FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Technická dokumentace aplikace

KIV/ASWI, KIV/TSP1, KIV/TSP2 – Týmový projekt

Vypracovali: Tomáš Zikmund, Viktorie Pavlíčková,
František Kaiser, Michal Schwob (*One team to rule them all*)

Datum: 29. 5. 2023 (v1.0)

Obsah

Technologie a architektura

Sekce je dedikována použitým technologiím a softwarovému modelu (architektuře).

Použité technologie

Při tvorbě mobilní aplikace byly využity následující technologie. Multiplatformní open-source softwarový rámec React Native umožnil psát aplikaci jedním jazykem (JavaScript/TypeScript) a spouštět ji na různých platformách. Pro spouštění a testování byl zvolen nástroj Expo, který poskytuje sadu nástrojů pro vývojáře a usnadňuje spouštění aplikace na emulátorech či fyzických zařízeních.

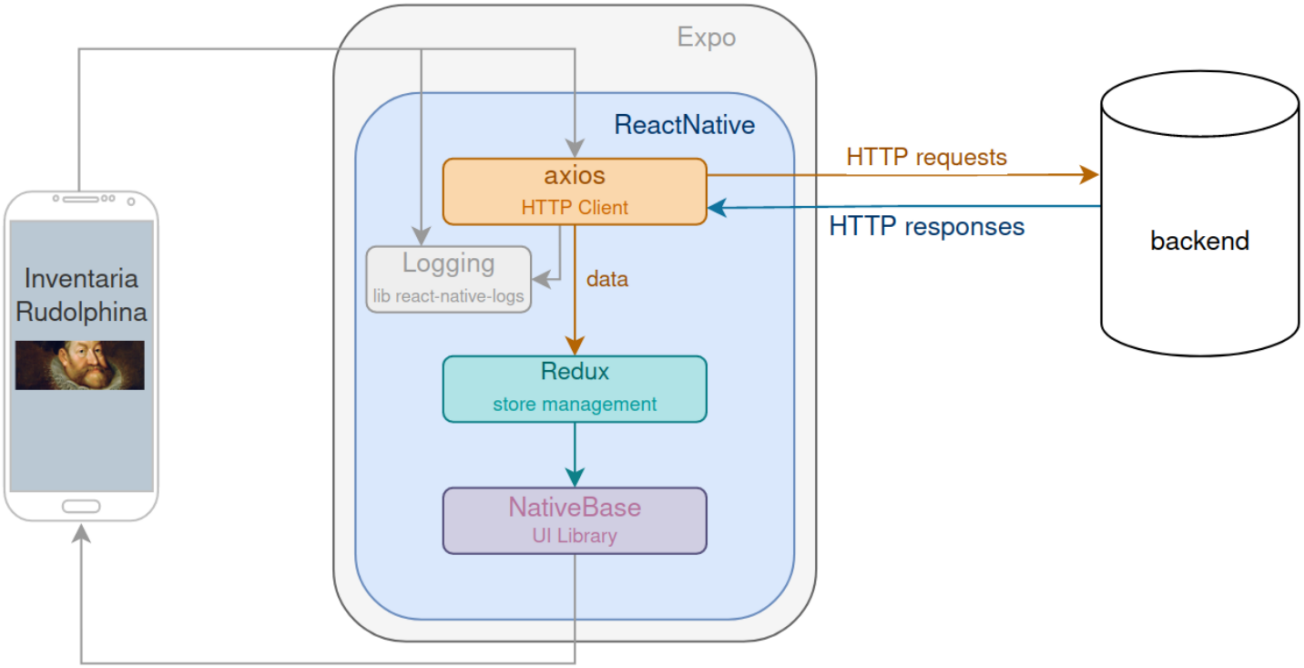
Uživatelské rozhraní bylo vyvinuto pomocí UI knihovny NativeBase, která poskytuje sadu předpřipravených komponent pro snadnou tvorbu rozhraní. Stavová správa aplikace byla realizována s využitím knihovny Redux, která efektivně řídí stavová data. Pro provádění HTTP požadavků byla implementována knihovna axios, která usnadňuje komunikaci s backendovými službami. Pro logování událostí v aplikaci byla využita knihovna react-native-logs, která umožňuje zaznamenávat a sledovat logy vývojového prostředí.

Níže následuje přehled uplatněných technologií/nástrojů společně s korespondující verzí:

Tabulka 1 - Přehled všech dependencies

Název:	**Verze:**
react-navigation/drawer	6.6.2
react-navigation/native	6.1.6
react-navigation/native-stack	6.9.12
reduxjs/toolkit	1.9.3
axios	1.3.4
expo	48.0.9
expo-status-bar	1.4.4
native-base	3.4.28
react	18.2.0
react-dom	18.2.0
react-native	0.71.6
react-native-deck-swiper	2.0.13
react-native-gesture-handler	2.9.0
react-native-logs	5.0.1
react-native-multiple-select	0.5.12
react-native-reanimated	2.14.4
react-native-safe-area-context	4.5.0
react-native-screens	3.20.0
react-native-svg	13.4.0
react-native-tab-view	3.5.1
react-native-vector-icons	9.2.0
react-redux	8.0.5
redux	4.2.1
typescript	4.9.4
babel/core	7.20.0
tsconfig/react-native	2.0.3
types/jest	29.5.0
types/react	18.0.28
types/react-test-renderer	18.0.0

Architektura



Obrázek 1 - Aplikační architektura (SW model)

Při návrhu aplikace byla uplatněna existující implementace backendu desktopové verze aplikace. Zadavatelem byla poskytnuta vývojářská verze a server následně zprovozněn na virtuálu skrze nuada.zcu.cz (*OpenNebula*), jež je spravován CIV.

Při vývoji aplikace byl využit multiplatformní open-source softwarový rámec React Native, který umožňuje vytvářet mobilní aplikace pro různé platformy, jako je iOS a Android, s použitím jednoho kódu. Pro spouštění a testování aplikace byl zvolen nástroj Expo, který poskytuje vývojářům sadu nástrojů a služeb pro usnadnění vývoje a testování React Native aplikací.

Pro tvorbu uživatelského rozhraní byla použita UI knihovna *NativeBase*. Tato knihovna poskytuje sadu předdefinovaných komponent, stylování a funkcí, které umožňují rychlé a jednoduché vytváření uživatelských rozhraní v React Native.

Za účelem správy stavu aplikace byla využita knihovna *Redux*. *Redux* je knihovna pro správu stavu aplikace, která umožňuje centralizovaně ukládat a aktualizovat stav aplikace. Tímto způsobem lze snadno sdílet data mezi komponentami a zajišťovat konzistentní stav v celé aplikaci.

Pro zajištění komunikace mezi uživatelem a serverem (tedy provádění HTTP požadavků) byla použita knihovna axios. Jedná se o jednoduchou a elegantní knihovna pro provádění HTTP požadavků z React Native aplikace. Poskytuje možnosti jako nastavování hlaviček, zpracování odpovědí a správu stavu požadavků.

Logování událostí a chyb je umožněno skrze knihovnu *react-native-logs*. Tato knihovna poskytuje nástroje pro zaznamenávání logů z React Native aplikace, což umožňuje jednoduše sledovat a analyzovat události a chyby v průběhu provozu aplikace.

Díky kombinaci výše uvedených nástrojů a knihoven bylo možné efektivně vyvíjet a testovat mobilní aplikaci v rámci React Native prostředí, zajišťovat správu stavu, provádět HTTP požadavky a logovat události pro snadnou analýzu a opravu chyb.

Struktura repositáře

Struktura aplikačního repositáře se skládá z **kořenového** adresáře a dvou zanořených adresářů – ***assets*** a ***src***. Všechny tyto adresáře obsahují zdrojové soubory různých formátů – od JSON, přes JavaScript až po TypeScript. Níže následuje podrobná programátorská dokumentace všech zdrojových souborů:

Kořenový adresář

Soubor app.json

Obsahuje konfiguraci pro aplikaci Expo. Název aplikace je "RudolfII" a slug je také zároveň "RudolfII". Verze aplikace je "1.0.0" a orientace je nastavena na "portrait" (režim svislého zobrazení). Ikona aplikace se nachází v souboru "./assets/icon.png". Uživatelské rozhraní je nastaveno na světlý režim. Splash obrazovka je definována souborem "./assets/splash.png". Obrázek se přizpůsobuje obsahu pomocí režimu "contain" a pozadí má bílou barvu. Nastavení assetBundlePatterns zahrnuje všechny soubory v projektu.

Pro iOS je povolena podpora pro tablety. Pro Android je definována adaptivní ikona, která má přední obrazek v souboru "./assets/adaptive-icon.png" a bílé pozadí. Pro web je nastaven favicon na soubor "./assets/favicon.png".

Soubor babel.config.js

Soubor babel.config.js je konfigurační soubor pro Babel. Tato konkrétní konfigurace používá přednastavení "babel-preset-expo" a zásuvný modul "react-native-reanimated/plugin". Funkce v tomto souboru také zakazuje používání mezipaměti (cache) pro Babel.

Soubor App.tsx

Soubor App.tsx obsahuje kód pro základní konfiguraci a vytvoření kořenového komponentu aplikace. Nejprve je importován modul "react-native-gesture-handler", který slouží pro zachytávání a zpracování gest v aplikaci. Dále je importován modul StyleSheet z "react-native", který umožňuje definovat styly pro komponenty pomocí CSS-like syntaxe.

Následují importy dalších modulů, a to NativeBaseProvider a Box z "native-base". NativeBaseProvider je součástí knihovny NativeBase, která poskytuje sadu UI komponent pro React Native. Box je jedním z těchto komponentů. Dále je importován Provider z "react-redux", což je součást knihovny Redux pro správu stavu aplikace. Provider je komponenta, která zajišťuje propojení Redux store se stromem komponent. Je též importován modul store z "./src/stores/store", jež reprezentuje Redux store aplikace. Následuje import komponenty Navigation z "./src/components/Navigation", která slouží pro navigaci v aplikaci a objektu nativeBaseTheme z "./src/theme/nativeBaseTheme", který obsahuje nastavení vzhledu pro komponenty z knihovny NativeBase.

Funkce *App* je výchozím exportem tohoto souboru. Tato funkce vrací JSX kód, který reprezentuje kořenovou komponentu aplikace. V této kořenové komponentě je vložen Provider, který obaluje zbylou část aplikace a propojuje ji s Redux store.

Uvnitř Provider komponenty je umístěn NativeBaseProvider, který nastavuje vzhled komponent z knihovny NativeBase pomocí poskytnutého nativeBaseTheme.

V rámci NativeBaseProvider je vložena komponenta Navigation, která se stará o navigaci mezi obrazovkami aplikace. Nakonec je definován objekt styles pomocí metody StyleSheet.create. Tento objekt obsahuje definici stylů pro komponentu, v tomto případě pro container. Styly určují flexibilitu, barvu pozadí, zarovnání a justifikaci komponenty.

Soubor tsconfig.json

Slouží k nastavení kompilátoru TypeScript a definování jeho parametrů. Soubor začíná objektem s dvěma klíči: "extends" a "compilerOptions".

- Klíč "extends" určuje, že se má použít rozšíření "expo/tsconfig.base". Tímto způsobem lze rozšířit existující konfiguraci TypeScriptu a využít přednastavená nastavení specifická pro Expo framework.
- Klíč "compilerOptions" obsahuje podklíč "strict", který je nastaven na hodnotu "true". Tento parametr zajišťuje přísnou kontrolu typů při kompilaci. Když je nastaven na hodnotu "true", TypeScript bude vyžadovat, aby byly všechny typy přesně deklarovány a použity v kódu. Tímto se minimalizují možné chyby a zvýší se bezpečnost a spolehlivost aplikace.

Celkově soubor tsconfig.json definuje, že pro kompilaci TypeScript kódu v rámci Expo projektu se má použít přednastavená konfigurace "expo/tsconfig.base" a přísná kontrola typů je povolena. Tím se zajistí správná kompilace a typová kontrola v rámci projektu.

Soubor package.json

Jedná se o konfigurační soubor, který obsahuje informace o projektu a jeho závislostech. Soubor začíná objektem s různými klíči, které určují různé vlastnosti projektu.

- Klíč "name" definuje název projektu jako "rudolfii". Jedná se o unikátní identifikátor, který slouží k identifikaci projektu.
- Klíč "version" udává verzi projektu jako "1.0.0". Verze umožňuje sledovat změny a aktualizace projektu.
- Klíč "main" určuje hlavní vstupní bod aplikace, který je nastaven na "node_modules/expo/AppEntry.js". Tento soubor je spuštěn při spuštění aplikace.
- Klíč "scripts" obsahuje různé skripty pro spouštění různých příkazů v rámci projektu. Skripty jsou definovány jako klíč-hodnota páry, kde klíč je název skriptu a hodnota je příkaz, který se spustí. Zde jsou definovány skripty pro spuštění aplikace v různých prostředích, jako "start", "android", "ios" a "web".
- Klíč "dependencies" obsahuje seznam závislostí projektu. Jedná se o externí knihovny a balíčky, které jsou potřebné pro běh aplikace. Každá závislost je definována jako klíč-hodnota pár, kde klíč je název závislosti a hodnota je verze této závislosti (podrobnosti viz: Použité technologie).
- Klíč "devDependencies" obsahuje seznam vývojových závislostí projektu. Tyto závislosti jsou potřebné pouze během vývoje aplikace a nejsou součástí výsledného produktu. Opět jsou definovány jako klíč-hodnota páry, kde klíč je název závislosti a hodnota je verze této závislosti (podrobnosti viz: Použité technologie).
- Klíč "private" je nastaven na hodnotu "true", což označuje, že tento projekt je soukromý a není určen k publikaci jako balíček.

Celkově soubor package.json obsahuje důležité informace o projektu, jako název, verzi, seznam závislostí a skripty pro spouštění příkazů v rámci projektu. Tyto informace jsou důležité pro správné nastavení a provoz aplikace.

Adresář *assets*

Složka *assets* v rámci projektu obsahuje několik rastrových obrázků, které slouží pro různé účely. Následuje popis jednotlivých obrázků:

- "adaptive-icon.png": Tento obrázek představuje adaptační ikonu, která se používá pro dynamické přizpůsobení ikony aplikace v závislosti na zařízení, operačním systému nebo nastavení uživatele. Adaptační ikona je často používána na mobilních zařízeních s různými velikostmi a tvary ikon.

- "favicon.png": Tento obrázek slouží jako favicon, což je malá ikona zobrazená na v záložkách webového prohlížeče nebo v panelu záložek. Favikona se nejčastěji zobrazuje v adresním řádku, na panelu se stránkou, v nabídce záložek/oblíbených či mezi aplikacemi na displeji mobilního zařízení.
- "icon.png": Tento obrázek představuje ikonu aplikace. Ikona aplikace je zobrazena na ploše, v seznamu aplikací nebo v liště úloh a slouží k identifikaci a spuštění aplikace.
- "splash.png": Tento obrázek je využíván při zobrazování úvodního obrazovky nebo načítacího obrazovky aplikace. Splash obrázek je často používán k vytvoření vizuálně přitažlivého prvního dojmu při spouštění aplikace.

Složka "assets" je tedy umístění, ve kterém jsou uloženy tyto rastrové obrázky, které slouží jako grafické prvky aplikace, jako ikony, favicony a načítací obrazovky.

Adresář *src*

Podadresář *api*

Soubor api.ts

Soubor nese zodpovědnost za konfiguraci instance axiosu, která se používá pro komunikaci se serverem pomocí HTTP požadavků. První řádek kódu importuje knihovnu axios, která poskytuje funkcionality pro provádění HTTP požadavků. Druhý řádek importuje konstantu BASE_URL ze souboru constants.ts, která obsahuje URL adresu základního rozhraní API. Poté je vytvořena instance axiosu pomocí metody create(). Tato instance je přiřazena do konstanty axiosInstance.

Nastavení instance axiosu zahrnuje několik klíčových vlastností:

- "baseURL" definuje základní URL adresu serveru, ke kterému budou odesílány HTTP požadavky. Tato hodnota je získána z konstanty BASE_URL, která byla importována ze souboru constants.ts.
- "withCredentials" je nastaveno na hodnotu "true". Tato vlastnost určuje, zda se mají s každým HTTP požadavkem posílat přihlašovací údaje (credentials) ve formě souborů cookie nebo autentizačních tokenů.

Tímto způsobem je instance axiosu nakonfigurována a připravena k použití pro komunikaci se serverem. Tato instance bude použita pro provádění různých typů HTTP požadavků, jako je GET, POST, PUT, DELETE atd. Vytvoření instance s konkrétními nastaveními umožňuje konzistentní komunikaci se serverem a poskytuje flexibilitu v případě změn nebo rozšíření komunikačních požadavků.

Soubor authservice.ts

Obsahuje funkce pro autentizaci pomocí HTTP požadavků na server. Soubor začíná importem axiosInstance ze souboru api.ts. Tato instance axiosu je již nakonfigurována a připravena pro komunikaci se serverem.

Následují dvě funkce:

- loginRequest - Tato funkce provádí asynchronní HTTP POST požadavek na cílovou URL "/login" na serveru. Přijímá dva parametry: "username" (uživatelské jméno) a "password" (heslo). Tyto údaje jsou zasílány ve formě JSON objektu v těle požadavku. Funkce vrací asynchronní odpověď získanou z axiosInstance.post().
- isAuthRequest - Tato funkce provádí asynchronní HTTP GET požadavek na cílovou URL "/api/isauth" na serveru. Tato URL slouží k ověření, zda je uživatel autentizován. Funkce nevyžaduje žádné parametry a vrací asynchronní odpověď získanou z axiosInstance.get().

Obě funkce využívají předem nakonfigurovanou instanci axiosu (axiosInstance) pro provádění HTTP požadavků. Tímto způsobem je zajištěna konzistence využívaných komunikačních parametrů a URL adres. Funkce používají metody z axiosu pro odesílání požadavků a získávání odpovědí. Tyto funkce mohou být použity ve vaší aplikaci pro autentizaci uživatele a ověřování jeho stavu pomocí HTTP požadavků na server.

Soubor constants.ts

Skládá se z deklarace konstantních hodnot. Soubor začíná s exportem konstanty BASE_URL, která je nastavena na hodnotu 'http://147.228.173.159/api'. Tato konstanta slouží jako základní adresa (URL) pro komunikaci s API serverem. Využívá se v různých částech aplikace, kde je potřeba navázat spojení s tímto serverem.

Následuje export konstanty AVATAR_URL, která je nastavena na hodnotu 'http://147.228.173.159/static/avatars'. Tato konstanta slouží jako adresa pro získávání avatary uživatelů ze statických souborů na serveru. Předpokládá se, že na této adrese jsou uloženy avatarové obrázky uživatelů.

Dále je exportována konstanta IMAGE_URL, která je nastavena na hodnotu 'http:147.228.173.159/static/images'. Tato konstanta slouží jako adresa pro získávání obecných obrázků ze statických souborů na serveru. Předpokládá se, že na této adrese jsou uloženy různé obrazové soubory, které jsou používány v aplikaci. Tyto konstanty slouží k centralizaci a jednoduchému upravování URL adres, které se v aplikaci používají pro komunikaci se serverem. Díky tomu je možné jednoduše měnit adresy serveru nebo adresy statických souborů bez potřeby úprav v mnoha částech aplikace.

Soubor itemservice.ts

Obsahuje implementaci funkcí pro komunikaci s API týkající se položek (items).

Soubor začíná s importem axiosInstance z modulu "./api". Tento import zajišťuje, že se využívá instance axios s konfigurací, která byla vytvořena v souboru api.ts. Tato instance je již připravena s baseURL a dalšími nastaveními pro komunikaci s API serverem.

Následuje definice funkce getItemRequest, která je asynchronní a přijímá jeden parametr itemId typu string. Tato funkce zajišťuje získání detailů položky pomocí HTTP GET požadavku na konkrétní URL cesty /item/${itemId}. Výsledkem je promise, která vrátí odpověď z API serveru. Komentářem je zakomentován kód funkce getItemConcordancesRequest. Tato funkce není momentálně používána a je tedy zakomentována. Původně měla sloužit k získání souvisejících informací k dané položce pomocí HTTP GET požadavku na URL cestu /api/concordances/${itemId}.

Soubor itemservice.ts tedy poskytuje rozhraní pro získávání detailů položek z API serveru. Funkce v tomto souboru využívají předem vytvořenou instanci axios pro provádění HTTP požadavků na správné URL cesty.

Soubor noteservice.ts

Zahrnuje implementaci funkcí pro komunikaci s API týkající se poznámek (notes).

Soubor je započat importem typu SortOptions z modulu "../types/general" a typu Note z modulu "../types/note". Tyto importy představují definice typů používaných v implementovaných funkcích. Dále je importována axiosInstance z modulu "./api". Tento import zajišťuje, že se využívá instance axios s konfigurací, která byla vytvořena v souboru api.ts. Tato instance je již připravena s baseURL a dalšími nastaveními pro komunikaci s API serverem.

Následuje definice funkce getItemNotesRequest, která je asynchronní a přijímá jeden parametr itemId typu string. Tato funkce zajišťuje získání poznámek pro danou položku pomocí HTTP GET požadavku na konkrétní URL cestu /notes/?item_id[]=${itemId}. Výsledkem je promise, která vrátí odpověď z API serveru.

Další definovaná funkce je updateNoteRequest, která je asynchronní a přijímá jeden parametr note typu Note. Tato funkce provádí aktualizaci existující poznámky pomocí HTTP PUT požadavku na URL cestu /note a předávaného objektu note. Výsledkem je promise, která vrátí odpověď z API serveru.

Následuje funkce createNoteRequest, která je asynchronní a přijímá jeden parametr note typu Note. Tato funkce slouží k vytvoření nové poznámky pomocí HTTP POST požadavku na URL cestu /note a předávaného objektu note. Výsledkem je promise, která vrátí odpověď z API serveru.

Poté je definována funkce deleteNoteRequest, která je asynchronní a přijímá jeden parametr noteId typu string. Tato funkce slouží k odstranění existující poznámky pomocí HTTP DELETE požadavku na konkrétní URL cestu /note/${noteId}. Výsledkem je promise, která vrátí odpověď z API serveru.

Poslední definovaná funkce je getAllNotesRequest, která je asynchronní a přijímá tři parametry: myComments typu boolean, generalComments typu boolean a sortOptions typu objektu s vlastnostmi items a date typu SortOptions. Tato funkce slouží k získání všech poznámek pomocí HTTP GET požadavku na dynamicky vytvářenou URL cestu na základě předaných parametrů. Výsledkem je promise, která vrátí odpověď z API serveru.

Soubor noteservice.ts tedy poskytuje rozhraní pro manipulaci s poznámkami v API serveru. Funkce v tomto souboru využívají předem vytvořenou instanci axios pro provádění HTTP požadavků na správné URL cesty.

Soubor searchFormService.ts

Obsaženy jsou implementace funkcí pro komunikaci s API týkající se načítání dat pro vyhledávací formulář. První řádek představuje import axiosInstance z modulu "./api". Tento import zajišťuje, že se využívá instance axios s konfigurací, která byla vytvořena v souboru api.ts. Tato instance je již připravena s baseURL a dalšími nastaveními pro komunikaci s API serverem.

Následuje definice funkce fetchInventoriesRequest, která je asynchronní a nevyžaduje žádné parametry. Tato funkce zajišťuje načítání inventářů pomocí HTTP GET požadavku na URL cestu "/inventories". Výsledkem je promise, která vrátí odpověď z API serveru.

Dále jsou definovány funkce pro načítání konkrétních typů dat. Například funkce fetchCountriesRequest se stará o načítání seznamu zemí pomocí HTTP GET požadavku na URL cestu "/form/country". Obdobně fungují i ostatní funkce jako fetchInstitutionsRequest, fetchCitiesRequest, fetchNationalitiesRequest, fetchTechniquesRequest, fetchSubjectsRequest, fetchArtistNamesRequest a fetchPlanRequest. Každá z těchto funkcí provádí odpovídající HTTP GET požadavek na specifickou URL cestu a vrátí odpověď z API serveru.¨

Tyto funkce slouží k načítání různých datových sad, které se používají ve vyhledávacím formuláři, například seznam zemí, institucí, měst, národností, technik, předmětů, jmen umělců nebo seznam plánů. Celý soubor searchFormService.ts poskytuje rozhraní pro komunikaci s API a načítání potřebných dat pro vyhledávací formulář.

Soubor searchService.ts

Zahrnuje implementaci funkcí pro provádění vyhledávání pomocí API.

Prvotní je import axiosInstance z modulu "./api". Tento import zajišťuje, že se využívá instance axios s konfigurací, která byla vytvořena v souboru api.ts. Tato instance je již připravena s baseURL a dalšími nastaveními pro komunikaci s API serverem.

Dále je importována funkce log z modulu "../logging/logger", která slouží k logování informací. Následuje definice rozhraní SearchParams, které obsahuje různé parametry pro vyhledávání. Tyto parametry zahrnují inventáře, místnosti, umělce, národnosti, předměty, techniky a další. Každý parametr má odpovídající typ.

Poté je definována funkce composeSearchParams, která slouží k sestavení parametrů vyhledávání pro URL. Tato funkce přijímá pole a identifikátor a vytváří z nich část URL, která odpovídá danému parametru. Pokud pole není prázdné, prochází se všechny prvky pole a vytváří se jejich reprezentace v URL formátu. Následuje definice funkce searchRequest, která provádí vyhledávání. Tato funkce přijímá objekt SearchParams obsahující všechny potřebné parametry pro vyhledávání. Funkce sestavuje URL pro vyhledávací požadavek pomocí předaných parametrů a volá funkci axiosInstance.get s tímto URL. Výsledkem je *promise*, která vrátí odpověď z API serveru.

Podadresář *components*

Soubor Navigation.tsx

Soubor "Navigation.tsx" obsahuje implementaci navigace pro aplikaci pomocí React Navigation. Zde je popis toho, jak soubor funguje:

Nejprve se importují potřebné závislosti, jako je funkce `createNativeStackNavigator` pro vytvoření zásobníku navigace, `NavigationContainer` pro obalování navigačních komponent, `IconButton` z knihovny "native-base" pro zobrazení tlačítka, `AntDesign` z knihovny "@expo/vector-icons" pro použití ikon, a další.

Dále je uvedna definice typu `RootDrawerParamList`, který reprezentuje seznam možností pro hlavní navigační panel. Tyto možnosti zahrnují různé obrazovky, jako je "Home" (domovská obrazovka), "Search" (obrazovka vyhledávání), "Logout" (obrazovka odhlášení), "Item" (obrazovka položky s předáním identifikátoru položky), "Login" (obrazovka přihlášení) a "Notes" (obrazovka poznámek).

Funkce `Navigation` je komponenta, která vytváří navigační strukturu aplikace. V této funkci je vytvořen instance `DrawerNavigator` pro vytvoření navigačního panelu typu "drawer", který poskytuje boční menu pro navigaci mezi různými obrazovkami. Dále je pomocí hooku `useSelector` z knihovny "react-redux" získána informace o přihlášení uživatele z globálního stavu aplikace uloženého v Redux store.

Komponenta vrací kontejner `<NavigationContainer>` pro obalování navigačních komponent. Uvnitř kontejneru jsou definovány různé obrazovky, které jsou zobrazeny v závislosti na přihlášení uživatele. Pokud je uživatel přihlášen (`loggedIn` je `true`), zobrazí se obrazovky "Home", "Search", "Notes" a "Logout". Pokud uživatel není přihlášen, zobrazí se pouze obrazovka "Login". Pro každou obrazovku jsou nastaveny různé konfigurace, jako je název obrazovky, komponenta, možnosti zobrazení v hlavičce, atd.

Nakonec je komponenta `Navigation` exportována jako výchozí export souboru. Celkově soubor "Navigation.tsx" obsahuje implementaci navigace pro aplikaci pomocí React Navigation. Navigační panel umožňuje uživatelům pohodlně přecházet mezi různými obrazovkami v závislosti na jejich přihlášení.

Podadresář *general*

Soubor Dialog.tsx

Soubor "Dialog.tsx" obsahuje implementaci dialogového okna pro zobrazení potvrzovacího dialogu. Nejprve se importují potřebné závislosti, jako je komponenta AlertDialog a Button z knihovny "native-base" pro zobrazení dialogového okna a tlačítek.

Funkce ConfirmDialog je komponenta, která přijímá několik vlastností jako vstupní parametry, včetně isShown (určuje, zda je dialogové okno viditelné), headerText (text zobrazený v záhlaví dialogového okna), bodyText (text zobrazený v těle dialogového okna), confirmText (text tlačítka pro potvrzení), confirmColor (barva tlačítka pro potvrzení), cancelRef (reference na tlačítko pro zrušení dialogu), onClose (funkce pro zavření dialogového okna) a onSubmit (funkce pro odeslání formuláře).

V komponentě se používá podmíněný renderování. Pokud je vlastnost isShown true, dialogové okno je zobrazeno. Uvnitř dialogového okna jsou definovány různé části, jako je záhlaví, tělo a zápatí. Obsah těla dialogového okna (bodyText) je zobrazen pouze tehdy, pokud je definován.

V zápatí dialogového okna jsou umístěna tlačítka "Cancel" (zrušení) a "Confirm" (potvrzení). Tlačítko "Cancel" volá funkci onClose a je spojeno s referencí cancelRef. Tlačítko "Confirm" volá funkci onSubmit a má nastavenou barvu confirmColor. Oba prvky jsou součástí skupiny tlačítek definovaných pomocí komponenty Button.Group. Celá komponenta je obalena v prázdných značkách <>...</>. Tím je umožněno vykreslování komponenty v závislosti na hodnotě vlastnosti isShown.

Celkově soubor "Dialog.tsx" poskytuje implementaci dialogového okna pro zobrazení potvrzovacího dialogu. Toto dialogové okno umožňuje uživatelům potvrdit nebo zrušit akci pomocí tlačítek "Confirm" a "Cancel".

Soubor Icons.tsx

Soubor "Icons.tsx" obsahuje implementace dvou ikon: MessageIcon a EditIcon. Tyto ikony slouží k vizuálnímu zobrazení různých akcí nebo funkcí ve webové aplikaci.

V souboru jsou použity importy z knihoven "native-base" a "react-native-svg", které umožňují používat komponenty pro ikony a SVG prvky.

Komponenta MessageIcon je definována jako funkční komponenta, která přijímá vstupní parametr color, což je barva ikony. Vnitřně je použita komponenta Icon z knihovny "native-base", která slouží jako kontejner pro SVG ikonu. Uvnitř komponenty Icon jsou použity prvky G a Path z knihovny "react-native-svg", které definují tvar ikony pomocí atributu d. Vzhled ikony je nastaven pomocí atributů fill a stroke, které určují vyplnění a obrys ikony.

Komponenta EditIcon má obdobnou strukturu jako MessageIcon a slouží k vykreslení ikony pro úpravu. I zde je možné nastavit barvu ikony pomocí vstupního parametru *color*.

Tyto implementace ikon umožňují jednoduché použití ikon v rámci webové aplikace a poskytují možnost různých vizuálních akcí a funkcionalit.

Podadresář item

Soubor ItemTabBar.tsx

Soubor "ItemTabBar.tsx" obsahuje implementaci komponenty `ItemTabBar`, která slouží k zobrazení navigačního panelu pro záložky položky.

V souboru jsou importovány různé komponenty. Komponenta `ItemTabBar` je definována jako funkční komponenta, která přijímá vstupní parametry `navigationState`, `concordances`, `index`, `onIndexChange` a `onBackPressed`. Tato komponenta zobrazuje navigační panel pro záložky položky na základě poskytnutých dat.

V závislosti na existenci záznamů v `concordances` je vykreslen buď kompletní navigační panel pro záložky nebo komponenta `LoadingBox`, která zobrazuje zprávu o načítání.

Navigační panel je implementován pomocí komponent `HStack`, `Button`, `ScrollView`, `Pressable`, `Text` a dalších. Záložky jsou vykresleny dynamicky na základě dat v `navigationState.routes` a `concordances`. Každá záložka obsahuje ikonu, identifikátor a reaguje na kliknutí uživatele. Uživatel může přepínat mezi záložkami pomocí tlačítek s ikonami `ChevronLeftIcon` a `ChevronRightIcon`. Komponenta také obsahuje tlačítko "Back" pro návrat na předchozí stránku.

Tímto způsobem umožňuje komponenta `ItemTabBar` pohodlnou navigaci mezi záložkami položky a poskytuje uživateli interaktivní prostředí pro prohlížení obsahu.

Soubor ItemView.tsx

Zdrojový soubor obsahuje implementaci dvou komponent: `ItemDetail` a `ItemView`.

Na začátku souboru jsou importovány různé komponenty z knihovny "native-base". Dále jsou importovány typy `Item` a `Note` z jiných souborů a komponenty `LoadingBox` a `NotesListView` z dalších souborů.

Komponenta `ItemDetail` je definována jako funkční komponenta, která přijímá vstupní parametr `item`. Tato komponenta zobrazuje detaily položky na základě poskytnutých dat. V závislosti na existenci `item` se vykreslí kompletní detail položky, který obsahuje informace o autorovi, názvu díla, konkordanci, inventární položce, obrázku a dalších popisných údajích.

Komponenta `ItemView` je také funkční komponenta, která přijímá vstupní parametry `item`, `itemLoading`, `notes`, `notesLoading` a `navigation`. Tato komponenta slouží k zobrazení zobrazení položky a poznámek. V závislosti na aktuálním stavu (`itemShown`) se buď zobrazí detail položky pomocí komponenty `ItemDetail` nebo se zobrazí seznam poznámek pomocí komponenty `NotesListView`. Pokud je položka nebo poznámky ještě ve fázi načítání, zobrazí se komponenta `LoadingBox`. Uživatel může přepínat mezi zobrazením položky a poznámek pomocí tlačítek "Item" a "Notes".

Podadresář listView

Soubor ItemPreview.tsx

Soubor zahrnuje implementaci komponenty `ItemPreview`.

Na začátku souboru jsou importovány některé komponenty z knihovny "native-base", jako například `Center`, `HStack`, `Image`, `Pressable`, `Text` a `VStack`. Dále je importována funkce `useEffect` z Reactu a typ `DrawerScreenProps` z "@react-navigation/drawer" knihovny. Také je importován typ `RootDrawerParamList` z jiného souboru s navigací.

Komponenta `ItemPreview` je definována jako funkční komponenta, která přijímá vstupní parametry `caption`, `title`, `name`, `image`, `itemId` a `navigation`. Tato komponenta slouží k zobrazení náhledu položky. V rámci komponenty je použita funkce `useEffect`, která vypisuje hodnoty props do konzole v případě jejich změny.

Výstupem komponenty je JSX kód, který obsahuje tlačitelný prvek (`Pressable`), který slouží k navigaci na podrobnosti položky. Uvnitř tohoto prvku je umístěn horizontální kontejner (`HStack`), který

obsahuje obrázek položky, název, popisek a titulek. Obrázek je získáván z URL na základě poskytnutého názvu obrázku (`props.image`) nebo v případě neexistence je použit výchozí obrázek. Ostatní textové informace jsou zobrazovány pomocí komponenty `Text`.

Soubor ListView.tsx

Soubor "ListView.tsx" obsahuje implementaci komponenty `ListView`.

Na začátku souboru jsou importovány některé komponenty z knihovny "react" a "native-base", jako například `useEffect`, `useState`, `ScrollView`, `Text` a `VStack`. Dále je importována funkce `useSelector` z knihovny "react-redux" a typ `RootState` z jiného souboru obsahujícího Redux store. Importována je také komponenta `ItemPreview` a typ `ItemPreviewType` z jiného souboru s definicemi typů. Nakonec je importována funkce `log` z loggeru.

Komponenta `ListView` je definována jako funkční komponenta, která přijímá vstupní parametry `navigation`. Tato komponenta slouží k zobrazení seznamu položek. Uvnitř komponenty je použita funkce `useSelector`, která získává data ze stavu Redux store.

Ve funkci `ListView` jsou také definovány další pomocné funkce. Funkce `parseNameByRole` slouží k vyhledávání jmen podle dané role v rámci objektu položky. Funkce `parseArtists` zpracovává informace o umělcích a v závislosti na tom, zda se jedná o originální dílo nebo kopii, vrací odpovídající textový řetězec. Funkce `parseImage` slouží k získání názvu obrázku z položky.

Výstupem komponenty je JSX kód, který obsahuje mapování položek ze stavu Redux store pomocí metody `map`. Pro každou položku je vytvořena komponenta `ItemPreview`, které jsou předány odpovídající vstupní parametry jako `caption`, `title`, `name`, `image`, `itemId` a `navigation`.

Podadresář loading

Soubor LoadingBox.tsx

Implementuje komponenty `LoadingBox`. Na začátku souboru jsou importovány některé komponenty z knihovny "native-base", jako například `Center`, `Box`, `VStack`, `Button`, `HStack`, `Text`, `Image`, `ScrollView`, `Heading` a `Spinner`. Dále je importována knihovna "react".

Komponenta `LoadingBox` je definována jako funkční komponenta, která přijímá vstupní parametry `text` s typem `string`. Tato komponenta slouží k zobrazení informace o načítání nebo čekání s animovaným spinnerem a textovým popiskem.

Ve funkci `LoadingBox` je vytvořen JSX kód, který obsahuje horizontální kontejner (`HStack`) s centrováním a zarovnáním prvků na ose Y. V rámci tohoto kontejneru je umístěn komponenta `Spinner`, která zobrazuje animovaný spinner. Dále je zde komponenta `Heading`, která zobrazuje textový popisek načítání. Hodnota popisku je předána získáním vstupního parametru `text`.

Výstupem komponenty je zobrazení animovaného spinneru spolu s textovým popiskem načítání ve vertikálně a horizontálně zarovnaném kontejneru.

Podadresář notes

Soubor NotesListView.tsx

Zajišťuje implementaci komponenty `NotesListView`. Na začátku souboru jsou importovány některé komponenty z knihovny "native-base", jako například `Center`, `VStack`, `Box`, `Text`, `HStack` a `ScrollView`. Dále je importován typ `Note` ze souboru "../../types/note" a komponenta `NoteView` ze souboru "./NoteView".

Komponenta `NotesListView` je definována jako funkční komponenta, která přijímá vstupní parametry `notes` (pole poznámek), `handleReply` (funkce pro zpracování odpovědi na poznámku), `handleDelete` (funkce pro zpracování smazání poznámky), `handleEdit` (funkce pro zpracování úpravy poznámky), `setConfirmDialog` (funkce pro nastavení potvrzovacího dialogu) a `navigation` (objekt pro navigaci).

Ve funkci `NotesListView` je vytvořen JSX kód, který obsahuje kontejner `ScrollView`. Uvnitř tohoto kontejneru je umístěn ternární operátor, který zjišťuje, zda existují nějaké poznámky (`notes && notes.length > 0`). Pokud ano, je vytvořen vertikální kontejner `VStack`, který obsahuje iteraci přes všechny poznámky pomocí metody `map`. Pro každou poznámku je vytvořena instance komponenty `NoteView` s předáním vstupních parametrů z props a unikátním klíčem `key={index}`. Tím se vytváří seznam zobrazení poznámek. Pokud neexistují žádné poznámky, je zobrazeno textové zpráva "*There are no notes.*"

Výstupem komponenty je kontejner `ScrollView`, který buď zobrazuje seznam poznámek nebo zprávu o jejich absenci.

Soubor NoteView.tsx

Soubor "NoteView.tsx" obsahuje implementaci komponenty `NoteView`.

Na začátku souboru jsou importovány některé komponenty z knihovny "native-base. Dále jsou importovány další moduly, jako například typ `Note` ze souboru "../../types/note", komponenta `EditIcon` z "../general/Icons" a konstanta `AVATAR_URL` z "../../api/constants". Také je importována funkce `useState` z knihovny "react" a modul `React`.

Komponenta `NoteView` je definována jako funkční komponenta, která přijímá vstupní parametry `note` (poznámka), `handleReply` (funkce pro zpracování odpovědi na poznámku), `handleDelete` (funkce pro zpracování smazání poznámky), `handleEdit` (funkce pro zpracování úpravy poznámky), `setConfirmDialog` (funkce pro nastavení potvrzovacího dialogu) a `navigation` (objekt pro navigaci).

Ve funkci `NoteView` jsou destrukturovány vstupní parametry a inicializovány některé lokální stavy pomocí funkce `useState`. Stavy `isEdited` (určuje, zda je poznámka upravována) a `text` (obsahuje text poznámky) jsou nastaveny na počáteční hodnoty.

Dále jsou definovány pomocné funkce:

- `getDateString` přijímá časové razítko (timestamp) a vrací formátovaný řetězec datumu a času ve formátu "YYYY-MM-DD HH:MM".
- `getUsernameInitials` přijímá uživatelské jméno a vrací iniciály uživatele, které jsou získány z prvního písmene každého slova v jméně.
- `toggleEdited` je funkce, která přepíná stav `isEdited` mezi true a false a provádí další logiku v závislosti na hodnotě `isEdited`.
- `deleteClicked` je funkce, která zobrazuje potvrzovací dialog pro smazání poznámky.
- `saveEdit` je funkce, která ukládá provedené změny v poznámce po úpravě.

Ve výsledném JSX kódu komponenty je definována struktura zobrazení poznámky. Obsahuje horizontální kontejner `HStack`, který obsahuje avatara, vertikální kontejner `VStack` a další prvky pro zobrazení uživatelského jména, datumu a času vytvoření poznámky, textu poznámky, tlačítka pro navigaci na položky související s poznámkou, tlačítka pro odpověď, úpravu a smazání poznámky.

Avatara je zobrazen pomocí komponenty `Avatar` z knihovny "native-base" s použitím pozadí `note.noteColor` a zdroje obrázku definovaného v konstantě `AVATAR_URL` a připojeného s identifikátorem poznámky (`note.avatarUrl`).

V textových prvcích se zobrazují uživatelské jméno, datum a čas vytvoření poznámky a samotný text poznámky. Text poznámky může být buď v normálním zobrazení nebo v režimu úpravy, který se ovládá pomocí stavu `isEdited`. Pokud existují položky související s poznámkou (`note.items`), je zobrazeno tlačítko pro navigaci na první z těchto položek. Na konci komponenty jsou zobrazena tlačítka pro odpověď, úpravu a smazání poznámky. Zobrazení těchto tlačítek závisí na přítomnosti odpovídajících funkcí `handleReply`, `handleEdit` a `handleDelete` v props.

Celkově tato komponenta slouží k zobrazení jedné poznámky a umožňuje její úpravu, smazání a reakci na ni.

Podadresář search

Soubor MultiSelect.tsx

Soubor "MultiSelect.tsx" obsahuje implementaci komponenty `MultiSelect`.

Na začátku souboru jsou importovány některé komponenty z knihovny "native-base“. Dále jsou importovány moduly `FlatList` a `GestureResponderEvent` z knihovny "react-native". V neposlední řadě je importována komponenta `Ionicons` z knihovny "@expo/vector-icons" a funkce `log` z modulu "../../logging/logger".

Komponenta `MultiSelect` je definována jako funkční komponenta se vstupními parametry `data`, `label`, `selectedItems` a `onSelectedItemsChange`.

Ve funkci `MultiSelect` jsou destrukturovány vstupní parametry a inicializovány některé lokální stavy pomocí funkce `useState`. Stav `isSelected` je inicializován jako pole s délkou odpovídající délce `props.data` a obsahující hodnoty `false`. Stav `query` je inicializován prázdným řetězcem. Dále jsou pomocí funkce `useDisclose` inicializovány stavy `isOpen`, `onOpen` a `onClose`.

Pomocí funkce `useMemo` je definována proměnná `filteredData`, která obsahuje filtrovaná data z `props.data` na základě hodnoty `query`. Tato proměnná se aktualizuje pouze tehdy, když se změní hodnota `query` nebo `props.data`.

Funkce `updateSelected` slouží k aktualizaci stavu `isSelected` při změně výběru položek. Pomocí funkce `useEffect` se sleduje změna stavu `props.selectedItems` a provádí se aktualizace stavu `isSelected` na základě nových vybraných položek. Funkce `onCancel` slouží k uzavření komponenty `Actionsheet` po stisknutí tlačítka "Cancel". Funkce `onDone` se volá po stisknutí tlačítka "Done" a aktualizuje vybrané položky na základě stavu `isSelected` a volá funkci `props.onSelectedItemsChange` s novým seznamem vybraných položek.

Funkce `unselectItem` slouží k odstranění vybrané položky ze seznamu vybraných položek.

V JSX kódu komponenty jsou zobrazeny vybrané položky, tlačítko pro otevření `Actionsheet` a samotná komponenta `Actionsheet`, která obsahuje filtr, seznam dostupných položek a tlačítka pro zrušení a potvrzení výběru. Celkově tato komponenta slouží k výběru více položek z poskytnutých dat a umožňuje filtrování a úpravu výběru.

Soubor SearchForm.tsx

Uskutečňuje implementaci komponenty `SearchForm`.

V souboru jsou, mimo jiné, importovány také typy `AppDispatch` a `RootState` z příslušného souboru v adresářové struktuře. Dále je importována třída `Inventory` z `searchFormTypes.ts`, funkce `search` z `listViewThunks.ts` a funkce `log` z `logger.ts`.

Rozhraní `SearchFormProps` je definováno s vlastností `isFilterOpen` typu `boolean`. Komponenta `SearchForm` je definována jako funkční komponenta se vstupním parametrem `props` typu `SearchFormProps`.

V těle komponenty jsou definovány různé proměnné, které získávají data ze stavu pomocí funkce `useSelector`. Tyto proměnné reprezentují různé části formuláře a jejich hodnoty jsou aktualizovány v závislosti na stavu aplikace.

Dále jsou definovány různé stavy pomocí funkce `useState`, které reprezentují vybrané položky v různých částech formuláře. Funkce `dispatch` slouží k dispečeru Redux akcí. Funkce `searchSubmit` je volána po stisknutí tlačítka "Search" a odesílá vyhledávací data pomocí funkce `search` a dispečeru. Funkce `clearForm` slouží k vyčištění formuláře a odstranění vybraných položek.

V JSX kódu komponenty jsou zobrazeny různé části formuláře, jako například komponenty `MultiSelect` a `SwitchWithLabel`. Tlačítka "Reset" a "Search" jsou spojeny s odpovídajícími funkcemi.

Celkově tato komponenta reprezentuje vyhledávací formulář s možností výběru a filtrování různých položek. Po vyplnění formuláře a stisknutí tlačítka "Search" jsou data odeslána a vyvolána akce pro vyhledání odpovídajících výsledků.

Soubor SwitchWithLabel.tsx

Soubor "SwitchWithLabel.tsx" obsahuje implementaci komponenty `SwitchWithLabel`.

V souboru je importována komponenta `Switch` z knihovny "native-base" a také komponenty `View` a `Text` pro manipulaci s rozložením a zobrazením. Rozhraní `SwitchWithLabelProps` je definováno s vlastnostmi `label` typu `string`, `value` typu `boolean` a `onValueChange` typu `(value: boolean) => void`.

Komponenta `SwitchWithLabel` je definována jako funkční komponenta se vstupním parametrem `props` typu `SwitchWithLabelProps`. V těle komponenty je vrácen JSX kód, který zobrazuje popisek a přepínač (switch) vedle sebe. Komponenta `View` je použita pro nastavení rozložení prvků v řádku s flexibilním uspořádáním. Komponenta `Text` zobrazuje hodnotu `label`, která je předána jako vstupní vlastnost. Komponenta `Switch` má hodnotu `value` a `onValueChange` předané jako vstupní vlastnosti.

Celkově tato komponenta reprezentuje přepínač (switch) s textovým popiskem, který umožňuje uživateli zapínat a vypínat určitou funkci nebo možnost.

Podadresář *logging*

Soubor logger.ts

Jedná se o konfiguraci a inicializaci loggeru pro zaznamenávání událostí.

Klíčový je import loggeru z modulu 'react-native-logs'. Tento modul poskytuje funkce pro logování zpráv v prostředí React Native. Následuje definice konfiguračního objektu, který se nazývá config. Tento objekt obsahuje různé nastavení pro logger. Mezi tato nastavení patří:

- *severity*: určuje úroveň závažnosti logovaných zpráv, ve specifikaci je nastavena na 'debug'

- *transportOptions*: objekt obsahující možnosti pro přenos logovaných zpráv, zde jsou nastaveny hodnoty jako barvy zpráv (vypnuto), zobrazení úrovně logování, zobrazení data, povolení logování, značka (tag) a formát zprávy
- *levels*: definuje úrovně logování (debug, info, warn, error) a přiřazuje jim číselné hodnoty pro porovnání
- *async*: určuje, zda mají být logovací operace prováděny asynchronně, zde je nastaveno na false, což znamená synchronní provádění

Následuje inicializace loggeru pomocí funkce logger.createLogger, která přijímá konfigurační objekt a vytváří instanci loggeru. Tato instance loggeru je přiřazena do proměnné log.

Podadresář *pages*

Soubor HomePage.tsx

Představuje definici komponenty pro domovskou stránku (Home Page).

Zahájen je importem několika komponent z modulu "native-base". Importované komponenty jsou Center, Image a Text. Tyto komponenty slouží k vytváření struktury a vizuálního obsahu stránky. Následuje definice komponenty HomePage, která je funkčním komponentou napsanou v JSX syntaxi. Tato komponenta slouží k zobrazení obsahu domovské stránky. Obsahem komponenty jsou:

- Center: Tato komponenta slouží k vycentrování svého obsahu. Všechny další komponenty uvnitř této komponenty budou vycentrovány na stránce.
- Text: Tato komponenta slouží k zobrazení textového obsahu. Zde je zobrazen text "Home Page".
- Image: Tato komponenta slouží k zobrazení obrázkového obsahu. Komponenta má několik atributů, které nastavují vlastnosti obrázku:
- source: určuje zdroj obrázku pomocí URI (Uniform Resource Identifier). V tomto případě je zdrojem obrázek na adrese "http:/147.228.173.159/static/home/Rudolf-Aachen-crop.png".
- w: nastavuje šířku obrázku na "100 %".
- h: nastavuje výšku obrázku na "50 %".
- alt: určuje alternativní text pro případ, že obrázek nelze načíst. Zde je alternativní text "Rudolf-Aachen".

Komponenta HomePage slouží k zobrazení domovské stránky s textovým obsahem "Home Page" a obrázkem Rudolf-Aachen.

Soubor ItemViewPage.tsx

Soubor obsahuje definici komponenty pro zobrazení detailu položky (Item View Page).

Soubor začíná importem několika modulů a komponent z různých knihoven a vlastních souborů. Importuje se například React, useEffect a useState z knihovny "react", useDispatch a useSelector z knihovny "react-redux", TabView z knihovny "react-native-tab-view" a další. Tyto importy slouží k použití funkcionalit těchto knihoven a komponent v rámci souboru.

Následuje definice komponenty ItemViewPage, která je funkční komponentou napsanou v JSX syntaxi. Tato komponenta slouží k zobrazení detailu položky a obsahuje několik hooků, stavových proměnných a efektů. V komponentě se nejprve definuje proměnná layout, která využívá hook useWindowDimensions pro získání rozměrů okna.

Následují definice dalších proměnných pomocí hooků useState, které slouží k nastavení stavu a indexu pro záložky (routes) a současné zobrazené záložce (index). Dále se využívá hook useDispatch pro získání

dispatch funkce ze store, a useSelector pro získání stavových proměnných z redux store pomocí funkce selectora.

Následuje blok useEffect, který se spouští při inicializaci komponenty. V tomto bloku se pomocí dispatch funkce volá akce getItem pro načtení hlavní položky na základě id, které je předáno jako parametr route. Zároveň se loguje informace o volání této akce. Další blok useEffect se spouští při změně stavové proměnné concordances. V tomto bloku se získávají konkordance a na jejich základě se vytváří seznam záložek (routes) pro TabView komponentu. Zde se také volá akce setConcordances pro uložení konkordancí do store.

V neposlední řadě je nutné uvést blok useEffect, který se spouští při změně stavové proměnné item. V tomto bloku se získává hlavní položka a na základě ní se volá akce getItemNotes pro načtení poznámek k položce. Zde se také kontroluje, zda je index aktuální záložky 0 (tj. záložka hlavní položky), a pokud ano, volá se akce setConcordances pro uložení konkordancí do store.

Následuje definice funkce handleIndexChanged, která se volá při změně zobrazené záložky. Funkce aktualizuje stavovou proměnnou index a volá akci getItem pro načtení položky na základě id konkordance zvolené záložky. Další funkce onBackPressed se volá při stisknutí tlačítka zpět. V této funkci se pomocí navigace naviguje zpět.

Na závěr je uvedena návratová hodnota komponenty. Zde je podmínka, která se kontroluje, zda nejsou konkordance prázdné. Pokud jsou prázdné, zobrazí se LoadingBox s textem "Loading item {route.params.itemId}...". Pokud nejsou prázdné, zobrazí se TabView komponenta, která obsahuje záložky, zobrazení položky (ItemView) a další nastavení.

Komponenta ItemViewPage slouží k zobrazení detailu položky s možností prohlížení různých konkordancí a poznámek k položce.

Soubor LoginPage.tsx

Reprezentuje definici komponenty pro přihlašovací stránku (Login Page). Jako v předcházejících případech dochází k importu několika modulů a komponent z různých knihoven a vlastních souborů.

Následuje definice komponenty LoginPage, která je funkční komponentou napsanou v JSX syntaxi. Tato komponenta slouží k zobrazení přihlašovací stránky a obsahuje několik hooků, stavových proměnných a funkcí. V komponentě se nejprve definují stavové proměnné username a password pomocí hooku useState.

Dále se využívá hook useSelector pro získání stavové proměnné loggedIn ze store. Následuje definice dispatch funkce pomocí hooku useDispatch. Následuje definice funkce loginUser, která se volá při stisknutí tlačítka "Sign in". Tato funkce volá akci login s parametry username a password, které jsou získány ze stavových proměnných.

Navazuje návratová hodnota komponenty. Zde se využívá komponenta KeyboardAvoidingView, která slouží k přizpůsobení klávesnice na mobilním zařízení. Uvnitř této komponenty je Centrum, které obsahuje Box, ve kterém se nachází nadpis, texty, formulářové prvky (Input), tlačítko (Button) a odkazy (Link).

LoginPage slouží k zobrazení přihlašovací stránky s formulářem pro zadání uživatelského jména a hesla, tlačítkem pro přihlášení a odkazy na registraci.

Soubor Logout.tsx

Jedná se o definici komponenty Logout. Importuje se useDispatch z knihovny "react-redux", useEffect z knihovny "react", AppDispatch z vlastního souboru store, a Heading, HStack, Spinner z knihovny "native-base". Tyto importy slouží k použití funkcionalit těchto knihoven a komponent v rámci souboru.

Následuje definice komponenty Logout, která je funkční komponentou napsanou v JSX syntaxi. Tato komponenta slouží k zobrazení načítání při odhlášení uživatele. V komponentě se nejprve definuje proměnná dispatch pomocí hooku useDispatch.

Následuje definice efektu useEffect, který se spustí po vykreslení komponenty. Tento efekt volá akci logout pomocí dispatch. Akce logout slouží k odhlášení uživatele. Dále je uvedena návratová hodnota komponenty. Zde je HStack, který obsahuje komponenty Spinner a Heading. Spinner zobrazuje načítací ikonu a Heading zobrazuje text "Loading". Tato část komponenty slouží k vizuálnímu zobrazení načítání při odhlášení uživatele.

Logout zobrazuje načítání při odhlášení uživatele a volání akce logout prostřednictvím useDispatch.

Soubor NotesViewPage.tsx

Zahrnuje definici komponenty NotesViewPage.

Soubor začíná importem modulů a komponent z různých knihoven a vlastních souborů. Následuje definice komponenty NotesViewPage, která je funkční komponentou napsanou v JSX syntaxi. Tato komponenta slouží k zobrazení stránky s poznámkami. V komponentě se nejprve definuje proměnná notes, notesLoading, requestPending, triggerRefresh pomocí hooku useSelector, který získává stav z redux store.

Následuje definice proměnné dispatch pomocí hooku useDispatch a dále jsou definovány další proměnné pomocí hooku useState. Jedná se o isSortOpen, newComment, myCommentsCheck, generalCommentsCheck a replyingTo. Tyto proměnné slouží k uchovávání stavu komponenty. Definovány jsou i proměnné cancelRef a confirmDialog pomocí hooku useRef a useState. Tyto proměnné slouží k ovládání dialogového okna pro potvrzení.

Poté je definována proměnná sortSettings pomocí hooku useState, která obsahuje stav týkající se nastavení třídění. Následují funkce handleSortChanged a getSortIcon. Funkce handleSortChanged slouží k aktualizaci stavu sortSettings při změně třídění. Funkce getSortIcon vrací ikonu pro zobrazení aktuálního stavu třídění.

Efekt useEffect, který se spustí po vykreslení komponenty a při změně hodnoty triggerRefresh. Tento efekt volá funkci getNotes a aktualizuje stavy newComment a replyingTo. Další efekt useEffect reaguje na změnu hodnoty generalCommentsCheck a myCommentsCheck. Pokud notesLoading není true, efekt volá funkci getNotes.

Následuje definice funkce getNotes, která volá akci getAllNotes prostřednictvím dispatch. Tato akce slouží k získání poznámek s různými filtracemi a tříděním. Definovány jsou i funkce handleReply, handleEdit, handleDelete a handleCreateComment. Tyto funkce slouží k manipulaci s poznámkami, jako je odpovídání na poznámky, úprava poznámek, mazání poznámek a vytváření nových poznámek.

Podstatná je i návratová hodnota komponenty. Zde je vnořeno několik komponent, jako Box, VStack, HStack, Text, Switch, Button, Flex, TextArea, Icon, IconButton, Popover, Pressable, ScrollView, NotesListView a ConfirmDialog. Tyto komponenty slouží k vizuálnímu zobrazení stránky s poznámkami a interakci s uživatelem.

Celá komponenta NotesViewPage slouží k zobrazení stránky s poznámkami a manipulaci s poznámkami prostřednictvím akcí volaných pomocí useDispatch.

Soubor SearchPage.tsx

Definuje komponenty pro SearchPage. Definice komponenty SearchPage, která je funkční komponentou napsanou v JSX syntaxi. Tato komponenta slouží k zobrazení stránky s vyhledáváním. V komponentě se nejprve definuje proměnná isFilterOpen pomocí hooku useState, která uchovává stav otevření filtru.

Následuje definice proměnné dispatch pomocí hooku useDispatch. Dále je definována efekt useEffect, který se spustí po vykreslení komponenty a při změně hodnoty dispatch. Tento efekt volá různé akce pro načítání dat pomocí dispatch.

Následuje návratová hodnota komponenty. Zde je vnořeno několik komponent, jako Center, Button, ScrollView, VStack, SearchForm a ListView. Tyto komponenty slouží k vizuálnímu zobrazení stránky s vyhledáváním a interakci s uživatelem.

Celá komponenta SearchPage slouží k zobrazení stránky s vyhledáváním a načítání dat prostřednictvím akcí volaných pomocí useDispatch.

Podadresář *stores*

Soubor store.ts

Obsažena je definice Redux store pro správu stavu aplikace. Vše začíná importováním funkce configureStore z knihovny "@reduxjs/toolkit" a importy různých reducerů z vlastních souborů.

Následuje definice proměnné store, ve které je volána funkce configureStore. Tato funkce konfiguruje Redux store s několika vlastnostmi. Jednou z vlastností je reducer, který specifikuje jednotlivé reducery pro správu stavu. Reducer přijímá objekt, kde klíče jsou názvy stavových sliců a hodnoty jsou samotné reducery. V tomto souboru jsou použity reducery userReducer, itemReducer, noteViewReducer, searchFormReducer a listViewReducer.

Následují exporty z tohoto souboru. Exportuje se výchozí hodnota store, což je vytvořený Redux store. Dále jsou exportovány typy RootState, AppStore a AppDispatch. Typ RootState reprezentuje návratový typ funkce store.getState, která vrací aktuální stav aplikace. Typ AppStore reprezentuje typ výchozí hodnoty store, tedy Redux store. A typ AppDispatch reprezentuje typ dispatch funkce store, která slouží k odesílání akcí do Redux store.

Soubor store.ts definuje Redux store a exportuje jej spolu s příslušnými typy pro použití v aplikaci.

Podadresář *actions*

Soubor itemThunks.ts

Definovány jsou asynchronních *thunks* funkce pro získání položek (item) a poznámek (notes) v Redux store. Soubor začíná obligádním importem funkce createAsyncThunk z knihovny "@reduxjs/toolkit" a importem funkcí pro získání položek a poznámek ze služebních API. Importuje také typy Certainty a Concordance z modulu "../../types/item".

Následuje definice thunks funkce getItem, která slouží k asynchronnímu získání informací o položce na základě zadaného identifikátoru (itemId). Tato thunk funkce je vytvořena pomocí createAsyncThunk. V této thunk funkci je implementována logika pro získání dat z API a jejich transformace na odpovídající formát pro Redux store. Funkce getItemRequest je volána s parametrem itemId a vrací odpověď z API. Podle odpovědi jsou získávány různé atributy položky a vytvářen je objekt s odpovídajícími daty. V

závislosti na dostupných datových položkách jsou vytvářeny různé varianty objektu. Pokud dojde k chybě nebo výjimce, je vrácena odpovídající chybová zpráva.

Následuje definice thunks funkce getItemNotes, která slouží k asynchronnímu získání poznámek položky na základě zadaného identifikátoru (itemId). Tato thunk funkce je vytvořena pomocí createAsyncThunk. V této thunk funkci je implementována logika pro získání dat z API a jejich transformace na odpovídající formát pro Redux store. Funkce getItemNotesRequest je volána s parametrem itemId a vrací odpověď z API. Pokud je odpověď úspěšná a obsahuje nějaké poznámky, jsou tyto poznámky zpracovány a vytvořen je pole objektů s odpovídajícími daty. Pokud nejsou žádné poznámky k dispozici, je vytvořeno prázdné pole. V případě chyby je vrácena odpovídající chybová zpráva.

Na konci souboru je exportována thunk funkce getItem a getItemNotes pro použití v jiných částech aplikace.

Soubor listViewThunks.ts

Vydefinuje asynchronní *thunk* funkce pro vyhledávání v seznamu (list view) v Redux store. Obsahuje definici thunk funkce search, která slouží k asynchronnímu vyhledávání v seznamu na základě zadaných vyhledávacích parametrů (searchParams). Tato thunk funkce je vytvořena pomocí createAsyncThunk. V této thunk funkci je implementována logika pro volání API a získání odpovědi. Funkce searchRequest je volána s parametrem searchParams a vrací odpověď z API. Pokud je odpověď úspěšná (status 200), je vracená data typu SearchResponse a předána do Redux store. V případě chyby je vrácena odpovídající chybová zpráva.

Na konci souboru je exportována thunk funkce search pro použití v jiných částech aplikace.

Soubor notesThunks.ts

Skládá se z definic asynchronních thunk funkcí pro manipulaci s poznámkami (notes) v Redux store.

Definuje například thunk funkce deleteNote, která slouží k asynchronnímu mazání poznámky na základě zadaného identifikátoru poznámky (noteId). Tato thunk funkce je vytvořena pomocí createAsyncThunk. V této thunk funkci je implementována logika pro volání API a získání odpovědi. Funkce deleteNoteRequest je volána s parametrem noteId a vrací odpověď z API. Pokud je odpověď úspěšná (status 200), je vracen objekt s informací o smazané poznámce (deletedNoteId). V případě chyby je vrácena odpovídající chybová zpráva.

Následuje definice thunk funkce createNote, která slouží k asynchronnímu vytváření nové poznámky na základě zadaných informací (newNote). Tato thunk funkce je vytvořena pomocí createAsyncThunk. V této thunk funkci je implementována logika pro volání API a získání odpovědi. Funkce createNoteRequest je volána s parametrem newNote a vrací odpověď z API. Pokud je odpověď úspěšná (status 201), je vracen objekt s informací o vytvořené poznámce (createdNote). V případě chyby je vrácena odpovídající chybová zpráva.

Poté je uvedena definice thunk funkce updateNote, která slouží k asynchronní aktualizaci poznámky na základě zadaných informací (note). Tato thunk funkce je vytvořena pomocí createAsyncThunk. V této thunk funkci je implementována logika pro volání API a získání odpovědi. Funkce updateNoteRequest je volána s parametrem note a vrací odpověď z API. Pokud je odpověď úspěšná (status 201), je vracen objekt s informací o aktualizované poznámce (updateNote). V případě chyby je vrácena odpovídající chybová zpráva.

Důležitá je i definice thunk funkce getAllNotes, která slouží k asynchronnímu získání všech poznámek. Tato thunk funkce je vytvořena pomocí createAsyncThunk. V této thunk funkci je implementována

logika pro volání API a získání odpovědi. Funkce getAllNotesRequest je volána s parametry myComments, generalComments a sortOptions a vrací odpověď z API. Pokud je odpověď úspěšná (status 200), jsou získané poznámky zpracovány a připraveny pro uložení do Redux store. Pokud jsou k dispozici nějaké poznámky (response.data.length > 0), jsou tyto poznámky transformovány do požadovaného formátu a vráceny jako objekt s polem poznámek (notes). V opačném případě, pokud nejsou k dispozici žádné poznámky, je vrácen objekt s prázdným polem poznámek. V případě chyby je vrácena odpovídající chybová zpráva.

Na konci souboru jsou exportovány všechny definované thunk funkce pro použití v jiných částech aplikace.

Soubor searchFormThunks.ts

Definuje asynchronních thunk funkcí pro získávání dat z formuláře vyhledávání (search form) v Redux store. Zde je podrobný popis tohoto souboru:

Následuje definice thunk funkce fetchInventories, která slouží k asynchronnímu získání inventářů (inventories). Tato thunk funkce je vytvořena pomocí createAsyncThunk. V této thunk funkci je implementována logika pro volání API a získání odpovědi. Funkce fetchInventoriesRequest je volána bez parametrů a vrací odpověď z API. Pokud je odpověď úspěšná (status 200), jsou získané inventáře vráceny. V opačném případě, pokud dojde k chybě, je vrácena odpovídající chybová zpráva.

Podobným způsobem jsou definovány i další thunk funkce, jako fetchCountries, fetchInstitutions, fetchArtists, fetchTechniques, fetchNationalities, fetchCities, fetchSubjects a fetchPlans. Každá z těchto thunk funkcí obsahuje logiku pro volání příslušné API metody a zpracování odpovědi. Pokud je odpověď úspěšná, jsou získaná data vrácena. V případě chyby je vrácena odpovídající chybová zpráva.

Na konci souboru jsou exportovány všechny definované thunk funkce pro použití v jiných částech aplikace. Tyto thunk funkce lze použít k asynchronnímu načítání dat z formuláře vyhledávání a jejich uložení do Redux store.

Soubor userThunks.ts

Zahrnuje definice asynchronních thunk funkcí souvisejících s uživatelem v Redux store.

Zásadní je definice thunk funkce login, která slouží k asynchronnímu přihlášení uživatele. Tato thunk funkce je vytvořena pomocí createAsyncThunk. V této thunk funkci je implementována logika pro volání API a získání odpovědi. Metoda loginRequest je volána s uživatelským jménem a heslem předanými jako payload. Po získání odpovědi z API je zkontrolován status odpovědi. Pokud je odpověď úspěšná (status 200), jsou vráceny informace o přihlášeném uživateli, jako je uživatelské jméno a role. V případě chyby je vrácena odpovídající chybová zpráva.

Další definovanou thunk funkcí je checkAuth, která slouží k asynchronnímu ověření stavu přihlášení uživatele. Tato thunk funkce je vytvořena pomocí createAsyncThunk. V této thunk funkci je implementována logika pro volání API a získání odpovědi. Metoda isAuthRequest je volána bez parametrů. Po získání odpovědi z API je zkontrolován status odpovědi. Pokud je odpověď úspěšná (status 200), je vrácen informace o stavu přihlášení uživatele. V případě chyby je vrácena odpovídající chybová zpráva.

Na konci souboru jsou exportovány všechny definované thunk funkce pro použití v jiných částech aplikace. Tyto thunk funkce lze použít k asynchronnímu provádění akcí souvisejících s přihlášením a stavem uživatele v Redux store.

Podadresář *reducers*

Soubor itemSlice.ts

Pojímá definici slice pro správu stavu položek (items) v Redux store.

Struktura zdrojového kódu je zahájena konstantou CertaintyWithColors, která je typu Record a slouží k definici různých jistot (certainty) s přiřazenými barvami. Tyto hodnoty budou použity při zobrazení jistoty položek.

Poté je definován počáteční stav (initialState) pro stav položky (item). Tento stav obsahuje prázdný objekt pro položku, pole pro souhlasné položky (concordances), poznámky (notes), poslední chybovou zprávu (lastError) a příznaky (itemLoading a notesLoading) indikující, zda se položky načítají.

Následuje definice slice pomocí funkce createSlice. Slice je pojmenován "item" a má počáteční stav initialState. Slice obsahuje také definici reduktorů (reducers) a extra reduktorů (extraReducers).

V rámci reduktorů je definován reduktor setConcordances, který slouží k nastavení pole souhlasných položek. Tento reduktor aktualizuje stav předaný jako první argument (state) a používá payload akce jako nové pole souhlasných položek. Extra reduktory definují reakce na akce spojené s načítáním položek a poznámek. Například při splnění akce getItem je stav aktualizován na základě payload akce, který obsahuje informace o položce. Pokud je fullView položky nastaveno na true, jsou aktualizovány další informace o položce (institution, authorName, imageUrl, title, repository, provenance, description). V případě, že fullView je false, jsou tyto informace odstraněny ze stavu. Podobně je nastaven stav pro načítání položek a zpracování chyb. Nakonec je exportován výchozí reducer slice pro použití v Redux store.

Tento soubor tedy definuje *slice* pro správu stavu položek v aplikaci a obsahuje reduktory, které reagují na různé akce a aktualizují stav položek v Redux store.

Soubor listViewSlice.ts

Definuje slice pro správu stavu seznamu položek (list view) v Redux store. Následuje definice rozhraní ListViewState, které obsahuje vlastnosti inventories (pole inventářů), data (pole typu ItemPreviewType), loading (příznak indikující načítání) a lastError (poslední chybová zpráva).

Poté je definován počáteční stav (initialState) pro stav seznamu položek. Tento stav obsahuje prázdná pole inventories a data, a příznak loading je nastaven na false.

Následuje definice slice pomocí funkce createSlice. Slice je pojmenován "listView" a má počáteční stav initialState. Slice obsahuje také definici reduktorů (reducers) a extra reduktorů (extraReducers). V rámci reduktorů je definován reduktor resetListView, který slouží k resetování stavu seznamu položek. Tento reduktor jednoduše nastaví stav na počáteční hodnoty initialState.

Extra reduktory definují reakce na akce spojené se vyhledáváním v seznamu položek. Například při splnění akce search je stav aktualizován na základě payload akce, který obsahuje informace o inventářích (inventories) a datech (data). Příznak loading je nastaven na false. Podobně je nastaven stav pro načítání dat a zpracování chyb.

Nakonec jsou exportovány akce slice pomocí listViewSlice.actions a výchozí reducer slice pro použití v Redux store. Tento soubor tedy definuje slice pro správu stavu seznamu položek v aplikaci a obsahuje reduktory, které reagují na různé akce a aktualizují stav seznamu položek v Redux store.

Soubor notesSlice.ts

Podstatná je definice slice pro správu stavu poznámek (notes) v Redux store.

Klíčová je definice počátečního stavu (initialState) pro stav poznámek. Tento stav obsahuje prázdné pole notes (poznámky), příznaky notesLoading (indikuje načítání poznámek), requestPending (indikuje probíhající požadavek na aktualizaci nebo odstranění poznámky), triggerRefresh (číslo, které slouží k obnovení poznámek) a lastError (poslední chybová zpráva).

Následuje definice slice pomocí funkce createSlice. Slice je pojmenován "note" a má počáteční stav initialState. Slice neobsahuje žádné reduktory (reducers).

Extra reduktory definují reakce na akce spojené s načítáním, aktualizací, vytvářením a odstraňováním poznámek. Například při splnění akce getAllNotes je stav aktualizován na základě payload akce, který obsahuje pole poznámek (notes). Příznak notesLoading je nastaven na false. Podobně jsou nastaveny stavy pro aktualizaci, vytváření a odstraňování poznámek v závislosti na splnění nebo zamítnutí příslušných akcí. Nakonec je exportován výchozí reducer slice pro použití v Redux store.

Tento soubor tedy definuje slice pro správu stavu poznámek v aplikaci a obsahuje extra reduktory, které reagují na různé akce a aktualizují stav poznámek v Redux store.

Soubor searchFormSlice.ts

Nachází se v něm definice slice pro správu stavu vyhledávacího formuláře (search form) v Redux store.

Soubor začíná importem funkce createSlice z knihovny "@reduxjs/toolkit". Také jsou importovány thunk funkce fetchArtists, fetchCities, fetchCountries, fetchInstitutions, fetchInventories, fetchNationalities, fetchPlans, fetchSubjects a fetchTechniques z jiného souboru searchFormThunks.ts. Kromě toho jsou importovány typy Artist, Inventory a Room související se stavem vyhledávacího formuláře.

Následuje definice počátečního stavu (initialState) pro stav vyhledávacího formuláře. Tento stav obsahuje pole inventories (inventáře), rooms (místnosti), artists (umělci), nationalities (národnosti), techniques (techniky), institutions (instituce), countries (země), cities (města) a subjects (předměty). Také obsahuje volitelné pole lastError pro uchování poslední chybové zprávy.

Uvedena je i definice slice pomocí funkce createSlice. Slice je pojmenován "searchForm" a má počáteční stav initialState. Slice obsahuje reduktory resetSearchForm, setInventories, setRooms, setArtists a setNationalities, které slouží k aktualizaci příslušných částí stavu vyhledávacího formuláře.

Extra reduktory definují reakce na akce spojené s načítáním dat pro jednotlivé části vyhledávacího formuláře. Například při splnění akce fetchInventories je stav inventories aktualizován na základě payload akce, který obsahuje pole inventářů. Příznak lastError je nastaven na prázdný řetězec. Podobně jsou definovány reakce pro načítání institucí, národností, zemí, umělců, místností, předmětů, měst a technik. Na závěr jsou exportovány reduktory resetSearchForm, setInventories, setRooms, setArtists a setNationalities pro použití v Redux store.

Tento soubor tedy definuje slice pro správu stavu vyhledávacího formuláře v aplikaci a obsahuje extra reduktory, které reagují na různé akce a aktualizují příslušné části stavu vyhledávacího formuláře v Redux store.

Soubor userSlice.ts

Zahrnuje v sobě definici slice pro správu stavu uživatele (user) v rámci Redux store.

Po požadovaných importech následuje definice rozhraní UserState, které popisuje strukturu stavu uživatele. Stav zahrnuje pole username (uživatelské jméno), loggedIn (přihlášený/odhlášený), role (role uživatele) a volitelné pole lastError pro uchování poslední chybové zprávy.

Po definici stavu následuje inicializace počátečního stavu (initialState) pro stav uživatele. Počáteční stav obsahuje prázdné hodnoty pro username, loggedIn a role.

Následuje definice slice pomocí funkce createSlice. Slice je pojmenován "user" a má počáteční stav initialState. Slice obsahuje reduktor logout, který slouží k odhlášení uživatele a nastavení stavu na počáteční hodnoty. Extra reduktory definují reakce na akce spojené s přihlášením a ověřením autentizace. Například při splnění akce login je stav aktualizován na základě payload akce, který obsahuje uživatelské jméno (username), příznak přihlášení (loggedIn), uživatelskou roli (role) a lastError je nastaven na prázdný řetězec. Podobně jsou definovány reakce pro ověření autentizace (checkAuth) a zpracování odpovědí při úspěšném nebo neúspěšném provedení akce. Nakonec jsou exportovány reduktory logout pro použití v Redux store.

Tento soubor tedy definuje slice pro správu stavu uživatele v aplikaci a obsahuje extra reduktory, které reagují na akce související s přihlášením a ověřením autentizace a aktualizují příslušné části stavu uživatele v Redux store.

Podadresář *theme*

Soubor nativeBaseTheme.ts

Lze jej charakterizovat jako definici tématu pro použití v knihovně NativeBase, která poskytuje sadu komponent a stylů pro vývoj mobilních aplikací. Uvedena je exportovaná konstanta nativeBaseTheme, která je vytvořena voláním funkce extendTheme. Tato funkce přijímá objekt s konfiguračními možnostmi pro rozšíření tématu.

V rámci konfiguračního objektu je definován klíč "colors", který obsahuje definici primárních barev. Primární barvy jsou definovány pomocí číselných hodnot v rozsahu od 50 do 900. Každá číselná hodnota odpovídá určitému odstínu primární barvy. Například pro klíč "primary" jsou definovány hodnoty pro různé odstíny od světlejších (50) po tmavší (900).

Vytvořené téma (nativeBaseTheme) je následně exportováno a může být použito při vývoji mobilních aplikací s využitím knihovny NativeBase. Toto téma rozšiřuje základní téma knihovny a poskytuje definici barev pro primární barvy, které mohou být použity v různých komponentách a stylování aplikace.

Podadresář *types*

Soubor general.ts

Složen je z vývojového kódu, jež exportuje výčtový typ (enum) nazvaný SortOptions. Zde je podrobný popis tohoto souboru:

Soubor obsahuje export definice výčtového typu SortOptions, který slouží k reprezentaci možností třídění. Výčtový typ je určený pro představování omezené množiny hodnot, které jsou vzájemně výlučné.

V rámci výčtového typu jsou definovány tři možnosti třídění: None, Asc a Desc. Každá z těchto možností má přiřazenou řetězcovou hodnotu. Hodnoty jsou přiřazeny pomocí syntaxe "hodnota = 'řetězec'". Konkrétně zde jsou definovány následující hodnoty:

- None: Hodnota "none" představuje možnost bez třídění.
- Asc: Hodnota "asc" představuje možnost třídění vzestupně.
- Desc: Hodnota "desc" představuje možnost třídění sestupně.

Výčtový typ SortOptions může být importován a použit v dalších částech kódu, kde je potřeba reprezentovat možnosti třídění. Tento enum usnadňuje správu a kontrolu povolených hodnot a zvyšuje čitelnost kódu při práci s tříděním.

Soubor item.ts

Jedná se o kód definující několik typů a výčtového typu.

Uvádí definici typu Item, reprezentující položku. Typ Item obsahuje různé vlastnosti, jako je id (identifikátor), seznam souvisejících textů (concordances), zobrazení jména autora, název díla, položku inventáře, vyhledávací předměty, indikaci, zda je zobrazený celý obsah, jméno autora, URL obrázku, titul, instituci (která obsahuje název, inventární číslo, zemi a město), úložiště, provenience a popis položky. V některých případech jsou některé vlastnosti nepovinné, což je indikováno pomocí otazníkového zápisu.

Dále je definován typ ItemViewState, který reprezentuje stav zobrazení položky. Tento typ obsahuje vlastnosti, jako je položka (typ Item), načítání položky, seznam souvisejících textů (concordances), poznámky (typ Note), načítání poznámek a případnou poslední chybu.

Poté následuje definice typu Concordance, který představuje související text. Tento typ obsahuje vlastnosti id (identifikátor) a cert (jistota), která je výčtovým typem Certainty.

Nakonec je definován výčtový typ Certainty, který slouží k reprezentaci jistoty. Tento výčtový typ obsahuje několik možných hodnot, včetně SameAs, High, Medium, Low a Unknown. Každá z těchto hodnot představuje určitou úroveň jistoty. Tyto typy a výčtový typ mohou být importovány a použity v dalších částech kódu pro reprezentaci a manipulaci s informacemi o položkách, stavu zobrazení položky a souvisejících textech.

Soubor listViewTypes.ts

Představuje kód definující různé typy, které se týkají zobrazení seznamu položek.

Soubor listViewTypes.ts začíná importem typu Inventory z jiného souboru s názvem searchFormTypes.ts. Následují definice několika typů, které se používají při zobrazení seznamu položek.

- Typ SearchResponse představuje odpověď na vyhledávání. Obsahuje vlastnost pagination (stránkování) a data (pole s typem ItemPreviewType[]), které obsahuje náhledy položek.
- Typ Pagination reprezentuje informace o stránkování. Obsahuje vlastnosti cursor (kurzor), records (počet záznamů), items (počet položek) a inventories (pole s typem Inventory[]), které obsahuje informace o inventářích.
- Typ ItemPreviewType představuje náhled položky. Obsahuje vlastnosti xml_id (identifikátor v XML), iconclass_external_id (externí identifikátor ikonické třídy), text (text), title (název), object (typ Item), inventory (typ Inventory) a volitelně src (typ ItemSource), který představuje zdroj položky.
- Typ Item reprezentuje položku. Může mít dvě možné formy. První forma je pole s jedním objektem, který obsahuje vlastnosti caption (popisek), type (typ) a volitelně name (pole s typem PersonName[]), které obsahuje informace o jménech osob. Druhá forma je pole objektů, které obsahují vlastnosti images (pole s typem ItemImage[]) a volitelně origDate (původní datum).
- Typ PersonName představuje jméno osoby. Obsahuje vlastnosti value (hodnota, tedy jméno), role (role osoby), getty_external_id (externí identifikátor z Gettyského institutu) a getty_data (typ GettyData), který obsahuje informace o datu zobrazení.
- Typ GettyData obsahuje vlastnost display_name (zobrazené jméno).

- Typ ItemImage reprezentuje obrázek položky. Obsahuje vlastnosti file (soubor) a text (popis).
- Typ ItemSource reprezentuje zdroj položky. Obsahuje vlastnost value (hodnota, tedy zdroj).

Tyto typy jsou používány pro reprezentaci a manipulaci s informacemi o náhledech položek v seznamu.

Soubor note.ts

Definuje rozhraní a typů, které se týkají poznámek. Zde je podrobný popis tohoto souboru:

Soubor note.ts začíná exportem rozhraní Note, které představuje poznámku. Rozhraní obsahuje vlastnosti uuid (unikátní identifikátor), username (uživatelské jméno), userId (identifikátor uživatele), note (text poznámky), avatarUrl (URL adresa avataru), items (pole identifikátorů položek), createdTime (datum vytvoření), updatedTime (datum aktualizace), a noteColor (barva poznámky) a reply_to (identifikátor rodičovské poznámky).

Následuje definice typu NotesViewState, který reprezentuje stav poznámek. Obsahuje vlastnosti notes (pole s typem Note[], tedy seznam poznámek), notesLoading (příznak indikující načítání poznámek), requestPending (příznak indikující probíhající požadavek), triggerRefresh (číslo, které slouží k vyvolání obnovení), a volitelně lastError (textový popis poslední chyby).

Tyto rozhraní a typy jsou používány pro reprezentaci a manipulaci s poznámkami a jejich stavem v aplikaci.

Soubor searchFormTypes.ts

Skládá se z definic typů, které souvisejí s formulářem pro vyhledávání.

Soubor searchFormTypes.ts začíná definicí typu Inventory, který reprezentuje inventář. Tento typ obsahuje vlastnosti count (volitelný počet), name (název inventáře), label (popisek inventáře) a order (řadové číslo).

Následuje definice typu Room, který představuje místnost. Typ Room obsahuje vlastnosti id (identifikátor místnosti), in_plan (příznak indikující, zda je místnost v plánu), label (popisek místnosti) a volitelně places (pole míst v místnosti typu RoomPlace[]).

Typ RoomPlace představuje konkrétní místo v místnosti. Obsahuje vlastnosti id (identifikátor místa) a label (popisek místa).

Poslední definovaný typ je Artist, který reprezentuje umělce. Typ Artist obsahuje vlastnosti display_name (zobrazované jméno umělce) a getty_id (identifikátor umělce získaný z Getty databáze).

Tyto definice typů slouží k reprezentaci a manipulaci s daty týkajícími se inventáře, místností a umělců v souvislosti s vyhledávacím formulářem v aplikaci.

Závěr

Dokument obsahuje programátorskou dokumentaci a také dokumentaci k architektuře navrhované aplikace. Repositář projektu je přehledně strukturován a zdrojový kód je v rámci tohoto dokumentu okomentován. Přehledná struktura zajistí, že při případném doplňování rozšíření, či údržbě, je možné aplikaci snadno modifikovat. Aplikace splňuje zadání (akceptační kritéria) zákazníka.