{coding academy

MissBug

CRUDL end 2 end

Node.js

MissBug is a bug management system, it allows users to add / remove and update bugs.

Phase 1

Step 1 – Backend

Create a new folder: miss-bug-proj

- *npm init*
- Install the modules *express* and *cookie-parser*
- Create a *server.js* file:

```
import express from 'express'
const app = express()
app.get('/', (req, res) => res.send('Hello there'))
app.listen(3030, () => console.log('Server ready at port 3030'))
```

- Check it out
- Add a .gitignore file, commit and push your code to github: 'Initial backend setup'

- Our app manages a bug entity:

```
{
    "_id" : "abc123",
    "title" : "Cannot save a new car",
    "description" : "problem when clicking Save",
    "severity" : 3,
    "createdAt" : 1542107359454,
}
```

- Provide an API for Bugs CRUDL:
  (Implement one by one along with a bugService)

```
app.get('/api/bug', (req, res) => {})
app.get('/api/bug/save', (req, res) => {})
app.get('/api/bug/:bugId', (req, res) => {})
app.get('/api/bug/:bugId/remove', (req, res) => {})
```

- Test your API from the browser
- Remember to commit at every working stage

Use a frontend

- Get familiar with the provided frontend code
- Add a description to the bug entity (another prompt for now)
- In the BugDetails page, show the bug's description
- Commit your work
- Refactor the bugService to use your API instead of using localStorage

Backend - Add a cookie for usage limit

- Let's limit the user for viewing no more than 3 bugs during some time
  - Later on, we might want to encourage the user to signup and remove this limit, but this is out-of-scope for now
- So we need to keep track of the bugs the user visited
- From the backend we will send a cookie: *visitedBugs*
  in which we will store an array of visited bug ids
  Make sure the array is sent as cookie and saved by the browser (check the dev tools)
- Make sure it works by printing a message to the backend console:
  User visited at the following bugs: […]
- When user visits more than 3 different bugs we will respond with an error:

```
return res.status(401).send('Wait for a bit')
```

- Make that cookie last for 7 seconds
- Note that you can clear the cookies at any time using the dev-tools
- git commit your work

Bug Filter

Allow filtering the bugs,

Bonus – Get a PDF

Allow the user to download a PDF file of the bugs 

Phase 2 – REST API + Sorting, Paging, Filtering

Model

The bug should now have the following properties:

```
{
    _id : "abc123",
    title : "Cannot save a Car",
    description : "problem when clicking Save",
    severity : 3,
    createdAt : 1542107359454,
    labels : ['critical', 'need-CR', 'dev-branch'],
}
```

Backend

Convert your backend to provide a RESTful API on the entity bug.

Support server side filtering, sorting and paging:

1. Sorting examples:
    a. ?sortBy=title
    b. ?sortBy=severity
    c. ?sortBy=createdAt&sortDir=-1
2. Paging: ?pageIdx=3
3. Filtering:
    a. by txt
    b. by min severity
    c. by labels (check if any of the labels is included)

Use postman to test your API

Frontend

- Update your frontend to use the REST API
- Add more filtering options, sorting and pagination

Phase 3 – User Support

Model

The bug should have the following properties:

```
{
    "_id": "abc123",
    "title": "Cannot save a Car",
    "description": "errors when clicking Save",
    "severity": 3,
    "createdAt": 1542107359454,
    "creator": {
      "_id": "u101",
      "fullname": "Puki Ja"
    }
}
```

Backend

- Add user.json – that holds all the users and a userService
- Add userRoute
    - /api/auth/signup – add a new user to the file
    - /api/auth/login – check if username and password are correct - generate a loginToken and return a *mini-user* to the frontend
        - When bug is added – get the creator from the loginToken
        - Only the bug's creator can DELETE/UPDATE a bug
          Update only updatable fields
    - /api/auth/logout – clear the cookie
- Test your API from POSTMAN

Frontend

- Use or Create the component: <LoginSignup>
- Add a userService
    - Implement the functions: `login`, `signup`, `logout`, `getLoggedinUser`
    - Use the sessionStorage to hold the loggedinUser and survive browser refresh
      W
- Add a <UserDetails> page
    - This is a user profile page
    - Show the user's **bugs** (bugs that he has created)
        - Can you reuse your bug-list component?
    - At the header, add a Profile link that route to UserDetails page of the logged-in user.

Implement ownership

- When adding a new Bug – add the creator (the loggeinUser)

- Only the bug's creator can DELETE/UPDATE a bug
- Use **postman** to test the APIs

Add Admin Support

- Add isAdmin to the user entity
- Hard-coded mark a user (username: admin, pass: admin) as admin in your user.json file
- Admin can delete / edit all bugs
    - Admin has a link to UserIndex page where he can view and delete users
    - Bonus: Prevent deletion of users that own bugs

Deploy to Render

Follow the needed steps to upload your project to Render.com

Set up a SECRET1 environment variable holding the encryption key

Done!