1. Overview

The Accessibility Checker is a **client-server application** designed to evaluate the accessibility compliance of HTML files. The system identifies issues such as missing attributes, skipped heading levels, and other accessibility barriers. It generates a compliance score and provides suggestions for fixes. The architecture is modular, scalable, and follows standard software design principles.

2. Architecture

The application uses a **3-tier architecture**:

- 1. Presentation Layer (Frontend):
 - Built with **React** for a user-friendly interface.
 - Styled with **Tailwind CSS** for responsiveness and modern design.
 - Handles file upload and displays the analysis results.
- 2. Application Logic Layer (Backend):
 - Built with **Node.js** and **Express**.
 - Processes uploaded HTML files using a rule-based algorithm.
 - Manages API endpoints for file upload and analysis response.
- 3. Data Processing Layer:
 - Utilizes Cheerio to parse HTML and traverse its structure.
 - Implements accessibility rules to evaluate compliance.

3. System Components

Frontend

- React Components:
 - FileUpload: Handles file selection and uploads.
 - AccessibilityReport: Displays the compliance score and issue details.
- Libraries:
 - **Axios**: Communicates with the backend API.
 - **Cors**: Allow requests from specific origins or all origins during development.(Middleware)
 - Tailwind CSS: Provides pre-built styles for rapid UI development.

Backend

- Node.js Modules:
 - Express: Sets up API endpoints.
 - Multer: Handles file uploads securely.
 - Cheerio: Parses and analyzes HTML documents.
- Endpoints:
 - POST /upload: Accepts the uploaded HTML file and returns the analysis results.

Scoring Logic

- The compliance score starts at 100% and deducts points for each identified issue.
- Rules:
 - 1. Missing alt attributes: Deduct 10 points per instance.
 - 2. Skipped heading levels: Deduct **5 points** per instance.
 - 3. Empty anchor tags: Deduct 5 points per instance.
- The final score is rounded to the nearest integer.

4. Scoring Logic Design

Algorithm

1. Parse the HTML:

• Use **Cheerio** to traverse the DOM structure.

2. Apply Rules:

- Iterate through elements (, <h1>, <h2>, etc.).
- Detect issues based on predefined criteria.

3. Deduct Points:

• Subtract points from a base score of 100 based on issue severity.

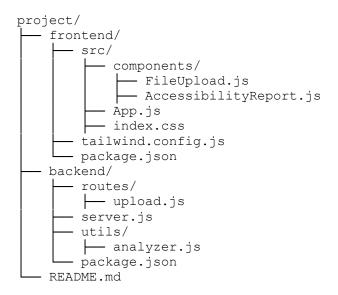
4. Generate Suggestions:

• For each issue, provide a fix recommendation.

Example Rule Implementation (Backend)

```
// Rule 1: Missing alt attributes
('img').each((index, img) => {
    if (!$(img).attr('alt')) {
        issues.push({
            issue: 'Missing alt attribute',
            element: $.html(img),
            suggestion: 'Add a descriptive alt attribute to this image.',
        });
        score -= 10;
    }
});
// Rule 2: Skipped heading levels
const headings = [];
('h1, h2, h3, h4, h5, h6').each((_, heading) => {
   headings.push(parseInt(heading.tagName[1]));
headings.forEach((level, index) => {
    if (index > 0 \&\& level - headings[index - 1] > 1) {
        issues.push({
            issue: 'Skipped heading level',
            suggestion: `Use <h${headings[index - 1] + 1}> instead of
<h${level}>.`,
        });
        score -= 5;
    }
});
```

5. Folder Structure



Key Files

Frontend:

- FileUpload.js: Handles file selection and uploads.
- AccessibilityReport.js: Displays results and compliance score.

Backend:

- server.js: Configures Express and API routes.
- analyzer.js: Contains logic for parsing and analyzing HTML.

6. Data Flow

1. Frontend:

- Accepts file input from the user.
- Sends the file to the backend using Axios (POST /upload).

2. Backend:

- Processes the file upload using Multer.
- Parses HTML and detects issues with Cheerio.
- Computes a compliance score and suggestions.
- Returns results to the frontend as a JSON response.

3. Frontend:

• Displays the compliance score and issue details.

7. Use Case Scenarios

Scenario 1: Missing alt Attributes

• HTML Input:

```
<img src="image1.jpg">
```

- Detected Issue:
 - Missing alt attribute.
- Suggested Fix:

• Add a descriptive alt attribute:

```
<img src="image1.jpg" alt="Description">.
```

Scenario 2: Skipped Heading Levels

• HTML Input:

```
<h1>Main Title</h1><h3>Subsection</h3>
```

- Detected Issue:
 - Heading levels are skipped ($\langle h1 \rangle \rightarrow \langle h3 \rangle$).
- Suggested Fix:
 - Use <h2> instead of <h3>.