Python Software Foundation

Sub-org: EOS Design System
Project: EOS Icons Next.js & TypeScript Migration
By: Ayush Satyam

Contact information :

Name: Ayush Satyam

Email ID:

Github Username: https://github.com/ayushsatyam146

Mobile: (+91)

Country: India

Timezone: **Indian Standard Time** (UTC +05:30)

Primary Language: English

Linkedin: https://www.linkedin.com/in/ayush-satyam-179ba4197/

University Info:

University Name: **Shri Mata Vaishno Devi University, Jammu & Kashmir**

Current year and Degree: 3rd Year, Bachelor of Technology in Computer Science & Engineering

Expected Graduation Date: August 2023

Code Contribution:

I have been actively contributing to the EOS Icons repository since January 2022 and have made substantial contributions to it. I have also suggested and worked on some ideas for UI/UX improvement along the way. Some of the major ones are listed below:

- **[Merged][Bug]** !22, !126: Fixing the site behavior on refresh and making icon links shareable. Persisting URL parameters, search value, and vertical position on refreshing the page or sharing an icon link fixing issues #16, #124.
- **[Merged][UX Enhancement]** !81: Added functionality to take the user to the home screen whenever the eos-icons logo in the navbar is clicked. Fixes issue #80.
- **[Merged][UX Enhancement]** !119: Added functionality to open the icon editor panel

on double-clicking an icon, giving users a smooth UX to edit and download icons. Fixes issue #118.

- **[Merged][Code Enhancement]** !123: Wrapped animated icons into a defined component for better understanding & readability of the codebase. Fixes issue #122.
- **[Merged][UX Enhancement]** !101: Closing the icon editor Panel on clicking outside the icon editor component. Fixes issue #100.
- **[Merged][Bug & UX Enhancement]** !129: Fixing the app's behavior on clicking the tags in HowTo Panel. Make tag links shareable by retaining the app state for shared links. Fixes issue #128.

All other Pull Requests are listed **here** (17 Merged, 1 Closed). All Issues created are listed **here** (10 Closed, 1 Open).

Synopsis:

I will be working on the EOS Icons website to make it more reliable, faster, operational, and easy to maintain. I will also **add various UI changes** like adding a **navigation panel for icon categories** and adding a **side drawer for the HowTo section & icon editor panel**. Along with these, I will also add some other significant UI/UX changes to the website after a thorough discussion with mentors to improve the overall user experience.

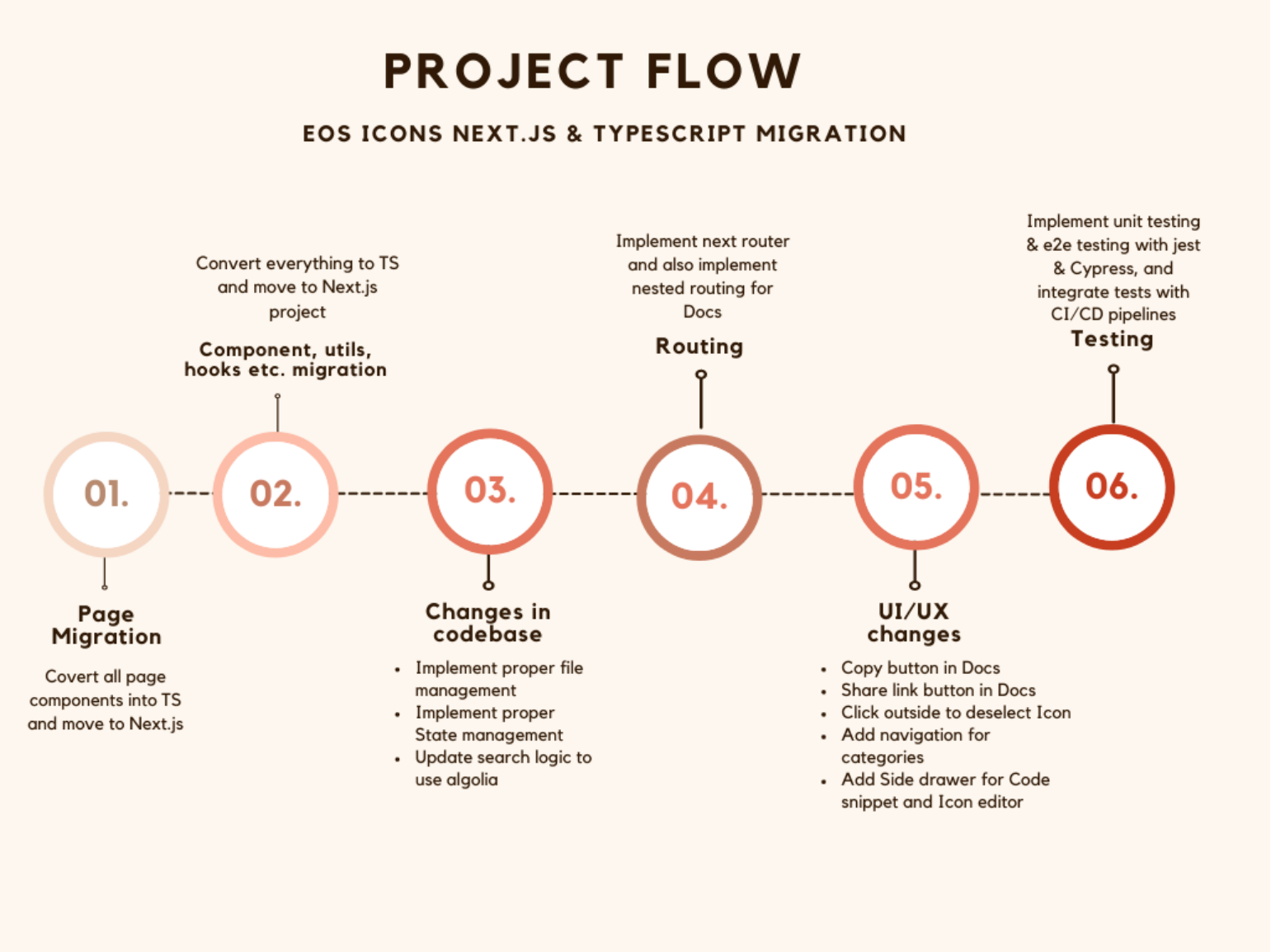
I will **migrate the project to TS and Next.js** and also implement proper file management and state management in the project ensuring clean code and easier scaling. Migration to TS and Next.js will ensure **better SEO** and fewer bugs. I am planning changes in categories and icon editor UI for a better user experience. I plan to add **unit tests** by using **Jest & React-testing-library**, and **e2e testing** with **Cypress.js**. I will integrate these tests with Github CI/CD pipelines and ensure proper development workflow using es-lint, prettier and husky.

I've already dug deep into the code, fixed some major functional and UI/UX bugs, and introduced and worked upon new ideas that might help make a positive change in the project. I am comfortable writing JS/TS, React, and Node.js for a while now. I also have some experience working with Next.js during my internship last year. I try to use the best coding practices while working. I am a fairly quick learner and open to learning new concepts and ideas along the way. I hope for a fruitful collaboration with mentors and other contributors to complete this project.

Which of the published tasks are you interested in? What do you plan to do?

I am interested in [Project idea 3: EOS Icons - TypeScript, Next.js, and guides] for the EOS Design System.

The goal is to migrate the entire EOS Icons landing app to Next.js and transition towards using TypeScript for the application. I also intend to change some UI/UX elements to make the app more intuitive and operational.



Project Plan & Milestones:

I am planning to migrate the EOS icons website to the Next.js ecosystem and convert the existing codebase to use TypeScript resulting in more manageable and scalable code. I also plan to add new features, and several new UI/UX changes to make the existing app more intuitive and easy to use for the end-user.

The Project is divided into 4 milestones, gradually explaining the needs and the workflow of the project.

1. TypeScript & Next.js Migration

Currently, the EOS icons website is a Single Page Application (SPA). Due to the client-side rendering nature of React, it makes the app hard to crawl and difficult to render for the first contentful paint. All these things hinder the SEO capabilities of the website.

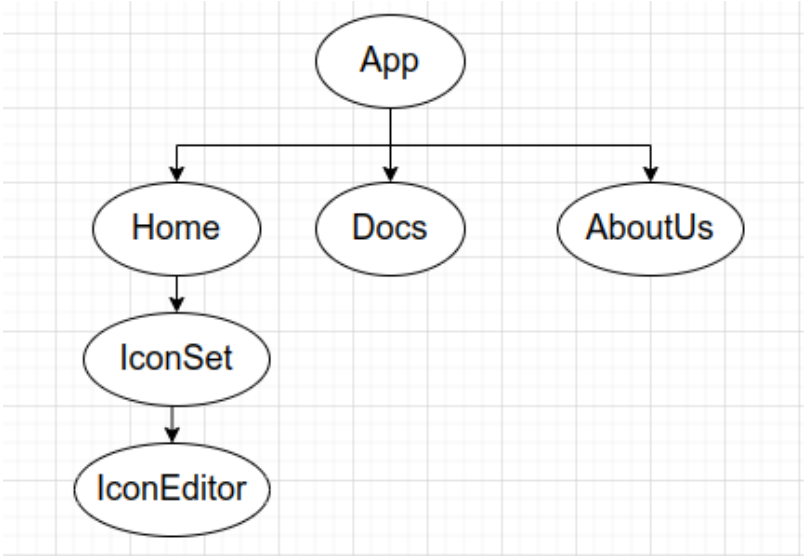
We can use custom meta tags for each page to boost SEO rankings. Along with the migration of the existing public/index.HTML page headers in the _document.js sample page provided by Next.js

I am planning to serve all pages statically as it will drastically improve the performance of the app, and also it is not much needed as we don't need to render anything on the server. I will do the migration in a series of steps.

a. **Migrating the pages, utils, hooks, components & modules**

   I plan to begin the migration from **`Pages`** as it is the most important part of any Next.js application because routing and Project structure depends on the Pages directory.

   I will move the pages, utils, hooks, and modules from the old project to the new Next.js project and simultaneously convert the code & logical elements to TypeScript from Javascript.



TypeScript Migration Workflow Diagram

During the migration of components, I will follow a bottom-up approach i.e. converting smaller components first so that bigger components that are using them get migrated easily. According to the above diagram, IconEditor will get migrated first, then IconSet, Home, and then App.

b. Routing

Managing the routing in Next.js won't be a big issue as the routing in Next.js depends on the files and sub-directory structure of **`Pages`** Folder. I plan to handle routing after the migration of all logical and fundamental Javascript to TypeScript.

c. Styling

The older eos-icons project uses SASS for styling. It follows the 7-1 Ladder architecture for writing SASS. Since most of the app's styling is going to remain the same with very little addition for newer components we can use the exact SCSS styles from the older project. There won't be any issues in integration with Next.js as it supports SASS natively.

d. Improvements in Codebase

The current codebase is well catered to the needs of the app but we can further improve it to ensure better scalability.

I am planning to do a lot of significant improvements to the existing codebase. Since it will get a lot cumbersome to migrate unmanaged code and then add the improvements, I am planning to do these **improvements along with the code migration process**. I am describing all the possible improvements in the next section.

2. Improvements in Codebase

The codebase is well suited for the current version of the app. However, it will get more difficult and complex to work with as the app grows. It doesn't enforce any particular guideline for declaring or managing the states, using hooks, or component nesting.

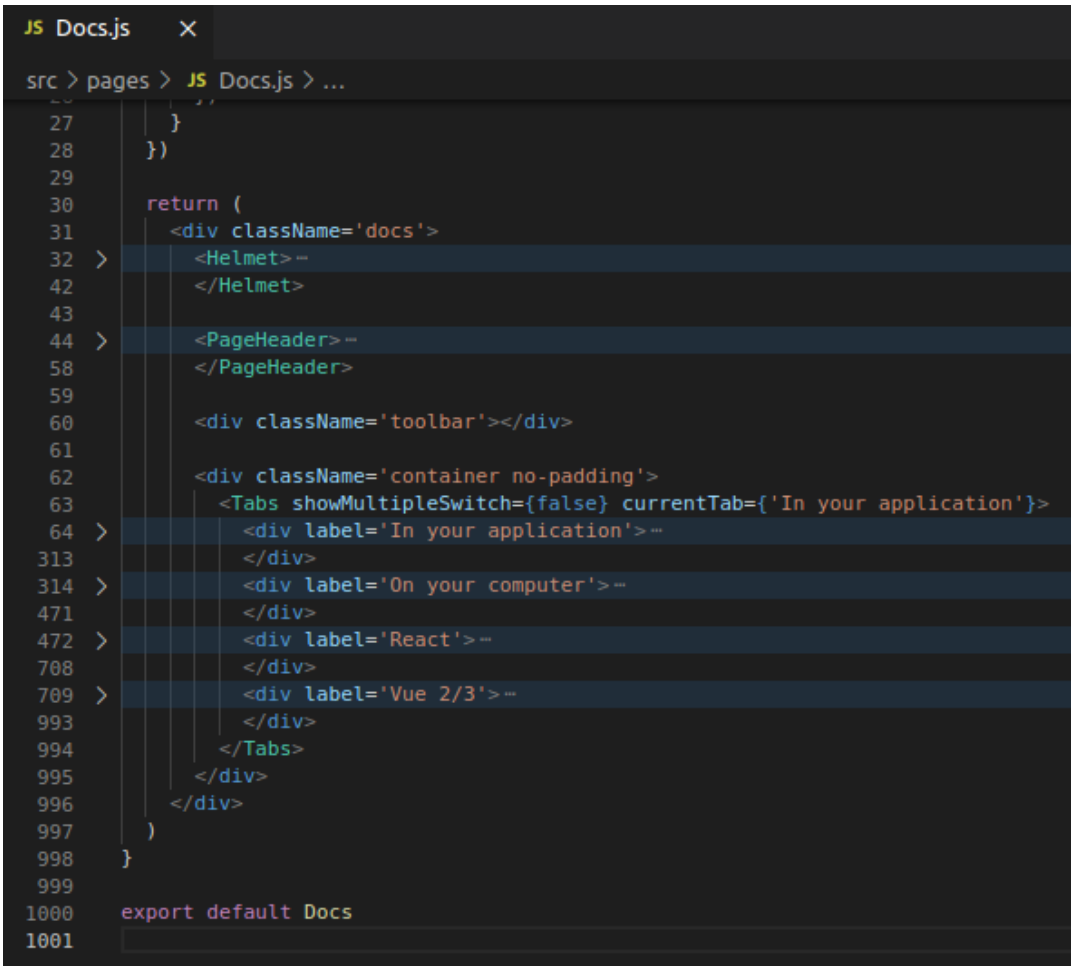
- Splitting the code base into well-managed file/directories

There are many instances in the codebase where multiple separable components are present in the same file making the component excessively complicated.

One such example is IconSet.js in the present codebase.

This component contains most of the app logic for one of the most important sections of the site. But this section contains components and props for other child components which should ideally be separated from each other.
For example, the ShowHowToUse component

should be separated into a different file in the components directory.



Another such example is the Docs.js file. This file contains documentation for the app for all use cases like React, Vue 2/3 & HTML inside the component, making it really lengthy.
I am planning to separate each section into its own component so that there is a logical separation between different sections, also it will help in routing the docs section in a better user-friendly way.

I am planning to add sub-routes to each section (React, Vue, HTML, In-your-application) of the Docs Page, so that users can share a direct URL of a particular section.

Sample URL - **"https://eos-icons.com/docs/react"**

The above URL will lead the user directly to the react Docs Page.

- ## Introducing Proper State management

  Proper state management is vital for the app. We are using React's Context API to manage some important global states of the project present in EosIcons.store.js. We control various global app states and their actions through this store for the whole app.
  But, a major portion of the app containing the majority of the business logic is contained in the IconSet.js file. There is no active state management present for the states of this file.

```
const IconsSet = (props) => {
  const [iconSelected, setIconSelected] = useState('')
  const [showPanel, setShowPanel] = useState(false)
  const [searchValue, setSearchValue] = useState('')
  const [size] = useWindowsSize()
  const [tab, setActiveTab] = useState('Static Icons')
  const [staticHistory, setStaticHistory] = useState('')
  const [animatedHistory, setAnimatedHistory] = useState('')
  const searchRef = useRef(null)
  const [selectMultiple, setSelectMultiple] = useState(true)
  const [emptySearchResult, setEmptySearchResult] = useState(false)
  const [suggestedString, setSuggestedString] = useState('')
  const [iconEditor, setIconEditor] = useState(false)
  const [userSearchInput, setUserSearchInput] = useState(false)
  const [tagSelected, setTagSelected] = useState('')
```

  These are all the states that are associated with the IconSet.js file.
  I am planning to use the **Context API** for generating a global object of IconSet's state variables. I will use the **`useReducer`** hook to manage the states and their actions. The main reason for using Context API is to maintain consistency across the whole project. We are already using Context API for passing the Eoc.IconStore.js file's state. Also, our data doesn't update very frequently like a social media app hence using something like Redux will unnecessarily increase the bundle size significantly.
  The majority of the code in the IconSet.js file comprises functions to alter the states of the file. If we move the states and actions of this component into a dedicated file, then we will have a centralized place for introducing new features and fixing bugs by having new states and actions. Also, it will help in abstracting out a lot of logical parts from the main component making the code a lot cleaner and easier to work with.
  The component shown below is present in the IconSet.js file to call the Tabs component, taking a lot of states as props for the Tabs component.

If we manage states properly we won't need to pass every required prop manually to the component; we can just use **`useContext`** in the corresponding component's file and use the value of whatever state we want.

```
<Tabs
  setTab={(e) => tabSwitch(e)}
  customize={state.customize}
  showPanel={showPanel}
  currentTab={tab}
  toggleCustomize={(callback) => toggleCustomize(callback)}
  showMultipleSwitch={true}
  resetTabsState={true}
  resetTabsStateRef={resetTabsStateRef}
>
  <div label='Static Icons'>⋯
  </div>
  <div label='Animated Icons'>⋯
  </div>
</Tabs>
```

Tabs component in Current file

After we transfer all states into a store we can just call components in the IconSet.js file and use the state values from the nested children at any level eliminating the need to pass props manually for every level. The Tabs component will look something like shown below.

```
<Tabs resetTabsStateRef={resetTabsStateRef} >
  <div label='Static Icons'>⋯
  </div>
  <div label='Animated Icons'>⋯
  </div>
</Tabs>
```

A lot of components will get cleaned up, making the code in IconSet.js cleaner and more scalable; which will be a huge productivity boost for other developers working on this project in the future.

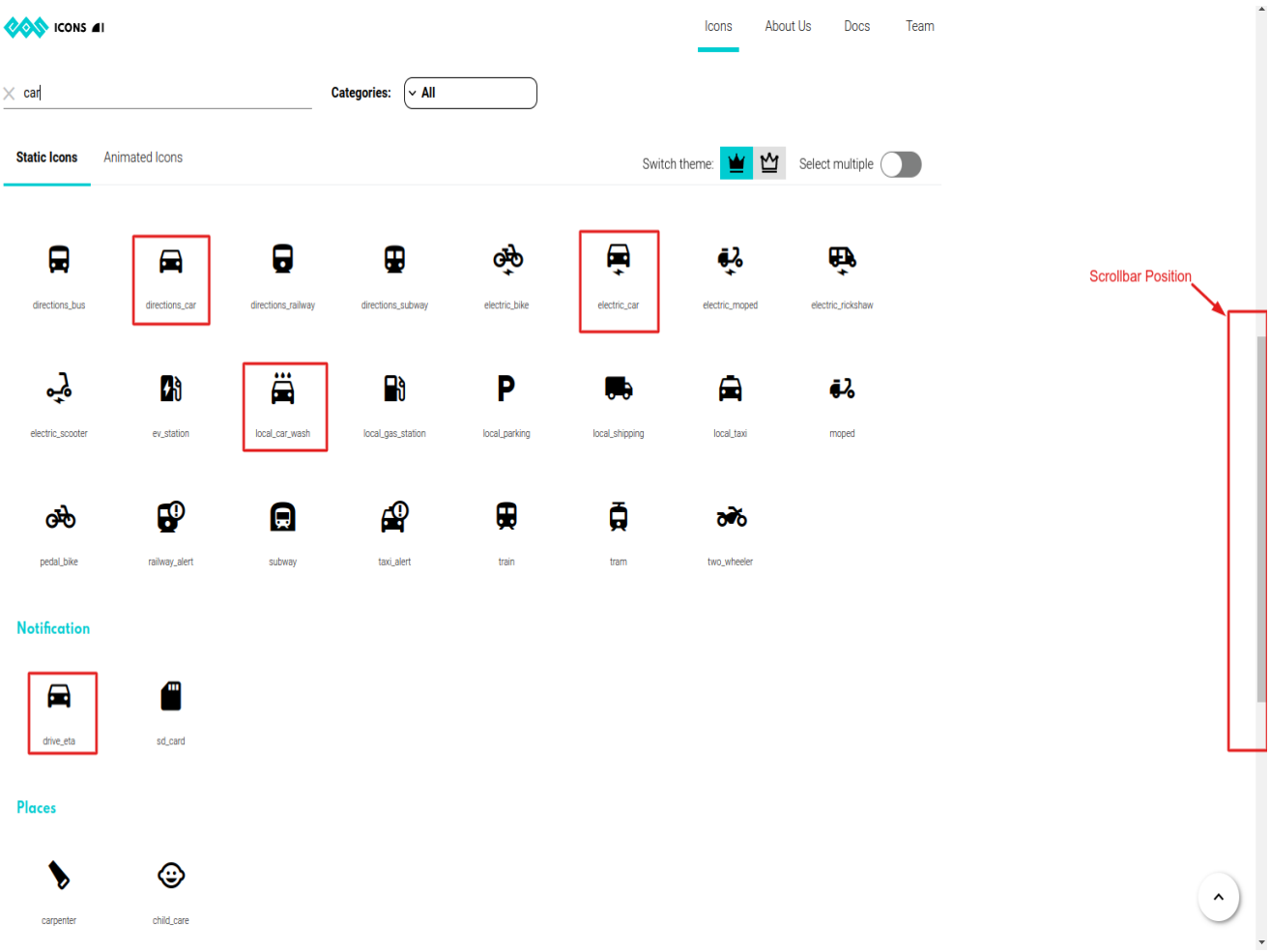
- ## Routing

  Since Next.js comes with its own router we will need to migrate to next's inbuilt router. While we are migrating to the new router we can separate out the components of the Docs Page and use nested routing for each section of the Docs Page, so that users can directly share the URLs to

specific documentation of their needs.
Apart from separating the URL to each section, I'm planning to add functionality for shareable links to every relevant part of the page. I will further explain this in UI/UX design Changes section

- ## Updating the search logic to use algolia

  We use custom search functions implemented in the EosIcons.store.js file for implementing the search logic in the app.
  This approach will work in most cases but sometimes it also leads to bad search results.



For example, if we search the keyword "car" the term "add_card" is treated as a better suggestion than the term "car_rentals" affecting the search results in a bad way.
Also, we find the matches for the search value in each category of icons and show all these icons category-wise. An icon that might be a better

match doesn't show up first because it belongs to a category that is positioned lower alphabetically.
I am planning to remove categories in search results for better search recommendations.
Also, as **discussed** in the **development slack channel** of the EOS community we currently have an **algolia** instance for search queries. But currently, it is inactive and we are not using it for search queries because frontend code logic is not updated accordingly to consume search results from algolia.
I am planning to update the frontend code logic to consume algolia for search queries. Since algolia provides a search engine as a service for custom datasets, our search results will get more accurate and any scope for logical flaws in the implementation of search logic will be eliminated.

3. UI/UX changes

I have made a Figma mockup for all the UI changes that I have proposed. You can take a look at the Figma wireframe here. I am also including a Figma prototype for the designs which I have linked here. I am attaching a video link here.

- ### Introducing navigation for Categories (UI Changes)

Currently, icon categories are shown as a small dialog box besides the search bar. Icons in all the categories have been laid out neatly in the UI, but because there are so many icons present in each category it may not be intuitive for the user to know that there are several more categories of icons that they can explore.

I am planning to introduce a navigation menu below the search bar listing all the categories of icons. It will instantly let the user know that there are more categories of icons that can be explored, and also make the workflow easier by making the categories more navigable.

Along with all the default categories I'm planning to add a category called **"Popular icons"**. This will contain all the icons that are used most frequently. As discussed with mentors we already have the analytics in our backend so we can use the number of downloads or number of clicks to populate "Popular icons". This will be the default Tab on the first-page load. All other categories will come after the popular icons category.

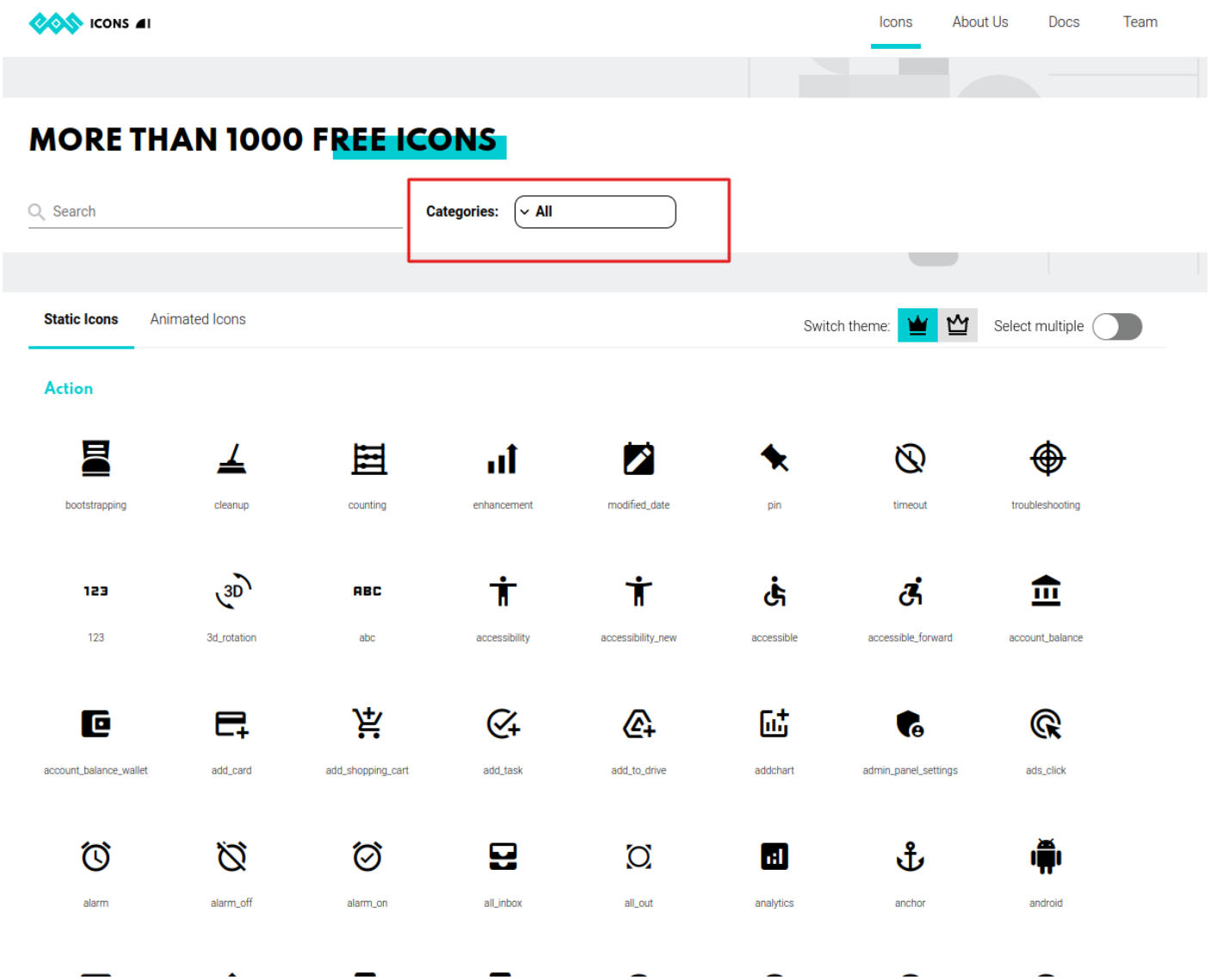
These categories will automatically get highlighted according to the

scrolling of the page, and we can also click on a particular category to instantly scroll that section of the page.

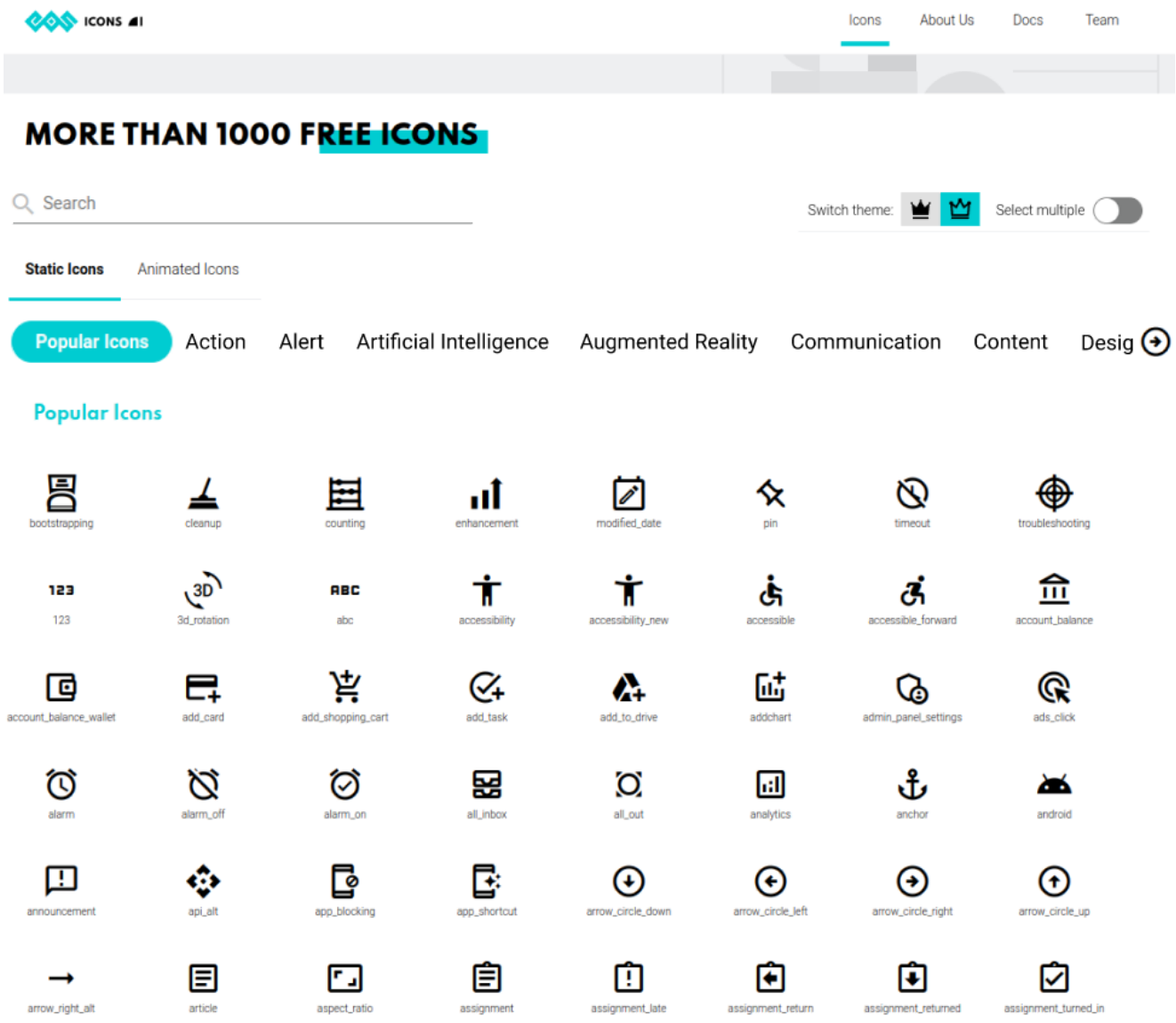
This way it will be easier for a user to navigate to all the categories, and they will instantly know about all the available categories. This will solve a major UX hurdle that we currently have on the site.

I am attaching the current and expected view of the home page.

Current view



Expected View



- ## Introducing a Side drawer (UI Changes)

  As mentioned in the GSoC project ideas list we need to show the users all the possible options for using the icon in the HowToUse section. But, it is already taking up a lot of vertical space in the current UI. If we include all the options in the HowTo section of the current UI, then the icons will get pushed further down.

  Currently, in the HowTo Use section, we are just showing how to use the EOS icons in the form of an HTML tag. We can use the icons with React & Vue2/3 as well. We are not showing how to use the icons in React and Vue. Also, we are not showing any of the customizations that we can achieve with the HTML tags.

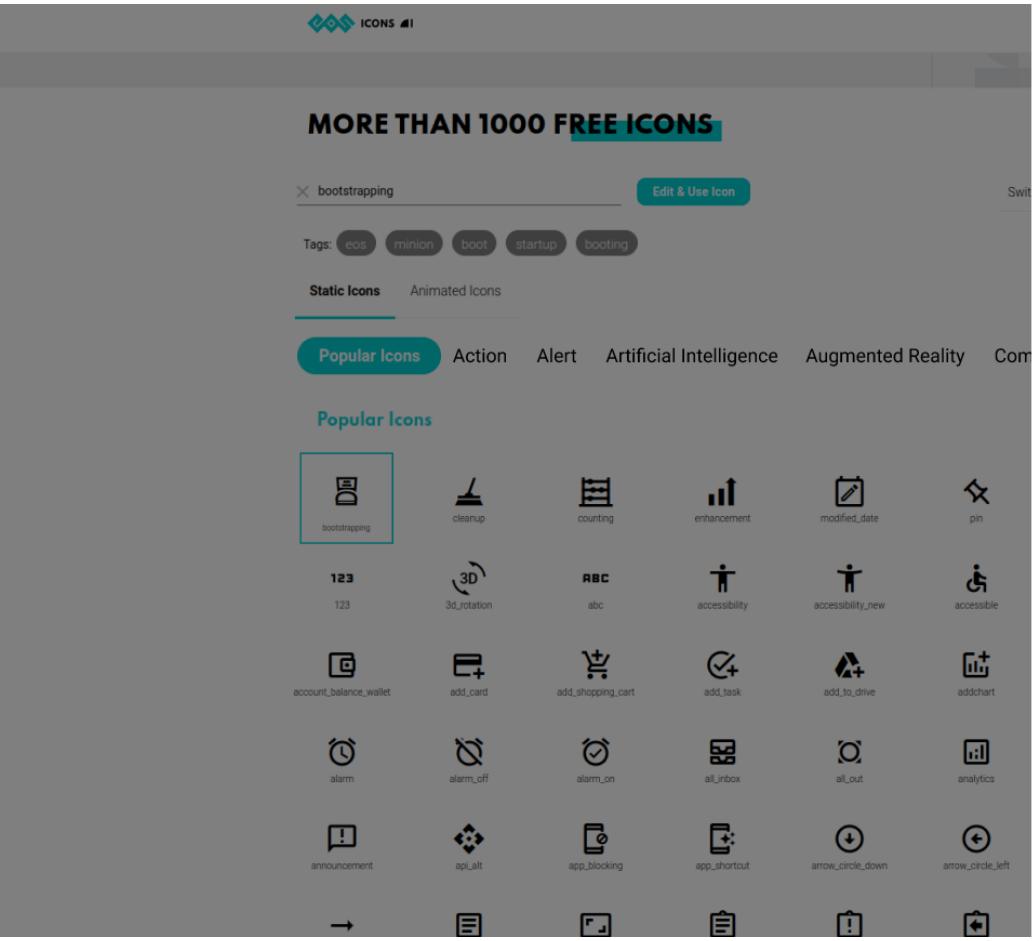
  We need to show the user all the possible ways of using an icon without taking up much space. I'm thinking of pushing down all the information to a side drawer widget. We will move the icon editor panel too inside the widget so that the user gets all the things that they might want from a single widget.

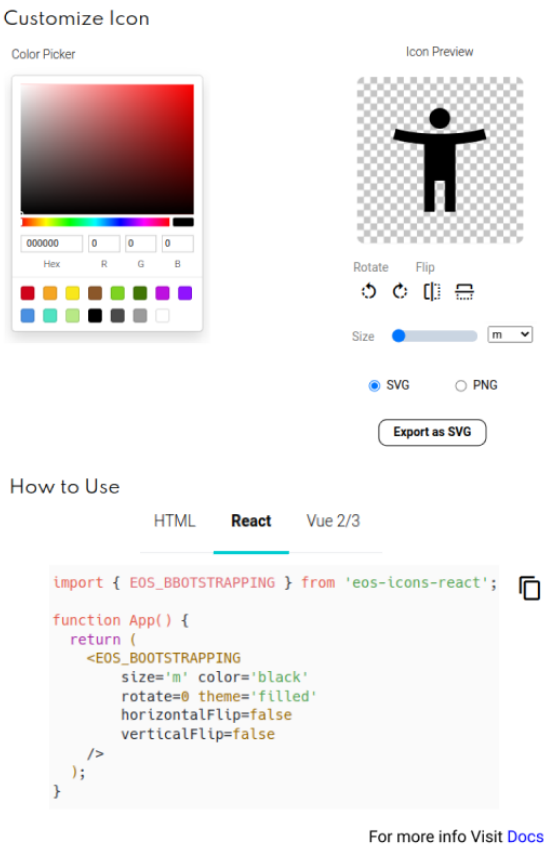
  We will give customized code snippets according to the icon selected and an option to toggle the service user wants to use (React, Vue 2/3, HTML). The **code snippets will contain all the available prop values and if the user changes any props like color, rotation, or size in the icon editor the code snippet will get updated accordingly**. The user will edit the icon and copy the generated code snippets to use the icon. We will also link the Docs section of the selected section in the side drawer panel.

  Since the side drawer panel will show all the places users can use the icon. It will instantly let the user know about the flexibility that eos-icons provide for using the icons, right after they select a particular icon.

  I am attaching pictures below to show a rough overview of the expected UI.

Expected UI





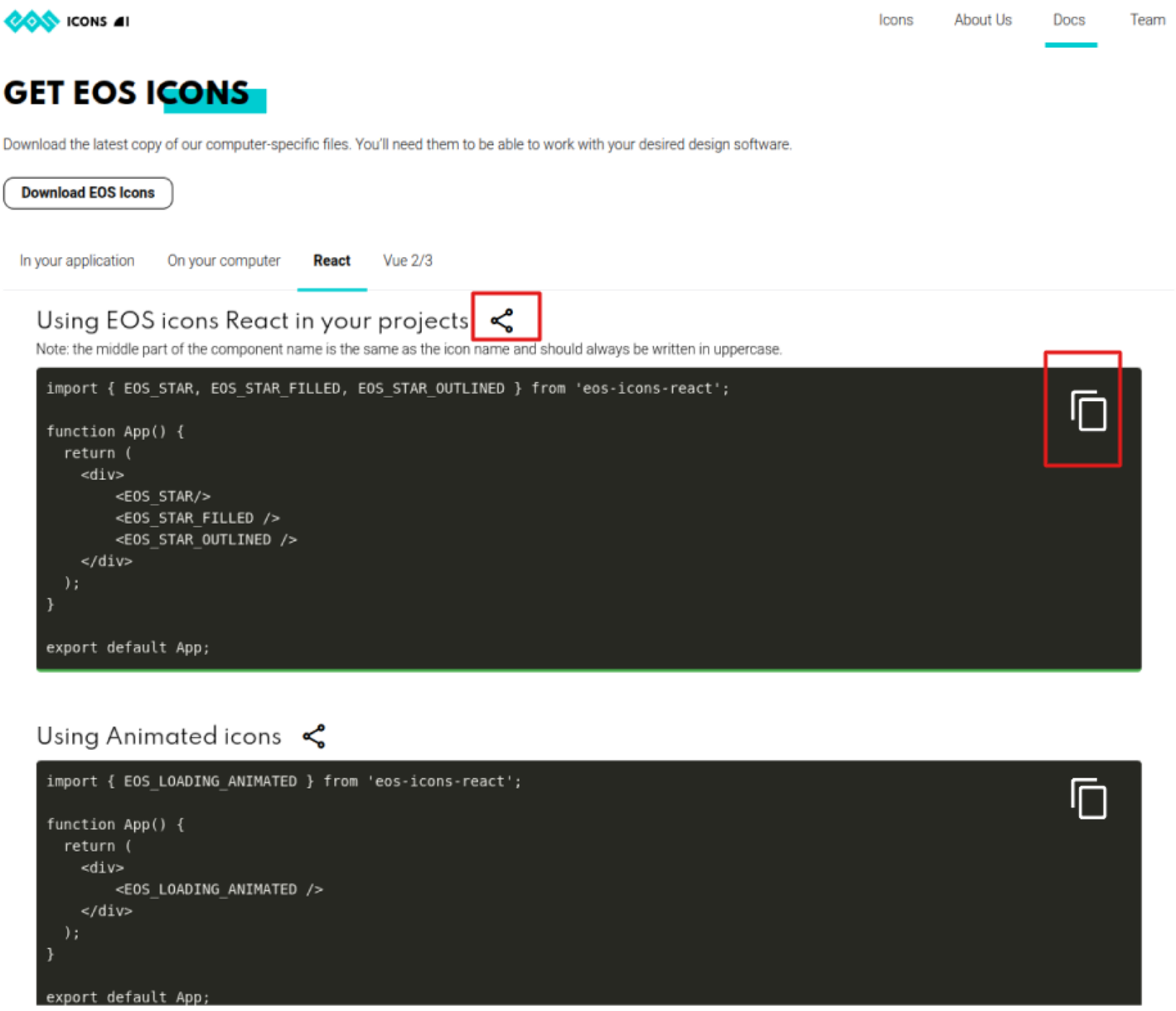
- **Copy button in code snippet and sections in Docs (UI Changes)**

  There's no button to copy code for any section of code in the Docs. Users can view all the sample code snippets for all documentation but they can't copy the code snippet directly from the Docs. I will add a button to copy all the code snippets directly with a single click.

  Also, currently, there is no way for a user to share the link of a specific section of the Docs page. For example, if the user wants to share a specific link pointing to the "Using Animated Icons" in the "React" section they can't do it.

I'm planning to **add a share link button** right next to the heading of each section which will copy the link of that particular section to the clipboard. When the user shares this link with another user, they will just land on that specific section of the website. I am adding a mockup of the expected UI below

Expected UI



- ## Click outside to deselect an Icon (UX Change)

  In the current UI if we want to use an icon we can select it, but when we want to deselect an icon we have to click on that icon again or just click on the logo of the website present in the navbar.

  A very normal and expected user behavior is to click on an empty space to deselect the currently selected icon. We don't support this functionality currently.

  I'm planning to add functionality to deselect an icon selection by clicking on any empty space inside the webpage.

  I have already implemented a similar feature to close the Icon editor panel on clicking outside. I created a custom hook called useClickOutside for implementing this. We might need to look out for deeper nested components for the proposed feature but it can be done in a similar fashion as we did with the icon editor panel in PR !101.

```
import { useEffect, useCallback } from 'react'

const useClickOutside = (ref, callback) => {
  const handleClick = useCallback(
    (event) => {
      if (ref.current && !ref.current.contains(event.target)) {
        callback()
      }
    },
    [callback, ref]
  )

  useEffect(() => {
    document.addEventListener('click', handleClick)

    return () => {
      document.removeEventListener('click', handleClick)
    }
  }, [handleClick])
}

export default useClickOutside
```

4. Testing & CI/CD Pipeline Integration

I am planning to implement unit testing and end-to-end (e2e) testing using Jest, react testing library, and Cypress.js respectively for the whole project. I plan to use Jest & Cypress.js to maintain consistency among the other EOS design system projects. Jest & React testing library are recommended by the Next.js team for unit testing and it integrates well with React and Next.js ecosystem. Cypress offers fast test execution and better debugging capabilities. It eliminates the need for any other frameworks or assertion libraries.

Due to the nature of the project, we have a lot of complex user interfaces in the app. Unit testing will ensure that all stand-alone components and functions are working properly after any changes, while E2E testing will let us test exact user behavior while using the app across various user flows. We can also implement vertical e2e testing for certain components to ensure that other important nested children components are working fine. The majority of the tests will be unit tests as they provide better insights for debugging, and are not that computationally expensive compared to e2e tests. We will use e2e tests for testing the behavior in some of the major user workflows like editing and downloading an icon, Navigating around the site, visiting Docs, etc.

After implementing the tests I will also integrate them into the Github CI/CD Pipelines. This will eliminate the need for manual testing of the future Pull Requests.

Following the true unit testing methodology, I will make **unit tests for all the components, hooks, and standalone pure utility functions,** to ensure that all the smaller components are working correctly inside the app.

Some of the e2e tests that I plan to implement -

- The user is able to navigate across all pages and tabs and access the default view of each page
- The user is able to select and download all icons (static and animated) from the top and bottom of the page.
- The user is able to access and scroll to the Docs of each section.
- The user is able to edit and do the icon customization in the side drawer panel and download it.

Timeline:

Community Bonding (May 20 - June 12):

I'll use this time to set out my objectives and discuss the project's workflow with my mentors. I'll also discuss and try to finalize all the UI/UX changes that I have proposed. I'll solidify the requirements and the plan of action. I'll also familiarize myself with the EOS workflow and begin working with the scrumban technique. I'll also take advantage of this community bonding time to get to know all of the other GSOC 2022 participants as well as the mentors.

Week 1 (June 13 - June 19):

- Setting up the Next.js project with the initial configuration.
- Setting up tsconfig.json for TypeScript along with the initial setup of ESLint, prettier and husky.
- Planning out the migration process by sorting the order of migration of pages, components, and functions.

Week 2 (June 20 - June 26):

- Start with the migration of the pages directory of the project.
- Work upon migration of components, utils, hooks, and functions.
- Rearrange Docs component for nested routing.

Week 3 (June 27 - July 3):

- Complete the component migration process.
- Ensure proper file management of the code-base

- Splitting the code-base into separate files and directories if needed.

Week 4 (June 4 - July 10):

- Ensure proper state management in the code-base.
- Implement state management using Context API and useReducer hook.
- Separating functions and other logical units of the code-base.

Week 5 (July 11 - July 17):

- Complete the migration of the functions and other units.
- Migrate code to use the default Next.js router.
- Implement nested routing in the Docs section.

Week 6 (July 18 - July 24):

- Resolving issues that might appear during the migration process and ensuring that a minimal number of side effects are running in the app.
- Remove current search logic implemented in the frontend.
- Use algolia for implementing the search feature.
- Update UI to show the best search results regardless of the category of the icon.

Week 7 (July 25 - July 31):

- Make a navigation bar to list all categories on the home screen so that users can look and scroll through all available categories.
- Add a category called “Popular icons” to the list of icon categories.

Week 8 (August 1 - August 7):

- Introduce one of the major UI changes by making a side drawer
- Refactor the UI of the icon editor Panel and move it inside the side drawer.
- Add a feature for having code snippets in the HowTo section for React/Vue/HTML and move it inside the side drawer panel.

Week 9 (August 8 - August 14):

- Change the UI elements of the Docs section
- Include a share link button for every section in the documentation.
- Introduce a copy button in code snippets present in the docs section.

Week 10 (August 15 - August 21):

- Discuss and plan out unit tests for the project.
- Start implementing unit tests for various components.
- Implement unit tests for other functions, hooks, and utils.

Week 11 (August 22 - August 28):

- Continue with unit testing of various components.
- Design user flows for implementing e2e testing.
- Implement e2e tests for all user flows in Cypress.js.

Week 12 (August 29 - September 4):

- Continue implementing e2e tests.
- Debug and resolve any new issues that might appear during the development or testing process.
- Integrate Github CI/CD pipelines to the tests.
- Resolve any final issues that might appear in the code-base and wrap up the project

Previous experience:

I am well accustomed to the Javascript-based ecosystem of web development and most of my work till now has been around frameworks like React and Node.js in the field of web development.

I got started with HTML, CSS, and JS in my first year of bachelor's and slowly learned all the new popular web development frameworks as I moved ahead.

I had done a web development internship at Pravegak Technologies last year, which was a great learning opportunity for me. During my internship, I made backends for a couple of apps in Node.js. The first app is a chat app named unicohub. The second app is a multiplayer online game app named Skribo.

I migrated Skribo to a Node.js backend, earlier it was using Firebase. I also redesigned certain API endpoints and implemented content and query caching to optimize the backend. The app's user base has grown significantly since then and now the app has over **100,000+ downloads**.

I also made various plugins for Shopify during my internship period. I also worked on some websites in React.js and Next.js during my internship period, like a CMS (Content Management System) for the above-mentioned apps and some blog-based sites for clients.

Other than this I am very passionate about Operating systems and compiler design, and in my 5th semester, I made a project called OPUS. OPUS is an

esoteric programming language designed and optimized to capture MIDI (Musical Instrument Digital Interface) input from musical instruments and compile those inputs with the OPUS compiler that I implemented in C++ to generate outputs. Basically, it’s a programming language that allows users to write code with the help of musical instruments (MIDI).
I have also done competitive programming for quite some time and I have also qualified for ACM ICPC regionals 2020.

Tell us a bit about you:

I am a computer science engineering student passionate about building out-of-the-box stuff. I am a firm believer in picking utility over technology in every project. I like solving problems and try to be as efficient and practical as possible with my solutions.
I like playing and listening to all kinds of music. I am currently trying my hands-on learning guitar and piano. Apart from the web; Operating Systems, Compilers, and System design are the fields that intrigue me the most.

Commitments:

I won’t be applying to any other organizations in GSoC, nor do I have any internship or a full-time job during the GSoC period. I'll be giving 10-day long semester exams somewhere from late June to early July, thus I'll only be able to devote 2-3 hours per day during that period. I will be able to devote more than 6 to 8 hours every day except for the examination period.