**Docker-compose.yml:**

Ez a fájl egy Docker Compose konfigurációs fájl, amely segítségével több szolgáltatást indíthatunk el egyszerre konténerekben. A fájl a version: '3' szintaxist használja, amely a Docker Compose harmadik verziójának megfelelő formátum.

Az egyes szakaszok a következőket határozzák meg:

1. Kommentben lévő frontend és backend szolgáltatások:
   * A fájlban ki vannak kommentelve, így jelenleg nem futtatódnak. Azonban, ha aktiválnánk őket, ezek a szolgáltatások külön konténerekben indítanák el az alkalmazás frontendjét és backendjét.
   * A frontend a 3000-es porton, a backend pedig az 5000-es porton érhető el. A backend kapcsolatot létesít egy db nevű adatbázissal.
2. keycloak szolgáltatás:
   * Ez a szolgáltatás a Keycloak szervert futtatja, amely felhasználói hitelesítéshez és jogosultságkezeléshez használható.
   * A Keycloak adminisztrátor felhasználóneve és jelszava az admin és admin értékre van állítva, amit az environment rész definiál.
   * A Keycloak szerver a 8080-as porton lesz elérhető.
3. db szolgáltatás:
   * Ez egy PostgreSQL adatbázis-szolgáltatást futtat.
   * A POSTGRES\_USER, POSTGRES\_PASSWORD, és POSTGRES\_DB változók segítségével létrejön egy alapértelmezett adatbázis (loposg), egy felhasználó (loposg) és egy jelszó (loposg123).
   * Az adatbázis a 5432-es porton lesz elérhető.
4. Hálózatok (networks):
   * Egy app-network nevű hálózat van definiálva, amely lehetővé teszi a különböző szolgáltatások közötti kommunikációt.

Ez a fájl lehetőséget ad a különböző szolgáltatások egyszerű egyidejű indítására és kezelésére Docker konténerekben.

**auth/Dockerfile:**

Ez a fájl egy Dockerfile, amely a Keycloak hitelesítési és jogosultságkezelési rendszer beállításait tartalmazza. A fájl a következőket definiálja:

1. Alap image (FROM quay.io/keycloak/keycloak:latest):
   * A Docker image alapját a quay.io repozitórium legfrissebb Keycloak verziója képezi. A Keycloak egy nyílt forráskódú identitás- és hozzáférés-kezelő eszköz.
2. Környezeti változók beállítása:
   * ENV KEYCLOAK\_ADMIN=admin: Beállítja az adminisztrátor felhasználónevet admin-ra.
   * ENV KEYCLOAK\_ADMIN\_PASSWORD=admin: Beállítja az adminisztrátor jelszavát admin-ra.
   * Ezeket a változókat a Keycloak indításakor használja az adminisztrációs felülethez való hozzáférés biztosítására.
3. Indítási parancs (ENTRYPOINT):
   * ENTRYPOINT ["/opt/keycloak/bin/kc.sh", "start-dev"]: Meghatározza, hogy a Keycloak szerver start-dev módban induljon el. Ez egy fejlesztési üzemmód, amely lehetővé teszi a gyorsabb beállítást és tesztelést.

Helye az auth mappában:

* Mivel ez a fájl az auth mappában található, valószínűleg az alkalmazás többi komponense számára biztosít hitelesítési szolgáltatásokat.

**Backend/.gitignore:**

Ez a .gitignore fájl a backend mappában található, és a Git verziókezelő számára jelzi, hogy ne vegye figyelembe a \_\_pycache\_\_/ mappát.

A \_\_pycache\_\_/ mappa a Python által létrehozott gyorsítótárat (cache-t) tartalmazza, ahol a futtatott Python fájlok fordított verziói találhatók. Ezek a fájlok gyorsítják a Python programok futását, de nem szükségesek a kód működéséhez vagy a verziókezeléshez, ezért gyakran kihagyják őket a Git tárolókból.

**Backend/Dockerfile:**

Ez a Dockerfile a backend mappában található, és egy Python alapú backend alkalmazás Docker konténerben való futtatására szolgál. Az egyes parancsok a következőket csinálják:

1. Python alap image (FROM python:3.9):
   * Ez a sor egy Docker image-ből alapozza meg a környezetet, amely a Python 3.9-es verzióját tartalmazza. Ez biztosítja, hogy a Python környezet telepítve legyen a konténerben.
2. Mappa létrehozása a kódfájlok számára (WORKDIR /app):
   * A WORKDIR utasítás létrehozza és beállítja az /app mappát a konténeren belül, ahol az alkalmazás kódfájljai elhelyezkednek. Ezt a mappát fogja használni a konténer a következő lépésekhez.
3. Függőségek telepítése (COPY requirements.txt . és RUN pip install -r requirements.txt):
   * Az első parancs (COPY requirements.txt .) bemásolja a requirements.txt fájlt a konténer /app mappájába.
   * A második parancs (RUN pip install -r requirements.txt) telepíti a Python projekt függőségeit, amelyeket a requirements.txt fájl tartalmaz.
4. Backend indítása Quart szerverrel (CMD ["quart", "run", "--host=0.0.0.0", "--port=5000", "--reload"]):
   * Ez a parancs indítja el a backend alkalmazást a quart szerver segítségével.
   * A quart egy Python web framework, amely aszinkron működést támogat.
   * A --host=0.0.0.0 paraméter lehetővé teszi, hogy a szerver minden IP-címről elérhető legyen.
   * A --port=5000 beállítja a szerver portját, hogy az a 5000-es porton fusson.
   * A --reload bekapcsolja a rejtett újraindítást fejlesztési módban, tehát ha módosítasz a kódban, a szerver automatikusan újraindul.

Ez a Dockerfile tehát egy Python 3.9 alapú környezetet hoz létre, telepíti a szükséges függőségeket, és elindítja a backend alkalmazást Quart használatával.

**Backend/requirements.txt:**

A requirements.txt fájl a backend mappában található, és azokat a Python csomagokat tartalmazza, amelyek szükségesek az alkalmazás futtatásához. Az alábbi csomagok szerepelnek benne:

1. **quart**:
   * A Quart egy aszinkron web framework, amely a Flask-ra épít, de támogatja az aszinkron működést (például async/await), ami lehetővé teszi a nagyobb teljesítményt és skálázhatóságot, különösen az I/O műveletek során. Az alkalmazás a Quart segítségével működik.
2. **asyncpg**:
   * Az asyncpg egy aszinkron PostgreSQL adatbázis könyvtár. Az alkalmazás valószínűleg egy PostgreSQL adatbázissal dolgozik, és az asyncpg segíti az aszinkron adatbázis kapcsolatok kezelését, amely javítja az I/O műveletek hatékonyságát.
3. **python-keycloak-async**:
   * Ez egy Python könyvtár, amely lehetővé teszi a Keycloak, egy nyílt forráskódú identitás- és hozzáférés-kezelő rendszer aszinkron használatát. A python-keycloak-async könyvtár segítségével a backend alkalmazás képes kapcsolódni a Keycloak rendszerhez, például hitelesítéshez és jogosultságok kezeléséhez.
4. **quart-cors**:
   * A quart-cors egy bővítmény a Quart-hoz, amely lehetővé teszi a CORS (Cross-Origin Resource Sharing) konfigurálását. A CORS beállítása biztosítja, hogy a backend szerver képes legyen kezelni a különböző domain-ekről érkező kéréseket, ami különösen fontos, ha az alkalmazás frontendje más domain-en fut.

Összefoglalva, a requirements.txt fájl az alábbi Python könyvtárakat tartalmazza, amelyek biztosítják az aszinkron működést, a PostgreSQL adatbázis kezelését, a Keycloak hitelesítést és a CORS támogatást a Quart alapú backend alkalmazás számára.

**Backend/app.py:**

A app.py fájl egy Quart alapú backend alkalmazást tartalmaz, amely hitelesítési funkciókat és szobák kezelését valósítja meg a Keycloak rendszeren keresztül. Az egyes részek a következő funkciókat látják el:

**1. Importok:**

* A szükséges könyvtárak importálása, például:
  + Quart: Web szerver és aszinkron framework.
  + keycloak: A Keycloak hitelesítési könyvtár, amely segítségével kommunikálhatunk a Keycloak szerverrel.
  + quart\_cors: A CORS támogatás beállítására.
  + random: Véletlenszám-generálásra, ami a szoba ID-k generálásához szükséges.

**2. Keycloak beállítása:**

* A KeycloakOpenID példány létrehozása, amely kapcsolódik a Keycloak szerverhez. Az openid változó segítségével az alkalmazás képes a Keycloak OpenID Connect implementációját használni a felhasználók hitelesítésére.
* Paraméterek:
  + server\_url: A Keycloak szerver URL-je.
  + client\_id: Az alkalmazás azonosítója a Keycloak rendszerében.
  + realm\_name: A használt realm neve.
  + client\_secret\_key: Az alkalmazás titkos kulcsa.

**3. CORS beállítás:**

* Az alkalmazás CORS (Cross-Origin Resource Sharing) engedélyezése, hogy a frontend (más domain-en lévő alkalmazás) elérhesse az API-t.

**4. Aszinkron függvények:**

* **check\_user\_loggedin**: Ellenőrzi, hogy a felhasználó be van-e jelentkezve a Keycloak rendszerébe, a hozzáférési token alapján. Ha a felhasználó nincs bejelentkezve vagy érvénytelen a token, akkor None-t ad vissza.
* **generate\_room\_id**: Generál egy egyedi szoba ID-t, amely egy 6 számjegyű véletlenszerű szám lesz. A generálás addig folytatódik, amíg nem találunk egy egyedi ID-t.

**5. API végpontok:**

* **/create\_room (POST)**:
  + Ellenőrzi, hogy a felhasználó be van-e jelentkezve.
  + Ha a felhasználó nincs bejelentkezve, akkor hibaüzenetet küld.
  + Ha túl sok szoba van már létrehozva (a limitet elérve), akkor hibaüzenetet küld.
  + Ha a fenti feltételek nem teljesülnek, akkor új szoba ID-t generál, és hozzáadja a szobákhoz.
* **/login (GET)**:
  + Előállítja a Keycloak hitelesítési URL-t, amelyre a felhasználó átirányítható a bejelentkezéshez.
  + Az auth\_url a frontend számára szükséges, hogy elindítsa a bejelentkezési folyamatot.
* **/callback (POST)**:
  + Ez a végpont kezeli a Keycloak által visszaküldött kódot.
  + A kódot használva az alkalmazás megszerzi a hozzáférési tokent, amely tartalmazza a felhasználó hitelesítési információit.
* **/logout (POST)**:
  + A felhasználó kijelentkezését kezeli a Keycloak rendszerből a token\_refresh segítségével.
  + A kijelentkezés sikerességét egy státusz üzenetben jelzi.

**6. Futtatás:**

* A run() függvény indítja el az alkalmazást.
* Ha a fájl közvetlenül futtatásra kerül, akkor az if \_\_name\_\_ == "\_\_main\_\_": blokkban meghívja a run() függvényt, és elindítja a Quart szervert.

**Összefoglalás:**

Ez az alkalmazás egy backend szolgáltatást biztosít, amely lehetővé teszi a felhasználók hitelesítését a Keycloak rendszerén keresztül. Az alkalmazás lehetővé teszi szobák létrehozását, hitelesítési URL generálását, valamint a felhasználó bejelentkezését és kijelentkezését. Az API aszinkron működik, és a CORS is engedélyezve van a különböző domain-eken lévő frontend alkalmazások számára.

**frontend/index.html:**

Az általad megosztott index.html fájl a frontend mappában található, és a következőket tartalmazza:

HTML fájl felépítése:

1. <!DOCTYPE html>:
   * Ez a sor jelzi, hogy a dokumentum HTML5-ös típusú.
2. <html lang="">:
   * A <html> elem jelzi, hogy HTML dokumentumot kezdünk. A lang attribútum üres, de itt célszerű lenne megadni az oldal nyelvét, például: <html lang="en"> angol nyelvű oldal esetén.
3. <head>:
   * A <head> szakasz tartalmazza a dokumentum meta-információit és egyéb beállításokat.
   * <meta charset="UTF-8">:
     + Ez a sor beállítja a karakterkódolást UTF-8-ra, ami a legtöbb modern weboldalon használt kódolás.
   * <link rel="icon" href="/favicon.ico">:
     + Ez a sor beállítja a weboldal ikonját (favicon), amely a böngésző fülén jelenik meg. A fájl az alapértelmezett /favicon.ico útvonalon kereshető.
   * <meta name="viewport" content="width=device-width, initial-scale=1.0">:
     + A meta viewport beállítás segít optimalizálni az oldal megjelenését mobil eszközökön, így az oldal szélessége megegyezik az eszköz szélességével, és a kezdeti nagyítás 1.
   * <title>Vite App</title>:
     + A <title> elem határozza meg a böngésző fülén megjelenő címet, jelen esetben "Vite App" lesz.
4. <body>:
   * A <body> elem tartalmazza az oldalon megjelenő tényleges tartalmat.
   * <div id="app"></div>:
     + Ez egy üres div elem, amely a frontend alkalmazás root konténereként szolgál. Itt kerül majd megjelenítésre az alkalmazás dinamikus tartalma. A JavaScript kód itt "mountolja" a frontend alkalmazást.
   * <script type="module" src="/src/main.js"></script>:
     + Ez a sor betölti és futtatja a /src/main.js fájlt, amely valószínűleg a frontend alkalmazás belépési pontja, és innen indítódik a JavaScript alkalmazás, például egy React vagy Vue.js alkalmazás.

Összefoglaló:

Ez egy alap HTML fájl, amely a frontend alkalmazás számára van beállítva, és azt várja, hogy a dinamikus tartalom a <div id="app"></div> elembe legyen renderelve. A JavaScript alkalmazás (main.js) futtatásával kezdődik a frontend működése. A Vite, amely valószínűleg a build és fejlesztési környezet, biztosítja a gyors újratöltést és moduláris felépítést az alkalmazás számára.

**frontend/Dockerfile:**

Ez a Dockerfile a frontend mappában található, és a következő lépéseket hajtja végre a frontend alkalmazás konténerizálására:

1. Alap image kiválasztása: A Dockerfile egy Node.js alapú képet használ (node:18), amely a Node.js 18-as verzióját tartalmazza. Ez biztosítja, hogy a konténer tartalmazza az összes szükséges futtatási környezetet a Node.js alkalmazásokhoz.
2. Munkakönyvtár létrehozása: A Docker a konténerben egy /app nevű mappát hoz létre, amely a kódfájlok tárolására szolgál. Ez lesz a munkaállomás, ahonnan az alkalmazás futtatása történik.
3. Függőségek telepítése: A package.json fájlokat bemásolják a konténerbe, és a szükséges csomagokat telepítik az npm install parancs segítségével. Ez biztosítja, hogy a szükséges könyvtárak és eszközök telepítve legyenek a környezetbe.
4. Alkalmazás kódjának másolása: A teljes kódbázist átmásolják a konténerbe. Ezzel biztosítva van, hogy az alkalmazás összes fájlja elérhető legyen a futtatáshoz.
5. Alkalmazás futtatása: A Docker konténer a npm run dev parancsot futtatja, ami elindítja a fejlesztői szervert. Ez általában azt jelenti, hogy a konténer elindítja az alkalmazást egy helyi szerveren, amely a fejlesztéshez szükséges környezetet biztosít, például az automatikus újratöltést, amikor a kódot módosítják.

Ez a Dockerfile tehát egy fejlesztői környezetet készít elő a frontend alkalmazás számára, amely lehetővé teszi a könnyű futtatást és tesztelést konténerben.

**frontend/.gitignore:**

A **.gitignore** fájl a frontend mappában található, és meghatározza azokat a fájlokat és mappákat, amelyeket a Git nem fog verziókezelésbe venni. Ez azt jelenti, hogy ezek a fájlok nem kerülnek bele a verziókövetett repóba, hogy elkerüljük a fölösleges vagy érzékeny adatokat.

A fájl tartalmának jelentése a következő:

1. **Log fájlok**:
   * logs, \*.log, npm-debug.log\*, yarn-debug.log\*, pnpm-debug.log\*, lerna-debug.log\*: Ezek a log fájlok, amelyeket a különböző csomagkezelők és eszközök hozhatnak létre. Ezek a fájlok nem szükségesek a verziókezeléshez.
2. **Csomagok és függőségek**:
   * node\_modules: A Node.js függőségek tárolására szolgáló mappa. Nem szükséges verziókezelni, mivel a package.json és a package-lock.json elegendő információt ad a telepítési folyamathoz.
3. **Platform specifikus fájlok**:
   * .DS\_Store: A macOS operációs rendszer által automatikusan létrehozott fájlok, amelyek a mappa nézetének beállításait tartalmazzák. Nem szükségesek a verziókezeléshez.
4. **Fordítási és build fájlok**:
   * dist, dist-ssr: Általában a build folyamat során létrejövő kimeneti mappák. Ezeket a fájlokat nem szükséges verziókezelni, mivel újra generálhatók.
   * coverage: A tesztelés során generált lefedettségi jelentések.
   * \*.local: Lokális környezetekhez tartozó fájlok, amelyek nem fontosak a verziókezeléshez.
5. **Cypress teszteléshez kapcsolódó fájlok**:
   * /cypress/videos/ és /cypress/screenshots/: A Cypress tesztkeretrendszer által készített videók és képernyőképek, amelyek tesztfutások során keletkeznek. Ezeket nem szükséges verziókezelni.
6. **Szerkesztői beállítások**:
   * .vscode/\*: A Visual Studio Code IDE által létrehozott fájlok és beállítások.
     + A !.vscode/extensions.json sor kivételt képez, tehát a .vscode/extensions.json fájl verziókezelésbe kerül, amely a projekthez telepített kiterjesztéseket tartalmazza.
   * .idea: A JetBrains IDE-k (pl. IntelliJ IDEA) által létrehozott fájlok és beállítások.
   * \*.suo, \*.ntvs\*, \*.njsproj, \*.sln: A Visual Studio és más Microsoft fejlesztői eszközök által létrehozott fájlok.
   * \*.sw?: Az IDE-k által használt ideiglenes fájlok.
7. **Fordítási információk**:
   * \*.tsbuildinfo: A TypeScript build információkat tartalmazó fájlok, amelyek nem szükségesek a verziókezeléshez.

Összességében ez a .gitignore fájl biztosítja, hogy a fejlesztési környezetből és a build folyamatokból származó ideiglenes fájlok, függőségek és egyéb nem fontos fájlok ne kerüljenek a verziókezelés alá, így tisztábbá és kezelhetőbbé téve a Git repót.

**frontend/jsconfig.json:**

A **jsconfig.json** fájl a frontend mappában található, és a JavaScript projekt beállításait tartalmazza, különösen a kód szervezésére és a modulok kezelésére vonatkozó információkat. Itt a fájlban található beállítások jelentése a következő:

1. **compilerOptions**:
   * **paths**: A paths kulcs segítségével aliasokat hozhatunk létre a fájlrendszerben található mappákhoz.
     + @/\*: Ez egy alias, ami azt jelenti, hogy a @/-ot a projektben bárhol használva a ./src/\* mappára hivatkozik. Például, ha a @/components/Button kifejezést használjuk, az a ./src/components/Button fájlt jelenti. Ez lehetővé teszi a tisztább és könnyebben kezelhető fájlimportokat a kódban, mivel nem kell relatív vagy hosszú elérési utakat használni.
2. **exclude**:
   * A exclude kulcsban megadott mappák és fájlok nem kerülnek figyelembevételre a projektben.
     + **node\_modules**: A node\_modules mappa, amely az összes projekt függőséget tartalmazza, nem kerül bele a projekt fordításába vagy a fájlkövetésbe.
     + **dist**: A dist mappa, amely általában a build folyamat során keletkező kimeneti fájlokat tartalmazza, szintén nem kerül figyelembevételre.

Összességében a **jsconfig.json** fájl azt biztosítja, hogy a JavaScript projektben könnyen kezelhetőek legyenek az importálási utak, és hogy a nem szükséges fájlok és mappák (mint a node\_modules és a dist) ne zavarják a fejlesztést vagy a kód minőségét.

**frontend/package.json:**

A **package.json** fájl a frontend mappában található, és a projekt függőségeit, fejlesztési beállításait, valamint a parancsokat tartalmazza. Az egyes részek jelentése:

1. **name**: A projekt neve, amely ebben az esetben "frontend".
2. **version**: A projekt verziója. Jelenleg 0.0.0, ami azt jelenti, hogy a projekt még nem érte el a stabil kiadást.
3. **private**: Az érték **true**, ami azt jelenti, hogy a projekt nem publikálható a npm (Node Package Manager) registry-be.
4. **type**: **module**, ami azt jelenti, hogy a projekt ES (ECMAScript) modulokat használ.
5. **scripts**: A parancsok, amelyeket a projekt során futtathatunk a npm run <script> formában:
   * **dev**: Futtatja a vite-ot fejlesztői módban, így elindítja a Vite szervert a projekt fejlesztéséhez.
   * **build**: A vite build parancsot futtatja, hogy létrehozza a projekt production (éles) verzióját.
   * **preview**: A vite preview parancsot futtatja, hogy előnézeti módot indítson a kész build-re.
6. **dependencies**: A projekthez szükséges külső függőségek:
   * **@popperjs/core**: A Popper.js könyvtár legújabb verziója, amely a dinamikusan pozicionált elemek kezelésére szolgál.
   * **axios**: HTTP kliens, amely segít az API-k vagy más külső szerverek elérésében.
   * **bootstrap**: A Bootstrap CSS keretrendszer, amely előre elkészített stílusokat és komponenseket biztosít a gyors frontend fejlesztéshez.
   * **pinia**: Állapotkezelő könyvtár Vue 3 projektekhez.
   * **vue**: A Vue.js JavaScript keretrendszer legújabb verziója.
   * **vue-router**: A Vue.js-hez tartozó hivatalos router könyvtár, amely lehetővé teszi az alkalmazás oldalainak navigálását.
7. **devDependencies**: A fejlesztéshez szükséges függőségek:
   * **@vitejs/plugin-vue**: A Vue.js támogatásához szükséges plugin a Vite számára.
   * **sass**: CSS előfeldolgozó, amely lehetővé teszi a Sass (Syntactically Awesome Stylesheets) használatát a projektben.
   * **vite**: A Vite, egy gyors fejlesztői szerver és bundler, amely segíti a projekt fejlesztését és buildelését.

Összefoglalva, ez a **package.json** fájl egy Vue.js alapú frontend projektet ír le, amely a Vite fejlesztői szervert és más népszerű könyvtárakat (pl. Bootstrap, Axios) használ a fejlesztés során.

**frontend/README.md**

Ez a **README.md** fájl a frontend mappában található, és a projekt beállításához és használatához nyújt segítséget. A dokumentáció az alábbiakat tartalmazza:

* **Projekt Célja**: A sablon segít elindítani a fejlesztést Vue 3 és Vite segítségével. Ez egy modern JavaScript keretrendszer, amely lehetővé teszi gyors webalkalmazások létrehozását.
* **Ajánlott Fejlesztői Környezet**: Az ajánlott eszközök közé tartozik a **VSCode** (Visual Studio Code) szövegszerkesztő és a **Volar** kiegészítő, amely biztosítja a Vue 3 projektek számára a legjobb támogatást. A **Vetur** kiegészítőt érdemes kikapcsolni, mert Volar jobb integrációt nyújt a Vue 3-hoz.
* **Konfiguráció Testreszabása**: A dokumentáció említi a Vite konfigurációs lehetőségeit, ha a fejlesztő személyre szeretné szabni a build környezetet.
* **Projekt Beállítása**: Az első lépés a szükséges függőségek telepítése. Ehhez egy egyszerű parancsot kell futtatni, ami letölti és telepíti a szükséges csomagokat a projekt számára.
* **Fejlesztési Környezet**: A fejlesztési folyamatot könnyíti meg egy másik parancs, amely elindítja a Vite fejlesztői szervert, lehetővé téve a gyors visszajelzést és a kódváltozások azonnali alkalmazását.
* **Éles Verzió Készítése**: A dokumentáció arról is szól, hogyan készíthető el a produkciós verzió. A build folyamat során a kód minimalizálása és optimalizálása történik, így az alkalmazás a lehető legjobban fut éles környezetben.

Ez a fájl tehát a frontend alkalmazás elindításához és fejlesztéséhez szükséges lépéseket és eszközöket ismerteti.

**frontend/.vscode/extension.json:**

Ez a extensions.json fájl a .vscode mappában található, és a VSCode fejlesztői környezethez tartozó kiegészítőket (extensions) ajánlja.

A fájl tartalma:

* recommendations: Az ajánlott kiegészítők listáját tartalmazza, jelen esetben csak egy ajánlás található benne: Vue.volar.

A Vue.volar egy kiegészítő, amely a Vue 3 projektekhez biztosít fejlesztői támogatást, például szintaxis kiemelést, kód automatikus kiegészítését és hibajavítást. Az ajánlás arra szolgál, hogy a fejlesztők telepíthessék ezt a kiegészítőt a VSCode-ban a projekthez való optimális fejlesztéshez.

**frontend/src/App.vue:**

Ez az App.vue fájl, amely a frontend Vue.js alkalmazásának fő komponense. A fájl egy template, script és style részből áll, amelyeket az alábbiak szerint magyarázhatunk el:

Template rész:

A HTML szerkezet a felhasználói felületet tartalmazza, beleértve a következő elemeket:

* Négy gomb, amelyek különböző funkciókat hívnak meg: Login, Logout, Szoba létrehozása, és Lap húzás.
* Két <p> elem, amelyek dinamikusan mutatják a lapot (card) és a szoba ID-ját (room\_id).

Script rész:

A Vue 3 Composition API-t használva:

* Imports: Az axios és a ref importálásra kerül.
* Refs: Két ref van definiálva, room\_id és card, amelyek a szoba ID-ját és az aktuális lapot tartalmazzák.
* Functions:
  + login: Ez a funkció egy GET kérést küld a backendhez (http://localhost:5000/login), és átirányítja a felhasználót a visszaadott auth\_url-ra, hogy bejelentkezzen.
  + create\_room: Ez egy POST kérést küld a backendhez (http://localhost:5000/create\_room), és a válaszban lévő szoba ID-t menti el a room\_id változóba. Az Authorization headerben az aktuális hozzáférési tokent küldi.
  + logout: Ez a funkció a backendhez küldött POST kéréssel végzi el a kijelentkezést (http://localhost:5000/logout), eltávolítva a hozzáférési és frissítő tokent a helyi tárolóból, és egy alerttel értesíti a felhasználót.
  + get\_card: Ez a funkció véletlenszerűen kiválaszt egy lapot a cards tömbből és beállítja azt a card változóba.

Style rész:

A CSS stílusok a komponens specifikus stílusait tartalmazzák, mint például:

* A header és nav stílusok, amelyek az oldal elrendezését és a navigációs sávot formázzák.
* A gombok és a szoba-id és kártya megjelenítése is szabályozva van, hogy az UI megfelelően nézzen ki különböző képernyőméreteken.

Ez a fájl tehát egy Vue alkalmazás fő komponense, amely a backenddel való kommunikációt kezeli, beleértve a felhasználói autentikációt, a szoba létrehozását és a lapok kezelését.

**frontend/src/main.js:**

Ez a **main.js** fájl a Vue.js alkalmazás belépési pontja, amely a következő funkciókat látja el:

**Importálás:**

* **import './assets/main.css'**: Betölti az alkalmazás globális CSS fájlját.
* **import { createApp } from 'vue'**: Importálja a Vue 3 fő komponensét, amely lehetővé teszi az alkalmazás létrehozását.
* **import { createPinia } from 'pinia'**: A Pinia egy állapotkezelő könyvtár a Vue 3-hoz, amely az alkalmazás állapotának kezelésére szolgál.
* **import App from './App.vue'**: Az alkalmazás fő komponenst, az App.vue fájlt importálja.
* **import router from './router'**: A Vue Router-t importálja, amely az alkalmazás navigációját kezeli.

**Az alkalmazás inicializálása:**

* **const app = createApp(App)**: Létrehozza a Vue alkalmazást az App.vue fő komponenssel.
* **app.use(createPinia())**: Használja a Pinia állapotkezelőt, amely biztosítja az alkalmazás állapotának kezelését.
* **app.use(router)**: A Vue Routert is hozzáadja az alkalmazáshoz, lehetővé téve a navigációt az alkalmazás különböző oldalai között.
* **app.mount('#app')**: A Vue alkalmazás a #app ID-jú DOM elemhez csatlakozik, amely az index.html fájlban található.

**Összegzés:**

Ez a fájl alapvetően a Vue alkalmazás inicializálásáért felelős, és biztosítja, hogy az alkalmazás működjön a Pinia állapotkezelő és a Vue Router segítségével. Az App.vue a fő komponens, és a main.css fájlban találhatók a globális stílusok.

**frontend/src/assets/base.css:**

Ez a **base.css** fájl alapvető stílusokat és színbeállításokat tartalmaz az alkalmazás számára, különös figyelmet fordítva a színpalettákra és a sötét mód kezelésére.

**Főbb elemek:**

1. **Színpaletta:** A CSS fájl egy színpalettát definiál a projekt számára, amelyet különböző változókban tárol:
   * **--vt-c-white, --vt-c-black, --vt-c-indigo**: Alap fehér, fekete és indigó színek.
   * **--vt-c-divider-light-1, --vt-c-divider-dark-1**: Világos és sötét elválasztó színek a UI elemek elválasztásához.
   * A :root pseudo-osztályban ezek a színek változók formájában elérhetők az egész alkalmazásban.
2. **Szemantikus színváltozók:** A színpalettát kiterjeszti a projekt igényeihez:
   * **--color-background, --color-border**: Háttér- és szegély színek.
   * **--color-heading, --color-text**: Címsorok és szövegek színei.
3. **Sötét mód támogatás:** A @media (prefers-color-scheme: dark) médiakérdés segítségével a fájl biztosítja a sötét módot. Ha a felhasználó eszköze sötét módot használ, a változók más színekkel cserélődnek, hogy illeszkedjenek a sötét téma igényeihez:
   * A háttér színek, a szegélyek és a szöveg színei sötét tónusúra váltanak, hogy jobban olvasható legyen alacsony fényviszonyok mellett.
4. **Alap stílusok:**
   * **\* (összes elem)**: A box-sizing: border-box és a margók törlése biztosítja, hogy minden elem helyesen pozicionálódjon és ne legyenek nem kívánt eltolódások.
   * **body**: Az alap betűtípus és a színbeállítások, valamint a háttér színének változása az alkalmazás színtémájától függően.
   * A betűtípusok középpontjában az **Inter** és a rendszer fontok állnak, hogy biztosítsák a jól olvasható szöveget különböző eszközökön.
5. **Áttűnések:** A **transition** használatával az alkalmazás színeinek váltása zökkenőmentes lesz, például a háttér és a szöveg színének változása fokozatosan történik 0.5 másodperc alatt.

**Összegzés:**

Ez a **base.css** fájl a projekt alapvető stílusait és színbeállításait tartalmazza, és biztosítja a sötét mód megfelelő kezelését is. Az alkalmazás összes elemére kiterjedő alapértelmezett stílusokat definiál, beleértve a betűtípusokat, színeket és a felhasználói élmény javításához szükséges átmeneteket.

**frontend/src/assets/main.css:**

Ez a **main.css** fájl egy alap stíluslap, amely a **base.css** fájlra épít, és további dizájn elemeket ad hozzá az alkalmazás felhasználói felületéhez.

**Főbb elemek:**

1. **Alap stílusok:**
   * Az **#app** elem stílusa meghatározza, hogy a fő alkalmazás konténer maximális szélessége 1280px legyen, és automatikusan középre igazodik a képernyőn. A padding értéke 2rem, ami azt jelenti, hogy van egy kis távolság az alkalmazás tartalma és a konténer szélei között.
   * A **font-weight** alapértelmezetten "normal", tehát a betűk nem lesznek vastagok.
2. **Linkek és zöld szín:**
   * A **a** (link) és **.green** osztályú elemek stílusa a szöveget zöld színűre állítja. A szín **hsla(160, 100%, 37%, 1)**, amely egy élénk zöld árnyalatot képvisel.
   * A **transition** 0.4 másodperces átmenetet ad a stílusok változásához, különösen az interakciók (például hover) során.
   * A **hover** hatás csak akkor lép életbe, ha az eszköz támogatja a hover állapotot (pl. egérrel rendelkező eszközök), ahol a háttér színének világosabb zöld árnyalatúra változik.
3. **Média lekérdezések (Media Queries):**
   * **@media (min-width: 1024px)**: Ha az eszköz képernyője 1024px vagy annál nagyobb szélességű (pl. asztali gépek), akkor a következő változások történnek:
     + A **body** elemet **flexbox** elrendezésűvé teszi, középre helyezve a tartalmat.
     + Az **#app** elem **grid** elrendezésűvé válik, két egyenlő szélességű oszlopra osztva a tartalmat (grid-template-columns: 1fr 1fr), hogy jobban kihasználja a nagyobb képernyőt.
4. **Fontos megjegyzés:**
   * A **@import** utasítások közül az egyik le van kommentálva, amely a **Bootstrap 5.2.3** stílusait importálná egy külső CDN-ből. Ha aktiválnák, akkor a projekthez hozzáadódnának a Bootstrap alapvető dizájnelemei, de jelenleg nem használják.

**Összegzés:**

Ez a **main.css** fájl az alap **base.css** stílusokat bővíti ki, és testreszabott dizájnelemeket ad hozzá az alkalmazás felületéhez. A főbb funkciók közé tartozik a maximális szélesség beállítása, a linkek és zöld színű elemek stílusának definiálása, valamint a nagyobb képernyőkön alkalmazott flexbox és grid elrendezések. A fájl biztosítja, hogy az alkalmazás jól jelenjen meg különböző eszközökön, és az interakciók simán történjenek.

**frontend/src/router/index.js:**

A **frontend/src/router/index.js** fájl Vue Router-t konfigurál a Vue alkalmazáshoz. A router lehetővé teszi a különböző nézetek közötti navigációt az alkalmazásban, és az alábbi fő funkciókkal rendelkezik:

**Főbb részek:**

1. **Router létrehozása:**
   * A **createRouter** és **createWebHistory** funkciók segítségével egy router példányt hoz létre. Az alkalmazás **createWebHistory(import.meta.env.BASE\_URL)** történeti módot használ, amely a URL-t változtatja, de nem frissíti újra az oldalt.
   * A router definiálja az alkalmazás útvonalait **routes** tömbben.
2. **Az útvonalak (Routes):**
   * **/ (HomeView):** Az alapértelmezett útvonal, amely a **HomeView.vue** komponenst rendereli.
   * **/about (AboutView):** Az **AboutView.vue** komponens lazán betöltődik (lazy-loading), amikor az útvonalra navigálunk. Ez javítja az alkalmazás betöltési idejét, mivel csak akkor töltődik be, amikor ténylegesen szükség van rá.
3. **Route Guard - beforeEach:**
   * A **router.beforeEach** egy globális navigációs őr (guard), amely minden útvonalváltás előtt lefut.
   * **to.query.code**: Az útvonal kérdéseihez tartozó **code** paramétert ellenőrzi. Ha megtalálja, akkor a kódot a következő módon kezelik:
     + **axios.post** hívást küld a backend szerverre (http://localhost:5000/callback) a **code** paraméterrel, és megpróbálja megszerezni az **access\_token**-t és a **refresh\_token**-t a válaszból.
     + A kapott **access\_token**, **refresh\_token** és egyéb információk (például a felhasználói név) a **localStorage**-ba kerülnek mentésre, hogy a felhasználói bejelentkezési állapotot megőrizzük.
     + Ha sikeresen megkapjuk az adatokat, akkor a **next()** függvény hívása biztosítja, hogy a navigáció folytatódjon, és a felhasználó beléphet az alkalmazásba.
     + Ha hiba történik, a hibát a **catch** blokk kezeli, és a navigációt tovább engedjük.
4. **Navigáció kezelése:**
   * A **router.replace({ path: to.path })** átirányítja a felhasználót ugyanarra az útvonalra, de eltávolítja a kódsor paramétert, hogy az URL-t tisztán tartsuk a kódtól.
   * Az **next()** hívás biztosítja a navigáció folytatását, akkor is, ha nem található kód a lekérdezésben, vagy ha a kód sikeresen feldolgozásra került.

**Összegzés:**

A **router/index.js** fájl egy Vue Router konfiguráció, amely az alkalmazás különböző nézeteihez tartozó útvonalakat határoz meg. A router biztosítja, hogy a felhasználó bejelentkezéskor kezelje a kódot, amelyet a backend API hívása után elment a **localStorage**-ba, és ezen keresztül kezeli a felhasználó hitelesítési állapotát. Az útvonalak között navigálás közben biztosítja a felhasználói élményt és a zökkenőmentes működést.

**frontend/src/stores/counter.js:**

A **frontend/src/stores/counter.js** fájl a **Pinia** könyvtárat használja a globális állapot kezelésére a Vue alkalmazásban. A **Pinia** az új állapotkezelési megoldás Vue 3-ban, amely a **Vuex**-et váltja fel, és egyszerűbb, könnyebben kezelhető API-t biztosít.

**Főbb részek:**

1. **Pinia Store létrehozása:**
   * Az **defineStore** függvény használatával egy új store-t hozunk létre, amelynek neve **counter**. Ez egy név, amivel hivatkozhatunk a store-ra a komponenseken belül.
   * Az első argumentumban a store neve, a második pedig egy **kompozíciós API**-val megadott objektum, ami tartalmazza a store állapotát, számított értékeit és metódusait.
2. **Állapot (State):**
   * A **count** változó a store állapotát reprezentálja. A **ref** segítségével egy reaktív változót hozunk létre, amely a Vue reaktív rendszere által figyelt és frissített.
   * A kezdeti értéke **0**, tehát a szám számláló állapota.
3. **Számított értékek (Computed Properties):**
   * A **doubleCount** egy számított érték, amely mindig a **count** érték kétszerese. A **computed** segítségével egy olyan reaktív változót hozunk létre, amely automatikusan újraszámítódik, amikor a **count** értéke változik.
   * Ez azt jelenti, hogy minden alkalommal, amikor a felhasználó növeli a számlálót, a **doubleCount** is automatikusan frissül.
4. **Metódusok:**
   * Az **increment** függvény növeli a **count** értékét. Mivel a **count** reaktív, a Vue automatikusan frissíti a DOM-ot, ha a **count** változik.
5. **Visszatérési érték:**
   * A store a következő három dolgot ad vissza:
     + **count**: A számláló állapotát.
     + **doubleCount**: A számított értéket, amely a számláló kétszerese.
     + **increment**: A számláló növelésére szolgáló függvény.

**Összegzés:**

Ez a store a **Pinia**-t használva hoz létre egy egyszerű számlálót, amelynek állapota a **count** változó, és van egy számított érték **doubleCount**, amely a számláló kétszerese. A **increment** függvény növeli a **count** értékét. Ez a store jól illeszkedik a Vue 3 kompozíciós API-jába, és könnyen integrálható a Vue alkalmazásba a globális állapot kezelésére.

**frontend/src/views/AboutView.vue:**

Az **AboutView.vue** fájl egy Vue komponens, amely az "about" oldalt jeleníti meg a webalkalmazásban. Az oldal tartalmaz egy egyszerű szöveges üzenetet: "This is an about page", amely egy <h1> HTML elemben jelenik meg.

A stílusok csak akkor alkalmazódnak, ha a képernyő szélessége 1024px vagy nagyobb, tehát asztali képernyők esetén. Ilyenkor a .about osztályú div legalább akkora magas, mint a böngészőablak, és a szöveg középre van igazítva a képernyőn, hogy vizuálisan középre kerüljön.

Ez az oldal statikus, információs jellegű, és a Vue Router segítségével érhető el az /about útvonalon.

**frontend/src/views/HomeView.vue:**

A HomeView.vue fájl a webalkalmazás főoldalát jeleníti meg. A fájl Vue 3-ban készült és az új Composition API-t használja a kód felépítéséhez. A script setup blokkban egy TheWelcome nevű komponenst importál, amely a főoldalon megjelenik. Az importált komponens a components/TheWelcome.vue fájlban található, tehát annak tartalma határozza meg, hogy mi jelenik meg az oldalon.

A sablon (template) részben a <main> HTML elem tartalmazza a TheWelcome komponenst, amely valószínűleg üdvözlő szöveget vagy egyéb információt jelenít meg a felhasználónak a főoldalon.

Ez az oldal statikus és egyszerű, és a Vue Router segítségével érhető el az alapértelmezett / útvonalon.