**Project Folder Structure and Files**

**1. Backend**

* **src/** (Rust backend)
  + **main.rs**: Sets up the main server configurations and initializes necessary services.
  + **config.rs**: Manages configuration settings (e.g., database connection, secret keys, environment variables).
  + **db/** (Database module for MS SQL interaction)
    - **mod.rs**: Defines the module structure for database operations.
    - **connection.rs**: Establishes a connection pool to the MS SQL database using sqlx.
    - **models.rs**: Defines data models that map to database tables (e.g., User, Provider, Appointment, etc.).
    - **queries.rs**: Contains SQL queries for CRUD operations and any specific data retrieval logic.
  + **routes/** (API endpoints)
    - **mod.rs**: Sets up and groups all API routes.
    - **auth.rs**: Handles user authentication endpoints (e.g., login, register, logout).
    - **providers.rs**: Manages endpoints related to telehealth providers (e.g., GET /providers, POST /providers for searches).
    - **appointments.rs**: Manages appointment-related endpoints, including real-time availability updates.
    - **education.rs**: Manages endpoints for educational content, such as articles on telehealth options.
    - **notifications.rs** (Optional): Endpoint to manage user notifications (e.g., appointment availability alerts).
  + **services/** (Business logic)
    - **mod.rs**: Defines the services module.
    - **auth\_service.rs**: Handles authentication, authorization, and encryption of sensitive user data.
    - **provider\_service.rs**: Manages logic related to provider searches, including filters by specialty, location, and insurance compatibility.
    - **appointment\_service.rs**: Handles appointment scheduling, real-time data fetching, and updates.
    - **education\_service.rs**: Provides data on telehealth services, pulling resources for educational outreach.
  + **utils/** (Helper utilities)
    - **mod.rs**: Defines the utilities module.
    - **hashing.rs**: Implements password hashing and other security utilities.
    - **validation.rs**: Defines input validation functions.
    - **error\_handling.rs**: Centralized error handling and logging.
  + **middleware/** (Middleware for security and logging)
    - **auth\_middleware.rs**: Ensures protected routes are accessible only by authenticated users.
    - **logging\_middleware.rs**: Logs user actions and system activity for monitoring and debugging.
* **Improvement Suggestions**:
  + **Real-Time Data Synchronization**: Implement an additional module in services that syncs real-time data with an external API if telehealth providers offer such data.
  + **Caching**: For performance, consider adding a caching layer in utils to store frequently accessed data like provider lists or educational content.

**2. Frontend**

* **public/** (Static files)
  + **index.html**: Main HTML page template with <div id="app"></div> for the JavaScript frontend app to load.
  + **styles/**
    - **main.css**: Styles for the dashboard. Include a clean, user-friendly design, with focus on accessibility.
  + **images/** (Optional): Placeholder images for educational articles or icons for telehealth providers.
* **src/** (JavaScript application)
  + **app.js**: Entry point for the frontend, bootstrapping the main view and setting up routes.
  + **components/** (UI Components)
    - **Header.js**: Renders the navigation bar, allowing users to switch between educational resources, provider search, and appointment booking.
    - **Footer.js**: Basic footer with project credits and additional links.
    - **ProviderSearch.js**: Displays the search form with filters and dynamically lists telehealth providers based on user preferences.
    - **AppointmentBooking.js**: Manages appointment scheduling and displays availability in real-time.
    - **EducationalContent.js**: Showcases articles, videos, or resources educating users on telehealth services.
    - **Login.js** and **Register.js**: Handle user authentication.
    - **Notifications.js** (Optional): Alerts users about upcoming or newly available appointments.
  + **api/** (API interaction)
    - **auth.js**: Manages authentication-related API calls.
    - **providers.js**: Manages provider search and filter API calls.
    - **appointments.js**: Handles appointment-related API interactions, including real-time updates.
    - **education.js**: Manages educational content requests from the backend.
  + **utils/**
    - **config.js**: Holds the frontend configuration variables.
    - **helpers.js**: Utility functions for tasks like form validation and date formatting.
  + **store/** (Application state management)
    - **index.js**: Centralized state management for handling shared data like user sessions, notifications, and provider search filters.
* **Improvement Suggestions**:
  + **Responsive Design**: Ensure the dashboard is fully responsive, catering to users on mobile devices.
  + **Accessibility**: Incorporate ARIA roles and ensure keyboard navigability.
  + **Error Handling UI**: Display informative messages to users if something goes wrong with API requests.

**3. Database**

* **Tables**
  + users: Stores user information (hashed password, email, preferences).
  + providers: Contains telehealth provider details (name, location, specialties, contact info).
  + appointments: Tracks scheduled appointments, including user and provider IDs and time.
  + educational\_content: Holds educational articles, videos, or links on telehealth.
  + notifications (Optional): Stores alerts or reminders for users.
* **Improvement Suggestions**:
  + **Indexes**: Add indexes on fields like provider location and specialty for optimized search.
  + **Audit Logs**: Consider adding an audit log table for tracking user actions (e.g., login, search queries).

**4. Environment and Configuration**

* **Environment Variables**
  + Ensure environment variables are stored in .env and referenced securely in config.rs and config.js.

**5. Security**

* **Backend**
  + JWT-based authentication for secure login sessions.
  + Use Rust's crypto library for hashing and encrypting sensitive data.
* **Frontend**
  + CSRF tokens for secure API calls.
  + Input sanitization to prevent XSS.

**6. Testing**

* **Unit Tests**: Test individual components, especially for services and API calls.
* **Integration Tests**: Test database interactions, ensuring data is handled correctly.
* **E2E Tests**: Simulate real user interactions with tools like Selenium.