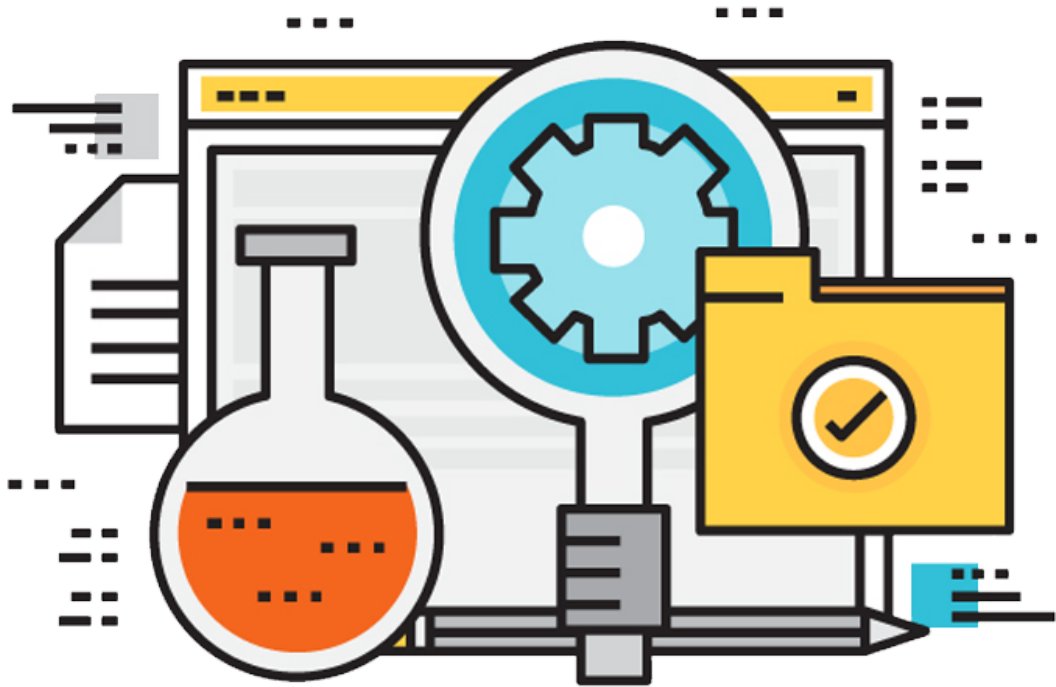


Project Report R&D Jakob Udovič



Jakob Tadej Udovič - S1049877
Research and development project
Date: June 2020



Radboud Universiteit

Project Report R&D Jakob Udovič	1
1. Introduction	3
2. Description	5
2.1 Properties	5
2.2 Product justification	6
2.3 Specifications	7
3. Design	8
3.1 Global design	8
Frontend	8
Backend	10
3.2 Detailed design	12
3.3 Design justification	16
4. Evaluation	17
5. Other work	19

1. Introduction

This was more of a research than a development project for me. To keep things simple, I will divide my project in 3 parts:

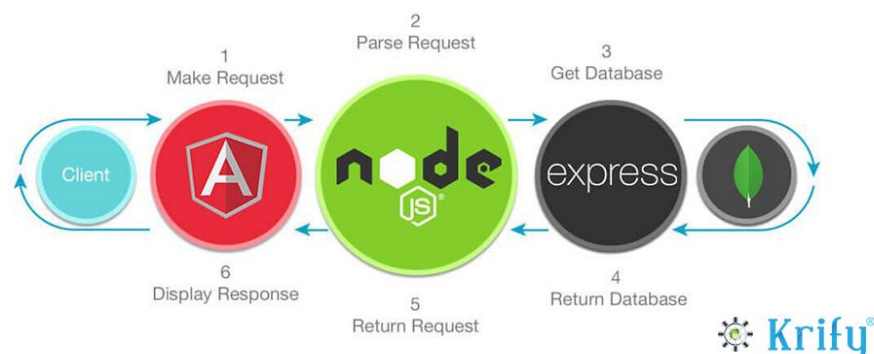
1. Setting up an LED strip, controlling it via NodeMCU and making it responsive to music/output of my laptop.
2. Exploring and using my newest addition, Raspberry Pi 4.
3. Setting up a personal website, using Angular framework, learning the MEAN stack.

I did not know how to do any of the things above. My main focus majority of the time were LEDs but I had to abandon the idea soon. It was way too much troubleshooting with MCU and I got worried there won't be an actual product at the end. Node MCU I was using is known to have a lot of issues and I just could not afford to wait for the new one to arrive (mine did not want to flash anymore).

I still set up my LEDs, gained some knowledge about circuits and connected them with my RPI 4. I also learned some stuff about writing python scripts for the LED strip, which is run on the RPI.

This brings us to our second point. I started off by learning how to SSH into my RPI console. I then gave it an ethernet cable and made it into a Pi-hole server (<https://pi-hole.net/>). It blocks all ads, on my laptop as well as my phone at the same time, so the browser extensions like uBlock and Adblock are almost redundant. Running a server, scripts for LEDs etc started to heat up my RPI a little. Based on its temperature, I implemented a script controlling the fan spinning and cooling it down (I just did not like the constant noise coming from it).

Why you Should go for Mean stack Development for your Project



Finally, I started to explore my third and last point, Angular, so I would actually have something to turn in. I knew some stuff about web development, since I was working on a project for my home university, using Vue.js. However, I wanted to try something else also. Angular “won” against React, my second option. Because I only worked on the frontend project in my past, I wanted to get to know the full stack development, so

called “MEAN stack” (MongoDB, express.js, Angular and Node.js). This is the part I worked on mostly for the rest of the time I had left.

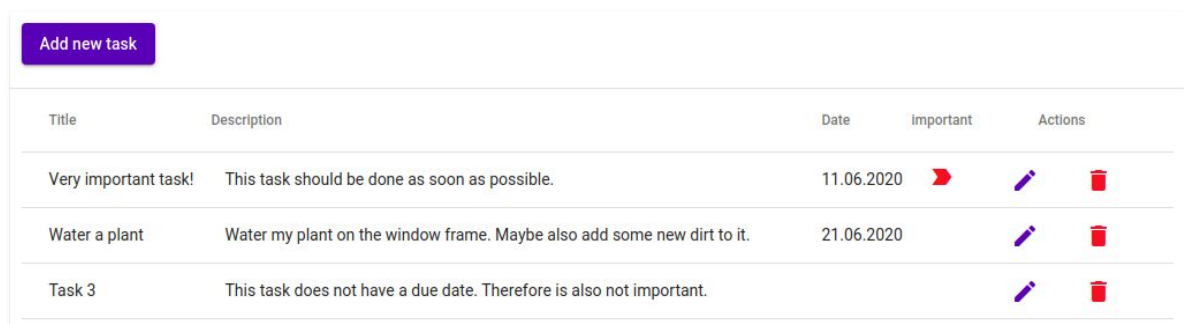
The MEAN stack will be covered by making a fairly simple application that displays the tasks of a certain user (in this case, only me). It receives the data from MongoDB, where I created a free account. After returning the request, we visualize tasks for the user to interact with. He can edit a certain task or delete it. He can also add a completely new one, if he chooses so.

My final wish is to merge not only points 1 and 2, but also 2 and 3. This means hosting my website on my RPI server, obtaining the static IP and making it available to the public.

2. Description

2.1 Properties

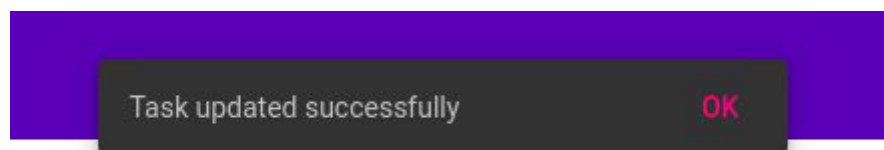
The core of my project would be a fairly simple Angular website, served locally (for now) on port 4200. The website has some functionalities which I will continue to add in the future, one by one. The main functionality, which is also the most complicated one so far is a task manager. There, you can inspect your TODO list, add, remove or edit your tasks. Each one of them is equipped with the title, description, due date and importance level.



Add new task				
Title	Description	Date	Important	Actions
Very important task!	This task should be done as soon as possible.	11.06.2020	🔴	✎ 🗑
Water a plant	Water my plant on the window frame. Maybe also add some new dirt to it.	21.06.2020		✎ 🗑
Task 3	This task does not have a due date. Therefore is also not important.			✎ 🗑

All fields are optional, except for the title of the task. After you add/save certain tasks, you can always modify all of the fields at any time. If you decide to delete it, however, it will be removed forever.

After updating/adding or deleting a task, you, as a user, also receive a beautiful notification about successful update.



The website also contains a beautiful landing homepage and an “about” section, where some general information about the author (me) can be found¹.

¹ This later on changes to be the place for diagrams, rather than my personal information.

2.2 Product justification

Pretty much everyone nowadays has their own personal website. On many platforms around the web you are frequently asked to enter your information when creating an account. One of them is your website. It is a great tool of expressing your character, achievements, fields of interest, and so much more.

Besides providing general information I wanted my website to serve me as a task scheduler too. All of the todo lists currently available online always seemed too complicated and “overdone”. So far, I have been using google keep as my tool to organize work. For more complicated projects I have been using Trello² instead. They have served me well but, I always wanted something more minimalistic, yet effective. It was a great opportunity for me to explore the full (MEAN) stack development – see section 1.



² Trello: <https://trello.com>

2.3 Specifications

Use case 1: Viewing all tasks

- User can view all tasks by navigating to the “/tasks” section. If there are none, the website will not show any.

Use case 2: Adding a task

- User can click on a button to add a task. He will get redirected to the “/create” route, where he will be asked to enter information about a task. After doing so, he can either confirm to add a new task or go back to the previous screen, viewing all tasks.

Use case 3: Edit a task

- After creating a new task, it can also be modified. To modify its information, user can click on the blue pen button near the task, which will redirect him to the “/edit” route. He will be able to edit the task there and after being satisfied with the result, he can either confirm the submission or navigate to the previous screen, abolishing the edit.

Use case 4: Delete a task

- Every task can be deleted by pressing the red bin icon next to the task, on the screen displaying all tasks.

Use case 5: View general info about the author

User can navigate to the “/about” route where he can see the information about an author.

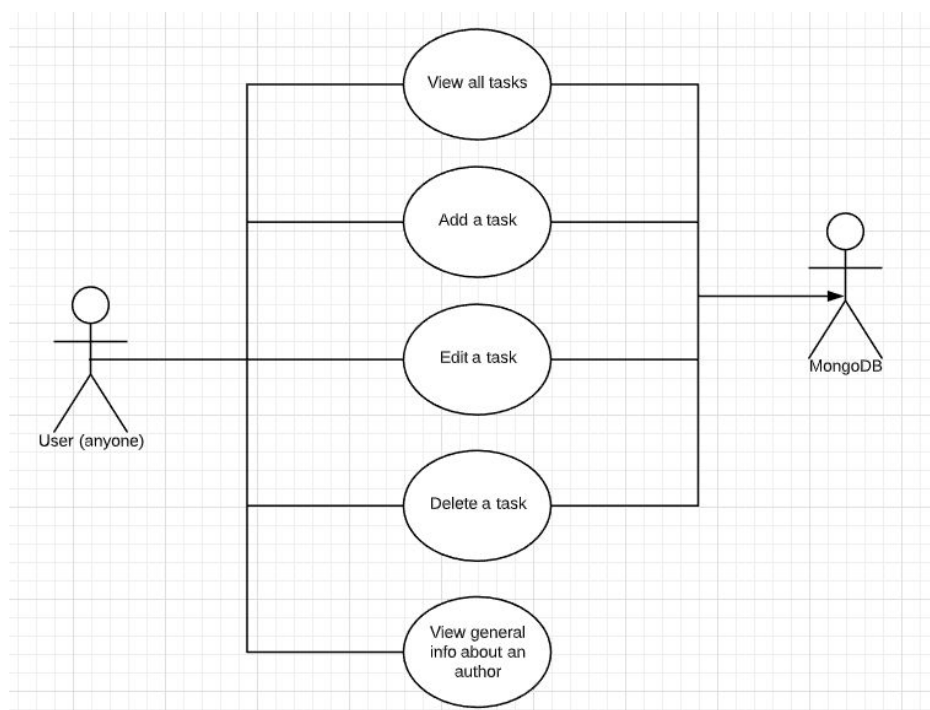


Image: Use case diagram

3. Design

3.1 Global design

In this part I will focus mostly on a sequence diagram of the most complicated functionality - editing a task - as an example how components connect with each other. It nicely shows the interconnections between the majority of the tasks. You can see messages and function names on top of arrows to easier follow the flow.

But before we inspect the sequence diagram, a few words on the frontend and the backend of the project follow.

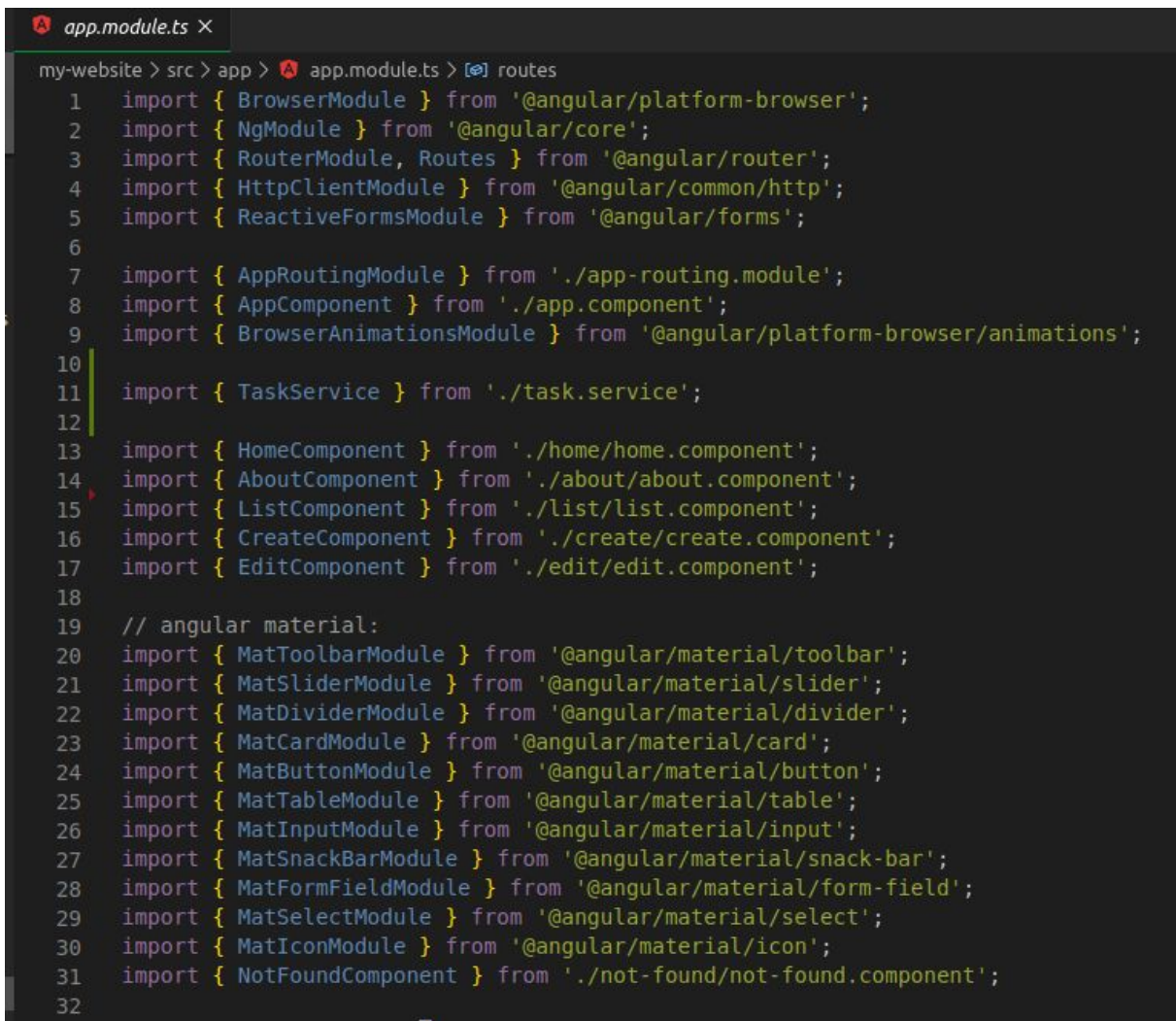
Frontend

Angular is made of many components, each of which has its own html, scss and typescript file. Html and scss determine how the component looks on the website, whereas typescript file serves as a logic behind it. It tells the component how to react and communicates with the backend system.

You can add as many components as you want but they all share the same root component, `app.component.html`. Besides that, they also share the same global scss file - `app.component.scss`.

To keep things more organized, I put all functions related to tasks in one global `task.service.ts` file and as the name suggests, it is now a service. We can inject or import it whenever we need its functionalities. There, we use `HttpClient` to access all requests from the backend.

One should keep in mind that when importing some external classes/modules; `RouterModule`, `HttpClientModule`,.. other components; `HomeComponent`, `AboutComponent`, `EditComponent`... or Angular material components; `MatToolbarModule`, `MatIconModule`, `MatTableModule`, just to name a few, he should do it all in root typescript file `app.module.ts` as well as in the desired component, where he uses the certain class/module.



```
app.module.ts X
my-website > src > app > app.module.ts > routes
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { RouterModule, Routes } from '@angular/router';
4 import { HttpClientModule } from '@angular/common/http';
5 import { ReactiveFormsModule } from '@angular/forms';
6
7 import { AppRoutingModule } from './app-routing.module';
8 import { AppComponent } from './app.component';
9 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
10
11 import { TaskService } from './task.service';
12
13 import { HomeComponent } from './home/home.component';
14 import { AboutComponent } from './about/about.component';
15 import { ListComponent } from './list/list.component';
16 import { CreateComponent } from './create/create.component';
17 import { EditComponent } from './edit/edit.component';
18
19 // angular material:
20 import { MatToolbarModule } from '@angular/material/toolbar';
21 import { MatSliderModule } from '@angular/material/slider';
22 import { MatDividerModule } from '@angular/material/divider';
23 import { MatCardModule } from '@angular/material/card';
24 import { MatButtonModule } from '@angular/material/button';
25 import { MatTableModule } from '@angular/material/table';
26 import { MatInputModule } from '@angular/material/input';
27 import { MatSnackBarModule } from '@angular/material/snack-bar';
28 import { MatFormFieldModule } from '@angular/material/form-field';
29 import { MatSelectModule } from '@angular/material/select';
30 import { MatIconModule } from '@angular/material/icon';
31 import { NotFoundComponent } from './not-found/not-found.component';
32
```

Image: A lot of imports in the app.module.ts. Everything we ever use on the website.

In this file we can also find a route logic. How and what components will the website display when navigating to a certain route. We also define a default as well as redirect route (optional). I created a new component that redirects you to a page 404 and informs you that the page requested does not exist.



```
const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'list', component: ListComponent },
  { path: 'about', component: AboutComponent },
  { path: 'create', component: CreateComponent },
  { path: 'edit/:id', component: EditComponent },
  { path: '404', component: NotFoundComponent },
  { path: '**', redirectTo: '404' },
];
```

Picture: Routes object used for taking care of routing.

There is an assets folder with some image material for the website.

A bit further down the folder structure, we can find a well known index.html. This is the real root of the website. The look of it is familiar to anyone who has ever been taking part in web development. We have our well-known head and body section, where the body is interested to us. In the <body> tags there is just one element alone - <app-root>. This is the root of our components, mentioned above.

We also have a bunch of other files for various reasons. For example, we need to modify some json files when the project is production-ready. There are also some files used for testing purposes that I will not dive into, because they are not relevant at the moment.

Backend

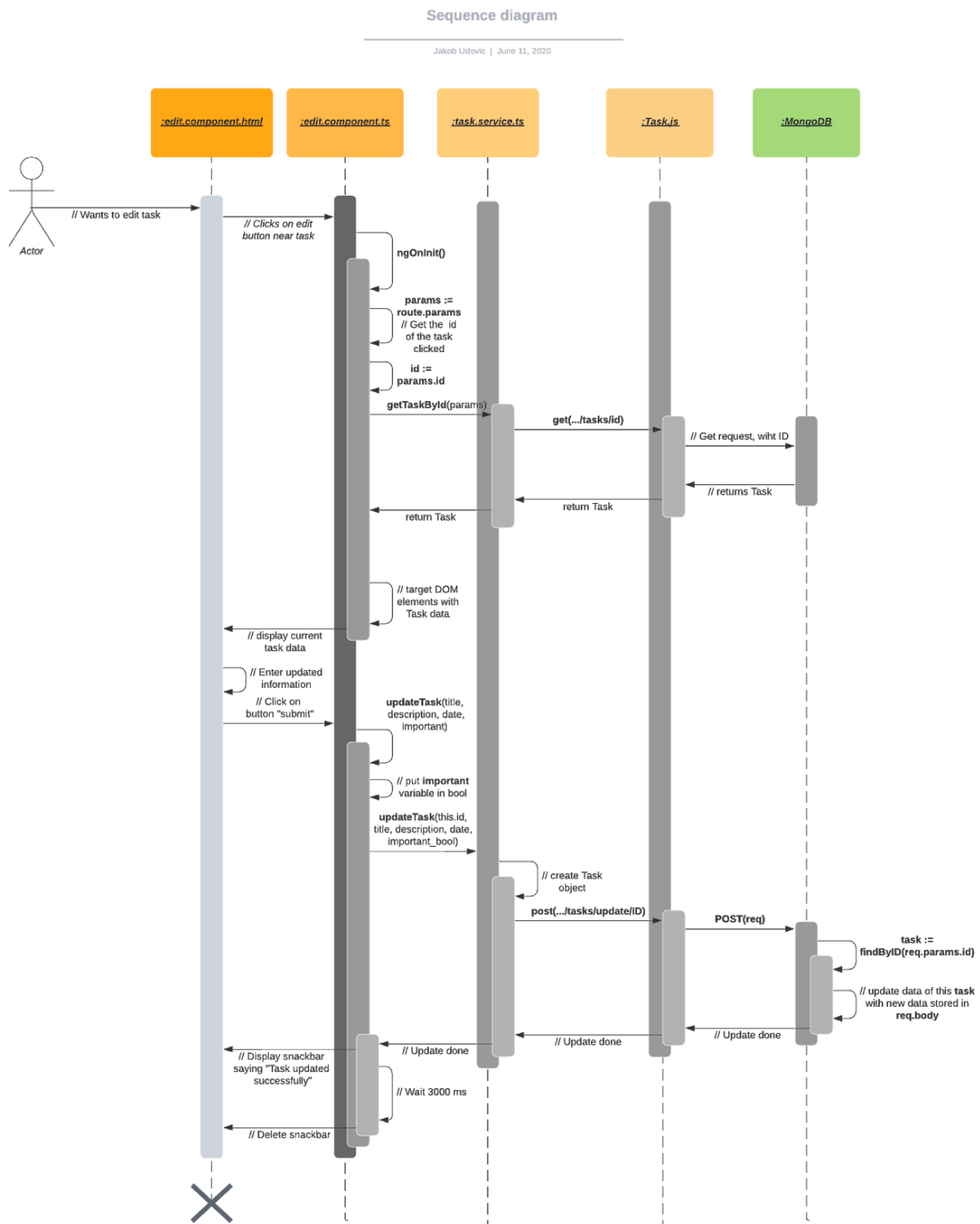
In the backend folder, we can find all the logic happening “under the hood”. I used a mongoose schema for the architecture of the Task objects stored in MongoDB. It acts as a separate data structure. Task.js (starting with a capital) serves as a tool to use for the backend data. We can see all tasks in raw format on localhost:4000/tasks. If we were to delete a task, we would copy one id from the visible tasks and navigate to localhost:4000/tasks/delete/ID. That would delete the task with an id ID and return us a message "Remove successfully".

In package.js file we can find all the libraries we are using in this backend folder, as well as other information about our project (version, dependencies, author, license...). Package.js in the my-website folder (frontend) has a similar feature. When trying to run this project with commands “npm run dev” in the backend folder and then “ng serve” in my-website folder, one should make sure he has those dependencies installed.

server.js file acts as a backbone of the server. It unites other js files in the backend folder, just to keep things more organized. It establishes a connection with a mongoDB server, by passing my username and password in the URI. I use cors to avoid some errors regarding the required connection between my localhost and mongoDB³.

³ More on CORS can be found here:

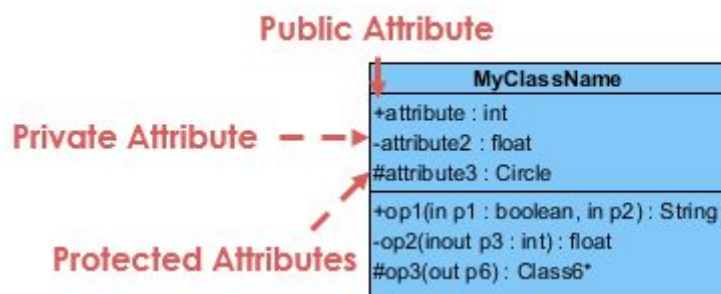
[https://en.wikipedia.org/wiki/Cross-origin_resource_sharing#:~:text=Cross%2Dorigin%20resource%20sharing%20\(CORS,scripts%2C%20iframes%2C%20and%20videos.](https://en.wikipedia.org/wiki/Cross-origin_resource_sharing#:~:text=Cross%2Dorigin%20resource%20sharing%20(CORS,scripts%2C%20iframes%2C%20and%20videos.)



Picture: Sequence diagram of the task editing. (Can be also found on my website in full resolution)

3.2 Detailed design

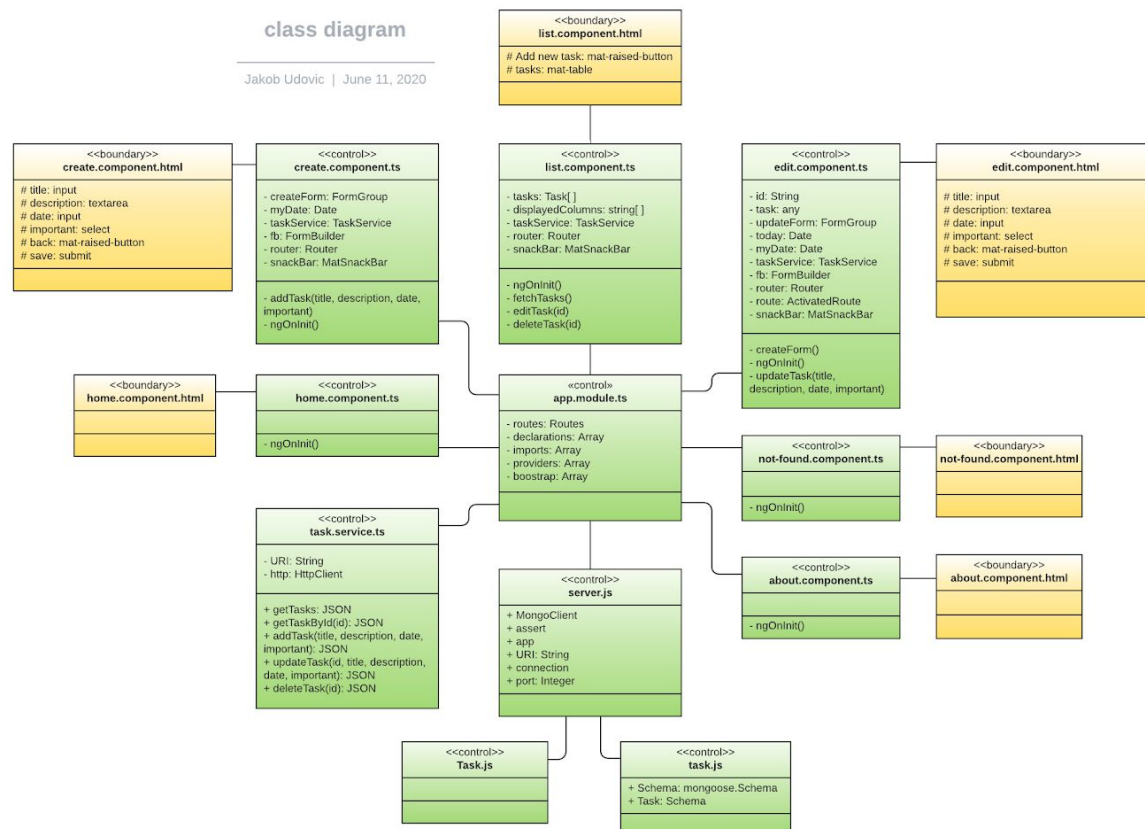
In this section a reader will be introduced to all components used, their methods and corresponding attributes. This will be represented with a class diagram⁴, which can also be found, as well as all other diagrams, in better resolution on the website itself (“about” section).



- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations

Picture: Explanation of the class diagram below.

⁴ More on class diagrams here: [UML Class Diagram Tutorial](#)



Picture: Class diagram.

As we can see, most of the attributes are private. That is so the components do not rewrite/use the same objects at the places you do not want them to.

Green squares represent control and yellow ones boundary elements of the page.

We can also see all the methods and functions typescript files use, as well as their return type (if there is any).

The most important idea behind the whole system is fetching the tasks from the Mongo database and updating existing as well as adding new ones.

```

// getting the tasks method
fetchTasks() {
  this.taskService.getTasks().subscribe((data: Task[]) => {
    this.tasks = data;
    console.log('tasks:');
    console.log(this.tasks);
  });
}

```

Picture: fetchTasks() method in list.component.ts

When obtaining the methods from the database, we fetch them using method `fetchTasks()` in `list.component.ts`. Here, after injecting the `taskService` in constructor, we subscribe to its function `getTasks()` and wait for the `Tasks[]` array to be returned from the database. After receiving it from the backend, we store the result in a private variable called `tasks`. Angular takes care to automatically update html components, whenever a certain data being used gets changed. This is done by Angular variable listeners. I will not dive into that here, but in short, it provides us with more events, like `OnChange`, so whenever an element with this listener binded to it changes, we can do something in between⁵.

```
// function to add task, using task service
addTask(title, description, date, important) {
  let important_bool = false;
  if (important === 'High') {
    important_bool = true;
  }
  this.taskService.addTask(title, description, date, important_bool).subscribe(() => {
    this.router.navigate([ '/list' ]);
    this.snackBar.open('Task added successfully!', 'OK', {
      duration: 3000,
      horizontalPosition: 'center',
      verticalPosition: 'top'
    });
  });
}
```

Picture: `addTask()` method in `create.component.ts`

The other important feature of the website is to create and add a new task to the database. We do this in a create component element. First we build a form with injected `FormBuilder` and store it in a (private) `createForm` variable. After a user has inputted the data for the new task, we call the function `addTask()` with a click on the button of the submit type. Here, we have some more checks done, for example we check if the required field “Title” is not empty. If it is, the form is invalid and therefore the button disabled. This prevents us from sending a false form producing an error in the backend, since we explicitly stated that the field “Title” is required.

```
addTask(title, description, date, important) {
  const task = {
    title: title,
    description: description,
    date: date,
    important: important
  };
  return this.http.post(`${this.URI}/tasks/add`, task);
}
```

Picture: `addTask()` method in `taskService`

⁵More on variable `OnChange` listener: <https://angular.io/api/core/OnChanges>

After getting all the data, I transform the important variable from string to boolean with one simple if clause. After that, an already known taskService is called and subscribed to. We pass it our (updated) data, where it creates a constant object task and makes a POST request to the backend. Because we are subscribed to this event in the frontend, we can wait for the reply 200 OK, stating the POST request was successful. Finally, we display a snackBar message saying "Task added successfully!" for 3 seconds.

Those were 2 most important examples of the website. Everything else connected with the tasks is pretty similar to this communication flow.

3.3 Design justification

In this section we discuss my design decisions and motivations behind them.

Besides the task scheduler and providing general information about me I have many more ideas to add. Therefore the site will serve me as a template for my future projects and functionalities being implemented in the near future. Due to limited time of this course many of which will have to be postponed to the summertime.

Just to name a few:

- Instagram bot script made with Selenium⁶
- MP3 music converter
- Favorite spotify music streams
- ...

Picking Angular for this project amongst various other options was not an easy task. As I said before, I worked on a project previous summer, learning Vue.js whilst developing the frontend part of the website. I will not say I am now confident with Vue and want to learn something new instead. There is a lot more for me to improve there. I could say though I wanted to see what else is on the market.

There were many available options, Angular, React and Django, just to name a few. I ended up choosing between Angular and React due their popularity⁷ and React's compatibility with Facebook apps⁸. I also watched plenty of Angular tutorial videos in the past, but I lacked practical knowledge. After many days of researching and decision making I agreed on using Angular this time.

For the actual design of the website I did not (yet) pay great attention nor spent a lot of time perfecting it. I wanted to learn the actual logic behind it rather than spending already precious time on making the user interface polished. Angular has provided me with a great amount of pre-built elements⁹ that I could simply just import and use anywhere I wanted.

I am of the opinion that the website looks good, but can always be improved.

⁶ One of the first ideas for this course that I actually implemented this semester and will be included either as a script alone, if time will not let me implement some sort of GUI for it on my website.

⁷ Most famous web frameworks in 2020: <https://medium.com/front-end-weekly/10-most-popular-web-frameworks-in-2020-167b9103e08a>

⁸ Facebook uses React as their framework: <https://developers.facebook.com/>

⁹ Angular material: <https://material.angular.io/>

4. Evaluation

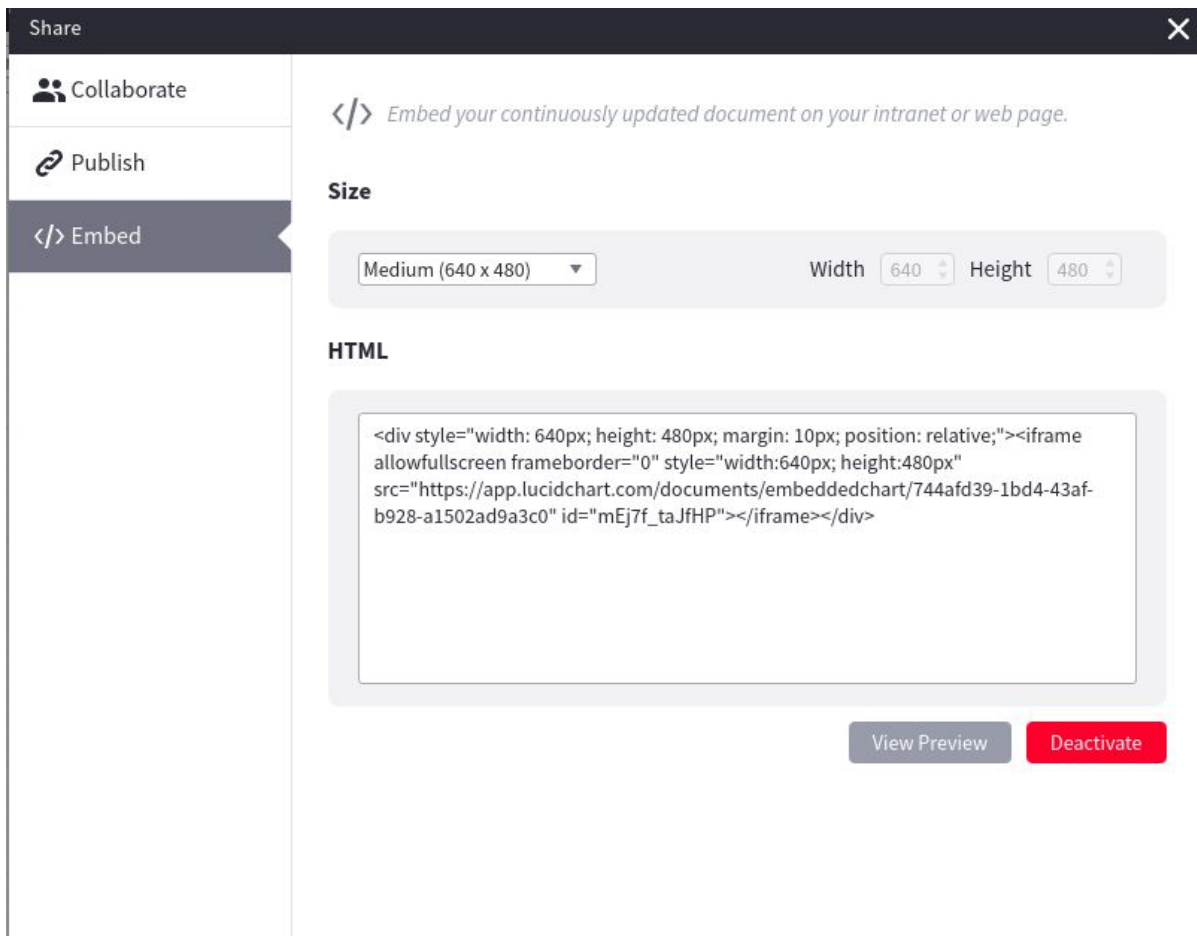
This project was pretty interesting to work on. The biggest struggle was picking a right theme, because I just had too many ideas. For a long time now, I wanted to invest my time into learning some framework, from scratch. Now I understand better how a certain framework communicates with the backend of the system.

I will spend more time on the website in the future, improving it over time. I am proud that it is somewhat responsive, so I will be able to use it from the phone too. Users can try and resize it. The empty sides on the side dynamically disappear, which gives it a more professional look.

Most of the time I was working behind the schedule, which is pretty normal for me. But the good thing is, I was only 1 week behind, at most. I am glad I did not postpone this project to the last week, because I would not be able to finish it. Originally, I set my goals too high, too many things to implement. But this was, again, more of a research than a development project, since with my 1 and a half years programming experience I could not produce anything breathtaking.

For the time of the project, I have dedicated the “about me” section for the diagrams, because I do not believe they will be readable from this document. The Lucidchart¹⁰ tool that I have used for them provides me with a great option to publish graphs by sharing embedded code on my website.

¹⁰ Lucidchart: <https://www.lucidchart.com>



Picture: Embedded code sharing option for Use case diagram.

When the project is over and I hopefully pass the course, I plan on changing that view to my actual information/contact page and making it as a default component route as well as presentable for the eyes.

I am not known for good time management but I think this project helped me a bit to understand that not everything can be done in one day. Now, that I look back, I appreciate every day that I implemented at least one thing, wrote one paragraph here or just did some research about a certain thing.

The weekly emails regarding the project helped me to be on track with the work too. I guess I will have to set those reminders myself in the future and plan the whole project beforehand, by estimating time frames needed for each part.

A lot of times I prioritised other university work so I mostly worked on the R&D during the weekends. This was the biggest and quite possibly the only factor that determined my work schedule, since most of my exchange friends left me due to the Corona crisis.

5. Other work

As the last part, I would like to include other work I tried to add to the website or just wanted to work on as a part of the R&D project.

1. Raspberry Pi fan controller.

The fan is controlled by the GPIO pin 12. I wanted the fan to run for some time and then stop as well. Otherwise there were too many quick changes which was annoying. I made an interactive bash output, where the user can see some sort of bar indicating how much time it needs for the pin to check the temperature again. Every for loop we change one dash to | and see the animation as a smooth loading bar. In between we wait 0.167 second which leads to a total of 7.5¹¹ seconds of fan either spinning or waiting for the temperature to increase.

The script is 80 lines long and will be included in the OTHER folder.

```
pi@raspberrypi:~/fan-project $ python3 fan1.py
fan1.py:15: RuntimeWarning: No channels have been set up yet - nothing to clean up! Try cleaning up at the end of your program instead!
  gpio.cleanup()
fan1.py:17: RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(False) to disable warnings.
  gpio.setup(FAN_PIN, gpio.OUT)
Average: 42.4, Target: 44.0
T: 42.355 ; Cold enough. Stopped.
|||||
Average: 42.4, Target: 44.0
T: 42.355 ; Cold enough. Stopped.
|||||
Average: 42.4, Target: 44.0
T: 42.355 ; Cold enough. Stopped.
|||||
Average: 42.5, Target: 44.0
T: 42.842 ; Cold enough. Stopped.
|||||
Average: 42.4, Target: 44.0
T: 41.868 ; Cold enough. Stopped.
|||||-----
```

Picture: Animation, sadly only as a picture, run on RPI.

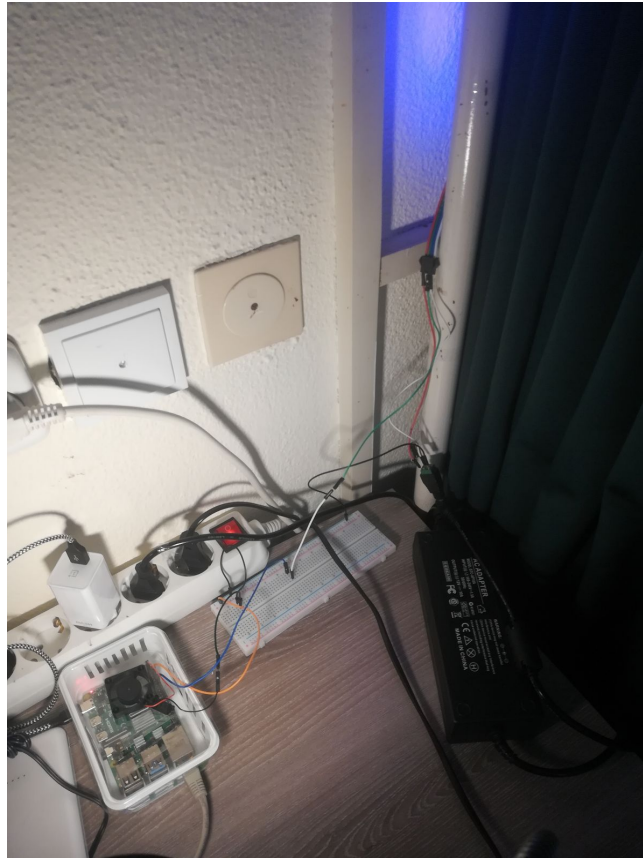
2. LED strip.

The python script can be run with

“sudo PYTHONPATH=".:build/lib.linux-armv7l-2.7" python strandtest.py“.

Here, I was just experimenting with some LED strip animations. I managed to get them to work, for which I required quite some knowledge about the circuits. Sadly, I will not be able to include a picture of the circuit I made, but there is a picture of the whole setup. LEDs are controlled by the RPI, but I originally wanted to use NodeMCU. Due to MCU suddenly not responding to my flashes from android studio, I had to abolish the project and move on with RPI.

¹¹ 0.167 * 45 = 7.5



Picture: My LED setup, controlled by the RPI. We can also see the fan on top of the RPI server.

3. Python instagram bot made with Selenium.

This project was supposed to be merged with the actual website, where a user would input their credentials - instagram username and password, and choose one of the many options bot provides. For now I have finished 2 functionalities:

- Like (or unlike) all user's posts
- Unfollow all people you follow.

This script was made with Selenium¹², a tool that is normally used for web scraping. After running it, it opens a separate chrome window, logs in your account and does whatever you ordered it to do. At first, it was pretty hard to learn the Selenium library and what it does, but gradually I moved on from hard coded time delays to await methods. It made the script a lot smoother and professional. I wish to work on it more, but first, I will try to put it on the website, create a GUI. I have made some research and it is not as trivial as it seems. Since python does not live in the browser, but rather on the server in the backend, I should make this service as a REST API, using Flask, for example¹³.

How it works is, you put your password in a separate file called secrets and import it in the python script at the beginning like: "from secrets import pw".

Now your password is stored in the pw variable, hidden from everyone looking at your code.

¹² Selenium: <https://selenium-python.readthedocs.io/locating-elements.html>

¹³ <https://stackoverflow.com/questions/43917263/how-to-call-python-script-from-angularjs-app>

Bot can also read all people who you follow, but do not follow you back and store them in the file called “unfollowers”. This way you know with whom you need to have “the talk”.