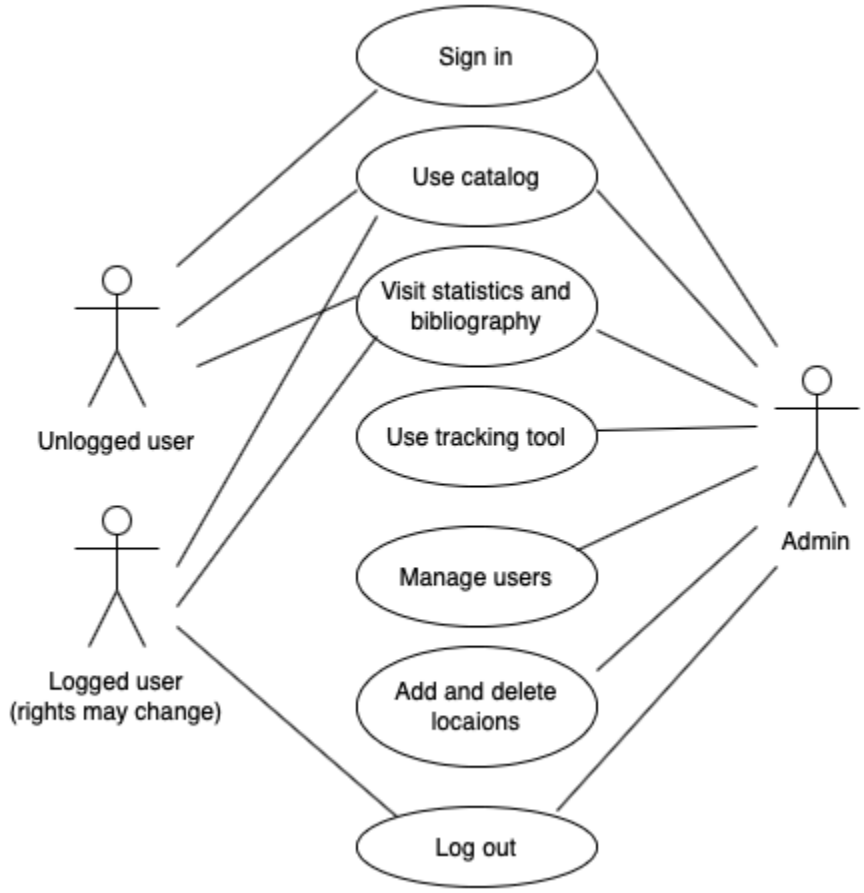
Architektura Aplikace

Tento dokument popisuje architekturu aplikace.

Use Case diagram aplikace



Obr. 1 - Use case diagram.

V aplikaci se mohou pohybovat 3 typy uživatelů - nepřihlášený uživatel, přihlášený uživatel a admin (viz. Obr. 1).

Nepřihlášený uživatel

Nepřihlášený uživatel má na stránkách omezené možnosti. Může používat lokální katalog, který je veřejně přístupný. V tomto katalogu lze zobrazit všechny položky a zároveň existuje možnost filtrace dle názvu, typu lokace apod. Kromě katalogu smí prohlížet stránky s literaturou a statistikou. Poslední právo je přihlášení se do aplikace.

Admin

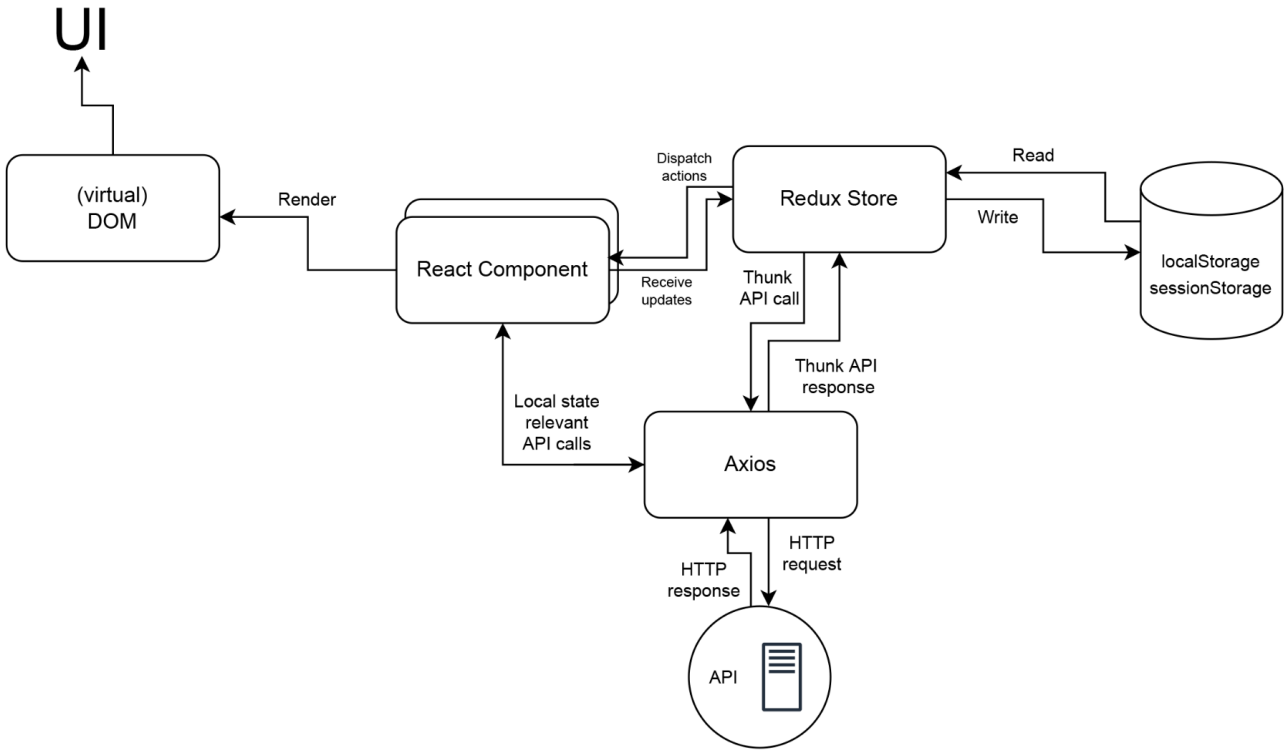
Admin má na stránce všechna práva. Smí navštěvovat všechny stránky a má možnost editace titulní strany, přidávání a mazání lokací v katalogu a přístup k trasovacímu nástroji. Také má na starosti správu uživatelů - včetně jejich registrace a odstranění. U každého uživatele může nastavovat jednotlivá práva.

Přihlášený uživatel

Uživatel, který je do aplikace přihlášen, má již více možností, které však záleží na nastavení adminem. Mezi tyto práva patří přístup k trasovacímu nástroji, přidání a editace lokací či jejich mazání. Tato práva jsou nastavena pouze adminem a uživatel je nesmí měnit.

Architektura frontendu

Architekturu frontendu ukazuje obr. 2.



Obr. 2 - Architektura frontendu.

Základní moduly

Frontend je single page application (SPA) postavená na JavaScript frameworku React. Jako hlavní programovací jazyk se používá TypeScript, který se pro klienta transkompiluje do ES2022 JS. Aplikace se skládá z několika logických modulů:

- **Komponenty** - renderují samotné HTML, CSS a JS, a obsahují logiku pro manipulaci s view (lokální)
- **State** - stav aplikace - např. přihlášení uživatele, barevné téma, načtené předměty v katalogu, apod.
- **HTTP klient pro volání API** - abstrakce volání backendu, abstrakce chyb pro snazší zobrazení uživateli
- **Routing** - zobrazení správného view při dané URL

Business logika

Při návrhu SPA je typicky největším problémem kam umístit business logiku aplikace. Zde byl zvolen nejčastěji používaný přístup a to takový, že základní logiku, která ovlivňuje pouze danou komponentu umístíme do těla funkce komponenty. Pokud je potřeba funkce sdílet mezi více komponentami implementujeme funkce v oddělených modulech.

K některým částem stavu aplikace, jako např. přihlášení uživatele, musíme přistupovat z více komponent a potřebujeme nějaký mechanismus jak toho deterministicky docílit. K tomu slouží framework Redux, díky němuž můžeme vytvořit jeden globální přístup ke stavu - Store. Následně můžeme business logiku, která tento stav upravuje uložit do tzv. reducers, které nad storem provádějí operace.

Kromě úpravy stavu potřebujeme také (ideálně) jednotný přístup k API. Pro to existuje v aplikaci instance HTTP klientu Axios, která má konfiguraci pro správnou komunikaci s backendem (URL serveru, správné headery, apod.) a automaticky obnovuje vypršené přístupové tokeny (access token). Modul také zajistí správné chování při nedostupnosti serveru, které mohou při komunikaci nastat.

Routing

Routing zajišťuje knihovna react-router-dom, která pro dané URL zobrazí požadované view. Router je v podstatě speciální komponenta, která si přečte aktuální URL prohlížeče a najde view, které je pro URL nebo jeho regex namapované.

Vzhled

Základem pro vzhled aplikace je Material Design, který pro React implementuje knihovna MUI (Material UI). Knihovna kromě snadno použitelných React komponent, které dodržují material design obsahuje i velké možnosti přizpůsobení, které budou využity dle potřeb zákazníka.

Verze stěžejních knihoven / frameworků

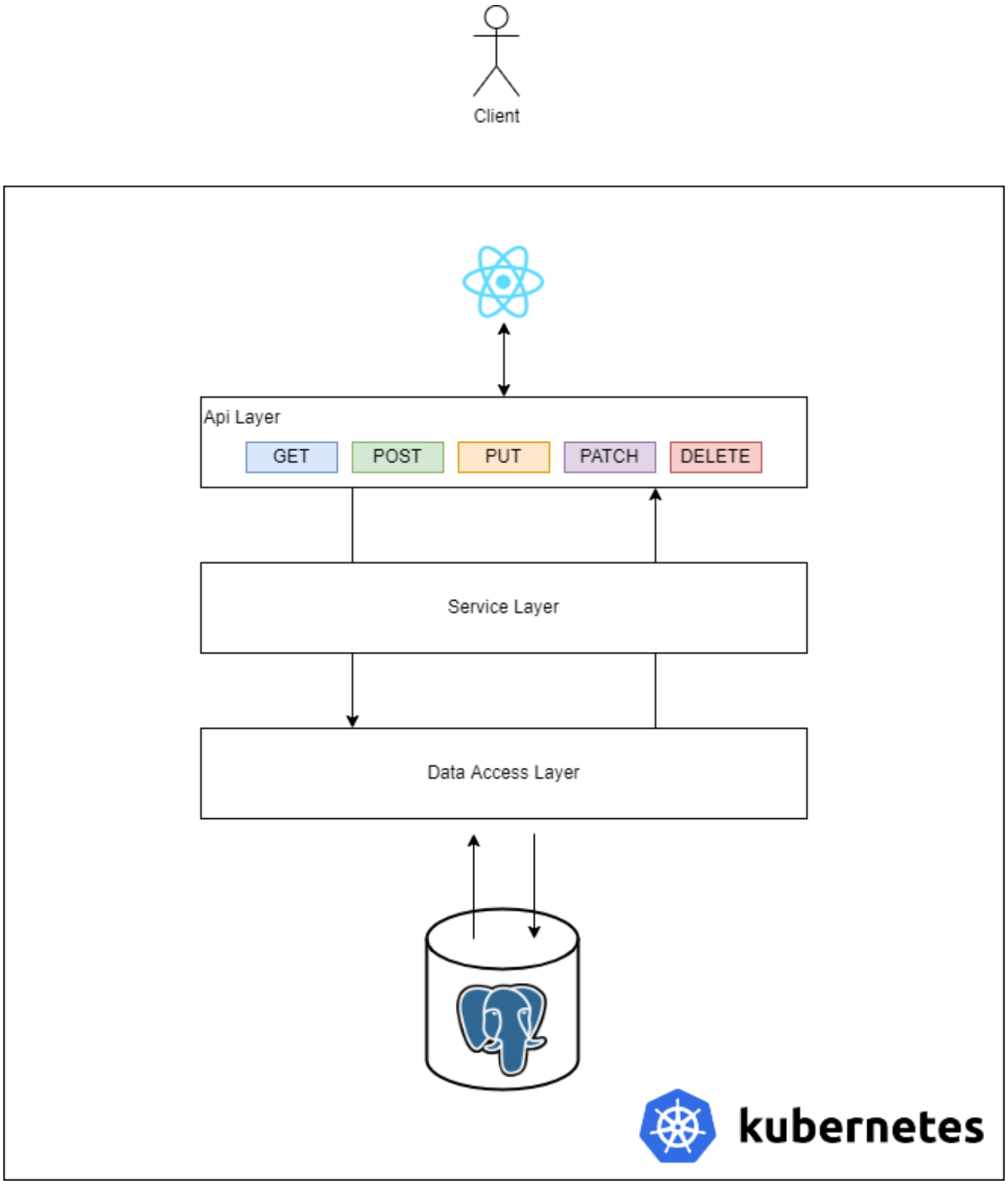
Verze stěžejních knihoven a frameworků použitých pro frontend viz tabulka 1.

Název	Verze
React	17.0.2
typescript	4.4.2
redux	4.1.2
redux-persist	6.0.0
reduxjs/toolkit	1.8.1
axios	0.26.0
dotenv	16.0.0
react-router-dom	6.2.2
mui/material	5.5.2

Tab. 1 - Verze knihoven / frameworků pro frontend.

Architektura backendu

Architekturu backendu s naznačeným propojením s frontendem, zabalením do nástroje Kubernetes a interakcí s klientem ukazuje obr. 3.



Obr. 3 - Architektura backendu.

Technologie

Pro backend byl zvolen framework programovacího jazyka Java Spring Boot. Jako nástroj pro build je používán Apache Maven. Jakožto databázový systém byl zvolen PostgreSQL. Pro propojení frontendu a backendu je využíváno specifikace OpenAPI, která popisuje rozhraní REST API (zobrazuje např. dostupné endpointy a operace k daným endpointům). Verze použitých technologií a závislostí viz tabulka 2.

Název	Verze
Java	11
Spring Boot Starter Web	2.6.4
Spring Boot Configuration Processor	2.6.4
Spring Boot Starter Test	2.6.4
Spring Boot Starter Data JPA	2.6.4
Spring Boot Starter Validation	2.6.4
Spring Boot Starter Security	2.6.4
Spring Security Test	5.6.2
Lombok	1.18.22
H2 Database Engine	2.1.210
PostgreSQL JDBC Driver	42.3.3
Apache Commons CSV	1.9.0
Java JWT	3.19.1
Guava: Google Core Libraries For Java	31.1-jre
Springdoc OpenAPI UI	1.6.7
PostgreSQL	14.2

Tab. 2 - Verze technologií a závislostí pro backend.

Architektura

Backend aplikace se skládá ze tří hlavních vrstev – datové, servisní a prezentační vrstvy. Každá vrstva má danou vlastní funkcionalitu a může volat / být volána jinými vrstvami.

Datová vrstva (Data Access Layer) slouží jako spojení k databázi – ukládá, mění, maže a získává data z databáze. Obsahuje tedy hlavně SQL příkazy pro práci s databází.

K datové vrstvě má přístup servisní vrstva (Service Layer). Tato vrstva obsahuje hlavní logiku aplikace, např. mění stav entit, volá datovou vrstvu pro získání entit z databáze, převádí entity

na DTO (data transfer object – objekty, se kterými pracuje prezentační vrstva) a naopak, vrací výsledek prezentační vrstvě.

Prezentační vrstva (Api Layer) zpracovává požadavky od frontendu (konkrétně GET, POST, PUT, PATCH a DELETE), volá servisní vrstvu (a tedy hlavní funkce aplikace), výsledek vrací zpět frontendu. V případě chyby při špatném požadavku vrací chybovou hlášku a příslušný HTTP stavový kód.

Aplikace též obstarává bezpečnost Určuje, které endpointy jsou volně přístupné, pro které je nutné být přihlášen a pro které je nutné mít specifické právo nebo roli ADMIN. Autentizace i autorizace je zajištěna pomocí JWT (JSON Web Token), který má v sobě uložené potřebné informace (uživatelské jméno, práva). Tento token získá uživatel při přihlášení a musí jej poslat zpět backendu spolu s požadavky.

Nasazení

Pro nasazení je využíván open source software Kubernetes. Na clusteru spravovaném univerzitou byl vytvořen namespace, kde je aplikace nasazena. Aplikace se skládá ze 3 deploymentů, pro které běží vždy jeden Pod a jedna Service. Jeden deployment je pro databázi, jeden pro backend aplikace a jeden pro frontend.

Service zajišťují přístupový bod k jednotlivým deploymentům. Aby byl frontend přístupný i mimo cluster, je její typ nastaven na LoadBalancer. Přesměrování requestů na backend je zajištěno nginx serverem, na kterém běží frontend, a requesty začínající */api* přepošle do Service pro backend.