



PYTHON SOFTWARE FOUNDATION

Sub-organization

EOS Design System



Project

User Story - Frontend and UX

By

Kailash Kejriwal

Contents

1.	Personal Details	2
2.	Synopsis	3
3.	Outline and methodology	5
4.	Timeline	22
5.	Contributions	24
6.	Previous Experiences	24
7.	About me	25
8.	Stretched Goals	26

Personal Details

Contact Details

Name	Kailash Kejriwal
Email	kailashkejriwal21@gmail.com
Timezone	Indian Standard Time (UTC + 5:30)

Profiles

GitHub	kailash360
LinkedIn	kailash360
Portfolio	kailashk.me
Medium	@kailash360

Education Details

University	Indian Institute of Information Technology, Gwalior
Degree	Bachelor of Technology
Major	Computer Science and Engineering
Expected Graduation	June, 2024

Synopsis

The EOS User Story project seeks to create a user-friendly and participatory site where users may log in and request features, report errors, and so on. Users can keep track of the status of their issues, vote on them, and leave comments on them. The issues can then be reviewed, resolved, closed, or assigned status by the project administrators. It functions as an effective feedback mechanism for organizations, which is required for the creation of various products.

The goal of this project is to improve the user experience by improving different areas. These ideas are as follows:

- **UX/UI improvements in Desktop and Mobile devices**

This will include refactoring of the *Story page*, and improvements in the *My Profile* and *Notifications* page to enhance their UI and provide a better user experience with more details.

- **Making the application a Progressive Web App (or PWA)**

This will allow the application to be installed in mobile devices and provide a native-application-like interface to the user. It will also introduce pre-caching the common files and assets in the browser cache. This, in turn, will decrease the loading time of the application. These features of PWA will be in accordance with web.dev/pwa-checklist. Apart from that, I will also work on improving the UI for mobile devices to give a better native-app-like experience.

- **Polish existing features and increase test coverage**

This will include enhancing the current functionalities of the application like reflecting network requests on buttons and adding requests templates for products. This will also include adding more tests to the frontend and creating a testing environment for the backend.

- **Improving session management for user authentication**

This will introduce a refresh-token in the authentication process, and this token can be used to regenerate the JWT when it expires. This will also prevent the user from getting logged out of the application in the middle of an activity.

- **OAuth 2.0 and SAML Integration**

OAuth 2.0 allow us to register on a platform without going through the entire sign-up process. Integrating OAuth also allows the users to log in quickly and easily. In EOS User Story, we can add OAuth 2.0 for popular services like [Google](#) and SAML 2.0 using [Okta](#). This will make the website more accessible and the UX will be smoother.

- **Report feature for stories**

This feature will allow the users to report a story to the admin if it contains any inappropriate content. The admin can then view the report and take action accordingly. The user who created the story can also make this request to get the story unpublished after a review by the admin.

- **Email notifications**

Email notifications will be sent to a user when any of the different cross-user activities take place like when someone comments or votes on a story, replies to a comment of the user or mentions the user in a comment.

- **Resolve issues on GitHub and User Story**

There are several issues open in both - GitHub and User Story. Apart from adding and modifying new features, all these open issues will also be handled and resolved.

Outline and Methodology

1. UX/UI Improvements in Desktop/Mobile devices

This will focus on improving the interface of the application to provide the users with a smooth experience and make the frontend more interactive. The primary tasks for this idea will be:

- **Refactor the Story page**

This will reconfigure the Story page and enhance the layout of the page. It will introduce better story details, a more organized comment section, and also add new sections like more stories from this product. The link to the design is [User Story - Story Page](#)

- **Fix Notifications page and My Profile page**

The footer in both - *Notifications page* and *My Profile* page jumps to the top of the page. This leads to UI inconsistency. The focus of this task will be to fix this by rewriting the styling sheets and modifying the content or message to be shown in that case.

Link for the design of notifications page: [User Story - Notifications Page](#)

2. Making the application a Progressive Web App

A Progressive Web App (or PWA) provides a native-application-like interface to the users on mobile devices. It is installable and allows the files to be cached in the browser.. This reduces the loading time of the application and comes with a custom offline page. The EOS User Story frontend will be converted into a PWA through the following tasks:

- **Adding a manifest file**

A *manifest.json* file will be added to the application. This file will contain all the necessary information to make the application installable. It will define the necessary properties of the application like:

- Name:** The name of the application that will be shown when it gets installed, here **EOS User Story**
- Short Name:** The short name is used when there is insufficient space to show the name, here **UserStory**
- Icons:** All the icons to be displayed when the PWA is installed
- Start URL:** The entry point of the application, here **index.html**
- Display:** Allows the app to be used as a native application. The value will be **standalone**

- f. Theme Color: #86EAE4
- g. Background Color: #091D2D

```
{
  "name": "EOS User Story",
  "short_name": "UserStory",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": "/index.html",
  "display": "standalone",
  "theme_color": "#86EAE4",
  "background_color": "#091D2D"
}
```

- **Adding a service-worker**

A service-worker will be added that can run parallel to the main thread and behave as an interceptor for the requests. It will contain the different events and respective callbacks that are to be executed when that event is fired, in the background.

The behavior of the service-worker corresponding to each event in this task will be:

- **Install:** When the service worker is installed in the browser, all the assets and files will be saved in the browser cache.

```

// install event
self.addEventListener('install', event => {

  //Wait until the cache is saved
  event.waitUntil(
    caches.open(staticCacheName)
      .then((cache) => {
        console.log('Caching all assets');
        cache.addAll/assets);
      })
  )
})

```

- **Activate:** When the service worker will get activated, the old cache is filtered out and removed from the browser.

```

// activate event
self.addEventListener('activate', event => {

  // Wait until the old cache is cleared
  event.waitUntil(
    caches.keys().then(keys => {
      return Promise.all(keys
        // Get the old cache
        .filter(key => key !== staticCacheName && key !== dynamicCacheName)
        // Clear the old cache
        .map(key => caches.delete(key))
      );
    })
  );
});

```

- **Fetch:** This is the event when the browser makes any request to the backend, either to fetch any file, asset, or data. In this event, it will be checked if the requested file already exists in the cache, else the request will be continued. This will also be responsible to handle cases like when the user goes offline, or any unexpected error occurs.


```

// fetch events
self.addEventListener('fetch', event => {
  event.respondWith(

    // Check if the request matches with any key in cache
    caches.match(event.request)
    .then(cacheResponse => {

      // If no such key exists in cache, then make the request,
      // otherwise return the value stored in cache
      return cacheResponse || fetch(event.request)
      .then(fetchResponse => {

        // Save the response in cache for future use
        return caches.open(dynamicCacheName)
        .then(cache => {
          cache.put(evt.request.url, fetchResponse.clone());
          return fetchRes;
        })
      });
    })
    // Handle any error like failed connection
    // By displaying an error page
    .catch((error) => {
      console.error('Error in making request:', error.message)
      return caches.match('/fallback.html');
    })
  );
});

```

- **Custom offline page**

The service-worker itself will display the custom offline page to the user. The fallback page to be displayed is [User Story - Fallback Page](#).

- **UI developments**

Apart from adding the configuration to make the application a PWA, I will also improve the UI for mobile devices such that it provides a native-app-like experience. The key modifications will include

- **Responsiveness:** I will work on improving the responsiveness of the entire application to suit all mobile devices. This will require reconfiguring

the style sheets of different pages and adding more detailed styling configurations.

- **Removing scrolling glitches:** Native applications are free from any form of scrolling glitch and provide a smooth UI. I will also remove such scrolling glitches from the User Story PWA and make sure that the content does not jump when the page loads.
- **Adding hamburger menu:** Native applications provide a sliding menu for navigation instead of a navbar on the top. I will create this sliding menu and add a hamburger menu to the application to toggle the sliding menu.
- **Finger-friendly touch targets:** Small touch targets like buttons and links make it difficult for users to locate and press them, especially on mobile devices. This can be fixed by enlarging the button size such that the finger pad can cover the target, and by highlighting the links sufficiently so that the user can locate them easily.
- **Disable text selection and text highlighting:** Native applications do not provide the option for text selection or highlighting. I will also work on removing these actions from the PWA.

3. Polish existing features and increase test coverage

This idea will focus on polishing the existing features of the application like:

- **Reflect network request on buttons**

This will make the buttons disabled until the network request is complete. If the network connection is slow, it will prevent the user from making consecutive requests to the same endpoint. This will also keep the UI updated with the current status of the action.

This will be done by adding a `isSubmitting` state that will check if a network request has been made.

```
<Button
  type='submit'
  data-cy='btn-submit'
  className={
    isSubmitting ? 'btn btn-default btn-disabled': 'btn btn-default'
  }
  onClick={descError}
  disabled={isSubmitting}
>
  {isSubmitting ? 'Submitting...' : 'Submit'}
</Button>
```

- **Adding more tests**

The new tests will cover a broader range of functionalities of the application and will also include the situation where the user is not logged into the application. The current tests already consider a wide number of functionalities. Apart from them, the tests that I am planning to add are:

- **For frontend**

User is not logged in	User is logged in
Home page: <ul style="list-style-type: none"> ▶ Should show <i>Sign In</i> button ▶ User cannot vote stories ▶ Clicking on story opens the story page ▶ Clicking on author's name opens the author's profile ▶ Stories can be sorted by votes ▶ Pagination should display more stories 	Home page: <ul style="list-style-type: none"> ▶ Should show <i>New Story</i> option ▶ Should show the <i>Notifications</i> icon ▶ User can vote the story, and then unvote it
Forgot Password page <ul style="list-style-type: none"> ▶ Should have field to enter email ▶ Should show error on providing invalid email ▶ Should submit a request when email is valid 	Forgot Password page <ul style="list-style-type: none"> ▶ Should redirect to homepage
Story page <ul style="list-style-type: none"> ▶ Should show all details of the story ▶ Should show all the comments and replies ▶ Should not show input field to add a comment ▶ Should show all the status of the story ▶ Should display the list of voters ▶ Should not allow to vote the story 	Story page <ul style="list-style-type: none"> ▶ Same as when the user is not logged in ▶ Should show input field to add a comment ▶ Should allow to vote and unvote on the story

Cookies and Policies page <ul style="list-style-type: none"> ▸ Should display to the privacy policy 	Cookies and Policies page <ul style="list-style-type: none"> ▸ Should display the privacy policy
--	---

- **For backend**

In the Strapi-based backend, tests will be added for the custom API endpoints and controllers. This will be helpful to make the backend more robust and help to eliminate any possible security issues in the backend. I will be using [Jest](#) - a testing framework for creating the test environment and writing the tests, suggested by the [Strapi documentation](#) itself. A Strapi server instance will need to be developed in the test environment to run the tests.

```

const Strapi = require('strapi');
const http = require('http');

let instance;

// Function to create the instance
async function setupStrapi() {

  if (!instance) {

    // Load up the function that parses the configs, hooks, middlewares and APIs of
    the application
    await Strapi().load();
    instance = strapi; // Strapi is global now

    await instance.app
      .use(instance.router.routes()) // populate Koa routes
      .use(instance.router.allowedMethods()); // populate Koa methods

    // create the server
    instance.server = http.createServer(instance.app.callback());
  }
  return instance;
}
module.exports = { setupStrapi };

```

The primary tests for the backend will be:

Collection	Tests
User Story	<ul style="list-style-type: none"> ● Should create a story with a multipart request body ● Should create a story without multipart request

	body <ul style="list-style-type: none"> ● Should not create a story without title and description ● Should not allow creating a story if the user is not logged in
Custom APIs	Logout <ul style="list-style-type: none"> ● Should log out the user CheckAuthor <ul style="list-style-type: none"> ● Should return 'true' if the author is correct ● Should return 'false' if the author is incorrect

I will also add more tests and modify the existing tests based on the feedback by the mentors.

These tests will be then integrated into the GitHub workflows to automate the entire test process using CI/CD pipelines of GitHub Actions.

4. Improving session management for user authentication

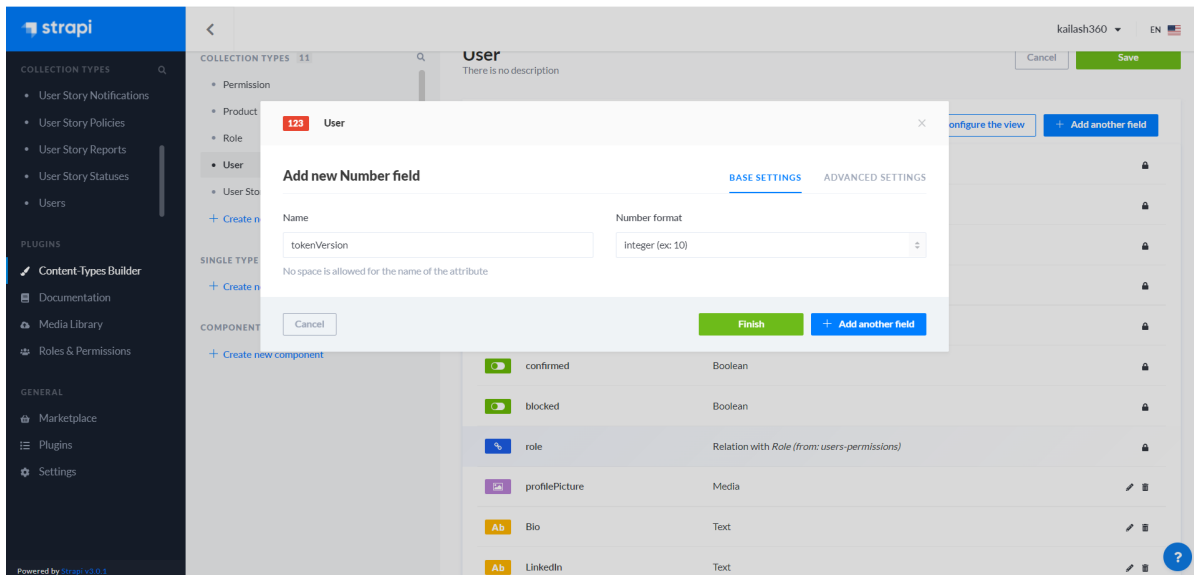
The backend of EOS User Story is developed using Strapi which is built on Koa.js. In this framework, authentication tokens are made to store in the cookies by default. After the token expires, the user will need to log in again to the application. The session may expire when the user is writing a story, or commenting on a story. This can spoil the poor user experience.

I will add refresh tokens for the session management.. The refresh tokens will be used to regenerate a new authentication token or JWT, and store it in the cookie. So, if the token expires when the user is performing an important activity, it will get the new token in the background, and provide a smooth user experience.

This will be done in the following tasks:

- **Create a field for the version of the refresh token**

A field called tokenVersion will be created in the Users collection that will denote the current version of refresh-token for the corresponding user. It will allow us to regenerate new tokens if the previous tokens get revoked. Since it should not be exposed to the API, the field will be private.



● Add a controller to generate the refresh token

I will add a controller that will be needed to generate the refresh token after all checks have been passed. It will be used for the following actions:

- When a user logs into the application
- When the password is changed by the user.
- When the refresh token itself gets expired

The refresh token will consist of the current version of the refresh token (tokenVersion), objectId of the user (id), and the expiration time (exp) of the refresh token.

```
// Controller to generate a new refresh token
const generateRefreshToken = (user) => {
  return strapi.plugins["users-permissions"].services.jwt.issue(
    {
      tokenVersion: user.tokenVersion, // Token Version
    },
    {
      id: user.id.toString(),
      expiresIn: "60d",
    }
  );
}
```

● Adding a controller for regenerating the JWT

I will create a controller `refreshToken` that will regenerate the authentication token or JWT after necessary verifications. This controller will check if the refresh token is valid by comparing the versions. This controller will be also used to regenerate a new refresh token after the old one gets revoked.

```
async refreshToken(ctx) {  
  
  const params = ctx.request.body;  
  
  // Params should consist of:  
  // * token - string - jwt refresh token  
  // * renew - boolean - if true, also return an updated refresh token.  
  // Parse Token  
  try {  
  
    // Unpack refresh token  
    const {tokenVersion, iat, exp, id} =  
      await strapi.plugins["users-permissions"].services.jwt.verify(params.token);  
  
    // Check if refresh token has expired  
    if (Date.now() / 1000 > exp) return ctx.badRequest(null, "Expired refresh token");  
  
    // fetch user based on objectId  
    const user = await strapi.query("user", "users-permissions").findOne({ id: id });  
  
    // Check here if user token version is the same as in refresh token  
    // This will ensure that the refresh token hasn't been made invalid by a password change or similar.  
    if (tokenVersion !== user.tokenVersion)  
      return ctx.badRequest(null, "Refresh token is invalid");  
  
    // Otherwise we are good to go.  
    ctx.send({  
      jwt: strapi.plugins["users-permissions"].services.jwt.issue({  
        id: user.id,  
      }),  
      refresh: params.renew ? generateRefreshToken(user) : null  
    });  
  } catch (e) {  
    return ctx.badRequest(null, "Invalid token");  
  }  
},
```

● Adding a controller to revoke the refresh token

The refresh token may be exposed to any third party, or the user may change the password for security reasons. In both cases, it is necessary to revoke (or render useless) the current refresh token, so that a new refresh token can be generated. This will use a controller where the existent version of the present refresh token will be checked, and the value for `tokenVersion` field in the `Users` collection is increased by 1. Since the current token contains the old value of `tokenVersion`, it gets revoked.

- **Adding routes for the controller**

The routes will be defined in a JSON file present in the format `extensions/user-permissions/config/routes.json`, with each route dedicated for each of the controllers.

Route	Method	Controller
/refreshToken	POST	refreshtoken
/revoke	POST	revoke

- **Integrating the routes with the frontend**

I will define custom services for making the required API calls to regenerate or revoke a token. I will use a timeout tracker to check if the JWT in the cookies has expired or not. If it expires, a request to `/refreshToken` will be made to fetch the new JWT.

```
// Timeout tracker to check if the token has expired
setInterval(async ()=>{
  try{
    // If both - the refresh token and the JWT have expired,
    // then get regenerate both the tokens
    if($.cookie('token') === null){
      // Generate both the tokens using the service 'regenerateToken'
      const { jwt,refreshToken } = await regenerateToken({
        renew: $.cookie('refreshToken') === null
        // check if the refresh token has also expired
      })
      // Save the tokens in cookies
      document.cookie = `token=${jwt}`
      document.cookie = `refreshToken=${refresh}`
    }
  }catch(err){
    //If an error occurs, then logout the user
    console.error(err.message)
    toast.error(err.message)
    dispatch({
      type: 'DEAUTHENTICATE'
    })
    navigate('/')
  }
},1000)
```


This will also make sure the user gets logged out if an invalid token is sent in the request. I can also modify the method of implementation, based on the feedback by the mentors.

5. OAuth 2.0 Integration

OAuth 2.0 allows us to register on a platform without going through the entire sign-up process. Integrating OAuth also allows the users to log in quickly and easily. In EOS User Story, we can add OAuth 2.0 for popular services like [Google](#) and [Okta](#). This will make the website more accessible and the UX will be smoother.

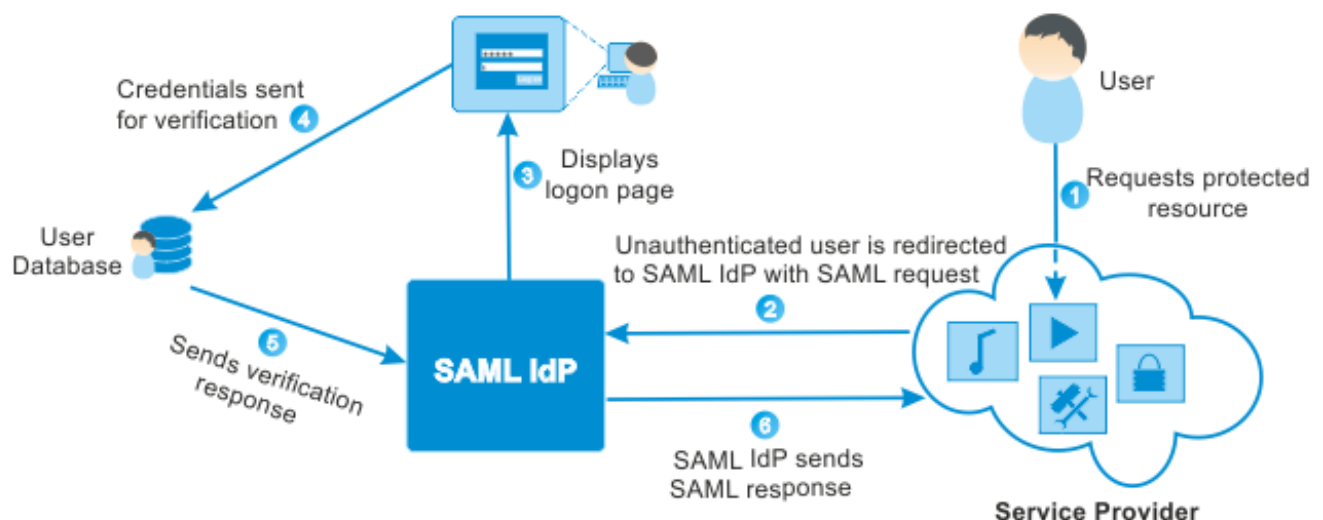
Google OAuth 2.0

The screenshot displays the Google Cloud Platform console interface for configuring an OAuth consent screen. The top navigation bar shows 'Google Cloud Platform' and 'EOS User Story'. A search bar is present on the right. The left sidebar lists various API and service options, with 'OAuth consent screen' selected. The main content area is titled 'EOS User Story' and includes an 'EDIT APP' link. It features sections for 'Publishing status' (with a 'PUBLISH APP' button), 'User type' (set to 'External' with a 'MAKE INTERNAL' option), and 'OAuth user cap' (showing a progress bar for '1 user (1 test, 0 other) / 100 user cap'). Below these is a 'Test users' section with an 'ADD USERS' button. At the bottom, there is a 'Filter' input field and a table of 'User information' with one entry: 'panthera.tigris360@gmail.com'.

SAML Integration using Okta

I will also add SAML for user authentication using Okta SAML 2.0 provider. It will enable Single Sign-On (SSO) which in turn will allow the users to access the EOS User Story platform by logging into the application only once.

I will use SP-Initialized SSO so that when an unauthenticated user visits our application, he/she will be redirected to the login page.



To integrate SAML with Okta, we will need to create an application on Okta Developer Console and configure the SAML integration settings. The configuration settings will include the credentials of the client to be used for verification. In this case, it will be an email. This will provide us with an application URL to be used for making the authentication request from the client-side.

Now when a user tries to access the User Story application, he/she will be redirected to the login page if that user is not authenticated.

General Settings

[Edit](#)

Okta domain

dev-34182814.okta.com



APPLICATION

App integration name

EOS User Story

Application type

Single Page App (SPA)

Grant type

Client acting on behalf of a user

- ☒ Authorization Code
- ☐ Interaction Code
- ☐ Refresh Token
- ☐ Implicit (hybrid)

USER CONSENT

User consent ⓘ

☒ Require consent

Terms of Service URI ⓘ

Policy URI ⓘ

Logo URI ⓘ

LOGIN

Sign-in redirect URIs ⓘ

☐ Allow wildcard * in login URI redirect.

http://localhost:3000/login/callback

https://userstory.eosdesignsystem.com/login

Sign-out redirect URIs ⓘ

http://localhost:3000

https://userstory.eosdesignsystem.com/

Login initiated by

App Only

Initiate login URI ⓘ

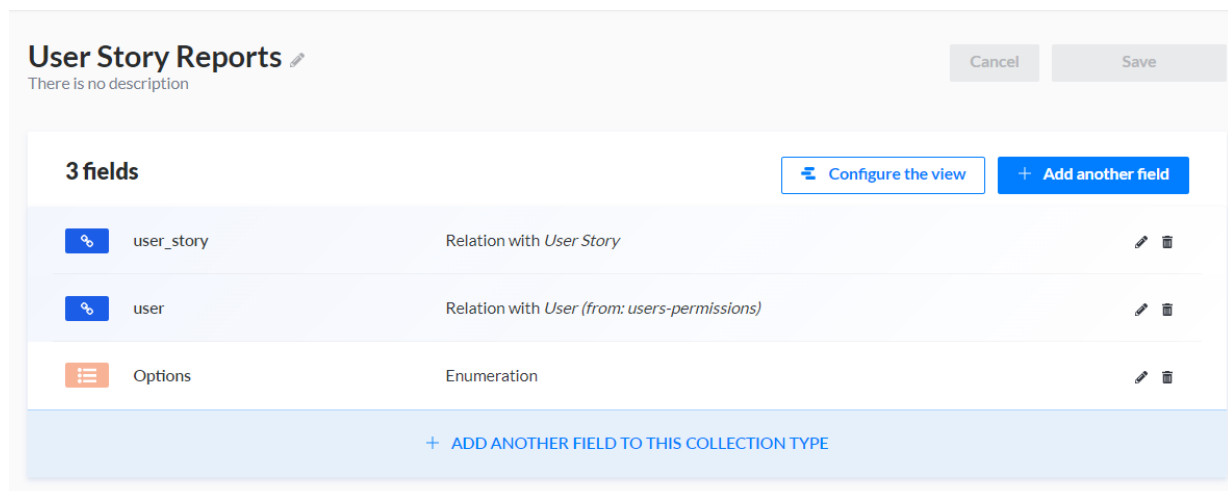
6. Report feature for stories

This will allow the users to report any story that contains any inappropriate content. The reported stories will be displayed in a User Story Reports collection, and the admin will be able to take action for each report.

While reporting a story, the users will need to select a reason that the story is reported. It can be done by providing pre-defined options like Abusive Content, Spam, Not relatable to EOS, Duplicated Story, and so on.

Link for the design of Report modal: [User Story - Report Modal](#)

In the Strapi-based backend, the reports will be stored in the User Story Reports collection, which is a many-to-one relationship with both Products and Users collection. So, a single user can make multiple reports, and each product can have multiple products under it.



The screenshot shows the 'User Story Reports' collection configuration interface. At the top, there's a title 'User Story Reports' with a pencil icon and a subtitle 'There is no description'. To the right are 'Cancel' and 'Save' buttons. Below this is a section titled '3 fields' with two buttons: 'Configure the view' and 'Add another field'. The fields are listed in a table:

Field Name	Field Type	Actions
user_story	Relation with User Story	Edit, Delete
user	Relation with User (from: users-permissions)	Edit, Delete
Options	Enumeration	Edit, Delete

At the bottom, there is a button: '+ ADD ANOTHER FIELD TO THIS COLLECTION TYPE'.

When a new report is created, an email will also be sent to all the administrators regarding the new report. This will be done by adding the logic to send email in the afterCreate life cycle in the controllers of the User Story Reports collection.

7. Email Notifications

Emails notifications can be added for different functionalities to notify the user with any update of a story or comment. This will increase the engagement of the system with the user.

I will enable Email notifications for the following actions:

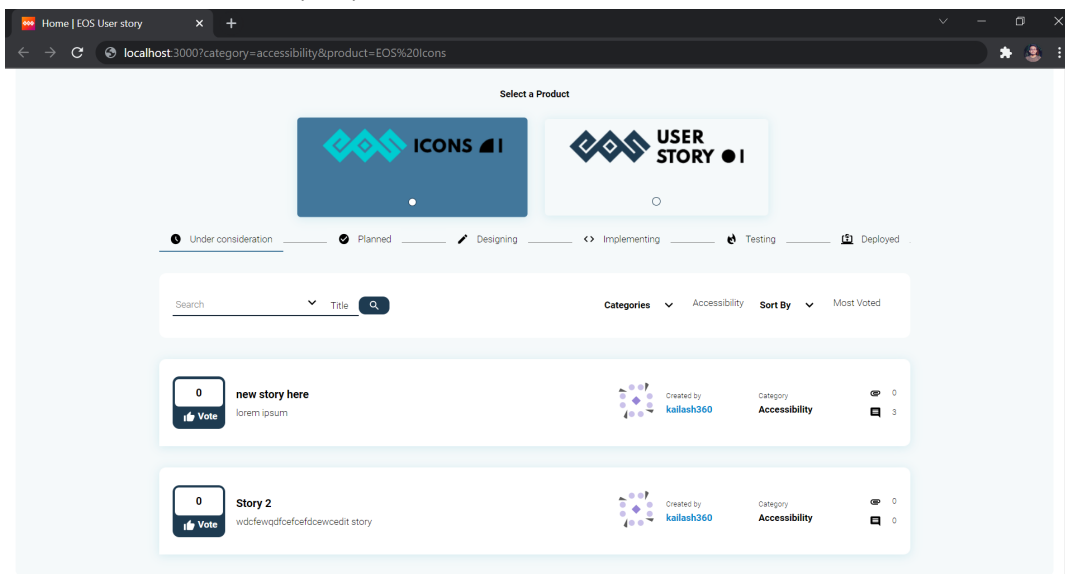
- When someone comments on my story
- When someone replies to my comment
- When someone mentions me in a comment

8. Resolve existing issues on GitHub and User Story

Apart from implementing the ideas discussed above, I will also work on fixing the current issues on GitHub and User Story. Some issues that need special attention are mentioned below:

- **Issue #28: Shareable search result link for stories**

This will need the use of query strings to share the field-value pair of the filters. The query can then be then parsed using the [query-string](#) package. So, if a query is '?category=accessibility&product=EOS%20Icons', it would mean that 'all the stories of category Accessibility for product EOS Icons' should be displayed.



- **Issue #29: Improve code quality, bug fixing**

This issue mentions replacing the excessive use of the useState hook with a better state management technique. I will be using the useReducer hook for pages like Story.js.

There will be an initial state to contain the default values of the fields, and a reducer will be defined to update the values in the state based on the type of action that is dispatched. This will help to maintain a single source of state.

- **Improve design in case no stories are found**

The application currently shows a simple message if no stories are found - be it for a filter, or a particular user. This can be fixed by modifying the view if no stories are found

Link for new design: [User Story - No Stories Component](#)

- **Fix Request Templates**

The feature of rendering request templates while creating a story, has already been implemented. But the templates are not rendered on the client-side. I will also work on fixing this issue.

Timeline

1. Community Bonding Period (May 20 - June 12)

- Continue to work on minor bugs and possible improvements.
- Spend more time learning about the workflow of EOS, and getting familiar with the community members and mentors
- Get the proposed designs reviewed by the mentors, to have a final outcome
- Learn more about the best practices of Progressive Web Apps

2. Week 1 (June 13 - June 19)

- Work on refactoring the Story page according to the proposed design
- Add new sections on the Story page
- Fix responsiveness of the Story page

3. Week 2 (June 20 - June 26)

- Fix the notifications page according to the proposed design
- Remove UI inconsistencies on the My Profile page
- Start working on making the application a PWA

4. Week 3 (June 27 - July 3)

- Add manifest and service workers
- Improve the UI for mobile devices
- Test the features by installing them in a mobile device

5. Week 4 (July 4 - July 10)

- Add network request status on buttons
- Add tests for the frontend
- Fix any failing tests

6. Week 5 (July 11 - July 17)

- Create test configuration for backend
- Write new tests for the backend
- Integrate testing job in CI/CD pipeline for backend
- Fix any failing test in the backend

7. Week 6 (July 18 - July 24)

- Define controllers for generating and revoking refresh tokens
- Create routes for the controllers
- Integrate the routes with the frontend in custom services

8. **Week 7 (July 25 - July 31 | Phase 1 Evaluation)**
 - Take review from mentors
 - Work on any possible improvements
 - Start working on OAuth 2.0 Integration
9. **Week 8 (August 1 - August 7)**
 - Add Google OAuth 2.0 authentication
 - Integrate with SAML 2.0 using Okta
 - Finish working on OAuth 2.0
10. **Week 9 (August 8 - August 14)**
 - Develop the Report modal component
 - Integrate the APIs with the Report modal
 - Write Cypress tests for the Report feature
11. **Week 10 (August 15 - August 21)**
 - Add new Email templates for different actions
 - Add logic for sending Email Notifications
12. **Week 11 (August 22 - August 28)**
 - Start working on fixing the existing issues on GitHub and User Story
 - Add the feature to share search results via URL
13. **Week 12 (August 29 - September 4)**
 - Modify state management in various pages by replacing multiple useState hooks with a useReducer hook
 - Modify the view when no stories are found, according to the design
 - Finish any pending task
14. **Week 13 (September 5 - September 12 | Final Evaluation)**

Contributions

Contributions in EOS User Story Frontend

- [Pull Requests](#)
- [Issues](#)

Contributions in EOS User Story Backend (Strapi)

- [Pull Requests](#)
- [Issues](#)

Contributions in EOS Icons Landing

- [Pull Requests](#)

Contributions in EOS Wiki

- [Pull Requests](#)

Previous Experience

Internships

- **Blockchain Developer Intern | WalletSocket**
(January 2022 - April 2022)
 - Developed an automated CiceroMark markdown generator using Cicero Engine and Concerto data models of Accord Project
 - Integrated Web 3.0 services into the document generator to exhibit the on-chain block history
 - Optimized the document generation scripts to reduce compilation time by 70%
- **Full Stack Developer Intern | InfuzeX Ventures**
(August 2021 - December 2021)
 - Developed a multi-level academics management system and cross-language learning platform having 50,000+ registered users
 - Built and modified 100+ reusable components using ReactJS, Redux and Material UI, leveraging existing components
 - Designed and implemented a scalable RCM-based backend in NodeJS + ExpressJS, WebSocket protocol to exchange data using REST APIs
- **Web Developer | Uthaan IIITM**
(July 2021)
 - Designed and developed a website for my institute's journalism and recreational club Uthaan, using MERN stack.

- Integrated Cloudinary services into the RCM-based backend to host multimedia dynamically

Open Source

- **Git Init FOSS 2021 | IIIT Gwalior**

I contributed to more than 10 repositories, by successfully making 20+ Pull Requests and creating 15+ new issues. I also emerged to be one of the Top 5 contributors in the same.

- **Hacktoberfest 2021**

I successfully completed the milestones of Hacktoberfest 2021 by making 25+ Pull Requests over 15+ repositories.

- **Other Contributions**

Apart from that I have contributed to major organizations like AOSSIE, Oppia, The Palisadoes Foundation, Mozilla, and NodeJS.

About Me

I am a second-year undergraduate pursuing a Bachelor of Technology in Computer Science and Engineering, from the Indian Institute of Information Technology, Gwalior. I have always been keen to learn about new technologies, and use them to build scalable products. This enthusiasm landed me in the field of development, Web Development to be precise. I found it very interesting and gradually went on learning deep into this domain. I also developed several personal projects. It was a wonderful experience. Then, I came across Blockchain technology and I am still learning more about this field.

I was told that Open Source is a rewarding field. So, after my freshman year ended, I ventured out to learn about contributing to open source, and then started contributing to different projects which suited my knowledge. It really feels amazing when your PR gets merged and you know that your contribution has left an impact on the thousands of users who use your software. Open source also helped me to learn a lot of best practices which are difficult to learn when you are working on some personal project.

I also have a knack for writing. So I tried my hands in writing articles on the latest technologies. I frequently publish my articles on [Medium.com](https://medium.com).

Apart from coding, I love to read books on history and listen to music in my free time.

Stretched Goals

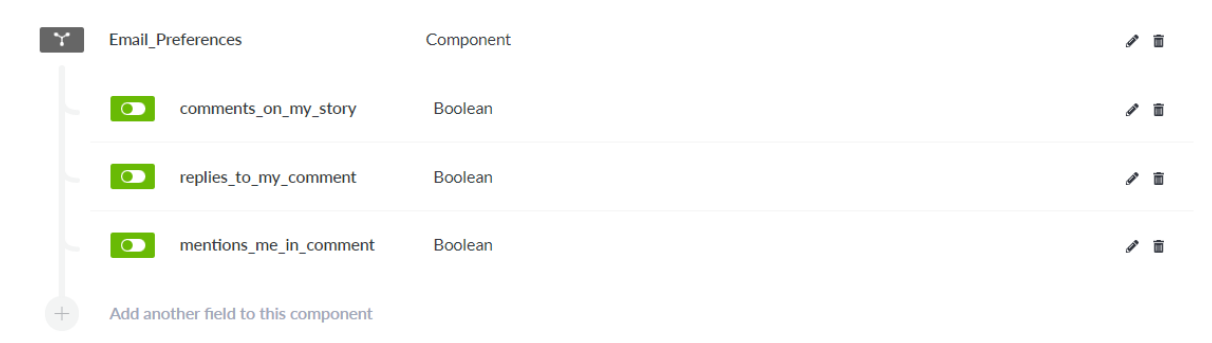
These are the ideas that can be implemented once all the ideas discussed above are completed successfully.

- **Email Preferences**

An **Email Preferences** section will be added in the User Story, that will allow the users to choose the type of emails that he/she wants to receive.

Link to the design of Email Preferences Modal: [User Story - Email Preferences](#)

In the backend, the Email Preferences will be stored as a Strapi collection component, and this component will contain each preference in Boolean format.



- **Latest Updates page**

This will be a new page in the application to show all the latest developments in the EOS organization. The content will be uploaded solely by the admin, while the users will have the liberty to upvote/downvote an update. Each update will be linked to a product. This will allow the users to filter the updates of a particular product.

Link to design for Latest Updates page: [User Story - Latest Updates page](#)

In the backend, the updates will be stored in a collection - User Story Latest Updates. This collection will be in a many-to-one relationship with the Products collection such that there can be multiple updates for a single product.

