**Accessibility Analysis System Documentation**

## **Table of Contents**

1. Overview
2. Architecture
3. Technology Stack
4. Backend Documentation
   * Features
   * Project Structure
   * Routes and Endpoints
   * Database and Migrations
   * Third-Party Integrations
5. Frontend Documentation
   * Features
   * Project Structure
   * Routing and UI Components
   * Environment Variables
6. Setup and Installation
7. Testing
8. Future Work and Enhancements
9. Scoring Logic
10. References

## **1. Overview**

The **Accessibility Analysis System** is composed of a backend API and a frontend React application. The system is designed to help users analyze HTML files for accessibility issues. It performs a rule-based analysis for missing attributes (like alt for images) and skipped heading levels. Additionally, it integrates with OpenAI’s API to provide enhanced recommendations and fixed HTML output. The backend persists the results in a PostgreSQL database, while the frontend provides an interactive interface for file upload and result display. Please run the command “chmod +x start-projects.sh” and “./start-projects.sh” to start and install all dependencies.

## 

## **2. Architecture**

* **Backend**:  
   The backend is built using Express.js with TypeScript. It uses TypeORM to interact with a PostgreSQL database, and Multer to handle file uploads. The backend exposes RESTful endpoints for uploading HTML files, performing accessibility analysis, and returning results. OpenAI is integrated to provide additional analysis and recommendations.
* **Frontend**:  
   The frontend is a React application set up using Vite for fast development and bundling. It uses React Router for navigation between pages, Tailwind CSS for styling, and fetches data from the backend API. The app allows users to upload HTML files and view analysis results.
* **Data Flow**:  
  1. A user uploads an HTML file via the frontend.
  2. The frontend sends the file to the backend API endpoint.
  3. The backend processes the file using Cheerio for rule-based accessibility analysis.
  4. The analysis results are enriched via an OpenAI integration.
  5. The backend stores the results in PostgreSQL and returns the data as JSON.
  6. The frontend displays the analysis results, including the compliance score, identified issues, and OpenAI recommendations.

## 

## 

## 

## 

## 

## 

## **3. Technology Stack**

* **Backend**:  
  + **Framework**: Express.js, TypeScript
  + **ORM**: TypeORM
  + **Database**: PostgreSQL
  + **File Upload**: Multer
  + **HTML Parsing**: Cheerio
  + **External API Integration**: OpenAI API
  + **Testing**: Jest, SuperTest
  + **Containers**: Docker
* **Frontend**:  
  + **Framework**: React, TypeScript
  + **Bundler/Dev Server**: Vite
  + **Routing**: React Router DOM (v6)
  + **Styling**: Tailwind CSS
  + **Testing**: Vitest, @testing-library/react, @testing-library/user-event

## 

## 

## 

## 

## 

## 

## 

## 

## **4. Backend Documentation**

### **Features**

* Accepts HTML file uploads.
* Performs accessibility analysis including:
  + Missing alt attributes.
  + Skipped or incorrect heading levels.
* Calculates a compliance score.
* Enhances results using OpenAI completions.
* Persists analysis results in PostgreSQL.
* Supports database migrations and seeding.
* Supports Docker but its not required

### **Project Structure**

backend/

├── config/

└── database.ts // Database connection configuration

├── controllers/

└── accessibility.controller.ts // Handles file upload and analysis

├── models/

├── AccessibilityIssue.ts // Entity for accessibility issue definitions

└── AccessibilityResult.ts // Entity for storing analysis results

├── routes/

└── accessibility.routes.ts // API endpoints for analysis

├── seeders/

└── seed.accessibilityIssues.ts // Seeds known accessibility issues

├── services/

├── accessibility.service.ts // Core logic for file analysis and persistence

└── openai.service.ts // Handles integration with OpenAI API

├── validations/

└── fileUpload.validation.ts // Middleware for file upload validation

├── utils/

└── htmlAnalyzer.ts // Utility to parse and analyze HTML content

├── app.ts // Express application configuration

└── server.ts // Application entry point (initializes DB, seeds, starts server)

├── migrations/ // TypeORM migration files

├── Dockerfile // Dockerfile images

├── docker-compose.yml //For building the Dockerfile images

├── .env // Environment variable definitions

├── package.json // NPM scripts and dependencies

├── tsconfig.json // TypeScript configuration

└── README.md // Project documentation

### **Routes and Endpoints**

#### **POST /api/accessibility/analyze**

* **Description**:  
   Accepts an HTML file upload, analyzes the file for accessibility issues, enhances the analysis with OpenAI, and returns a JSON object containing the results.
* **Middleware**:  
  + **Multer**: Handles file uploads.
  + **File Validation**: Ensures a file is present.

**Response Format**:  
  
 {

"id": 16,

"complianceScore": 80,

"issues": [

{

"ruleName": "Missing Alt Attribute",

"description": "Images should have an alt attribute for accessibility.",

"suggestedFix": "Add alt attribute describing the image content."

},

{

"ruleName": "Skipped Heading Levels",

"description": "Headings should be used sequentially to maintain hierarchy.",

"suggestedFix": "Ensure that heading tags (h1-h6) are used in a sequential order."

}

],

"createdAt": "2025-02-01T09:00:14.604Z",

"openAI": {

"fixedHtml": "<!DOCTYPE html> ... </html>",

"recommendations": "Added an alt attribute... \nRearranged headings..."

}

}

### 

### 

### 

### **Database and Migrations**

* Uses **PostgreSQL** for data storage.
* **TypeORM Migrations** are used to manage schema changes.
* **Seeder Scripts** populate known accessibility issues into the database.

### **Third-Party Integrations**

* **OpenAI API**:  
   The backend sends the raw HTML file to OpenAI's completions endpoint to receive an improved HTML version and recommendations.
* **Environment Variables**:  
   Configurable via the .env file (e.g., database credentials, OpenAI API key).

## 

## 

## 

## 

## 

## 

## 

## **5. Frontend Documentation**

### **Features**

* Upload HTML files to the backend for analysis.
* Display analysis results including compliance score and accessibility issues.
* Show OpenAI enhanced HTML and recommendations.
* Provide smooth navigation between Home, Upload, and Result pages.
* Responsive UI using Tailwind CSS.

### **Project Structure**

accessibility-app/

├── public/

│ └── index.html // Main HTML template

├── src/

│ ├── \_\_*tests*\_\_/ // Test cases for vitest

│ ├── components/ // Reusable UI components (e.g., Header, Footer)

│ ├── pages/ // Page components (e.g., Home, Upload, AnalysisResult)

│ ├── services/ // API service files (e.g., accessibilityAPI.ts)

│ ├── App.tsx // Application root with routing

│ ├── index.tsx // Vite entry point

│ ├── index.css // Global styles and Tailwind directives

│ ├── vite-env.d.ts // Vite TypeScript environment definitions

│ └── tests/ // Vitest test files

├── .env // Environment variable definitions

├── package.json // NPM scripts and dependencies

├── tsconfig.json // TypeScript configuration

├── tailwind.config.js // Tailwind CSS configuration

├── vite.config.js // Vite configuration

└── postcss.config.js // PostCSS configuration

### **Routing and UI Components**

* **React Router DOM (v7)** is used for navigation.
* **Pages**:
  + **Home**: The upload page.
  + **Upload**: Form for selecting and uploading an HTML file.
  + **Analysis Result**: Displays the results returned by the backend API.
* **Components**:  
   Reusable components include a header and footer for consistent UI across pages.

### 

### **Environment Variables**

**VITE\_ACCESSIBILITY\_API\_URL**:  
 This variable defines the backend API endpoint URL.  
 Example (in .env):  
 VITE\_ACCESSIBILITY\_API\_URL=http://localhost:3000/api/accessibility/analyze

* *Note: Vite exposes only environment variables prefixed with VITE\_.*

## **6. Setup and Installation**

**Install Dependencies**:  
  
 npm install

**Configure Environment Variables**:  
 Create a .env file in the project root:  
  
 VITE\_ACCESSIBILITY\_API\_URL=http://localhost:3000/api/accessibility/analyze

**Run the Development Server**:  
  
 npm run dev

1. The app will be served by Vite (typically on http://localhost:5173).

## **7. Testing**

* **Test Runner**: Vitest

**Running Tests**:  
 To run tests, execute:  
 npm run test

* **Test Files**:  
   Tests are located in the src/\_\_tests\_\_/ directory
* **Example Test**:  
   A sample test for the Upload page verifies that the upload form renders correctly.

## **8. Future Work and Enhancements**

* **Backend Enhancements**:
  + Improve accessibility analysis with more rules.
  + Enhance OpenAI integration for better recommendations.
  + Implement authentication and user management.
* **Frontend Enhancements**:
  + Improve UI/UX based on user feedback.
  + Add more detailed analytics and reporting.
  + Integrate real-time progress updates during analysis.
* **Testing**:
  + Increase test coverage for both backend and frontend components.
  + Set up end-to-end tests with a tool like Cypress.

## **9. Scoring Logic**

**Scoring Logic**

The scoring logic is a central part of the backend’s accessibility analysis. It serves as a quantitative measure of how well the submitted HTML meets accessibility standards.

**Base Score**

* **Starting Point**:  
  Every HTML file begins with a base compliance score of **100**.

**Issue Penalties**

* **Per Issue Penalty**:  
  Each accessibility issue identified deducts a fixed amount from the base score and in our case **10 points** per issue detected.
* **Minimum Score**:  
  The score is never allowed to drop below **0**. This ensures that even if many issues are present, the score remains within a predictable range.

**Example Calculation**

Suppose an HTML file is analyzed and the following issues are detected:

* **Missing Alt Attribute** on one or more <img> elements.
* **Skipped Heading Levels**, where the heading hierarchy is not properly maintained.

If the rule-based analysis detects 2 issues (for example, one missing alt attribute and one skipped heading level) and each issue carries a penalty of 10 points, the compliance score would be calculated as follows:

Compliance Score = 100 - (Number of Issues \* 10)

= 100 - (2 \* 10)

= 100 - 20

= 80

**Mapping to Known Accessibility Rules**

* **Issue Identification**:  
  When an issue is found (e.g., an <img> tag without an alt attribute), the system maps this to a known accessibility rule that has been seeded in the database. This rule includes a description and a suggested fix.
* **Enhanced Recommendations**:  
  In addition to the rule-based detection, the backend leverages OpenAI to provide enhanced analysis. The output from OpenAI includes:
  + **Fixed HTML**: A revised version of the HTML that addresses the detected issues.
  + **Recommendations**: Additional insights and recommendations for further improvements.
* **Final Response**:  
  The final JSON response sent to the frontend includes the compliance score, a list of issues (with detailed descriptions and fixes), and the OpenAI enhancements. This information is then presented to the user in the Analysis Result page of the frontend application.

## **10. References**

* [Express.js Documentation](https://expressjs.com/)
* [TypeORM Documentation](https://typeorm.io/)
* [React Documentation](https://reactjs.org/)
* [Vite Documentation](https://vitejs.dev/)
* [Vitest Documentation](https://vitest.dev/)
* [Tailwind CSS Documentation](https://tailwindcss.com/)
* [OpenAI API Documentation](https://beta.openai.com/docs/)