Organization: Python Software Foundation
Sub-organization: EOS Design System
Project Title: User Story - frontend and UX

Name	**Hrishikesh Agarwal**
Email	hrishikeshagarwalv@gmail.com
Phone number	**(+91) 9625239524**
Timezone	**UTC+05:30**
Github	**https://github.com/codetheorem**
Linkedin	**https://www.linkedin.com/in/hrishikesh-agarwal-794801195/**

Code Contribution to EOS

- Contributions to eos-user-story:
    1. Pull requests
    2. Created Issues

- Contributions to eos-strapi:
    1. Pull requests
    2. Created Issues

Project Background

EOS is an open-source, and customizable Design System for front-end developers and UX designers.It has a whole range of products which deals with making the frontend more accessible and available to the world.

The EOS User Story project is a project that supports development and testing of other projects. In this you can raise issues or make new feature requests or discuss some improvements. It helps developers at EOS get better feedback which results in development of quality products. The better the feedback the better will be the product so EOS which aims for high quality requires this for its developers and users. This project can help any organization in their development cycle.

Synopsis

Currently EOS user story has many features but most of them are in their primitive stage and there is a need to work more on those features and also add some more important features. My plan to work on this project is that first I want to improve existing features and services and then implement

new features. The features can be divided in three parts based on implementation

- Frontend only
- Frontend + Backend
- Backend only

So, first I will focus on the frontend then I will move onto the backend. The main reason for this is that the frontend requires more effort as it is directly facing the user hence needs more care.

The features that I want to improve or add are -

1. **Revamping comment section (Frontend + Backend)**
   Discussions are a very important part of any product development but currently we have very few features in the comment section which gives a boring experience to the users. Our aim is to add more features to it.
2. **Adding state management (Frontend)**
   Currently communication between different components is only through props and there are so many states lying in each file so to regulate that and make this application scalable state management is necessary.
3. **Improving search and sort (Frontend + Backend)**
   We can only search using title and author currently but we can add a general search which searches according to title,author,content etc on its own and also add more sorting options.
4. **Adding labels to story (Frontend + Backend)**
   Any story can be categorized under different categories but we can also have user created labels under each category which will give more information about a story.
5. **Revamping notification system (Frontend + Backend)**
   There is a need to improve both the frontend and backend of the notification service. In the backend we need to add notification of

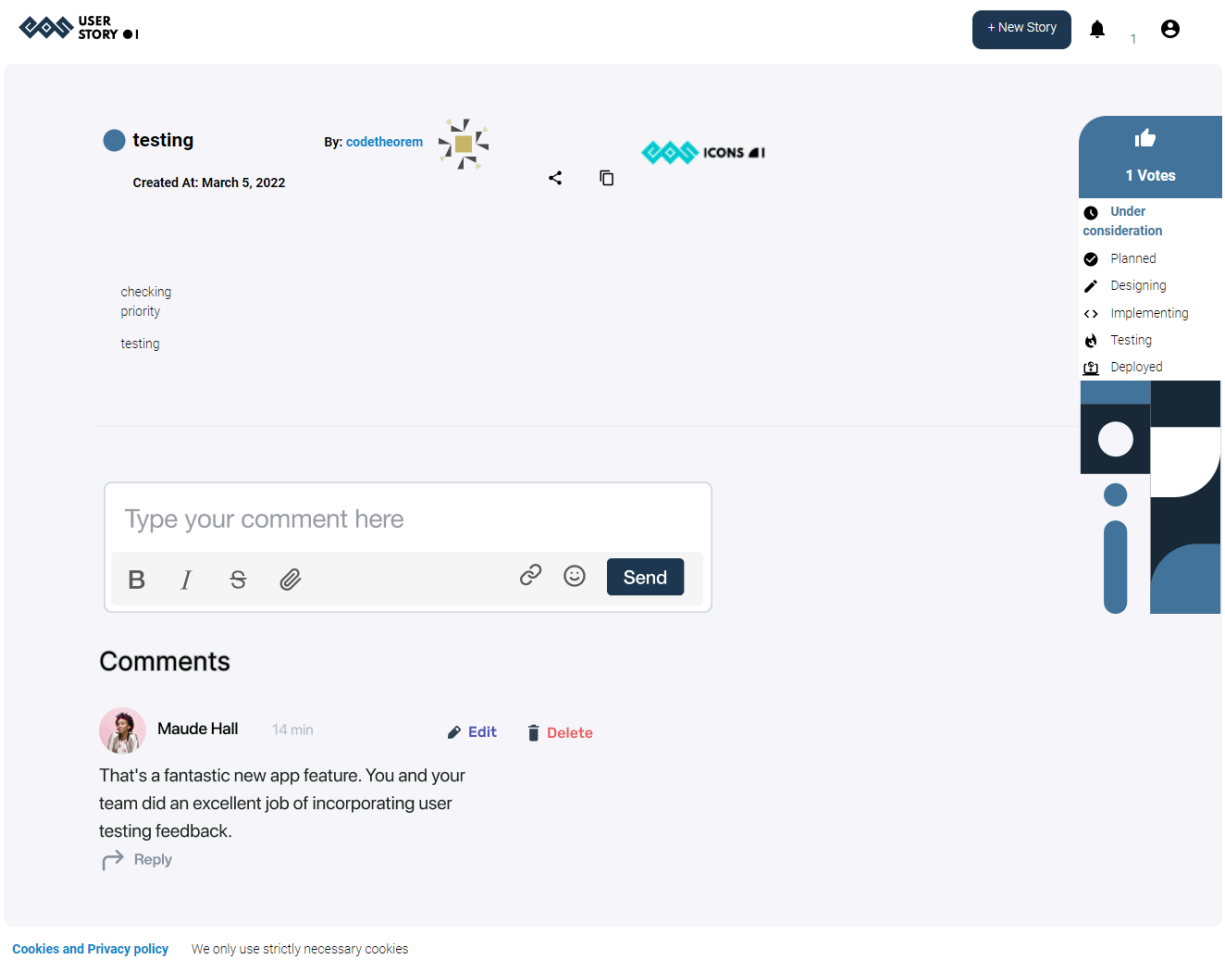
important actions and in the frontend we need to improve the appearance.

6. **Migration to Typescript (Frontend)**
   Typescript is a static typing language, so for future development it is important to implement it which will result in less bugs and quality development.
7. **Migration to Strapi V4 (Backend)**
   We are using Strapi version 3.x for our backend but currently Strapi has released its version 4 so it is necessary to migrate to version 4 to avoid future problems.
8. **Progressive Web App (Frontend)**
   PWAs work like a native app but originally it is only a web app so it gives benefits of both the technologies and provides users a unique experience.
9. **Adding testing in Strapi (Backend)**
   Add unit testing in the backend to make the application more safe and robust.

Outline and Methodology

- **Revamping comment section (Frontend + Backend)**
  Discussions are a very important part of any development cycle and user stories help developers and designers to make a good product.Currently the comment section of a story don't have much features you can only just add comment and reply on any comment.Idea is to add more features to it like edit , delete and add formatting options for the comment section, reactions.

  This idea requires changes in both backend and frontend-



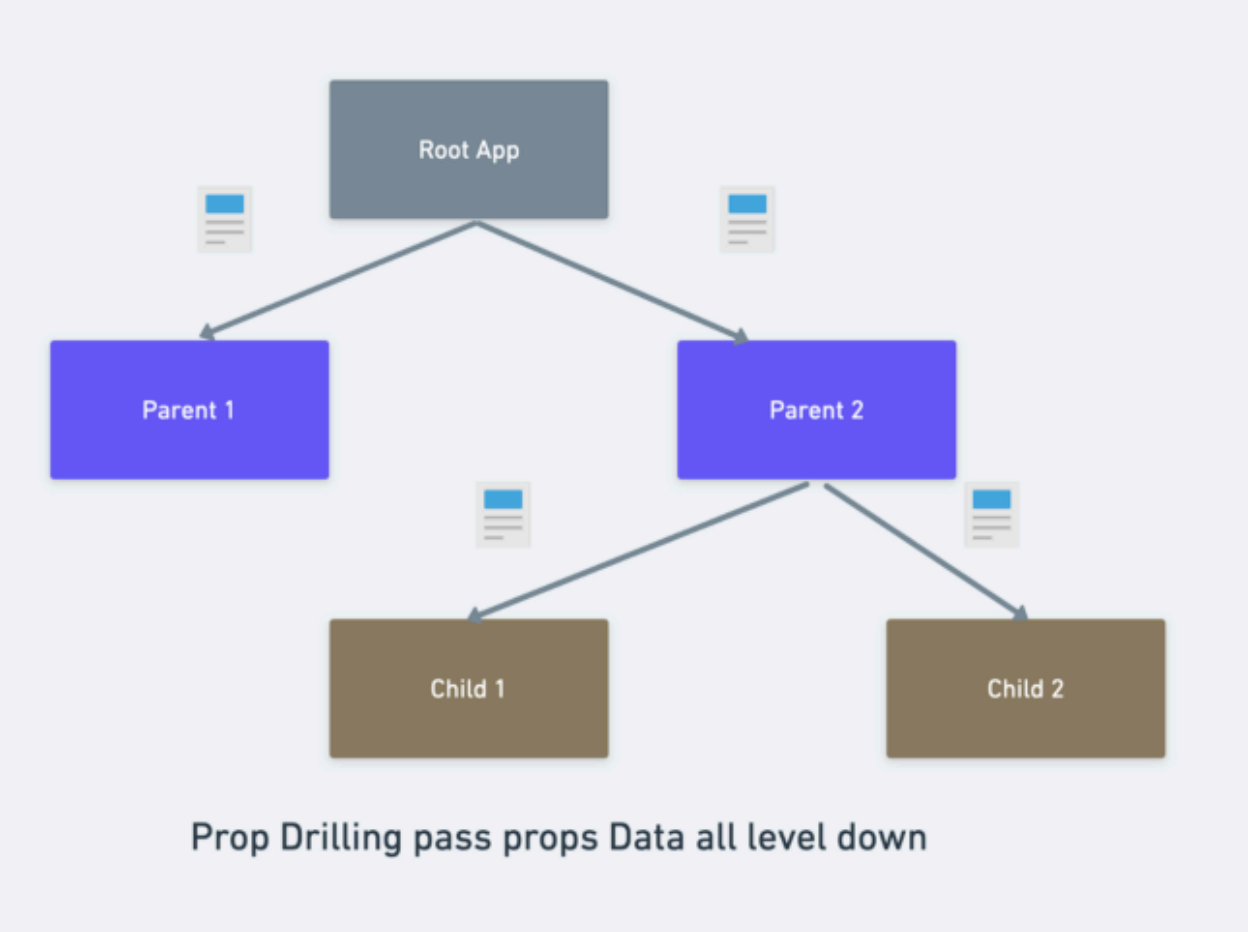
This is the mockup for the comment section which will have a new comment form with more features and also the comment will have an edit and delete button which will be visible only to the author of that comment.

In backend we need to create a model for reactions and also update model of user-story-comments and user-story-comments-thread.Here it will look like

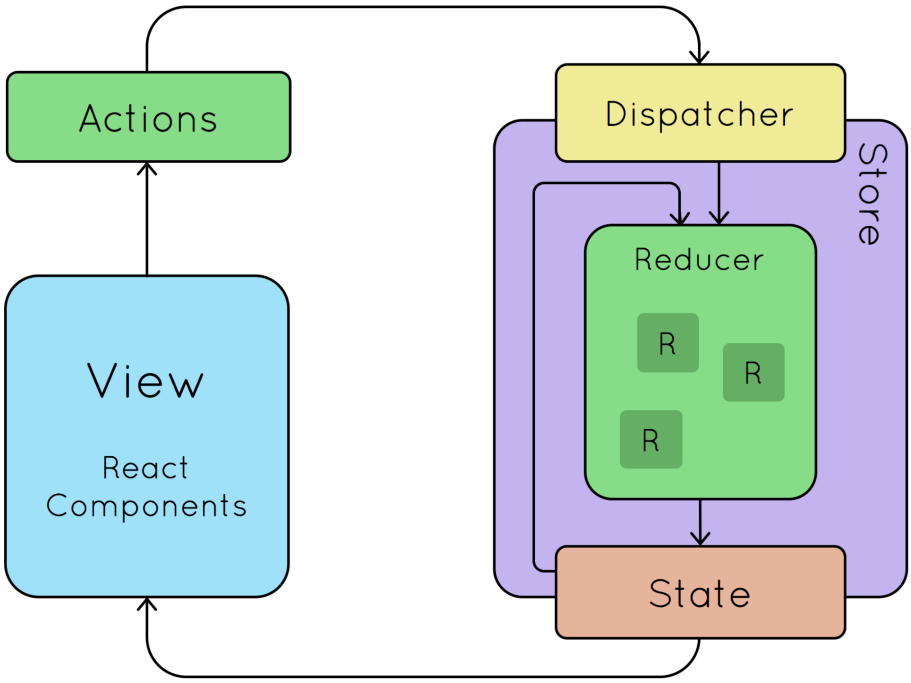
```
{
  "kind": "collectionType",
  "collectionName": "user_story_reactions",
  "info": {
    "name": "User Story Reaction"
  },
  "options": {
    "increments": true,
    "timestamps": true
  },
  "attributes": {
    "emoji": {
      "type": "string",
      "unique": true
    },
    "label": {
      "type": "string",
      "unique": true
    },
    "user": {
      "plugin": "users-permissions",
      "collection": "user"
    },
    "user_story_comment": {
      "collection": "user-story-comment",
      "via": "user-story"
    },
    "user_story": {
      "model": "user-story",
      "via": "user_story_comments"
    }
  }
}
```

[Figma Link]

- **Adding state management (Frontend)**
  In frontend we have multiple components and pages and we are communicating between these using prop drilling However, prop drilling can become an issue in itself because of its repetitive code.



So the solution is to add a state management service like Redux. To add redux we have to first create a store and to create a store we need to first create a reducer and initial state then we will create our store. A reducer is a reducing function which uses actions to mutate the state of the store. So all the actions currently in components will be migrated to reducers then we will create a store. To listen to all changes of the store we can subscribe to the changes using `store.subscribe()` method provided by redux. An action is dispatched using `store.dispatch({type: 'mutation',...arguments})` . Here is workflow of redux state management -



An action is dispatched from a react component which then makes a reducer mutate the state and then using subscribe we can listen to all the changes from the state and so every component is in sync with all other components and results in better performance. Here is example of a code -

```
const story = (state = [], action) => {
  switch (action.type) {
    case 'SET_AUTHOR':
      userstory.author.name=action.text
      return userstory
    default:
      return state;
  }
};

const store = createStore(story);

store.subscribe(() => {          // listen to all changes in the state
  console.log(store.getState());
});


store.dispatch({
  type: 'SET_AUTHOR',
  text: 'Example Author',
});
```

After creating the store and reducers we will wrap our app.js file with the provider that redux offers the code will look like this -

```
import { Provider } from "react-redux"
import { store } from "store.js"

render(
   <Provider store={store}>
      <App />
   </Provider>,
   document.getElementById('root')
);
```

After this we can use state and actions anywhere in our app.

We can also add middleware on the top of our dispatch layer. We can add redux-persist to make our state persist and we can also check all the reads and writes to the store which helps in debugging more easily and perfectly.

- **Improving search and sort (Frontend + Backend)**

  Searching for stories is important from a UX point of view. Currently you have to first select whether you have to search by author or title. But what if a person have to search by content so to implement we have two options-

  1. Make an option for search by content.
  2. Let the user enter anything and give them best results.

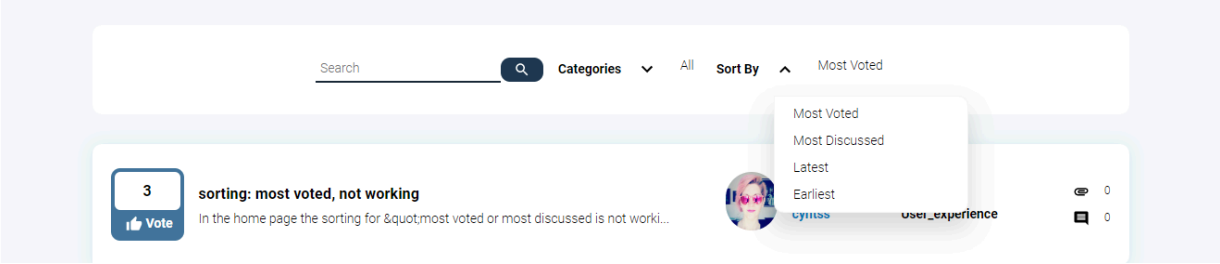
  According to my judgment, the second option is best.

  To implement the second option we need to change the backend and In frontend we need to just send the search query to the backend. In order to implement this we have to create a controller in the user-story API

in strapi here how the controller will look like

```
async find(ctx) {
    let entity=[];
    const authorResult = await strapi.services['user-story'].model.find({where:{author:{username: /.*ctx.query._search*/}}}); //searches for
author
    const titleResult = await strapi.services['user-story'].model.find({where:{Title: /.*ctx.query._search*/}}); //searches for title
    const contentResult = await strapi.services['user-story'].model.find({where:{Description: /.*ctx.query._search*/}}); //searches for content
    entity.push(authorResult);
    entity.push(titleResult);
    entity.push(contentResult);
   if(ctx.query._limit || ctx.query._start){ //pagination
     entity = entity.slice(ctx.query._start,ctx.query._limit)
   }
   return entity.map(story => sanitizeEntity(story, { model: strapi.models['user-story'] }));
 }
```

The above controller catches for all find requests and it first searches for stories having matching author then it checks for title and then content and all the relevant stories are sent to frontend in a paginated format.
Alternatively we can also use third party services like Algolia to implement search.
Another important thing is sorting the stories. So currently sorting is not working properly in the user story. To resolve this issue I have created a PR in backend as well as a PR in frontend. Idea is to add more sorting methods like latest,earliest and to implement this I have to update the frontend by adding more options in dropdown and updating the controller in the backend. Latest will sort the stories in descending order of created date and earliest in ascending order.



- **Adding labels to story (Frontend + Backend)**
  We have categories that we can add while creating a new story but it does not give exact description or enough metadata about the story

so the need for labels can be explained through a example let's assume that there is a story related about a bug so a user will have to look into the description of the story to know whether it is related to him or not but what if there are labels so that he will see the labels and can decide that whether he should go and read the description or move ahead.

To implement we have to add label attribute in model of user-story and make a new collection for labels in strapi

```
{
  "kind": "collectionType",
  "collectionName": "user_story_label",
  "info": {
    "name": "User Story Label"
  },
  "options": {
    "increments": true,
    "timestamps": true
  },
  "attributes": {
    "description": {
      "type": "string",
      "required": true
    },
    "color": {
      "type": "string",
      "required": true
    },
    "user_story": {
      "via": "user_story_status",
      "collection": "user-story"
    },
    "author": {
      "plugin": "users-permissions",
      "model": "user",
      "required": true
    }
  }
}
```

The frontend changes will look like this-

USER STORY

+ New Story 1

New Story

Title

Product

Select a product

Category

Select a category

Priority

Select priority

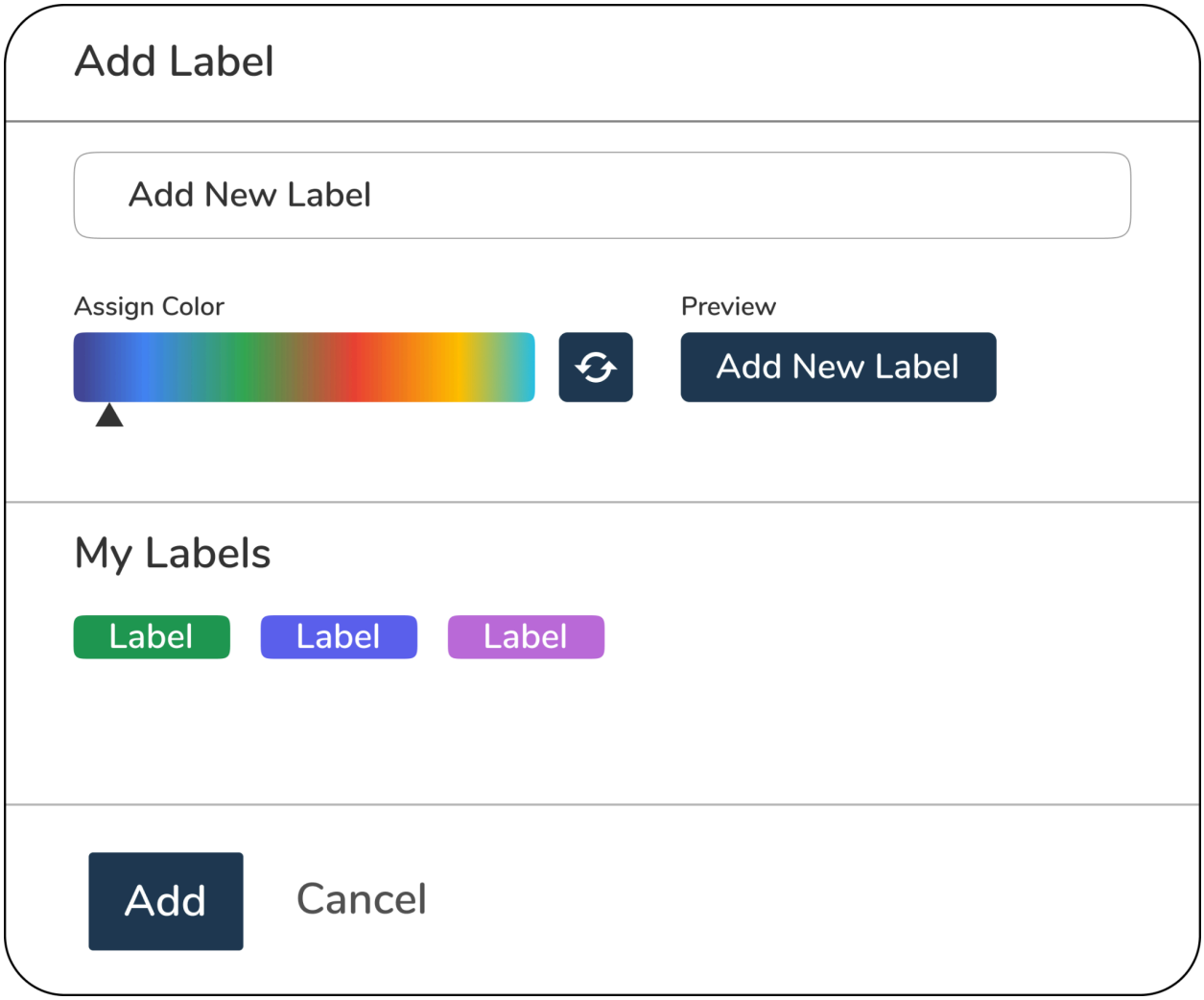
Description

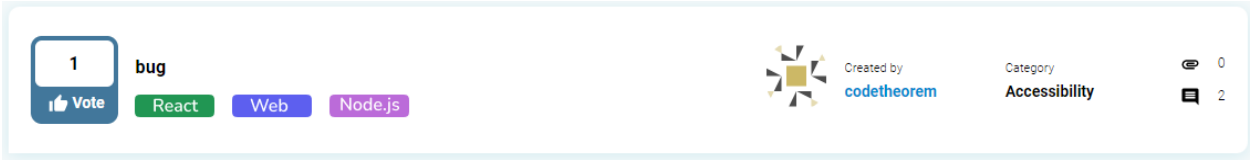
Drag 'n' drop some files here, or click to select files

Labels

Add

Label Label Label Label Label

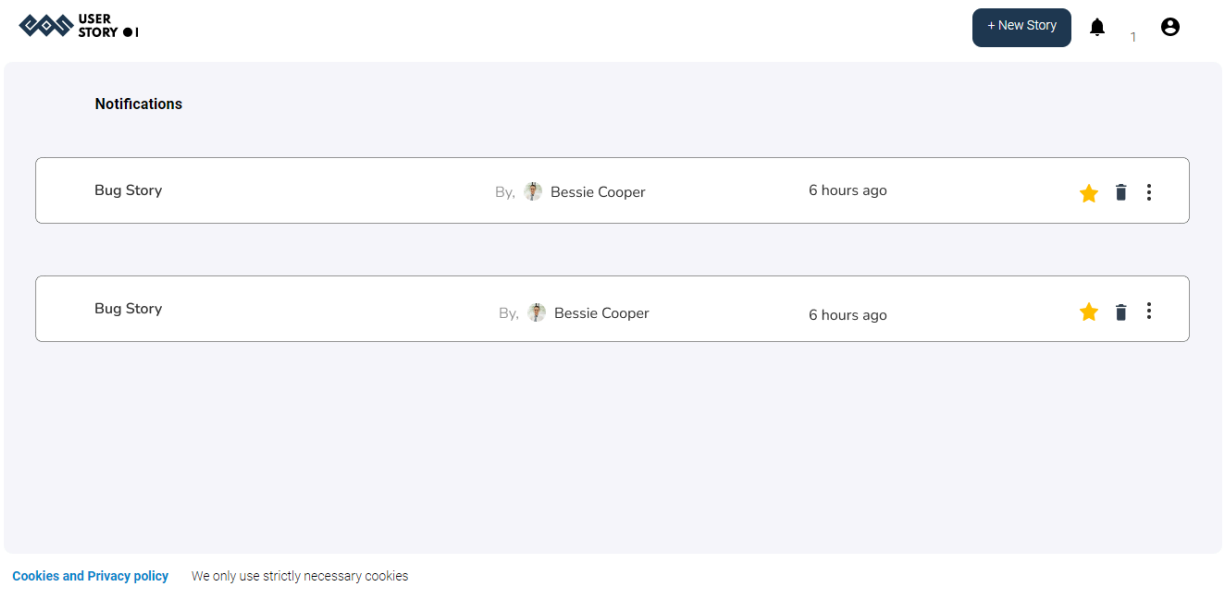




After clicking the add button a modal will open having title Add Label which is shown above and a user can add label and delete its label from there and choose appropriate labels there. Even after adding the label it can be deleted by using the red badge above the label. These labels will be visible to users in the story component shown above.
[Figma Link]

- **Revamping notification system (Frontend + Backend)**

  Notifications are very important for user engagement in an application. Any user wants to get notified of important events.Currently in our app we have a very primitive type of notification system. I propose to add features like delete, save and unsubscribe to a notification. I want to change the frontend of the notification page as well which will look like this



  To implement this in backend we need to update all our controllers and model lifecycle hooks where we are using notification service and in addition to this we need to add notification service for other important actions to like reply of comment,mention of user in a story,story report etc.In this regard I have updated strapi by adding notification for a vote action that when somebody upvotes your story. PR related to this link. Figma Link

- **Migration to TypeScript (Frontend)**

TypeScript adds type support to JavaScript and catches type errors during compilation to JavaScript. Bugs that are caused by false assumptions of some variable being of a certain type can be completely eradicated by adding TypeScript to the project.

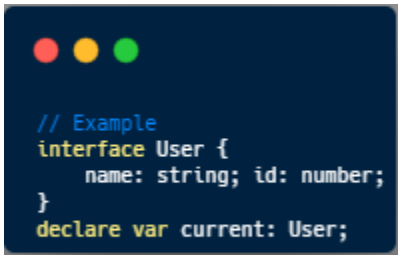
Migration Steps-

1. Add TypeScript to your project by `npm i -D typescript`
2. Make a tsconfig.json file that will have all definitions

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": [
      "dom",
      "dom.iterable",
      "esnext"
    ],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react"
  },
  "include": [
    "src"
  ]
}
```

3. Install all TypeScript definition files of project dependencies
4. Start changing JavaScript files to TypeScript files in a phased manner
   Various methods that we will employ to change the code-

- Creating Interfaces (Structure of a variable)



```
// Example
interface User {
    name: string; id: number;
}
declare var current: User;
```

- Defining prop types for every file.
- Defining definition of hooks in every file
  After changing files in TypeScript we will test all functionality thoroughly and we will also change cypress unit tests and migrate them to TypeScript.Cypress itself comes with a global type definitions we will add our type definitions in that to support custom functions.

```
// in cypress/support/index.ts
// load type definitions that come with Cypress module
/// <reference types="cypress" />

declare global {
  namespace Cypress {
    interface Chainable {
      /**
       * Custom command to select DOM element by data-cy attribute.
       * @example cy.dataCy('user-story')
       */
      dataCy(value: string): Chainable<Element>
    }
  }
}
```

- **Migration to Strapi V4 (Backend)**

  The need for migrating the current backend from Starpi V3 to Starpi V4 is because Strapi has introduced many code breaking changes in the new version and Strapi can also end its support for V3 and project is currently in development so it can undergo such change but when launched it would be difficult to make such change.

  Major changes to be done while migration-

1. Changing Database configuration file and setting up `PostgreSQL` because Strapi has removed support for MongoDB.We are using PostgreSQL because it is highly extensible.
2. Adding admin.js which will contain auth secret.
3. Updating all dependencies according to Strapi V4
4. Creating new controller files which contains new implementation eg-
5. Migrating GraphQL resolvers to the register method found in the `./src/index.js` file of Strapi v4
6. Other than this we will use codemods provided by strapi for migration.

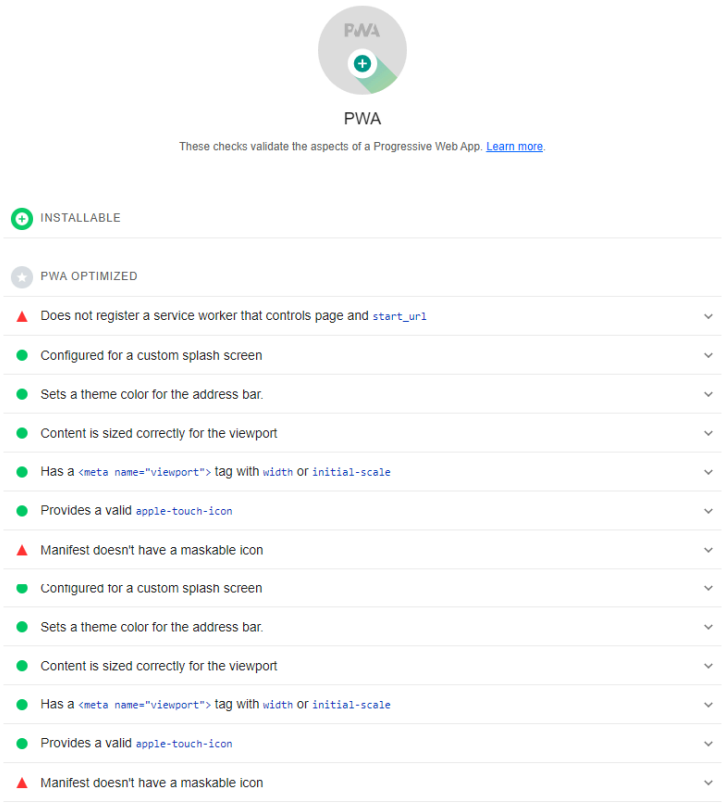
```
// path: ./src/api/<content-type-name>/controllers/<controller-name>.js

const { createCoreController } = require('@strapi/strapi').factories;

module.exports = createCoreController('api::api-name.content-type-name', ({ strapi }) => ({
  // wrap a core action, leaving core logic in place
  async find(ctx) {
    // some custom logic here
    ctx.query = { ...ctx.query, local: 'en' }

    // calling the default core action with super
    const { data, meta } = await super.find(ctx);

    // some more custom logic
    meta.date = Date.now()

    return { data, meta };
  },
}));}
  return go(f, seed, [])
}
```

(Controller in Strapi V4)

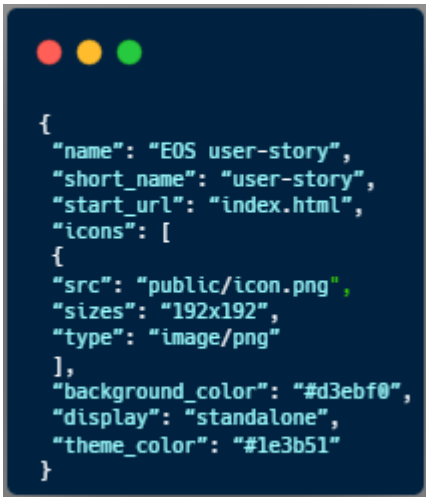
To solve the authentication problem in user story we will update our cookie configuration and will add an option to save cookies in localStorage if the browser does not allow storing cookies like safari browser does not allow sites to set cookies.

- **Progressive Web App (Frontend)**

  Progressive Web Apps (PWAs) are web apps that use service workers, manifests, and other web-platform features in combination with progressive enhancement to give users an experience on par with native apps.
  In order to make user-story a PWA we need to check a list provided by google which lists main aspects of a PWA.
  I ran a test on lighthouse to check performance of our app here is the test-



Main issue that comes in front of us in registering a service worker. Service worker loads instantly for the user regardless of the network state.
After adding service worker we will install that and add a manifest.json file it takes care of app name,path to icons etc



```
{
 "name": "EOS user-story",
 "short_name": "user-story",
 "start_url": "index.html",
 "icons": [
 {
 "src": "public/icon.png",
 "sizes": "192x192",
 "type": "image/png"
 ],
 "background_color": "#d3ebf0",
 "display": "standalone",
 "theme_color": "#1e3b51"
}
```

After adding manifest.json we will add files that need to be cached in the service worker.

We can also introduce push notification because a PWA has access device specific functionality also.

- **Adding testing in Strapi (Backend)**

Adding testing helps in delivering a safe system and makes the development process more secure so like our frontend we should also have unit testing in the backend.Lot of people face problems while setting up strapi so this helps us to figure out many problems easily.

We will use the jest framework for adding testing . Our major focus of testing will be controllers and model lifecycle methods.

In order to run tests we need to make a strapi instance that runs in a testing environment and also a database that create and delete between tests.

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

let instance;

async function setupStrapi() {
  if (!instance) {
    /** the following code in copied from `./node_modules/strapi/lib/Strapi.js` */
    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

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

- **UI/UX Improvements (Frontend)**
    1. Adding shareable links for profiles of users along with a copy button to copy the URL of the user-story.
    2. Allowing mention of user and story in description of a story.
    3. Adding network status toast notification to notify users when they are offline.
    4. Updating responses of different actions that users do with appropriate responses.
    5. Updating and adding test cases for increasing test coverage.

Schedule

Community Bonding Period (May 20 - June 12)

- Working on minor bugs and improvements .
- Learning related technologies like TypeScript.
- Refining the ideas and making a blueprint of the work.
- Discussion with mentors and other contributors about the project.

Week 1 (June 13 - June 19)

- Will start work on migration of Strapi V3 to V4.
- Preparing file structure.
- Adding new dependencies.
- Setting up configuration files.

Week 2 (June 20 - June 26)

- Complete configuration of Strapi V4.
- Start working on route migration.
- Resolving bugs that come during the migration

Week 3 (June 27 - July 3)

- Migration of controllers and services.

- Working on middleware.
- Discussion with mentors over the progress of the project.

Week 4 (4 July - 10 July)

- Working on GraphQl resolvers.
- Finishing up migration.
- Updating documentation of Strapi backend.

Week 5 (11 July - 17 July)

- Will start working on typescript migration of user-story.
- Initial setup required for migration.
- Working on the reviews received from mentors on Strapi migration.

Week 6 (18 July - 24 July)

- Finalizing tsconfig.json file.
- Installing dependencies compatible with Typescript.
- Start migration of code files of Javascript to TypeScript.

Week 7 (25 July - 31 July) First Evaluation

- Working on code files and migrating them to Typescript.
- Discussion with mentors over the progress.

Week 8 (1 August - 7 August)

- Testing the work done on migration of code.
- Updating documentation of user-story.
- Working on reviews received from mentors.

Week 9 (8 August - 14 August)

- Working on the comment section frontend part.
- Working on implementation of the comment section backend.

Week 10 (15 August - 21 August)

- Backend part of comment section to be completed.

- Testing the implemented feature.
- Discussion with mentors over the implemented features.

Week 11 (22 August - 28 August)

- Working on adding state management to user-story.
- Working on reviews received from mentors.
- Documenting the implemented feature.

Week 12 (29 August - 4 September)

- Continue with the previous week's work.
- Finalizing state management implementation.
- Testing state management.

Week 13 (5 September - 11 September) Second Evaluation

- Improving search and sort functionality.
- Working on reviews received from mentors.

Week 14 (12 September - 18 September)

- Working on adding labels functionality.
- Removing bugs that come during the implementation.

Week 15 (19 September - 25 September)

- Finalizing adding labels functionality.
- Testing the added functionality.
- Documenting the functionality.

Week 16 (26 September - 2 October)

- Start working on notification service.
- Working on reviews received from mentors.

Week 17 (3 October - 9 October)

- Implementing notification service in backend and frontend.
- Documenting the implementation.

Week 18 (10 October - 16 October)

- Will start implementing PWA.
- Working on reviews received from mentors.
- Discussion with mentors over PWA

Week 19 (17 October - 23 October)

- Will work on PWA service worker implementation.
- Testing PWA thoroughly.
- Documenting the implementation.

Week 20 (24 October - 30 October)

- Start adding tests to the Strapi backend.
- Working on reviews received from mentors.

Week 21 (31 October - 6 November)

- Complete writing all test cases for Strapi backend.
- Documenting the work done.

Week 22 (7 November - 13 November)

- Working on miscellaneous improvements.
- Testing the integration of backend and frontend.
- Discussion with mentors.

Week 23 (14 November - 21 November) Third Evaluation

- Working on final updates
- Documenting all the work done till now.
- Final submission

Previous Experience

I started working in JavaScript from my first year and slowly I started to learn new technologies like Vue , React, Node.js etc along with making self projects in them and also contributing to various open source organizations. I also learnt C and python during my studies.

In my first year I did an internship at APIcon.io. I worked on projects related to ApexCharts, Kanban system using React-beautiful-dnd, Rich text editor using Draft.js,Mobile app using React.js, Redis implementation for caching, API calls count and API analytics dashboard.

Other than this I have done an internship at Famstar in which I worked on a dashboard build using Next.js and express.js. Recently I completed my internship with Workhack in which I built a dashboard from scratch and used technologies like react.js,FASTAPI (python),Airtable. I worked on both the frontend and backend part of the dashboard.

I have also contributed to open source organizations like Catalyst, Layer5,Codeuino etc. I also served as mentor for KWOC(Kharagpur winter of code) 2020. I also conducted an international hackathon named Hack the Mountains 2020 which was sponsored by MLH.Other than this I have worked on numerous small and medium scale projects.

About Me

I am currently studying in third year at Shri Mata Vaishno Devi University (SMVDU), Katra. I like to explore things, especially when it is related to programming and computers. I also like reading books about the world,philosophy,and metaphysics. I like playing musical instruments in my free time. I have always worked for open source

and EOS is a great platform from which I can utilize my skills for solving real world problems. EOS gives a great opportunity to learn new things and polish our skills. Looking forward to working with nice people at EOS on new projects and technologies .

Stretched Goals

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

Working on a CLI that provides a simple method to run user story and integrates frontend and backend and gives the user as a single package. THe CLI will customize many settings for the user. This CLI will help other people to launch user story for their own organization.