20. DEZEMBER 2020

THE MICROBLOGGER

REPORT OF THE COURSE PROJECT IN WEB APPLICATIONS

DENNIS NAUJOKAT
0609886
LUT University

Description

The Program is called “The MicroBlogger” and is a simple micro blogging service. Everybody can see the posts (if the author wants so) even without registration. To post something users’ needs to register first and then can login to make posting available. Each user can control his privacy configuration to show his post to: everyone, registered users, only friends or to only himself. The configuration can be changed in the profile and is by default on everyone.

A post must contain a title and a post text, but can also optional include an image. If the transmitted file is not an image or the image type is not supported the post will not be posted and the website will show the error message: "Couldn’t process Image".

The main page shows all posts which the user is allowed to see. It loads at first only the 20 newest posts but with the button “More” more post can be loaded. The website caches loaded post to minimize the traffic and the cache will be deleted ones the user leaves the website. In these view images will only be shown in a small resolution of max. 450x450 pixels. To show the image in higher resolution (max 1000x1000 pixels) the user needs to open the post by clicking on its title. And by clicking at an author his profile page will be opened including a list of only his posts. If the user is logged in the main page also shows an interface to post new posts.

The view of a single post shows additional to the post the comments under the post and enables logged in users to write new comments to the post. In the view of another user profile you can see his username, can ask him to be your friend and see his posts (if his privacy allows it). If you are already a friend of him, you also see his Email address and his friends.

To befriend someone both users’ needs to asked each other to be friends. A user can see who has asked him on his own profile site. At the current state friends cannot be removed so the user needs to be careful who he is friending.

Program Design

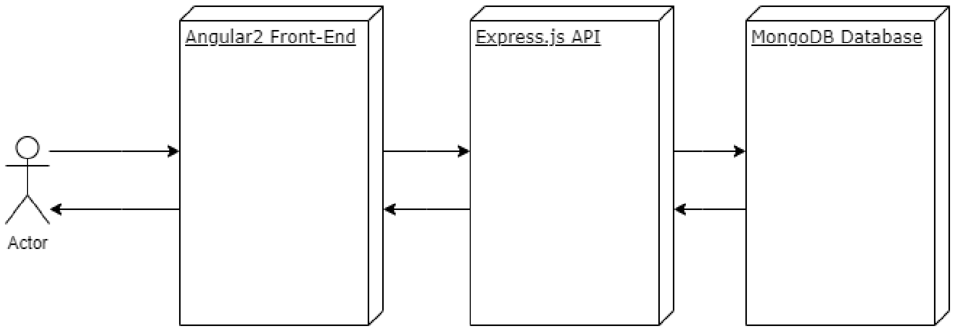


Figure 1: Layers Architecture

The Program is basically structured in 4 parts. The API, the Database, the Front-End and the API Documentation. The Front-End and API Documentation will be delivered to the user by the Webserver and will then run in the user's browser powered by JavaScript. The API is running directly on the Webserver and the Database needs to be a MongoDB but can be located everywhere as long as she is accessible via a connection string which is provided by an environment variable.

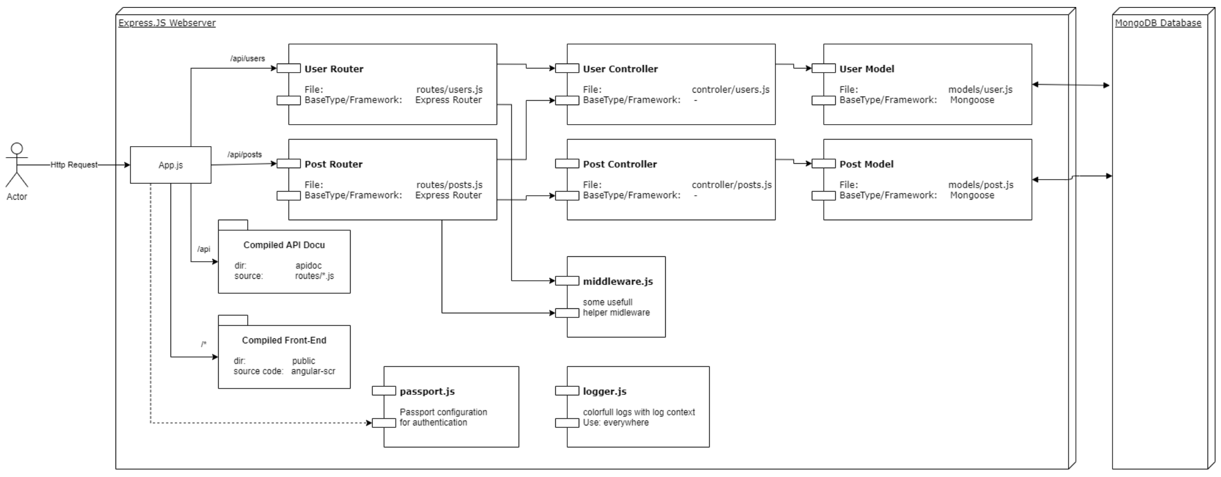
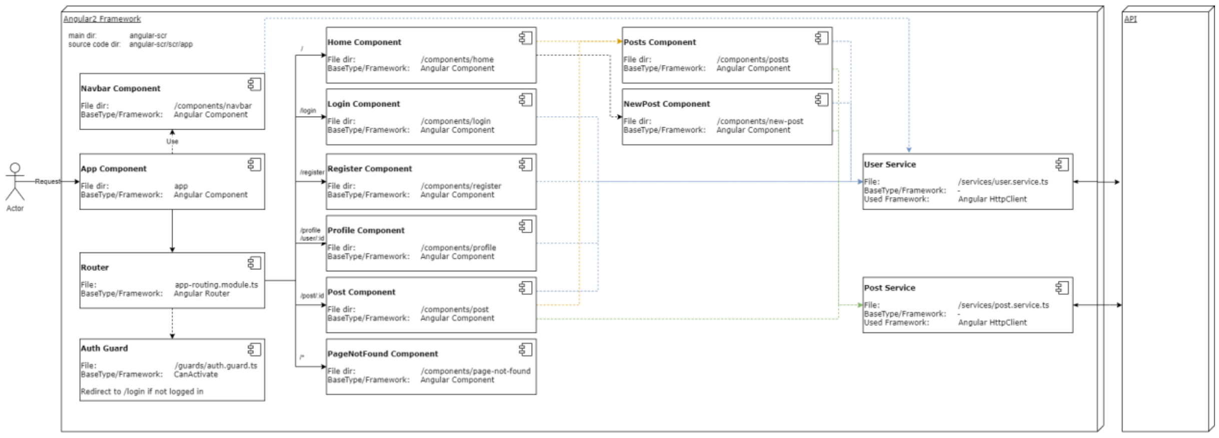


Figure 2: Webserver Structure

As Webserver is used Express.js because it is one of the most common for Node and its easy use. The communication to the Database takes place via Mongoose with is a comment Node Framework for MongoDB. While development a MongoDB Version 3.6 was used, but do to restrictions of CSC-Rathi the code was adapt to use Version 3.2. The Version can be set via an environment variable, default is 3.6. The API is split in user management and post management, where the post management is depending on the user management to enable privacy functions and to deliver parts of the user data in posts and comments to reduce the number of needed requests. The Routing and execution of preparations (like authentication, authorization, etc.) are managed by the routers and the program logic is contained in the corresponding controllers. The interactions with the Database are defined in the models.

To communicate with the API the Content-Type of the requests (except GET) needs to be "application/json" and the responses will also be in these format. The Only exceptions are "Write a Post" and the response of "Get Image from Post". "Write a Post" needs the Content-Type "multipart/form-data" because it can contain an Image beside the normal data, but it delivers also JSON as response. The other exception "Get Image from Post" doesn't deliver JSON it delivers the Image directly, because these ways it is possible to use the images like every other image on a normal webserver. Because of these it's also possible to deliver the authorization token not only as header but also as Cookie, because normal requests for images doesn't contain special headers.

The webserver delivers the folders public and apidoc static to the user, because these contains the compiled JavaScript programs of the Front-End and the API Documentation. The documentation is generated by apiDoc and is written as comment blocks in the router at the associated routes. ApiDoc was used because it is very similar to normal code documentation what's makes it easy to understand and use.



Angular2 Framework
main dir: angular-scr
source code dir: angular-scr/scr/app
Navbar Component
File dir: /components/navbar
BaseType/Framework: Angular Component
Use
Actor
Request
App Component
File dir: app
BaseType/Framework: Angular Component
Router
File: app-routing.module.ts
BaseType/Framework: Angular Router
Auth Guard
File: /guards/auth.guard.ts
BaseType/Framework: CanActivate
Redirect to /login if not logged in
/
Home Component
File dir: /components/home
BaseType/Framework: Angular Component
/login
Login Component
File dir: /components/login
BaseType/Framework: Angular Component
/register
Register Component
File dir: /components/register
BaseType/Framework: Angular Component
/profile
/user/:id
Profile Component
File dir: /components/profile
BaseType/Framework: Angular Component
/post/:id
Post Component
File dir: /components/post
BaseType/Framework: Angular Component
/*
PageNotFound Component
File dir: /components/page-not-found
BaseType/Framework: Angular Component
Posts Component
File dir: /components/posts
BaseType/Framework: Angular Component
NewPost Component
File dir: /components/new-post
BaseType/Framework: Angular Component
User Service
File: /services/user.service.ts
BaseType/Framework: -
Used Framework: Angular HttpClient
Post Service
File: /services/post.service.ts
BaseType/Framework: -
Used Framework: Angular HttpClient
API

Figure 3: Front-End Structure

For the Front-End the Framework Angular2 is used. These Framework was chosen because basic structor of Components/Views is similar to other UI Frameworks like WPF or the UI Framework of Android Studio. Each component is split in two main parts the UI design and the code behind (by Angular 4 because additional CSS-file and component test file). The code behind can be used like each other normal class (with some exceptions e.g. You normally don't create instances on your own). To display data from the code behind in the UI, you simple use an databinding which automatically update on changes or call functions of the code behind. This clean separation of UI and functional code is one of the main reasons I chose these Framework, it's makes it easy to handle without restricting your freedom.

The route management is handled by the Angular AppRoutingModule which is defined in the "app-routing.module.ts" it automatically changed the displayed components per route. The next import part are the services. They control the communication to the API and cache the data. They are configured to be handled as singletons by Angular to make the secure caching possible. The user service handles all data transfer concerning the user and also implements the login and authorization managements. That's why the post service is also depending on the user service to get the authorization information. The post service handles the data transfer concerning posts and also caches the data to minimize the data traffic. He also managed the share of data between components to minimize the traffic even further.

The main components which are directly accessible by the user via routes are the Home-, Login-, Register-, Post-, PageNotFound- and the Profile-Component. The Login- and Register-Components show each interfaces to login and register users and execute these functions via the user service. The Home-Component shows the newest posts of all users and shows logged in users an interface to post new posts. Both functions are also structured in sub components to make them also usable in other components.

The Posts-Component, which shows the new posts, for example is also used in the Profile-Component to show the posts of the user. For these the Component has an optional input variable with defines the user id if the post should be filtered. Beside the posts the Profile-Component also displays the data of the user and provide the ability to change your own privacy configuration and befriend other users. These component is used to show your own profile and also the profiles of other users, depending on these some functions can sometimes be disabled. E.g. you are not able to change the privacy of other users and there for these controls are hidden in these scenario.

The last important component is the Post-Component which shows a single post in detail, including its comments. It also provides an interface to write new comments under the post. The displayed image in these view is large version of the image while the Pages-Component only shows the small version as a preview.

Other components are the PageNotFound-Component which is shown when in invalid route is called or not existing resources were required. And The Navbar-Component contains the navigation bar, which is shown in every situation of the Front-End.

Libraries

Some of the used libraries were already explained in the text before. The following Tables show all includes Libraries and for what they are used:

Libraries API

Library	Use
cors	Allow request to other Servers
express	Webserver Framework
express-validator	Parameter validation
bcryptjs	Hashing of Passwords
jsonwebtoken	Generation of JWT
mongoose	Database ORM
passport	User authorization
passport-jwt	User authorization by JWT
multer	To temporary save uploaded Images on disk
sharp	Resize Images
cookie-parser	Load Cookie, to allow Image access with privacy

Development Library	Use
dotenv	Emulate Environment Variables
nodemon	Auto reload of Server after changes

Libraries Front-End

Library	Use
@angular/animations	Generated by Angular CLI
@angular/common	Generated by Angular CLI
@angular/compiler	Generated by Angular CLI
@angular/core	Generated by Angular CLI
@angular/forms	Generated by Angular CLI, to communicate with Forms
@angular/platform-browser	Generated by Angular CLI
@angular/platform-browser-dynamic	Generated by Angular CLI
@angular/router	Generated by Angular CLI, to enable routing
@auth0/angular-jwt	To check if JWT is expired
angular2-flash-messages	To show Status Popups to the User
rxjs	Generated by Angular CLI
tslib	Generated by Angular CLI
zone.js	Generated by Angular CLI

Development Library	Use
@angular-devkit/build-angular	Generated by Angular CLI
@angular/cli	Generated by Angular CLI
@angular/compiler-cli	Generated by Angular CLI
@types/jasmine	Generated by Angular CLI
@types/node	Generated by Angular CLI
codelyzer	Generated by Angular CLI
jasmine-core	Generated by Angular CLI
jasmine-spec-reporter	Generated by Angular CLI
karma	Generated by Angular CLI
karma-chrome-launcher	Generated by Angular CLI
karma-coverage	Generated by Angular CLI
karma-jasmine	Generated by Angular CLI
karma-jasmine-html-reporter	Generated by Angular CLI
protractor	Generated by Angular CLI
ts-node	Generated by Angular CLI
tslint	Generated by Angular CLI
typescript	Generated by Angular CLI, because Angular used TypeScript instead of direct JavaScript

How to Run

The application needs an installed version of Node and needs to be provided with a link to a MongoDB Database. To run the program following environment variables needs to be set:

Enviroment Variable	Desciption
PORT	Port on which the Webserver should run
DATABASE_URL	Connection String to the Database
TOKEN_SECRET	Secret for encrypting JWT, should be an random string
DEBUG	true/false: If true API shows some additional debug information on requests
MongoDbVersion	Min. Version of the MongoDB, allowed Values: 3.2 and 3.6, default is 3.6

To run the application run:

```
npm install
npm run start
```

Selected Requirements

The is aiming for 66pts in total. The selected requirements with the number of points are explained in the table below and for some of them also how they were interpreted while developing. For further information about how they are achieved please watch the explanation video.

Requirement	Points	Adaption
Using React.js front-end	**15**	**Using Angular2 front-end**
Using redux and an advanced React architecture	5	Using of global services as data management
Running your application in a Docker container (or having the application Docker compatible)	**5**	
Running your application in Rahti	5	
Having more than one container (such as a separate database server or a load balancer)	3	Additional Database container
Using a database, such as Mongo, Redis, or any SQL-compatible	**5**	**Using MongoDB**
Use an ORM and models in backend, such as the Mongoose (MongoDB) or Sequelize (SQL)	3	Using Mongoose
Supporting pictures storage and display	**5**	
Resize and validate the images after download - create a set of thumbnails on loading to be shown in different places	3	
Using XHR (also know as AJAX/AJAJ) for data transfer and frontend syncronization	**4**	**Using Angular2 Http-Client**
User registration, authentication, and password storage	**5**	**User management + Authentication via JWT**
Add access permissions: blog is open to everyone, to registered users, to specific users, to owner only	2	
Provide data from your application through an API and document it with a suitable tool, e.g. with apiDoc or API Blueprint.	**3**	
Using a cache (frontend)	**3**	
	66 **pts**	