**ISSUE TRACKING TOOL**

**PROJECT REPORT**

**PROJECT OVERVIEW :**

This project is a ready to deploy, Issue Tracking Tool.

Reporter reports an issue and assigns it to a user.

Reporter can add/edit title of issue, Description of issue, add/edit any related attachments, like screenshots. Users including reporter and assignee, can assign the issue, to any another user (called assignee, hereafter). Also there a comments section, where any user, should can comment around this issue

It is a MEANSTACK based application.

**There are two separate parts of the application :**

* FRONTEND - ANGULAR 8, HTML5, CSS3, JS, Bootstrap
* BACKEND - NodeJS, ExpressJS and Socket.IO Database Allowed - MongoDB and Redis

Angular is a platform that makes it easy to build web based applications. It combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop

**How to create the angular boilerplate application ?**

Firstly create the boilerplate frontend application using following commands

* ng new Issue-Tracking-Tool
* cd Issue-Tracking-Tool
* ng serve

Create module folders according to the requirement of the project by the following command:

ng g module modulename

and inside module folder create components:

ng g component componentname

**Getting Started**

The instructions provided will get a copy of the project up and running on the local machine for development and testing purposes.

**Prerequisites**

**Make sure before creating an angular application following things are installed :**

* Visual studio code (IDE) (Download from the official site)
* Node JS and NPM (npm comes preinstalled in nodejs now a days)

To check you have node and npm installed on your system,run following commands:

* node -v

npm –v (NPM is a separate project from node js and tends to update itself more frequently.

Therefore we can update the npm with its latest version by this command , npm install npm@latest -g)

* Typescript installed- (npm install –g typescript)
* Install git
* Angular-CLI –(npm install –g @angular/cli)

**Running locally**

1. Create a folder named as Issue-Tracking-Tool
2. change directory to Issue-Tracking-Tool

cd Issue-Tracking-Tool

1. Fetch the source code from my github library

git init

git remote add origin https://github.com/Surabhie/Issue-tracker-Frontend.git

git pull origin master

1. Install all the modules required to run the given application with following command

npm install (This command will install all the modules that is inside package.json)

1. Run the application by using following command

ng serve --open

1. This will navigate to <http://localhost:4200/> on the browser .

And then the application runs.

**REQUIREMENTS AND FLOW:**

APPLICATION MODULES :

1. Personalized-Dashboard Module
2. User Module
3. Shared Module
4. Issue-description Module
5. Server-error Module

Each module contains components based on the functionalities of the application.

COMPONENTS :

1- USER MODULE

Sign-up, Login ,Reset-password ,Update password.

2-PERSONALIZED-DASHBOARD MODULE :

Home Component

1. ISSUE-DESCRIPTION VIEW- Create-Issue component, Edit- Issue component, view-issue component.

Note : (I have kept the components separate intentionally and not merged in just one view for better understanding and my learning)

4-SHARED MODULE

my-nav

4- server-error component

5-Page-not-found component

**Features of the Application**

Features of the platform​-

1. Login View : comes under user module🡪 login component

2. Personalized Dashboard View : comes under personalized dashboard module ->home component

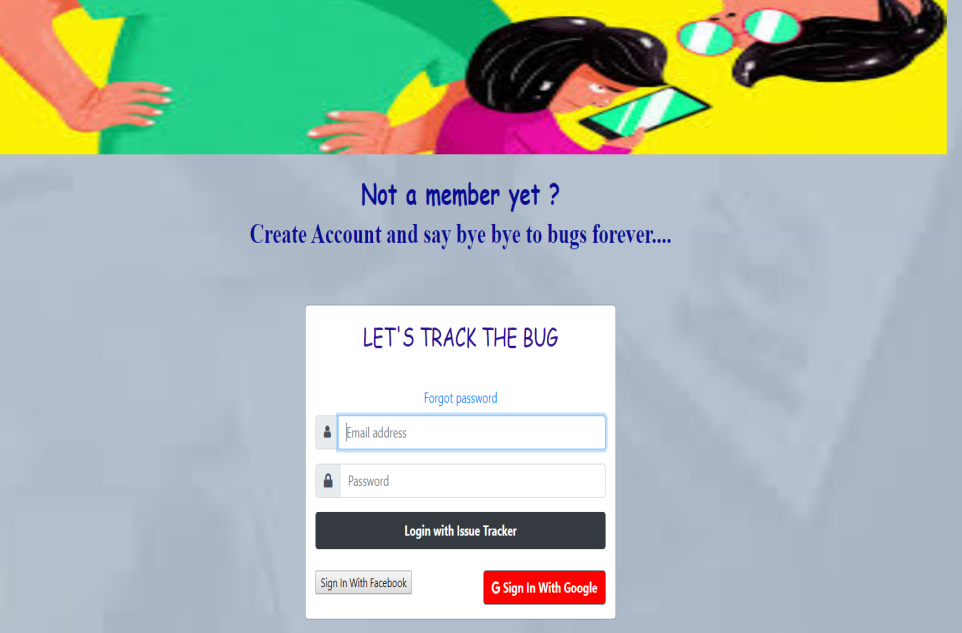
3. Issue description view : comes under issue-description module -> (create issue component,edit –issue component,view-issue component)

4. Search view : comes under personalized dashboard module -> (home component in the form of table)

**Login Page**

[nikisha@gmail.com](mailto:nikisha@gmail.com)

12345678



a. User is able to login to the system through his username/password or social logins. . (user module -> login component)

NORMAL LOGIN

//LOGIN

  signinFunction() {

    this.progress = true;

    // check for email

    if (this.email) {

      // check for password

      if (this.password.length >= 8) {

        let data = {

          email: this.email.toLowerCase(),

          password: this.password

        }

        this.appService.signinFunction(data)

          .subscribe((apiResponse) => {

            if (apiResponse.status === 200) {

              this.toastr.success("Login Successfull");

              this.\_cookieService.set('authtoken', apiResponse.data.authToken);

              this.\_cookieService.set('receiverId', apiResponse.data.userDetails.userId);

              this.\_cookieService.set('receiverName', `${apiResponse.data.userDetails.firstName} ${apiResponse.data.userDetails.lastName}`);

              this.appService.setUserInfoInLocalStorage(apiResponse.data.userDetails);

              setTimeout(() => {

                this.router.navigate(['/user/home']);

              }, 500);

            } else if (apiResponse.status === 404) {

              this.progress = false;

              this.snackBar.open(`Email or Password wrong`, "Dismiss", {

                duration: 5000,

              });

            } else {

              this.snackBar.open(`${apiResponse.message}`, "Dismiss", {

                duration: 5000,

              });

            }

          }, (err) => {

            this.snackBar.open(`some error occured`, "Dismiss", {

              duration: 5000,

            });

            setTimeout(() => {

              this.router.navigate(['/500'])

            }, 500);

          });

      } else {

        this.snackBar.open(`Make sure your password is more than 8 random characters`, "Dismiss", {

          duration: 5000,

        })

      }// check for password ends here

    } else {

      this.snackBar.open(`Please enter a valid Email and Password`, "Dismiss", {

        duration: 5000,

      })

    } // check for email ends here

  }

HOW TO USE SOCIAL LOGIN CONCEPT:

Steps :

**Install angularx-social-login**

* npm install angularx-social-login–save

**STEPS TO SETUP A FACEBOOK APP LOGIN**

* To get an AppID/Secret ,goto facebook developers login with your facebook developers account or credentials.
* Click on add a new app and type the name of the application,then click

‘**create App ID button’**

* Then App id gets created .Note it down.
* Then goto settings,add platform,website and mention the URL and save changes.

**STEPS TO SETUP A GOOGLE APP LOGIN**

* Create a google app and get client id.
* The first thing we need is create a google project and get user credentials.
* Go to Google API Console and click on credentials .
* Create credentials -> choose OAuth client ID
* Select web applications,enter project URL and click on Create button
* This will create a client ID and secret key

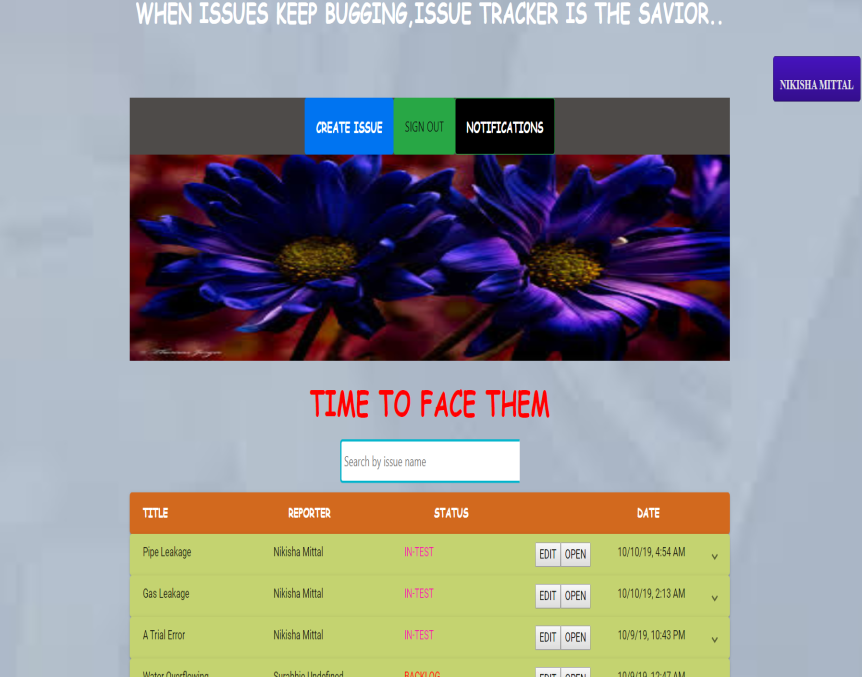
b. User is able to register also. (user module -> sign-up component)

c. Upon login, user should land on his ​Personalized Dashboard View​.

(personalized-dashboard view->home component)

                this.router.navigate(['/user/home']);

**Personalized Dashboard View**



A table showing all issues currently assigned to logged-in user is shown.

It has following columns.

1. Status : current status of the issue i.e backlog, in- progress,­­­­­­­ in-test, done.

2. Title:​ Title of the bug.

3. Reporter:​ User who reported the bug

4. Date:​ Date when this bug was reported.

5. A search box where, User should be able to search for an issue, which would open

​Search View.

6. A create button: To log a new issue. On clicking, user should be taken to issue description view. (CREATE ISSUE BUTTON)

NOTE:

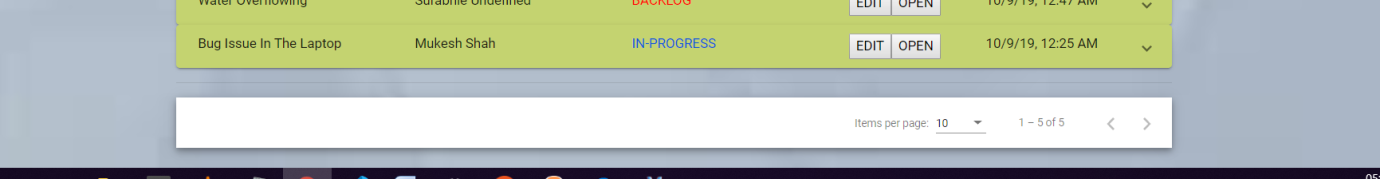
* 1. **Table is paginated**

**In** [**home.component.ts**](http://home.component.ts)

 public pageSize: number = 10;

  public pageSizeOptions: number[] = [5, 10, 25, 100];

  pageIndex: number = 0;



2. Table has sorting on columns

//  Table should have sorting on columns ,So sorting function

  sortBy(type: string) {

    if (type == "title") {

      if (this.sort == "title.1") {

        this.sort = 'title.-1'

      } else {

        this.sort = 'title.1'

      }

    } else if (type == "reporter") {

      if (this.sort == "reporter.1") {

        this.sort = 'reporter.-1'

      } else {

        this.sort = 'reporter.1'

      }

    } else if (type == "status") {

      if (this.sort == "status.1") {

        this.sort = 'status.-1'

      } else {

        this.sort = 'status.1'

      }

    } else {

      if (this.sort == "createdOn.1") {

        this.sort = 'createdOn.-1'

      } else {

        this.sort = 'createdOn.1'

      }

    }

    this.getAllIssue(this.pageSize, this.pageIndex, this.sort);

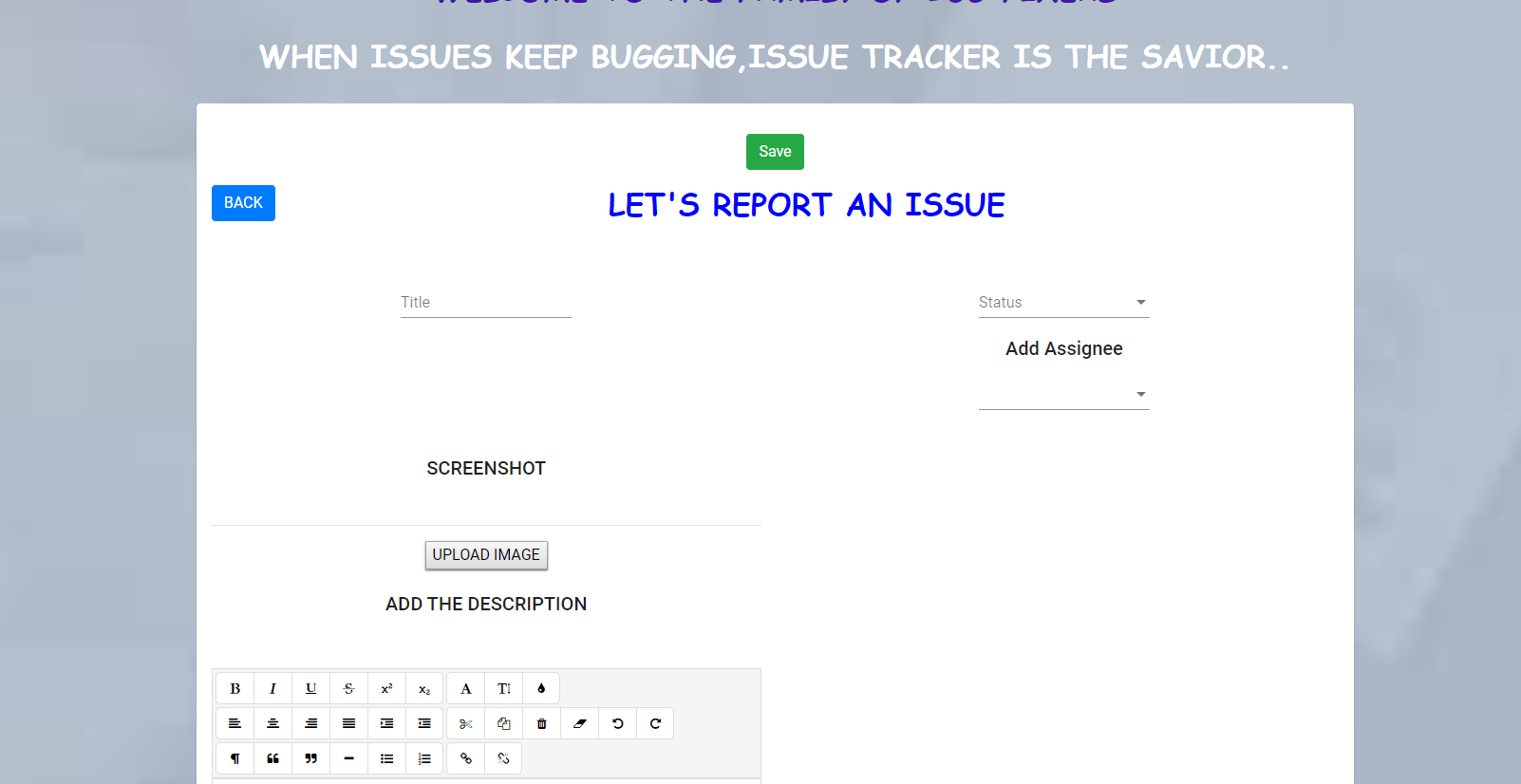
  }

3. User can filter rows based on any columns.

4. Upon clicking on any row, ​Issue Description View​ opens (ARROW IS AT THE LAST )

1. Issue description view :

a-create-issuecomponent



For upload image:

  onFileSelected(event) {

    this.warning = false

    this.selectFile = <File>event.target.files[0];

    if (this.selectFile) {

      let reader = new FileReader();

      reader.onload = (event: any) => {

        this.imageUrl = event.target.result;

      }

      reader.readAsDataURL(this.selectFile);

      if (this.selectFile.size > 5000000) {

        this.warning = true

        this.message = "Please make sure your image is less than 5Mb for ensuring the performance of the app"

      }

      if (this.selectFile.type == "image/png" || this.selectFile.type == "image/jpeg") {

      } else {

        this.warning = true

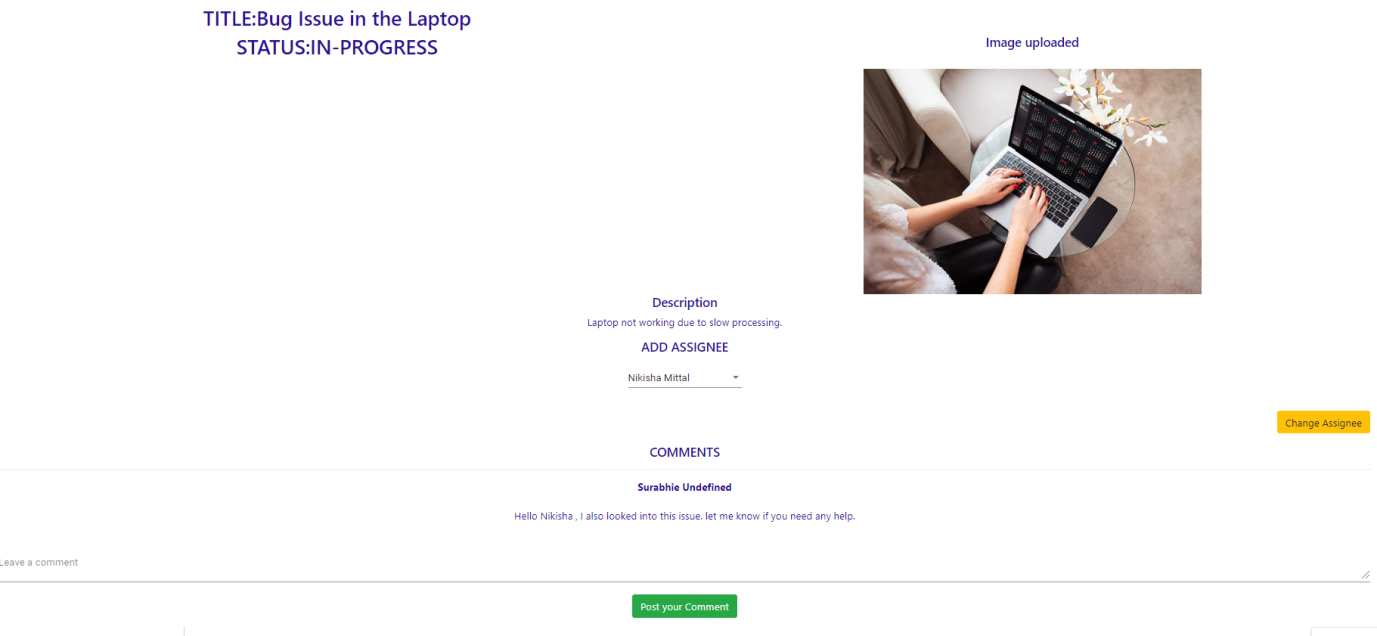
        this.message = "Please make sure your image format is Jpeg/Png"

      }

    }

  }

b-edit-issue component (this view comes when userid and reporter id are different)



GET ISSUE DETAILS

//get Issue details

  getIssueDetails() {

    this.issueService.getIssueInfo(this.currentUrl).subscribe(

      data => {

        if (data['status'] == 200) {

          let response = data['data']

          this.imageUrl = `http://localhost:3000/uploads/${response.screenshot}`

          this.title = response.title

          this.status = response.status

          this.editorContent = response.description

          this.reporterId = response.reporter[0].userId

          this.reporter = response.reporter

          this.previous = response.screenshot

          this.commentsArray = response.comments

          this.watchers = response.watching

          // To check userId of assignee and give them rights to edit

          response.assignee.filter(x => this.assigneeArray.push(x.userId));

          response.watching.filter(x => {

            if (x.userId == this.userId) {

              this.watchee = true;

            }

          })

          setTimeout(() => {

            this.anotherList = [];

            response.assignee.filter(x => {

              for (let y of this.users) {

                if (y.userId == x.userId) {

                  // for Default Checking of value

                  this.anotherList.push(y);

                }

              }

            });

            this.assignee.setValue(this.anotherList);

          }, 3000);

        } else {

          this.snackBar.open(`some error occured`, "Dismiss", {

            duration: 5000,

          });

          setTimeout(() => {

            this.router.navigate(['/500'])

          }, 500);

        }

      }, (err) => {

        this.snackBar.open(`some error occured`, "Dismiss", {

          duration: 5000,

        });

        setTimeout(() => {

          this.router.navigate(['/500'])

        }, 500);

      });

  }

c-issue-view (simple-view)



1. Here user(Reporter) can add/edit title of issue(create issue and edit issue component) ,

Description of issue, add/edit any related attachments, like screenshots. Description Box is a rich text editor having options like font styling, underline etc. .

1. Any user, including reporter and assignee, is able to assign this issue, to any another user (called assignee, hereafter).

 public addAssignee() {

    let data = {

      assignee: this.assignee.value,

      id: this.currentUrl

    }

    this.issueService.addAssignee(data).subscribe(

      data => {

        if (data['status'] == 200) {

          // this.router.navigate(['/home'])

          this.notify(`${this.name} has Added Assignee on ${this.title}`);

          this.snackBar.open(`${data['message']}`, "Dismiss", {

            duration: 5000,

          });

        } else {

          this.snackBar.open(`some error occured`, "Dismiss", {

            duration: 5000,

          });

          setTimeout(() => {

            this.router.navigate(['/500'])

          }, 500);

        }

      }, (err) => {

        this.snackBar.open(`some error occured`, "Dismiss", {

          duration: 5000,

        });

        setTimeout(() => {

          this.router.navigate(['/500'])

        }, 500);

      });

  }

1. Any user can make changes to this issue.
2. Also, there is a comments section, where any user, is able to make comments around this issue.

 public postComment() {

    if (this.comment) {

      let data = {

        id: this.currentUrl,

        comment: this.comment

      }

      this.issueService.postComment(data).subscribe(

        data => {

          if (data['status'] == 200) {

            this.notify(`${this.name} has Commented ${this.comment} on ${this.title}`);

            this.snackBar.open(`${data['message']}`, "Dismiss", {

              duration: 5000,

            });

            this.router.navigate(['user/home'])

          } else {

            this.snackBar.open(`some error occured`, "Dismiss", {

              duration: 5000,

            });

            setTimeout(() => {

              this.router.navigate(['/500'])

            }, 500);

          }

        }, (err) => {

          this.snackBar.open(`some error occured`, "Dismiss", {

            duration: 5000,

          });

          setTimeout(() => {

            this.router.navigate(['/500'])

          }, 500);

        });

    } else {

      this.snackBar.open(`Comment box empty!`, "Dismiss", {

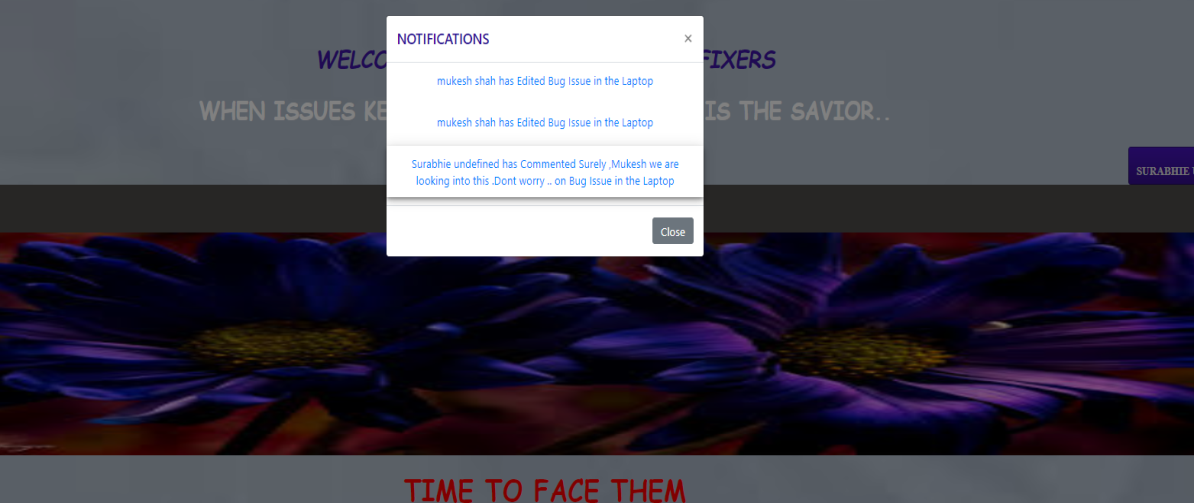
        duration: 5000,

      });

    }

  }

1. Further, Any user should is able to add himself, as watcher, to this issue.(if he is not the reporter)
2. A button called "Watch" is there, for this.
3. List of watchersis also shown(There can be more than one watcher).
4. All watchers, assignee and reporter, receive notifications, for any changes or comments on this issue.
5. Notification comes on screen, having a short description of what changed,



 // get notifications of the user

  public getNotify: any = () => {

    this.SocketService.notify(this.userId)

      .subscribe((data) => {

        let message = data;

        this.snackBar.open(`${message.message}`, "Dismiss", {

          duration: 5000,

        });

      }, (err) => {

        this.snackBar.open(`some error occured`, "Dismiss", {

          duration: 5000,

        });

        setTimeout(() => {

          this.router.navigate(['/500'])

        }, 500);

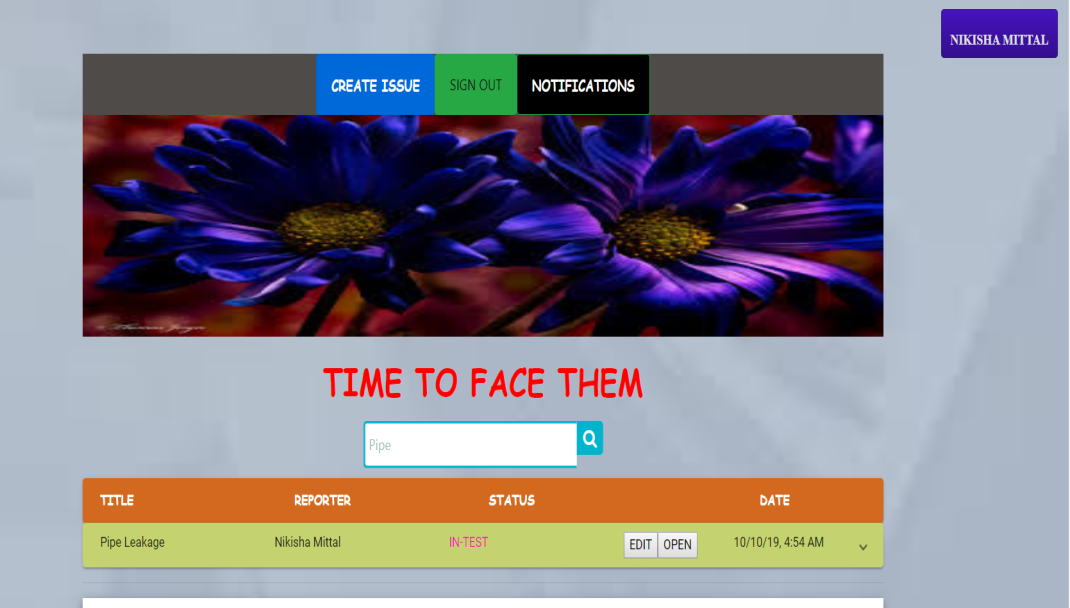
      });//end subscribe

  }// end get message from a user

1. Also, when clicked on notification, user lands on the ​Issue Description View ​of related issue.

(edit-issue view)

4. Search view



1. User is able to search for any text.

2. User is shown a results table, having all the issues related to the search text.

3. This table is similar to the table in ​Personalized Dashboard​ ​View​.

search(e) {

    this.cross = true;

    this.issueService.searchIssue(this.searchText).subscribe(data => {

      let response = data['data']

      this.length = data['count']

      if (data['status'] == 200) {

        this.issues = []

        this.issues = response;

      } else if (data['status'] == 404) {

        this.none = true

        this.snackBar.open(`${data['message']}.`, "Dismiss", {

          duration: 2000,

        });

        setTimeout(() => {

          this.none = false

        }, 5000);

      } else {

        this.snackBar.open(`some error occured`, "Dismiss", {

          duration: 5000,

        });

        setTimeout(() => {

          this.router.navigate(['/500'])

        }, 500);

      }

    }, () => {

    });

  }

**In home.html for search input:**

      <div class="wrap">

        <div class="search">

          <input type="text" class="searchTerm" placeholder="Search by issue name" (change)="search($event)"

            [(ngModel)]="searchText">

          <button \*ngIf="cross" type="submit" class="searchButton" (click)="ngOnInit()">

            <i class="fa fa-search"></i>

          </button>

        </div>

      </div>

Error Views and messages - Each major error response (like 404 or 500) are handled by different pages.

Suppose if a user tries to access the page that is not in the application he/she will get a 'Page Not Found Page'.

Backend PART : NodejS

**APPLICATION STRUCTURE :**

**APP**

**Controllers**

**Libs**

**Middlewares**

**Models**

**Routes**

**CONFIG :**

**appConfig.js**

**STEPS TO CREATE THE APPLICATION :**

1. Firstly create a folder issue-tracking-tool-backend
2. Then run npm init command. It basically tries to generate package.json file
3. npm install express –save

Now we have the node\_modules folder. In the similar manner, we will install all the packages that are required in our application.

FOR HANDLING ERRORS

* 1. Created an appErrorHandler middleware and routeLevelMiddleware to log the information of the routes.

appErrorHandler- checks for errors and sends the error response. Route not found error handler is also executed .

LIBRARIES:

Created response format

WHILE GENERTAING TOKENS:

1. Token is generated after password validation.
2. token.generateToken (userDetails)where token specifies token library and generateToken is the method in the library.
3. Then passed the userDetails as it is expecting some data.

**Why do we create Auth models :**

JWT has a token that expires(we have set it).

But prob is once token is set,it contains user info.you cannot delete it by going on everyone’s browser and deleting the cookie. So we create a Auth model.

And in login function we use it and add save token function

PASSWORD HASHING:

Password was encrypted before storage to prevent unauthorized access.

Two functions created for this: one to create hashing(used in sign up code later),other to compare the password(used in login code)

EVENT DRIVEN PROGRAMMING –

There is an event and an event handler.

Whenever an event is emitted, even handler function does its work.

The system is waiting for some input from the user. And that input is translated to an event. And when that event is trigerred,whatever associated with that event is called.

**Built With**

* [Angular](https://angular.io/) - The web based framework used for Frontend Design
* [NPM](https://www.npmjs.com/) - Most of the modules are used
* [nodemailer](https://nodemailer.com/about/) - NPM module to send the mails
* [apiDoc](http://apidocjs.com/) - NPM module to create the apiDoc and eventDoc

**Authors**

* Surabhi J – Application created
* Edwisor for providing instructions

**License**

Use of this source code is governed by an MIT-style license that can be found in the LICENSE file at https://angular.io/license.

**Acknowledgments**

* Problem statement provided by Edwisor.