Fakultät für Ingenieurwissenschaften, Informatik und Psychologie

Institut für Datenbanken und Informationssysteme (DBIS)

Konzeption und Realisierung einer interaktiven und Feedback-orientierten Webplattform für Stresspatienten

Abschlussarbeit an der Universität Ulm

Vorgelegt von:
Ana Luciana Anisie Coderea
ana.anisie-codrea@uni-ulm.de
2000527

Gutachter:
Prof. Dr. Manfred Reichert
Prof. Dr. Rüdiger Pryss

Betreuer:
Rüdiger Pryss

2021

Fassung July 26, 2021

Satz: PDF-LaTeX 2_ε

Abstract

Stress is a normal biological reaction to stressors. Stress is more and more present in our lives. Exposure to high stress levels for a long period of time can have negative effects on emotional and physical health. It is important to control and reduce stress and anxiety. Technology can help to measure, track and reduce the stress level.

At University Ulm there exists several applications where patients suffering from stress can track their stress levels.

The goal of this project is to develop an interactive and feedback-oriented web platform for patients that participate in the stress tracking program. The web platform should offer to users the opportunity to visualize their recorded stress data. Statistic related with their own data and in comparison with other participants should be presented.

This web platform could help patients to become more conscious of their stressors or stressful situations, manage and maybe reduce their stress levels.

Contents

1 Introduction

1.1 The Aim of this Work

The purpose of this project is to develop a web platform that allows people to visualize the data about their stress levels that has been collected during their participation in the **TrackYourStress mHealth platform** running at University of Ulm. The detailed goals are explained in detail in Project Goals.

1.2 The Overview of Chapters

The document is structured as follows.

The chapter **Background Information** presents a brief definition of stress, introduces the **TrackYourStress mHealth platform** and briefly explains the goals of this project.

The next two chapters focus on the high-level architecture of the project. The chapter **System Context and Architectural Patterns** presents the highest-level structure of the project. It explains the main concepts and patterns that have been used in this project. The chapter **Container Level Architecture and Used Technologies** describes the frameworks and tools that have been used to implement the concepts and patterns explained in **System Context and Architectural Patterns**.

Once the high-level architecture decisions have been explained, the following two chapters detail the design of the application on the component level. The chapter **Backend Architecture** presents the details of the server part of the application. The chapter **Frontend Architecture** presents the details of the client part of the application.

Next three chapters describe various specific aspects of application development. The chapter **Statistic** presents in detail how the statistic is calculated. The chapter **User interface** presents the user experience and user interface of the project. The chapter **Testing and Documentation** describes the testing of the application and the code documentation.

1.3 C4 Diagramming Standard

To present the architecture of the project I use the **Diagramming Standard C4**.

This diagramming technique **C4** was developed by Simon Brown to modernize the UML diagrams and address certain deficiencies of UML approach.

1.3.1 Context

The first **C** in the *Diagramming Standard C4* is the **Context diagram**. The **context** refers to the business use case, the whole system, external dependencies, user roles, and other systems that it interacts with.

1.3.2 Container

The second **C** in the *Diagramming Standard C4* is the **Container diagram**. The **container** refers to the applications, data stores or microservices that make up the system.

1.3.3 Component

The third **C** in the *Diagramming Standard C4* is the **Component diagram**. The **component diagram** shows the components inside of the containers and interactions between them.

1.3.4 Class

The fourth **C** in the *Diagramming Standard C4* is the **Class diagram**. The **class diagram** defines and provides the overview and structure of the system in terms of classes, attributes and methods, and the relationships between them [18].

2 Background Information

To understand why this project takes place, it is necessary to understand what the stress and its implications are. I will briefly describe the problem of stress and explain how the goals of this project stand in the context of the **TrackYourStress mHealth platform**.

2.1 Stress

The World Health Organization qualifies stress as the "Health Epidemic of the 21st Century"[4]. Stress can have serious effects on emotional and physical health. Hans Selye was an endocrinologist who did important scientific work on the hypothetical nonspecific response of an organism to stressors. According to Selye stress is the non-specific response of the body to any demand [4].

Stress is a natural and essential mental and physical reaction but if the stress level remains elevated for long it has a negative impact on human health. Selye developed the concept of general adaptation syndrome (GAS) which represents a three-stage reaction of coping with stress.

The **alarm** is the first stage when the body reacts with a "fight-or-flight" response to a stressor. This stage prepares a person to respond to the stressor. In normal conditions, the alarm reaction dos not last for long.

The **resistance** is when the body begins to prepare itself to handle the stressful situations. The cortisol is being released, heart rate and blood pressure are increasing. If the stressful situation remains unsolved the stress hormone, heart rate and blood pressure remain elevated. This can lead to serious health problems.

The **exhaustion** is the third stage and appears if the stressor continues to exceed the body's handling capacity. At this stage, the body no longer can fight stress and the risks of developing stress-related health conditions are high [3].

Stress is an ordinary occurrence and it is impossible to avoid all the stressors in our life. It is important to manage the stress levels to prevent ourselves from the negative health effects of stress.

2.2 The TrackYourStress mHealth platform

The Institut für Datenbanken und Informationssysteme at University Ulm runs several projects related to stress tracking. The users can track their stress by answering questionnaires and monitoring the stress level over time. There are several mobile and desktop applications to measure stress levels. For example, at the website trackyourstress.org the user can log in and answer daily, weekly, and monthly questionnaires [12], [11].

2.3 Project Goals

The goal of this project is to develop a website where the verified user can:

- view the questionnaires they have answered,
- download the answered questionnaires,
- view graphical statistic related with their own data,
- view graphical statistic in comparison with other users.

This application, the same as the trackyourstress.org website supports both English and German language.

3 System Context and Architectural Patterns

In this chapter, the project context and the used architectural patterns are presented. The context diagram is presented in the figure 3.1.

3.1 System Context

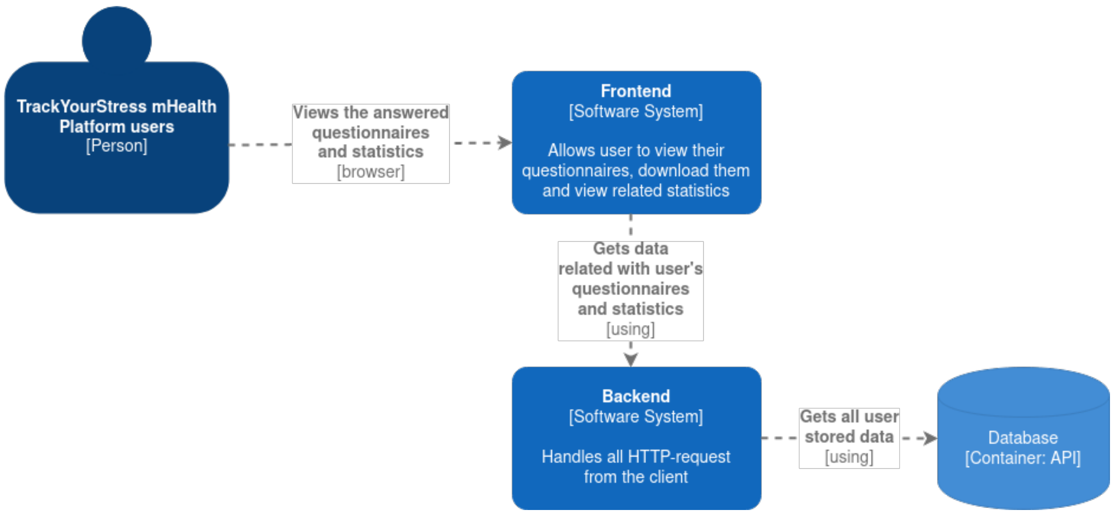

TrackYourStress mHealth Platform users
[Person]
Views the answered questionnaires and statistics
[browser]
Frontend
[Software System]
Allows user to view their questionnaires, download them and view related statistics
Gets data related with user's questionnaires and statistics
[using]
Backend
[Software System]
Handles all HTTP-request from the client
Gets all user stored data
[using]
Database
[Container: API]

Figure 3.1: Context diagram

In this project, the user of **TrackYourStress mHealth platform** can visualize information related to their answered questionnaires. The frontend system communicates with the backend system to get all the data and process them. The backend system uses the Rest API to retrieve the user's data from the database.

The project consists of three parts, each being responsible for a specific task. The **single-page application (SPA)**, serves the static content and runs in the user's browser. The **backend** application processes and serves the requested user's data. The **Database API** is used by the **backend** to retrieve the raw data stored in database.

The software systems and the interactions between them are presented in Context diagram.

Each component is defined and described in more detail below.

3.1.1 Single-page Application

A **multi-page application** sends an HTTP request every time the user interacts with it. The server sends in response the HTTP content back to the browser and the entire page is reloaded.

A **single-page application (SPA)** is a web application that dynamically rewrites a web page in a browser instead of loading the entire page every time. An initial HTML document is requested from the server and sent to the browser. The user interaction will request small fragments of data that are inserted into the initial HTML document.

A single-page application can be much faster than a multi-page application because it does not require full content reload . The single-page application downloads only the needed data that can reduce the network traffic and possibly improve responsiveness of the application. In this model, the server is no longer responsible for creating dynamic HTTP content. [16]

Single-page applications may perform better when the data needed to render a page is small in comparison to the HTTP content. Better performance means a better user experience as the application responds faster to the user's request. This approach splits the backend (server part) and the frontend (client part) development which makes a single-page application highly decoupled. This separation brings the opportunity to use serverless architecture. The SPA applications are easier to port from web to mobile by reusing the backend code.[5]

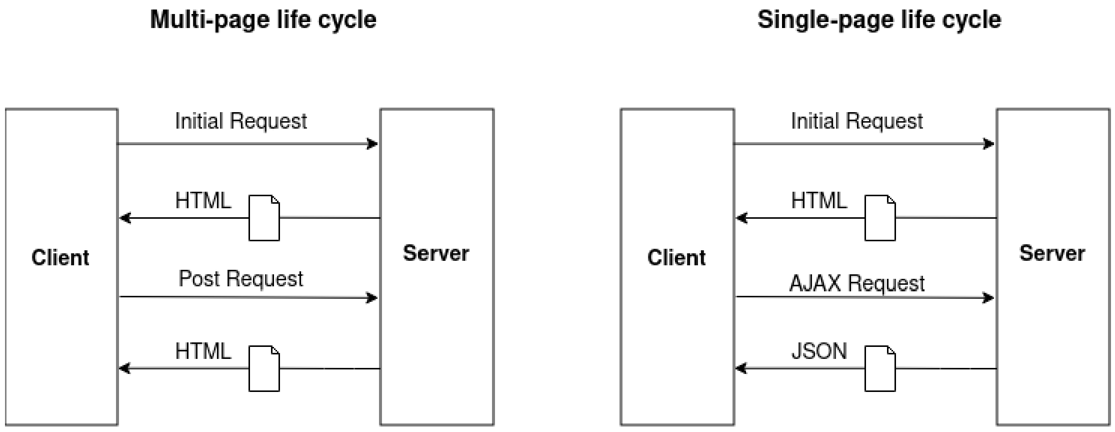


Figure 3.2: Page life cycle

The choice between a single or multi-page application depends mainly on the characteristics of the project. The biggest drawback of a single-page application is that it is not Search Engine Optimization SEO-friendly. Since single-page applications load the content only after a user interaction for a search engine it seems to be a page with no content. Another disadvantage of SPA is security because they are prone to *cross-site scripting (XSS)* attacks. Since they run on JavaScript, it does not compile code, making it more vulnerable to malware.

All these drawbacks can be overcome by implementing good practices like keeping the logic on the server side to avoid sensitive information leaks.

Some of the reasons I chose to develop a single-page application for this project are:

- the web is geared towards user interaction,
- the data needed to render a page is small,

- the database is only accessible via Rest API.

3.1.2 Backend

In software engineering, the terms front-end and back-end refer to the separation between the presentation layer and the data access layer.

The presentation layer also called **frontend** refers to the client-side, where the focus is on the interactions with the end user. The frontend includes everything that the user experiences directly, from text, colors to menus and navigation buttons.

The data access layer also called **backend** refers to the server-side, where the focus is on storing, processing, and organizing data. The backend includes everything that is related with the data processing, logic hardware, and storage[27].

The following figure represents these two layers and their interactions.

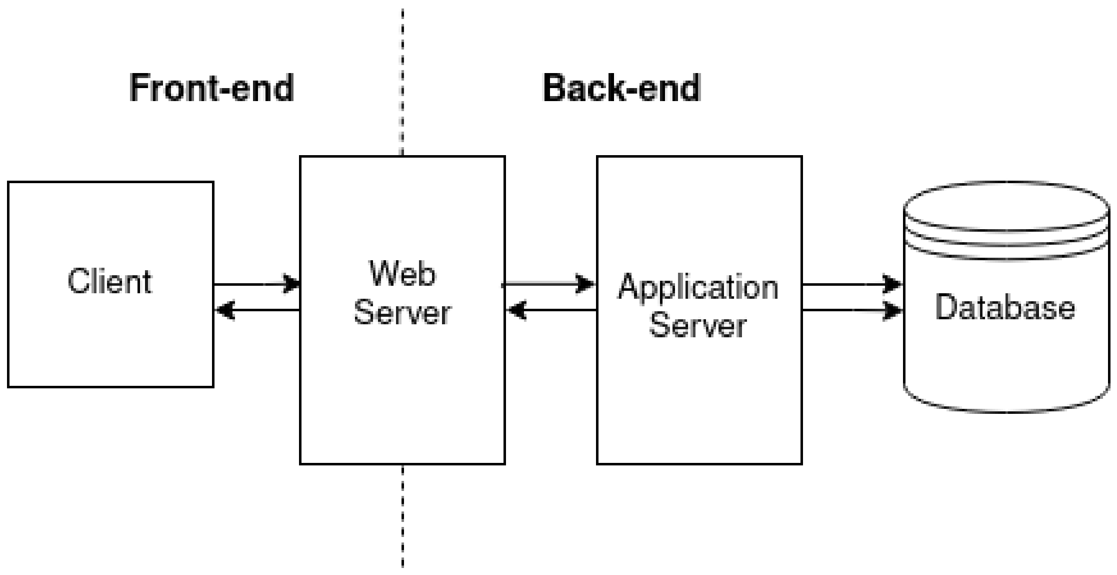


Figure 3.3: Back-end and Front-end Architecture

This project contains a backend application where the logic and the data processing are done. In this way, the presentation layer and the data access layer are separated, making the application more organized and secure. The sensitive data is not exposed and the information sent to the client-side is ready to be displayed.

3.1.3 REST API

The **Representational state transfer (REST) application programming Interface API** is a software intermediary that allows the interaction of multiple applications. The REST API defines the functions and procedures that govern the data access point. The REST API handles requests from the client and returns a response to those requests[13].

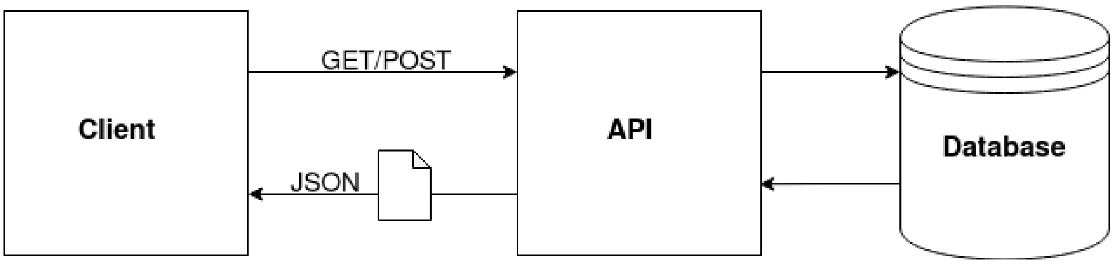


Figure 3.4: Application programming Interface (API)

In this project the used REST API is provided by the *Instituts für Datenbanken und Informationssysteme (DBIS)* at university Ulm. The REST API was created using the PHP web framework Laravel.

REST API services are addressed via URL using the basic HTTP request methods: **GET**, **POST** or **DELETE**.

3.2 Architectural Patterns

The architectural patterns used in the project are presented next.

3.2.1 Single Responsibility Principle

The containers and components are structured following the **Single-Responsibility Principle (SRP)**. Historically, the **SRP** says that: *A class should have only one reason to change*. Each component has responsibility for a single part of the application's functionality. [14]

3.2.2 Component Design Principle

Software systems are changed to satisfy needs of users and stakeholders, so the final definition of the **SRP** is: *A module should be responsible to one, and only one, actor* [20].

The **SRP** is about functions and classes but it reappears in different levels. At components levels, it becomes the **Common Closure Principle (CCP)**. The **CCP** says:

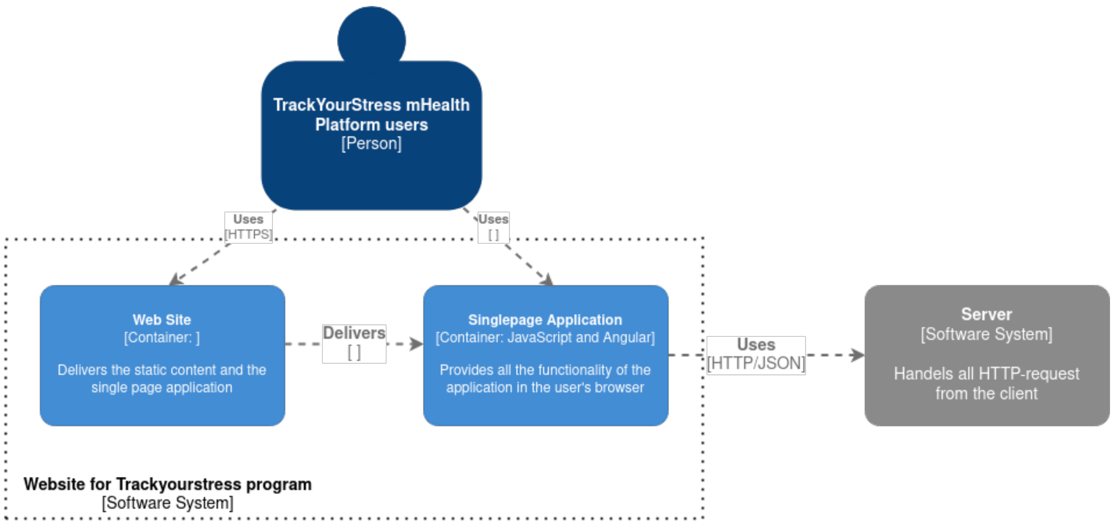
> *Gather together those things that change at the same times and for the same reasons. Separate those things that change at different times or for different reasons* [20].

4 Container Level Architecture and Used Technologies

In this chapter, the container diagrams and the chosen technologies for the frontend and the backend applications are presented and explained in detail.

4.1 Container View of the Frontend

The single page application container diagram is shown below.



TrackYourStress mHealth Platform users
[Person]
Uses
[HTTPS]
Uses
[ ]
Web Site
[Container:]
Delivers the static content and the single page application
Delivers
[ ]
Singlepage Application
[Container: JavaScript and Angular]
Provides all the functionality of the application in the user's browser
Uses
[HTTP/JSON]
Server
[Software System]
Handels all HTTP-request from the client
Website for Trackyourstress program
[Software System]

Figure 4.1: SAP Container diagram

The frontend container diagram describes the bigger parts of the frontend application. The web application serves the static content. The single-page application is an Angular application, and runs in the user's web browser. The single-page application uses the backend application to request the needed data.

4.2 Technologies of the Frontend

The frontend – the single page application, is made using Angular framework due to several reasons:

- the main objective of the project is to build an interactive and user experience-oriented website.
- no need for data storage and database
- not much need of page reloads

4.2.1 TypeScript

TypeScript is a compile-to-JavaScript language, which was released as an open-source project by Microsoft in 2012. TypeScript is a strongly typed and object oriented language. It adds static type definitions and provides a way to describe an object. The types are not automatically converted and they are checked at compile time. TypeScript is a superset of JavaScript [26].

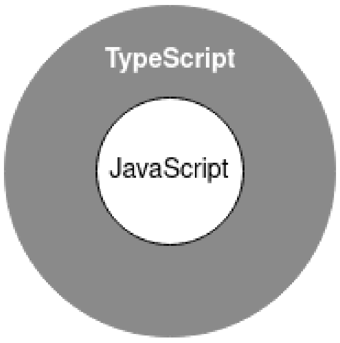


Figure 4.2: TypeScript as a superset of JavaScript

4.2.2 Overview of Angular Framework

Angular is a TypeScript-based open-source web application framework developed by Google.

The angular core value proposition is making it possible to build apps that work for almost any platform, mobile, web, or desktop.

According to the Stack Overflow survey from 2020 Angular Framework is getting more and more popular and the TypeScript is the second laved programming language [28].

Model View Controller

Angular uses a variation of the classical **Model View Controller (MVC)** pattern. In the **MVC** pattern the data (Model), the logic (Controller) and the HTML (View) are separated. The **Model** manages the data of the application in terms of what data is presented to the user. The **View** manages the presentation of the data and control elements such as buttons, menus, etc. to the user. The **Contoller** manages the user interactions and prepares the data for the view. In the logic part of the application the calculations and transformations of the data are handled in the controller. [21]

The Angular MVC pattern uses different terms for the Model View Controller. The *Angular Model* contains the data and the logic to modify and manipulate the data. The model is isolated from the view and from the controller. The *Angular View/Templates* contains the markup and logic to present the data to the user and is defined using HTML elements. The *Angular Controller/Components* contains the logic and behavior required to present or update the data. The *RESTful Services* is in this case a database that manages the data and is accessed through HTTP requests. [6]

Benefits of Using Angular Framework

Some of the benefits of using Angular framework for the frontend are:

- **Component-based architecture** Angular supports custom components that include functionality along with their rendering logic in reusable parts.
- **Data binding** Angular supports bidirectional event binding where changes reflected in the application code or within HTML DOM are reflected in both places.

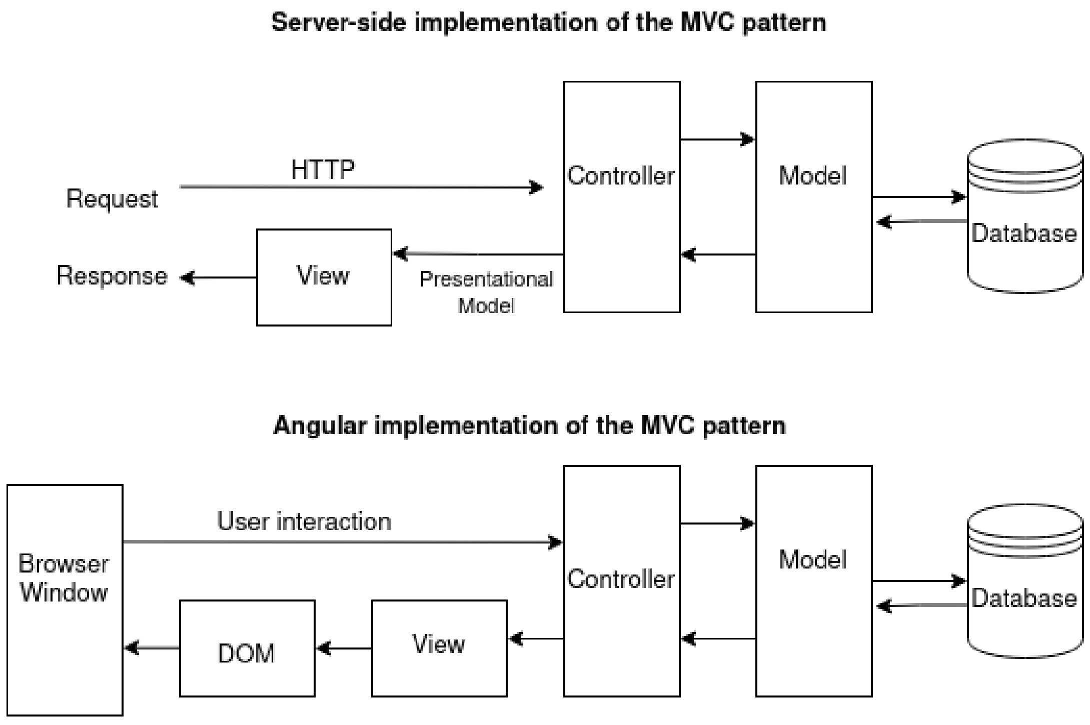


Figure 4.3: Model View Controller

- **Dependency injection** Angular supports modular services that can be injected wherever they are needed – increasing the flexibility of the application [27].

4.2.3 Detailed Description of Angular Framework

In the next section, the Angular concepts are introduced and described.

Component-based Architecture

Angular components are organized in modules. Angular application is bootstrapped by a root module that renders the root component. The root component connects complete component hierarchy to the **document object model (DOM)** of the page. The root component is the **app.component** and is called from the single page – *index.html* – that the server provides. Angular router provides smart hotlink tracking for navigating between components and enables changing parts of a page.

The image below explains how Angular routing works.

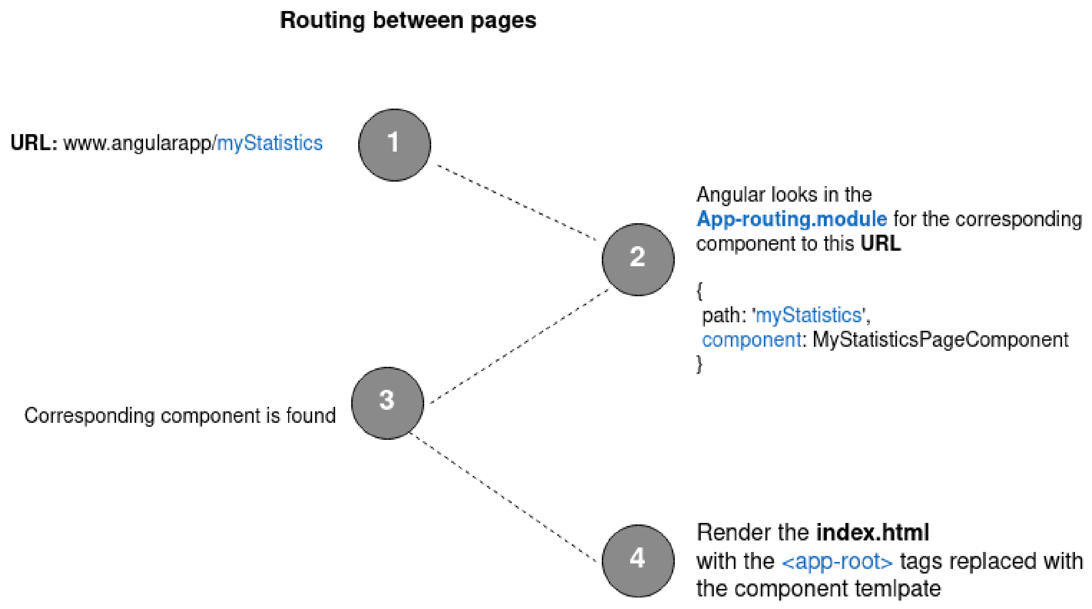

Routing between pages
URL: www.angularapp/myStatistics
1
2
Angular looks in the App-routing.module for the corresponding component to this URL
{
path: 'myStatistics',
component: MyStatisticsPageComponent
}
3
Corresponding component is found
4
Render the index.html with the <app-root> tags replaced with the component temlpate

Figure 4.4: Angular Router

The Angular Components are the basic building blocks of an Angular application. A component is a directive that defines it own HTML content and CSS style. The data, logic, and the HTML for a specific view are encapsulated in a component. They allow dividing large projects into smaller pieces.

The Component is a TypeScript class with the *@Component* decorator. It has an associated HTML file and a style-sheet file. The class contains the logic and data that is available to the HTML template [23].

Elements of the Component

All angular components have a selector, templateUrl and stylesUrls.

The **Selector** is used to identify a component uniquely within the component tree. The selector informs Angular to create and insert an instance of this component wherever it finds the corresponding tag in the HTML template.

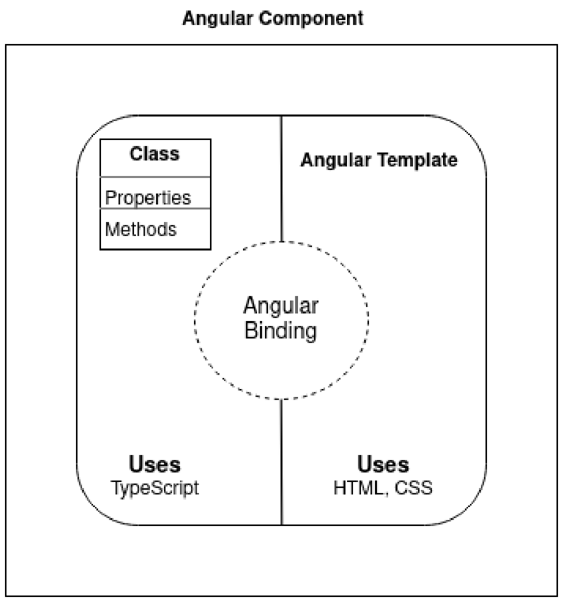


Figure 4.5: Angular Component

The **TemplateUrl** defines the template to be used along with a component. A template is an HTML snippet that tells Angular how to render the component. It defines the component's view.

The **StylesUrls** are used to add CSS styles to the component template. By defining the styles for each component, one gets complete encapsulation and isolation of the styles. The styles used in one component do not affect any other component.

Angular is based on components that have to interact with each other. This interaction can be done in several ways, some of them are: interaction parent to child component, child to parent component, or component to component (siblings). The components follow the parent tree structure, each component is a child of another component, up to the root.

Communication Between Components

To pass information from parent to child the *@Input() decorator* can be added to the child component so that it can receive messages from its parent component.

To pass information from child to parent the *@Output() decorator* and *EventEmitter* can be used. The child has to emit a message and the parent will receive it.

To access a child property or methods the *@ViewChild() decorator* can be used. It is the most flexible form of communication since the parent component has full access to all the attributes and methods within the child component.

Interactions between siblings components in real-time will be primarily dynamic and should be updated with real-time data without the need to refresh the page. This can be done using Subject. A Subject allows values to be transmitted to many Observers. An Observer can subscribe to the Subject and receive values from it. Subject adds them to his collection of observers. Whenever there is a value in the stream the Subject will notify all his observers.

Data Binding Overview

Data binding is the way information is exchanged between the component and the DOM. The exchange of data between the component and the view will help to build dynamic and interactive web applications.

There are three ways of binding the data based on the direction of the data flow: *from source to view, from view to the source or bidirectional sequence from view to source to view*.

One-way Data Binding from Source to View

There are several types of binding data to the view, some examples are:

- *Interpolation binding* is used to display property in the respective view template. The data is moved from the component to the HTML elements.
- *Property Binding* is used to set a property of a view element. It involves updating the value of a property in the component and binding it to an element in the view template.

One-way Data Binding from View to Source

Event binding is used when information flows from the view to the component when an event is triggered. When the event occurs in the view, it calls the method specified on the component.

Two-way Data Binding

Two-way data binding means that changes made to the model in the component are propagated to the view and any changes made to the view are immediately updated to the data in the underlying component. The Two-way data binding sets a specific element property and listens for an element change event. The component and the view are always in sync, and changes made at both ends are immediately updated in both directions [6].

Dependency Injection

Dependency injection is a design pattern in which a class requests dependencies from external sources. Dependency injection keeps code flexible, testable, and mutable. Classes can inherit external logic without knowing how to create it. In Angular, dependency injection is done by injecting a service class into a component. A service is a class of data or logic that is not associated with any specific views to share between components. Services can be used for many things besides requesting data from APIs.

4.3 Container View of the Backend

The backend container diagram describes the bigger parts of the backend application. In this case, the backend has a simple structure and it contains only one container. The user does not interact directly with the backend application. The express-server accepts requests from the frontend. It uses the remote database to get the raw data.

The backend application container diagram is shown below.

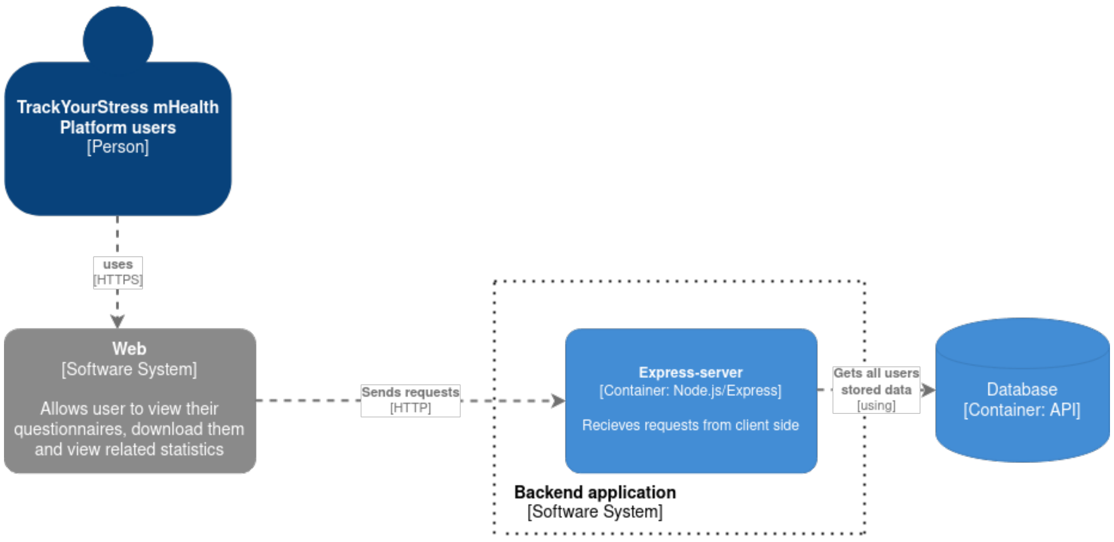

TrackYourStress mHealth Platform users
[Person]
uses
[HTTPS]
Web
[Software System]
Allows user to view their questionnaires, download them and view related statistics
Sends requests
[HTTP]
Express-server
[Container: Node.js/Express]
Recieves requests from client side
Gets all users stored data
[using]
Database
[Container: API]
Backend application
[Software System]

Figure 4.6: Backend container diagram

4.4 Technologies of the Backend

The technologies used fro the backend implementation are presented next.

4.4.1 Node.js

Node.js is a JavaScript runtime engine designed to run outside the browser. It can be used to build scalable network applications, asset compilation, scripting, or monitoring [8]. Some features of Node.js are:

- Fast processing by handling concurrent requests. The Node.js-based server never waits for an API to return data. The server passes to the next API after it is called and a Node.js Event notification mechanism helps the server to get a response from the previous API call.
- As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.

- Single programming language, since it uses JavaScript, there is no need to learn another programming language for the backend.

4.4.2 Express Framework

Express was developed by TJ Holowaychuk and is maintained by Node.js foundation and open source community.

Express.js is a web framework for Node.js. Express provides features for building robust web applications and APIs. It is often used to power single-page applications that normally require a server component. It includes several middleware modules that can be used to perform additional tasks on demand. Express gives a high performance due to the execution of multiple operations independently of each other through asynchronous programming.

It associates an HTTP method to a function or set of functions that are called to handle the path. The way an application responds to a client request is determined by routing. Each route can have one or more handler functions, which are executed when the route is matched. Routers are useful for separating concerns and keeping related parts of code together. This way the code is easier to maintain.

One of the reasons for using the Express framework was the ease of integration with the Angular application. The frontend application request data over HTTP to the express server and receives JSON files as a response. For each request, the server accesses the API and processes the data before sending the response to the frontend [25].

All sensitive data is processed on the express server to prevent information leaks. The data sent to the client is ready to be shown to the user.

Communication is done using the HTTP protocol which is explained next.

4.4.3 HTTP

The **Hypertext Transfer Protocol HTTP** is an application layer protocol used for communication between clients and servers. The browser sends HTTP requests to the server and the server sends an HTTP response.

The transactions between client and server are done through *HTTP request methods*. The method tells the server what action to take [1]. In this project the HTTP methods used are:

- **GET** method requests a representation of the specific resource. Requests using GET should only retrieve data. The server should send the named resource to the client.
- **POST** method submits an entry to the specified resource. Requests using POST often cause a state change or side effects on the server. The server stores the data from the client on a named server resource.

5 Backend Architecture

This chapter describes in detail the software architecture of the backend application.

5.1 Component Diagram

The backend application component diagram is shown below.

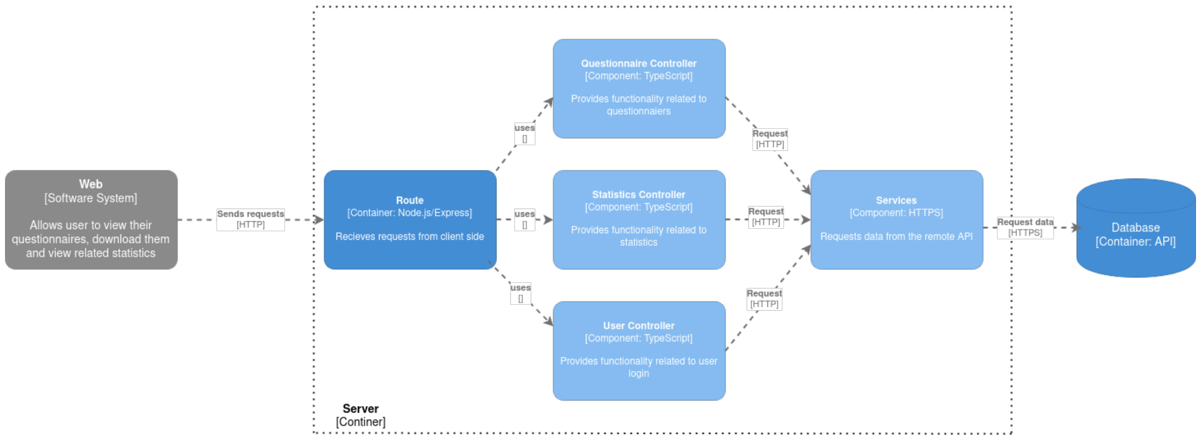


Figure 5.1: Backend component diagram

Each component is responsible for one functionality of the application. Each component is described in detail below.

5.2 Router

The client, in this case, the single page application (angular app) requests data from the express server. Incoming requests are handled in the router's file.

The Express router is a class library that defines the way the HTTP requests from the client are handled.

For each incoming request, the router forwards the request to the appropriate controller. The corresponding controller holds the logic and processes the requests. The controllers request data from the external database using the services component.

5.3 Questionnaire Controller

The **Questionnaire controller** processes the requests related to the questionnaires answered by the user. It is responsible for requesting all answered questionnaires of the user and the answersheet for a specific questionnaire. The questionnaire controller requests the data from the database and contains the logic that translates the answersheets. The controller returns the answerseets processed through the routes to the client. The data is ready to be consumed and does not need to be further processed on the client side.

5.4 Statistic Controller

The **Statistic Controller** processes the requests related to the statistic of the user and all the participants in the *TrackYourStress mHealth platform*. It is responsible for requesting all answersheets for all users from the database. This information is processed separately for the user and the other participants. How statistic are actually calculated is presented in the **Statistic** chapter later in the document. The data is ready to be consumed and does not need to be further processed on the client side.

5.5 User Controller

The **User Controller** processes the login requests. The data sent to the client side is the login token that is attached to all other requests from the user.

5.6 Services Component

Communication with the remote API is done through the **Services**. All the HTTP requests to remote API are grouped in the services class. This class is in charge of requesting the data to the database. The data is sent in JSON format to the corresponding controller without any processing.

5.7 Testing

The methods processing the questionnaires and the statistic are tested using **Uni Tests**. More details about the testing are presented in the chapter **Testing and Documentation** later in the document.

5.8 Backend Design

The backend application is designed using the **single-responsibility principle**, so each component is responsible for a specific part of the application. By doing this the system is very flexible. Adding new features is easy due to the split of responsibilities and only the corresponding controller needs to be changed.

5.9 Tree Diagram

The backend tree diagram is presented in the figure below.

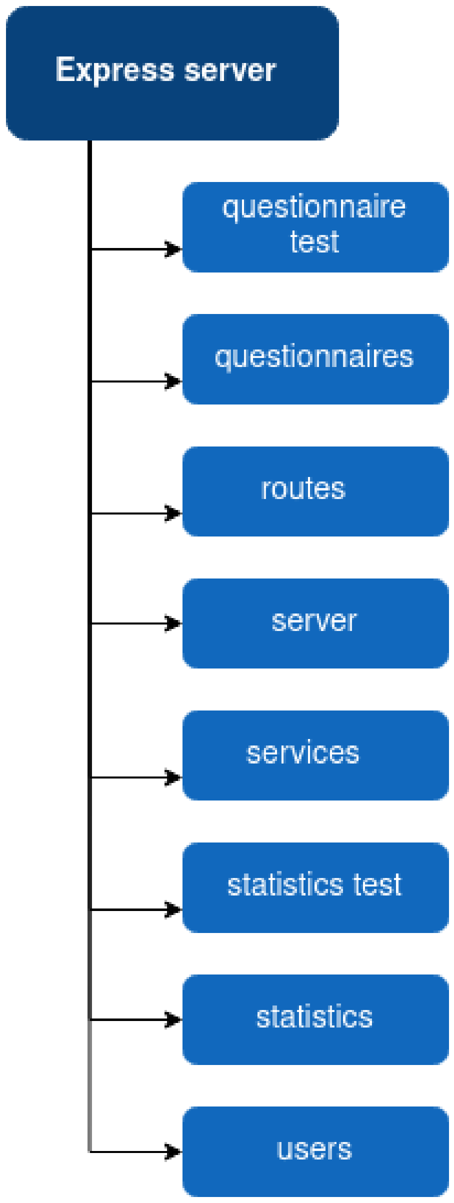


Figure 5.2: Server tree diagram

6 Frontend Architecture

This chapter describes in detail the software architecture of the frontend application.

6.1 Component Diagram

The component diagram for the frontened application is shown in the diagram below.

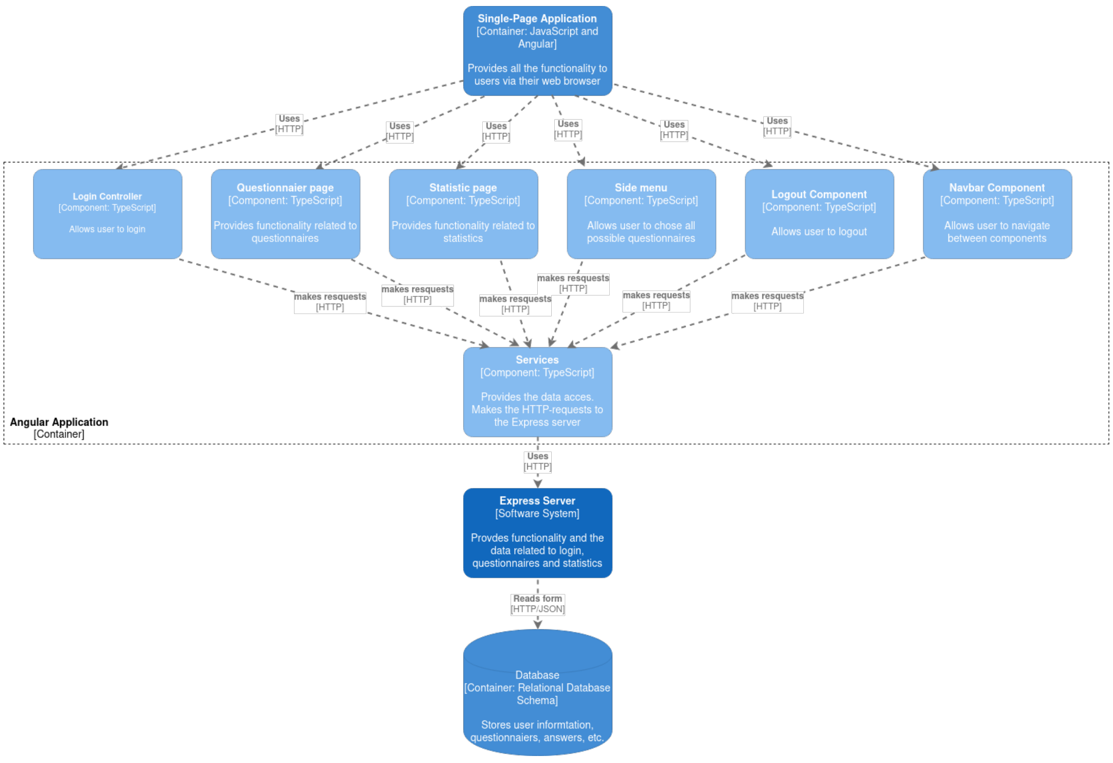


Figure 6.1: Single page application Component diagram

The single page application made with Angular is divided into several components. Each component has a single responsibility and can contain other components to split the tasks. The components communicate and interact with one another to build the complete application.

6.2 Services Component

The **Services** allow components to communicate with the server side application. The services folder contains the classes for fetching the data from the server. The components do not need to request the data directly, so they delegate the task to the services.

The services are responsible for making the HTTP requests to the server side and deliver the requested data. They are divided into three classes, each serving the data related to a specific part of the application.

The *user-services* contains the business logic related to the user login and logout. The *questionnaires-services* contains the business logic related to the questionnaires. And the *answersheets-services* contains the business logic related to the answersheets.

Depending on the type of information that is needed, the controllers use the corresponding services.

The services are also used to communicate between the components. The *navbar-services* are used to send messages to the navigation bar informing when the user logged in or out. The communication is done using Subject and Observers as explained in the chapter *Used technologies* 4.2.3.

6.3 Login Romponent

The **Login component** is responsible for the user login. It sends the login form to the server side using the user-services. When the login is successful the application routes to the questionnaires page. A message is sent to the *Navbar Component* informing that the user logged in.

6.4 Questionnaire Page Component

The **Questionnaire page component** holds other components that are used to display the questionnaires page as a whole. Because the questionnaire page displays everything related to the questionnaires, it consists of several smaller components. Each of them is responsible for a specific part of the page.

On the *Side-Menu Component* the user can select a type of questionnaire. The *Answersheet List Component* is responsible for displaying the list of all answersheets for a specific questionnaire type. The *Answersheet View Component* is responsible for displaying one selected answersheet.

6.5 Statistic Page Component

Similar to the questionnaire page component the **Statistic Page component** holds smaller components that are used to display the statistic as a whole. On the *Side-Menu Component* the user can select a type of questionnaire. The *Statistic List Component* is responsible for displaying the list of all questions for a specific questionnaire type. The *Statistic View Component* is responsible for displaying the statistic for the selected question.

6.6 Side-menu Component

The **Side-Menu Component** is a shared component used to display the questionnaire type. The user can select for which type of questionnaire the information should be displayed. The data is passed to the corresponding component using the EventEmmitter explained in the chapter *Used technologies* 4.2.3.

6.7 Logout Component

The **Logout Component** is responsible for the user logout. The **Logout Component** uses the *user-services* to send a request to the server for the user logout. The

Logout Component uses the *navbar-services* to inform the *Navbar component* that the user logged out.

6.8 Navbar Component

The **Navbar Component** is responsible for the navigation bar that is used in the root component, explained in the chapter *Used technologies*. 4.2.3
This navigation bar is always shown to the user with the most links when the user is logged in.

6.9 Models

The **Models** are used for modeling the infromation that flows between the template, the controller and the backend application. Each model is a class that represents a data structure, they are used to isolate the data structure from the component code.

In this project, the models are very simple, and they contain the property related with the data they model. They store the data delivered by the backend application, and they are used in the templates to display the data to the user.

6.10 Frontend Design

The user interface design and how the data is presented to the user is described in the chapter **User interface**. Dividing the page into smaller parts makes the code easy to debug and flexible to changes. The components can be modified separately and a change in the answersheet or statistic view component does not affect the answersheet or statistic list component.

6.11 Tree Diagram

The frontend tree diagram is presented in the figure below.

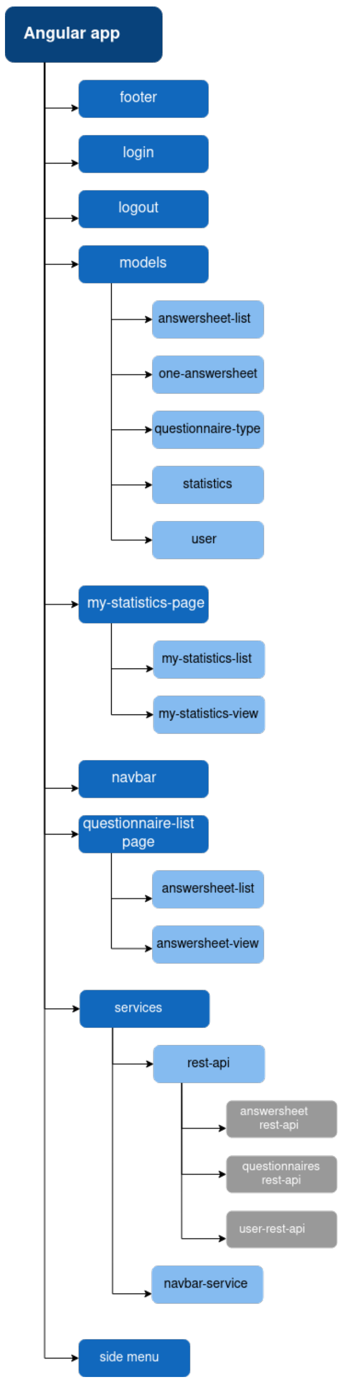


Figure 6.2: Server tree diagram

7 User interface

This project is intended to offer an interactive and feedback-oriented website. The **user interface (UI)** and the **user experience (UX)** are important concepts.

7.1 Easy and Intuitive Design

The **user interface (UI)** is anything a user may interact with to use a software product. The goal is to create interfaces that users find easy to use and intuitive. Easy to use means that the interface should be simple and clear in the language used in the labels, buttons, etc. Intuitive means that the interface should be understandable to the user. The user should not need to remember or memorize things to use it. An intuitive interface is an interface that looks familiar to the user.

7.2 User Experience

The **user experience (UX)** refers to how the user experiences a product. It includes all the aspects of the interaction between the end-user and the website.

The website is a self-service product, where the user interacts with it alone. The user experience is crucial in web development. The users have no instructions on how to use the website, and the only guide they have is their previous experience [7].

The user experience design requires understanding the needs of the end-users. The design is user-centered and focuses on their needs.

7.3 Design Context

This project has a design inspired by the website trackyourstress.org, and the University Ulm web-based email, sogo.uni-ulm.de.

The user of this project web application might have some experience with the mentioned websites. The design should look familiar, and the user should not need to memorize or remember new things to use the web application.

7.4 Main Elements of User Interface

The interface elements such as the navigation bar, menu, and the different pages are presented next.

7.4.1 Home Page

The **Home page** is the page where the user can log in. It has a simple design that looks similar to the login page at the University Ulm web-based email, sogo.uni-ulm.de.

In the middle of the page is shown the **trackyourstress** name next to the login form. The login form is very simple, showing where the user can introduce the login code and the login button. The login form is placed in the center, making it easy to view. The page displays also the horizontal **navigation-bar** placed at the top of the page.

The login page is shown in the figure below.

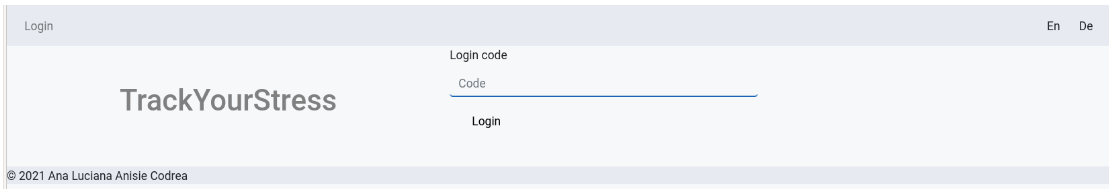


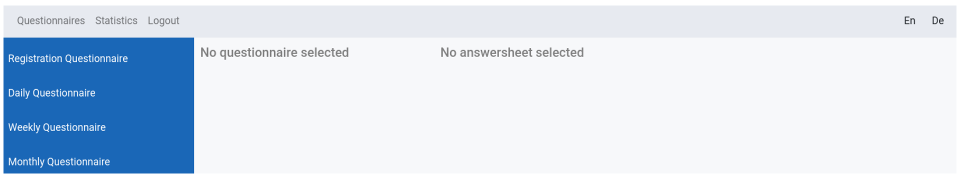
Figure 7.1: Login page

7.4.2 Navigation Bar

The horizontal **navigation-bar** is placed on the top of the page, the same as the navigation bar in the trackyourstress.org website. Many websites use this type of navigations bars, so the users are familiar with it. They are easy to notice, uses little space, and takes a prominent position on the page.

Navigation bars are the most useful when having only a few items. Otherwise, they become quickly cluttered and confuse the user. In the case of this application, there are very few items, so the interface is clear.

The navigation bar is used to switch pages, and for changing the language. The **side menu** is used to navigate inside the page, between the views.



No questionnaire selected

No answersheet selected

Figure 7.2: Navigation bar and side menu

7.4.3 Questionnaire Page

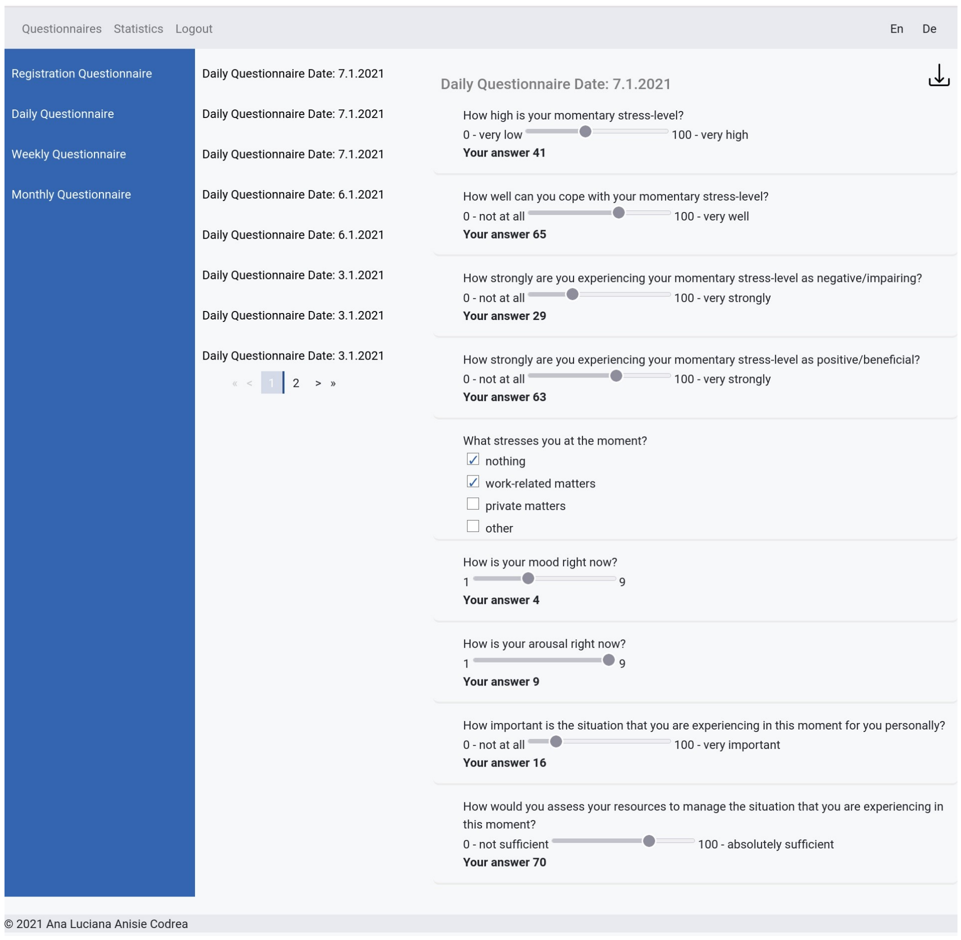
When the user logs in, the **Questionnaire-page** is shown by default.

The design of **Questionnaire-page** (and of the **Statistic-Page**) is inspired by the the University Ulm web-based email, sogo.uni-ulm.de. The page is divided into three columns.

On the left column, the **side menu** is shown. The users can select the questionnaire type, they want to see. The middle column lists all answersheets for a specific questionnaire type. They are ordered by date, in descending order. The content of a single answersheet is displayed on the right column. The design is the same as the university Ulm web-based email. The list of emails is shown in the middle column, and the selected email is displayed in the right column.

The user can answer the questionnaires at the trackyourstress.org website. In this project, the content of the answersheets is displayed similarly to the questionnaire on the mentioned website.

The questionnaire page is shown in the figure 7.3.



Daily Questionnaire Date: 7.1.2021
Daily Questionnaire Date: 7.1.2021
Daily Questionnaire Date: 7.1.2021
Daily Questionnaire Date: 6.1.2021
Daily Questionnaire Date: 6.1.2021
Daily Questionnaire Date: 3.1.2021
Daily Questionnaire Date: 3.1.2021
Daily Questionnaire Date: 3.1.2021

Daily Questionnaire Date: 7.1.2021

How high is your momentary stress-level?
0 - very low 100 - very high
Your answer 41

How well can you cope with your momentary stress-level?
0 - not at all 100 - very well
Your answer 65

How strongly are you experiencing your momentary stress-level as negative/impairing?
0 - not at all 100 - very strongly
Your answer 29

How strongly are you experiencing your momentary stress-level as positive/beneficial?
0 - not at all 100 - very strongly
Your answer 63

What stresses you at the moment?
- [✓] nothing
- [✓] work-related matters
- [ ] private matters
- [ ] other

How is your mood right now?
1 9
Your answer 4

How is your arousal right now?
1 9
Your answer 9

How important is the situation that you are experiencing in this moment for you personally?
0 - not at all 100 - very important
Your answer 16

How would you assess your resources to manage the situation that you are experiencing in this moment?
0 - not sufficient 100 - absolutely sufficient
Your answer 70

© 2021 Ana Luciana Anisie Codrea

Figure 7.3: Questionnaire page

PDF Download Button

The download button is shown on the top right corner of the page. The user can download the questionnaire content in pdf format.

The positioning of the button is due to several reasons:

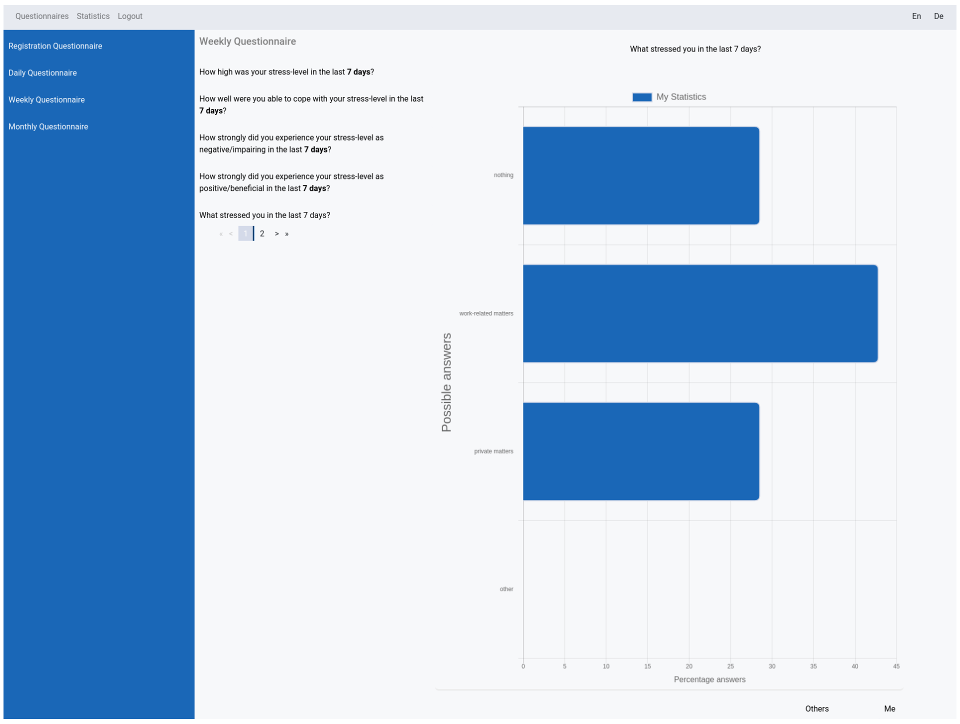
- the position of this type of button is at the top of the page in the web-based email, sogo.uni-ulm.de.
- the user does not need to scroll to the end of the page, to download the document

- some questionnaires, like the registration questionnaire, can be very large, so the download button should be easily visible.

7.4.4 Statistic Page

The **Statistic-page** has the same structure and design as the **Questionnaire-page**. The **side menu** is placed on the left column. The questions are listed in the middle column. The list of questions is displayed in a bunch of five questions. The user can navigate to the next bunch of questions using the pagination buttons.

The statistic page is shown in the figure 7.4 and 7.5.



Weekly Questionnaire

How high was your stress-level in the last **7 days**?

How well were you able to cope with your stress-level in the last **7 days**?

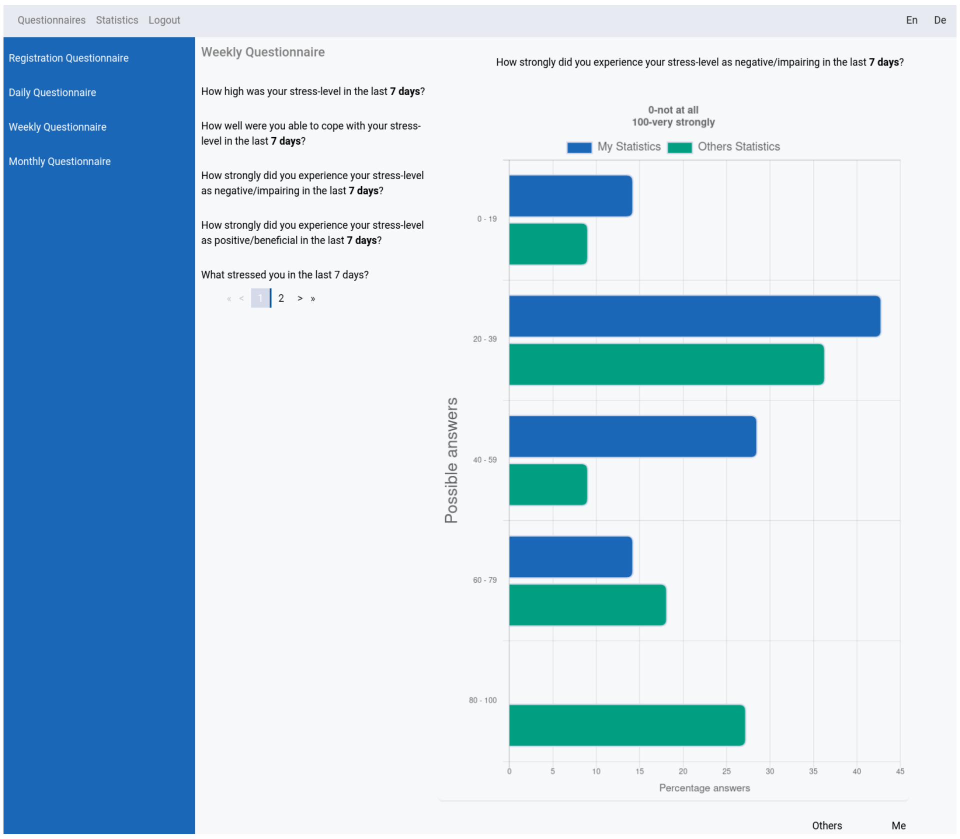
How strongly did you experience your stress-level as negative/impairing in the last **7 days**?

How strongly did you experience your stress-level as positive/beneficial in the last **7 days**?

What stressed you in the last 7 days?

What stressed you in the last 7 days?

Figure 7.4: Logged in user statistic



Weekly Questionnaire

How high was your stress-level in the last **7 days**?

How well were you able to cope with your stress-level in the last **7 days**?

How strongly did you experience your stress-level as negative/impairing in the last **7 days**?

How strongly did you experience your stress-level as positive/beneficial in the last **7 days**?

What stressed you in the last 7 days?

How strongly did you experience your stress-level as negative/impairing in the last **7 days**?

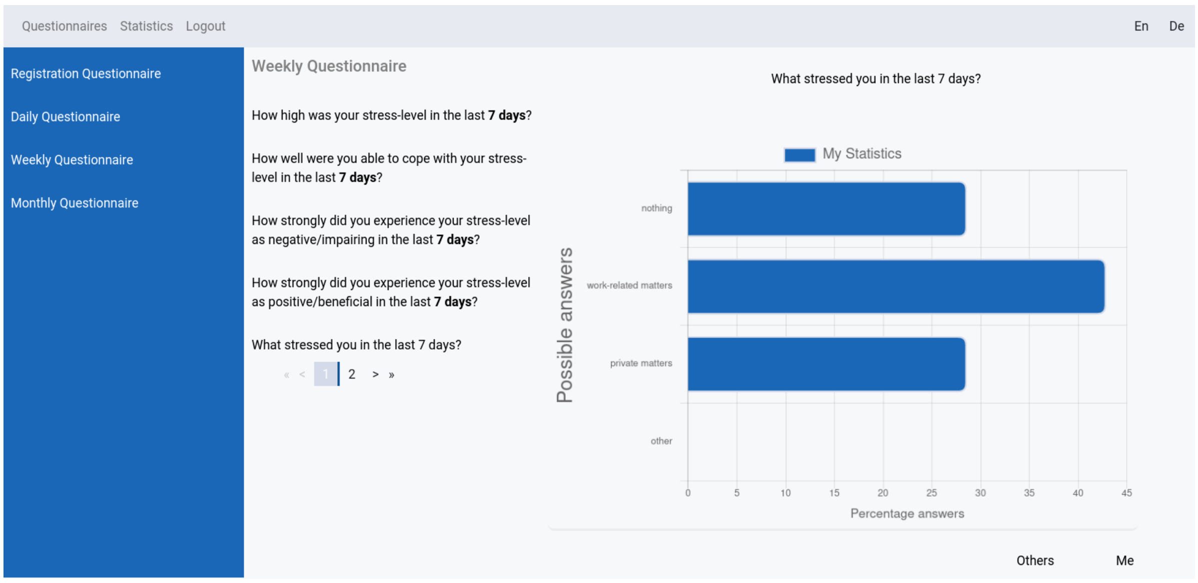
Figure 7.5: All users statistic

Horizontal Bar Chart

The statistic related to a specific question is displayed in the right column. The data is displayed on a horizontal bar chart. Some reasons for using this type of chart are:

- easy to read, the text is displayed horizontally,
- the labels are the possible answers to a specific question, so the bars are a continuation of the answers,
- some of the labels are very large, they are displayed horizontal, so the entire text is shown.

The chart displays the user statistic by default. The user can see it data in the context of the entire population by clicking on a button. The horizontal bar chart is shown in the figure 7.6.



Weekly Questionnaire

How high was your stress-level in the last **7 days**?

How well were you able to cope with your stress-level in the last **7 days**?

How strongly did you experience your stress-level as negative/impairing in the last **7 days**?

How strongly did you experience your stress-level as positive/beneficial in the last **7 days**?

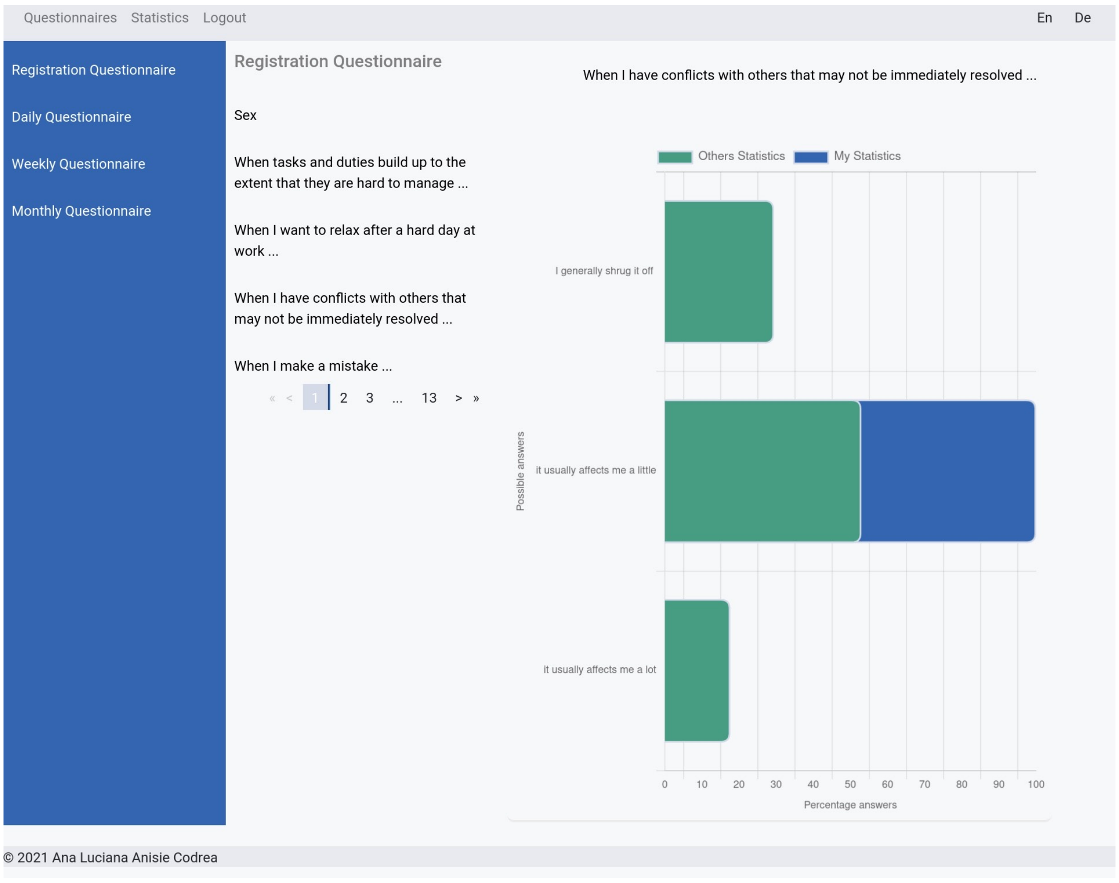
What stressed you in the last 7 days?

What stressed you in the last 7 days?

Figure 7.6: Horizontal bar chart

Registation Questionnaire

The statistic for the registration questionnaire is displayed slightly differently. The user answers this questionnaire only once and gives one answer per question. The data is shown in a stacked horizontal bar chart. The focus of the chart is to compare the totals and the part of the totals (the user answer). The data is easy to read on a stacked chart. The statistic page for the registration questionnaire is shown in the figure 7.7.



Registration Questionnaire

Sex

When tasks and duties build up to the extent that they are hard to manage ...

When I want to relax after a hard day at work ...

When I have conflicts with others that may not be immediately resolved ...

When I make a mistake ...

« < 1 2 3 ... 13 > »

When I have conflicts with others that may not be immediately resolved ...

I generally shrug it off

it usually affects me a little

it usually affects me a lot

0 10 20 30 40 50 60 70 80 90 100

© 2021 Ana Luciana Anisie Codrea

Figure 7.7: Statistic page registation questionnaire

7.4.5 Colors

To make the website familiar to the users the main color used is the same as the one used in the navigation bar in the trackyourstress.org website. This color is used for the **side menu**, and the user statistic on the bar chart.

7.4.6 Interaction

A good interaction design provides feedback to the user. Feedback communicates the results of any interaction. The links and buttons are changing color on hovering to tell that they are clickable. In case of errors or no information available, the system will prompt an informative message.

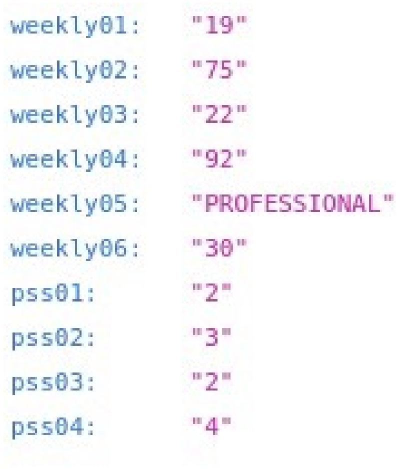
8 Statistic

This chapter describes in detail how the statistic is calculated. The processing of the statistic is done in the **User Controller** part of the application, in the statistics controller.

8.1 Answersheet

The controller requests the answers of all users to the database. The answersheet is parsed into a key-value pair object (dictionary). The key is the question label and the value is the user given answered to that question.

The parsed answersheet for the weekly questionnaire is shown in the figure below.



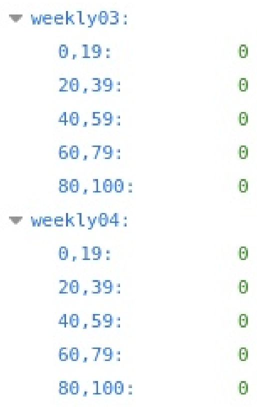
```
weekly01:   "19"
weekly02:   "75"
weekly03:   "22"
weekly04:   "92"
weekly05:   "PROFESSIONAL"
weekly06:   "30"
pss01:      "2"
pss02:      "3"
pss03:      "2"
pss04:      "4"
```

Figure 8.1: Parsed answersheet

8.2 Histogram

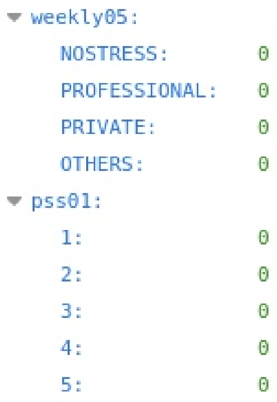
A histogram is created for each question type. The histogram is a counter for each question and possible answer. It is a key-value pair object (dictionary). The key represents a possible answer for a given question, and the initial value is set to zero. For the slider question type, the histogram divides the answers into five ranges. The possible answers for the slider question typo are between 0 and 100. The histograms are assigned to the corresponding question.

The histograms for weekly questionnaire are presented in the figure 8.2.



```
▼ weekly03:
    0,19:       0
    20,39:      0
    40,59:      0
    60,79:      0
    80,100:     0
▼ weekly04:
    0,19:       0
    20,39:      0
    40,59:      0
    60,79:      0
    80,100:     0
```

(a) Slider



```
▼ weekly05:
    NOSTRESS:       0
    PROFESSIONAL:   0
    PRIVATE:        0
    OTHERS:         0
▼ pss01:
    1:              0
    2:              0
    3:              0
    4:              0
    5:              0
```

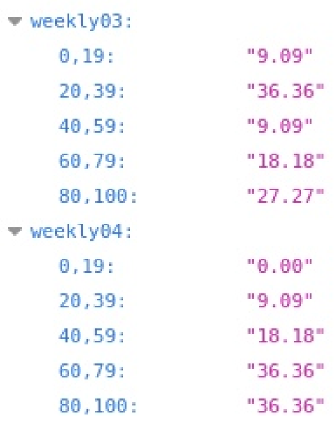
(b) Multiple and Single choice

Figure 8.2: Histograms

The initial value in the histograms is incremented based on the users' answers. For all answers in the answersheet, the corresponding value in the histogram is incremented. In the end, each histogram has the answers count for each question and each possible answer.

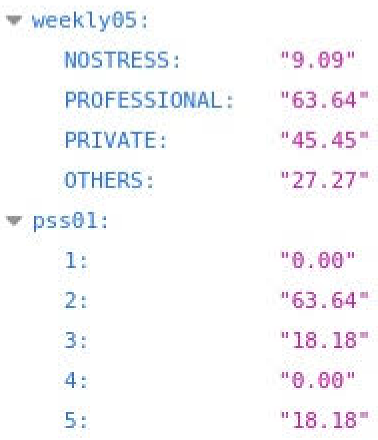
The last step is to calculated and add the percentage for each histogram. The percentage is calculated based on how many users answered a specific questionnaire. For each possible answer, the percentage is shown.

In the end, the histogram for the weekly questionnaire will look like in the figure 8.3.



```
▼ weekly03:
    0,19:       "9.09"
    20,39:      "36.36"
    40,59:      "9.09"
    60,79:      "18.18"
    80,100:     "27.27"
▼ weekly04:
    0,19:       "0.00"
    20,39:      "9.09"
    40,59:      "18.18"
    60,79:      "36.36"
    80,100:     "36.36"
```

(a) Slider



```
▼ weekly05:
    NOSTRESS:       "9.09"
    PROFESSIONAL:   "63.64"
    PRIVATE:        "45.45"
    OTHERS:         "27.27"
▼ pss01:
    1:              "0.00"
    2:              "63.64"
    3:              "18.18"
    4:              "0.00"
    5:              "18.18"
```

(b) Multiple and Single choice

Figure 8.3: Final histograms

9 Testing and Documentation

This chapter focuses on testing and code documentation of the project. Before getting into details, the definition and the importance of testing are explained.

9.1 Software Testing

Software testing is the process of evaluating the functionality of a software application. The main goal is to test if the application functions as expected and to identify the defects.

Testing is a part of the programming phase. It is a very important part of software development due to several reasons [9]:

- errors and bugs can be found in early stages,
- unit testing leads to a better design,
- testing allows sustainable project growth.

There are three level of testing: **end to end testing**, **integration testing** and **unit testing**, [2].

The **end to end testing** tests the fully integrated application. This type of test is the least frequent.

The **integration tests** validates how the different pieces of the application work together. The individual software modules are combined and tested as a group.

The **unit testing** is a way of testing an individual piece of code. The goal of the unit tests is to verify a small unit of code, quickly and in an isolated manner. The unit tests are the most frequent tests. They are used to ensure that the building blocks of the software work independently from each other [9].

This project uses unit tests to test small pieces of code. The tests are divided into backend and frontend testing, which are described below.

9.1.1 Backend-Testing

The backend testing is done using **Jest Testing Framework** for JavaScript unit testing. **Jest** is a node.js package and does not need extra configuration. It is very fast due to the parallelization of the tests.

The tests isolation is done by replacing the dependencies of the system under test with test doubles [9]. The backend application contains tests for the **Questionnaire controller**, the **Statistic controller**, and for the **Routes**. A test class exists for each of these files – and tests most of the methods. The JSON data used for testing is declared in a separate folder.

An example of a unit test used in the **Statistic controller** is presented below.

```

test("create all Histograms" , () => {
  var qqStructure = require('./testJsonData/qstructure.json');
  var qallHisto = require('./testJsonData/allHistograms.json');
  var qtypes = require('./testJsonData/types.json');
  expect(stats.createAllHistograms(qqStructure)).toEqual([qallHisto, qtypes])
});
```

The example test, tests the *createAllHistograms()* function that created all the histograms. The first variable is the questionnaire structure, which is the parameter passed into the function. The next two variables are the returning object. The function should return an array with the histograms, and the question type for all questions from the questionnaire structure.

The **Routes** file is also tested. The HTTP calls are tested using the **SuperTest** module. The **SuperTest** provides a high-level abstraction for testing HTTP requests.

An example of a unit test used in the routes files is presented below.

```

test("get statistics for questionnaire id = 2", done ↩
    => {
  request(appExpress)
    .get("/statistics/2/6b86b273ff34fce19d6b804eff5a
    3f5747ada4eaa22f1d49c01e52ddb7875b4b")
    .expect(200, done);
});
```

The example test tests the statistic route. It makes an HTTP GET request using the **resquest (SuperTest)** variable. The statistic for the given questionnaire id is requested, and the user token is also attached to the request URL. Once the response is received, the expected status code is 200.

9.1.2 Frontend-Testing

The frontend application in this project is an Angular application, built from components, so the place to start testing is components tests.

Each component in Angular has a *.spec.ts* file to unit test the source file. The tests are created using the **Jasmine** framework and run with the **Karma** test runner.

Karma is a JavaScript command-line tool used to open a browser, load an application, and execute tests. It is used to run tests using the **Jasmine** framework, which is a behavior-driven development framework for testing JavaScript code [17].

An example of a test case used in angular *side-menu.comment.spec.ts* is presented below.

```
   it('button should display  Daily', () => {
    let q = new QuestionnaireType("2","Daily");
    let questionnaire: QuestionnaireTypeList = new ↩
       QuestionnaireTypeList ();
    questionnaire.data.push(q);
    component.questionnaires = questionnaire;
    fixture.detectChanges();
    let buttonLoop = fixture.debugElement.↩
       nativeElement.querySelector('button');
    expect(buttonLoop.textContent).toContain('Daily')
 });
```

In the test above, the buttons created for the side menu component are tested. First, the object containing the questionnaire data, and the array list storing these objects are created. After having the data needed to display on the button, the list in the component has to be assigned to the new list that has been created in the test. The component has to trigger the changes using the **detectChanges()** function. Once the component has noticed the change, the assertion is that the button is created, and contains the correct label.

This example shows just how Angular testing works, and each component has its own test file. In these test files, everything can be tested, the component class, the interactions with the DOM, and with other components.

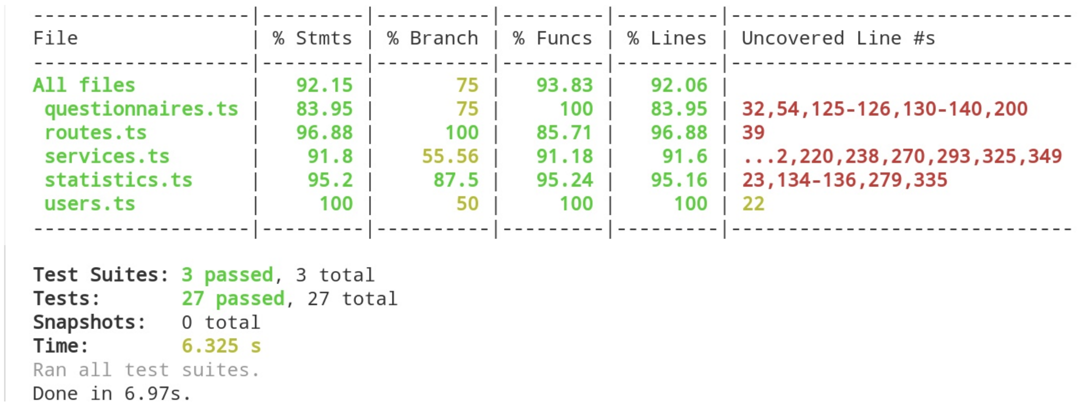
9.1.3 Test Coverage

The **Test Coverage** is a metric that reflects how much of the code is exercised by running the test. The coverage is presented in percentage of code covered in the test suites. The report contains four measurements of coverage.

- **Statement coverage** reflects how many statements from the code run in the tests,
- **Branch coverage** reflects how many executions paths the test went through considering the total possible number of paths,
- **Function coverage** reflects how many functions run out of the total that exists in the code,
- **Line coverage** reflects how many lines of code the test executes.

The most important type of coverage is branch coverage. It indicates if all the possible branches from each decision point are executed. The branch coverage shows if each possible choice that the code can make is validated [2].

As an example for the test coverage, the test coverage report for the backend application is presented in the figure below.



```
-------------------|---------|----------|---------|---------|-----------------------------
File               | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------|---------|----------|---------|---------|-----------------------------
All files          |   92.15 |       75 |   93.83 |   92.06 |
 questionnaires.ts |   83.95 |       75 |     100 |   83.95 | 32,54,125-126,130-140,200
 routes.ts         |   96.88 |      100 |   85.71 |   96.88 | 39
 services.ts       |    91.8 |    55.56 |   91.18 |    91.6 | ...2,220,238,270,293,325,349
 statistics.ts     |    95.2 |     87.5 |   95.24 |   95.16 | 23,134-136,279,335
 users.ts          |     100 |       50 |     100 |     100 | 22
-------------------|---------|----------|---------|---------|-----------------------------

Test Suites: 3 passed, 3 total
Tests:       27 passed, 27 total
Snapshots:   0 total
Time:        6.325 s
Ran all test suites.
Done in 6.97s.
```

Figure 9.1: Backend-tests coverage report

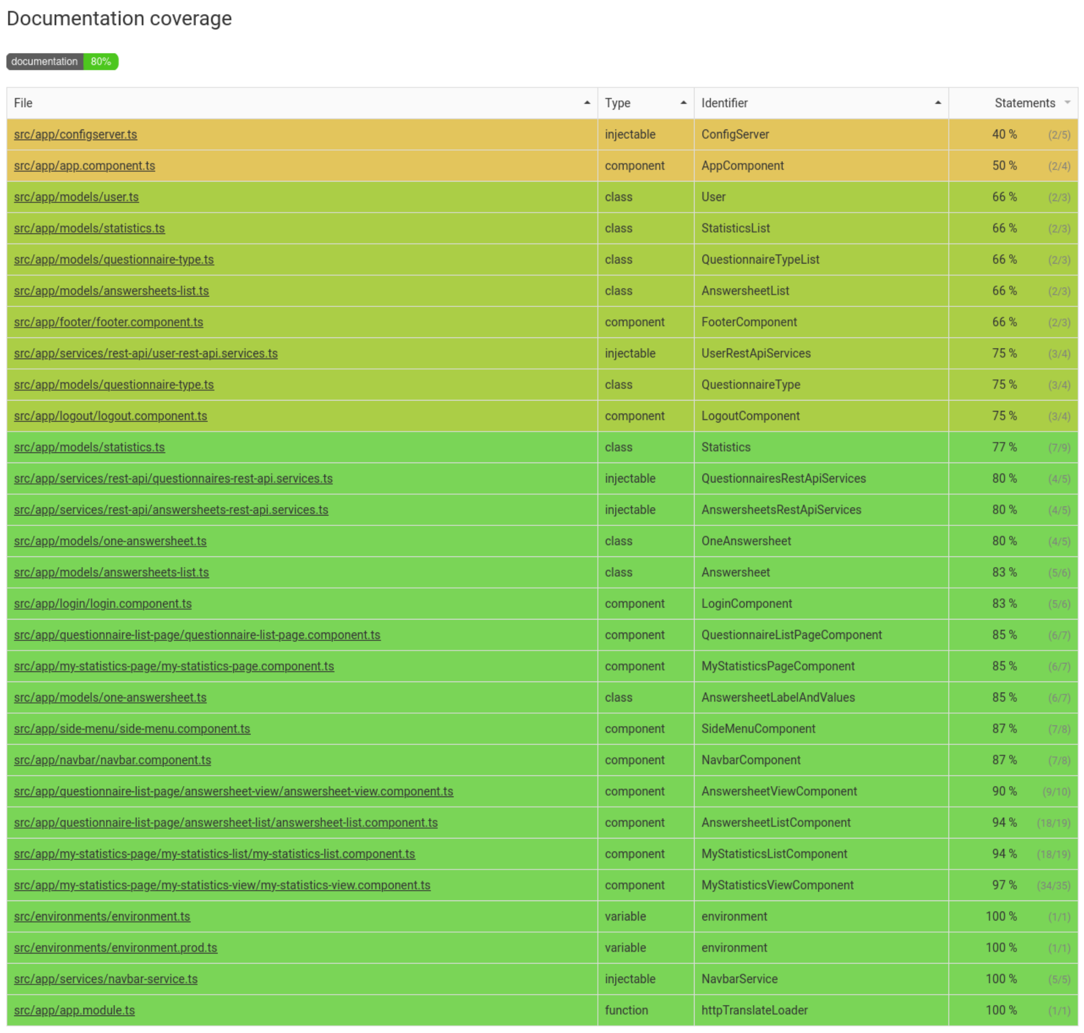
9.2 Code Documentation

Code documentation is a very important part of the development due to several reasons:

- the code and the project is easily maintainable,
- easy for all parties involved to read, change and improve the code,
- helps others to understand the code without reading it.

There are many tools for generating documentation. They generate static documentation for software applications. In this project, for the frontend application They generate a web API that contains different details about the project. In this project, the **Compdoc** documentation tool is used for the frontend application. Compdoc is an open-source tool for documenting Angular applications.

The documentation coverage for the frontend application is presented in the image below.



Documentation coverage

documentation 80%

File	Type	Identifier	Statements	
src/app/configserver.ts	injectable	ConfigServer	40 %	(2/5)
src/app/app.component.ts	component	AppComponent	50 %	(2/4)
src/app/models/user.ts	class	User	66 %	(2/3)
src/app/models/statistics.ts	class	StatisticsList	66 %	(2/3)
src/app/models/questionnaire-type.ts	class	QuestionnaireTypeList	66 %	(2/3)
src/app/models/answersheets-list.ts	class	AnswersheetList	66 %	(2/3)
src/app/footer/footer.component.ts	component	FooterComponent	66 %	(2/3)
src/app/services/rest-api/user-rest-api.services.ts	injectable	UserRestApiServices	75 %	(3/4)
src/app/models/questionnaire-type.ts	class	QuestionnaireType	75 %	(3/4)
src/app/logout/logout.component.ts	component	LogoutComponent	75 %	(3/4)
src/app/models/statistics.ts	class	Statistics	77 %	(7/9)
src/app/services/rest-api/questionnaires-rest-api.services.ts	injectable	QuestionnairesRestApiServices	80 %	(4/5)
src/app/services/rest-api/answersheets-rest-api.services.ts	injectable	AnswersheetsRestApiServices	80 %	(4/5)
src/app/models/one-answersheet.ts	class	OneAnswersheet	80 %	(4/5)
src/app/models/answersheets-list.ts	class	Answersheet	83 %	(5/6)
src/app/login/login.component.ts	component	LoginComponent	83 %	(5/6)
src/app/questionnaire-list-page/questionnaire-list-page.component.ts	component	QuestionnaireListPageComponent	85 %	(6/7)
src/app/my-statistics-page/my-statistics-page.component.ts	component	MyStatisticsPageComponent	85 %	(6/7)
src/app/models/one-answersheet.ts	class	AnswersheetLabelAndValues	85 %	(6/7)
src/app/side-menu/side-menu.component.ts	component	SideMenuComponent	87 %	(7/8)
src/app/navbar/navbar.component.ts	component	NavbarComponent	87 %	(7/8)
src/app/questionnaire-list-page/answersheet-view/answersheet-view.component.ts	component	AnswersheetViewComponent	90 %	(9/10)
src/app/questionnaire-list-page/answersheet-list/answersheet-list.component.ts	component	AnswersheetListComponent	94 %	(18/19)
src/app/my-statistics-page/my-statistics-list/my-statistics-list.component.ts	component	MyStatisticsListComponent	94 %	(18/19)
src/app/my-statistics-page/my-statistics-view/my-statistics-view.component.ts	component	MyStatisticsViewComponent	97 %	(34/35)
src/environments/environment.ts	variable	environment	100 %	(1/1)
src/environments/environment.prod.ts	variable	environment	100 %	(1/1)
src/app/services/navbar-service.ts	injectable	NavbarService	100 %	(5/5)
src/app/app.module.ts	function	httpTranslateLoader	100 %	(1/1)

Figure 9.2: Frontend documentation coverage

10 Conclusions

The purpose of this project was to deliver an interactive and feedback-oriented web application for patients suffering from stress.

The web application should offer to users the possibility to view questionnaires they answered and view statistics related to those questionnaires. From the beginning, the focus of the project was on the end-user. This helped to develop a familiar user interface as explained in the **User interface** chapter. The user interface is, in the end, the part of the project with which the end-user interacts.

Stress is a serious problem in modern society. Tracking the stress, and monitoring the stress levels can help people protect their health in the long term.

This web platform can help people suffering from stress to better understand what causes their stress, and how their stress changes over time. The statistic shows some of the reasons for stress. It is also interesting to see how the user's stress looks comparing with the global population. All this information can help to get a better diagnose or treatment.

In the future, new features could be implemented. For example, showing to the user how their stress level changed over a period of time, or how their answers have changed over the time for the same question.

This could help to detect new stressors or to see if some stress treatment helped to improve their stress situation.

Bibliography

[1] Brian Totty; Sailu Reddy; David Gourley; Marjorie Sayer; Anshu Aggarwal. *HTTP: The Definitive Guide*. : O'Reilly Media, Inc, 2002.

[2] Lucas Fernandes da Costa. *Testing JavaScript Applications*. : Manning Publications, 2021.

[3] Guibert Ulric Crevecoeur. *A system approach to the General Adaptation Syndrome*. `https://www.researchgate.net/publication/305609884_A_system_approach_to_the_General_Adaptation_Syndrome`. [Online; accessed 13-January-2021]. 2016.

[4] George Fink. *Stress: Concepts, Definition and History*. : Elsevier Inc., 2017.

[5] Ido Flatow; Gil Fink. *Pro Single Page Application Development: Using Backbone.js and ASP.NET*. : Apress, 2014.

[6] Adam Freeman. *Pro Angular 9: Build Powerful and Dynamic Web Apps*. : Apress, 2020.

[7] Jesse James Garrett. *The Elements of User Experience, Second Edition: User-Centered Design for the Web and Beyond*. : New Riders, 2010.

[8] James Hibbard; Craig Buckler; Paul Orac; Mark Brown; M. David Green; Florian Rappl; James Kolce; Nilson Jacques. *Your First Week With Node.js, 2nd Edition*. : SitePoint, 2020.

[9] Vladimir Khorikov. *Unit Testing Principles, Practices, and Patterns*. : Manning Publications, 2020.

[10] Jörg Knappen. *Schnell ans Ziel mit LATEX 2e*. 3., überarb. Aufl. München: Oldenbourg, 2009.

[11] Rüdiger Pryss; Dennis John; Winfried Schlee; Wolff Schlotz; Johannes Schobel; Robin Kraft; Myra Spiliopoulou; Berthold Langguth; Manfred Reichert; Teresa O'Rourke; Henning Peters; Christoph Pieh; Claas Lahmann and Thomas Probst. *Exploring the Time Trend of Stress Levels While Using the Crowdsensing Mobile Health Platform, TrackYourStress, and the Influence of Perceived Stress Reactivity: Ecological Momentary Assessment Pilot Study*. [Online; accessed July 2021]. 2019.

[12] Rüdiger Pryss; Dennis John; Manfred Reichert; Burkhard Hoppenstedt; Lukas Schmid; Winfried Schlee; Myra Spiliopoulou; Johannes Schobel; Robin Kraft; Marc Schickler; Berthold Langguth and Thomas Probst. *Machine Learning Findings on Geospatial Data of Users from the TrackYourStress mHealth Crowdsensing Platform*. [Online; accessed July 2021]. 2019.

[13] Neil Madden. *API Security in Action*. :Manning Publications, 2020.

[14] Micah Martin; Robert C. Martin. *Agile Principles, Patterns, and Practices in C sharp*. : Pearson, 2006.

[15] Frank Mittelbach, Michel Goossens, and Johannes Braams. *Der Latex-Begleiter*. 2., überarb. und erw. Aufl. ST - Scientific tools. München [u.a.]: Pearson Studium, 2005.

[16] Fernando Monteiro. *Learning Single-page Web Application Development*. : Packt Publishing, 2014.

[17] Harmeet Singh Ravi Kumar Gupta Hetal Prajapati. *Test-Driven JavaScript Development*. : Packt Publishing, 2015.

[18] Neal Ford; Mark Richards. *Fundamentals of Software Architecture*. : O'Reilly Media, Inc., 2020.

[19] Joachim Schlosser. *Wissenschaftliche Arbeiten schreiben mit LATEX : Leitfaden für Einsteiger*. 5., überarb. Aufl. Frechen: mitp, 2014.

[20] Robert C. Martin Series. *Cleas Architecture A Craftsman's Guide to Software Structure and Design*. : Prentice Hall, 2018.

[21] Peter Späth. *Beginning Java MVC 1.0: Model View Controller Development to Build Web, Cloud, and Microservices Applications*. : Apress, 2020.

[22] Thomas F. Sturm. *LATEX : Einführung in das Textsatzsystem*. 9., unveränd. Aufl. RRZN-Handbuch. Hannover [u.a.]: Regionales Rechenzentrum für Niedersachsen, RRZN, 2012.

[23] Doguhan Uluca. *Angular for Enterprise-Ready Web Applications - Second Edition*. : Packt Publishing, 2020.

[24] Herbert Voß. *LaTeX Referenz*. 2., überarb. u. erw. Aufl. Berlin: Lehmanns Media, 2010.

[25] Hage Yaapa. *Express Web Application Development*. : Packt Publishing, 2013.

[26] Anton Moiseev Yakov Fain. *TypeScript Quickly*. : Manning Publications, 2020.

[27] Simon Holmes; clive harber. *Getting MEAN with Mongo, Express, Angular, and Node, Second Edition*. : Manning Publications, 2019.

[28] stackoverflow. *Stack Overflow Survey*. `https://insights.stackoverflow.com/survey/2020`. [Online; accessed 11-January-2021]. 2020.

Name: Ana Luciana Anisie Coderea Matrikelnummer: 2000527

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 26. 07. 2021 .... [illegible]

Ana Luciana Anisie Coderea