Christoph Tiffner, BSc

# Online Time Management For University Students

**Master's Thesis**

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

**Graz University of Technology**

Supervisor

Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Nikolai Scerbakov

Institute of Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, March 2019

# 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.

| | |
|---|---|
| Date | Signature |

# Abstract

University students often face the problem of organizing their courses, grades and exams and thereby keeping track of their time. This problem extends even further as courses often require students to write documents that are needed for the course exams and should always be accessible when needed. Students that are not technically versed often struggle with the applications that exist right now or cannot afford them, so they fall back to pen and paper. It is also important that besides all of the organizational work for the university, students need to organize and keep track of the rest of their time not spend in a university context. This thesis highlights technologies to develop an application to solve the problems of fulfilling the students' tasks. It features the applications concept and a prototype developed as a starting point based on the concept. As the software project is specifically developed for students, it is free to use and therefore all of the source code for the prototype and further development is licensed under the MIT license [24].

# Contents

# 1 Introduction

In today's society, it becomes more and more important to stay organized. This is especially true for university students. Students face problems and tasks in all kind of directions. Starting from keeping track of their courses, grades and exams to organizing their documents and keeping their social contacts during all of this. Although there are already several application appliances for the task of time management, pen and paper is still often the first weapon of choice. People are drawn to intuitive and simple solutions, so working with what they know is the logical step and we must build applications with this in mind. Existing applications often only focus on one or few of the everyday tasks and are mostly not that simple to use. Therefore, the idea is that it should be possible to handle time management and organizational tasks with ease and provide additional functionality. The additional functions have to be tailored to fit students' needs and tackle their everyday problems. [25] While some features are needed for specific studies or universities others are not. In the end, students should be convinced to work with a software solution with the features they need and is not stuffed with a lot of overhead. It is not an easy task and there are many requirements to take into consideration.

## 1.1 Current Situation

Currently there are already some applications available that focus on the problem of online time management and organization. Some of them also claim to be designed for university students. However, analyzing the applications it is clear that most of them are missing key features like including social elements.

Following there are some state-of-the-art applications dealing with this functionality premises listed.

### 1.1.1 Firefly Student Planner

The *Firefly Student Planner* [8] is a web application and online learning platform developed by *Firefly Learning Ltd*. It provides a lot of possibilities including direct communication between students and teachers. This includes learning material, tests and time planning. This platform however is neither free to use nor open source.

### 1.1.2 myHomework Student Planner

The *myHomework Student Planner* [18] is a multi-platform application for mobile devices. It supports iOS, Android, Microsoft Windows, Kindle Fire and is available as a *Google Chrome* Add-On. The application is free to use (including advertisements) in the basic version but with costs in the premium version. It is strictly designed to work with tasks concerning university course, test and homework planning.

### 1.1.3 Evernote

*Evernote* [7] is a popular multi-platform application to create and manage notes. It is available on iOS, Android and Microsoft Windows. It enables the user to create and share notes but does not include a time management.

### 1.1.4 Focus Booster

Focus Booster [10] is an online application to manage your time efficiently. It is not free to use and not specifically adapted to university students' needs. On the pro side, it features a lot of graphical evaluation for the planned tasks.

### 1.1.5 Remember The Milk

Remember The Milk [49] is a free online application that works as a reminder. It is not specifically designed for students but has a good feature to remind you over multiple platforms. It is also missing a fully functional calendar and the feature to include external calendars.

### 1.1.6 iStudiez Pro

iStudiez Pro [19] is a specialized student time planning application for iOS, Mac and Microsoft Windows. It can keep track of courses, grades and assignments. It has most of the needed features for students but is missing the possibility to link courses and full notes (e.g. PDFs, Microsoft Word Documents or Google Docs).

### 1.1.7 Student Agenda

Student Agenda [5] is a simple android app to manage courses and exams in a calendar and include the possibility to create notifications and enter marks. This application is sufficient for fast and simple tasks, but for a total replacement of analog student planners there still are features missing.

### 1.1.8 My Study Life

My Study Life [35] is a free to use online application specialized on planning for university students. Like in most of the other student time management applications you can manage your courses, homework and exams. You have the possibility to create to-do lists and put calendar entries into specific categories.

### 1.1.9 Studo

Studo [32] is a partially free Android and iPhone application specialized on time and course planning. It has the possibility to link your calendar to some of their partner universities to present timetables and course updates. A big drawback is the price you have to pay for the full range of functions.

## 1.2 Comparison

All the mentioned apps are compared in table 1.1. Comparison indicators mainly focus on the students' point of view but also include several technical requirements.

| | General | Time management | | | | | | Course management | | | | | | Personal timeline | | | | Export | Share | Adapt-ability | User management | Application Platform | | | Open Source | Free to use | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Application | Designed for students | Regular Course Schedule | Single course dates | Exams | Extended Categories | Private appointments | Reminder | Courses | Exams | Grades | Learning Material | Course Notes | Reporting | Personal Notes | Media Content | Social Content | Arbitrary Conent | | | | | Client | Mobile | Web Application | | | |
| Firefly Student Planner [8] | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | | | | | ✓ | ✓ | | ✓ | | | ✓ | | | |
| myHomework Student Planner [18] | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | | | | | | | ✓ | | | ✓ | ✓ | ✓ | ✓ | | ✓* | *Free Premium |
| iStudiez Pro [19] | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | | | | ✓ | | | ✓ | ✓ | ✓ | | | | |
| Student Agenda [5] | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | | | ✓ | | | ✓ | | ✓ | | | ✓ | |
| My Study Life [35] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | | | | | ✓ | | | ✓ | | ✓ | | | ✓ | |
| Evernote [7] | | | ✓ | | | ✓ | ✓ | | | | | | | ✓ | | | | ✓ | ✓ | | ✓ | ✓ | ✓ | | | ✓ | |
| Remember the Milk [49] | | | | | | ✓ | ✓ | | | | | | | ✓ | | | | ✓ | | | ✓ | ✓ | ✓ | | | ✓ | |
| Focus Booster [10] | | | ✓ | | | ✓ | ✓ | | | | | | ✓ | | | | | ✓ | ✓ | | ✓ | | | ✓ | | ✓ | |
| Studo [32] | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | | | ✓ | | | | ✓ | ✓ | | ✓ | | ✓ | | | ✓* | *Free Premium |

Table 1.1: Application Comparison

## 1.3 Conclusion

If you compare all these planning applications, they are missing a feature to keep track of items that are on first thought not directly linked to the university work. They are missing a feature to save social elements like images, social events and other items you want to memorize. This feature is provided by social networks but not on a private personal level. Another missing feature is the possibility to link documents (self-written or provided) and other learning material for different courses. Therefore, the application should combine a learning oriented planning and tracking environment with some of the features offered by social networks. Features like these have to be easy working with, in order to replace analog alternatives like paper notebooks and student planners. To achieve this level of accessibility it is necessary to work with "what you see is what you get" (WYSIWYG) elements. People tend to distribute their generated data over a lot of different services and systems. It should be possible to easily include elements from these different external systems like Facebook, Google Drive, Dropbox etc. The system also should be open source or have an open API (application programming interface) to make development of add-ons for including elements from different systems possible.

# 2 Requirements

The requirements incorporate the comparison indicators mentioned in section 1.2. The requirements are split into functional requirements and technological requirements.

## 2.1 Functional Requirements

The application has three main issues to handle: time management, course management and personal features. These main requirements are a result of my personal university student experience. They reflect what students are currently handling either with many different applications or on paper. Beside the main functionalities, there are some secondary functional requirements that should help the application to get a higher acceptance.

### 2.1.1 Time Management

The main focus of this application has to be a simple and fitted time management for a university setting. This includes the regular courses and corresponding exams. Besides that, there is the requirement to add off schedule appointments to make it possible handling private and irregular university appointments. Another requirement is a configurable reminder to keep track of all the due dates. For this requirement, an intuitive calendar would be the best choice.

### Regular Courses

The main appointments for university students are their regular courses. These courses, in contrast to simple appointments, need to provide more flexibility. Therefore, they are categorized into scheduled lectures and unscheduled lectures.

**Scheduled Lectures** Scheduled lectures are recurring events on a timely basis (e.g. weekly, daily, ...). To manage these events there is a schedule for those courses with the possibility to exclude or include certain dates.

**Unscheduled Lectures** Unscheduled lectures can always happen during the semester. This is especially needed for technical exercises that do not follow a schedule at all. It is possible to define these unscheduled lecture appointments as a single appointment.

### Exams

Exams are an important part of time management for students. The exam dates have to be configurable for higher priority. The reminder provides the possibility to show how far the learning progress for the exam is. This data is presented in a statistical way.

#### Homework

Homework is handled like the exams and therefore it is also linked to the courses.

#### Semester

Semesters are used for statistical purposes but also for planning ahead. It is possible to accept pre-defined time spans or define a new time spans to count as a semester.

#### Appointment Categories

Appointments can be tagged with one or multiple Categories. It is possible to add own categories to these appointments. These categories fit the specific academic studies. One tag for example could be "exercise interview" specifically for technical studies as there are a lot of exercise courses.

#### Private Appointments

To provide the full functionality of a calendar application private appointments are an essential part of the time management. This is especially true to maintain the application as personal and comfortable to use as possible.

#### Reminder

This feature is important to keep track of the appointments in the calendar. The user is notified whenever homework, exams, important or other marked appointments are due. The way the user is reminded is configurable for example per e-mail.

### 2.1.2 Course Management

This core feature extends the simple time management with tools that are important for university students. It has to be possible to keep track of their tasks, accomplishments and progress regarding all of their courses throughout the semester.

#### Courses

Courses provide the starting point for further management options. Newly created courses provide the functionality to set a schedule. This feature is directly linked to the time management.

The current course progress including time and grade is viewable. It is also possible that a course is marked as passed or failed. This can either be dependent on current grade of the course, time or a manual selection.

Courses can be defined to have a set amount of ECTS [6]. You can link learning material and course notes to a course.

#### Exams

Exams have date and time and are linked to a single course. It is possible to define a weight on how much the exam influences the grade. You can link specific learning material to the exam.

#### Homework

Homework is handled like exams with the difference that you can link the homework documents.

#### Grades

For any created course it is required that a student can enter grades. Either the grade can be linked to an exam or homework or it can be a standalone grade in case of for example an extended category like an exercise interview.

#### Learning Material And Course Notes

It is possible to link documents created by either teachers or students to the corresponding course. These documents can be accessed using the course management but also using the time management. This is important for students to access them in a timely manner. The documents have to be accessible by the application. This can be done either via upload or via access through cloud-based services.

#### Reporting

Reports can be generated to show different statistics like

- Grade development
- Missed appointments
- Overall statistics
- ECTS statistics

These reports can be stored in the students' documents and exported for convenience.

## 2.2 Technological Requirements

The requirements for the application allow different possible implementation approaches.

### 2.2.1 Platform

The selection of the platform is important as it defines the available technologies and the target group of users. For the management application platform, there are three possibilities:

- Desktop
- Mobile
- Web Application

### Desktop Application

Desktop applications have the advantage that you can access operating system functions directly. Using a web-service the problem with central storage can be handled. The problem with desktop applications however is the accessibility. Especially students spend a lot of their time without a desktop environment with only mobile devices accessible. Therefore, a desktop application is not very well suited for the stated requirements.

### Mobile Application

Mobile applications are an important consideration. They are designed for mobile devices and have simple access.

The disadvantage of the platform is the screen size. For an application like the planner, the more space available the easier it is for the user to work with. Another problem are the different operating systems. As there are three operating systems Android, iOS, Windows Mobile a lot of the effort goes into multi-platform support. There are however solutions that can create applications for all platforms at once. When considering the prior disadvantages, a mobile only application should not be the first choice.

### Web Application

Web applications are very well suited for the stated requirements. There are many state-of-the-art frameworks to work with. Time management, course management and personal timeline can be visualized in full size on fitting devices. For smaller devices, there is still an option for a different view. Good performance with the correct usage of up-to-date technologies.

Further advantages include:

- accessible from everywhere
- usable on both desktop and mobile devices (if developed using certain frameworks)
- platform (operating system) independent
- refactor or pack into a mobile application later

Therefore, the best choice to develop an online application for personal time and course management with social features is a web application.

### 2.2.2 Open Source

In the university setting applications should be open source licensed. [25] It provides other students with the option to extend on the work in the future. For the developed software the MIT license [24] should be used.

### 2.2.3 User Interface

Applications that try to mimic an analog medium have to be as accessible as the paradigm. It needs to provide an UI that provides:

- a very low learning curve
- intuitive controls
- easy access to all areas
- fast responses
- quick findings
- modern and simple design

# 3 Problem Statement

The problem statement focuses on user interface design, user interface logic and backend logic. It is important to separate frontend from backend to keep the application flexible. A publicly accessible API handles connection between frontend and backend.

## 3.1 User Interface Design

User Interface design is important to not only draw the user in but to encourage the user to keep using the application.

As Silva and Andrade point out in "Development of a Web Application for Management of Learning Styles"

> *"According [2] and [27], user satisfaction in the interaction with the product and its utility are critical factors to be considered in software design. These factors are governed by the guidelines set out in international standards regarding usability (ISO/DIS 9241:11, European Usability Support Centres). In this context and according to this standard, usability is defined as «the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use»"* [52]

Regarding the *Student Online Management*, it is important that it has a clean and modern feel to it. Therefore, a "what you see is what you get" (WYSIWYG) approach is crucial. The look should follow the following rules:

- Modern font
- Font size appropriate for large display sizes
- Small and fixed main navigation area
- Meaningful Icons enhancing or replacing text when possible
- Buttons execute exactly one action
- Drag and drop when helpful
- Loading indication
- Customization options
- WAI-ARIA

## 3.2 User Interface Logic

The user interface logic is the "backend" of the user interface. It should act as the user interface data processor and provide the connection to the backend via the defined API. It is responsible for handling the HTTP requests and responses and in this function reacting to and updating HTTP headers.

## 3.3 Backend Logic

The backend is split into modules and each module is assigned to a specific task. The modules can easily be replaced, removed or new modules can be added. All modules are connected and can exchange information. A publicly accessible API is provided, which allows access to backend functions and data.

### 3.3.1 Authorization Module

The authorization module handles user accounts, authorities and login information. With these authorities, it is possible to restrict access to certain parts of the application.

### 3.3.2 University Management Module

The university management module should handle everything related to university administration data. This includes data for universities, professors, students, studies, courses etc.

### 3.3.3 Time Management Module

The time management module is dedicated to on users' or students' time planning. It handles everything that can be added to the calendar e.g. personal appointments or memories, recurring course appointments and social events.

### 3.3.4 Course Management Module

The course management module handles course related material like exams, grades and learning material.

### 3.3.5 Document Management Module

The document management module handles all documents, either personal or public accessible. A Document can be any resource a user can upload or link from an external provider.

#### Personal Documents

The document management module has to provide the possibility to manage personal documents that are only visible to the current user. The user has the possibility to allow and deny public or individual access to these documents.

#### Document Library

The document management module has to provide a library that consists of all public accessible documents. These documents are categorized and searchable. They have the minimum requirement of having a connection to the creator but can also include the study, course, ...

## 3.4 Mobile

Most people use smartphones on a daily basis and therefore accessing applications via a mobile device is necessary to get the needed acceptance. Fortunately, nowadays this is no longer a big problem. Many frameworks are available for using applications developed for the web on mobile devices without downsides. If we don't want to rely on a framework, using HTML5 in combination with CSS3 and JavaScript is sufficient to build applications for most display sizes and performances [23].

# 4 State-Of-The-Art Technologies

This chapter describes state of the art technologies that are capable of fulfilling all of the mentioned requirements in chapter 2. These Technologies are split into backend and frontend technologies although some technologies fit both tasks.

## 4.1 Backend Technologies

The backend technologies will mainly focus on current web server technologies. Furthermore, we have a look at the details of frameworks that already implement most of these stateless web technologies. After the basic concepts, we have a look at state-of-the-art frameworks applying these concepts.

### 4.1.1 Concepts

The concepts show a small selection of state-of-the-art web technologies. In fact, there are a lot more technologies but the ones listed are some of the most prominent today.

### SOAP Web Services

> *"SOAP or Simple Object Access Protocol is a protocol designed to exchange information in the form of Web Services. It is primarily based on XML documents exchanged over HTTP, but it's possible to transmit messages through other mediums like email and JMS."*[46]

SOAP web services use *Web Services Description Language* [59] files. As [46] mentions both the client and the server commit to the WSDL contract for exchanging information and making remote procedural calls [54].

As compared in [53], SOAP is more commonly used for enterprise applications in shared environments, it is platform independent and it is standardized.

### RESTful Web Services

*REST* or *Representational State Transfer* is an abstraction that mostly builds upon the HTTP methods GET, POST, PUT, PATCH, DELETE. Most modern browsers also use an OPTIONS request to verify which of these methods are allowed.

> *"REST is an architectural style for designing distributed systems. It is not a standard but a set of constraints, such as being stateless, having a client/server relationship, and a uniform interface. REST is not strictly related to HTTP, but it is most commonly associated with it."* [45]

The principles of REST as mentioned in [45] are:

- Resources - Every request accesses a resource that is accessible via a simple URI

- Representations - Data transferred is either a simple text, or more commonly a JSON or XML object
- Messages - Only use the defined HTTP methods and their required format
- Stateless - Between server and client there is no context stored, therefore it can scale very well. Session information for one client is stored directly on this client.

"*RESTful* webservice" implies that we take the basic REST approach and strictly stick to the HTTP standard described in [36]. [50]

As compared in [53], RESTful webservices and SOAP webservices, both have advantages in certain areas.

### HATEOAS

> *"HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture. A hypermedia-driven site provides information to navigate the site's REST interfaces dynamically by including hypermedia links with the responses. This capability differs from that of SOA-based systems and WSDL-driven interfaces. With SOA, servers and clients usually must access a fixed specification that might be staged somewhere else on the website, on another website, or perhaps distributed by email. "* [44]

HATEOAS provides us with a new set of opportunities, as we always define where our resources are located with links and know that these links implement the required standard methods.

### 4.1.2 Frameworks

Frameworks in software technology are used to apply certain implemented technologies or parts repeatedly in applications. [26] The frameworks in this section implement the above-mentioned concepts and provide a comfortable way to incorporate them in the application. These frameworks are very powerful and extensive and therefore we do not cover ever single detail.

#### Spring

The *Spring framework* [39] is developed for Java and provides a vast number of components for developing business applications. It consists of several projects [42] that can be used for example for building web applications, accessing databases or authentication. For our application *Spring Boot* [40], *Spring Data* [41] and *Spring Security* [43] are the most relevant projects.

#### Spring Boot

*SpringBoot* [40] provides all needed components to build a running web application without the need for an external web server. It includes an embedded web server and does most of the configuration automatically via Java annotations as shown in code listing 4.1.

**Code listing 4.1: Spring Boot Demo**

```
package at.tugraz.stups;

import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.web.bind.annotation.*;

@RestController
@EnableAutoConfiguration
```

```
public class DemoApplication {

  @RequestMapping("/")
  String demo() {
    return "Response";
  }

  public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
  }

}
```

Further information on how to build Spring Boot applications can be found in the "Building an Application with Spring Boot" [38] guide.

### Spring Data

*SpringData* [41] provides the needed components for accessing data using underlying frameworks like *Java Database Connectivity* or the *Java Persistence API*. The configuration is mostly done via annotations.

### Spring Security

*SpringSecurity* [41] provides the needed components for authentication and authorization with Java. It supports *Kerberos*, *OAuth* and *SAML* security.

### JavaServer Faces

*JavaServer Faces* or *JSF* [37] is a framework developed by Oracle to build web applications. It has many similarities to the Spring framework but follows a different approach concerning frontend-backend split. JSF is designed to use the MVC approach. XHTML pages that are linked to the models represent the view part. Models are domain objects that can be used by the view and get their data from the persistence layer using the controllers. Parts of the JSF framework are called components and they also follow the MVC approach. There are many JSF component frameworks for building web applications. One of the best-developed and active frameworks is *Primefaces* [17].

#### ASP.NET

*ASP.NET* [29] is an open source web framework that builds upon the proprietary *.NET Framework* [28]. Both technologies are developed by Microsoft and the main programming language is C#. It is well suited for building web applications as it offers nearly all the features and components mentioned in the previous Java based frameworks. When choosing between these technologies it is up to personal preferences.

## 4.2 Frontend Technologies

The frontend technologies will mainly focus on libraries and frameworks to build client-side user interfaces depending on *Java Script* [33] or *TypeScript* [30]. After basic concepts, we have a look at some of the modern frameworks applying these concepts.

### 4.2.1 Concepts

The concepts relevant for the frontend development are the same as described in section 4.1.1.

### 4.2.2 Frameworks and Technologies

Frameworks in general are described in section 4.1.2. Frontend frameworks for the most part only differ in the used programming languages.

#### JSF

The core concepts of the JSF [37] framework are already described in section 4.1.2. The frontend consists of XHTML pages that contain JSF tag libraries. These tag libraries allow us to access variables and functions from our backend. This can either be synchronous or asynchronous using *AJAX (Asynchronous JavaScript and XML)* calls. The corresponding client-side Java Script for this functionality is generated by JSF.

#### Java Script

*Java Script* [33] is a script language that most modern browser can interpret. Therefore, it enables us to execute programs inside the browser that acts as the client for the web application. You can say Java Script enhances the mostly static web experience when using only HTML and CSS (see section 4.2.3). It is possible to develop the frontend using only *Native Java Script* but there are many useful libraries, like *JQuery* [9], that make the development process much easier.

#### Angular

*Angular* [11] is a client-side web framework. In this section when talking about Angular, we refer to Angular versions 2 and above. In contrast to version 1 or *AngularJS*, Angular is based on *TypeScript* rather than pure *Java*

*Script*. Browser cannot directly interpret Typescript, therefore we need a pre-processor for converting the Typescript code into Java Script code.

Angular Material

*Angular Material* [12] contains *Material Design* components for Angular. All components follow the *Material Design Guide* [13] written by Google. This includes most of the styling and functionality and allows for quick and appealing web development.

#### React

In contrast to Angular, React [16] is only a node library rather than a complete framework. It is designed to build reactive we applications by enhancing the basic java script functions for modifying the *Document Object Model* of the HTML page.

### 4.2.3 HTML and CSS

The *Hypertext Markup Language* or *HTML* and *Cascading Style Sheets* or *CSS* build the structure of our website or web application. For appealing and web applications with good performance it is mandatory that during the development process they do not come too short. One of the biggest issues with HTML and CSS today is the *Cross Browser Compatibility*, where the web application should look the same in every browser and on mobile devices.

### HTML5

*HTML5* [57] is the current standard for building web pages. It defines the tags browser can understand to display the web page. One of the biggest advancements in the HTML5 standard is the canvas tag, that allows us to draw directly in the browser e.g. with Java Script.

### CSS3

*CSS3* [56] is the current standard for defining the style of web pages. It allows to style and layout the tags or components defined in HTML. Some of the biggest advancements in version 3 of CSS are the *Flex Box Layout* for dynamic web page building and *CSS Animations* that do not require Java Script.

#### *SASS*

*Syntactically Awesome Style Sheets* or *SASS* [15] allows us to write CSS in a more "logical" fashion. It is possible to define style classes hierarchical and nested and use so called *Mixins* for function like calls. For this to work we need a pre-processor that converts our SASS syntax into CSS syntax that can be interpreted by the browser.

### Accessible Rich Internet Applications (WAI-ARIA)

WAI-ARIA is a standard defined by the W3 group that aims to make web applications more accessible to people with disabilities. [58] For example all controls should be accessible in different ways and it should always be clear which actions can be executed from the current state.

## 4.3 Database Technologies

Business grade applications need to persist their large amount of data in databases. The field of database technologies is very wide and therefore we take a look at some of the more established technologies for smaller business web applications. We differentiate between database technologies based on the *Structured Query Language* and so-called *NoSQL* databases.

### 4.3.1 Structured Query Language

Technologies based on the *Structured Query Language* or *SQL* build their core around tables and the connection of these tables.

#### MySQL

*MySQL* [1] is a largely distributed *Relational Database Management System* based on SQL. It is open source and capable of handling business application sized data amounts. Because it is so widely used, there is a lot of documentation and community support for development.

#### PostgreSQL

*PostgreSQL* [14] is also widely used and provides the same functionality as MySQL. It only differs in minor implementation of different features and can, under the right circumstances, outperform MySQL by a tiny bit.

### 4.3.2 NoSQL

In comparison to *SQL databases*, *NoSQL databases* are non-relational. They are designed to work with huge amounts of data as they have a high scalability. NoSQL databases can store any kind of unstructured data with a high performance. [22]

#### MongoDB

*MongoDB* is a NoSQL database where the data is represented by documents (similar to JSON format) and collections of documents. It is highly flexible and scalable. [31]

> *"In MongoDB, there are no database outlines or tables. Documents are like rows and are assembled into collections which are like tables. The document is an information structure made out of field and value pairs. The value of fields may incorporate different records, clusters, and varieties of documents. MongoDB consequently produces a primary key (id) to uniquely recognize each record. MongoDB endeavors to hold the greater part of the information in memory so straightforward questions take less time by staying away from costly hard disk recovery operations."* [22]

### 4.3.3 In-memory Database

*In-memory databases* store all the data directly in the RAM of the server. This leads to a very high performance if the server has enough RAM. It should only be used for testing because the data is lost the minute the server is turned off. Example for an in-memory database is the *H2 database* [34].

# 5 Solution

The solution proposal takes optimal prerequisites and includes some features not implemented in the developed prototype.

## 5.1 Prototyping

For demonstration on how to apply the technologies from chapter 4, two prototypes where developed using two different technology stacks.

### 5.1.1 JSF

The first prototype was developed using JSF with the Primefaces [17] UI Framework. Data storage was handled by a PostgreSQL database. The application was deployed using a Wildfly [48] server instance. The problem with this prototype was, that due to the limitations that came by the incorporated UI Framework, it was not possible to develop a modern appealing application. From a strictly functional approach, it works but the acceptance by students would be very low.

### 5.1.2 JHipster

The second prototype and all of the following described solutions are developed with JHipster [20]. JHipster generates the backend (Spring Boot) application, the database and database access, the Rest API and a basic web user interface. The developed prototype is based on many modern technologies [21]. For the prototype, the support of the following technologies is relevant:

- Java 8 (Spring Framework)
- Angular 6+
  - Typescript
  - HTML5
  - CSS3
    - Sass
    - Twitter Bootstrap
- Maven
- Yarn
- PostgreSQL

With this technology stack there are several advantages in comparison to the JSF approach. The differences between these two technologies are shown in table 5.1

## 5.2 User Interface Design

The user interface design is important and if done correctly it makes the process of implementing the user interface logic much easier. For the design process, it is essential to start with general mockups. After the mockups

| | JSF | Spring Framework + Angular |
|---|---|---|
| Frontend and backend connection | Frontend and backend are deployed as one packed web application file and are dependent of each other | Frontend and backend are independent and can be exchanged as long as the API is implemented |
| Application server | A configured application server like Glassfish, Apache Tomcat or Wildfly is needed to run the application | Handled by Spring Boot, no need for an external application server instance to run the application |
| Database Access | Need to configure most of the database access | Easy pre-configured database access with Spring Data |
| Security | Choose a security framework and configure this framework | Easy pre-configured security management with Spring Security |
| User Interface | Depends a lot on external frameworks like Primefaces to get a modern feel | State of the art Typescript/JavaScript based user interface backed by a growing community |

Table 5.1: JSF vs. JHipster

are completed we select the needed UI frameworks to create a website that represents the previous mockups. We introduce a style that is applied throughout the application user interface. This includes

- Color palette

- Font palette
- Control sizes
- Fixed elements
- Element spacing

### 5.2.1 Mockups

The mockups represent the basic user interface idea for the prototype. Figure 5.1 shows a simple login page without a lot of distraction and the possibility to have more than one method of logging in. Figures 5.2, 5.3 and 5.4 show the planner page that also acts as the central access point or dashboard for the application. Figure 5.5 shows a basic concept for the course management.

### 5.2.2 Navigation

Navigation in the web application should feel natural and intuitive. Therefore, we need a navigation bar that is fixed in place, so the user always has the same feel and opportunity to change to a different part of the application. The application should feel like a desktop or single page application where the different sections only change the current possible functions.

#### Navigation Bar

For displaying the navigation bar there were three options:

1. Sidebar
2. Top-bar
3. Sidebar and Top-bar

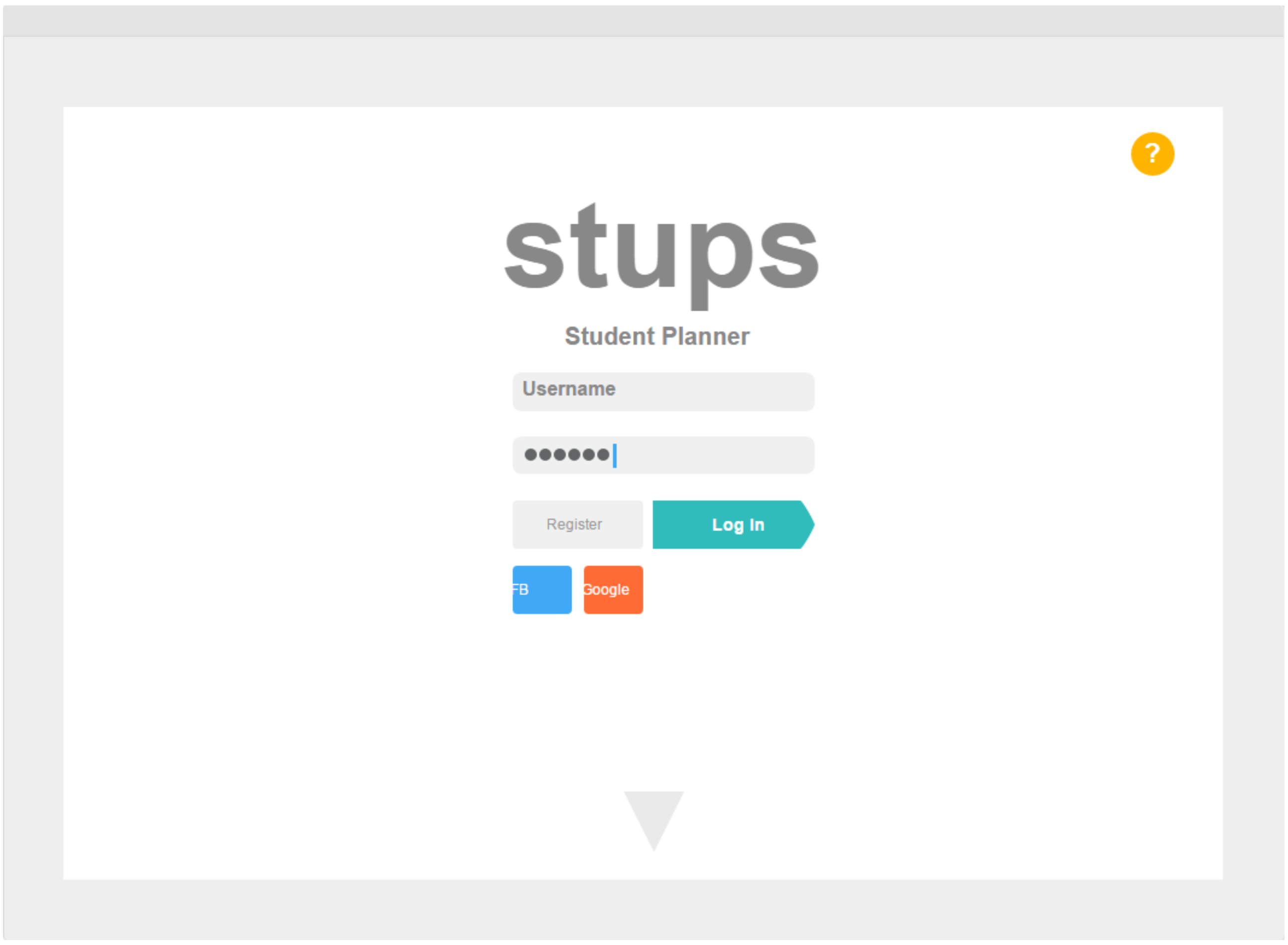


Figure 5.1: Mockup Login Page

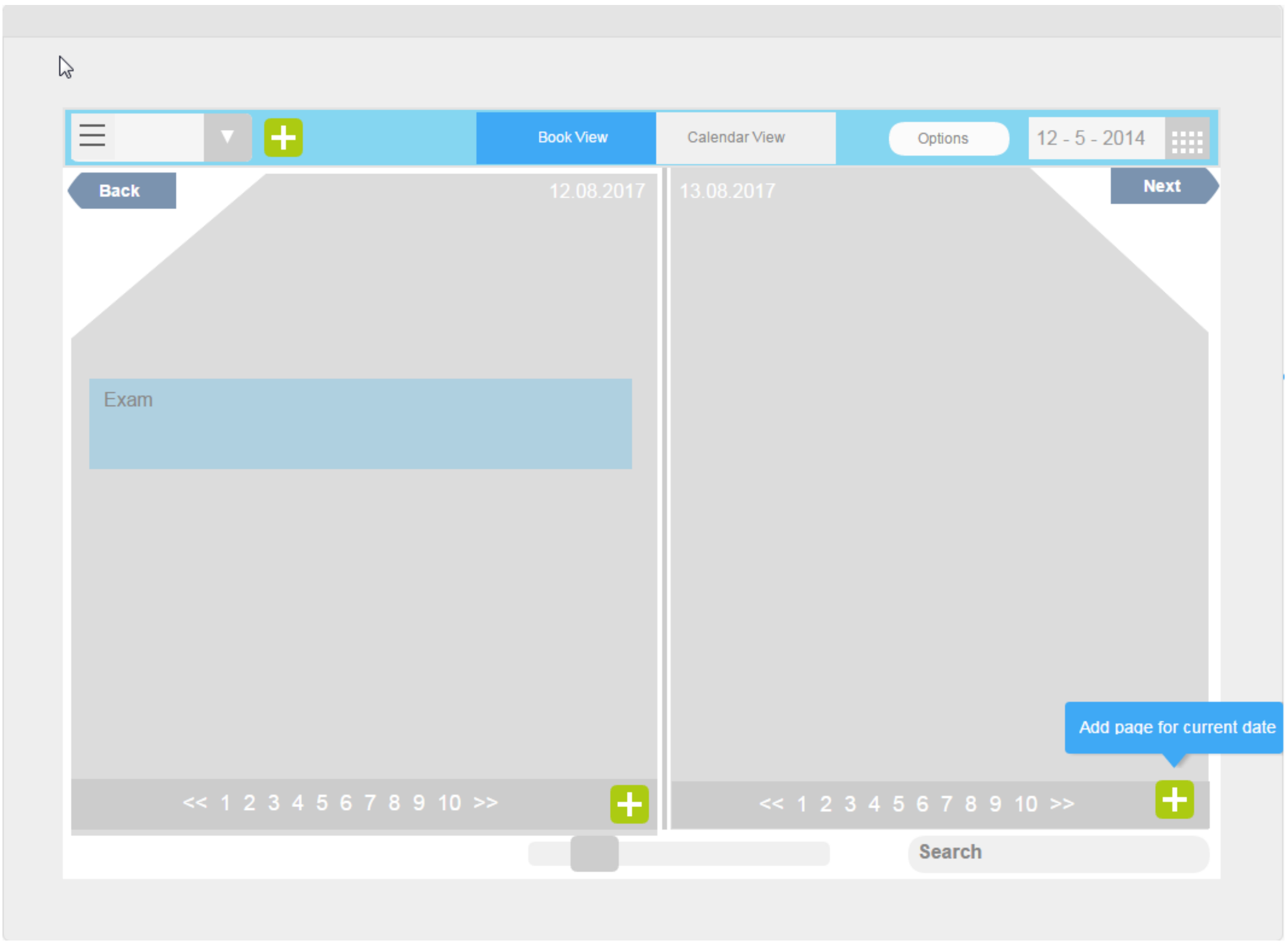


Figure 5.2: Mockup Planner Page 1

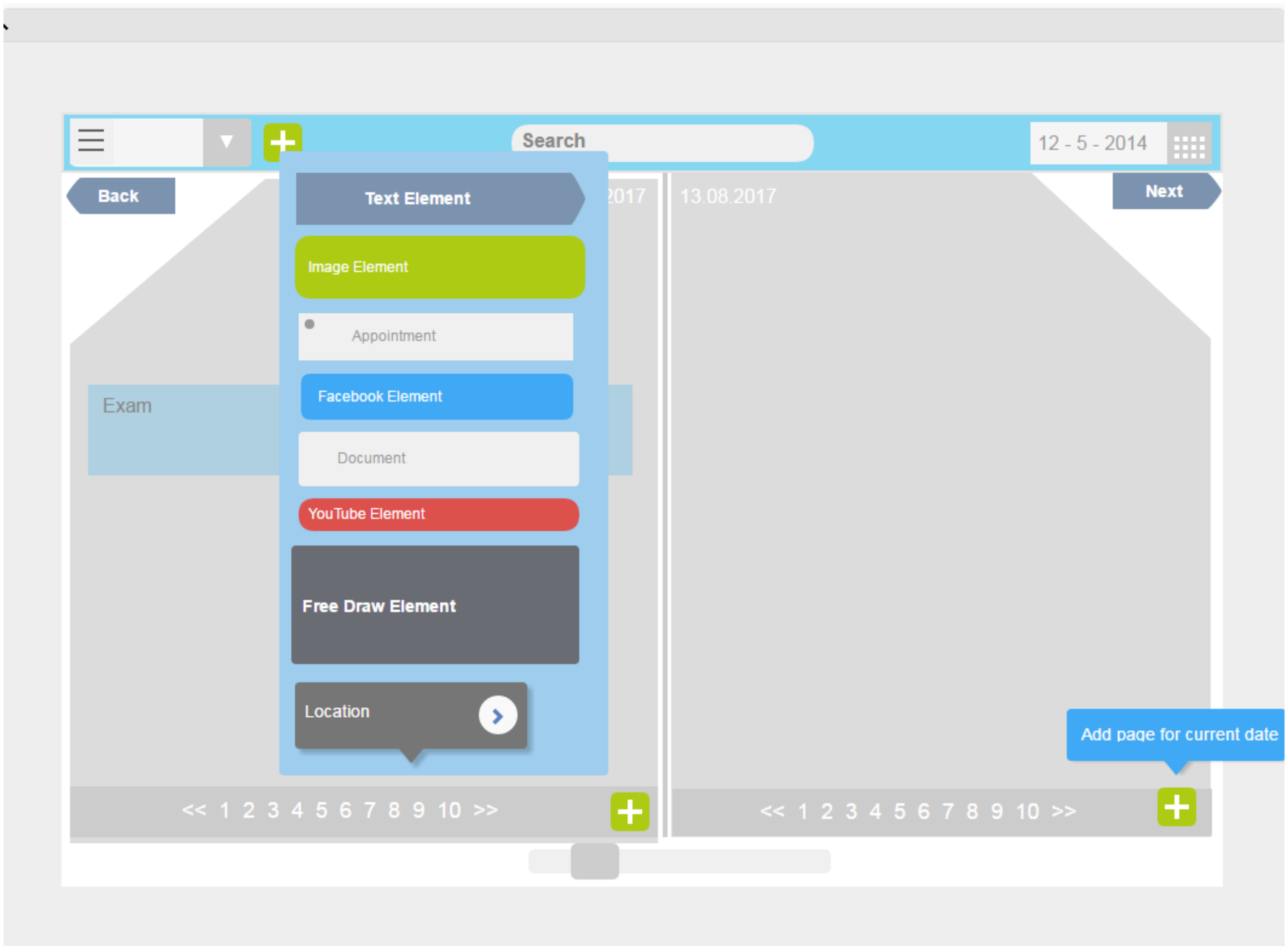


Figure 5.3: Mockup Planner Page Add

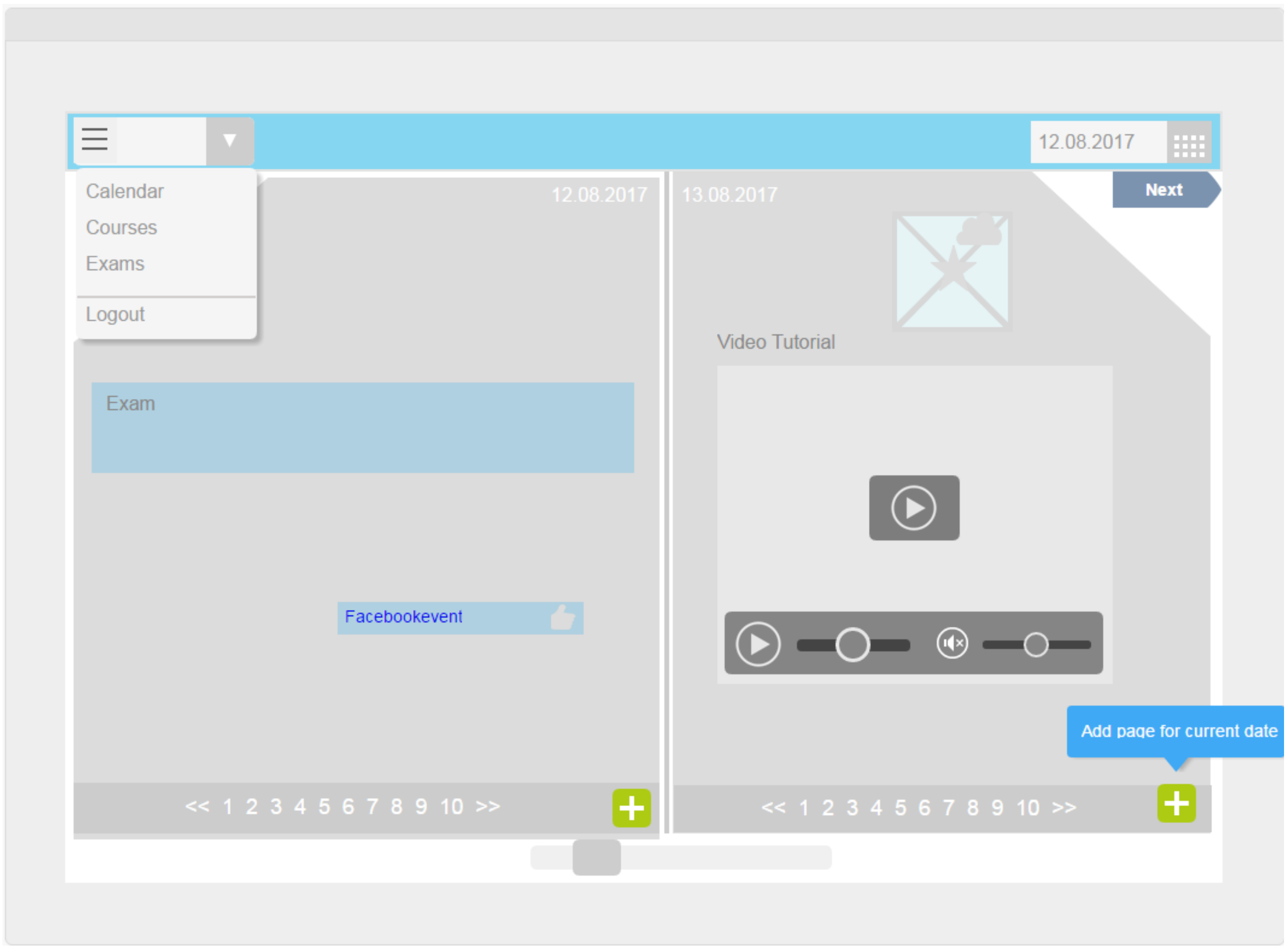


Figure 5.4: Mockup Planner Page 2

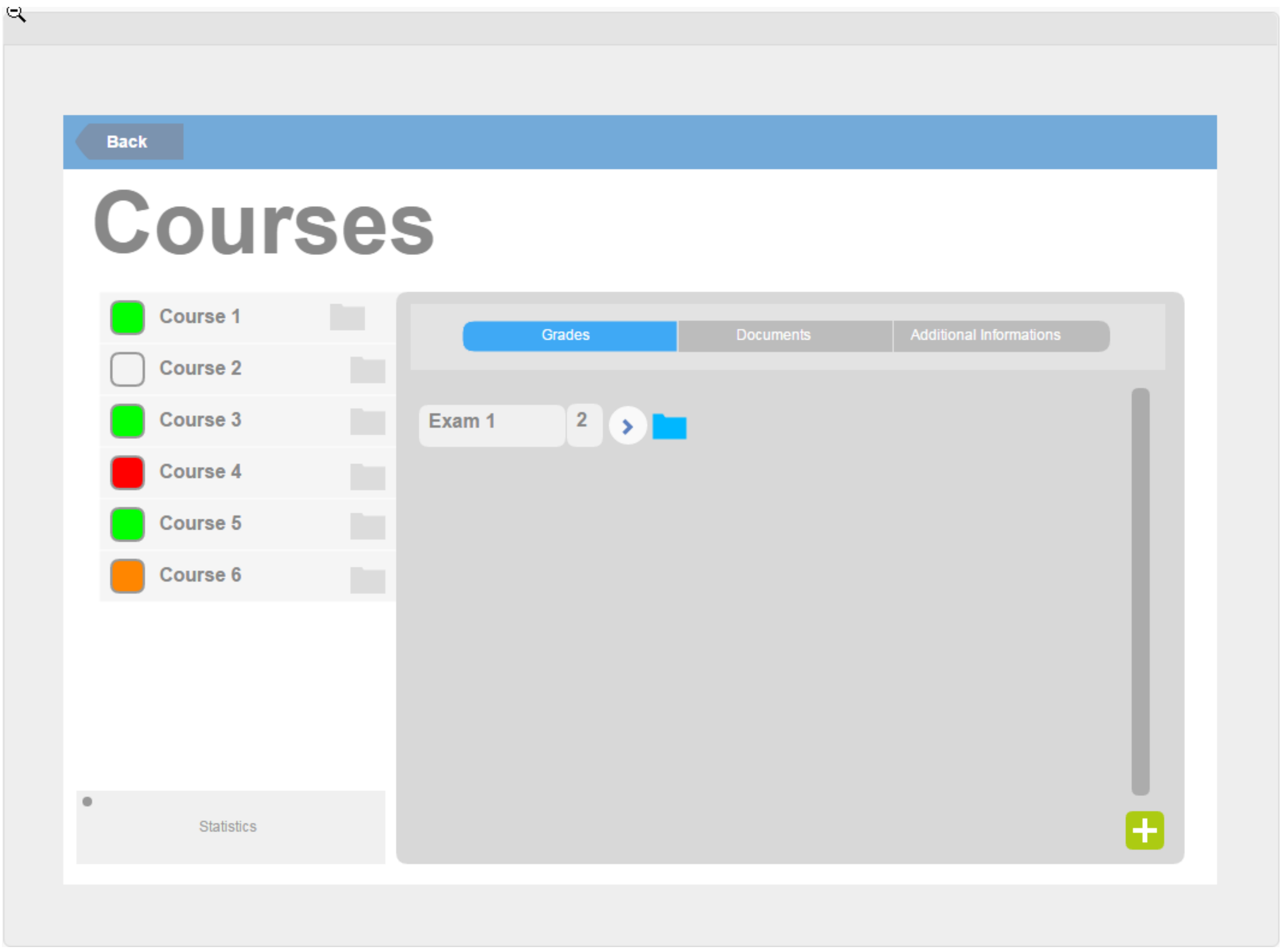


Figure 5.5: Mockup Courses Page

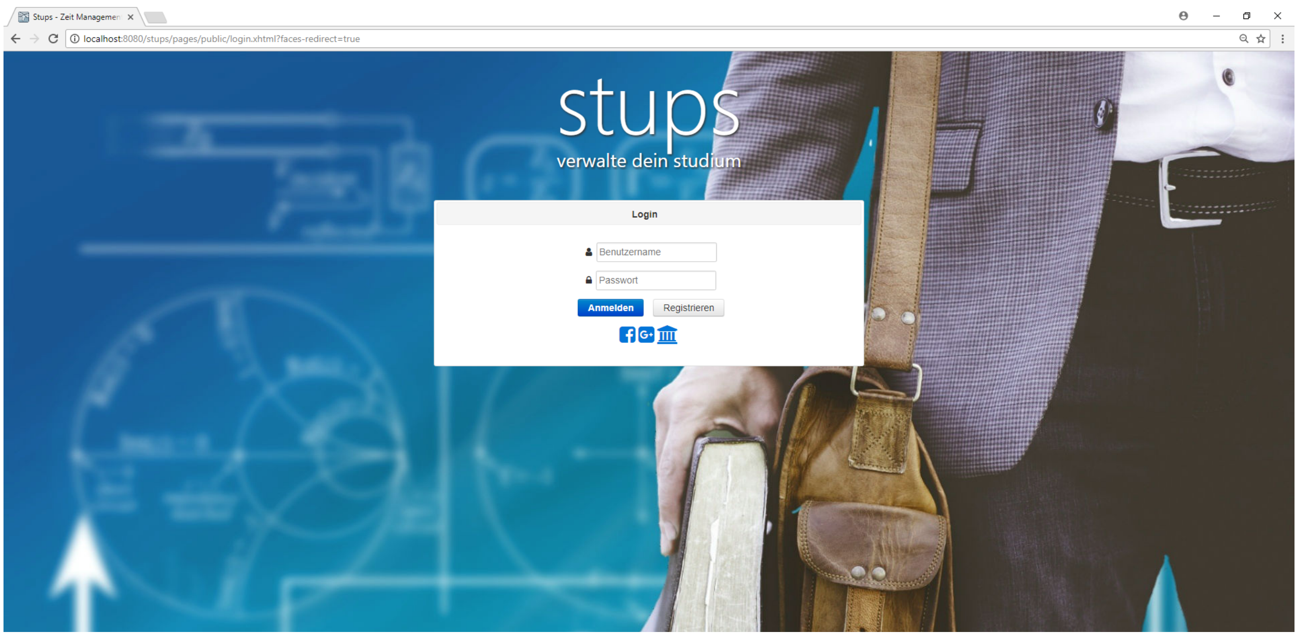


Figure 5.6: JSF Prototype Login Page

The prototype only uses a top-bar as for the planner it is much more convenient to have more horizontal space. The navigation bar for the prototype is e.g. shown on the top in figure 5.7. The menu items are displayed as highly recognizable icons with their description as tooltip text.

### Login Page

The login page as shown in figure 5.6 is also the landing page and provides a simplistic login form. The design is friendly but also professional. Existing users can login easily and fast and new users can register easily using the corresponding button that takes them to the register page described in section 5.2.2. If the user credentials are invalid, a general error message is presented. Unused buttons that could be used in the future are login via linked social network and university accounts. After successful login the user is redirected to the dashboard or time manager described in section 5.2.2.

### Registration

The registration page only consists of a registration form with required and optional fields. The required fields are

- Username
- First name
- Last name
- E-Mail
- University (list of universities pre-persisted in the database)

The optional fields are

- Prefix/Postfix title
- Street
- House number
- City
- Zip code
- Courses (after university selection)

After filling out the form, the user can use a button to register and go back to the login page.

### Dashboard/Time Manager

The dashboard or time manager is the main unit of the application. In this view, the user can perform several actions. These actions include

- Go to any date
- Create an appointment (unscheduled lecture, exam, private appointment)
- View appointments created by scheduled lectures or exams

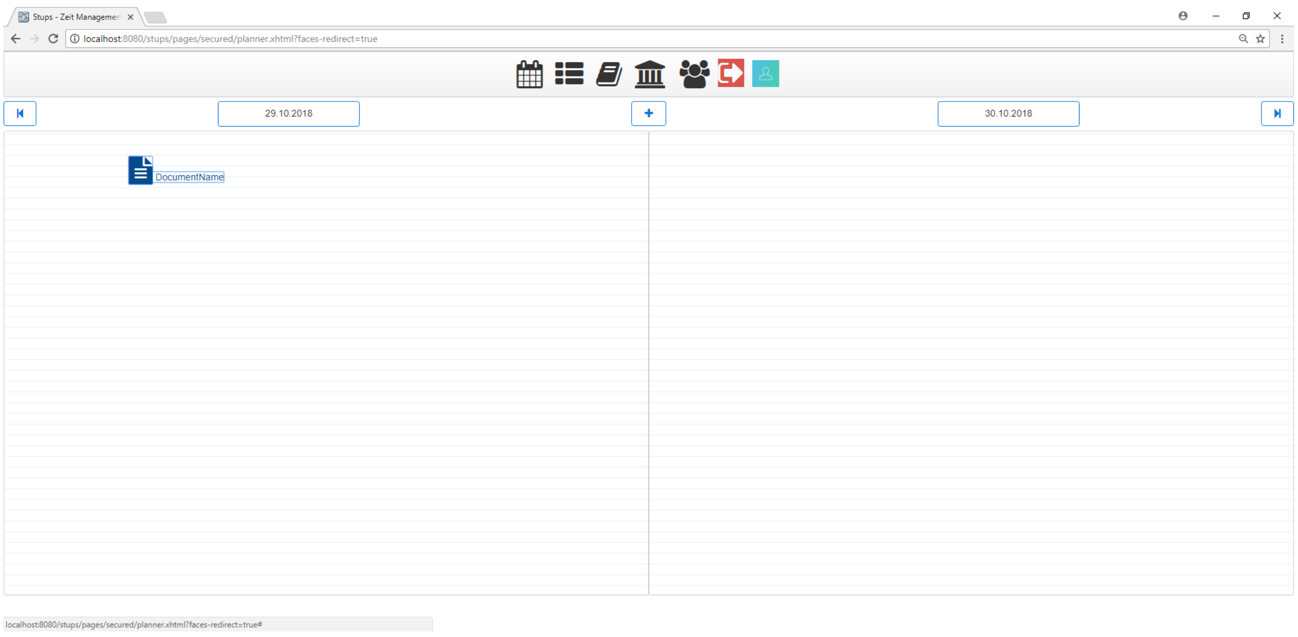

Stups - Zeit Managemen
localhost:8080/stups/pages/secured/planner.xhtml?faces-redirect=true
29.10.2018
30.10.2018
DocumentName
localhost:8080/stups/pages/secured/planner.xhtml?faces-redirect=true#

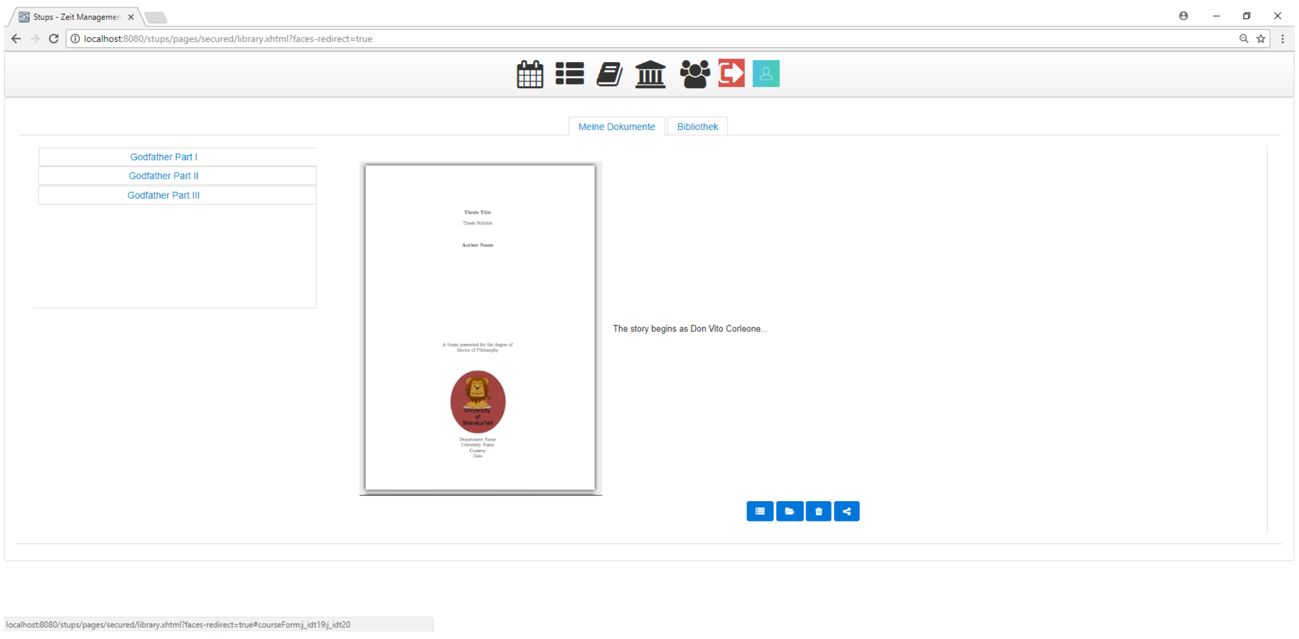
Figure 5.7: JSF Prototype Time Management Page

- Change view between scrapbook and calendar
- In scrapbook view the user can add any number of pages to a specific date
- Add item that is not an appointment (e.g. image, document from library, external document, link, note, video, ...)
- Delete any of the created items

The positions of items in the scrapbook view as shown in figure 5.7 are saved.

### Library

The library page consists of two subpages. The first subpage is the users' private library. This library contains documents grouped by course and loose documents not assigned to any course grouped by a user defined label. In the user library, it is possible to add new documents, edit existing documents, delete documents and publish documents to the general library or revoke


Stups - Zeit Managemen
localhost:8080/stups/pages/secured/library.xhtml?faces-redirect=true
Meine Dokumente
Bibliothek
Godfather Part I
Godfather Part II
Godfather Part III
Thesis Title
Thesis Subtitle
Author Name
A thesis presented for the degree of Doctor of Philosophy
University of ShareLaTeX
Department Name
University Name
Country
Date
localhost:8080/stups/pages/secured/library.xhtml?faces-redirect=true#courseForm:j_idt19:j_idt20

The story begins as Don Vito Corleone...

Figure 5.8: JSF Prototype Library Page

the publication. The second subpage contains the general library where all published documents from users can be viewed. Because of the larger amount, these documents are grouped in a treelike structure going from university level, over study level to course level.

In both libraries a full text search with optional filters is possible as all documents are indexed. Also, a preview image for easier identification of documents is generated and presented when searching or browsing the library.

As the prototype uses a SQL database the full text search is problematic and does not have a very good performance, therefore a NoSQL database would be better suited in future builds.

An example of the user library is shown in figure 5.8.

#### University

The university page is all for information. It presents information about the universities, their studies, the corresponding courses and the professors.

#### Courses

The courses page shows all courses the user has added either during registration or afterwards on this page. The user can add or delete courses and view the course details that include

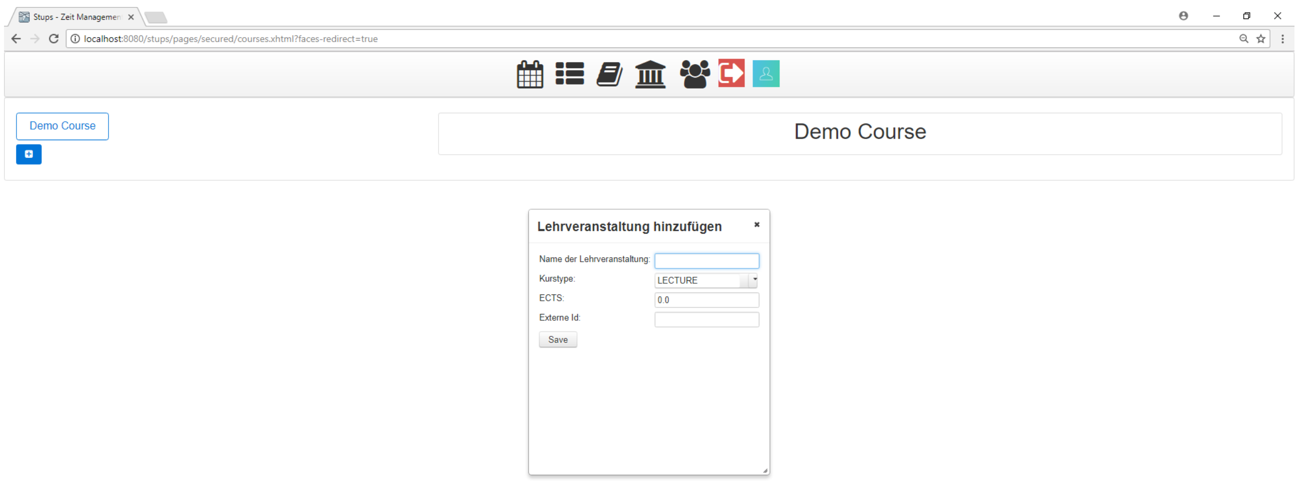
- Grades
- Documents
- Professor
- Lectures
- Exams
- ECTS
- Notes

When adding the user can choose to add an existing course from the system or create a course by providing the necessary information.

A demonstration for a course page is shown in figure 5.9.

#### Settings and Personalization

The settings page enables the user to customize the look and feel of the application. This page should be update continuously and contains only settings for the time management page in the prototype.


Stups - Zeit Managemen
localhost:8080/stups/pages/secured/courses.xhtml?faces-redirect=true
Demo Course
Demo Course

Lehrveranstaltung hinzufügen

|  |  |
|---|---|
| Name der Lehrveranstaltung: |  |
| Kurstype: | LECTURE |
| ECTS: | 0.0 |
| Externe Id: |  |

Save

Figure 5.9: JSF Prototype Course Page

### 5.2.3 Social Features

This is a future improvement to enable the user to add other students as friends and chat with them.

## 5.3 Frontend Logic

The frontend is stateless, and the logic mostly consists of checking form validity and presenting data from the backend. The data is retrieved and updated via Data Transfer Objects or DTOs (see section 5.4.5) over a defined REST API.

## 5.4 Backend Logic

The backend is implemented in a MVC pattern using Java extended by the Spring framework. The MVC classes for university management, time management, course management and grade management are generated by JHipster and extended with additional methods.

### 5.4.1 Authorization

The authorization method can be selected during the application generation. For the prototype, the authorization is handled via JSON Web Tokens or JWT [3]. JWT is stateless and therefore suited for our stateless angular application. JWT is not included in Spring Security and JHipster is using an implementation of their own. For productive installations JWT should be switched to a more secure authorization like OAuth2 that is also included in Spring Security.

### 5.4.2 Repository

The repository handles database operations. JHipster can generate a new database or use an existing database. It also generates data access classes for the provided or generated database using the Spring Data JPA.

#### Database

For the prototype, we use a generated MySQL database for all of the data. Real world applications should consider using a NoSQL database for the document management part. It is possible to prepare a database in advance and only provide the connection to the existing database. For applications beyond prototypes, this should always be the preferred choice.

### JHipster Domain Language

JHipster uses JHipster Domain Language (JDL) to describe the database relationships. The JDL file is the base on which JHipster generates the database (and services) independent of the selected technology. The complete data model is shown in figure 5.10. The prototype includes the following entities (and enumerations)

- User
- Student
- Address
- Course
- CourseType
- Study
- Exam
- Appointment
- LibraryDocument
- Library
- Professor
- University
- Planner
- PlannerPage
- PlannerPageItem
- PlannerItemTye
- Preferences
- Locale

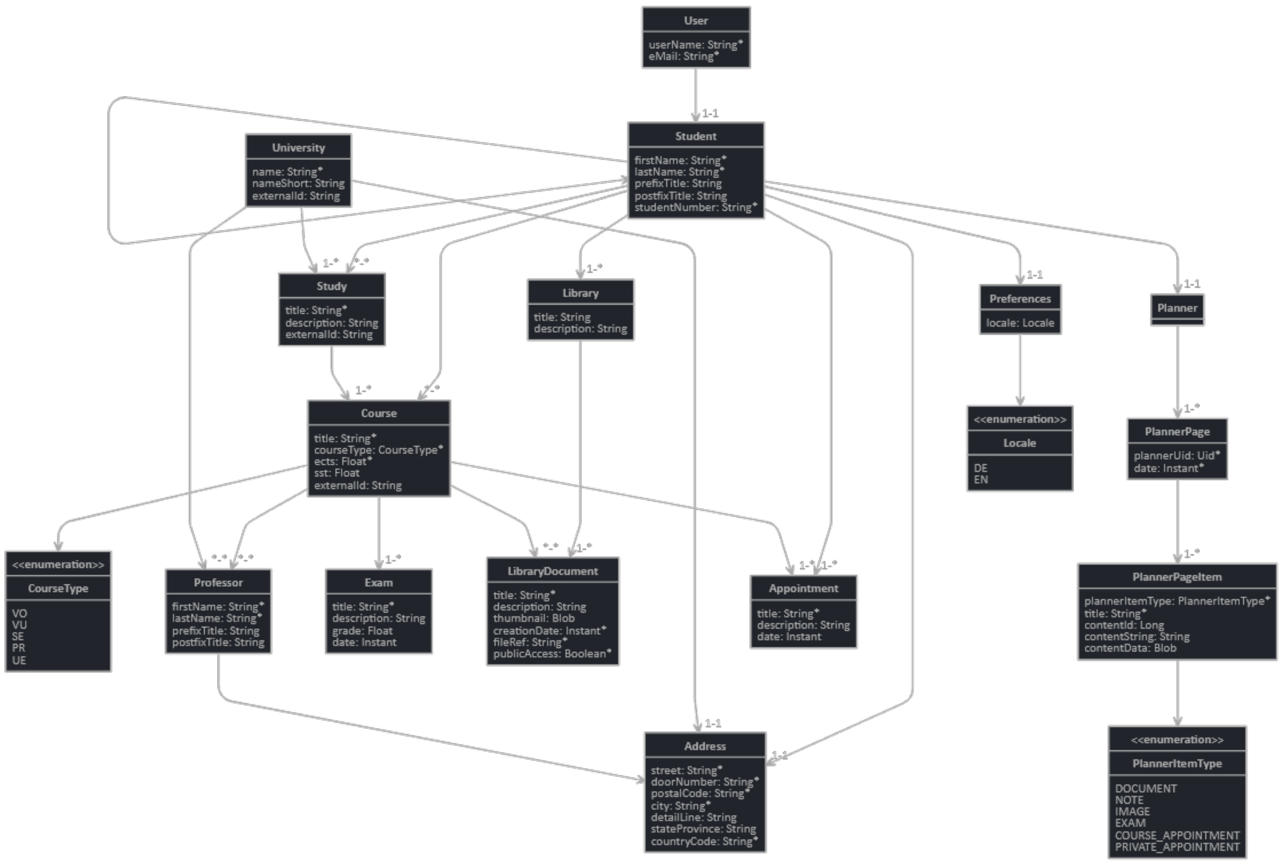


Figure 5.10: JDL Diagram

Code listing 5.1 contains the JDL code behind figure 5.10. It includes the entities, relationships and instructions for the database and service generation.

**Code listing 5.1: JDL**

```

entity User {
  userName String required,
  eMail String required
}

entity Student {
  firstName String required,
  lastName String required,
  prefixTitle String,
  postfixTitle String,
  studentNumber String required
}

entity Address {
  street String required,
  doorNumber String required,
  postalCode String required,
  city String required,
  detailLine String,
  stateProvince String,
  countryCode String required
}

entity Course {
  title String required,
  courseType CourseType required,
  ects Float required,
  sst Float,
  externalId String
}

enum CourseType {
  VO, VU, SE, PR, UE
}

entity Study {
  title String required,
  description String,
  externalId String,
}

entity Exam {
  title String required,
  description String,
  grade Float,
  date Instant
}

entity Appointment {
  title String required,
  description String,
  date Instant
}
```

```

entity LibraryDocument {
  title String required,
  description String,
  thumbnail Blob,
  creationDate Instant required,
  fileRef String required,
  publicAccess Boolean required
}

entity Library {
  title String,
  description String
}

entity Professor {
  firstName String required,
  lastName String required,
  prefixTitle String,
  postfixTitle String,
}

entity University {
  name String required,
  nameShort String,
  externalId String
}

entity Planner {
}

entity PlannerPage {
  plannerUid Uid required,
  date Instant required
}

entity PlannerPageItem {
  plannerItemType PlannerItemType required,
  title String required,
  contentId Long,
  contentString String,
  contentData Blob
```

```
}

enum PlannerItemType {
  DOCUMENT,
  NOTE,
  IMAGE,
  EXAM,
  COURSE_APPOINTMENT,
  PRIVATE_APPOINTMENT
}

entity Preferences {
  locale Locale
}

enum Locale {
  DE, EN
}

relationship ManyToMany {
  Student{studies} to Study{students},
  Student{courses} to Course{students},
  Course{documents} to LibraryDocument{courses},
  Course{professors} to Professor{courses},
  University{professors} to Professor{universities},
  Student{friends} to Student{friends}
}

relationship OneToMany {
  Planner to PlannerPage,
  PlannerPage{Item} to PlannerPageItem,
  Course{exam} to Exam,
  Study{course} to Course,
  Student{library} to Library,
  Library{document} to LibraryDocument{library},
  University{study} to Study{university},
  Course{appointment} to Appointment,
  Student{appointment} to Appointment
}

relationship OneToOne {
  User{student} to Student{user}
  Student{planner} to Planner{student},
  Student{preferences} to Preferences,
  Student{address} to Address,
  Professor{address} to Address,
  University{address} to Address
}

paginate LibraryDocument, Course, Study, Student, Professor with pagination

dto * with mapstruct

// Set service options to all except few
service all with serviceImpl
```

### Spring Data JPA

Spring Data wraps the JPA (Java Persistence API) functions that connect our java application to our database. For every entity, a Spring Data JPA repository is generated (*@Repository* Spring Data annotation). The basic generated repository only includes query methods for find one and find all but can be easily extended. Code listing 5.2 displays the generated repository class for the student entity.

**Code listing 5.2: Spring Data JPA Student Repository**

```
package at.tugraz.stups.repository;

import at.tugraz.stups.domain.Student;
import org.springframework.stereotype.Repository;

import org.springframework.data.jpa.repository.*;
import org.springframework.data.repository.query.Param;
import java.util.List;

/**
 * Spring Data JPA repository for the Student entity.
 */
@SuppressWarnings("unused")
```

```
@Repository
public interface StudentRepository extends JpaRepository<Student, Long> {
    @Query("select distinct student from Student student left join fetch
        student.studies left join fetch student.courses")
    List<Student> findAllWithEagerRelationships();

    @Query("select student from Student student left join fetch student.
        studies left join fetch student.courses where student.id =:id")
    Student findOneWithEagerRelationships(@Param("id") Long id);

}
```

## 5.4.3 Domain

The domain layer connects the repositories to their corresponding service classes. JHipster uses Hibernate as the persistence framework.

### Hibernate

Hibernate extends the Java Persistence API and enables us to use annotations in our entity classes for almost all database related actions. In our case JHipster uses hibernates caching mechanism for faster data access. An example of an entity class is the student entity presented in code listing 5.3

**Code listing 5.3: Student Entity**

```
package at.tugraz.stups.domain;

import com.fasterxml.jackson.annotation.JsonIgnore;
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;

import javax.persistence.*;
import javax.validation.constraints.*;

```

```java
import org.springframework.data.elasticsearch.annotations.Document;
import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;
import java.util.Objects;

/**
 * A Student.
 */
@Entity
@Table(name = "student")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
@Document(indexName = "student")
public class Student implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "
        sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    private Long id;

    @NotNull
    @Column(name = "first_name", nullable = false)
    private String firstName;

    @NotNull
    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "prefix_title")
    private String prefixTitle;

    @Column(name = "postfix_title")
    private String postfixTitle;

    @NotNull
    @Column(name = "student_number", nullable = false)
    private String studentNumber;

    @OneToOne
    @JoinColumn(unique = true)
    private Planner planner;

    @OneToOne
    @JoinColumn(unique = true)
    private Preferences preferences;

    @OneToOne
    @JoinColumn(unique = true)
    private Address address;

    @OneToMany(mappedBy = "student")
    @JsonIgnore
    @Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
    private Set<Library> libraries = new HashSet<>();

    @ManyToOne
    private Student student;

    @OneToMany(mappedBy = "student")
    @JsonIgnore
    @Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
    private Set<Student> friends = new HashSet<>();

    @OneToMany(mappedBy = "student")
    @JsonIgnore
    @Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
    private Set<Appointment> appointments = new HashSet<>();

    @ManyToMany
    @Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
    @JoinTable(name = "student_study",
               joinColumns = @JoinColumn(name="students_id",
                  referencedColumnName="id"),
               inverseJoinColumns = @JoinColumn(name="studies_id",
                  referencedColumnName="id"))
    private Set<Study> studies = new HashSet<>();

    @ManyToMany
    @Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
    @JoinTable(name = "student_course",
            joinColumns = @JoinColumn(name="students_id",
               referencedColumnName="id"),
            inverseJoinColumns = @JoinColumn(name="courses_id",
               referencedColumnName="id"))
    private Set<Course> courses = new HashSet<>();

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public Student firstName(String firstName) {
        this.firstName = firstName;
        return this;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public Student lastName(String lastName) {
        this.lastName = lastName;
        return this;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getPrefixTitle() {
        return prefixTitle;
    }

    public Student prefixTitle(String prefixTitle) {
        this.prefixTitle = prefixTitle;
        return this;
    }

    public void setPrefixTitle(String prefixTitle) {
        this.prefixTitle = prefixTitle;
    }

    public String getPostfixTitle() {
        return postfixTitle;
    }

    public Student postfixTitle(String postfixTitle) {
        this.postfixTitle = postfixTitle;
        return this;
    }

    public void setPostfixTitle(String postfixTitle) {
        this.postfixTitle = postfixTitle;
    }

    public String getStudentNumber() {
        return studentNumber;
    }

    public Student studentNumber(String studentNumber) {
        this.studentNumber = studentNumber;
        return this;
    }

    public void setStudentNumber(String studentNumber) {
        this.studentNumber = studentNumber;
    }

    public Planner getPlanner() {
        return planner;
    }

    public Student planner(Planner planner) {
        this.planner = planner;
        return this;
    }

    public void setPlanner(Planner planner) {
        this.planner = planner;
    }

    public Preferences getPreferences() {
        return preferences;
    }

    public Student preferences(Preferences preferences) {
        this.preferences = preferences;
        return this;
    }

    public void setPreferences(Preferences preferences) {
        this.preferences = preferences;
    }

    public Address getAddress() {
        return address;
    }

    public Student address(Address address) {
        this.address = address;
        return this;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    public Set<Library> getLibraries() {
        return libraries;
    }

    public Student libraries(Set<Library> libraries) {
        this.libraries = libraries;
        return this;
    }

    public Student addLibrary(Library library) {
        this.libraries.add(library);
        library.setStudent(this);
        return this;
    }

    public Student removeLibrary(Library library) {
        this.libraries.remove(library);
        library.setStudent(null);
        return this;
    }

    public void setLibraries(Set<Library> libraries) {
        this.libraries = libraries;
    }

    public Student getStudent() {
        return student;
    }

    public Student student(Student student) {
        this.student = student;
        return this;
    }

    public void setStudent(Student student) {
        this.student = student;
    }

    public Set<Student> getFriends() {
        return friends;
    }

    public Student friends(Set<Student> students) {
        this.friends = students;
        return this;
    }

    public Student addFriend(Student student) {
        this.friends.add(student);
        student.setStudent(this);
        return this;
    }

    public Student removeFriend(Student student) {
        this.friends.remove(student);
        student.setStudent(null);
        return this;
    }

    public void setFriends(Set<Student> students) {
        this.friends = students;
    }

    public Set<Appointment> getAppointments() {
        return appointments;
    }

    public Student appointments(Set<Appointment> appointments) {
        this.appointments = appointments;
        return this;
    }

    public Student addAppointment(Appointment appointment) {
        this.appointments.add(appointment);
        appointment.setStudent(this);
        return this;
    }

    public Student removeAppointment(Appointment appointment) {
        this.appointments.remove(appointment);
        appointment.setStudent(null);
        return this;
    }

    public void setAppointments(Set<Appointment> appointments) {
        this.appointments = appointments;
    }

    public Set<Study> getStudies() {
        return studies;
    }

    public Student studies(Set<Study> studies) {
        this.studies = studies;
        return this;
    }

    public Student addStudy(Study study) {
        this.studies.add(study);
        study.getStudents().add(this);
        return this;
    }

    public Student removeStudy(Study study) {
        this.studies.remove(study);
        study.getStudents().remove(this);
        return this;
    }

    public void setStudies(Set<Study> studies) {
        this.studies = studies;
    }

    public Set<Course> getCourses() {
        return courses;
    }

    public Student courses(Set<Course> courses) {
        this.courses = courses;
        return this;
    }

    public Student addCourse(Course course) {
        this.courses.add(course);
        course.getStudents().add(this);
        return this;
    }

    public Student removeCourse(Course course) {
        this.courses.remove(course);
        course.getStudents().remove(this);
        return this;
    }

    public void setCourses(Set<Course> courses) {
        this.courses = courses;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        Student student = (Student) o;
        if (student.getId() == null || getId() == null) {
            return false;
        }
        return Objects.equals(getId(), student.getId());
    }

    @Override
    public int hashCode() {
        return Objects.hashCode(getId());
    }

    @Override
    public String toString() {
        return "Student{" +
            "id=" + getId() +
            ", firstName='" + getFirstName() + "'" +
            ", lastName='" + getLastName() + "'" +
            ", prefixTitle='" + getPrefixTitle() + "'" +
            ", postfixTitle='" + getPostfixTitle() + "'" +
            ", studentNumber='" + getStudentNumber() + "'" +
            "}";
    }
}
```

### 5.4.4 Service Interface

The service interface defines all functions that have to be implemented in order to access our persistence layer. The interface for the student service is shown in code listing 5.4

**Code listing 5.4: Student Service Interface**

```java
package at.tugraz.stups.service;

import at.tugraz.stups.service.dto.StudentDTO;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

/**
 * Service Interface for managing Student.
 */
public interface StudentService {

    /**
     * Save a student.
     *
     * @param studentDTO the entity to save
     * @return the persisted entity
     */
    StudentDTO save(StudentDTO studentDTO);

    /**
     * Get all the students.
     *
     * @param pageable the pagination information
     * @return the list of entities
     */
    Page<StudentDTO> findAll(Pageable pageable);

    /**
     * Get the "id" student.
     *
     * @param id the id of the entity
     * @return the entity
     */
    StudentDTO findOne(Long id);

    /**
     * Delete the "id" student.
     *
     * @param id the id of the entity
     */
    void delete(Long id);

    /**
     * Search for the student corresponding to the query.
     *
     * @param query the query of the search
     *
     * @param pageable the pagination information
     * @return the list of entities
     */
    Page<StudentDTO> search(String query, Pageable pageable);
}
```

### 5.4.5 Data Transfer API

The data transfer API defines the connection between the frontend and the backend. It consists of the DTO definition, a mapping that maps DTOs to domain objects and the implementation of the domain services to update the objects on the persistence layer.

#### Data Transfer Objects

DTOs are serializable objects that are transferred between the backend and a client or in our case the frontend. The fields of the DTO can vary in comparison to the domain object but for the prototype, they are the same. DTOs are transferred and received in JSON format as in code listing 5.6. The DTO for the student domain object is presented in code listing 5.5

**Code listing 5.5: Student DTO**

```
package at.tugraz.stups.service.dto;


import javax.validation.constraints.*;
import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;
import java.util.Objects;

/**
 * A DTO for the Student entity.
 */
public class StudentDTO implements Serializable {

    private Long id;

    @NotNull
    private String firstName;

    @NotNull
    private String lastName;

    private String prefixTitle;

    private String postfixTitle;

    @NotNull
    private String studentNumber;

    private Long plannerId;

    private Long preferencesId;

    private Long addressId;

    private Long studentId;

    private Set<StudyDTO> studies = new HashSet<>();

    private Set<CourseDTO> courses = new HashSet<>();

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getPrefixTitle() {
        return prefixTitle;
    }

    public void setPrefixTitle(String prefixTitle) {
        this.prefixTitle = prefixTitle;
    }

    public String getPostfixTitle() {
        return postfixTitle;
    }

    public void setPostfixTitle(String postfixTitle) {
        this.postfixTitle = postfixTitle;
    }

    public String getStudentNumber() {
        return studentNumber;
    }

    public void setStudentNumber(String studentNumber) {
        this.studentNumber = studentNumber;
    }

    public Long getPlannerId() {
        return plannerId;
    }

    public void setPlannerId(Long plannerId) {
        this.plannerId = plannerId;
    }

    public Long getPreferencesId() {
        return preferencesId;
    }

    public void setPreferencesId(Long preferencesId) {
        this.preferencesId = preferencesId;
    }

    public Long getAddressId() {
        return addressId;
    }

    public void setAddressId(Long addressId) {
        this.addressId = addressId;
    }

    public Long getStudentId() {
        return studentId;
    }

    public void setStudentId(Long studentId) {
        this.studentId = studentId;
    }

    public Set<StudyDTO> getStudies() {
        return studies;
    }

    public void setStudies(Set<StudyDTO> studies) {
        this.studies = studies;
    }

    public Set<CourseDTO> getCourses() {
        return courses;
    }

    public void setCourses(Set<CourseDTO> courses) {
        this.courses = courses;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }

        StudentDTO studentDTO = (StudentDTO) o;
        if(studentDTO.getId() == null || getId() == null) {
            return false;
        }
        return Objects.equals(getId(), studentDTO.getId());
    }

    @Override
    public int hashCode() {
        return Objects.hashCode(getId());
    }

    @Override
    public String toString() {
        return "StudentDTO{" +
            "id=" + getId() +
            ", firstName='" + getFirstName() + "'" +
            ", lastName='" + getLastName() + "'" +
            ", prefixTitle='" + getPrefixTitle() + "'" +
            ", postfixTitle='" + getPostfixTitle() + "'" +
            ", studentNumber='" + getStudentNumber() + "'" +
                "}";
    }
}
```

Code listing 5.6: Student DTO JSON

```
{
  "id":23842,
  "firstName":"John",
  "lastName":"Student",
  "prefixTitle":"DI",
  "postfixTitle":"BSc",
  "studentNumber":"0011223344",
  "plannerId":23841,
  "preferencesId":23840,
  "addressId":23823,
  "studies":[...],
  "courses":[...]
}
```

## Mapping

The mapping is the connection between our DTOs and the corresponding domain objects. For every DTO a mapping is present. Code listing 5.7 shows the mapping for the StudentDTO.

Code listing 5.7: Student Mapping

```
package at.tugraz.stups.service.mapper;

import at.tugraz.stups.domain.*;
import at.tugraz.stups.service.dto.StudentDTO;

import org.mapstruct.*;

/**
 * Mapper for the entity Student and its DTO StudentDTO.
 */
```

```
@Mapper(componentModel = "spring", uses = {PlannerMapper.class,
    PreferencesMapper.class, AddressMapper.class, StudyMapper.class,
    CourseMapper.class})
public interface StudentMapper extends EntityMapper<StudentDTO, Student> {

    @Mapping(source = "planner.id", target = "plannerId")
    @Mapping(source = "preferences.id", target = "preferencesId")
    @Mapping(source = "address.id", target = "addressId")
    @Mapping(source = "student.id", target = "studentId")
    StudentDTO toDto(Student student);

    @Mapping(source = "plannerId", target = "planner")
    @Mapping(source = "preferencesId", target = "preferences")
    @Mapping(source = "addressId", target = "address")
    @Mapping(target = "libraries", ignore = true)
    @Mapping(source = "studentId", target = "student")
    @Mapping(target = "friends", ignore = true)
    @Mapping(target = "appointments", ignore = true)
    Student toEntity(StudentDTO studentDTO);

    default Student fromId(Long id) {
        if (id == null) {
            return null;
        }
        Student student = new Student();
        student.setId(id);
        return student;
    }
}
```

## Service Implementation

The service implementation class implements the interface from section 5.4.4. The service methods define the writing and loading of the persistence layer. Code Listing 5.8 shows the implementation of the student service interface from code listing 5.4.

**Code listing 5.8: Student Service Implementation**

```java
package at.tugraz.stups.service.impl;

import at.tugraz.stups.service.StudentService;
import at.tugraz.stups.domain.Student;
import at.tugraz.stups.repository.StudentRepository;
import at.tugraz.stups.repository.search.StudentSearchRepository;
import at.tugraz.stups.service.dto.StudentDTO;
import at.tugraz.stups.service.mapper.StudentMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;


import static org.elasticsearch.index.query.QueryBuilders.*;

/**
 * Service Implementation for managing Student.
 */
@Service
@Transactional
public class StudentServiceImpl implements StudentService {

    private final Logger log = LoggerFactory.getLogger(StudentServiceImpl.
        class);

    private final StudentRepository studentRepository;

    private final StudentMapper studentMapper;

    private final StudentSearchRepository studentSearchRepository;

    public StudentServiceImpl(StudentRepository studentRepository,
        StudentMapper studentMapper, StudentSearchRepository
        studentSearchRepository) {
        this.studentRepository = studentRepository;
        this.studentMapper = studentMapper;
        this.studentSearchRepository = studentSearchRepository;
    }

    /**
     * Save a student.
     *
     * @param studentDTO the entity to save
     * @return the persisted entity
     */
    @Override
    public StudentDTO save(StudentDTO studentDTO) {
        log.debug("Request to save Student : {}", studentDTO);
        Student student = studentMapper.toEntity(studentDTO);
        student = studentRepository.save(student);
        StudentDTO result = studentMapper.toDto(student);
        studentSearchRepository.save(student);
        return result;
    }

    /**
     * Get all the students.
     *
     * @param pageable the pagination information
     * @return the list of entities
     */
    @Override
    @Transactional(readOnly = true)
    public Page<StudentDTO> findAll(Pageable pageable) {
        log.debug("Request to get all Students");
        return studentRepository.findAll(pageable)
            .map(studentMapper::toDto);
    }

    /**
     * Get one student by id.
     *
     * @param id the id of the entity
     * @return the entity
     */
    @Override
    @Transactional(readOnly = true)
    public StudentDTO findOne(Long id) {
        log.debug("Request to get Student : {}", id);
        Student student = studentRepository.findOneWithEagerRelationships(id);
        return studentMapper.toDto(student);
    }

    /**
     * Delete the student by id.
     *
     * @param id the id of the entity
     */
    @Override
    public void delete(Long id) {
        log.debug("Request to delete Student : {}", id);
        studentRepository.delete(id);
        studentSearchRepository.delete(id);
    }

    /**
     * Search for the student corresponding to the query.
     *
     * @param query the query of the search
     * @param pageable the pagination information
     * @return the list of entities
     */
    @Override
    @Transactional(readOnly = true)
    public Page<StudentDTO> search(String query, Pageable pageable) {
        log.debug("Request to search for a page of Students for query {}",
            query);
        Page<Student> result = studentSearchRepository.search(queryStringQuery
            (query), pageable);
        return result.map(studentMapper::toDto);
    }
}
```

### 5.4.6 Document Management

The data for the document management is also stored in the database. As documents have to possibility to be either private or publicly accessible, there is an access control in place.

*"Access control seeks to restrict access to protected resources by enforcing policies that state which subjects can perform which actions under which conditions on which resources.[51]"* [55]

### Personal Documents

These documents are only accessible by the student that uploaded them. They are marked by the public access flag that is set to false by default.

### Document Library

The library is the connection to all of the documents. Therefore, the API offers the possibility to do searches over all documents accessible by the current user. The search technology used in the prototype is Elasticsearch [4].

## 5.5 Mobile

The prototype is developed specifically as a web application for desktop devices but with the used technologies, the user interface is also responsive and can be used on mobile devices.

# 6 Outlook

The application prototype in chapter 5 provides the base for a lot of further development and expansions. Because there are so many opportunities, we will focus only on three that could allow for the most benefit in the short term.

- Including University Frameworks
- Extensions
- Microservices

Further improvements for the long term could include

- Social features (e.g. Friends, Chat)
- Including social networks (e.g. Facebook, Twitter, Instagram)
- Mobile optimized application

All improvements should raise the acceptance factor but keep the usability the same or make it better.

## 6.1 Including University Frameworks

Most universities offer an API that can be used to access the information provided to the students if logged in. With the integration of these frameworks, students would only have to provide their credentials for the university login and the available information would be transferred automatically into the time management application. This would make the application much more convenient and the acceptance would rise.

## 6.2 Extensions

This would allow for custom extensions that users could write e.g. in Groovy [47] or JavaScript that would automate certain tasks or add new types of content to the planner.

## 6.3 Microservices

This would mean that we refactor the application to use a microservice approach, where every module of the application is a microservice and they are connected via a message bus. For the frontend nothing would change, as we would still have our single API gateway, but the backend would get a lot more scalable and in the long term more performant.

# 7 Summary

This work analysis all requirements for an online time management web application that is tailored for university students. Existing applications on their own are missing features for a higher acceptance. Therefore, we checked different aspects on how to get higher acceptance for our application. The development process contained the definition of several requirements to stay on track tackling the existing problems. Choosing the best-suited technology stack is one of the biggest challenges and therefore we analyzed most state-of-the-art concepts and frameworks in the field of web application development. Two java-based prototypes were developed using Java Server Faces with Prime Faces and the JHipster generator including Angular 2+ and Spring. The later one proved to be more viable and should be used for further development. We demonstrated that, on the right basis, it is possible to build a web application helping university students time management with manageable effort. A final solution to this problem would still require more development work and usability testing in cooperation with students from different universities.

# Appendix

# List of Figures

# List of Tables

# List of Code listings

# Bibliography

[1] Oracle Corporation and/or its affiliates. *What is MySQL?* 2019. URL: `https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html` (visited on 01/07/2019) (cit. on p. 30).

[2] António Manuel Andrade. "Ensino a Distância e e-learning." In: *Instituto Educação-Universidade Católica Portuguesa* (2005) (cit. on p. 15).

[3] Autho. *JSON Web Token*. 2019. URL: `https://jwt.io/introduction/` (visited on 01/27/2019) (cit. on p. 48).

[4] Elasticsearch B.V. *Elasticsearch*. 2019. URL: `https://www.elastic.co/de/products/elasticsearch` (visited on 02/24/2019) (cit. on p. 76).

[5] Claudivan de Carvalho. *Firefly Student Planner*. 2019. URL: `https://play.google.com/store/apps/details?id=com.clawdyvan.agendadigitalaluno` (visited on 12/05/2017) (cit. on pp. 4, 5).

[6] European Commission. *ECTS Users' Guide 2015*. Publications Office of the European Union, 2015 (cit. on p. 10).

[7] Evernote Corporation. *Evernote*. 2019. URL: `https://evernote.com/` (visited on 01/04/2019) (cit. on pp. 3, 5).

[8] Firefly Learning Ltd. *Firefly Student Planner*. 2019. URL: `https://fireflylearning.com/` (visited on 01/04/2019) (cit. on pp. 2, 5).

[9] The JQuery Foundation. *JQuery*. 2019. URL: `https://jquery.com/` (visited on 01/27/2019) (cit. on p. 27).

[10] Francesco Cirillo. *Focus Booster*. 2019. URL: `https://www.focusboosterap` `com/` (visited on 01/04/2019) (cit. on pp. 3, 5).

[11] Google. *Angular Docs*. 2019. URL: `https://angular.io/docs` (visited on 01/07/2019) (cit. on p. 27).

[12] Google. *Angular Material*. 2019. URL: `https://material.angular.io/` (visited on 01/27/2019) (cit. on p. 28).

[13] Google. *Material Design*. 2019. URL: `https://material.io/design/` (visited on 01/27/2019) (cit. on p. 28).

[14] The PostgreSQL Global Development Group. *PostgreSQL About*. 2019. URL: `https://www.postgresql.org/about/` (visited on 01/07/2019) (cit. on p. 30).

[15] Chris Eppstein Hampton Catlin Natalie Weizenbaum. *Sass (Syntactically Awesome StyleSheets)*. 2019. URL: `https://sass-lang.com/documentation/file.SASS_REFERENCE.html` (visited on 01/27/2019) (cit. on p. 29).

[16] Facebook Inc. *React*. 2019. URL: `https://reactjs.org/` (visited on 01/27/2019) (cit. on p. 28).

[17] PrimeTek Informatics. *Primefaces for JavaServer Faces*. 2019. URL: `https://www.primefaces.org/showcase/` (visited on 01/05/2019) (cit. on pp. 26, 33).

[18] instin, LLC. *myHomework*. 2019. URL: `https://myhomeworkapp.com/` (visited on 01/04/2019) (cit. on pp. 2, 5).

[19] iStudiez Team. *iStudiez Pro*. 2019. URL: `https://istudentpro.com/` (visited on 01/04/2019) (cit. on pp. 3, 5).

[20] JHipster. *JHipster*. 2019. URL: `https://www.jhipster.tech/` (visited on 01/05/2019) (cit. on p. 34).

[21] JHipster. *JHipster Tech Stack*. 2019. URL: https://www.jhipster.tech/tech-stack/ (visited on 01/05/2019) (cit. on p. 34).

[22] J. Kumar and V. Garg. "Security analysis of unstructured data in NOSQL MongoDB database." In: *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*. Oct. 2017, pp. 300–305. DOI: 10.1109/IC3TSN.2017.8284495 (cit. on p. 31).

[23] Ivano Malavolta. "Beyond Native Apps: Web Technologies to the Rescue! (Keynote)." In: *Proceedings of the 1st International Workshop on Mobile Development*. Mobile! 2016. Amsterdam, Netherlands: ACM, 2016, pp. 1–2. ISBN: 978-1-4503-4643-6. DOI: 10.1145/3001854.3001863. URL: http://doi.acm.org/10.1145/3001854.3001863 (cit. on p. 19).

[24] Massachusetts Institute of Technology. *MIT License*. 2019. URL: https://opensource.org/licenses/mit-license.php (visited on 01/04/2019) (cit. on pp. iii, 14).

[25] Ritesh Mehra et al. "Configuring Java-Based Web Application Development Environment for an Academic Setting." In: Jan. 2004, pp. 111–118 (cit. on pp. 1, 14).

[26] Y. Mei and F. Lingjie. "ATS Software Framework Design Pattern and Application." In: *2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)*. Sept. 2015, pp. 141–146. DOI: 10.1109/IMCCC.2015.37 (cit. on p. 24).

[27] Luis Mena Tobar, Pedro M. Latorre Andrés, and Elena Lafuente Lapena. "WebA: A Tool for the Assistance in Design and Evaluation of Websites." In: *J. UCS* 14 (Jan. 2008), pp. 1496–1512 (cit. on p. 15).

[28] Microsoft. *.NET*. 2019. URL: https://dotnet.microsoft.com/ (visited on 03/16/2019) (cit. on p. 26).

[29] Microsoft. *ASP.NET*. 2019. URL: https://dotnet.microsoft.com/apps/aspnet (visited on 02/17/2019) (cit. on p. 26).

[30] Microsoft. *TypeScript*. 2019. URL: https://www.typescriptlang.org/ (visited on 02/17/2019) (cit. on p. 26).

[31] Inc. MongoDB. *MongoDB*. 2019. URL: https://www.mongodb.com/de/what-is-mongodb (visited on 01/05/2019) (cit. on p. 31).

[32] Moshbit GmbH. *Studo*. 2019. URL: https://studo.co/ (visited on 01/04/2019) (cit. on pp. 4, 5).

[33] Mozilla and individual contributors. *JavaScript*. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (visited on 01/27/2019) (cit. on pp. 26, 27).

[34] Thomas Müller. *H2*. 2019. URL: http://www.h2database.com/html/main.html (visited on 01/05/2019) (cit. on p. 31).

[35] My Study Life, Ltd. *My Study Life*. 2019. URL: https://www.mystudylife.com/ (visited on 01/04/2019) (cit. on pp. 4, 5).

[36] Henrik Frystyk Nielsen et al. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. June 1999. DOI: 10.17487/RFC2616. URL: https://rfc-editor.org/rfc/rfc2616.txt (cit. on p. 23).

[37] Oracle. *JavaServer Faces Technology*. 2019. URL: https://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html (visited on 01/27/2019) (cit. on pp. 25, 27).

[38] Inc. Pivotal Software. *Building an Application with Spring Boot*. 2019. URL: https://spring.io/guides/gs/spring-boot/ (visited on 02/17/2019) (cit. on p. 25).

[39] Inc. Pivotal Software. *Spring*. 2019. URL: https://spring.io (visited on 02/17/2019) (cit. on p. 24).

[40] Inc. Pivotal Software. *Spring Boot*. 2019. URL: https://spring.io/projects/spring-boot (visited on 02/17/2019) (cit. on p. 24).

[41] Inc. Pivotal Software. *Spring Data*. 2019. URL: https://spring.io/projects/spring-data (visited on 02/17/2019) (cit. on pp. 24, 25).

[42] Inc. Pivotal Software. *Spring Projects*. 2019. URL: https://spring.io/projects (visited on 02/17/2019) (cit. on p. 24).

[43] Inc. Pivotal Software. *Spring Security*. 2019. URL: https://spring.io/projects/spring-security (visited on 02/17/2019) (cit. on p. 24).

[44] Inc. Pivotal Software. *Unterstanding HATEOAS*. 2019. URL: https://spring.io/understanding/HATEOAS (visited on 01/05/2019) (cit. on p. 23).

[45] Inc. Pivotal Software. *Unterstanding REST*. 2019. URL: https://spring.io/understanding/REST (visited on 01/05/2019) (cit. on p. 22).

[46] Inc. Pivotal Software. *Unterstanding SOAP*. 2019. URL: https://spring.io/understanding/SOAP (visited on 01/05/2019) (cit. on p. 22).

[47] Apache Groovy project. *Groovy - A multi-faceted language for the Java platform*. 2019. URL: http://groovy-lang.org/ (visited on 02/23/2019) (cit. on p. 78).

[48] Inc. Red Hat. *Wildfly*. 2019. URL: http://wildfly.org/ (visited on 01/05/2019) (cit. on p. 33).

[49] Remember The Milk. *Remember The Milk*. 2019. URL: https://www.rememberthemilk.com/ (visited on 01/04/2019) (cit. on pp. 3, 5).

[50] Usha Sakthivel, Neha Singhal, and Pethuru Raj Chelliah. "RESTful web services composition & performance evaluation with different databases." In: Dec. 2017, pp. 1–4. DOI: 10.1109/ICEECCOT.2017.8284608 (cit. on p. 23).

[51] R. S. Sandhu and P. Samarati. "Access control: principle and practice." In: *IEEE Communications Magazine* 32.9 (Sept. 1994), pp. 40–48. ISSN: 0163-6804. DOI: 10.1109/35.312842 (cit. on p. 76).

[52] Rosa Silva and António Andrade. "Development of a Web Application for Management of Learning Styles." In: *J. UCS* 15 (Jan. 2009), pp. 1508–1525 (cit. on p. 15).

[53] SmartBear Software. *SOAP vs REST Infographic*. 2018. URL: https://www.soapui.org/resources/infographic/api-testing/soap-vs-rest-infographic.html (visited on 01/05/2019) (cit. on pp. 22, 23).

[54] Robert Thurlow. *RPC: Remote Procedure Call Protocol Specification Version 2*. RFC 5531. May 2009. DOI: 10.17487/RFC5531. URL: https://rfc-editor.org/rfc/rfc5531.txt (cit. on p. 22).

[55] Kam S. Tso, Michael J. Pajevski, and Bryan Johnson. "Access Control of Web and Java Based Applications." In: *Proceedings of the 2011 IEEE 17th Pacific Rim International Symposium on Dependable Computing*. PRDC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 320–325. ISBN: 978-0-7695-4590-5. DOI: 10.1109/PRDC.2011.54. URL: http://dx.doi.org/10.1109/PRDC.2011.54 (cit. on p. 76).

[56] W3C. *Cascading Style Sheets*. 2019. URL: https://www.w3.org/Style/CSS/ (visited on 01/27/2019) (cit. on p. 29).

[57] W3C. *HTML 5.2*. 2017. URL: https://www.w3.org/TR/html5/ (visited on 01/27/2019) (cit. on p. 29).

[58] W3C. *WAI-ARIA*. 2019. URL: https://www.w3.org/WAI/standards-guidelines/aria/ (visited on 01/05/2019) (cit. on p. 29).

[59] W3C. *XML WSDL*. 2019. URL: https://www.w3schools.com/xml/xml_wsdl.asp (visited on 01/05/2019) (cit. on p. 22).