Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Webová aplikácia na efektívne plánovanie a správu vetiev v nástroji Jenkins

Systémová dokumentácia

2024 **Roman Yehorov**

Obsah

1 Úvod

Táto dokumentácia popisuje implementačné detaily webovej aplikácie vytvorenej v rámci diplomovej práce. Cieľom projektu je zjednodušiť správu vývojových a testovacích vetiev v prostredí Jenkins, a zároveň umožniť pohodlné plánovanie ich spúšťania prostredníctvom používateľsky prívetivého rozhrania.

Aplikácia je rozdelená na backendovú a frontendovú časť. Backend bol implementovaný v jazyku Java s využitím frameworku Spring Boot, pričom na perzistenciu dát sa používa relačná databáza PostgreSQL. Komunikácia s Jenkinsom prebieha pomocou REST API. Frontend je vytvorený pomocou frameworku React s využitím TypeScriptu, Vite.js a moderných webových technológií na tvorbu responzívneho používateľského rozhrania.

V nasledujúcich kapitolách sú podrobne zdokumentované jednotlivé triedy backendu, ich funkcie, vlastnosti, ako aj spôsob ich vzájomnej interakcie. Dokumentácia slúži ako technický sprievodca pre vývojárov, ktorí by chceli systém ďalej rozvíjať alebo upravovať.

2 Spustenie programu

Pred začiatkom je potrebne nainštalovať ďalšie inštrumenty a knižnice:

- NPM (Node Package Manager) – `https://nodejs.org/en/download/`
- Maven – `https://maven.apache.org/download.cgi`
- Jenkins – `https://www.jenkins.io/download/`
- PostgreSQL – `https://www.postgresql.org/download/`
- Vytvoriť PostgreSQL databázu.

2.1 Lokálne spustenie

- Stiahnuť projekt z git repozitára.
- V priečinku `/backend/` spustiť backendovú časť pomocou príkazu **mvn clean install && mvn spring-boot:run**.
- Backend bude bežať na adrese `http://localhost:8080`. Toto je možné zmeniť v priečinku `/backend/src/main/resources/` v súbore application-properties.xml.
- V priečinku `/frontend/` spustiť frontendovú časť pomocou príkazu **npm install && npm run dev**.
- Frontend bude bežať na adrese `http://localhost:5173`.

3 Backend

Tato časť dokumentácie predstavuje podrobný popis architektúry backendu, ktorý aktívne využíva Java, Spring Boot a PostgreSQL.

3.1 Závislosti

- **spring-boot-starter** – základná závislosť pre každú aplikáciu postavenú na Spring Boot, poskytuje základnú konfiguráciu a runtime podporu.
- **spring-boot-starter-web** – umožňuje vytvárať REST API a webové aplikácie pomocou Spring MVC.
- **spring-boot-starter-webflux** – poskytuje podporu pre reaktívne webové aplikácie postavené na Project Reactor.
- **spring-boot-starter-security** – integruje Spring Security pre autentifikáciu a autorizáciu používateľov.
- **spring-boot-starter-data-jpa** – zabezpečuje jednoduchú prácu s databázou pomocou JPA a Hibernate.
- **lombok** – generuje boilerplate kód ako gettery, settery či konštruktory pomocou anotácií počas kompilácie.
- **spring-boot-starter-test** – obsahuje nástroje a knižnice pre testovanie Spring aplikácií (napr. JUnit, Mockito).
- **postgresql** – JDBC ovládač pre komunikáciu s PostgreSQL databázou.
- **modelmapper** – zjednodušuje mapovanie medzi entitami a DTO objektmi.
- **gson** – knižnica od Googlu na prevod objektov na JSON a späť.
- **dotenv-java** – umožňuje načítavať konfiguráciu z `.env` súboru počas behu aplikácie.

3.2 Štruktúra projektu

- `src/main/java/com/thesis/yr` – priečinok obsahujúci všetky zdrojové kódy:
  - config – priečinok pre konfiguráciu aplikácie.
  - controller – priečinok s Rest kontrolérmi.
  - dto – priečinok s DTO (Data Transfer Object).
  - entity – priečinok s entitami reprezentujúcimi tabuľky v databáze.
  - exception – priečinok s vlastnými výnimkami.
  - model – priečinok s modulmi.
  - repository – priečinok s JPA repozitármi.
  - service – priečinok so službami.
  - BackendApplication.java – súbor používaný na spustenie Spring Boot aplikácie.
- `src/main/resources` – priečinok so zdrojmi
  - application.properties – súbor používaný na konfiguráciu Spring Boot aplikácie.
- pom.xml – súbor so závislosťami

3.3 Súbory

3.3.1 AppConfig.java

Trieda `AppConfig` slúži ako konfiguračná trieda pre Spring aplikáciu. Pomocou anotácie `@Configuration` označuje, že obsahuje definície beanov. V tomto prípade definuje jeden bean typu ModelMapper, ktorý je dostupný v celom kontexte aplikácie. ModelMapper sa používa na jednoduché mapovanie medzi entitami a dátovými objektmi (DTO).

3.3.2 SecurityConfig.java

Trieda `SecurityConfig` obsahuje konfiguráciu zabezpečenia pomocou Spring Security. Je označená ako konfiguračná trieda pomocou anotácie `@Configuration`. Definuje dva beany:

- **SecurityFilterChain** – konfiguruje bezpečnostný filter pre HTTP požiadavky. V tomto prípade sú povolené všetky požiadavky (/**) bez autentifikácie a je vypnutá ochrana proti CSRF.
- **CorsConfigurationSource** – definuje pravidlá pre CORS (Cross-Origin Resource Sharing), ktoré povoľujú požiadavky z lokálnych frontendových adries `http://localhost:5173` a `http://localhost:4173` so špecifikovanými HTTP metódami a hlavičkami. Takisto povoľuje zasielanie cookies.

3.3.3 WebClientConfig.java

Trieda `WebClientConfig` konfiguruje HTTP klienta pre komunikáciu s Jenkins serverom pomocou knižnice `WebClient` zo Spring WebFlux. Je označená ako konfiguračná trieda anotáciou `@Configuration`.

- Vytvára sa bean `jenkinsWebClient`, ktorý načítava prihlasovacie údaje pre Jenkins (používateľské meno a API token) zo súboru `.env` pomocou knižnice `dotenv-java`.
- Tieto údaje sú zakódované do formátu Basic Auth a následne použité pri každej požiadavke ako HTTP hlavička `Authorization`.
- WebClient je nakonfigurovaný tak, aby posielal požiadavky na lokálny Jenkins server `http://localhost:8888` s hlavičkou `Content-Type: application/json`.

3.3.4 BranchController.java

Trieda `BranchController` slúži ako REST kontrolér, ktorý sprístupňuje API endpoint pre získavanie informácií o vetvách (branches). Je označená anotáciou `@RestController` a mapuje sa na URL cestu `/api/branches`.

- Používa `BranchService` ako závislosť.
- Metóda `getAllBranches()` je mapovaná na HTTP GET požiadavku a voliteľne prijíma parameter `timeScope`, ktorý však aktuálne nie je využívaný.
- Táto metóda vracia zoznam objektov typu `BranchDTO`, ktoré predstavujú jednotlivé vetvy.

3.3.5 JenkinsController.java

Trieda `JenkinsController` poskytuje REST API na komunikáciu s Jenkins serverom. Je anotovaná ako `@RestController` a sprístupňuje endpointy pod cestou `/api/jenkins`.

- Používa `JenkinsService` ako závislosť.
- Metóda `buildJob()` je mapovaná na HTTP POST požiadavku s dynamickým segmentom `itemName` v URL.
- Táto metóda spustí danú Jenkins úlohu pomocou služby `JenkinsService` a vráti reťazec s odpoveďou.

3.3.6 ProfileController.java

Trieda `ProfileController` sprístupňuje REST API na získavanie profilov, ktoré sú priradené k určitej vetve. Je označená anotáciou `@RestController` a mapovaná na cestu `/api/profiles`.

- Používa `ProfileService` ako závislosť.
- Metóda `getProfilesByBranchId()` spracováva HTTP GET požiadavky na `/branch` a očakáva parameter `id`, ktorý reprezentuje identifikátor vetvy.
- Vráti zoznam objektov typu `ProfileDTO`, ktoré sú naviazané na danú vetvu.

3.3.7 BranchDTO.java

Trieda `BranchDTO` predstavuje dátový objekt (DTO), ktorý slúži na prenos informácií o vetve medzi backendom a frontendom.

- Obsahuje polia: `id` (identifikátor vetvy), `name` (názov vetvy), `startTime` (čas začiatku spustenia), `totalTests` (celkový počet testov), `failedTests` (počet zlyhaných testov), `status` (stav vetvy), `duration` (dĺžka trvania) a `hasRetries` (informácia, či vetva obsahuje opakované pokusy).
- Trieda je anotovaná pomocou Lombok anotácií `@Data`, `@AllArgsConstructor` a `@NoArgsConstructor`, ktoré automaticky generujú gettre, settre, konštruktory a metódy ako `toString()`, `equals()` a `hashCode()`.

3.3.8 ProfileDTO.java

Trieda `ProfileDTO` slúži ako dátový objekt (DTO) na prenos informácií o konkrétnych profiloch vetiev medzi backendom a frontendom.

- Obsahuje nasledujúce polia: `id` (identifikátor profilu), `name` (názov profilu), `startTime` (čas začiatku), `totalTests` (počet vykonaných testov), `failedTests` (počet neúspešných testov), `duration` (dĺžka spustenia), `status` (stav profilu) a `hasRetries` (informácia o opakovaniach).
- Pomocou anotácií z knižnice Lombok — `@Data`, `@AllArgsConstructor` a `@NoArgsConstructor` – sú automaticky generované gettre, settre, konštruktory, a ďalšie pomocné metódy.

3.3.9 Branch.java

Trieda `Branch` predstavuje entitu, ktorá je mapovaná na tabuľku v databáze a slúži na uchovávanie informácií o jednotlivých vetvách.

- Trieda je anotovaná anotáciou `@Entity`, čo znamená, že je spracovávaná ako entita JPA.
- Polia zahŕňajú: `id` (identifikátor vetvy), `name` (názov vetvy), `startTime` (čas začiatku), `totalTests` (celkový počet testov), `failedTests` (počet neúspešných testov), `status` (stav vetvy), `duration` (dĺžka spustenia) a `hasRetries` (informácia o opakovaniach).
- V triede sa nachádza vzťah `OneToMany` s entitou `Profile`, čo znamená, že jedna vetva môže mať viacero profilov.
- Použitie anotácie `@EqualsAndHashCode(exclude = "id")` zabezpečuje, že pri porovnávaní objektov sa neberie do úvahy pole `id`.
- `@Column` anotácie sú použité na mapovanie názvov stĺpcov v databáze, ktoré sa líšia od názvov polí v triede.

3.3.10 Profile.java

Trieda `Profile` predstavuje entitu, ktorá je mapovaná na tabuľku v databáze a uchováva informácie o profile vetvy.

- Trieda je anotovaná `@Entity`, čo znamená, že je spracovávaná ako entita JPA.

- Polia zahŕňajú: `id` (identifikátor profilu), `name` (názov profilu), `startTime` (čas začiatku), `totalTests` (celkový počet testov), `failedTests` (počet neúspešných testov), `status` (stav profilu), `duration` (dĺžka spustenia) a `hasRetries` (informácia o opakovaniach).
- Použitie anotácie `@ManyToOne` znamená, že každý profil je priradený k jednej vetve (entita `Branch`), pričom tento vzťah je mapovaný cez `@JoinColumn` s názvom `branch_id`.
- Anotácia `@EqualsAndHashCode(exclude = "id")` zabezpečuje, že pri porovnávaní objektov sa neberie do úvahy pole `id`.
- `@Column` anotácie sú použité na mapovanie názvov stĺpcov v databáze, ktoré sa líšia od názvov polí v triede.

3.3.11 JenkinsServiceException.java

Trieda `JenkinsServiceException` je vlastná výnimka, ktorá rozširuje `RuntimeException` a slúži na zachytávanie a hlásenie chýb súvisiacich s komunikáciou so službou Jenkins.

- Trieda poskytuje dva konštruktory:
  - Prvý konštruktor prijíma správu (`message`), ktorá bude zaznamenaná v výnimke.
  - Druhý konštruktor umožňuje prijať správu a príčinu výnimky (`cause`), čím poskytuje podrobnejšie informácie o tom, čo spôsobilo túto výnimku.
- Trieda je rozšírením `RuntimeException`, čo znamená, že ide o nekontrolovanú výnimku, ktorú môže vyhodiť akýkoľvek kód počas behu programu.

3.3.12 Status.java

Trieda `Status` je výčtový typ (enum), ktorý predstavuje rôzne stavy, ktoré môže mať profil alebo vetva v systéme.

- Tento výčtový typ obsahuje nasledujúce hodnoty:
  - `PASSED` - Profil alebo vetva boli úspešne dokončené.
  - `FAILED` - Profil alebo vetva zlyhali počas vykonávania.

    - `IN_PROGRESS` - Profil alebo vetva sú v procese vykonávania.
    - `STOPPED` - Profil alebo vetva boli zastavené pred ich dokončením.
    - `INTERRUPTED` - Profil alebo vetva boli prerušené z nejakého dôvodu.
- Tento výčtový typ sa používa na reprezentáciu stavu procesu alebo operácie v systéme.

3.3.13 BranchRepository.java

Rozhranie `BranchRepository` je rozšírením `CrudRepository` a slúži na interakciu s databázou a správu entít `Branch`.

- Rozhranie dedí od `CrudRepository`, čo znamená, že poskytuje základné metódy na vykonávanie CRUD operácií (Create, Read, Update, Delete) na entitách `Branch`.
- Metóda `findAll()` sa používa na načítanie všetkých záznamov z databázy, ktoré sú reprezentované entitou `Branch`.

3.3.14 ProfileRepository.java

Rozhranie `ProfileRepository` je rozšírením `CrudRepository` a poskytuje metódy na interakciu s databázou pre entitu `Profile`.

- Rozhranie dedí od `CrudRepository`, čo znamená, že umožňuje vykonávať základné CRUD operácie na entitách `Profile`.
- Metóda `findAllByBranchId(int branchId)` umožňuje získať zoznam profilov, ktoré sú priradené k určitej vetve podľa ID vetvy (`branchId`).

3.3.15 BranchService.java

Trieda `BranchService` poskytuje logiku služieb pre entitu `Branch` a vykonáva operácie týkajúce sa vetiev systému.

- Trieda je označená anotáciou `@Service`, čo znamená, že ide o Spring Service komponentu, ktorá poskytuje business logiku.
- `BranchService` obsahuje dve hlavné závislosti:
    - `BranchRepository` na komunikáciu s databázou.
    - `ModelMapper` na mapovanie entít `Branch` na DTO objekty `BranchDTO`.

- Metóda `getAllBranches()`:
  - Načíta všetky vetvy zo systému pomocou metódy `findAll()` z `BranchRepository`.
  - Každú vetvu následne mapuje na objekt `BranchDTO` pomocou `modelMapper`.
  - Všetky DTO objekty sú zoradené podľa času začiatku (`startTime`) v zostupnom poradí.

3.3.16 JenkinsService.java

Trieda `JenkinsService` poskytuje logiku na interakciu so službou Jenkins, konkrétne na spúšťanie Jenkins úloh.

- Trieda je označená anotáciou `@Service`, čo znamená, že ide o Spring Service komponentu, ktorá poskytuje business logiku.
- `JenkinsService` používa `WebClient` na interakciu s Jenkins API. `WebClient` je konfigurovaný ako `jenkinsWebClient` a injektovaný do triedy pomocou anotácie `@Qualifier`.
- Konštruktor triedy prijíma `WebClient` ako závislosť a inicializuje internú premennú `jenkinsWebClient`.
- Metóda `buildJob(String jobName)`:
  - Posiela POST požiadavku na spustenie Jenkins úlohy pomocou URI `/job/jobName/build?delay=0sec`, kde `jobName` je názov úlohy.
  - Výsledok požiadavky je spracovaný pomocou `bodyToMono(String.class)` na získanie odpovede ako reťazca.
  - Ak dôjde k chybe počas požiadavky, vyvolá sa vlastná výnimka `JenkinsServiceException` s popisom chyby.

3.3.17 ProfileService.java

Trieda `ProfileService` poskytuje logiku služieb pre entitu `Profile` a vykonáva operácie týkajúce sa profilov systému.

- Trieda je označená anotáciou `@Service`, čo znamená, že ide o Spring Service komponentu, ktorá poskytuje business logiku.

- Trieda obsahuje dve hlavné závislosti:
  - `ProfileRepository` na komunikáciu s databázou.
  - `ModelMapper` na mapovanie entít `Profile` na DTO objekty.
- Metóda `getAllProfilesByBranchId(int branchId)`:
  - Načíta všetky profily pre danú vetvu zo systému pomocou metódy `findAllByBranchId()` z `ProfileRepository`.
  - Každý profil je následne mapovaný na objekt `ProfileDTO` pomocou `modelMapper`.
  - Všetky DTO objekty sú zoradené podľa času začiatku (`startTime`) v zostupnom poradí.

3.3.18 BackendApplication.java

Trieda `BackendApplication` je hlavným bodom vstupu do aplikácie a obsahuje metódu `main`, ktorá spúšťa celú aplikáciu Spring Boot.

- Trieda je označená anotáciou `@SpringBootApplication`, čo je zjednodušený spôsob, ako spustiť Spring Boot aplikáciu. Anotácia zahŕňa v sebe `@Configuration`, `@EnableAutoConfiguration` a `@ComponentScan`.
- Hlavná metóda `main(String[] args)` spúšťa aplikáciu.
- Táto trieda neobsahuje žiadnu ďalšiu logiku, jej účelom je len spustiť Spring Boot kontext a načítať konfigurácie a komponenty aplikácie.

4 Frontend

Tato časť dokumentácie predstavuje podrobný popis architektúry frontendu, ktorý aktívne využíva TypeScript a React.

4.1 Závislosti

V projekte sa závislosti delia na dve hlavné kategórie: **dependencies** a **devDependencies**, pričom každá z nich zohráva inú úlohu počas vývoja a behu aplikácie.

4.1.1 Dependencies

Dependencies sú knižnice, ktoré aplikácia potrebuje na svoj chod v produkcii – teda keď ju používateľ spustí. Patria sem napríklad React, React Router, alebo stavový manažér ako Zustand.

- **@fluentui/react** – knižnica komponentov od Microsoftu pre tvorbu moderného UI.
- **@typescript-eslint/eslint-plugin** – plugin pre ESLint na kontrolu TypeScript kódu.
- **@typescript-eslint/parser** – parser umožňujúci ESLintu analyzovať TypeScript.
- **eslint-plugin-jsx-a11y** – pomáha dodržiavať prístupnosť v JSX komponentoch.
- **eslint-plugin-prettier** – ESLint plugin integrujúci Prettier pre formátovanie kódu.
- **eslint-plugin-react** – plugin pre ESLint so špecifickými pravidlami pre React.

- **monaco-editor** – editor kódu ako vo VS Code, vhodný pre webové aplikácie.
- **react** – základná knižnica na tvorbu používateľského rozhrania.
- **react-dom** – poskytuje metódy na rendering React komponentov do DOMu.
- **react-router-dom** – knižnica pre routing v React aplikáciách.
- **react-select** – komponent pre výber z možností (dropdown) s rozšírenými funkciami.
- **xml-formatter** – nástroj na formátovanie XML dokumentov.
- **zustand** – ľahký stavový manažér pre React aplikácie.

4.1.2 DevDependencies

DevDependencies sú knižnice, ktoré slúžia iba na vývoj a testovanie aplikácie, ako napríklad ESLint, TypeScript, Vite alebo typové definície.

- **@eslint/js** – oficiálna kolekcia pravidiel ESLint pre JS.
- **@types/react** – typové definície pre React v TypeScripte.
- **@types/react-dom** – typové definície pre react-dom v TypeScripte.
- **@vitejs/plugin-react** – plugin pre Vite umožňujúci použitie Reactu.
- **eslint** – linter na kontrolu kvality a chýb v JavaScript/TypeScript kóde.
- **eslint-plugin-import** – plugin na kontrolu a správu importov v projekte.
- **eslint-plugin-react-hooks** – zabezpečuje správne používanie React hookov.
- **eslint-plugin-react-refresh** – pomáha s hot-reloadom v React vývoji.
- **eslint-plugin-unused-imports** – odhaľuje a odstraňuje nepoužívané importy.
- **globals** – zoznam globálnych premenných pre rôzne prostredia (napr. browser, node).
- **typescript** – programovací jazyk rozširujúci JavaScript o typy.
- **typescript-eslint** – integruje TypeScript s ESLintom.
- **vite** – moderný build nástroj pre vývoj frontendu s rýchlym HMR.

4.2 Štruktúra projektu

- `src` – priečinok obsahujúci všetky zdrojové kódy:
  - assets – priečinok s ikonkami.
  - components – priečinok s React komponentami.
  - constants – priečinok s konštantami.
  - model – priečinok s modulmi.
  - store – priečinok s Zustand úložiskom.
  - utils – priečinok s volaniami API.
  - App.css – súbor so spoločnými štýlmi pre kontajnery React.
  - App.tsx – súbor zodpovedný za zobrazenie konkrétnych stránok na webe.
  - index.css – súbor so spoločnými štýlmi a fontmi pre celú webovú stránku.
  - main.tsx – hlavný súbor pri spustení frontendu.
  - profiles.json – súbor obsahujúci všetky možné názvy profilov vetiev.
- index.html – hlavný html súbor.
- package.json – súbor so závislosťami.
- vite.config.ts – súbor s konfiguráciou Vite.js.

4.3 Súbory

4.3.1 LaunchesActionButtons.tsx

Komponent `LaunchesActionButtons` predstavuje ovládacie tlačidlá na manipuláciu so spusteniami v aplikácii. Poskytuje funkcie ako pridanie nového spustenia, filtrovanie, vyhľadávanie, výber časového rozsahu a reštartovanie vybraných spustení.

- **Použité knižnice a komponenty:**
  - `@fluentui/react` – na dizajnové komponenty ako `DefaultButton`, `Dropdown`, `FontIcon`, `SearchBox`.

  - `zustand` – na správu globálneho stavu filtrov.
  - `react` – základ pre funkčný komponent a hooky ako `useState`.

- **Vstupné vlastnosti komponentu (`Props`):**
  - `selectedLaunches` – pole vybraných položiek typu SelectableLaunchItem.
  - `unselectAllLaunches` – funkcia na zrušenie výberu všetkých spustení.
  - `changeTimeScope` – funkcia na zmenu časového rozsahu.
  - `searchLaunches` – funkcia na vyhľadávanie podľa reťazca.
  - `clearSearch` – funkcia na vymazanie vyhľadávacieho vstupu.
  - `filterLaunches` – funkcia na filtrovanie spustení podľa stavu.

- **Hooky a stavy:**
  - `useState<boolean>` – `isNewLaunchFooterOpen`, určuje či je otvorená spodná časť s formulárom na nové spustenie.
  - `useState<string>` – `message`, text správy pre používateľa.
  - `useState<MessageType | null>` – typ zobrazenej správy (napr. RESTART).
  - `useState<string>` – `toastMessage`, správa pre `Toast`.
  - `useLaunchesFilterStore` – získava a upravuje filter stav pomocou zustand.

- **Funkcia `handleTimeScopeChange`** — pri zmene hodnoty v Dropdown komponentu aktualizuje časový rozsah v `localStorage` a resetuje filtre.

- **Funkcia `timeScope`** — načítava hodnotu časového rozsahu z `localStorage`, nastaví predvolenú hodnotu na `24`, ak tam ešte nie je.

- **Funkcia `filterCount`** — vypočíta počet aktívnych filtrov (súčet stavov + retry flag).

- **Funkcia `generateRestartMessage`** — pripraví správu pre potvrdenie reštartu podľa počtu vybraných položiek.

- **Funkcia `restartLaunches`** — asynchrónne zavolá funkciu `restartItem` pre každé vybrané spustenie a zobrazí správu po dokončení.

- **Funkcia `closeToastMessage`** — vymaže správu v toast notifikácii.
- **Renderovaná štruktúra:**
  - Sekcia s tlačidlami „Add new launch“ a „Restart“ (ak sú položky vybrané).
  - Pravá sekcia s filtrami, vyhľadávaním a výberom časového rozsahu.
  - Komponent `AddNewLaunchFooter`, ktorý sa otvorí kliknutím.
  - Komponent `GeneratedMessage`, ktorý sa zobrazí pri potvrdení reštartu.
  - Komponent `Toast` na zobrazenie výslednej správy po reštarte.

4.3.2 LaunchesFilterButton.tsx

Komponent `LaunchesFilterButton` poskytuje možnosť filtrovať spustenia podľa ich stavu a opakovaní. Tento komponent je zodpovedný za zobrazenie filtra v podobe `Callout` menu, ktoré sa otvorí po kliknutí na tlačidlo.

- **Použité knižnice a komponenty:**
  - `@fluentui/react` – na dizajnové komponenty ako `Callout`, `CommandBarButton`, `Checkbox`, `Separator`, `Stack`, `Text`.
  - `zustand` – na správu globálneho stavu filtrov v `useLaunchesFilterStore`.
  - `react` – základ pre funkčný komponent a hooky ako `useState`, `useRef`.
- **Vstupné vlastnosti komponentu (`Props`):**
  - `filterLaunches` – funkcia na filtrovanie spustení na základe vybraných filtrov.
  - `filterCount` – počet aktívnych filtrov, ktorý sa zobrazuje pri tlačidle.
- **Hooky a stavy:**
  - `useState<boolean>` – `isCalloutVisible`, určuje, či je `Callout` zobrazený.
  - `useRef<HTMLDivElement | null>` – `buttonRef`, referenčný hák pre `CommandBarButton`.
  - `useLaunchesFilterStore` – na čítanie a aktualizáciu stavu filtrov.
- **Funkcia `toggleCallout`** – preklopí viditeľnosť `Callout` panelu.

- **Funkcia `handleStatusesChange`** – aktualizuje stav filtrovania podľa vybraného stavu spustenia (úspech, zlyhanie, prebiehajúce).
- **Funkcia `handleRetryChange`** – aktualizuje stav filtrovania podľa opätovného spustenia.
- **Funkcia `getCheckboxCheckedState`** – kontroluje, či je daný stav filtrovania aktívny.
- **Renderovaná štruktúra:**
  - Tlačidlo `CommandBarButton` pre zobrazenie filtra.
  - `Callout`, ktorý zobrazuje možnosti filtrovania, ako sú stavy spustení a opakované spustenia.
  - `Checkbox` komponenty pre jednotlivé možnosti filtrovania.
  - `Separator` pre vizuálne oddelenie sekcií.

4.3.3 ProfilesActionButtons.tsx

Komponent `ProfilesActionButtons` slúži na zobrazenie akcií pre profily v aplikácii, ako je reštartovanie vybraných profilov, filtrovanie a vyhľadávanie profilov. Tento komponent obsahuje tlačidlá pre vykonávanie týchto akcií, spolu s funkcionalitou na zobrazenie správ a toast notifikácií.

- **Použité knižnice a komponenty:**
  - `@fluentui/react` – na dizajnové komponenty ako `DefaultButton`, `SearchBox`.
  - `react` – základ pre funkčný komponent a hooky ako `useState`.
  - `zustand` – na správu globálneho stavu filtrov v `useProfilesFilterStore`.
  - `restartItem` – pomocná funkcia pre reštartovanie profilov.
- **Vstupné vlastnosti komponentu (`Props`):**
  - `selectedProfiles` – pole vybraných profilov typu `SelectableProfileItem`.
  - `unselectAllProfiles` – funkcia na zrušenie výberu všetkých profilov.
  - `searchProfiles` – funkcia na vyhľadávanie profilov podľa reťazca.
  - `clearSearch` – funkcia na vymazanie vyhľadávania.

- `filterProfiles` – funkcia na filtrovanie profilov podľa stavu.

- **Hooky a stavy:**
  - `useState<string>` – `message`, text správy pre používateľa.
  - `useState<MessageType | null>` – `messageType`, typ zobrazenej správy (napr. RESTART).
  - `useState<string>` – `toastMessage`, správa pre `Toast`.
  - `useProfilesFilterStore` – na čítanie a aktualizáciu stavu filtrov.

- **Funkcia `filterCount`** – vypočíta počet aktívnych filtrov (stavy + retry flag).

- **Funkcia `generateRestartMessage`** – pripraví správu pre potvrdenie reštartu podľa počtu vybraných profilov.

- **Funkcia `restartProfiles`** – asynchrónne reštartuje vybrané profily a zobrazí toast správu po dokončení.

- **Funkcia `closeToastMessage`** – vymaže správu v toast notifikácii.

- **Renderovaná štruktúra:**
  - Sekcia s tlačidlom `Restart`, ktoré sa zobrazuje, ak sú položky vybrané.
  - Komponent `ProfilesFilterButton` pre filtrovanie profilov.
  - Komponent `SearchBox` na vyhľadávanie profilov.
  - Komponent `GeneratedMessage` pre potvrdenie reštartu profilov.
  - Komponent `Toast` na zobrazenie výslednej správy po reštarte.

4.3.4 ProfilesFilterButton.tsx

Komponent `ProfilesFilterButton` slúži na zobrazenie tlačidla pre filtrovanie profilov. Po kliknutí na tlačidlo sa zobrazí `Callout` s možnosťami pre filtrovanie profilov podľa rôznych stavov a možnosti opakovania.

- **Použité knižnice a komponenty:**
  - `@fluentui/react` – na dizajnové komponenty ako `Callout`, `Checkbox`, `CommandBarButton`, `Separator`, `Stack`, `Text`.
  - `react` – základ pre funkčný komponent a hooky ako `useState`, `useRef`.

- `zustand` – na správu globálneho stavu filtrov v useProfilesFilterStore.

- **Vstupné vlastnosti komponentu (`Props`):**
  - `filterProfiles` – funkcia na aplikovanie filtrovaných položiek do stavu aplikácie.
  - `filterCount` – počet aktívnych filtrov, ktorý sa zobrazuje vedľa tlačidla.

- **Hooky a stavy:**
  - `useState<boolean>` – `isCalloutVisible`, určuje, či je `Callout` zobrazený.
  - `useRef<HTMLDivElement | null>` – `buttonRef`, odkaz na tlačidlo, ktoré spúšťa otvorenie `Callout`.
  - `useProfilesFilterStore` – na čítanie a aktualizáciu stavu filtrov.

- **Funkcia `toggleCallout`** – prepína viditeľnosť `Callout`, ktorý obsahuje filtre pre statusy a retry flag.

- **Funkcia `handleStatusesChange`** – aktualizuje stav filtrov pre konkrétny status profilov (Passed, Failed, In progress).

- **Funkcia `handleRetryChange`** – aktualizuje stav filtru pre retry flag, ktorý označuje, či sa profil pokúša o opakovaný pokus.

- **Funkcia `getCheckboxCheckedState`** – kontroluje, či je daný status v aktívnych filtroch.

- **Renderovaná štruktúra:**
  - Tlačidlo `Filter profiles`, ktoré po kliknutí zobrazí `Callout`.
  - `Callout` s možnosťami filtrovania podľa statusov (Passed, Failed, In progress) a retry flag.
  - Každý filter je reprezentovaný `Checkbox` komponentom.
  - `Separator` medzi sekciami statusov a iných filtrov.

4.3.5 BreadcrumbNav.tsx

Komponent `BreadcrumbNav` poskytuje navigačný panel zobrazený ako `Breadcrumb`, ktorý zobrazuje aktuálnu cestu a umožňuje navigáciu medzi jednotlivými úrovňami hierarchie stránok. Tento komponent využíva `react-router-dom` pre prácu s trasami a navigáciou.

- **Použité knižnice a komponenty:**
  - `react-router-dom` – pre získanie aktuálnej lokácie a navigáciu medzi stránkami.
  - `@fluentui/react/lib/Breadcrumb` – pre komponent `Breadcrumb`, ktorý zobrazuje navigačný panel.
  - `localStorage` – na uloženie názvu aktuálneho spustenia (`launchName`).
  - `Icon` – vlastný komponent na zobrazenie ikony pre rozdeľovač (`divider`).
- **Stavy a hooky:**
  - `useLocation` – hook z `react-router-dom`, ktorý umožňuje prístup k aktuálnej URL ceste.
  - `useNavigate` – hook z `react-router-dom`, ktorý umožňuje navigáciu medzi stránkami.
- **Funkčnosť komponentu:**
  - Komponent extrahuje relatívnu cestu z `location.pathname`, zohľadňuje základnú cestu `BASE_PATH` (ak je prítomná).
  - Cesta sa rozdelí na segmenty, ktoré sa použijú na generovanie položiek navigačného panelu.
  - Ak je segment "Profiles", nahrádza sa hodnotou z `localStorage`, ak je k dispozícii (`launchName`).
  - Ak je segment posledný v zozname, stáva sa aktuálnym prvkom v navigácii, inak umožňuje navigáciu po kliknutí.
- **Renderovaná štruktúra:**
  - Zobrazuje komponent `Breadcrumb` s položkami, ktoré predstavujú jednotlivé úrovne cesty.
  - Na rozdeľovanie medzi položkami sa používa vlastný komponent `Icon` so symbolom "divider".

- **Vlastnosti komponentu:**
  - `items` – zoznam položiek pre `Breadcrumb`, kde každá položka má text, kľúč, a možnosť kliknutia.
  - `dividerAs` – umožňuje nahradiť predvolený rozdeľovač vlastnou ikonou.

4.3.6 FooterCheckboxes.tsx

Komponent `FooterCheckboxes` umožňuje užívateľovi vybrať medzi dvoma možnosťami: `True` alebo `False`. T ento komponent používa SVG ikony na zobrazenie začiarknutých a nezačiarknutých políčok, ktoré reprezentujú stav výberu.

- **Použité knižnice a komponenty:**
  - `React` – základná knižnica pre vytváranie komponentov v Reacte.
  - `styles` – vlastné CSS triedy pre štýlovanie komponentu.
  - `svg` – na zobrazenie ikon pre začiarknuté a nezačiarknuté políčka.
- **Funkčnosť komponentu:**
  - Komponent obsahuje dve možnosti: `True` a `False`, ktoré sú reprezentované pomocou SVG ikon.
  - Po kliknutí na ikonu sa stav výberu prepne medzi `True` a `False`, pričom tieto stavy sú spravované pomocou funkcie `setSelected`.
  - Stav (boolean) je riadený cez `selected` a môže sa meniť kliknutím na jednotlivé možnosti.
- **Vlastnosti komponentu:**
  - `selected` – boolean hodnota, ktorá určuje, ktorá možnosť (True alebo False) je aktuálne vybraná.
  - `setSelected` – funkcia na aktualizáciu hodnoty `selected` pri zmene výberu.
- **Renderovaná štruktúra:**
  - Zobrazuje dve možnosti (`True` a `False`) v podobe začiarknutých políčok.
  - Každé políčko je reprezentované SVG ikonou, ktorá mení svoj stav v závislosti od hodnoty `selected`.

4.3.7 TableCheckbox.tsx

Komponent `TableCheckbox` poskytuje prispôsobený začiarkavací box, ktorý zobrazuje rôzne stavy (predvolené, pri najetí myšou, začiarknutý) v závislosti od interakcie užívateľa.

- **Použité knižnice a komponenty:**
  - `useState` – React hook na správu stavu komponentu (určuje, či je checkbox prechádzaný myšou).
  - `styles` – vlastné CSS triedy pre štýlovanie komponentu.
  - `svg` – na zobrazenie rôznych ikon pre checkbox podľa stavu.
- **Funkčnosť komponentu:**
  - Komponent zobrazuje začiarkavací box, ktorý môže mať tri rôzne stavy: predvolený, začiarknutý alebo pri najetí myšou.
  - Používa sa tu stav `isHovered`, ktorý určuje, či je myš nad začiarkavacím boxom.
  - Pri zmenu stavu `checked` sa zobrazuje iná ikona pre začiarknutý stav, ako aj pre prechod myšou.
- **Vlastnosti komponentu:**
  - `checked` – určuje, či je checkbox začiarknutý alebo nie.
  - `readOnly` – checkbox je iba na čítanie, takže užívateľ nemôže meniť jeho stav.
- **Renderovaná štruktúra:**
  - Zobrazuje SVG ikony pre rôzne stavy checkboxu:
    - Predvolený stav – `#checkbox-default`.
    - Pri najetí myšou – `#checkbox-hovered`.
    - Začiarknutý stav – `#checkbox-checked`.

4.3.8 LaunchesContent.tsx

Komponent `LaunchesContent` je trieda v React, ktorá spravuje stav a interakcie so zoznamom `launches`. Tento komponent obsahuje logiku na získavanie údajov, filtrovanie a vyhľadávanie medzi rôznymi štartmi. Využíva sa tu React

lifecycle metóda `componentDidMount`, ktorá spúšťa asynchrónnu požiadavku na načítanie dát pri prvotnom načítaní komponentu.

- **Použité knižnice a komponenty:**
  - `React.Component` – React trieda na vytvorenie komponentu.
  - `LaunchesActionButtons` – Komponent pre akčné tlačidlá týkajúce sa štartov (napríklad filtrovanie, zmena rozsahu času).
  - `LaunchesTable` – Komponent pre zobrazenie tabulky so štartmi.
  - `Separator` – Komponent pre vizuálne oddelenie sekcií.
  - `useLaunchesFilterStore` – Hook na získanie a aktualizáciu stavu filtra.
  - `getAllLaunches`, `convertToLaunches` – Funkcie na získavanie a konverziu údajov.
- **Funkčnosť komponentu:**
  - Komponent získava dáta pomocou funkcie `fetchBranches`, ktorá načítava údaje o štartoch a konvertuje ich na vhodný formát.
  - Vyhľadávanie a filtrovanie štartov je umožnené pomocou funkcií searchLaunches a filterLaunches.
  - `setSelectedLaunches` a `unselectAllLaunches` umožňujú aktualizovať stav vybraných štartov.
  - Používa sa `localStorage` na uchovávanie voľby časového rozsahu.
- **Stavy komponentu:**
  - `selectedLaunches` – pole objektov reprezentujúcich vybrané štarty.
  - `launches` – pole všetkých štartov načítaných zo servera.
  - `filteredLaunches` – filtrované štarty podľa aktuálneho filtra.
  - `isLaunchesRetrieved` – boolean indikujúci, či boli štarty úspešne načítané.
- **Vlastnosti komponentu:**
  - Komponent neakceptuje žiadne vstupné vlastnosti (props).
- **Renderovaná štruktúra:**

- Zobrazuje nadpis `Launches`, komponent `BreadcrumbNav` a tlačidlá akcií `LaunchesActionButtons`.
- Po zobrazení sa zobrazí tabulka s `LaunchesTable` na zobrazenie filtrovaných alebo všetkých štartov.

4.3.9 ProfilesContent.tsx

Komponent `ProfilesContent` je trieda v React, ktorá spravuje stav a interakcie so zoznamom profilov. Tento komponent zodpovedá za získavanie profilov podľa identifikátora štartu, filtrovanie a vyhľadávanie medzi profilmi. Používa sa tu React lifecycle metóda `componentDidMount`, ktorá spúšťa asynchrónnu požiadavku na načítanie profilov pri načítaní komponentu.

- **Použité knižnice a komponenty:**
  - `React.Component` – React trieda na vytvorenie komponentu.
  - `ProfilesActionButtons` – Komponent pre akčné tlačidlá týkajúce sa profilov (napríklad filtrovanie, vyhľadávanie).
  - `ProfilesTable` – Komponent pre zobrazenie tabulky s profilmi.
  - `Separator` – Komponent pre vizuálne oddelenie sekcií.
  - `useProfilesFilterStore` – Hook na získanie a aktualizáciu stavu filtra profilov.
  - `getAllProfilesByLaunchId`, `convertToProfileTableItems` – Funkcie na získavanie a konverziu údajov profilov.
- **Funkčnosť komponentu:**
  - Komponent načítava dáta o profiloch pomocou funkcie `fetchProfiles`, ktorá získava údaje podľa `launchId` a konvertuje ich na vhodný formát.
  - Vyhľadávanie a filtrovanie profilov je umožnené pomocou funkcií searchProfiles a filterProfiles.
  - `setSelectedProfiles` a `unselectAllProfiles` umožňujú aktualizovať stav vybraných profilov.
- **Stavy komponentu:**
  - `selectedProfiles` – pole objektov reprezentujúcich vybrané profily.

- `profiles` – pole všetkých profilov načítaných zo servera.
- `filteredProfiles` – filtrované profily podľa aktuálneho filtra.
- `isProfilesRetrieved` – boolean indikujúci, či boli profily úspešne načítané.
- `launchName` – názov štartu, ktorý je použitý v hlavičke.

- **Vlastnosti komponentu:**
  - `launchId` – identifikátor štartu, podľa ktorého sa profil načítava.
  - `launchName` – názov štartu, ktorý sa zobrazuje v hlavičke.

- **Renderovaná štruktúra:**
  - Zobrazuje nadpis `Profiles`, komponent `BreadcrumbNav` a tlačidlá akcií `ProfilesActionButtons`.
  - Po zobrazení sa zobrazí tabulka s `ProfilesTable` na zobrazenie filtrovaných alebo všetkých profilov.

4.3.10 DropdownMenu.tsx

Komponent `DropdownMenu` je React komponent, ktorý poskytuje používateľovi viacnásobný výber z možností v podobe rozbaľovacieho menu. Tento komponent využíva knižnicu `react-select` na implementáciu rozbaľovacieho menu s podporou viacerých výberov. Komponent je tiež obalený v `forwardRef`, čo umožňuje odovzdanie ref do komponentu `Select`.

- **Použité knižnice a komponenty:**
  - `react-select` – Knižnica na implementáciu rozbaľovacieho menu s podporou viacerých výberov.
  - `Label` – Komponent od Fluent UI na zobrazenie štítku pre rozbaľovacie menu.

- **Funkčnosť komponentu:**
  - Komponent umožňuje používateľovi vybrať viacero položiek zo zoznamu.
  - Pri zmene výberu sa vyvolá funkcia `handleChange`, ktorá aktualizuje stav vybraných položiek pomocou funkcie `setSelectedItems`.

- **Stavy komponentu:**
  - Komponent neudržiava vlastný stav, ale využíva `setSelectedItems`, ktorý aktualizuje stav vo vyššej úrovni aplikácie.
- **Vlastnosti komponentu:**
  - `labelName` – Názov štítku pre rozbaľovacie menu.
  - `items` – Pole položiek typu `ProfileOptionType`, ktoré budú zobrazené v rozbaľovacom menu.
  - `setSelectedItems` – Funkcia na nastavenie vybraných položiek (zobrazovaných v menu).
- **Vlastnosti `Select` komponentu:**
  - `isMulti` – Povolenie viacnásobného výberu.
  - `closeMenuOnSelect` – Po výbere položky sa menu neuzatvára.
  - `onChange` – Funkcia na spracovanie zmien vo výbere.
  - `theme` – Prispôsobenie štýlu, vrátane okrajov a farby primárneho prvku.
- **Vlastnosti štýlov:**
  - Použitie `styles.dropdownContainer` pre obalenie celého komponentu.
  - Použitie `styles.dropdownLabel` pre štýl štítku nad rozbaľovacím menu.
- **Renderovaná štruktúra:**
  - Zobrazuje štítok s názvom `labelName`.
  - Zobrazuje `Select` komponent pre viacnásobný výber položiek.

4.3.11 AddNewLaunchFooter.tsx

Komponent `AddNewLaunchFooter` je React komponent, ktorý slúži na zobrazenie a spracovanie formulára na pridanie nového spustenia v Jenkins. Tento komponent umožňuje zadávať rôzne parametre konfigurácie spustenia, vrátane názvu codeline, verzie testovaného softvéru, verzie runtime, režimu vývoja a výberu profilov.

- **Použité knižnice a komponenty:**
  - `TextField` a `PrimaryButton` z Fluent UI – Komponenty pre textové polia a tlačidlá.
  - `DropdownMenu` – Komponent pre výber viacerých profilov.
  - `FooterCheckboxes` – Komponent na zobrazovanie a výber možností vývojového režimu.
  - `GeneratedXMLMessage` – Komponent na zobrazenie vygenerovaného XML.
  - `Toast` – Komponent na zobrazenie notifikácie o vytvorení úlohy.
  - `useCodelineParams` – Vlastný hook pre získavanie a nastavenie parametrov codeline.
  - `collectCodelineParamsToXML` – Funkcia na konverziu parametrov do XML formátu.
  - `createGitLabIssue` – Funkcia na vytvorenie GitLab issue pre nový launch.
- **Funkčnosť komponentu:**
  - Komponent umožňuje zadávať a upravovať rôzne parametre pre nový launch (codeline name, tested version, runtime version, atď.).
  - Po úprave parametrov sa automaticky generuje XML obsahujúce tieto parametre.
  - Používateľ môže uložiť konfiguráciu, zobraziť vygenerované XML alebo vytvoriť GitLab issue.
- **Stavy komponentu:**
  - `selectedItems` – Uchováva vybrané položky (profily) z `DropdownMenu`.
  - `isUserEdited` – Stav, ktorý indikuje, či bol používateľom upravený niektorý z parametrov.
  - `resultXML` – Vygenerované XML, ktoré obsahuje všetky aktuálne nastavené parametre.
  - `isResultOpened` – Stav, ktorý určuje, či sa zobrazuje vygenerované XML.
  - `toastMessage` – Správa pre toast notifikáciu po vytvorení úlohy.

- **Vlastnosti komponentu:**
  - `profiles` – Pole profilov typu `ProfileOptionType`, ktoré sú k dispozícii na výber.
  - `isOpen` – Určuje, či je panel otvorený.
  - `setOpen` – Funkcia na zmenu stavu otvorenia panelu.
- **Metódy komponentu:**
  - `handleClose` – Zatvára panel a resetuje všetky zadané údaje.
  - `handleSave` – Ukladá aktuálne nastavené profily a generuje XML.
  - `handleCreateIssue` – Vytvára GitLab issue na základe zadaných parametrov a XML.
  - `taskCreated` – Nastavuje správu pre toast po vytvorení úlohy.
  - `closeToastMessage` – Zatvára toast správu.
- **Renderovaná štruktúra:**
  - Zobrazuje formulár s textovými poľami pre zadanie codeline name, tested version, runtime version a výberom vývojového režimu.
  - Zobrazuje `DropdownMenu` pre výber profilov.
  - Tlačidlá pre uloženie konfigurácie, zrušenie a zobrazenie vygenerovaného XML.
  - Po uložení sa zobrazí vygenerované XML alebo toast notifikácia o vytvorení úlohy.

4.3.12 Header.tsx

Komponent `Header` je jednoduchý React komponent, ktorý slúži na zobrazenie hlavičky aplikácie. Tento komponent obsahuje logo vo forme ikony vafle a názov aplikácie "ARIS TA Service".

- **Použité knižnice a komponenty:**
  - `img` – HTML tag pre zobrazenie obrázka ikony.
  - `styles` – CSS modul pre štýlovanie komponentu.
- **Funkčnosť komponentu:**

- Zobrazuje logo aplikácie v podobe ikony vafle (`WaffleIcon`) a text "ARIS TA Service".

- **Stavy komponentu:**
  - Tento komponent nemá žiadne stavy, pretože slúži len na zobrazenie statického obsahu.

- **Vlastnosti komponentu:**
  - Tento komponent neakceptuje žiadne vstupné vlastnosti (props).

- **Metódy komponentu:**
  - Tento komponent nemá žiadne metódy.

- **Renderovaná štruktúra:**
  - Komponent zobrazuje dva hlavné elementy:
    * `img` element pre zobrazenie ikony vafle.
    * `div` element s názvom aplikácie "ARIS TA Service".

4.3.13 ProfilesContentWithParams.tsx

Komponent `ProfilesContentWithParams` slúži ako premostenie medzi routingom aplikácie a komponentom `ProfilesContent`. Zabezpečuje extrakciu parametra `id` z URL a hodnoty `launchName` z `localStorage`, ktoré následne odovzdáva ako vlastnosti (props) do `ProfilesContent`.

- **Použité knižnice a funkcie:**
  - `useParams` – hook z knižnice `react-router-dom` na získanie parametrov z URL.
  - `localStorage.getItem` – získava názov spustenia (`launchName`) z lokálneho úložiska prehliadača.

- **Funkčnosť komponentu:**
  - Získa parameter `id` z aktuálnej URL adresy (napr. `/profiles/123`).
  - Získa `launchName` z `localStorage`, pričom predvolene nastaví prázdny reťazec, ak hodnota neexistuje.
  - Odovzdá tieto údaje ako `props` do komponentu `ProfilesContent`.

- **Vstupné a výstupné údaje:**
  - **Vstup:** implicitne cez URL a `localStorage`.
  - **Výstup:** komponent `ProfilesContent` s vlastnosťami launchId a launchName.
- **Poznámka:**
  - Použitie `id!` (tzv. non-null assertion) predpokladá, že hodnota `id` bude vždy definovaná. Toto by mohlo byť doplnené o ochranu (fallback) v prípade, že `id` nie je k dispozícii.

4.3.14 Icon.tsx

Komponent `Icon` je univerzálny React komponent na zobrazovanie SVG ikon podľa zvoleného názvu. Umožňuje konzistentne zobrazovať rôzne typy ikon vo vizuálnom rozhraní aplikácie pomocou jedného komponentu.

- **Vstupné parametre (`props`):**
  - `name` (`'restart' | 'add' | 'delete' | 'divider' | 'retry'`) – identifikátor požadovanej ikony. Tento parameter je povinný.
  - `width` (`number`, nepovinný) – šírka ikony v pixeloch, predvolená hodnota je `16`.
  - `height` (`number`, nepovinný) – výška ikony v pixeloch, predvolená hodnota je `16`.
- **Použité SVG ikony:**
  - `RestartIcon` – reštartovanie profilu.
  - `AddIcon` – pridanie novej položky.
  - `DeleteIcon` – odstránenie položky.
  - `DividerIcon` – vizuálny oddeľovač.
  - `RetryIcon` – opakovanie akcie.
- **Implementácia:**
  - Pomocou `switch` vetvy sa podľa hodnoty `name` vyberie príslušný SVG súbor.

- Ak sa zadaný názov nezhoduje so žiadnym z definovaných prípadov, komponent vracia `null`, čím sa žiadna ikona nezobrazí.
- Výstupom komponentu je HTML element `<img>` s atribútmi `src`, `alt`, `width` a `height`.

- **Výhody riešenia:**
  - Centralizované spracovanie ikon znižuje duplikáciu kódu.
  - Možnosť jednoduchého rozšírenia o nové ikony pridaním ďalšieho `case`.
  - Zabezpečenie konzistentného štýlu ikon v celej aplikácii.

4.3.15 GeneratedMessage.tsx

Komponent `GeneratedMessage` slúži ako modálne okno s potvrdením, ktoré používateľovi zobrazuje vygenerovanú správu (napr. XML konfiguráciu) a umožňuje potvrdiť alebo zrušiť vykonanie akcie (napr. vytvorenie úlohy).

- **Vstupné parametre (`props`):**
  - `message` (`string`) – text správy, ktorý sa zobrazí v hlavičke okna.
  - `setOpen` (`Dispatch<SetStateAction<boolean>>`) – funkcia na zatvorenie modálneho okna.
  - `onSave` (`() => void | Promise<void>`) – callback funkcia, ktorá sa spustí po potvrdení akcie (napr. vytvorenie GitLab úlohy).

- **Implementácia:**
  - Po kliknutí na tlačidlo `Yes` sa vykoná funkcia `onSave` a následne sa modálne okno zatvorí.
  - Po kliknutí na tlačidlo `No` sa modálne okno len zatvorí pomocou funkcie `setOpen(false)`.
  - Komponent využíva Fluent UI tlačidlá `PrimaryButton` a `DefaultButton`.
  - Súčasťou komponentu je aj polo-transparentné pozadie (`bluredBackground`), ktoré vizuálne oddelí modálne okno od zvyšku aplikácie.

- **Štruktúra:**
  - `messageBox` – hlavný kontajner s obsahom správy.
  - `messageContainer` – vnútorný blok so zobrazením textu a tlačidiel.

- `actionButtons` – kontajner pre ovládacie tlačidlá `Yes` a `No`.
- `bluredBackground` – rozostrené pozadie na zneaktívnenie obsahu aplikácie počas zobrazenia modálu.

- **Použitie:**
  - Komponent sa využíva pri potvrdzovaní operácií, ktoré používateľ vyvolal (napr. pri vytváraní issue na základe XML konfigurácie).
  - Zaručuje, že používateľ musí vedome potvrdiť akciu, čím zvyšuje bezpečnosť a použiteľnosť aplikácie.

4.3.16 GeneratedXMLMessage.tsx

Komponent `GeneratedXMLMessage` slúži ako modálne okno na zobrazenie vygenerovaného XML dokumentu (napr. konfigurácia pipeline) s možnosťou jeho potvrdenia a uloženia. Využíva editor `Monaco Editor` v režime `read-only`, aby používateľ mohol pohodlne prezerať vygenerovaný kód.

- **Vstupné parametre (`props`):**
  - `message` (`string`) – text XML správy, ktorá sa zobrazí v editore.
  - `setOpen` (`Dispatch<SetStateAction<boolean>>`) – funkcia na zatvorenie modálneho okna.
  - `onSave` (`() => void`) – callback funkcia, ktorá sa spustí po potvrdení akcie (napr. uloženie konfigurácie).

- **Implementácia:**
  - Používa `Monaco Editor` z balíka `monaco-editor`, ktorý je umiestnený do referenčného DOM prvku `editorRef`.
  - Editor je nakonfigurovaný ako `read-only`, so zvýrazňovaním syntaxe XML a vypnutou minimapou.
  - Pri unmountovaní komponentu sa editor uvoľní pomocou `editor.dispose()`.
  - Tlačidlo `Add` zavolá funkciu `onSave` a zatvorí okno.
  - Tlačidlo `Cancel` len zatvorí okno bez vykonania akcie.

- **Štruktúra:**
  - `messageBox` – hlavný kontajner modálneho dialógu.

- `messageContainer` – vnútorný blok s hlavičkou, editorom a akčnými tlačidlami.
- `messageHeader` – textový nadpis dialógu, napr. Generated XML.
- `dialog` – kontajner pre editor s XML správou.
- `actionButtons` – blok s ovládacími tlačidlami.
- `bluredBackground` – rozostrené pozadie slúžiace na zvýraznenie modálneho okna.

- **Použitie:**
  - Tento komponent sa zobrazuje po úspešnom vygenerovaní XML konfigurácie pre vybraný profil alebo vetvu.
  - Poskytuje používateľovi možnosť skontrolovať obsah predtým, ako sa rozhodne ho pridať do systému (napr. vytvoriť z neho GitLab issue).
  - Vizuálne a funkčne oddeľuje tento krok od zvyšku aplikácie, čím zvyšuje prehľadnosť a bezpečnosť procesu.

4.3.17 Sidebar.tsx

Komponent `Sidebar` slúži ako jednoduchý vertikálny navigačný panel, ktorý umožňuje používateľovi prepínať medzi jednotlivými sekciami aplikácie. V tejto implementácii sa zobrazuje len jedna možnosť – `Launches`, ktorá je zvýraznená pri výbere alebo pri prejdení kurzorom myši.

- **Použité hooky:**
  - `useState<string>` na uchovanie aktuálne vybranej položky v stave `selected`.
  - `useState<boolean>` pre detekciu hover efektu pomocou stavu isHovered.

- **Funkcionalita:**
  - Kliknutím na tlačidlo `Launches` sa nastaví stav `selected` na `'launches'`, čo zabezpečí vizuálne zvýraznenie vybranej položky.
  - Pomocou udalostí `onMouseEnter` a `onMouseLeave` sa mení stav isHovered, ktorý ovplyvňuje výber SVG ikony.

- Používa sa SVG sprite s dvoma stavmi ikonky – `launches-icon-default` a `launches-icon-hovered`, ktoré sa prepínajú v závislosti od aktuálneho výberu alebo hoveru.

- **Štruktúra komponentu:**
  - `sidebarContainer` – hlavný kontajner postranného panela.
  - `sidebarButtonContainer` – obal pre jednotlivé tlačidlo vrátane dekorácie pre výber.
  - `selection` – dekoratívny pásik naľavo, ktorý indikuje aktívnu sekciu.
  - `sidebarButton` – samotné tlačidlo, ktoré obsahuje SVG ikonu a text.
  - `sidebarButtonText` – textový popis položky navigácie.

- **Zobrazenie ikon:**
  - SVG ikony sa renderujú cez tag `<use>` s atribútom `href`, ktorý sa dynamicky mení podľa stavu komponentu.
  - Tento spôsob umožňuje jednoduché prepínanie ikon bez potreby meniť celý SVG obsah.

- **Rozšíriteľnosť:**
  - Komponent je navrhnutý tak, aby bolo možné jednoducho pridať ďalšie navigačné položky – stačí pridať nové bloky sidebarButtonContainer s príslušnými ikonami a stavovými podmienkami.

4.3.18 LaunchesTable.tsx

Komponent `LaunchesTable` je zodpovedný za vykreslenie tabuľky so zoznamom `Launch` objektov v rámci používateľského rozhrania. Tento komponent poskytuje podporu pre triedenie, výber viacerých riadkov a navigáciu k detailom spustení.

- **Vstupné vlastnosti (`props`):**
  - `launches: Launch[]` – pole objektov reprezentujúcich spustenia vetiev.
  - `selectedLaunches: SelectableLaunchItem[]` – aktuálne vybrané položky.

- `setSelectedLaunches: (item: SelectableLaunchItem[]) => void` – funkcia na aktualizáciu výberu.
- `launchesRetrieved: boolean` – príznak označujúci, či boli dáta načítané.

- **Stav komponentu (`state`):**
  - `selectedLaunches: SelectableLaunchItem[]` – lokálne vybrané položky.
  - `selection: Selection` – inštancia výberového modelu z Fluent UI.
  - `columns: IColumn[]` – konfigurácia stĺpcov tabuľky.
  - `launches: Launch[]` – lokálne uchovávané spustenia.

- **Štruktúra a renderovanie tabuľky:**
  - Komponent využíva `ShimmeredDetailsList` z Fluent UI pre zobrazenie shimmer efektu počas načítania dát.
  - `MarqueeSelection` umožňuje výber viacerých riadkov potiahnutím myši.
  - Tabuľka podporuje triedenie kliknutím na hlavičky stĺpcov, ktoré je implementované cez metódu `onColumnClick`.
  - Výber je synchronizovaný so stavom komponentu pomocou onSelectionChanged, kde sa mapujú vybrané objekty na typ `SelectableLaunchItem`.

- **Vlastnosti jednotlivých stĺpcov:**
  - `Launch` – obsahuje názov spustenia s odkazom na profily, zvýraznený stav `isRetry` ikonou.
  - `Start Time` – čas začatia spustenia.
  - `Status` – aktuálny stav, štýlovaný podľa mapy `statusClassMap`.
  - `Total`, `Failed`, `Duration` – štatistické údaje o priebehu testov.

- **Interakcie a pomocné funkcie:**
  - `handleProfilesLinkClick` – pri kliknutí na názov spustenia uloží názov do `localStorage` pre neskoršie použitie.
  - `findLaunchName` – pomocná funkcia na získanie názvu spustenia podľa jeho kľúča.

- `copyAndSort` – externá pomocná funkcia pre triedenie stĺpcov.
- Vlastné komponenty ako `TableCheckbox` a `CustomIcon` slúžia na prispôsobenie vzhľadu.

- **Zaujímavosti:**
  - Výber sa obnovuje pri aktualizácii vstupných údajov alebo pri odstránení výberu.
  - Triedenie podporuje prepínanie medzi vzostupným a zostupným smerom.
  - Ikony a tooltipy v tabuľke zlepšujú používateľskú skúsenosť.

4.3.19 ProfilesTable.tsx

Komponent `ProfilesTable` slúži na zobrazenie tabuľky profilov v rámci jedného spustenia (`Launch`). Tabuľka umožňuje triedenie, výber viacerých riadkov a zobrazuje doplnkové informácie o stave, trvaní a počte testov.

- **Vstupné vlastnosti (`props`):**
  - `setSelectedProfiles: (profile: SelectableProfileItem[]) => void` – funkcia na aktualizáciu vybraných profilov.
  - `selectedProfiles: SelectableProfileItem[]` – aktuálne vybrané profily.
  - `launchId: string` – identifikátor aktuálneho spustenia.
  - `launchName: string` – názov aktuálneho spustenia.
  - `profiles: ProfileTableItem[]` – pole profilov určených na zobrazenie.
  - `isLoading: boolean` – príznak, či sa dáta práve načítavajú.

- **Stav komponentu (`state`):**
  - `selectedProfiles: SelectableProfileItem[]` – lokálne vybrané profily.
  - `selection: Selection` – Fluent UI mechanizmus na správu výberu.
  - `columns: IColumn[]` – konfigurácia stĺpcov tabuľky.
  - `profiles: ProfileTableItem[]` – profily, ktoré sa majú zobraziť.

- **Štruktúra a renderovanie tabuľky:**
  - Používa komponent `ShimmeredDetailsList` na zobrazenie shimmer efektu počas načítavania.
  - Obalený v `MarqueeSelection`, ktorý umožňuje výber potiahnutím myši.
  - Používa komponent `TableCheckbox` na prispôsobenie vzhľadu výberových políčok.
- **Stĺpce tabuľky:**
  - `Profile` – názov profilu, zvýraznený príznakom `isRetry` s ikonou (tooltip).
  - `Start Time` – čas začatia vykonávania profilu.
  - `Status` – stav profilu, zobrazený s CSS triedou podľa `statusClassMap`.
  - `Total` – celkový počet testov.
  - `Failed` – počet zlyhaných testov.
  - `Duration` – trvanie spustenia profilu; ak chýba, zobrazí sa znak en dash.
- **Funkcie a interakcie:**
  - `onSelectionChanged` – aktualizuje `selectedProfiles` podľa používateľského výberu.
  - `findProfileName` – získa názov profilu podľa jeho kľúča.
  - `onColumnClick` – triedenie podľa vybraného stĺpca (vzostupne alebo zostupne).
- **Logika spracovania zmien:**
  - Pri odznačení všetkých riadkov sa vyčistí výber cez `Selection`.
  - Pri zmene profilov v `props` sa aktualizuje lokálny stav.
  - Zmeny vo výbere sa propagujú pomocou `setSelectedProfiles`.

4.3.20 Toast.tsx

Komponent `Toast` slúži na dočasné zobrazovanie krátkych správ používateľovi. Ide o notifikačný komponent, ktorý sa automaticky zobrazuje a následne po krátkom čase zmizne.

- **Vstupné vlastnosti (`props`):**
  - `message: string` – textová správa, ktorá sa má používateľovi zobraziť.
  - `onClose: () => void` – callback, ktorý sa zavolá po skrytí toastu (po skončení animácie).
- **Vnútorný stav (`useState`):**
  - `visible: boolean` – príznak určujúci, či je toast aktuálne viditeľný.
- **Použité hooky:**
  - `useEffect` – spustí logiku zobrazenia toastu pri zmene správy:
    - okamžite nastaví `visible` na `true`, čím sa spustí animácia zobrazenia,
    - po 3 sekundách nastaví `visible` na `false`, čo spustí animáciu skrytia,
    - po 3.3 sekundách zavolá `onClose`, čím zabezpečí úplné odstránenie komponentu.
  - `clearTimeout` – zabezpečí zrušenie časovačov pri odmontovaní komponentu alebo zmene správy.
- **Renderovanie:**
  - Toast je `<div>` s dvomi triedami: základná (`toast`) a dodatočná (`show`) ak je `visible = true`.
  - Obsahuje vnorený `<div>` s textom správy, ktorému je priradená štýlovacia trieda `message`.
  - Triedy sú definované v CSS module `toast.module.css`, čo zabezpečuje lokálny rozsah štýlov.

4.3.21 launchesFilterStore.ts

Súbor `launchesFilterStore.ts` definuje stavový manažment pre filter zoznamu spustení pomocou knižnice Zustand. Ide o jednoduchý *global store*, ktorý uchováva a aktualizuje stav filtrov použiteľných pri zobrazení zoznamu profilov.

- **Použitá knižnica:** `zustand/react` – ľahká knižnica na správu stavu, vhodná pre menšie aplikácie alebo moduly, ktoré nevyžadujú zložité zdieľanie stavu.

- **Rozhrania:**
  - `LaunchesFilterState`
    - `statuses: ItemStatus[]` – pole možných stavov, ktoré sa majú filtrovať (napr. `PASSED`, `FAILED`, `SKIPPED`).
    - `retry: boolean` – príznak, či sa majú zahrnúť iba profily s opakovaným spustením testov.
  - `LaunchesFilterStore`
    - `filterState: LaunchesFilterState` – aktuálny stav filtra.
    - `setFilterState: (field, value) => void` – metóda na aktualizáciu konkrétneho poľa filtra.
- **Implementácia:**
  - Store sa vytvára pomocou funkcie `create`, ktorá prijíma funkciu s `set` ako argumentom.
  - Počiatočný stav obsahuje:
    - `statuses` – prázdne pole (žiadny filter),
    - `retry` – `false` (všetky profily).
  - Funkcia `setFilterState` umožňuje meniť jednotlivé polia filtra. Ak je hodnota funkcia, táto funkcia sa aplikuje na predchádzajúci stav.
- **Export:**
  - Exportovaný hook `useLaunchesFilterStore` sa používa v React komponentoch na čítanie a úpravu stavu filtrov.

4.3.22 profilesFilterStore.ts

Súbor `profilesFilterStore.ts` definuje stavový manažment pre filter zoznamu profilov, ktorý je implementovaný pomocou knižnice Zustand. Tento stavový manažment sa stará o ukladanie a manipuláciu so stavom filtra, ktorý sa využíva pri zobrazení a filtrovaní profilov v aplikácii.

- **Použitá knižnica:** `zustand/react` – knižnica na jednoduchú správu globálneho stavu v aplikácii, ktorá nevyžaduje komplexnú implementáciu a je vhodná pre menšie projekty.
- **Rozhrania:**

- `ProfilesFilterState`
  - `statuses: ItemStatus[]` – pole, ktoré uchováva zoznam stavov profilov, ktoré sa majú filtrovať. Tieto stavy sú typu `ItemStatus`, ako napr. `PASSED`, `FAILED`, `SKIPPED`.
  - `retry: boolean` – príznak, ktorý určuje, či sa majú filtrovať len profily s opakovaným spustením testov.
- `ProfilesFilterStore`
  - `filterState: ProfilesFilterState` – aktuálny stav filtra, ktorý obsahuje zoznam stavov a príznak opakovania.
  - `setFilterState: (field, value) => void` – funkcia na nastavenie hodnôt jednotlivých polí v stave filtra.
  - `resetFilterState: () => void` – funkcia, ktorá resetuje filter na počiatočný stav (t. j. odstráni všetky filtre).

- **Implementácia:**
  - Store je vytvorený pomocou funkcie `create` z knižnice `zustand`, ktorá umožňuje jednoducho spravovať stav v React aplikácii.
  - Počiatočný stav `initialFilterState` obsahuje:
    - `statuses` – prázdne pole, čo znamená, že žiadny stav nie je filtrovaný.
    - `retry` – `false`, čo znamená, že nie je aktivovaný filter na opakované spustenie.
  - Funkcia `setFilterState` aktualizuje konkrétne pole v stave filtra. Ak je hodnota funkcia, táto sa aplikuje na aktuálny stav.
  - Funkcia `resetFilterState` resetuje filter na počiatočný stav, čo znamená, že sa odstránia všetky filtre a používateľ vidí všetky profily bez obmedzenia.

- **Export:**
  - Exportovaný hook `useProfilesFilterStore` je použiteľný v React komponentoch, kde je potrebné čítať a aktualizovať stav filtra profilov.

4.3.23 CodelineToXMLUtils.ts

Súbor `CodelineToXMLUtils.ts` definuje funkciu collectCodelineParamsToXML, ktorá je zodpovedná za generovanie XML reťazca na základe parametrov, ktoré

sú potrebné pre zostavenie informácií o kódovej línii. Táto funkcia zohráva kľúčovú úlohu v procese generovania konfigurácie pre spustenie testov v prostredí Jenkins.

- **Použité knižnice:**
  - `xml-formatter` – knižnica na formátovanie XML reťazcov, ktorá zabezpečuje čitateľnosť výsledného XML dokumentu pomocou vhodného odsadenia a zbalenia obsahu.
- **Funkcia:** `collectCodelineParamsToXML`
  - **Vstupné parametre:** `params: CodelineParameters` – objekt obsahujúci rôzne parametre potrebné pre generovanie XML, ktoré definujú konfiguráciu pre zostavenie kódovej línie v prostredí Jenkins. Tento objekt obsahuje nasledovné polia:
    - `codelineName` – názov kódovej línie.
    - `testedVersion` – verzia, ktorá bola testovaná.
    - `runtimeVersion` – verzia runtime prostredia.
    - `devMode` – režim vývoja.
    - `fromJenkins` – označuje, či je proces spustený z Jenkins.
    - `templateSelector` – názov šablóny, ktorá je vybraná pre spustenie.
    - `profiles` – pole profilov na testovanie, pričom každý profil má atribút `name`, ktorý sa použije na vytvorenie zoznamu v XML.
  - **Výstup:** Funkcia vracia formátovaný XML reťazec reprezentujúci parametre pre kódovú líniu. Tento reťazec je generovaný takto:
    - Generuje XML element `<codeline>` ako koreňový element.
    - Každý parameter zo vstupného objektu `params` je umiestnený v samostatnom XML elemente
    - V prípade pole profilov sa vytvorí jeden XML element <profilesToTest>, ktorý obsahuje názvy všetkých profilov spojené čiarkou.
  - **Formátovanie:** Výsledné XML je formátované pomocou knižnice xml-formatter, ktorá zabezpečuje:
    - Odsadenie obsahu pomocou dvoch medzier.
    - Zbalenie obsahu, aby bol výstup kompaktný a čitateľný.

4.3.24 GitLabIssueUtils.ts

Súbor `GitLabIssueUtils.ts` obsahuje funkciu `createGitLabIssue`, ktorá slúži na vytvorenie nového problému (issue) v systéme GitLab prostredníctvom jeho REST API. Táto funkcia je využívaná na automatizáciu procesu správy problémov, čo zjednodušuje integráciu medzi aplikáciou a nástrojom na správu zdrojového kódu.

- **Použité knižnice:**
  - `fetch` – vstavaná funkcia JavaScriptu na vykonávanie HTTP požiadaviek.
- **Funkcia:** `createGitLabIssue`
  - **Vstupné parametre:**
    - `title` – názov problému (issue), ktorý bude vytvorený v GitLab.
    - `body` – popis problému, ktorý bude pridaný do tela problému. Tento text je formátovaný do XML bloku pomocou syntaxe ``` ```xml ... ``` ```.
  - **Výstup:** Funkcia nevracia žiadnu hodnotu, ale vytvára problém na serveri GitLab, ak je požiadavka úspešná.
  - **Priebeh vykonávania:**
    - Funkcia vykoná asynchrónnu požiadavku typu `POST` na GitLab API endpoint https://git.kpi.fei.tuke.sk/api/v4/projects/68189/issues, ktorý slúži na vytváranie nových problémov v projekte s ID 68189.
    - Do tela požiadavky je pridaný objekt obsahujúci:
      - `title` – názov problému.
      - `description` – formátovaný popis problému s obsahom vo formáte XML.
      - `assignee_ids` – pole obsahujúce ID používateľov, ktorým je problém priradený.
    - V prípade úspešného vytvorenia problému API vráti dáta, ktoré sú následne vypísané do konzoly.
    - Ak požiadavka zlyhá, zobrazí sa chybová správa vrátane detailov o zlyhaní.

- **Bezpečnosť:** Pre autentifikáciu na GitLab sa používa `PRIVATE-TOKEN`, ktorý je uložený v environmentálnej premennej `VITE_GITLAB_API_TOKEN`. Tento token je potrebný na autentifikáciu pri komunikácii s GitLab API.

4.3.25 LaunchesUtils.ts

Súbor `LaunchesUtils.ts` obsahuje niekoľko funkcií, ktoré umožňujú manipuláciu s vetvami v systéme Jenkins a ich spracovanie v rámci aplikácie. Tento súbor komunikuje s API servera na získanie a manipuláciu s údajmi o vetvách a následne ich prevádza na formát, ktorý môže byť zobrazený v používateľskom rozhraní.

- **Použité knižnice:**
  - `fetch` – vstavaná funkcia JavaScriptu na vykonávanie HTTP požiadaviek.
- **Funkcia:** `getAllLaunches`
  - **Vstupné parametre:**
    - `timeScope` – parameter, ktorý určuje rozsah času (napríklad obdobie, za ktoré sa načítajú dáta).
  - **Výstup:**
    - Funkcia vracia pole objektov typu `Branch`, ktoré obsahujú informácie o vetvách.
  - **Priebeh vykonávania:**
    - Funkcia vykoná HTTP požiadavku typu `GET` na endpoint http://localhost:8080/api/branches so zadaným parametrom `timeScope`.
    - Ak požiadavka zlyhá, vyvolá sa výnimka a v konzole sa vypíše chybová správa.
- **Funkcia:** `convertToLaunches`
  - **Vstupné parametre:**
    - `data` – pole objektov typu `Branch`.
  - **Výstup:**
    - Funkcia vracia pole objektov typu `Launch`, ktoré sú upravené a pripravené na zobrazenie v používateľskom rozhraní.

- **Priebeh vykonávania:**
  - Funkcia mapuje každú vetvu z `data` na nový objekt `Launch` obsahujúci požadované vlastnosti, ako napríklad názov, stav a ďalšie.

- **Funkcia:** `getBranchStatus`
  - **Vstupné parametre:**
    - `branch` – objekt typu `Branch`, ktorý obsahuje stav vetvy.
  - **Výstup:**
    - Funkcia vracia stav vetvy ako hodnotu typu `ItemStatus`.
  - **Priebeh vykonávania:**
    - Funkcia vyhodnotí stav vetvy a v prípade neúspechu (ak sú testy zlyhané) vráti hodnotu `FAILED`.
    - Ak je vetva v procese, vráti stav `IN_PROGRESS`.
    - Ak všetky testy prebehli úspešne, vráti stav zodpovedajúci úspešnému vykonaniu.

- **Funkcia:** `restartItem`
  - **Vstupné parametre:**
    - `item` – identifikátor položky, ktorá sa má reštartovať.
  - **Výstup:**
    - Funkcia nevytvára žiadny výstup, ale spustí Jenkins úlohu, ktorá reštartuje daný proces.
  - **Priebeh vykonávania:**
    - Funkcia vykoná HTTP požiadavku typu `POST` na URL http://localhost:8080/api/jenkins/job/{item}/build, aby spustila reštartovanie úlohy v Jenkins.
    - V prípade zlyhania požiadavky sa vypíše chybová správa.

- **Bezpečnosť:** Pri komunikácii s backendom sa predpokladá, že komunikácia prebieha v rámci dôveryhodného lokálneho servera (localhost), kde nie je potrebná ďalšia autentifikácia.

4.3.26 ProfilesUtils.ts

Súbor `ProfilesUtils.ts` obsahuje funkcie, ktoré umožňujú získať informácie o profiloch na základe ID spustenia a konvertovať tieto informácie do formátu vhodného na zobrazenie v tabuľkách používateľského rozhrania.

- **Použité knižnice:**
  - `fetch` – vstavaná funkcia JavaScriptu na vykonávanie HTTP požiadaviek.
- **Funkcia:** `getAllProfilesByLaunchId`
  - **Vstupné parametre:**
    - `launchId` – identifikátor spustenia, podľa ktorého sa vyhľadávajú profily.
  - **Výstup:**
    - Funkcia vracia pole objektov typu `Profile`, ktoré obsahujú informácie o profiloch spustených v rámci daného `launchId`.
  - **Priebeh vykonávania:**
    - Funkcia vykoná HTTP požiadavku typu `GET` na endpoint http://localhost:8080/api/profiles/branch?id={launchId}.
    - Ak požiadavka zlyhá, vyvolá sa výnimka a v konzole sa vypíše chybová správa.
- **Funkcia:** `convertToProfileTableItems`
  - **Vstupné parametre:**
    - `data` – pole objektov typu `Profile`.
  - **Výstup:**
    - Funkcia vracia pole objektov typu ProfileTableItem, ktoré obsahujú upravené a formátované údaje o profiloch.
  - **Priebeh vykonávania:**
    - Funkcia mapuje každú položku z `data` na nový objekt ProfileTableItem, ktorý je vhodný na zobrazenie v tabuľke. Obsahuje vlastnosti ako názov profilu, čas spustenia, počet testov a stav.
- **Funkcia:** `getProfileStatus`

- **Vstupné parametre:**
  - `profile` – objekt typu `Profile`, ktorý obsahuje informácie o profile.
- **Výstup:**
  - Funkcia vracia stav profilu ako hodnotu typu `ItemStatus`.
- **Priebeh vykonávania:**
  - Funkcia vyhodnotí stav profilu a v prípade neúspechu (ak sú testy zlyhané) vráti hodnotu `FAILED`.
  - Ak je profil v procese, vráti stav `IN_PROGRESS`.
  - Ak všetky testy prebehli úspešne, vráti stav zodpovedajúci úspešnému vykonaniu.

- **Bezpečnosť:** Pri komunikácii s backendom sa predpokladá, že komunikácia prebieha v rámci dôveryhodného lokálneho servera (localhost), kde nie je potrebná ďalšia autentifikácia.

4.3.27 SortingUtils.ts

Súbor `SortingUtils.ts` obsahuje pomocné funkcie na manipuláciu a spracovanie údajov, ktoré sa používajú v iných častiach aplikácie.

- **Použité knižnice:**
  - `match` – metóda na vyhľadávanie regulárnych výrazov v reťazcoch.
- **Funkcia:** `convertDurationToMinutes`
  - **Vstupné parametre:**
    - `duration` – reťazec reprezentujúci trvanie vo formáte `Xh Ym Zs`, kde X, Y, Z sú čísla (hodiny, minúty, sekundy), alebo `null`.
  - **Výstup:**
    - Funkcia vracia celkový čas v minútach (typ `number`).
  - **Priebeh vykonávania:**
    - Funkcia rozparsuje vstupný reťazec `duration` a použije regulárne výrazy na získanie hodín, minút a sekúnd.
    - Na základe týchto hodnôt prepočítava celkový čas na minúty a vracia ho.

        * Ak je vstupný reťazec `null`, funkcia vráti hodnotu 0.

- **Funkcia:** `copyAndSort`
    - **Vstupné parametre:**
        * `launches` – pole objektov, ktoré reprezentujú rôzne spustenia.
        * `columnKey` – kľúč stĺpca, podľa ktorého sa bude zoradovať (napr. `duration`).
        * `isSortedDescending` (voliteľné) – určuje, či bude zoradenie zostupné.
    - **Výstup:**
        * Funkcia vracia nový zoznam `launches`, zoradený podľa hodnoty v určenom stĺpci.
    - **Priebeh vykonávania:**
        * Funkcia vytvorí kópiu pôvodného poľa `launches`.
        * Podľa hodnoty `columnKey` sa vykoná triedenie:
            · Ak je stĺpec `duration`, funkcia použije convertDurationToMinutes na prepočítanie hodnôt na minúty a zoradí podľa nich.
            · Ak sú hodnoty číselné, zoradí ich podľa číselnej hodnoty.
            · Ak sú hodnoty reťazcové, zoradí ich podľa lexikografického poradia.
        * Funkcia vráti nový zoradený zoznam, pričom triedenie môže byť zostupné alebo vzostupné na základe hodnoty `isSortedDescending`.

4.3.28 useCodelineParamsUtils.ts

Súbor `useCodelineParamsUtils.ts` obsahuje `custom hook`, ktorý sa používa na správu parametrov codeline v rámci aplikácie. Tento hook poskytuje spôsob, ako uchovávať a aktualizovať rôzne parametre codeline, vrátane názvu codeline, verzií a profilov. Parametre sa inicializujú predvolenými hodnotami z konštánt a umožňujú manipuláciu s nimi prostredníctvom viacerých setter funkcií.

- **Použité knižnice:**
    - `useState` – hook z Reactu na uchovávanie stavu v komponente.
- **Funkcia:** `useCodelineParams`
    - **Vstupné parametre:**

    - Funkcia neberie žiadne vstupné parametre.
- **Výstup:**
    - Funkcia vracia objekt obsahujúci:
        - `params` – aktuálny stav parametrov codeline (CodelineParameters).
        - Setter funkcie pre každé pole v parametri params (napr. setCodelineName, setTestedVersion, atď.).
        - `resetToDefaults` – funkcia na obnovenie predvolených hodnôt.
- **Priebeh vykonávania:**
    - Hook používa `useState` na uchovávanie aktuálnych parametrov codeline, ktoré sú inicializované predvolenými hodnotami z konštánt `BRANCH_DEFAULTS`.
    - Hook poskytuje viaceré setter funkcie, ktoré umožňujú meniť jednotlivé parametre codeline:
        - `setCodelineName` – nastavuje názov codeline.
        - `setTestedVersion` – nastavuje verziu, ktorá bola testovaná.
        - `setRuntimeVersion` – nastavuje runtime verziu.
        - `setDevMode` – nastavuje vývojový režim (boolean).
        - `setFromJenkins` – nastavuje, či sú parametre načítané z Jenkins.
        - `setProfiles` – nastavuje zoznam profilov.
    - Funkcia `resetToDefaults` obnovuje všetky parametre na predvolené hodnoty definované v `BRANCH_DEFAULTS`.