Projektdokumentation: TotalPETView

1. Einleitung

1.1 Projektübersicht

TotalPETView (TPV) ist ein web-basiertes System zur Verwaltung und Visualisierung von medizinischen Bilddaten im DICOM-Format, mit einer Spezialisierung auf PET-Scans. Die Anwendung bietet eine benutzerfreundliche Oberfläche für den Upload, die Suche und die Anzeige von medizinischen Studien und richtet sich an medizinisches Fachpersonal wie Radiologen und Forscher.

1.2 Projektziel

Das Hauptziel des Projekts ist die Schaffung eines DICOM-Viewers, der die sequenzielle Darstellung von PET/CT-Scans ermöglicht. Eine zentrale Anforderung ist die Implementierung einer standardisierten DICOM-Schnittstelle (PACS) zur Anbindung an externe Systeme. Die ursprünglich geplante Integration des "TotalSegmentator" zur automatisierten Organsegmentierung wurde aus Zeitgründen zurückgestellt.

2. Systemarchitektur

2.1 Architekturübersicht

Das System ist als klassische Drei-Schichten-Architektur konzipiert:

1. **Frontend (Vue.js)**: Eine Single-Page-Application (SPA), die im Browser des Benutzers ausgeführt wird und die gesamte Benutzeroberfläche bereitstellt.
2. **Backend (Django)**: Ein Python-basierter Server, der als Schnittstelle (API) zwischen dem Frontend und dem PACS-Server dient. Er übernimmt die Geschäftslogik, die Benutzerauthentifizierung und die Weiterleitung von Anfragen.
3. **PACS (Orthanc)**: Ein leichtgewichtiger, Open-Source DICOM-Server, der als Docker-Container läuft. Er ist für die Speicherung, Verwaltung und den Abruf der DICOM-Dateien zuständig.

2.2 Technologiestack

- **Backend**: Python, Django, Django REST Framework, Simple JWT
- **Frontend**: JavaScript, Vue.js, Vuex, Vue Router, Axios
- **PACS**: Orthanc
- **Datenbank**: SQLite (für die Django-Benutzerverwaltung)
- **Containerisierung**: Docker, Docker Compose

2.3 Kommunikationsfluss

- **Authentifizierung**: Der Benutzer meldet sich über das Vue-Frontend an. Die Anmeldedaten werden an das Django-Backend gesendet, das mit Simple JWT ein JSON Web Token (JWT) generiert und zurückgibt. Dieses Token wird im Frontend gespeichert und für alle nachfolgenden authentifizierten Anfragen verwendet.
- **Upload**: Eine DICOM-Datei wird im Frontend ausgewählt und an einen geschützten Endpunkt im Django-Backend gesendet. Das Backend leitet die Datei an die REST-API des Orthanc-Servers weiter, wo sie gespeichert wird.
- **Suche**: Suchanfragen vom Frontend werden an das Django-Backend gesendet. Dieses übersetzt die Anfrage in eine für Orthanc verständliche Abfrage (`/tools/find`), sendet sie an den Orthanc-Server und leitet die Ergebnisse an das Frontend zurück.
- **Anzeige**: Wenn eine Studie im Frontend zur Anzeige ausgewählt wird, ruft dieses die detaillierten Studieninformationen über das Backend vom Orthanc-Server ab. Die Darstellung der Bilder im Viewer ist geplant, aber noch nicht vollständig implementiert.

3. Installation und Inbetriebnahme

Das gesamte System ist containerisiert und wird mit Docker Compose verwaltet.

3.1 Voraussetzungen

- Docker und Docker Compose müssen installiert sein.

3.2 Ausführung

1. Klonen Sie das Git-Repository:

```
git clone https://github.com/VlntnLnhrdt/TotalPETView.git
cd TotalPETView
```

2. Starten Sie alle Dienste (Frontend, Backend, Orthanc) mit Docker Compose:

```
docker compose up -d
```

3. Die Anwendung ist nun verfügbar:
    - **Frontend**: `http://localhost:5173`
    - **Backend API**: `http://localhost:8000`
    - **Orthanc Web-UI**: `http://localhost:8042` (Login: `orthanc`, Passwort: `password`)

3.3 Beispieldaten

Für die Erstinbetriebnahme können Sie Beispieldaten herunterladen und über die Upload-Funktion der Anwendung oder direkt über die Orthanc-Weboberfläche importieren.

4. Backend (Django)

Das Backend ist modular in zwei Haupt-Apps aufgeteilt: `authenticator` und `api`.

4.1 App: `authenticator`

Diese App ist für die Benutzerverwaltung und Authentifizierung zuständig. Sie verwendet `django.contrib.auth` für die Benutzermodelle und `rest_framework_simplejwt` für die Token-basierte Authentifizierung.

Endpunkte:

- `POST /api/auth/register/`: Registriert einen neuen Benutzer.
- `POST /api/auth/login/`: Authentifiziert einen Benutzer und gibt ein Access- und Refresh-Token zurück.
- `POST /api/auth/login/refresh/`: Erneuert ein abgelaufenes Access-Token.

4.2 App: `api`

Diese App stellt die Kernfunktionalität für die Interaktion mit dem Orthanc-Server bereit. Alle Endpunkte erfordern eine gültige JWT-Authentifizierung.

Endpunkte:

- `POST /api/upload/`: Lädt eine DICOM-Datei (`.dcm`) hoch und speichert sie in Orthanc.
- `GET /api/search/?query=<Patientenname>`: Sucht in Orthanc nach Studien, die auf den Patientennamen passen.
- `GET /api/studies/<study_id>/`: Ruft detaillierte Informationen zu einer bestimmten Studie aus Orthanc ab, inklusive der zugehörigen Serien.

4.3 Konfiguration

Die Backend-Konfiguration befindet sich in `backend/settings.py`. Hier werden die installierten Apps, Middleware, Datenbank (SQLite) und die Verbindungsdaten zum Orthanc-Server (`ORTHANC_URL`, `ORTHANC_USER`, `ORTHANC_PASSWORD`) definiert.

5. Frontend (Vue.js)

Das Frontend ist eine moderne Single-Page-Application, die mit Vue.js und dem Vue-CLI-Tooling aufgebaut ist.

5.1 Projektstruktur

- `src/assets/`: Globale CSS-Dateien und Bilder.
- `src/components/`: Wiederverwendbare Vue-Komponenten (z.B. `Layout.vue` für das Hauptlayout).
- `src/views/`: Seitenkomponenten, die durch den Router aufgerufen werden (z.B. `Login.vue`, `Search.vue`, `Upload.vue`).
- `src/router.js`: Definiert die URL-Pfade und die zugehörigen Komponenten. Implementiert Navigations-Guards, um den Zugriff auf geschützte Routen zu steuern.
- `src/store/`: Enthält den Vuex-Store für das globale State-Management.
    - `auth.js`: Verwaltet den Authentifizierungsstatus (Token, Benutzer) und stellt Aktionen für Login, Logout und Registrierung bereit.
    - `api.js`: Ein zentralisierter API-Client (basierend auf `axios`), der die Kommunikation mit dem Django-Backend kapselt. Ein Interceptor fügt automatisch den JWT-Bearer-Token zu jeder Anfrage hinzu.

5.2 Wichtige Ansichten (Views)

- **Login/Register**: Formulare zur Benutzerauthentifizierung.
- **Search**: Bietet ein Suchfeld zur Abfrage von Patientenstudien. Die Ergebnisse werden als Liste angezeigt.
- **Upload**: Ermöglicht das Auswählen und Hochladen von DICOM-Dateien.
- **Viewer**: Soll die DICOM-Bilder einer Studie anzeigen. Die Integration der `Cornerstone.js`-Bibliothek ist vorbereitet, aber die finale Implementierung zum Laden und Anzeigen der Bilder steht noch aus.

6. Sicherheit

- **Authentifizierung**: Der Zugriff auf die Kernfunktionen der Anwendung wird durch eine Token-basierte Authentifizierung (JWT) geschützt.
- **Autorisierung**: Das Backend schützt API-Endpunkte, sodass nur authentifizierte Benutzer darauf zugreifen können. Das Frontend leitet nicht-authentifizierte Benutzer zur Login-Seite um.
- **CORS**: Das Django-Backend ist mit `django-cors-headers` so konfiguriert, dass Cross-Origin-Anfragen vom Vue.js-Frontend (das auf einem anderen Port läuft) akzeptiert werden (`CORS_ALLOW_ALL_ORIGINS = True` für die Entwicklung).
- **Netzwerksicherheit**: Im Docker-Setup kommunizieren Backend und Orthanc über ein internes Docker-Netzwerk. Der Orthanc-Server ist nicht direkt aus dem Internet erreichbar, was die Angriffsfläche reduziert.

7. Zukünftige Entwicklung

- **Viewer-Implementierung**: Die dringendste Aufgabe ist die vollständige Implementierung des DICOM-Viewers in der `Viewer.vue`-Komponente mit `Cornerstone.js`, um die Bilder darzustellen und Werkzeuge (Zoom, Pan, etc.) bereitzustellen.
- **Sicherheit**: Für eine Produktionsumgebung muss `CORS_ALLOW_ALL_ORIGINS` auf eine spezifische Whitelist von Domains beschränkt werden. Zudem sollte die Datenübertragung mit HTTPS verschlüsselt werden.
- **TotalSegmentator**: Die ursprünglich geplante Integration zur automatischen Segmentierung könnte als nächstes großes Feature umgesetzt werden.