

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DOKUMENTÁCIA K ZADANIU
SEMINÁRNA PRÁCA

2024 Bc. Miroslav Malíšek, Bc. Peter Kopecký, Bc. Michal
Vrbovský, Bc. Adela Radičová, Bc. Richard Körösi

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DOKUMENTÁCIA K ZADANIU
SEMINÁRNA PRÁCA

| | |
|-------------------|---|
| Študijný program: | Aplikovaná informatika |
| Predmet: | I-ASOS – Architektúra softvérových systémov |
| Prednášajúci: | RRNDr. Igor Kossaczký, CSc. |
| Cvičiaci: | Ing. Stanislav Marochok |

Bratislava 2024 Bc. Miroslav Malíšek, Bc. Peter Kopecký, Bc.
Michal Vrbovský, Bc. Adela Radičová, Bc. Richard Körösi

Obsah

| | | |
|----------|--|-----------|
| 1 | README z Github | 1 |
| 1.1 | Frontend | 1 |
| 1.2 | Backend | 2 |
| 2 | Projekt setup a dokumentácia | 4 |
| 2.1 | Projektová štruktúra | 4 |
| 2.2 | Version control a README súbor | 7 |
| 3 | Backend | 8 |
| 3.1 | API dizajn | 8 |
| 3.2 | Databáza | 10 |
| 3.3 | Security | 11 |
| 3.3.1 | Autentifikácia | 11 |
| 3.3.2 | Autorizácia | 12 |
| 3.3.3 | Ochrana proti XSRF | 12 |
| 3.3.4 | Ochrana hesiel používateľov | 12 |
| 3.3.5 | Ochrana voči bruteforce útokom | 13 |
| 3.3.6 | Ochrana voči SQL injection | 13 |
| 3.3.7 | Validácia | 13 |
| 4 | Frontend | 16 |
| 4.1 | Responzívny dizajn | 16 |
| 4.2 | User Experience | 16 |
| 4.3 | State Management a Interakcia | 17 |
| 4.4 | Integrácia s API | 19 |
| 5 | Best Practices and Code Quality | 21 |
| 5.1 | Code Structure and Readability | 21 |
| 5.2 | Error Handling and Logging | 22 |
| 6 | Testovanie | 24 |
| 6.1 | Unit Testy | 24 |
| 6.2 | Integration Testy | 24 |
| 6.3 | Load Testy | 26 |
| 7 | Continuous Integration | 28 |

| | | |
|-----------------------------------|-------------------------------|-----------|
| 7.1 | Kontajnerizácia | 28 |
| 7.2 | Github CI | 30 |
| 7.2.1 | Github actions | 30 |
| 7.2.2 | Pravidla na branche | 31 |
| 7.2.3 | Branch strategia | 32 |
| Zoznam použitej literatúry | | 33 |
| Prílohy | | I |

Zoznam obrázkov a tabuliek

| | | |
|------------|---|----|
| Obrázok 1 | Screenshot priečinkovej štruktúry frontendu | 5 |
| Obrázok 2 | Screenshot priečinkovej štruktúry projektu | 6 |
| Obrázok 3 | Screenshot ciest aplikácie 1 | 8 |
| Obrázok 4 | Screenshot ciest aplikácie 2 | 8 |
| Obrázok 5 | Screenshot databázovej štruktúry | 10 |
| Obrázok 6 | Screenshot tabuľky 'sessions' | 11 |
| Obrázok 7 | Cookies | 12 |
| Obrázok 8 | Screenshot tabuľky 'users' | 13 |
| Obrázok 9 | Príklad rate limitingu | 13 |
| Obrázok 10 | Ukážka responzívneho dizajnu | 16 |
| Obrázok 11 | Ukážka štruktúry projektu | 22 |
| Obrázok 12 | Ukážka <i>laravel.log</i> súboru | 23 |
| Obrázok 13 | Ukážka výpisu logov z local storage | 23 |
| Obrázok 14 | Screenshot časti výsledkov testov | 25 |
| Obrázok 15 | Screenshot časti výsledku load testu | 27 |
| Obrázok 16 | Screenshot docker štruktúry pre backend | 29 |
| Obrázok 17 | Screenshot github workflows | 30 |
| Obrázok 18 | Screenshot pravidiel na branche | 31 |
| Obrázok 19 | Screenshot branchovej stratégie | 32 |

1 README z Github

1.1 Frontend

Tento repozitár obsahuje kód pre Frontendovú časť projektu na predmet ASOS v 2. ročníku API na FEI STU. Frontend je naprogramovaný v knižnici React použitím nástroja Vite. Celý projekt je dockerizovaný.

Projektom je webová aplikácia e-shop. Pre správne fungovanie je potrebné mať spustenú aj Backendovú časť aplikácie, ktorú nájdete tu:

<https://github.com/ASOS-Semestralny-projekt/Back-end.git>.

Aplikácia poskytuje základné funkcionality bežného e-shopu:

- Používateľ môže prezerať produkty, filtrovať podľa kategórií alebo hľadať podľa kľúčového slova.
- Každý produkt má detailnú stránku.
- Produkty je možné pridávať do košíka, meniť ich množstvá a vykonať objednávku.
- Používateľ má možnosť registrácie a prihlásenia.
- Prihlásený používateľ vidí históriu objednávok, môže meniť osobné údaje (okrem emailu) a heslo.
- Neprihlásený používateľ nemá prístup k histórii objednávok ani k úpravám údajov.

Proces objednávky:

1. Prehľad obsahu košíka s celkovou sumou.
2. Zadanie údajov (automaticky predvyplnené pre prihlásených používateľov).
3. Súhrn objednávky a údajov s možnosťou úprav.
4. Odoslanie objednávky, presmerovanie na domovskú stránku.

Návod na lokálne spustenie

1. Stiahnite a nainštalujte Node.js: <https://nodejs.org/en/download/prebuilt-installer>.
2. Naklonujte repozitár:

```
1 git clone https://github.com/ASOS-Semestralny-projekt/Front-end.git
```

3. Prejdite do domovského adresára projektu:

```
1 cd cesta_k_prieinku/Frontend
```

4. Nainštalujte závislosti:

```
1 npm install
```

5. Spustite aplikáciu:

```
1 npm run dev
```

6. Otvorte aplikáciu v prehliadači: <http://localhost:3000/>.

Návod na spustenie v Dockeri

1. Otvorte aplikáciu Docker vo vašom počítači.

2. Naklonujte repozitár:

```
1 git clone https://github.com/ASOS-Semestralny-projekt/Front-end.git
```

3. Prejdite do domovského adresára projektu:

```
1 cd cesta_k_prieinku/Frontend
```

4. Vytvorte Docker kontajner aplikácie:

```
1 docker-compose up --build
```

5. Otvorte aplikáciu v prehliadači: <http://localhost:3000/>.

1.2 Backend

This repository provides a back-end service for e-shop written in the PHP framework Laravel. Further documentation about endpoints can be viewed on this link.

How to Run Back-end Clone Repository

```
1 git clone https://github.com/ASOS-Semestralny-projekt/Back-end.git
```

Create .env File

Go to the Back-end folder:

```
1 cd Back-end
```

Depending on your operating system, use one of the following commands:

- Linux/MacOS:

```
1 cp .env.example .env
```

- Windows Command Prompt:

```
1 copy .env.example .env
```

- Windows Powershell:

```
1 Copy-Item .env.example .env
```

Start Application

Run the following commands:

```
1 docker compose build
2 docker compose up
```

Caution: Changed Port in Requests

NGINX port was changed to 83 instead of the default 80 because of Windows-specific issues. For example: localhost:83/products.

2 Projekt setup a dokumentácia

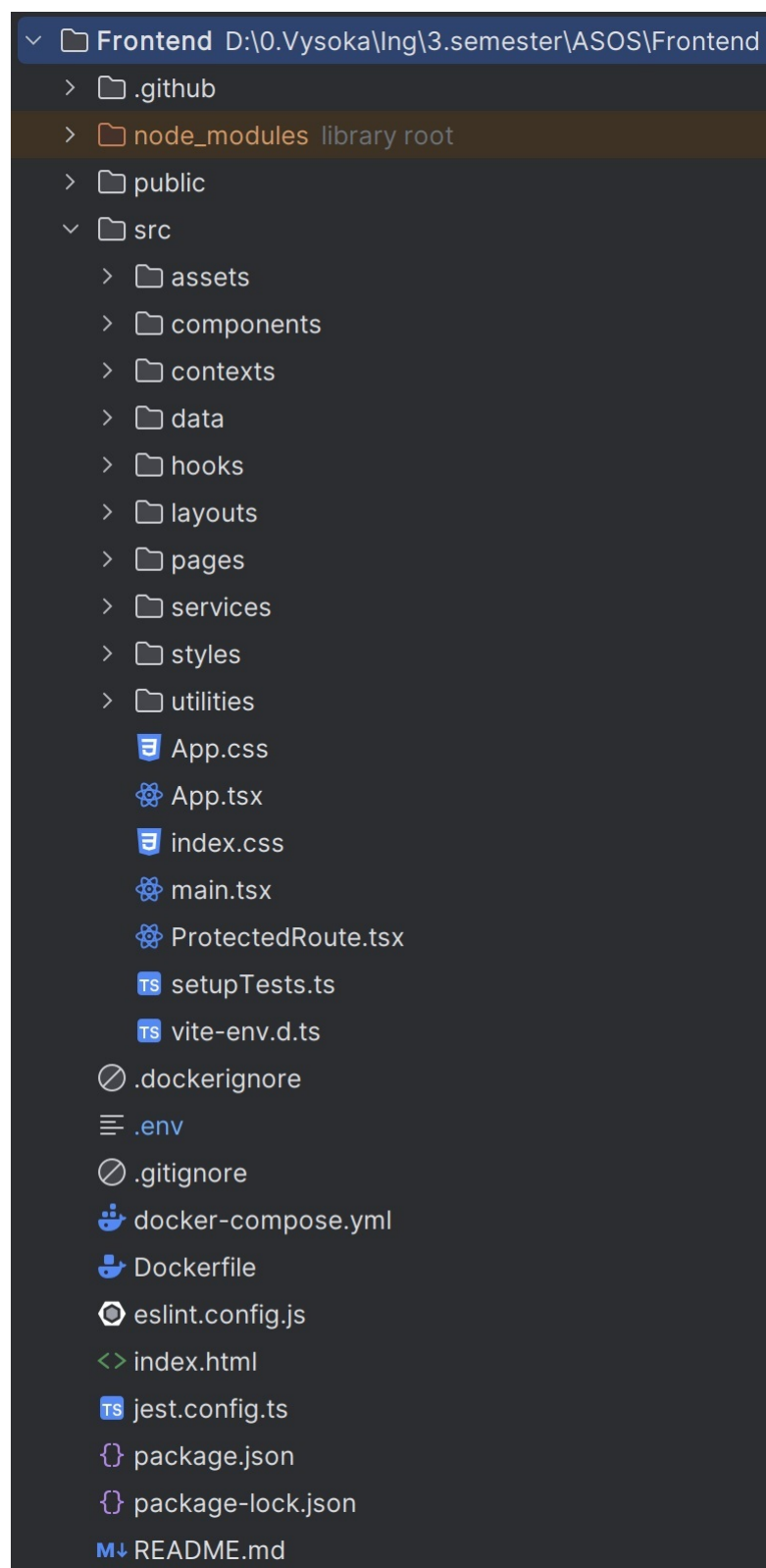
Obsahom tohto projektu je webová aplikácia simulujúca E-shop. Zameriava sa primárne na predaj elektroniky. Aplikácia poskytuje základné funkcionality bežného e-shopu, ako prezeranie tovaru, registráciu a prihlasovanie používateľov, nákupný košík s možnosťou objednania tovarov, históriu objednávok a zmena údajov v používateľovom profile.

2.1 Projektová štruktúra

Aplikácia sa skladá z dvoch samostatných častí, ktoré medzi sebou môžu komunikovať.

Frontendová časť, teda prezentačná vrstva, je vytvorená pomocou knižnice React s použitím nástroja Vite. Ako programovací jazyk sme zvolili TypeScript a na dizajn knižnice ako 'react-bootstrap', '@mui/material' alebo 'primereact'. React ponúka vzorovú štruktúru projektu, čo sprehľadňuje celkový kód aplikácie. Štruktúru projektu, ktorý je použitý v aplikácii, môžeme vidieť na obr. č. 19.

Hlavný súbor je App.tsx, kde sú definované všetky stránky aplikácie a navigácia. Priechinok 'pages' obsahuje samotné jednotlivé stránky, ktoré sa ďalej skladajú z komponentov z priechinku 'components'. Okrem toho v priechinku 'context' nájdeme definície Context API pre košík a používateľskú autentifikáciu. Priechinok 'services' v sebe obsahuje všetko potrebné pre komunikáciu s inými službami, v našom prípade primárne s backendovou časťou. Nachádzajú sa tu jednak data transfer objekty, ale aj samotná API pre komunikáciu s backendom.

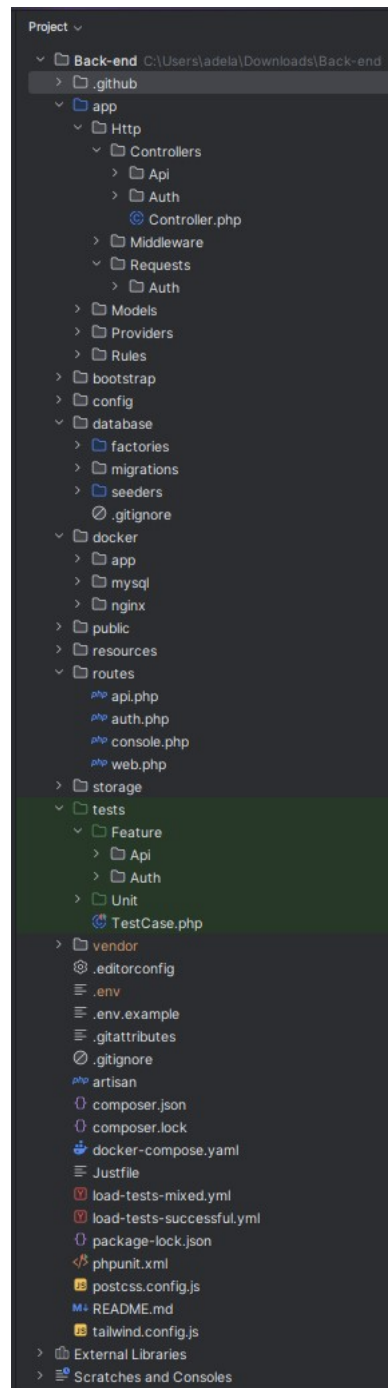


Obr. 1: Screenshot priečinkovej štruktúry frontendu

Backendová časť aplikácie je vytvorená pomocou PHP frameworku Laravel. Ten so

sebou pri vytvorení projektu prináša už istú štruktúru, čo sa týka priečinkov projektu, ktorú sme sa snažili dodržiavať a zmysluplne rozšíriť.

Projekt sa delí medzi rôzne podpriečinky, ako ‘Controllers’, ‘Middleware’, ‘Requests’, ‘Models’, ‘database’, ‘docker’, ‘routes’, ‘tests’ a ďalšie.



Obr. 2: Screenshot priečinkovej štruktúry projektu

2.2 Version control a README súbor

Obe časti aplikácie boli vyvíjané v samostatných repozitároch. To výrazne zvýšilo prehľadnosť celej aplikácie. Oba repozitáre však boli súčasťou jedného projektu, ktorý bol vytvorený na github.com.

Link na projekt sa nachádza tu: <https://github.com/ASOS-Semestralny-projekt>.

Link na frontend repozitár sa nachádza tu: <https://github.com/ASOS-Semestralny-projekt/Frontend>.

Link na backend repozitár sa nachádza tu: <https://github.com/ASOS-Semestralny-projekt/Backend>.

Počas vývoja sme pri commitoch využívali konvenciu Conventional Commits.

Oba repozitáre obsahujú README.md súbory s presným postupom spustenia (lokálneho alebo dockerizovaného) danej časti aplikácie. Frontend repozitár navyše obsahuje aj stručný popis prezentačnej vrstvy webovej aplikácie.

3 Backend

3.1 API dizajn

RESTful princípy v projekte:

1. Identifikácia zdrojov prostredníctvom URL : každý zdroj je identifikovaný jedinečnou URL.

```
Route::middleware(['checkUserLoggedIn'])->group(function () {
    Route::get( uri: '/user', [UserController::class, 'show']);
    Route::put( uri: '/user', [UserController::class, 'update']);
    Route::put( uri: '/user/password', [UserController::class, 'updatePassword']);
    Route::get( uri: '/orders', [OrderController::class, 'getOrders']);
});

Route::get( uri: '/products', [ProductController::class, 'index']);
Route::get( uri: '/categories', [CategoryController::class, 'index']);
Route::get( uri: '/products/{categoryId}', [ProductController::class, 'getByCategory']);
Route::get( uri: '/product/{productId}', [ProductController::class, 'getById']);
Route::post( uri: '/place-order', [OrderController::class, 'placeOrder']);
```

Obr. 3: Screenshot ciest aplikácie 1

```
Route::post( uri: '/register', [RegisteredUserController::class, 'store'])
    ->name( name: 'register');

Route::post( uri: '/login', [AuthenticatedSessionController::class, 'store'])
    ->name( name: 'login');

Route::get( uri: '/logout', [AuthenticatedSessionController::class, 'destroy'])
    ->name( name: 'logout');
```

Obr. 4: Screenshot ciest aplikácie 2

2. Použitie HTTP metód : aplikácia používa štandardné HTTP metódy na vykonávanie akcií na zdrojoch.
 - GET na načítanie dát.
 - POST na vytvorenie nových zdrojov.
 - PUT na aktualizáciu zdrojov.

3. Statelessness : Každá požiadavka od klienta na server obsahuje všetky informácie potrebné na pochopenie a spracovanie požiadavky. Server neukladá žiadny kontext klienta medzi požiadavkami.
4. Reprezentácia zdrojov : zdroje sú reprezentované vo formáte JSON.

Príklad response na request GET user :

```
1 {  
2   "first_name": "Jozko",  
3   "last_name": "Mrkvicka",  
4   "street": "Ilkovicova",  
5   "house_number": "11B",  
6   "city": "Bratislava",  
7   "zip_code": "82103",  
8   "country": "Slovensko",  
9   "email": "example@example.com",  
10  "phone": "421914567890"  
11 }
```

5. Jednotné rozhranie : v projekte je dodržiavaná jednotná štruktúra, ktorú možno vidieť, ako na priečinkovej štruktúre tak na routes, dodržiavaní MVC architektúry (rozdelenie na model, view, controllers aj v rámci súborov) a ďalších aspektoch.
6. HTTP status kódy : aplikácia používa vhodné HTTP status kódy na indikáciu úspechu alebo zlyhania operácií.
 - 200 OK pre úspešné načítanie.
 - 201 Created pre úspešné vytvorenie.
 - 400 Bad Request pre neúspešnú validáciu.
 - 401 Unauthorized pre nepovolené operácie.
 - 404 Not Found pre neúspešné vyhľadávanie.
 - 409 Conflict pre neúspešnú validáciu.
 - 500 Internal Server Error pre neúspešné vykonanie operácie na strane servera.

Pre jasné zdokumentovanie všetkých endpointov sme vytvorili Swagger projekt. K nemu mali prístup všetci členovia tímu, čo uľahčilo vývoj a integráciu backendovej s frontendovou časťou aplikácie. Projekt je dostupný tu:

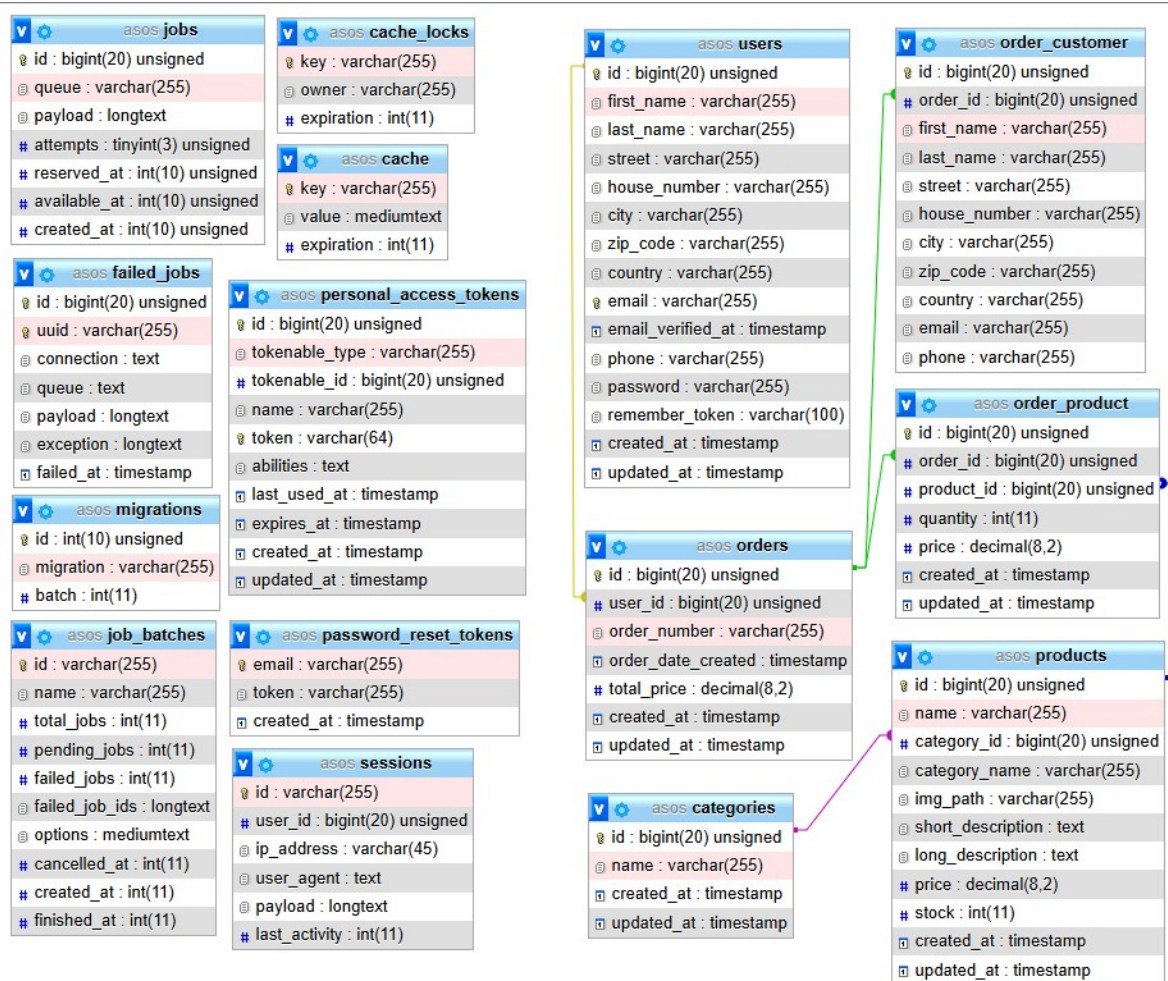
https://app.swaggerhub.com/apis/XMALISEK/ASOS_Eshop_2/1.0.0

3.2 Databáza

Pre potreby projektu sme zvolili relačnú databázu, ktorá je reprezentovaná štyrmi data modelmi : ‘User‘, ‘Product‘, Category‘ a ‘Order‘.

Vzťahy sú definované, ako :

- 1:N pre user-order
- N:M pre product-order
- 1:N pre category-product



Obr. 5: Screenshot databázovej štruktúry

Pre vytvorenie uvedenej databázovej štruktúry sme použili Laravel migrácie, ktoré nám umožňujú automatizované vytvorenie tabuľky jednoducho na základe definovanej štruktúry/stĺpcov v danej tabuľke.

Na naplnenie databázy dátami sme využili Laravel seeder-y, ktoré umožňujú zdefinovanie dát v rámci konkrétnej tabuľky, do ktorej majú byť vložené.

3.3 Security

3.3.1 Autentifikácia

Základy autentifikácie a autorizácie boli implementované za pomoci základných funkcionalít framework-u Laravel doplnených o balík *laravel/breeze*. V našom projekte využívame session-based autentifikáciu (*config/session.php*).

| | id | user_id | ip... | user_ag... | payload | last_activity |
|---|-------------------|---------|-----------|--------------------|---|---------------|
| 1 | 0LRPUxAi1mR3fQ... | 10 | 127.0.0.1 | Artillery (http... | YToz0ntz0jY6Il90b2tlbiI7czo0MDoiVnlnTV... | 1733493798 |
| 2 | 0NsSeecjpIhsuh... | 11 | 127.0.0.1 | Artillery (http... | YToz0ntz0jY6Il90b2tlbiI7czo0MDoiFlZWE... | 1733493783 |
| 3 | 1E3foQVILWKPf9... | 12 | 127.0.0.1 | Artillery (http... | YToz0ntz0jY6Il90b2tlbiI7czo0MDoicnVSOF... | 1733493793 |
| 4 | 1vJ94yRiZMwVrE... | 13 | 127.0.0.1 | Artillery (http... | YToz0ntz0jY6Il90b2tlbiI7czo0MDoiRMBn... | 1733493795 |
| 5 | 32Z30LJTGE6wGd... | 14 | 127.0.0.1 | Artillery (http... | YToz0ntz0jY6Il90b2tlbiI7czo0MDoiWHRPek... | 1733493789 |
| 6 | 494BCQN0oZHK1E... | 15 | 127.0.0.1 | Artillery (http... | YToz0ntz0jY6Il90b2tlbiI7czo0MDoiQXdpdU... | 1733493786 |
| 7 | 6wZUoV1IaBa6HX... | 16 | 127.0.0.1 | Artillery (http... | YToz0ntz0jY6Il90b2tlbiI7czo0MDoiWNJYj... | 1733493798 |
| 8 | 9IcEL45bU9HnJr... | 17 | 127.0.0.1 | Artillery (http... | YToz0ntz0jY6Il90b2tlbiI7czo0MDoic3FHRm... | 1733493799 |
| 9 | 9zoD7PTf5drwOK... | 18 | 127.0.0.1 | Artillery (http... | YToz0ntz0jY6Il90b2tlbiI7czo0MDoiSm16Z2... | 1733493791 |

Obr. 6: Screenshot tabuľky 'sessions'

```

1 return [
2     'driver' => env('SESSION_DRIVER', 'database'),
3     'lifetime' => env('SESSION_LIFETIME', 120),
4     'expire_on_close' => env('SESSION_EXPIRE_ON_CLOSE', false),
5     'connection' => env('SESSION_CONNECTION'),
6     'table' => env('SESSION_TABLE', 'sessions'),
7     'store' => env('SESSION_STORE'),
8     'cookie' => env(
9         'SESSION_COOKIE',
10        Str::slug(env('APP_NAME', 'laravel'), '_').'_session'
11    ),
12     'secure' => env('SESSION_SECURE_COOKIE'),
13     'http_only' => env('SESSION_HTTP_ONLY', true),
14     'same_site' => env('SESSION_SAME_SITE', 'lax'),
15 ];

```

Vyššie zobrazený kód predstavuje časť konfigurácie pre správu session v našom projekte, kde je použitý databázový driver na ukladanie session údajov[1].

3.3.2 Autorizácia

Niektoré endpointy sú dostupné iba autentifikovaným/prihláseným používateľom. Pri pokuse neprihláseného používateľa o prístup server vráti chybu 401 Unauthorized.

```
1 public function update(Request $request): JsonResponse {
2     /** @var User $user */
3     $user = auth()->user();
4
5     if (!$user) {
6         return response()->json([
7             'message' => 'Please log in'
8         ])->setStatusCode(401);
9     }
10    ...
11 }
```

3.3.3 Ochrana proti XSRF

V aplikácii máme ošetrenú aj ochranu voči XSRF útokom. Využívame na to cookie (XSRF-TOKEN), ktorý obsahuje unikátny token, ktorý musí byť prítomný pri požiadavkách na server aby sa potvrdilo, že požiadavka pochádza od daného prihláseného používateľa (*config/session.php*)[1].

| Name | Value | Domain | Path | Expires |
|-----------------|------------------------------|-----------|------|-------------------------------|
| XSRF-TOKEN | eyJpdil6lnVocnlxdzJzaW5mR... | localhost | / | Fri, 06 Dec 2024 16:00:55 GMT |
| laravel_session | eyJpdil6ljRHZGNhcGh4ZGFh... | localhost | / | Fri, 06 Dec 2024 16:00:55 GMT |

Obr. 7: Cookies

3.3.4 Ochrana hesiel používateľov

Heslá používateľov sú hashované za pomoci algoritmu bcrypt() (*app/Http/Controllers/Auth/RegisteredUserController.php*).

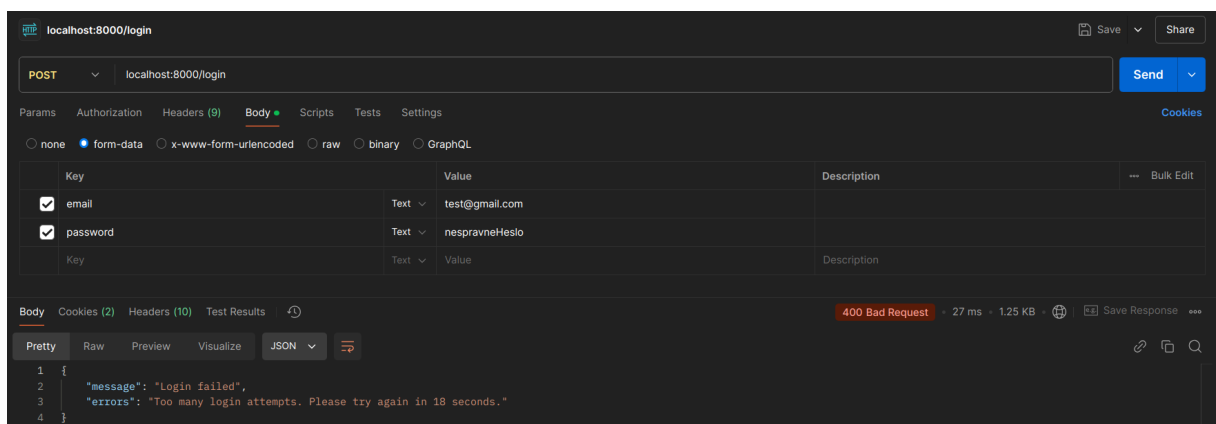
```
1 $user = User::create([
2     'first_name' => $request->first_name,
3     'last_name' => $request->last_name,
4     'street' => $request->street,
5     ...
6     'password' => Hash::make($request->string('password')),
7 ]);
```

| | id | first_name | last_name | username | email | password | city | zip | country | |
|---|----|------------|-----------|----------------|-------|------------------|-----------------------------------|------------|-------------|-----------|
| 1 | 1 | Test... | Maye... | 746 Rodrigu... | 265 | test@example.com | \$2y\$12\$02Cri6EuUiFmUdYLSpbk... | Kleinmouth | 55619-00... | Belarus |
| 2 | 2 | Test | Test | Ilkovicova | 11B | test@gmail.com | \$2y\$12\$E490n2N6QeNqhpJYjzx3... | Bratislava | 82103 | Slovensko |
| 3 | 3 | Test | Test | Ilkovicova | 11B | test2@gmail.com | \$2y\$12\$.ZWQCyC/B7oZoghdEHnv... | Bratislava | 82103 | Slovensko |
| 4 | 4 | Test | Test | Ilkovicova | 11B | test3@gmail.com | \$2y\$12\$3.Azsbk9j7qosgcdWMq5... | Bratislava | 82103 | Slovensko |
| 5 | 5 | Test | Test | Ilkovicova | 11B | test4@gmail.com | \$2y\$12\$jnt9XAVN4SL0p8W.pS6H... | Bratislava | 82103 | Slovensko |

Obr. 8: Screenshot tabuľky 'users'

3.3.5 Ochrana voči bruteforce útokom

Ako ochranu voči bruteforce útokom využívame rate limiting metódu (*app/Http/Requests/Auth/LoginRequest.php*).



Obr. 9: Príklad rate limitingu

3.3.6 Ochrana voči SQL injection

Ochranu proti SQL útokom zabezpečuje nástroj Eloquent, ktorý je bežnou súčasťou Laravel frameworku. Eloquent využíva prepared statements a parameterized queries pri práci s databázou. To znamená, že všetky vstupy používateľa sú automaticky escapované (spracované tak, aby sa neinterpretovali ako SQL kód) pred ich vložením do SQL dotazu.[1].

3.3.7 Validácia

Validáciu dát bolo potrebné implementovať v prípadoch, ako je práca s formulármi resp. v prípade, že používateľ poskytol dáta, ktoré bolo pred spracovaním treba overiť.

Príkladom je prípad, keď si používateľ chce zmeniť informácie vo svojom profile, ako je napríklad heslo. Kontroluje sa, či používateľ zadal správne aktuálne heslo a či sa nové heslá zhodujú (nové heslo treba zadať vo formulári 2x).

Ďalším prípadom je registrácia nového používateľa. Validujú sa poskytnuté dáta z registračného formulára a či sú vyplnené povinné polia. Na validáciu hesla a emailovej adresy sme použili built-in Laravel metódy.

Pre validáciu vytvorenej objednávky sme si okrem použitia build-in Laravel metód

vytvorili aj vlastnú class-u, ktorá kontroluje, či sú všetky produkty z objednávky validnými produktami, či sa cena produktu na objednávke zhoduje s cenou produktu v databáze, či je dostatok produktu na sklade podľa množstva daného produktu v objednávke a takisto, či výsledná cena (súčet cien produktov v objednávke) sedí s reálnym súčtom cien.

```
1 class ValidOrder implements ValidationRule
2 {
3     protected float $totalPrice;
4     public function __construct($totalPrice)
5     {
6         $this->totalPrice = $totalPrice;
7     }
8     /**
9      * Determine if the validation rule passes.
10     * Is used for order validation.
11     *
12     * @param string $attribute
13     * @param mixed $value
14     * @param Closure $fail
15     * @return void
16     * @throws ValidationException
17     */
18     public function validate(string $attribute, mixed $value, Closure $fail):
19         void
20     {
21         foreach ($value as $product) {
22             if (!isset($product['id'])) {
23                 throw ValidationException::withMessages([
24                     $attribute => 'Product ID is required.'
25                 ]->status(409);
26             }
27             $productId = $product['id'];
28             if (!Product::find($productId)) {
29                 throw ValidationException::withMessages([
30                     $attribute => 'The product does not exist.'
31                 ]->status(404);
32             }
33             $productModel = Product::find($productId);
```

```

33         if ($productModel->price != $product['price']) {
34             throw ValidationException::withMessages([
35                 $attribute => 'The price of the product does not match the
36                     price in the database.'
37             ]->status(409);
38         }
39         if ($productModel->stock < $product['quantity']) {
40             throw ValidationException::withMessages([
41                 $attribute => 'The quantity of the product is not
42                     sufficient.'
43             ]->status(409);
44         }
45     }
46     $this->validateTotalPrice($value);
47 }
48 /**
49  * Validate the total price of the order.
50  * Sum of the prices of the products in the order must match the total
51  * price.
52  *
53  * @param array $value
54  * @return void
55  * @throws ValidationException
56  */
57 protected function validateTotalPrice(array $value): void
58 {
59     $sumOfPrices = array_reduce($value, function ($sum, $product) {
60         return $sum + ($product['price'] * $product['quantity']);
61     }, 0);
62     if ($sumOfPrices != $this->totalPrice) {
63         throw ValidationException::withMessages([
64             'total_price' => 'The total price does not match the sum of the
65                 product prices.'
66         ]->status(409);
67     }
68 }
69 }
70 }

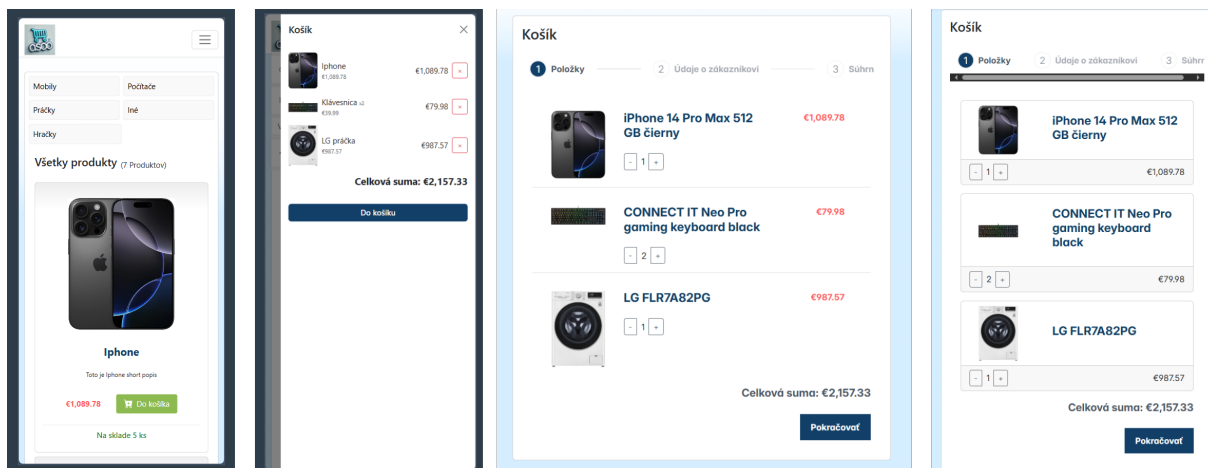
```

4 Frontend

Pre vývoj frontendu našej aplikácie bol použitý framework React s použitím TypeScriptu. Projekt bol vytvorený na základe šablóny Vite, ktorá poskytuje vývojové prostredie optimalizované pre najmodernejšie aplikácie. Aplikácia je kompletne vybavená routing rozhraním pomocou knižnice ‘react-router-dom’, ktorá poskytuje mnohé komponenty pre routing moderných aplikácií. Aplikácia je taktiež plne dockerizovaná.

4.1 Responzívny dizajn

Na štylizáciu našej aplikácie sme použili viaceré populárne knižnice ako ‘react-bootstrap’, ‘@mui/material’ alebo ‘primereact’. To nám poskytlo využitie pokročilých komponentov a tiež plne responzívny dizajn aplikácie, ktorý sme si podľa potrieb a rozloženia komponentov na jednotlivých page-och vedeli customizovať. Názornú ukážku môžete vidieť na obrázku 10. Na obrázku môžeme vidieť, že dizajn je optimalizovaný predovšetkým pre mobilné zariadenia a teda aplikácia je plne responzívna. Ako konkrétny príklad responzivnosti môžeme ešte uviesť domovskú stránku aplikácie so všetkými produktmi. Stránka obsahuje bočný panel s kategóriami a samotné produkty. Pre menšie veľkosti obrazovky sa bočný panel zobrazí nad produktmi a produkty sa zobrazia jednotlivo pod sebou. Pre veľké obrazovky sa vedľa seba zobrazia až 4 produkty, navyše s bočným panelom vľavo.



Obr. 10: Ukážka responzívneho dizajnu

4.2 User Experience

Aplikácia poskytuje základné funkcionality bežného e-shopu. Používateľ si môže prezerať všetky produkty, ktoré máme v ponuke. Produkty si tiež môže pozerať podľa dostupných kategórií, prípadne vyhľadať podľa kľúčového slova. Každý produkt má aj svoju samostatnú

stránku s detailným popisom. Používateľ si môže produkty vkladať do košíku, následne ľubovoľne pridávať, resp. uberať množstvo a neskôr aj vykonať objednávku.

Aplikácia ponúka možnosť registrácie. Ak už je používateľ registrovaný, môže sa do svojho účtu prihlásiť. Prihlásený používateľ si vie v aplikácii pozrieť históriu svojich objednávok, pričom ku každej nájde dátum vytvorenia, číslo objednávky, celkovú sumu a jednotlivé produkty objednávky.

Okrem toho, prihlásený používateľ si môže vo svojom profile zmeniť osobné údaje (okrem emailovej adresy), ktorými sa registroval. Taktiež má možnosť zmeny hesla, pričom však musí poznať svoje aktuálne heslo.

Neprihlásený používateľ nemá v aplikácii možnosť vidieť históriu svojich objednávok a ani nemá prístup k zmene údajov či zmene hesla.

Každý používateľ má možnosť vytvoriť objednávku z našich tovarov. Ikona košíku sa nachádza v hornom menu a zobrazí sa po pridaní aspoň jedného tovaru do košíku. V prvom kroku procesu objednávky je používateľovi prehľadne zobrazovaný obsah košíku aj s celkovou sumou. V druhom kroku používateľ zadá svoje údaje. Ak je používateľ prihlásený, tak sa mu tieto údaje automaticky predvyplnia, môže si ich však ľubovoľne zmeniť. Ak používateľ nie je prihlásený, tak musí tieto údaje vyplniť. V treťom kroku si používateľ môže pozrieť súhrn objednávky, teda všetky produkty, ich množstvá a celkovú sumu. Okrem toho si tiež môže skontrolovať svoje údaje. Ak je spokojný, môže objednávku odoslať. Ak nie, môže sa vrátiť na niektorý z predošlých krokov a upraviť čo potrebuje. Po odoslaní objednávky sa zobrazí hláška o odoslaní objednávky a následne je používateľ presmerovaný na domovskú stránku.

Aplikáciu sme sa snažili urobiť čo najviac používateľsky jednoduchú a intuitívnu. Používateľ má dostupné všetky stránky už priamo z domovskej obrazovky aplikácie. Používateľ má niekoľko možností ako pridávať, upravovať a odoberať produkty z košíku, pričom všetky sú spolu prepojené. Ak je to možné, používateľ má predvyplnené údaje (napr. v košíku alebo v profile), čo zlepšuje UX. Taktiež používame tzv. Protected Routes pre blokovanie neautorizovaného prístupu (napr. neprihlásený používateľ si chce pozrieť profil). V neposlednom rade sa aplikácia snaží informovať používateľa o výsledkoch jednotlivých operácií. Či už je to zmena údajov, odoslanie objednávky, nevyplnené polia, príp. informovanie používateľa, že ešte nemá žiadne objednávky pri prehliadaní histórie objednávok.

4.3 State Management a Interakcia

Na spravovanie stavov v našej aplikácii sme použili Context API. V podstate ide o akýsi parent komponent, ktorý poskytuje jednoduché a efektívne rozhranie na zdieľanie a prácu

s údajmi vo všetkých child komponentoch. V ukážke 1 si môžeme všimnúť, že prakticky celú aplikáciu zaobalujeme do context provider-ov, čím sa každému child komponentu pri požiadaní sprístupnia údaje a operácie nad týmito údajmi.

```
1 return (  
2   <>  
3     <ShoppingCartProvider>  
4       <AuthProvider>  
5         <MainLayout>  
6           <Routes>  
7             <Route path="/" element={<Home />} />  
8             <Route path="/register" element={<Register />}/>  
9             <Route path="/login" element={<Login />} />  
10            ...  
11            <Route path="/orders" element={  
12              <ProtectedRoute>  
13                <UserOrders />  
14              </ProtectedRoute>  
15            } />  
16            <Route path="/cart" element={<Cart />} />  
17          </Routes>  
18        </MainLayout>  
19      </AuthProvider>  
20    </ShoppingCartProvider>  
21  </>  
22 );
```

Listing 1: Ukážka kódu volania context providera

Na uchovávanie údajov pre náš context používame local storage. Pomocou React hook-u ‘useContext‘ si v akomkoľvek child komponente požiadame o čítanie a sledovanie daného contextu. To pomáha udržiavať logiku na jednom mieste, jednoduchší a efektívnejší kód a zdieľať údaje medzi komponentmi.

Ako príklad si vezmime ‘ShoppingCartContext‘. Tento context poskytuje používateľovi údaje o jeho košíku, tj. produkty a ich počet, ktorý do košíka vložil. Context poskytuje navyše metódy ako ‘getItemQuantity‘, ‘increaseCartQuantity‘, ‘decreaseCartQuantity‘, ‘isCartEmpty‘, ‘openCart‘ a ďalšie, ktorými vieme spravovať údaje v košíku, V ukážke 2

môžeme vidieť ako sú tieto metódy priamo odovzdané do provider-a čím umožní prístup z nižších leveloch. Pokiaľ teda v jednom komponente spravujeme stav košíka, to sa automaticky aktualizuje pre všetky child komponenty operujúce nad košíkom.

```
1 return (
2     <>
3         <ShoppingCartContext.Provider
4             value={{
5                 openCart,
6                 closeCart,
7                 getItemQuantity,
8                 increaseCartQuantity,
9                 decreaseCartQuantity,
10                removeFromCart,
11                isCartEmpty,
12                clearCart,
13                cartQuantity,
14                cartItems
15            }}
16        >
17            {children}
18            <ShoppingCart isOpen={isOpen} />
19        </ShoppingCartContext.Provider>
20    </>
21 )
```

Listing 2: Ukážka kódu definovania context providera

Okrem košíka využívame context na veľmi dôležitú časť a tou je autentifikácia. Pomocou contextu udržiavame stav autentifikácie používateľa čím vieme obmedziť prístup v určitých častiach aplikácie. Kompletnú implementáciu kontextov nájdete v repozitári Front-end v priečinku ‘contexts’.

4.4 Integrácia s API

Ako sme už viackrát spomenuli, frontend aj backend boli vyvíjané ako samostatné časti, čiže bola potrebná ich vzájomná integrácia. Komunikácia medzi frontendom a backendom prebieha pomocou REST služieb a na prenos dát sa používa formát JSON. Na simuláciu spojenia s backendom, resp. na testovanie sme tiež vytvorili mock službu, v ktorej používame

lokálne vytvorené dáta v rovnakom formáte, ako ich očakávame z backendu.

Celá API logika sa nachádza v priečinku ‘services’. Tento priečinok obsahuje data transfer objects (DTO), ktoré definujú štruktúry jednotlivých objektov, ktoré sa budú prenášať medzi frontendom a backendom. DTO zvyšujú prehľadnosť kódu.

Okrem toho sa tu nachádza interface ‘IApiService’, ktorý definuje všetky metódy, ktoré sa môžu využívať na komunikáciu medzi frontendom a backendom. Následne triedy ‘ApiService’ a ‘Mockservice’ už tieto metódy konkrétne implementujú. Prvý menovaný nasleduje návrhový vzor Singleton a slúži na komunikáciu s reálnou backendovou časťou aplikácie, ktorá beží na inom porte (v našom prípade 83). Komunikácia prebieha pomocou ‘fetch’, kde sa ďalej špecifikujú API request metódy, hlavičky, prípadne telo. Druhý menovaný slúži na získanie dát z lokálne vytvorených objektov.

Takto vytvorený interface s jeho implementáciami spôsobuje to, že dokážeme efektívne prepínať medzi rôznymi zdrojmi dát bez potreby zmeny kódu. To uľahčuje ďalší vývoj a tiež testovanie.

Na výber zdroja dát slúži ‘ServiceSelector’. Je to jednoduchá premenná, ktorá je typu ‘IApiService’ a môžeme jej priradiť hodnotu buď ‘MockService’ alebo ‘ApiService’. Túto jednu premennú potom používame v celej aplikácii na rozhodovanie, z ktorého zdroja dát sa bude čerpať. Tento spôsob centralizuje celý výber do jedného miesta a teda pri zmene nie je potrebné meniť kód na rôznych miestach aplikácie.

5 Best Practices and Code Quality

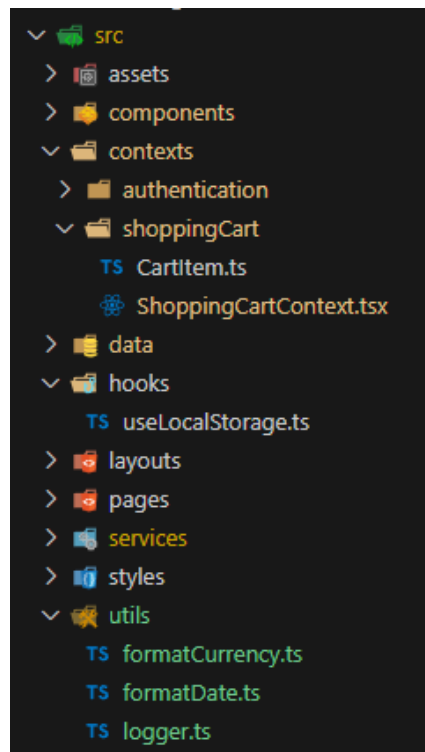
5.1 Code Structure and Readability

Pri návrhu štruktúry frontendovej časti aplikácie sme sa držali všeobecnej architektúry a pravidiel pre React TypeScript aplikáciu:

- Prvý level štruktúry obsahuje priečinky podľa typu súborov, tzn. developer okamžite bude vedieť, kde hľadať komponenty, spomínané kontexty alebo jednotlivé stránky (pages).
- Druhý level umožňuje zoskupenie vlastností pre konkrétnu inštanciu typu, napr. pre konkrétny komponent zoskupuje všetky súbory, ktoré sú s nim spojené, tj. tsx, test, css, ...

Na obrázku 11 môžeme vidieť názornú ukážku štruktúry priečinkov. Týkajúc sa samotného kódu, dodržali sme základy SOLID princípov a teda každá vytvorená trieda má jeden zámer a kód v nej je čitateľný, napr. ‘ShoppingCartContext’ sa stará o logiku košíka, ale logika ukladania dát do local storage je oddelená v ‘useLocalStorage’ . Taktiež sme rozdelili funkcionality tak, aby naše komponenty neboli príliš veľké a chaotické.

Okrem toho sme pre prehľadnosť kódu oddeľovali CSS štýly jednotlivých elementov do samostatných súborov.



Obr. 11: Ukážka štruktúry projektu

Na strane back-end-u sme sa držali MVC architektúry, čo umožnilo udržiavať prehľadnú štruktúru kódu a súborov. Brali sme do úvahy aj modularitu kódu, preto sme sa rozhodli pre vytvorenie vlastnej validačnej class-y a middleware-u na autentifikáciu, ako samostatných súborov. Jednak sme predišli duplicitu kódu a jednak sme z nich vytvorili znovu-použiteľné moduly.

5.2 Error Handling and Logging

Laravel poskytuje robustný systém logovania, ktorý umožňuje zaznamenávať rôzne typy správ, ako sú informácie o priebehu, varovania a chyby. Pomocou tejto technológie zapisujeme hlásenia z BE do externého súboru *laravel.log*.

```

[2024-12-07 12:15:37] local.INFO: Fetching all categories
[2024-12-07 12:15:37] local.INFO: Categories fetched successfully {"categories_count":6}
[2024-12-07 12:16:05] local.INFO: Fetching products {"search":null}
[2024-12-07 12:16:05] local.INFO: Products fetched successfully {"products_count":25}
[2024-12-07 12:16:12] local.INFO: Fetching products by category {"category_id":"3"}
[2024-12-07 12:16:12] local.INFO: Products fetched successfully by category {"category_id":"3","products_count":3}
[2024-12-07 12:16:19] local.INFO: Fetching product by ID {"product_id":"300"}
[2024-12-07 12:16:19] local.WARNING: Product not found {"product_id":"300"}
[2024-12-07 12:18:28] local.INFO: Updating authenticated user {"user_id":1}
[2024-12-07 12:18:28] local.INFO: User updated successfully {"user_id":1}
[2024-12-07 12:18:47] local.INFO: Updating password for authenticated user {"user_id":1}
[2024-12-07 12:18:47] local.INFO: Password validation successful {"user_id":1}
[2024-12-07 12:18:48] local.WARNING: Old password does not match {"user_id":1}
[2024-12-07 12:19:03] local.INFO: Fetching orders for user {"user_id":1}
[2024-12-07 12:19:03] local.INFO: Orders fetched successfully {"orders_count":0}
[2024-12-07 12:19:16] local.INFO: Placing an order {"user_id":1}
[2024-12-07 12:19:16] local.INFO: Order data validated successfully {"data":{"customer":{"first_name":"Jožko","last_name":"Mrkvička","street":"Ilkovičova","house_number":"118","city":"Bratislava"},
[2024-12-07 12:19:16] local.INFO: Order created successfully {"order_id":1}
[2024-12-07 12:19:16] local.INFO: Products attached to order {"order_id":1,"products":[{"id":1,"quantity":1,"price":1099.99}]}
[2024-12-07 12:19:27] local.INFO: Placing an order {"user_id":1}
[2024-12-07 12:19:27] local.ERROR: Order validation failed {"error":"The quantity of the product is not sufficient."}

```

Obr. 12: Ukážka *laravel.log* súboru

Pre FE sme zadefinovali vlastnú implementáciu loggeru pomocou knižnice ‘loglevel’. Vďaka nej vieme registrovať typy správ ako info, error, debug alebo warn. Výpisy sa ukladajú do local storage-u pod názvom ‘appLogs’, príklad môžeme vidieť na obrázku 13. Správy jednotlivých hlásení majú zaužívaný formát, ktorý je customizovateľný.

```

▶ 60: {message: "[2024-12-07T08:17:27.866Z] ERROR: Failed to fetch", timestamp: "2024-12-07T08:17:27.866Z"}
▶ 61: {message: "[2024-12-07T08:17:27.867Z] ERROR: [2024-12-07T08:17:27.866Z] ERROR: Failed to fetch",...}
▶ 62: {message: "[2024-12-07T08:17:27.883Z] ERROR: TypeError: Failed to fetch",...}
▶ 63: {...}
▶ 64: {message: "[2024-12-07T08:17:27.883Z] ERROR: Failed to fetch", timestamp: "2024-12-07T08:17:27.883Z"}
▶ 65: {message: "[2024-12-07T08:17:27.883Z] ERROR: [2024-12-07T08:17:27.883Z] ERROR: Failed to fetch",...}
▶ 66: {message: "[2024-12-07T08:17:27.884Z] ERROR: TypeError: Failed to fetch",...}
▶ 67: {...}
▶ 68: {message: "[2024-12-07T08:17:27.885Z] ERROR: Failed to fetch", timestamp: "2024-12-07T08:17:27.885Z"}
▶ 69: {message: "[2024-12-07T08:17:27.885Z] ERROR: [2024-12-07T08:17:27.885Z] ERROR: Failed to fetch",...}
▶ 70: {message: "[2024-12-07T08:33:28.544Z] ERROR: useAuth must be used within an AuthProvider",...}
▶ 71: {...}
▶ 72: {message: "[2024-12-07T08:33:28.560Z] ERROR: useAuth must be used within an AuthProvider",...}
▶ 73: {...}
▶ 74: {message: "[2024-12-07T08:33:28.570Z] ERROR: useAuth must be used within an AuthProvider",...}
▶ 75: {...}
▶ 76: {message: "[2024-12-07T08:33:28.574Z] ERROR: useAuth must be used within an AuthProvider",...}
▶ 77: {...}
▶ 78: {message: "[2024-12-07T08:56:51.278Z] INFO: Product with id 1 has been added to Cart.",...}
▶ 79: {...}
▶ 80: {message: "[2024-12-07T09:00:34.635Z] INFO: Product's amount with id 1 has been decreased by 1.",...}
▶ 81: {...}
▶ 82: {message: "[2024-12-07T09:00:35.282Z] INFO: Product with id 1 has been added to Cart.",...}
▶ 83: {...}
▶ 84: {message: "[2024-12-07T09:00:45.568Z] INFO: Product with id 2 has been added to Cart.",...}
▶ 85: {...}
▶ 86: {message: "[2024-12-07T09:00:46.195Z] INFO: Product's amount with id 2 has been increased by 1.",...}

```

Obr. 13: Ukážka výpisu logov z local storage

6 Testovanie

Plnohodnotné výsledky testov sú dostupné v zvlášť vytvorenom repozitári *ASOS-Semestrálny projekt - Attachments*. Implementované testy na back-ende sú dostupné v priečinku */tests* v "Back-end" repozitári.

6.1 Unit Testy

Na vytvorenie Unit a Integračných testov na back-ende sme využili technológiu PHPUnit. Jednalo sa o testy, ktoré kontrolovali správnu funkcionálnu napr. vytvárania tried.

```
1 public function test_create_order_mock(): void {  
2     $order = $this->createMock(Order::class);  
3     $order->method('save')  
4         ->willReturn(true);  
5     $this->assertTrue($order->save());  
6 }
```

6.2 Integration Testy

Rolu integračných testov v Laravel projekte zastávali tzv. Feature testy. Tieto testy kontrolovali správnu integráciu medzi viacerými časťami projektu.

```
1 public function test_get_product_list_by_category_with_items_success() {  
2     $category = Category::factory()->create();  
3     $product = Product::factory()->create(['category_id' => $category->id]);  
4     $product2 = Product::factory()->create(['category_id' => $category->id]);  
5     $uri = '/products/' . $category->id;  
6     $response = $this->getJson($uri);  
7     $response->assertStatus(200);  
8     $response->assertJsonCount(2);  
9     $response->assertJson([$product->toArray(), $product2->toArray()]);  
10 }
```

Nato aby vyššie spomenutý test prešiel musel splniť všetky požiadavky. Po vytvorení "dummy" hodnôt sa zavolała požiadavka na získanie produktov v danej kategórii. V tomto teste sa kontroluje najmä schopnosť komunikácie s Laravel aplikáciou prostredníctvom REST API.

```

1 public function test_create_category(): void {
2     $category = Category::factory()->create(['name' => 'Electronics']);
3     $this->assertInstanceOf(Category::class, $category);
4     $this->assertEquals('Electronics', $category->name);
5     $this->assertDatabaseHas('categories', ['name' => 'Electronics']);
6 }

```

Medzi integračné testy vieme zaradiť aj "jednoduchšie" testy, ktoré majú za úlohu kontrolovať integrácie medzi modelom a databázou tým, že overujú prítomnosť vytvoreného záznamu priamo v databáze. Tento konkrétny test teda kontroluje, či Eloquent dokáže úspešne vytvoriť záznam v databáze a že medzi modelom a databázou funguje správna integrácia[1].

```

PASS Tests\Feature\Api\UserTest
✓ get user success                                0.01s
✓ get user not logged in error                    0.01s

PASS Tests\Feature\Auth\AuthRoutesTest
✓ register user success                            0.01s
✓ register user weak password error                0.01s
✓ register user email exists error                0.01s
✓ register user missing required field error      0.01s
✓ login user success                              0.01s
✓ login user wrong password error                 0.21s
✓ login user wrong email error                   0.23s
✓ login user already logged in error              0.02s
✓ logout user success                             0.01s
✓ logout user not logged in error                 0.01s

Tests:      40 passed (65 assertions)
Duration: 1.11s

```

Obr. 14: Screenshot časti výsledkov testov

6.3 Load Testy

```
1 config:
2   target: 'http://localhost:8000'
3   plugins:
4     fake-data: {}
5     apdex: {}
6     metrics-by-endpoint: {}
7   apdex:
8     threshold: 100
9   phases:
10    - duration: 60
11      arrivalRate: 1
12      rampTo: 5
13      name: Warm up phase
14    - duration: 30
15      arrivalRate: 10
16      rampTo: 30
17      name: Spike phase
18 scenarios:
19   - flow:
20     - post:
21       url: "/register"
22       json:
23         first_name: "Test"
24         last_name: "Name"
25         street: "Street 14"
26         ...
27         email: "{{ $randEmail() }}"
28         password: "password"
29     - get:
30       url: "/logout"
31   - flow:
32     - get:
33       url: "/logout"
```

Load testy boli vytvorené za pomoci technológie *Artillery*. Vyššie zobrazený kód

reprezentuje zjednodušenú verziu reálneho naimplementovaného testu. Scenarios respektíve flow-y predstavujú scenáre, ktoré každý virtuálny user náhodne vykoná (vyberie si jeden z možných flow-ov). Phases predstavuje "fázy", ktoré sa postupne zapínajú. Sú definované časom, počiatočnou a konečnou hodnotou virtuálnych user-ov.

All VUs finished. Total time: 2 minutes, 32 seconds

Summary report @ 17:50:12(+0100)

| | |
|------------------------|---------|
| apdex.frustrated: | 0 |
| apdex.satisfied: | 3111 |
| apdex.tolerated: | 329 |
| http.codes.200: | 1254 |
| http.codes.201: | 329 |
| http.codes.401: | 658 |
| http.codes.404: | 596 |
| http.codes.409: | 603 |
| http.downloaded_bytes: | 1019263 |
| http.request_rate: | 34/sec |
| http.requests: | 3440 |
| http.response_time: | |
| min: | 5 |
| max: | 233 |
| mean: | 27.9 |
| median: | 7 |
| p95: | 206.5 |
| p99: | 214.9 |

Obr. 15: Screenshot časti výsledku load testu

Výsledkom spusteného testu je finálny report, ktorý obsahuje informácie ako napríklad počet request-ov za sekundu počas testovania alebo taktiež štatistiku časov odpovedí. Vďaka využitému pluginu *apdex* taktiež máme aj jednoducho pochopiteľnú skórovaciu hodnotu "*Apdex score: 0.9521802325581395 (excellent)*".

7 Continous Integration

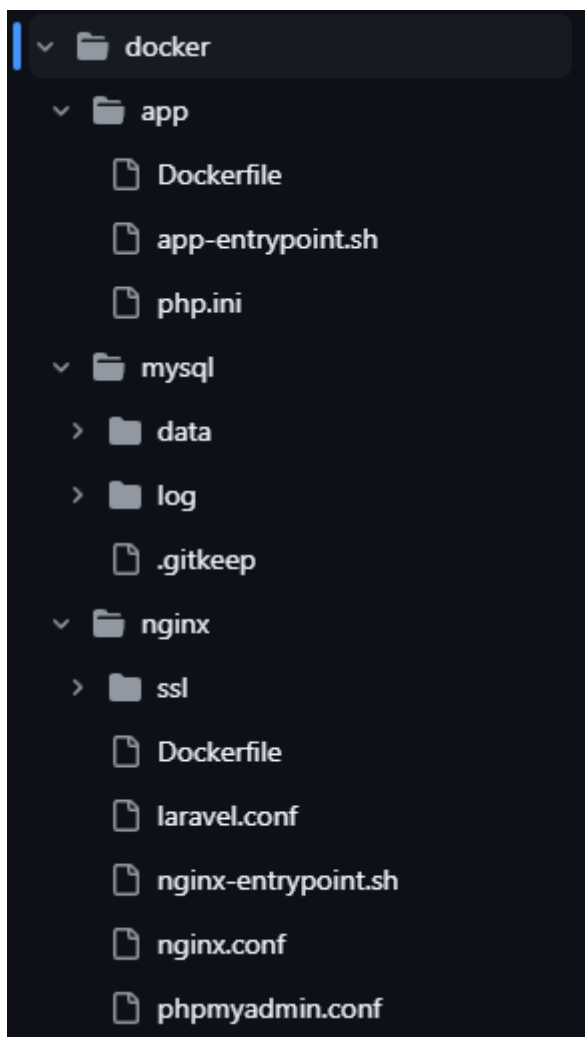
7.1 Kontajnerizácia

Obe časti aplikácie sú kontajnerizované v Dockeri, čo uľahčuje ich nasadenie bez ohľadu na prostredie. Pre prehľadnosť sme sa rozhodli, že obe časti pripravíme v samostatných kontajneroch. Jednak tým udržujeme oddelenosť dátovej od prezentačnej vrstvy, ale zároveň umožňujeme používateľovi si jednoduchšie zvoliť, ktorú časť aplikácie si chce spustiť, nakoľko obe sú samostatné a dokážu pracovať aj nezávisle od seba. Návody na spustenie nájdete v súboroch README.md jednotlivých repozitárov.

Z technických detailov, frontend beží v developer móde, čo je štandard pre React v Dockeri. Používa port 3000, čo je taktiež štandard pre React.

Backend kontajner sa skladá z troch častí. Je to jednak databáza MySQL, ktorá beží na porte 3306. Ďalej samotný server Nginx, ktorý je centrom celej backendovej časti aplikácie, pracuje na porte 83. Navyše sme ešte pridali aj nástroj phpMyAdmin na port 8080, ktorý umožňuje správu MySql databázy priamo vo webovom prehliadači.

Takto vyzerá štruktúra dockeru pre backend. Obsahuje samostatné dockerfiles pre aplikáciu aj nginx s príslušnými entrypoint.sh skriptami pre dodatočne nastavovania.



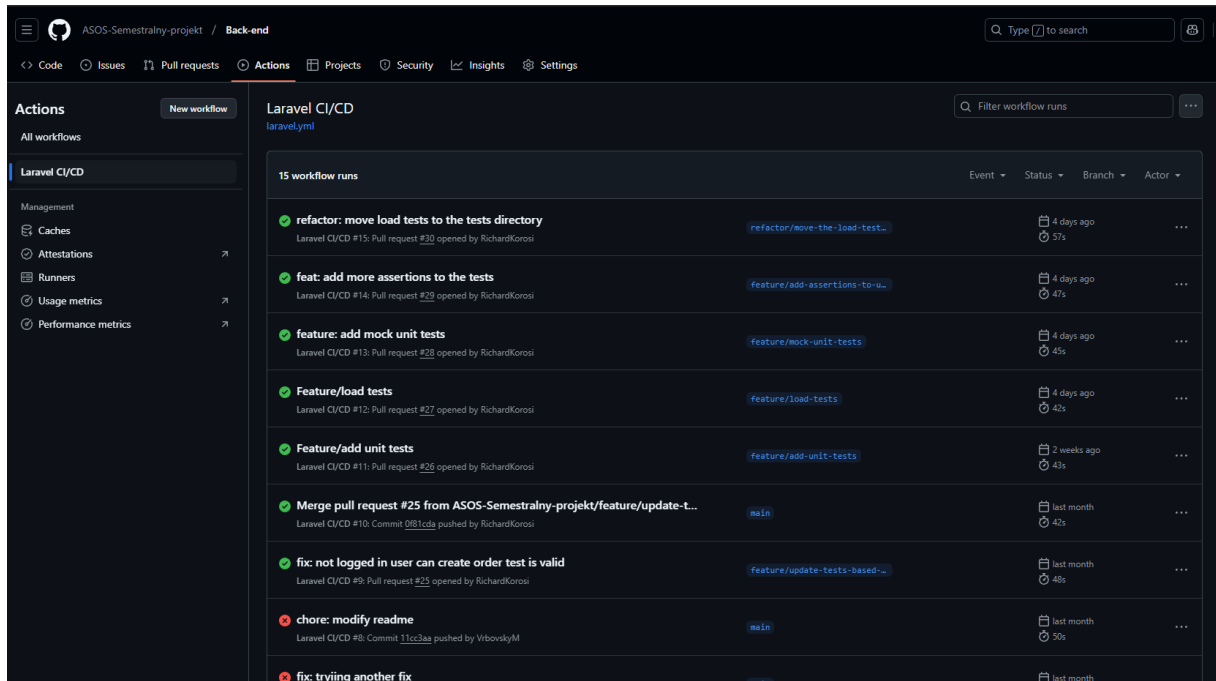
Obr. 16: Screenshot docker štruktúry pre backend

7.2 Github CI

Pre plynulosť developmentu bolo využité rôzne github tools.

7.2.1 Github actions

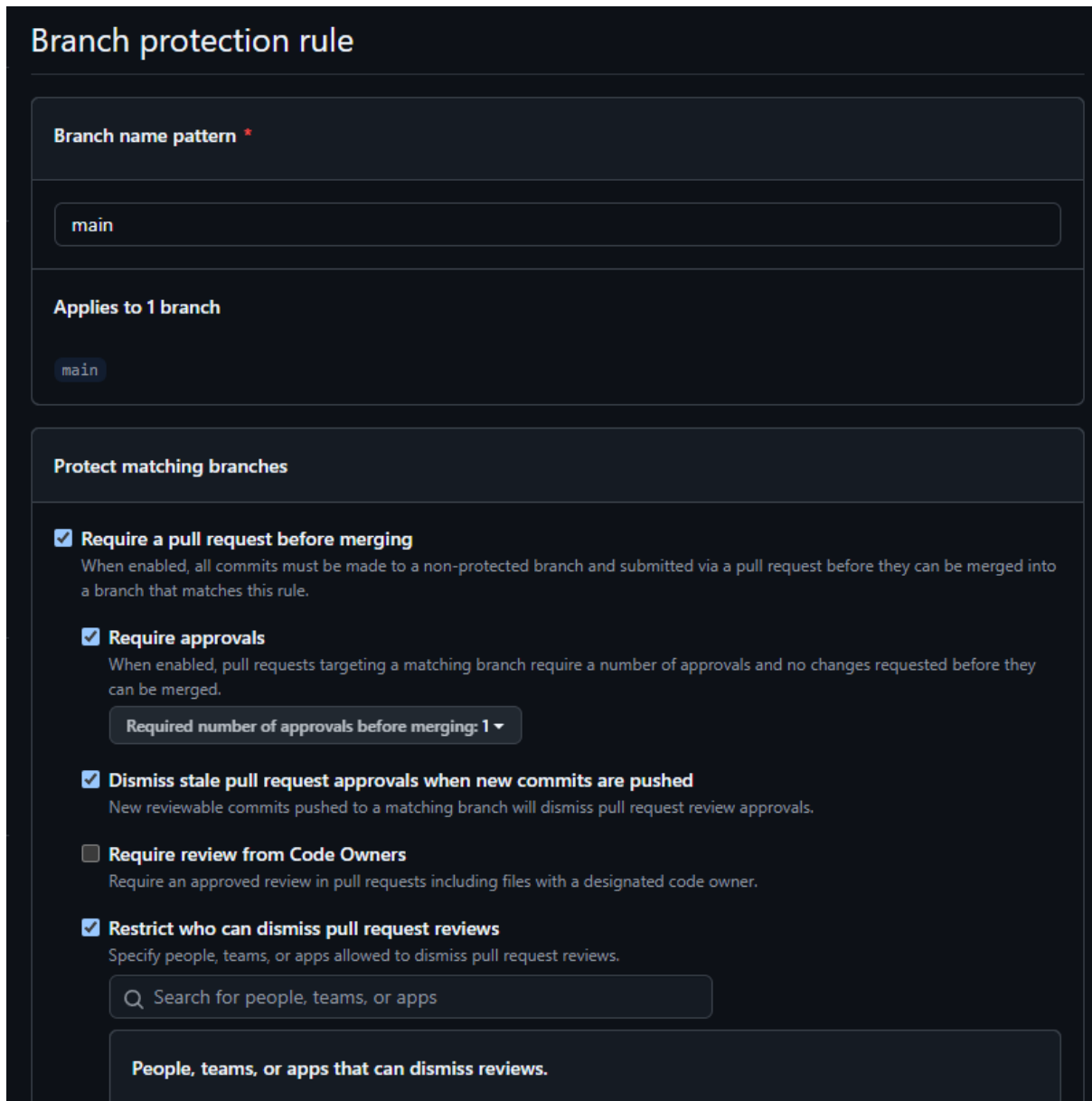
Bola vytvorená github action ktorá automaticky spúšťa testy pri vytvorení pull requestu.



Obr. 17: Screenshot github workflows

7.2.2 Pravidla na branche

Bolo vytvorene pravidlo ktoré nútilo vytvoriť pull request a taktiež review pred mergovaním do mainu.

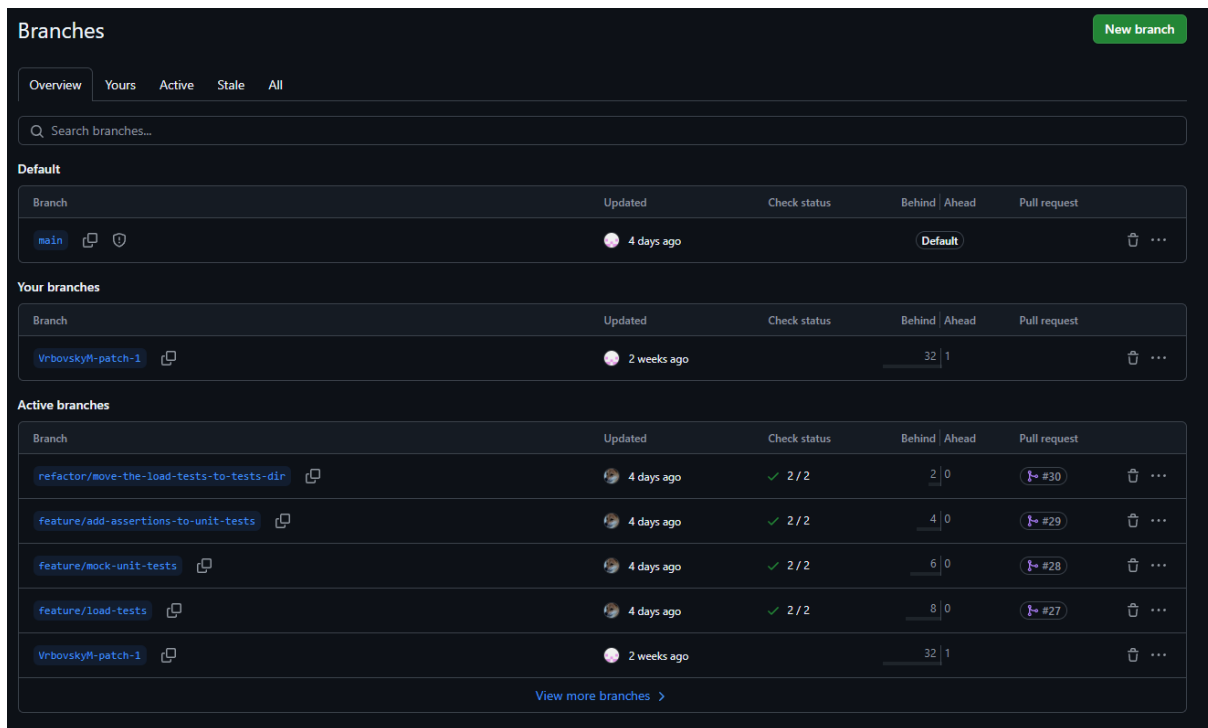


The screenshot shows the 'Branch protection rule' configuration page in GitHub. The interface is dark-themed. At the top, the title 'Branch protection rule' is displayed. Below it, there are several sections: 1. 'Branch name pattern *' with a text input field containing 'main'. 2. 'Applies to 1 branch' with a list of branches containing 'main'. 3. 'Protect matching branches' which contains several checkboxes: - 'Require a pull request before merging' (checked), with a description: 'When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.' - 'Require approvals' (checked), with a description: 'When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.' Below this is a dropdown menu showing 'Required number of approvals before merging: 1'. - 'Dismiss stale pull request approvals when new commits are pushed' (checked), with a description: 'New reviewable commits pushed to a matching branch will dismiss pull request review approvals.' - 'Require review from Code Owners' (unchecked), with a description: 'Require an approved review in pull requests including files with a designated code owner.' - 'Restrict who can dismiss pull request reviews' (checked), with a description: 'Specify people, teams, or apps allowed to dismiss pull request reviews.' Below this is a search input field with the placeholder text 'Search for people, teams, or apps'. At the bottom, there is a section titled 'People, teams, or apps that can dismiss reviews.' which is currently empty.

Obr. 18: Screenshot pravidiel na branche

7.2.3 Branch strategija

Ako branchovú stratégiu sme použili Github Flow, ktorý funguje cez main/master branchu a feature branchu ktoré sa následne merguju do main branchu.



Obr. 19: Screenshot branchovej strategije

Zoznam použitej literatúry

1. OPENAI. *ChatGPT*. 2024. Generovanie a úprava textu do dokumentácie.

Prílohy