listings. They can also buy books if they wish. The application was coded in JavaScript using Node.js and the Next.js framework. The data is stored in a PostgreSQL database and on cloud-based Firebase Storage

KEYWORDS

Node.js, JavaScript, Next.js, React, PostgreSQL, textbook

ANNOTATION

Das Ziel des diesjährigen Projekts war die Entwicklung einer Webanwendung für die Schulbuchbörse. Die Nutzer können ihre Bücher zum Verkauf anbieten und die Angebote anderer Nutzer einsehen. Sie können auch Bücher kaufen, wenn sie dies wünschen. Die Anwendung wurde in JavaScript mit Node.js und dem Next.js Framework programmiert. Die Daten werden in einer PostgreSQL-Datenbank und auf dem Cloud-basierten Firebase Storage gespeichert.

SCHLÜSSELWÖRTER

Node.js, JavaScript, Next.js, React, PostgreSQL, Kursbuch

OBSAH

Úvod.....	4
1. Použité technologie	5
2. Frontend	7
2.1. Přidání knihy do nabídky	7
2.2. Zobrazení nabídky knih.....	9
2.3. Stránka pro zakoupení knihy.....	11
2.4. Profilová stránka	12
2.5. Další stránky.....	13
3. Databáze.....	14
3.1. Schéma databáze	14
3.2. PrismaClientSingleton.....	16
3.3. Hosting na serveru Avava	17
4. Backend.....	18
4.1. Přihlašování.....	18
4.2. REST API.....	18
4.2.1. GET metody	18
4.2.2. POST metody	19
4.2.3. PUT metody	19
4.3. Zasílání emailu	19
Závěr	20

ÚVOD

Tento projekt se zabývá naprogramováním aplikace Burza učebnic, jež bude sloužit jako místo, kde studenti mohou prodávat své staré učebnice a nakupovat učebnice z druhé ruky. Cílem tohoto projektu je vytvořit funkční web, který těchto požadavků bude schopný. Pro její vývoj je použita technologie Node.js a programovací jazyk JavaScript. Jakožto webový framework je použit Next.js Pro ukládání dat je použita databáze PostgreSQL, jež je hostovaná na školním serveru Avava.

Mezi další požadavky patří přihlašování pomocí účtu Google, filtrování knih podle předmětu, ke kterému jsou přiřazeny. Dále také zasílání emailu prodejci v případě, že si někdo chce knihu koupit.

Bonusovým zadáním je hosting na serveru Avava a implementace platební brány.

Práce je rozdělena do tří hlavních částí: frontend, backend a databáze. O každé z nich je pojednáváno v této dokumentaci.

1. POUŽITÉ TECHNOLOGIE

Během tvorby tohoto projektu bylo použito mnoho technologií, které nejen umožnily vznik, ale také v mnoha případech velmi pomohla. V následujících kapitole budou použité technologie postupně představeny a přiblíženy.

Hlavním stavebním prvkem aplikace je prostředí Node.js. Ten se používá převážně pro tvorbu webových aplikací, respektive jejich serverové části neboli backendu. Jeden z důvodů, proč je Node.js mezi webovými vývojáři tak oblíbený je jeho schopnost asynchronních I/O operací a neblokujícího I/O modelu. Tím se zajistí rychlejší běh aplikace, který by za použití tzv. *vanilla* JavaScriptu. Dále je také oblíbený, neboť s ním stačí vývojáři jeden jazyk jak na frontend, tak na backend. Ale bezesporu největší předností je komunita kolem Node.js, díky které vývojáři jednoduše naleznou dokumentace, podporu nebo různé nástroje a knihovny, které velmi příjemní celkový vývoj.

Jak již bylo zmíněno, Node.js využívá objektově orientovaný skriptovací jazyk JavaScript. Tento jazyk se nejčastěji využívá právě pro vývoj webových stránek a aplikací. Jedná se o interpretovaný jazyk, který běží na straně klienta (v prohlížeči) a umožňuje interaktivní uživatelské rozhraní a dynamické efekty na webových stránkách. JavaScript byl vytvořen v roce 1995 a postupem času se stal jedním z nejpoužívanějších jazyků pro tvorbu webových aplikací a rozšířil se i na jiné platformy, jako jsou desktopové aplikace a mobilní aplikace. JavaScript umožňuje vývojářům tvorbu interaktivních, dynamických a responzivních webových stránek a aplikací s moderními funkcemi a efekty.

Jakožto programovací prostředí byl využit Visual Studio Code od společnosti Microsoft, který společně s IntelliJ patří k nejlepším programovacím IDE, které může programátor použít. Velkou výhodou VSCode je velké množství rozšíření, které si může uživatel stáhnout, aby si co nejvíce personifikoval prostředí pro svoji práci.

Většina webových aplikací využívají pro svůj chod tzv. webový framework. Ani tento projekt není výjimkou. Využívá jeden z nejmodernějších frameworků, které na dnešním trhu nachází, framework *Next.js*. Tento framework je postaven na technologii *React* (viz níže) a umožňuje vytvářet rychlé webové aplikace. Přitom zprostředkovává jak uživatelské prostředí, tak routing nebo server-side rendering.

Ve frontendové části byl využit výše zmíněný React. Jedná se JavaScriptovou knihovnu, která umožňuje vytvářet interaktivní uživatelské prostředí, které se dynamicky mění pomocí JavaScriptu. Používá tzv. komponenty, které umožňují opětovné využití, což se velmi často využívá, např. pro dynamické zobrazování na klientu.

Dále byl pro stylizaci stránek použit open-source CSS framework TailwindCSS. Tento framework umožňuje vývojářům rychle vytvářet a stylizovat své stránky přímo ve zdrojovém kódu stránky, bez toho, aby připojovali externí CSS soubory. Jeho použití je

velmi intuitivní pro každého, kdo má dostatečnou úroveň stylizování v CSS, která je klíčová pro užití TailwindCSS.

Další z frontendových technologií, které byly použity je Material Design UI (zkráceně MUI), což je velká knihovna předdefinovaných React component, které může web-developer použít pro své stránky.

Nyní technologie, které byly použity pro databázi a úschovu dat. Jakožto databáze byla použita databáze PostgreSQL. PostgreSQL je open-source relační databázový systém, který se používá pro ukládání a správu dat. Jeho největšími přednostmi je vysoká stabilita a spolehlivost. Podporuje velké množství datových typů a umožňuje vytváření složitých dotazů.

Pro operaci s databází je využit open-source ORM framework Prisma. Zprostředkovává jednoduchou správu relačních databází. Velkou výhodou je definování modelů pomocí Prisma Schema souboru, ve kterém nadefinujeme veškeré relace mezi tabulkami a tabulky samotné podobně.

Pro schraňování obrázků je použita Firebase od společnosti Google. Jedná se o cloudovou platformu, která disponuje relační databází, autentifikací uživatele nebo právě cloudovým uložištěm Firebase Storage.

Pro verzování kódu byl použit Git a platforma GitHub, kde byl uložen v repozitáři. Díky tomu je možné upravovat kód na více zařízeních bez použití flash-disku, na kterém by bylo uloženo několik verzí kódu.

2. FRONTEND

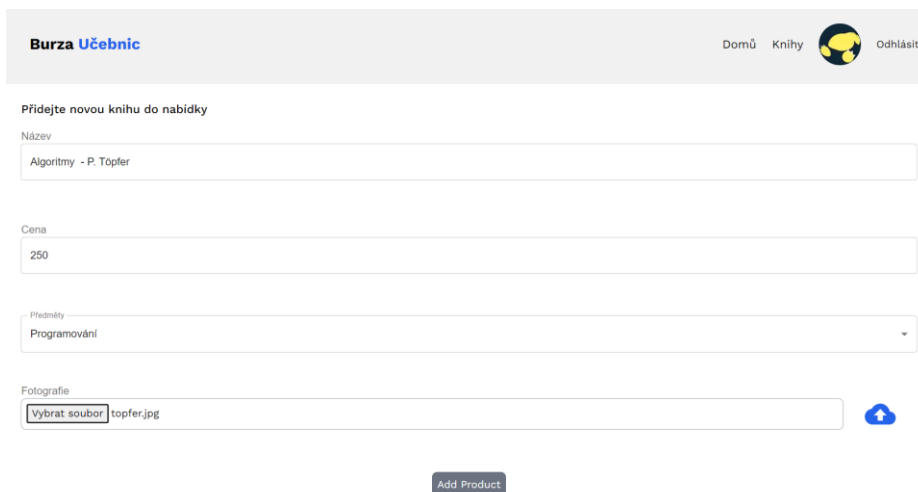
V této kapitole bude přiblížen celkový frontend aplikace. Představeny budou React komponenty a jednotlivé stránky. Komponenty jsou uloženy ve složce `/components` a jednotlivé stránky jsou reprezentovány složkami v adresáři `/pages`. Tím se také v Next.js zajišťuje routing.

2.1. Přidání knihy do nabídky

Pokud chce uživatel přidat nějakou ze svých knih do nabídky, tak může pomocí stránky na endpointu `/addProduct`. Zdrojový kód stránky samotné je uložen v souboru `/pages/addProduct/index.jsx`. V něm se nachází pouze komponenta `AddProductForm`.

Komponenta `AddProductForm` je nejkompexnější komponenta ze všech. Neboť kromě svého vyobrazení na obrazovce je také zodpovědná za zpracování dat a jejich poslání na Firebase Storage.

Celá komponenta se skládá ze dvou tlačítek a čtyř vstupních polí, pole na název, cenu, kategorii a fotografii knihy. Vstupní pole jsou převzata z knihovny *Material Design*. Tyto input na sobě mají tzv. *React hooky*. Ty sledují, co se s tímto prvkem děje a pokud s ním je manipulováno tak jednají, tak jak jim je řečeno. V tomto případě si během uživatelské interakce zapamatují, jaké hodnoty do vstupních polí zadal, a ty poté zpracují na odeslání pro další práci s nimi.



Obrázek 1 UI přidání knihy do databáze

Zajímavou funkcí je asynchronní funkce `uploadImage()`, která se stará o posílání fotografií a jejich uložení na externím cloud-uložišti Firebase po stisknutí tlačítka pro nahrání. Nejprve je pomocí funkce `uuidv4()` vygenerováno „unikátní“ jméno souboru. Poté se zavolá asynchronní funkce `uploadBytes()`, která si jako argument bere referenci na uložisko, tj. námi

vytvořený storage a URL, kde má být soubor vytvořen. Po zpracování tohoto požadavku je vytvořena URL adresa, která se uloží do hooku a je připravena pro práci s ní. Návrátové hodnoty funkce jsou `1` pro úspěšné provedení a `2` pro neúspěšnou akci. Zdrojový kód této funkce naleznete níže.

```
const uploadImage = async () => {
  if (file === null) {
    return 0;
  }
  setUploading(true);

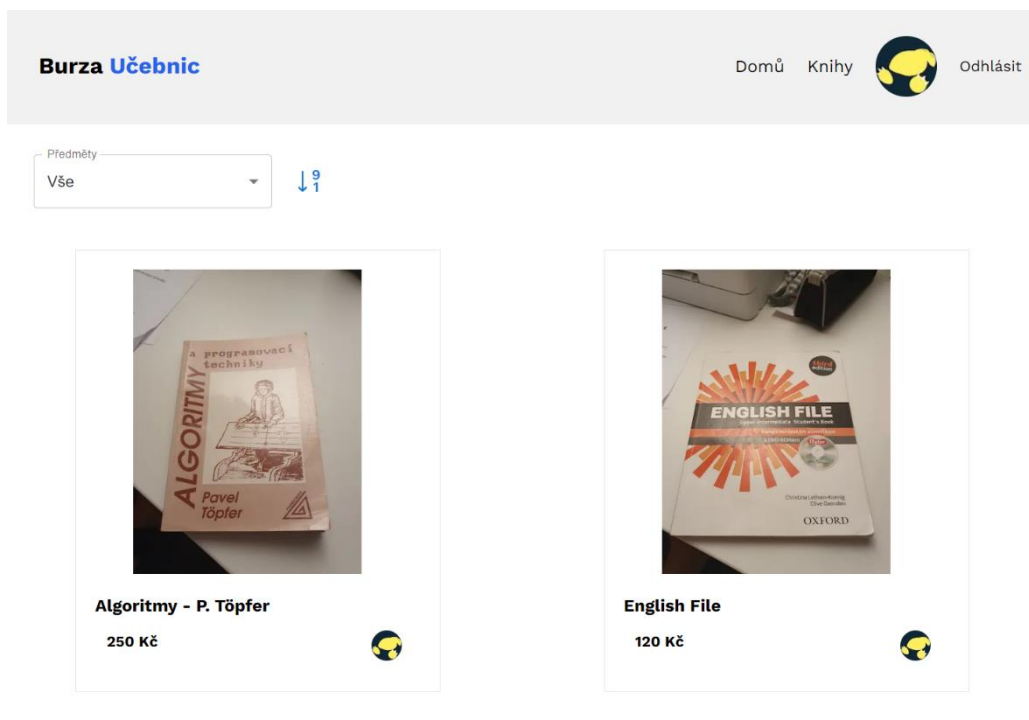
  let fileNameFB = uuidv4() + (file.type === "image/jpeg" ? ".jpg" : ".png");

  const imageRef = ref(storage, `/images/${fileNameFB}`);
  await uploadBytes(imageRef, file)
    .then(() => {
      getDownloadURL(imageRef)
        .then((url) => {
          setUrl(url);
          uploaded = true;
          setUploading(false);
          return 1;
        })
        .catch((error) => {
          console.error(error.message, "error getting image url");
          return 2;
        });
    })
    .catch((error) => {
      console.log(error.message);
    });
};
```

Pro posílání dat do vlastní databáze je vytvořena funkce sendDataToDatabase(). Ta je vyvolána akcí na Buttonu. Do proměnných se nejprve uloží hodnoty z input fieldů a z useSession() hooku se získá ID uživatele. Poté je v try-catch bloku volána asynchronní funkce fetch(). Tou jsou pomocí HTTP POST poslány do API data uložená v proměnných v podobě JSON. Více se dozvíte v kapitole 4.2. Po tomto požadavku na server je uživatel přesměrován na stránku s poděkováním.

2.2. Zobrazení nabídky knih

Pro zobrazení celé nabídky knih je vytvořena stránka, ke které se uživatel dostane na endpointu `/product`. Zdrojový kód této stránky je uložen v souboru `/pages/products/index.jsx``.



Obrázek 2 Nabídka knih

Pro získání dat z REST API je použita asynchronní metoda `getServerSideProps()`, která bere data volání `fetch()` na `/api/products`. Tato metoda běží pouze na server-side a je zodpovědná za pre-render stránky, tudíž je obsah načten ještě před načtením stránky.

Obsah stránky si lze také dynamicky změnit, pomocí ovládání, které se nachází v horní části stránky (viz obrázek 2). Uživatel může filtrovat knihy podle předmětu, ke kterému byly prodejcem přiřazeny. Také je možnost řadit knihy podle ceny. K tomu je použita vestavěná javascriptová funkce `sort()`.

Knihy jsou vloženy v divu, který má nastavené zobrazení pomocí gridu, které se mění podle velikosti viewportu.

Pro zobrazení jednotlivých knih, které jsou na prodej, je určena komponenta `ProductComponent`. Ta přejímá argument *props*, ve kterém se při použití komponenty předávají data, která se mají zobrazit jako například URL obrázku nebo název.



Obrázek 3 Komponenta pro zobrazení knihy

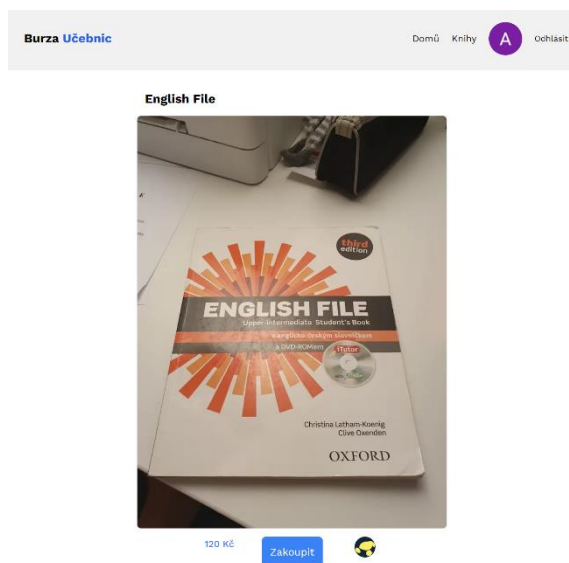
Komponenta obsahuje obrázek učebnice, její název a cenu. Dále obsahuje profilovou fotografii, která slouží jako odkaz na profil prodejce. K vidění je na obrázku č. 3.

Při kliknutí na název je uživatel přesměrován na stránku produktu samotného. Aby bylo zajištěno, že se ke koupi dostane pouze přihlášený uživatel, je odkaz na stránku podnícen hodnotou booleanu `canGoForward` v argumentu. Ten nabyde kladné hodnoty pouze v případě, že získaná `session` z hooku `useSession()` bude nabývat nenullové hodnoty.

2.3. Stránka pro zakoupení knihy

Pro zobrazení detailního zobrazení knihy je vytvořena stránka, ke které se uživatel dostane na dynamickém endpointu `/products/<id-knihy>`. Zdrojový kód této stránky je uložen v souboru `/pages/products/[...id].jsx`.

Pro získání dat je použita funkce `getServerSideProps()`, která přejímá v argumentu context požadavku, ze kterého získá parametr URL, který je stejný jako id knihy. Poté se metodou `fetch()` získají data z API endpointu `/api/products/${id}`, který je definován tzv. literal expression, tj. vložení řetězce z proměnné do již definovaného řetězce.



Obrázek 4 Detail knihy

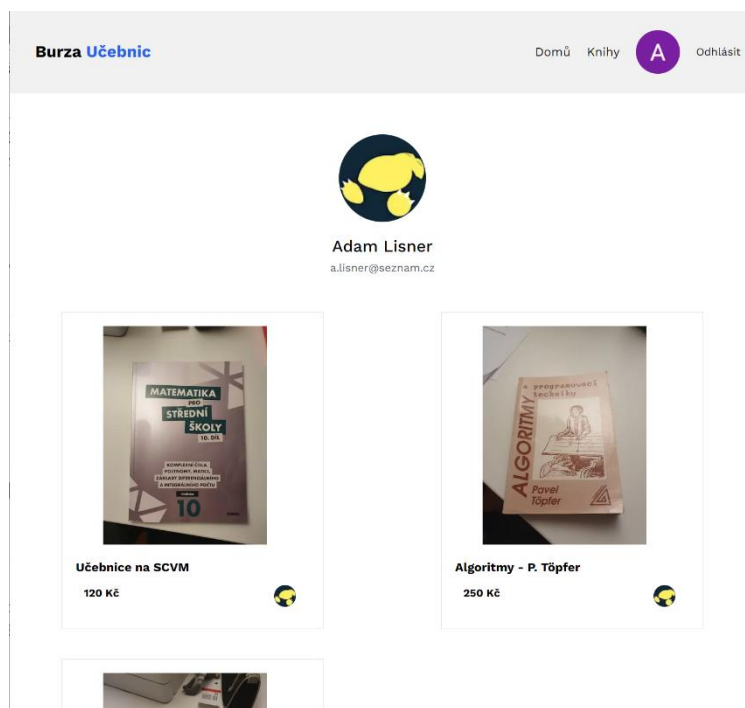
Struktura stránky je velmi prostá (jak lze vidět na obrázku č. 4), člověk na ni nalezne název knihy, její fotografii, cenu, odkaz na prodejce a výzvu ke koupi. Výzva ke koupi je reprezentovaná tlačítkem, které se zobrazí pouze uživatelům oprávněným ke koupi, tj. všichni přihlášení - `{prodejce}`. Po stisknutí tlačítka se zobrazí pop-up panel, ve kterém se uživatele naposledy uživatele aplikace ptá, jestli si opravdu chce koupit knihu. Po stisknutí se zavolá více metod, nejprve se nastaví status knihy na prodaná, poté se přidá do tabulky prodejů záznam o prodeji, a nakonec se pošle email prodejci, kde je informován o zájmu o jeho knihu. Bližší informace o těchto metodách naleznete v kapitole 4.2.

2.4. Profilová stránka

Stránka, na které se nachází informace o uživateli, je reprezentována dvěma soubory: `/pages/profile/index.jsx` a `/pages/profile/[...id].jsx`. Nacházejí se na endpointech `/profile` a `/profile/<id-uživatele>`. Stránka bez UID v parametru URL je určena pouze pro zobrazení profilu právě přihlášeného uživatele, zatímco stránka v jejímž URL najdeme UID je pro zobrazení libovolného uživatele.

Na stránce najdeme profilovou fotku, jméno a email, který po kliknutí odkáže uživatele do emailové schránky a umožní mu napsat email majiteli tohoto profilu. Kolonka s emailem se nenachází na stránce právě přihlášeného uživatele.

Pod profilovými informacemi se nachází seznam všech produktů, které vlastní daný uživatel. Tyto produkty získá pomocí metody `getServerSideProps()`, která vrátí produkty v JSONu pomocí fetch API routu. Více o API se dozvíte v kapitole 4.2. Vizual profilové stránky si můžete prohlédnout na obrázku č. 5.



Obrázek 5 Stránka profilu

2.5. Další stránky

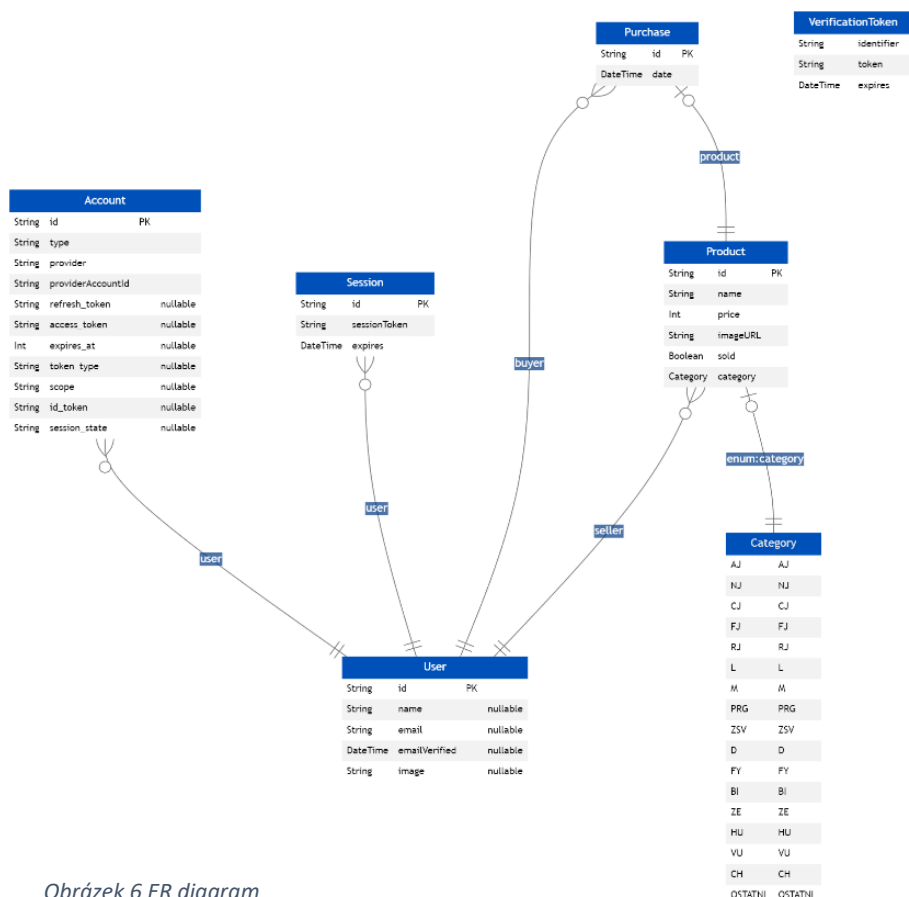
Kromě všech výše zmíněných stránek se v aplikaci nachází také několik dalších. Například stránka pro případ odpovědi serveru 404, tedy stránka nebyla nalezena, je vytvořena vlastní React funkce v souboru 404.jsx. V tomto je velmi nápomocný právě framework Next.js, jehož tvůrci mysleli na tento případ a do svého produktu integrovali možnost vytvoření *Page Not Found* jednoduše jedním souborem. Toto řešení je jednodušší než v jiných frameworkcích, např. Django kde je nutné zasahovat do konfiguračního souboru aplikace

3. DATABÁZE

Následující kapitola se zabývá databázovým systémem aplikace.

3.1. Schéma databáze

V databázi je celkově šest tabulek. Tabulky: *Account*, *Session*, *User*, *VerificationToken* jsou generované přihlašovacím enginem NextAuth (více o přihlašování v kapitole 4.1.). Dále se zde nachází tabulka *Product* a *Purchase*. Na obrázku č. 6 naleznete databázové schéma.



Obrázek 6 ER diagram

Tabulka *Product* obsahuje primární klíč *id*, který je automaticky generován ve tvaru uuid. Dále obsahuje kolonku *name*, *price*, *imageUrl*. Relačně je propojena s tabulkou *User*, čímž se v produktu ukládá jeho prodejce. Produkt si také udržuje kategorii, ke které spadá. Ta je v Prisma schématu definovaná výčtem hodnot, které vyjadřují jednotlivé předměty. Další relací je vztah s tabulkou *Purchase*, o které se dozvíte více níže.

Tabulka *Purchase* slouží k zapamatování jednotlivých prodejů. Obsahuje primární klíč *id*, který je vytvořen pomocí funkce *uuid* přímo Prismou. Záznamy jsou dále relačně propojeny s tabulkou *User*, což reprezentuje kupujícího, a také s tabulkou *Product*. Touto relací

získáme informace o prodejci, tudíž není nutné vytvářet další záznam o prodejci, byl by nadbytečný.

3.2. PrismaClientSingleton

Pro obsluhu databáze je využíván ORM framework Prisma, který se skládá ze tří systémů: *Prisma Studio* (GUI pro prohlížení databáze), *Prisma Migrate* (migrační systém) a *Prisma Client* (query builder pro Node.js). Poslední jmenovaný je využíván pro tvorbu dotazů. Využitím tohoto nástroje není nutné vytvářet často komplikované dotazy pomocí SQL příkazů, namísto toho se jednoduchým příkazem určí podmínky, které musí daný příkaz splňovat.

Jedním z problémů, které komunita s našla je přetékání *connection poolu* z důvodu vytváření neustále nových instancí *PrismaClient*, jak radila dokumentace. Řešením tohoto problému je využití návrhového vzoru *Singleton*, který zajišťuje vytvoření jedné instance tohoto objektu na uživatele. Níže můžete vidět implementaci návrhové vzoru Singleton.

```
import { PrismaClient } from "@prisma/client";

/**
 * Singleton class for creating and managing a single instance of the `PrismaClient`.
 */
class PrismaClientSingleton {
  /**
   * Private static property that holds the singleton instance of the `PrismaClient`.
   * @type {PrismaClient}
   * @private
   * @static
   */
  static instance = null;

  /**
   * Private constructor to prevent creating new instances of the `PrismaClientSingleton`
   class.
   * @private
   */
  constructor() {}

  /**
   * Public static method that returns the singleton instance of the `PrismaClient`.
   * If the instance doesn't exist, it creates a new instance of the `PrismaClient`.
   * @returns {PrismaClient} - The singleton instance of the `PrismaClient`.
   * @public
   * @static
   */
  static getInstance() {
    if (!PrismaClientSingleton.instance) {
      PrismaClientSingleton.instance = new PrismaClient();
    }
    return PrismaClientSingleton.instance;
  }
}

export default PrismaClientSingleton.getInstance();
```

3.3. Hosting na serveru Avava

Součástí bonusového zadání bylo hostovat stránku na serveru Avava. Nyní se na tomto serveru nachází databáze aplikace. Připojení k databázi probíhá přes SSH tunel, který je zprostředkován pomocí aplikace Putty. V raném vývoji byla aplikace hostována na lokálním PostgreSQL serveru. Toto řešení bylo jednoduché a fungovalo dobře, ovšem během vývoje z různých desktopů nebo laptopu bylo nutné pracovat se stejnými daty, aby nemuselo být vytvářeno velké množství testovacích dat po každé změně prostředí. Jelikož je Avava na linuxovém serveru, bylo nutné ovládat základy Bash.

4. BACKEND

Backend tohoto projektu je psán v jazyce JavaScript a využívá technologii REST API pro získávání dat. REST API v tomto projektu funguje na sdílení dat mezi serverem a klientem pomocí JSON, ze kterého vyzíská data jak server, tak i uživatel. K backendu je připojena databáze PostgreSQL, kterou ovládá ORM framework Prisma.

4.1. Přihlašování

Pro přihlašování je využita open-source knihovna NextAuth.js, která slouží pro autentizaci uživatelů v Next.js aplikacích. Tato knihovna poskytuje nespočet poskytovatelů pro autentizaci, jako například GitHub, Facebook nebo Google. V této aplikaci je použit pro autentizaci poslední jmenovaný. Pro implementaci stačí použít kód poskytnutý autory v jejich dokumentaci. Kromě autentizace také poskytuje React hook *useSession()*, kterým lze získat informace o uživateli. V základu poskytuje pouze email, jméno a profilový obrázek uživatele. Pro získání dalších informací je nutné v API upravit *callback*, aby vracel i data, která jsou potřeba, například ID uživatele.

NextAuth má implementované metody *signIn()* a *signOut()*, které po zavolání přihlásí nebo odhlásí uživatele z aplikace.

4.2. REST API

Pro komunikaci mezi backendem a frontendem aplikace je využito REST API, které odpovídá na dotazy ve formě JSON. Celé API je uloženo ve složce */pages/api*.

4.2.1. GET metody

V následující kapitole budou představeny všechny GET metody z API.

Pro získání produktů z databáze jsou vytvořeny dva javascriptové soubory, které využívají *PrismaClientSingleton* (viz kapitola 3.2.) pro hledání v databázi.

Pro nalezení všech produktů slouží soubor */api/product/index.js*. V asynchronní metodě *handler*, je v try-catch bloku je do konstanty uložen výsledek metody *prisma.product.findMany(...)*, která nahrazuje klasické SQL příkazy. Tato konstanta je poté vrácena v *response* v podobě JSON.

Pro získání dat o jednom produktu je použit skript v JavaScriptu, jenž se nachází ve stejné složce jako předchozí v souboru *[id].js*. Díky tomuto zápisu umožňuje Next.js vytvářet dynamický routing. Skript volá v metodě *handler*, která v argumentu přejímá http request a response. Z argumentu získává *id* knihy, díky čemuž pak hledá tuto knihu v databázi

metodou *prisma.product.findFirst(...)*. Výsledek tohoto hledání v databázi je vrácen v odpovědi serveru v podobě JSON.

Pro nalezení všech knih, které inzeruje daný uživatel, použit skript */api/userProduct/[id].js*. Ten hledá všechny produkty, u kterých se záznam od *id* prodejce shoduje s *id*, které je předáno v *requestu*. Výsledek tohoto hledání je poté vrácen v JSON.

Pro získání informací o jednom daném uživateli je použit skript */api/user/[id].js*. Ten hledá v tabulce *User* informace o uživateli, jehož *id* je stejné jako *id* v *requestu*.

4.2.2. POST metody

Na stránce, kde lze přidat knihu k inzerci (viz kapitola 2.1.), je po vyplnění dotazníku odeslán požadavek na server pomocí http metody POST. Ta pomocí metody *fetch()* předá API na endpointu */api/createProduct* informace o knize v JSONu. Z něj jsou dekonstrukcí získány informace zpět a voláním metod *prisma.product.create(...)* je poté vytvořen nový záznam v databázi. Kód tohoto skriptu naleznete v adresáři */pages/api/createProduct* v souboru *index.js*.

Metoda POST je také využita pro vytvoření záznamu o prodeji v tabulce *Purchase*. Ta ukládá data o prodejích knih v databázi.

4.2.3. PUT metody

Při procesu koupě knihy je nutné zasahovat do atributu *sold* dané knihy. Tento atribut se mění pomocí HTTP metody PUT, která pomocí *fetch()* pošle požadavek na se těla požadavku *id* dané knihy. Po získání identifikátoru se pomocí *PrismaClientu* u dané knihy změni hodnota ze *sold: false* na *sold: true*.

4.3. Zasílání emailu

Aby byl prodejce obeznámen o tom, že o jeho knihu projevil nějaký uživatel zájem, tak ses používá knihovna *email.js*, kterou lze nainstalovat pomocí příkazu *npm install emailjs*. Tato knihovna slouží k posílání emailových zpráv pomocí protokolu SMTP. Nejprve je nutné inicializovat novou instanci objektu *SMTPClient*, tento objekt je nutné nejprve importovat z *emailjs*. Během této inicializace se předává v argumentu emailová adresa, kterou chceme použít pro zasílání. Dále heslo k tomuto emailu, při použití dvoufázového ověření je nutné použít tzv. *application password*, které lze získat v nastavení svého Google účtu. Tato knihovna má výhodu oproti jiným, že používá zabezpečený přenos pomocí protokolu SSL.

Zaslání emailu proběhne po odkliknutí tlačítka na klientu pomocí asynchronní metody *sendEmail()*. Tato metoda vezme metodu *fetch()* a metodou POST pošle požadavek na endpoint */api/email*, ze kterého se zavolá asynchronní metoda *sendAsync()*, jež obsahuje veškeré informace z těla požadavku.

ZÁVĚR

S výsledkem práce jsem spokojený. Cíl vytvořit funkční webovou aplikaci pro inzerci učebnic se povedlo. Uživatel může knihy filtrovat podle předmětu, ke kterému jsou přiřazeny, a může je také řadit podle ceny. Plně funkční je také přidávání nové knihy do nabídky pomocí formuláře. Mezi cíle, které byly naplněny, patří také zasílání emailu pomocí protokolu SMTP. Velmi spokojený jsem také s tím, jak aplikace vypadá. Je plně responsivní, má moderní design.

Z bonusové části byl částečně naplněn cíl hostingu na Avava serveru. Na tomto serveru je nyní hostovaná databáze. Ovšem musím říci, že s tímto serverem už nechci v budoucnu pracovat, neboť je velmi neintuitivní. Bonus v podobě platební brány nebyl z časových důvodů splněn.

Tento projekt byl mojí první full-stack aplikací, kterou jsem vytvořil. Doposud jsem se vždy soustředil na frontendovou část vývoje, tudíž mě tato aplikace zasvětila do tvorby backendu. Byl to také můj první projekt, kde jsem využil React. Tato technologie je podle mého názoru velmi dobrá i přes všechny její strasti. Webový framework Next.js bych také doporučil.

Do budoucna by se dal vyřešit celkový hosting aplikace, a hlavně platební brána pomocí PayPal API. Ale i přes tyto nedostatky jsem s výsledkem mého projektu spokojený a chtěl bych se podobným projektům z oblasti webového vývoje věnovat i v budoucnu.

OBRÁZKY

Obrázek 1 UI přidání knihy do databáze.....	7
Obrázek 2 Nabídka knih.....	9
Obrázek 3 Komponenta pro zobrazení knihy	10
Obrázek 4 Detail knihy	11
Obrázek 5 Stránka profilu	12
Obrázek 6 ER diagram	14

POUŽITÉ ZDROJE

FreePik. (2023). *StorySet - Awesome free customizable illustrations for your next project*. Načteno z StorySet: <https://storyset.com/>

Google Developers. (2023). *Dokumentace Firebase*. Načteno z Firebase: <https://firebase.google.com/docs>

Google Material. (2022). *Material.io*. Načteno z material.io: material.io

Material UI SAS. (2023). *Dokumentace MUI*. Načteno z mui.com: <https://mui.com/material-ui/getting-started/overview/>

Meta Platforms, Inc. (2023). *Dokumentace React*. Načteno z React: <https://reactjs.org/docs/getting-started.html>

Node.js. (2023). *Dokumentace Node.js*. Načteno z Node.js: <https://nodejs.org/en/docs/>

PluralSight. (2023). *JavaScript.com*. Načteno z JavaScript: <https://www.javascript.com/learn>

Prisma Data, Inc. (2023). *Dokumentace Prisma*. Načteno z Prisma.io: <https://www.prisma.io/docs>

Uživatelé GitHubu. (2021). > *For those having this issue, I solved it by creating a singleton: #5139*. Načteno z GitHub: <https://github.com/prisma/prisma/issues/5139>

Vercel. (2023). *Dokumentace Next.js*. Načteno z NEXT.js: <https://nextjs.org/docs>

Vercel. (2023). *Dokumentace NextAuth.js*. Načteno z NextAuth.js: <https://next-auth.js.org/>

zackschuster, e. (2023). *Repository EmailJS*. Načteno z Github: <https://github.com/eleith/emailjs#readme>