



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelor Thesis

Cristian-Ion Cheibas

Creating a secure web service for generating
personalized calendar-files

Cristian-Ion Cheibas
Creating a secure web service for generating
personalized calendar-files

Bachelor Thesis based on the study regulations
for the Bachelor of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the Hamburg University of Applied Sciences

Supervising examiner : Prof. Dr. rer. nat. Henning Dierks
Second Examiner : Prof. Dr.Ing. Rainer Schoenen

Day of delivery 9. Dezember 2016

Cristian-Ion Cheibas

Title of the Bachelorthesis

Creating a secure web service for generating personalized calendar-files

Keywords

Generate, Calendar-File, Secure, Service, Java, Web, Personalized, Application, Schedule

Abstract

Report describing the implementation and design of a secure web service generating personalized calendar files. The service is a web application running on a server. It offers HAW Hamburg students the ability to create their own personalized calendar file.

Cristian-Ion Cheibas

Titel der Arbeit

Herstellung eines sicheren Webdienstes für personalisierte Kalenderdateien

Stichworte

Generieren, Kalenderdatei, Sicher, Dienst, Java, Web, Personalisiert, Anwendung, Stundenplan

Kurzzusammenfassung

Bericht, der das Design und die Umsetzung einer web-basierten Anwendung zur Erstellung personalisierter Kalenderdateien. Der Dienst ist eine Java Web Anwendung, die dazu ausgelegt wurde auf Servern zu laufen. Dies bietet Studenten der HAW Hamburg die Möglichkeit, ihre eigenen Stundenpläne zu erstellen.

Contents

List of Tables	6
List of Figures	7
Listings	8
1. Introduction	9
1.1. Requirements	9
2. Analysis	10
2.1. Java Web Application	10
2.2. iCalendar Standard	11
2.3. Maintainability	13
2.4. Security	14
2.4.1. XSS Attacks	14
2.4.2. Preventive Measures	15
2.5. Java-bean Concept	15
3. Design	17
3.1. Design Pattern	17
3.2. Front-end Design	18
3.2.1. Website Flow	18
3.2.2. Java Server Faces	20
3.2.3. Primefaces	22
3.2.4. Selection Phase	23
3.2.5. Visualization Phase	25
3.3. Back-end Design	27
3.3.1. Build Automation and Dependency Management	27
3.3.2. Software Architecture	29
3.3.3. Web Server	30
3.3.4. reCAPTCHA API	30

4. Realisation	32
4.1. Building the Project	32
4.2. Managing Dependencies	33
4.3. Testing	34
4.4. Logging	36
4.5. Web Parser	37
4.6. Java Mail	38
4.7. JSON Parsing	39
4.8. Regular Expressions	40
4.9. Java Source Codes	42
4.9.1. The Model Package	42
4.9.2. The Util Package	49
4.9.3. The Singleton Concept	49
4.9.4. The Controller	58
4.9.5. The View	60
4.9.6. Index.html	61
4.9.7. Scroller.xhtml	64
4.9.8. calendar.xhtml	65
4.10. Issues	67
5. Conclusion	68
5.1. Summary	68
References	69
Appendices	71
A. Setup Guide	72
B. Default Configuration File	73
C. DVD Contents	74

List of Tables

2.1. Java-bean Advantages and Disadvantages	15
2.2. Context Annotations and their Life-cycle	16
4.1. Speed Comparison of Logging platforms	36
4.2. Regular Expressions Patterns	40

List of Figures

3.1. Model View Controller Design Pattern	17
3.2. Flow Activity Diagram	18
3.3. SelectManyCheckbox	20
3.4. SelectOneRadio	20
3.5. Pie Chart	22
3.6. Donut Chart	22
3.7. Select One Menu with Default Majors	23
3.8. JSF accordionPanel & dataTable Components	23
3.9. Inactive BooleanButton	24
3.10. Pressed BooleanButton	24
3.11. Loading Bar	24
3.12. Calendar Month View	25
3.13. Calendar Week View	25
3.14. Email Input Panel	26
3.15. Email Content Example	26
3.16. Default Maven Project Layout	28
3.17. Captcha Validation	31
4.1. JUnit Testing Interface	34
4.2. JSON Object Hierarchy	39
4.3. Pattern Class Diagram	42
4.4. DateFormat Class Diagram	44
4.5. Department and Major class Diagrams	45
4.6. Course and Event Class Diagrams	47
4.7. Event.toString() Sequence Diagram	48
4.8. ICS Provider and Course Class Diagrams	51
4.9. InitializeService.contextInitialized() Sequence Diagram	55
4.10. EntityConverter.getAsString() Sequence Diagram	57
4.11. CombinedView.getFile() Sequence Diagram	59
4.12. Maven Web Application Hierarchy	60
4.13. The tabView Component	62
4.14. The growl Component	63

Listings

2.1. iCalendar Example	12
3.1. Value Reading without EL	21
3.2. Value Reading with EL	21
4.1. Maven Build Configuration	32
4.2. Maven JSF Dependency	33
4.3. Logger Usage	36
4.4. Jsoup element selection	37
4.5. JavaMail Send-Email Pseudocode	38
4.6. Json Object Constructor Declaration	39
4.7. Applying a Regex in Java	41
4.8. Creating Event Objects	47
4.9. Synchronized Singleton Implementation	49
4.10. Locating the configuration.json File	50
4.11. Mapping the JSON File to Objects	51
4.12. Binding Courses to a Map	53
4.13. tab element	62
4.14. selectOneMenu element	64
4.15. schedule element	65

1. Introduction

Every semester, the students of HAW Hamburg's "Technik und Informatik" department are assigned a schedule based on their study-year[11]. Presented in a `.pdf` format the schedules are not customizable. Most students however, require a custom schedule based on multiple factors like the time of laboratory groups chosen and whether or not they take other courses from different study years. In order to offer students the flexibility of creating their own timetable a secure website has to be created.

Throughout the thesis, Unified Modeling Language(UML) diagrams are used to detail how the website is functioning.

1.1. Requirements

Additionally, the bachelor thesis has to conform to a set of requirements:

- In order to offer the users a selection of courses, the required calendar files have to be downloaded from an online repository.
- The fact that calendar files are updated often also requires the web service to keep them up to date.
- Security of the system is another important requirement. Even though the schedule information is public the possibility of linking it to a student is not. Every timetable has to be kept private.
- The back end logic of the service has to be written in the Java programming language.
- The service should be easy to maintain and allow performing small changes and configurations (like study regulation updates) without modifying the source code.
- The generated, personalized calendar files have to be compliant with the iCalendar[4] (`.ics`) file format standard to ensure compatibility with most existing calendar software.

2. Analysis

2.1. Java Web Application

Web applications are, by nature, programs able to run in a web browser that communicate with the user through a server. Not only are web applications easily accessible, but the ability to maintain one without installing hundreds of software copies on user computers is a big reason for their popularity.

Like most applications, a Java web application consists of several parts, some of which have graphical user interfaces and some of which do not require one. In order to create a graphical interface capable of being interpreted by a web browser the application requires usage of additional markup and scripting languages(HTML, JavaScript, CSS). Additionally the Java programming language provides the Java Enterprise Edition(Java EE) platform which is designed to "provide programming interfaces and runtime environments for developing and running large-scale, multi-tiered, scalable¹, reliable, and secure network applications"[17].

Java Enterprise offers a vast selection of frameworks and technologies to use in a web application, but the core of it all is the Servlet API that is used to specify how Java behaves over Hypertext Transfer Protocols(HTTP). Servlet allows an application to capture a user request, process it and send an appropriate response.

In order to understand Servlet one must first of all realize that it is an Application Programming Interface (API), a well defined set of functions and procedures for request/response handling and is rarely used directly, however higher level Java web frameworks often implement the API to benefit from its well defined communication strategies. More detail on higher level frameworks implementing Servlet that are used throughout this thesis can be found in Chapter 3.2.2.

Java Web Applications are packaged in a `.war` (Web Application Archive) file which constitutes a collection of Java classes, configuration files, Servlet implementations, HTML pages and related files. All the Java EE compatible servers and containers support deployment of `.war` files.

¹Scalable - Able to be changed in size or scale.

2.2. iCalendar Standard

iCalendar(RFC 2445) is a computer file format specification created in 1998 for calendar data interchange between users and devices. Since being produced, the specification has become dominant on the web and is supported by almost all desktop and mobile calendar software applications. The specification has been revised in 2009 and updated to iCalendar(RFC 5545)[4, p. 7] or version 2.0, the one used in the scope of this thesis. iCalendar files usually have the `.ics` file extension, however `.ical` and `.icalendar` are also used.

The RFC 5545 specification requires iCalendar files to follow a well defined data model and consists of multiple calendar components or objects, each with a set of unique properties and parameters. The start and end of an `.ics` file is denoted by the tags `BEGIN:VCALENDAR` and `END:VCALENDAR` respectively. Everything between these two tags is referred to as the "calendar body".

Inside the calendar body, all components have to start with the letter "V"[4, p. 7]. For example, `VEVENT` is an event component, `VJOURNAL` denotes a daily journal and `VTODO` refers to the to-do component. Additionally a set of requirements and limitations is in place to define the data model.

- 1) An iCalendar body must include the `PROID` and `VERSION` properties, but neither must occur more than once.
- 2) Throughout the file, at least one calendar component has to be defined.
- 3) Lines of text should not be longer than 75 octets, excluding the line break[4, p. 9].
- 4) Component properties like `DTSTART` must not occur more than once (Exception: `DTEND`).

All the iCalendar files that belong to the "Technik und Informatik" department can be found online, in a web repository¹[10]. Upon inspecting the files it can be noted that they contain only `VEVENT` components to denote lecture times. The start and end of a event component is specified by the `BEGIN:VEVENT` and `END:VEVENT` keywords respectively.

¹Repository - A location where data is stored and managed.

Below is a breakdown of all the event component properties used by the repository files:

- `SUMMARY` - contains the study major, semester and course name information
- `LOCATION` - specifies the location of the event
- `UID` - unique identifier for the calendar component
- `DTSTART` - specifies the start time together with the time-zone information
- `DTEND` - specifies the end time together with the time-zone information

The start and end time properties, specify the timezone using a `TZID` parameter. Since all events happen in the same city and timezone the parameter is constant - "Europe/Berlin". Date and time information has to follow a specific format as well, `20161017T094500` is an example of the date-time settings for 9:45 A.M, 17th of October 2016. Note that the character 'T' is used to separate date from time.

The following snippet is an iCalendar(RFC 5545) file example that describes an event taking place in room 1283 which starts at 8:10 A.M. CET, October 26, 2016 and ends on the same day at 11:25 A.M. CET.

```
1 BEGIN:VCALENDAR
2 VERSION:2.0
3 PRODID:-//HAW-Hamburg - Dept EuI//
4 CALSCALE:GREGORIAN
5 BEGIN:VEVENT
6   SUMMARY:IE3-SS1
7   LOCATION:1283 Stand 26-10-2016
8   UID:161026200919.100054@etech.haw-hamburg.de
9   DTSTART:TZID=Europe/Berlin:20160923T081000
10  DTEND:TZID=Europe/Berlin:20160923T112500
11 END:VEVENT
12 END:VCALENDAR
```

Listing 2.1: iCalendar Example

2.3. Maintainability

Maintainability is one of those properties that is much easier to recognize by its absence. For instance when something a developer thinks should take an hour, takes days to fix or implement. For an author knowing much more about the code, it is difficult to evaluate the maintainability of a software, however a set of coding practices and standards has been set up to aid developers in creating easily maintainable code. For the scope of this thesis, a code is considered maintainable if another developer is able to read, understand, test, modify and extend the code without asking for the author's help.

In order to achieve that the code has to be, among many others:

- Well documented - Understandable comments and javadocs¹, descriptive package, class, function and variable names.
- Clearly formatted - Indentation present, formatting guidelines applied consistently.
- Limited in flow - Only one software entry and exit point exists and both are clearly visible.
- Testable - Unit and Integration tests (defined in Chapter 4.3) are set up to notify developers if changes modify expected behavior.
- Avoiding "code smell" - No excessively large or unreasonably small classes exist, methods do not take more parameters or return more data than functionally required, identical or very similar code does not exist.
- Cyclomatically simple - Large branches, recursive functions or loops are broken down into smaller functional parts.
- Extensive Logging - Keeping a log of important events and transactions in order to provide insight in cases of unexpected behavior and help diagnose issues.

Additionally, every semester the university changes currently existing curricula to improve the quality of studies offered. Majors and courses are added, removed or renamed and in turn, calendar files are updated to match the new courses.

Curriculum changes are not published in a fixed format, which makes fetching them from the web impractical, while at the same time hard-coding the current majors and courses will cause the service to quickly become obsolete. The solution is reading the current curricula from a configuration file. This way the software is able to build itself accordingly each study semester as long as the configuration file is updated by the service maintainer.

¹Javadoc - Large multi-line documentation placed before class or method declarations that describes the functionality, exceptions thrown, parameters used and is visible in the development environment.

2.4. Security

Another crucial aspect to analyze is web security, and most importantly the safety of personal information - privacy. As mentioned in the introduction, even though course information is public, it should not be linked to a student. Generally, the more technologies and frameworks used, the greater is the risk of creating security loopholes. According to Open Web Application Security Project (OWASP)[6] the three most popular security vulnerabilities of 2013 are:

- 1) SQL Injection Attacks
- 2) Faulty User Authentication
- 3) Cross-site Scripting(XSS)

The web application does not rely on user authentication or database communication, thus SQL injections and faulty user authentication do not pose a threat, however the presence and usage of Javascript on the front-end make the service vulnerable to XSS Attacks.

2.4.1. XSS Attacks

Cross-site scripting(XSS) attacks are a form of injection vulnerability that relies on injecting scripts into input fields or the address bar and is best understood by example:

Consider a website having the search functionality, searching for an item withing the website is referenced by the following url `www.example.com/search=input`. In case an attacker notices the vulnerability and choses to replace the search term with `<script>alert('Input banking information');</script>` as well as successfully sends the new url to an unsuspecting user, a poorly designed website referencing text directly from the address bar will cause a browser prompt to appear asking for sensitive information.

In simpler terms, the application should not directly reference(output on a web page) the inputs provided by users through both regular HTML input fields or the address bar. If this is negated, harmless looking text like `Welcome '<input>'!` can hide a script behind it.

2.4.2. Preventive Measures

XSS attacks can be detected and defended against at two points in an application. By validating the user interface input (and refusing faulty inputs) as well by sanitizing fields on the back end. Sanitizing Inputs means detecting potentially dangerous characters and replacing them with their ASCII character code, forcing their treatment as chars and not keywords. Together with a clever input validation strategy the web application can reliably prevent XSS attacks, which are impossible to perform if the attacker cannot make use of the HTML `<script>` tags. Additionally the web application has to be designed independent from address bar inputs.

2.5. Java-bean Concept

A term specific to the Java Platform, Java-beans represent classes that encapsulate many objects into one single object, the bean. Beans are serializable¹, required to have a zero-argument constructor and their properties are accessed using getter and setter functions. The bean has been created with the goal of providing reusable software components. Beans directly represent the Model side of a MVC pattern (described in Chapter 3), as they are designed to carry information.

Advantages	Disadvantages
Properties and methods of a bean can be controlled by another application	Accessing the properties requires creation of getters and setter for most, which leads to boilerplate code.
The state of a bean can be saved to a persistent storage and then reloaded or restored	Java-beans can only be mutable and lack the advantages of immutable objects.
Beans can generate and receive events to communicate with other objects	Java-beans require a default constructor which can lead to beans being instantiated in an invalid state.

Table 2.1.: Java-bean Advantages and Disadvantages

Java-beans can be further improved by using the Contexts and Dependency Injection framework.

¹Serializable - A java object can be converted to a series of bytes and saved or transported, after which it can be converted back and is returned to the initial state.

CDI for Java EE has been introduced as part of Java EE 6[18], and has become one of the most important components of the platform, it can be separated into two packages by functionality:

- Context - The ability to bind Java components to life-cycle contexts
- Dependency Injection - The ability to inject components into an application

A context, in simple terms is a bean storage repository, however multiple contexts can exist at the same time, and most importantly, every context has a timed life-span. Meaning the data in each can expire if, for example the user session is over. This functionality can be used to limit the time Java-beans are kept alive. For example, once a user opens the course selection screen, a Java-bean is assigned to his session which will contain a list of selected courses. Once the user has generated his calendar, there is no more need to keep his selection information and the bean is destroyed and removed from the context, the same happens if the user is inactive for more than 15 minutes. This helps reduce the resources required for the application to run.

Binding a bean to a specific context is done with the help of CDI, by marking a Java-bean class with one of the respective annotations:

Annotation	Bean Lifetime
@ApplicationScoped	Alive as long as the application(server) is running
@SessionScoped	Alive until the user finishes his browsing session
@ViewScoped	Lives until the user stops interacting with the page
@RequestScoped	Lives exactly one HTTP request

Table 2.2.: Context Annotations and their Life-cycle

3. Design

This chapter describes the decisions, reasonings and results for the application design. The source code design, however is not covered in this chapter, even though snippets of code might be used to enforce a statement.

3.1. Design Pattern

One of the most successful and widely known design patterns for web applications is the Model View Controller(MVC) pattern. It is based around dividing the software into three interconnected parts that are designed to separate internal information from the user.

- Model - Carries data on the back-end
- View - Displays the web page
- Controller - Handles user requests from the page

MVC also provides a set of rules and limitations to each actor. The model manages data based on controller commands and sends them to the view for display. The view reads changes and generates an output. The Controller receives user instructions and updates the model accordingly. ¹

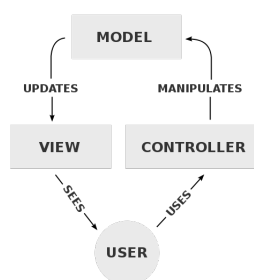


Figure 3.1.: Model View Controller Design Pattern

¹Picture 3.1 credit – [Wikimedia](#).

3.2. Front-end Design

3.2.1. Website Flow

Before the service can generate a calendar file, the user has to be offered a selection of courses. Since there are over 600 calendar files, filtering the unused ones and dividing them according to their specific major would greatly simplify the user interaction as well as the server load.

Every course should also provide the basic information like the event time and date, so the user can easily choose between courses with multiple options.

Below is the UML activity diagram of the flow from the user's perspective:

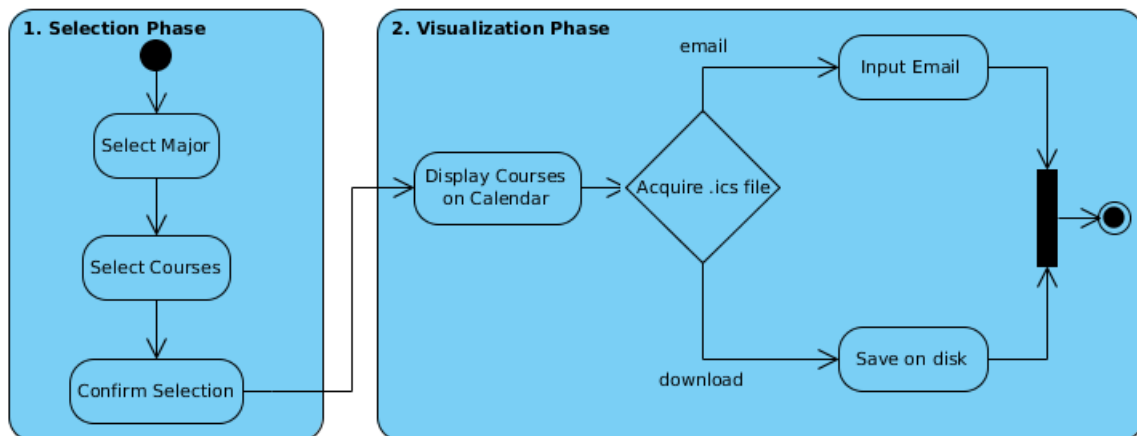


Figure 3.2.: Flow Activity Diagram

Once a course is selected it has to be visibly highlighted to provide a visual indicator for the user and help avoid confusion. When all the necessary courses have been chosen, the selection is confirmed by clicking the "Proceed" button and the user is redirected to the second part of the flow - displaying the selection on a calendar, an interface component where a user can visualize the chosen courses and has the option of sorting either by month or week. The calendar component also offers the ability of downloading and emailing the generated calendar-file to the user. If mailing the file is chosen, an windows is opened containing an input field asking the user to introduce an electronic mail address.

However, it is never a good idea to bind the user to a restrained flow, which means the UML diagram describes the simplest way a user can navigate. In reality the user can switch between the selection phase and visualization phase at will and jump to any action at any time. Faulty requests however, like asking for a file download with no courses selected are not executed, and instead notify the user of his/her mistake.

3.2.2. Java Server Faces

Java Server Faces(JSF) technology is a server-side, component-based MVC framework for developing user interfaces. JSF is part of the Java EE specification for building a web application.

By using JSF the base Servlet API is extended with additional functionality and simplified in terms of development complexity. For example, the core HTTP communication class of JSF, `FacesServlet.class` acts as an improved request/response controller. FacesServlet removes the need of explicitly programming the HTTP Communication from the developer's hands. Tasks such as, rendering the response, validating inputs or converting objects are done automatically at the cost of less fine-grained control(as is the case with every component based MVC framework).

JSF offers a huge set of improvements to all the elements of the Model View Controller pattern, below are only the ones relevant for this web application.

- Model - Automated addition of Java-beans to the context, if marked with the `@ManagedBean` annotation, ability to develop converters from HTML text to Java Objects and back.
- View - Ability to use multiple xHTML(extended HTML) components, as well as declare custom ones thanks to Facelets, a powerful and easy to use declaration language.
- The Controller - Automated request/response communication, allowing the developer to focus only the relevant logic.

JSF components are the basic building block of the web application user interface. A component is one highly-configurable and well-defined interface element, that can range in complexity from a simple text output field to entire data tables. JSF components can be directly associated with Java-beans and are backed up by helper converters, action listeners and data validators by definition. Below is an example of two basic components used for data selection, `SelectManyCheckbox` and `SelectOneRadio`, both displaying the same bean containing an array of 3 Strings.

Item 1 Item 2 Item 3

Figure 3.3.: SelectManyCheckbox

Item 1 Item 2 Item 3

Figure 3.4.: SelectOneRadio

Facelets on the other hand, offer support for the Expression Language (EL), which can be used to access back-end data. EL expressions take the form of `#{bean.variable}` and can be used to reference bean properties, solve arithmetical, logical operations or execute back-end functions from front-end. In order to better understand the advantage in code maintainability and simplicity offered by EL here is a comparison between a View reading a bean(`user`) property(`name`) value with and without the Expression Language.

```
<%
User user = applicationContext.findUser();
if (user != null) {
    String name = user.getName();
    if (name != null) {
        return name;
    }
}
%>
```

Listing 3.1: Value Reading without EL

```
#{user.name}
```

Listing 3.2: Value Reading with EL

EL does all the tedious, boilerplate work like checking if the properties referenced beans are null, finding the right user from the context relevant to the expression and returns the result. It is also much easier to read, understand or develop, and is used extensively throughout this thesis as the main method of communication between Model-Controller and View.

3.2.3. Primefaces

Due to the nature of JSF and the ability to define custom components, multiple developers are able to contribute with their own components to enrich the already large UI library.

As a result, component libraries have been created like Primefaces. Primefaces is a set of rich components, making use of powerful libraries and technologies like jQuery to offer more variety as well as more fine-grained communication capabilities, and are designed to be responsive to mobile devices and touch screens. Compared to the simple and basic JSF components, the ones offered by Primefaces look like finished functional products on their own, for example here are two Primefaces chart components:

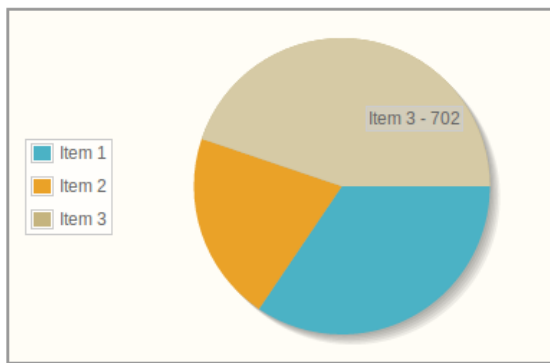


Figure 3.5.: Pie Chart

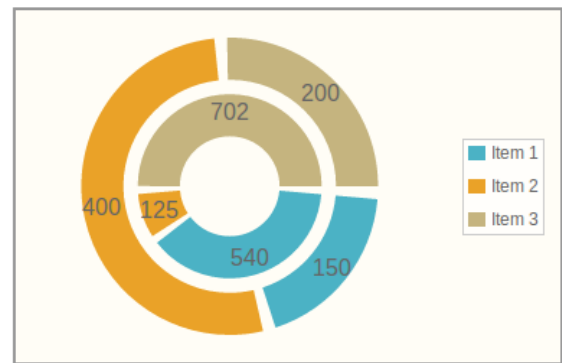


Figure 3.6.: Donut Chart

There are a total of 18 chart components available in Primefaces, ranging from static images to updating live or even interactive.

3.2.4. Selection Phase

As mentioned previously, selecting courses has been chosen as the entry point into the web flow. In order to select courses, the user has to specify a major first. The list of currently available majors is fetched from the [configuration file](#). A JSF `selectOneMenu` element has been chosen as it is the most intuitive design-wise.

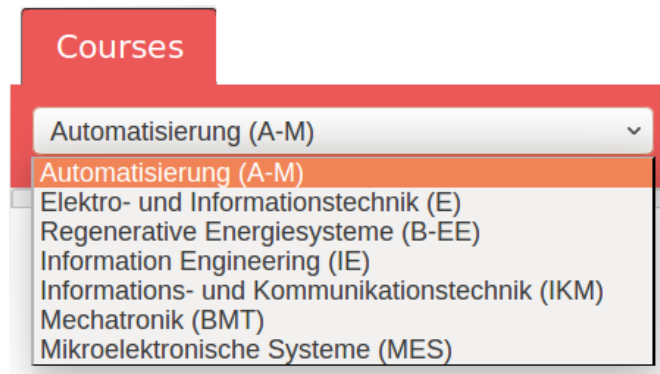


Figure 3.7.: Select One Menu with Default Majors

By default, the major is selected alphabetically, thus "Automatisierung(A-M)" is the default choice on page load. Selecting a major controls the list of relevant courses offered to a user in the form of a list of Primefaces JSF components named `accordionPanel`, which can be expanded or contracted on click.

IE3-DI				+ Add
IE3-DIL-01				+ Add
#	Week Number	Date	Start/End Time	
1	Week 42	18-10-2016	08:10/11:25	
2	Week 45	08-11-2016	08:10/11:25	
3	Week 48	29-11-2016	08:10/11:25	
4	Week 51	20-12-2016	08:10/11:25	
IE3-DIL-02				+ Add

Figure 3.8.: JSF accordionPanel & dataTable Components

The image depicted in Figure 3.8, shows the mobile (desktop version contains more information, like location and day) selection screen for 3 courses, each with their respective `accordionPanel` element. The first panel IE3-DI is inactive, the second panel IE3-DIL-01 is expanded revealing the necessary information for the user to make the decision whether or not the course fits into the schedule, and the last panel IE3-DIL-02 is highlighted on tap (mouse-over for desktop).

From the image it can also be understood that all the `accordionPanel` elements contain a `dataTable` element inside. The tables are not intractable, adding an ability to sort or remove events at will would only needlessly complicate the service. The difference between plain HTML table and JSF tables is that the latter can accept Java-beans containing data structures like Lists or Sets and populate the table in one variable read with the entire List. In the web application a List-containing bean is referenced to the `dataTable` element for every column seen.

Every `accordionPanel` also has a `selectBooleanButton` element on the opposite side of the title. A big difference between a JSF boolean button and a regular HTML button is in changing the status on click. Regular buttons are used only for performing an action. While `selectBooleanButton` acts as an indicator as well, lighting up green and changing the text and icon when pressed once. Pressing the boolean button in its active state will reset it back to inactive and remove the course from selection.

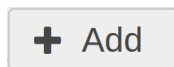


Figure 3.9.: Inactive BooleanButton



Figure 3.10.: Pressed BooleanButton

When selecting a new major in the `selectOneMenu` which is particularly large, the loading process might take anywhere from 0.5 to 4 seconds, depending on the internet speed of the user. In order to provide a visual indicator for the loading to users, the website displays a `.gif` image of a progress bar over the faded out selection tab.

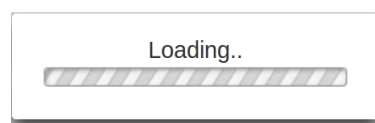


Figure 3.11.: Loading Bar

Finally, once the selection is over the user can proceed to the second part of the flow by either selecting the calendar tab, or clicking the "Proceed" button located on top of the selection tab.

3.2.5. Visualization Phase

Once the user has selected the desired courses, the tab view is replaced with a calendar containing his selection arranged by day. As well as the buttons for downloading and emailing a iCalendar file. Compared to the Selection Phase, this view is the same on both mobile and desktop versions of the website because the space allows it to still be visible.

W	Sun	Mon	Tue	Wed	Thu	Fri	Sat
49	27	28	29	30	1	2	3
		12:10p IE1-SO1	12:10p IE1-EE1	8:10a IE1-EE1-01	8:10a IE3-DI		
50	4	5	6	7	8	9	10
	8:10a IE3-EM	12:10p IE1-EE1		8:10a IE3-DI			
	12:10p IE1-SO1						
51	11	12	13	14	15	16	17
	12:10p IE1-SO1						
52	18	19	20	21	22	23	24
	8:10a IE3-EM		8:10a IE1-EE1-01	8:10a IE3-DI			
53	25	26	27	28	29	30	31
1	1	2	3	4	5	6	7

Figure 3.12.: Calendar Month View

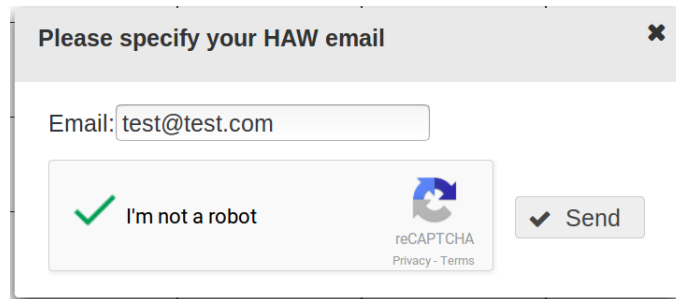
The calendar can also be viewed by week, where the current day is highlighted in blue.

W50	Sun 12/4	Mon 12/5	Tue 12/6	Wed 12/7	Thu 12/8	Fri 12/9	Sat 12/10
All Day							
8am		8:10 - 11:25 IE3-EM			8:10 - 11:25 IE3-DI		
9am							
10am							
11am							
12pm		12:10 - 3:40 IE1-SO1	12:10 - 3:40 IE1-EE1				
1pm							
2pm							

Figure 3.13.: Calendar Week View

The calendar allows switching between months back and forward. When desired, the user can return to the current date by clicking the "Current Date" button.

The upper left icon depicting an arrow pointing on a hard drive, prompts a download of the generated, personalized calendar file on the hard disk, while the email icon opens a widget asking for the user email, a Captcha confirmation (described in Chapter 3.3.4) and a Send button to confirm sending the email.



The image shows a dialog box titled "Please specify your HAW email". It contains an "Email:" label followed by a text input field containing "test@test.com". Below the input field is a reCAPTCHA widget with a green checkmark and the text "I'm not a robot". To the right of the reCAPTCHA is a "Send" button with a checkmark icon. The reCAPTCHA logo and "Privacy - Terms" link are also visible.

Figure 3.14.: Email Input Panel

The emails are intentionally configured to be treated as calendar events by the most well-known mailing clients (ex. Gmail, Thunderbird). And instead of showing a simple attachment, offer the user the choice of saving the calendar file in their built-in calendars.

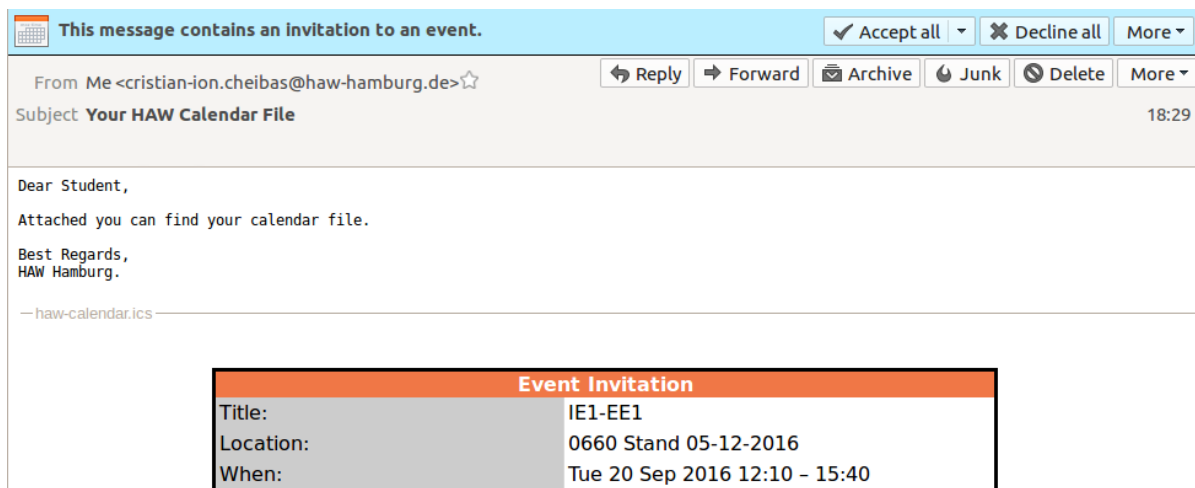


Figure 3.15.: Email Content Example

3.3. Back-end Design

3.3.1. Build Automation and Dependency Management

Two crucial issues in software development are build description and dependency management.

The process in which all the files are combined into one resulting software or product in a consumable form is called build. Building the software compiles all the source codes into binary code, packages the result assuming all the automated tests run successfully.

In today's world build automation has become standard practice, developers specify details of the process and with the usage of foreign tools, the code is automatically compiled and packaged on any machine and operating system capable of running the tool. A build automation process can also be configured to package the result in a `.war` file, making it much easier to deploy the resource on a server, however, in our case, the greatest benefit offered by a build automation tool is the often included dependency management engine. Previously, a projects dependency management consisted of manually downloading and importing all the `.jar` files required, or for more advanced developers, creating scripts that would download all the libraries at once. Manually doing all of this is a discouraging process, and even more so when you consider the existence of transient dependencies(dependencies of dependencies).

Java EE offers multiple technologies for usage in various applications, however grouping them up into one development kit would lead to excessively large downloads and bloated software. Not every developer is building a web application or plans to send emails. As a result, multiple drivers, frameworks and libraries that are part of the Java Enterprise Family have to be imported into the project. The Servlet functionality, for example, which is a core library for this thesis is not present in the default Java packages and has to be imported.

Fortunately, a dependency management engine is the solution to this problem, by configuring the dependency engine a developer can be sure that any machine running the project automatically downloads all the required libraries of the same version and from the same repository and builds it in the exact same way.

The Java ecosystem is dominated by three competing build and dependency management tools, Apache Ant&Ivy, Apache Maven and Gradle.

Upon analyzing every tool, the conclusion that each have their strong and weak points has been made, however, ultimately, the decision to use Apache Maven has been made because of the following advantages over the competition:

- Standard Project Layout – Maven projects provide developers with a well defined layout, and even though it limits the flexibility, it also greatly boosts understanding what type of files go where, which makes it more maintainable.
- Superior Dependency Management – Maven identifies unused and transitive dependencies, is able to display them in a hierarchical tree and most importantly dependency downloads are built by default into the project classpath¹.
- Biggest IDE and Server Support – Supported by most development environments and servers without the use of plugins², which means that migrating from an IDE or Server to another does not have to be explicitly configured.
- Largest User Base – Most Java developers are more familiar with Maven when compared to Ant or Gradle, which also means most developers do not have to learn a new build tool to be able to understand how it works.

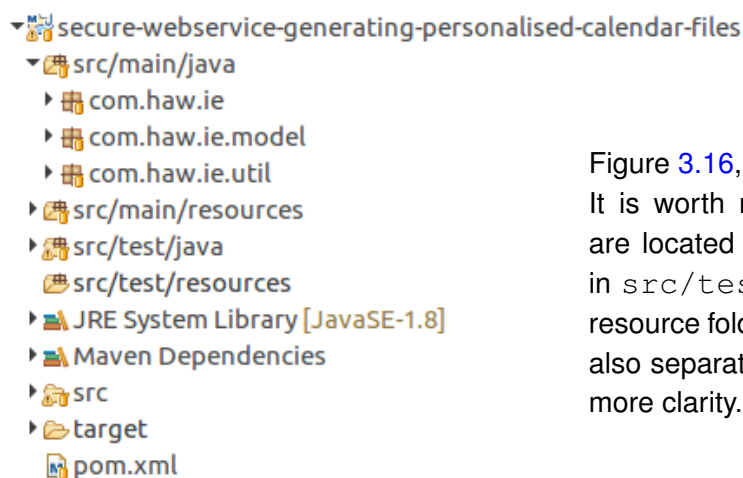


Figure 3.16, shows the application layout. It is worth noting that the source packages are located in `src/main/java`, and tests in `src/test/java`, each with a dedicated resource folder. The Maven dependencies are also separated from the JRE ones, to provide more clarity.

Figure 3.16.: Default Maven Project Layout

The Maven configuration is defined by the `pom.xml` file which is stored in the root directory of the project and contains the dependency, build and packaging details. Maven is open-source and free (unlike a developer's time spent manually managing dependencies).

¹Java Classpath - Parameter specifying the location of the packages and classes used by an application.

²Plugin(syn. Addon) - Software component that adds additional functionality to an already existing application.

3.3.2. Software Architecture

When building a project from scratch, a developer has to make a set of decisions that will shape the software architecture. A crucial part in designing the service is compatibility between all the tools used. For the scope of this thesis the key tools are - source codes, frameworks and libraries as well as the web application server. It is generally good practice to use the latest stable versions of all the assets since most tools used during this thesis have been under development for tens of years by hundreds of developers, and every release multiple improvements are added to readability, speed, security and the overall integrity of the tools. However, in some cases, the choices made during software development can shape the decision as well.

As mentioned previously, the calendar files describing courses are stored in a web repository. With an active internet connection, the service has access to the required information, but the downloaded calendar files still have to be parsed¹ in order to extract and sort the relevant information.

One option would be saving files on the local storage and accessing them whenever required, however due to the similarity of calendar `VEVENT` components with Java objects and the benefit of Java being an Object Oriented Programming Language, the decision to save parsed calendar events to objects and properties to variables has been made. Instead of storing the calendar files and then accessing them for information, the application keeps parsed Objects inside the Heap (RAM) memory.

After taking into account the large number of .ics files and even more so, the events described in each, it is concluded that the quickest and least memory consuming parsing algorithm is of most importance to the overall service speed. All the calendar files are parsed on server launch as well as updated every 24 hours, so the algorithm has to work in the background without disrupting users from using the website. Here is where choosing the correct Java version shapes the overall software architecture. Java 8 release offers developers a new way of extending Java Applications with a new development style called functional programming, which is in summary:

- Based on mathematical computation models
- Uses functions as the main data structure
- Designed for performing a high number of operations on data that matches a specific pattern.

¹Parsing - Analyzing or searching a text for logical syntactic components

Design-wise, functional programming is exactly the feature needed for the thesis in order to achieve fast sorting of data. Its application during the scope of this thesis is explained in detail in Chapter [4.9.3](#).

By using Java 8, our sorting algorithm becomes faster and more efficient, but greater attention has been paid to the dependencies and tools used, to ensure all of them are compatible with the latest Java version.

3.3.3. Web Server

In simple terms, the server is a software tool that provides services to other software applications. For the scope of this thesis, only a web server is required. A web server is a tool used to serve HTML web pages or files to users by using HTTP communication. All the websites hosted on the internet, reside on a server and this web application is no exception.

Fortunately, there is a large variety of servers available for Java. The `.war` web archive can be deployed on any Server that supports Java 8. For the scope of this thesis the Apache Tomcat 9 server has been used in order to test the application locally. Tomcat handles web configurations, such as the ports used, the overall memory and thread load allocation or the website icon, among many others and is Java based, meaning it shares the same Java Runtime Environment(JRE) resources as the web application hosted.

3.3.4. reCAPTCHA API

To prevent flooding the server with email send requests the service has to detect and block bots, scripts and any abusive agents. In order to achieve that Google's reCAPTCHA API is used to verify if the user is human.

reCAPTCHA is one of the most reliable and easy to use Captcha currently available. Google's API checks the user behavior prior to clicking, whether the cursor moves in a flow motion (or jumps from point to point), the browser fingerprint, web cookies and location history tied to a user specific fingerprint account. All these steps and information are done and stored remotely by Google while the web application receives only a boolean value which is used to validate the email input. It is difficult to fool a continuously learning pattern detection engine, and in cases where the analysis engine is not sure if an user is human or an abusive agent, it still requires the user to match a String Captcha.

One drawback, however is the fact that the Captcha needs to be registered with Google, and the website owner need to specify the domains it will be used for. After doing so a private

and public keys are provided and need to be set as parameters in the web application. More details on how to do so are found in Attachment [A](#).

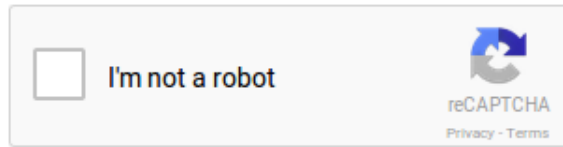


Figure 3.17.: Captcha Validation

4. Realisation

In order to expand on the design decisions taken, this chapter details the actual coding and implementation process, as well as providing examples, snippets and reasoning for the methods used. This chapter will also elaborate on challenges and issues faced.

4.1. Building the Project

As previously described, Maven is a build and dependency management engine. Before proceeding with development, configuring how the project will be built, which version of Java be used and what dependencies need to be downloaded needs to be done.

Below is the XML configuration of the build process:

```
1 <modelVersion>4.0.0</modelVersion>
2 <groupId>com.haw.ie</groupId>
3 <artifactId>calendar-files</artifactId>
4 <packaging>war</packaging>
5 <version>1.0.0</version>
6 <properties>
7   <maven.compiler.source>1.8</maven.compiler.source>
8   <maven.compiler.target>1.8</maven.compiler.target>
9 </properties>
```

Listing 4.1: Maven Build Configuration

As well as, a description of relevant lines and their role in the application build:

- 1) `modelVersion` - The `pom.xml` model version which is always required, however Maven 3 currently has only one model making `4.0.0` default.
- 2-5) `groupId`, `artifactId`, `version`, `packaging` - An artifact, in Maven is the output of a build. Artifacts in Maven are identified by a coordinate system `groupId/artifactId-version.packaging`. The configuration results in the output `calendar-files-1.0.0.war` file being generated containing the entire

project and ready to be deployed. The group id stores the file at `repository/com/haw/ie/` and can be later used to dictate inheritance in case multiple Maven projects are combined. All the mentioned properties are required by the model for a web application.

- 7,8) `maven.compiler.source`, `maven.compiler.target` - Properties specifying the Java versions to be used by the Maven compiler plugin (used to compile the classes). `source` indicates the minimum Java version compatible with the project, while the `target` specifies the Java version desired to use for compilation (i.e Java 6 can compile Java 5 sources), however keeping them both the same ensures a higher level of stability, due to the fact that Java 1.9 is not released yet and there is no telling how the software will behave then.

The compiled sources and artifacts can be found in the `./target/` folder.

4.2. Managing Dependencies

Below is a list of all the top level dependencies managed by Maven, some of these have already been described while others are expanded upon in the following chapters. All the dependencies used for this thesis are downloaded from the default Maven repository, which, at the time of writing this thesis contains over 5.62 Million artifacts.[5]

- 1) `cdi-api (v1.2)` - Context and Dependency Injection API
- 2) `jackson-databind (v2.8.3)` - JSON File Parser
- 3) `jsoup (v1.9.2)` - HTML Web Page Parser
- 4) `javax.servlet-api (v3.1.0)` - The Servlet API
- 5) `jsf-api & jsf-impl (v2.2.13)` - Java Server Faces API & Implementation
- 6) `junit (v4.12)` - Framework used for unit testing
- 7) `primefaces (v6.0)` - Library of components for JSF
- 8) `javax.mail (v1.4.2)` - Framework for Email Communication

In order to add a dependency to the project its artifact should be added to the `pom.xml` in the following way.

```
1 <dependency>
2   <groupId>com.sun.faces</groupId>
3   <artifactId>jsf-api</artifactId>
4   <version>2.2.13</version>
5 </dependency>
```

Listing 4.2: Maven JSF Dependency

4.3. Testing

In computer programming, software testing is necessary to avoid mistakes, the impact of which can range from irrelevant to dangerous. Multiple ways of testing software exist, however out of those which are able to be performed by the developer, Unit testing is one of the most effective and widely used in the industry. Unit testing is the automated process in which the smallest testable parts of an application (variable manipulations, procedures, functions) are individually and independently tested for proper operation.

Unit testing is one of the maintainability criteria discussed in Analysis, and is executed using a testing framework named JUnit.

JUnit, unlike the other `.jar` libraries imported, is only used at compile-time and does not use resources or slow down the service during runtime. A JUnit test is a Java object, designed to test other Java objects and is located in the `src/test/..` folder, separate from the functional source-codes.

In order to create a new Unit test a function has to be marked with the `@Test` annotation. All functions annotated will be executed by Maven before compiling the source codes and have a dedicated visual interface in most Java IDEs. It is also possible, if required to configure methods that are executed before or after the function or class, which help in pre-loading resources that are expected to be present during runtime, like configuration settings or web repository files. In order to achieve that, the annotations `@Before`, `@After`, `@BeforeClass`, `@AfterClass` need to be placed before the method declaration.

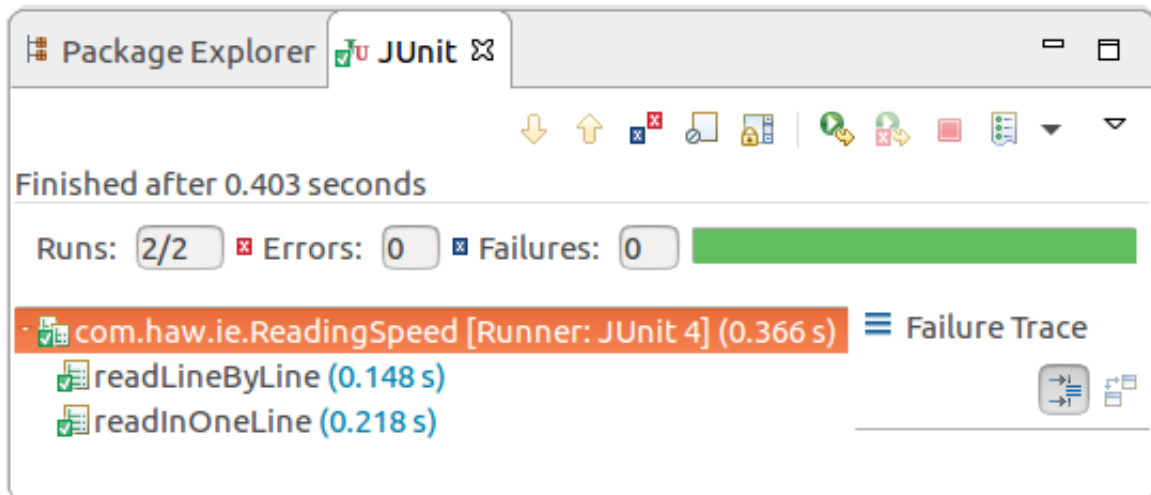


Figure 4.1.: JUnit Testing Interface

Thanks to all the tools offered by JUnit, creating unit tests is made simple, the complicated part is developing as many scenarios as possible.

Unit tests offer the following improvements to an application:

- **Simplified Integration** - The process in which all the individual modules of a software are combined and tested as a group is called Integration testing. It is done by combining multiple unit tests together and helps diagnose problems on a larger scale, for example – incompatibility. Integration testing is usually done manually at the end of a development cycle.
- **Early Problem Detection** - Due to the fact that unit tests are created in parallel with the functional source codes (a methodology called Test-driven development), unit tests, by design help locate problems early and allow tracing the location of an error with ease, as well as offering a timer that can potentially detect poorly-written code. Additionally, during the process of developing the tests, a developer is forced to think through inputs, outputs, behavior and error conditions which helps in clearly defining a unit's behavior. The severity of finding bugs and errors during development is much lower than having to treat them later.
- **Tolerance to Changes** - The application becomes more tolerant, and even invites changes; for example refactoring the code at a later date, with the goal of optimizing or improving the execution speed will already have a benchmark in place to check if the modules still work correctly and whether or not, the improvements are successful.

Testing also adds a new programming concept for developers called assertions. An assertions is a boolean expression that has to be true, unless there is a bug or error in the program. Or in simpler words, a conditions that has to be met. Only false assertions are flagged by JUnit.

There are multiple types of assertions available, for instance, checking for equality is done by invoking `assertEquals(Expression 1, Expression 2)` or verifying if an object is null `assertNull(Object)`. Array data structures are also supported thanks to `assertArrayEquals(arrayExpected, arrayResult)`.

Assertions are the building blocks of JUnit tests and are generally present in every test function.

Although unit tests are present in this bachelor thesis, their meaning and functionality will not be analyzed. Tests do not contribute to the web application functionally and going into detail on them will only create confusion.

4.4. Logging

One of the maintainability criteria discussed previously is creating a log of relevant information. Java 5 added logging support to the EE package in the form of `java.util.logging` (JUL). The way information is presented to the developer can be configured, by specifying if the output should be redirected to a file, console or even emailed.

Logging in Java is divided by gravity of occurrence, during this thesis there are only three levels used:

- INFO - Relevant information regarding the activity of the service.
- WARNING - A problem not crucial to the integrity of the resource.
- SEVERE - An issue is impairing the functionality of the service.

Currently there are two severe exceptions, the inability to fetch the calendar files, and inability to parse the JSON configuration file.

An alternative to the default logger is Log4j, a logging tool developed by Apache since November 1999, which is conceptually and functionally identical to JUL. In order to determine if there is any advantage in using Log4j, both loggers have been tested for speed in the `LoggingSpeed.class` test, and the results of writing 100,000 messages to the stdout console with the following results:

log4j	java.util.logging
10.042 s	6.875 s
11.231 s	7.150 s
10.439 s	6.747 s

Table 4.1.: Speed Comparison of Logging platforms

After running the tests, it has been concluded that the standard Java logging utility outperformed Log4j in terms of speed and has all the required features like selecting the logging output type and division of events by severity. Which removes the need of configuring an extra dependency in Maven. In Java, a logger is defined and called in the following way:

```
private static final Logger logger =
    Logger.getLogger(CalendarEvent.class.getName());
logger.warning("Unable to find internet connection");
```

Listing 4.3: Logger Usage

Notice that the logger is static and final for each class, because it is impractical to have multiple loggers created pointing to the same location.

4.5. Web Parser

In order to access the web repository[10], Java has to connect to the internet. Unfortunately, the Java EE 8 package does not have reliable web parsing functionality, even though it does contain the Java Swing package – used to build user interfaces. Swing offers the ability to build a web browser, which can be used for accessing the page, but naturally it is not enough. Even though it has been attempted, Swing is unfortunately, not designed for parsing web pages. It is slow and extremely resource intensive, the code is also bloated with multiple foreign objects referencing each other, which make it complicated and hard to understand.

The thesis requires another framework, able to access the web pages in a reliable fashion, and after multiple trials the "jsoup: Java HTML Parser" library has been implemented as a dependency. Jsoup provides an API parsing the HTML DOM¹, it allows extracting data from a web page or file. Upon a closer look at the web repository holding the calendar files, it can be seen that the links lead to separate web pages containing the file. Jsoup is able to directly access files, however we cannot benefit from it as HTML links to a file look this way ``, while the web repository has elements that reference other url locations ``. Jsoup is not able to find any files in the repository, but rather sees a list of web addresses for each calendar file.

```
Document doc = Jsoup.connect (
    "http://www.etch.haw-hamburg.de/Stundenplan/ICS/").get ();
Elements Links = doc.select ("a[href*=.ics]");
```

Listing 4.4: Jsoup element selection

The snippet above shows how the repository is accessed in the web application. First Jsoup fetches the web page as a document object. Next, using the framework's selector-syntax[13], all elements that are web links and contain `.ics` are added to a list (defined by the data structure `Elements.class`). The selector is a feature borrowed from other programming languages (CSS, jQuery), which are more suited towards web development than Java. This is the only place in the application where a selector is applied, for this reason it will not be expanded on general usage of selectors on various elements, however understand what exactly the snippet `a[href*=.ics]` means is important. The `a` in the selector is called "tagname", it searches for all the tags named "a", `[href]` is an attribute, and specifies a HTML attribute belonging to the previously found element. The `*=.ics` operator means "contains the value .ics".

¹Document Object Model(DOM) - A world-wide-web standard for accessing web documents, separated into the core, xml and html dom elements that combined make a webpage.

4.6. Java Mail

One of the design decisions mentioned previously, is the ability of sending a file through email. Java EE does have a dedicated library intended for email messaging called – Java-Mail. JavaMail is an API that "provides a platform-independent and protocol-independent framework to build mail and messaging applications"[20].

The first step before sending an email, is configuring the connection information, the server url, encryption method, ports, username and password as well as the emailing protocol used. Multiple mailing protocols exists, for example POP3 or IMAP that vary in terms of complexity and features offered. Because the web application does not access any emails, but rather sends them, using a complex protocol makes little sense, for this reason, Simple Mail Access Protocol (SMAP) has been selected as the technology used for sending emails. SMAP is designed to fall back on IMAP (a more complex protocol) without requiring extra configuration, in case it is not supported by the mailing server.

The authentication before sending a Java email is used to determine the sender and avoid getting the message flagged as junk. In order to configure the SMAP connection to the HAW mailer, the `.pdf` document[9] from the university set of resources has been used.

Additionally, by deciding to send an email with a calendar file attachment, the web application is forced to adapt the Multipurpose Internet Mail Extensions Object(MIME). A MIME email is different from a traditional email in the aspect that it has to be built from multiple parts. MIME has also been designed specifically for the SMAP protocol, offering better compatibility. Below is the simplified pseudo-code describing the way JavaMail sends a message:

```
MimeMessage email = new MimeMessage(session);
BodyPart text = new MimeBodyPart();
// set the text content //
BodyPart attachment = new MimeBodyPart();
// set the attachment content //
Multipart combinedParts = new MimeMultipart();
// add both text and attachment to the Multipart object //
email.setContent(combinedParts);
Transport.send(email);
```

Listing 4.5: JavaMail Send-Email Pseudocode

In the last line, Transport is an abstract service used to sends the email and does not need to be instantiated. Naturally, the function is too big to describe without UML, however the general idea of building parts and combining them should be clear.

4.7. JSON Parsing

JSON (JavaScript Object Notation) is a data-interchange format, that is considered easy to write and read by a human. This is also true if one considers that a JSON file has a similar object, property structure as Java objects; as well as consists of name and value pairs similar to Java variables.

The best way to understand a json object is to visualize the code as a hierarchy, for example, the configuration used for the web application (Annex B), has the following hierarchy:

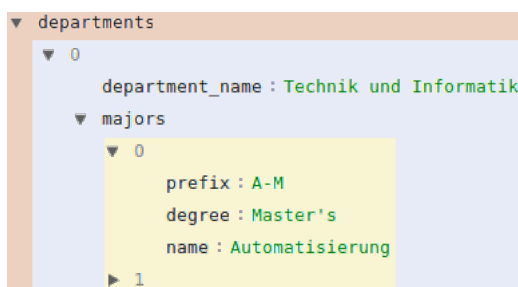


Figure 4.2.: JSON Object Hierarchy

- Yellow - The `Major` objects, containing 3 properties, prefix, degree and name.
- Blue - The `Department` objects, containing 2 properties, `department_name` and an array of `Major` objects.
- Brown - The unnamed JSON Array, contains one instance of `Department`.

The Jackson-Json dependency provides simple JSON to Java Object conversion, however classes holding the objects described have to be declared before parsing the file. JSON objects, can be instantiated by Jackson with the respective values from the file and in order to achieve that two annotations are provided `@JsonCreator` and `@JsonProperty`. The first annotation marks the class constructor used to instantiate objects, while the second marks variables to be set according to the JSON properties.

Below is an example constructor for the `Major.class`.

```
@JsonCreator
public Major(@JsonProperty("prefix") String prefix,
@JsonProperty("degree") String degree, @JsonProperty("name")
String name){
    this.prefix = prefix;
    this.degree = degree;
    this.name = name;
}
```

Listing 4.6: Json Object Constructor Declaration

After creating the necessary Java objects, with the correct annotations and hierarchy. Jackson maps the file with the `ObjectMapper().readValue(..)` function.

4.8. Regular Expressions

A regular expression (regex) is a concept from linguistic theories, it is used to search for a pattern in a sequence of characters, strings or text. The search pattern is able to match anything starting from one character, to complex expression and may match multiple times, or none at all for a given string. The process of analyzing or modifying a string with a regex is called: applying the regular expression(regex) to the text.

In order to apply a regular expression to a string, in Java, the following algorithm has to be applied:

1. Generating a Pattern Java Object – The pattern is applied from left to right and can not match a character twice, for example the regex `aba` will only match the text `ababa` once (`aba__`). The pattern represents a combination of characters and metacharacters², for example `[a-z]` matches all the letters from a to z. In Java, a Pattern is an object, and has to be compiled before being usable. Compiling is done by calling `Pattern.compile("[a-z]")` from the Pattern class and creates a set of properties describing it, like the encoding and number of whitespace characters for faster filtering. Since this service uses multiple patterns, the decision of combining them together into an enumeration data structure has been made, this approach is the most efficient, as compiling patterns takes time, resources and is generally, best done only once per pattern.

Name	Pattern
EVENT	<code>\QBEGIN:VEVENT \E (.*) \QEND:VEVENT \E</code>
SUMMARY	<code>\QSUMMARY:\E (.*)</code>
LOCATION	<code>\QLOCATION:\E (.*)</code>
STAND	<code>\QStand \E (.*)</code>
UID	<code>\QUID:\E (.*)</code>
STARTTZID	<code>\QDTSTART;TZID=\E (.*) :</code>
ENDTZID	<code>\QDTEND;TZID=\E (.*) :</code>
STARTTIME	<code>\QDTSTART;TZID=\E .* : (.*) \QDT\E</code>
ENDTIME	<code>\QDTEND;TZID=\E .* : (.*)</code>

Table 4.2.: Regular Expressions Patterns

²Metacharacters - characters with special meaning within a software application.

2. Creating a `Matcher.class` Object – The `Matcher` instance can be extracted from compiled patterns by running the `Pattern.matcher(text)` function. `Matcher` is an engine, it is tasked with performing match operations on text by interpreting a compiled `Pattern`.

3. Applying the Regular Expression – In order to apply the pattern to a text, the `Matcher.find()` function is used. What `find()` does is return a boolean value depending if the text matched the `Pattern` or not, and if true, generates a group of `Strings` that can be accessed with the `String group(index)` function. In cases where a `Pattern` is matched by the text at least once, the group function will return a result in the following manner:

Index 0 : All of the matches combined

Index 1 : The first occurrence of the `Pattern`

Index 2 : The second occurrence of the `Pattern` ...

Repeatedly calling the `find()` function will shift the index of `group()` by one.

The following example is a combination of the three steps above. It applies the regex to a calendar `VEVENT` component attempting to get the summary. The reason this part is explained in greater detail is due to the fact that this snippet of code (but with the pattern being an enumeration object) is called for every event in each calendar file and for every pattern described in Table 4.2.

```
Matcher matcher =  
Pattern.compile("\\QSUMMARY:\\E(.*) ").matcher(raw_text);  
summary = (matcher.find()) ? matcher.group(1) : null;
```

Listing 4.7: Applying a Regex in Java

The large number of repetitions of the same instruction, requires a performance test in to decide whether or not regular expressions are a good fit for the application.

A benchmark JUnit test has been created for this purpose, and after separating the time it takes to connect to the internet and fetch the files, the test has been instructed to apply the `EVENT` pattern to all the files.

The time it took to extract all the `VEVENT` components from calendar files averaged at 0.184 seconds between 5 runs, leading to a conclusion that the used regular expressions are a good fit.

4.9. Java Source Codes

The following chapter focuses on describing the Java source codes used by the application. For better understanding, UML diagrams are used.

The Java side of the application has been divided into packages according to the functionality of the classes inside. The `com.haw.ie.model` package is used to define classes describing Java Objects or Beans, while `com.haw.ie.util` is reserved for utility classes and services.

Inside the `com.haw.ie` package the only class is the Controller element of the MVC architecture. It has been placed higher in hierarchy due to the fact that it combines everything from `util` and `model`.

4.9.1. The Model Package

The `com.haw.ie.model` package consists of 2 Enumeration classes and 4 Object defining classes.

Patterns.java

Patterns is the class containing the enumeration of compiled Patterns, described in Table 4.2. Regular expression Patterns that have been compiled, can be reused multiple times to match texts, therefore compiling a Pattern should be done only once in the application. Being an enumeration, the class has only a constructor and getter functions.

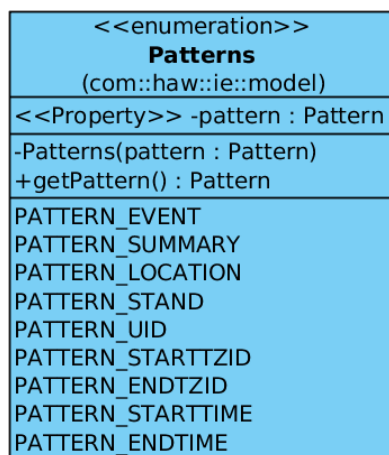


Figure 4.3.: Pattern Class Diagram

Below is the breakdown of the Java regex pattern for the element `PATTERN_EVENT` – `\QBEGIN:VEVENT \E (.*)\QEND:VEVENT \E :`

The first metacharacter `\Q` describes the beginning of a direct quote in the regular expression. The ending of a quote is denoted by the `\E` metacharacter. This means the `\QBEGIN:VEVENT \E` part of the Pattern can be verbally interpreted as "look for exactly these characters BEGIN:VEVENT<space>". The <space> in a pattern is interpreted as both, space or new line.

Following after the quote, is the `(.*)` part of `PATTERN_EVENT`, where `()` denotes a group of text, `.` means "any character", `*` is interpreted as "zero or more times" and the final metacharacter `?` makes the previous `*` quantifier – reluctant. A reluctant quantifier means the `*?` is interpreted as "find the smallest in terms of length" and also makes the regex stop at the first match. When combining all the parts together, the pattern so far can be verbally interpreted as "search for text beginning with BEGIN:VEVENT<space>, followed by a group of any number of any characters (but prioritize as few as possible / but stop at the first match)". The last part of the pattern is a direct quote of the iCalendar event end tag `END:VEVENT` described in similar fashion to the first quote. The reason for limiting the result group to the first match, will be made clear further.

All the regular expressions in the enumeration use the previously described metacharacters in a similar fashion, but compared to `PATTERN_EVENT` look for different quotes and usually stop at a <space> or `:` character.

DateFormat.java

Similar to `Patterns.java` is the `DateFormat.java` enumeration. `DateFormat` is used to declare the format of a date and can be used in two ways, parsing a string in search of a pattern or converting an existing Date Object to a defined String format. There are a total of 5 elements in the enumeration as well as a constructor and getter method.

Next, is an example of the enumeration element `DATEFORMAT_INPUT`:

```
DATEFORMAT_INPUT(new SimpleDateFormat("yyyyMMdd'T'HHmmss"
, Locale.UK))
```

`DATEFORMAT_INPUT` is used to parse the information extracted from a calendar file and corresponds to the iCalendar format found in the `.ics` files.

The metacharacters of the DateFormat Pattern mean the following:

- `YYYY` – Year in 4 numbers.
- `MM` – Month in 2 numbers.
- `dd` – Day in 2 numbers.
- `'T'` – Direct quote T character.
- `HH` – Hour, 2 numbers, 24-hour format.
- `mm` – Minutes, 2 numbers.
- `ss` – Seconds, 2 numbers.

All the other enumeration elements use the same metacharacters and are used to format existing dates to String representations for the View.

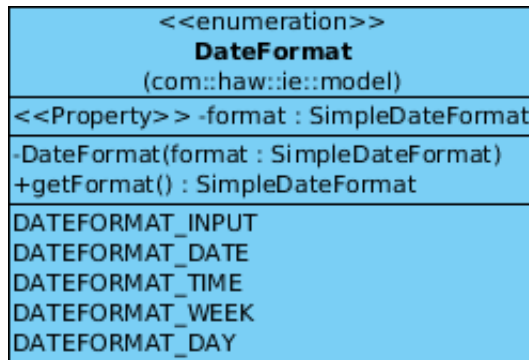


Figure 4.4.: DateFormat Class Diagram

Major.java

A class describing an object of type Major. All the objects of this class are instantiated by the Jackson JSON library from the `configuration.json` file. The class constructor code and functionality has already been detailed in Chapter 4.7.

Department.java

The class describing an object of type Department. Similar to the Major objects, this class is instantiated by the Jackson JSON library from the `configuration.json` file. The class contains the `department_name` and a List of Major objects as variables, exactly matching the JSON file.

Below is the class diagram of both Department and Major combined, notice the one to many relationship caused by the List of Major variable from the Department class.

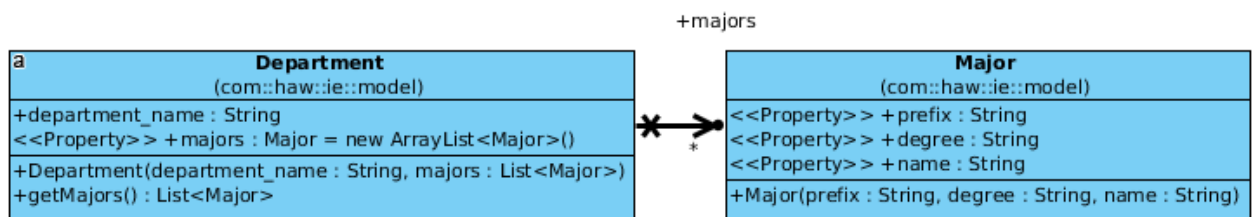


Figure 4.5.: Department and Major class Diagrams

Event.java

The class defining an object of type event. Event Objects are representations of the iCalendar `VEVENT` components. The class has all the properties encountered in the `.ics` files, summary, location and time information which is extracted by applying the regular expressions defined in the `Patterns.java` enumeration.

The information extraction is done in the constructor function.

`public Event(String raw_text)`, where the input `raw_text` is the `VEVENT` text passed as a parameter. The constructor can also possibly throw a `ParseException`, in case the input date is not corresponding to the iCalendar file standard. The calendar files are expected to have their date in the `"yyyyMMdd'T'HHmmss"` format. Aside from logging the exception as a warning, no action is performed to treat the exception and is up to the person responsible for the creation of the `.ics` files to adapt the correct standard.

The Event class also uses all elements from the `DateFormat.java` enumeration. The functions `String getFormattedDate()`, `String getFormattedTime()`, `String getWeek()` and `String getDay()` all return Date Objects formatted by an element from the enumeration. Event elements are also referenced in the View via EL expressions and populate the table component hidden inside each `accordionPanel`.

Additionally, the class overrides the default `java.Object toString()` method. The method is called by the function generating calendar files from the Controller class, and returns the String representation of a calendar EVENT.

It might seem there is no need for a `toString()` method, considering the property `raw_text`. However, one issue of regular expressions is that the patterns do not differentiate between a space character, a tab or a newline, and as a result the `raw_text` parsed is a string represented in one line.

One of the constraints discussed in the Analysis (Chapter 2.2) is that `.ics` files do not recognize lines longer than 75 octets (characters). Outputting `raw_text` in its one-line-form results in the file being unreadable by existing software. The `toString()` method is designed to split the information inside the Object line by line and return it in a compatible form.

Course.java

Course is the Java Object representation of an iCalendar file. It is a class consisting of 3 variables and a constructor. The variables represent the iCalendar file name, URL(full web address) and a List of `Event.class` elements described previously.

The course constructor `public Course(String name, String absUrl)` is a method that receives 2 parameters, the file name and its location on the internet. The constructor, trims the file name removing the `.ics` extension and stores the result in the name property. For example, the file `IE1-EE1.ics` is translated to a Course Object named `IE1-EE1`.

The constructor will also throw a `IOException` if connecting to the url is not successful. Once again from the developer side of things, no action except logging is performed because the service is expected to have internet connection to function.

When a course is constructed, Jsoup is used to fetch the content of each calendar file from the repository. The `PATTERN_EVENT` regular expression is then applied to the text and all the iCalendar `VEVENT` components are extracted. Thanks to the Pattern `(.*?)` metacharacter `?` the regular expression stops at the first match and returns the text inside one `VEVENT` component at a time. This content is passed to the `public Event(String raw_text)` constructor and, one by one Event Objects for the Course are created and added to the List.

In programming terms, the `?` metacharacter allows shaping the function in the following manner:

```
while (matcher.find())
    events.add(new Event (matcher.group(1)));
```

Listing 4.8: Creating Event Objects

As mentioned previously, if a previous invocation of `matcher.find()` returns true, it will match the first instance not previously matched. As a result `matcher.group(1)` will have a different event each iteration as long one exists in the file.

Each course, is referenced in the View and represents an `accordionPanel` component. Below is the class Diagram of the iCalendar to Java Object model classes:

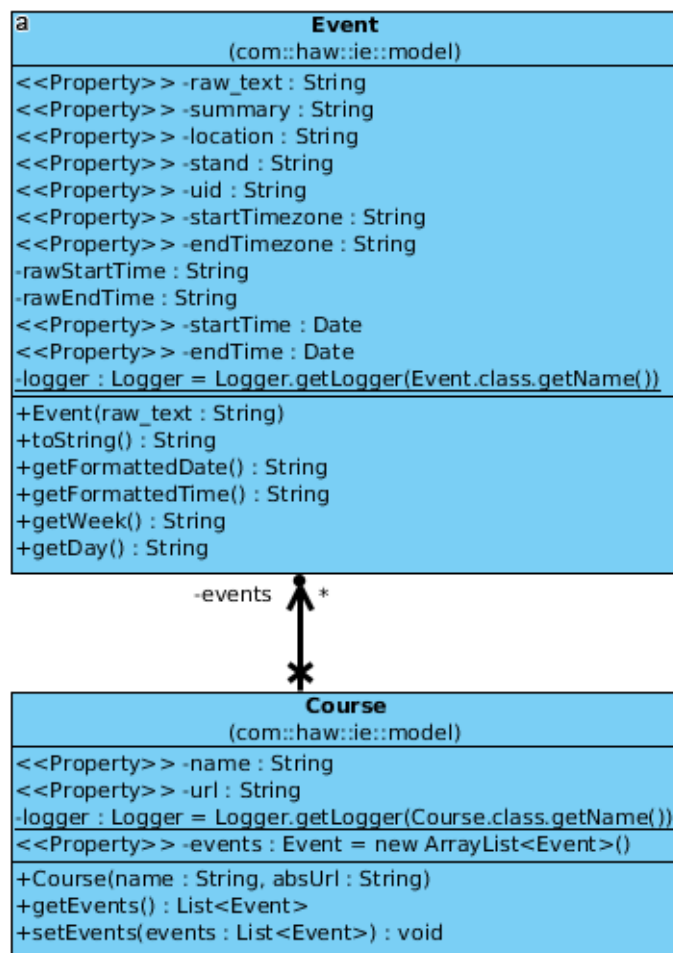


Figure 4.6.: Course and Event Class Diagrams

Even though the `com.haw.ie.model` package contains mostly Objects for Data manipulation with no complicated functions, one method can be expanded on if visualized as a sequence diagram.

Event.toString()

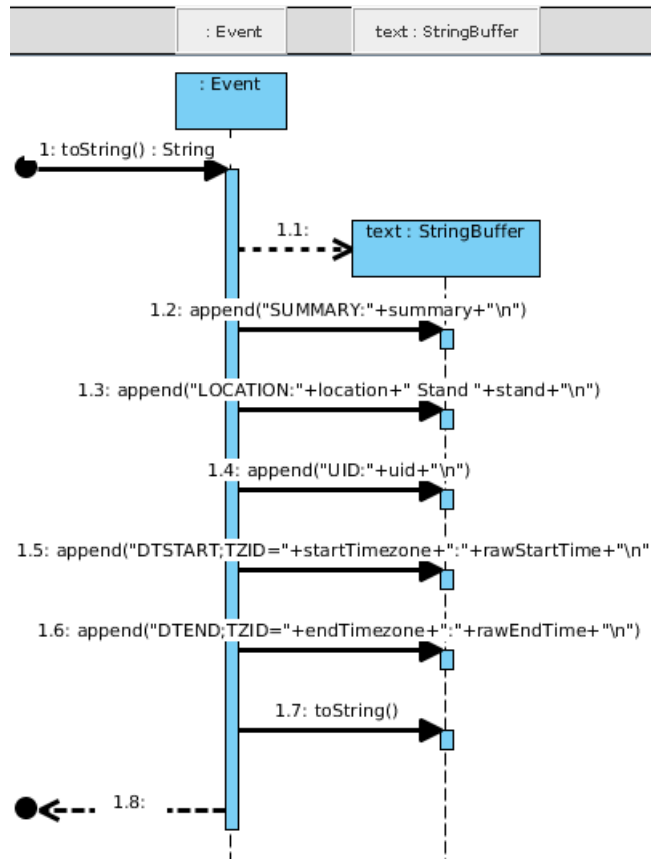


Figure 4.7.: Event.toString() Sequence Diagram

The function is building the output using a Java Standard Edition object `StringBuffer`. `String Buffer` is similar in functionality to the more popular `StringBuilder` however, compared to the builder the `StringBuffer` Object is thread-safe. Information from the Object variables is appended to the `StringBuffer`, which is surrounded by the respective iCalendar format tags. Additionally, the `rawStartTime` and `rawEndTime` properties (Strings defined in the constructor) are used in order to avoid formatting the Date multiple times, convert it from String to Date and back into a String.

4.9.2. The Util Package

The `com.haw.ie.util` package consists of 2 Providers, 2 Services and one JSF Converter class. The package covers the utility functionality inside the application.

4.9.3. The Singleton Concept

A singleton, is a class which can only be instantiated once and provides easy access to the said instance.

Singleton classes are to be taken with a grain of salt, as there is large noise among developers around their usage, firstly a singleton is viewed as a global variable, which has always been considered bad programming practice, secondly it goes against Java's Object Oriented Programming design. Therefore, singleton class only considered worth using if multiple instances of the said class might harm the application resources, speed or productivity in any way.

Several ways of declaring a singleton exist in Java, for the scope of this thesis, it has been decided to follow the thread-safe singleton approach as a web application, accessed by multiple users opens new threads often.

1. The singleton class has to be marked with the `final` keyword, a final class means it can not be subclassed (extending it is not allowed).
2. The singleton class has to contain a static variable holding an instance of its type and provide a getter method to access it, but no setter. A static variable is shared by all the instances of the class (in this case hopefully one).
3. Below is the getter method for a thread-safe singleton:

```
private static ConfigurationProvider instance;
public static ConfigurationProvider getInstance() {
    if(instance == null){
        synchronized (ConfigurationProvider.class) {
            if(instance == null){
                instance = new ConfigurationProvider();
            }
        }
    }
    return instance;
}
```

Listing 4.9: Synchronized Singleton Implementation

The singleton `getInstance()` shows the standard way of defining in Java a software engineering pattern known as "double-checked locking". The design pattern, performs the first check, without locking the object, if an instance exists, it will be returned, however if it does not, the class is locked (by redeclaring the class inside itself with the `synchronized` keyword).

Acquiring the lock, could have also been done in parallel by another thread, thus the instance variable is checked again to determine if the class has been instantiated, if not it can safely be assumed that this thread is the first one to instantiate it and the Object is constructed.

An overly complicated approach to be sure, however it outweighs the risk of having multiple Providers or Services which the web application does not need.

ConfigurationProvider.java

The ConfigurationProvider is the class tasked with reading the `configuration.json` file. The class represents the highest hierarchy JSON Object (the anonymous array colored in brown in Figure 4.7) and contains a List of Departments. The class constructor is the method reading the configuration file and consists of two steps, acquiring the file content as a String and mapping the said String to JSON Objects.

1. Acquiring a file from the disk in Java is a trivial algorithm, however when it comes to web applications, the locations or file hierarchy is hard to predict. Traditional access methods from the disk might not work when the application is deployed, because the server decides where the files are going to be located each deployment. As a result, we have to use a reference point – the `ClassLoader`. The `ClassLoader` bound to the current object is responsible for loading the class, and holds the information about project location. By accessing the `ClassLoader`'s `.getResource("path")` function the URL of the location is returned. Finally the resource URL is converted to URI (which holds both the URL content as well as identification information) and returned to be read as a stream of bytes.

The algorithm described above corresponds to the following code from the web application:

```
String content = newString(Files.readAllBytes(Paths
    .get(getClass().getClassLoader()
        .getResource("configuration.json").toURI())));
```

Listing 4.10: Locating the configuration.json File

After looking at the code, one may wonder, why gather all the calls in such a cumbersome value assignment. The answer is simple, by keeping the calls anonymous, Java does not have to create extra objects that pollute the Heap memory.

2. In order to map the content to the JSON Objects from `com.haw.ie.model`, Jackson-json provides the `ObjectMapper.class`, which given the content and the highest hierarchy class mapped to the file, will populate all the objects using their constructors.

```
instance = new ObjectMapper().readValue(content,
ConfigurationProvider.class);
```

Listing 4.11: Mapping the JSON File to Objects

ICSPProvider.java

The ICSPProvider class is responsible for a set tasks regarding iCalendar Files. It contains 3 variables and 5 functions that are not getters or setters.

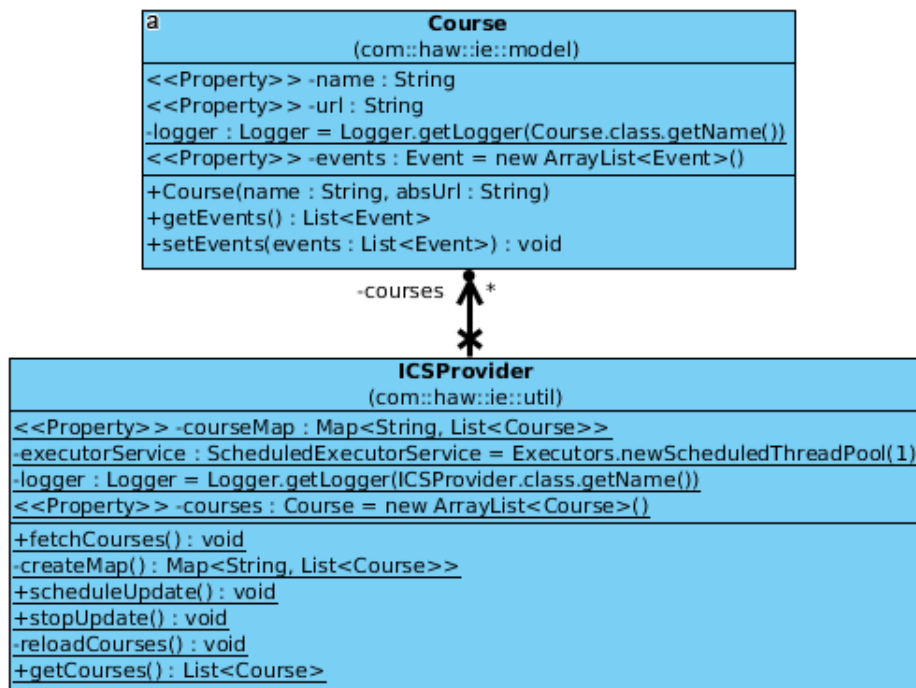


Figure 4.8.: ICS Provider and Course Class Diagrams

ICSPProvider, offers the web application functions for sorting through the web repository and converting hyperlink information to Course Objects, it also offers the ability to filter through the information and only create Courses relevant to students, as there are other calendar files in the repository as well.

Another functionality is grouping the created Courses by Major and inserting them into a HashTable³ data structure, which is used as the thread-safe alternative to a HashMap.

Synchronization is important because the class is also tasked with periodically updating the List of Courses, which should not disrupt the user from browsing the site.

`void fetchCourses()` - The function accessing the repository, sorting through the results and afterwards, creating all the Course Objects and saving them in a List. Once all the Courses have been created the function triggers a call of the `Map<String, List<Course>> CreateMap()` which is described further.

Naturally, in order to access the web, the Jsoup library is used, in this instance a connection Timeout is configured lasting 10 seconds. This helps avoid exception throws when the internet connection is present, but slow. After the document has been accessed and all hyperlink elements have been selected as described previously in Chapter 4.5, the function eliminates files that are not relevant to students – the calendar files for professors and rooms.

In order to achieve this, it checks if the filename (example `IE1-EE1.ics`) contains the dash(-) character, as it is used in all the calendar files to separate major and semester information from the course name. It also makes sure that no elements containing the text "Raum" are selected, because some Room calendar files use dashes in their nomenclature. The filtering is achieved with the default `java.lang.String boolean contains("text");` function. Calendar files that successfully match the criteria are converted into Java Objects and added to the List variable.

The function may throw an `IOException` if it is unable to connect to the internet, the exception causes a severe logger message, as failure at this stage means no Course and Event objects are defined for the application to present for the users. The function will keep attempting to connect to the repository until it is successful, by calling this function recursively on Exception throw.

`fetchCourses()` ends with an assignment of the `courseMap` variable with the help of the `createMap()` function described further.

³HashTable - <Key, Value> Map data structure, that uses hashing functions to compute index for an array of "buckets" that describe the value

`Map<String, List<Course>> createMap()` is tasked with binding `Course` objects to their majors for sorting purposes. `CreateMap` uses functional programming described previously together with another Java 8 feature called Streams. The `Map` data structure, represents a set of `<Key, Value>` pairs. A `Key` is the `prefix` property of the `Major` Objects from the configuration file, while a `value` is a `List` of `Courses` whose names start with the previously mentioned prefix. The key is available, as it already has been mapped from the JSON file, however the `List` of `Courses` needs to be created.

```
hashTable.put(major.getPrefix(), courses.stream()
    .filter(course ->
        course.getName().startsWith(major.getPrefix()))
    .collect(Collectors.toList()));
```

Listing 4.12: Binding Courses to a Map

The `courses.stream()` function, converts the `List` of `Courses` to an abstraction called `Stream`. A `Stream` Object is able to pipeline⁴ at most 4 instructions one after another, `filter`, `sort`, `map`, `collect` in the following form `stream.filter().sort().map().collect()`.

In order to select relevant `Courses` and create a list of objects, `filter` and `collect` are enough. Filtering applies an anonymous function (denoted by the `->` operator) to every element `course` of the now `Streamed` data structure.

The `(course -> course.getName().startsWith(major.getPrefix()))` snippet can be translated verbally to, "for every `course`, check if its name starts with the respective major prefix and return the boolean result", because the `java.lang.String.startsWith()` function returns a boolean value.

The boolean value is then passed to the `filter` pipeline part, and if true, is passed further to the `collect` part, where the `Course` is converted back from a `stream` and added to a `List` using the `java.util.stream.Collectors` utility.

⁴Pipelining - Dividing the execution of instructions into stages.

`void reloadCourses()` is a function that clears the Lists of Courses by instantiating a new empty List in its place and re-runs the `fetchCourses()` function to populate the List with Objects once again. It is called by `scheduleUpdate()` every time it is executed.

`void scheduleUpdate()` The function initiating the calendar file update routine. Inside, an object of type `java.lang.Runnable` is declared, with its `run()` function overwritten to one instruction, calling the `reloadCourses()` function. After which the runnable object is passed to the `executorService` variable of type `java.util.concurrent.ScheduledExecutorService`, a class designed for periodic scheduled events. By default, the Executor Service is configured to run every 24 hours, starting from the moment the web application has been deployed on the server.

`void stopUpdate()` Function designed to shutdown the `executorService` routine that updates iCalendar Files. It is required to avoid memory leaks and frees the dedicated thread, otherwise the thread leak may persist even after the Java Runtime is shut down.

InitializeService.java

A service class extending `ServletContextListener` and overriding the two methods declared by it. When the web application loads Servlet, this class will be scanned and identified as a listener thanks to the annotation `@WebListener`. Servlet listeners are used to detect events in the web application. In this particular instance, the events are the initialization and destruction of the Servlet context. Or in simple terms, the starting and ending points of the web application.

The two functions are `contextInitialized` and `contextDestroyed`, both are called by the Servlet API when the application start and end are initiated. `contextDestroyed` runs only the `stopUpdate()` routine from the `ICSPProvider.class`, which stops the thread resource allocation for the scheduled event.

On application launch, the Provider classes described previously are instantiated. The courses are fetched and the Timezone information is configured to match the university location as well as differentiate from winter and summer time.

Below is the sequence diagram executed on starting the web application.

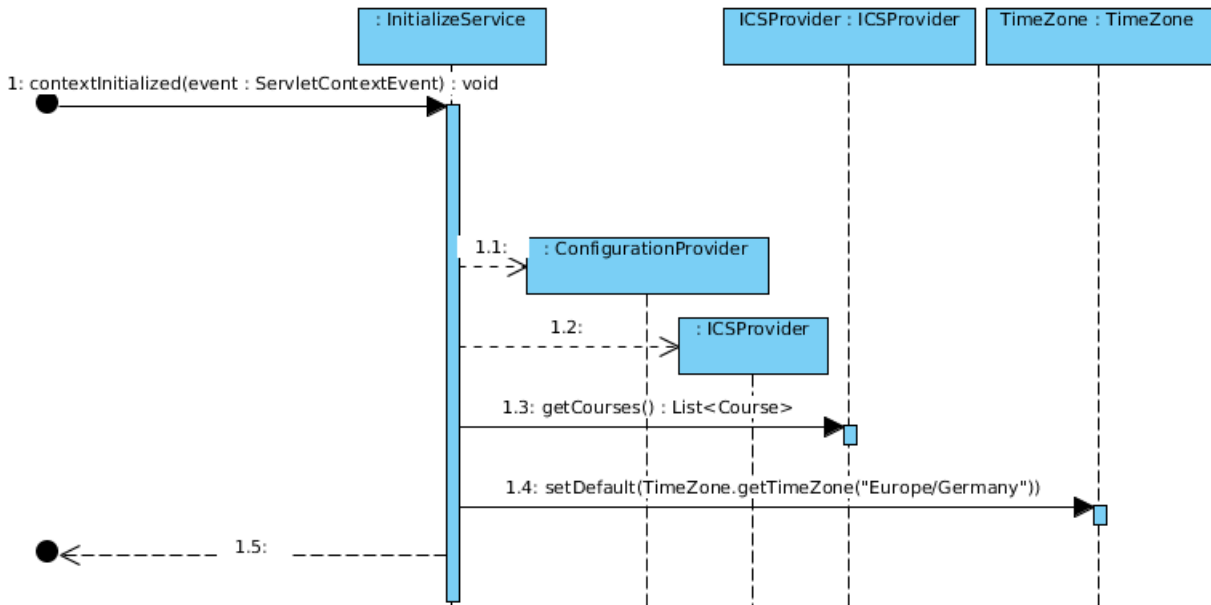


Figure 4.9.: `InitializeService.contextInitialized()` Sequence Diagram

EmailService.java

Email Service is the class used for sending the calendar files via Email. However, it has been designed to also allow regular mailing functionality if required, for example logs can be configured to be sent through email. The class consists of multiple properties to configure the access to the `haw-mailer.haw-hamburg.de` email server over SMAP. The class is a singleton because there is no need for multiple configurations and instances of the class.

Email Service offers the `void sendMail(subject, body, recipient, file)` which uses the parameters to build a MIME Message and send it. The method is too large to be presented in the form of a Sequence diagram or code listing, however, its functionality principle has been covered in Chapter 4.6.

The message is built in a puzzle fashion, properties such as header, sender/recipient address, subject, body text, the message content and an optional attachment are all assembled and sent. If the attachment property in the function call is set to null, the service will not attempt to add an attachment MIME Body Part. Another important setting, that allows the message to be detected as a set of calendar events is the header property `Content-Type`, which has to be set to `text/calendar`.

The function may throw two exceptions caused by, an invalid assignment of `javax.mail` properties which leads to a `MessagingException` or incorrect file content leading to an `IOException`. Naturally, if any exception occurs, the email is not sent and a warning is logged.

EntityConverter.java

Entity Converter is a generic class used by JSF to convert from Objects to String representation and back. The Entity Converter is required to populate JSF components, because sometimes, the default conversion can not interpret which Java Object represents which String in the HTML View.

A converter has to implement the `javax.faces.convert.Converter` class and Override the methods provided by it, namely `getAsString()` and `getAsObject()`.

`getAsString()` converts Java Objects into String representations used by the JSF Renderer⁵ in the View. This, however does not mean that the object is converted to text, but rather a pseudo random universally unique identification 128 bit value that is used to reference objects during the Rendering phase. For example, the JSF `selectOneMenu` uses the converter, the 6 Majors displayed in it are going to be referenced internally in JSF, by their UUID value, even though users will see the regular bean text. When a user selects a field, the converter receives the UUID value of that field and performs the conversion.

The pair of Object and UUID is mapped to a `Hashtable<Object, StringUUID>` variable inside the class, where the Object represents the key and the UUID is the value. Object being the key prevents duplicates and null values.

Every user accessing the website, is rendered a personal View by JSF, by mapping the values to a `Hashtable` and referencing them whenever required, the converter functions more efficiently, as the 6 Majors used in the `selectOneMenu` that have been converted once, can be reused by thousands of users.

⁵Render a JSF Page - Analyze and convert the `.xhtml` document with EL expressions to a plain HTML representation.

Below is the Sequence Diagram of the `getAsString()` converter function described previously:

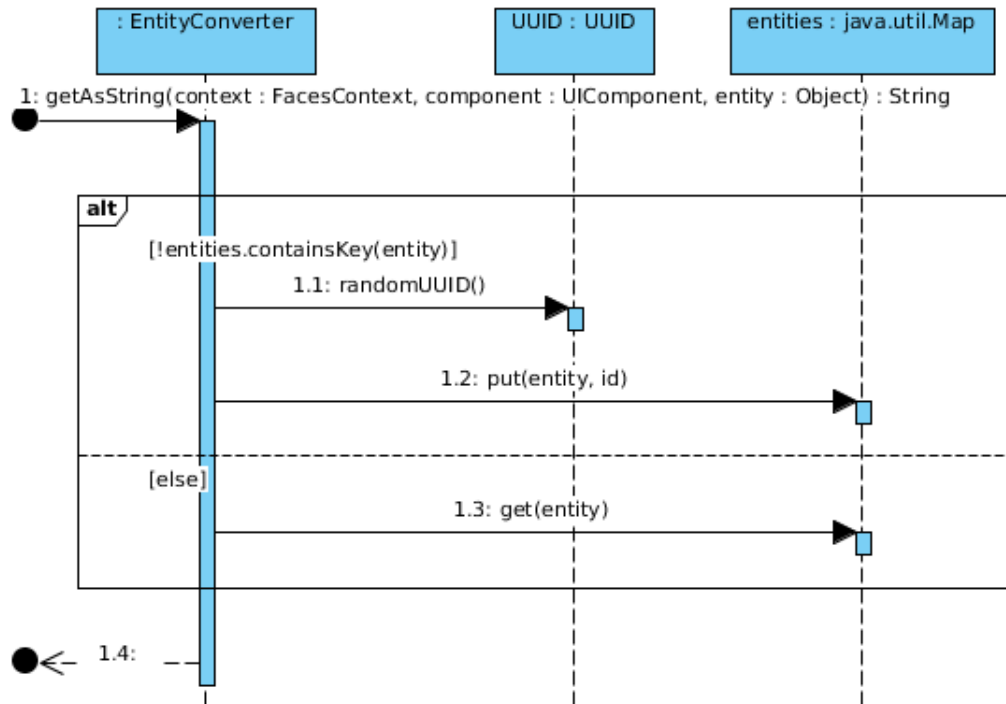


Figure 4.10.: `EntityConverter.getAsString()` Sequence Diagram

Which in verbal language can be interpreted as "if the Hashtable contains the object, return its UUID ; else generate a UUID and save it with the object before returning."

`getAsObject()` is the second function of the converter. The function receives UUID from JSF during rendering, looks for the `<Object, UUID>` entry (pair) in the Hashtable, and if found returns it ; else returns null because it is impossible to generate an object from an ID if it has not been mapped previously.

There is no standard way of building JSF converters, developing a powerful JSF converter leads to fast and correct View rendering. The approach developed for this application comes with the concern that multiple objects converted, will populate the Hashtable, and eventually slow the resource down as searching the Hashtable for an entry takes longer if the Map is large. This however, is unlikely to happen, as the 6 Major + 412 Course Objects converted and reused are a tiny amount compared to what Hashtable is able to handle.

For the duration of this thesis, the `EntityConverter.class` has been used reliably on a daily basis, without showing any sign of faulty or slow behavior.

4.9.4. The Controller

`CombinedView.java` is the Controller of the application, it controls the combined view of the scroller and calendar UI components, hence the name – `CombinedView`. The class implements `java.io.Serializable` so the server may persist the current state of the class on the storage, in case an user is inactive but returns later.

The controller is the MVC element responsible for the back-end to front-end communication through model manipulations. A JSF controller has to be annotated with `@ManagedBean`, which lets JSF scan and convert to a bean the class together with everything it holds as variables, with the use of bean injections.

For example, the value of the courses variable from the `ICSPProvider.class` is injected into the `combinedView` Java-bean and may be referenced. Everything is done implicitly, without developer interaction, thanks to the clever combination of frameworks and technologies used, even though a Course is not declared as a Java-bean, by referencing (and injecting) Course into one, it becomes a bean.

The controller is also limited in context, by the `@ViewScoped` annotation. `ViewScoped` binds the bean to a View, and its life-cycle. When the view is destroyed – the bean is destroyed. It also allows using `@PostConstruct` annotation on methods and functions inside the class. Functions marked with `@PostConstruct` are executed before the bean is fully initialized.

`CombinedView` consists of 6 functions and 9 properties (excluding the logger and getters and setters). Each function is designed to react to the user interaction with the View and modifies at least one property.

`void init()` Function annotated with `@PostConstruct`. The first method executed when a view is loaded and the backing bean is instantiated.

When a user loads the website, he sees the default major selection "Automatisierung (A-M)" with the courses corresponding to that Major. This is decided by the `init()` method, inside the default values and constructors are defined. The variables `selectedMajor`, `majorCourses` and `scheduleModel` are instantiated with values before the view is rendered in order to not provide the user with an empty web page.

`void processCourse(course)` function called when the `booleanButton` element is clicked. The Course object (offered by the entityConverter) comes from the View as a function parameter. The function checks the list of user selected courses `addedCourses` and if the object exists removes it, else adds it.

`void onProceed()` is the method called when a user clicks on the "Proceed" button or changes from the scroller to calendar UI component. It checks the user selection and, if empty, sends an error message back to the user. If the selection is present however, the class adds all the events selected to the calendar component. The calendar backing bean is a fixed data structure called `scheduleModel` and is designed to be interpreted by the Primefaces calendar element.

`void onMajorChange()` the function called when the `selectOneMenu` changes its value. Clears the user selection and offers a new instance of `majorCourses` to populate the scroller component.

`StreamedContent getFile()` the function generating calendar files based on user selection. Triggered by a click on the download icon from the calendar component. The function output type `StreamedContent` is specific to Primefaces downloads and combines the file data, encoding, filename. The output is created from an `InputStream`, which is saved in the `fileAsStream` variable because it is the valid input of a email attachment, unlike `StreamedContent`.

The picture below depicts the incomplete diagram, describing only the creation of a file:

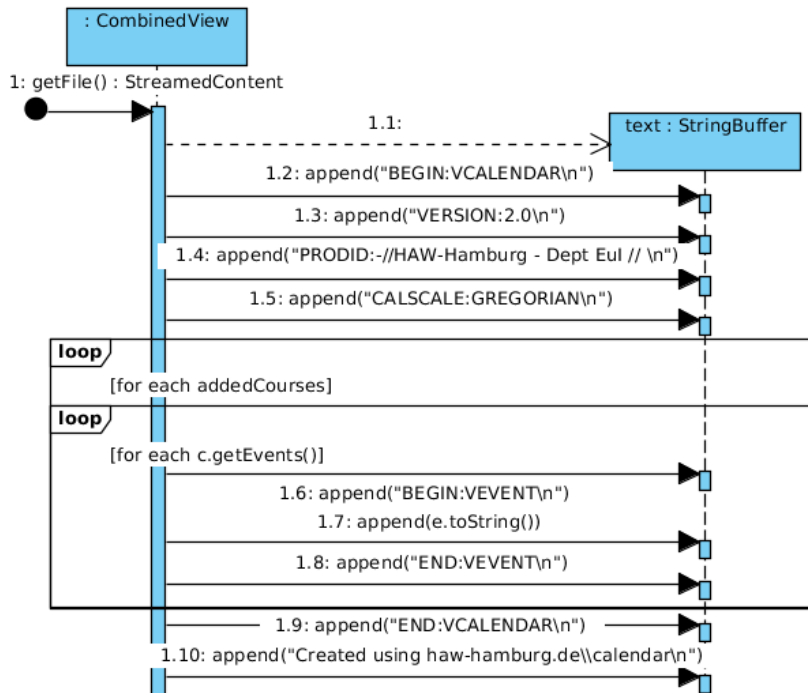


Figure 4.11.: `CombinedView.getFile()` Sequence Diagram

The function may throw an `UnsupportedEncodingException`, although highly unlikely, in case the resulting file contains characters from other languages or unsupported by the UTF-8 encoding.

`void sendFile()` is the function used to send the file as an email attachment. It is called when the user presses the envelope icon inside the calendar component. The function checks whether or not an email address exists and if so, attempts to send an email with the help of the `EmailService.class`. However, unless the user has previously clicked on the download button, a file is not yet generated. The function checks if the user made a selection but does not have a calendar file and executes the `getFile()` method to acquire it. It also sends messages to the JSF `FacesContext` aimed at notifying users if an email is invalid, no Courses have been selected or the email has been successfully sent.

4.9.5. The View

The View is the User Interface element of the application. It is used to represent Model data and contains instructions for the Controller. When compared to the back-end, Maven offers a different hierarchy for the web application front-end. View files are located inside the `src/main/webapp` folder, where all the assets related to web development are stored. When deployed, everything in the folder except the `web.xml` file is accessible by the user, for this reason it is not recommended to keep files other than HTML files and their resources inside the `webapp` folder.

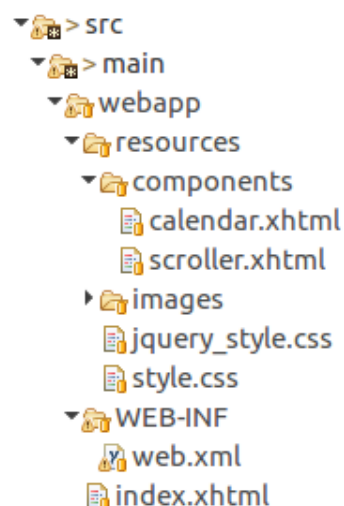


Figure 4.12.: Maven Web Application Hierarchy

An important file located in the `webapp/WEB-INF` folder is `web.xml`. It is used to configure the Servlet behavior, similar to how Maven is configured by the `pom.xml`. Inside the `web.xml`, Servlet properties are assigned values, for example, the property defining the welcome-file (file offered when the website is accessed) is set to refer to `index.html`. Other properties include but are not limited to: mapping the url to `www.domain.com/`, allowing direct access only to `*.xhtml` files and resources referred inside them, removal of the JSF Primefaces default customization theme, and in the case of reCAPTCHA usage, private and public keys bound to the domain.

CSS

Cascading Style Sheets is a style sheet language for HTML elements. It is used to specify the appearance of HTML elements.

Inside the `webapp/resources` folder, the `jquery_style.css` style sheet is defined. The file is used to override the jQuery stylesheet used by the Primefaces components. jQuery offers a theme generation tool for the framework on its website[14]. This style sheet has not been developed by the author of this thesis, but rather generated by the tool according to the author's specifications. All the credits and copyright information are provided inside the file itself. The application also makes use of a custom font named "Open Sans"[15], and a set of icons named "FontAwesome"[8] which are imported in the `index.html` file and credited by referencing.

The web application also has a developed `.css` file used by `.xhtml` files. Properties like elements height, font, text and container wrappings are declared inside.

More detail on style sheets and customization will not be offered by this report, details about styles will also be omitted when analyzing the `.xhtml` files because, similar to Unit tests it does not provide a functional aspect and will confuse the reader.

4.9.6. Index.html

The `index.html` file, consists of 3 parts, the header where a title and brief description of the application is provided, the center content, where either a calendar or a scroller component is displayed and the footer used only for spacing. `Calendar.xhtml` and `scroller.xhtml` are called composite JSF components and are located inside the `webapp/resources/components` folder. A composite JSF component is a `.xhtml` page that is created by the developer, rather than available as part of JSF standard tags. Composite components must reside in the `/resources/` folder to be accessible.

In order to switch between the composite components, a Primefaces `tabView` element is used. A `tabView` is panel used to group content into tabs.

tabView

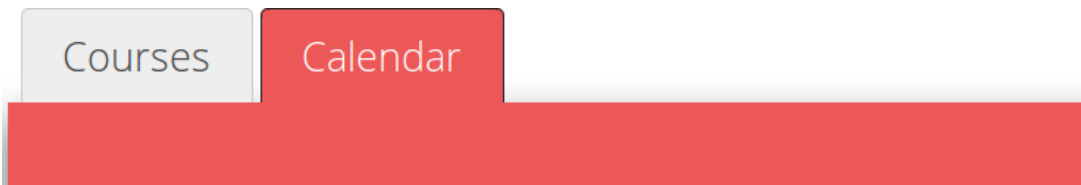


Figure 4.13.: The tabView Component

A `tabView` element separates the flow into two tabs, one intended only for course selection and one used for visualization and downloads. When a user clicks on the "Courses" tab it becomes active and hides the currently used "Calendar" tab. `TabView` has been chosen (instead of the multiple web pages approach) in case any components are developed in the future, as those can be added to the `tabView` in the following manner:

```
1 <p:tab title="Courses">
2   <cc:scroller />
3 </p:tab>
```

Listing 4.13: tab element

As all HTML components, the contents of a tab element are described between its tags. In the example above, the composite component – scroller is added. It can be noticed that JSF tags, use a prefix for identification, `p:` looks for a Primefaces component, while `cc:` searches for a composite component. The declaration of the prefix tags is done inside the `<html>` tag at the beginning of the file and are referred to as namespaces.

Namespace

A namespace is a collection of elements with unique names. The most well known namespace is the default HTML collection, defined in the following way `<html xmlns='http://www.w3.org/1999/xhtml'>` which contains details about the standard HTML elements (for example `<table>`, `<h2>`). All `.xhtml` files require the previously mentioned namespace to be declared inside the `html` tag. In a similar fashion, JSF and Primefaces offer a number of namespaces available.

For this thesis, 4 namespaces are used:

- `h:` – provides the JSF implementation of HTML elements
- `f:` – offers a functional extension for the JSF `h:` elements, for example functions populating the object with a bean value.
- `cc:` – JSF namespace defining the location of the custom components in the project
- `p:` – namespace for the Primefaces components and functions.

AJAX

The Asynchronous JavaScript and XML(AJAX) framework is used and provided by Primefaces. Ajax is able to communicate to the server without reloading the web page or sending a postback⁶ request. Thanks to Ajax, the website can update only relevant parts of the web application instead of having to reload the entire page when sending requests, for example a `booleanButton` value change event or updating the list of courses on changing the major. What Ajax is able to achieve, is communicate with the back-end without disrupting the flow and as a result, the user has a better browsing experience.

Growl

In order to communicate with the user in case of an issue, a Primefaces `growl` component is used. The component provides an overlay for timed messages, which can be received from the Controller class or from inside the view and is displayed in the upper right corner of the screen when triggered.

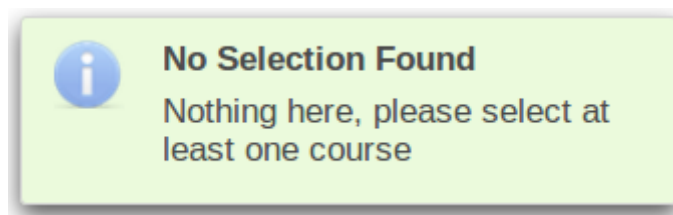


Figure 4.14.: The growl Component

⁶Postback Request - a HTTP POST request to a web page containing a form element. Sending a postback request usually reloads the page to render the HTTP response.

4.9.7. Scroller.xhtml

The `scroller.xhtml` is a composite component used for serving courses to a user in the form of `accordionPanels` and `dataTables`. There are multiple components referenced and configured inside the scroller, for this reason simple elements like text fields or hyperlinks are not described.

selectOneMenu

In order to understand how JSF is written, a code snippet has to be analyzed. Below is definition of JSF `selectOneMenu` component:

```

1 <h:selectOneMenu value="#{combinedView.selectedMajor}"
  converter="entityConverter">
2   <f:selectItems
  value="#{combinedView.departments.get(0).majors}"
3   var="major" itemValue="#{major}"
4   itemLabel="#{major.name} ({major.prefix})" />
5   <p:ajax event="valueChange"
6   listener="#{combinedView.onMajorChange()}"
7   update="...:datatable" />
8 </h:selectOneMenu>

```

Listing 4.14: selectOneMenu element

JSF code looks confusing at first, however it is easy to understand once broken down into smaller parts. Below is the description of source code from the snippet.

- 1) `selectOneMenu` – JSF component, is bound to the variable from the controller class `selectedMajor` and uses the converter to convert from the UUID to an object of type `Major`
- 2-4) `selectItems` – JSF functional component, contains a `List<Major>` as the backing bean and a variable assignment available only inside its tags. Every instance of the list, will be referenced with the variable "major". The `itemValue` property will keep the converter UUID of the major variable which is not visible to the user, the visible part is `itemLabel`, the property displaying the "major" variable as text in the form of `name (prefix)`.
- 5-7) `ajax` – An ajax event detecting the value change. When triggered, calls the `onMajorChange()` function from the Controller class. Updates the table with the newly acquired bean values once the function is executed.

It is crucial to understand the snippet above, since all the other components are also declared in a similar fashion inside the file.

Below is the list of components found inside `scroller.xhtml` together with a short description, organized by hierarchy:

- `blockUI` – Detects long duration ajax calls and shows a loading bar to notify the user.
- `dataTable` – Table containing multiple `accordionPanels`, bound to `majorCourses`.
- > `accordionPanel` – Each element is a `course` variable from `dataTable`.
- > `selectBooleanButton` – Bound by ajax to `processCourse(course)`.
 - » `dataTable` – Table containing rows with event information. Populated from the `course.events` variable. Table rows are bound to a bean each, the week, day, date, time or location of the event.

4.9.8. calendar.xhtml

`calendar.xhtml` is a composite component used for viewing the selection made in the `scroller` component and downloading or emailing the calendar file. There are 3 important elements described in this file.

Schedule

A schedule is a Primefaces component that offers a highly customizable calendar interface. It uses a specially defined `ScheduleModel` data type for its backing bean. Below is the listing for the schedule element.

```
1 <p:schedule value="#{combinedView.scheduleModel}"
  editable="false" draggable="false" resizable="false"
  showWeekNumbers="true" leftHeaderTemplate="prev, next, today"
2 centerHeaderTemplate="title" rightHeaderTemplate="month,
  agendaWeek" />
```

Listing 4.15: schedule element

The element is bound to the bean `scheduleModel` from the controller, it is set un-editable, un-drag-able and un-resizable. The week numbers are configured to show. The schedule component also offers a header configuration via a set of defined templates. The template elements previous, next and current day are placed on the left of the header, the title is centered while the view changing buttons are aligned right.

Download & Email

For visibility, two icons are placed above the schedule component, one for downloading the file locally and the other for sending an email. Each icon is masking the following component respectively:

`fileDownload` – component offering a file download by accessing the `getFile()` function return value.

`dialog` – The pop-up window used when entering the email, contains a form inside with a Primefaces `captcha` component that offers the reCAPTCHA, an email input field bound to the `emailAddress` variable and a button bound to the `sendFile()` controller function. The email field is only considered valid and accepted if the value matches a regex limiting the allowed characters and forcing the `haw-mailer.de` ending.

Each JSF component described and used has a different set of values required as well as customization options. All the information on each component is handily provided in the respective JSF[21] and Primefaces[3] documentations referenced in the bibliography.

4.10. Issues

Context Reloading

Upon launching the web application on the server using a Java Integrated Development Environment(IDE) Eclipse, the Servlet will reload, causing the start Listener to be triggered twice. This causes fetching the calendar files and loading the configuration a second time, effectively doubling the application loading time. The application is fully usable once the context is reloaded.

This happens because the server is designed to detect changes to files, and when deploying via Eclipse, the hidden `.settings` folder is updated information (for exmple, the current time) by the IDE. This triggers the context to detect file changes on launch and it schedules a reload immediately once the initial loading is complete.

The bug is not manifested when deploying the `.war` file directly to the server.

User Notifications

One feature developed but unused in the thesis due to erroneous behavior is the notification system in case a calendar file changes on the repository. The idea consisted of implementing a notification system that would send the users who opt in, an email with the updated calendar file in case any of their courses selected are changed for the duration of the semester.

The database architecture and connection functionality has been developed, Jsoup has also been configured to fetch the update dates for each file. Unfortunately after an extensive period of trial and error, the notification algorithm developed was not functional. Sending the users calendar files, required modifications to the project architecture and addition of several services. Ultimately, due to the tightening deadlines, it has been decided against destabilizing an already functional service. All dependencies, sources and documentation regarding notifications has been scrapped and removed from the thesis.

5. Conclusion

As demonstrated throughout the thesis, creating a secure web service for generating personalized calendar files has produced successful results with all the initial requirements met. The generated web service is fast, safe, reliable and well documented. It is able to provide up-to-date information and generate calendar files that comply with the required standard and are recognized by existing scheduling software. Through a well defined combination of Java native as well as foreign libraries and frameworks, the web application is able to be easily adapted to changes. Thanks to the presence of detailed documentation, multiple tests and well written javadocs the application can be understood, modified and extended without the author's help.

5.1. Summary

From a developer perspective, this thesis provides an insight on the knowledge and level of complexity required to build an effective Java web application. It combines a set of modern tools, technologies and utilities to ensure the developed application benefits from the most recent technological improvements. Every library or framework used by the software has been detailed, often with code examples. The web application also provides information about the inner workings of the software, which are made visible to the service maintainer through the use of logging, additionally, over 400 lines of unit tests have been provided to aid developers attempting to modify the code. In order to make implementing the thesis easier, the web side of the application is offered in form of components that are able to be placed inside already existing web pages by adding only one line to the source code.

From a user perspective, the web service offers a safe, non-restrictive environment used to develop the university timetables in a way unavailable previously. The web site provides a detailed selection of courses to choose from, together with an interface for visualizing the generated schedules sorted by month or week. The application uses a set of messages to guide the user towards successfully generating calendar files and, once achieved offers the option of downloading the files or receiving them by email. The components developed have also been configured to be mobile compatible and can be used from any device.

References

- [1] T. Calendaring and S. Consortium. An introduction to internet calendaring and scheduling. <https://www.calconnect.org/resources/introduction-internet-calendaring>, 2011 (accessed December 6, 2016).
- [2] S. Chodnicki. An introduction to regular expressions. <http://type-exit.org/adventures-with-open-source-bi/2011/05/an-introduction-to-regular-expressions/>, 2011 (accessed December 6, 2016).
- [3] C. Civici. Primefaces user guide. http://www.primefaces.org/docs/guide/primefaces_user_guide_6_0.pdf, 2016 (accessed December 8, 2016).
- [4] B. Desruisseaux. Internet calendaring and scheduling core object specification (icalendar). <https://tools.ietf.org/html/rfc5545>, 2009 (accessed November 14, 2016).
- [5] A. S. Foundation. The central repository search engine. <http://search.maven.org/>, 2011 (accessed December 6, 2016).
- [6] O. Foundation. Top 10 (security risks) 2013. https://www.owasp.org/index.php/Top_10_2013-T10, 2013 (accessed December 6, 2016).
- [7] T. A. S. Foundation. Maven - introduction. <https://maven.apache.org/what-is-maven.html>, 2016 (accessed December 6, 2016).
- [8] D. Gandy. Font awesome, the iconic font and css toolkit. <http://fontawesome.io/>, 2016 (accessed December 8, 2016).
- [9] H. Hamburg. Konfiguration des pop3 oder imap zugriffs auf den haw mailer. https://www.haw-hamburg.de/fileadmin/user_upload/HAW-Mailer/Konfiguration_des_POP3_oder_IMAP_Zugriffs_auf_den_HAW-Mailer.pdf, 2011 (accessed December 6, 2016).
- [10] H. Hamburg. Index of /stundenplan/ics. <http://www.etch.haw-hamburg.de/Stundenplan/ICS/>, 2016 (accessed November 14, 2016).

-
- [11] H. Hamburg. Studienplan studiendepartment informations- und elektrotechnik, fakultät technik und informatik. <http://www.etechnik.haw-hamburg.de/>, 2016 (accessed November 14, 2016).
- [12] J. Hedley. jsoup java html parser. <https://jsoup.org/>, 2016 (accessed December 6, 2016).
- [13] J. Hedley. Use selector-syntax to find elements. <https://jsoup.org/cookbook/extracting-data/selector-syntax>, 2016 (accessed December 6, 2016).
- [14] T. jQuery Foundation. Themeroles | jquery ui. <https://jqueryui.com/themeroller/>, 2016 (accessed December 8, 2016).
- [15] S. Matteson. Open sans - google fonts. <https://fonts.google.com/specimen/Open+Sans>, 2016 (accessed December 8, 2016).
- [16] O. T. Network. Java servlet technology. <http://www.oracle.com/technetwork/java/index-jsp-135475.html>, 2011 (accessed November 14, 2016).
- [17] Oracle. Differences between java ee and java se. <http://docs.oracle.com/javase/6/firstcup/doc/gkhoy.html>, 2012 (accessed November 15, 2016).
- [18] Oracle. Introduction to contexts and dependency injection for the java ee platform. <http://docs.oracle.com/javase/6/tutorial/doc/giwhb.html>, 2013 (accessed December 6, 2016).
- [19] Oracle. Overview (javamail api documentation). <https://javamail.java.net/nonav/docs/api/>, 2015 (accessed December 6, 2016).
- [20] Oracle. Javamail api. <http://www.oracle.com/technetwork/java/javamail/index.html>, 2016 (accessed December 6, 2016).
- [21] Oracle. Jsf 2.2 documentation. <https://jaserverfaces.java.net/nonav/docs/2.2/>, 2016 (accessed December 8, 2016).

Appendices

A. Setup Guide

Below is a step by step guide on how to setup the project in a Linux environment.

- 1: Install Java 8 Development Kit
- 2: Install Tomcat (v9.0.0.M10) or any other Java 8 compatible Server.
- 3: Install Maven 3.3.9
- 4: Set the Environmental Variables

Launch the terminal and run `sudo gedit /etc/environment`, scroll to the end of the opened file and add the following text to it

```
JAVA_HOME=/<JAVAPATH>
export PATH=/<MAVENPATH>/bin:$PATH
export JAVA_HOME
```

Where `<JAVAPATH>` and `<MAVENPATH>` take the value of your installation paths respectively. Save your file and exit. Reload the system path with the following command `./etc/environment`.

- 5: Modify the Hardcoded Strings

Replace the private and public Captcha keys in `web.xml`. In order to register the Captcha with Google and link it to your domain access <https://www.google.com/recaptcha/admin>

Replace the SMAP properties and login information in the `EmailService.java`.

- 6: Compile the Web Archive (.war)

Navigate to the source directory of the project (containing the `pom.xml` file) and start a terminal from that location. Run `mvn clean install` in the terminal window and wait until Maven downloads all the dependencies and compiles the classes into one archive.

- 7: Deploy the Archive on the Server

Deploy th file `./target/calendar-files-1.0.0.war` on the server.

B. Default Configuration File

Structure and content of the default .json configuration file:

```
{
  "departments": [
    {
      "department_name": "Technik und Informatik",
      "majors": [
        {
          "prefix": "A-M",
          "degree": "Master's",
          "name": "Automatisierung"
        },
        {
          "prefix": "E",
          "degree": "Bachelor's",
          "name": "Elektro- und Informationstechnik"
        },
        {
          "prefix": "B-EE",
          "degree": "Bachelor's",
          "name": "Regenerative Energiesysteme"
        },
        {
          "prefix": "IE",
          "degree": "Bachelor's",
          "name": "Information Engineering"
        },
        {
          "prefix": "IKM",
          "degree": "Master's",
          "name": "Informations- und Kommunikationstechnik"
        },
        {
          "prefix": "BMT",
          "degree": "Bachelor's",
          "name": "Mechatronik"
        },
        {
          "prefix": "MES",
          "degree": "Master's",
          "name": "Mikroelektronische Systeme"
        }
      ]
    }
  ]
}
```

C. DVD Contents

This Bachelor Thesis contains an appendix of program listings, configuration files and other assets required for setting up the service. This appendix is deposited with Prof. Dr. rer. nat Henning Dierks within Berliner Tor 7, 20099 Hamburg.

Declaration

I declare within the meaning of section 25(4) of the Examination and Study Regulations of the International Degree Course Information Engineering that: this Bachelor report has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, December 9, 2016

City, Date

sign