GSOC 2024 PROPOSAL

by Sai Srikar Dumpeti

on

build a developer dashboard for tracking repository build status

TABLE OF CONTENTS

Introduction

Full Name: Sai Srikar Dumpeti

University status: Yes

University name: Birla Institute of Technology, Mesra

University program: Bachelor of Technology in Computer Science

Expected graduation: 2026

Short biography:

I am a 2nd year B.Tech. Student in the department of Computer Science and Technology at Birla Institute of Technology, Mesra. I started my programming journey from 2019 in High School i.e 12th Standard.

I am a self learnt programmer who likes to learn new things by creating new projects. Here are some programming languages and frameworks which I have learnt:

- Go
- Python
- JavaScript
- TypeScript
- Reactjs
- Svelte
- HTML/CSS
- Jest
- Stimulus

In College, I have completed many courses like ***Problem Solving with Programming in C***, ***Object Oriented Programming***, ***Data Structure and Algorithms*** and many more.
Currently attending courses of ***Database Management Systems***, ***Design Analysis of Algorithms***, ***Operating System***.

My Interests lie in **Web Development** and **creating new things**.

Timezone: Asia/Kolkata UTC +0530

Contact details:

- **Email:** saisrikardumpeti@gmail.com
- **GitHub:** https://github.com/the-r3aper7
- **Gitter:** https://matrix.to/#/@the-r3aper7:gitter.im

Platform: Linux (Fedora)

Editor:

For development I use **Fedora** as an Operating System and **VSCode** as my editor because it is simple to use, learning curve is less and comes with lots of built-in features like **emmet** features and extensions. However, I have learned the basics of **vim** for working on servers.

Programming experience:

I have significant knowledge of programming, when I first started to learn programming in high school where I started my first language with **Python** I was fascinated by it. Afterwards I started to explore various fields but Web Development caught my interest. I have created some projects using **Django** Framework. Then, I started to learn **JavaScript** so that I can learn **React** for the front-end. Then I started contributing to **Open Source** to learn new things and it was great for exploration. I have learned a lot from Open Source and met a lot of amazing people.

JavaScript experience:

I've been having a great time learning JavaScript and working with it on front-end projects in open source environments. The best part about **JavaScript** is that you can literally build anything with it, from mobile apps with frameworks like **React Native** to web apps, games, and much more. Recently, I even came across a project where they built a whole operating system in the browser called **OS.js**

Problems with JavaScript is that it has loose typing which can lead to problems if you don't use tools like **TypeScript** to add type safety. While **JSDoc** can help with documentation, it doesn't enforce types.However, there is a RFC going on where we can Static Types in JavaScript in Future. Another challenge is ***callback hell*** where nested callbacks can make code difficult to read and hard to maintain.

Here are some of my works with **JavaScript**:

- Checkout my personal site which I created while I was learning **Svelte** link to Repo.
- Checkout Next.js file server which I created while I was learning `Next.js`.
- Revamped the media page of Qwik checkout the PR.
- Small Changes to Sidebar to Qwik checkout the PR.
- Another Small Improvement in Qwik's Media Page checkout the PR.
- Here is a Achieve Chess Academy Website which I created for a client which was needed in 24hrs.

Node.js experience

I haven't directly worked with **Node.js** yet, but I did use the API features of Next.js v12 to create a file server interface for a local network. This project allowed for basic upload, download, and streaming functionalities. Check it out on GitHub here: Next.js File Server

I'm definitely interested in getting more hands-on experience with **Node.js** in the future.

C/Fortran experience

While I haven't had the chance to create a full project in C yet, I did take a course in **Programming for Problem Solving in C** during college. This course involved implementing basic algorithms in C, which provided a strong foundation in the language's core concepts.

Interest in stdlib

While I am still developing my knowledge of statistics and machine learning algorithms, I have been impressed by the stdlib-js/stdlib library. It offers a collection of smaller, focused packages instead of a single, massive library that could bloat your bundle size. What particularly excites me is the ability to perform NumPy-like operations directly in the browser. The plotting API seems like a fantastic example of this functionality. I have also found the stdlib community to be very welcoming and approachable.

Version control

Yes, Currently I am using Git as primary VCS.

Contributions to stdlib

Here are the statuses of some PR's which I have authored.

Status Merged

All PRs which are merged

- feat: added utils/every-in-by
- feat: add math/base/special/acscd
- feat: add math/base/special/asecd
- feat: add math/base/special/secd
- feat: add math/base/special/cscd
- feat: add math/base/special/acotd
- feat: add math/base/special/atand
- feat: add math/base/special/acosd
- feat: add math/base/special/asind

- feat: add math/base/special/cotd
- feat: add math/base/special/tand
- feat: add math/base/special/cosd

Status Opened

All Prs which are opened

- docs(added readme examples for random/iter): added examples

Issues Opened
All issues which are currently opened

- none

Goals

Abstract

Stdlib is a Open Source Project written in **JavaScript**, Stdlib provides 3500+ high performance packages which can be used for numerical and scientific computation, statistics. Since it is written in **JavaScript** we can use it anywhere on mobile via **react-native**, **Node.js** and **Browsers**. My proposal is to create a developer dashboard for tracking build status of those 3500+ repositories. We may want to achieve this using **Reactjs** and **Fastify** for backend which will serve static files and API responses.

Description for Project

For my completing my project I am going to divide it into 2 parts:

- Roadmap to Status Dashboard.
- Roadmap to Historical overviews and per package metrics.

Here are some libraries and tools which will be useful in this project.

- Vitejs is a great build tool as it provides various features like **code-splitting**, **optimized builds** and **faster workflows**.
- Reactjs is a popular framework for building websites. Since we are in the initial stages we may not want to bloat the application with various technologies. Let's go simple and clear. We will be using Class Based Components as it is familiar to the community and documentation sites are built with it.
- postgres we will use this library to query data from the database directly in **JSON** format which can be easily used for formatting and sending it to the **frontend** for rendering.
- Fastify as the name suggests is fast and less overhead for creating REST API's. Fastify Ecosystem provides lots of plugins which will make DX easier and simpler. Since, we will be serving our react app from the same port as **Fastify** this will avoid the process of **CORS**.

Roadmap to Status Dashboard

- Here are some designs which I am thinking of for the Status Dashboard.

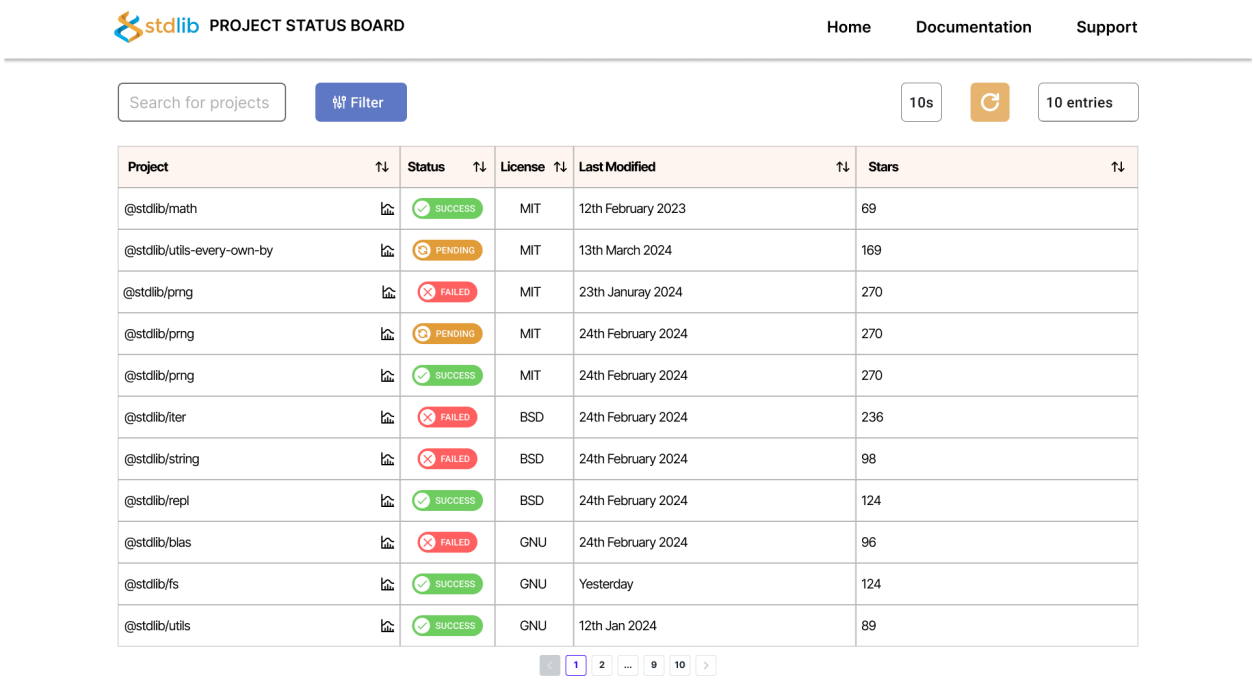


Fig 1.1: Index Page

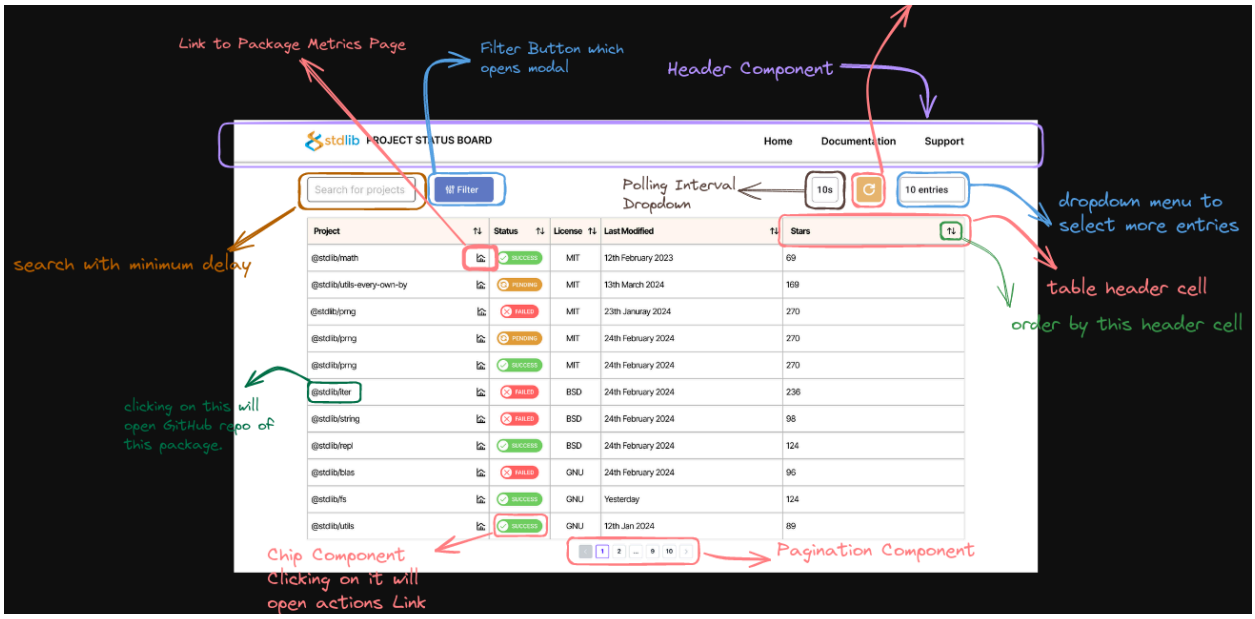


Fig 1.2: Explanation of Index Page

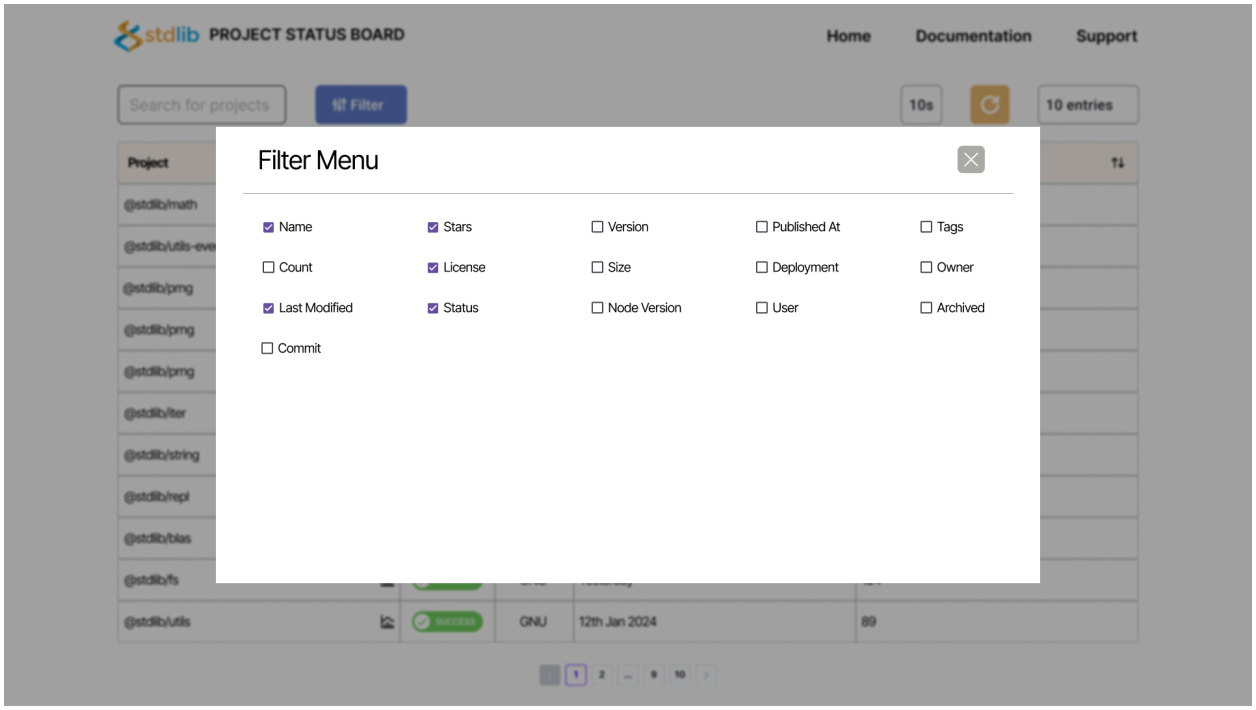


Fig 1.3: when filter button is clicked

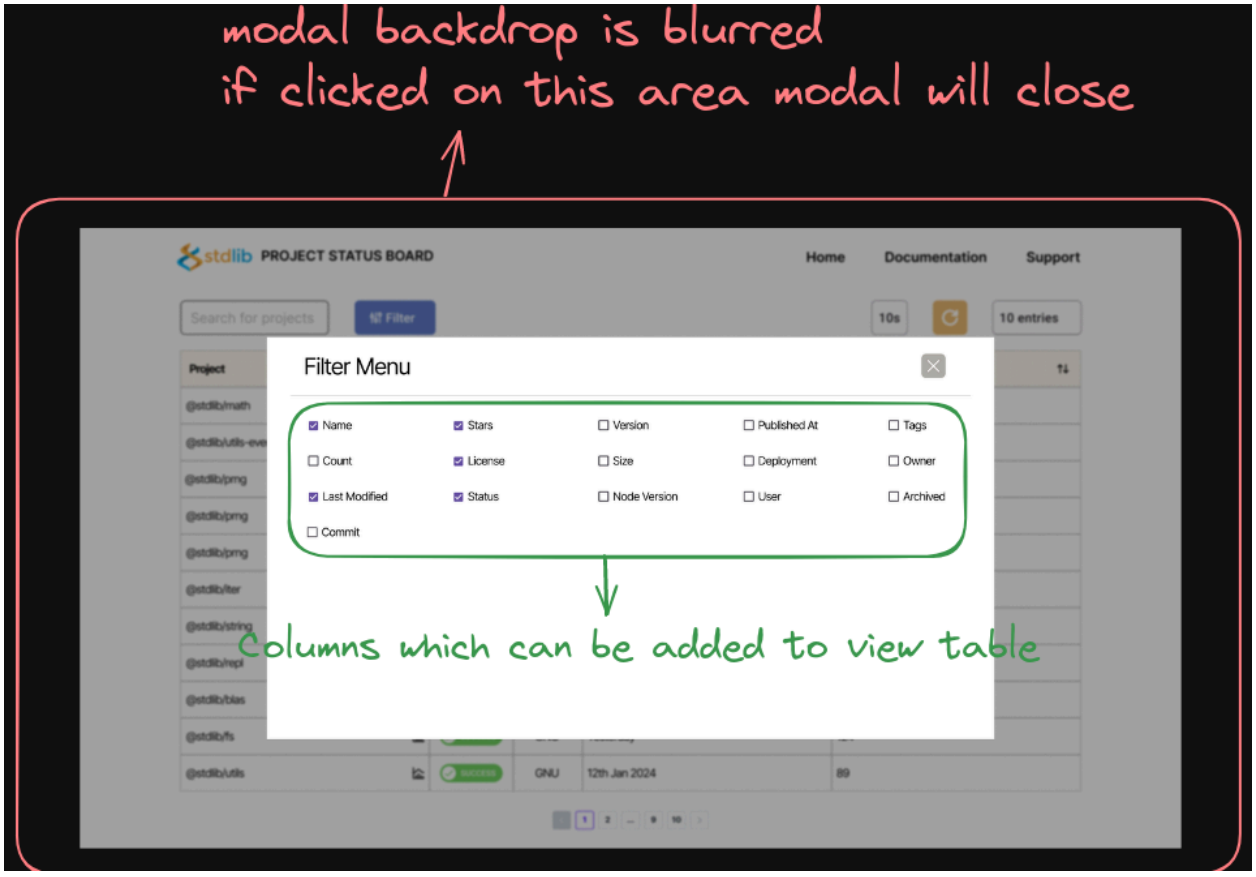


Fig 1.4. Explanation of when filter button is clicked

Fig 1.5. when data is fetching/loading.

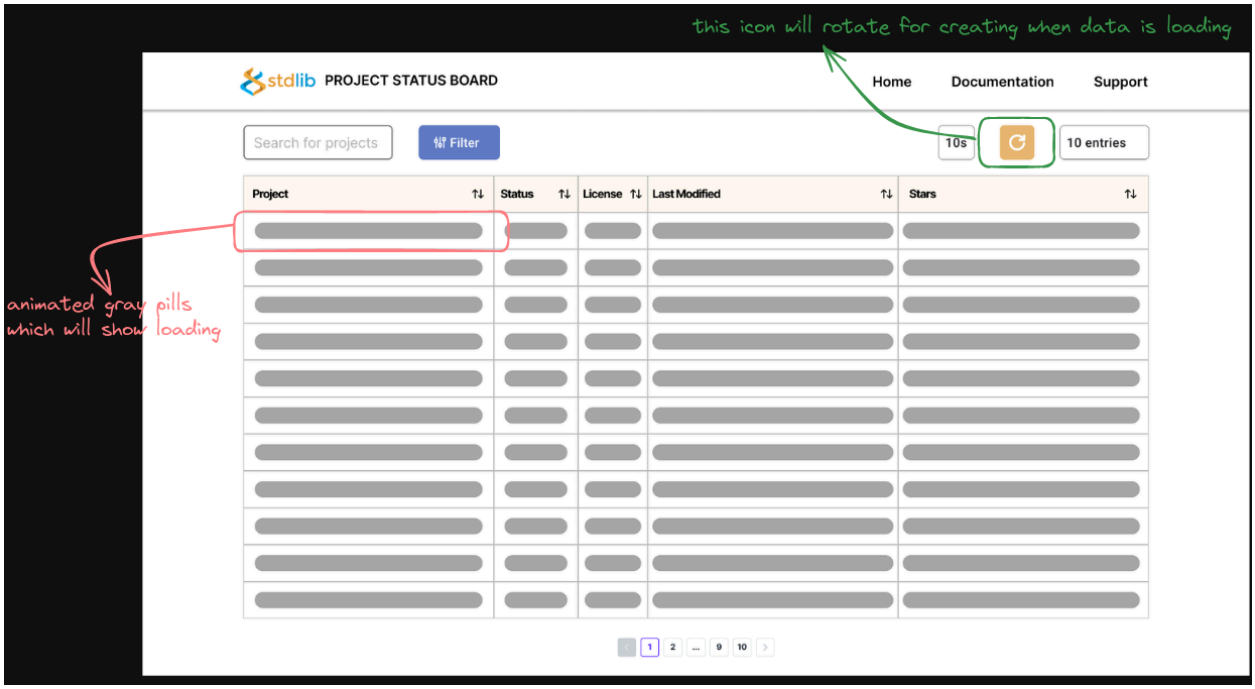


Fig 1.6. Explanation of when data is fetching/loading.

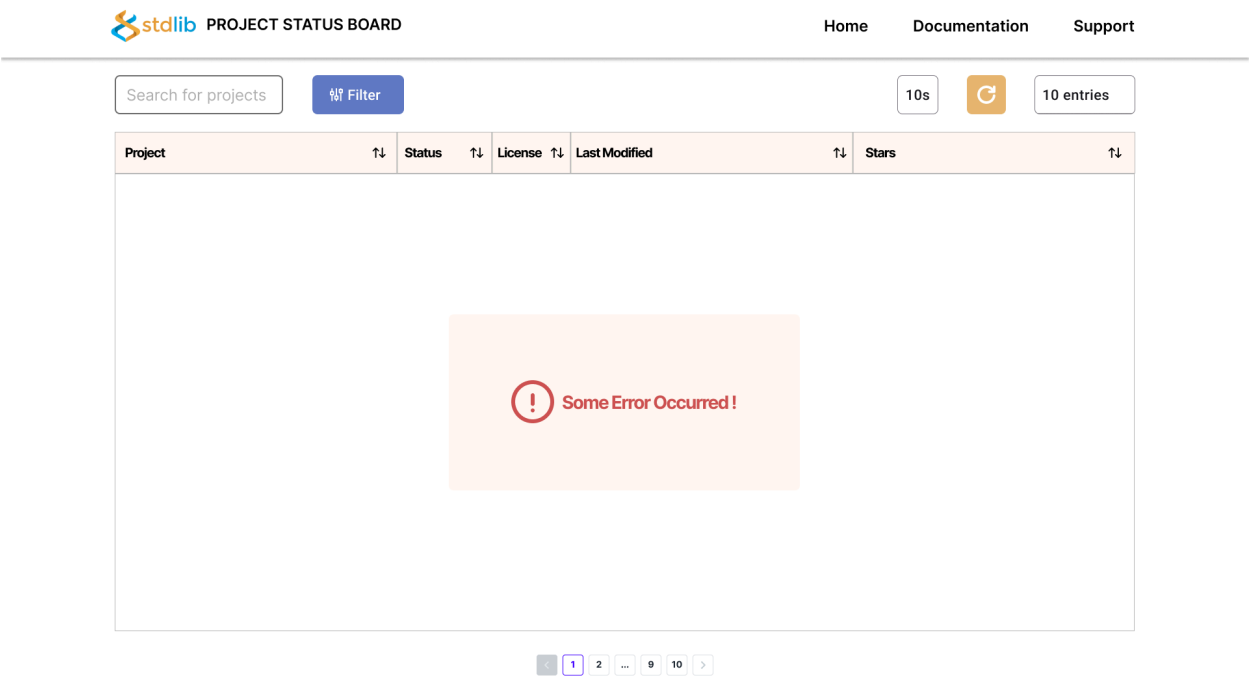


Fig 1.7. when error occurs while data fetching

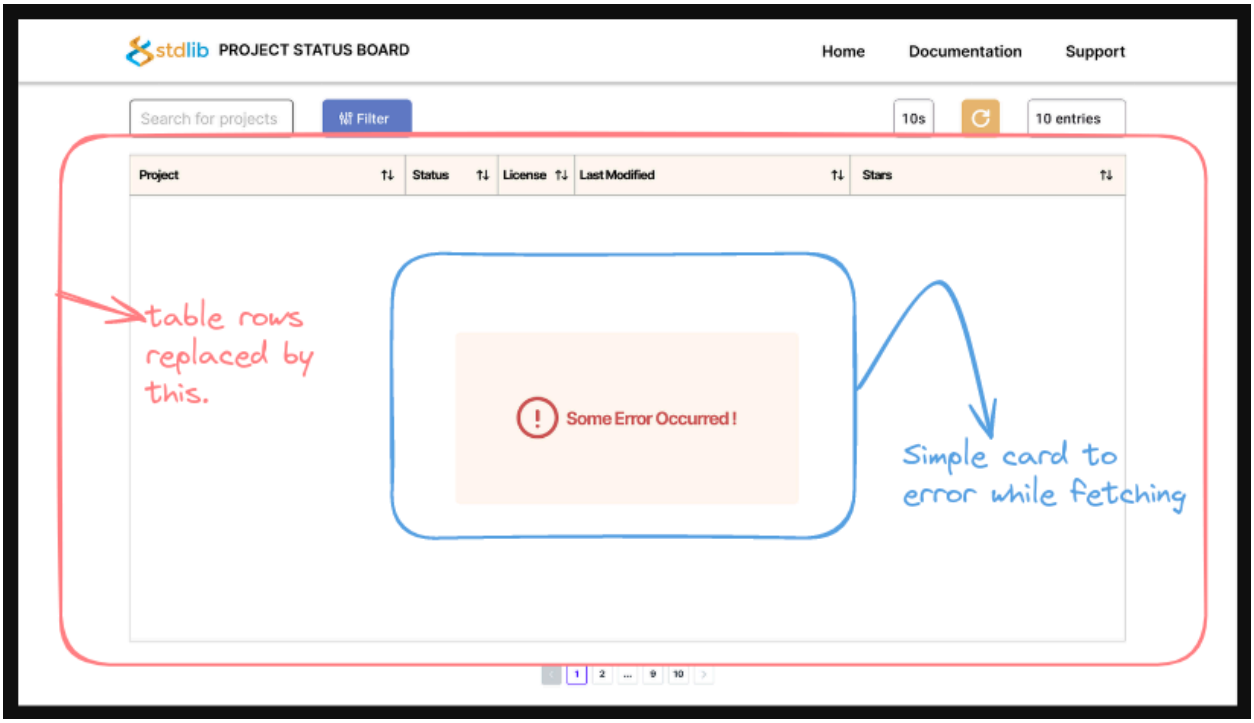
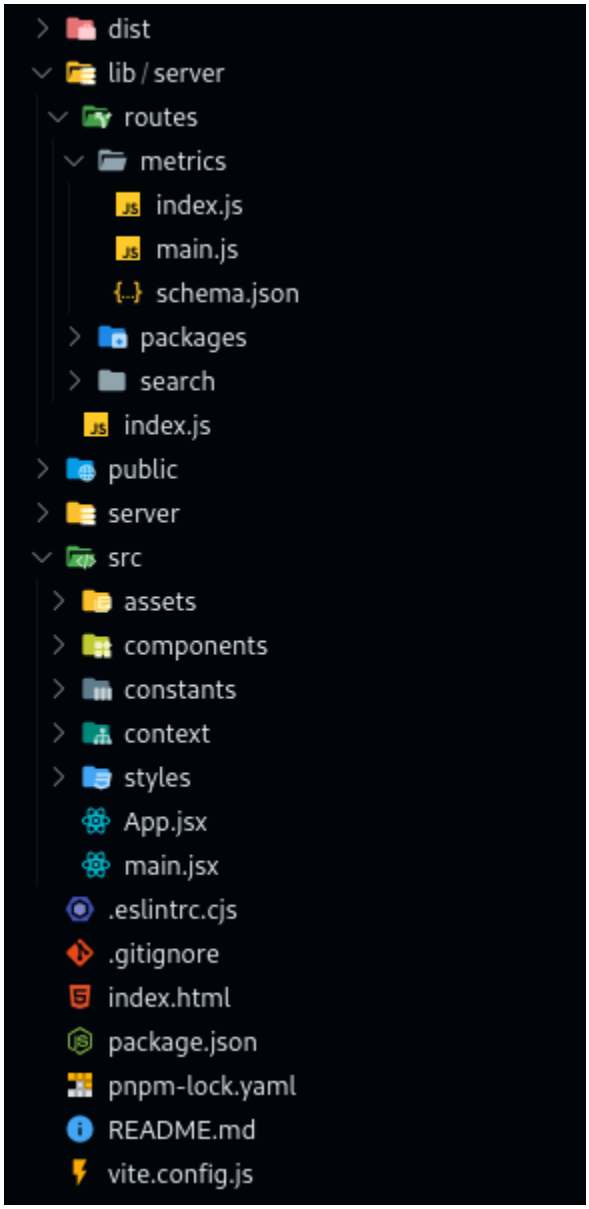


Fig 1.8. Explanation of when error occurs while data fetching

Links for the above designs:

1. Prototype of the websitePro
2. Designs of the website pages and instances
3. Designs of the website components
4. Designs of the website table

- This may be the project structure for our frontend part of the website.



Components: from the above design we may use these components initially.

1. Table.jsx
2. Header.jsx
3. Chip.jsx
4. Icon.jsx
5. Card.jsx
6. Modal.jsx

Context: Since, we are making it leaner and simple we may want to use **Reactjs Context API** to avoid prop drilling. We may store recently fetched data, search query and as project requirements while developing.
Constants: we will store all constants like **API** urls that may be removed as per suggestion.

- Communication with APIs

  Since, we are seeking the project to be as simple as possible so we will use the Fetch API which is used by browsers natively for communication. We can create a Component for rendering table data.

  Checkout the example component which fetched the data from an API.

  NOTE! This is a mockup component for the proposal that might change in the real project.

  Now, we easily render table data with changing urls and store the api response to a context as it would be easy to modify the data when a user clicks on **Sort Icon** as this would be deeply nested in the component tree.

- Implementation of REST API
  - */* (*GET*): for serving the **index.html** file**.**
  - **/assets** (*GET*): for serving static files like css, js, svgs etc.
  - **/search** (*GET*): this route serves **JSON** data for a given query.

    Here are some search query parameters which it will take.
    - **q** (*required*) (*type: string*) for querying by the package-name.

      Here are some DX improving shortcuts to search.

      Basic Syntax will **[header cell/filter type]=[...]**
      1. if type is string then use **^**.
      2. if type is number then use **-** (for range)**, >=, >, <=, <**.

      Initially when you type into the search bar it will search for a package which contains the package name in SQL which will look like this *select packagename from packages packagename like '%q%';* **(NOTE JUST A MOCK QUERY)**.

      Suppose you want to search for a package.
      - Starting with a given query you can type **name=^name-of-package**.
      - Ends with a given query you can type **name=name-of-package^**

- If you type **name=name-of-package** this will be treated as default.
- Examples: **name=^iter**

Suppose you want to filter your searches based on stars which is a number.

- Range of star you can type **stars=1-10**
- Less than a specific number you can type **stars=<10**
- Greater than or Equal a specific number you can type **stars=<=10**

You can also combine multiple shortcuts which are delimiter by ' ' (*space*) will look like this.

- stars which range from 1 to 10, having math in their package name you can type something like **name=^math stars=1-10**.

- **page** (*required*) (*type: integer*) which will be useful for pagination.
- **size** (*required*) (*type: integer*) size of table rows which is currently selected.
- **filters** (*required*) (*type: string delimiter by ","*) to serve only the data required by the client.
- **Response Skeleton:**

```
{
  data: [
        {
        id: string, // useful for as key while mapping
        name: string,
        stars: number,
        status: string,
        license: string,
        lastModified: string,
        ... // add more to response as filters are applied
        }
  ],
  nextPage: string, // url with filters, size and q to next page.
  length: number, // current length of data.
}
```

- **/packages** (*GET*) this route will the table data.
  - **page** (*required*) (*type: integer*) which will be useful for pagination.
  - **size** (*required*) (*type: integer*) size of table rows which is currently selected.
  - **filters** (*required*) (*type: string delimiter by ","*) to serve only the data required by the client.
  - Response Skeleton:

```
{
  data: [
        {
        id: string, // useful for as key while mapping
        name: string,
        stars: number,
        status: string,
        license: string,
        lastModified: string,
        ... // add more to response as filters are applied
        }
  ],
  nextPage: string, // url with filters, size to next page.
  length: number, // current length of data.
}
```

- Reason Behind choosing pagination for tables.

  So What I was thinking was to store an object in the context provider like this **{key: 'name(default)', orderBy: 'ascending'}**. Whenever a user clicks on the header cell this changes the key in context and we will use memoize to sort the data accordingly or create an api route if performance degrades, and now we can render the data which we have fetched initially which has been sorted. That's why I have chosen pagination over infinite scroll. Problem with infinite scroll will be reordering the elements when the fetched data becomes huge and this will degrade the performance of the website.

Roadmap to Historical overviews and per package metrics.

- Here are some designs which I am thinking of for the metrics page.

Fig 2.1 Main Metrics Page

Fig 2.2 Explanation of Main Metrics Page

Here are some Ideas for Other Metrics like **Build Failures**.

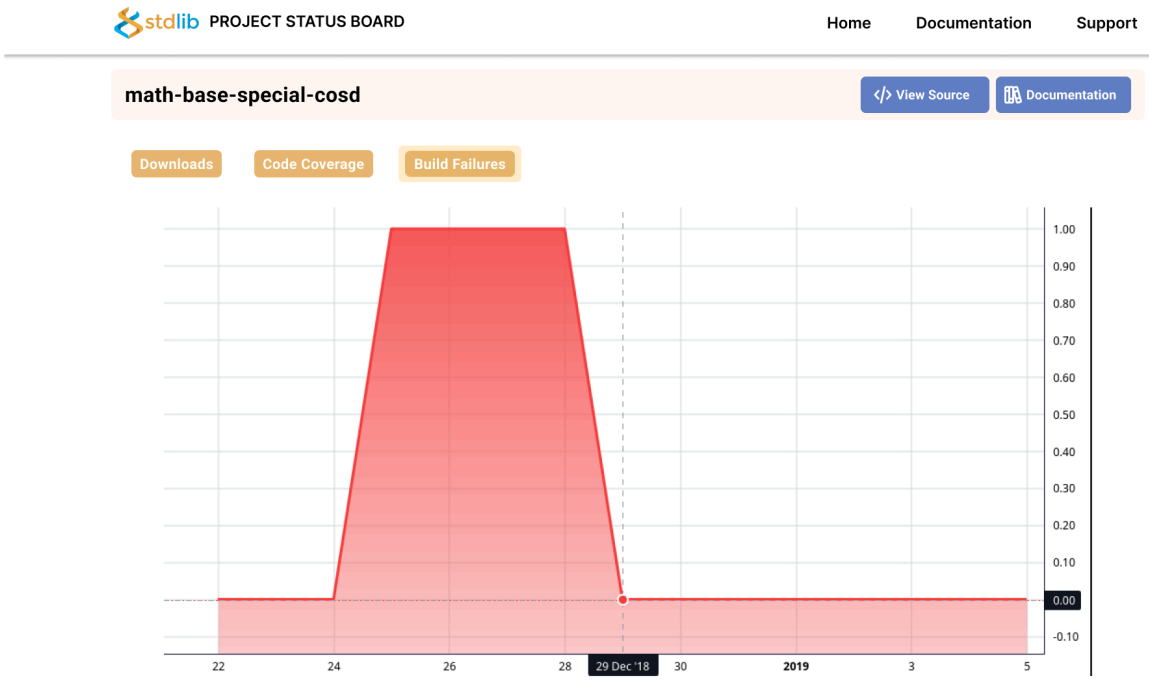


Fig 2.3 Build Failures Data Representation (*only failures*)

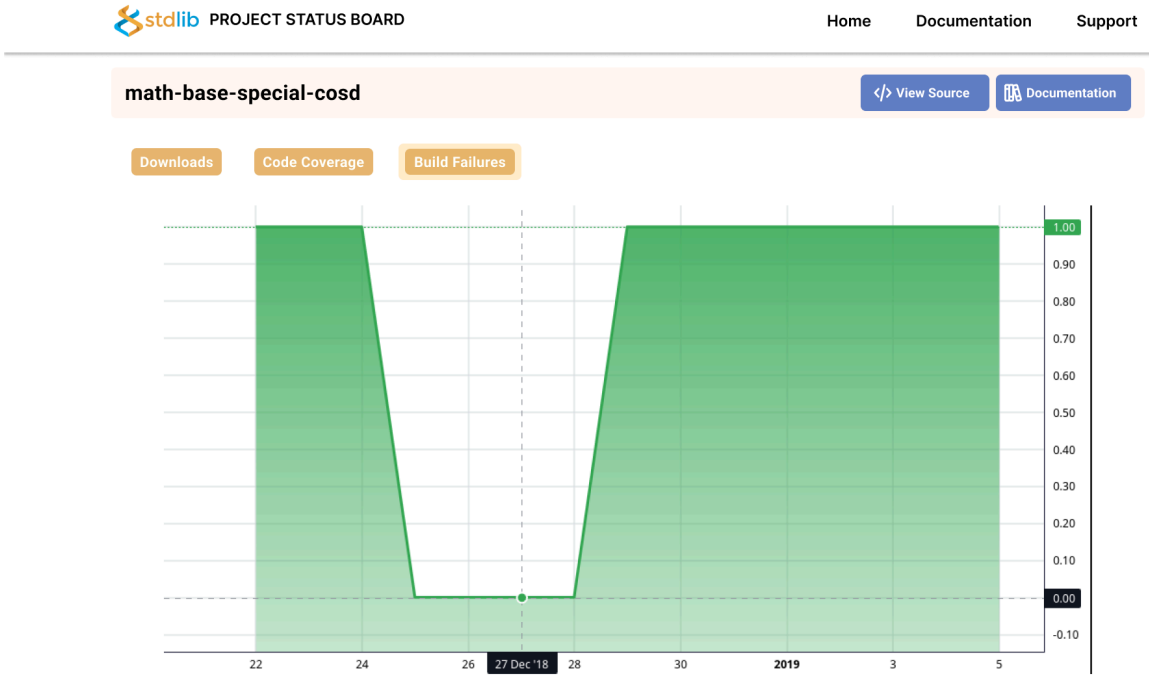


Fig 2.4 Build Failures Data Representation (*only success*)

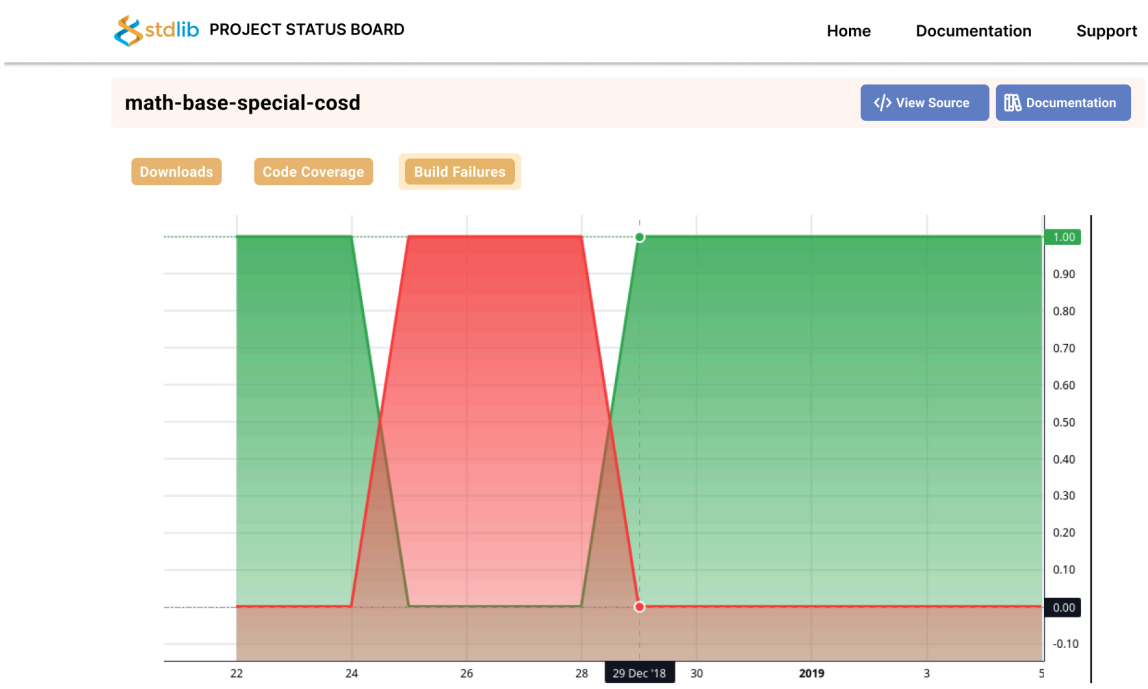


Fig 2.5 Build Failures Data Representation (*fusion of both*)

Links for the above designs:

1. Prototype of the website

- Extending Frontend
    - Currently for demo purposes I have used Lightweight Charts library by Trading View.
      NOTE! this may change as recommended by mentors in the Community Bonding Period.
    - Adding Polling to Status Dashboard with dropdown menu for selecting the time interval.

- Extending API
    - **/metrics/:package-id** (*GET*) this route will server **JSON** data for a given metric by user. Here are some search parameters which will be required for fulfilling a request.
        - **type** (*required*) (*type: string delimiter by ","*) by this we can have **downloads**, **code coverage**, **build failures** and other metrics which can be used.
        - Response Skeleton:

```
{
  pkgName: string,
  githubURI: string,
  documentation: string,
  data: {
```

```
        downloads: [ { date: string, value: number }, … ],
        codeCoverage: [ { date: string, value: number }, … ],
        buildFailures: [ { date: string, value: number }, … ]
        // add more data as defined by type search parameter
    },
}
```

Additional Things for Improving DX

Since, we are creating this for stdlib developers It would be nice for us to have faster navigation around the page for example focusing the search bar, navigating between pagination etc..

Here, are some initial shortcuts which can be useful:

1. For Focusing Search Bar: / (forward slash key)
2. For Navigation to next page of pagination: **ALT +** →
3. For Navigation to previous page of pagination: **ALT +** ←

So, we will also have a modal which displays all these shortcuts when you press **CTRL + /**

Project Documentation

To ensure the codebase remains well-documented, I plan to document it with each pull request (PR) I submit. This way, the documentation burden stays manageable, especially as deadlines approach. Specifically, I will focus on documenting the changes I introduce in each PR, keeping the overall documentation up-to-date without requiring a separate, time-consuming documentation task.

Why this project?

I am particularly interested in this project because it allows me to use my skills to create a tool that will positively affect the community. Best part of this, I can apply the frontend development skills I have learned through contributions to various open-source projects, along with my experience creating and working with APIs. This project presents a fantastic opportunity to use my knowledge and experience to work on something.

With the help of the developer dashboard, we can quickly view which packages have problems in their build status. This allows us to address issues for specific packages easily. Without the dashboard, you would have to go to every single package repository to check their build statuses. By using the dashboard, we can navigate and work more efficiently.

Qualifications

I've made significant contributions to several well-respected projects. Here are some highlights:

- Qwik's website A frontend framework which introduced **Resumability** to the frontend world.
    - I helped enhance the user experience of Qwik's website by redesigning the Media Page
    - I'm also nearing completion on the development of the package manager tabs which will be integrated into production soon.

- Wagtail
    - Implementing a production-ready url generator copy button using stimulus.
    - Migration of components from vanilla js to stimulus here are some migrations enableDirtyFormCheck and tabs.js.

In addition to my open-source experience, I'm currently enrolled in a **Database Management System (DBMS)** course at college. This course is equipping me with the skills to execute queries efficiently and apply optimization techniques, which I'm eager to leverage in this project.

Prior Art

Yes, there is a similar project like this from npm where they are also tracking the build status, total downloads and many more things. From what I have understood from their Github Repo. They have created a folder named **workspaces** which contains 2 projects

- **www** from here they are serving simple **HTML**, **CSS**, **JS** and using jquery for DOM manipulation.
- **data** from here they are serving their content from **REST service** or **GraphQL** as needed.

All these things are automated by using GitHub Actions.

Commitment

I plan on devoting my entire summer to this. I won't have any other kind of job, so I will be able to work a standard 40 hour week for the coding period. Also, my classes will end well before the coding period begins and will not start up again until well after the pencils down date, so I will be totally free for the whole period.

Schedule

Weeks 1: Community Bonding & Planning

- Activities:
    - Finalize frontend design with mentors (including user interface (UI) mockups or wireframes).
    - Discuss and document API routes and website themes.
    - Set up the development environment (including database setup if needed).
- Deliverables:
    - Approved UI mockups/wireframes.
    - Documented API routes and website theme.
    - Development environment configured (*including database setup, if applicable*).

Weeks 2-4: Frontend Development & Core Functionality (Roadmap to Status Dashboard)

- Activities:
    - Implement core frontend components with clear and concise code.
    - Planning making Pull Requests for each component for both parts.
    - Thoroughly document components with comments and per-PR documentation.
    - Integrate front-end with APIs (if possible at this stage).
- Deliverables:
    - Functional frontend components.
    - Well-documented code with per-PR documentation.
    - (Optional) Initial frontend-API integration.

Weeks 5-7: Midterm and Per Package metrics (Roadmap to Historical overviews and per package metrics)

- Activities:
    - Conduct a mid-term review with mentors to assess progress and address any concerns.
    - Start working on the Package Overview page.
    - Continue documenting code throughout development.
- Deliverables:
    - Fully functional website with core functionality.
    - Functional Package Overview page.
    - Ongoing code documentation.

Week 8: Performance Optimization

- Activities:
    - Analyze and optimize database queries using the Postgres library.
    - Consider performance optimization techniques for the frontend (e.g., caching, lazy loading).
- Deliverables:
    - Optimized database queries.
    - Improved frontend performance (if applicable).

Weeks 9-11: Testing & Refinement

- Activities:
    - Write unit tests for frontend components and APIs using Jest or a similar testing framework.
    - Fix any bugs identified during testing.
    - Continue with code documentation.
- Deliverables:
    - Comprehensive unit tests for frontend components and APIs.
    - Bug-free website.
    - Well-documented codebase.

Week 12: Finalization & Review

- Activities:
    - Conduct a final review with mentors to ensure project completion aligns with initial goals.
    - Document any remaining parts of the codebase.
    - Prepare for presentation or deployment (if applicable).
- Deliverables:
    - Project reviewed and approved by mentors.
    - Fully documented codebase.
    - Project ready for presentation or deployment.