



THESIS PORTAL FRONT END DEVELOPMENT PROCESS

PREPARED FOR

Dr. Claudia Krull

Support Internationals at FIN

PREPARED BY

Mariam Riaz

Abstract

This document will serve as the technical documentation, of development process of the "Thesis Portal". This portal is being development under the supervision of Dr.Claudia Krull as a project under Support Internationals and FIN. Thesis Portal main purpose will be to become a common touch point for all the Supervisors/Professors at FIN to upload available thesis topics and for the students to have a one stop solution to search for a thesis topic.

Main objectives of Thesis Portal are:

- To reduce the time students spend in finding master thesis topics.
- To give more visibility to ongoing research works and possible master thesis topics in various groups at the **Faculty of Informatics and beyond.**



FRONT END DEVELOPMENT

1. Front End Overview

Front end development of Thesis Portal Web Application is being done via Angular. Angular is a framework which allows us to create reactive, single page applications.

2. Angular

Angular is a TypeScript-based open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular helps front end developers throughout the world to develop creative, reactive and light single page applications. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly.

You can learn more about angular at;

[Angular Website](#)

[Wikipedia](#)

[Get started with Angular, video tutorial](#)

3. Technical Prerequisites

Before getting started with angular, we have to make sure that our development environment includes;

1. Node js

Angular requires Node.js version 10.9.0 or later.

To check the version, run **node -v in a terminal/console window**.

To get Node.js, go to nodejs.org.

2. Node package manager

Angular, the Angular CLI and Angular apps depend on features and functionality provided by the libraries that are available as [npm packages](#). To download and install npm packages, one must have an npm package manager.

This setup guide uses the [npm client](#) command line interface, which is installed with Node.js by default. To check that one has the npm client installed, run;

npm -v in a terminal/console window.

3. The npm version that I installed is 6.12.0, Angular version is 8.3.19 and the node is 12.13.0. If you want to run the code from our repository please make sure to have the local versions of npm and angular as the same in the environment you are running the project, else you might get some errors.
4. If you run in the problem of global and local version differences, you can do; ng update, npm update and then ng serve again (this should solve the problem)

4. Install Angular

We use the Angular CLI (command line interface) to create projects, generate application and library code, and perform a variety of ongoing development tasks such as testing, bundling, and deployment.

Install the Angular CLI globally.

To install the CLI using npm, open a terminal/console window and enter the following command:

```
npm install -g @angular/cli
```

To check if the installation was successful run;

ng --version in a terminal/console window

Once the Angular CLI is downloaded we can create an empty example app like this: create a workspace for our application, create a new folder where you want to store all of your application packages and then run command prompt there.

To create a new workspace and initial starter app:

Run the CLI command ng new and provide the name my-app, as shown here:

```
ng new my-app
```

The Angular CLI installs the necessary Angular npm packages and other dependencies.

Once you have made the app, you can see it live working by using the command in the command line by

ng serve --open

This will open your app and it will usually be hosting at <http://localhost:4200/>.

This page will keep on reloading every time you make any change in the code.

5. Install Code Editor

To start coding we need a code editor or IDE, the code editor that we have used is “Visual Studio Code” or VS code. It is cross platform light editor which one can download from;

code.visualstudio.com

You can use any other editor as well. Now you can open your project from Visual studio code by going to -> File -> Open Folder.

6. Version Control System

The version control system that we are using is GitHub, and the repository link is;

[Tp-frontend-service](#)

7. Issue Tracking System

The Issue Tracking system that we are using is JIRA, to decide on sprint tasks and division of work between the team. JIRA board for this project can be accessed at;

[JIRA board](#)



MILESTONE-1

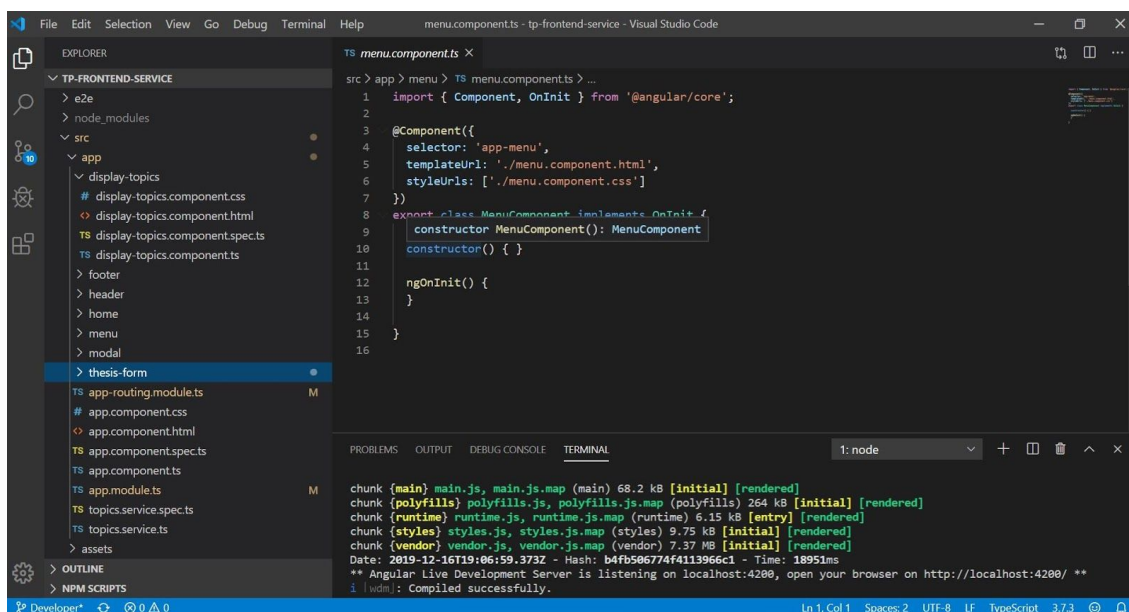
1. Design of Thesis Topic Submission Page

To design the thesis topic submission page, we have added a component (to know more about what a component is please refer, [here](#)) in the existing application by running a command in the vs code terminal.

ng generate component [component-name]

This will create a new component in the angular application that will basically contains 4 files.

1. .html file, in which you will write your html code.
2. .css file, in which you will giving the styling to the html script.
3. .ts file, in which you will write the logic of the component.
4. .spec.ts, to write tests for your component.



After the generation of thesis form component the design looks like this;

Thesis Portal

Home Create topic Available Topics Search

Title : Enter Thesis Topic Title

Research Group : Enter Research Group Here

Supervisor : Name of the Supervisor

Description : Please Enter Detailed description here

Must Have : Must Have Skills for People Applying

Nice to Have : Must Have Skills for People Applying

Contact Information : Enter the mode of contact and details

Start Date : YYYY-MM-DD

Create Topic

A project of Support International @ FIN

2. Display all thesis topics

To display all the thesis topics a component was created in the application named display-topics.

It is linked with a mock json server which is serving as a database to store all the topics and display them, later it will be replaced with a rest API of the backend server.

The mock json server is linked to the display-topics through a topics.service that is created in the project.

```

src > app > TS topics.service.ts > TopicsService > setTopics
5 import { Observable, of } from 'rxjs';
6 import { catchError, map, tap } from 'rxjs/operators';
7
8 @Injectable({
9   providedIn: 'root'
10 })
11 export class TopicsService {
12
13   constructor(private http: HttpClient) {}
14
15   getTopics () {
16     return this.http.get(environment.dbUrl);
17   }
18
19   //this is a post service which isnt used yet in the code
20   setTopics (topic) {
21     return this.http.post(environment.dbUrl, {
22       topic
23     })
24     .subscribe(console.log);
25   }
26 }
27
28
29

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

i | wdms | Compiled successfully.

Ln 24, Col 5 Spaces: 2 UTF-8 CRLF TypeScript 3.7.3



MILESTONE-2

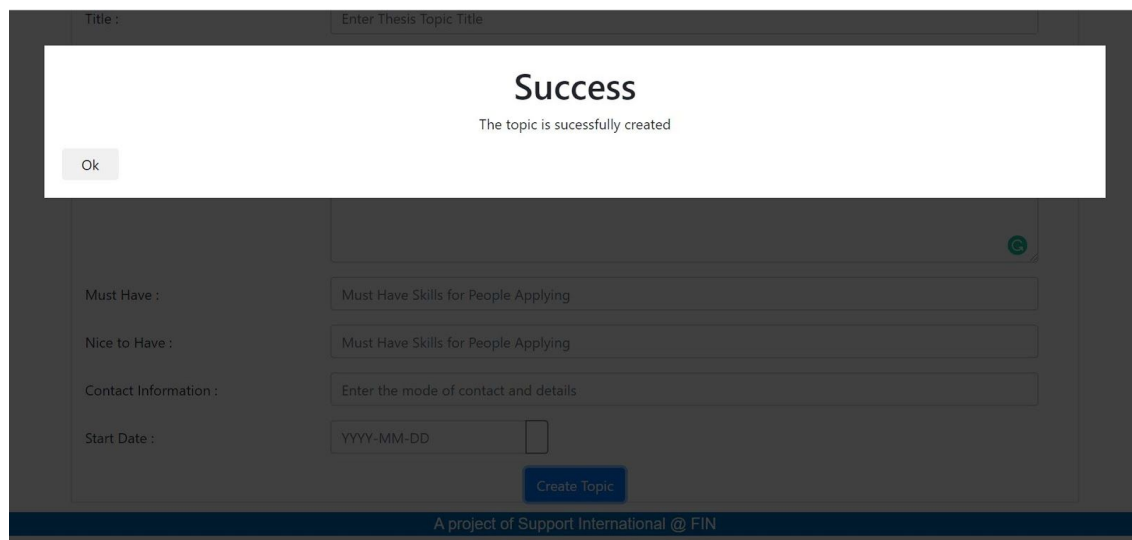
1. Popup On click for create topic

In bootstrap and angular when you want to have a popup on click of any button, or text, this component is basically called a modal. You can read more about it [here](#).

Each modal should have a unique id, as when called from modal.service that particular modal will be open and closed in the app, incase you have multiple modals in the app.

Service is a broad category encompassing any value, function, or feature that an app needs. A component can delegate certain tasks to services, such as fetching data from the server, validating user input, or logging directly to the console. By defining such processing tasks in an *injectable service class*, you make those tasks available to any component.

To read more about services; [visit here](#)

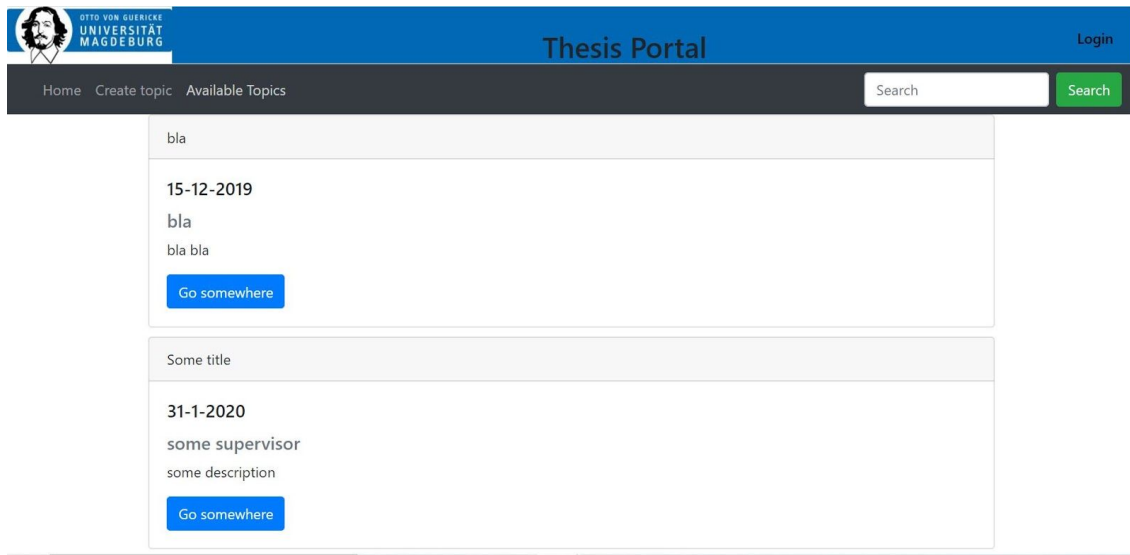


The screenshot displays a web application interface. At the top, there is a header with the text "Title : Enter Thesis Topic Title". Below this, a large white modal box is centered, featuring the word "Success" in a large, bold font. Underneath "Success", it says "The topic is sucessfully created". A small "Ok" button is located in the bottom left corner of the modal. In the background, a form is visible with several input fields. The form has a dark theme. The fields are labeled: "Must Have :", "Nice to Have :", "Contact Information :", and "Start Date :". The "Must Have" and "Nice to Have" fields contain the text "Must Have Skills for People Applying". The "Contact Information" field contains the text "Enter the mode of contact and details". The "Start Date" field contains the text "YYYY-MM-DD" and has a calendar icon to its right. A blue "Create Topic" button is positioned at the bottom right of the form. At the very bottom of the page, a dark blue footer contains the text "A project of Support International @ FIN".

2. Change display-topics from table to a list view

I have used bootstrap cards to display the topics in a list format.

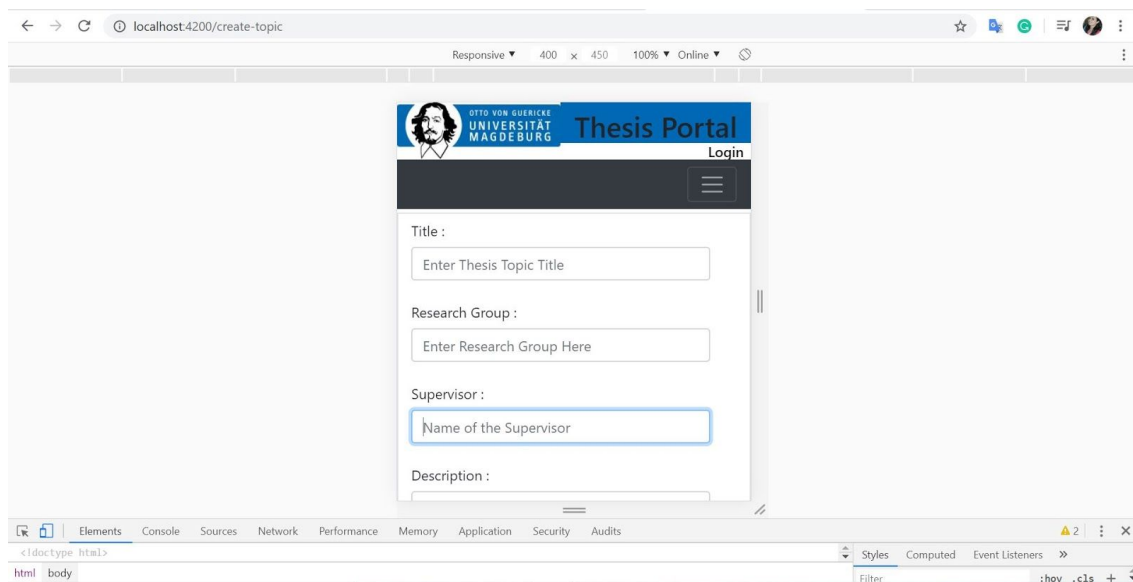
The definition of cards are done in the .html file and the styling in the .css file



3. Mobile view

The third task for this milestone for me was to check the website capability to change its view for mobile. This can be checked in following flow;

1. While hosting the app , from your browser go to tools. I have used chrome so the option say “More tools”.
2. Under more tools option → Developer tools.
3. When you open the developers tools you can see there is an icon to change the view from a computer view to mobile view.
4. When you click that, you will see if your website is responsive or not. At the moment our website looks like this;



If the header and footer of the app and the pages are not getting resized with the mobile view, it means your app is not responsive to mobile view.



MILESTONE-3

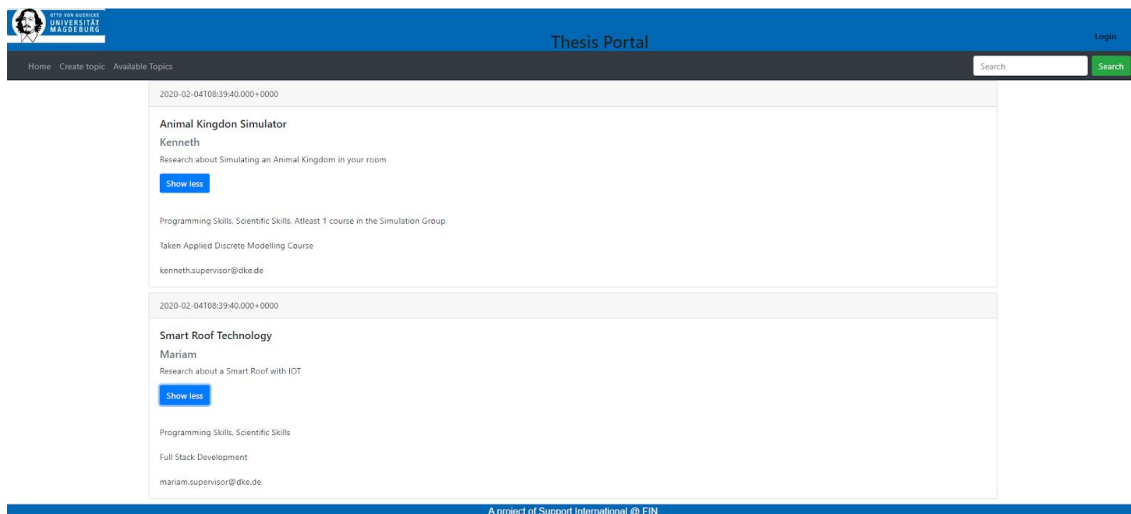
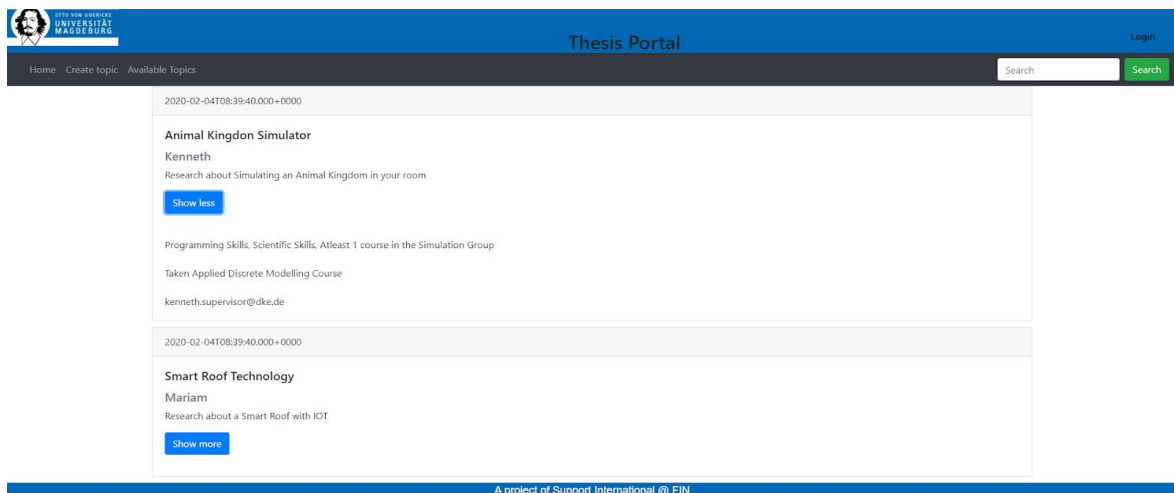
1. “Show More” Button

To incorporate a show more button, the idea was to add a drop down kind of portion which when clicked show the rest of the fields of the available topics, like Must have, Nice to have and the contact of the supervisor.

The show more button and the collapsable portion is added in the display-topics-component, toggling the visibility of a content by pressing a button or text is done in bootstrap by collapsable class .

You can read more about collapsible elements of bootstrap [here](#).

The implementation of the collapsable element can be seen in the code, in files display-topics-component.ts, display-topics-component.html, display-topics-component.css.



2. Consuming the Back-End API

The main task in consuming the Back End API was to set up, backend and front end services on the same computer and then let them run simultaneously and check if they are working fine.

For Front-End API consumption, the main part is to write the methods that will take the inserted data from front end, (Thesis-form-component) and then store them in the back end database.

All the methods to get stored the topics from front end to back end are written in a topics.service.ts (set topics). Get topics to get data from back end to front end)

After doing that the main part was to match the fields of Back-End database(attributes naming) to be the same as the topic fields used in front end.

```
TS topics.service.ts  TS environment.ts  TS environment.prod.ts
src > environments > TS environment.ts > environment > dbUrl
1  // This file can be replaced during build by using the `fileReplacements` array.
2  // `ng build --prod` replaces `environment.ts` with `environment.prod.ts`.
3  // The list of file replacements can be found in `angular.json`.
4
5  export const environment = {
6    production: false,
7    dbUrl: "http://localhost:8080/core/topics/all"
8  };
9
10 /*
11  * For easier debugging in development mode, you can import the following file
12  * to ignore zone related error stack frames such as `zone.run`, `zoneDelegate.invokeTask`.
13  *
14  * This import should be commented out in production mode because it will have a negative impact
15  * on performance if an error is thrown.
16  */
17 // import 'zone.js/dist/zone-error'; // Included with Angular CLI.
18
```

```
TS topics.service.ts X  TS environment.ts  TS environment.prod.ts
src > app > TS topics.service.ts > TopicsService > getTopics
1  import { Injectable } from '@angular/core';
2  import { HttpClient } from '@angular/common/http';
3  import { environment } from '../environments/environment';
4
5  import { Observable, of } from 'rxjs';
6  import { catchError, map, tap } from 'rxjs/operators';
7
8  @Injectable({
9    providedIn: 'root'
10 })
11 export class TopicsService {
12
13   constructor(private http: HttpClient){ }
14
15
16   getTopics () {
17     return this.http.get(environment.dbUrl);
18   }
19
```


3. Consuming the Search API

For consuming the search API, we almost need to follow the same approach.

- Storing and transmitting the text that is written in the search section of the front end and sending it to the Search API.
- For that, Write methods in the Menu-component.ts file of the code, which will hold the logic of the how to send the text to API.
- The next step would be to create a separate component, which will hold the logic of how to display the search results.

This task could not be completed in this sprint because I did not have access to search service. The elastic search when I tried on my system is not running because of some system limitations that I have on my laptop.



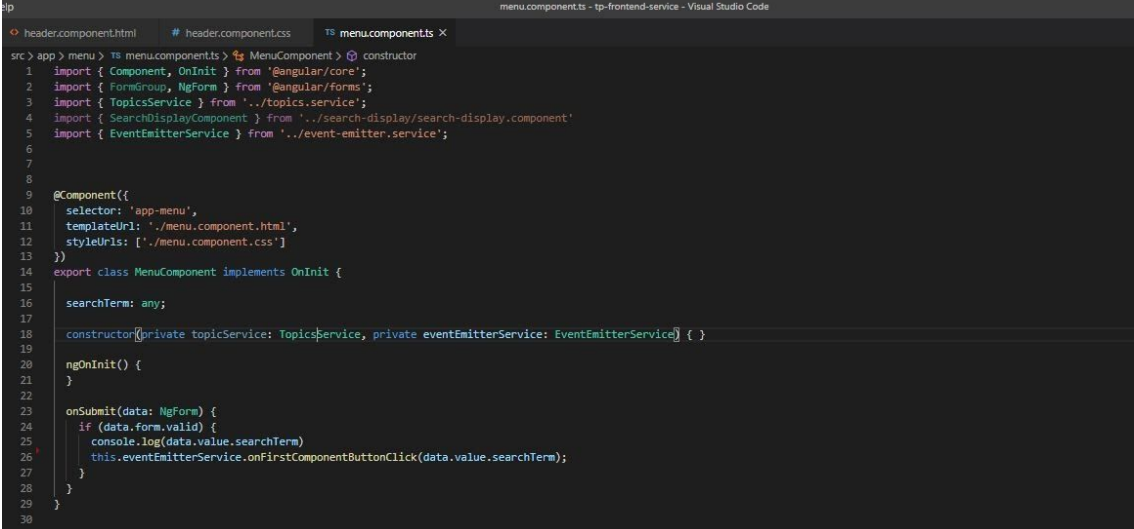
STONE-4

1. Consuming the Search API

For consuming the search API, we need to follow the approach.

- Storing the text that is written in the search section of the front end and sending it to the Search API.

The search field in the webpage is defined under the menu component. So, the variable to store and send the input to topic service is called “searchTerm”.



```
src > app > menu > menu.component.ts > MenuComponent > constructor
1 import { Component, OnInit } from '@angular/core';
2 import { FormGroup, NgForm } from '@angular/forms';
3 import { TopicsService } from '../topics.service';
4 import { SearchDisplayComponent } from '../search-display/search-display.component';
5 import { EventEmitterService } from '../event-emitter.service';
6
7
8
9 @Component({
10   selector: 'app-menu',
11   templateUrl: './menu.component.html',
12   styleUrls: ['./menu.component.css']
13 })
14 export class MenuComponent implements OnInit {
15
16   searchTerm: any;
17
18   constructor(private topicService: TopicsService, private eventEmitterService: EventEmitterService) {}
19
20   ngOnInit() {
21   }
22
23   onSubmit(data: NgForm) {
24     if (data.form.valid) {
25       console.log(data.value.searchTerm);
26       this.eventEmitterService.onFirstComponentButtonClick(data.value.searchTerm);
27     }
28   }
29 }
30
```

In the menu.component.ts you can see the method (onSubmit) that is assigning the input to searchTerm and sending it to TopicsService, where the setSearchTerm and searchTopics (connected with the search API) method is defined.

- The next step is to create a separate component, which will hold the logic of how to display the search results.

The component to display all the search results is called `search-display.component.ts` and it displays all the results that it receives from the search API, in the same form as it displays the results in the available topics tab. The component will appear after clicking the search button.

```

1  import { Component, OnInit } from '@angular/core';
2  import { TopicsService } from '../topics.service';
3  import { EventEmitterService } from '../event-emitter.service';
4
5  @Component({
6    selector: 'app-search-display',
7    templateUrl: './search-display.component.html',
8    styleUrls: ['./search-display.component.css']
9  })
10 export class SearchDisplayComponent implements OnInit {
11   topics: any = [];
12   showContent: boolean = false;
13   togglePanel: boolean[];
14   constructor(private topicService: TopicsService, private eventEmitterService: EventEmitterService) {
15     this.togglePanel = []
16     console.log(this.togglePanel)
17   }
18
19   ngOnInit() {
20     this.setToggle()
21     this.togglePanel = []
22
23     if (this.eventEmitterService.subsVar === undefined) {
24       this.eventEmitterService.subsVar = this.eventEmitterService
25       invokeFirstComponentFunction.subscribe((name:string) => {
26         this.firstFunction(name);
27       });
28     }
29
30     firstFunction(name){
31       console.log("Request arrived");
32       console.log("name : " + name)
33       this.topicService.searchTopics(name).subscribe((x) => {
34         this.topics = x
35         this.setToggle()
36         // console.log(x)
37       });
38       console.log(this.topics)
39     }
40     setTopics(T: any){

```

(As the time for this sprint was just one week, so that's why the tasks for this milestone is just one.)



MILESTONE-5

1. Dockerizing the front-end:

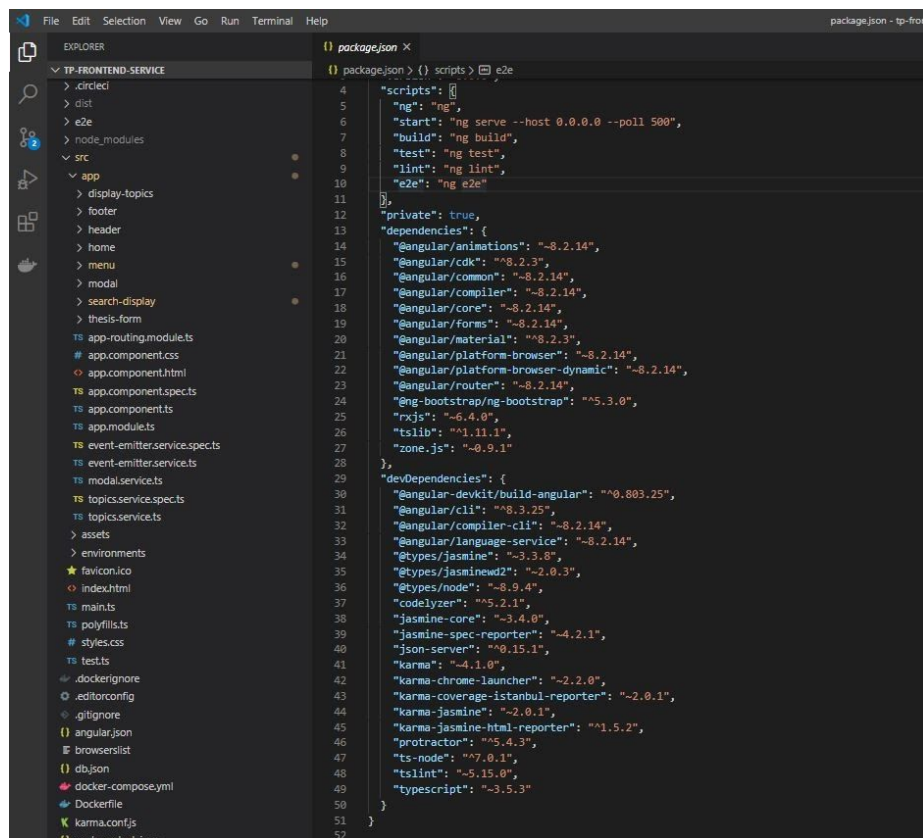
Docker is an open source containerization technology, docker allows you to package an application with its environment and all its dependencies into a container. Usually, a container consists of an application running in a stripped-to-basics version of a Linux operating system. A docker image is the blueprint for a container, a container is a running instance of an image.

To know more about docker you can read more, [here](#).

For dockerizing a node.js app, the first step is to have a working Docker installation on your system. Guide to install docker on your system can be found [here](#).

In our application we defined a working directory called “package.json”, which will contain all the dependencies of your application.

With the new package.json file, run npm install. This will generate a package-lock.json file which will be copied to Docker image.



The screenshot shows a Visual Studio Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure for 'TP-FRONTEND-SERVICE' with folders like 'dist', 'e2e', 'node_modules', 'src', and 'app'. The code editor displays the 'package.json' file with the following content:

```
1 {
2   "name": "tp-front",
3   "version": "0.0.0",
4   "scripts": {
5     "ng": "ng",
6     "start": "ng serve --host 0.0.0.0 --poll 500",
7     "build": "ng build",
8     "test": "ng test",
9     "lint": "ng lint",
10    "e2e": "ng e2e"
11  },
12  "private": true,
13  "dependencies": {
14    "@angular/animations": "~8.2.14",
15    "@angular/cdk": "~8.2.3",
16    "@angular/common": "~8.2.14",
17    "@angular/compiler": "~8.2.14",
18    "@angular/core": "~8.2.14",
19    "@angular/forms": "~8.2.14",
20    "@angular/material": "~8.2.3",
21    "@angular/platform-browser": "~8.2.14",
22    "@angular/platform-browser-dynamic": "~8.2.14",
23    "@angular/router": "~8.2.14",
24    "@ng-bootstrap/ng-bootstrap": "^5.3.0",
25    "rxjs": "~6.4.0",
26    "tslib": "^1.11.1",
27    "zone.js": "~0.9.1"
28  },
29  "devDependencies": {
30    "@angular-devkit/build-angular": "^0.803.25",
31    "@angular/cli": "^8.3.25",
32    "@angular/compiler-cli": "~8.2.14",
33    "@angular/language-service": "~8.2.14",
34    "@types/jasmine": "~3.3.8",
35    "@types/jasminewd2": "~2.0.3",
36    "@types/node": "~8.9.4",
37    "codelyzer": "~5.2.1",
38    "jasmine-core": "~3.4.0",
39    "jasmine-spec-reporter": "~4.2.1",
40    "json-server": "~0.15.1",
41    "karma": "~4.1.0",
42    "karma-chrome-launcher": "~2.2.0",
43    "karma-coverage-istanbul-reporter": "~2.0.1",
44    "karma-jasmine": "~2.0.1",
45    "karma-jasmine-html-reporter": "^1.5.2",
46    "protractor": "^5.4.3",
47    "ts-node": "^7.0.1",
48    "tslint": "~5.15.0",
49    "typescript": "~3.5.3"
50  }
51 }
```

Create three empty files, Dockerfile, .dockerignore and docker-compose.yml .

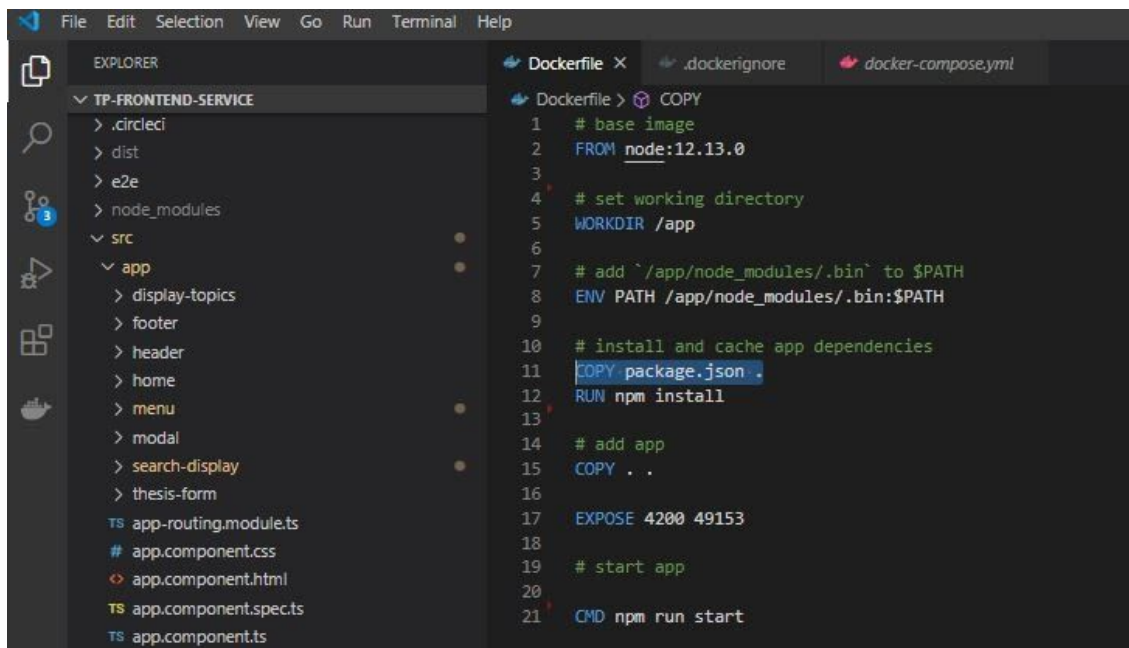
Dockerfile should have from what image we want to build, (FROM node: version), then a working directory to hold application code inside the image (WORKDIR use working directory of application).

Then rather than copying the whole working directory, we only copied the package.json (COPY package.json .). (RUN npm install) command, helps provide faster, reliable reproducible builds.

To bundle up the source code inside docker image the command (COPY . .) is used. Our app binds to the port 4200, so using (EXPOSE 4200 49153) maps it by docker daemon.

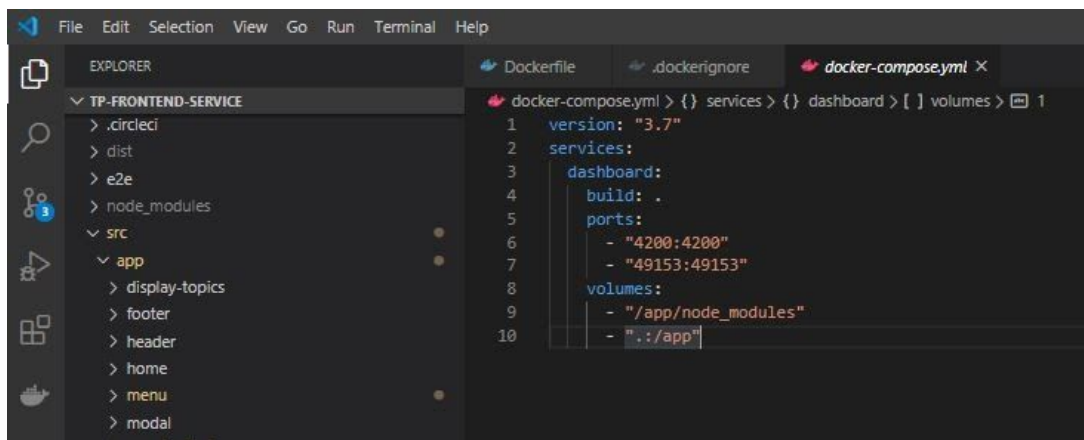
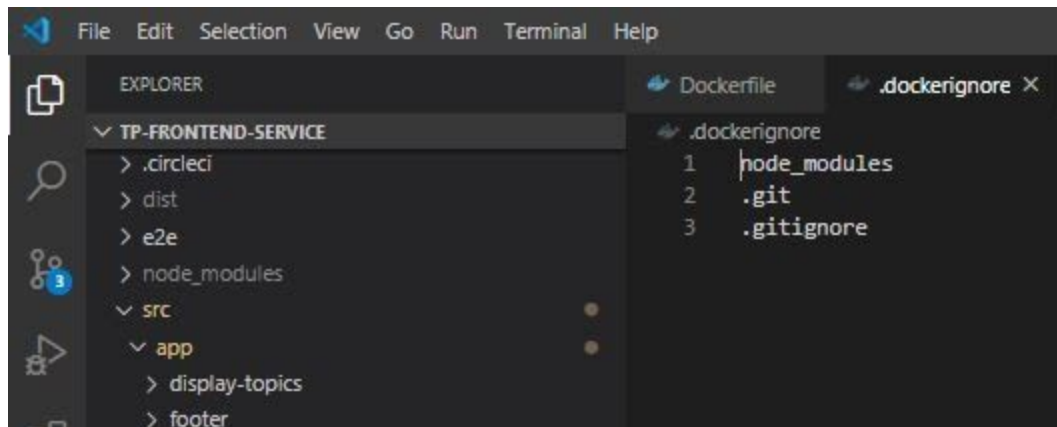
Last but not least, the command to run the app is CMD.

The docker file and .dockerignore looks like this.



The screenshot shows the Visual Studio Code interface with the Explorer, Dockerfile, and .dockerignore files open. The Explorer shows the project structure for 'TP-FRONTEND-SERVICE', including directories like .circleci, dist, e2e, node_modules, and src, and files like app-routing.module.ts, app.component.css, app.component.html, app.component.spec.ts, and app.component.ts. The Dockerfile and .dockerignore files are shown in the right pane.

```
File Edit Selection View Go Run Terminal Help
EXPLORER
TP-FRONTEND-SERVICE
  > .circleci
  > dist
  > e2e
  > node_modules
  > src
    > app
      > display-topics
      > footer
      > header
      > home
      > menu
      > modal
      > search-display
      > thesis-form
    TS app-routing.module.ts
    # app.component.css
    < app.component.html
    TS app.component.spec.ts
    TS app.component.ts
Dockerfile X .dockerignore docker-compose.yml
Dockerfile > COPY
1 # base image
2 FROM node:12.13.0
3
4 # set working directory
5 WORKDIR /app
6
7 # add `/app/node_modules/.bin` to $PATH
8 ENV PATH /app/node_modules/.bin:$PATH
9
10 # install and cache app dependencies
11 COPY package.json .
12 RUN npm install
13
14 # add app
15 COPY . .
16
17 EXPOSE 4200 49153
18
19 # start app
20
21 CMD npm run start
```



At the end you can use the command “docker-compose up --build” to build the application and produce a docker image. (make sure your docker application is already running on your system)

You can read more about writing effective docker file, [here](#).

2. Continuous integration of the front-end:

Continuous Integration or commonly known as CI is a software development practice of integrating code continuously, it has been commonly used in agile development process, where more than one developer is working on the software. It verifies the new code that you just wrote broke or not the code that was already working. Automated tests and other tasks (defined in the build job) are executed when integrating the code.

You can know more about CI, [here](#).

There are a lot of build systems available that you can work with for your project. We chose to work with circle CI, because the technologies that we were using it integrated seamlessly with all of them.

Here is a guideline of how we have configured our project with Circle CI. [Source 1](#), [source 2](#). After the integration was done.

The main file under which all the build jobs are defined can be found in the folder of our project in each repository as .circleci > config.yml. At every new code push the build job runs and if it's successful it means nothing in your code broke.

```
.circleci > ! config.yml > ## version
1 |version: 2
2 |executorType: machine
3 |jobs:
4 |  # The build job
5 |  build:
6 |    working_directory: ~/app
7 |    docker:
8 |      - image: circleci/node:13.8-browsers
9 |    steps:
10 |      # Checkout the code from the branch into the working_directory
11 |      - checkout
12 |      # Restore local dependencies from cache
13 |      - restore_cache:
14 |        keys:
15 |          - v1-dependencies-{{ checksum "package-lock.json" }}
16 |          - v1-dependencies-
17 |      # Install project dependencies
18 |      - run:
19 |        name: Install node@v12.16.2
20 |        command: |
21 |          export NVM_DIR="/opt/circleci/.nvm"
22 |          [ -s "$NVM_DIR/nvm.sh" ] && . "$NVM_DIR/nvm.sh"
23 |          nvm install v12.16.2 && nvm use v12.16.2 && nvm alias default v12.16.2
24 |          node -v
25 |          npm install @angular/cli@9.1.3
26 |
27 |      # Cache local dependencies if they don't exist
28 |      - save_cache:
29 |        key: v1-dependencies-{{ checksum "package-lock.json" }}
30 |        paths:
31 |          - node_modules
```

3. Setting up a Login page:

A new component of Login was created (login.component) through which an admin user can login with the credentials of Username and password, which will lead the logged user to a homepage through which the user can either create a new topic or to a upload/delete page.


The fields of user name and password are linked with a backend API which send the field to backend and then returns a token of login and Research group ID, as the usernames are connected with their particular research groups.

```
src > app > login > ts login.component.ts > LoginComponent > @ ngOnInit
1 import { Component, OnInit } from '@angular/core';
2 import { FormGroup, NgForm } from '@angular/forms';
3 import { Router } from '@angular/router';
4 import { AuthService } from '../auth.service';
5 import { CookieService } from 'ngx-cookie-service';
6
7 @Component({
8   selector: 'app-login',
9   templateUrl: './login.component.html',
10  styleUrls: ['./login.component.css']
11 })
12 export class LoginComponent implements OnInit {
13
14   username:String;
15   password:String;
16
17   constructor(private cookieService: CookieService, private router: Router, private auth: AuthService) { }
18
19   ngOnInit() {
20     if(this.cookieService.get("tp_user")){
21       // console.log("user present")
22       this.router.navigate(['/', 'supervisor-dashboard']).then(nav => {
23         console.log(nav); // true if navigation is successful
24       }, err => {
25         console.log(err) // when there's an error
26       });
27     }
28     else {
29       // console.log("user not present")
30     }
31   }
32
33   login(data: NgForm){
34     if (data.form.valid) {
35       this.auth.login(data.value.username, data.value.password)
36     }
37   }
38 }
39
40 }
41
```

The login page looks like;



Sign in

 Username

 password

Login

[Navigate to home](#)

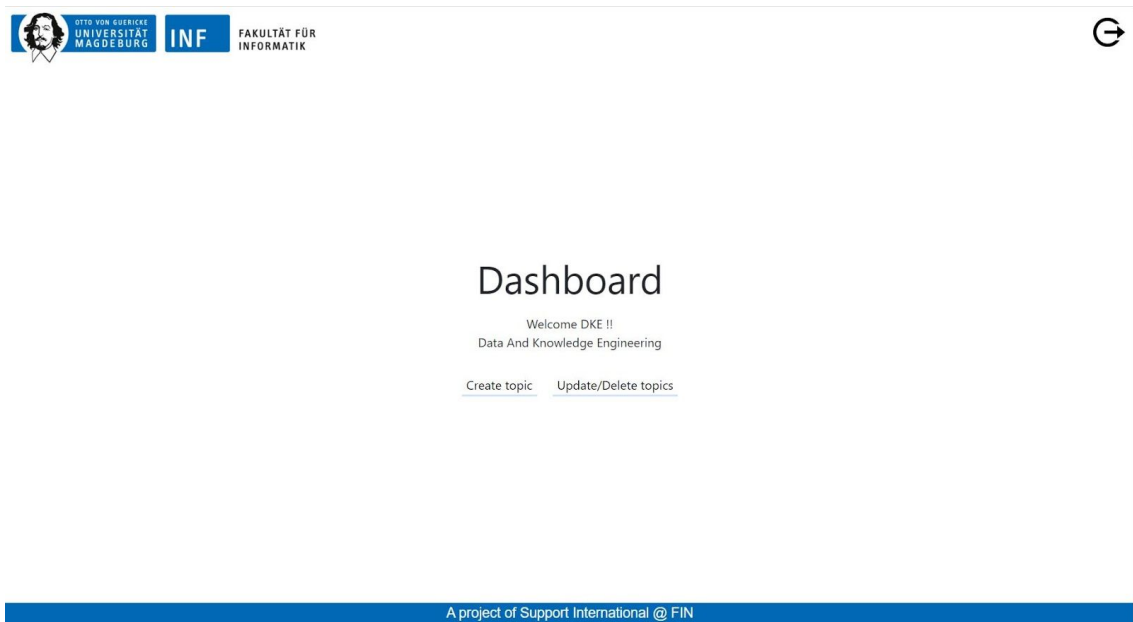


MILESTONE-6

1. Home page/Supervisor Dashboard:

We have created one of other homepage for an admin when he or she logs in. it redirects to the two features/functionality pages that the admin can do, 1) Create a topic 2) Update or delete a topic. The homepage for supervisors/admin is called supervisor-dashboard in the code and it looks like this.

```
src > app > supervisor-dashboard > supervisor-dashboard.component.html > div.aligner > div.home-container > div.profile-info
1 <div class="aligner">
2   <div class="home-container">
3     <div class="title">
4       Dashboard
5     </div>
6     <div class = "user-name profile-info">
7       Welcome {{user}} !!
8     </div>
9     <div class="profile-info">
10      {{researchGroup}}
11    </div>
12    <div class="options">
13      <button type="button" class="all-topics-button" [routerLink]="['/create-topic/NULL']">Create topic</button>
14      <button type="button" class="all-topics-button" [routerLink]="['/supervisor-display-topics']">Update/Delete topics</button>
15    </div>
16  </div>
17 </div>
```



2. Update and Delete page:


Admin roles include creating a topic and updating or deleting existing topics. Creating a topic page has already been developed in previous phases. Now we worked on updating and deleting functionalities. For this a page similar to display all topics has been created. Which is connected to a backend API, which returns the topic that the articular research group has uploaded previously and shows an option of updating or deleting it. This component is called supervisor-display-topics in the code. The display cards of this components have two additional button, one is EDIT button and one is Delete button.

```
# supervisor-display-topics.component.css
src > app > supervisor-display-topics > supervisor-display-topics.component.html > div.topics-container > div.card > div.card-body
1 <div class="options">
2   <button type="button" class="all-topics-button" [routerLink]="['/supervisor-dashboard']">Go to dashboard</button>
3   <button type="button" class="all-topics-button" [routerLink]="['/create-topic/NULL']">Create topic</button>
4 </div>
5 <div class="topics-container">
6   <div class="card" *ngFor="let topic of topics; let i=index">
7     <h4 class="card-header">
8       {{topic.title}}
9     </h4>
10    <div class="card-body">
11      <div class="card-text mb-2 text-muted"><p class="label">Supervisor : </p>{{topic.supervisor}} </div>
12      <!-- [div class="card-text"><p class="label">Research Group : </p>{{topic.researchGroup}}</div> -->
13      <div class="card-text"><p class="label">Start Date : </p>{{topic.startDate| date}}</div>
14      <div class="card-text">{{topic.description}}</div>
15      <div id="collapseable" [ngbCollapse]="!togglePanel[i]">
16        <div class="card-text"><p class="label">Must have : </p>{{topic.mustHave}} </div>
17        <div class="card-text"><p class="label">Nice to have : </p>{{topic.niceHave}} </div>
18        <div class="card-text text-muted">{{topic.contactInfo}} </div>
19      </div>
20
21      <p class="action-buttons">
22        <button type="button" class="btn btn-outline-primary" (click)=toggle(i) [attr.aria-expanded]="!togglePanel[i]" aria-controls="collapseable">
23          <div *ngIf="togglePanel[i]">Show less <i class="fas fa-chevron-up"></i></div>
24          <div *ngIf="!togglePanel[i]">Show more <i class="fas fa-chevron-down"></i></div>
25        </button>
26        <button class="btn btn-outline-primary edit-button" (click)=editTopic(i)>Edit <i class="fas fa-pencil-alt"></i></button>
27        <button class="btn btn-outline-danger delete-button" (click)=confirmDelete(i)>Delete <i class="fas fa-times"></i></button>
28      </p>
29    </div>
30  </div>
31  <app-modal id="topic-del-modal">
32    <h1 class="text-center" mat-dialog-title>Delete Topic</h1>
33    <div mat-dialog-content>
34      <p class="text-center">Are you sure you want to delete this topic?</p>
35    </div>
36    <div class="right-align" mat-dialog-actions>
37      <button mat-button (click)=deleteTopic() cdKFocusInitial id="yes-button">Yes</button>
38      <button mat-button (click)=closeModal('topic-del-modal');cdKFocusInitial id="no-button">No</button>
39    </div>
40  </app-modal>
41 </div>
```


First let's see the delete button, delete button is connected to a modal, which once pressed the user if they really want to delete the selected topic. If the user clicks yes it sends this information to the backend API of delete and the topic gets deleted from the backend.

Second is Edit once this button is pressed it leads back to the thesis form component but this time the form is prefilled with the values that are returned from a backend API of update. The admin can then fill the required information or change it and save the topic again which then gets updated in the backend.

```
src > app > supervisor-display-topics > ts supervisor-display-topics.component.ts > delete topic > subscribe() callback
1 import { Component, OnInit } from '@angular/core';
2 import { TopicsService } from '../topics.service';
3 import { CookieService } from 'ngx-cookie-service';
4 import { Router } from '@angular/router';
5 import { ModalComponent } from '../modal/modal.component';
6 import { ModalService } from '../modal.service';
7
8 @Component({
9   selector: 'app-supervisor-display-topics',
10   templateUrl: './supervisor-display-topics.component.html',
11   styleUrls: ['./supervisor-display-topics.component.css']
12 })
13 export class SupervisorDisplayTopicsComponent implements OnInit {
14   topics: any = [];
15   showContent: boolean = false;
16   togglePanel: boolean[];
17   private toDeleteId: string;
18
19   constructor(private router: Router, private cookieService: CookieService, private topicService: TopicsService, private modalService: ModalService) {
20     this.togglePanel = []
21     // console.log(this.togglePanel)
22   }
23
24   ngOnInit() {
25
26     if(this.cookieService.get("tp_user")){
27       // console.log("user present")
28     }
29     else {
30       this.router.navigate(['/', 'login']).then(nav => {
31         console.log(nav); // true if navigation is successful
32       }, err => {
33         console.log(err) // when there's an error
34       });
35       // console.log("user not present")
36     }
37
38     this.getTopics()
39     this.togglePanel = []
40   }
41
42   getTopics() {
43     this.topicService.getResearchGroupTopics().subscribe(x => {
44       this.topics = x
45       this.setToggle()
46       console.log(this.topics)
47     });
48   }
49
50   // setting toggle of each topic to false.. ie. every card in closed
51   setToggle() {
52     for(var i=0; i<this.topics.length; i++){
53       this.togglePanel[i] = false;
54     }
55   }
56
57   toggler(i){
58     if(this.togglePanel[i] == true)
59       this.togglePanel[i] = false;
60   }
61 }
```



INF
FAKULTÄT FÜR
INFORMATIK



[Go to dashboard](#) [Create topic](#)

Edited Demo Topic

Supervisor : Dr. David Akworko

Start Date : May 31, 2020

The emancipation of three nodes distributed network graph data

[Show more](#) [Edit](#) [Delete](#)

Frontend Developer

Supervisor : hello




Start Date : May 31, 2020

abc

[Show more](#) [Edit](#) [Delete](#)

last topic

Pre filled form for update function.

OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURGINFFAKULTÄT FÜR
INFORMATIK

[Go to dashboard](#)[Update/Delete topics](#)

Title :

Research Group :

Supervisor :

Description :

Must Have :

Nice to Have :

Contact Information :

Start Date :

Enter Thesis Topic Title

Data And Knowledge Engineering

Name of the Supervisor


Please Enter Detailed description here

Must Have Skills for People Applying

Must Have Skills for People Applying

Enter the mode of contact and details

YYYY-MM-DD



Create Topic

A project of Support International @ FIN



Summary

1. Software Usage:

The software at its current state can be used by two set of people:

- 1) Students, to search for the recent topics offered by our faculty.
- 2) Faculty, to upload available thesis topics on the portal, update a particular uploaded topic and delete it, if needed.

It will save students' time in searching for a topic as all the topics will be at one place and they can use keywords to search for topics and if those words appear in any of the topics in the database the search results will show all the topics.

2. Experiences/Learnings:

The major things that I learned during the course of this project was the importance of teamwork, other than the obvious technical learning that I did. At the start of this project we started off with different people in the team, two of them left due to some miscommunication that occurred in the team. So it showed me how important it is to communicate well when you are working in a team. After that it was a smooth sail when Madhuri joined our team.

As we were all working on different parts of the project and it was absolutely necessary for us to communicate and inform the other person about the changes that they made at their end because it affected the code and working of all of us.

I also learned to be okay with the agile working environment because what i really worked hard for in one phase can completely change in the next sprint and with that i have to make a lot of changes and be fluid with my approach towards the development process.

Other than that i learned Angular, that i have not even heard of before starting the project. Slowly but steadily i got comfortable with working with Angular, and linking every aspect of front end with backend and search.

I learned container of the project via docker and why it is important, I learned to integrate our project with a continuous build system and many times it saved us from breaking down our code and for trouble shooting.

I also learned to deploy my frontend service on the server where it is hosted.

Testimonial:

“ The development of a thesis portal under the supervision of Dr. Cluadia Krull has allowed me to use all aspects of a Software development, from an idea to a final product that serves students and faculty of FIN gives a sense of pride in its own different way. I am very pleased with the Application that we are able to produce and I hope it serves its purpose of reducing the time students spend in finding master thesis topics and give more visibility of the ongoing research work of our department”.