PYTHON SOFTWARE FOUNDATION

Sub Org: EOS Design System

Project: **User Story - Improvements and new features**

By: Sundeep Chand

Contact Information:

- Timezone: UTC +5:30
- Country: India
- Gitlab username: SundeepChand
- Github username: SundeepChand
- LinkedIn: sundeep-c-418862135

- University: Delhi Technological University
- Program: B.Tech in Computer Engineering
- Current Year: 2nd
- Expected graduation date: August 2023

Code Contributions to EOS:

Contributions to eos-user-story:
- Merge requests
- Created issues

Contributions to eos-strapi:
- Merge requests

Contributions to eos-icon-picker:
- Merge requests

Project Background:

EOS is an open-source, and customizable Design System for front-end developers and UX designers at SUSE and SMEs to efficiently deliver and implement industry-standard practices that will help brands achieve consistency and success.

The EOS User Story project aims at providing a user-friendly and interactive platform for users to sign in and request new icons, report bugs, etc for EOS Icons. The users can track the progress of their issues, vote and comment on issues that are of interest to them. The project admins can then review, resolve, close or assign a status to the issues. It serves as an efficient feedback mechanism for organisations, which is necessary for the development of different products of an organisation. The project is totally reusable and can be used by any other organisation for their products.

Synopsis:

Most of the basic features of the EOS User Story project are developed but some important features are yet to be implemented. I'm going to improve the project by adding the following features:

1) Request Templates - This feature will enable the admins to configure custom templates for certain products. This will provide users the guide to fill the story details as well as speed up the review process of the stories.
2) Shareable Links for Stories - This feature will provide the users the ability to copy the URL of a particular story with the single click of a button, which can then be shared.
3) Searching for Stories - This will enable the user to search for stories by title.
4) Shareable Search results - This feature will generate unique links for each of the search filters that are applied to the stories, so that the results can be shared easily.

5) OAuth Integration - Integrating OAuth LogIn will streamline the user experience.

In addition to these features, I will also be working on adding unit, integration and end-to-end tests to the project using Jest and Cypress. After implementing the tests I will also integrate them to the Gitlab CI/CD Pipelines. This will eliminate the need for manual testing of the future Pull Requests.

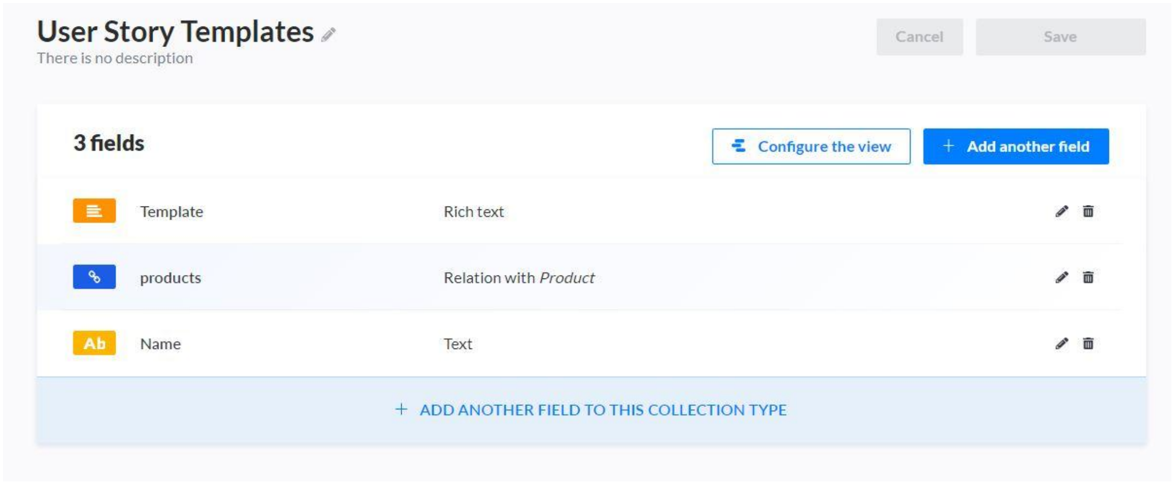
User Story Improvements:

1. Request templates:

(View video in GDrive) (View video in youtube)
(Link to eos-strapi branch) (Link to user-story branch)

This feature will provide the project admins the ability to configure custom request templates for a product. This template will provide the user some hints about the request format and would streamline & speedup the process of request reviews.

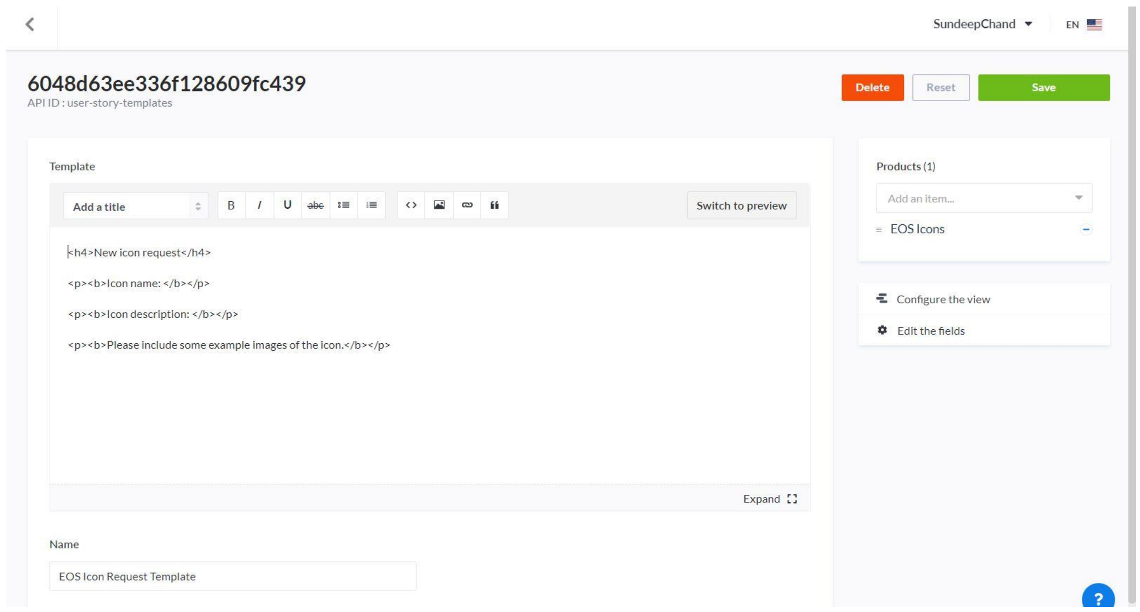
- Research done so far:
I've created a collection type User-Story-Template in strapi, which can be associated to a product, by a one to many relationship. So basically a template can be used by many products whereas a product can have a single template.


User Story Templates
There is no description

3 fields

Template	Rich text
products	Relation with *Product*
Name	Text

Then templates can be configured in the strapi admin as shown in the image below:



6048d63ee336f128609fc439
API ID : user-story-templates

Template

```
<h4>New icon request</h4>
<p><b>Icon name: </b></p>
<p><b>Icon description: </b></p>
<p><b>Please include some example images of the icon.</b></p>
```

Name: EOS Icon Request Template

Products (1): EOS Icons

Since the template collection is a part of the product collection, the template for a product can be fetched as shown below:

```
query {
  products {
    id
    Name
    user_story_template {
      Name
      Template
    }
  }
}
```

This template content is updated in the frontend when a product is selected, by passing the templateText as data prop to CKEditor.

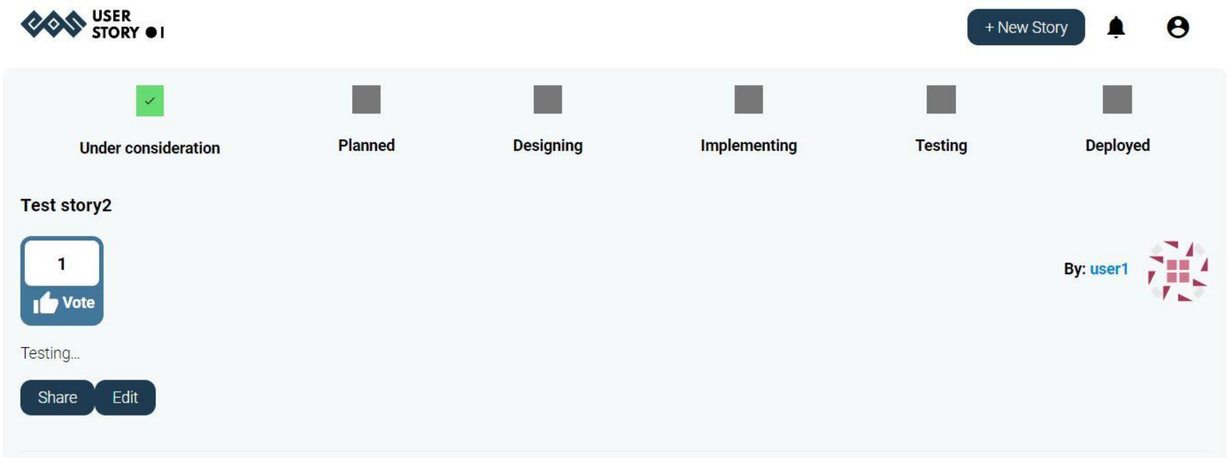
2. Shareable Link for Stories:

(Link to user-story branch)

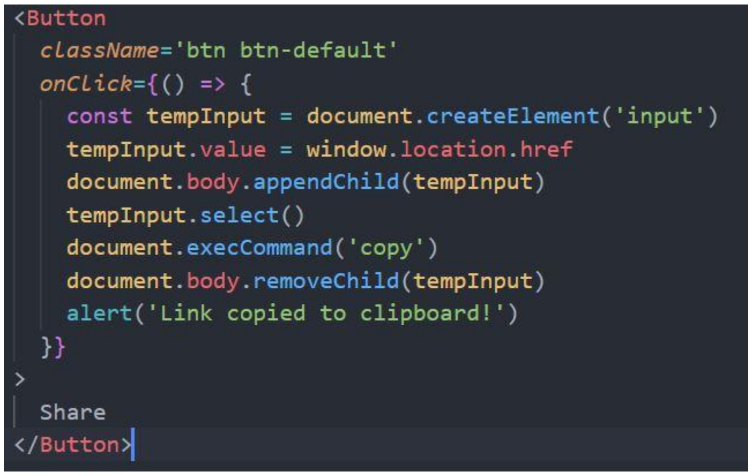
This feature will provide the users the ability to copy the URL of a particular story with the single click of a button, which can then be shared.

- Research done so far:

I've placed the share button as shown in the image below inside the story page:

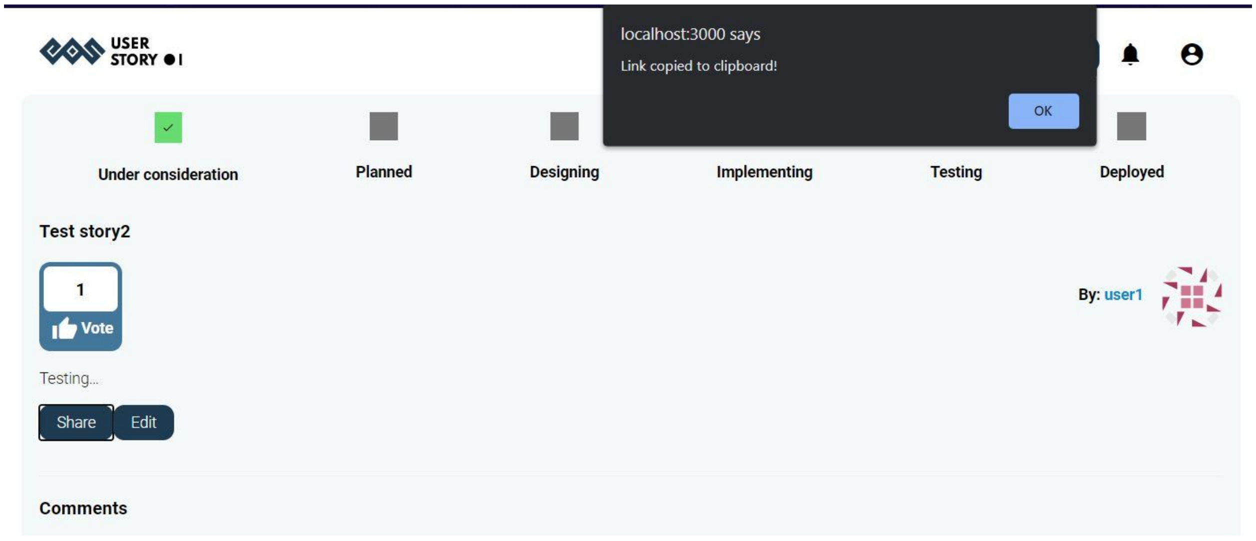


The code to copy the current URL is as shown below:



```
<Button
  className='btn btn-default'
  onClick={() => {
    const tempInput = document.createElement('input')
    tempInput.value = window.location.href
    document.body.appendChild(tempInput)
    tempInput.select()
    document.execCommand('copy')
    document.body.removeChild(tempInput)
    alert('Link copied to clipboard!')
  }}
>
  Share
</Button>
```

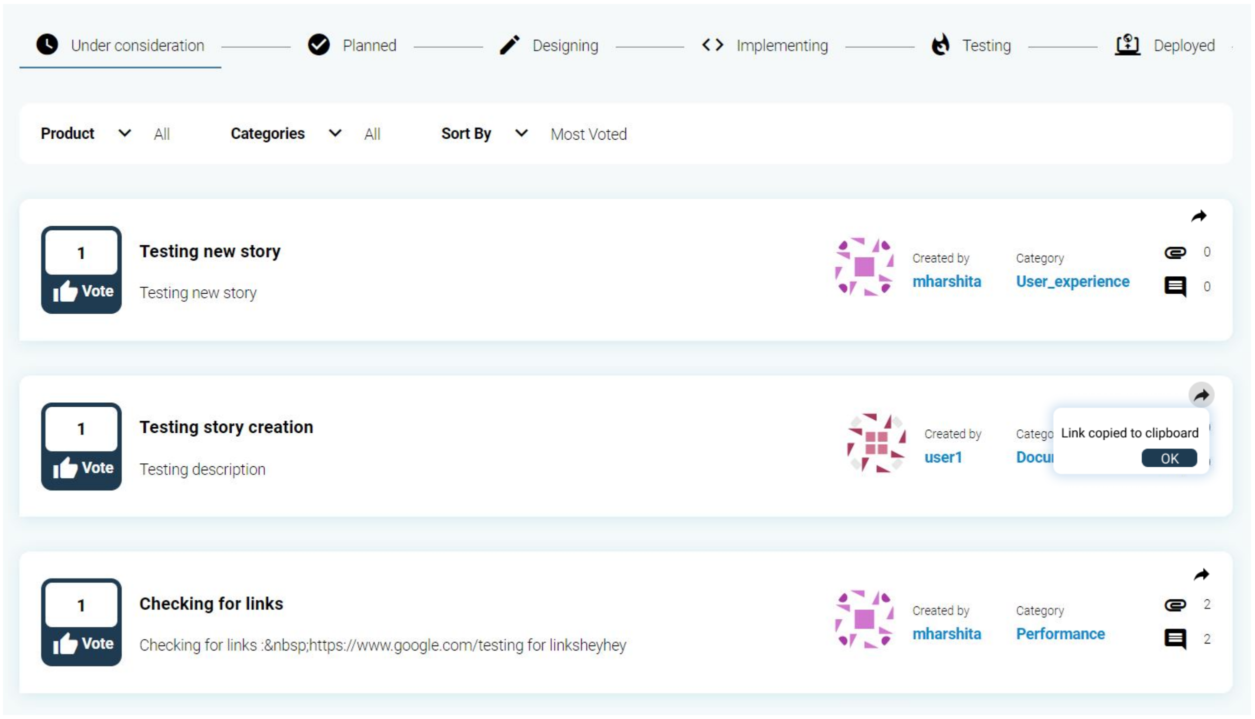
And when user clicks on the share button, the current URL is copied to the clipboard (instead of the alert, we can show a nice modal):



localhost:3000 says

Link copied to clipboard!

I'm planning to put the share button on the Home page as well. Here is a design of the UI (Link to design):



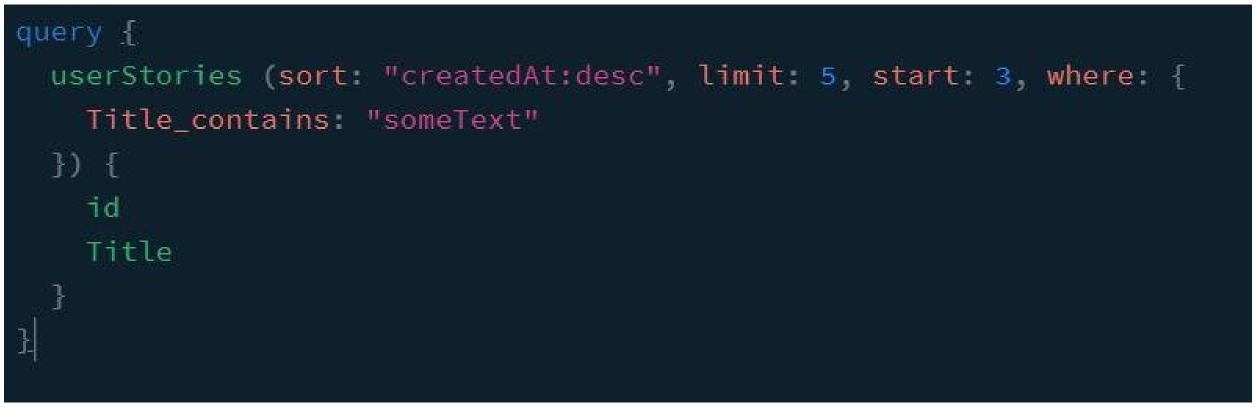
3. Enable Searching of Stories:

(View video in GDrive) (View video in youtube) (Link to user-story branch)

This feature is going to enable the users to search for stories by title.

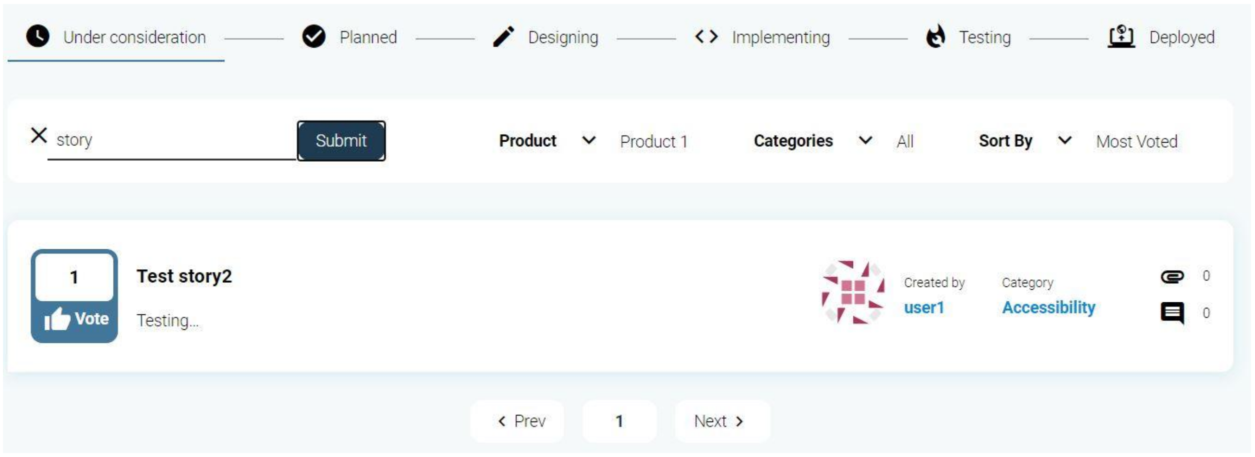
- Research done so far:

In order to implement the search, I used the default "where" alongwith "field_contains" filter provided by strapi. My implementation matches whether the story title contains the search text & returns a result based on that. Here is the code to fetch stories containing `someText` in their title:



```
query {
  userStories (sort: "createdAt:desc", limit: 5, start: 3, where: {
    Title_contains: "someText"
  }) {
    id
    Title
  }
}
```

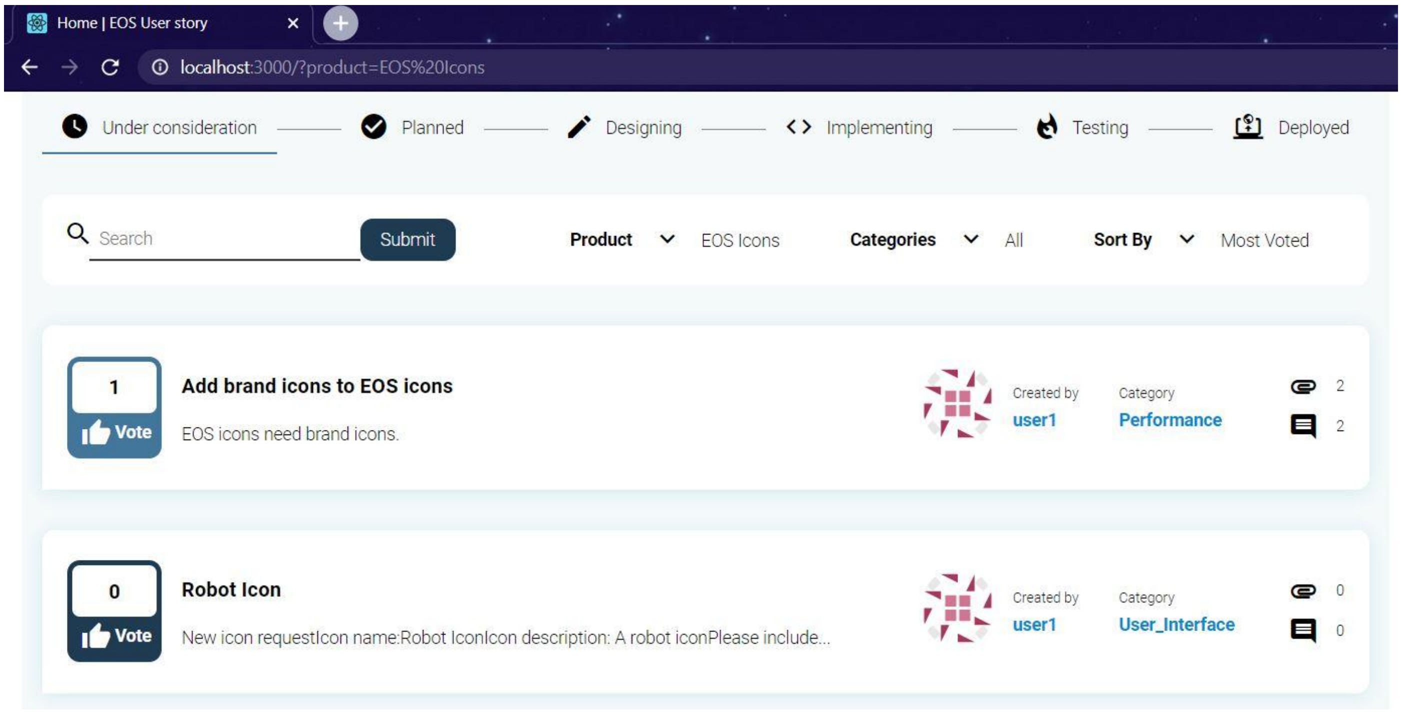
I've used the results from this query to implement the search. The code for handling pagination was already written, so implementing this will be a little straightforward.



But I think implementing a search based on matching the title along with description, would require a custom search endpoint to be implemented in strapi as I'm unable to find a way of applying "OR" operation through default strapi graphql while querying the backend.

4. Shareable search result links:

In order to improve the search functionality further, I'm looking at ways to make the search results shareable. The approach I'm currently trying is to update the Frontend route with query parameters when a search filter is applied. This will give a unique link for each combination of search filters which can be shared easily. So opening a link like 'http://localhost:3000/?product=EOS%20Icons' would open the product filter set to 'EOS Icons' by default. Currently I've implemented this for the product field (shown in the screenshot below). I've used an npm package `query-string` to parse the query string & update the frontend state variables accordingly to display the results.



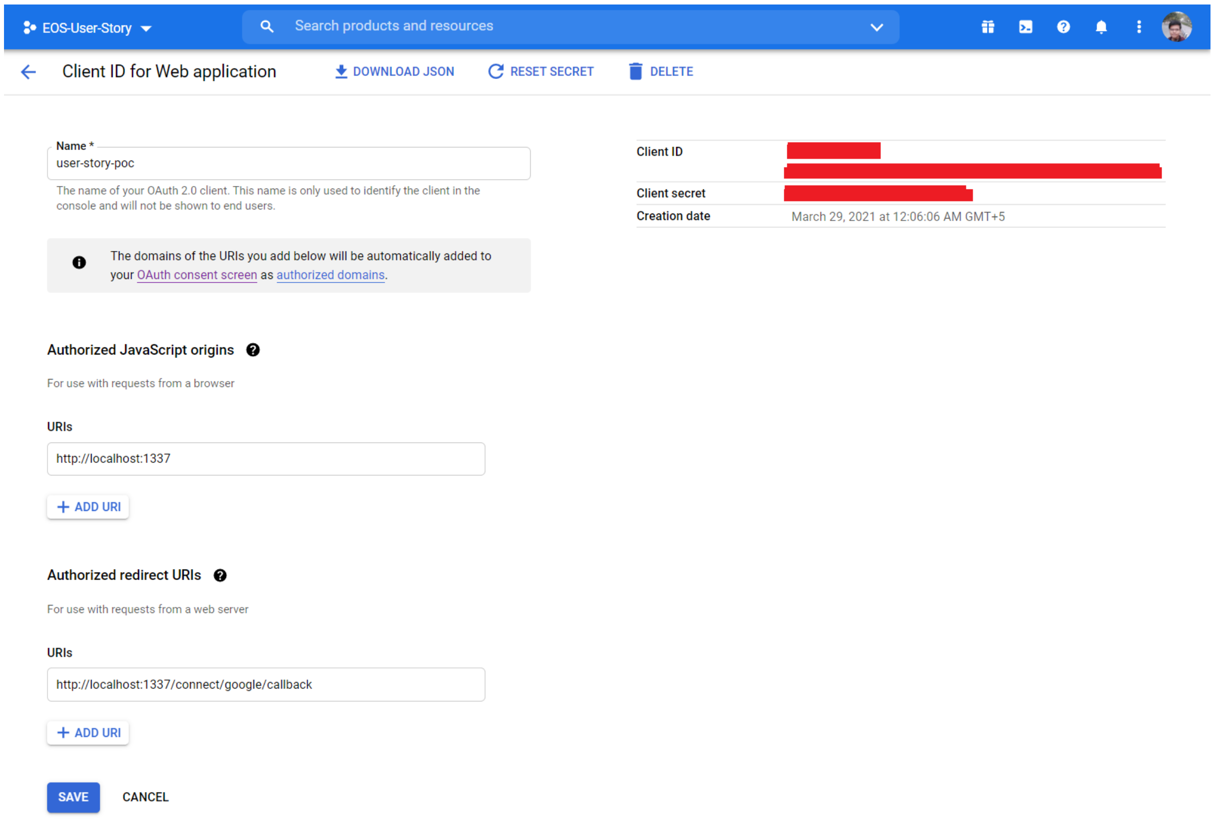
5. Integrating OAuth SignUp/Login:

(View video in GDrive) (View video in youtube)
(Link to eos-user-story branch) (Link to eos-strapi-branch)

Integrating OAuth will help provide easy & quick Login options to users. This will make the website more accessible and will make the UX frictionless.

- Research done so far:

I integrated OAuth using Google provider. The link to the demo of that is provided above. Integrating the Google OAuth provider requires the project to be registered in Google Cloud console, and after the setup we get an OAuth Client ID & a Client Secret that are used to communicate with the OAuth provider.



Name *
user-story-poc
The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorized domains.

Authorized JavaScript origins

For use with requests from a browser

URIs

http://localhost:1337

Authorized redirect URIs

For use with requests from a web server

URIs

http://localhost:1337/connect/google/callback

Client ID	[redacted]
Client secret	[redacted]
Creation date	March 29, 2021 at 12:06:06 AM GMT+5

Integrating other OAuth providers would require a similar process of registering the application in their Developer consoles and setting up the Client ID & Client Secret.

I am planning to integrate Google, Github, Twitter & Facebook OAuth providers as part of this project.

6. Improving test coverage:

(View video in GDrive) (View video in Youtube)
(Link to eos-user-story branch) (Link to eos-strapi branch)

As the application grows in size, it becomes necessary to perform checks before putting it to production/merging any patch. So I'll be adding more unit/end-to-end tests using Jest/Cypress testing libraries respectively.

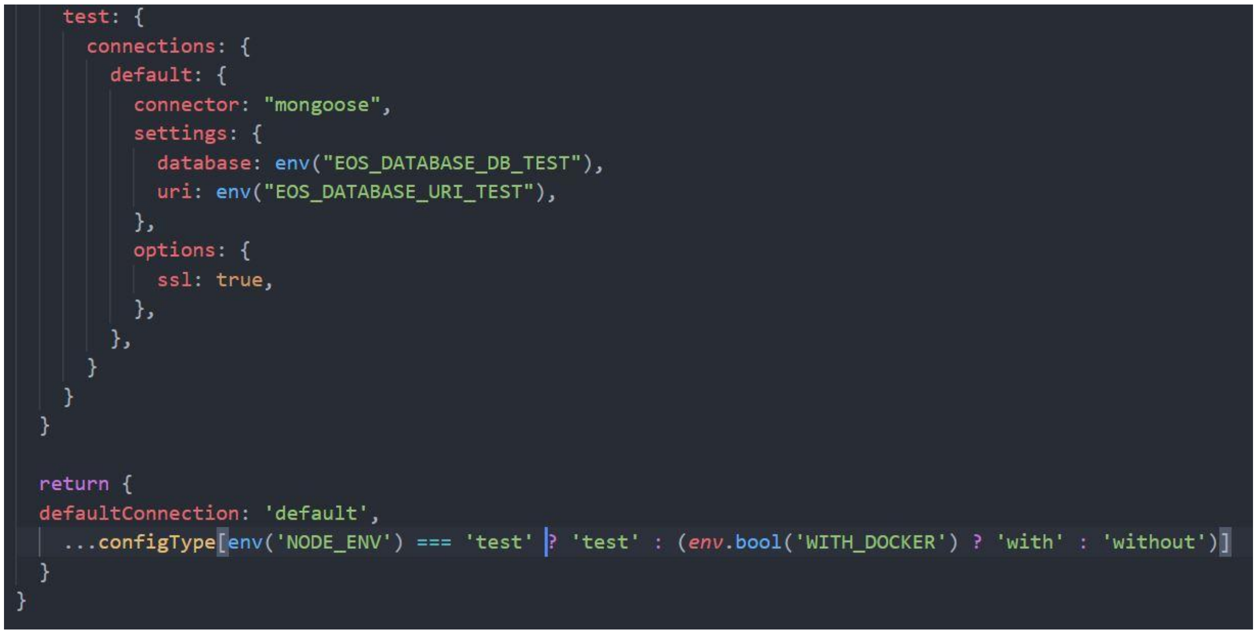
The tests can be divided into 3 categories:
a) End-to-end tests - They check the entire flow of a feature in the application.
b) Integration tests - They check how a unit integrates with other components

c) Unit tests - They check the individual standalone components of the codebase.

The tests mentioned above are in the decreasing order of the resources they need to run. So an end-to-end test will take the most resources (as well as time) while a unit test will take the least resources to run in the CI pipeline. Hence having a large number of unit tests and very few end-to-end tests is desirable.

- Research done so far:

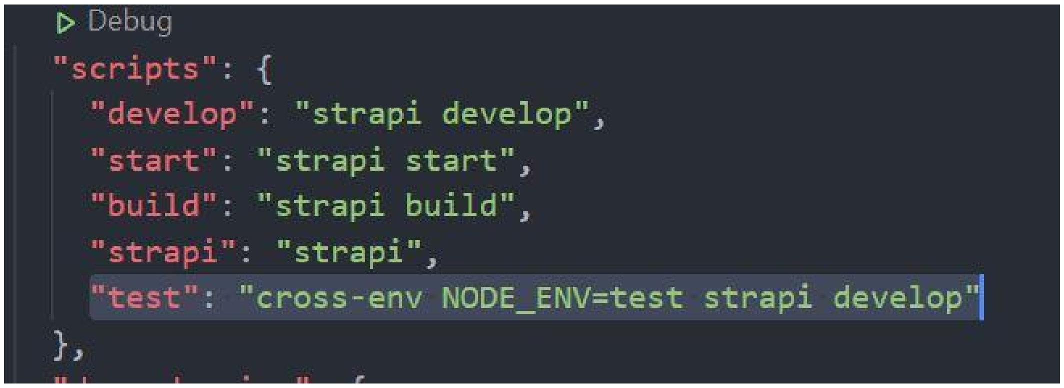
a) Setting up a test environment for end-to-end tests:
Currently I'm planning to connect to a separate MongoDB instance in order to run the end-to-end tests, so that it does not affect the data in the production/development database. Here is the configuration to connect to a new database:



```
    test: {
      connections: {
        default: {
          connector: "mongoose",
          settings: {
            database: env("EOS_DATABASE_DB_TEST"),
            uri: env("EOS_DATABASE_URI_TEST"),
          },
          options: {
            ssl: true,
          },
        },
      }
    }
  }

  return {
  defaultConnection: 'default',
    ...configType[env('NODE_ENV') === 'test' ? 'test' : (env.bool('WITH_DOCKER') ? 'with' : 'without')]
  }
}
```

And here is the command to run strapi in test mode:



```
▷ Debug
"scripts": {
  "develop": "strapi develop",
  "start": "strapi start",
  "build": "strapi build",
  "strapi": "strapi",
  "test": "cross-env NODE_ENV=test strapi develop"
},
```

It uses the `cross-env` npm package to set the `NODE_ENV` environment variable's value.

b) Writing tests:

Given below are the specs that I'm planning right now for implementing the tests. Most of these tests can be implemented as unit/integration tests. And certain user-workflows in these specs can be implemented as end-to-end tests.

I will modify these if I find a better workflow, or based on any feedback that I receive from my mentors:

User is not logged in	**User is logged in**
Navbar: - Displays only EOS Logo & a sign in button - Logo should link to / - Sign in button should link to /login	Navbar: - Displays EOS Logo, new story button, notification & myAccount - New story button links to /newStory - Clicking on notifications shows unread notifications & has link to /notifications - My account displays username, email & has links to /myStories & /myProfile - Has logout button
Homepage (/): - Stories are loaded properly in different sections - Stories can be filtered. - Users cannot vote stories - Clicking on a story should open the story page - Clicking on the author should open author's profile - Pagination should load more stories	Homepage (/): - Same as not logged in case - User should be able to vote/un-vote stories
Register page (/register): - Should have relevant fields - Error messages are displayed correctly when some field is empty. - Terms & conditions should link to /policies	Register page (/register): - Should redirect to home

- 'Existing user' should link to /login - User can register	
Login page (/login): - Should have relevant fields - Error messages are displayed correctly when some field is empty. - 'Forgot password' links to /forgotPassword - Has link to register page - User is able to login	Login page (/login): - Should redirect to home
Forgot Password (/forgotPassword): - Should have relevant fields - Error messages are displayed correctly when some field is empty. - User is able to submit a request.	Forgot Password (/forgotPassword): - Should redirect to home
NewStoryPage (/newStory): - Should display login form	NewStoryPage (/newStory): - Should display new story form - User should be able to create a new story - The new story appears in home page
Story Page (/story/:storyId): - Should display story status - Story title, description is displayed - Story comments are displayed - Replies to comments are visible	Story Page (/story/:storyId): - Same as not logged in - Should display edit button, if the author is current user - User is able to update the story - User should be able to comment on the story & reply to comments
My Stories Page (/myStories): - Should display login form	My Stories Page (/myStories): - Should display the stories created by the user & the stories the user follows. - Should link to the stories
My Profile Page (/myProfile): - Should display login form	My Profile Page (/myProfile): - Should display user info - User should be able to edit their info - Should link to /changePassword
User Profile Page (/profile/:profileId): - Should display user info	User Profile Page (/profile/:profileId): - Same as not logged in

- User info cannot be edited - Stories can be filtered	
Notifications (/notifications): - Should display login form	Notifications (/notifications): - Should display the notifications
Change Password (/changePassword): - Should display login form	Change Password (/changePassword): - Should display relevant fields - Should display error messages properly - User should be able to change password
Policies (/policies): - Should display the privacy policy	Policies (/policies): - Should display the privacy policy
Footer - Should link to /policies	Footer - Should link to /policies
	Sign out: - Should sign out the user, clear localstorage & cookies

- A large number of them, like testing the links, the content on the page, etc can be implemented as unit tests for individual react components (using Jest and react-testing-library).
- Certain specs like users can vote/un-vote stories are suitable candidates for integration tests.
- And certain mission critical user work-flows like user can login, user can change password, etc can be tested using end-to-end tests.

a) I've implemented the specs for the navbar (as unit test) using Jest. Given below is a part of the test where I am testing the state of the navbar when the user is logged in. For this I have used a package called `jest-localstorage-mock` to mock the localstorage for testing. We can use this package to test for other components as well, that rely on localStorage & sessionStorage:

- Navigation.spec.js:

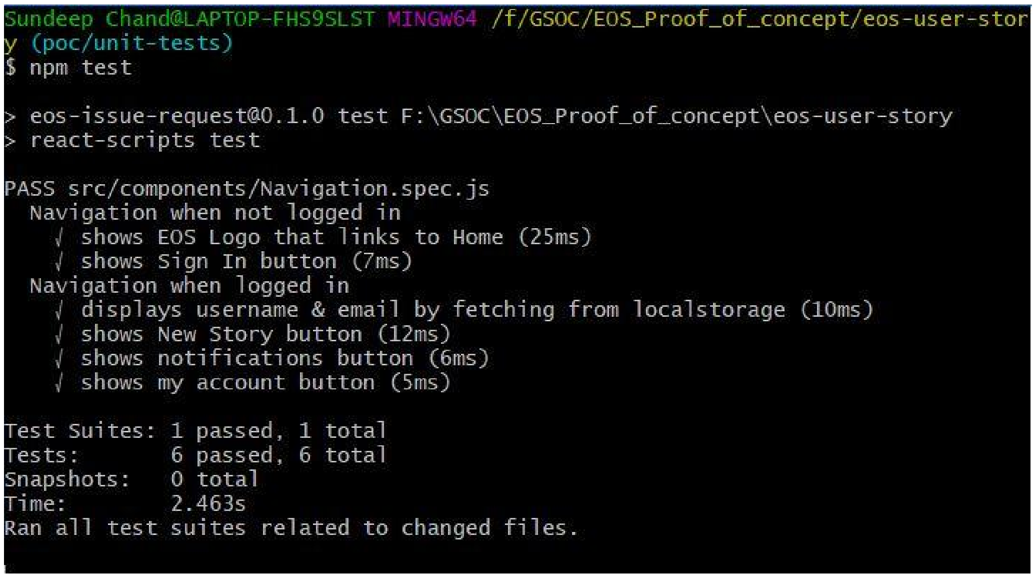
```
describe('Navigation when logged in', () => {
  let component
  const mockHandler = jest.fn()

  const TEST_USERNAME = 'user'
  const TEST_EMAIL = 'test@gmail.com'
  beforeAll(() => {
    localStorage.setItem('username', TEST_USERNAME)
    localStorage.setItem('email', TEST_EMAIL)
  })

  beforeEach(() => {
    const initialState = {
      auth: true,
      errorCode: null
    }
    component = render(
            <Context.Provider value={{ state: initialState, dispatch:
mockHandler }}>
        <Navigation />
      </Context.Provider>
    )
  })

  test('displays username & email by fetching from localstorage', () => {
    expect(component.container).toHaveTextContent(TEST_USERNAME)
    expect(component.container).toHaveTextContent(TEST_EMAIL)
  })
```

The image below shows the results after running the entire Navigation.spec.js.



```
Sundeep Chand@LAPTOP-FHS9SLST MINGW64 /f/GSOC/EOS_Proof_of_concept/eos-user-stor
y (poc/unit-tests)
$ npm test

> eos-issue-request@0.1.0 test F:\GSOC\EOS_Proof_of_concept\eos-user-story
> react-scripts test

PASS src/components/Navigation.spec.js
  Navigation when not logged in
    √ shows EOS Logo that links to Home (25ms)
    √ shows Sign In button (7ms)
  Navigation when logged in
    √ displays username & email by fetching from localstorage (10ms)
    √ shows New Story button (12ms)
    √ shows notifications button (6ms)
    √ shows my account button (5ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        2.463s
Ran all test suites related to changed files.
```

Currently I’ve also implemented the Homepage spec using Cypress. However, in order to reduce the size of end-to-end tests this can be divided into smaller unit tests using Jest as there are components that can be tested in isolation.

- home.spec.js:

```
/// <reference types="cypress" />

describe('Home page', function () {
  before('Open app', function () {
    cy.visit('/')
  })

  it('loads stories', function () {
    cy.get('.story:nth-child(1) > .stories-content > h3', {
      timeout: 10000
    }).contains('This is a test story')
  })

  it('does not allow user to vote', function () {
    cy.get('.votes-count')
      .invoke('text')
      .then((val1) => {
        cy.get('.vote-button').click({
          timeout: 10000
        })

        cy.get('.votes-count')
          .invoke('text')
          .then((val2) => {
            expect(val1).to.equal(val2)
          })
      })
  })

  it('links to story author', function () {
    cy.get('[data-cy=author-link]').should(
      'have.attr',
      'href',
      '/profile/605a30f6a4aa16137c4254a5'
    )
  })
  it('loads stories in different sections', function () {
    const buttonCount = 6
    for (let i = 2; i <= buttonCount; i++) {
      cy.get(`.roadmap > button:nth-child(${i})`).click({
        timeout: 10000
      })
      cy.contains('No stories')
    }
  })

  it('should open stories page when user clicks on a story', function () {
    cy.visit('/', {
      timeout: 10000
    })
    cy.get('.story:nth-child(1) > .stories-content')
      .click()
      .click()
      .url()
                                                                   .should('equal',
'http://localhost:3000/story/605a35e2a4aa16137c4254a7')
  })
})
```

Here is the demo of these tests running (links given above as well):
(Demo video in GDrive) (View video in Youtube)

TImeline:

Community Bonding (May 17 - June 7):

During this period I will get familiar with the community members and understand the workflow at EOS. I will interact with my mentor and decide the workflow for the project and refine the features presented(if any) in this proposal after discussing with the mentors. I will also fix any existing issues in the application, that might be a roadblock for the project. As well as I will learn in more detail about Cypress best practices and Docker during this period, so that I can enhance the testing part.

Outcomes:

- Learn more about EOS.
- Finalise the project roadmap.
- Improve my knowledge of Cypress best practices & Docker.

Week-1 (June 7 - June 13):

I will work on implementing Shareable Links for Stories. After this I will also start working on the initial setup for Searching of Stories.

Outcomes:

- Shareable Links for Stories is implemented.
- Start working on Searching of Stories.

Week-2 (June 14 - June 20):

I will finish implementing searching of stories. I will also work on implementing sharing of search/filter results. I will also work on the initial setup for implementing request templates.

Outcomes:

- Finish implementation of search and shareable search results.
- Initial setup for implementing request templates.

Week-3 (June 21 - June 27):

I will work on integrating request templates with the Front end. I will also work on the initial setup for OAuth Integration.

Outcomes:

- Finish implementation of Request Templates feature.
- Initial setup for OAuth integration.

Week-4 (June 28 - July 4):

I will work on finishing OAuth Integration this week. This week I will also start working on the unit/integration tests using Jest and React testing library.

Outcomes:

- Finish OAuth Integration.
- Implemented unit/integration tests for certain components.

Week-5 (July 5 - July 11):

I will continue working on writing tests this week.

Outcomes:

- Finish a majority of unit/integration tests.

Week-6 (July 12 - July 18) (1st review from July 12 - July 16):

This is the time for first evaluations. This week I will take the feedback from my mentor & work on improvements (if any). I will further work on implementing the end-to-end tests this week.

Outcomes:

- Merging of the patches to the main tree.
- Finish unit/integration tests.
- Initial work on implementing the end-to-end tests.

Week-7 (July 19 - July 25):

This week I will finish writing the end-to-end tests in Cypress.

Outcomes:

- Completion of the end-to-end tests.

Week-8 (July 26 - August 1):

This week I will focus on getting the tests reviewed, and get them merged with the main branch. I will also make changes to the documentation during this week.

Outcomes:

- Merging of tests to the main branch.
- Improved documentation.

Week-9 & Week-10 (August 2 - August 15):

I will work on deploying the tests to the Gitlab CI/CD Pipeline, so that the tests are run every time a commit is pushed.
As well as I am using this time as a buffer period, to avoid any delays arising out of any unforeseeable circumstances.
If everything goes as per schedule, I will be working on fixing existing issues and implementing feature enhancements during these weeks.

Outcomes:

- Tests deployed to CI/CD pipelines.
- Fixing existing issues.

Final Evaluations (August 16 - August 31)

Previous Experience:

I started learning web development a little before the start of my first year at college. By the second semester of my college I started contributing to some of my college societies as a web developer. My role involved developing the sites of the societies along with my team-mates.
I've also created a couple of full-stack web projects which can be found on my Github. I started contributing to p5.js-web-editor since my second year, before I started contributing to EOS. Here is a link to all the Pull Requests that I had made to the p5.js-web-editor project. I have also participated in some hackathons where I worked on creating full-stack web apps.
I am also currently a part of the Underwater Robotics Team of our college where I have worked on the computer vision part. However, it would not affect my GSOC project as the workload is not high in the team.

About Me:

I'm a second year Computer Engineering student at Delhi Technological University, New Delhi. I'm enthusiastic about learning new technologies, and I'm interested in Web Development in particular. I also code in C++/Python and I used to participate in competitive programming as well. However I did not enjoy doing it, and continued learning web development. I started contributing to open source since August last year. And I really enjoyed doing it as I was applying my skills to real world projects. Contributing to open-source also helped me feel confident about my development skills. So now I would love to contribute to an open-source project in a more focussed way under the guidance of professional mentors who have considerable experience in the field, and this project is the best opportunity for the same. I am a keen learner and never miss my deadline, so I will strictly adhere to my timeline.

Apart from coding I also enjoy going on cycling trips with my friends and listening to music. I also play video games during my free time.

Stretch Goals:

(These ideas can be implemented once the above goals are accomplished)

Integration of EOS User Story with Slack workspace:

Taking the example of EOS-Icons, we can integrate the website to our slack workspace so that whenever any new story is posted to the website, it is also updated on the internal Slack workspace. Similarly any other organisations who choose to use EOS User Story may integrate their Slack workspace just by changing an environment variable. This will enable the entire internal team to stay updated about the things being posted to the website and would ensure quick responses.

For implementing this we can use the `incoming-webhook` Slack app. Whenever a new story is created we can make a POST request from strapi-server to this Slack app and thus the message will be posted. I will work on implementing this once the above mentioned features are implemented.