Tina Promitzer, BSc

Concept and Visualization of News Articles and the Registration Process for a Soccer Platform

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany

Institute for Softwaretechnology
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Wolfgang Slany

Graz, February 2021

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Signiert von: Tina Promitzer

Datum: 28.02.2021 11:00:49

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Dieses Dokument ist digital signiert!

Prüfinformation:
Informationen zur Prüfung der elektronischen Signatur finden Sie unter:
www.handy-signatur.at

www.a-trust.at

TRUST
einfach sicher

Date, Signature

Abstract

Nowadays, there are many different approaches for developing software or a product before it can be delivered to the customer. The development of software usually consists of different phases since software development is a step-by-step process. These phases start with the planning of the product and end with the finalization of the product. The phases that software has to go through are called the Software Development Life Cycle (SDLC). It is crucial to choose an appropriate SDLC to make the desired product development clear and make complex software maintainable. Besides, the appropriate SDLC ensures an optimal implementation of the individual phases. Moreover, including the wishes and requirements of potential users is essential when developing software to create an ideal product that users accept and is adapted to their needs.

Therefore, we discuss different phases of possible SDLCs and choose an appropriate one in the further course of this work to create a concept and implement essential parts of a web application for a soccer platform. These essential parts are the two main elements of the web application that is currently under construction. One of these elements is the primary means of communication between soccer clubs and their fans. The soccer platform utilizes so-called news articles as main communication medium. During our thesis, we chose an appropriate SDLC to visualize these articles. The second feature is the registration process that sets this platform apart from existing ones on the domestic market since this platform allows fans to create an account. As a result, they can follow their favorite teams and have the option to create an individual area with the latest news from their favorite clubs. In order to create an ideal product that meets the requirements of the potential user, the wishes and requirements are identified, communicated, and documented with the help of Requirement Engineering (RE).

Kurzfassung

Heutzutage gibt es sehr viel verschiedene Wege eine Software oder ein Produkt zu entwickeln bevor es dem Kunden übergeben werden kann. Die Entwicklung einer Software besteht normalerweise aus verschiedenen Phasen, die mit der Planung des Produktes starten und der tatsächlichen Fertigstellung des End-Produktes enden, da die Entwicklung einer Software ein schrittweiser Prozess ist. Die Phasen, die eine Software durchlaufen muss, nennt man Software Development Life Cycle. Die Wahl des geeigneten SDLC ist wichtig um die Entwicklung des gewünschten Produktes übersichtlich zu gestalten und komplexe Software wartbar zu machen. Außerdem wird dadurch eine optimale Umsetzung der einzelnen Phasen gewährleistet. Außerdem ist es wichtig die Wünsche und Anforderungen der potentiellen User miteinzubeziehen um ein ideales Produkt zu erschaffen, welches die User akzeptieren und an ihre Wünsche angepasst ist.

Aus diesem Grund werden im Zuge dieser Masterarbeit verschiedene Phasen von möglichen Software Development Life Cycles theoretisch gegenübergestellt und ein geeigneter SDLC ausgewählt um ein Konzept und die Implementierung von essentiellen Teilen einer Webanwendung für eine Fußball-Plattform zu ermöglichen. Es handelt sich um zwei Hauptelemente der Webanwendung, die sich momentan noch im Aufbau befindet. Eines dieser Elemente ist das Hauptkommunikationsmittel der Fußballvereine mit ihren Fans. Dabei handelt es sich um die Newsartikel, für die eine entsprechende Visualisierung gefunden werden soll. Das zweite Feature ist der Registrierungsprozess durch den sich diese Plattform von bereits existierenden auf dem heimischen Markt abhebt. Denn diese Plattform ermöglicht es erstmals, dass sich Fans registrieren können und in Folge dessen ihren Lieblingsmannschaften folgen können. Dadurch wird ihnen die Option geboten einen individuellen Bereich zu schaffen mit den aktuellsten

Neuigkeiten ihrer Lieblingsvereine. Um ein ideales Produkt zu erschaffen, dass den Anforderungen der potentiellen User entspricht, werden die Wünsche und Anforderungen mithilfe von RE identifiziert, kommuniziert und dokumentiert.

Contents

List of Figures

Acronyms

API	Application Programming Interface	6
CLI	Command Line Interface	19
CSS	Cascading Style Sheets	21
DI	Dependency Injection	23
DOM	Document Object Model	16
HTTP	Hypertext Transfer Protocol	49
HTML	Hypertext Markup Language	12
ID	Identifier	25
IDE	Integrated Development Environment	75
JS	JavaScript	19
JSON	JavaScript Object Notation	48
RE	Requirement Engineering	iii
REST	Representational State Transfer	6
SDLC	Software Development Life Cycle	iii
SPA	Single Page Application	25
UI	User Interface	6
UML	Unified Modeling Language	viii
URL	Uniform Resource Locator	25

1 Introduction

This section explains the initial situation and the motivation behind this thesis. Moreover, it describes the objectives and goals that should be solved in this work's further course. Afterwards, the last paragraph provides an overview of this master thesis.

1.1 Motivation

This master thesis's focus is the creation of a concept and the implementation of parts of a soccer platform. A platform that is still at the starting point of development and of getting successful. An interdisciplinary collaboration between developers, representatives from sales, potential users, and the product owner is essential for this development. This platform specializes in Austrian Amateur Soccer, intending to create an ideal product for the more than 2000 soccer teams of the Austrian Soccer Association (ÖFB) and all their fans. The soccer platform enables soccer clubs to publish their content and, thus, reach their fans. Moreover, the platform provides game results, a live ticker, and all kinds of information about the soccer teams to their fans.

For developing an ideal product, the integration of user requirements and user feedback is crucial, and that is why the platform is working with the soccer clubs and their fans as partners. Moreover, it should be possible for the soccer teams to publish contributions on several other platforms with just one click to reach as many fans as possible. Publishing contributions with just one click means that the soccer clubs can communicate their content on our applications, which already exist, on the website, which is under development, and on the clubs' related social media sites, like Facebook, simultaneously. Furthermore, these contributions also get published on the paper's editing website with whom the soccer platform works. This "one-click publication" should be made possible by developing a web application offering homepages for the teams to publish their content. The soccer platform arranges about spreading the content on the other mentioned platforms. Therefore, the soccer teams do not have to have the technical know-how to reach as many people as possible.

Such contributions include news articles written by the soccer clubs themselves or by the paper's editorial staff with whom the soccer platform works. These news articles are the soccer team's communication medium to reach all their existing and potential fans. Therefore, the news articles are one of the most crucial elements of this platform. Besides the "one-click publishing process", this soccer platform has another outstanding attribute, which protrudes this platform from the few others on the domestic market. This attribute or feature is the possibility to create an account for fans and, therefore, create a personal area with the latest news of someone's favorite soccer teams.

Currently, the entire content of this soccer platform is only accessible on the native applications for the mobile operating systems Android and iOS. However, a web application is needed to provide the "one-click publishing mechanism", as mentioned before. Furthermore, an additional web application brings the advantage of addressing a larger target group. The content will be accessible on each device regardless of whether a user is using a smartphone, a tablet, or a computer.

As a result, the following problems can be identified from the current situation. As mentioned before, the soccer team's primary communication means are the news articles. Therefore, the concept and visualization of these news articles are needed. Furthermore, the feature, which lets this platform differ from the few others, should be implemented as well. Therefore, the possibility of creating an account and consequently getting the option to create one's personal area should be realized. Therefore, a registration process with all necessary components is needed.

In summary, the two derived problems are:

- The need of a representation for the news articles.
- For providing the creation of an individual area, a registration process is needed.

1.2 Goals and Objectives

This soccer platform needs a web application, and essential parts of this application should be developed in the further course of this work. Therefore, this master thesis aims to create the concept and implementation of these crucial parts: the news articles and the registration process. These elements are explained in the following paragraphs.

1.2.1 Goal 1: Creation of the concept and the visualization of news articles

As mentioned in Section 1.1, the soccer team's primary communication means are the news articles. Therefore, the concept and visualization of these news articles are needed.

The news articles should be accessible at different locations on the website. This platform offers a homepage to the clubs that they can manage themselves without needing the necessary technical know-how, as described in Section 1.1. This homepage, or let us call it the "team page", is one location where the news articles written by that team should be accessible. Furthermore, there will also be "league pages". Each league has an own homepage that provides the news articles that correspond to that specific league. Finally, there will be an area where all news articles will be presented, regardless of which team has published them or what league they correspond to. Besides the different locations, there are also different representations needed. On the one hand, the articles should be visualized in a sort of news feed to get an overview of several articles available. On the other hand, they need a detail view representation for showing the entire article. An essential component of the news articles is a picture gallery to display the images attached to the article. Besides, the fans should get the possibility to share these articles on other platforms. Because of that, a sharing feature for these articles should be developed as well.

1.2.2 Goal 2: Implementation of a registration process with all necessary components

In contrast to the few on the domestic market, an outstanding attribute of this soccer platform is the option for creating an individual area, mentioned in Section 1.1. The fans should be allowed to get all the news associated with their favorite soccer teams. For providing this feature, the fans must have the option to create an account on this platform. Therefore, a registration process with all necessary components is needed.

These components are the registration, the login, the handling if the user forgot their password or their email address, the handling for changing the users' data, and the option for contacting the support team. Furthermore, the platform should provide login via other social media platforms like signing in with Facebook and Apple.

Objectives

All these features go through a process with several phases. These phases are the analysis, the conception of the implementation and design, the implementation itself, and the testing and evaluation process.

1. Analysis:

During the analysis phase, the requirements have to be determined and documented. Furthermore, the requirements must incorporate the desires and feedback of the users. After the specification of the needs, other websites and social media platforms get examined and analyzed to derive the conception from the requirements.

2. Concept:

The conception follows after the analysis phase. This phase consists of the consideration and definition of the details that are needed for the implementation. Furthermore, this stage also consists of the creation of mock-ups. The mock-ups have to consider the different screen sizes because the application has to be developed for mobile phones, tablets, and desktops.

3. Implementation:

This phase contains the realization regarding the definitions of the details that were determined in the conception.

4. Feedback:

This stage consists of the testing and evaluation phase of the implemented features. The features get tested and evaluated before they can be incorporated into the platform. They have to fulfill the functionality. Furthermore, this phase enables suggestions for improvement by the testers.

1.3 Thesis Overview

First of all, Section 2 starts with the literature review, including the theoretical aspects needed to create the concept and the implementation of the goals described before. The literature review includes background knowledge about SDLC and RE. SDLC describes the process software has to run through from the very beginning until the software is finished and can be delivered to users. RE describes the process of gathering, modeling, and documenting requirements for a system. Furthermore, Section 2 also contains a description of requirements and suggestions of how to divide them to define what the system should do and how the system should implement them. Since the used technology for the implementation part is Angular, the literature review also contains a description of this framework.

Section 3 explains the used methodology and consists of a description of the entire process necessary to achieve the results and, therefore, reach the goals and objectives. Afterwards, the results are presented in a separate section. Section 4 includes but is not limited to the analysis of the actual situation and the analysis of the requirements. The requirements are divided regarding the theoretical aspects of the literature review. Therefore, they are divided into functional and non-functional requirements. Furthermore, the functional ones are transformed into story cards and prioritized afterwards. The concept part afterwards derives the conception from the defined requirements. This part shows the system architecture of the desired state, along with an overview of the Representational State Transfer (REST) Application Programming Interface (API). Afterwards, the software architecture is described in detail and illustrated with some diagrams. The diagrams and explanations describe how the features should be implemented from a technical point of view. Furthermore, the User Interface (UI) conception is also described and illustrated with some mock-ups. The UI conception is divided into the features that should be implemented in this work's further course. After creating the concept, the implementation of the features follows and describes how the features are implemented using the Angular Framework. The Section 4 ends with comparing the requirements and the implementation to emphasize if the requirements are fulfilled. The thesis ends with Section 5, which is the conclusion including a description of lessons learned. Furthermore, this last section includes some limitations that have to be faced and a Future Work section describing features that have to be considered to be added in the future.

2 Literature Review

This master thesis aims to develop a concept and the implementation of news articles and a registration process, with all essential components, for a soccer platform. For developing a concept, the users' and systems' requirements have to be defined and analyzed first. For this purpose, literature is used that deals with the subject of SDLC and RE. These terms are described in more detail in Sections 2.1 and 2.2. The definition of requirements itself is explained afterwards. Finally, a description of the Angular Framework, which will be used for the development, follows.

2.1 Software Development Life Cycle (SDLC)

In the following paragraphs, we use the definition and explanations of IntroBooks Team, 2020. Software development contains several steps before software, or a product is finished. These steps are called SDLC. The software development process consists of several separate steps because it is impossible to create software all at once. The product gets repeatedly tested by potential customers. Their valuable feedback has to be considered to change the product in a way that it gets integrated. Testing and implementing the wished changes of the product is a repeating process with the aim that the customer likes and approves of the product. This process finishes not until the customer is satisfied with the software. The customers are the most valuable thing for a company since the goal is to create the ideal product that meets the potential users' requirements. The SDLC is the medium to communicate the different phases of software development to us, starting with the conception of the idea of the product up to the final stage of product shutdown.

According to IntroBooks Team, 2020, the SDLC consists of seven stages, which are:

- Planning
- Feasibility Analysis
- Software Design
- Programming
- Implementation and Integration
- Software Test
- Installation and Maintenance

The starting point of SDLC is the planning of a product. All specialists come together and determine how they develop the product, how to record the requirements, and how to analyze all aspects of the future product thoroughly. The requirements of the customers have to be taken into account by the developers. It is essential to know which features the potential user wants and which are unnecessary. Furthermore, the developers have to discuss the likely problems that could arise during the development process. The whole team communicates with each other. In the analysis part, the project team defines the whole Project in detail.

In the second phase, the Feasibility Analysis, the team has to check whether the product is feasible or not. Moreover, the entire project team manages the whole workflow. The workflow itself gets divided into small tasks and assigned to the corresponding team members. Usually, the team consists of testers, designers, developers, and project managers. By splitting the whole flow into smaller tasks, it is easier for each team member to evaluate their functions.

Design is the third and one of the main phases of the SDLC and considers the whole product design. This stage takes into account what hardware and system requirements are needed. Furthermore, the entire system architecture is considered.

The fourth stage, namely the programming phase, is considered the critical phase of the SDLC. Implementing the desired product is done by a team of programmers responsible for this software, putting much effort into it.

The fifth stage of the SDLC is the Implementation and Integration. Usually, software consists of many different programs that have to be implemented carefully. Therefore, the integration must be gradual. Furthermore, the software has to be checked by the project team if it runs on the various available systems. When errors occur, they have to be fixed by testers.

The sixth phase, namely the testing stage, is when the testers come into play after the software implementation is finished. The quality and the performance of the software has to be guaranteed. Therefore, the testers have to inspect the software before it can be passed on to the customer.

The last stage of the SDLC is the installation and maintenance. The software implementation is finished, tested, and approved by testers. Therefore, the product can be deployed or delivered to the customer. By using the product, the customers identify the real problems, and the developers are responsible for starting the maintenance by fixing these problems.

According to Delfmann and Rieke, 2007, the classic phases of a SDLC are:

- RE
- Design
- Implementation
- Testing
- Deployment and Maintenance

As Delfmann and Rieke, 2007 says, the first phase of the SDLC is RE which is discussed in more detail in Section 2.2. This phase combines the Planning and Feasibility Analysis stages, as defined by IntroBooks Team, 2020. Furthermore, Delfmann and Rieke, 2007 also combines the Programming and Integration stage into the Implementation phase.

There are many different approaches of how companies implement such an SDLC.

2.2 Requirement Engineering (RE)

"Requirements engineering is concerned with identifying, modeling, communicating and documenting the requirements for a system, and the contexts in which the system will be used." (Paetsch, Eberlein, and Maurer, 2003)

According to Chakraborty et al., 2012, RE is indispensable regarding software engineering. They describe RE as the most crucial part of the SDLC because it can be very costly if errors occur at this phase and remain undetected.

RE is the process of recognizing, modeling, informing, and documenting the requirements for a system. Furthermore, these requirements describe the contexts of how the system will be used. So, to avoid that the software has to be revised because it is not clear what the system should do before developing it, RE comes into play. (Paetsch, Eberlein, and Maurer, 2003)

In summary, RE is crucial for developing an ideal product that satisfies the potential users instead of developing one past the customer's needs.

2.3 Requirements

The first steps of system development include the specification of what the system should do. According to Sommerville and Sawyer, 1997, this specification describes the systems' behavior, a systems property, or a system's attribute, which is called requirement. As stated by them, requirements could describe different aspects, which are the following ones:

- a facility at the user level
- a general feature of the system
- a specific restriction of the system
- a condition for developing a system

So, these requirements could also be limitations of the system's development process.

The requirements of a system must be documented in some way. Now the question may arise what the best way to document or write such requirements.

According to Sommerville and Sawyer, 1997, the answer to that question is that there is no best way to do so. Sommerville and Sawyer, 1997, describe that the documentation of the requirements depends on notations used by readers and writers of the requirement and their organizational practice.

They could be written in natural language as well as in engineering terms. It depends on who the source of the requirement is. Sommerville and Sawyer, 1997 recommends dividing the requirements into functional and non-functional ones. They say the functional requirements tell what the system should do, and the non-functional ones tell how the functional requirements should be implemented.

To be more precise, the functional requirements describe the system itself or its components and how the behavior in certain situations should be. These requirements are recorded in use cases and are mandatory with a focus on user requirements. (Guru 99, 2021)

The non-functional requirements are concerned with the attributes of the system. These attributes could be the performance, usability, etc. Sommerville and Sawyer, 1997 also say that the quantitative values for system needs should be specified when possible. Paetsch, Eberlein, and Maurer, 2003 share the same opinion as Sommerville and Sawyer, 1997. They say that most non-functional requirements should be specified since these needs may affect the choice of database, programming language, or operating system.

According to Cohn, 2004, there is a good way to describe at least functional requirements. This solution are "user stories". He also says that such stories should represent the users' needs instead of documenting them. That is why these stories are described in writing, and the description should be kept short. Furthermore, he says that the crucial facts will be established in conversations. Therefore, the way in which the user stories are written is intended to give the person reading them straightaway an idea of how a real user wants to use the application and should not tell how this feature should be implemented or any other technical details. For the technical details, the conversations come into the picture. A common and recommended way to write such stories is explained below.

The user stories should contain the following three aspects:

- Persona: It should be clear who the customer of the software to be developed is.
- Requirement: It should describe the customer's intention.
- Purpose: It should describe the benefits the customer derives from it.

(Rehkopf, 2021)

2.4 Angular Framework

Angular[1] is an open-source framework for mainly developing . An SPA consists of a single Hypertext Markup Language (HTML) document, and its content gets dynamically reloaded. Classic web applications, on the other hand, consist of several linked HTML documents. Furthermore, Angular is a client-side framework that can be used in conjunction with server-side web frameworks such as Node.js[2]. Client-side is the software that runs on the web-browser of users while server-side software runs on the server. Besides, Angular is based on JavaScript and HTML.

When someone talks about Angular, they always mean the latest version of the framework. Currently, Angular is available in Version 11. (As of January 2021). The framework is co-developed by the American company Google and has a very large community in the development scene. It is recommended by Angular to use TypeScript as a programming language, as it has many advantages.

Since TypeScript is a superset of the JavaScript programming language, any valid TypeScript code is automatically valid JavaScript code because it is transpiled back to JavaScript. Some advantages of TypeScript over JavaScript are the integration from classes, modules, interfaces, or the optional type system, which provides type safety. (Kasagoni, 2017)

[1]https://angular.io/ [accessed on: 13.01.2021]
[2]https://nodejs.org/en/ [accessed on: 13.01.2021]

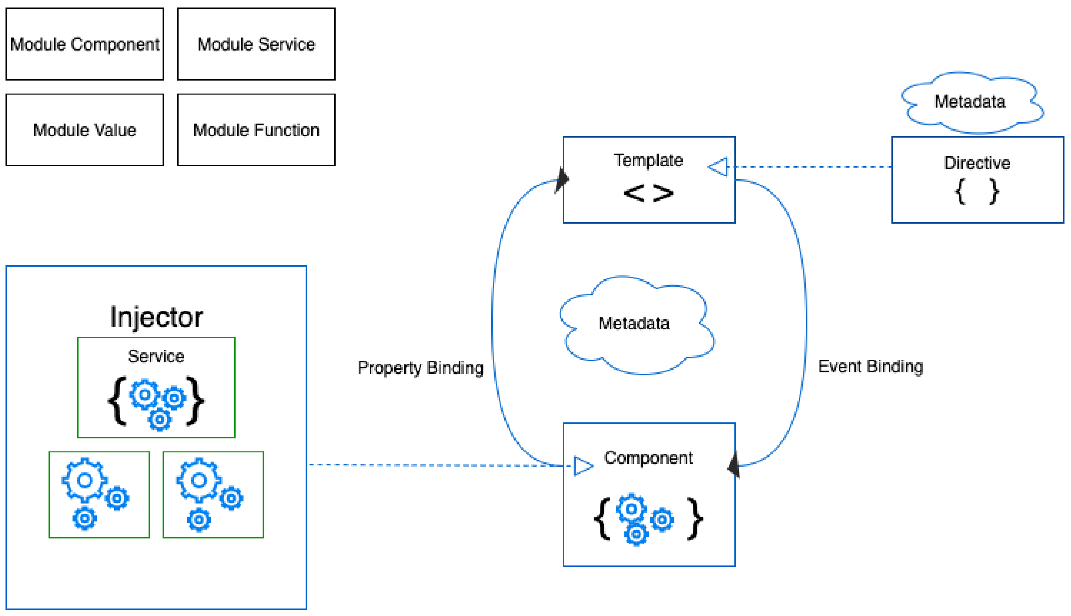


Figure 2.1: This diagram shows the main building blocks of an Angular application and how they are connected. (Angular Team, 2021)

The main building blocks of an Angular application are Components (2.4.2), Templates (2.4.3), Metadata (2.4.4), Directives (2.4.5), Services (2.4.7), Modules (2.4.8), Data binding (2.4.9), and Dependency Injection (2.4.10) as it is shown in Figure 2.1. These building blocks will be explained in detail in the following sections.

2.4.1 Decorators

Decorators are a design pattern for separating modification of a class without modifying the original source code. Therefore, the Angular decorator modifies JavaScript-Classes by storing metadata about a class. Somehow Angular has to know which functionality the classes have to fulfill. Each decorator has a base configuration, and it can be passed through by using the relevant factory. Assuming that a component, explained in Section 2.4.2, gets configured, its decorator provides metadata. This metadata tells Angular that it is a component and has a component-specific configuration. In

Section 2.4.4 the metadata is explained in more detail.

2.4.2 Components

The most basic UI building block of an Angular application are components. A component consists of the template, which contains all HTML and is explained in more detail in Section 2.4.3, the corresponding TypeScript code, which is the component class and defines the logic, and the optionally styling applied to the template. Through an API consisting of properties and methods, the class can interact and support the view. Every Angular application is built up modularly from several components, and each has at least one root component. (Angular Team, 2021)

2.4.3 Templates

As mentioned before, a template is a form of HTML. So, the template is responsible for defining what to display in the browser and tells Angular in which way it should render the corresponding component. On one hand, the template consists of typical HTML elements, like <p >or <h1 >, and, on the other hand of, additional Angular template syntax.(Angular Team, 2021)

An example of Angular template syntax would be <app-list >as shown in listing 2.1. What this element exactly does is explained in Section 2.4.4. Listing 2.1 shows a basic example of how such an Angular template could look.

Listing 2.1: Example of a basic Angular template with common HTML elements and Angular template syntax (Code adapted by Angular Team 2021)

```
<h2>My List</h2>
<ul>
  <li *ngFor="let element of listWithItems">{{element.title}}</li>
</ul>
<app-list></app-list>
```

2.4.4 Metadata

In TypeScript, metadata is appended to the class using a decorator. Listing 2.2, for example, shows how the component metadata is appended using the @Component decorator. Using this decorator, Angular is told what kind of code it is and where the class and template can be found. In addition, it also indicates which services are required. The selector ensures that the component can be identified within the template. An instance of this component is created and added to the corresponding position in the parent template. Let us assume that <app-list ></app-list >as in the Listing 2.1, is included in the HTML of the Angular application, then Angular would create and insert an instance of this ListComponent between these closing tags. The templateUrl tells Angular the location of the HTML template. As mentioned before, it is possible to tell Angular which Services are required. That happens with the "providers" property. It is an array of providers for services.(Angular Team, 2021)

Listing 2.2: Example of a basic component metadata (Code adapted by Angular Team 2021)

```
@Component({
  selector:   'app-list',
  templateUrl: './list.component.html',
  providers: [ ListService ]
})
export class ListComponent implements OnInit {
/* . . . */
}
```

2.4.5 Directives

An Angular directive is a class with the @Directive decorator. Angular transforms the Document Object Model (DOM) during the rendering process according to the respective directives' instructions. Therefore, the appearance or behavior of the DOM is changed.(Angular Team, 2021)

There are exactly three types of directives:

- Components
- Structural directives
- Attribute directives

From a technical point of view, components are directives with a template. Nevertheless, components are so unique and central to Angular applications that the @Component decorator was introduced. It enhances the @Directive decorator with template-oriented functions.

Structural directives are responsible for the HTML layout and change the structure of the DOM, by adding, removing, and replacing elements. These directives are marked with an asterisk. Typical structural directives are *ngIf and *ngFor.(Angular Team, 2021)

In Listing 2.3 are examples of each one.

Listing 2.3: An example of the *ngIf and *ngFor structural directive. (Code adapted by Angular Team 2021)

```
<div *ngIf="showAnimal" class="name">{{animal.name}}</div>
<ul>
  <li *ngFor="let animal of animals">{{animal.name}}</li>
</ul>
```

The *ngIf structural directive takes a boolean as an expression, adds the corresponding element to the DOM when it evaluates to true, or removes it from the DOM when it evaluates to false. According to listing 2.3 the *ngFor structural directive tells Angular to add the <li>HTML element to the DOM for each element in the animals list. (Angular Team, 2021)

2.4.6 Pipes

Pipes are used for transforming data in a user-friendly form before they can be displayed. The pipe gets the data as input and transforms it before the appropriately transformed data is displayed to the user. Angular provides built-in pipes, such as DatePipe, shown in listing 2.4, UpperCasePipe, LowerCasePipe, etc., but of course, it is possible to create custom pipes. The built-in DatePipe transforms a date according to local rules. The UpperCasePipe transforms text to all upper case in contrast to the LowerCasePipe. This pipe converts text to all lower case. (Angular Team, 2021)

An example implementation of a custom pipe is shown in listing 2.5, where the pipe's name is ExponentialPipe. It increases an input value exponentially. The pipe class and the PipeTransform interface have to be imported, and the @Pipe decorator has to be added. The exponent with which the value is to be increased is given as a parameter. If no parameter is given, the default value of the exponent is 1. The whole calculation happens in the transform method. The name property within the @Pipe decoration is needed to reference it in the template to use this pipe.

Pipes have the advantage that they can be used within the entire Angular application by creating them only once.(Angular Team, 2021)

Listing 2.4: An example of the Angular built-in DatePipe, which transforms a date according to local rules. (Code adapted by Angular Team 2021)

```
<p>My birthday is: {{ birthday | date }}</p>
```

Listing 2.5: An example of a custom Pipe, which increases a value exponentially. (Code adapted by Angular Team 2021)

```
import { Pipe, PipeTransform } from '@angular/core';
/* Usage: value | exponential:exponent
 * Example: {{ 2 | exponential:10 }} formats to: 1024 */
@Pipe({name: 'exponential'})
export class ExponentialPipe implements PipeTransform {
  transform(value: number, exponent?: number): number {
    return Math.pow(value, isNaN(exponent) ? 1 : exponent);
  }
}
```

2.4.7 Services

A service is typically a clearly defined purpose. Services are commonly used by components. The component class should be kept as simple as possible and complex tasks and functions should be outsourced to a service created for this purpose since the component should only take care of the data binding in order to mediate between the view and the application logic. Angular does not enforce this principle, but Angular supports this procedure in that services can be added very easily in components using dependency injection, which is explained in more detail in Section 2.4.10. (Angular Team, 2021)

2.4.8 Modules

Angular applications have a modular structure. They have a modularity system available, the so-called ngModules. Modules can be used to group related components, directives, pipes, and services in order to connect them to other modules. Every Angular application has at least one NgModule, loaded during the start-up as the first step. With this root module, the application will be started and is called AppModule. The corresponding file is named app.module.ts. For allowing to define a module, the ngModule decorator is needed. (Angular Team, 2021)

The ngModule metadata tells Angular which components, directives, and pipes belong to the module. It declares some of them as public in order to allow other module's components to use them, and it imports other modules.(Angular Team, 2021)

Furthermore, modules may be loaded eagerly, which is the default loading strategy of Angular. Eagerly-loaded means that it is loaded when the application is started. Nevertheless, modules may also be loaded lazy, which means that the modules are loaded asynchronously from the router. The router is explained in more detail in Section 2.4.12. Modules also make it easy to expand the application with external libraries. Many third-party libraries, such as Material Design[3], as well as the Angular libraries, such

[3]https://material.angular.io/ [accessed on: 14.01.2021]

as the FormsModule, are NgModules or available as such.(Angular Team, 2021)

Modules are already known from JavaScript, but they differ from ngModules. There is no metadata associated with the JavaScript modules[4] and the module is simply a separate file. JavaScript (JS) Modules can load each other and exchange functionalities with the export and import directives. With JS modules, the module's functionalities can be distributed over several files. Furthermore, it is possible to add services through modules to the application. Whether the service was developed internally, like the Angular Router, or externally. The involvement of the services happens via the ngModule metadata, which provides services that the components can use.(Angular Team, 2021)

In listing 2.6 an example of a basic AppModule is shown. It is generated by the Angular Command Line Interface (CLI) when a new application is created. With the Angular CLI, a command-line interface tool, it is easy to initialize, develop, and maintain an Angular application from a command shell.

Listing 2.6: Basic AppModule genreated by the Angular CLI (Code adapted by Angular Team 2021)

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

// @NgModule decorator with its metadata
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

[4]https://javascript.info/modules [accessed on: 15.01.2021]

In summary, modules are a fantastic way for structuring and organizing an application.

2.4.9 Data Binding

The communication between the template and its components and between parent and child components is guaranteed through data binding. There are three categories of possible directions for the data flow. One possibility is from source to view, while the opposite is from view to source. Both possibilities are called one-way data binding. This binding ties the data from the component to the view (DOM), which is the template, or from view to the component. Therefore, one-way data binding is unidirectional. Nevertheless, there is also the possibility of a two-way sequence of view to source to view. (Angular Team, 2021)

The property binding will be represented with some examples concerning the binding direction as follows:

From data source to view target

To bind from source to view, [] is used with exception of the interpolation.

- Property binding:

There are different forms of property bindings, often named one-way-binding, because of its direction flow. It is possible to set the target element's value, but it is not possible to read it. So, the value flows from the property of a component to a target element's property and not the other way around. The target element can be an element, a component, or a directive, as shown in the example listings below. (Angular Team, 2021)

Listing 2.7: An example of a property binding with an **element property as target**. (Code adapted by Angular Team 2021)

```
<img [src]="someImageUrl">
```

Listing 2.8: An example of property binding with a **component property as target**. (Code adapted by Angular Team 2021)

```
<app-some-detail [detailToDisplay]="currentDetail"></app-some-detail>
```

Listing 2.9: An example of property binding with a **directive property as target**. (Code adapted by Angular Team 2021)

```
<div [ngClass]="{'special': isSpecial}"></div>
```

- Attribute binding:

With attribute binding, it is possible to set values for attributes directly as shown in listing 2.10. Anyhow, the usage of attribute binding is not recommended by Angular. Angular recommends using property binding as long as it is possible.(Angular Team, 2021)

Listing 2.10: An example of attribute binding with an **attribute as target**. (Code adapted by Angular Team 2021)

```
<button [attr.aria-label]="help">help</button>
```

- Class binding:

Class binding makes it possible to add and remove Cascading Style Sheets (CSS) class names as shown in listing 2.11. In that listing, Angular adds the class name ″special″ if the bound expression ″isSpecial″ evaluates true. If the bound expression evaluates false Angular removes the class name.(Angular Team, 2021)

Listing 2.11: An example of class binding with a **class property as target**. (Code adapted by Angular Team 2021)

```
<div [class.special]="isSpecial">Special</div>
```

- Style binding:

Style binding allows it to set styles of the template dynamically. In the example in listing 2.12, Angular sets the color of the button to red when the bound expression is truthy or green when the expression is falsy.(Angular Team, 2021)

Listing 2.12: An example of style binding with a **style property as target**. (Code adapted by Angular Team 2021)

```
<button [style.color]="isSpecial ? 'red' : 'green'">
```

- Interpolation:

The curly braces are the delimiters of the interpolation. Interpolation is the most straightforward possibility of displaying some class properties in the DOM of the browser. The embedded expression will be replaced by the value of the corresponding component property. The view will be changed automatically when the property changes. (Angular Team, 2021) Listing 2.13 represents an example of interpolation with an element property as target.

Listing 2.13: An example of interpolation with an **element property as target**. (Code adapted by Angular Team 2021)

```
<p>{{expression}}</p>
```

From view target to data source

To bind from view to source, () is used.

- Event binding: Event binding comes into play in order to be able to react to user interactions, such as clicks, keystrokes, or touches. For event binding, a target event name within parentheses and a quoted template statement are needed. In listing 2.14 for example, click is the target event name and onSave the template statement.(Angular Team, 2021)

As at property binding, the target can be an element event (listing 2.14), a component event (listing 2.15), or a directive event (listing 2.16).

Listing 2.14: An example of event binding with a **element event as target**. (Code adapted by Angular Team 2021)

```
<button (click)="onSave()">Save</button>
```

Listing 2.15: An example of event binding with a **component event as target**. (Code adapted by Angular Team 2021)

```
<app-some-detail (deleteRequest)="deleteDetail()"></app-some-detail>
```

Listing 2.16: An exampe of event binding with a **directive event as target**. (Code adapted by Angular Team 2021)

```
<div (myClick)="clicked=$event" clickable>click me</div>
```

Two-way-binding

To bind in a two-way sequence, [()] is used.

As the name indicates, the data flow is given in both directions. So, the two-way binding allows to simultaneously listen for events and update the value when it changes. So, this gives us the possibility to share data between parent and child components. As can be seen from the syntax, two-way binding combines property binding (the parenthesis) and event binding (the brackets). (Angular Team, 2021)

An example of how to use two-way binding based on the Angular ngModel directive is shown in listing 2.17. The value of the property "name" flows into the HTML input element while, at the same time, changes made by a user flow back into the component by, in this case, entering a value in the input field.

Listing 2.17: An example of two-way binding based on the Angular ngModel directive. (Code adapted by Angular Team 2021)

```
<input [(ngModel)]="name">
```

2.4.10 Dependency Injection

A class needs certain services and objects to perform its functionality. Such services and objects are called dependencies, and these are injected using Dependency Injection (DI). DI is an important design pattern known from software development. In this design pattern, a class requests dependencies from external resources instead of creating them. Angular has its DI framework, which is used to provide new components with corresponding services or other things that are needed by the components.(Angular Team, 2021)

Angular recognizes by the parameters of the constructor of the component which services are needed by the component. As soon as Angular creates a component, the required services are requested from a so-called injector. The injector is one of the main components of the DI design pattern. Its task is to manage a container of service instances. If a requested service instance is not already in the container, the injector creates a new instance and adds it to the container before it is transferred to Angular. Angular calls the constructor of the component class with the required services as arguments, just when all requested services have been resolved using the injector.

Listing 2.18: An example of a default service, which can be used in the DI system, created by the Angular CLI. (Code adapted by Angular Team 2021)

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class DataService {
  constructor() { }
}
```

As shown in Listing 2.18, a class, which is decorated with the @Injectable decorator tells Angular that it can be used in the DI system. The "providedIn" metadata with the value "root" tells Angular that the DataService is visible within the whole application. It declares that the root application injector should create this service. By configuring providers, services are made available to specific parts of the application. That parts, which need the service. (Angular Team, 2021)

2.4.11 Lifecycle-Hooks

Every instance of a component or directive in Angular has a lifecycle. Angular manages this lifecycle. The lifecycle starts as soon as the instance of a component class is instantiated, and the component view is rendered with all of its child views. Angular provides so-called lifecycle hooks that a developer can react to events in the component's lifecycle. Unlike the components, directives do not have view-specific lifecycle hooks but all other ones. The corresponding interface has to be implemented to react to one of these lifecycle hooks. Each interface has a single hook method, and its name corresponds to that interface with the additional prefix "ng". All lifecycle hooks are listed in Table 2.1, relating to the order Angular calls them.

2.4.12 Routing and Navigation

A Single Page Application (SPA) consists of a single HTML document and so all the application's functions exist in this single document. Therefore, the browser has to render the parts related to particular components to change what the user sees. (Angular Team, 2021)

The Angular Router was introduced to allow users to switch between different views created relating to specific application tasks. The Router can be used to navigate between different component views. The Router interprets the browser Uniform Resource Locator (URL) and navigates to the corresponding component view. So-called routes are used to define in which way users can navigate in the application. Often it is the case that someone would like to pass on information between the components. (Angular Team, 2021)

For example, let us assume that the user clicks on a particular news article in a news article overview and would like to navigate to this particular article. Each article has a unique Identifier (ID), and this ID is required to display the article that the user has clicked on. This ID can be given via the optional URL parameters, as shown in Listing 2.19. Each routed Angular application has exactly one instance of the router service. As soon as the

Hook Method Name	**Timing and Purpose**
ngOnChanges	It is called before the ngOnInit method and always when an input property changes. So, the purpose is to react to changes of a data-bound input property. A SimpleChange object with the current and previous value of the input property is passed to the method.
ngOnInit	It is called only once after the first ngOnChanges call. It is for the initialization of the component after all input properties are set.
ngDoCheck	It is called immediately after every change detection run, which is the ngOnChanges call, and immediately after ngOnInit. To recognize and react to changes that Angular does not recognize itself.
ngAfterContentInit	It is called once after the first ngDoCheck. For reaction after Angular has projected external content in the component view or into the view that is a directive.
ngAfterContentChecked	It is called after ngAfterContentInit and after each subsequent ngDoCheck. To respond after Angular has checked the external content that was projected into the component.
ngAfterViewInit	It is called once after the first ngAfterContentChecked. For reaction after Angular has initialized the component view and its child component views.
ngAfterViewChecked	It is called after the ngAfterViewInit and every subsequent ngAfterContentChecked. For reaction after Angular has checked the component view and its child component views.
ngOnDestroy	It is called immediately after Angular destroys the component or directive. To clean up before the directive or the component is destroyed by Angular. A typical use case is that an observable get unsubscribed so that no memory leaks occur.

Table 2.1: Lifecycle hooks relating to the order Angular calls them. (adapted by Angular Team 2021)

URL in the address line changes, the router searches for the corresponding route, which specifies which component has to be displayed. The developer is responsible for configuring the routes to allow the router to load a specific component. Without the routes, this mechanism is not possible, and the router is not able to load the corresponding component. The following example in listing 2.19 creates two route definitions and configures the router using the RouterModule.forRoot method. The result is added to the Module via the imports array. (Angular Team, 2021)

Listing 2.19: An example configuration of two route definitions. Each of these routes represents a URL path to a component. (Code adapted by Angular Team 2021)

```
const appRoutes: Routes = [
  { path: 'news-overview', component: NewsOverviewComponent },
  { path: 'news-detail/:id',   component: NewsArticleDetailComponent
      },
];

@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes)
  ],
  ...
})
export class AppModule { }
```

3 Methodology

This section describes the process of how we obtained the results using the introduced concepts described in the literature review. There are many different approaches of how companies implement an SDLC. We are interested in the SDLC steps necessary to implement the goals and objectives described in Section 1.2. These steps include the RE in the analysis phase, the design or concept, the implementation, and the testing or feedback phase.

3.1 Analysis

The first step is the analysis phase, including the actual state analysis and RE, to develop the product regarding the customer's needs. The actual state analysis should examine the domain of the application and all relevant actors. Moreover, it should also include the analysis of the system architecture of the current situation. Besides, the requirements have to get determined and documented in this phase of the SDLC.

It was determined that the visualization for news articles and a registration process must be implemented. Nevertheless, the exact requirements have to be determined with the user's involvement to find out what these components need for additional functionalities or elements. Besides, these requirements are needed to be able to find a suitable design. The gathering of the requirements is effected by different means. On the one hand, the colleagues from sales are in contact with the soccer clubs because their task is to convince them to be part of this platform. They persuade them by showing the native apps and telling them about the added value of this platform. In that way, the soccer teams' feedback and wishes can be included in the creation of the concept. Therefore, there will be discussions and interviews with the colleagues from sales to work out requirements. On the other hand, observations and analysis of similar platforms already available on the domestic market are other means of gathering requirements. So, state-of-the-art observations are crucial. Another method is the consideration of requirements that had already been implemented in the native apps and were well received by the end-users.

When the gathering is finished, the requirements have to be documented. As described by Sommerville and Sawyer, 1997, the requirements will be divided into functional and non-functional requirements to ensure that the non-functional requirements also get devised. The consideration of the non-functional requirements is important since they could consider attributes like performance and usability. For defining the non-functional requirements, meetings with the developer team are essential since these are decisions about the implementation and affect the whole application. The functional requirements will be described as user stories as suggested by Cohn, 2004 since they define the requirements from the user's point of view.

Before the design can be derived from the defined requirements, the needs' prioritization has to be done. The functionalities that contribute to the basic functionality have to be implemented first.

3.2 Design / Concept

After the specification of the needs, other websites and social media platforms will be examined and analyzed to derive the conception from the requirements. So, the involvement of state-of-the-art observations is crucial. Different platforms have to be considered, and they have to be carefully analyzed to allow filtering out what is usable for the case of this soccer platform. So, these regarded platforms' features have to be examined to determine which are well and which are badly realized. The gained knowledge will incorporate into the design of this work. Other soccer platforms on the market and generally, platforms with some kind of feed and pictures are the main focus since the news feed and the image gallery are parts of this thesis.

Moreover, the desired state's system architecture has to be considered before the concept can be derived. The next step includes discussions and meetings with the developer team for defining the logic about the implementation. It has to be declared how each particular software component interacts with each other. Therefore, the developers must define the communication between the back-end and front-end and how all necessary things for the implementation will be available. This communication should be represented utilizing UML diagrams for better understanding.

Furthermore, the whole team, including the product owner, must agree upon a Style Guide to allow a consistent design throughout the whole application. When these things are declared, the next step is to use them and derive the UI conception from the requirements. In order to develop the UI, mock-ups for each needed view have to be created. There are lots of tools for prototyping that could be used. We chose Figma,[1] because the company is used to it and every one of the team has access to it. After finishing the prototypes, they have to be discussed with the product owner.

3.3 Implementation

After all necessary design decisions are set, the implementation phase can start. The requirements will be implemented regarding the prioritization. The used Framework for the development of this application will be Angular. Furthermore, there will be regular meetings to discuss the progress of the implementation and to estimate how long the implementation for the particular features will take. Moreover, if someone needs help, this will also be discussed in this meeting.

3.4 Testing / Feedback

The testing phase is the stage where the testers of the team come into play. All features have to go through a detailed testing phase before they can

[1]https://www.figma.com/ [accessed on: 07.02.2021]

be released. The testers put themselves in the role of the users and click themselves through the application. They test the new features and also have to consider other things that could be affected by the new feature. Furthermore, the source code has to be reviewed by at least two other developers, that can give their feedback. When they are not satisfied, the developer of the feature has to adapt the code regarding their complaints or to discuss why they have implemented the feature in this manner. Only after the developers have approved the implementation and the testing phase was successful, the feature can be released. The testing phase is successful when the functionality is fulfilled.

3.5 Summary of the Methodology

Each phase of the SDLC that is used in the further course of this work is explained in detail. Each phase describes the process of how the goals defined in Section 1.2 can be achieved. The SDLC consists of analysis, design/concept, implementation, and testing/feedback. Each of these stages includes reflection meetings with the entire team because each member is involved in the whole development process.

4 Results

This section presents the results that could be achieved by implementing the phases described in Section 3. Therefore, the section is divided into the analysis, the concept, and the implementation phase. Furthermore, a requirement comparison follows after the implementation to emphasize if the requirements are fulfilled.

4.1 Analysis

"Software development can be defined as the process of designing, programming, specifying, documenting, testing, conceptualizing and debugging that go together in the development and maintenance of various Frameworks, applications and other components of the software help." (IntroBooks Team, 2020)

In the first step of a software development process, it is necessary to figure out what potential users expect from the application in order to be able to specify and design the application. Besides, the user's needs have to be analyzed, structured, and adjusted by the whole project team to ensure that the needs are technically feasible, that they have been understood, and will be met.

Therefore, the following section deals with RE and the subsequent prioritization of the resulting user needs. The requirements themselves are structured into functional and non-functional ones. The desired state can be established from these requirements. With the help of sketches of the system architecture, the actual state will be demonstrated.

4.1.1 Requirement Engineering (RE)

RE, explained in more detail in Section 2.2, is the process of generating requirements. Gathering potential users' requirements for the application is an essential and mandatory step at the beginning of the development process. In our case, the needs could be determined by analyzing existing similar platforms and working closely with soccer fans and soccer clubs, which will be mentioned in the actual analysis part below. As a result, all the resulting software requirements are explained and divided into functional and non-functional requirements below, as recommended by Sommerville and Sawyer, 1997.

Actual State

In the process of capturing, analyzing, and specifying the requirements for a software product, it is essential to examine the domain of the application. In the first step, the relevant actors are determined. In the case of this work, the relevant actors are fans of certain soccer clubs who are represented on this platform or simply soccer enthusiasts in general and the soccer clubs of course.

The soccer teams are also actors since they use this platform to distribute their content. The news articles are part of this content. This platform has a separate web page for creating the news articles, which get distributed on the native apps and the web application that gets developed in the further course of this work.

After the actors have been determined, it is also important to mention which mediums are available in order to be able to gather and integrate requirements, especially user requirements. First and foremost, the native applications already mentioned in Section 1, which are still being revised, are used as a prototype. Therefore, there are already some users who inform the platform about missing features or features that need improvement. Thus, the wishes of those interested in soccer are caught up and taken to heart, which is absolutely crucial for the success of a product. Because it is an interdisciplinary collaboration between the developers, the product owner, and the colleagues from sales, there is still another medium through

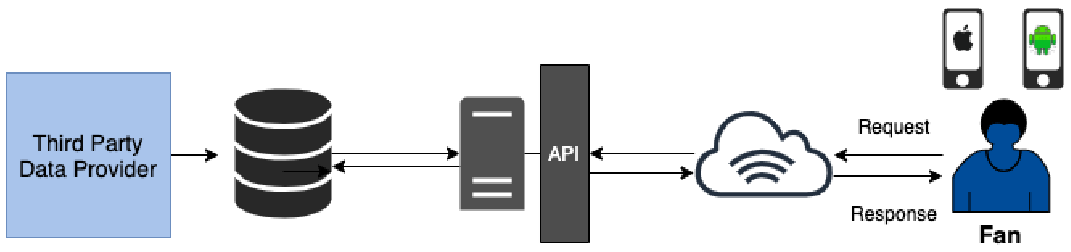


Figure 4.1: The figure sketches the system architecture of the actual state.

which feedback is obtained. That feedback comes from the soccer clubs themselves and is obtained through the colleagues who work in sales. They maintain contact with the soccer clubs by showing them the product and convincing them to become part of this soccer platform.

Another method in determining requirements is the observation and analysis of similar platforms already available on the market. Therefore, state-of-the-art observations are an essential part of gathering additional user requirements. Nevertheless, there is still another method for gathering needs, namely, involving requirements that had already been implemented in the apps and were well received by the end-users. These needs will be transferred to the web application. The requirements are adjusted and checked in consultation with the entire team on-site in discussions or via a business communication platform. These discussions ensure that objections, suggestions for improvement, or problems can be addressed. Therefore, regular reflection meetings are the most crucial part of developing this whole product.

System Architecture Overview of the Actual State

Figure 4.1 illustrates the architecture of the current situation. As already mentioned in Section 1, the content of the soccer platform is currently only available in the native apps for Android and iOS. Nevertheless, the native apps are still being developed and cannot be considered complete. Since the soccer platform is still in its infancy, there are currently many changes in terms of requirements and design. However, these apps at the current state can be viewed as a prototype to obtain users' opinions. The users share their suggestions for improvement to the support team and the product owner via email. Thus, the wishes of the users can flow directly into the development. Furthermore, the colleagues who work in sales use these apps as a prototype for convincing the soccer teams to be part of this platform, as mentioned before. They must must be able to demonstrate already existing functions when they want to win the teams over for this platform by convincing them about this platform's value. Furthermore, a few teams are already part of it. Thus, their feedback and wishes can also flow directly into the development process.

Functional Requirements

After the requirements have been captured through observation and analysis of other platforms, as well as through direct input of user feedback, these requirements must be divided. The requirements are divided into two parts: the functional and the non-functional requirements. As already mentioned above, user feedback is obtained through the existing native apps. The reuse of requirements by the apps is also included here.

In the following sections, the functional requirements are described as user stories and are given sequential numbers in order to be able to refer to them later in the work. Besides, they are already assigned regarding the features that have to be implemented. Therefore, the user stories are structured into three parts. One contains all user stories concerning the news articles themselves, one relating to the sharing of the news articles, and the last one regarding the registration and login process.

News Articles

Story Card N1

As a soccer enthusiast, I want to see several news articles from all soccer clubs at one glance so that I have an overview of all clubs and their latest news.

The user should have the opportunity to see all the latest news from the clubs. For this purpose, there must be a news article overview or news feed in which all news articles are visualized in an overview variant.

Story Card N2

As a soccer fan, I want to filter the news articles, such that I only see the articles associated with the soccer clubs I like the most, so that I can create my individual scope.

The user should be able to get all the latest news of their favorite teams visualized. With this option, the user can create their personal scope and decide which news article they will get.

Story Card N3

As a soccer fan, I want to be able to follow the team through the news article so that I can easily add it to my individual area.

Somewhere in the news article, there should be a follow button so that the user can follow the team. Following a team means adding a specific team to the favorite teams and, therefore, getting all the latest news and notifications about games, for example.

Story Card N4

As a user, I also want to get the chance to see pictures in the news articles for a better user experience.

The user should be guaranteed a better user experience by scrolling over several articles with imaged attached to the news articles.

Story Card N5

As a user, I want to see how many images have been attached to know how many images I have to view if I want to see all of them.

It should be visualized how many images have been attached to the news article if there are more than one.

Story Card N6

As a user, I want to get the option to click through all the images in the news article overview so that I am not forced to read the whole article.

There should be something like a picture gallery so that the user can see all the pictures in the news feed.

Story Card N7

As a user, I want to be able to zoom the images attached to the news articles so that I can inspect it in every detail.

The user should be able to zoom the attached pictures to get the chance to see every detail when it may be too small or for other reasons.

Story Card N8

As a user, I want to get the possibility to see the entire news article.

The user should be able to click on the news article in the news feed and should be navigated to the detail view of the article. There should be much more information shown on the detail view.

Story Card N9

As a user, I want to know which team the article is from and who the author is to ensure credibility.

The user wants to know the author of the article they are reading to ensure its credibility. Therefore, there should be a field with the name of the author somewhere in the visualization of the detail view. It should be noted that the author is not the team itself. This should be stated separately. In addition, the team should navigate the user to the corresponding team page.

Story Card N10

As a user, I want to know when the article was published because I do not want to read news articles about soccer from the past.

So that the user knows when the article was published the publication date should be visible to the user. The published time should be visualized as followed:

- ″One minute ago″ when the news article was published less than a minute ago
- ″mm minutes ago″ when the news article was published less than an hour ago
- ″HH hours ago″ when it was published less than 24 hours ago
- ″DD days ago″ when it was published less than 7 days ago
- ″DD.MM″ when the article was published more than 7 days ago

Sharing of News Articles

Story Card S1

As a user, I want to get the option to share the news article on different platforms.

The user should have the opportunity to share the contributions on different platforms. This requires a sharing button and several options on which platforms it can be shared. So, this story card can be split into those four options:

- Sharing via WhatsApp
- Sharing on Facebook
- Sharing on Twitter
- Just copying the link

Registration/Login

Story Card L1

As a user, I need the possibility to create an account on the platform.

In order to allow the user to design an individual area, the users should be able to register and to log in again with the created account.

Story Card L2

As a user, I want to log in with Facebook so that I do not have to create an extra account.

The user should be able to register on this platform via Facebook with just one click.

Story Card L3

As a user, I want to log in with my Apple ID so that I do not have to create an extra account.

The user should be able to log into this platform with just one click using their Apple ID.

Story Card L4

As a user, I want to change my account settings for a better user experience.

The logged-in user should be able to change their account data afterward.

Story Card L5

As a user, I want to reset my account if I forgot my email address or password.

The user must be able to reset their password if they have forgotten the password or their email address. Otherwise, the user can no longer log in to the platform.

Story Card L6

As a user, I want to be able to contact the support.

The user must be able to contact the support if they have forgotten their used email address or because of some other issues.

Non-functional Requirements

In contrast to the functional requirements, the non-functional requirements are not about what the system should implement. They describe how the functionality should be implemented.

A non-functional requirement goes beyond functional requirements and defines the quality feature of a software system. It is about quality requirements such as performance, reliability, usability, and effectiveness of the system. These requirements are not mandatory but are recommended because failing to meet them may result in user requirements not being met either. They ensure that the application is easy to use. (Guru 99, 2021)

The collected non-functional requirements are partly derived from the functional requirements since they describe how the functional requirements should be implemented. They are described below.

Platform Independence

Since the application has to be implemented as web application, the independence of operating systems is fulfilled. Besides, the application should be optimized for the following browsers:

- Chrome[1]
- Firefox[2]
- Microsoft Edge[3]
- Safari[4]

Furthermore, it should be possible to visualize the application in the browser on a wide variety of devices. The visualization must therefore be implemented for the most varied of screen sizes. It is planned that the application should ultimately be able to be displayed on mobile phones, tablets, and desktops in order to capture the largest possible target group.

Stability

Errors that occur during the registration and login process during run-time must be communicated to the user via appropriate error messages. Besides, the error messages of the native apps and the ones of the web application created in the course of this work should be standardized.

[1]https://www.google.com/intl/de/chrome/ [accessed on: 09.02.2021]
[2]https://www.mozilla.org/de/firefox/new/ [accessed on: 09.02.2021]
[3]https://www.microsoft.com/de-de/edge [accessed on: 09.02.2021]
[4]https://support.apple.com/de-at/safari [accessed on: 09.02.2021]

Another aspect of ensuring stability is user feedback. Users should be allowed to transmit their feedback to the platform by contacting support via a dialog that sends the message to the back-end.

Usability, UI

The appearance of the application should be as clear and intuitive as possible for the user. In relation to the visualization on small devices, which are usually Smartphones, should contain the standard, known mechanisms of use that the user of a mobile device is used to. The data or information displayed should be structured and mapped in such a way that, despite the relatively small display, it is not a problem to read them and therefore may differ from the visualization of larger screens, as tablet and desktop.

The most common screen sizes must be considered when designing the User Interface (UI) corresponding to good usability. The usability must be as efficient and straightforward as possible. Every function should be accessible with just a few clicks.

Maintainability

The application's architecture should be modular and structured so that further development and maintenance of the existing system is guaranteed. Thus, it should not cause any additional effort if new or existing functionalities have to be maintained or added.

Spam Protection

As already mentioned above, it should be possible to contact a support team. To prevent the back-end from being spammed, a protective mechanism such as a honeypot should be built to protect against spammers and bots.

A honeypot[5] is a mechanism designed to deflect an attacker from the actual target somehow. The honeypot aims to attract and "trap" people who try to break into other computer systems.

Password Specification

The password must consist of at least six characters. Besides, there should be a confirmation field to avoid that the user mistypes their password and can not recognize it.

Username Specification

Duplicate usernames are not supported. In order to fulfill this requirement, there should be a check during the registration process if the username is still available. Furthermore, no special characters except underline, dash, and point should be allowed, as well as the length limited from 4 to 25 characters.

Name and Surname Specification

Since the username is already mandatory, the name and surname should only be provided optionally. However, if the user wants to specify their name, it must consist of 2 to 50 characters. The allowed characters should be UNICODE characters. The name should not be limited too much, as personal names are quite different depending on the countries and cultures.

On the one hand, people may have several surnames, and therefore limiting the length of the surname would not make any sense. On the other hand, people can have an initial in the middle of their name. Consequently, special characters such as the full stop should also be allowed. UNICODE characters should be allowed because, in many countries and cultures, the writing

[5]https://www.computerweekly.com/de/definition/Honeypot-Honigtopf [accessed on: 10.01.2021]

system consists of ideographic descriptive characters such as Japan, for example. (Richard Ishida, 2011)

Different Visualisations of News Articles

In addition to the news articles created by teams themselves, there will also be editorial articles. It should be noted that the articles written by the editorial team differ from the team news. Therefore differences will appear in the visualization.

With the team news, there will be the possibility to follow the teams. Following the teams allows getting all the latest news and notifications about games, for example. However, this option is not given by editorial teams. Therefore, a solution is needed that the following mechanism is only available for team news. Nevertheless, the editorial news should support the option to share these articles on other social media platforms instead.

In the case of team articles, the team name should be clickable and forward to the corresponding team page provided by this soccer platform, as mentioned in Section 1.1. With the editorial reports, this forwarding mechanism does not make sense because this soccer platform provides no separate homepage for editorial development.

4.1.2 Prioritization

After the requirements analysis has established which requirements are needed and how they should be implemented, the next step is to decide the order in which the user stories will be implemented. First, the essential requirements should be implemented, which are those that contribute to the basic functionality. Additional functions that are to be implemented in the future are based on this basic functionality.

Table 4.1 shows the order according to the requirements' priority. Reference was made to the previously defined Story Cards in Section 4.1.1 to describe and sort the requirements. The most important are those that concern the implementation of the news article visualization. The news items are a

Requirements	Category
Story Card N8	News
Story Card N1	News
Story Card N4	News
Story Card N9	News
Story Card N10	News
Story Card L1	Login
Story Card L4	Login
Story Card L5	Login
Story Card L6	Login
Story Card N3	News
Story Card N2	News
Story Card N6	News
Story Card N5	News
Story Card N7	News
Story Card S1	Sharing
Story Card L2	Login
Story Card L3	Login

Table 4.1: Illustration of the prioritization of the requirements.

central feature of the platform, and thus the most basic functionality needed at the beginning. They are the clubs' means of communication with the fans and the means for drawing attention to themselves.

The starting point is the detailed news view with all the information an article has. Before we can display the overview of several news articles, we have to realize the detailed view. So, the first implementation should be Story Card N8. As already mentioned, the users also want to see the images attached to the article. However, in the first development steps, it should be realized that only one image is displayed in the view. It is not the most crucial functionality to display all images. It is a feature which will be implemented in the further course. Therefore, the next Story Card to implement is Story Card N1, the news feed overview. After we have the detailed view, it should be realizable with little effort because it contains much less information than the detailed view. Story Card N9 will be realized simultaneously with both Story Cards mentioned before because it is contained in the articles, as well as Story Card N10. These cards have to be considered in the process of prototyping the drafts for the detailed view and the overview, as well as Story Card N4. This visualization relates to just one single image attached to the news articles. The visualization of more than one image will be discussed later on, as it does not contribute to the basic functionality.

After completing these requirements, the log-in and sign-up process are needed to create a more significant user experience. The users want to have an individual area and decide which articles they get to see. Therefore, the next Story Card we are going to implement is Story Card L1, which contains the log-in and the registration. Afterwards, Story Card L4 and L5 are needed. The user should be able to change the account settings and get help when they forgot their password or email address. In order to get help with the forgotten email address or with something similar, they need the option to contact the support team, so Story Card L6 has to be realized next. After these implemented Story Cards, the application has a good starting point, and the users can already do something on the website.

To increase the user experience, the next Story Cards to implement are Story Card N3 and N2. In order to visualize Story Card N3, the articles of the favorite teams of the user, there has to be an option to follow these teams via the news article, which is Story Card N2. The remaining Story Cards are features, which will maximize the user experience. The first feature relates to Story Card N6 and N5. A kind of picture gallery is needed so that the user has access to all attached pictures and can easily navigate through them. Furthermore, the user wants to know how many images they have to navigate through. Therefore, this has to be considered in the visualization. The next step is to allow the user to zoom these images, which is Story Card N7. The next feature relates to Story Card S1, which allows the user to share the articles on other social media platforms. The last to remaining Story Cards deal again with the log-in process. Story Card L2 and L3 should allow the user to decide if they want to use Facebook or their Apple ID to log-in on the soccer platform instead of creating a new account.

4.2 Concept

In this section, the application gets conceptualized by taking into account the results obtained in Section 4.1. First of all, the section describes the system architecture of the desired state derived from the previous results. It also describes the REST API since it is the connection between the front-end and the back-end and, therefore, essential. Right after the system architecture overview, the software architecture description follows. This description consists of sequence diagrams for the features that have to be implemented. They illustrate the interaction of all system elements. Before the design for the features can be derived, the style guide, the colors, and the icons sections determine general rules for the procedure. Furthermore, a description of the state-of-the-art observation is also contained. Finally, the features are examined and designed based on mock-ups.

4.2.1 System Architecture Overview of the Desired State

The web app is currently under construction, and fundamental parts of it will be created in the course of this work. This work focuses on the development of the visualization in the front-end. The front-end task is to visualize the data accordingly for the respective end device. As already

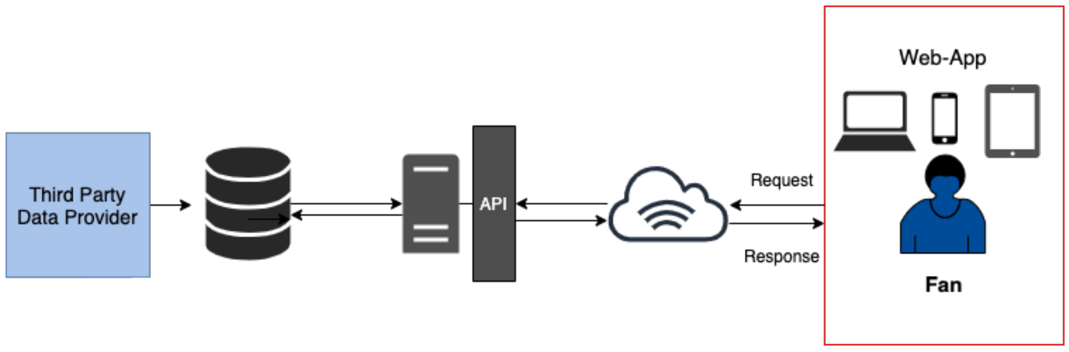


Figure 4.2: The figure sketches the system architecture of the desired state.

mentioned in the requirements, the visualization should be optimized for smartphones, tablets, and desktops. So, the user should be able to access the web app and thus the soccer platform's content via any internet-enabled device. In Figure 4.2, the system remains the same as illustrated in Figure 4.1 in the actual state description. Only the end device is replaced. Therefore, the native apps are replaced by the web application.

As shown in the draft, the soccer data regarding game results and soccer tables are loaded into the database by a third-party provider. Contributions like the news articles, which will be created by this platform, will be created in the back-end and saved in the database. These contributions do not need a third-party provider. A REST API that returns the requested data in JavaScript Object Notation (JSON) format is implemented to access all the data.

REST API Overview

This API is the connection between the front-end and back-end. There is a back-end for development and production. Each API-Call consists of the appropriate back-end and the corresponding route or address for the endpoint to provide the data.

Almost every API-Call needs to have a header field named API-access-token. This API-access-token is created the first time a user is registered. Therefore a registration process is an essential part of the application. There are just

Code	Type	Description
400	Bad Request	Token is unknown
401	Unauthorized	No token is supplied for restricted Resource
403	Forbidden	User level not sufficient for desired operation
500	Internal Server Error	Something else went wrong (very unlikely)

Table 4.2: Error Codes of the API depending on user level

a handful of exceptions that do not need this API-access-token. One of these exceptions, for example, is the request for getting the latest news. The value of the API-access-token field must be a 24 character long hex-string. This header identifies users and hence their permission level. Each user has different permissions since users have the possibility to become reporters of their favorite teams. Admins, of course, also have a different permission level. Depending on the required user level, the API call may return one of the errors, demonstrated in Table 4.2.

An important fact concerning the authentication process is that a representation of the user needs to be created via an Hypertext Transfer Protocol (HTTP) POST request to the /users route first. After the successful creation of a user, the necessary API-access-token can be requested via logging in. This request happens via an HTTP POST request to the /users/login route. The back-end returns the API-access-token when the requested user is found in the database.

4.2.2 Software Architecture

As explained and showed in Section 4.2.1, the system architecture consists of a database, a REST API, and the web application itself. Furthermore, the requirements were gathered and divided into three features: the visualization of the news articles, the sharing of the items, and the login or registration process. In discussions with the entire developer team, it is determined how the logic should be implemented. In some cases, specific logic models are already available, as these had already been defined for the implementation of the native applications. In other cases, new logic models have to be defined. The logic is recorded using UML diagrams to illustrate

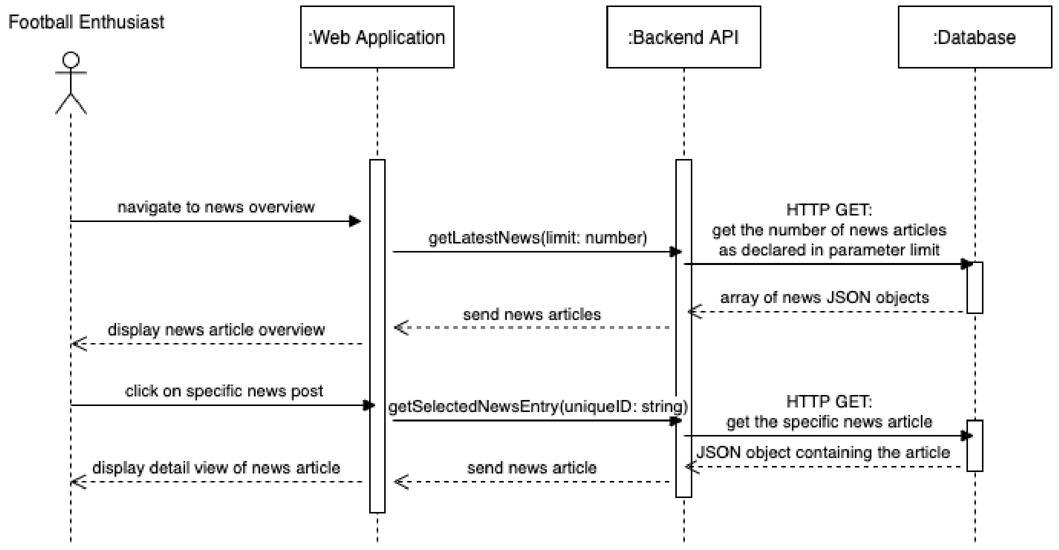
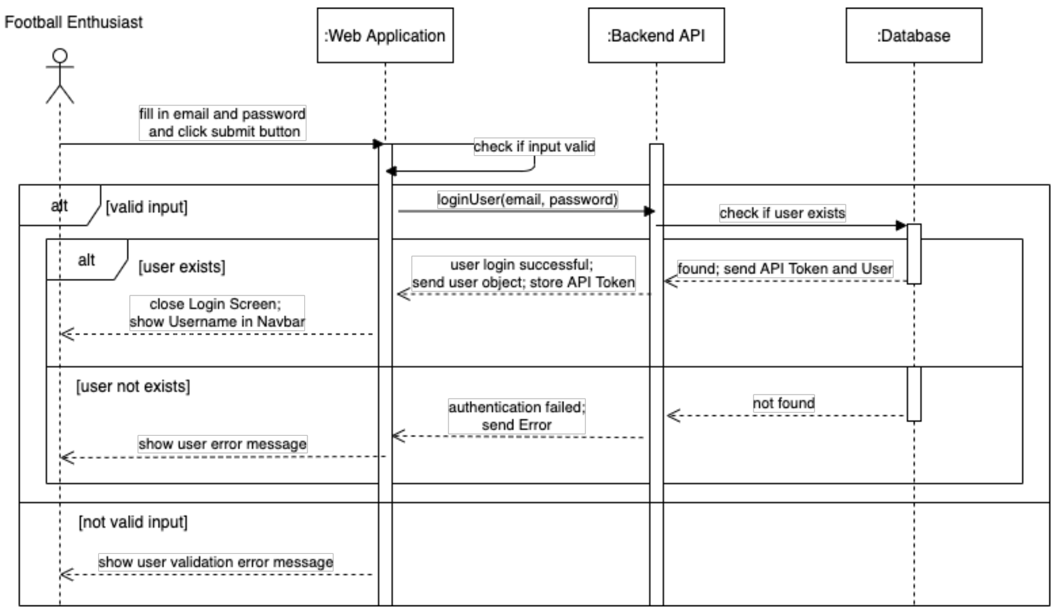

Football Enthusiast
:Web Application
:Backend API
:Database
navigate to news overview
getLatestNews(limit: number)
HTTP GET:
get the number of news articles
as declared in parameter limit
array of news JSON objects
send news articles
display news article overview
click on specific news post
getSelectedNewsEntry(uniqueID: string)
HTTP GET:
get the specific news article
JSON object containing the article
display detail view of news article
send news article

Figure 4.3: The UML Sequence Diagram shows the interaction of the application, the API, and the database when a user wants to see an overview of the latest news.

the process. How the software should interact concerning these features is illustrated in the sequence diagrams in the next few paragraphs.

Figure 4.3, shows the interaction of all components when the user wants to get an overview of the latest news. First of all, the user navigates to this overview. When navigating to the overview, the front-end has to send an HTTP GET request to the API with a parameter that sets a limitation. This limitation is needed that not all database articles get loaded, which could be a vast dataset. Loading all the news entries at once would have several disadvantages. Firstly it would take a long loading time and would have a negative impact on the user experience. The more articles exist, the more time it would need to load them. Secondly, it makes no sense to show the user old news items that are, for example, five months old. Therefore there will be a limitation set by the developers, and if the user wishes, they can also display older news articles and thus more. The application requests the news article data based on the identifier by clicking on a specific news article. After successfully obtaining the object containing the entry, the news detail view should open.

The process when the user wants to log-in on the platform is shown in the



Football Enthusiast
:Web Application
:Backend API
:Database
fill in email and password and click submit button
check if input valid
alt
[valid input]
loginUser(email, password)
check if user exists
alt
[user exists]
user login successful; send user object; store API Token
found; send API Token and User
close Login Screen; show Username in Navbar
[user not exists]
authentication failed; send Error
not found
show user error message
[not valid input]
show user validation error message

Figure 4.4: The UML Sequence Diagram shows the application's interaction, the API, and the database when a user wants to log in.

second sequence diagram, Figure 4.4. This diagram is explained before the sign-up process because the registration process also needs this logic after the successful creation of the user. The user has to fill in their user information, which is their email address and the password, first. The next step is to validate this data. The front-end should check if the email address is a valid one. When the input data is not valid, the user should get an error message displayed to fix their input. When the input is valid, the front-end sends an HTTP POST request to the corresponding route. The back-end checks if the user exists with this email address in the database and that the password is correct. Again, if this process fails, the user should get corresponding user feedback. If the process is successful, the API-access-token contained in the user object gets sent back to the front-end, which has to store this token for later requests and query the username from the user object to display it for the user.

The next sequence diagram, Figure 4.6, shows the process when a user wants to sign-up on the platform. First of all, they have to fill in their user information and click the submit button afterward. The front-end has to check if the inputted data is valid. In relation to the username, it has to fulfill some restrictions as mentioned in the non-functional requirements (section 4.1.1). Furthermore, there should be a password confirmation so that the user inputs a password they can recognize without any typos. When the input is not valid, the user should get corresponding user feedback in the form of error messages. Because of the fact that the username is unique, the application has to check if it is available when the user clicks the submit button. The front-end has to send an HTTP POST request with the username to the corresponding endpoint to check the availability. When the username already exists in the database, an error message should be displayed to the user. When the username does not exist in the database, the process continues. A representation of the user gets created and stored in the database with their unique API-access-token. The back-end sends the user object containing the token to the front-end to log in the user, which is the same process as shown and explained before in Figure 4.4.

The sequence diagram 4.5 illustrates what messages are sent when the user wants to share the news article on another social platform. The Story Card S1 describes the associated requirement. The sharing feature has to include the options to share on Facebook, Twitter, WhatsApp, and copy the article's link to share it somewhere else. The user clicks on one of the sharing buttons for one of these platforms mentioned before to start the sharing process. The application sends an HTTP GET request to the back-end, with the news article's unique ID attached to it, and waits for a response. The response consists of the title, the description, the content URL, and the image URL. There are two alternatives for the sharing process. One of these alternatives describes the sharing when the user uses a non-mobile device or when the mobile phone browser does not support the Web Sharing API [6]. Therefore, this alternative is also the fallback solution at the same time. In this case, the application should open a new tab with the sharing dialog of the particular platform the user chose. When the user uses a mobile device, and the browser supports the Web Sharing API, the particular system's sharing screen should open, where the user can select the specific platform.

[6] https://www.w3.org/TR/web-share/ [accessed on: 20.02.2021]

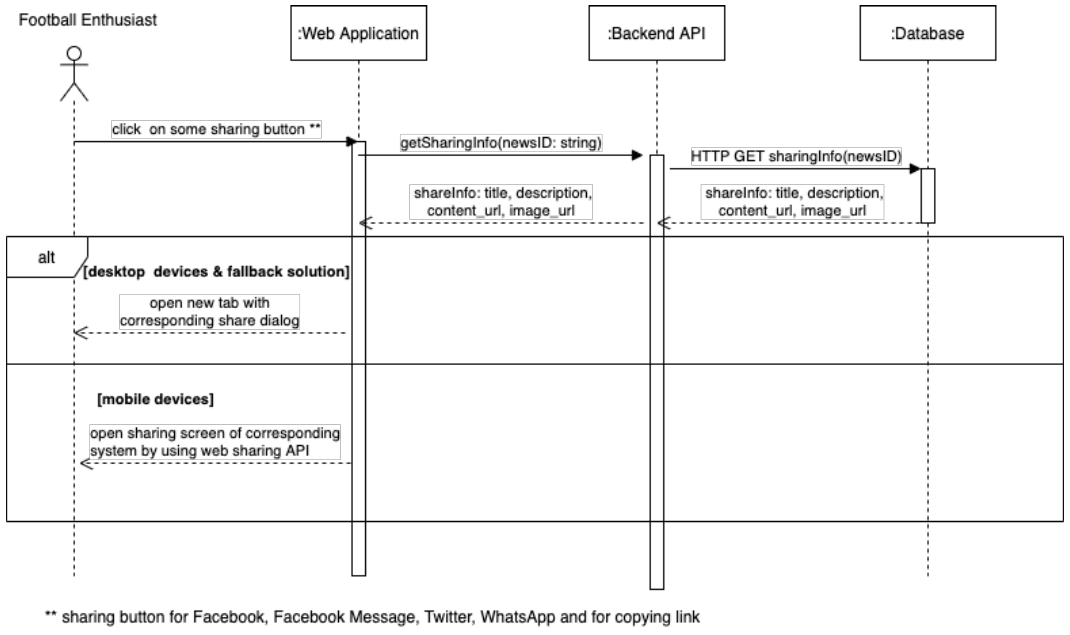
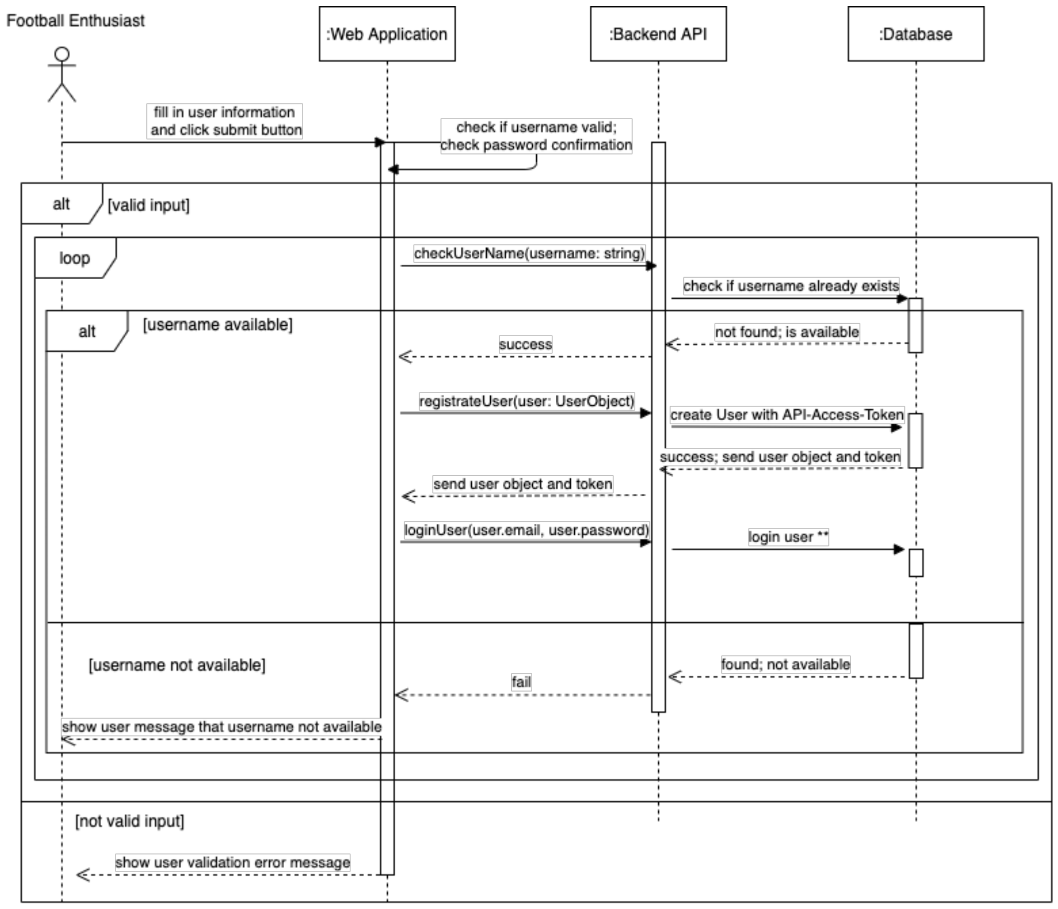

Football Enthusiast
:Web Application
:Backend API
:Database
click on some sharing button **
getSharingInfo(newsID: string)
HTTP GET sharingInfo(newsID)
shareInfo: title, description, content_url, image_url
shareInfo: title, description, content_url, image_url
alt
[desktop devices & fallback solution]
open new tab with corresponding share dialog
[mobile devices]
open sharing screen of corresponding system by using web sharing API
** sharing button for Facebook, Facebook Message, Twitter, WhatsApp and for copying link

Figure 4.5: The UML Sequence Diagram shows the interaction of the application, the API, and the database when a user wants to share a news article on another platform.


Football Enthusiast
:Web Application
:Backend API
:Database
fill in user information and click submit button
check if username valid; check password confirmation
alt
[valid input]
loop
checkUserName(username: string)
check if username already exists
alt
[username available]
not found; is available
success
registrateUser(user: UserObject)
create User with API-Access-Token
success; send user object and token
send user object and token
loginUser(user.email, user.password)
login user **
[username not available]
found; not available
fail
show user message that username not available
[not valid input]
show user validation error message

** login User: same process as shown in log-in process sequence diagram

Figure 4.6: The UML Sequence Diagram shows the interaction of the application, the API, and the database when a user wants to sign up on the soccer platform.

4.2.3 State-of-the-art Observation

Before the design can be derived from the requirements, different platforms have to be considered. These platforms have to be carefully analyzed to allow filtering out what is usable for the case of this soccer platform. So, these regarded platforms' features have to be examined to determine which are well and which are badly realized. The useful features and elements could be incorporated into the design of this work. Other soccer platforms on the market and generally, platforms with some kind of feed and pictures are the main focus since the news feed and the image gallery are parts of this thesis.

There are not many soccer platforms available on the market yet regarding "Austrian Amateur Soccer". By analyzing these few, it can be noticed that they do not have the possibility to create an account. This missing feature is the difference to the application that is developed as part of this thesis, as mentioned in Section 1. Therefore, these soccer platforms do not have the option for creating an individual area with news articles of the favorite soccer teams. However, what our platform and these other platforms have in common is the representation of the news articles, which we will implement during our work. They provide a news feed as well as a detailed view of these articles. Furthermore, all the articles consist of images, title, text, and publication date. One of these analyzed platforms also provides the author of the content.

All these soccer forums implement a feature to share the items on other social media channels. Facebook and Twitter are the ones that are always represented in that feature. Moreover, the news articles include public posts from other social media channels, like Facebook and co., by adding the "embed-links" in their websites.

Nevertheless, these soccer platform's articles always contain only one single image. So, there is no kind of picture carousel or something similar provided. However, in our soccer platform's application, a feature for displaying several images at the same time is needed.

So, other social media platforms, which do not have anything to do with soccer, are examined for getting ideas of such a picture carousel feature.

Many social media channels handle more than one image in a feed, for example, Facebook and Instagram. Facebook is not the appropriate choice for our case because the images of a post are their main focus. Generally, the arrangement and representation of the images cannot be incorporated into our design. Instagram, on the other hand, offers a picture gallery where one can navigate through all images. Besides, there is an indicator for showing how many pictures are available and which picture is displayed at the moment. Those are elements, which can be considered in our design.

4.2.4 Style guide

As already mentioned in the non-functional requirements, the application's appearance should be as clear, as intuitive, and as simple as possible. Hence, all elements which have the same functionality should look the same or at least similar, and the functionalities must be easy to learn and accessible with just a few clicks. Therefore, the selected layout must be uniformly and used consistently in all places. For example, this applies to the news articles available in several places on the website. It is displayed on the team and league page and in the news view. Thus, both the News Feed and the News Detail View must look the same in all places and must have the same functionalities.

4.2.5 Colors

The application should by no means become too colorful and thereby distract from the actual functionalities. In order not to confuse the application, two primary colors are chosen.

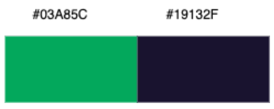


Figure 4.7: Colors with HEX codes.

As shown in Figure 4.7 the base colors used in the application are green and dark blue. So, there are two primary colors to prevent that the application gets overloaded with too many colors. The whole application uses these two colors.

4.2.6 Icons

In order to achieve an intuitive design, icons are used instead of text so that the user recognizes the functionality behind them. Thus, the user should be able to find their way around relatively quickly.

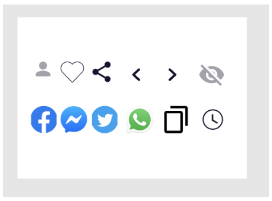


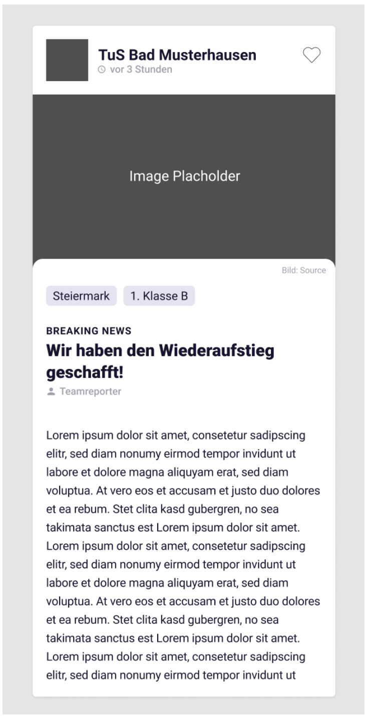
Figure 4.8: The used icons.

In Figure 4.8 the used icons are illustrated. Most of them are Material Design Icons[7], which are an icon set provided by Google. The soccer platform itself creates the icons which are not available in this icon set.

4.2.7 UI Conception

After the requirements have been gathered and structured, and the general rules relating to design have been established, the design requirements can be derived. In the course of the entire development process, the mock-ups have changed again and again. The changes were always discussed and agreed upon with everyone involved. The finalized drafts are recorded and discussed in the following section, divided into the features of displaying news articles, the login process, and the sharing feature.

[7]https://materialdesignicons.com/ [accessed on: 23.01.2021]


TuS Bad Musterhausen

vor 3 Stunden

Image Placholder

Bild: Source

Steiermark

1. Klasse B

BREAKING NEWS

Wir haben den Wiederaufstieg geschafft!

Teamreporter

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut

Figure 4.9: This figure shows the mock-up for the news detail view

News Articles

In the following section, the News Article component is examined and designed concerning the Story Cards defined in Section 4.1.1. The mobile view's mock-ups are represented since there are only minimal differences in the view for larger screens. Only the sizes, respectively, the widths change. How the news articles should be presented remains the same or is very similar. This part is divided into the News Detail View, the News Feed, and the Image Gallery. Each of these components is discussed and visualized utilizing appropriate mock-ups.

News Detail View

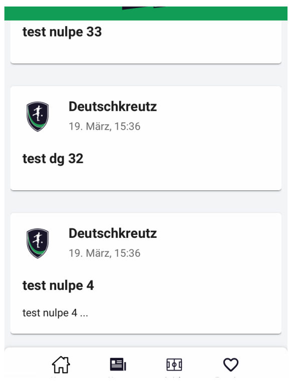
In Section 4.1.2 it was stated that Story Card N8 is the starting point. Therefore, a mock-up relating to the detail view of the news articles is needed. Moreover, Story Cards N9 and N10 are included in this view, as well as Story Card N3.

By clicking on a specific news article, the user has the option to navigate to the corresponding detail view, where all information relating to this article should be collected and visualized. This visualization is represented in mock-up 4.9. The view should be composed of a header, an optional image, the corresponding tags, and the article's content.

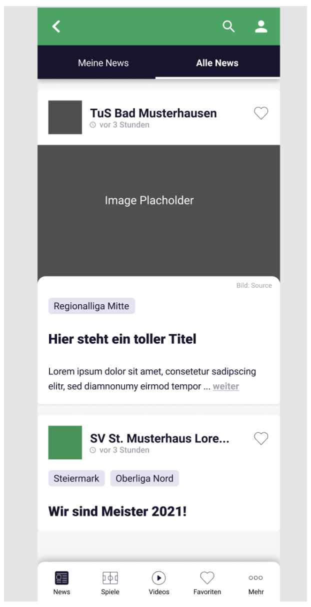
The tags are just simple breadcrumbs, which dedicate the news article to the corresponding soccer state and league. By default, these two tags are displayed, but as already mentioned, the news articles posted by teams are not the only ones. There will also be editorial articles, which will differ a little bit from the team news. Therefore, there will also be an "Allgemein" tag, which means general and indicates that the news article cannot be dedicated to a specific state or league.

The header contains the team logo, the team name, the information, which indicates when the platform published the article, and a heart icon to allow the user to add this soccer club to their favorite teams. When an image was attached to the article, it should be displayed between the header and the text. For that moment, the visualization and the implementation should realize showing just one single picture, which relates to Story Card N4. The article's text should be placed in a container with rounded corners, which protrudes into the image a little bit. In the right corner of this container, an optional picture credit, which indicates the picture's source, should be displayed. The pictures will be discussed in more detail in the further course.

The content consists of an author, a title, and a pre-title. If a team itself writes the article, the author name displayed will be "Teamreporter", in contrast to an editorial article, where the author name is the name and surname. The title is an optional field, as well as the pre-title, which is limited to 22 characters. However, the pre-title is not available for team news. Moreover,



(a) first draft of the news feed



(b) final draft of the news feed

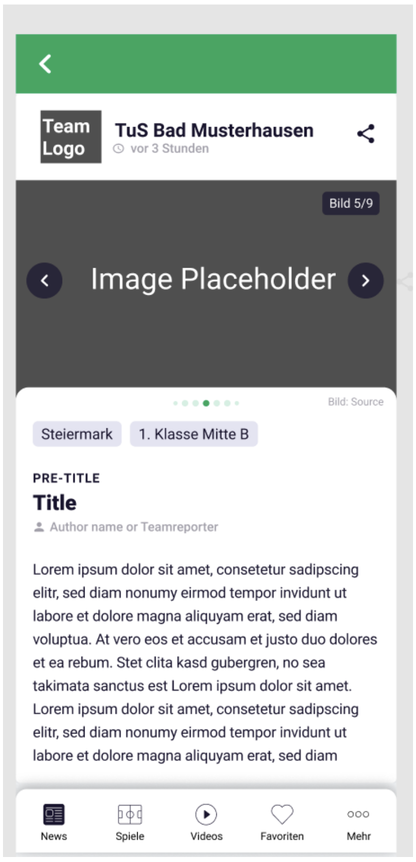
Figure 4.10: This figure shows the first and final version of the news feed mock-up

it is not possible to only have the pre-title and no title. If an editorial article contains a pre-title, then it also has to hold a title. The title has a limitation of 50 characters. Furthermore, the content contains the text itself, which is optional and has no limitation. Due to the editorial news, the dissolution of social media embeds must be supported. Therefore, the required scripts for Facebook, Twitter, and Instagram hast to be added.

News Feed

The News Feed is an overview of the soccer clubs' latest news and relates to Story Card N1. The user can navigate through the articles and get an overview of them. The visualization should be kept as simple as possible to avoid overcrowding with information and a consequent negative user experience. Nevertheless, it should contain all information required to catch the attention of the user. During the development process and after several reflection meetings, changes to the news feed view were made. The first prototype of the mock-up, illustrated in Figure 4.10 on the left side, was not satisfactory and was discarded after some conversations in the meetings. The reason for discarding this first draft is the too minimalistic design and information visualization. Just the title and only a few words from the beginning of the article are not expressive. This non-appealing overview does not catch the user's attention.

Besides, it is not apparent enough that the detail view opens by clicking on the news card. Furthermore, a revision of this mock-up was needed because of the additional requirement that several pictures can be added to the news article, which relates to Story Card N6. After the mock-up for the news detail view was completed, it was decided to adopt this representation for the news feed with minimal deviations. The final mock-up, seen in Figure 4.10 on the right side, shows these deviations. Instead of displaying the full content, as in the detail view, or displaying just a few words, as in the first draft, it should show more characters. This clipped preview text is part of the news article object, coming from the back-end side so that the front-end does not have to handle it. Moreover, a button should be added to the clipped text to convey that the entire content can be seen by clicking on the button or the news article in the overview. Another deviation to the detail view relates to the omission of the author. Furthermore, appending a picture is optional. So, there can be news cards just containing text and no image, which was also considered in mock-up 4.10 on the right side. Considering Story Card N2, there is no additional mock-up needed because it contains the same visualization, as illustrated in Figure 4.10, with the only difference that it is shown within the "Meine News" tab, which are the german words for "My News". Moreover, it has to be considered that the news feed view and the detailed view of the news articles are also available at different locations on this website, which are the team page and league page. However, the visualization does not differ from the standard visualization described above. Therefore, these views have to be implemented modular, so that it is possible to add these views at each position where they are needed.


Team Logo

TuS Bad Musterhausen

vor 3 Stunden

Bild 5/9

Image Placeholder

Bild: Source

Steiermark

1. Klasse Mitte B

PRE-TITLE

Title

Author name or Teamreporter

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam

Figure 4.11: This figure shows the mock-up for the image gallery

Image Gallery

The feature to realize a kind of image gallery, also known as an image carousel, relates to Story Card N6. For allowing a good user experience, the user should be able to navigate through all attached images both in the detailed view of the article and in the news feed. Therefore, the only option to allow visualization in the news feed without position the images below the other and consequently overloading the feed is a picture carousel. In order to derive a good and simple design, other social media platforms were analyzed. Hence, Facebook[8] and Instagram[9] were considered, because both platforms are dealing with a feed and images. For the soccer platform, Instagram was the more appropriate choice. Therefore, the idea with the carousel itself and the carousel dots was assumed. Figure 4.11 shows a visualization of the picture carousel. The user should get the first image displayed and get the possibility to navigate to the next ones and to navigate back to the previous ones. On the one hand, navigation should be possible via clicking the button displayed within the image on the left and right side and, on the other hand, with the keyboard's arrow keys. Moreover, it should be possible to use swiping as well because of the visualization on mobile devices, because users are familiar with the swiping gesture when they are using their Smartphones.

Furthermore, the dots should indicate in which position the user is located at the moment. For example, in Figure 4.11, the user has navigated to image number five of nine. Therefore, there are four previous images and five images next. The maximum number of dots is seven: five big dots and two small dots on each side. The bigger dots are the index of the current image. If the gallery contains three pictures and the user has navigated to the first one, then the first dot should be marked. If the user has navigated to the second one, the second dot should be marked, and so on. The maximum number of the bigger dots is five. So, if the gallery consists of more than five images, the smaller dots should indicate that there are more pictures left in the corresponding direction. More examples are attached in Figure 4.13 to provide an image of this procedure. Story Card N5 is considered as well in this visualization. There is a container in the right corner of the

[8]https://www.facebook.com/ [accessed on: 23.01.2021]
[9]https://www.instagram.com/[accessed on: 23.01.2021]

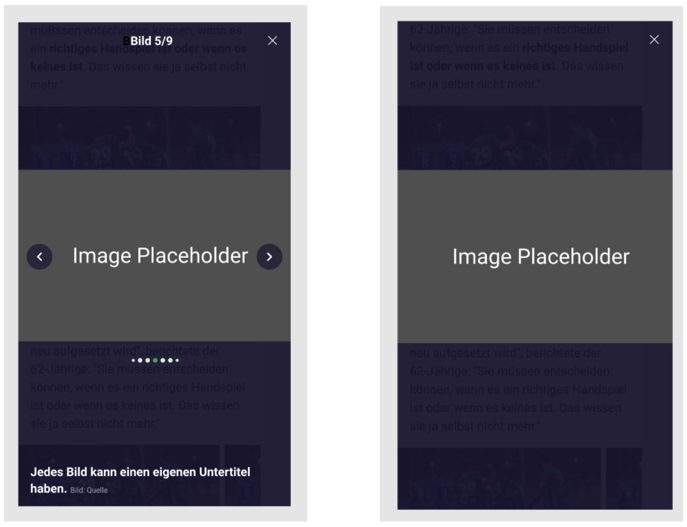


Bild 5/9
Image Placeholder
Jedes Bild kann einen eigenen Untertitel haben. Bild: Quelle
Image Placeholder

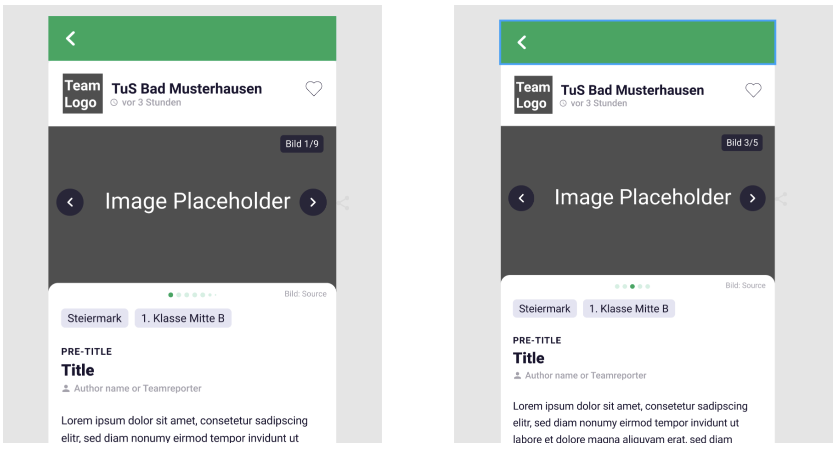
(a) full screen with image carousel

(b) full screen with one single image

Figure 4.12: Two examples of the full screen mode .

image itself. It contains the current index and the number of pictures that are added to the news article.

Furthermore, the images have to be adapted to an aspect ratio of 16:9 so that the carousel's functionality is given. Therefore, if the image does not have the desired size, it must be zoomed to fit the gallery. Nevertheless, the user should be allowed to get the original picture visualized. Figure 4.12 shows the needed full-screen mode of the images. The full-screen mode is reached by clicking on the image in the detail view and has a transparent background. (Clicking on the image in the news feed opens the detailed view of the article.) The container with the current index and the number of images should be available at the top, next to the close button. In the middle of the screen, the picture itself should be displayed in the original size it was attached. Additionally, it should be possible to zoom the image in full-screen mode, as mentioned in Story Card N7. The carousel dots should be positioned beneath the image. At the bottom, the optional picture caption should be displayed. The full screen is the only place where it gets displayed. Next to the caption, the picture credit is positioned.



Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam

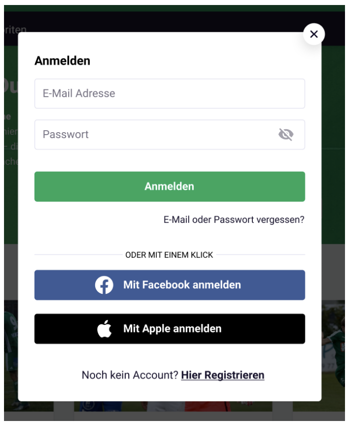
(a) first image of nine is displayed

(b) third image of five is displayed

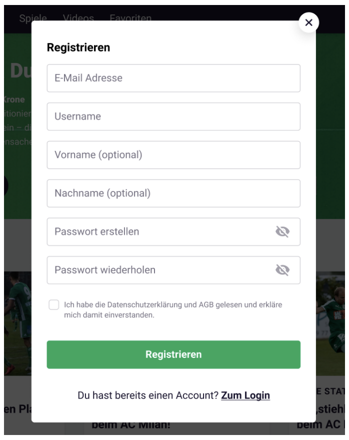
Figure 4.13: Two examples show how the carousel dots should act as indicators.

Registration Process

In the following section, the whole login process views and logic are examined and designed regarding the Story Cards, defined in Section 4.1.1. The registration process views include the login screen, the registration screen, and the screens for editing the user profile and the password. Furthermore, the process comprises the screens for getting help when the user forgot their email address or password and the screen for contacting the support. A dialog window should open with the corresponding content for larger screens, illustrated in the mock-ups below. The only difference to the mobile view is that the dialog extends over the entire screen. The first step is to open the login screen by clicking on the login button in the navigation bar's right corner. The registration screen could be reached from the login window and the window for getting help if the user forgot their password. When they cannot remember their email address, the window for getting help can be reached from the "forgot password" screen. The screen for contacting support can be reached through the "forgot email address" window. All these screens are discussed below.



(a) draft of the log-in window



Ich habe die Datenschutzerklärung und AGB gelesen und erkläre mich damit einverstanden.

(b) draft of the sign-up window

Figure 4.14: This figure shows the mock-ups for the registration and the login process.

Login

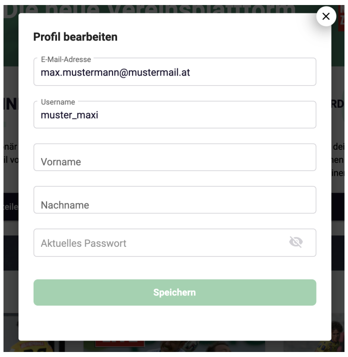
Before the user can create an account on this platform, they get navigated to the login window. The mock-up for the login screen is illustrated in Figure 4.14 on the left side. This representation is related to Story Card L1, L2, and L3. This screen consists of two mandatory input fields for the email address and the password of the user. In the password input field, the user can show the entered password if they click on the eye icon. The submit button is positioned beneath the input fields and has to be clicked after the two form fields are filled in. Beneath the submit button, the user can click on the "forgot email or password" text to navigate to the appropriate screen. As mentioned before, the user can use their Facebook account or their Apple ID for a single sign-on instead of creating an additional account for this platform. The close button is positioned on the top right corner. At the very bottom, the underlined text navigates to the registration window, which is described in the next subsection.

Registration

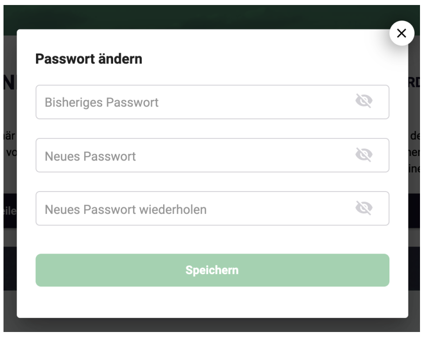
When the user has clicked on the registration text, they get navigated to the registration window, illustrated in Figure 4.14 on the right side. The user has to fill in all necessary data. The email address, the username, the password, and the password confirmation fields are mandatory. Only the name and the surname are optional fields. As defined in the non-functional requirements, the password must consist of at least six characters, while the username must consist of at least 4 and at most 25 characters. No special characters except underline, dash, dot, and the umlauts are allowed for the username. When the user wants to fill out the optional name and surname fields, they must consist of at least 2 and at most 50 characters. Furthermore, the user has to confirm that they have read the privacy policy and the general business terms by marking the form fields' checkbox. As in the login window, the green submit button is at the bottom and the close button in the top right corner. At the very bottom, the user can navigate back to the login screen by clicking on the underlined text.

Edit Account

After the user has successfully created an account and is able to log in, they also want to get the chance to edit the user data, which concerns Story Card L4. The already existing data should be filled in in the corresponding input fields when the user opens the window, represented in Figure 4.15 on the left side. Therefore, at least the email address field and the username field should be filled out beforehand because the name and the surname are optional fields, as mentioned before. The user can change all fields if they want. The user has to enter their password to verify their identity to send the data to the back-end. As mentioned in Section 4.2.2 and illustrated in the sequence diagram 4.6, the username has to be checked if it is available. This availability check has to happen at creating the user account and when the user wants to change it. After the user has filled in all the necessary data, they can click the submit button at the bottom to confirm the changes they made. A separate window opens for changing their password, represented in Figure 4.15 on the right side. It contains an input field for the previous



(a) screen for editing the account data



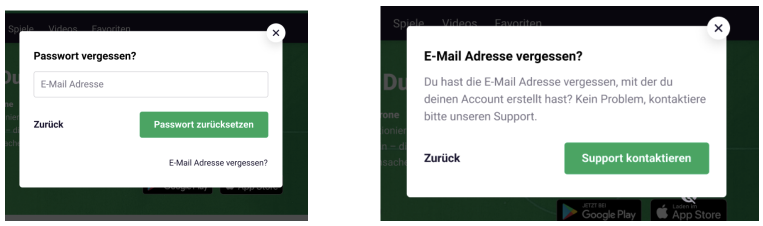
(b) screen for changing the password

Figure 4.15: The figure shows the screens for allowing to change the user data.

password, an input field for the new one, and an input field for password confirmation.

Forgot Email or password

As defined in Story Card L5, the user wants to get help if they forgot their password or email address. The screen for getting help when the user forgot their password could be reached through the login screen. It consists of a single input field for entering the email address, visualized in Figure 4.16 on the left side. After clicking the submit button, the user gets an email sent to the entered email address. The user can click on the text in the lower right corner to get to the window when they forgot their email address, represented in Figure 4.16 on the right side. It consists of an information text that says that the user should contact the support team if they cannot remember their email address. The user has to click the green button in the lower right corner, which leads to the window for contacting the support team explained in the next section.



E-Mail Adresse vergessen?

Du hast die E-Mail Adresse vergessen, mit der du deinen Account erstellt hast? Kein Problem, kontaktiere bitte unseren Support.

(a) Window when the user forgot their password

(b) Window when the user forgot their email address

Figure 4.16: The figure shows the screens for helping the user when they forgot their password or email address.

Support

As defined in Story Card L6, the user needs to contact support if they cannot remember their email address. The representation of this screen, visible in Figure 4.17, consists of in pre-filled input field for the subject, which can be changed, of course, an input field for the phone number, and a text area field for describing the problem. As defined in the non-functional requirements, a honeypot has to be implemented for this field to avoid spam. The subject field only contains a support request. After entering the necessary information, the user can send their request to the team by clicking the lower right corner's green button.

Sharing

Story Card S1 describes the feature to share the articles on other social media platforms. There are three different visualizations because the view for smaller screens differs from that for larger screens. There are articles written by an editorial team instead of the soccer club, as mentioned in Section 4.1.1 relating to news articles' different visualizations. It does not make sense to show the heart icon because there is no option to follow the editorial team. Therefore, after some discussions, it was decided to show a sharing button instead, which is represented in Figure 4.18. Simultaneously, it was decided to do the same with the articles, where the user is already following the team. This visualization relates to the news detail view as

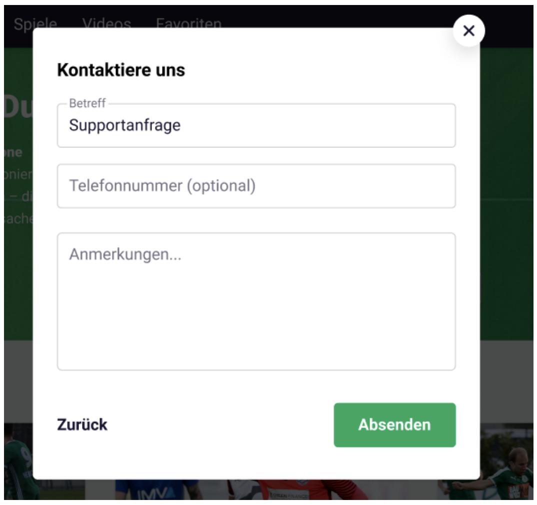


Figure 4.17: This figure shows the mock-up for the support contacting screen

well as the news feed view. By clicking the sharing button, a drop-down with the different sharing options opens. It should be possible to share the article on Facebook, via Facebook Message, on Twitter, and WhatsApp. Besides, it should be possible to copy the link and share the article wherever the user wants. Sharing the news article via Facebook Message differs if the user uses their smartphone because then the article should be sent via Facebook Messenger, which is only available for smartphones. If the user uses their Laptop or something similar, the article should be sent via the "Send Dialog"[10] from Facebook, which is available via Facebook-SDK for JavaScript.

Furthermore, the Web Sharing API should be supported for mobile phones to allow opening the sharing screen of the respective system in mobile browsers as this permits sharing on all kinds of apps. If the browser does not support this functionality, a fallback solution is given with the implementation explained further.

The visualization for the mobile view can be seen in Figure 4.19 on the left side. It represents the detail view of a specific news article, where the

[10]https://developers.facebook.com/docs/sharing/reference/send-dialog/ [accessed on: 09.02.2021]

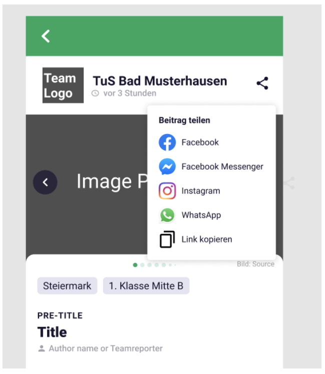
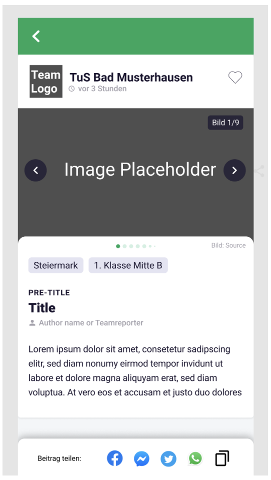


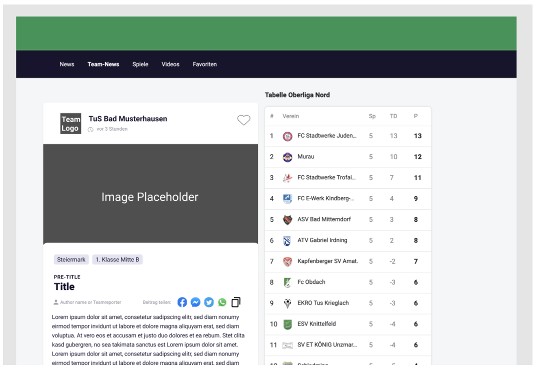
Figure 4.18: This figure shows the mock-up for the sharing button with the corresponding drop down

buttons for sharing are arranged in a bar at the bottom. The sharing options for the news feed are only available via the sharing button as described above. The representation for larger screens, shown in Figure 4.19 on the right side, differs in that the sharing buttons are visualized to the author's right.



Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores

(a) Representation of the sharing-bar for small screens



Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam

Tabelle Oberliga Nord

#	Verein	Sp	TD	P
1	FC Stadtwerke Juden...	5	13	**13**
2	Murau	5	10	**12**
3	FC Stadtwerke Trofai...	5	7	**11**
4	FC E-Werk Kindberg-...	5	4	**9**
5	ASV Bad Mitterndorf	5	3	**8**
6	ATV Gabriel Irdning	5	2	**8**
7	Kapfenberger SV Amat.	5	-2	**7**
8	Fc Obdach	5	-3	**6**
9	EKRO Tus Krieglach	5	-3	**6**
10	ESV Knittelfeld	5	-4	**6**
11	SV ET KÖNIG Unzmar...	5	-4	**6**

(b) Representation of the sharing buttons for large screens

Figure 4.19: Visualization of the sharing feature for small and large screens.

4.3 Implementation

This section covers the implementation of the requirements, defined in Section 4.1 and designed in Section 4.2. First of all, it gives an overview of the implementation basics, describing the entire development process. Furthermore, the general structure of the project is shortly described and represented with some listings. Afterwards, the navigation via the Angular Router is explained to give an understanding of how the navigation in this application works. Finally, the implementation of the concept defined in Section 4.2 follows.

4.3.1 Implementation Basics

This section deals with the choice of the used technology. It describes why the selection fell on Angular. The development process is also explained in detail to give an overview of how this project is developed. Furthermore, it describes the procedure to check whether the features can be classified as resolved.

Technology Selection

The Angular Framework, summarized in Section 2.4, is used in the course of this work. The choice of a suitable software architecture depends on many factors like the type of software application and the problem. Last but not least, it also depends on the people included and their experience in software development. Based on these facts, the choice fell on Angular because the soccer platform has already had good experiences with this framework. Furthermore, TypeScript is used as recommended by the Angular Team itself because of the many provided advantages, as mentioned in Section 2.4.

Development Environment and Process

Section 4.2.7 defines all the required mock-ups before the development process can start. At the beginning of the process, the requirements have to be clearly defined, to be able to derive the design. Thus, Section 4.1.1 records and examines these requirements more closely and divides them into functional and non-functional requirements. The entire team held regular discussions to talk about the logic and the realization of it. A specially designed system for this purpose records all these requirements. At the beginning of the development process, Trello[11] was used for this purpose. The discussions took place on-site, as everyone was always present on the same days. However, only the product owner and the respective developer had access to the Trello board, which turned out to be a disadvantage. It was not always clear who was currently working on which issue. After a while, we switched to the Gitlab[12] board, as the full source code for all projects is there, and it is easier when one system is used instead of several at the same time. Moreover, each team member has access to each Gitlab board. Every team member uses an own board. The panel starts with a Backlog with all open issues. When the development of an issue starts, it gets moved to the ″in progress″ status. When the development process finishes, the issue gets transferred to the ″in review″ status and ″testing″ status. ″In review″ means that the developer made a Merge Request, and two other developers have to review the code and approve it. Simultaneously, the testers of the team have to test the feature in detail. Only if the issue moves to ″testing ok″ by the tester and the other developers approve the issue it can be merged. Otherwise, the developer of the issue has to implement the requested changes by the approver or fix that issue because of the ″testing failed″ status. After merging the issue on the master branch, it can be closed.

Besides, a jour fixe[13] was introduced to include all members involved to discuss who implements which issue and whether someone needs help. At the same time, sprints were introduced where everyone divided up the issues that they thought they could implement during this time. Initially, the

[11]https://trello.com/de [accessed on: 20.02.2021]
[12]https://about.gitlab.com/ [accessed on: 20.02.2021]
[13]https://www.teamazing.at/was-ist-ein-jour-fixe/ [accessed on: 20.02.2021]

sprints lasted a week. During development, this turned out to be disadvantageous, and the sprints were extended to two weeks. A list for the sprint was added to the Gitlab board. Another communication channel that we use is Slack[14] . There are channels for all kinds of purposes. If someone notices that something is broken or does not work as expected, they communicate over these channels. The user feedback is also shared here.

For development, an Integrated Development Environment (IDE) is used, precisely the IDE "WebStorm" from Jetbrains.

4.3.2 General Structure

The main starting point of the application is the main.ts file. This file is responsible for bootstrapping the application by passing the app module to the method, as shown in listing 4.1.

Listing 4.1: The main.ts file which bootstraps the angular application

```
import { AppModule } from './app/app.module';

document.addEventListener('DOMContentLoaded', () => {
  platformBrowserDynamic().bootstrapModule(AppModule);
});
```

The app module refers to the app.module.ts file, illustrated in listing 4.2 and is the most important part when starting the application. In this file, it is defined which components angular needs for rendering the index.html file. The App component itself is added in the index.html file. The index.html file is the central file of the project. In this file, the basic framework gets defined. It includes all required libraries, all necessary files for displaying, and the program logic.

[14]https://slack.com/intl/de-at/ [accessed on: 20.02.2021]

Listing 4.2: The app.module.ts file which defines that the App component is needed when the app is bootstrapped

```
@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [... ],
  providers: [...],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

The website consists of three different parts: the team page, league page, and home page. Each of these pages has an own module. The team page and league page are not discussed in detail because this work focuses on the home page.

4.3.3 Navigation via Angular Routing

The Angular Router is used to load the corresponding components regarding the different routes. There are different routes for the news feed and the news detail view. Since these are entirely different views, different components with their templates are needed. Besides, the Apple Login requires a separate route and an individual component for handling the user's API-access-token. The routing of these views is explained in this section.

News Feed and news detail view Routing

The Angular Router is used to provide the navigation to the news feed and the news detail view. As already stated, the application's starting point is the App module with the declared App component. The Routing gets included in the template of the App component through the inclusion of <router-outlet ></router-outlet >. The routes get added by importing the corresponding module, where the routes themselves are defined. In our case, the Home module has to be loaded. The Home module only gets loaded when the user is neither located on a team page nor a league page. The Home module is the location where the News module with the necessary routes gets loaded.

Listing 4.3: Loading of the News module, when path is composed of "news" (home.routing.ts)

```
export const routes: Routes = [
  {
    path: 'news',
    loadChildren: () =>
        import('@modules/home/news/news.module').then(m =>
        m.NewsModule),
  },
    ...
];
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
```

Listing 4.3 shows that the Home module loads the News module when the path of the URL contains *"news"*, while Listing 4.4 represents the routes, that load the corresponding components.

Listing 4.4: The news.routing.ts file where the necessary routes for the Detail View are defined.

```
export const routes: Routes = [
  {
    path: '',
    children: [
      {
        path: '',
        component: NewsPageWrapperComponent,
          children: [
          { path: '', component: NewsPageComponent },
          { path: 'meine', component: NewsPageMyNewsComponent },
        ],
      },
      { path: ':id', component: NewsDetailComponent, resolve:
          {newsEntry: NewsResolver}},
    ]
  },
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
```

The NewsPageWrapper component gets loaded when the user navigates to the following URL: *"https://${websiteURL}/news"*. This URL contains the domain of the soccer platform and the "news" path. This components' template displays two tabs, namely *"Meine News"* and *"Alle News"* as captured in Figure 4.10. As the name suggests, this component is just a wrapper. The corresponding components for the standard news feed and the personal news feed have to get loaded in this position. Again the loading of the corresponding components happens via routing. As illustrated in the listing 4.4, the NewsPage component gets loaded when the user navigates to the standard news feed, the "Alle News" tab. Equally, the NewsPageMyNews component gets loaded when the user navigates to the personal news feed, the "Meine News" tab, which can be reached via *"https://${websiteURL}/news/meine"*. In both templates, the NewsList component is included in the HTML via

<app-news-list ></app-news-list >. This component is the actual news feed and contains the NewsCard component for displaying each single news article.

Listing 4.4 shows the route that provides the navigation to a specific news article when the user clicks on a specific article. It is the last route shown in this listing. The NewsDetail component's template is reachable through the news path attached with the unique ID of the article.

Routing for the Apple Login

Another route is required to provide the ability to use Apple single sign-on. The "Implementation of the concept" section below will explain the implementation of the process itself. After completing the authorization process on the Apple side as well as on the back-end side, the required API-access-token in the query parameters of the URL gets sent to the front-end to the following route: *"https://${websiteURL}/apple/login?= api_access_token=api-access-token"*. The API-access-token is the intern token required in almost every API-Call, as explained in the section of the "REST API Overview" in Section 4.2. Listing 4.5 shows the component that gets loaded when the URL contains *"apple/login"*. That component is responsible for requesting the user from the back-end with the corresponding API-access-token and logging the user in.

Listing 4.5: The route and component defined for the Apple Login in the app-routing.module.ts

```
export const routes: Routes = [
  {
    path: 'apple/login', component: AppleDigestLoginComponent
  },
  ...
];
```

4.3.4 Implementation of the Concept

This section deals with the implementation of the Story Cards. Therefore it explains how the functional and non-functional requirements are implemented with the Angular Framework.

News

This section concentrates on the implementation of all parts regarding the news articles. Therefore, the implementation of the news detail view, as well as the news feed, are explained below.

News Detail View / Story Card N8

As mentioned several times before, the starting point for the implementation is Story Card N8, the Detail View of the news article. Listing 4.4 declares that Angular loads the NewsDetail component when the URL path contains the 'news' route with the news identifier attached to it. The resolver, more precisely the News resolver, gets called first, and the code gets executed. The News resolver takes the ID from the URL with the help of the Angular Router interface. The resolver calls the News service that is responsible for all API calls regarding the news articles. The News service sends an HTTP GET request to the back-end with the ID. The back-end returns the news entry object if it exists. If it does not, the resolver redirects to the PageNotFound component, a simple template to show the user that the requested page does not exist.

The team and the product owner have decided that the news detail view must support social media embeds. The dissolution of the embeds happens in the NewsDetail component. The index.html file must include the corresponding Facebook SDK, Twitter Widget, and Instagram Embed libraries to allow using this mechanism.

In addition to the actual news article, the detail view template consists of the league table, a component that displays advertising, the soccer game table, and the sharing bar, defined in Figure 4.19 on the left side. A separate component, the NewsPost component, contains the actual news article itself to avoid an unnecessarily large template that would not be easy to read and maintain. The NewsPost component is subdivided into individual components providing a modular structure. Furthermore, these components can be used anywhere in the application independently. The NewsPost component contains the following components:

- NewsHeaderComponent
- GalleryComponent
- NewsBreadcrumbsComponent
- SharingButtonsComponent

The NewsHeader component contains the team link to allow the user to navigate to the corresponding team page by clicking on the team name. Furthermore, the header displays the time when the article was published (Story Card N10). The sharing button is also available in the header, which is a separate component and discussed in Section 4.3.4 . Moreover, the header holds the author's object, which consists of the name and the logo, to show the club's name or the editorial team name (Story Card N9).

Finally, the current user is needed to check if the user already follows the team. If the user follows the team, the sharing button has to be displayed. If the team is not added to their favorite teams, the heart icon has to be displayed. The AppGallery component, or the image carousel, receives all images via input and is discussed in Section 4.3.4. The Breadcrumbs component is the visualization of the tags.

The content or text of the news article gets added via innerHTML property binding. Besides, a pipe for the built-in Angular DomSanitizer is implemented and used to prevent Cross-Site Security bugs. It sanitizes the value to be safe. Listing 4.6 illustrates usage of this pipe to add the news article content.

Listing 4.6: Content of the news entry added via innerHTML property binding and DomSanitizer Pipe.

```
<div [innerHTML]="(entry | async)?.content | safe: 'html'"
    class="news-detail__content"></div>
```

The SharingButtons component, discussed in Section sec:sharing-s1, needs the news ID to share the corresponding article on other social media platforms.

News Feed / Story Card N1

By navigating to the news feed, the NewsList component gets loaded as mentioned in Section 4.3.3. This component contains a separate component for the news article card. Listing 4.7 shows how such an implementation of the template could look.

Listing 4.7: This listing shows how the news feed template could look.

```
<ng-container *ngFor="let entry of newsEntries; let i = index">
  <news-card
    *ngIf="entry !== undefined;"
    [entry]=entry>
  </news-card>
</ng-container>
```

By using the ngFor structural directive, the news card gets displayed as often as the length of the array containing the articles. In our case, a limitation for ten articles is set. Therefore, ten articles get displayed in the news feed. The user can click on a ”more” button to get ten more articles. The NewsList component shares the news entry with the NewsCard component.

The NewsCard component itself consists of the following components:

- NewsHeaderComponent
- GalleryComponent
- NewsBreadcrumbsComponent

These are the same components used in the news detail view. The NewsCard component does not display the entire text of the article, like the NewsPostComponent. On this occasion, just the preview text gets displayed. The preview text is a separate property of the news entry object. Like in the NewsPost component, the preview text gets sanitized by the DomSanitizer before it gets rendered in the template.

Image Gallery / Story Card N6

Story Cards N4 and N6 define the requirement to add images to the visualization of the news articles. The image gallery component implements this visualization. This component gets used in the news feed and the news detail view as well. The image gallery consists of a container visualizing the Story Card N5, showing the current picture's index and showing the number of images attached to the news article. The full-screen gets activated once the user clicks on the image in the news detail view and the position of the image changes. Furthermore, the background color of the container changes. The gallery itself works the same in full-screen mode as in non-full-screen mode, except in full-screen mode, the user can zoom the image. Story Card N7 declares the requirement that a user wants to be able to zoom the pictures. The full-screen mode closes by clicking the close button, clicking outside the gallery, or using the escape key.

The image gallery component's content consists of an individual component that handles the displaying of the images. Therefore, the image gallery component shares the pictures with this child component, which is called AppCarousel. The AppCarousel implements the zoom functionality, the swipe functionality, and the navigation via button and arrow keys functionality.

Moreover, the image gallery component also contains the component that implements the carousel dots and the added picture credit and picture text.

Login Process

The following section discusses the registration process's implementation, which includes the login, the registration, the possibility to change user data and password, the screens for getting help when the user forgot their password or email address, and the contact support screen. Besides, the section also includes the features providing Facebook Login and Apple Login as well. The implementation of each component regards the functional and non-functional requirements and the conception of them.

Login and Registration / Story Card L1

When the user clicks the login button, the sign-in window, realized as defined in Figure 4.14 on the left side, gets loaded. By clicking the button, an event gets triggered, opening a modal dialog provided by Material Design. The Dialog loads the SignInWindow component with the corresponding template. The template consists of two input fields for entering the email address and the password, a submit button and the links for navigating to the registration, and the window for getting help if the user has forgotten their email address or password.

The controls for the Facebook login and the Apple login are not realized at this moment.

By clicking the link for navigating to the registration process, a click event gets emitted, opening the Registration component, which again is loaded by a Material Design dialog. This template is implemented as illustrated in Figure 4.14 on the right side, and consists of six input fields for the corresponding entries. The "required" HTML attribute sets the email address, the username, and the password input as mandatory. Furthermore, validators for the username, email, name, surname, and password fields are created to accomplish the non-functional requirements, defined in Section 4.1.1. Listing 4.8 represents how one can implement such validators.

Listing 4.8: Implementation of Validators for input fields.

```
this.formGroup = formBuilder.group({
    'username' : ['', Validators.compose([Validators.required,
       Validators.minLength(4),
       Validators.pattern('^[a-z0-9A-Z-.äöüÄÖÜß_]*'),
       Validators.maxLength(25)])],
    'email' : ['', Validators.compose([Validators.required,
       Validators.email,
       Validators.maxLength(255)])],
    'first-name' : ['' , Validators.compose([
       Validators.pattern('([^\u0000-\u007F]*[a-Àzÿ-A-Z-.\\s]*)*'),
       Validators.minLength(2), Validators.maxLength(50)])],
    'surname' : ['', Validators.compose([
       Validators.pattern('([^\u0000-\u007F]*[a-Àzÿ-A-Z-.\\s]*)*'),
        Validators.minLength(2), Validators.maxLength(50)])],
    'password' : ['', Validators.minLength(6)],
    'passwordConfirm' : ['', Validators.minLength(6)],},
{ validator: passwordMatchValidator });
```

According to listing 4.8, three Angular built-in validator functions validate the username input. These functions are the minLength function setting the minimum length of the username to four characters. The maxLength method sets the maximum length to 25 characters. The pattern function with a regular expression allowing the german alphabet with umlauts, dash, dot, and underline.

The require validator function marks the input as mandatory The email validator function validates the input if it is an email address containing the @-sign. Again, the maxLength method gets used to set the maximum to 255 characters. The name and surname use the same validators. The minLength setting the length to a minimum of two characters and the maxLength setting the maximum number to 25 characters. Besides, the pattern validator allows all UNICODE characters so that even Chinese ideographs can be used. The password input and the password confirmation use the minLength validator, setting the minimum length to six characters. Moreover, the last line sets the passwordMatchValidator, which is a custom function. It checks if the password input value evaluates to the same entered in the password confirmation input. If not, it returns passwordMismatch: true and an error message gets displayed to the user. The same is true when some of the built-in validators evaluate to false. The input fields get validated while the user enters the data. The user does not have to click the submit button for the validation. Besides, the submit button is disabled until the user has entered all required data, which has to be successfully validated. The registration process is implemented according to the sequence diagram, as seen in Figure 4.6. Therefore, the User service checks if the username is available when the user clicks the submit button.

The User service handles all calls corresponding to the user. The user object consists of a unique ID, an API-access-token, an email address, a username, and many more. Therefore, the User service is responsible for storing, getting, and deleting the API-access-token, logging in the user, or checking if the username is available. For checking the username availability, an HTTP GET request to the back-end is sent. The username is attached to the URL. The response consists of a ″free″ field with a boolean as value. It is set to true when the username is still available and false when it is not. When it is not free, the user gets an error message, and when it is free, the user gets created with an HTTP POST request. After successfully creating the user, the User service logs the user in by storing the API-access-token locally at the user's client. Again if something does not work as it should, the user gets an error message.

The same process applies to the login window. The email input field uses the validators to set the field as required and to check if the input is a valid email address. The password input field uses the validator to set the field as required. When the user has completed the form, the submit button is enabled. When the button gets clicked, the User service gets called to login the user. It is the same function call and the same process as explained before in the registration process.

Edit Account / Story Card L4

By clicking on the user data button, a small menu gets opened. The menu contains the option to change the profile data and the password as defined in Story Card L4. When the button for changing the profile gets clicked, a Material Design Dialog opens where the Registration component gets loaded. However, this time, an optional configuration object is attached. The object contains a parameter to communicate that the user wants to edit their profile instead of creating a new account. Therefore, the template gets loaded with settings to change the registration window. These settings adapt the template to look as described in Figure 4.15. The already existing data gets automatically filled in in the input fields. The user has to enter their password to be able to change the data. The username can only be changed when it is still available. Therefore, the corresponding request to the back-end has to get called. It is the same process as described in the login process. The User service sends the changed data to the API service, where an HTTP PUT request gets executed to overwrite the user data in the database.

When the button for changing the password is clicked, a Material Design Dialog opens where the corresponding component gets loaded. This component consists of three input fields. One for the actual password and one for the new password. Like in the registration template, a confirmation field is added. By clicking the submit button, the API service gets called to save the data in the database through an HTTP PUT request.

Forgot Email or Password / Story Card L5

The dialog for getting help when the user has forgotten their password consists of a single input field for entering the email address. When the submit button gets clicked, the user receives an email to the entered email address with the link for resetting their password.

When the user has forgotten their email address, they can click on the text right after the submit button on this screen. A separate dialog opens just containing the information to contact support for getting help with this issue. When the submit button gets clicked, an individual component gets loaded to provide contacting support. The Section 4.3.4 discusses this component in detail.

Contact Support / Story Card L6

The template for contacting the support team consists of a subject field, an optional field for the phone number, and a text area for the user's message as defined in the corresponding Figure 4.17. Because of the honeypot implementation, this time, the fields occur twice, even when they are not visible to the user. These fields should block spammers and bots. The corresponding settings in the styling sheet are responsible that these fields are not visible to anyone. These settings define that the fields are displayed but not seen. These hidden honeypot fields have common names as "phone, subject, and comment" to lure the spammers and bots. The "real" fields, the visible ones, have strange random names like "namedksjfldjflj". With this pattern, the bot cannot recognize what these fields are. The bot would use the hidden inputs with the common names to fill.

The component does not send the message to the back-end when one of these fields gets touched. The component only calls the API service when the user uses the real fields. Then, the API service sends an HTTP POST request with the message attached to the body.

Facebook Login / Story Card L2

Additionally to the standard login process, a feature is needed to provide a login via Facebook as described in Story Card L2. For this purpose, Facebook provides a Facebook-SDK for JavaScript [15], which has to be imported in the application. Besides, a Facebook developer account and a registered Facebook-App are required. The soccer platform already provides a Facebook Login Process on the website for reporters, so the required app ID and other initialization settings are already registered in the Facebook developer account.

The method for adding the Facebook-SDK is defined in the Facebook service and is called in the App component. When the SDK gets loaded, the configuration for the authorization object gets set too. Listing 4.9 shows the process of loading the SDK.

Listing 4.9: Configuration of the Facebook authorization object.

```
window.fbAsyncInit = () => {
  FB.init({
    version: 'v7.0',
    appId  : '[APP_ID]',
    status : true, // check login status
  });
};
```

The window.fbAsyncInit method loads the Facebook-SDK and calls the init function of the SDK. The init function contains all the necessary configurations required for the authorization. Therefore, the app ID, registered in the Facebook developer account, has to be added here as well as the used

[15]https://developers.facebook.com/docs/facebook-login/web/ [accessed on: 27.01.2021]

version. The SDK tries to get info about the current user right after the init by defining the status to true. A button is needed in the SignInWindow component's template to allow using the Facebook login.

The component injects the Facebook service, and when the user clicks the Facebook login button, a method from that service gets called. This method calls the getLoginStatus function from the Facebook-SDK, which can be used to request if a user is logged in via Facebook. It returns the authentication object attached with the status of a user. The status could be connected, not authorized, or unknown. Connected means that the user is connected to Facebook as well as to this website. Not authorized means that the user is connected to Facebook but not connected to this platform. Unknown means that the user is not connected to Facebook, and therefore, it is unknown if the user is connected to this platform. Furthermore, the authentication object consists of an authResponse if the user status results connected, which consists of accessToken, expiresIn, reauthorize_required_in, signedRequest, and userID. The accessToken is the access key for the user on the website. The expiresIn field contains the time when the key expires. The time is a UNIX-timestamp. When the token expires, the user has to log in again. The signedRequest is a signed parameter containing information of the user. Last but not least, the userID is the ID of the user, as the name says.

The access token is crucial to be able to login the user. When the user is connected, and the getLoginStatus request returns the authorization object, the application can query the access token from that object. If the user is not connected, the Facebook service has to login the user. In that case, a method gets executed, where the FB.login method from the Facebook-SDK gets called. Listing 4.10 shows an example of how such a call can look.

Listing 4.10: The Login method provided from Facebook-SDK.

```
async loginViaFacebook(): Promise<any> {
  return new Promise(async (resolve) => {
    FB.login(function(response) {
        resolve(response);
      }, {scope: 'email, publish_pages'}
    );
  });
}
```

Field	Type	Description
success	JSON	indicates if the operation has been successful
registration	JSON	indicates if the user has been created the first time
user	JSON	the corresponding user object

Table 4.3: Possible responses when /users/fblogin/:token has been requested successfully.

The optional list defines which permissions are needed by the website. Therefore, the user has to agree that the website can access the data specified in that list. In our case, the platform is allowed to access the email address and the public profile when the user permits. Afterwards, the authentication object with the user status gets returned, and the access token can be queried.

In the next step, the User service needs the access token to login the user. To do so, the API service gets called and sends an HTTP POST request. In our case the endpoint URL looks like this:
"https://${websiteURL}/users/fblogin/${access-token}". The back-end sends a response that can look like described in Table 4.3. Either the response consists of success with the corresponding user object or registration with the corresponding user object. After successfully receiving the user object, the API-access-token gets stored locally at the user's client, and the representation of the user gets saved. Finally, the AppState service sets the corresponding state to set the user to logged in. If this process fails, an error gets thrown, and the user receives an error message.

Apple Login / Story Card L3

In Story Card L3 the requirement of implementing the single sign-on feature via Apple is defined to lure more users since they do not have to create a separate account for this soccer platform. To allow Apple single sign-on, there are some things needed. First of all, an Apple Developer Program Account[16] is needed in order to provide the Apple login service in the application. Since the soccer platform offers an application for iOS, this

[16]https://developer.apple.com/support/compare-memberships/
[accessed on: 27.01.2021]

was no challenge. In association with the platform's iOS developer, a client ID and a redirect URI, which is crucial because the authorization object's configuration needs them, was registered via the Apple developer console. Nevertheless, before configuring that object, the "Sign in with Apple JS" framework [17] has to be embedded in the web page as well as the button for triggering the single sign-on flow.

The SignInWindow component's template includes the button just below the single sign-on via Facebook control. The button gets displayed by opening the login window, and the Apple JS library loaded. A separate service, in our case the AppleSignIn service, implements the loading process of the library. The SignInWindow component injects this service to access the corresponding function. After successfully loading the library, the init method, which configures the authorization object, gets executed. Listing 4.11 shows an example of how such a configuration can look.

Listing 4.11: Configuration of the Apple authorization object.

```
init(): void {
  const redirectURI = this.apiService.getAppleRedirectUri()
  AppleID.auth.init({
    clientId: '[CLIENT_ID]',
    scope: 'email name',
    redirectURI: redirectURI,
  });
}
```

The init method gets called using the Javascript API. All crucial parts get set in this function. The client ID is the one defined in the Apple developer console, as well as the redirect URI. The redirect URI is the URI to which Apple posts the authentication result. Apple does not allow to define local URLs or IP addresses as redirect URI, so it must include a domain name. The API service defines all endpoint calls as well as the redirect URI. The AppleSignIn service injects the API service, which returns the redirect URI. The scope is the requested data of the user. So, in this case, the email address and the name of the user are asked. It is essential to know that Apple does

[17]https://developer.apple.com/documentation/sign_in_with_apple/sign_in_with_apple_js [accessed on: 27.01.2021]

send this user information only once when the user authenticates the first time. So, this data has to be saved in the back-end.

By clicking the Apple login button, the signIn method from the Apple JS library gets called. Listing 4.12 represents an example of the call of that signIn method.

Listing 4.12: Apple's sign-in call of the Apple JS library.

```
public async signIn(): Promise<AppleSignInAuthorizationObject> {
  try {
    return await AppleID.auth.signIn();
  } catch (e) {
    throw e;
  }
}
```

With this function call, the authentication process with Apple starts. The user gets redirected to Apple's single sign-on page, where the user has to permit to use their Apple ID for this platform. After submitting the authorization, Apple sends the authorization response to the defined redirect URI. The back-end has to handle the authorization response, which is not discussed in detail as it is not part of this work.

The authorization response consists of a code, a single-use authentication valid for five minutes, and an id_token, a JSON web token consisting of the users identifying information. Furthermore, it contains the user object when the user authorizes for the first time. The user object contains the requested data, which is the email address and the name. After the back-end has decoded the identity token, it sends the intern user's API access token to following route: "https://${websiteURL}/apple/login?=api_access_token={api-access-token}". For allowing the front-end to get this access token, a route for that path gets added to the Angular Router. By navigating to that route, a component, the AppleDigest component, gets loaded. This component access the token, login the user and navigates back to the landing page. In case of failure, the component navigates to the landing page with the sign-in window opened, and an error message is displayed. Listing 4.13 shows the implementation of the AppleDigestLogin component and how such handling could look.

Listing 4.13: Implementation of the AppleDisgestLogin component

```
async ngOnInit() {
    const api_access_token =
       this.route.snapshot.queryParams['api_access_token'];
    const loginFailed = this.route.snapshot.queryParams['failed'];

    if (!api_access_token || loginFailed) {
      this.handleError();
      return;
    }
    try {
      await this.userService.appleDigestLogin(api_access_token);
    } catch (e) {
      this.handleError()
      throw e;
    }
  }
private handleError() {
    this.router.navigate(['/']);
    this.appStateService.set(this.appStateService.OPEN_SIGNIN_WINDOW,
       'digestLoginFailed', true);
  }
```

The Angular ActivatedRoute interface gets used to traverse the RouterState tree to get the query parameters, where all information about a route can be accessed. This interface provides access to the API-access-token or the failed parameter. When there is no value for the token or the failure value is available in the URL parameters, the handleError method gets called. This method navigates the user back to the landing page by calling Angular's Router. The AppState service triggers the sign-in window to open with a corresponding error message.

If none of both cases arrive, the User service gets called to send an HTTP GET request with the API-access-token. The back-end returns the user object to store the access token in the local storage and save the user. Finally, the application navigates the user back to the landing page.

Sharing / Story Card S1

The user can share news articles by clicking on the sharing button in the news header or by clicking on one of the buttons in the sharing bar. The sharing bar contains all the buttons for the platforms on which the articles can be shared. Only the news detail view provides the sharing bar, but both the news feed and the detail view contain the header with the sharing button. By clicking the sharing button in the header, the drop-down with the different sharing options opens. Story Card S1 describes that feature and the Figure 4.18 and 4.19 defines its design. The sharing works for Facebook, Twitter, and WhatsApp. Besides, it should be possible to copy the link. The different sharing button options are implemented as an individual component, the SharingButtons component. This component is included in three different places. One of these places is the component for the sharing bar in the detail view for small screens. The second place is the drop-down menu, which opens by clicking the news header's sharing symbol. The last area where the component is included is the news detail view for larger screens next to the author. The SharingButtons component requires the news ID as input to request the sharing object from the back-end, which is illustrated in sequence diagram 4.5. The sharing object contains the title, the description, the content URL, and the image URL. When the user clicks on one of these buttons, a new tab opens, and the user can share the article on the corresponding platform. Listing 4.14 shows how such a call looks when the user wants to share the article on WhatsApp, for example.

Listing 4.14: Implementation of the sharing feature for WhatsApp.

```
public shareOnWhatsApp() {
   const win = window.open('about:blank', '_blank');

   this.getSharingLink().then(url => {
     win.location.href =
         `https://wa.me/?text=${encodeURIComponent(url)}`;
   });
 }
```

The window.open method opens a new tab or window corresponding to the browser settings where the URL gets loaded. The getSharingLink function requests the sharing object from the back-end and sets the URL afterwards. The URL that gets loaded is the click-to-chat function provided by WhatsApp to begin a chat with someone without having their phone number.

The SharingButtons component also includes the implementation of the Web Sharing API usage, which invokes the native sharing mechanism of the device.

4.4 Requirements Comparison

After implementing the requirements, defined in Section 4.1.1, a comparison of the achieved functionalities and the requirements is described. This comparison emphasizes if the requirements are fulfilled. All the implemented features have been tested by testers before they are classified as resolved. First of all, the implementation is compared with the functional requirements. Afterwards, the comparison with the non-functional requirements follows.

4.4.1 Functional Requirements

The next few paragraphs compare the implementation with the functional requirements.

Story Card N1

The visualization of the news feed has been fully implemented. The user gets an overview of all the latest news when he navigates to the website's corresponding tab. The news articles are visualized as small news cards with the images and the title as main attributes on these cards.

Story Card N2

This Story Card also has been fully implemented. It is the same visualization as in Story Card N1, with the only difference that only the soccer clubs' articles are shown, which the user is following. The user has just to navigate to the corresponding tab on the website.

Story Card N3

The functionality of the option to follow the team, which has written the article, has been fully implemented. The user can click on the follow button, represented as a heart icon, in the news article to follow the team.

Story Card N4 / Story Card N6

Story Card N4, which states that the user wants to see pictures attached to the news articles has been fully implemented. The feature is implemented as picture gallery, which is included in the news feed as well as in the detail view of the articles. Thus, Story Card N6 also has been implemented.

Story Card N5

The visualization of an indicator of how many pictures are attached to the article has been fully implemented. This indicator is attached to the picture gallery. Therefore, the user always knows how many images are available in the gallery.

Story Card N7

The zooming feature, defined in Story Card N7 has also been fully implemented. The picture gallery has the option to open a full-screen mode by clicking on the picture in the news detail view. When the full-screen mode is activated, the user has the possibility to zoom the image.

Story Card N8 / Story Card N9 / Story Card N10

Story Cards N8, defines the wish to display an entire news article. This feature has been fully implemented and also includes the Story Cards N9 and N10. The author and the publish date are included in the news detail view.

Story Card S1

The sharing feature has been fully implemented, and, therefore, Story Card S1 is also fulfilled. The user can share all articles through different options. Either they can click the share button in the header of the news article regardless of whether in the news feed or the detail view. This sharing button is visible when the user is following the team or when the editorial team writes the article. Alternatively, the user can also share the article by clicking one of the social media buttons in the sharing bar in the article's detail view. These social media buttons include sharing on Facebook, Twitter, WhatsApp, and copying the article's link.

Story Card L1

Story Card L1 has been fully implemented. Therefore, the user is able to create an account and login with this account. Furthermore, a validation of the user data is realized with the help of the Angular Framework. After successfully creating an account, the user gets an access token for identification. With this access token, the user is able to log in again.

Story Card L2 / Story Card L3

Story Card L2, as well as Story Card L3, have been fully implemented. The user is able to log in with Facebook or with their Apple ID instead of creating a separate account on this platform.

Story Card L5 / Story Card L6

The option for resetting the account if the user has forgotten their password or email address has been fully implemented. Therefore, Story Card L5 has been fulfilled. The user can open the screen for getting help by clicking on the corresponding text on the log-in screen. When the user has forgotten their password, they can enter their email address for getting an email with the link for resetting their password. When they forgot their email address, they get navigated to a screen where they can send a message to the support team to get help with this issue.

4.4.2 Non-functional Requirements

The next few paragraphs compare the implementation with the non-functional requirements.

Platform Independence

All the features are implemented in a way that the visualization adapts to the browser window. Therefore, the application can be displayed on mobile phones, tablets, as well as desktops. So, the platform independence is fulfilled.

Stability

When something fails during the login process or the registration, the user gets a corresponding error message displayed on the screen. The error messages were discussed in the team and compared with those of the native apps to provide a consistent form of user messages. Furthermore, the user can contact the support team to transmit their feedback when they have some issues. Therefore, this issue can be classified as resolved.

Usability, UI

The application is implemented in a way that it can be displayed on all screen sizes. Furthermore, the design is consistent and intuitive. It is realized as efficient and straightforward as possible. Nevertheless, to allow thorough research into usability, usability tests would be necessary, where randomly selected users test the application. We did not do any usability tests, but the website has been released, and there has been no negative user feedback regarding the usability of these features until now.

Maintainability

The application's architecture has been implemented as modular as possible by creating Angular components, which can be used in the entire application. It should not cause any additional effort if new or existing functionalities have to be maintained or added.

Spam Protection

To prevent the back-end from being spammed, when someone wants to contact the support team, a honeypot has been implemented. This honeypot is a protection mechanism that no bots or attackers can spam this platform. Therefore, this requirement can be classified as resolved.

Password Specification

The front-end validates if the entered password consists of at least six characters before the user can submit. Moreover, there is an additional input field for the confirmation of the password. Therefore, this non-functional requirement can be seen as resolved.

Username Specification

The front-end sends a request to the back-end to check if the suggested username is still available or already used. When someone already uses the username, the user has to take another one before creating an account. Furthermore, the front-end validates if the entered username contains special characters except for underline, dash, and dot. If it contains some special characters, the user cannot submit the data and, therefore, they cannot create an account. Moreover, the front-end also sets a minimum and maximum number for the username. The username has to have a length between 4 and 25 characters. So, this requirement has been fully implemented.

Name and Surname Specification

The front-end sets a minimum and maximum number for the name and surname. They have to have a length between 2 and 50 characters. Furthermore, these two input fields allow all UNICODE characters. Therefore, this requirement can be classified as resolved.

Different Visualisations of News Articles

The news article visualization has been implemented for different cases. One case is the article written by a soccer team, and the other case is the article written by an editorial team. There are small changes in the representation. This requirement has been fully implemented.

4.5 Summary of the Results

This section covers all phases necessary to implement the news articles and registration features and finally to pass them on to the user. The first step is the analysis phase, where all requirements are defined and divided into functional and non-functional ones, as recommended by Sommerville and Sawyer, 1997. Besides, the functional requirements are transformed into user stories according to Cohn, 2004 suggestion. After the definition and documentation of the requirements, they are prioritized. This prioritization is illustrated in Table 4.1 and starts with the most crucial requirements. The most crucial needs are the ones that contribute to the basic functionality. Therefore, the first phase defines and documents all requirements using Requirement Engineering. Afterwards, the concept phase follows, where all design decisions were derived from the defined requirements. First of all, the software architecture is described and the logic defined with the help of UML diagrams. Each feature is presented in its separate diagram. The next step is the conception of the UI design with the help of mock-ups that were designed using Figma[18]. Each screen that is needed for the implementation is represented in a separate mock-up and described in detail. The UI conception is the last step of the concept or design phase. The implementation phase follows afterwards and contains the realization regarding the design decisions and the Story Cards, defined in the analysis phase. Besides, Angular is used for the implementation. Finally, a comparison between the Story Cards and the implementation follows in order to emphasize if the requirements are fulfilled.

[18]https://www.figma.com/ [accessed on: 07.02.2021]

5 Conclusion

This master thesis aimed to create two of the most crucial parts of a soccer platform. One of these essential parts was creating the concept and the implementation of the news articles, which are the primary communication medium of the soccer clubs with their fans. The realization of the registration process was the second main element and, therefore, also one of the two goals. The registration process includes the login, the handling if the user has forgotten their password or their email address, and the option for contacting the support. Besides, the handling for changing the user data and the single sign-on via Facebook and Apple are also included.

By searching for the appropriate SDLC, we found a way to accomplish these goals. Based on this SDLC, it can be concluded that the entire development process is an incremental one. There are several steps or phases the development process has to run through until the desired end product is achieved. Furthermore, the inclusion of the potential users is as essential as the involvement of the whole project team. The involvement of all those involved took place mainly through verbal communication. The inclusion of the potential end-users concerns the soccer teams and their fans or just soccer enthusiasts. The involvement of the users is most notably in the first phase of the SDLC, the analysis phase. This phase is the one where all the requirements for the desired end-product were defined. Once the requirements were set, the design could be derived from it, which happens in the design phase. In this phase, state-of-the-art observations came into play to get ideas related to UI design. How the logic should be implemented was discussed with the developers and sketched using UML diagrams. Finally, the design decisions were implemented in the implementation phase. Again, the entire development team was also involved. The progress and problems were discussed in weekly meetings. These meetings enabled certain things to be dealt with more quickly by getting the help of other developers, who might already be familiar with the problem or provided another point of view. After implementing the features, they were thoroughly tested by the testers, and the other developers reviewed the source code. When both sides approved, the features were released and thus brought to the users.

5.1 Limitations

Due to limited time resources, it was not possible to cover usability testing in this master thesis. Nevertheless, usability testing would be essential to determine whether the design of the features was created as simple and as intuitive as possible or whether something could be improved in this regard. Although the features have been checked by the testers, these tests are more about functionality and whether it was implemented according to the predefined design. However, the features are in production, which means several soccer clubs and soccer fans are already using these features. Of course, the users have the possibility to give their feedback by contacting the support team, but it seems they are more likely to use it to draw attention to missing features or things that do not work.

5.2 Future Work

In the further course of this work, crucial parts of the application have been developed, but some things need to be considered in the future. Nowadays, it is almost necessary to offer single sign-on on a website, as many users do not want to create a separate account. In the course of this work, two single sign-on solutions were implemented, namely Facebook and Apple ID, but at least the most common ones should be implemented. Therefore it would certainly be interesting to add more in the future, such as the Google login. Another thing that could be considered is that the editorial teams' news articles should also be shown in the personal news area. Currently, they are just accessible on the main page for all different news articles.

Bibliography

Angular Team (2021). *Angular Architecture*. URL: https://angular.io/guide/architecture (cit. on pp. 13–25, 27).

Chakraborty, A. et al. (May 2012). "The Role of Requirement Engineering in Software Development Life Cycle." In: *Journal of Emerging Trends in Computing and Information Sciences* 3.5, pp. 723–729 (cit. on p. 10).

Cohn, Mike (2004). *User Stories Applied: For Agile Software Development*. USA: Addison Wesley Longman Publishing Co., Inc. ISBN: 0321205685 (cit. on pp. 11, 29, 100).

Delfmann, P. and T. Rieke (2007). *Effiziente Softwareentwicklung mit Referenzmodellen*. Ed. by J. Becker. ISBN: 978-3-7908-1994-6. DOI: 10.1007/978-3-7908-1994-6 (cit. on p. 9).

Guru 99 (2021). *Functional Requirements vs Non Functional Requirements: Key Differences*. URL: https://www.guru99.com/functional-vs-non-functional-requirements.html (visited on 01/08/2021) (cit. on pp. 11, 41).

IntroBooks Team (Sept. 2020). *Grundlagen der Softwareentwicklung*. IntroBooks. ISBN: 9781393404866 (cit. on pp. 7–9, 32).

Kasagoni, S.K. (2017). *Building Modern Web Applications Using Angular*. Packt Publishing. ISBN: 9781785880032. URL: https://books.google.at/books?id=qnc5DwAAQBAJ (cit. on p. 12).

Paetsch, F., A. Eberlein, and F. Maurer (2003). "Requirements engineering and agile software development." In: *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003*. Pp. 308–313. DOI: 10.1109/ENABL.2003.1231428 (cit. on pp. 9–11).

Rehkopf, Max (2021). *User Storys mit Beispielen und Vorlage*. URL: https://www.atlassian.com/de/agile/project-management/user-stories (visited on 01/09/2021) (cit. on p. 12).

Richard Ishida, W3C. (Aug. 2011). *Personal names around the world*. URL: `https://www.w3.org/International/questions/qa-personal-names.en` (visited on 01/11/2021) (cit. on p. 44).

Sommerville, Ian and Pete Sawyer (1997). *Requirements Engineering: A Good Practice Guide*. 1st. New York, NY, USA: John Wiley & Sons, Inc. ISBN: 0471974447 (cit. on pp. 10, 11, 29, 33, 100).