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: 16. 12. 2023 (v2.0)

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.

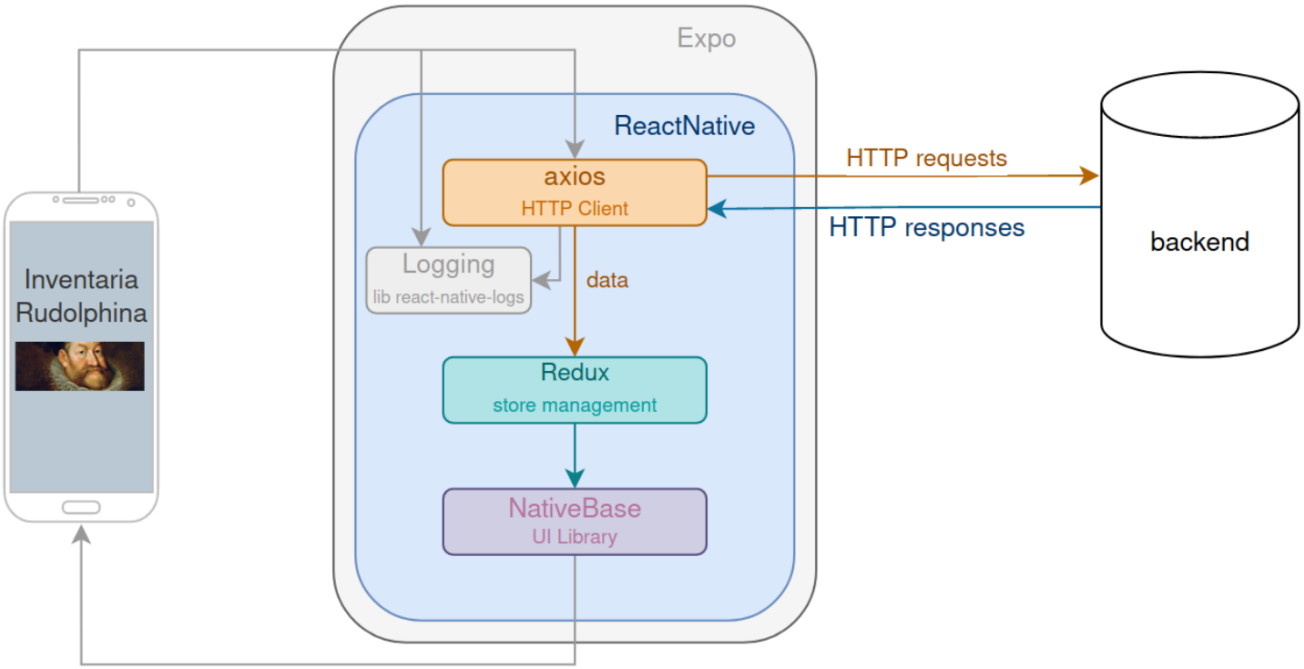
Uživatelské rozhraní bylo vyvinuto pomocí UI knihovny *NativeBase*, která poskytuje sadu předpřipravených komponent pro snadnou tvorbu rozhraní. Stavová správa aplikace byla realizována s využitím knihovny *Redux*, která efektivně řídí stavová data. Pro provádění *HTTP* požadavků byla implementována 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-native-community/viewpager	^5.0.11
@react-navigation/drawer	^6.6.2
@react-navigation/native	^6.1.6
@react-navigation/native-stack	^6.9.14
@reduxjs/toolkit	^1.9.3
axios	^1.5.1
expo	^49.0.13
expo-build-properties	~0.8.3
expo-splash-screen	^0.20.5
expo-status-bar	~1.6.0
native-base	^3.4.28
react	18.2.0
react-dom	^18.2.0
react-native	0.72.5
react-native-deck-swiper	^1.1.7
react-native-gesture-handler	~2.12.0
react-native-logs	^5.0.1
react-native-multiple-select	^0.5.12
react-native-reanimated	~3.3.0
react-native-safe-area-context	4.6.3
react-native-screens	~3.22.0
react-native-svg	13.9.0
react-native-tab-view	^3.5.2
react-native-vector-icons	^9.2.0
react-native-xml2js	^1.0.3
react-redux	^8.0.5
redux	^4.2.1
redux-persist-transform-encrypt	^5.0.0
typescript	^5.2.2
@babel/core	^7.20.0
@tsconfig/react-native	^2.0.3
@types/jest	^29.5.0
@types/react	~18.2.14
@types/react-test-renderer	^18.0.0
typescript	^5.1.3

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

Tento zdrojový soubor obsahuje konfiguraci pro mobilní aplikaci vytvořenou pomocí frameworku Expo. Celkově definuje různá nastavení pro různé platformy a specifikace pro uživatelské rozhraní. Níže je souvislý popis toho, co zdrojový soubor obsahuje:

Tento soubor obsahuje metadata a konfiguraci pro mobilní aplikaci s názvem "RudolfII". Aplikace má specifikováno několik vlastností, včetně verze, orientace obrazovky (portrait), ikony a rozložení uživatelského rozhraní ve světlém režimu. Splash screen aplikace je definován obrázkem "Rudolf-Aachen-large.png" s barvou pozadí "#E3A400".

Pro platformu iOS je nastavena podpora pro tablety, zatímco pro platformu Android je definován adaptivní ikonový formát s předním obrázkem "./assets/icon.png" a barvou pozadí "#E3A400". Specifikuje se také balíček pro Android s názvem "cz.zcu.kiv.tsp.inventaria_rudolphina".

Pro webovou platformu je určen favicon "./assets/favicon.png". V sekci "extra" je zahrnut projekt v rámci EAS (Expo Application Services) s ID "9cbe973a-6a5e-4b5e-85dd-02a80f7e38cc".

Za zmínku stojí také použití pluginu "expo-build-properties", který nastavuje konkrétní vlastnosti pro Android a iOS, jako je povolení nezabezpečené komunikace na Androidu a využívání nevýjimečného šifrování na iOS.

Celkově lze z tohoto souboru vyčíst informace o konfiguraci vlastností a chování mobilní aplikace na různých platformách a v různých situacích.

Soubor App.tsx

Soubor App.tsx představuje vstupní bod mobilní aplikace napsané v React Native. Celý soubor je navržen tak, aby propagoval potřebné konfigurace a poskytoval správné kontexty pro běh aplikace. Níže naleznete souvislý popis obsahu tohoto souboru:

Soubor App.tsx zahajuje svou funkcionalitu importem potřebných knihoven, včetně 'react-native-gesture-handler' pro podporu gest a komponenty NativeBaseProvider ze "native-base" pro poskytnutí základních komponent v nativním designu. Dále je použita knihovna Provider pro propojení s Redux storagem, a je importován reduxový store z cesty "./src/stores/store".

Samotná funkce App je exportována jako výchozí a vrací hierarchii komponent. Celý strom komponent je obklopen komponentou Provider, která propojuje Redux store s aplikací. Vnitřním prvkem je NativeBaseProvider, který zajišťuje nativní vzhled komponent a je konfigurován tématem nativeBaseTheme.

V těle Provider a NativeBaseProvider se nachází hlavní komponenta aplikace, a to Navigation. Tato komponenta zřejmě obsahuje navigační strukturu pro celou aplikaci a řídí přepínání mezi jednotlivými obrazovkami. Celkově lze soubor App.tsx chápat jako klíčový bod pro inicializaci aplikace, nastavení globálních kontextů a definici celkové struktury uživatelského rozhraní pomocí komponent poskytovaných knihovnou native-base.

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 eas.json

Soubor eas.json slouží ke konfiguraci některých aspektů vývoje a sestavování mobilní aplikace pomocí nástroje Expo Application Services (EAS). Níže je souvislý popis obsahu tohoto konfiguračního souboru:

V části "cli", konkrétně ve vlastnosti "version", je definována minimální verze nástroje Expo CLI, kterou projekt vyžaduje pro správné fungování. V tomto případě je vyžadována verze rovna nebo vyšší než 3.12.1.

Sekce "build" obsahuje konfigurace pro sestavení aplikace v různých režimech. V oddílu "development" jsou specifikovány parametry pro vývojový režim, kde je aktivní vývojový klient a distribuce je interního typu. Dále jsou definovány tři náhledové režimy - "preview", "preview2" a "preview3". Pro "preview" režim Androidu je nastaven typ sestavení na "apk". "preview2" specifikuje vlastní gradle příkaz pro sestavení release verze pro Android. "preview3" opět aktivuje vývojového klienta. Pro produkční sestavení je vytvořen prázdný objekt, protože pravděpodobně nejsou potřebné specifické konfigurace.

V části "submit" jsou specifikovány konfigurace pro odesílání aplikace. Opět je zde vytvořen prázdný objekt pro produkční odeslání, což naznačuje, že v této fázi nejsou žádné specifické konfigurace uvedeny.

Celkově vzato, soubor eas.json poskytuje konfigurace pro Expo Application Services, které se týkají verze Expo CLI a specifikují různé režimy a konfigurace pro vývoj, náhled a produkční sestavování a odesílání mobilní aplikace.

Soubor package.json

Soubor package.json obsahuje metadata a konfigurace pro Node.js projekt. Zde je souvislý popis obsahu tohoto souboru:

Projekt je pojmenován "rudolfii" a má verzi 1.0.0. Hlavní soubor pro vstupní bod aplikace je "node_modules/expo/AppEntry.js". Skripty pro spouštění různých částí aplikace jsou definovány v sekci "scripts". Například, příkaz "start" spouští vývojový server pomocí Expo CLI, "android" a "ios" spouští aplikaci na připojeném zařízení nebo emulátoru, a "web" spouští vývojový server pro webovou verzi.

Závislosti projektu jsou rozděleny do dvou částí - "dependencies" a "devDependencies". Mezi hlavní závislosti patří nástroje jako React Native, Expo, a několik knihoven pro navigaci, stavový management, HTTP requesty, logování, a další. Verze těchto závislostí jsou pevně definovány, což může pomoci zajistit konzistentní prostředí při instalaci závislostí na jiném zařízení nebo pro jiného vývojáře.

V oddílu "devDependencies" jsou uvedeny závislosti, které jsou používány pouze během vývoje, například TypeScript pro typovou kontrolu, Babel pro kompilaci kódu, Jest pro testování, a další.

Celkově vzato, soubor package.json poskytuje klíčové informace o projektu, včetně jeho názvu, verze, skriptů pro spouštění různých částí aplikace, a seznamu závislostí nezbytných pro běh a vývoj projektu postaveného na platformě React Native.

Soubor package-lock.json

Soubor package-lock.json obsahuje podrobný záznam o konkrétních verzích a závislostech používaných balíčků v projektu. Zde je souvislý popis několika vybraných částí souboru:

V souboru jsou uvedeny informace o projektu s názvem "rudolfii" ve verzi 1.0.0. Soubor obsahuje informace o verzování zámkového souboru (lockfileVersion je 2), zda jsou vyžadovány závislosti (requires je true) a seznam balíčků, které jsou součástí projektu.

V samotném seznamu balíčků jsou záznamy o různých balíčcích, včetně jejich verzí, odkazů na jejich distribuci a integritu. Například, balíček "@react-native-community/viewpager" ve verzi "^5.0.11" má specifikované závislosti na dalších balíčcích, které jsou také uvedeny v souboru. Tyto informace jsou důležité pro zajištění konzistence a správného stavu balíčků při instalaci projektu na jiném místě nebo pro jiného vývojáře.

Pro každý balíček jsou také uvedeny informace o kompatibilitě s verzí Node.js. To je užitečné pro zajištění, že balíčky budou správně fungovat s používanou verzí Node.js v daném prostředí.

Souhrnně lze soubor package-lock.json chápat jako "zamrzlý" stav projektu, který obsahuje konkrétní verze všech závislostí, což zajišťuje konzistentní prostředí pro spuštění projektu.

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.

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. Ikona 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.
- „Rudolph-Aachen-large.png“: Zdrojový soubor s portrétem císaře Rudolfa II. Je využit při spuštění aplikace a také na hlavní stránce (*My home*).

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 sadu funkcí, které slouží k provádění autentizačních operací v rámci mobilní aplikace. Využívá se zde knihovna axiosInstance, která je importována ze souboru api.

První funkce, loginRequest, je asynchronní a slouží k odeslání požadavku na server pro ověření přihlašovacích údajů. Funkce přijímá uživatelské jméno a heslo jako parametry a pomocí metody POST odesílá tyto údaje na konkrétní cestu "/login" na serveru.

Druhá funkce, isAuthRequest, také asynchronní, slouží k provedení požadavku typu GET na cestu "/isauth". Tato cesta pravděpodobně slouží k ověření, zda je uživatel stále autentizován. Server na tuto cestu pravděpodobně odpovídá informací o stavu autentizace.

Třetí funkce, logoutRequest, je opět asynchronní a slouží k odeslání požadavku na odhlášení. Volá metodu GET na cestu "/logout", což naznačuje, že server by měl provést odhlášení uživatele.

Celkově vzato, soubor authservice.ts poskytuje jednoduché a jasné rozhraní pro autentizační operace v rámci aplikace a využívá pro komunikaci se serverem knihovnu axios.

Soubor constants.ts

Skládá se z definice několika konstant, které zahrnují URL adresy, zejména pro přístup k různým zdrojům na serveru.

První konstanta, BASE_URL, určuje základní adresu serveru, která je nastavena na 'http://147.228.173.159'. Tato adresa pravděpodobně představuje kořenovou cestu ke zdrojům na serveru.

Druhá konstanta, BASE_API_URL, je odvozena z předchozí konstanty a přidává '/api', což naznačuje, že se jedná o základní adresu pro API operace na serveru.

Třetí a čtvrtá konstanta, AVATAR_URL a IMAGE_URL, obsahují cesty pro přístup k statickým souborům na serveru, konkrétně k avatárům a obecným obrázkům.

Pátá konstanta, HOME_PAGE_IMG_URL, poskytuje adresu obrázku pro domovskou stránku.

Šestá konstanta, ITEM_PREVIEW_IMG_PREFIX_URL, představuje prefix adresy pro náhledy obrázků.

Celkově vzato, soubor constants.ts centralizuje informace o URL adresách, což usnadňuje údržbu a změny v případě potřeby. Tyto konstanty pravděpodobně poslouží k jednoduchému a konzistentnímu přístupu k různým zdrojům a endpointům na serveru z různých částí aplikace.

Soubor homePageService.ts

Soubor homePageService.ts obsahuje funkci pro provádění asynchronního HTTP požadavku na server za účelem získání dat pro domovskou stránku aplikace. Funkce je pojmenována fetchHomeDataRequest a využívá externího modulu axiosInstance importovaného ze souboru api.ts.

Samotná funkce fetchHomeDataRequest je definována jako asynchronní a vrací výsledek asynchronního volání metody get objektu axiosInstance. Toto volání je konfigurováno s konkrétním URL endpointem "/homePage", což naznačuje, že účelová cesta na serveru pro získání dat domovské stránky je dostupná pod tímto konkrétním endpointem.

Soubor homePageService.ts slouží k oddělení logiky pro provádění HTTP požadavku na server pro získání dat pro domovskou stránku. To napomáhá kódové modularitě a údržbě, přičemž komunikace se serverem a zpracování dat pro domovskou stránku jsou odděleny od ostatních částí aplikace.

Soubor itemservice.ts

Soubor `itemservice.ts` obsahuje funkci pro provádění asynchronního HTTP požadavku na server za účelem získání informací o konkrétním položce. Tato funkce je pojmenována `getItemRequest` a využívá externího modulu `axiosInstance`, který je importován ze souboru `api.ts`.

Samotná funkce `getItemRequest` je definována jako asynchronní a přijímá jeden parametr `itemId` typu string, který představuje identifikátor konkrétní položky. Funkce poté vrací výsledek asynchronního volání metody `get` objektu `axiosInstance`. Toto volání je konfigurováno s konkrétním URL endpointem `/item/${itemId}`, což naznačuje, že účelová cesta na serveru pro získání informací o položce obsahuje dynamický parametr `itemId`.

Komentáře v kódu ukazují, že původně byla v souboru i další funkce `getItemConcordancesRequest` pro získání souvisejících informací o položce. Tato funkce byla zneaktivněna a není aktuálně používána ve zdrojovém kódu.

Soubor `itemservice.ts` slouží k oddělení logiky pro provádění HTTP požadavků na server, konkrétně pro získání informací o položkách, což přispívá k modularitě a čitelnosti kódu.

Soubor noteservice.ts

Soubor noteservice.ts obsahuje sadu funkcí pro provádění asynchronních HTTP požadavků na server v souvislosti s poznámkami (notes). Tyto funkce slouží k interakci s backendovým API, zejména pro získání, aktualizaci, vytvoření a odstranění poznámek. Importuje také potřebné typy a modul pro správu HTTP požadavků (axiosInstance z api.ts).

První funkce, getItemNotesRequest, je navržena pro získání poznámek týkajících se konkrétní položky (Item). Funkce umožňuje také specifikovat, zda mají být zahrnuty i související komentáře. Vytváří dynamickou URL na základě poskytnutých parametrů.

Následují tři další funkce (updateNoteRequest, createNoteRequest, deleteNoteRequest), které slouží k aktualizaci, vytvoření a odstranění poznámky. Každá z těchto funkcí provádí příslušný HTTP požadavek na specifický endpoint serveru.

Poslední funkce, getAllNotesRequest, je koncipována pro získání všech poznámek a umožňuje specifikovat různé filtry a řazení. Funkce opět vytváří dynamickou URL na základě poskytnutých parametrů.

Výsledkem každé z těchto funkcí je asynchronní volání metody get, put, post nebo delete objektu axiosInstance s odpovídající URL. Použití těchto funkcí v rámci aplikace by mělo umožnit efektivní správu poznámek na serveru.

Soubor planservice.ts

Soubor planservice.ts obsahuje sadu funkcí pro provádění asynchronních HTTP požadavků na server v souvislosti s plány a inventářem. Tyto funkce slouží k interakci s backendovým API a umožňují získání informací o plánech, inventáři a položkách v místnostech.

První dvě funkce, getPlanAllRequest a getPlanListRequest, jsou navrženy pro získání informací o všech plánech nebo seznamu plánů. Každá z těchto funkcí provádí HTTP požadavek na specifický endpoint serveru.

Třetí funkce, getPlanFloorImageRequest, slouží k získání obrázku patra na základě poskytnutého identifikátoru patra (floorId). Tato funkce také provádí HTTP požadavek na specifický endpoint serveru.

Poslední funkce, getPlanItemsRequest, je koncipována pro získání položek na základě různých parametrů, jako je místnost, místo, inventář atd. Funkce vytváří dynamickou URL na základě poskytnutých parametrů a poté provádí HTTP požadavek na server.

Výsledkem každé z těchto funkcí je asynchronní volání metody get objektu axiosInstance s odpovídající URL. Použití těchto funkcí v rámci aplikace by mělo umožnit efektivní správu plánů a inventáře na serveru.

Soubor searchFormService.ts

Zdrojový soubor searchFormService.ts obsahuje sadu funkcí pro provádění asynchronních HTTP požadavků na server, které jsou zaměřeny na načítání dat pro různá formulářová pole a seznamy ve vyhledávacím formuláři. Tyto funkce jsou navrženy tak, aby získávaly informace o inventářích, zemích, institucích, městech, národnostech, technikách, předmětech, jménech umělců a plánech.

Prvních sedm funkcí, fetchInventoriesRequest až fetchTechniquesRequest, jsou určeny pro získání dat pro různá formulářová pole. Každá funkce provede HTTP požadavek na specifický serverový endpoint (/inventories, /form/country, /form/institution atd.) a vrátí odpověď obsahující potřebná data.

Následující funkce fetchSubjectsRequest a fetchArtistNamesRequest slouží k získání seznamu předmětů a jmen umělců z endpointů /subject a /name/all.

Poslední funkce fetchPlanRequest má za úkol získat seznam plánů pomocí HTTP požadavku na /plan/list.

Výsledkem těchto funkcí je asynchronní volání metody get objektu axiosInstance s odpovídajícími URL. Tyto funkce umožňují efektivní načítání dat pro různá pole a seznamy ve vyhledávacím formuláři aplikace. Jejich použití přispívá k funkcionalitě vyhledávacího rozhraní, které může být využíváno uživateli k filtrování a hledání specifických informací v aplikaci.

Soubor searchService.ts

Zdrojový soubor searchService.ts obsahuje sadu funkcí pro provádění asynchronních HTTP požadavků na server, které slouží k vyhledávání specifických informací v aplikaci. Tato funkcionalita umožňuje uživatelům provádět pokročilé vyhledávání s využitím různých parametrů.

Soubor začíná definicí rozhraní SearchParams, které obsahuje mnoho vstupních parametrů pro vyhledávání, například inventáře, místnosti, umělce, národnosti, předměty, techniky, a další. Tyto parametry slouží k filtraci výsledků vyhledávání.

Následují dvě pomocné funkce: composeSearchParams a composeBooleanSearchParams. Tyto funkce slouží k sestavení části URL pro vyhledávací dotaz na základě předaných parametrů. První funkce je

určena pro parametry, které mohou mít hodnotu v poli (např. inventáře, místnosti), zatímco druhá se používá pro boolean hodnoty (např. true/false pro identifikované umění).

Samotná hlavní funkce searchRequest sestavuje kompletní URL pro vyhledávací dotaz pomocí předaných parametrů. Tato funkce využívá dříve definované pomocné funkce pro vytvoření specifických částí URL pro různé typy parametrů. Následně volá metodu get objektu axiosInstance s vytvořeným URL a vrátí výsledek asynchronního požadavku.

Celkově soubor umožňuje dynamické sestavení URL pro vyhledávání na základě různých kombinací vstupních parametrů a předává tuto konfiguraci na server pomocí asynchronního HTTP požadavku. Tím poskytuje robustní vyhledávací mechanismus, který lze snadno rozšiřovat o další parametry podle potřeby.

Podadresář *components*

Podadresář *general*

Soubor Dialog.tsx

Zdrojový soubor Dialogs.tsx obsahuje definici komponenty pro dialogové okno potvrzení (ConfirmDialog). Tato komponenta využívá knihovnu native-base pro vytváření uživatelského rozhraní v mobilních aplikacích napsaných v React Native.

Komponenta ConfirmDialog přijímá několik vstupních parametrů (props), které umožňují konfiguraci vzhledu a chování dialogového okna. Tyto parametry zahrnují:

- isShown: Booleovská hodnota, která indikuje, zda je dialogové okno aktuálně viditelné.
- headerText: Volitelný text pro zobrazení v hlavičce dialogového okna.
- bodyText: Volitelný text pro zobrazení v těle dialogového okna.
- confirmText: Volitelný text pro tlačítko potvrzení.
- confirmColor: Barva tlačítka potvrzení.
- cancelRef: Reference na tlačítko pro zavření dialogového okna.
- onClose: Funkce, která se spustí při zavření dialogového okna.
- onSubmit: Funkce, která se spustí při potvrzení dialogového okna.

Komponenta vrací JSX element reprezentující dialogové okno potvrzení. Pokud je isShown true, je zobrazen prvek AlertDialog s obsahem nastaveným na základě předaných parametrů. Dialogové okno obsahuje různé části, včetně záhlaví (AlertDialog.Header), těla (AlertDialog.Body), tlačítek pro zavření a potvrzení, a dalších. Tlačítka jsou definována jako součásti skupiny tlačítek (Button.Group), která obsahuje tlačítko "Cancel" a tlačítko pro potvrzení s odpovídajícími funkcemi obsluhy událostí.

Celkově lze tuto komponentu použít k vytváření dialogových oken potvrzení v mobilních aplikacích napsaných v React Native s využitím knihovny native-base.

Soubor Icons.tsx

Icons.tsx obsahuje definice několika ikon, které jsou implementovány jako komponenty pro knihovnu native-base a využívají komponenty Icon a grafické prvky z react-native-svg. Tyto ikony jsou používány v mobilních aplikacích napsaných v React Native pro vizuální reprezentaci různých prvků nebo akcí.

První ikona je MessageIcon, která zobrazuje symbol zprávy. Tato ikona má konkrétní tvar definovaný v elementu Path z react-native-svg.

Druhá ikona je EditIcon, která představuje symbol úpravy nebo editace. Tato ikona má dvě části: první část je tvar pera nebo nástroje pro kreslení, a druhá část je čtverec s dvěma výřezy, což může představovat položený list papíru.

Třetí ikona je FullscreenIcon, která představuje symbol na celou obrazovku. Tato ikona má tvar čtyř čtverců, které mohou symbolizovat jednotlivé obrazovky nebo okna.

Čtvrtá ikona je TargetIcon, která reprezentuje symbol cíle nebo terče. Tato ikona má dvě linie směřující ke středu a dvě vodorovné linie, což může představovat kruhový terč.

Poslední ikona je ExitfullscreenIcon, která představuje symbol pro opuštění režimu celé obrazovky. Tato ikona obsahuje kombinaci čtverců a linií, které mohou naznačovat snížení rozlišení nebo zmenšení obrazovky.

Každá ikona má parametr color, který určuje barvu ikony a je předáván jako vstupní vlastnost (prop). Tyto ikony mohou být použity v uživatelském rozhraní aplikace pro přidání vizuálních prvků a zlepšení uživatelského zážitku.

Podadresář item

Soubor ItemTabBar.tsx

Zdrojový soubor ItemTabBar.tsx obsahuje implementaci vlastního záložkového panelu pro komponenty v React Native. Tato komponenta je určena pro zobrazení záložek souvisejících s konkrétním prvkem (Item) a poskytuje uživatelské rozhraní pro navigaci mezi těmito záložkami.

Komponenta začíná importem různých komponent z knihoven native-base, včetně tlačítek, ikon, layoutů a dalších. Dále jsou importovány některé další komponenty z projektu, jako například InfoToast a LoadingBox. Také jsou načteny některé typy a konstanty používané v komponentě.

Samotná komponenta ItemTabBar přijímá několik propojovaných hodnot, jako jsou navigationState (stav navigace), concordances (seznam souvisejících prvků), index (index aktuálně vybrané záložky), onIndexChange (funkce pro změnu vybrané záložky), onBackPressed (funkce pro návrat zpět), nextItem (název nebo identifikátor následujícího prvku) a prevItem (název nebo identifikátor předchozího prvku).

V těle komponenty je podmínka, která zjišťuje, zda existují nějaké související prvky (concordances). Pokud ano, komponenta vytváří záložkový panel s tlačítky pro navigaci na předchozí a následující prvek. Tlačítko "Back" vrací uživatele na předchozí obrazovku. Tlačítka s ikonami šipky vlevo a vpravo provedou navigaci na předchozí a následující prvek. V závislosti na existenci předchozího nebo následujícího prvku může být navigace buď provedena, nebo se zobrazí oznámení (toast) o nedostatku předchozího nebo následujícího prvku.

Dále je implementováno zobrazení záložek v horizontálním scrollovatelném seznamu. Každá záložka obsahuje ikonu kruhu s určitou barvou a identifikátorem souvisejícího prvku. Vybraná záložka má zvýrazněnou spodní hranu.

Pokud neexistují žádné související prvky, zobrazí se komponenta LoadingBox s informací o načítání souvisejících prvků.

Celkově komponenta ItemTabBar poskytuje uživatelsky přívětivý a interaktivní záložkový panel pro procházení souvisejících prvků a navigaci v mobilní aplikaci napsané v React Native.

Soubor ItemView.tsx

Zdrojový soubor ItemView.tsx implementuje komponenty pro zobrazení detailů prvku (Item). Soubor obsahuje tři hlavní části: ItemDetail, ItemNotes, a samotný ItemView.

První část, ItemDetail, je zodpovědná za zobrazení detailů konkrétního prvku. Pokud jsou data o prvku načítána (props.itemLoading je true), zobrazí se načítací box, jinak bude zobrazena struktura obsahující informace o autorovi, názvu díla, inventory itemu, a dalších vlastnostech prvku. V případě existence obrázků, umožňuje uživateli scrollovatelný výběr mezi nimi.

Druhá část, ItemNotes, se stará o zobrazení poznámek k danému prvku. Může zobrazovat buď všechny poznámky, nebo pouze ty, které jsou přímo související s prvky (showRelatedComments). Uživatel může přepínat mezi oběma zobrazeními pomocí přepínače. Načítací box se zobrazí, pokud jsou poznámky načítány.

Třetí část, ItemView, je nadřazený komponent, který obsahuje buď ItemDetail nebo ItemNotes podle aktuálního stavu (itemShown). Zároveň obsahuje tlačítka pro přepínání mezi zobrazením detailů prvku a zobrazením poznámek. Při kliknutí na tlačítko se mění stav a vybraná část (ItemDetail nebo ItemNotes) se zobrazí.

Celý soubor využívá různé komponenty a prvky ze knihovny native-base pro vytváření uživatelsky přívětivého a responzivního uživatelského rozhraní v React Native. Také využívá React Hooks pro sledování stavu a událostí, což přispívá k interaktivitě komponenty. Využívá Redux pro správu stavu a komunikaci se storem aplikace.

Podadresář listView

Soubor ItemPreview.tsx

Soubor ItemPreview.tsx definuje komponentu ItemPreview, která slouží k zobrazení náhledu prvku. Tato komponenta je určena pro použití v seznamu náhledů prvků, například v seznamu výsledků vyhledávání.

Komponenta ItemPreview přijímá několik vstupních vlastností (props), včetně caption (popisek), title (název), name (jméno), image (cesta k obrázku), itemId (identifikátor prvku), a inventoryLabel (popis inventárního štítku). Také přijímá navigační objekt pro možnost přechodu na detaily prvku.

Vizuální struktura komponenty obsahuje Pressable (stisknutelný kontejner), který reaguje na stisknutí a přechází na detaily prvku pomocí navigačního objektu. Uvnitř Pressable je HStack, který obsahuje buď obrázek prvku nebo komponentu ItemPreviewMissingImage (v případě, že obrázek není k dispozici), a VStack, který obsahuje textové informace o prvku, jako je jméno, popisek a název.

Pokud má prvkek obrázek (props.image existuje), je zobrazen pomocí komponenty Image, jinak je zobrazena komponenta ItemPreviewMissingImage. V textovém bloku (VStack) jsou zobrazeny informace o prvku v různých velikostech písma. Celá struktura je responzivní, což znamená, že se přizpůsobuje dostupnému prostoru.

Komponenta využívá importovaných komponent ze native-base, včetně Box, Center, HStack, Image, Pressable, ScrollView, Text a VStack. Taktéž využívá komponenty ItemPreviewMissingImage a konstanty ITEM_PREVIEW_IMG_PREFIX_URL z externích zdrojů.

Celkově je účelem této komponenty vytvořit vizuálně přitažlivý a funkční náhled prvku, který umožňuje uživateli rychlý přechod na detaily prvku.

Soubor ItemPreviewMissingImage.tsx

ItemPreviewMissingImage.tsx obsahuje komponentu ItemPreviewMissingImage, která se používá v případě, že prvku chybí obrázek. Tato komponenta vytváří náhradní vizuální prvek, který zahrnuje popisek inventárního štítku.

Komponenta přijímá několik vstupních vlastností (props), včetně inventoryLabel (popis inventárního štítku), w (šířka) a h (výška). Výchozí hodnoty pro šířku a výšku jsou nastaveny na 65, pokud není specifikováno jinak.

Vizuální struktura komponenty obsahuje centrální kontejner (Center), v němž se nachází kontejner Box. Tento kontejner má nastavenou barvu pozadí (bg), šířku (w), výšku (h), barvu ohraničení (borderColor), šířku ohraničení (borderWidth), a je zarovnán na střed (alignItems a justifyContent).

Uvnitř kontejneru Box je vertikální kontejner (VStack), který obsahuje dvě textové komponenty (Text). První textová komponenta (Text) obsahuje první část popisku inventárního štítku (firstPart), která je získána z popisku od začátku do prvního mezerového znaku. Druhá textová komponenta obsahuje druhou část popisku inventárního štítku (secondPart), která je získána od prvního mezerového znaku do konce popisku.

Efekt useEffect je použit k aktualizaci stavu firstPart a secondPart v reakci na změny vlastnosti props.inventoryLabel. Při každé změně props.inventoryLabel se provede aktualizace první a druhé části popisku, a to na základě pozice prvního mezerového znaku.

Celkově tato komponenta vytváří vizuálně přitažlivý náhradní prvek pro případ, že prvku chybí obrázek. Tento náhradní prvek obsahuje informace z inventárního štítku a je přizpůsobený podle šířky a výšky definovaných vstupními vlastnostmi.

Soubor ListView.tsx

Zdrojový soubor ListView.tsx obsahuje komponentu ListView, která se stará o zobrazení seznamu položek na základě výsledků vyhledávání. Komponenta přijímá různé vstupy, včetně informací o inventářích, počtu výsledků, chybě, načtených stránkách dat a dalších.

Na začátku souboru jsou importovány potřebné knihovny, včetně useEffect a useState z Reactu, komponenty ScrollView, Text, VStack a useToast z knihovny NativeBase, a useSelector z knihovny React Redux. Dále jsou importovány další komponenty a typy, které jsou používány v rámci této komponenty.

Samotná komponenta ListView obsahuje JSX kód, který zajišťuje zobrazení seznamu položek na základě poskytnutých dat. Komponenta mapuje inventáře a pro každý inventář zobrazuje skupinu položek v rámci komponenty ListViewInventoryGroup. Tato skupina obsahuje informace o inventáři, seznam položek, stav načítání a načtených stránkách dat. Kromě toho komponenta obsahuje také logiku pro zobrazování chybového hlášení pomocí komponenty ErrorToast v případě, že došlo k chybě při získávání dat.

Lze tvrdit, že ListView.tsx slouží k organizaci a zobrazení výsledků vyhledávání, přičemž umožňuje uživateli procházet položky v rámci jednotlivých inventářů. Chybové hlášky jsou prezentovány pomocí informačního toastu.

Soubor ListViewInventoryGroup.tsx

Soubor ListViewInventoryGroup.tsx obsahuje komponentu ListViewInventoryGroup, která je odpovědná za zobrazení skupiny položek podle inventáře v rámci seznamu. Tato komponenta přijímá různé vstupy, včetně informací o inventáři, načtených položkách, stavu načítání, stránkách s načtenými daty a další.

Na začátku souboru jsou importovány potřebné komponenty z knihovny NativeBase, stejně jako typ ItemPreviewType a komponenta ItemPreview. Dále jsou importovány různé hooky z Reactu, včetně useDispatch, useSelector, AppDispatch a RootState, které jsou nezbytné pro práci s Redux store. Tato komponenta využívá akce loadItemsByInventory z Redux thunks pro načítání položek podle inventáře.

Samotná komponenta ListViewInventoryGroup obsahuje JSX kód, který zajišťuje zobrazení skupiny položek v rámci seznamu. Komponenta zahrnuje několik funkcí pro parsování informací o umělcích, obrazech a načítání dalších položek. Dále obsahuje logiku pro načítání dalších položek při rozkliknutí inventáře a také při stisknutí tlačítka "Load more", pokud jsou dostupná další data.

Slouží k organizaci a zobrazení skupiny položek podle inventáře v rámci seznamu. Tato komponenta se stará o zobrazování a načítání dat v reakci na uživatelské interakce.

Podadresář loading

Soubor LoadingBox.tsx

Soubor LoadingBox.tsx obsahuje komponentu LoadingBox, která je navržena pro zobrazení vizuálního prvku označujícího probíhající načítání nebo nějakou jinou aktivitu, která vyžaduje oznámení uživateli. Tato komponenta přijímá vstup ve formě textu, který slouží k popisu činnosti nebo procesu, který právě probíhá.

Na začátku souboru jsou importovány potřebné komponenty z knihovny NativeBase, konkrétně Box, HStack, Heading a Spinner. Dále je importován modul React. Komponenta je deklarována s rozhraním LoadingBoxProps, které definuje očekávaný vstup v podobě textového popisu.

Samotná komponenta LoadingBox obsahuje JSX kód pro vytvoření boxu s načítacím prvkem (Spinner) a nadpisem (Heading). Box je nastaven na šířku 100 %, s odstupem od horního okraje a pevnou výškou. Vnitřní HStack slouží k horizontálnímu rozmístění prvků, kde Spinner představuje vizuální indikátor načítání a Heading obsahuje textový popis, který je získán ze vstupních vlastností komponenty.

LoadingBox.tsx je komponentou pro zobrazení jednotného vizuálního prvku informujícího uživatele o probíhajícím procesu nebo činnosti. Tato komponenta slouží ke zlepšení uživatelského zážitku tím, že poskytuje vizuální zpětnou vazbu o stavu načítání.

Podadresář notes

Soubor NotesListView.tsx

Soubor NotesListView.tsx obsahuje komponentu NotesListView, která slouží k zobrazení seznamu poznámek (Note). Tato komponenta je navržena pro práci s poznámkami, umožňuje zobrazení, přidání, editaci a mazání poznámek. Importuje potřebné komponenty z knihovny native-base, včetně VStack, Box, Text, HStack, ScrollView, Flex, Button, CloseIcon, IconButton, TextArea, useToast. Dále importuje komponentu NoteView a několik dalších komponent.

Hlavní část komponenty obsahuje logiku pro zobrazení seznamu poznámek. V této části je také obsažen kód pro načítání dalších poznámek při dosažení konce seznamu. Při posledním záznamu se zobrazí odpovídající informační zpráva.

Komponenta umožňuje odpovídání na poznámky, editaci a mazání poznámek. K tomu využívá různé callback funkce, které jsou volány v reakci na interakce uživatele. Také obsahuje výškově dynamický box, který přizpůsobuje výšku prostoru pro poznámky a odpovědi na základě přítomnosti odpovědi.

Pod komponentou pro zobrazení poznámek je formulář pro vytvoření nové poznámky. Tento formulář je viditelný pouze v případě, že je nastavena odpovídající callback funkce handleCreateComment.

Obsahuje komplexní komponentu pro práci s poznámkami, poskytující uživatelsky přívětivé rozhraní pro interakci s daty.

Soubor NoteView.tsx

NoteView.tsx obsahuje React komponentu NoteView, která slouží k zobrazování jednotlivých poznámek (Note). Tato komponenta se stará o prezentaci informací o poznámkách včetně možnosti odpovědí, editací a mazání. Importuje potřebné komponenty z knihovny native-base a další součásti.

Hlavní část komponenty obsahuje prezentaci informací o poznámce včetně jména uživatele, data vytvoření, samotného textu poznámky a případného přiloženého předmětu (item). Zároveň umožňuje odpovídání na poznámky, zobrazuje odpovědi na stávající poznámky a nabízí možnost editace a mazání vlastní poznámky.

Komponenta obsahuje i logiku pro zobrazení odpovědí na konkrétní poznámku, které jsou dynamicky zobrazovány v odsazeném bloku (VStack). Pro uživatele se tak nabízí přehledné rozhraní k interakci s poznámkami.

V rámci vizuálního zvýraznění byla implementována možnost zvýraznit vybranou poznámku v rámci celkového seznamu. Požadavky na zvýraznění jsou určeny přes prop higlighted.

Významné funkce komponenty zahrnují možnost odpovědi na poznámku, zobrazení a skrytí odpovědí na konkrétní poznámku, editace a mazání poznámky. Všechny tyto akce jsou propojeny s odpovídajícími callback funkcemi, které jsou předány z nadřazené komponenty NotesListView.

Vyplývá z toho, že soubor NoteView.tsx obsahuje komplexní komponentu pro zobrazování a interakci s individuálními poznámkami v rámci seznamu

Podadresář plan

Soubor CastlePlanView.tsx

Soubor CastlePlanView.tsx obsahuje React komponentu CastlePlanView, která slouží k zobrazení plánu hradu s možností pohybu, přibližování a oddalování. Importuje potřebné komponenty z knihovny native-base, react-native, react-native-gesture-handler a další součásti.

Tato komponenta je vytvořena pro zobrazování plánů hradu s vyznačením jednotlivých místností. Přináší interaktivní možnosti přiblížení, oddálení a posunu plánu. Zahrnuje také ovládací prvky pro přepínání mezi režimem plné obrazovky a normálním režimem.

Hlavní funkce komponenty zahrnují:

1. **Interaktivní Pohyb a Zoom:**
    - Komponenta využívá knihovnu react-native-gesture-handler pro obsluhu gest a reaguje na různé dotykové události, jako je táhnutí prstem, přiblížení a oddálení dvěma prsty.
    - Využívá animované styly a sdílené hodnoty pro plynulé pohyby a změny měřítka.
2. **Zobrazování Plánu:**
    - Zobrazuje plán hradu, který je předán komponentě jako obrázek ve formátu SVG. Plán může obsahovat různé místnosti reprezentované cestami (Paths) v SVG.
    - Místnosti jsou vyplněny barvou a mají ohraničující obrys. Zvýrazněná místnost má jinou barvu než ostatní.
3. **Ovládací Panely:**
    - Obsahuje ovládací panely pro přepínání mezi plným režimem a normálním režimem. Tato tlačítka jsou vykreslena na horním pravém rohu komponenty.
4. **Funkce Zoomu na Místnost:**
    - Obsahuje funkci pro přiblížení k vybrané místnosti, což je užitečné, pokud uživatel chce získat detaily o konkrétní místnosti.
5. **Výpočet Rozměrů:**
    - Zahrnuje výpočty související s rozměry SVG pro správnou funkci gest a škálování.
6. **Označení Čísel Místností:**
    - Zobrazuje čísla místností na plánu.

Komponenta CastlePlanView poskytuje uživatelsky přívětivé rozhraní pro interaktivní prozkoumávání plánu hradu a získávání informací o jednotlivých místnostech.

Podadresář reusables

Soubor ApplicationHeading.tsx

Soubor ApplicationHeading.tsx obsahuje React komponentu s názvem ApplicationHeading, která je vytvořena pomocí knihovny native-base. Účelem této komponenty je zobrazit nadpis aplikace "Inventaria Rudolphina" ve středu stránky s definovanými stylovými vlastnostmi.

Komponenta využívá komponentu Heading z knihovny native-base, která je zodpovědná za zobrazení textu jako nadpisu. V rámci této komponenty jsou definovány následující stylové vlastnosti:

- size="2xl": Určuje velikost textu jako velký nadpis (2xl).
- color="primary.500": Nastavuje barvu textu na barvu definovanou v tématu aplikace (v tomto případě primární barva s odstínem 500).
- alignSelf={"center"}: Umožňuje umístit nadpis do středu vodorovně v rámci nadřazeného kontejneru.
- textAlign="center": Vytváří zarovnání textu na střed vodorovně.
- maxW={"80 %"}: Nastavuje maximální šířku nadpisu na 80% šířky nadřazeného kontejneru, což může být užitečné pro responzivní design.

Samotný text nadpisu je stanoven na "Inventaria Rudolphina". Tato komponenta by mohla být využívána v různých částech aplikace tam, kde je potřeba zobrazit tento konkrétní nadpis. Její jednoduchá implementace umožňuje snadné a opakované použití v různých částech aplikace, což přispívá k jednoduchosti a udržitelnosti kódu.

Podadresář search

Soubor MultiSelect.tsx

Zdrojový soubor MultiSelect.tsx obsahuje React komponentu s názvem MultiSelect, která implementuje víběr více prvků z předem definovaného seznamu. Komponenta je vytvořena s využitím knihovny native-base pro vzhled a chování komponent. Zde je stručný popis klíčových částí souboru:

1. **Importy a Knihovny:**
    - Importy jsou uvedeny na začátku souboru a zahrnují komponenty a knihovny nezbytné pro implementaci vícevýběru a vizuálních prvků.
2. **Rozhraní a Stav:**
    - MultiSelect přijímá několik vlastností pomocí props, včetně data (seznam položek k výběru), label (popisek komponenty), selectedItems (již vybrané položky) a onSelectedItemsChange (metoda pro aktualizaci vybraných položek).
    - Vnitřní stav komponenty zahrnuje isSelected (pole pro sledování vybraných položek), query (stav pro vyhledávací dotaz) a isOpen (stav zobrazování dialogu).
3. **Filtrování Dat:**
    - Komponenta obsahuje logiku pro filtrování dat na základě zadaného vyhledávacího dotazu. Filtrují se položky na základě shody s názvem.
4. **Správa Vybraných Položek:**
    - Komponenta udržuje stav vybraných položek a aktualizuje ho podle předaných vybraných položek před vykreslením.
5. **Otevření a Zavření Dialogu:**
    - Dialog pro výběr položek je otevřen nebo zavřen pomocí Actionsheet z knihovny native-base.

6. **Rozhraní Uživatele:**
    - Komponenta zobrazuje již vybrané položky v podobě štítků (Badge). Uživatel může přidávat nebo odebírat položky a používat vyhledávání.
7. **Ovládací Tlačítka:**
    - K ovládání dialogu jsou přidány tlačítka "Cancel" a "Done". Tlačítko "Cancel" zavře dialog, zatímco tlačítko "Done" uloží vybrané položky a zavře dialog.
8. **Zpracování Uživatelské Akce:**
    - Kód obsahuje funkce pro zpracování uživatelských akcí, např. přepínání stavu vybraných položek, zrušení výběru položky a potvrzení provedených změn.

Tento soubor obsahuje kompletní implementaci multi-výběru s možností vyhledávání a vizuálním zobrazením vybraných položek.

Soubor SearchForm.tsx

Soubor SearchForm.tsx obsahuje implementaci komponenty SearchForm, která představuje formulář pro vyhledávání v rámci inventářů. Následující text poskytuje souvislý akademický popis klíčových prvků a funkcí tohoto zdrojového souboru:

Soubor začíná importy potřebných komponent z knihovny native-base a také z React knihovny pro správu stavu a Redux pro správu globálního stavu aplikace. Dále jsou importovány definice typů a akcí, které jsou použity v rámci formuláře. Komponenta využívá také další vlastní komponenty, konkrétně MultiSelect a SwitchWithLabel.

Samotná komponenta SearchForm je funkcí, která přijímá vlastnost inventoryId. Vnitřně udržuje množství stavů pro sledování vybraných položek v rámci různých kategorií, jako jsou inventáře, národnosti, umělci, místnosti a další. Kromě toho udržuje stav pro vyhledávací dotaz a několik stavů pro filtrování.

Významnou částí komponenty je metoda searchSubmit, která je volána při odeslání formuláře. Tato metoda shromažďuje informace o vybraných položkách a stavech a poté dispečeruje akce Redux pro aktualizaci filtrů a provedení vyhledávání. Po provedení vyhledávání se stav formuláře zavře.

Další významnou funkcionalitou je metoda clearForm, která slouží k vymazání vybraných položek ve formuláři. Tato metoda je volána při resetování formuláře.

Komponenta rovněž obsahuje metodu getSearchHeader, která generuje hlavičku pro zobrazení vyhledávacího dotazu v závislosti na vybraných položkách.

V neposlední řadě, v souboru jsou efekty React Hook useEffect, který reaguje na změny vlastnosti props.inventoryId. Pokud je inventoryId definováno, provede se vyčištění formuláře, nastavení vybraného inventáře, a provede se vyhledávání pro daný inventář.

Souhrnně lze stanovit, že soubor SearchForm.tsx obsahuje kompletní implementaci formuláře pro vyhledávání v inventářích, s podporou výběru více kategorií a filtrování na základě různých kritérií.

Soubor SwitchWithLabel.tsx

SwitchWithLabel.tsx obsahuje implementaci komponenty SwitchWithLabel, která kombinuje přepínač (Switch) s textovým popiskem (Text). Tato komponenta slouží k zobrazení a ovládání boolean hodnoty spolu s popiskem.

Komponenta SwitchWithLabel je definována jako funkce, která přijímá vlastnosti (props) specifikované v rozhraní SwitchWithLabelProps. Tato rozhraní obsahuje tři klíčové vlastnosti: label pro popisek, value pro hodnotu přepínače a onValueChange pro obsluhu události změny hodnoty.

V těle komponenty je použita komponenta View z knihovny native-base, která definuje kontejner pro uspořádání obsahu. Tento kontejner má nastaveny vlastnosti flexDirection, alignItems a justifyContent, aby obsahoval řádek s textem a přepínačem. Text obsahuje hodnotu popisku z vlastnosti label.

Samotný přepínač (Switch) je zobrazen s aktuální hodnotou z vlastnosti value a obsluhou změny hodnoty z vlastnosti onValueChange. Velikost přepínače je explicitně nastavena na malou (size={"sm"}). Každý přepínač je identifikován pomocí klíče, který je odvozen od popisku (key={props.label}).

Soubor SwitchWithLabel.tsx obsahuje jednoduchou a znovupoužitelnou komponentu pro zobrazení přepínače s textovým popiskem a umožňuje ovládání hodnoty pomocí události změny.

Podadresář toast

Soubor SuccessToast.tsx

SuccessToast.tsx obsahuje implementaci komponenty SuccessToast, která slouží k zobrazení upozornění na úspěšnou událost. Tato komponenta využívá knihovnu native-base a sestává z několika komponent, včetně Alert, VStack, HStack, Text, IconButton, a dalších.

Komponenta SuccessToast má tři vstupní vlastnosti (props), a to headerText (pro nadpis upozornění), text (pro hlavní text upozornění) a onClose (pro obsluhu zavírání upozornění).

Celá komponenta je zabalena do komponenty Alert, což je kontejner pro zobrazení upozornění. Tato komponenta má nastaveny různé vlastnosti, jako je alignSelf, flexDirection, variant, status, a backgroundColor, které ovlivňují vizuální styl upozornění. Například, status="success" určuje, že upozornění bude mít zelený barevný kód, což je typické pro úspěšné události.

Vnitřek Alert obsahuje komponentu VStack, což je kontejner pro vertikální uspořádání prvků. Ve VStack jsou obsaženy další komponenty, jako například HStack, která slouží k horizontálnímu uspořádání prvků.

V rámci HStack se nachází ikona (Alert.Icon), text (buď headerText nebo text podle dostupnosti), mezeru (Spacer) a tlačítko (IconButton). Tlačítko má ikonu uzavíracího kříže a je definováno tak, aby zavolalo funkci onClose při stisknutí.

Pod HStack se nachází další text, který obsahuje text (hlavní text upozornění), pokud je definován headerText.

Komponenta SuccessToast vytváří vizuálně příjemné upozornění na úspěšnou událost, které zahrnuje nadpis, hlavní text a možnost zavření.

Soubor ErrorToast.tsx

Zdrojový soubor ErrorToast.tsx obsahuje implementaci komponenty ErrorToast, která slouží k zobrazení upozornění na chybu. Stejně jako u předchozí komponenty SuccessToast, tato komponenta využívá knihovnu native-base a skládá se z několika komponent, včetně Alert, VStack, HStack, Text, IconButton, a dalších.

Komponenta ErrorToast má tři vstupní vlastnosti (props), a to headerText (pro nadpis upozornění na chybu), text (pro hlavní text upozornění) a onClose (pro obsluhu zavírání upozornění).

Celá komponenta je zabalena do komponenty Alert, která slouží jako kontejner pro zobrazení upozornění. V rámci této komponenty jsou nastaveny různé vizuální vlastnosti, jako je alignSelf, flexDirection, variant, status, a backgroundColor. Specifikace status="warning" určuje, že upozornění bude mít žlutý barevný kód, což je typické pro chybová hlášení.

Vnitřek Alert obsahuje komponentu VStack, což je kontejner pro vertikální uspořádání prvků. Ve VStack se nachází další komponenty, například HStack, která slouží k horizontálnímu uspořádání prvků.

V rámci HStack se nachází ikona (Alert.Icon), text (buď headerText nebo text podle dostupnosti), mezera (Spacer) a tlačítko (IconButton). Tlačítko má ikonu uzavíracího kříže a je definováno tak, aby zavolalo funkci onClose při stisknutí.

Pod HStack se nachází další text, který obsahuje text (hlavní text upozornění), pokud není definován headerText.

Komponenta ErrorToast vytváří vizuálně výrazné upozornění na chybu, které zahrnuje nadpis, hlavní text a možnost zavření. Barva a ikony jsou vybrány tak, aby vizuálně odpovídaly chybě nebo varování.

Soubor InfoToast.tsx

Zdrojový soubor InfoToast.tsx obsahuje implementaci komponenty InfoToast, která slouží k zobrazení informativního upozornění. Stejně jako u předchozích komponent SuccessToast a ErrorToast, tato komponenta využívá knihovnu native-base a skládá se z několika komponent, včetně Alert, VStack, HStack, Text, IconButton, a dalších.

Komponenta InfoToast má tři vstupní vlastnosti (props), a to headerText (pro nadpis upozornění), text (pro hlavní text upozornění) a onClose (pro obsluhu zavírání upozornění).

Celá komponenta je zabalena do komponenty Alert, která slouží jako kontejner pro zobrazení upozornění. V rámci této komponenty jsou nastaveny různé vizuální vlastnosti, jako je alignSelf, flexDirection, variant, status, a backgroundColor. Specifikace status="info" určuje, že upozornění bude mít modrý barevný kód, což je typické pro informativní zprávy.

Vnitřek Alert obsahuje komponentu VStack, což je kontejner pro vertikální uspořádání prvků. Ve *VStack* se nachází další komponenty, například HStack, která slouží k horizontálnímu uspořádání prvků.

V rámci HStack se nachází ikona (Alert.Icon), text (buď headerText nebo text podle dostupnosti), mezera (Spacer) a tlačítko (IconButton). Tlačítko má ikonu uzavíracího kříže a je definováno tak, aby zavolalo funkci onClose při stisknutí.

Pod HStack se nachází další text, který obsahuje text (hlavní text upozornění), pokud není definován headerText.

Komponenta InfoToast vytváří vizuálně výrazné upozornění s modrým podkladem, které zahrnuje nadpis, hlavní text a možnost zavření. Barva a ikony jsou vybrány tak, aby vizuálně odpovídaly informativní zprávě.

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

HomePage.tsx obsahuje implementaci komponenty HomePage, která reprezentuje hlavní stránku aplikace. Tato komponenta využívá knihovny native-base a dalších modulů pro správu stavu, navigaci a asynchronní operace.

Komponenta HomePage začíná importem potřebných komponent a funkcí z knihoven. Mezi ně patří komponenty jako Center, Image, Pressable, ScrollView, Text a useToast z native-base. Dále jsou importovány funkce z react-redux pro práci se stavem Redux, včetně useDispatch a useSelector. Importuje také další potřebné věci, jako je DrawerScreenProps a RootDrawerParamList pro správu navigace, a několik funkcí pro práci s logováním a konstanty.

Samotná komponenta HomePage je funkcionální komponentou, která přijímá navigační vlastnosti DrawerScreenProps pro zajištění interakce s navigačním panelem.

V rámci této komponenty jsou definovány několik stavových proměnných, včetně data pro uchování dat načtených ze stavu Redux, loading pro sledování stavu načítání, lastError pro uchování poslední chyby, a toast pro zobrazení oznámení.

Další částí komponenty je efekt, který je spuštěn v reakci na změny v lastError. Pokud je detekována chyba, vytvoří se oznámení (toast) pomocí komponenty ErrorToast, která obsahuje informace o chybě a nabízí možnost jejího zavření.

Následuje další efekt, který zajišťuje načtení dat, pokud nebyla načtena nebo je načítání dokončeno.

Samotný JSX kód obsahuje strukturu stránky, zahrnující komponenty jako LoadingBox (při načítání), ApplicationHeading (nadpis aplikace), ScrollView pro rolování obsahu, a různé komponenty pro zobrazení inventářů na domovské stránce.

Celkově lze říci, že soubor HomePage.tsx obsahuje implementaci hlavní stránky aplikace, včetně obsluhy stavu, asynchronních operací, a vizuální prezentace dat.

Soubor ItemViewPage.tsx

Zdrojový soubor ItemViewPage.tsx obsahuje implementaci komponenty ItemViewPage, která představuje stránku pro zobrazení detailů konkrétní položky (itemu) v aplikaci. Tato komponenta využívá řadu knihoven, včetně react, react-redux, react-native-tab-view, native-base a dalších.

Komponenta ItemViewPage začíná importem potřebných modulů a komponent. Mezi ně patří komponenty pro práci se stavem Redux, navigací, zobrazením záložek (TabView), okenními rozměry a zpracování chyb (ErrorToast). Importována je také samotná komponenta pro zobrazení položky (ItemView).

Následuje definice komponenty ItemViewPage, která přijímá navigační vlastnosti (route, navigation) pomocí DrawerScreenProps pro interakci s navigačním panelem.

Ve třech efektech useEffect jsou řešeny následující logické kroky:

1. **Načtení hlavní položky (itemu):**
    - Po prvním načtení stránky (nebo při změně itemId) se spustí efekt pro načtení hlavní položky (itemu).
    - Metoda getItem ze stavu Redux se volá s aktuálním itemId z parametrů cesty.
2. **Načtení konkordancí:**
    - Jakmile jsou konkrétní položky (itemy) načteny, efekt sleduje změny v konkordancích.

- Pokud jsou konkordance dostupné, vytváří se seznam routes pro jednotlivé konkordance, které se zobrazují v záložkách. Tím se vytváří dynamicky měnící se seznam záložek pro každou konkordanci.
- Konkrétní položky (itemy) jsou inicializovány načítacím boxem.

3. **Reakce na změnu hlavní položky (itemu):**
   - Když je hlavní položka načtena, efekt reaguje na změny v hlavní položce.
   - Pokud index záložky je 0 (první záložka), jsou načteny konkordance položky a poznámky k této položce.
   - V případě změny indexu (přepínání mezi záložkami) je volána metoda pro načtení konkrétní konkordance.

V poslední části komponenty je JSX kód, který zahrnuje buď zobrazování načítacího boxu nebo komponentu TabView, která obsahuje záložky pro jednotlivé konkordance. Záložky obsahují komponentu ItemView pro zobrazení detailů položky a zpracování chyb pomocí ErrorToast.

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

Soubor LoginPage.tsx

Soubor LoginPage.tsx obsahuje implementaci komponenty LoginPage, která reprezentuje stránku pro přihlášení uživatele do aplikace. Tato komponenta využívá mnoho komponent z knihovny native-base a také několik komponent pro zpracování chyb a informativních zpráv.

Na začátku jsou importovány potřebné komponenty a moduly, včetně komponent pro pracování s rozložením stránky, textovými poli, tlačítky, zpracování zpráv, a dalších. Dále jsou importovány akce (thunks) ze stavu Redux pro autentizaci uživatele a také pomocné funkce.

Následuje definice samotné komponenty LoginPage, která v sobě obsahuje:

1. **Stav pro uživatelské jméno a heslo:**
   - useState pro sledování změn hodnot uživatelského jména (username) a hesla (password).
2. **Získání stavu poslední chyby a použití toasta pro její zobrazení:**
   - Pomocí useSelector jsou získány informace o poslední chybě z globálního stavu.
   - Efekt useEffect reaguje na změny poslední chyby a zobrazuje chybový toast.
3. **Odeslání požadavku na přihlášení:**
   - Metoda loginUser se stará o odeslání požadavku na přihlášení s uživatelským jménem a heslem pomocí funkce dispatch.
4. **Zjištění stavu autentizace při načítání stránky:**
   - Efekt useEffect se stará o zjištění stavu autentizace při načítání stránky.
5. **Strukturovaný layout stránky:**
   - Používá se KeyboardAvoidingView pro adaptaci vzhledu na různých zařízeních.
   - Centrování obsahu pomocí Center a VStack.
   - Zobrazení hlavičky aplikace, nadpisu a popisu stránky.
   - V případě chyby se zobrazí další text s popisem chyby.
   - Formulář pro zadání uživatelského jména a hesla.
6. **Zobrazení tlačítka pro přihlášení:**

- Využití komponenty Button pro provedení přihlášení.
- Stavové proměnné username a password jsou aktualizovány na základě uživatelského vstupu.

Jedná se o implementaci stránky pro přihlášení uživatele, obsahující interaktivní formulář a ovládací prvky pro uživatelskou autentizaci.

Soubor Logout.tsx

Soubor Logout.tsx obsahuje implementaci komponenty Logout, která reprezentuje stránku nebo komponentu sloužící k odhlášení uživatele. Tato komponenta využívá několik komponent z knihovny native-base a také komponentu LoadingBox pro zobrazení načítání během odhlašování.

1. **Import modulů a komponent:**
   - Importuje se useDispatch pro získání dispečera z Reduxu a useEffect pro spuštění odhlašovací akce po načtení komponenty.
   - Importuje se AppDispatch pro typovou kontrolu dispečera a několik komponent pro design, jako Center, Heading, HStack, Spinner a vlastní komponentu LoadingBox.
2. **Odhlašovací efekt:**
   - Efekt useEffect je použit pro provedení akce odhlášení (logout) po načtení komponenty.
   - V tomto efektu je získán dispečer z Reduxu pomocí useDispatch, a následně je zavolána akce logout prostřednictvím dispečera. To vede k odhlášení uživatele.
3. **Zobrazení načítacího boxu:**
   - Návratová hodnota komponenty je komponenta LoadingBox, která zobrazuje text "Logging out..." a pravděpodobně nějaký prvek, jako je například spinner (Spinner), indikující probíhající proces odhlašování.

Komponenta Logout.tsx slouží k provedení odhlašovací akce v rámci Redux stavu aplikace a zobrazuje uživateli informaci o probíhajícím odhlašování pomocí načítacího boxu. Tímto způsobem poskytuje uživatelsky přívětivé rozhraní pro proces odhlášení.

Soubor Navigation.tsx

Tento zdrojový soubor obsahuje implementaci navigačního systému aplikace, který využívá knihovnu React Navigation a komponenty z knihovny NativeBase pro vytvoření responzivního uživatelského rozhraní. Navigační systém je sestaven z různých obrazovek, které jsou reprezentovány komponentami a které jsou dostupné buď přihlášeným uživatelům nebo těm, kteří nejsou přihlášeni.

1. **Importy a konfigurace:**
   - Importuje se mnoho modulů a komponent z React Navigation, NativeBase, Expo a dalších knihoven.
   - Vytváří se typ RootDrawerParamList, který definuje názvy obrazovek a případné parametry, které mohou být předány mezi obrazovkami.
2. **Rozvržení navigačních obrazovek:**
   - Definuje se komponenta Navigation, která obsahuje navigační kontejner NavigationContainer.
   - Závisí na tom, zda je uživatel přihlášen (loggedIn), a podle toho zobrazuje různé obrazovky.

- V případě přihlášení zobrazuje obrazovky jako HomePage, SearchPage, NotesViewPage, Logout, ItemViewPage, a PlanViewPage v rámci Drawer.Navigator. Každá obrazovka má své konkrétní komponenty a konfigurace.
- V případě, že uživatel není přihlášen, zobrazuje obrazovku LoginPage s příslušnými konfiguracemi.

3. **Zobrazení prvků v záhlaví:**
    - Pro všechny obrazovky, kde je uživatel přihlášen, je definován pravý horní roh záhlaví (headerRight), který obsahuje tlačítko s ikonou vyhledávání.

Soubor Navigation.tsx implementuje navigační strukturu pro mobilní aplikaci, která poskytuje uživatelsky přívětivé rozhraní s možností přepínání mezi různými obrazovkami a funkcemi.

Soubor NotesViewPage.tsx

Soubor NotesViewPage obsahuje implementaci obrazovky pro zobrazení a správu poznámek. Tato obrazovka umožňuje uživateli filtrovat a třídit poznámky podle různých kritérií a také vytvářet nové poznámky. Následuje souvislý text popisující klíčové aspekty tohoto souboru:

Soubor NotesViewPage.tsx definuje komponentu pro obrazovku, která zobrazuje seznam poznámek a umožňuje uživateli provádět různé operace, jako je filtrování, třídění a vytváření nových poznámek. Komponenta využívá různé komponenty z knihovny NativeBase a React Navigation pro vytvoření responzivního uživatelského rozhraní.

V této implementaci je využíván Redux pro správu stavu a interakce s úložištěm aplikace. Komponenta obsahuje mnoho hooků z Reactu, jako jsou useEffect, useState a useCallback, pro správu stavu komponenty a reakce na události.

Funkce a klíčové prvky souboru:

1. **Filtrování a třídění:**
    - Uživatel může filtrovat poznámky podle dvou kritérií: "My comments" a "General comments". Tato volba je realizována pomocí přepínačů.
    - Je implementována možnost třídění poznámek podle dvou kritérií: "Items" a "Date". Uživatel může měnit směr třídění (vzestupně/sestupně).
2. **Vytváření nových poznámek:**
    - Uživatel může vytvářet nové poznámky pomocí tlačítka "Add Note". Po vytvoření nové poznámky je zobrazena informační zpráva o přidání poznámky.
3. **Zobrazení seznamu poznámek:**
    - Seznam poznámek je zobrazen pomocí komponenty NotesListView.
    - Pokud jsou poznámky načítány, je zobrazena zpráva o načítání.
4. **Vizuální prvky:**
    - Vizuální prvky jsou implementovány pomocí komponent z knihovny NativeBase, jako jsou tlačítka, přepínače, ikony a pop-up okno pro nastavení třídění.

Jedná se o implementaci obrazovky pro správu poznámek s možností filtrování, třídění a vytváření nových poznámek. Využívá moderních principů Reactu a knihoven pro tvorbu uživatelsky příjemného a interaktivního uživatelského rozhraní.

Soubor PlanViewPage.tsx

Soubor obsahuje implementaci obrazovky pro zobrazení plánu prostoru, včetně možnosti výběru patra, místnosti a místa. Tato obrazovka také umožňuje vyhledávat dostupné inventáře a zobrazovat seznam položek v daném prostoru. Následuje souvislý text popisující klíčové prvky tohoto souboru:

Soubor PlanViewPage.tsx představuje komponentu pro zobrazení plánu prostoru a správu položek v daném patře, místnosti a místě. Tato komponenta využívá různé komponenty z knihovny NativeBase a React Navigation pro vytvoření responzivního uživatelského rozhraní.

1. **Načítání dat:**
    - Komponenta využívá různé stavy z Redux úložiště pro sledování stavu načítání dat, včetně patra, seznamu položek, inventářů a obrázku plánu.
    - Načítání probíhá při spuštění komponenty a při změně vybraného patra.
2. **Výběr pater, místností a míst:**
    - Uživatel může vybírat z dostupných pater, místností a míst pomocí rozbalovacích seznamů (Select). Výběr ovlivňuje zobrazení plánu a seznamu položek.
    - Po vybrání patra se zobrazí obrázek plánu patra a seznam místností na tomto patře.
3. **Zobrazení plánu:**
    - Komponenta obsahuje možnost zobrazení plánu prostoru na základě vybraného patra. Obrázek plánu se zobrazuje pomocí komponenty CastlePlanView.
    - Uživatel může vybrat místnost na plánu, což aktualizuje stav vybrané místnosti.
4. **Vyhledávání inventářů a zobrazení seznamu položek:**
    - Komponenta umožňuje uživateli vyhledávat dostupné inventáře v daném patře, místnosti a místě.
    - Seznam nalezených inventářů je zobrazen v komponentě ListView.
5. **Ovládání zobrazení:**
    - Uživatel může přepínat mezi zobrazením plánu a zobrazením seznamu položek pomocí tlačítek v dolní části obrazovky.
    - Při změně zobrazení se aktualizuje stav zobrazení a případně probíhá načítání dalších dat.
6. **Ovládání interakcí na plánku:**
    - Komponenta reaguje na doteky uživatele na plánové mapě, což umožňuje uživateli interagovat s plánem prostoru.
    - Stav interakce s plánem je sledován pomocí isInteractingWithCastlePlan.
7. **Chybové zprávy:**
    - Při chybách v načítání dat nebo interakcích s komponentou jsou zobrazovány chybové zprávy pomocí komponenty ErrorToast.
8. **Ovládací prvky pro vyhledávání a resetování:**
    - Komponenta obsahuje prvky pro vyhledávání, resetování formuláře a potvrzení vyhledávání.

Komponenta implementuje obrazovku pro zobrazení plánu prostoru a správu položek v tomto prostoru. Využívá moderní principy Reactu, knihoven pro tvorbu uživatelsky příjemného rozhraní a Redux pro správu stavu aplikace.

Soubor SearchPage.tsx

Soubor SearchPage.tsx představuje implementaci obrazovky pro vyhledávání, která umožňuje uživateli provádět různé dotazy na dostupné inventáře. Definuje komponentu pro obrazovku vyhledávání, která využívá komponenty knihovny NativeBase a vlastní komponenty pro zobrazení formuláře a výsledků vyhledávání. Klíčové prvky této obrazovky jsou následující:

1. **Načítání dat při spuštění:**
    - Při inicializaci komponenty jsou spuštěny asynchronní akce pro načtení různých dat potřebných pro formulář vyhledávání. Tyto akce zahrnují načítání inventářů, národností, umělců, předmětů, plánů, technik, zemí, měst a institucí.
2. **Stav načítání dat:**
    - Komponenta sleduje stav načítání dat prostřednictvím globálního stavu Redux. Stavy obsahují informace o inventářích, počtu výsledků, načítání dat a načtených stránkách.
3. **Vyhledávací formulář:**
    - V komponentě je zahrnut formulář pro vyhledávání (SearchForm), který může obsahovat různé vstupní prvky pro zadání parametrů vyhledávání. Formulář je propojen s funkcionalitou Redux pro načítání dat.
4. **Seznam výsledků vyhledávání:**
    - Komponenta obsahuje seznam výsledků vyhledávání, který je zobrazen pomocí komponenty ListView. Tento seznam zahrnuje různé inventáře a položky, které odpovídají zadaným kritériím vyhledávání.
5. **Načítání dalších položek:**
    - Když uživatel dosáhne konce seznamu, může načíst další položky. Tato funkcionalita je implementována pomocí Redux akce loadItemsByInventory, která je volána při stisknutí tlačítka pro načítání dalších položek.
6. **Debugging informace:**
    - Komponenta vkládá do konzole informace o průběhu pomocí console.log. Tato funkcionalita může být užitečná při sledování stavu komponenty v průběhu jejího životního cyklu.

Implementována je tak komponenta pro obrazovku vyhledávání, která zahrnuje formulář a seznam výsledků vyhledávání. Využívá Redux pro správu stavu aplikace a využívá komponenty knihovny NativeBase pro vytvoření responzivního uživatelského rozhraní.

Podadresář *stores*

Soubor store.ts

Soubor store.ts obsahuje definici Redux store, který slouží k centralizovanému uchovávání stavu aplikace. Níže je souvislý popis obsahu tohoto souboru:

Store.ts je klíčovým prvkem v implementaci správy stavu pomocí knihovny Redux. Začíná importem funkce configureStore z knihovny @reduxjs/toolkit. Následně jsou importovány jednotlivé reducery, které definují, jak bude zpracován stav aplikace v reakci na akce.

Celková konfigurace Redux store je provedena voláním funkce configureStore. V této konfiguraci jsou specifikovány jednotlivé reducery, které jsou následně kombinovány do jednoho kořenového reducera. Tyto reducery zahrnují:

1. userReducer: Spravuje stav týkající se uživatelských informací a autentifikace.

2. itemReducer: Zpracovává stav týkající se detailů jednotlivých položek.
3. planViewReducer: Spravuje stav pro zobrazení plánu nebo mapy.
4. noteViewState: Zpracovává stav pro zobrazení a správu poznámek.
5. searchFormReducer: Řídí stav týkající se formuláře pro vyhledávání.
6. listViewReducer: Spravuje stav související s náhledem seznamu položek.
7. homePageReducer: Zpracovává stav týkající se domovské obrazovky.

Celkově je Redux store vytvořen s tímto souborem tak, aby obsahoval informace z různých částí aplikace a umožňoval uživatelskému rozhraní a stavu komunikovat jednotným způsobem. Nakonec jsou v souboru definovány typy, které umožňují silné typování akcí a stavu v Redux store, což zvyšuje bezpečnost a přehlednost kódu.

Podadresář *actions*

Soubor homePageThunks.ts

Zdrojový soubor homePageThunks.ts obsahuje definici asynchronní thunk akce pomocí funkce createAsyncThunk z knihovny @reduxjs/toolkit. Tato akce se zaměřuje na získání dat pro domovskou stránku. Níže je souvislý popis obsahu tohoto souboru:

Soubor homePageThunks.ts definuje asynchronní thunk akci nazvanou fetchData za pomocí funkce createAsyncThunk. Tato akce je vytvořena za účelem získání dat pro domovskou stránku aplikace. Asynchronní charakter této akce umožňuje provádění asynchronních operací, jako je volání API, a správu stavu v Redux store.

Asynchronní thunk akce fetchData má název "homePage/fetchData" a je definována s pomocí anonymní asynchronní funkce, která přijímá dvě parametry – první parametr je aktuální hodnota akce, a druhým parametrem je thunkAPI, což je objekt poskytující přístup k různým funkcionalitám pro asynchronní akce.

V těle asynchronní funkce je proveden asynchronní požadavek na získání dat domovské stránky pomocí funkce fetchHomeDataRequest z modulu homePageService. Odpověď na tento požadavek je ověřena zkontrolováním statusu odpovědi. Pokud není status odpovědi 200 (OK), akce odmítne s hodnotou "Failed to fetch data" pomocí thunkAPI.rejectWithValue. V opačném případě jsou data z odpovědi extrahována a vrácena jako výsledek asynchronní akce typu HomePageData.

Soubor homePageThunks.ts chápat jako modul obsahující asynchronní thunk akci pro získání dat domovské stránky, která může být použita v Redux store pro správu stavu domovské stránky aplikace.

Soubor itemThunks.ts

Soubor itemThunks.ts obsahuje implementaci asynchronní thunk akce, konkrétně funkci getItem, a dvě akce pro zpracování poznámek k položce: setConcordances a getItemNotes. Níže je souvislý popis obsahu tohoto souboru:

Soubor itemThunks.ts definuje několik asynchronních thunk akcí pro práci s informacemi o položkách. První a hlavní akcí je getItem, která slouží k získání informací o konkrétní položce. Tato akce je vytvořena pomocí funkce createAsyncThunk z knihovny @reduxjs/toolkit. Funkce getItem přijímá jako parametr ID položky a provádí asynchronní požadavek na získání dat této položky pomocí funkce getItemRequest ze služby itemservice.

Pokud je odpověď na požadavek úspěšná a obsahuje více než jednu položku (což značí, že obsahuje obrázky), pak jsou data zpracována do strukturovaného formátu, který obsahuje informace o autorovi, instituci, repozitáři, provenienci, popisu a dalších vlastnostech položky. V případě, že odpověď obsahuje pouze jednu položku (bez obrázků), jsou získané informace omezovány pouze na základní údaje, jako je název díla a autor.

Dále je v souboru definována akce setConcordances, která slouží k nastavení shod v položkách. Tato akce není asynchronní a přijímá seznam shod jako parametr.

Poslední akcí v souboru je getItemNotes, která slouží k získání poznámek k položce. Tato akce také pracuje asynchronně a přijímá objekt s informacemi o položce a volbou týkající se získávání souvisejících komentářů. Odpověď na tuto akci obsahuje pole poznámek, kde každá poznámka může obsahovat odpovědi.

Jedná se o modul, který poskytuje akce pro získání, zpracování a správu informací o položkách v Redux store aplikace.

Soubor listViewThunks.ts

Soubor listViewThunks.ts obsahuje dvě asynchronní thunk akce pro práci se stavem seznamu položek v Redux store. Tyto akce jsou vytvořeny pomocí funkce createAsyncThunk z knihovny @reduxjs/toolkit. První akce je nazvána search, a druhá je nazvána loadItemsByInventory.

Akce search slouží k provádění asynchronního hledání položek na základě poskytnutých parametrů vyhledávání (searchParams). V rámci akce je volána funkce searchRequest ze služby searchService, která zajišťuje komunikaci s API pro vyhledávání. Pokud odpověď od serveru indikuje úspěch (status 200), data jsou vrácena jako SearchResponse. V opačném případě je vrácena chybová zpráva.

Druhá akce, loadItemsByInventory, je také asynchronní a slouží k načítání položek na základě určitého inventárního čísla. Akce využívá informace o stavu seznamu položek, zejména o načtených stránkách (loadedPages). Akce získává aktuální kurzor pro konkrétní inventář a provádí hledání položek pomocí funkce searchRequest. Stejně jako v předchozí akci, v případě úspěchu jsou vrácena data typu SearchResponse, v opačném případě je vrácena chybová zpráva.

Tyto *thunk* akce vytvářejí asynchronní operace pro správu stavu seznamu položek v Redux store, což umožňuje efektivní a bezproblémové zpracování asynchronních operací ve webové aplikaci.

Soubor notesThunks.ts

Soubor noteThunks.ts obsahuje čtyři asynchronní thunk akce, které umožňují interakci s poznámkami v aplikaci. Tyto akce jsou vytvořeny pomocí funkce createAsyncThunk z knihovny @reduxjs/toolkit. První akce je deleteNote, druhá je createNote, třetí je updateNote a čtvrtá je getAllNotes.

Akce deleteNote je zodpovědná za asynchronní odstranění poznámky na základě poskytnutého identifikátoru poznámky (noteId). Akce volá odpovídající funkci deleteNoteRequest ze služby notesservice pro komunikaci s API. Pokud je odpověď od serveru úspěšná (status 200), je vrácen objekt s identifikátorem smazané poznámky. V opačném případě je vrácena chybová zpráva.

Akce createNote je určena pro vytváření nových poznámek. Přijímá objekt newNote typu Note, který obsahuje informace o nové poznámce. Akce volá funkci createNoteRequest ze služby notesservice pro odeslání požadavku na vytvoření poznámky na serveru. Pokud je odpověď od serveru úspěšná (status 201), je vrácen objekt s vytvořenou poznámkou. V opačném případě je vrácena chybová zpráva.

Akce updateNote slouží k aktualizaci existující poznámky. Přijímá objekt note typu Note s aktualizovanými informacemi. Akce volá funkci updateNoteRequest ze služby notesservice pro odeslání požadavku na aktualizaci poznámky na serveru. Pokud je odpověď od serveru úspěšná (status 201), je vrácen objekt s aktualizovanou poznámkou. V opačném případě je vrácena chybová zpráva.

Poslední akce, getAllNotes, slouží k získání všech poznámek podle zadaných parametrů, jako jsou poznámky od uživatele (myComments), obecné poznámky (generalComments) a možnosti řazení (sortOptions). Akce volá funkci getAllNotesRequest ze služby notesservice pro získání seznamu poznámek. Pokud je odpověď od serveru úspěšná (status 200), jsou data zpracována a vrácena ve strukturované podobě. V opačném případě je vrácena chybová zpráva.

Soubor planThunk.ts

Zdrojový soubor planThunks.ts obsahuje několik asynchronních thunk akcí pro práci s plány a jejich komponentami. První akce je getFloorList, která slouží k získání seznamu pater a místností v každém patře. Druhá akce, getPlanInventories, získává seznam inventáře v konkrétní místnosti. Třetí akce, getPlanItems, získává položky spojené s konkrétním inventářem. Poslední akce, getPlanFloorImage, získává obrázek podlahy a jeho metadata.

První akce getFloorList získává data ze serveru o všech místnostech na každém patře. Data jsou následně zpracována a strukturována do pole objektů reprezentujících jednotlivá patra. Každé patro obsahuje svůj identifikátor (id), popisek (label) a seznam místností (roomList). Tyto údaje jsou dále upraveny pro přizpůsobení jejich pozic vzhledem k patru.

Druhá akce getPlanInventories získává seznam inventáře v dané místnosti. Odpověď ze serveru obsahuje inventáře a metadata o všech inventářích v místnosti.

Třetí akce getPlanItems získává položky spojené s konkrétním inventářem. Odpověď ze serveru obsahuje seznam položek a metadata o stránkách položek v inventáři.

Poslední akce getPlanFloorImage získává obrázek podlahy a jeho metadata. Odpověď ze serveru obsahuje SVG obrázek a metadata o zobrazení (viewBox). Toto metadata je získáváno pomocí parseString z knihovny react-native-xml2js. Tím je umožněno získání rozměrů SVG obrázku.

Všechny tyto akce využívají knihovnu @reduxjs/toolkit a jsou vytvořeny pomocí funkce createAsyncThunk. Akce také obsahují logiku pro zpracování odpovědí a případných chyb ze serveru.

Soubor searchFormThunks.ts

Zdrojový soubor searchFormThunks.ts obsahuje několik asynchronních thunk akcí, které slouží k načítání dat potřebných pro formulář vyhledávání. Tyto akce komunikují s příslušnými API koncovými body a zajistí, že data jsou k dispozici pro formulář vyhledávání v uživatelském rozhraní.

První akce, fetchInventories, získává seznam inventářů voláním funkce fetchInventoriesRequest. Odpověď ze serveru obsahuje informace o inventářích.

Druhá akce, fetchCountries, získává seznam zemí voláním funkce fetchCountriesRequest. Odpověď ze serveru obsahuje informace o zemích.

Třetí akce, fetchInstitutions, získává seznam institucí voláním funkce fetchInstitutionsRequest. Odpověď ze serveru obsahuje informace o institucích.

Čtvrtá akce, fetchArtists, získává seznam jmen umělců voláním funkce fetchArtistNamesRequest. Odpověď ze serveru obsahuje informace o umělcích.

Pátá akce, fetchTechniques, získává seznam technik voláním funkce fetchTechniquesRequest. Odpověď ze serveru obsahuje informace o technikách.

Šestá akce, fetchNationalities, získává seznam národností voláním funkce fetchNationalitiesRequest. Odpověď ze serveru obsahuje informace o národnostech.

Sedmá akce, fetchCities, získává seznam měst voláním funkce fetchCitiesRequest. Odpověď ze serveru obsahuje informace o městech.

Osmá akce, fetchSubjects, získává seznam předmětů voláním funkce fetchSubjectsRequest. Odpověď ze serveru obsahuje informace o předmětech.

Devátá akce, fetchPlans, získává seznam plánů voláním funkce fetchPlanRequest. Odpověď ze serveru obsahuje informace o plánech.

Všechny tyto akce využívají knihovnu @reduxjs/toolkit a jsou vytvořeny pomocí funkce createAsyncThunk. Každá akce obsahuje logiku pro zpracování odpovědí a případných chyb ze serveru.

Soubor userThunks.ts

Zdrojový soubor userThunk.ts obsahuje asynchronní thunk akce, které souvisejí s autentizací a řízením uživatelských relací v aplikaci.

První akce, login, zpracovává požadavek na přihlášení uživatele pomocí uživatelského jména a hesla. Akce volá funkci loginRequest ze souboru authservice.ts, která je zodpovědná za komunikaci se serverem ohledně přihlášení. V případě úspěchu vrátí informace o uživateli, včetně uživatelského jména a role. V opačném případě vrátí chybu s odpovědí ze serveru.

Druhá akce, checkAuth, slouží k ověření aktuální autentizace uživatele. Volá funkci isAuthRequest ze souboru authservice.ts, která vrací informaci o aktuálním stavu autentizace. Pokud je uživatel přihlášen, vrátí informaci o tom, že je přihlášen. V opačném případě vrátí chybu s odpovědí ze serveru.

Třetí akce, logout, zpracovává požadavek na odhlášení uživatele. Volá funkci logoutRequest ze souboru authservice.ts, která zajišťuje komunikaci se serverem ohledně odhlášení. Pokud odhlášení proběhne úspěšně, vrátí rezolvovaný slib (Promise). V opačném případě vrátí chybu s odpovědí ze serveru.

Veškeré z těchto akcí jsou vytvořeny pomocí knihovny @reduxjs/toolkit a využívají funkci createAsyncThunk. Zároveň obsahují logiku pro zpracování odpovědí a chyb ze serveru, včetně logování relevantních informací.

Podadresář *reducers*

Soubor homePageSlice.ts

Soubor homePageSlice.ts obsahuje implementaci správy stavu a akcí pro domovskou stránku (home page) v aplikaci. Tento soubor využívá nástroje createSlice z knihovny @reduxjs/toolkit k definování reduktoru a akcí.

Na začátku souboru jsou importovány potřebné závislosti, včetně datového typu HomePageData z jiné části aplikace a akce fetchData z modulu homePageThunks.

Dále soubor obsahuje definici rozhraní HomePageState, které popisuje stav domovské stránky. Tento stav obsahuje atributy, jako je lastError (poslední chyba), loading (indikátor načítání dat) a data (data domovské stránky).

Následuje inicializace počátečního stavu initialState pro domovskou stránku, kde jsou výchozí hodnoty pro lastError, loading, a data nastaveny na prázdné nebo false hodnoty.

Hlavní část souboru obsahuje definici reduktoru pomocí createSlice. Reduktor, pojmenovaný homePageSlice, je členěn na dvě hlavní části: reducers a extraReducers. V sekci reducers je definována jediná akce resetHomePage, která v případě vyvolání vrátí stav domovské stránky do počátečního stavu initialState.

V sekci extraReducers jsou definovány reakce na asynchronní akce spojené s načítáním dat na domovské stránce. Konkrétně to jsou akce spojené s asynchronním načítáním dat pomocí fetchData. Reduktor reaguje na tři stavy této asynchronní akce: pending (probíhá načítání), fulfilled (úspěšné načtení) a rejected (chyba při načítání). V každém stavu jsou aktualizovány příslušné části stavu podle výsledku asynchronní akce.

Nakonec jsou exportovány akce definované v rámci homePageSlice a samotný reduktor jako výchozí export souboru, což umožňuje integrovat tento stav do celkové správy stavu aplikace.

Soubor itemSlice.ts

ItemSlice.ts obsahuje implementaci správy stavu a akcí pro zobrazení detailu položky v aplikaci. Tento soubor také využívá nástroje createSlice z knihovny @reduxjs/toolkit k definování reduktoru a akcí.

Na začátku souboru jsou importovány potřebné závislosti, včetně datového typu ItemViewState a Item z jiné části aplikace a akce getItem a getItemNotes z modulu itemThunks. Dále je importován výčet Certainty z datového typu item, který definuje úrovně jistoty a barvy pro zvýraznění těchto úrovní.

Následuje definice objektu CertaintyWithColors, který mapuje hodnoty výčtu Certainty na objekty obsahující informace o jistotě a příslušné barvě pro vizuální reprezentaci v uživatelském rozhraní.

Poté je definován počáteční stav initialState pro zobrazení detailu položky. Tento stav obsahuje atributy, jako je item (informace o položce), concordances (souhlasnosti), notes (poznámky), lastError (poslední chyba), itemLoading (indikátor načítání informací o položce), a notesLoading (indikátor načítání poznámek).

Hlavní část souboru obsahuje definici reduktoru pomocí createSlice. Reduktor, pojmenovaný itemSlice, je rozdělen do dvou hlavních částí: reducers a extraReducers. V sekci reducers je definována jediná akce setConcordances, která slouží k nastavení souhlasností v rámci stavu.

Sekce extraReducers obsahuje reakce na asynchronní akce, tj. načítání informací o položce pomocí akce getItem a načítání poznámek pomocí akce getItemNotes. Reduktor reaguje na tři stavy každé z těchto asynchronních akcí: pending (probíhá načítání), fulfilled (úspěšné načtení) a rejected (chyba při načítání). V každém stavu jsou aktualizovány příslušné části stavu podle výsledku asynchronní akce.

Nakonec je exportován reduktor itemSlice jako výchozí export souboru, což umožňuje integrovat tento stav do celkové správy stavu aplikace.

Soubor listViewSlice.ts

ListViewSlice.ts obsahuje implementaci správy stavu a akcí pro zobrazení seznamu položek v aplikaci. Tento soubor využívá nástroje createSlice z knihovny @reduxjs/toolkit k definování reduktoru a akcí.

Na začátku souboru jsou importovány potřebné závislosti, včetně datových typů jako Inventory, ItemPreviewType a LoadedPagesOfInventories z jiných částí aplikace, a akcí loadItemsByInventory a search z modulu listViewThunks. Dále je importován datový typ SearchParams z modulu searchService.

Následuje definice rozhraní ListViewState, které popisuje stav pro zobrazení seznamu položek. Tento stav obsahuje atributy, jako jsou inventories (seznam inventářů), data (datový objekt, kde klíčem je název inventáře a hodnotou jsou seznamy položek), loading (indikátor načítání), lastError (poslední chyba), filterState (filtr pro vyhledávání), loadedPages (načtené stránky inventářů), inventoriesDataLoading (indikátor načítání dat inventářů) a numOfResults (celkový počet výsledků).

Dále je definován počáteční stav initialState pro tento reduktor. Tento stav obsahuje prázdné seznamy a objekty pro uvedené atributy.

Následuje definice reduktoru listViewSlice pomocí createSlice. Reduktor, pojmenovaný listViewSlice, je rozdělen do tří částí: reducers a extraReducers. V sekci reducers jsou definovány dvě akce: resetListView pro resetování stavu na počáteční hodnoty a setFilterState pro nastavení filtru.

Sekce extraReducers obsahuje reakce na asynchronní akce, tj. načítání seznamu položek pomocí akce search a načítání položek podle inventáře pomocí akce loadItemsByInventory. Reduktor reaguje na tři stavy každé z těchto asynchronních akcí: pending (probíhá načítání), fulfilled (úspěšné načtení) a rejected (chyba při načítání). V každém stavu jsou aktualizovány příslušné části stavu podle výsledku asynchronní akce.

Závěrem jsou exportovány akce resetListView a setFilterState spolu s reduktorem listViewSlice jako výchozí exporty souboru, což umožňuje integrovat tento stav do celkové správy stavu aplikace.

Soubor notesSlice.ts

Zdrojový soubor notesSlice.ts obsahuje implementaci správy stavu a akcí pro poznámky (notes) v aplikaci. Tento soubor využívá nástroje createSlice z knihovny @reduxjs/toolkit k definování reduktoru a akcí.

Na začátku souboru jsou importovány potřebné závislosti, včetně akcí (deleteNote, getAllNotes, createNote, updateNote) a typů (Certainty, ItemViewState, Item, Note, NotesViewState) z dalších částí aplikace.

Dále v souboru je definován objekt initialState pro správu stavu poznámek. Tento stav obsahuje položky jako notes (seznam poznámek), notesLoading (indikátor načítání poznámek), requestPending (indikátor probíhající akce – mazání, aktualizace nebo vytváření poznámky), triggerRefresh (spouštěč pro obnovení dat) a lastError (poslední chyba).

Následuje definice reduktoru pomocí createSlice. Reduktor reaguje na akce související s načítáním všech poznámek (getAllNotes), mazáním poznámky (deleteNote), vytvářením poznámky (createNote) a aktualizací poznámky (updateNote). Při úspěšném provedení těchto akcí jsou aktualizovány příslušné části stavu a indikátory. V případě chyby je zachycena chyba a aktualizován atribut lastError.

Celkově soubor notesSlice.ts slouží k efektivní správě stavu a akcí pro poznámky v aplikaci, přičemž poskytuje strukturovaný a snadno udržovatelný kód pro tuto funkcionalitu. Reduktor je nakonec exportován jako výchozí export pro integrování do celkového stavu aplikace.

Soubor planSlice.ts

Soubor planSlice.ts obsahuje implementaci správy stavu a asynchronních akcí pro zobrazení plánu, který zahrnuje seznam podlaží, položek, inventářů a příslušné obrázky. Pro tuto implementaci je využíván nástroj createSlice z knihovny @reduxjs/toolkit.

Na začátku souboru jsou importovány potřebné závislosti, včetně datového typu PlanViewState ze souboru plan.ts a akcí jako getFloorList, getPlanFloorImage, getPlanInventories a getPlanItems z modulu planThunks.

Následuje definice počátečního stavu initialState pro tento reduktor. Stav obsahuje atributy, jako je lastError (poslední chyba), floorList (seznam podlaží), floorListLoading (indikátor načítání seznamu podlaží), itemInventories (seznam inventářů), itemInventoriesLoading (indikátor načítání inventářů), itemList (seznam položek), itemListLoading (indikátor načítání položek), itemListPagination (informace o stránkování položek).

Dále následuje definice reduktoru planSlice pomocí createSlice. Reduktor, pojmenovaný planSlice, má dvě hlavní sekce: reducers a extraReducers. V sekci reducers by mohly být definovány akce pro manuální změnu stavu, ale v tomto případě jsou prázdné, protože vývojář neimplementoval žádné akce.

Sekce extraReducers obsahuje reakce na asynchronní akce. Každá reakce je přiřazena k určité asynchronní akci (getFloorList, getPlanItems, getPlanInventories, getPlanFloorImage). V každé reakci jsou specifikovány tři stavy: fulfilled (úspěšné dokončení akce), pending (probíhající akce) a rejected (chyba při akci).

Například, v případě getFloorList.fulfilled jsou aktualizovány atributy stavu floorList a floorListLoading podle výsledku asynchronní akce. Stejně tak jsou zpracovány další asynchronní akce (getPlanItems, getPlanInventories, getPlanFloorImage), kde se aktualizují různé části stavu podle výsledků těchto akcí.

Soubor implementuje správu stavu pro plán, reaguje na různé asynchronní akce a udržuje konzistentní stav aplikace v závislosti na provedených akcích. Vytváří tak robustní mechanismus pro zobrazování a aktualizaci dat v rámci plánu.

Soubor searchFormSlice.ts

Zdrojový soubor obsahuje implementaci správy stavu a asynchronních akcí pro formulář vyhledávání. Tento soubor je napsán v TypeScriptu a využívá nástroj createSlice z knihovny @reduxjs/toolkit. Jeho hlavním účelem je udržovat aktuální stav pro různé položky formuláře, jako jsou inventáře, místnosti, umělci, národnosti, techniky, instituce, země, města a předměty.

Na začátku souboru jsou importovány potřebné závislosti, včetně datových typů jako Artist, Inventory a Room ze souboru searchFormTypes, a akcí pro načítání dat pomocí thunks.

Definice stavu (SearchFormState) obsahuje pole pro každou položku formuláře a také atribut lastError, který slouží k uchování poslední chyby v případě neúspěšného načítání dat.

Dále následuje definice reduktoru (searchFormSlice) pomocí createSlice. Reduktor má tři sekce: initialState, reducers a extraReducers.

V sekci initialState jsou definovány počáteční hodnoty pro každý atribut stavu.

V sekci reducers jsou definovány akce pro manuální změnu stavu. Akce jako resetSearchForm, setInventories, setRooms, setArtists, setNationalities slouží ke změně hodnot ve stavu přímo z komponent.

V sekci extraReducers jsou definovány reakce na asynchronní akce. Každá reakce reaguje na úspěšné nebo neúspěšné dokončení asynchronní akce a aktualizuje stav podle výsledků této akce.

Například, reakce pro fetchInventories zpracovává úspěšné nebo neúspěšné načítání inventářů a aktualizuje stav podle výsledků. To samé platí i pro ostatní akce, jako jsou načítání institucí, národností, zemí, umělců, plánů, předmětů, měst a technik.

SearchFormSlice.ts poskytuje robustní mechanismus pro udržování stavu formuláře vyhledávání a reaguje na různé akce tak, aby byl stav aplikace konzistentní s aktuálními daty.

Soubor userSlice.ts

Soubor userSlice.ts obsahuje implementaci správy stavu a asynchronních akcí pro uživatele v rámci Redux stavu aplikace. Tento soubor je napsán v TypeScriptu a využívá nástroj createSlice z knihovny @reduxjs/toolkit. Jeho hlavním účelem je udržovat stav spojený s uživatelským účtem, jako je jméno uživatele, stav přihlášení, uživatelská role, poslední chyba a informace o ověření autentizace.

Na začátku souboru jsou importovány potřebné závislosti, zejména akce pro asynchronní operace jako checkAuth, login a logout z userThunks.

Definice stavu (UserState) obsahuje atributy pro uživatelské jméno, stav přihlášení, uživatelskou roli, poslední chybu a informaci o ověření autentizace.

Dále následuje definice reduktoru (userSlice) pomocí createSlice. Reduktor má tři sekce: initialState, reducers a extraReducers.

V sekci initialState jsou definovány počáteční hodnoty pro každý atribut stavu.

V sekci reducers je definována jedna akce resetState, která resetuje stav na počáteční hodnoty. Tato akce může být použita například při odhlášení uživatele.

V sekci extraReducers jsou definovány reakce na asynchronní akce. Každá reakce reaguje na úspěšné nebo neúspěšné dokončení asynchronní akce a aktualizuje stav podle výsledků této akce.

Například, reakce pro login zpracovává úspěšné nebo neúspěšné přihlášení uživatele a aktualizuje stav podle výsledků. To samé platí pro akce checkAuth a logout.

Celkově soubor userSlice.ts poskytuje mechanismus pro udržování stavu uživatele v rámci aplikace Redux, s důrazem na bezpečné a spolehlivé zpracování asynchronních operací, jako jsou přihlášení, odhlášení a ověření autentizace.

Podadresář *theme*

Soubor nativeBaseTheme.ts

NativeBaseTheme.ts definuje téma pro uživatelské rozhraní pomocí knihovny native-base, což je knihovna pro React Native. Toto téma je vytvořeno pomocí funkce extendTheme poskytované touto knihovnou. Jeho hlavním účelem je definovat barvy pro různé části uživatelského rozhraní.

Celý soubor je exportován jako konstanta nativeBaseTheme. Toto téma obsahuje definice barev v různých odstínech pro hlavní barvu a také nastavuje barvu pozadí pro tlačítka.

Hlavní část tématu je uvnitř objektu colors, kde je definována hlavní barva pod klíčem primary. Tato barva má různé odstíny označené čísly od 50 do 900. Každé číslo představuje jiný odstín této barvy, přičemž nižší čísla obvykle znamenají světlejší odstíny a vyšší čísla tmavší odstíny. Tento postup umožňuje vývojářům flexibilně používat různé odstíny této barvy v jejich aplikaci podle potřeby.

Dále je definována specifická barva pro pozadí tlačítek pod klíčem buttonBackground. Tato barva je nastavena na odstín primary.500, což znamená, že používá odstín s číslem 500 z definované primární barvy. Tímto způsobem se zajistí, že pozadí tlačítek bude mít konkrétní vizuální vzhled definovaný v rámci tématu.

Slouží k definování konkrétních barev a vzhledu pro komponenty vytvářené pomocí knihovny native-base v rámci aplikace napsané v React Native. Tímto způsobem zjednodušuje a centralizuje správu vizuálního stylu v rámci aplikace.

Podadresář *types*

Soubor general.ts

Soubor obsahuje export definice výčtového typu (*enum*) 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 homePageTypes.ts

Zdrojový soubor homePageTypes.ts definuje typovou strukturu pro data na domovské stránce (HomePage). Tento soubor obsahuje jediný typ nazvaný HomePageData. Tento typ popisuje strukturu dat, která mohou být použita na domovské stránce aplikace.

HomePageData je typ objektu, který je definován jako pole objektů. Každý objekt v poli reprezentuje určitou část informací na domovské stránce. Klíče těchto objektů jsou řetězce (stringy), které mohou být použity jako identifikátory jednotlivých částí dat. Hodnota každého klíče je objekt obsahující dvě položky:

1. **Label (označení):** Toto je volitelná položka typu řetězec, která slouží k poskytnutí názvu nebo popisu dané části dat.

2. **Text (text):** Toto je povinná položka a představuje samotný textový obsah dané části dat. Text může obsahovat libovolné znaky a být využíván k zobrazení informací na domovské stránce.

Tento soubor definuje flexibilní typ datové struktury, která může být použita pro organizaci a reprezentaci různých textových informací na domovské stránce aplikace. To umožňuje vývojářům snadno rozšiřovat a upravovat obsah domovské stránky bez potřeby měnit samotný kód zpracovávající tyto informace.

Soubor item.ts

Zdrojový soubor items.ts definuje několik TypeScript typů, které slouží k reprezentaci a správě informací o položkách (Items) v aplikaci. Tato soubor obsahuje tři hlavní typy: Item, ItemViewState a Concordance, a také enum Certainty.

1. **Item:**
    - Item je hlavním typem reprezentujícím jednotlivé položky. Každá položka může mít různé vlastnosti, jako je ID, informace o autorovi, název práce, inventární číslo, seznam hledaných témat, zda je k dispozici celkový pohled (fullView), obrázky, instituce a další.
    - Některé vlastnosti jsou volitelné (např. ID) a některé jsou povinné (např. fullView).
    - Obrázky jsou reprezentovány polem objektů, kde každý objekt obsahuje URL obrázku a jeho popisek.
2. **ItemViewState:**
    - ItemViewState reprezentuje celkový stav jedné položky, včetně informací o samotné položce, načítání dat, seznamu konkordancí, poznámek a stavu načítání poznámek.
3. **Concordance:**
    - Concordance reprezentuje konkrétní konkordanci, což je spojení nebo shoda mezi různými položkami. Má vlastnosti, jako je ID a jistota (cert), která je typu enum Certainty.
4. **Certainty:**
    - Certainty je výčtový typ (enum), který obsahuje hodnoty pro různé jistoty spojené s konkrétními konkordancemi, jako je "same_as", "high", "medium", "low" a "unknown".

Takto je poskytován abstraktní a typově bezpečný způsob reprezentace a práce s informacemi o položkách v rámci aplikace. To zjednodušuje práci s těmito daty a minimalizuje chyby v kódu díky striktní typové kontrole TypeScriptu.

Soubor listViewTypes.ts

Zdrojový soubor listViewTypes.ts obsahuje definice TypeScript typů, které slouží k popisu dat v kontextu vyhledávání a náhledů položek v rámci aplikace. Tyto definice zahrnují odpovědi na hledání, informace o stránkování, náhledy položek a související údaje.

Prvním typem je **SearchResponse**, který reprezentuje odpověď na vyhledávání. Obsahuje informace o stránkování a seznam náhledů položek. Dále, **Pagination** definuje stránkování s položkami jako kurzor, počet záznamů, počet položek a seznam inventářů.

Dalším klíčovým prvkem je **ItemPreviewType**, který popisuje náhled položky. Obsahuje různé atributy, jako jsou XML ID, externí ID pro ikonu třídy, text, název, objekt Item, inventář a zdroj ItemSource.

Item je komplexnější typ, který může obsahovat pole s různými vlastnostmi, včetně popisu, typu, jména osoby a dalších informací. Typ **PersonName** reprezentuje jméno osoby a související údaje, jako je role a externí ID od Getty, a **GettyData** obsahuje informace od Getty, jako je zobrazené jméno.

Další komponentou jsou **ItemImage** a **ItemSource**, které popisují obrázky položky a informace o zdroji. Nakonec, **LoadedPagesOfInventories** uchovává informace o načtených stránkách inventářů v rámci aplikace, včetně kurzoru, počtu položek a záznamů pro každý inventář.

Poskytuje strukturované a typově bezpečné definice pro manipulaci s daty souvisejícími s vyhledáváním a náhledy položek v aplikaci, což usnadňuje práci s nimi v rámci TypeScriptu.

Soubor note.ts

Zdrojový soubor note.ts obsahuje definice pro práci s poznámkami (notes) v rámci aplikace. Základním stavebním kamenem je rozhraní Note, které popisuje strukturu jedné poznámky. Každá poznámka má několik klíčových vlastností:

- uuid: Unikátní identifikátor poznámky.
- username: Uživatelské jméno osoby, která poznámku vytvořila.
- userId: Unikátní identifikátor uživatele, který vytvořil poznámku.
- note: Textový obsah poznámky.
- avatarUrl: URL adresa obrázku avataru uživatele.
- items: Pole obsahující identifikátory položek (items), ke kterým je poznámka připojena.
- replies: Pole obsahující odpovědi na tuto poznámku, přičemž každá odpověď je opět typu Note.
- createdTime: Datum a čas vytvoření poznámky.
- updatedTime: Datum a čas poslední aktualizace poznámky.
- noteColor: Barva přiřazená poznámce.
- reply_to: Identifikátor rodičovské poznámky, na kterou je tato poznámka odpovědí.

Dále soubor definuje typ NotesViewState, který popisuje stav související s poznámkami na úrovni uživatelského rozhraní. Tento typ obsahuje:

- notes: Pole poznámek.
- notesLoading: Indikátor načítání poznámek.
- requestPending: Indikátor probíhající požadavku (například načítání, aktualizace apod.).
- triggerRefresh: Číslo, které může být použito jako spouštěč pro obnovení obsahu.
- lastError: Textový popis poslední chyby nebo problému.

Note.ts poskytuje strukturované definice pro práci s poznámkami v rámci aplikace, což usnadňuje správu dat a manipulaci s nimi.

Soubor plan.ts

Soubor plan.ts obsahuje klíčové definice pro práci s plánem budovy a souvisejícími informacemi. První z definic je Floor, která reprezentuje jednotlivá patra budovy. Každé patro má svůj unikátní identifikátor (id), popis (label) a seznam místností (roomList).

Dále je zde definice Room, která obsahuje informace o konkrétní místnosti. Každá místnost má svůj identifikátor (id), souřadnice v rámci plánu (kx, ky), úroveň (z), příslušnost k patru (floor), popisné označení (label), grafickou cestu ve formátu SVG (svg_path), informace o existenci seznamu místností (room_list) a seznam umístěných míst (places).

Place reprezentuje konkrétní umístění v místnosti. Má identifikátor (id) a popisné označení (label).

Další definicí je PlanImage, která obsahuje informace o obrázku plánu včetně textové reprezentace SVG (svg) a informací o zobrazované oblasti (viewBox).

Celkový stav plánu a jeho zobrazení je reprezentován definicí PlanViewState. Tato struktura obsahuje informace o načítání seznamu pater (floorListLoading), samotný seznam pater (floorList), informace o inventáři (itemInventories), načítání inventáře (itemInventoriesLoading), celkový počet položek (totalItemsCount), stránkování seznamu položek (itemListPagination), načítání položek (itemListLoading), samotný seznam položek (itemList), informace o obrázku plánu (mapImage), načítání obrázku plánu (mapImageLoading) a textový popis poslední chyby nebo problému (lastError).

Tyto definice jsou klíčové pro práci s informacemi o budově, umožňují strukturovaný způsob uchovávání a správu dat týkajících se plánu a jeho jednotlivých prvků.

Soubor searchFormTypes.ts

Zdrojový soubor searchFormTypes.ts obsahuje klíčové definice typů, které jsou využívány pro strukturování dat týkajících se vyhledávání a formuláře. Prvním definovaným typem je Inventory, který reprezentuje informace o inventáři. Každý inventář může obsahovat informace jako počet položek (count), název (name), popisek (label) a pořadí (order).

Následuje definice typu Room, který reprezentuje informace o místnosti. Každá místnost má unikátní identifikátor (id), indikátor, zda je součástí plánu (in_plan), popisek (label) a seznam umístěných míst (places).

RoomPlace je definice typu pro reprezentaci konkrétního umístění v místnosti. Má identifikátor (id) a popisek (label).

Poslední definovaný typ je Artist, který obsahuje informace o umělci. Každý umělec je charakterizován zobrazovaným jménem (display_name) a identifikátorem v databázi Getty (getty_id).

Definice typů jsou klíčové pro uchování a manipulaci s daty, která se týkají inventářů, místností a informací o umělcích. Poskytují strukturovaný způsob práce s těmito daty v rámci aplikace, což usnadňuje přehlednost a správu informací v průběhu vývoje a údržby systému.

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 důkladně rozebrán. Přehledná struktura zajistí, že při případném doplňování rozšíření, či údržbě, je možné aplikaci snadno modifikovat. Aplikace splňuje zadání (akceptační kritéria) zákazníka.