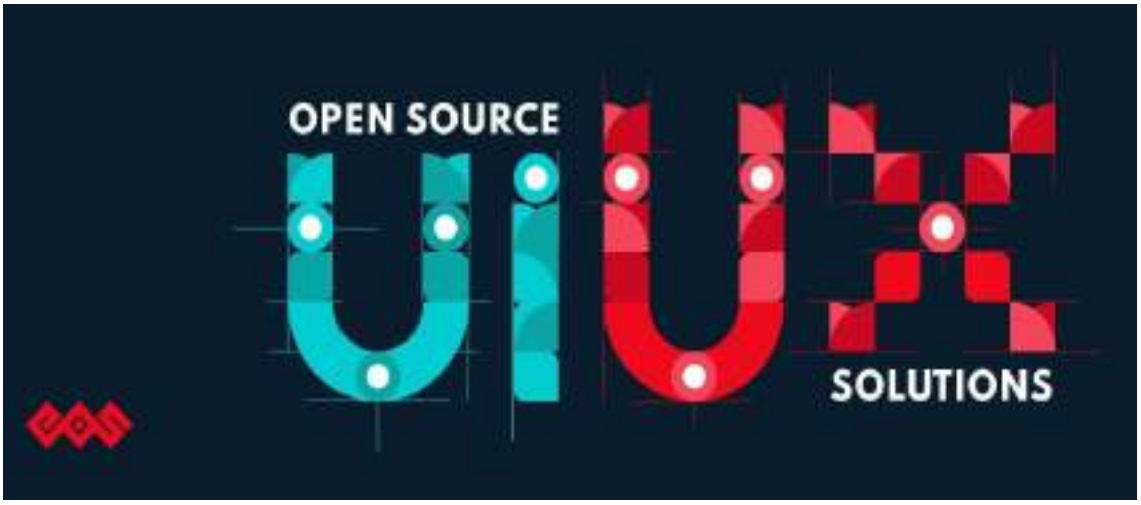


Python Software Foundation

Sub-org: EOS Design System
Project: EOS Icons react library
By: Vinayak Sharma

Contact information :

Name:	Vinayak Sharma
Email ID:	vinayaksh42@gmail.com
Github Username:	https://github.com/vinayaksh42
Gitlab Username:	https://gitlab.com/vinayaksh42
Mobile:	(+91) 6005857601 / (+91) 9839224762
Country:	India
Timezone:	**Indian Standard Time** (UTC +05:30)
Primary Language:	English
Linkedin:	https://www.linkedin.com/in/vinayak-sharma-141096193/

University Info:

University name: Shri Mata Vaishno Devi University, Jammu & Kashmir

Current year and Degree: 2nd Year, Bachelor of Technology in Electronics and Communication Engineering

Expected Graduation Date: August 2023

Code Contribution:

I have been contributing to the various different repositories of EOS since February and have made substantial contributions to it. Some of the major ones are listed below:

- **[Merged]** !43, !199: Addition of the Feature to copy SVG code of the selected icon, frontend UI of the feature as well as the backend route for providing the SVG.
- **[Merged]** !122: single graphql query for handling product and category filter
- **[Merged]** !198: Repositioning of the toggle switch to icons tab for better reach
- **[Closed]** !188: Background color testing while preview of icon
- **[Merged]** !190: Bug fix: Check HowToPanel state before switching tabs
- **[Merged]** Initial Commit: For creating EOS-Icons-Vue npm package
- **[Self-Repo]** Tree Shaking SVG Icons Library.

All other Pull Requests are listed **here** (17 Pull Requests). All Issues created are listed **here** (10 Issues).

Synopsis:

Currently EOS delivers icons via a set of well documented methods, which chiefly include EOS-Icons Package, CDN or direct download of icons in SVG/PNG formats. With the guidance and help of mentors, I want to develop an npm package that can deliver EOS Icons to its users as an independent component library and create a unified central system that will enable EOS to deliver its icons to various frameworks (e.g. React, Vue and Angular) independently. During the development process my chief goal would be to make this npm package as lightweight as possible in order to reduce the load put on the browser (using the tree shaking approach), which will create a smoother workflow for the users. Creating a central package of EOS Icons that can supply icons across React, Vue and Angular frameworks will go a long way in having a more consistent user experience as well as boost capability to accommodate users who migrate from one framework to another.

My qualifications to carry out this project arise from my sound knowledge of React, Angular and Vue frameworks. I have a robust experience in working with component libraries; I created a Vue-UI-preloader which sees over 1.7k downloads per month and has 60 unique users on github. During my contribution period with EOS I developed a trial EOS-Icons-React package which loads an icon using the tree shaking approach (500 - 600 B per icon). I have also contributed to the initial commit of the EOS-Icons-Vue package which was then merged to the repo.

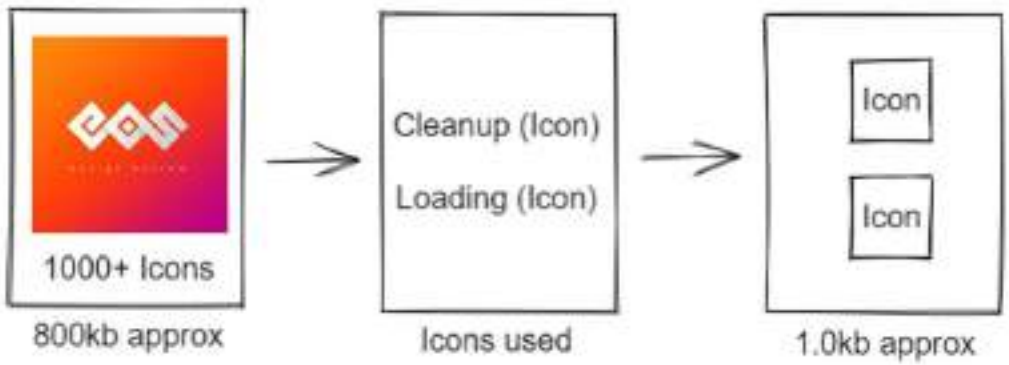
Apart from that, I am confident in my ability to pick up new skills and technology quickly should the opportunity to do so arise during my project.

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

I am interested in the Project-1 (EOS Icons React) of EOS Design System. My goal would be to create an npm package that can provide EOS Icons to various web frameworks such as Vue, React and Angular efficiently.

Project Plan:

The EOS Design System currently has 1442 icons and is only expected to only grow from here on out. Given the size complexity, I feel it prudent to ship icons by following the Tree Shaking approach, i.e. only ship icons that are requested by the browser instead of the whole icon library on request. This improves the web performance by significantly reducing loading time.

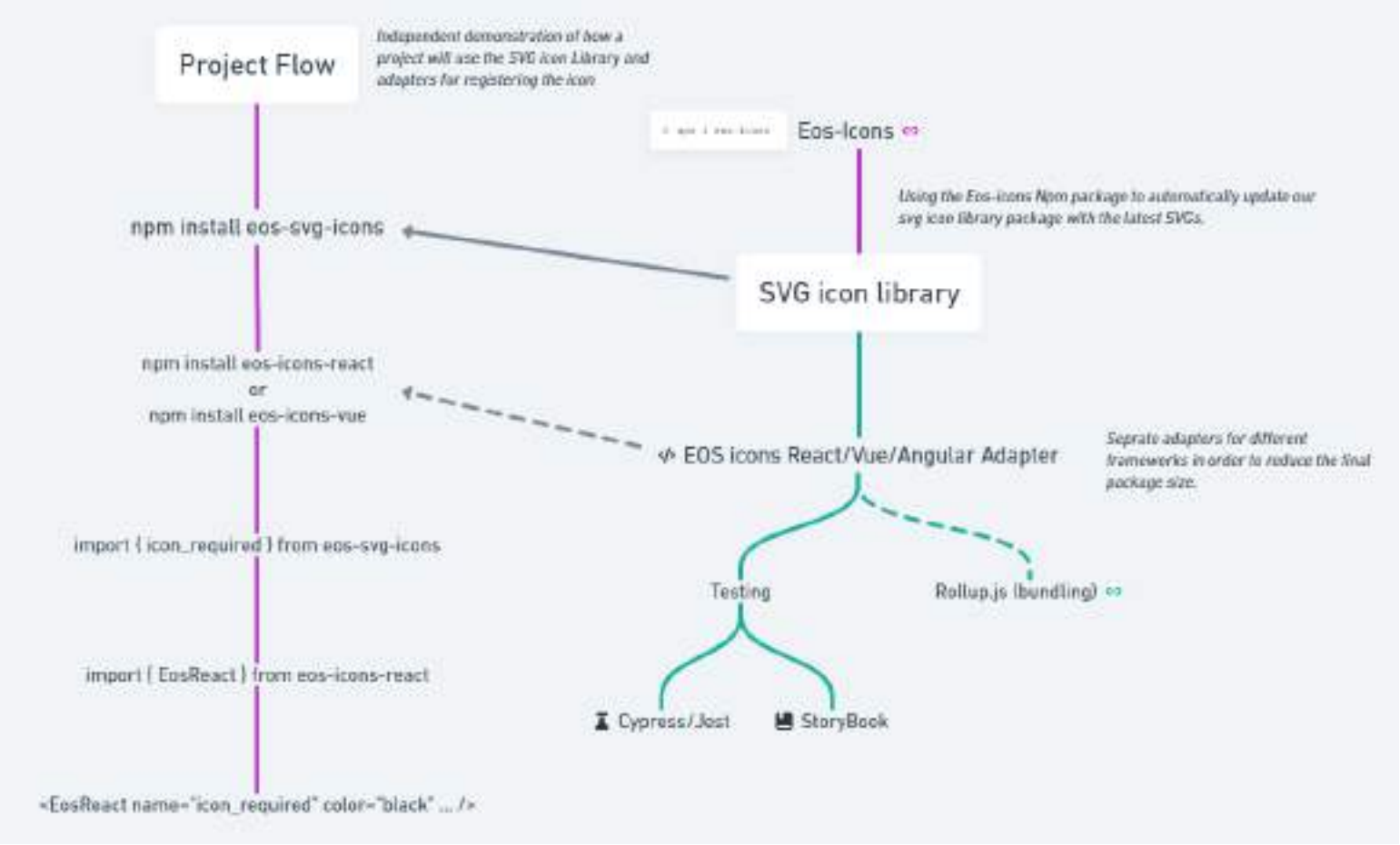


For instance, the EOS Icons npm package has a size of 3.19 Mb which becomes a significant bottleneck if the entire library is transferred on a single request. During my development of EOS-Icons-React Test Project I developed an icon library that deployed icons using the tree shaking approach and succeeded in reducing the frontend load of a single icon to 500 - 600 B *(Video example and further explanation included in this proposal)*.

Creation of a centralized npm package that deploys icons to all frameworks will reduce the complexity of package updates. Further, I also plan to include testing using Jest to make this package more robust.

The core SVG serving library will be used to provide SVGs to various framework adapters (e.g. React, Vue and Angular). The subsequent step will be to enable props on the SVGs. For building up the adapters we will use Rollup.js as it provides plenty of plugins and output format options. The last and the most important part of this project will be a proper documentation of "**How To Use**" the various adapters in respective frameworks.

The project steps can be discretized as depicted in the diagram below.



1) SVG serving icon library:
Currently, we have an EOS Icons package that contains all the latest EOS icons and is used by various EOS projects, e.g. EOS-Icon-picker (API), for providing SVGs to the EOS-Icons-Landing.

I plan on utilising this npm package for providing SVGs to the SVG serving icon library as it is the core container of all SVGs. Thus we will be able to fetch newly added SVGs directly from the package instead of having to manually update the SVG serving library, and thus save time and effort required for this operation.

We can utilize Dependabot for detecting version change and also use the github actions functionality for auto merging the Pull Request created by Dependabot.

Which would trigger the Github actions to rebuild and then republish our npm package along with the latest version of EOS Icons npm package.

For creating the SVG icon library we can use the svg-to-ts package, which is a helper utility that can convert raw SVG files into TypeScript files. This will create a strong base that will provide us with SVGs which can be further consumed by the adapters.

We can fetch all the SVGs from EOS Icons by using the operation below and then hand it over to the svg-to-ts package so that it can build our SVG serving library:

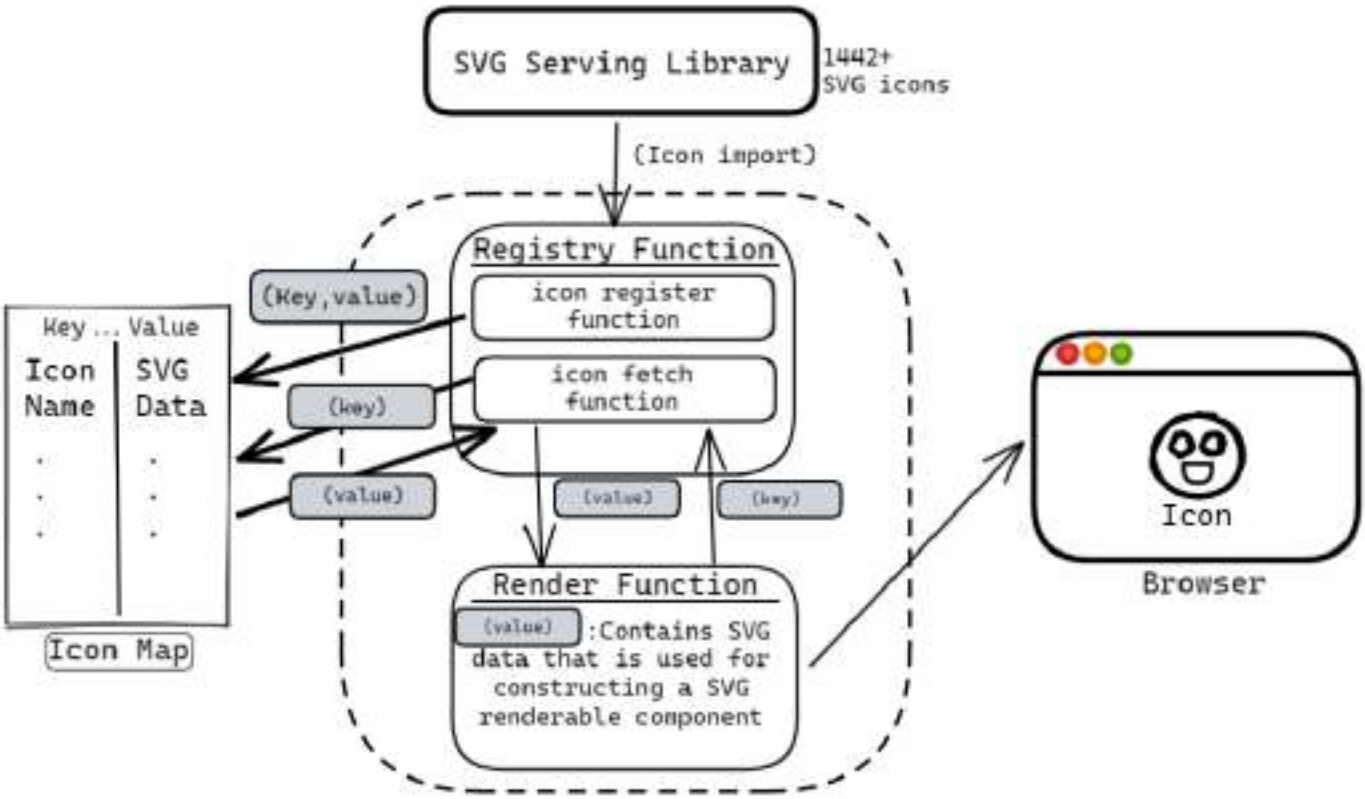
```
mkdir -p src/svg/ && cp node_modules/eos-icons/svg/*.svg src/svg/ && cp
node_modules/eos-icons/svg/material/*.svg src/svg/ && cp
node_modules/eos-icons/animated-svg/*.svg src/svg/
```

The key advantages of using Svg-to-ts npm package are:

- It can be configured to our needs easily.
- It provides Type safety to the SVG TypeScript files.
- It supports the tree shaking approach, which is key to creating our chosen package.

2) Adapters for React, Vue and Angular:

The next order of business will be to render the SVG Icons to the browser. The aforementioned library will provide us with the SVG data. This data will then be consumed by the adapter (component). The adapter is going to consist of a Registry function and a Render function. The flow of this adapter will be:



The adapter can be broken down into two main components:

1) **Registry Function**: As depicted in the above diagram this function is going to be responsible for storing the SVG data in a Map (key value pairs).
    a) The core logic of this function will be as follows:

```
const IconRegister = (IconArray) => {
      let IconMap = new Map()
      for (let i = 0; i < IconArray.length; i++) {
            IconMap.set(IconArray[i].name, IconArray[i].SvgData)
      }
}
```

In the first line of the above code, it takes in an array as an argument. This array will consist of the SVG data that is imported from the SVG serving library.

In the second line we declare a variable named “IconMap” which will contain all the key value pairs of the SVG name and SVG data.

Next we iterate over the array using a “For Loop” and pass down the SVG data to the map.

b) The fetch function will be as follows:

```
const IconFetch = (Icon) => {
      return IconMap.get(Icon);
}
```

The key (icon name) will be supplied as an argument to the function which is used for fetching the value from the Map. In case a wrong name is supplied we can add a check for that and return an error so that the user is aware about the wrong usage.

2) **Render Function**: This function will send a key (icon name) to the Icon fetch function of the “Registry function” and get back the required SVG data from the Map. This SVG data will then be used for construction of a renderable SVG component. This component will differ with respect to the framework that it is being used in. Each framework will have its own set of Registry and Render function. We will add our props and make changes to the style of the SVG icon in this component.

An example of the usage of the functions and the SVG serving library is as follows:

```
import { EosIconsRegister, EosIcons } from 'Eos-icons-adapter-React';
import { Loading } from 'Eos-icons-svg-lib';

const example = () => {

// passing the SVG data from the SVG serving library to the Icon Registry, Which
will create a Map
EosIconsRegister.IconRegister([Loading]);

return (
      <div>
            <EosIcons name="Loading" />
      </div>
      )
}
```

The above example is specifically for React. The structure for the other frameworks will be somewhat different.

3) Props for styling icon:

We are going to add different props for styling the SVG icon in the Render function. The base motivation for providing props is to eliminate the need for writing separate css by the user and instead use the props provided by the component for styling the SVG icon.

List of the props (styling) that I plan on adding:

1. Size (width and height of the SVG icon):
   Size of SVG can be modified using the Width and Height element of the SVG
2. Color (Fill/stroke color of the SVG icon):
   The color of the SVG can be modified using the fill property.
3. Rotation (in the same manner as it is available in the “Edit and download” option of EOS-icons-landing):
   Rotating an SVG can be achieved by using the `rotate()` Transformation
4. Flip (Same as rotation):
   Flipping of an Svg can be achieved by using the `Scale()` Transformation, transform: scale (-1, 1)
1. **Filled or Outlined**:
   As Eos is going to soon provide the user an option to pick between filled and outlined, we can also have it as a prop (more details below).

All of the above styling will be provided to the user in the form of props. In the future we can use the same method to add more customizations to our icons.

An example of props usage is as follows:

```
<EosIcons name="cleanup" size="12" color="#fcba03" rotate="170"
flipVertical="yes" flipHorizontal="yes" />
```

All of the above props will hold a default value so that in case the user decides to not provide any prop the component will simply render a default SVG icon.

For scalability purposes and better user experience, we can also ask the user to set up a config file within the project repo in order to set up a general config for their project.

The logical working of the above props would still remain the same but we will provide the user with an option to set up a config file within their project, so that they can have an initial theme setup for their icons. Using this config file the user can provide information for all the icons from a single file.

Props would overwrite the general config in case the user decides to change the styling of a particular icon from the general theme. Automatic generation of the config file using a CLI tool and other such options can also be explored for this particular use case.

4) Jest for testing:

I plan on adding Jest to test the components. This is highly necessary for this project as it will make the code more robust. The reason for why I chose Jest is that it works very well with React, Angular and Vue.
Also, I researched the other available Icon component libraries and font awesome, one of the most commonly used icon libraries also uses Jest to test their components. Not to mention, I have previously worked with Jest and hence will be able to write tests for the different components using this.

I would like to further discuss the topic of testing with the mentors in order to pick the best suited testing framework for this project.

5) StoryBook for testing props:

If we have time to spare during the coding phase, I would like to add StoryBook for testing the different props that we are supplying with the Icon adapter. We can really benefit from the usage of StoryBook as it will provide us with a Demo environment for testing our components.

Installation of storybook can be done by simply running this command:

```
Npx sb init
```

This will scaffold a basic storybook setup in our project. StoryBook looks into our project dependency and automatically configures itself for the best possible usage.

An example implementation of a component in StoryBook can be done in the following way:

```
Import React from 'react';
import { storiesOf } from '@storybook/react';
import { Wifi } from '../components';

const stories = storiesOf('Test', module);
stories.add('App', () => {
    return (
        <Wifi/>
    )
})
```

6) Rollup.js for bundling library (Publishing):

Automation of the publishing process can be done using Github actions. Semantic versioning can be configured in the same way as it is configured for the EOS icons repository on Gitlab. The tag used in the commit messages can be utilised for automating the publishing process and ensuring proper versioning. The workflow that will be used for publishing can include check for Jest tests. The same workflow can be utilized in all the different adapters (React, Vue and Angular) as well as for the SVG serving library.

Ultimately for publishing/building our Icon library we are going to require a module bundler for bundling the project. Rollup comes with a lot of configurable plugins

and output format options. I recently used Rollup for bundling the Eos-icons-React test project.

The config used for the Eos-icons-React test project is as following:

```
import babel from 'rollup-plugin-babel';
import resolve from '@rollup/plugin-node-resolve';
import external from 'rollup-plugin-peer-deps-external'
import { terser } from 'rollup-plugin-terser'

export default [
    {
        input: './src/components/index.js',
        output: [
            {
                file: 'dist/index.js',
                format: 'cjs'
            },
            {
                file: 'dist/index.es.js',
                format: 'es',
                exports: 'named',
            }
        ],
        plugins: [
            babel({
                exclude: 'node_modules/**',
                presets: ['@babel/preset-react']
            }),
            external(),
            resolve(),
            terser(),
        ]
    }
]
```

The above Rollup config can be used in the adapters with some minor changes. In the above configuration Rollup is targeting the src/components/index.js folder and outputting it in two formats (common JS - cjs and Es.js).

Rollup comes with a plugin named ‘terser’ which minimizes the final build of the project. It will be of great help to us in order to minimize our package before actually publishing it.

7) Documentation:

The documentation part of this component Library is really important and is going to be a decisive factor in how many users are actually going to be able to understand and use it. The essential material that we should absolutely cover under this documentation should be:

- Step by step instruction on how to utilize the package for different frameworks.
- List of all the props that can be passed to the components in a tabular format.
- Basic versioning updates of the package as we go on to bring more updates to the package.

The documentation should also be added to the [EOS-Icons-landing website](). We can have different tabs for the different frameworks in the following manner:

With the help of collaboration with the student who will be working on the landing website of Eos Icons we can also integrate the code snippets in the “Edit and Download” modal present at the EOS-Icons-landing website. We can provide the user with a direct code snippet that will live update according to the modifications that the user will do to the Icon.

An example of the idea suggested above:

1. User selects an icon from the EOS-Icons-landing website.
1. User opens the Edit and download modal for editing his chosen icon.
2. At the start the user is given a basic code snippet which will look like:

```
<EosIcons name="selected-Icon" />
```

3. User then changes the color of the icon to #36507 and rotates the icon by 180°, This will automatically update the live code snippet to:

```
<EosIcons name="selected-Icon" color="#36507" rotate="180"/>
```

4. Now the user can simply copy and paste this code snippet inside his project.

The above implementation will be of great value to EOS-Icons and make it really easy for the users to use and modify EOS Icons. More such features can be integrated in EOS-Icons-landing website in the future.

What have you done so far with this idea?

I have researched the currently available Icon libraries and found that all the popular ones use a Tree shaking approach. Iconify calls its approach as “Load on demand” which simply means that it will only render the icons that are being currently used in the application. I want to build this library with the same concept.

While researching and trying out some implementations of the tree shaking methodology I ended up building a SVG library for React which had an overhead of 0.5 Kb per Icon. I also gave the direct approach a try as well while building the EOS-Icons-Vue library. Elaboration on both the implementations:

1) React Tree Shaking SVG library:

In this approach all the SVGs provided by EOS-Icons (npm package) are converted into a React component using svgr/cli. All the Icons are then automatically registered in a separate index.js file. This file is then used as the entry point of the library. I used Rollup.js for building this library. Essentially, all the SVG React components act like one big registry, which contains all the SVG icons ready to be rendered on demand by the user. Only the Icon that is being called by the user puts load on the browser, while the rest of the icons remain on hold.

To test my above implementation I also configured a webpack analyzer to visually confirm the actual load that is being put on the browser. According to Webpack analyzer the “Loading” SVG used an exact space of 749 B (parsed).

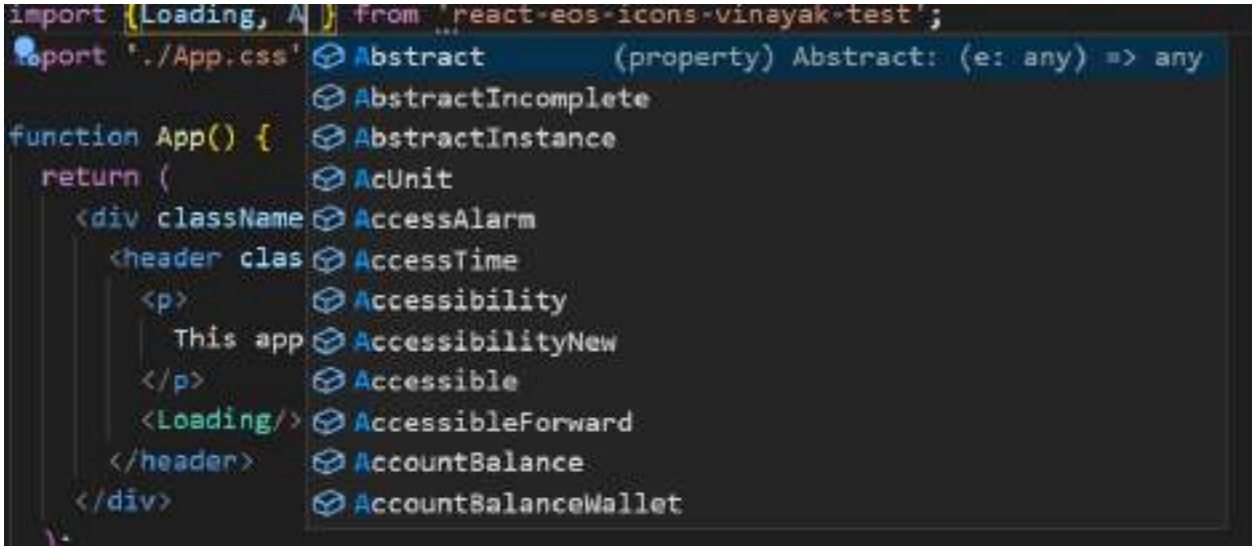
To test my implementation further, I decided to add another icon. As expected the load increased to almost 1 Kb (1.04 Kb).

Hence webpack analyzer proves that the above implementation uses a tree shaking approach. For reference, Iconify also claims to occupy only 0.5 Kb per icon, which is similar to what I achieved from the above implementation.

Other than the actual implementation I also implemented StoryBook for testing my component first hand. Due to StoryBook I did not have to set up a separate repo for running my component, which saved me a lot of time.

This component also supports IntelliSense. As soon as I start typing the name of the icon it starts to suggest to me icons based on keywords. I would love to add this feature to the final project as well because it is really helpful while importing a

new icon.



Click here to view the live demo

Github repo of the above project.

2) Vue EOS Icons (direct SVG approach):

The approach used in this implementation is rather direct in nature. EOS-Icons (npm package) is used once again to provide all the SVGs but instead of converting them into a Vue component I simply utilised “V-html” (a Vue directive) for rendering the SVGs to the browser. Vue comes with SVG support out of the box but it required me to set up a Vue.config.js file in order to configure SVG rules for the v-html loader. For building up this project into a component library I used the “Vue-cli-service build” command. It's a command that comes along with Vue.js but if required we can switch to Rollup.js in order to maintain a consistency among the different adapters.

In this project i also added a few props like Icon name and size of the icon:

```
  props: {
      name: {
     type: String,
     required: true,
   },
    size: {
     type: String,
     required: false,
```

```
    default: '10'
  },
}
```

The size of this library was almost equivalent to the actual size of all the SVGs combined:

Although the size of the final build is more than what we expected it to be, this project still provided me with a very good experience of working with props and the complexities faced while rendering SVGs in a Vue based application. I am sure this information will be of great use while creating the actual adapter for Vue.js framework. Github repo of the above project.

Timeline:

Community Bonding (May 17 - June 6):

I will utilize this time duration for planning out my goals as well as discussing the workflow of this project with my mentors. I will also adjust myself to the EOS workflow and start to work according to the scrumban methodology. I will have my semester exams from May 25th to June 4th, so I will be only able to dedicate 2-4 hours a day. After exams are over, I will be able to dedicate more than 6 to 8 hours per day. I will also use this community bonding time duration to be familiar with all the fellow GSOC 2021 participants as well as all the mentors.

Week 1 (June 7 - June 13):

For this week, my goal would be setting up the repository for the SVG serving library and configuring EOS Icons (npm package) for providing the SVG serving library with all the required SVG files. I will also add svg-to-ts to the project so that it can work with the SVG files provided by EOS Icons.

Output of this period:

- A repo setup with initial setup (svg-to-ts) of the SVG library.
- EOS Icons configured for providing SVG.

Week 2 (June 14 - June 20):

For this week, my goal would be configuring the Svg-to-ts helper utility according to the requirements of the project and setting up rollup.js or any other bundler for building this project and publishing it on npm. I will also utilize this week to know more about semantic versioning and configure the SVG serving library to be fully automated and strictly following the Semantic versioning specification.

Output of this period:

- Svg-to-ts configured for providing the SVG files.
- Npm package ready to be published.

Week 3 (June 21 - June 27):

For this week, my goal would be to set up a new repo for the React Adapter. Setting it up would involve creating the initial development environment for a React package. Linking the SVG serving library to this repo in order to proceed with the further development of the project. Construct the core Registry function for the react adapter as well as work on the Rendering function. Linking them together for rendering a basic SVG.

Out of this period:

- Initial repo setup for the React adapter.
- Creating the registry and render function for React.

Week 4 (June 28 - July 4):

Adding the different types of prop to the Render function. The props will include:

    - Size
    - Color
    - Rotation
    - Flip vertical/horizontal

Setting up default cases for the above props as well as Error checks on various different wrong use cases. Configuring Rollup.js for publishing this package to npm.

Output of this period:

- Addition of props.
- Error check on registry and render function.
- Rollup.js setup for publishing.

Week 5 (July 5 - July 11):

This week will be the first evaluation. I will add Jest to the React adapter repo in order to improve and test all the code that I have implemented until now. I will work closely with the mentors and try to improve the codebase as much as possible and ensure the proper functioning of both the packages.

Output of this period:

- Addition of Jest for testing the components.
- Submit feedback for evaluations.
- Package ready to be published on npm.

Week 6 (July 12 - July 18):

Initial setup of the Angular adapter. Linking it with the SVG serving library. Creating the initial development environment for an Angular package. Construction of the Registry as well as the Render function for the Angular Adapter (will require a bit less time as it will be based on the same logic that was used for creating the React adapter). Addition of props to the Render function. Same as the props that will be added to the React adapter.

Output of this period:

- Creating the registry and render function for angular.
- Addition of props.

Week 7 (July 19 - July 25):

This week I will work on configuring Rollup.js for the Angular adapter. Implement Error checks on the registry as well as the render function. Setup of Jest for the Angular adapter as well as basic test implementation. Working towards fully configuring the Angular adapter in order to make it ready for publishing.

Output of this period:

- Rollup.js configured for publishing.
- Jest setup for testing out the components.
- Package ready to be published on npm.

Week 8 (July 26 - August 1):

From this week onwards I will start working on the Vue adapter. Setup the initial environment required for creating a Vue based package. Create the Registry as well as the render function for the Vue adapter. Addition of props to the Render function in the same manner as it was for React. (this adapter might require some extra time compared to the other two adapters as the way of rendering SVGs in Vue is a little bit different.)

Output of this period:

- Initial setup of the Vue adapter.
- Registry and render function for Vue adapter.
- Addition of props.

Week 9 (August 2 - August 8):

This week will be utilized for configuring Rollup.js or Vue-cli-service build for publishing the package to npm. If time permits I will also try to add tests for this adapter using Jest. Work on improving the codebase in order to push it towards publishing.

Output of this period:

- Package configured to be bundled.
- Addition of Jest for testing.

Week 10 (August 9 - August 15):

This time will be used for creating the docs for the SVG serving library as well as the various different adapters. I will also construct the various different documentation sections for the various frameworks (React, Vue and Angular) under the EOS-Icons-landing website. These sections will be almost similar to the general documentation that will be constructed for each package. If possible i will also try to add the live code snippet feature to the EOS-Icons-landing website.

Output of this period:

- Construction of the docs of the various different packages on the website as well as for npm.
- Live code snippet feature for EOS-Icons website.
- Final project submission.

Tell us about your previous experience:

My main area of work till now has been around JavaScript. Most of my self projects as well as the contributions made to open source projects are in either JavaScript or Golang. Initially I started with basic Html, css and JavaScript then moved to various different web frameworks in JS, like Vue and React.

I have learned C and python as a part of my university courses. In my first year I did an internship at Apicon.io as a Software engineer intern. The projects on which i worked during my internship were based on ApexCharts, Docusaurus, Kanban system using React-beautiful-dnd, XLSX-npm package, Mobile app using React.js, Redis implementation for caching, API calls count and API analytics dashboard.

I am a very active open source contributor at layer5, I have developed a lot of sections and pages for their main website. I am also the maintainer for one of their projects, which is Getnighthawk.

Other than this I have taken part in many coding based competitions at my university as well as outside. I secured 1st position at a competitive coding competition held during the Technical Fest of SMVDU in my first year in college. I also secured 3rd position with my project image security (blog) at DevHack hackathon held by Deta. I have mentored 10+ teams working on projects based on VUE.js during the HobbyHacks hackathon (sponsored by Microsoft). I was the technical team lead for the hackathon HackTheMountain (sponsored by MLH), developed their website, web portal for query and the formal website of the organiser.

Tell us a bit about you:

Everyone on this planet has something that they are really passionate about, for me it is programming. My interest in programming originated from video games. Even though game development is a field that I have not yet explored a lot, I would definitely love to know more about it in the future. I play a lot of First Person Shooter games like Counter Strike: Global Offensive, Call of Duty and Valorant

(started playing it recently). I also love playing Real-time strategy games like Age of empires, sandbox games like Minecraft and Roblox. I like to watch Sci-Fi movies like Star Wars, Matrix, Source Code, Minority Report, etc. After joining university I have started playing lawn tennis and found I quite enjoy it. I am also the Outreach media head of my university's Competitive Coding Club (formed by CodeChef).