**Software Design Document**

**GIT WEB SCRAPPER**

**Development team:**

Marouthu, Sai Nikith

Rapolu, Ravi Teja

Gudiseva, Bhavya Sree

Cheedella, Prudhvi Lakshmi Sesha Sai

Gummadi, Karthik

Distribution clause:

This document was created as a deliverable for the Fall 2024 CSCi 6235 course at the George Washington University, 1918 F Street, NW, Washington, DC 20052. Distribution is unrestricted provided the source is referenced.

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Affected Pages** | **Explanation of Change** |
| 1 | 11/14 /2024 |  | Initial Draft |
| 2 | 11/27/2024 | 3, 10, 11,12,13 | Changed SBOM added dates, use case sequence diagrams, Added Activity diagram |
| 3 | 12/04/2024 | Title page, 1, 2, 3, 7,8, 11, 12, 13 | Updated figure numbers and small grammar errors |

TABLE OF CONTENTS

[**Scope** 1](#_Toc183625401)

[**Overview** 1](#_Toc183625402)

[**Theoretical background and definition of terms** 1](#_Toc183625403)

[**Software Architecture** 3](#_Toc183625404)

[**Software Bill of Materials (SBOM)** 3](#_Toc183625405)

[**File Structure** 6](#_Toc183625406)

[**Data Design** 8](#_Toc183625407)

[**Object Structure** 8](#_Toc183625408)

[**Classes** 9](#_Toc183625409)

[**Inter-Process Communication** 9](#_Toc183625410)

[**Software Functionality** 10](#_Toc183625411)

[**Use Cases** 10](#_Toc183625412)

[**Function Execution** 13](#_Toc183625413)

# **Scope**

This project consists of a web application designed to automate the retrieval and organization of information from Git repositories. It streamlines the process of searching, analyzing, and visualizing repositories, aiding users in better navigation and comprehension of repository data. The application is divided into two main components:

Frontend: Built with React and TypeScript, providing a user interface for interaction.

Backend: Developed using Node.js and Express, handling API requests and data processing.

# **Overview**

The Git Web Scraper is a web-based tool designed to streamline the discovery and exploration of GitHub repositories. Users can enter keywords to search for relevant projects, view detailed repository data—including descriptions, licenses, and owner info—and explore an interactive dependency tree that maps out each project’s components.

Following a client-server model, the tool’s backend fetches and categorizes data from GitHub based on user input, presenting it on a clean, responsive frontend. Each result includes direct links to GitHub, allowing seamless access to repositories for further exploration or contribution.

This tool provides an efficient way for developers, researchers, and enthusiasts to explore and engage with open-source projects on GitHub.

## **Theoretical background and definition of terms**

**Git Web Scraper Overview:**

A software tool designed to retrieve and categorize data from Git repositories based on user-specified keywords. The application provides a searchable interface and visualizes project relationships to assist users in identifying relevant Git repositories and their dependencies.

**Client-Server Model:**

The application operates on a client-server model where the client interface, accessed via a web browser, sends requests to the server. The server processes these requests, retrieves data from Git, and sends the results back to the client.

**Frontend and Backend:**

* **Frontend**: The user interface, accessible through a web browser, includes a search bar, keyword input, and visualizations, allowing users to manage and explore search results.
* **Backend**: The backend processes search requests, queries Git, manages data such as dependencies, and provides results for display on the frontend.

**Application Architecture:**

The tool’s architecture links the front-end search interface with backend Git data retrieval and database storage. The backend also processes dependency visualizations, enhancing scalability and maintainability.

**RESTful API Design:**

REST APIs connect the frontend search functionality with backend data-processing modules, facilitating smooth data flow for displaying repositories, descriptions, dependencies, and project details.

**User Experience (UX) and User Interface (UI) Design:**

A clean, minimalistic UI offers easy navigation through keyword searches and project details, while UX design focuses on simplifying the search and exploration of relevant Git repositories.

**Responsive Web Design:**

The front end is responsive, ensuring accessibility on various devices and screen sizes, making it suitable for mobile, tablet, and desktop users.

**Project Definitions:**

* **Controllers**: Defines REST API endpoints to manage user searches, queries, and results.
* **Services**: Collects data from the controller and processes it according to business logic, including handling dependency tree visualizations and scoring.

This structure offers a comprehensive, user-friendly tool for locating and exploring Git repositories based on user-defined criteria.

Figure 1 shows the overall architecture of the project and the dashed lines are implemented by routing.

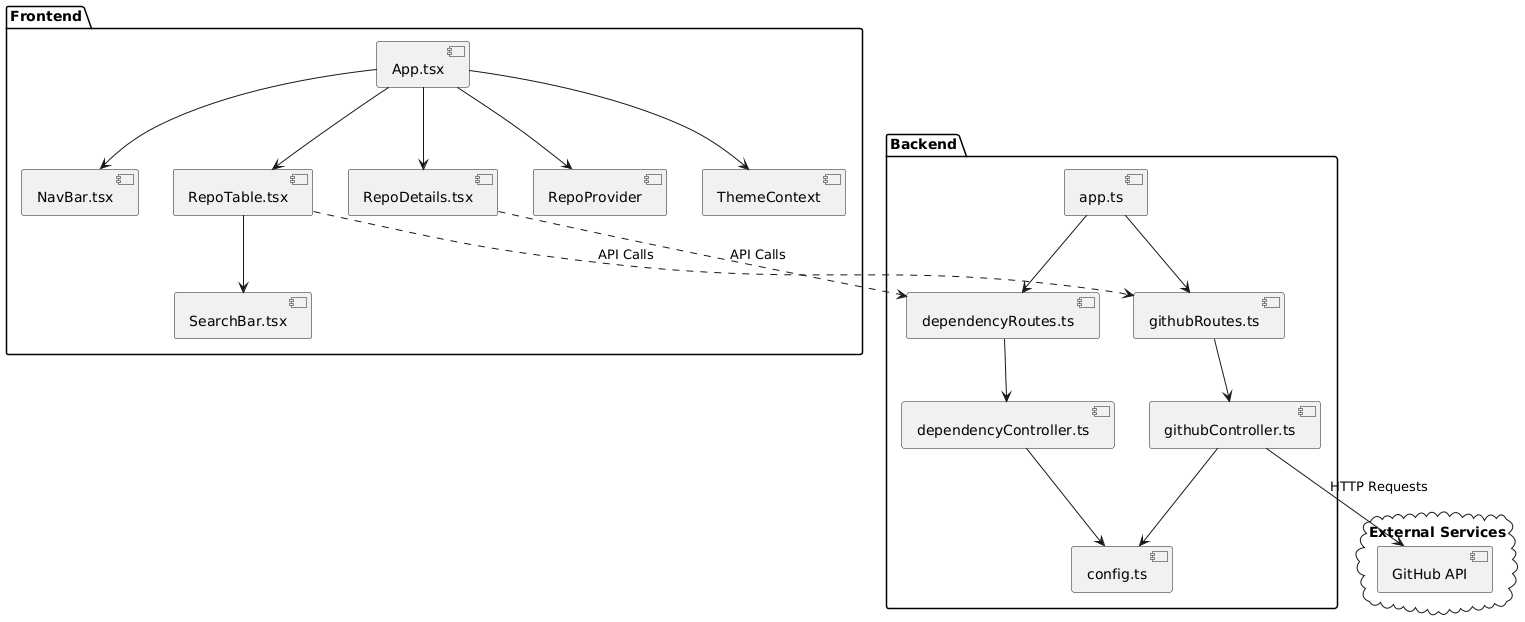


Figure Software Architecture

# **Software Architecture**

# **Software Bill of Materials (SBOM)**

The Software Bill of Materials (SBOM) provides a comprehensive list of all third-party components used in the project, their versions, usage within the system, and the sources from where they were retrieved. This SBOM aids in tracking dependencies, managing licenses, and identifying potential security vulnerabilities.

**Frontend Dependencies**

The front end of the application is built using React and TypeScript. It leverages several third-party libraries to enhance functionality and improve user experience.

The table with **full npm links** as the names:

Table : Frontend dependencies and versions and dates

|  |  |  |  |
| --- | --- | --- | --- |
| Package Name | Version | Release Date | npm Link |
| @mui/icons-material 6.1.2 | 6.1.2 | October 17, 2023 | <https://www.npmjs.com/package/@mui/icons-material/v/6.1.2> |
| @mui/material 6.1.1 | 6.1.1 | October 17, 2023 | <https://www.npmjs.com/package/@mui/material/v/6.1.1> |
| material-react-table 3.0.1 | 3.0.1 | October 4, 2023 | <https://www.npmjs.com/package/material-react-table/v/3.0.1> |
| react 18.2.0 | 18.2.0 | June 14, 2023 | <https://www.npmjs.com/package/react/v/18.2.0> |
| react-dom 18.2.0 | 18.2.0 | June 14, 2023 | <https://www.npmjs.com/package/react-dom/v/18.2.0> |
| react-router-dom 6.26.2 | 6.26.2 | October 17, 2023 | <https://www.npmjs.com/package/react-router-dom/v/6.26.2> |
| @types/react 18.3.10 | 18.3.10 | October 17, 2023 | <https://www.npmjs.com/package/@types/react/v/18.3.10> |
| @types/react-dom 18.3.0 | 18.3.0 | October 17, 2023 | <https://www.npmjs.com/package/@types/react-dom/v/18.3.0> |
| @typescript-eslint/eslint-plugin 7.1.1 | 7.1.1 | October 17, 2023 | <https://www.npmjs.com/package/@typescript-eslint/eslint-plugin/v/7.1.1> |
| @typescript-eslint/parser 7.1.1 | 7.1.1 | October 17, 2023 | <https://www.npmjs.com/package/@typescript-eslint/parser/v/7.1.1> |
| @vitejs/plugin-react 4.2.1 | 4.2.1 | October 18, 2023 | <https://www.npmjs.com/package/@vitejs/plugin-react/v/4.2.1> |
| eslint 8.57.0 | 8.57.0 | October 17, 2023 | <https://www.npmjs.com/package/eslint/v/8.57.0> |
| eslint-plugin-react-hooks 4.6.0 | 4.6.0 | October 17, 2023 | <https://www.npmjs.com/package/eslint-plugin-react-hooks/v/4.6.0> |
| eslint-plugin-react-refresh 0.4.5 | 0.4.5 | October 17, 2023 | <https://www.npmjs.com/package/eslint-plugin-react-refresh/v/0.4.5> |
| typescript 5.2.2 | 5.2.2 | October 17, 2023 | <https://www.npmjs.com/package/typescript/v/5.2.2> |
| vite 5.1.6 | 5.1.6 | October 25, 2023 | <https://www.npmjs.com/package/vite/v/5.1.6> |

**Backend Dependencies**

The backend is developed using Node.js and Express, facilitating API interactions and data processing.

Table : Backend dependencies and dates and their links

|  |  |  |  |
| --- | --- | --- | --- |
| Package Name | Version | Release Date | npm Link |
| axios 1.7.7 | 1.7.7 | October 18, 2023 | <https://www.npmjs.com/package/axios/v/1.7.7> |
| cors 2.8.5 | 2.8.5 | October 18, 2023 | <https://www.npmjs.com/package/cors/v/2.8.5> |
| dotenv 16.4.5 | 16.4.5 | October 18, 2023 | <https://www.npmjs.com/package/dotenv/v/16.4.5> |
| express 4.21.0 | 4.21.0 | October 24, 2023 | <https://www.npmjs.com/package/express/v/4.21.0> |
| mongoose 8.6.3 | 8.6.3 | October 11, 2023 | <https://www.npmjs.com/package/mongoose/v/8.6.3> |
| puppeteer 23.8.0 | 23.8.0 | October 18, 2023 | <https://www.npmjs.com/package/puppeteer/v/23.8.0> |
| @types/cors 2.8.17 | 2.8.17 | October 18, 2023 | <https://www.npmjs.com/package/@types/cors/v/2.8.17> |
| @types/express 4.17.21 | 4.17.21 | October 18, 2023 | <https://www.npmjs.com/package/@types/express/v/4.17.21> |
| @types/mongoose 5.11.97 | 5.11.97 | October 18, 2023 | <https://www.npmjs.com/package/@types/mongoose/v/5.11.97> |
| @types/node 22.6.0 | 22.6.0 | October 18, 2023 | <https://www.npmjs.com/package/@types/node/v/22.6.0> |
| nodemon 3.1.7 | 3.1.7 | October 18, 2023 | <https://www.npmjs.com/package/nodemon/v/3.1.7> |
| ts-node 10.9.2 | 10.9.2 | October 18, 2023 | <https://www.npmjs.com/package/ts-node/v/10.9.2> |
| typescript 5.6.2 | 5.6.2 | October 18, 2023 | <https://www.npmjs.com/package/typescript/v/5.6.2> |

## **File Structure**

A screenshot of a computer screen

Description automatically generated  
 **Class structure**

Class Diagram UML for the frontend.

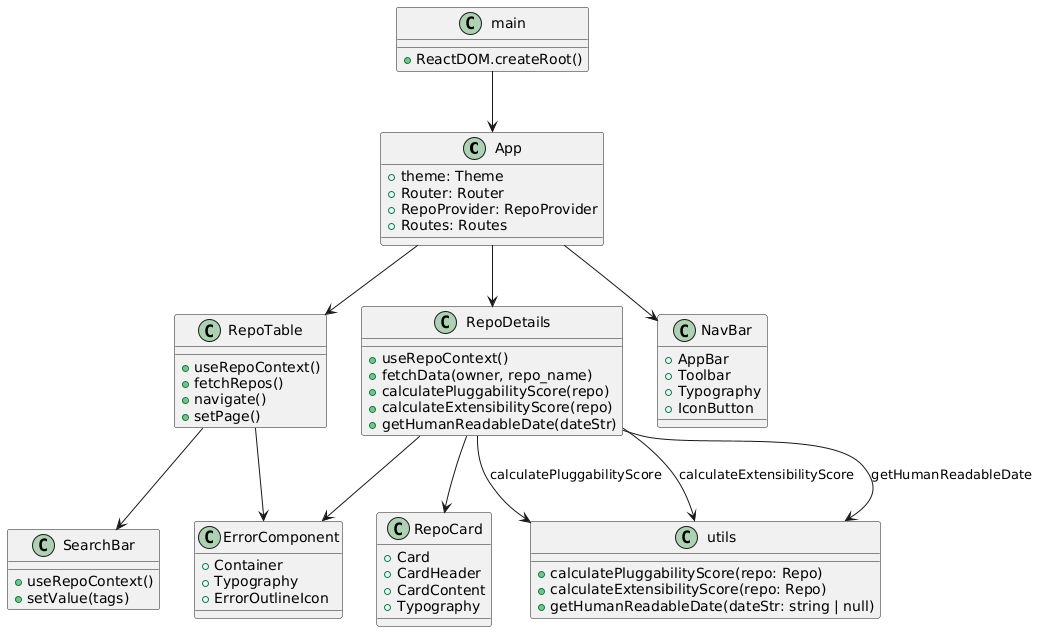


Figure 2 UML class diagram for Frontend of the application

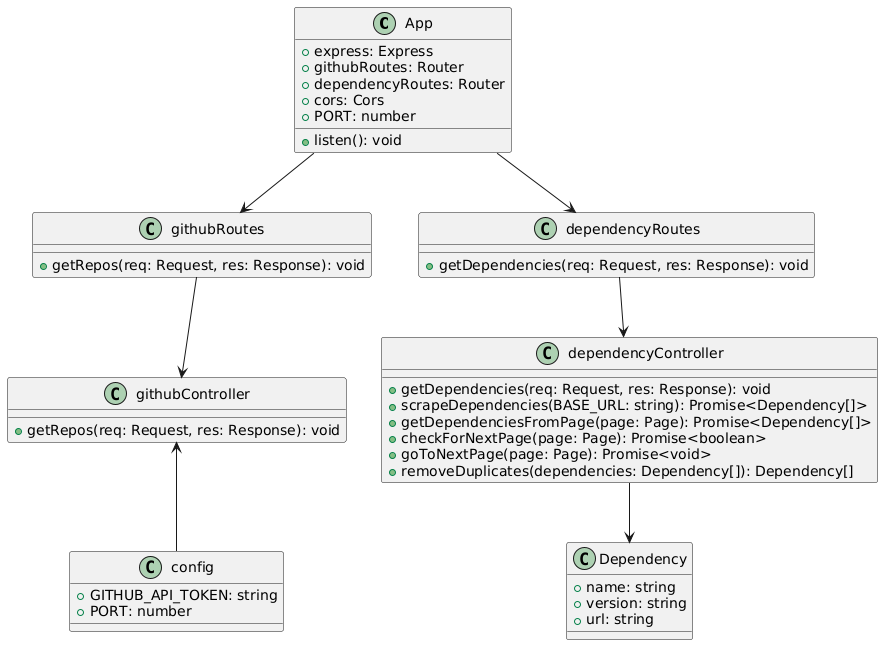


Figure 3 UML class diagram for backend

# **Data Design**

## **Object Structure**

**1. State Variables and Management**

* **Global State:** Managed using React Context (RepoProvider, ThemeProvider).
* **Local State:** Managed within components using the useState hook.
* **Variables:**
  + **repos:** Array of repository objects.
  + **loading:** Indicates data fetching status.
  + **keywords:** List of search keywords entered by the user.

**2. Data Flow Between Components**

* **Data Fetching:** Triggered in RepoProvider based on keywords, using fetchRepos function.
* **Context Sharing:** Components consume context data (repos, loading, keywords).
* **User Interaction:** SearchBar updates keywords, which in turn refreshes the repository data displayed in RepoTable.

## **Classes**

**1. Repo Class**

* **Properties:**
  + id: Unique identifier.
  + full\_name: Full repository name.
  + description: Brief description.
  + stargazers\_count: Number of stars.
  + Forks: Number of forks
  + owner: A User object representing the repository owner.
  + updated\_at: Last update timestamp.
  + license: A License object.
  + html\_url: URL to the repository.

## **Inter-Process Communication**

**1. Communication Between Frontend and Backend**

* **HTTP Requests:** The frontend communicates with the backend via HTTP calls using the fetch API or Axios.
* **API Endpoints**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Frontend File | Function/Component | Request Type | Endpoint | Description |
| [useRepoContext.tsx](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | [fetchRepos](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | GET | /api?keywords=${value.join(",")}&page=${page} | Fetches repositories based on keywords and page number |
| [RepoDetails.tsx](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | [fetchData](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | GET | /scrape-dependencies/${owner}/${repo\_name} | Fetches dependencies for a specific repository |

**2. Backend Interaction with External Services**

* **Axios Requests:** The backend uses Axios to fetch data from external GitHub PI.
* **Endpoint Handling:** Receives requests from the frontend, processes them, and forwards data accordingly.

|  |  |  |  |
| --- | --- | --- | --- |
| Backend File | FunctionComponent | Request Type | Endpoint |
| [githubController.ts](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | getRepos | GET | https://api.github.com/search/repositories?q=${query}+language:C&per\_page=20&page=${page}&sort=updated&order=desc |
| [dependencyController.ts](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | scrapeDependencies | GET | https://github.com/${owner}/${repoName}/network/dependencies |

# **Software Functionality**

## **Use Cases**

**Use Case 1: Keyword-Based Repository Search:**

**User Action:** Enters keywords into the search bar and clicks "Search."14

**System Outcome:**

The frontend sends a request to the backend API with the provided keywords.

The backend queries the GitHub API, filtering for repositories matching the keywords.

The backend returns a list of matching repositories to the frontend.

The frontend displays the results (likely using RepoTable with pagination).

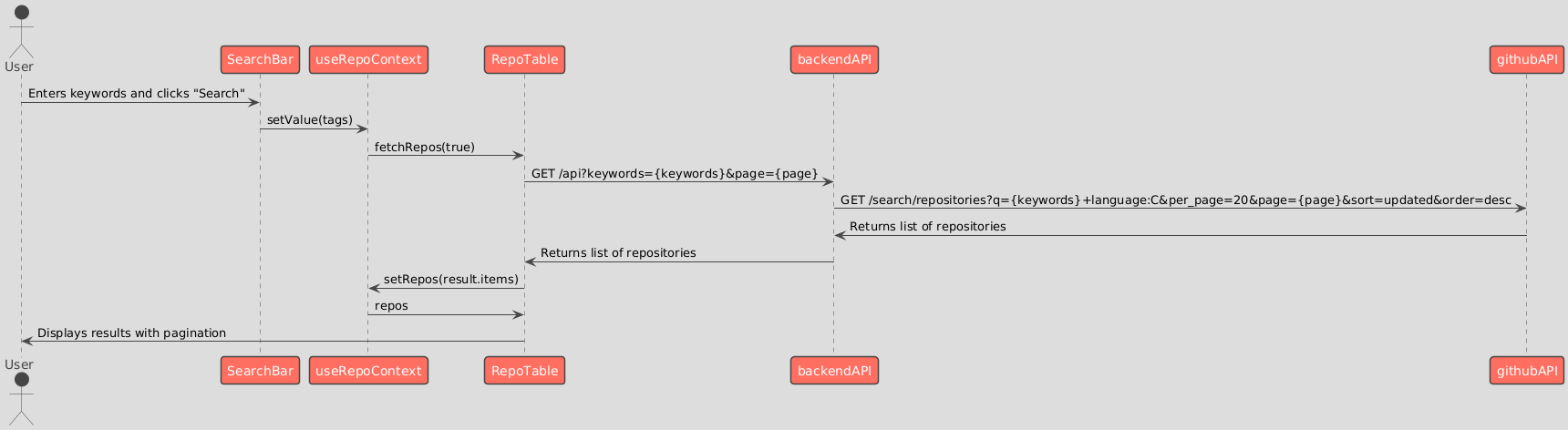


Figure 4 Sequence diagram for use case - 1

**Use Case 2: Pagination for Navigating Search Results:**

**User Action:** Clicks pagination controls (e.g., "Next Page") to view more results.

**System Outcome:**

Frontend updates the page number in its request to the backend.

Backend fetches the corresponding page of results from the GitHub API.

Frontend updates the displayed results accordingly.

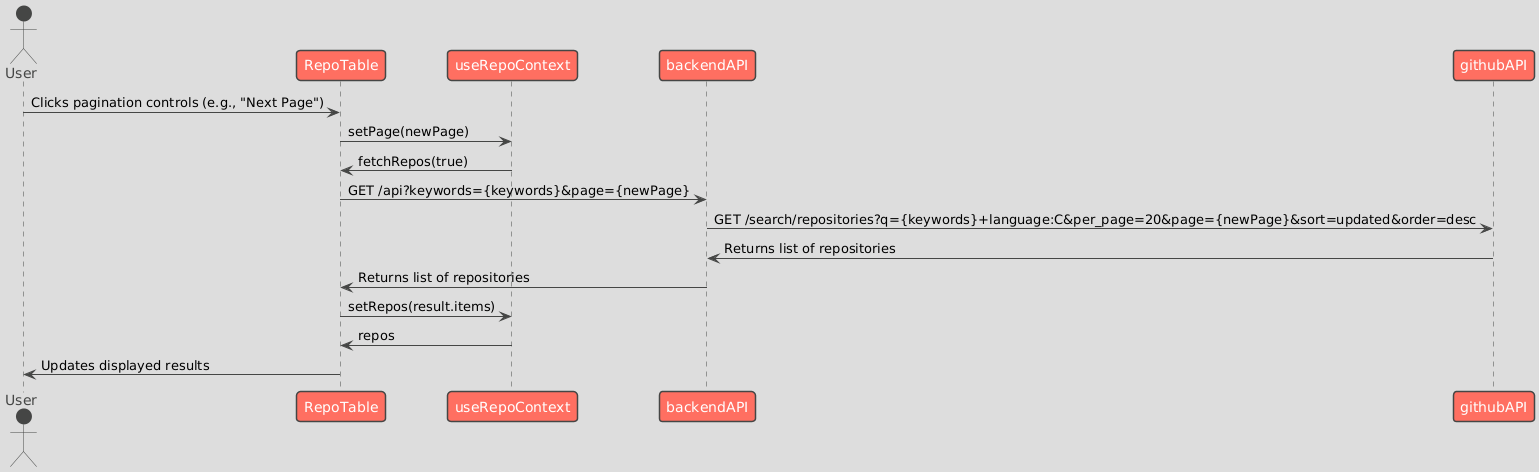


Figure 5 Sequence diagram for use case - 2

**Use Case 3: Viewing Detailed Repository Information:**

**User Action:** Clicks on a specific repository from the search results list.

**System Outcome:**

Frontend navigates to a new page or section displaying detailed repository information.

Data displayed likely includes:

Repository name, description, owner info, license details, star count, fork count.

Potentially includes a link to the repository on GitHub, the last updated timestamp, and more.

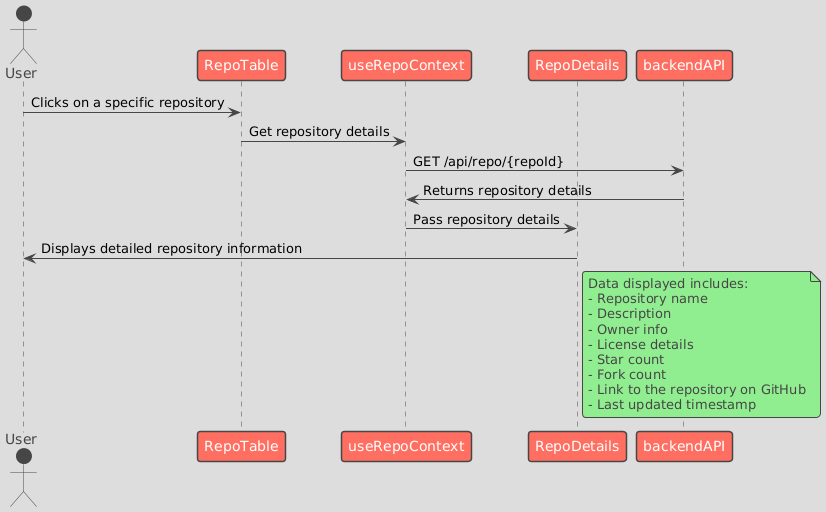


Figure 6 Sequence diagram for use case - 3

**Use Case 4: Expanding the Dependency Tree and Counting Dependencies:**

**User Action:** Interacts with a visual dependency tree representation (likely by expanding nodes).

**System Outcome:**

Frontend dynamically updates the dependency tree visualization as the user expands nodes.

Application displays the total number of visible dependencies, providing a measure of project complexity.

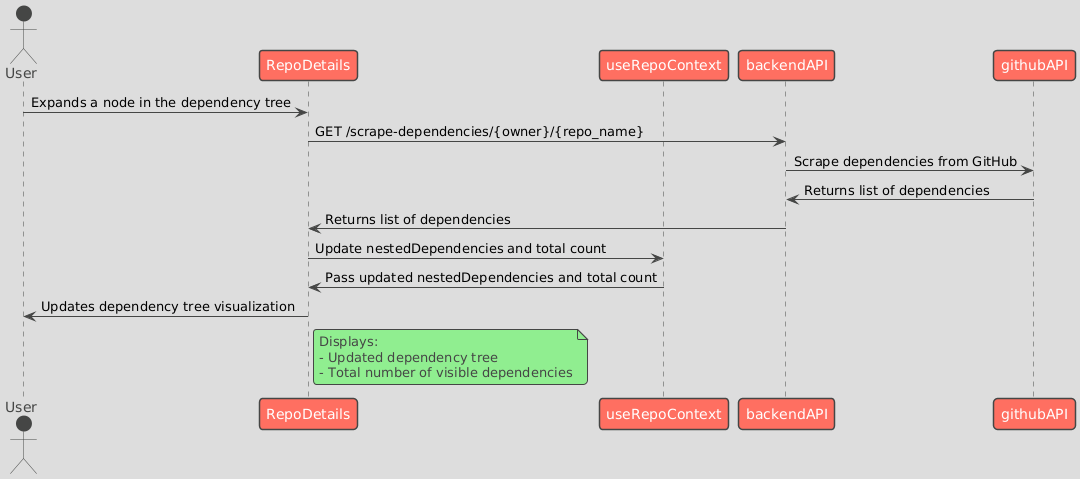


Figure 7 Sequence diagram for use case - 4

## **Function Execution**

**Functions and Their Usage in Components**

**1. fetchData**

**File:** RepoDetails.tsx

**Function:**

const fetchData = async (owner: string, repo\_name: string) => {

const response = await fetch(

`http://localhost:3000/scrape-dependencies/${owner}/${repo\_name}`

);

if (!response.ok) {

throw new Error("Network response was not ok");

}

return response.json();

};

**Description:**

1. This function fetches the dependencies of a given repository from the backend API.
2. It takes owner and repo\_name as parameters to construct the API endpoint.
3. It returns the JSON response containing the dependencies.

**Used In:**

1. **Component:** RepoDetails
2. **Method:** handleAccordionClick
3. **Purpose:** To fetch and display nested dependencies when a user expands a node in the dependency tree.

**2. calculatePluggabilityScore**

**File:** utils.ts

**Function:**

export const calculatePluggabilityScore = (repo: Repo) => {

let score = 0;

if ((repo.stars ?? 0) > 1000) {

score += 60;

} else if ((repo.stars ?? 0) > 100) {

score += 40;

} else if ((repo.stars ?? 0) > 10) {

score += 20;

}

if ((repo.forks ?? 0) > 1000) {

score += 15;

} else if ((repo.forks ?? 0) > 100) {

score += 10;

} else if ((repo.forks ?? 0) > 10) {

score += 5;

}

const description = repo.description ? repo.description.toLowerCase() : "";

if (description.includes("plugin")) {

score += 5;

}

if (description.includes("api")) {

score += 5;

}

if (description.includes("module")) {

score += 5;

}

if (description.includes("framework")) {

score += 5;

}

if (repo.is\_template) {

score += 5;

}

return score;

};

**Description:**

1. This function calculates the pluggability score of a repository based on its stars, forks, and keywords in the description.
2. It returns the calculated score.

**Used In:**

1. **Component:** RepoDetails
2. **Method:** useEffect
3. **Purpose:** To calculate and display the pluggability score of the repository.

**3. calculateExtensibilityScore**

**File:** utils.ts

**Function:**

export const calculateExtensibilityScore = (repo: Repo) => {

let score = 0;

if ((repo.stars ?? 0) > 1000) {

score += 15;

} else if ((repo.stars ?? 0) > 100) {

score += 10;

} else if ((repo.stars ?? 0) > 10) {

score += 5;

}

if ((repo.forks ?? 0) > 1000) {

score += 55;

} else if ((repo.forks ?? 0) > 100) {

score += 40;

} else if ((repo.forks ?? 0) > 10) {

score += 20;

}

const description = repo.description ? repo.description.toLowerCase() : "";

if (description.includes("extensible")) {

score += 5;

}

if (description.includes("extension")) {

score += 5;

}

if (description.includes("modular")) {

score += 5;

}

if (description.includes("framework")) {

score += 5;

}

return score;

};

**Description:**

1. This function calculates the extensibility score of a repository based on its stars, forks, and keywords in the description.
2. It returns the calculated score.

**Used In:**

1. **Component:** RepoDetails
2. **Method:** useEffect
3. **Purpose:** To calculate and display the extensibility score of the repository.

**4. getHumanReadableDate**

**File:** utils.ts

**Function:**

export function getHumanReadableDate(dateStr: string | null): string {

if (!dateStr) return "N/A";

const date = new Date(dateStr);

return date.toLocaleDateString("en-US", {

year: "numeric",

month: "long",

day: "numeric",

});

}

**Description:**

1. This function converts a date string into a human-readable date format.
2. It returns the formatted date string.

**Used In:**

1. **Component:** RepoDetails
2. **Method:** render
3. **Purpose:** To display the last updated date of the repository.
4. **Component:** RepoTable
5. **Method:** columns
6. **Purpose:** To display the last updated date of the repository in the table.

**5. fetchRepos**

**File:** useRepoContext.tsx

**Function:**

const fetchRepos = (clearRepos: boolean = false) => {

if (!navigator.onLine) {

setError(true);

setErrorMessage("No internet connection");

return;

}

setError(false);

setLoading(true);

fetch(

`http://localhost:3000/api?keywords=${value.join(",")}&page=${page}`,

{

method: "GET",

headers: {

"Content-Type": "application/json",

},

signal: signal,

}

)

.then((response) => {

if (response.ok) {

response.json().then((result) => {

console.log(result);

setTotalCount(result.total\_count);

if (clearRepos) {

setRepos(result.items);

} else {

setRepos((prevRepos) => [...prevRepos, ...result.items]);

}

setLoading(false);

});

} else {

throw new Error("Failed to fetch");

}

})

.catch((error) => {

console.error("Error:", error);

setLoading(false);

if (error.name === "AbortError") {

return;

}

setError(true);

setErrorMessage("Failed to fetch");

});

};

**Description:**

1. This function fetches repositories based on the provided keywords and page number.
2. It updates the state with the fetched repositories and total count.

**Used In:**

1. **Hook:** useRepoContext
2. **Purpose:** To fetch and manage the state of repositories in the context.

**6. handleAccordionClick**

**File:** RepoDetails.tsx

**Function:**

const handleAccordionClick = async (dep: Dependency, level: number) => {

const urlSplit = dep.url.split("/");

const owner = urlSplit[3];

const repo\_name = urlSplit[4];

if (owner === "N/A" || repo\_name === "N/A") {

return;

}

const key = `${dep.name}-${level}`;

if (!nestedDependencies[key]) {

setLoadingStates((prev) => ({ ...prev, [key]: true }));

try {

const result = await fetchData(owner, repo\_name);

setNestedDependencies((prev) => ({

...prev,

[key]: result.dependencies,

}));

setTotal((prevTotal) => prevTotal + result.dependencies.length);

} catch (error) {

console.error("Error fetching nested dependencies:", error);

} finally {

setLoadingStates((prev) => ({ ...prev, [key]: false }));

}

}

};

**Description:**

1. This function handles the click event on an accordion to fetch and display nested dependencies.
2. It updates the state with the fetched nested dependencies.

**Used In:**

1. **Component:** RepoDetails
2. **Method:** renderDependencies
3. **Purpose:** To fetch and display nested dependencies when a user expands a node in the dependency tree.

**7. reducer**

**File:** RepoDetails.tsx

**Function:**

const reducer = (state: State, action: Action): State => {

switch (action.type) {

case "FETCH\_INIT":

return { ...state, isLoading: true, isError: false, error: null };

case "FETCH\_SUCCESS":

return { ...state, isLoading: false, data: action.payload };

case "FETCH\_FAILURE":

return {

...state,

isLoading: false,

isError: true,

error: action.payload,

};

default:

throw new Error();

}

};

**Description:**

1. This function is a reducer that manages the state of fetching dependencies.
2. It handles actions for initializing fetch, successful fetch, and fetch failure.

**Used In:**

1. **Component:** RepoDetails
2. **Purpose:** To manage the state of fetching dependencies using the useReducer hook.

**8. handleOpen**

**File:** SearchBar.tsx

**Function:**

const handleOpen = () => {

if (tags.length === 0) {

setError({

error: true,

helperText: "Please enter at least one keyword.",

});

return;

}

setValue(tags);

};

**Description:**

1. This function handles the click event on the "Search" button.
2. It sets the keywords in the context if there are any tags.

**Used In:**

1. **Component:** SearchBar
2. **Purpose:** To handle the search action and set the keywords in the context.

**9. handleTagChange**

**File:** SearchBar.tsx

**Function:**

const handleTagChange = (\_event: any, newValue: string[]) => {

if (newValue.length <= 5) {

setTags(newValue);

setError({ error: false, helperText: "" });

} else {

setError({

error: true,

helperText: "You can only add up to 5 keywords.",

});

}

};

**Description:**

1. This function handles the change event for the tags input.
2. It updates the tags state and sets an error if the number of tags exceeds 5.

**Used In:**

1. **Component:** SearchBar
2. **Purpose:** To handle the change event for the tags input and update the tags state.

**10. getHumanReadableDate**

**File:** utils.ts

**Function:**

export function getHumanReadableDate(dateStr: string | null): string {

if (!dateStr) return "N/A";

const date = new Date(dateStr);

return date.toLocaleDateString("en-US", {

year: "numeric",

month: "long",

day: "numeric",

});

}

**Description:**

1. This function converts a date string into a human-readable date format.
2. It returns the formatted date string.

**Used In:**

1. **Component:** RepoDetails
2. **Method:** render
3. **Purpose:** To display the last updated date of the repository.
4. **Component:** RepoTable
5. **Method:** columns
6. **Purpose:** To display the last updated date of the repository in the table.

This document provides a detailed explanation of each function, its purpose, and the components that use it in the frontend codebase.