Höhere technische Bundeslehranstalt
und Bundesfachschule
im Hermann Fuchs Bundesschulzentrum

TableWizard

Diploma thesis

school focus
Cybersecurity

executed in school year 2024/2025 from:

Lukas Daxecker, 5AHITS

supervisor:

Dipl.-Ing. (FH) Dipl.-Ing. Christian Probst

November 28, 2025

Statutory declaration

We declare in lieu of oath that we have written this thesis independently and without outside help, that we have not directly used sources and aids other than those indicated and that we have identified the sources used as such, both literally and in terms of content.

Braunau/Inn, 28.11.2025	Lukas Daxecker	_____
Place, Date	*Author*	*Signature*

Abstract

This thesis aims to identify the most suitable technologies and architectures for developing a booking program, while also creating a functional system for training management. The research is valuable for developers trying to build similar applications and trainers who need an efficient way to manage training sessions and groups.

For this purpose, two applications were developed. The first using `Flutter` with a `tightly coupled architecture`, alongside one that uses `Plesk`, `Adminer`, `PHP`, and the `MVC architecture`. Both are web-based and where compared with the main focus lying on the front- and back-end. Mainly on feasibility but also in terms of development experience.

The study concludes that the `MVC architecture` is far more effective for building a robust system like the one proposed, while `Flutter` proves to be more enjoyable and efficient for developing the front-end and application styling.

Kurzfassung

Diese Arbeit zielt darauf ab, die am besten geeigneten Technologien und Architekturen für die Entwicklung eines Buchungsprogramms zu identifizieren und gleichzeitig ein funktionales System für das Trainingsmanagement zu erstellen. Die Forschung ist sowohl für Entwickler, die ähnliche Anwendungen erstellen möchten, als auch für Trainer, die eine effiziente Möglichkeit zur Verwaltung von Trainingseinheiten und Gruppen benötigen, von Wert.

Zu diesem Zweck wurden zwei Anwendungen entwickelt. Die erste verwendet `Flutter` mit einer `stark gekoppelten Architektur`, die andere basiert auf `Plesk`, `Adminer`, `PHP` und der `MVC-Architektur`. Beide sind webbasiert und wurden miteinander verglichen – mit dem Hauptaugenmerk auf Front- und Backend. Dabei lag der Fokus vor allem auf der Umsetzbarkeit, aber auch auf der Entwicklungserfahrung.

Die Studie kommt zu dem Schluss, dass die `MVC-Architektur` wesentlich effektiver für den Aufbau eines robusten Systems wie dem vorgeschlagenen ist, während sich `Flutter` als angenehmer und effizienter in der Frontend-Entwicklung und im Design der Anwendung erweist.

Contents

1 Introduction

If you are in a sports club or have a training group, you will need a way to organize your training sessions. This includes:

- Creating, cancelling or rescheduling training sessions
- Managing who is allowed to attend each session
- Checking who and how many people have booked a training
- Adding or removing trainees from a group
- Checking the training history of trainers and trainees alike

This thesis covers the creation of two different approaches to a software application capable of managing the tasks above.

1.1 Problem Setting

Consider yourself a trainer, whether in a club or a private setting. You have several groups, categorized by age or skill level, and you want to organize them efficiently. Most of the time, training sessions are managed through messaging services, which become increasingly disorganized as more people join. Therefore, you need an app or website where your trainees can book sessions, and you can create, cancel, and reschedule them.

At this point, having an overview of who and how many people have booked each session would be highly beneficial. Additionally, a simple way to manage the number and types of sessions a trainee is allowed to attend could make your life much easier.

1.2 Problem Statement

Developing such a program requires technical knowledge and a significant amount of time. If building it independently is not feasible, hiring a professional is an alternative, but that can be quite costly. Most clubs and private trainers lack the expertise or financial resources to hire professionals. As a result, they often continue struggling with their current, inefficient systems.

1.3 Solutions

The solution is a web application that allows trainers to manage their trainees and training sessions in one place. Additionally, a mobile app or website would enable trainees to book and view their sessions. These should be:

- Codeless
- Easy to learn
- Easy to navigate
- Easy to maintain
- Affordable
- Time-efficient

1.4 Structure of this Thesis

This thesis compares two approaches to developing such an application. It does this in the following chapters:

- **System Overview:** This chapter explains all the technologies that are mentioned in this thesis.
- **Training Management:** It describes how training management works today and what problems it faces.
- **Concept:** The `Concept` covers the functionalities of the two applications and how they solve the problems mentioned in the `Training Management` section.
- **Development:** This chapter covers how the applications get developed in which technologies they use.
- **Back-end:** It covers the hardware infrastructure as well as how the code works, including its architecture. Also containing a comparison between the two applications.
- **Front-end:** The `Front-end` covers the style and user interaction of the applications. As well as a comparison between the two applications.
- **Evaluation:** This chapter covers how the solutions for the problems mentioned in the `Introduction` were implemented.
- **Project Management:** This section goes over the project management conducted while creating the two apps.
- **Future Work:** It explains what changes can be made in the future if the projects are continued.
- **Related Work:** A few products and applications that are somewhat similar to those created for this thesis.
- **Conclusion:** The `Conclusion` covers some final thoughts about how it all worked out.

2 System Overview

The following sections will explain the technologies being used.

2.1 Programming Languages

2.1.1 SQL

Structured **Q**uery **L**anguage (`SQL`) is not considered a programming language. It does not have the ability to create loops, conditions, or functions and is therefore called a domain-specific language, with the domain being databases. SQL provides the syntax and functionality to insert, update, delete, and select data inside a database, as well as create tables within the database.[1]

2.1.2 JavaScript

`JavaScript`, or JS for short, is a programming language used to create the logic of websites. It is used for interactivity on websites as well as updating content on the site. Another important area of use is the management of cookies and their stored data.[2] The logo can be seen in figure 2.1.

Figure 2.1: JavaScript logo
[3]

Ajax

`Ajax` is used to update the content of a website. Updating only defined parts of a webpage, without the need to reload the entire site, enhancing speed and eliminating loading times in the process. This is done by periodically sending requests to the back-end. It is a technique used in `JavaScript` programming and can be performed without downloading any packages, making it a very useful tool.[4]

2.1.3 CSS

Cascading Style Sheets (CSS), the logo can be seen in figure 2.2, is not a programming language but a markup language. This means it describes how things should look and how they should be organized and displayed. It is used to design websites with colors, positioning, fonts, rudimentary effects, and more.[5] The reason it doesn't count as a programming language is that it does not have the ability to make decisions or contain any real logic. Although it can be used in tandem with JavaScript to create more complex animations.

Figure 2.2: CSS logo
[6]

2.1.4 Bootstrap

Bootstrap is an extension for CSS, making designing websites much easier. It contains pre-designed templates and components, which can be customized and changed as needed. Users also have the ability to create their own templates from scratch. The layout is based on a predefined grid, where the components of the website can be placed. Furthermore, it has the option to include JavaScript plug-ins to add functionality to items (tooltips, drop-down menus, ...).[7] The logo is shown in figure 2.3.

Figure 2.3: Bootstrap logo
[8]

2.1.5 HTML

HTML, **H**ypertext **M**arkup **L**anguage, provides the bare-bones structure of a website, consisting of a series of different elements like paragraphs and headers. It doesn't have the ability to implement logic on the website and is therefore not considered a programming language.[9] The actual design and logic of the webpage are implemented with CSS and JavaScript respectively. The logo is shown in figure 2.4.

Figure 2.4: HTML logo
[10]

2.1.6 PHP

PHP, the logo is presented in figure 2.5, is used to interact with databases and create dynamic content on webpages. It can be directly implemented into HTML files to provide logic to the webpage. Data can be directly stored in cookies, and it provides the ability to change the structure of the webpage by directly interacting with HTML code.[11]

Figure 2.5: PHP logo
[12]

2.1.7 Dart

Dart is a programming language that is mainly used to build web and mobile applications. Most of the time, it is used with the Flutter framework. It automatically converts the written code so that operating systems like iOS and Android can use it, removing the need for programmers to worry about the specific differences between the systems.[13] The logo can be seen in figure 2.6.

Figure 2.6: Dart logo
[14]

2.1.8 Flutter

Flutter, the logo is shown in figure 2.7, helps programmers build apps for multiple operating systems without them having to worry about the differences between those systems.

Especially in the area of design, Flutter offers many pre-built elements and automatically adapts to the look of the correct operating system. It is used with the programming language `Dart`, which is used to create the program's logic.[15]

Figure 2.7: Flutter logo
[16]

package: webview_flutter

The package `webview_flutter` is used to display an already existing website in your mobile app. It can be used as part of an application, like a window displaying the Google Maps location, or as the app itself. This eliminates the need to think about and rethink the design and logic of your app, as it simply takes it from your already existing website.[17] The package is not included by default in `Flutter`, and therefore must be imported.

package: provider

The `provider` package is an efficient tool for state management, which makes it possible to manage and share the state of your application and to access and update data without unnecessary rebuilds. State represents the current information, data, and processes available in the application. Rebuild means that the user interface is generated again with new data.[18] The package is not included by default in `Flutter`, and therefore must be imported.

package: image_field

An `image_field` simplifies the process of selecting, displaying, and managing images. It is helpful for applications that require the user to upload or update an image from their gallery, files, or camera.[19] The package is not included by default in `Flutter`, and therefore must be imported.

package: file_picker

A `file_picker` is used to select files so their contents can be read or edited. [20] The package is not included by default in `Flutter`, and therefore must be imported.

package: web_color_picker

With the `web_color_picker`, a color wheel to pick colors can be provided for the user. It then notifies the app's logic that a color has been chosen and is ready to be used. This package can only be used in web applications.[21] The package is not included by default in `Flutter`, and therefore must be imported.

package: archive

`archive` allows developers to work with compressed files like ZIP. With `archive`, compressed files can be imported or exported from the project, and they can be encoded and decoded.[22] The package is not included by default in `Flutter`, and therefore must be imported.

package: drift_dev

drift_dev can only be used in development and defines everything around databases in Flutter. This includes the database structure, as well as other functionalities like SQL queries and database migrations. Importantly, the package on its own cannot generate code but only defines schemas.[23] The package is not included by default in Flutter, and therefore must be imported.

package: build_runner

It is a tool that is used to generate code, which is only used in the development process. Some libraries, like drift_dev, can write schemas that are then unable to be executed. Those can then be run with the build_runner package.[24] The package is not included by default in Flutter, and therefore must be imported.

package: drift_flutter

Optimizes SQLite usage inside Flutter. Its main purpose is to handle database storage correctly. Important to note, the package cannot be used in pure Dart and needs a Flutter project to work.[23] The package is not included by default in Flutter, and therefore must be imported.

package: sqlite3_flutter_libs

It is used to bundle the newest SQLite versions into the application. This works on every operating system and is important to ensure that database operations run smoothly.[25] The package is not included by default in Flutter, and therefore must be imported.

2.2 Server & Database

2.2.1 Plesk

Plesk, the logo can be seen below in figure 2.8, is a web-based server management platform. It can be used for managing the entire back-end hardware infrastructure, such as web servers and storage servers. Providing the ability to manage your code and databases, as well as the permissions of users. Additionally, things like domain names and certificates can be created and managed on the platform.[26]

Figure 2.8: Plesk logo
[27]

2.2.2 SQLite

SQLite is a server-less and self-contained database management system. It doesn't need an extra database server like traditional databases, as it is embedded in the application, simply meaning it runs within the app. Furthermore, it doesn't require any configuration, is cross-platform (meaning it works on multiple operating systems), and is optimized for low-memory and high-speed performance.[28] The logo can be seen in figure 2.9.

Figure 2.9: SQLite logo
[29]

2.2.3 Drift

`Drift`, previously called `Moor`, is a database library for `Dart`, with the actual data being saved in a `SQLite` database. It gives users the ability to interact with databases using simple Dart code and, furthermore, supports real-time updates by allowing apps to listen to changes in the database with Dart. And because it is based on SQLite, it works perfectly on all operating systems.[30] Its logo is shown in figure 2.10

Figure 2.10: Drift logo
[30]

2.2.4 Adminer

`Adminer` is a database management tool that helps to store and organize data, like user credentials, that are saved in a database like `SQLite`. Users can manage their databases through a simple web interface provided by Adminer. It is installed by simply uploading a single file to the server and referencing it in the code. Therefore, it is not only easy to use but also easy to set up.[31] The logo is shown in figure 2.11.

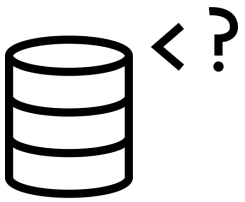


Figure 2.11: Adminer logo
[31]

2.3 Development-Tools

2.3.1 Visual Studio Code

`Visual Studio Code`, commonly referred to as VSCode, is a simple yet powerful programming environment developed and maintained by Microsoft. Its main advantages include a vast extension ecosystem maintained by both Microsoft and the developer community, `Git`

integration, the ability to work on remote machines via SSH, and compatibility with nearly all modern programming languages. Extensions can be installed with just a few clicks, offering functionalities ranging from simple syntax highlighting to AI-based code generation.[32] Visual Studio Code's logo is shown in figure 2.12.

Figure 2.12: Visual Studio Code logo
[33]

2.3.2 Git

Git is a version control software. This means it keeps track of the various versions, called branches, of your project, referred to as the repository. Giving you the ability to work, either alone or as a team, on different features at the same time without interfering with each other. It also gives you the opportunity to look back on changes you made a long time ago, without the need to store the code yourself.[34] The logo is shown below in figure 2.13.

Figure 2.13: Git logo
[35]

2.3.3 GitHub

GitHub, the logo can be seen in figure 2.14, is used, as the name implies, as a hub for your project and teams. It stores the code securely in the cloud and is therefore available from anywhere, making teamwork much easier. Differentiating itself from Git, which stores the data locally on your machine. However, it should be noted that GitHub is built around and controlled with Git. In addition, there are project management utilities available, such as milestones, boards, code reviews, status checks, and much more.[36]

Figure 2.14: GitHub logo
[37]

2.3.4 XAMPP

XAMPP is a software that allows you to set up local servers for development and testing purposes. The logo can be seen below in figure 2.15. Working directly on your server introduces significant security risks, as well as the possibility that the site could shut down, preventing users from accessing it. For these reasons, it is useful to set up a local web server and database, which are all provided by XAMPP, and upload the code when it is ready.[38]

Figure 2.15: XAMPP logo
[39]

2.3.5 Android Studio

Android Studio, the logo can be seen in figure 2.16, is the official IDE, **I**ntegrated **D**evelopment **E**nvironment, for writing code and developing Android applications. An example of an IDE would be Visual Studio Code. Offering tools to develop and test programs in multiple programming languages, such as Java, Kotlin, and C++. It also provides the possibilities to design the user interface with a visual editor, test on emulators, manage dependencies, and publish apps.[40]

Figure 2.16: Android Studio logo
[41]

Emulator

The emulator from `Android Studio` can simulate the functionality of a broad variety of Android phones. It can be used to test the functionality of an application locally on the developer's machine.[42]

2.4 Technologies

2.4.1 Language Server Protocol

`LSP`, **L**anguage **S**erver **P**rotocol, is a protocol that IDEs, **I**ntegrated **D**evelopment **E**nvironments, like `Visual Studio Code`, use to communicate with language servers. They have specific knowledge about the programming language the developer writes in, improving the development experience by providing quality-of-life features, such as auto-completing code and underlining errors.[43]

2.4.2 SSH

Secure Shell (`SSH`), the logo is shown in 2.17, is a protocol that provides the user with a secure way to connect to machines and servers. After logging in with one's credentials (username, password), the user has the ability to work on the machine as if they were physically on it. The connection is encrypted and therefore ensures a secure transmission of your data.[44]

Figure 2.17: SSH logo
[45]

2.4.3 Cookies

`Cookies` are used to save user data preferences, like items put into the shopping cart or the settings of a website. Depending on the relevance of the data, the duration of storage can be adjusted. They provide an easy way to send data between the front-end and the back-end but pose a certain security risk because they can be changed and viewed by the user.[46]

2.4.4 Web Socket

`WebSockets` build a connection between the back-end and the front-end. Through this connection, data can be transmitted as soon as changes happen making it extremely valuable for updating content on a webpage in real-time. The technology can be implemented in every programming language and is therefore easy to use.[47]

3 Training Management

Training management is an important part of the life of each and every trainer. It doesn't matter if it is in a club or a private setting. Most of the time, training sessions are managed in a simple messenger group, for example, in WhatsApp. The trainer writes when the training is, and the trainees write back if they can come. Sometimes the trainer also gives information about the contents of the training. If it is just a youth trainer at the local tennis club and they only have a group with a few trainees, this way of doing it may work. However, a lot of the time, this is not the case. Trainers often have multiple groups based on varying factors like skill or age, and those groups possibly have dozens of trainees. Now, having a single group chat can create a multitude of problems for the trainer and the trainees.

3.1 Problems

Modern training management runs into various problems, ranging from scheduling training sessions to attendance monitoring. They not only make it harder for the trainers but oftentimes also annoying for the trainees.

3.1.1 Rescheduling Trainings

As soon as a training session has to be rescheduled or gets cancelled, the problems begin. Some trainees may overlook the new message and still answer the old one. Others may not write again as they have already answered the previous message. These are only two quick examples of what could happen. Issues like these make it very hard for a trainer to prepare the training, as they cannot know for sure how many will attend.

The trainees also suffer from this, as there is always a multitude of new messages coming in. Each one could be important and, therefore, more or less has to be checked. And if they then overlook something, it could be that they come on the wrong date and miss the training.

That is if there is only one group. If the trainer now has multiple groups, as is almost always the case, a bunch of new problems arise. Each group has to have its own group chat, or things will become unorganized very quickly. Now, the trainer has to monitor multiple groups where the problems mentioned before all apply.

3.1.2 Training History

In some cases, like in a martial arts school, the trainer needs to know how often the student was in the training sessions and which trainings they attended. With that knowledge, the trainer has the ability to decide whether students could be ready for belt exams or not, especially if the groups are too big to always keep track of everyone. A second example

would be a trainer who requires their trainees to visit a certain number of training sessions in order to partake in tournaments. They either have to manually keep a list or look up which training sessions the students said they would attend. While the first option introduces unnecessary and tedious paperwork, the second option is very time-consuming and runs the risk of overlooking trainees. But information like that can help them keep track of what they have already taught and how often they held the same lessons. With that, they can then plan the contents of the next training sessions.

The same goes for the students. When they want to know which or how many training sessions they have attended, so they can do their next exam or compete in the next tournament, they either have to keep a list or track it back in the group chat.

3.1.3 Keeping Track Of The Trainings

Trainers somehow need a way to know when their trainings are and what contents they need to prepare for said training. Most of them keep the dates in apps like Google Calendar [48], but those have to be manually entered as well as manually changed when the training date changes. Students also have to keep track of the trainings somewhere, especially if they are in more than one club.

When stressed, a person can easily forget to put it in the calendar and then maybe completely forget about it and miss it. Even though this affects trainees more than trainers.

4 Concept

This chapter covers the functionality of the two applications and how they work, without going into the technical details. With the PAP application also explaining how it solves the problems mentioned in the training management chapter before. The Flutter app covers more general problems from the introduction, as it wasn't built specifically for training management.

4.1 Flutter

The application is a web-based software that gives users the ability to create their own booking program in a modular fashion. In this case, the app is not designed for training management specifically but for booking programs in general, and therefore also covers the more general problems from the introduction. This means it can be used for varying tasks, such as a court-booking system for a tennis club, a custom calendar for small organizations, as well as for training management.

An app being web-based simply means it runs online and is accessible via the browser, bringing several advantages. Firstly, it can always be used, no matter where or when, as long as there is a stable internet connection, removing the need for installations and complicated setups. Furthermore, there is no dependency on operating systems and can therefore be used on any device.

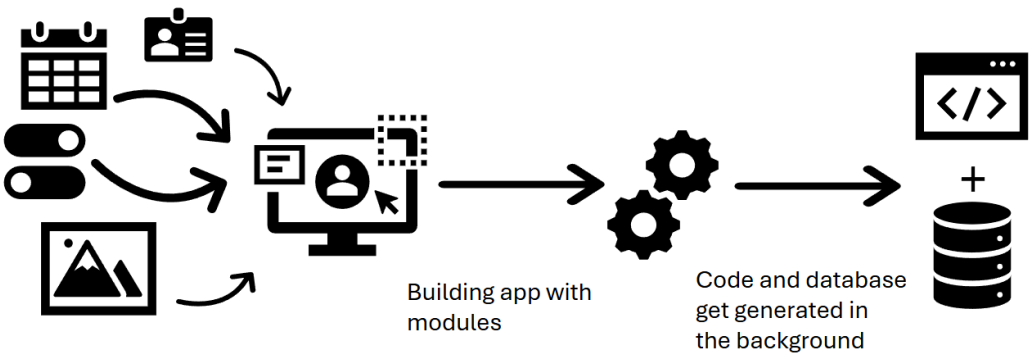


Figure 4.1: Modular application building

The booking program being modular means that modules needed for applications, like buttons and tables, are provided by the app itself and can be, through drag and drop, placed where they are needed. After being placed, the looks and attributes, like width, height, and position can be edited. With that system, users can create their own application without any coding knowledge and in a very short time. When the user has composed and designed

the app, the required code and databases are generated afterwards and ready to use. This process is presented simplified in figure 4.1 above.

4.1.1 Functionality

The structure of the application is displayed in figure 4.2. On the left is the list of the modules that can be used to build the application. In the middle is the workspace, the spot where the modules are dropped and changed. On the right is the inspector, which displays the attributes of the modules as well as the `render button` which is used to generate the built applications.

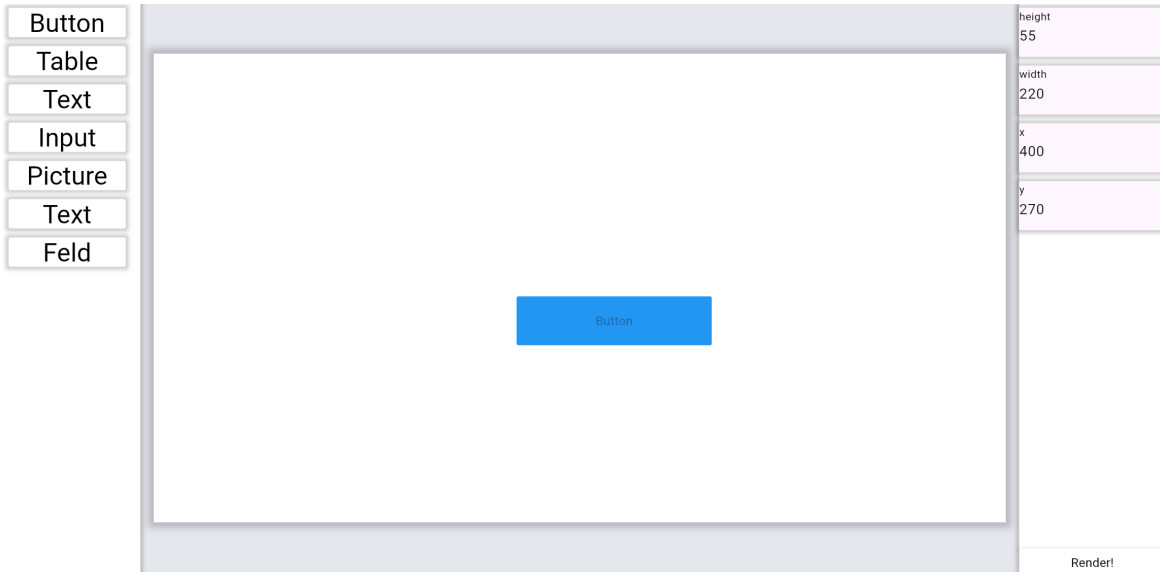
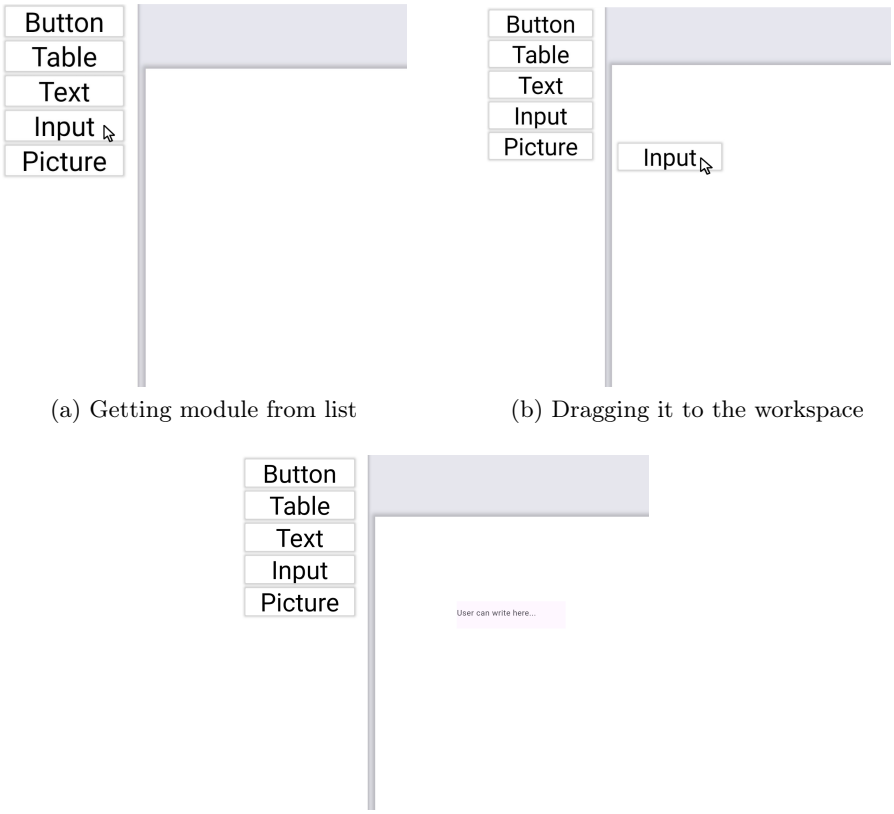


Figure 4.2: Structure of the application

A list of modules, also called widgets, is provided by the software. Those range from buttons to pictures, and with them, the user can build their own application and design it as they see fit.



(a) Getting module from list

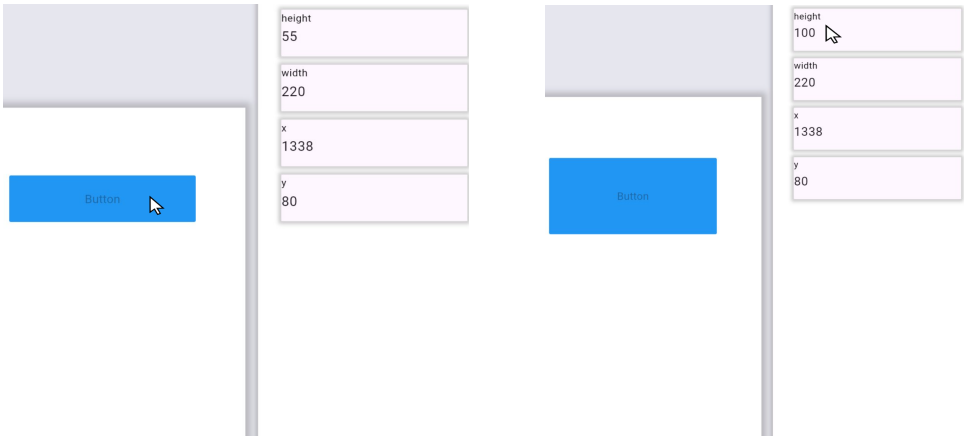
(b) Dragging it to the workspace

(c) Dropping it in the workspace

Figure 4.3: Example for inserting new module

In figure 4.3a is the list with said modules, positioned way to the left on the screen. They can be selected and dragged over to the workspace, as seen in figure 4.3b, and then dropped to place it, like in figure 4.3c.

The inserted parts are selectable and their attributes, like height and width, adjustable as needed. This process is shown in figures 4.4a and 4.4b below. They are also able to be moved manually by dragging them around and dropping them where they are wanted. Unwanted or obsolete widgets can be permanently deleted by dropping them outside the workspace.



(a) Selecting module (b) Edit attributes

Figure 4.4: The attribute of a module gets edited

Tables are used as the main method for managing bookings. Therefore the structure of the database adapts to the attributes and data types, like numbers and dates, present in the tables.

At any point in the process the application can be rendered. Rendering in this case refers to the process of taking everything that was placed in the workspace, like modules and their design, and converting it into usable Dart code. Importantly this also covers the creation of the databases for the final application. Those are then used to store the booking data of said program. The new application is then automatically downloaded and can be used.

4.1.2 Solutions

This section covers how the solutions mentioned in the introduction are implemented.

The user does not need any coding or technical knowledge for the application. Therefore, it is very easy to learn and use, so that everybody can create and maintain a booking program.

With its structured and simple layout, the application can be navigated effortlessly, and the wanted functionalities found in no time.

The program is rather cheap as it is completely free. Meaning that even small clubs or private individuals can afford to build their own application, without being reliant on external developers. Building it also takes only a small amount of time, especially when compared to the process of developing it from the ground up, as all of the building blocks are already provided.

4.2 PAP

It is a mobile and web-based application, giving trainers the ability to manage their groups and trainings. This means that tasks like creating, rescheduling trainings, or making attendee lists can all be done in the app. The focus hereby lies in the web, with the mobile app only

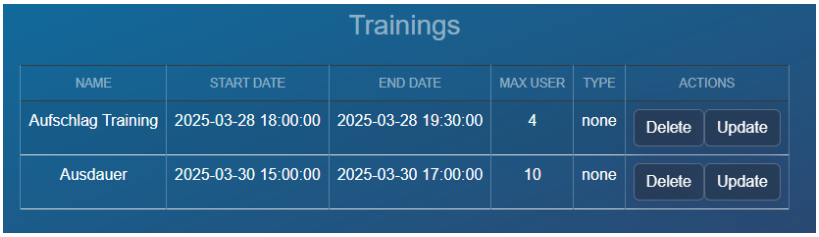
being an add-on for faster access on smartphones. An app being web-based brings several advantages, as already mentioned in the Flutter concept.

4.2.1 Functionality

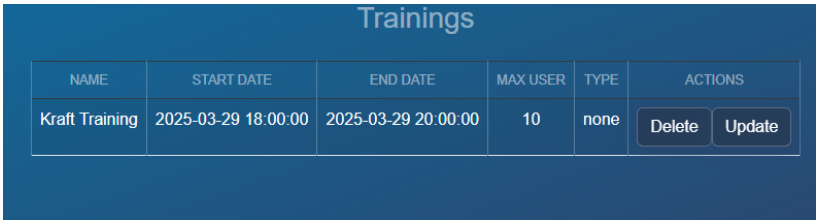
Trainers have the ability to create groups, which they then manage and control. Those groups are simple tools for the trainers to separate different clubs or organizations. For example, if they have trainees in two different tennis clubs, these can be separated with the help of those groups. In those groups they can perform various tasks:

- Manage training types
- Create, reschedule, and delete trainings
- Manage members
- Get training information of their trainees
- See their training history

All the things that are created or added inside a group are then only available in said group. So when a trainer schedules a training, it is only visible in the group they created it for, as shown in figures 4.5a and 4.5b.



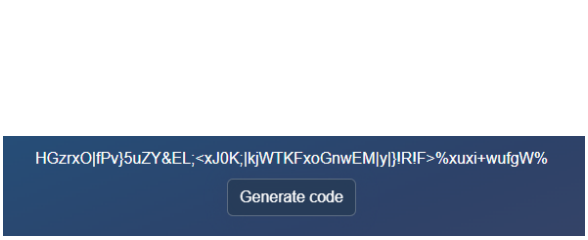
(a) Trainings UTC Neukirchen



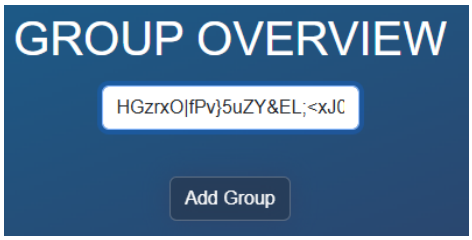
(b) Trainings TC Braunau

Figure 4.5: The training is only showing up in one group

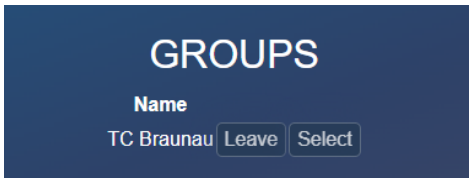
Those groups have no member limit and can also contain people who are not actually trainees, like supporting members. To add a member to a group, the application generates a code by simply pressing a button, which can be sent to potential members by the trainer. This code can then be used by the members to join the group, as shown in figures 4.6a, 4.6b, and 4.6c. There, the trainer first generates it by pressing the button `Generate code` and then sends it to a member. The member simply takes the code and enters it, on the `Groups overview` page, into the text field. When `Add group` is pressed, the member joins the group and it gets added to their group list.



(a) Trainer generates code



(b) Trainee enters code to join



(c) Group is added to list

Figure 4.6: Process of a member being added to a group

To organize a group, the trainer creates training types, which are just simple tags. These tags are then given to trainings to signify what kind of training this is, for example, if it's only for a certain skill or age range. Members can then be assigned one or multiple of those tags to control who can visit which training and how often they are allowed to attend. They also only receive messages when trainings are created, scheduled, or deleted that concern them. These tags can be added, changed, and deleted at any time, allowing the trainer to move trainees from one team to another.

This group structure can be seen in figure 4.7. At the top is a trainer who owns the groups. These groups consist of members, who all have tags to identify which trainings they are allowed to visit. Trainings also have those tags, called training types, assigned to them by the trainers. The tags and trainings only count within the group they were created.

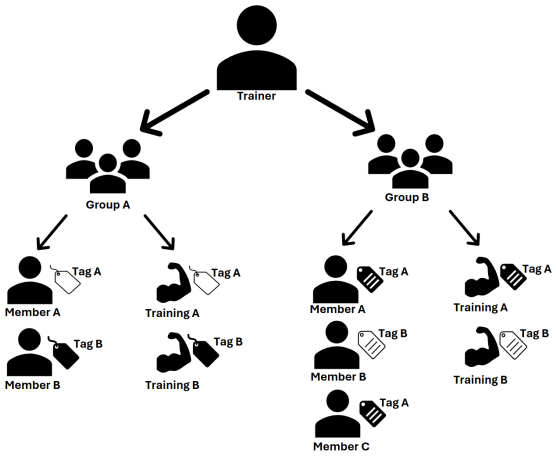


Figure 4.7: Group structure

Trainings themselves have multiple attributes that can be set when creating them and changed afterward. Those attributes include, as already mentioned, the training type, as well as the maximum number of participants, when it starts and ends, the name of the training, and how often it repeats. Trainers are also able to get a quick overview of the current status of the training registrations. As shown in figure 4.8, the trainer can view how many and who booked the trainings.

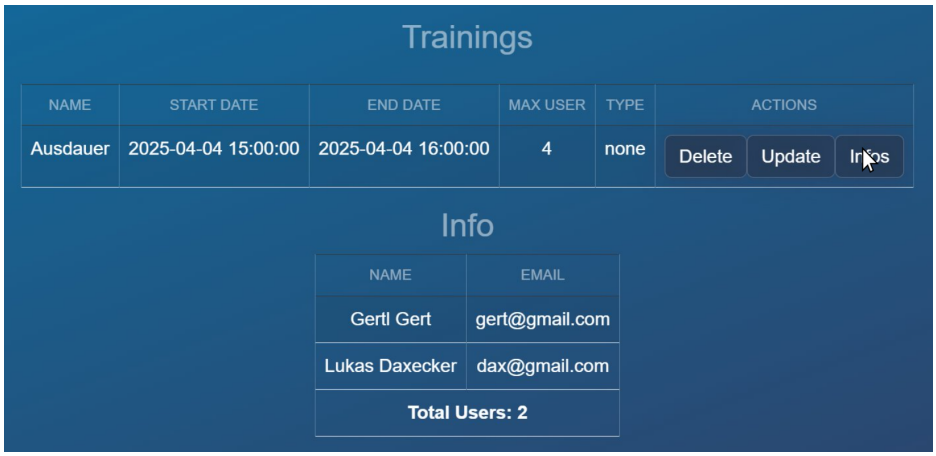
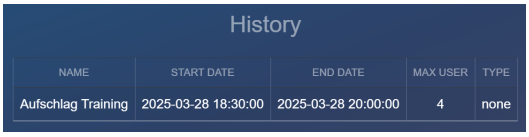


Figure 4.8: Overview of registered trainees

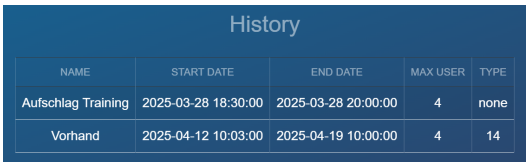
Trainees can book these trainings but also cancel the booking if something gets in between. They also get a message if something important changes, like the date of the training, with only trainees being notified that are directly impacted by the change. There is also the ability to connect the application to the user's calendar app to automatically save the dates. Trainees have the ability to look up their training history, as shown in figure 4.9a. In there, they can see statistics about their trainings, including which trainings they visited, how often they visit, and for how long they were in trainings.

The trainers can also view their history, like in figure 4.9b. They can see the trainings they held, as well as how many and who visited them. Statistics like how many trainees visit a training on average, the amount of trainings they have hosted, and the total duration of this in hours are also shown.

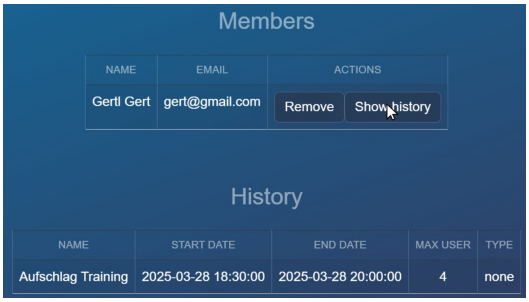
Furthermore, trainers are able to select trainees and view their history. Importantly, they can only see statistics about the group they are the trainer in. This is shown in figure 4.9c. Trainers can view which trainings the trainees have visited and how often they are attending trainings.



(a) Trainee training history



(b) Trainer training history



(c) Trainer watching trainee training history

Figure 4.9: Examples of the history function

4.2.2 Solutions

This section goes over the problems mentioned in the problems section, as well as general benefits of the application.

Problems like overlooking trainings or getting a message for everything, explained in more detail in the rescheduling trainings section, are solved with tags or more precisely training types. A trainer doesn't have to worry about multiple messenger groups, as they can put multiple groups into one. The tags give them the ability to easily separate the members of the group, on whichever basis they want, like contracts, age, or skill. Furthermore, trainees only get notifications when the contents actually affect them. Also, when a training is rescheduled, their booking is automatically terminated and has to be booked again. This makes it a lot easier for the trainer to plan their trainings.

Trainers, as well as trainees, get a better overview of trainings they held or visited. As mentioned in the training history paragraph, this normally is a rather tedious task. The application provides an overview for trainers to view the trainings they have held as well as who and how many attended them. Also, they are able to look up their students and check how often they train and which trainings they visited. Trainees are also able to access this overview and check their stats. These overviews can be seen in figures 4.9a, 4.9b, and 4.9c.

The sections keeping track of trainings briefly touch the subject of saving the training dates. Those aren't problems anymore as the application offers an overview of all of the trainings to come.

4.3 Cybersecurity

Cybersecurity is only represented in PAP as the Flutter app didn't reach the level of development that security was a big enough part of the project.

That being said, making the PAP application secure is a top priority. To do this, the main focus lies on the `OWASP top ten`[49] that shows the ten most common attacks conducted on websites and web apps, as well as the `OWASP coding practices`[50] that explain ways to prevent the attacks mentioned before. Including `session management`, `input validation`, and `output encoding`. These guidelines and tips are implemented to make the application as secure as possible.

5 Development

Development covers the process behind the creation of the application and code management, as well as a quick overview about which technologies are used and where they are used.

5.1 Technologies Overview

Here is explained which technologies are used by each of the applications without going into implementation details.

5.1.1 PAP

The acronym `PAP` stands for **P**lesk, **A**dminer, and **P**HP. HTML, CSS, and Bootstrap are being used to write the front-end, with the back-end being written in JavaScript and PHP. While Plesk provides us with the servers, which host the Adminer-managed SQLite database, and the web server used for hosting the website. Visual Studio Code is used as the programming environment, which provides the required LSPs for the programming languages in use. Furthermore, it provides the ability to connect to the web server via SSH. For testing the database and the functionality of the code, XAMPP is used, which provides the ability to host web and database servers locally.

5.1.2 Flutter

The app is written in Dart and Flutter. With Dart being used for the functionality and the logic. Flutter is used to design and create the looks of the application, with it automatically converting the looks to fit the operating system the application runs on. The figures 5.1a and 5.1b show how the same widgets, these are the building blocks of Flutter, would look, with figure 5.1a representing `Android` and figure 5.1b representing `iOS`.

(a) Switch and slider on Android

(b) Switch and slider on IOS

Figure 5.1: Flutter automatically converts the looks of the widget depending on the operating system

[51]

To get all the needed functionality, the libraries `provider`, `image_field`, `file_picker`, `web_color_picker`, and `archive` are utilized. Also, the database `Drift` is used to store user data, which requires the packages `drift_dev`, `build_runner`, `drift_flutter`, and `sqlite3_flutter_libs`. As the programming environment, `VSCode` is used, which also provides an LSP for Dart. Furthermore, it simulates a local web server for testing purposes, eliminating the need for extra software. This server can be accessed over any web browser without the need for an internet connection.

5.1.3 Mobile

The mobile app is written in Dart and Flutter. Because Flutter does not provide all the needed functionality, the package `webview_flutter` is also added. With that, it is possible to bind the already existing PAP application to the mobile app. `VSCode` is not only the programming environment but also provides an LSP for Dart. `AndroidStudio` provides an emulator to simulate a mobile phone which then runs locally. This can then be used for testing and running the code.

5.2 Process

This covers how the applications are created and their code is managed.

5.2.1 PAP

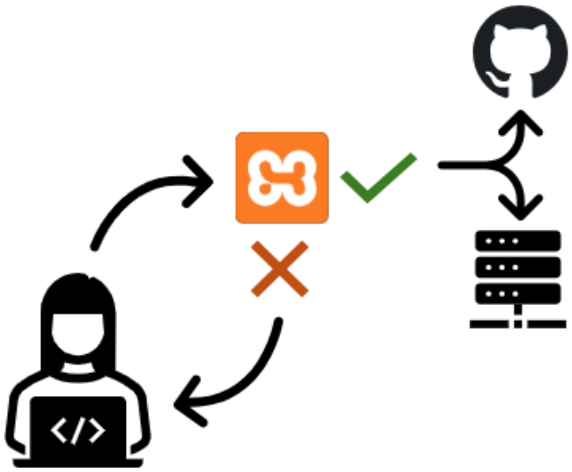


Figure 5.2: Process of the PAP-Application development

Features, which are small parts of an application like new functionalities or a new design, get developed. The code is written locally and then also tested locally by setting up a web- and database-server with XAMPP. If it works, it gets stored on GitHub to save the progress. When the wanted functionality is not given, the code gets reworked until it functions as intended. Then, if everything works as intended, we upload it to our actual web server via SSH. Meaning that the developed feature is instantly available on the webpage after upload. This process can be seen in figure 5.2 above.

5.2.2 Flutter

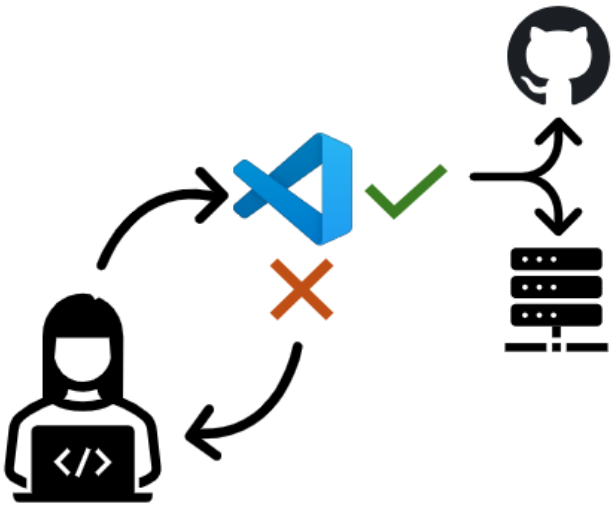


Figure 5.3: Process of the Flutter-Application development

Features, which are small parts of an application like new functionalities or a new design, get developed. The code is locally written on a computer or laptop. When a feature is ready,

it gets tested on the VSCode web server. If it then works as intended, the code is pushed to GitHub to store the progress. Is that not the case, it gets revised and then tested again until it works. The cycle continues until everything works properly. When there are enough features developed, the new version of the application is released. This process can be seen in figure 5.3 above.

5.2.3 Mobile

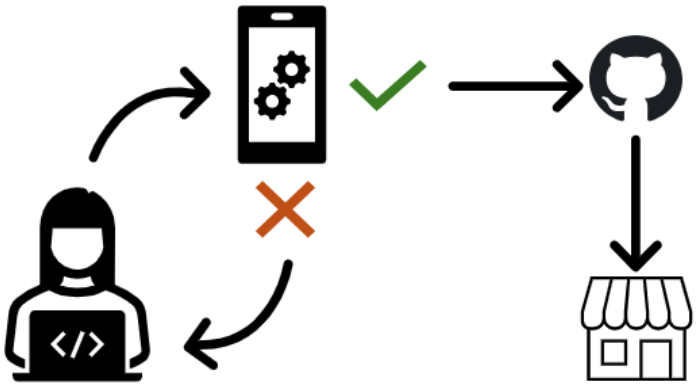


Figure 5.4: Process of the mobile app development

At first, the software is developed and then tested on the emulator provided by Android Studio. If the code works, a copy of it is saved on GitHub to store the progress. In the case that there are any errors or problems when testing, those are backtracked and reworked. Then, when the planned features are finished and ready, they are published. This process is shown in figure 5.4 above.

6 Back-end

The back-end is the server-side area of the application where all the hardware and the actual logic of a website reside. It is used to store and process data and manages the more complex interactions between users and the webpage.

6.1 Web-server and Data-server

Because both the PAP, **P**lesk **A**dminer **P**HP, and the Flutter application are web-based, Plesk is used for both.

It provides the web server for the application, as well as managing the HTTPS certificate and any additional requirements, such as blacklists and whitelists, that may be needed. Blacklists are used to block users, while a whitelist is used to allow users on a website. A database server could be hosted, but because both Adminer and Drift are lightweight databases and do not require an extra server, this Plesk service is not used. Plesk also provides tools to manage who can develop and access the servers, as well as managing what the developers can do on these servers. Furthermore, the ability to provide automatic backups for the servers is also offered.

6.2 Flutter - Database

The Flutter-Application uses Drift as its database manager and, therefore, SQLite as its actual database. Drift controls the database, allowing access to and manipulation of data in Dart code without the need to know how to use SQL queries.

It is used not only because it is easy to set up, but also because it is simple to maintain and manage. Furthermore, it can be used on any operating system without the need for any additional configurations or technologies.

6.2.1 Setup

To set this up, the libraries mentioned in the Flutter technologies section must be installed. Without them, not all of the necessary functionalities will be provided. A Dart file can now be created to define the structure of the database, which is then automatically generated using the `build_runner` package.

If the structure and the tables of the database need to be changed, the process can simply be repeated, and the database will be updated when generating it. In the generated applications, the general process is the same, with the structure of the database being provided through the users' configurations.

6.2.2 Structure

Drift databases are used in `TableWizard` as well as the applications that are generated. The generated applications requirements vary from application to application. Because of this, the database schema is always unique.

On the other hand, the `TableWizard` application itself has a simpler structure. Since its database only needs to store user and project data, the structure remains fairly simple. It consists of only a few tables, as shown in figure 6.1. The connections seen betweens these tables are one to many connection. This means that every `module` has one `project_id` and that every `project` has an `user_id`

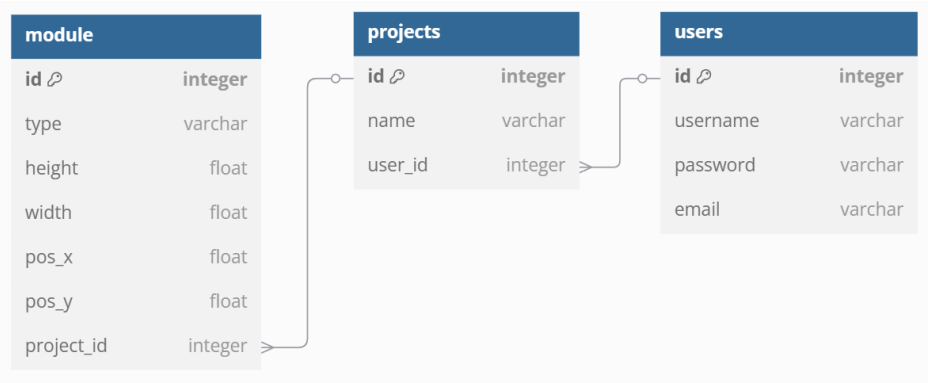


Figure 6.1: Flutter database structure

The `users` table, shown in figure 6.2, stores user-relevant data such as `email`, `name`, and `password`. It is important to note that while the first two are stored in plaintext, the `password` is stored in a hashed form for security reasons. This is done with the `SHA-2` algorithm who simply turns the password into an long and nearly irreversible string. The table also includes an `id` that uniquely identifies each user.

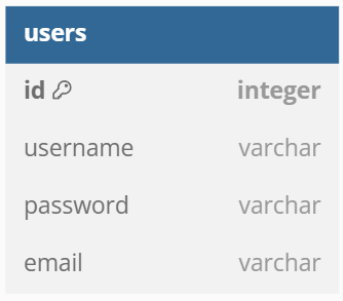


Figure 6.2: Users table

As shown in figure 6.3, this table stores the projects. Each project consists of a `name`, an unique identifier (`id`), and the `user_id`. The `user_id` refers to the user who owns the project. A user can have multiple projects, but a project can have only one user.

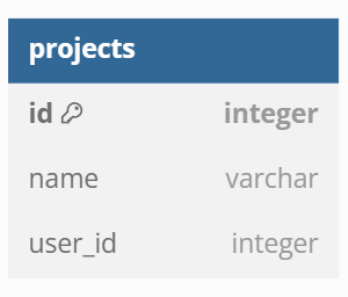


Figure 6.3: Projects table

Modules are the building blocks of projects, as explained in the basic functionality section. Those that are placed are saved in the modules table (figure 6.4). The table consists of a unique `id` as well as the `project_id`. Just like the relationship between `users` and `projects`, a `project` can have multiple `modules`, but a `module` can belong to only one `project`. The table also contains all the attributes concerning the `module`, with only a subset being represented in the picture for overview purposes.

Figure 6.4: Modules table

6.2.3 Functionality

The database has two main operational areas. Firstly, `TableWizard`, where it is simply used to store user and project data. Secondly, the generated applications, where the structure and functionality depend on the users' requirements.

The exact structure of the `TableWizard` database is explained in the structure section above. Due to the simplicity of the database, only rudimentary CRUD (**c**reate, **r**ead, **u**pdate, and **d**elete) functionalities are used. These are mainly used to update and store data required for the users' projects. This primarily covers managing the modules of the projects and inputting and outputting user and project data. These instructions are generated at the same time as the database and can then be used to interact with the database. The generation process is mentioned in the setup section. Because the data storage is so simple, no additional, more complex, custom queries need to be used.

Generated applications have unique requirements, so the database functionalities cannot be generated in advance. The same applies to the structure. Rudimentary CRUD commands are created anyway, but more specific functionalities are built depending on the users' demands.

6.3 Flutter - Software Architecture - Tightly Coupled Architecture

A tightly coupled architecture is a programming style where the different layers, such as the front-end and back-end, are interdependent. This means that business logic and the user interface are connected and cannot operate without each other. So, when something is changed in the design of the application, it is likely that changes will also need to be made in the logic of the program. An example of what this could look like can be seen in figure 6.5. There, the arrows show the dependencies between the different parts of the program.

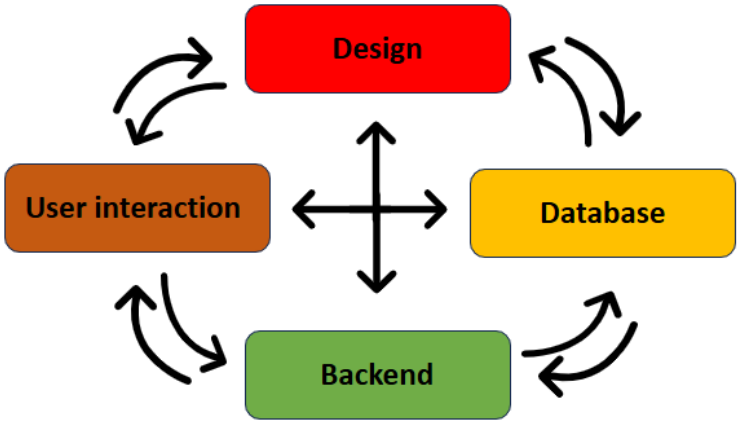


Figure 6.5: Tightly coupled architecture visualized

To fall under the categorization of tightly coupled, not every single part of the application has to be reliant on another. A system like the one seen in figure 6.6 would also be considered tightly coupled and is far more likely to be seen in actual development.

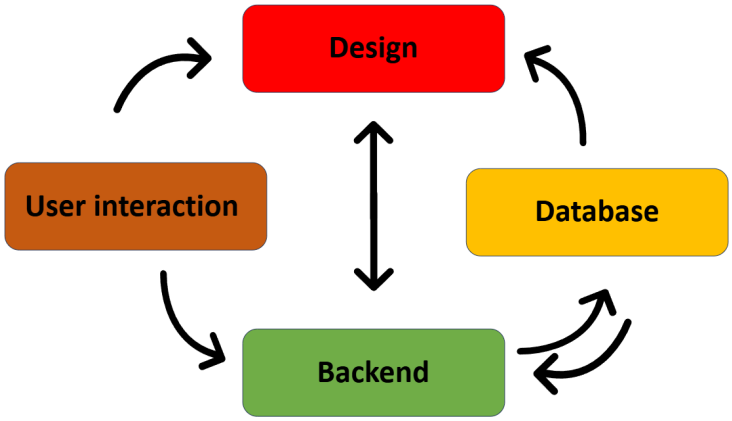


Figure 6.6: Tightly coupled architecture visualized

6.4 Flutter - System logic

The system logic refers to the underlying processes, functionalities, and rules by which an application operates. It ensures that the application works as intended and fulfils all the requirements, essentially being the `brain` of the application.

6.4.1 State Management

Notifiers handle the state management of the application. State[52] refers to all of the current data, actions, and settings in an application and must be updated when something changes to ensure everything functions as intended. While Dart handles some basic updates on its own, the more complex the software gets, the more has to be done manually.

For example, when something happens, like a widget being added to a list, it isn't displayed instantly. For this to happen, the state of the application must be updated so that the frontend is rendered again. Therefore, widgets are either marked as `listeners` or `senders`, with multiple widgets having the ability to be both `senders` and `listeners`. Widgets that will be added to the list are marked as `senders`. When they are added, they send a notification to the `notifier` with all of the needed data. This then sends a notification to the list, which is marked as a `listener`, that then adds the widget, with the data that was sent and updates its state.

In the case of `TableWizard` there are three notifiers. Those are the main method of managing the state of the application. They are used to:

- Manage the insertion of modules. This means it gets added to the workspace if dropped there or deleted if dropped outside the workspace.
- Manage the selection of modules. Meaning when a module gets selected, its attributes are displayed in the inspector.
- Manage the attributes of modules. When attributes in the inspector are changed, the modules change in accordance.

6.4.2 Module system

There are two different types of modules. The modules provided to build the application, those are the owns on the left side of the screen in the module tab, as seen in figure 4.2, and the ones that make up the inspector.

Building modules

A building module consists of a multitude of widgets with different functionalities. At the outermost level is a `base widget`. This manages the appearance of the widgets as well as the functions to drag them while they are still in the modules window. It is provided with a `type` so it knows which module it should represent. Types are things like buttons or tables. A system like this works because all of the widgets have the same functionality while they are in the modules window. When it is dragged to the workspace and dropped, it creates a `generated module` that already contains the correct position, size, and type. This is then sent to the notifier, which then provides the information to the workspace to append the module. As soon as it is appended, the workspace updates its state to display the new widget on the screen.

In the workspace, the `generated module` holds a container that manages the draggability and layout of the module inside the workspace. Inside of that container is the actual `widget` used for generation. It is mainly hidden as what is displayed in the workspace is the `generated module`. The purpose of the final `widget` is to hold all the information needed for the generation process. This is done to make it easier to extract the information from the module during generation, while still maintaining simple usability in the application itself. The layout is shown in figure 6.7.

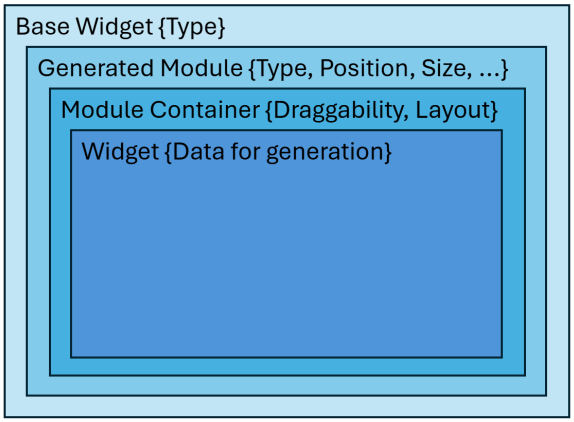


Figure 6.7: Building module structure

Such a system allows for a simple differentiation between the appearance and functionality of the module while it is still in the module window, versus when it is appended to the workspace. Furthermore, it provides a rather fast process for developing and implementing new modules, as the base structure and containers already exist. Therefore, only the unique functionalities of the new modules need to be created. It also makes it simple to extract the data from the widget for code generation, as it is mostly decoupled from the design and inner workings of `TableWizard`. To achieve this, each module gets an extra `Description` attribute, which is automatically filled out by the software. This attribute describes every

part of the module that is important in the generation process.

Inspector Modules

The `inspector modules` follow a different approach. Because they are only in the inspector window and cannot be moved around or changed while the application runs, they are much more simplified. Each of these modules essentially has its own unique structure. Functionality and appearance are coupled together because their features are mostly unique and cannot be contained in a single `base module`. Unifying them would quickly become overcomplicated.

Each `building module` type has its own unique list of `inspector modules`. When this list is created, the modules are assigned a type. Through this type, they know what input they accept. For example, whether they handle numbers or text.

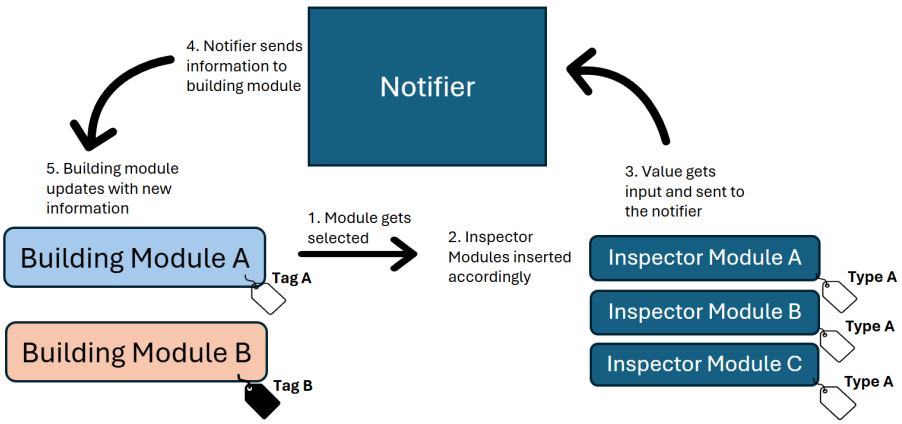


Figure 6.8: Relationship between inspector and building modules

When a `building module` is selected it appends the required `inspector modules` to the inspector. If now a new value is inserted into one of the `inspector modules`, they send the information to the corresponding notifier. This has to be done because the `inspector modules` do not know to which `building module` they belong. So they simply send their data to the notifier with a type that corresponds to their widget. The notifier than searches for the correct module and forwards the information to the correct `building module`, which updates its contents and state according to the data provided. A notifiers' main task is to link the `inspector modules` list to the `building modules` and transfer the data between them. In short, they are simply used to style and change the `building modules`. Figure 6.8 shows this entire process.

If a module in the workspace, or the workspace itself, is pressed, the list of `inspector modules` is displayed in the inspector window. If the user clicks outside the workspace, the inspector is cleared.

6.4.3 Code and Database Generation

As soon as the user has built an application they are satisfied with, they can start the generation. This process provides them with the code as well as the database.

Code Generation

The code for the final application can be generated at any time. When it is started, the folder structure and dependency files are copied. Dependency files in Flutter manage the libraries that are used, as well as some operating system-specific data. They are not made specifically for each application, as a few pre-written files are used for all of them. This can be done because they don't slow down the final application, even if there are things in them that are not actually used.

As already stated, each module has a description with which all the attributes can be retrieved with a few simple lines of code. Every module would already possess the ability to retrieve all the arguments needed, even without the description. However, having the description makes it much easier and faster to work with. Not only because it is unified for all modules, but also because it requires only a single line to get all the attributes.

These attributes are then inserted into the pre-written templates. Because of the way Flutter is written, the general structure of a module can simply be written and then filled with the attributes afterward. This also applies to the general structure of an entire application. Templates refer to the already existing structure of Flutter code. When everything is inserted, the module templates are put together into the application template.

```
class Generator(Description description, String application) {
    String Container = "Container(
        color: ${description.color},
        width: ${description.width},
        height: ${description.height},
    );";

    application += Container;
}
```

Listing 6.1: Pseudo code generation

A pseudo example is shown in the listing 6.1. To clarify, that is not actual working code, as this would be too long and complicated to include, but it still resembles a quick overview of how the templates are used. The example above shows a `Generator` that gets the `Description` of the container, as well as the rest of the `application`. A new `String`, which is simply text, with the name of `Container` gets created, and is filled with the attributes from the description. Then, the `Container` is simply attached to the rest of the application. In reality, a template consists of hundreds of lines of code and also includes logic that is needed for its creation.

A process like this has the advantage that the structure of the application, as well as the amount and types of modules used, does not matter. It is also relatively fast, as the software only has to execute a few simple instructions. However, it should be noted that as soon as the requirements become more complex, such as unique logic for completely custom buttons, this system may not be sufficient. For that reason, for now, only buttons with pre-defined functionality can be used.

Database Generation

This works similarly to the code generation. Needed information, such as the number of rows, the data type of the table, and the number of columns, are all stored in the description of the tables in use. The database then already has a pre-written layout, which is filled with

the attributes from the description. The same goes for the database operation: only the basic create, read, update, and delete functionalities are created, as anything more complex is extremely difficult to realize. When all the templates are filled, the database is completely generated before the entire application is downloaded.

6.5 PAP - Database

The PAP application uses Adminer, as the web application, to manage the SQLite database. It is accessed through PHP code in the application itself, which is used to write SQL statements.

Adminer is used because the setup is simple and fast, and it integrates seamlessly with Plesk and SQLite. It also works on every operating system automatically without issues.

6.5.1 Structure

The application needs a fairly complex set of tables to function properly. Everything is connected to everything, and therefore gets chaotic, as seen in figure 6.9. These connections are one to many connections. For example, each `training` has one `group`, while a group can have multiple trainings. However, the database can be divided into multiple table groups. These are the `trainings`, `trainers`, `users` who are the trainees, and `groups`.

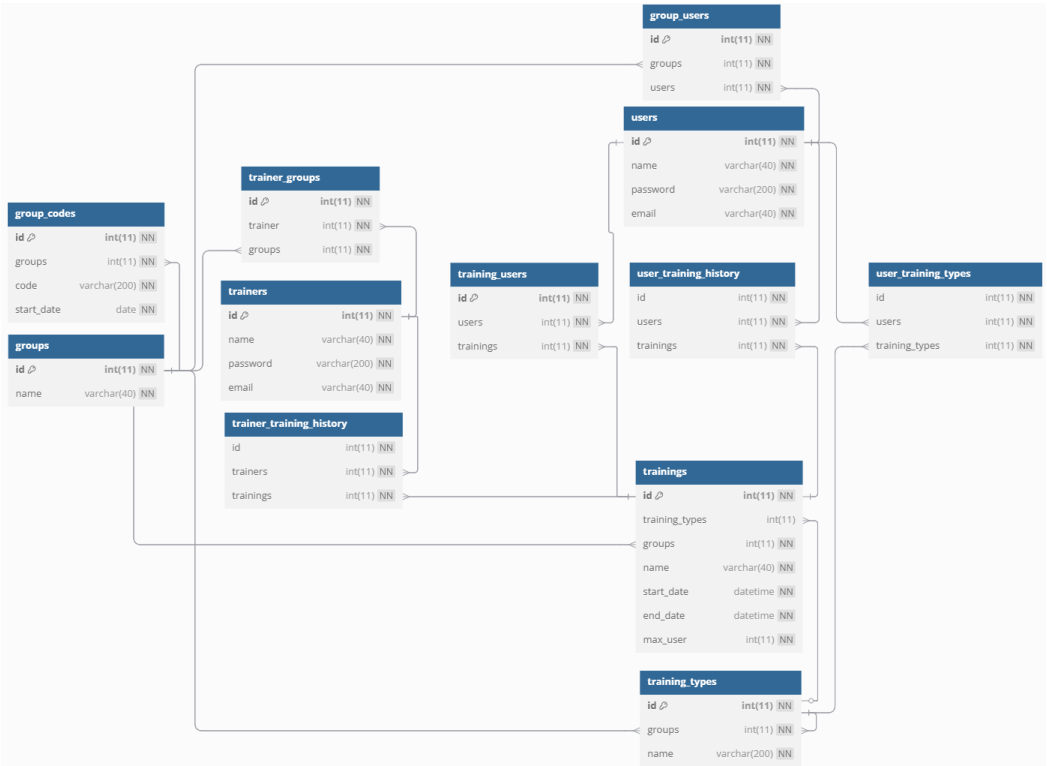


Figure 6.9: PAP database

The `groups` section mainly consists of the tables `groups` and `group_codes`, as presented in figure 6.10. A `group` simply consists of an unique `id` and a `name`. Each `group` can have

multiple `group_codes` (from now on referred to as `codes`), while each `code` can only be used for a single `group`. These `codes` are made up of an `id`, the `code` itself, which is simply a unique string of symbols, and a `start_date`. Trainers send the `codes` to the trainees so they can join the group by entering the `code`. This `code` is then deleted from the table so it cannot be used multiple times. The `start_date` is used to calculate how long the `code` is active. After a few days, the code expires and cannot be used anymore.

Figure 6.10: Groups

The `training` section is also made up of two tables: `Trainings` themselves and `training_types`. A `training_type` has a `group` it belongs to, a unique `id`, and a `name`. These `types` can be used to mark a `training`, which is made up of the said `training_type`, as well as an `id`, `name`, `start_date`, `end_date`, `max_users`, and the `group` it is held in. The `start_dates` and `end_dates` mark when the training is held and also its duration. `max_users` is, as the name implies, the maximum number of visitors a training can have. Each `training` can only be held in one `group` and can only have one `type`. It should also be noted that, due to the fact that each `training_type` can only belong to a single `group`, this `type` only works in the `group` it was created in. All of this can be seen in figure 6.11.

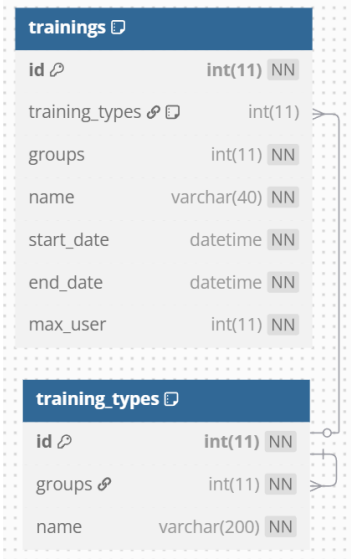


Figure 6.11: Trainings

`Trainer_groups`, `trainers`, and `trainer_training_history` make up the `trainer` section, as seen in figure 6.12. A `trainer` itself consists of an `id`, a `name`, a `password`, and a `email`, with the `id` and `email` being unique for reliably identifying the trainers. The `trainer_groups` table saves which `trainer` owns which `group`. Even though each entry consists of a `trainer` and a `group`, a `group` can only have a single `trainer`. A `trainer`, on the other hand, can have multiple `groups`.

It should be noted that the tables are designed in a way that, in the future, a `group` could be managed by multiple `trainers`. So, the current restriction is implemented with logic in the background, not in the database design itself, which would be possible.

The `trainer_training_history` (now referred to as `history`) saves all the `trainings` a trainer has held. To do this, each entry consists of a `trainer` and a `training`.

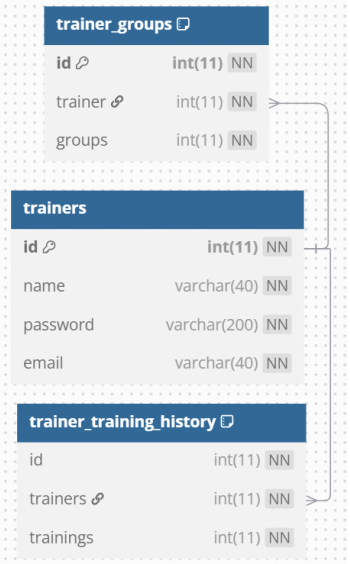


Figure 6.12: Trainers

By far, the largest section is the `user` section, shown in figure 6.13. It is made up of the `users`, `training_users`, `training_user_history` (from now on referred to as `history`), `user_training_types`, and `group_users`. A `user` has the attributes `id`, `name`, `password`, and `email`.

The `group_users` table stores the information about which `user` is in which `group`. For this, it is made up of an `id`, a `group`, and a `user`. A `user` can be in multiple `groups`, and a `group` can contain multiple `users`.

Information about which `user` has which `training_type` is stored in the `user_training_types` table. Each `user` can have multiple `types`, and each `type` can have multiple `users`. In order for that to be true, the table has to consist of a `user`, a `training_type`, and a unique `id` for identification.

Roughly the same way works the `training_users` table. It is used to store information about which `user` currently has booked which `trainings`. So the table has the attributes `users` and `trainings`, as well as an `id` for identification. `Users` can have booked multiple `trainings` at the same time, and `trainings` can have multiple `users` attached to them.

The `history` consists of `trainings` and `users`, as well as an `id` for identification. It is used to save the progress of the `users`, with the same rules from the `training_users` table applying.

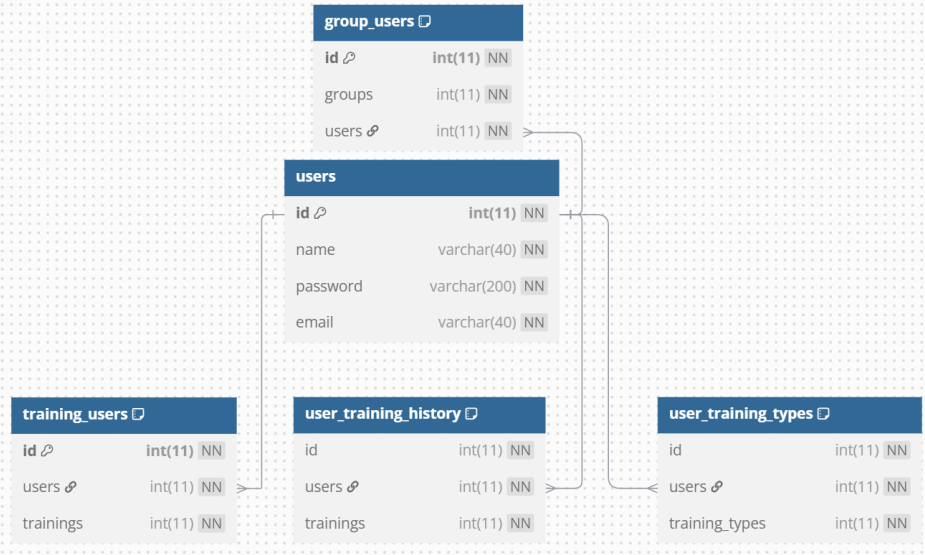


Figure 6.13: Users

Even though the tables `users` and `trainers` consist of the same attributes, they have separate tables. This is because they have different privileges and can perform different actions in the application. To make the process of identifying them simpler, they have their own tables. The same applies to things like `user_training_history` and `trainer_training_history`, which are nearly identical but are in separate tables to make managing the data more efficient.

It should also be noted that all the `passwords` are stored hashed for security reasons. For the Hash-Algorithm `SHA-2` is used.

6.5.2 Functionality

The database's functionality consists of CRUD, **c**reate, **r**ead, **u**pdate, and **d**elete actions. These form the foundation for building more complex queries. These queries not only complete a single task, like updating a value, but also look into other tables, check a few attributes, and then, depending on the results, update something. More complex queries like this can be used to simplify working with the database.

Even the simple CRUD queries are programmed by hand, but they are only written once and then reused instead of being written repeatedly.

6.6 PAP - Software Architecture - MVC

The MVC, **M**odel **V**iew **C**ontroller, is a widely-used software architecture. As shown in figure 6.14, it separates the application into three main parts: the `Model` and `Controller`, which contain the business logic, and the `View`, which is responsible for the user interface.

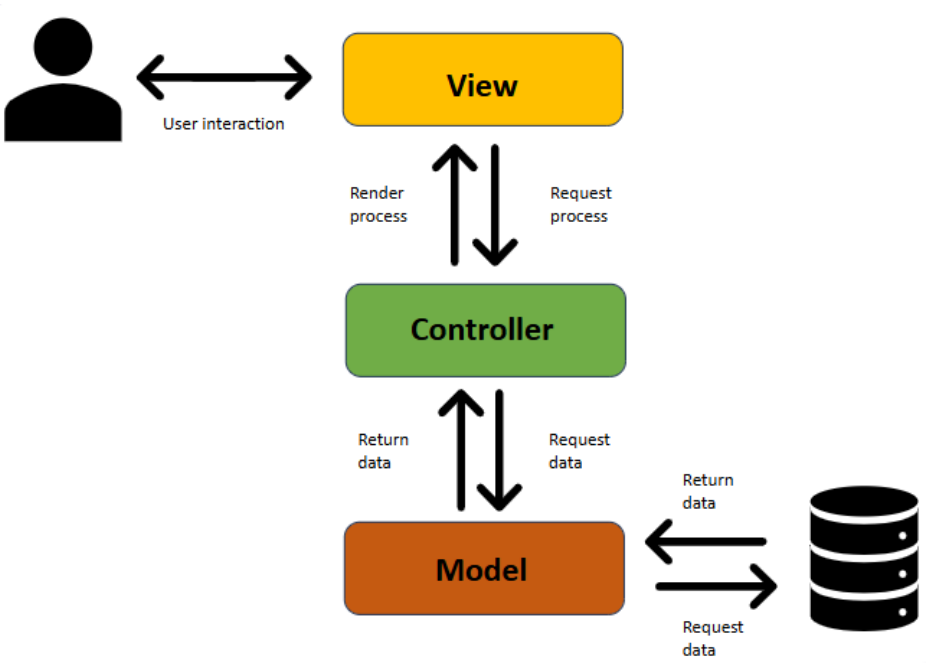


Figure 6.14: MVC architecture visualized

This idea was mainly taken from John Deacon's paper `Model-View-Controller Architecture` figure [53]. His ideas were adapted and rebuilt to fit the needs of the PAP-Application.

6.6.1 Model

The `Model` is used for retrieving and storing data in the database, as shown in figure 6.15. Requests to create, read, update, or delete data are sent from the Controller. These requests are then processed by the `Model`, and the response is sent back to the `Controller`. Without this request, the `Model` does not have the ability to access the database.
Example: In our web app, it manages data related to trainings and users, such as names, dates, emails, and so forth.

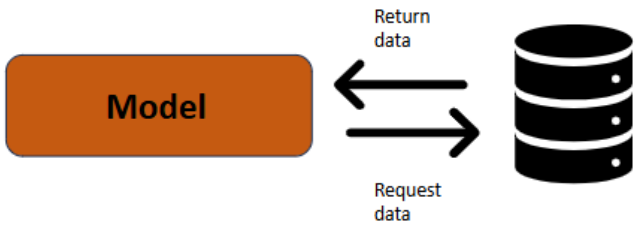


Figure 6.15: Model of the MVC

6.6.2 View

Interaction between the user and the website is handled by the `View`. It is responsible for displaying and updating the content of the website as well as managing user input, as shown in figure 6.16 following the paragraph. Also important to note is that the gathered information

isn't sent directly to the `Model`, but to the `Controller` instead. Furthermore, the data that should be displayed does not come from the `Model` itself. Communication occurs only through the `Controller`, thereby separating the user interface from the business logic.
Example: In our web app, it displays a list of trainings and provides buttons and input fields to book or update trainings.

Figure 6.16: View of the MVC

6.6.3 Controller

Communication between the `Model` and the `View` is handled by the `Controller`. It receives user input from the `View`, validates and interprets it, and then sends a request to the `Model`. After the `Model` sends back its response—whether it's data from the database or a simple acknowledgment that the operation was successful—the `Controller` sends the data to the `View` so it can update its content. The controller is shown in figure 6.17.
Example: In our web app, it is used to create or delete trainings or register a new user.

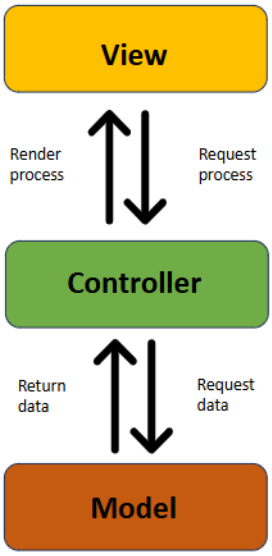


Figure 6.17: Controller of the MVC

6.6.4 Loosely Couple Architecture

A system where the different parts of the application are independent of each other is referred to as `loosely coupled`. MVC is one of the most commonly used loosely coupled architectures, with other examples being MVVM[54] and MVP[55]. The choice of which architecture to use highly depends on the tech stack—meaning the technologies being used—and the exact purpose and functionality of the application.

Building software with such an architecture offers various advantages.

- The separation means that the business logic and the user interface can be changed independently of each other
- Entire parts of a websites design can be replaced without the need to update the logic in the background and vice versa
- The codebase is highly maintainable and well-organized
- Parts of the View and the Model are easy to reuse
- Testing is simplified because the parts can be tested independently

In general cookies should only be used when storing data that is not security relevant. For relevant data `sessions` should be used as they are stored on the server on cannot be changed or accessed by the user.

6.7 Comparison

The comparison is not meant as a general comparison between the two architectures or databases. They are compared only in the scope of the `Flutter` and `PAP` projects and therefore project types like those. For that reason the insights given may not be transferable to other projects and ideas.

6.7.1 Server

The two of them use Plesk for their servers. This works seamlessly for both, with now problems arising. Not only for the web-server but also for getting all the needed certificates. Working fine from the beginning with small applications as well as with bigger apps. They still run smoothly and the websites are available all of the time.

6.7.2 Database

Between the two database setups, Drift with SQLite and Adminer with SQLite, are not two many differences. Both of them work on every operating system, are fast, easy to setup and work without problem with the chosen technologies.

Adminer has the advantage of a powerful, easy to use web-interface. Which, in the case of the PAP-Application with its somewhat complex database structure, helps a lot. On the other hand does the Flutter-Application not really need this interface due to its simplicity in database structure.

So for more complex databases Adminer is definitely the better choice, simply because of the web-interface, even for a Flutter application. While still, Drift works pretty well for simpler structures.

6.7.3 Architecture

The two architectures are different from the core up. While MVC tries to separate the different concerns of an application as much as possible, the tightly coupled architecture

does the exact opposite. In short was the tightly coupled architecture, for many reasons, the wrong decision for the Flutter project. The code base gets too confusing and implementing and changing features, or working with the code in general gets too complicated. But still both of them bring advantages as well as disadvantages with them.

MVC

- MVC keeps a great overview over the application. This not only includes the code but also the file structure of the application. Because everything is separated in their concerns and their functionality, searching for features is really simple. Especially if it is a larger and more sophisticated program.

  For smaller applications, this structure and separation can have the exact opposite effect. One can quickly loose track of things if the structure itself is more complicated than the code itself.

  For that reason it is the right choice for the PAP-Application and would have been by far the better choice for the Flutter application. While finding things in PAP is easy and takes no time, finding certain things in Flutter can be a real pain.

- Roughly the same goes for enhancing existing code. Finding things in the Flutter app often takes longer than actually implementing a new feature. While it is all neatly separated in PAP, the actual functionalities in Flutter are often all over the place and cannot be changed in a single place.

  That being said, while starting out, it took a really long time to write things in PAP. Separating functionalities takes a lot of extra code, especially in the beginning, while the first features were implemented in no time with Flutter. But as the projects grew bigger, the extra work in the beginning paid of, as now implementing new code is not only faster but also simpler in PAP.

- Changing existing features is also a pain in the Flutter-Application. While it went fast and without problem while the application was small, now it takes forever. It is not possible to simply change code in a certain place as it is scattered everywhere.

  The exact opposite was the case with PAP, at first changing code was pretty cumbersome but as it grew it got more and more comfortable. Now entire parts of the application can be changed without interacting with the larger code base.

- Catching and fixing errors is fast and easy. To see where they generate and what causes them can, through the structure and independency of the features, be really simple. And furthermore can you be sure that fixing the error won't cause any problems in any unforeseen places.

Tightly coupled architecture

- At the beginning the overview over the application is really good. The code base is small and everything is in one place, so finding things at the start is really simple.

  But as soon as the project gets larger, the structure gets really confusing and finding things gets really hard. So it really was the wrong choice for the Flutter-Application in this case.

- Changing and implementing new features, while again in the beginning being really easy and fast, is now really hard. To create a new functionality, the code has to be

changed in many different places that all depend on each other. Sometimes even the Front-end has to be modified even when the changes should only effect the Back-end.

This also creates a lot more risk to create new bugs and can easily destroy things that worked previously but depend on the code that has been changed.

- Fixing and finding errors is overall rather exhaustive. One can never be sure were exactly they generated and if fixing them causes any other problems somewhere unrelated in the code.

Summary: Overall can tightly coupled architecture work really well when creating smaller applications. There it is faster, programmers have a better overview and features can be changed or newly implemented faster as with MVC.

As soon as the code base gets larger MVC is the clear favourite though. Everything is better structured and in the long run, working with the code is easier. This goes not only for implementing new features, but especially for changing up existing code.

Bug fixing is easier in small as well as in large applications. Seeing and tracing back were the errors come from is easier and fixing them without having the worry about other code is an experience that the tightly coupled architecture doesn't provide.

6.8 PAP - System logic

The system logic consists of the underlying processes, functionalities, and rules that govern how an application operates. It ensures that the application functions as intended and fulfils all the requirements, essentially acting as the brain of the application.

6.8.1 Real-Time Data Management

Real-time data management refers to the information that is saved and used only while the user is on the website. In the case of `TableWizard`, as with most websites, this is done with `sessions` and `cookies`. They offer and easy, fast and, if done right, secure way to store temporary user data.

An example for that would be when a trainer or member selects a group. The `cookies` save the id of the group that has been selected. So when a trainer creates or a user books a training the database knows for which group to store the data. They also have the advantage that they can be stored for a longer period of time. This means even when the users closes the browser, the `cookies` may still be saved. The duration of how long they are saved can be adjusted by the developer.

Example: Someone logs into a website and closes the tab. They are still logged in after reopening the site.

But this opens up some security issues. Cookies are stored on the client, this means in the browser of the user and can therefore be viewed an altered by them. This might not seem like a big deal but a hacker could alter the cookie in a way that they delete a group they actually do not own. Luckily there are several ways to prevent this. First would be to use `sessions`. They are saved on the server and can therefore not be accessed by the users. So they can be used to store more secure data. Though this comes with some minor inconveniences. They

are a little bit more time intensive to implement and are cleared as soon as the browser closes.

Example: Someone logs into a website and closes the tab. After reopening the site they no longer are logged in and have to login again.

In the case of `TableWizard` cookies and sessions would be used with:

- Storing the chosen group
- Storing chosen user
- Storing the current user email
- Storing the type of user (admin, trainer, member)

6.8.2 Static data management

Static data management refers to the usage of data that is stored in the database. There are multiple ways to access and sent data.

For transferring the data from the back-end to the front-end and vice versa `AJAX` and `web sockets` are used. An example would be displaying the `groups` a trainer or member has. `AJAX` would send a request, which contains the user id is sent to the controller. It than unpacks the request and forwards it to the database. The answer is than sent back, again by the controller, and the data can than be used by the front-end.

Example: In the case of the groups, there is a request send every two seconds to display the up to date list.

This has some upsides as well as some downsides. First of all it is not real-time. Meaning that if, for example, a group gets added, the front-end has to wait until a new request is sent until it can display the new data. Furthermore the frequent requests do not only use more resources but can also slow down the webpage as well as slow down the internet in general. So in general `AJAX` is used for on-time request when loading a site or one time actions. A more detailed description look at the article from IBM about the AJAX[4].

For the more frequent request `web sockets` are used. They establish a persistent connection with the server. The connection than stays open until closed by either side. Through that the data transfer is real-time. Meaning changes get updated immediately and the front-end can display them instantly. This also reduces internet and resource usage as information is only exchanged if something is actually updated. For a more complete explanation look at the article from `Vishal Rana`[47] about web sockets.

Example: In the case of the groups, only a request is sent when a group is updated, deleted, changed.

`Web sockets` have some downsides though. They are harder to implement and therefore also take up far more time until completely implemented. Furthermore, they are really ineffective for one time requests. Therefore `TableWizard` uses mixture of `web sockets` and `AJAX` to manage the transfer of data.

WebSockets are used with:

- Getting trainings

- Getting groups
- Getting members
- Getting histories of trainers and members

AJAX is used with:

- Sending data when logging in
- Sending data when creating groups, trainings
- Sending data when updating trainings, groups
- Sending data when deleting trainings, groups

6.8.3 Authentication

Authentication includes registering and logging into the website. With both of these the user can choose between being a trainer and a member. This can simply be down by checking one of the boxes at the bottom of the side. As is shown in figure 6.18.

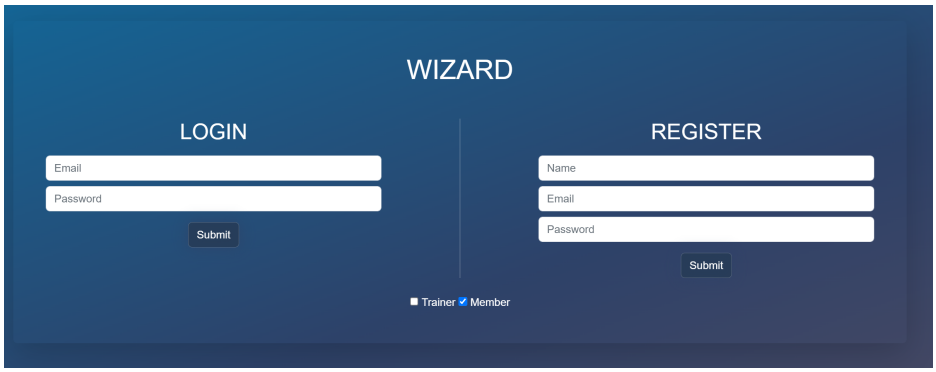


Figure 6.18: Authentication page

To register the user putts in their `email`, which has to be unique as it is used to identify them, their `password` and their `name`. It is important to note that when creating an trainer account, the same email can still be used to create a member account.

When logging in they simply put in their `email` and `password`. Than the `Submit` button is pressed. It sends an AJAX request to the server and if the data is correct the user gets logged in or registered. If the wrong data gets input, a message as in figure 6.19 is displayed in red to inform the user about it.

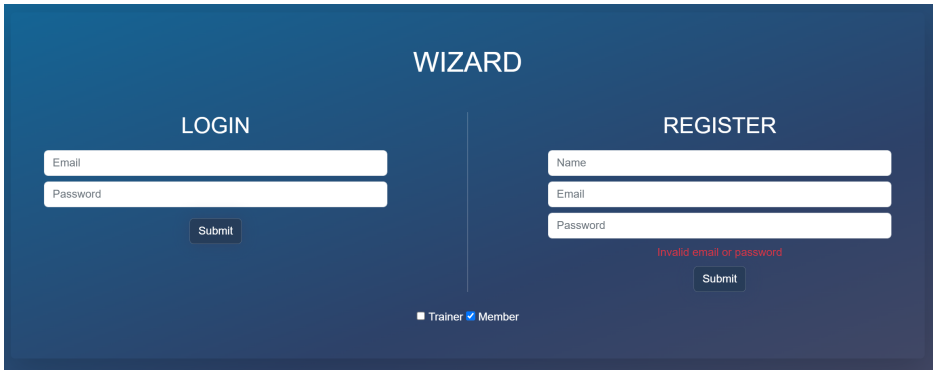


Figure 6.19: Authentication page wrong input

After submitting the data, the fields get cleared independent of if the input data was correct or not.

6.8.4 User management system

The user management system refers to the interaction and handling of `trainers`, `members` and `groups`. With trainers and members being users and groups being the structure that contains but is also managed by the users. Groups overall do not have permissions or actions as one of the other two have, as it purely works as a container.

After authentication sessions save if the user is a trainer or a member. Through them the web-app decides which pages to show. Because of that a member cannot access pages that are only mend for trainers and vice versa. This enhances security and ensures that members cannot suddenly change trainings or remove other members.

Trainers them selves have various abilities with which they manage the groups, trainings and members. These are:

- Create, delete and update groups, trainings and training types
- Add and remove group members
- View their own and their users training history
- View training details like who and how many visit a training

It has to be said that they only have these abilities in the groups they own. They cannot check the history of a random user and cannot create or delete trainings in any other group.

Members on the other hand only have the ability to book and dismiss trainings as well as view their own training history. Their general permissions are held low to increase the security aspect.

All of these actions are performed with `AJAX requests` and `web sockets` as mentioned before.

6.8.5 Training and booking system

Trainings are created by the trainer. They are created for a certain group and can only be accessed and managed in that group and can than be booked by the members. After all the information about the trainings is input, the trainer presses `submit` to send the data to the database via `AJAX`. With that the training is added. The creation of a training is shown in figure 6.20.

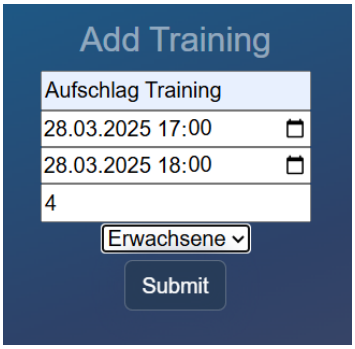


Figure 6.20: Training creation

The bookings and the training history of the trainers and members can than be viewed in extra interfaces. This are shown in the figures 4.9a, 4.9b and 4.9c. Trainings are simply stored in the database and stay there as they are needed for the history. All of the communication in these processes is handled with `AJAX` or `web sockets`.

6.8.6 Security

Security is important to ensure that data cannot be accessed or changed in any way by people that do not have permissions to do so. The `OWASP coding practices list`[50] is implemented to the extend needed.

Input validation

Checking and validating user input is one of the easiest way to make a page more secure. The things that are checked for are mainly symbols used in SQL-Scripting or other scripting languages.

To do this the input text first is trimmed, simply meaning that all of the white-spaces get removed. Than they are compared with a regular expression, regex for short, that contains the symbols the code looks for. A regex is a search pattern that describes which patterns and symbols are okay and which are not. An example for a regex can be seen in listing 6.3. The code takes the user input and compares it to a regex that forbids symbols like backslash and hashtags. It returns either false if no such symbols are in the input or true if there are, checking if there are any dangerous symbols in the code.

Most important is that all of the validation processes run on the server and not on the client. The client is the browser of the user who currently visits the website. If that is done, than there is no way for the user to evade those checks.

```
function injectionCheck(input) {
    const sqlInjectionPattern = /['";--/*%_()=|<>`\\]/;
    return sqlInjectionPattern.test(input);
}
```

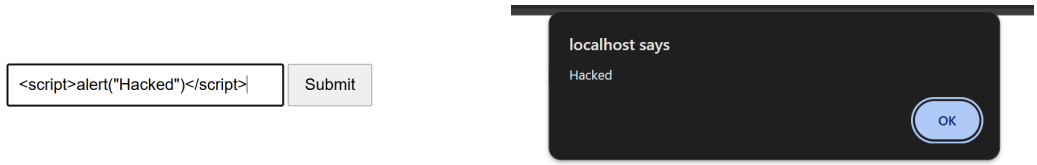
Listing 6.2: Regex to check for SQL-Injection

Before than storing the data to the database it is checked that it is not empty and that the data has the correct type for the database.

Sanitizing requests

Sanitizing the requests roughly functions the same way as the input validation. The only difference being that characters are not forbidden but converted to their `XML entities`. For example a < gets turn into `<`. The article `XML entities` from `PortSwigger`[56] explains what exactly these entities are.

When they would not be escaped, commands input on the website could execute automatically. This is shown in the pictures figure 6.21a where a JavaScript command to make an `alert` with the value `Hacked` is input. In the picture figure 6.21b this alert is executed. For that reason certain symbols are escaped to prevent such attacks.



(a) Inputting an alert command

(b) Command gets executed

Figure 6.21: Alert gets executed due to security issues

This is also done in the database to ensure no code gets executed and hackers do not have access to data they should not.

```
function escapeHtml(unsafe) {
    return unsafe
        .replace(/&/g, "&amp;")
        .replace(/</g, "&lt;")
        .replace(/>/g, "&gt;")
        .replace(/"/g, "&quot;")
        .replace(/'/g, "&#039;");
}
```

Listing 6.3: Sanitize requests

Authentication

When authenticating an user it is important to check the password, username and such for multiple things. First, as already stated, the input validation to check for any dangerous symbols. Then if the email and password are formatted correctly. An mail has to contain and `@` and have a ending like `.com`. The password has to contain `upper` and `lower` letters as well as `symbols` and `numbers`. It has to be said that the symbols that can be used for the password are those left over after the validation.

Secure cookies

As mentioned before, cookies can be used to store user data. This introduces security risks as they can be changed by the user. There cookies have options to make them more secure by setting certain flags when creating them. These flags are:

- **HttpOnly:** Through this flag cookies cannot be accessed with JavaScript code. With that, no dangerous data can be sent to the database as all the messages are sent with JS.
- **Secure:** This makes it that the cookies are only sent over HTTPS. This ensures that no data is sent over insecure connections like HTTP. The article `Comparison between HTTP and HTTPS` from `AWS`[57] covers the topic HTTP vs HTTPS in detail and perfectly explains why it is importuned to use HTTPS.
- **SameSite:** This flag can be set to three different modes `Strict`, `Lax` and `None`. When `None` is used, cookies saved on one site can be used by any other website as well. `Lax` means that the cookie can only be used on the site it is created and on sites the user is referred to by links. At last `Strict` means the cookie can only be used on the website it is created, without exception.

  By setting the flag as strict as possible, it can be ensured that no attacks can be committed from foreign websites.
- **Duration:** The duration states how long the cookie lasts until it expires, stated in seconds. This is very useful to ensure that user data is not saved forever.

In general cookies should only be used when storing data that is not security relevant. For relevant data `sessions` should be used as they are stored on the server on cannot be changed or accessed by the user.

6.9 Comparison

The comparison is not meant as a general comparison between the two architectures or databases. They are compared only in the scope of the `Flutter` and `PAP` projects and therefore project types like those. For that reason the insights given may not be transferable to other projects and ideas.

6.9.1 Server

The two of them use Plesk for their servers. This works seamlessly for both, with now problems arising. Not only for the web-server but also for getting all the needed certificates. Working fine from the beginning with small applications as well as with bigger apps. They still run smoothly and the websites are available all of the time.

6.9.2 Database

Between the two database setups, Drift with SQLite and Adminer with SQLite, are not two many differences. Both of them work on every operating system, are fast, easy to setup and work without problem with the chosen technologies.

Adminer has the advantage of a powerful, easy to use web-interface. Which, in the case of the PAP-Application with its somewhat complex database structure, helps a lot. On the other hand does the Flutter-Application not really need this interface due to its simplicity in database structure.

So for more complex databases Adminer is definitely the better choice, simply because of the web-interface, even for a Flutter application. While still, Drift works pretty well for simpler structures.

6.9.3 Architecture

The two architectures are different from the core up. While MVC tries to separate the different concerns of an application as much as possible, the tightly coupled architecture does the exact opposite. In short was the tightly coupled architecture, for many reasons, the wrong decision for the Flutter project. The code base gets too confusing and implementing and changing features, or working with the code in general gets too complicated. But still both of them bring advantages as well as disadvantages with them.

MVC

- MVC keeps a great overview over the application. This not only includes the code but also the file structure of the application. Because everything is separated in their concerns and their functionality, searching for features is really simple. Especially if it is a larger and more sophisticated program.

  For smaller applications, this structure and separation can have the exact opposite effect. One can quickly loose track of things if the structure itself is more complicated than the code itself.

  For that reason it is the right choice for the PAP-Application and would have been by far the better choice for the Flutter application. While navigating and finding things in PAP is easy and takes no time, finding certain things in Flutter can be a real pain.

- Roughly the same goes for enhancing existing code. Finding things in the Flutter app often takes longer than actually implementing a new feature. While it is all neatly separated in PAP, the actual functionalities in Flutter are often all over the place and cannot be changed in a single place.

  That being said, while starting out, it took a really long time to write things in PAP. Separating functionalities takes a lot of extra code, especially in the beginning, while the first features were implemented in no time with Flutter. But as the projects grew bigger, the extra work in the beginning paid of, as now implementing new code is not only faster but also simpler in PAP.

- Changing existing features is also a pain in the Flutter-Application. While it went fast and without problem while the application was small, now it takes forever. It is not possible to simply change code in a certain place as it is scattered everywhere.

  The exact opposite was the case with PAP, at first changing code was pretty cumbersome but as it grew it got more and more comfortable. Now entire parts of the application can be changed without interacting with the larger code base.

- Catching and fixing errors is fast and easy. To see where they generate and what causes them can, through the structure and independency of the features, be really simple. And furthermore can you be sure that fixing the error won't cause any problems in any unforeseen places.

Tightly coupled architecture

- At the beginning the overview over the application is really good. The code base is small and everything is in one place, so finding things at the start is really simple.

  But as soon as the project gets larger, the structure gets really confusing and finding things gets really hard. So it really was the wrong choice for the Flutter-Application in this case.

- Changing and implementing new features, while again in the beginning being really easy and fast, is now really hard. To create a new functionality, the code has to be changed in many different places that all depend on each other. Sometimes even the Front-end has to be modified even when the changes should only effect the Back-end.

  This also creates a lot more risk to create new bugs and can easily destroy things that worked previously but depend on the code that has been changed.

- Fixing and finding errors is overall rather exhaustive. One can never be sure were exactly they generated and if fixing them causes any other problems somewhere unrelated in the code.

Summary: Overall can tightly coupled architecture work really well when creating smaller applications. There it is faster, programmers have a better overview and features can be changed or newly implemented faster as with MVC.

As soon as the code base gets larger MVC is the clear favourite though. Everything is better structured and in the long run, working with the code is easier. This goes not only for implementing new features, but especially for changing up existing code.

Bug fixing is easier in small as well as in large applications. Seeing and tracing back were the errors come from is easier and fixing them without having the worry about other code is an experience that the tightly coupled architecture doesn't provide.

7 Front-end

The front-end is the visual part of an application. It covers the design of an app, as well as animations and basic user interaction.

7.1 Flutter

Flutter provides powerful design capabilities even without extra libraries. It handles both the structure as well as the actual style of an application.

This also extends to user interaction. Basic functionalities like buttons or text fields are already provided and can be used right from the start.

7.1.1 Design

Flutter makes it really easy and comfortable to design applications. It has base designs that automatically adapt to the operating system being used. These designs automatically change and handle much of the styling work for developers. An example of this can be seen in figures 5.1a and 5.1b, respectively. Both pictures show two widgets, each of which automatically received its design from Flutter, adjusted to the operating system.

```
class Button() {
    ButtonStyle buttonStyle = ButtonStyle(
        backgroundColor: WidgetStatePropertyAll(Colors.blue),
        shape: WidgetStateProperty.all(
          RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(0.2),
          ),
        );

    ElevatedButton(
          style: buttonStyle,
        );
}
```

Listing 7.1: Flutter button style

Also, creating new designs from scratch is simple. Flutter provides a lot of design templates that can easily be filled out and attached to widgets. An example of this would be a new button design, as seen in listing 7.1. A new `ButtonStyle` is created that sets the `background color` to `blue` and gives the button a `rounded edge`. It is then applied to the button itself, which takes over the new design. `ElevatedButtons` are just buttons that already have certain designs implemented. Importantly, this design does not change for every operating system but instead stays the same. Figure 7.1 shows the button that was styled in listing 7.1.

Figure 7.1: The button designed in 7.1

The `TableWizard` Flutter-Application mostly implements custom designs. Common designs were created once and reused in as many cases as possible. Some unique designs exist, such as the background behind the workspace, for example. While the common ones are saved separately and can be imported into any files, the unique ones are created within the widgets themselves.

This approach not only reduces code duplication but also keeps the codebase tidy and mostly focused on functionality.

7.1.2 User interaction

User interaction is written in the same programming language as the actual design. The logic is implemented, like the style, by appending a new tag. In this case, the button in listing 7.2 demonstrates how this works. `onPressed` means that the code inside the curly brackets gets executed when the button is pressed. It increases an `int`, which is a simple number, by one.

```
class Button() {
    ButtonStyle buttonStyle = ButtonStyle(
        backgroundColor: WidgetStatePropertyAll(Colors.blue),
        shape: WidgetStateProperty.all(
          RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(0.2),
          ),
        );

    int i = 0;

    ElevatedButton(
          style: buttonStyle,
          onPressed: () {
            i += 1;
          },
        );
}
```

Listing 7.2: Example for user interaction in Flutter

This process is the same with nearly all widgets who take care off user interaction. Only the

tags, which in listing 7.2 would be the `onPressed` tag, are changed to fit the required input mechanism.

7.2 PAP

The PAP-Application is designed with HTML and CSS/Bootstrap. HTML provides the base structure to the app, while CSS, with the help of Bootstrap, creates the actual style. Bootstrap itself is an add-on that provides pre-written designs and templates for a faster and more comfortable design experience. This also makes it easy to maintain a unified look for the entire application. Alternatives to Bootstrap would be `Pure CSS`[58], `Tailwind CSS`[59], and `Materialize`[60].

Bootstrap was chosen because it is easy to set up, free, and provides many good-looking, pre-written templates. Altering and creating templates is also relatively simple.

7.2.1 Design

Applications are designed with HTML and CSS. First, the structure, as seen in listing 7.3, of an app is built with HTML. The code snippets show a button with the content `Click me` and a text field with the text `Enter something...`. An `id` attribute gives an object a name. With this name, the object can then be referenced throughout the application. In this case, the button is given the id `btn`, and the text field is given the id `textField`.

```
<!DOCTYPE html>
<html>
<body>

    <button id="btn">Click me</button>
    <input type="text" id="textField" placeholder="Enter something...">

</body>
</html>
```

Listing 7.3: HTML application structure

This code produces a plain page that can be seen in figure 7.2. In the layout, the button is positioned next to the text field. At this stage, neither the button nor the input field has any design or styling applied.

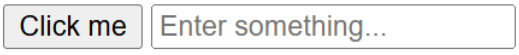


Figure 7.2: Page structured with HTML in 7.3

This design gets added by applying the Bootstrap templates written in CSS, as seen in 7.4. These templates can be added to an object by simply applying certain tags to it. In this

case, the button and the text field are centered and placed in a section that is lifted up from the rest of the screen. Additionally, the button is now positioned above the text field.

```
<!DOCTYPE html>
<html>
<body>
    <div class="vh-100 d-flex justify-content-center align-items-center">
        <div class="container shadow-lg p-5 rounded text-center">
            <div class="row">
                <div class="col">
                    <button id="btn" class="btn btn-success mt-2">Click me</
                        button>
                </div>
            </div>

            <div class="row justify-content-center">
                <div class="col-auto">
                    <input class="form-control my-2" style="width: 300px;" type=
                        "text" id="textField" placeholder="Enter something...">
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

Listing 7.4: Introduce CSS to HTML structure

Such a Bootstrap template can be seen in listing 7.5. At the top is the `name` or `tag` of the template that can be applied to parts of the HTML. In this case, `hr`, which specifically targets already existing `hr` elements. An `hr` is a simple horizontal rule (break line) used to separate contents.

Then, in the curly brackets, follow the `attributes` that describe the objects. These `tags` can be used in any file on nearly any object, ensuring great reusability. `margin` creates space around the object. It does this depending on the font size and only on the top and bottom, as specified by `1rem 0`. `color: inherit` ensures that the element gets its color from the parent object, which would be the object it is appended to, such as a container. `border: 0` removes any default border settings. A custom border setting like `border-top: 1px solid` can then be applied, adding a solid one-pixel border at the top of the `hr`. Finally, `opacity: 0.25` makes the element somewhat transparent.

```
hr {
  margin: 1rem 0;
  color: inherit;
  border: 0;
  border-top: 1px solid;
  opacity: 0.25;
}
```

Listing 7.5: Bootstrap template

The finished page can now be seen in figure 7.3.

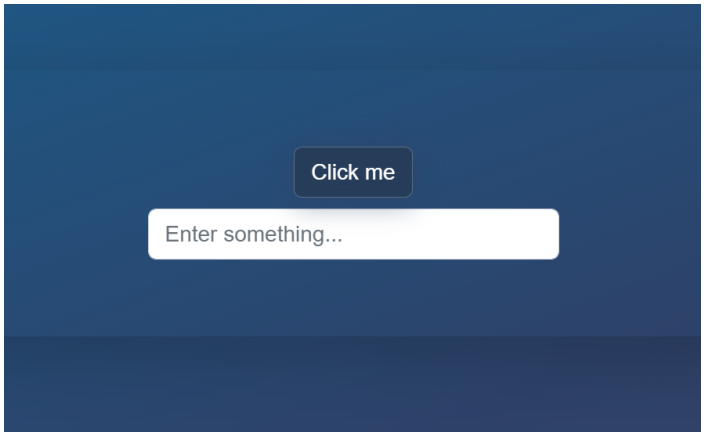


Figure 7.3: Page structured with HTML and styled with CSS in 7.4

7.2.2 User interaction

While the structure and the design are made with HTML and CSS, the actual user interaction is written in JavaScript. This interaction is done in two main ways. The first one, shown in listing 7.6, is to write the function that is called directly in the HTML, with the trigger set to `onclick`. In this case, it simply increases a number by one when the element (such as a button) is clicked.

```
<!DOCTYPE html>
<html>
<body>

    <button id="btn" onclick="increaseCounter()">Click me</button>

    <script>
        let count = 0;
        function increaseCounter() {
            count++;
        }
    </script>

</body>
</html>
```

Listing 7.6: Calling function in HTML

In the second approach, the trigger type and functionality are only defined in JavaScript. This is shown in listing 7.7. The code finds the HTML object by its tag and quickly saves it to the `button` variable for easier use. It then adds an `eventListener` that checks if the button is pressed, and when this happens, it simply increases a number by one.

```
<!DOCTYPE html>
<html>
<body>

    <button id="btn">Click me</button>

    <script>
        let count = 0;
        const button = document.getElementById("increaseBtn");

        button.addEventListener("click", function() {
            count++;
        });
    </script>

</body>
</html>
```

Listing 7.7: Calling function in JS

Normally, the JavaScript code would be written in a separate file, but to make it easier to see, it is directly appended to the HTML code. This is not wrong, but it is somewhat unusual, especially for larger applications.

7.3 Comparison

This comparison focuses on the differences between the design processes and functionalities of the two technologies. Some of the experiences described here are subjective and may not be applicable to everyone.

- Due to the architectural choice with the Flutter application, as mentioned in the Back-end section, the Front-end is somewhat dependent on the rest of the application. This leads to some problems that aren't present with PAP. When the Back-end is changed, the designs might be affected, and in some rare cases, even the other way around.
- Designing an application is a lot more comfortable with Flutter than with CSS/Bootstrap, mostly because of the pre-existing templates and how design is structured in Flutter. While Bootstrap provides templates, changing and reworking them is not only faster but also easier in Flutter. These templates can then be reused at any needed point.

  When creating unique designs, both of them produce extra code and are, in this form, quite similar.
- One thing that is better with CSS/Bootstrap is the fact that the Flutter widgets can get overloaded with designs. While they are simple tags with CSS, as the designs are saved in different files, in some cases, the whole design is saved in the widgets with Flutter. This leads to more code being present that does not have any functionality and can make finding things a lot more difficult.

  Even when the CSS code is saved in the same file as the HTML, it has its own section and doesn't affect the code structure like Flutter does.

- In terms of reusability, both of them are rather similar. The two systems work really well and do not have any stark benefits or drawbacks when compared to each other.
- User interaction goes to Flutter. With both applications, it is easy to use and pretty simple to set up. As long as the user interaction stays simple, Flutter spares a few code lines. But as soon as the application gets large enough and the user interaction gets more complicated, PAP handles it better.
- There is no need to install extra libraries for Flutter, as the base design capabilities are really good. While this is also possible with plain CSS, it takes a lot of time and effort, so installing an add-on like Bootstrap is somewhat necessary.
- With Flutter, there is no need to switch between different languages for design and logic. This makes for a better flow while programming.
- HTML and CSS also do not have the ability to change their look depending on the operating system, primarily because they are used exclusively in web development, while Flutter can be used on every system.

Summary: Overall, Flutter's system is partially better when it comes to designing the front-end. While it sometimes can lead to somewhat unstructured code, the benefits outweigh the drawbacks. The design process is simpler and offers more styling abilities without the need for additional add-ons, and user interaction is implemented more easily—above all, because there is no need to switch between different languages while programming.

This also goes for user interaction. The process of creating functionalities and dependencies through it is somewhat simpler, mostly because the user interaction stays rather simple, and therefore a few lines of code are spared, which the PAP application needs.

Furthermore, Flutter's base capabilities are stronger than those of CSS, which nearly always requires an add-on like Bootstrap, making the setup much more comfortable with Flutter.

That being said, the differences are pretty minimal and are rather nitpicks than choice-altering problems. Purely for design, Flutter would probably be the better choice, but taking into account the Back-end, the PAP technologies are just more scalable and work better for large applications.

8 Evaluation

The evaluation goes over the tasks which the final application should be able to perform, as mentioned in the introduction. It states how they were implemented and provides proof, where possible, in terms of screenshots from the application.

Creating, cancelling or rescheduling training sessions

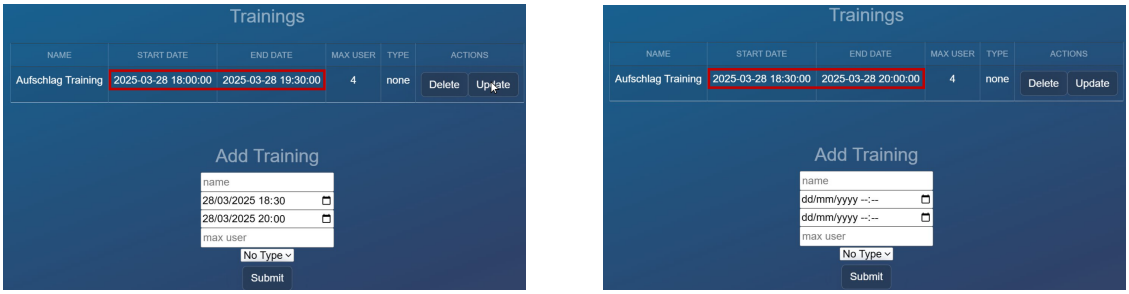
As seen in the figure 6.20, a training is simply created by entering all the information about it and than submitting it. After that all members with the same tag as the training can book it. It has to be said that if the training isn't given a type, everybody can book the session. Cancelling a training, as shown in figures 8.1a and 8.1b, can be performed by pressing the *delete* button next to the training.

(a) Cancelling training

(b) Training cancelled

Figure 8.1: A training gets cancelled

The figures 8.2a and 8.2b show how a training sessions gets rescheduled. The trainer simply enters the new information and presses the `update` button next to the training they want to update.



(a) Rescheduling training

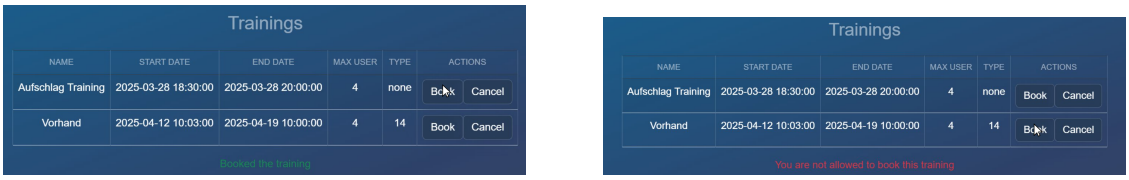
(b) Training rescheduled

Figure 8.2: A training gets cancelled

Managing who is allowed to attend which session

This has been done by implementing a training type. The type, also called tag, can be applied to a training as well as a trainee. As shown in the figure 8.3a both have the same tag

and the trainee can therefore book the training session. In the figure 8.3b they have different tags and the trainee cannot book the training.



(a) Trainee can book training

(b) Trainee cannot book training

Figure 8.3: The training and the trainee have the same tag

Checking who and how many people have booked a training

As seen in the figure 4.8 this has been implemented with a simple list that is displayed when selecting a training. It shows an overview of who exactly booked the training session as well as how many booked it.

Adding or removing trainees from a group

The process of adding a member can be seen in the figures 4.6a, 4.6b and 4.6c. At first the trainer generates a code, sends it to the user, who than enters the code, and with that, joins it.

If they want to remove a member, as shown in figures 8.4a and 8.4b, they can simply press the `remove` button next to the member. After it is pressed the user is, not only, removed from the group but also from all the trainings they have booked in said group.

(a) Member is till in group

(b) Member has been removed

Figure 8.4: The trainer removes a member from the group

Checking the training history of trainers and trainees alike

The problem was solved by creating different overviews for trainers and trainees. While the trainer has the ability the check his own history as well as the history of his trainees, the trainees themselves can only see their own history. This is shown in the figures 4.9a, 4.9b and 4.9c.

9 Project Management

As this is a one person project the project management mostly consists of writing a To-Do list and than working through it.

9.1 Planning

The planning process can be divided into two parts. To-Dos and work that was coming up the entire time and the establishing of the milestones at the beginning of the project.

GitHub helped to manage the To-Do list. It provides a service called *Projects* which can be used to create a table for the tasks with the ability to separate them in any way needed.

Though the milestones were put into an annual plan made in excel. It has to be said that they switched as it was clear that a new software (the PAP-Application) has to be built.

1. **Flutter:** Finish modules, 18. September 2024
2. **Flutter:** Finish rendering, 22. November 2024
3. **Flutter:** Finish inspector, 27. December 2024
4. **Flutter:** Finish project/DA, 21. February 2025

Those were partially aborted and simply set to *finish to project as fast as possible* when the building the PAP-Application was the new goal. The first one was completed and the second and third one were only partially done. While the last one was than refocused on the PAP app.

9.2 Evaluation

Overall it worked out pretty well. Because this was a one person project, there was no discussion about who is responsible for what and the things were simply done. Thorough that the entire process went pretty smooth.

On the other hand was there sometimes to problem with setting unrealistic goals and deadlines as there was no team mate to consult with, if the plan is even possible.

9.3 Time sheet

Date	Tasks	Home	School
5. Sep. 24	Overwork project profile, start planning project		9
12. Sep. 24	creating ToDos and milestones, start programming basis		9

19. Sep. 24	creating first modules and finishing planning		9
26. Sep. 24	finish basics for moduls TextField, TextBox and Button		9
30. Sep. 24	Bug fixes for the first modules	3	
1. Okt. 24	More bug fixes	1	
3. Okt. 24	Taking Flutter course for deeper understanding		9
4. Okt. 24	Gathering ideas for the generation program	2	
7. Okt. 24	Finishing first modules, better sizing for workspace	6	
8. Okt. 24	Making plans and researching Flutter tables	2	
10. Okt. 24	Bug fixes all over, planning and starting website	4	9
13. Okt. 24	Continue Website	3	
14. Okt. 24	Creating ability to manually resize workspace	4	
17. Okt. 24	Bug fixes workspace, starting selectable widgets		9
19. Okt. 24	Continuing selectable widgets	4	
20. Okt. 24	Bug fixes selectable widgets	1	
24. Okt. 24	Starting inspector, continuing table		9
26. Okt. 24	Bug fixes inspector and table	3	
27. Okt. 24	Continue inspector, continuing table	3	
28. Okt. 24	More functionallity to website	5	
30. Okt. 24	More functionallity to website, starting to restructure code	8	
1. Nov. 24	restructering modules, better layout for all modules	3	
2. Nov. 24	finish module restructure, bug fixes modules	4	
4. Nov. 24	Continue Website	2	
7. Nov. 24	Starting to set up webserver, simulating webserver setup	1	9
8. Nov. 24	Creating funtionality to copy/create files for code generation	3	
11. Nov. 24	Frontend for TdoT	4	
14. Nov. 24	Setting up Webserver, poster for TdoT		9
15. Nov. 24	Finish frontend for TdoT	2	
18. Nov. 24	Making plan for DA structure	2	
19. Nov. 24	Bug fixes for TdoT	3	
21. Nov. 24	Finish poster, creating first DA structure, function for TdoT		9
24. Nov. 24	Functionality for TdoT	4	
28. Nov. 24	Finishing TdoT design and functionality		9
30. Nov. 24	Rebuilding inspector	6	
1. Dez. 24	Rebuilding inspector and bug fixes inspector	6	
2. Dez. 24	Starting system overview DA	2	
4. Dez. 24	Major bug fixes	3	
5. Dez. 24	Starting rendering, trying out dependecies for rendering		9
6. Dez. 24	TdoT	7	
8. Dez. 24	Starting templates for rendering	2	
9. Dez. 24	Rethinking project	1	
12. Dez. 24	Enhancing templates and rendering dependecies		9
13. Dez. 24	Creating automatic download, generation structure	5	
16. Dez. 24	PROJECT OVERHALL Sketches for new software	4	
19. Dez. 24	Trying out databases and database structure		9
20. Dez. 24	Setting up database	2	
25. Dez. 24	Creating database management infrastructure, Design ideas	4	

26. Dez. 24	Establish scope of function	2	
31. Dez. 24	Creating database tables and code connections	5	
2. Jan. 25	Basic creating, reading, update and deletion functions	4	
3. Jan. 25	Fixing table datatypes	2	
5. Jan. 25	Finishing database excess from code and bug fixes	4	
6. Jan. 25	Creating creating and deleting trainings functions	4	
8. Jan. 25	Login and register functionality	6	
11. Jan. 25	Reserving and canceling trainings	3	
12. Jan. 25	Bug fixes trainings functionality	2	
13. Jan. 25	Bug fixes login, Training types and usermanagement	5	
16. Jan. 25	Training types and usermanagement		9
17. Jan. 25	Bug fixes everything	1	
20. Jan. 25	Bug fixes database access	3	
21. Jan. 25	Starting design	3	
23. Jan. 25	Continuing design		9
27. Jan. 25	Finishing basic design	2	
30. Jan. 25	Starting history and stats		9
1. Feb. 25	Continuing history and stats	4	
4. Feb. 25	Starting presentation for JobInfoBörse	1	
6. Feb. 25	Starting security		9
8. Feb. 25	Quality of live changes, more security	5	
10. Feb. 25	more security, continue DA	2	
13. Feb. 25	Diplomarbeit		9
14. Feb. 25	Infonachmittag	5	
16. Feb. 25	More design and DA	4	
19. Feb. 25	Finishing project basis functions	6	
24. Feb. 25	Diplomarbeit	4	
27. Feb. 25	Diplomarbeit		9
28. Feb. 25	Diplomarbeit	2	
3. Mrz. 25	Diplomarbeit	3	
6. Mrz. 25	Diplomarbeit		9
7. Mrz. 25	Diplomarbeit	3	
10. Mrz. 25	overworked front-end	4	
14. Mrz. 25	Diplomarbeit	2	
20. Mrz. 25	Diplomarbeit		9
21. Mrz. 25	Diplomarbeit	3	
22. Mrz. 25	Diplomarbeit	3	

Lukas Daxecker

Braunau/Inn, 28.11.2025 Daxecker Lukas _____

Place, Date *Signature*

10 Future Work

This chapter covers possible future enhancements to the applications, including improvements and fixes, as well as new features or functionalities.

10.1 Flutter

Because of the technical challenges, as already mentioned in the evaluation, the Flutter project will not be continued as such. Therefore, there is no real `Future Work`. But the idea might be rebuilt with other technologies that are more suited for the task.

10.2 PAP

All the wanted features are implemented, and the application is basically `ready to use`. But there still are a few enhancements and upgrades, as well as fixes, that could make the app better. The improvements below are ranked in importance of being implemented.

10.2.1 Code Restructure

The codebase in general should be overworked and simplified. It works and does its job, but is somewhat chaotic and needs some restructuring. A lot of functionalities could be implemented easier or more efficiently. This would increase the speed and reliability of the app, as well as make it easier to implement further features due to a better code overview.

10.2.2 Multiple Trainers per Group

The idea would be to have multiple trainers being able to manage a single group. This can be especially helpful if a group of trainers have the same students, or if a trainer has to be substituted.

10.2.3 Enhancing Group Invitations

Trainers should have the ability to enter one or multiple email addresses and then send the invitations via the app. This would remove the need for copying and pasting the code out of the app and into some external messenger, making it easier to add multiple students at once.

10.2.4 More Detailed Training Descriptions

The ability to add more detailed descriptions, such as what exactly is the plan or where the training is held (if there are multiple locations like in tennis). This would make it easier for the students to prepare for the training or to get to where they need to be.

10.2.5 Connecting to Calendar Applications

Giving the ability to connect to calendars, such as Google Calendar[48], to automatically save the dates of booked trainings. Even though they are saved in-app, some people might want an overview of all their appointments in one place.

10.2.6 Reward System

In-app achievements could help boost motivation to attend trainings. This would help people who have problems with motivation, as well as better visualize their progress.

10.2.7 In-App Messaging System

An in-app messaging system would be used to better connect trainers and trainees without the need for third-party applications. This way, trainees could ask the trainer questions regarding the trainings, and trainers could post news relevant to all trainees.

11 Related Work

The following companies have products with similarities to the solutions. Two examples are mentioned for both the Flutter and the PAP application. `Knack`[61] and `Glide`[62] for Flutter and `Vereinsplaner`[63] and `Spond`[64] for PAP.

11.1 Knack

Knack is a no-code development platform that allows users to create and control personalized databases and applications without programming knowledge. It is designed to assist companies and institutions manage structured data in an effective manner while providing an easy and friendly operating environment. The platform has features which include automation, reporting, and managing data securely. Due to its ability to integrate with other services, Knack is useful in many industries including healthcare, education, and construction, making it useful in meeting various administrative and operational requirements.

11.2 Glide

Glide is a no-code development environment that allows users to create mobile and web applications using structured data from Google Sheets. The platform automatically converts spreadsheet data into interactive applications, eliminating the need for coding expertise. It offers customization options for UI components, enabling users to tailor their applications to specific needs. With a focus on simplicity and accessibility, Glide is designed for individuals and businesses looking to rapidly develop functional apps without extensive development time.

11.3 Vereinsplaner

Vereinsplaner is a cloud-based software solution specifically designed for managing clubs and associations. It provides a range of features aimed at simplifying administrative tasks, including membership management, financial tracking, event organization, and communication tools. The platform includes both a mobile application and a web interface, ensuring accessibility for administrators and members alike. By automating routine processes and centralizing essential data, Vereinsplaner enhances efficiency and coordination within organizations of varying sizes.

11.4 Spond

Spond is a digital management tool tailored for sports teams, clubs, and activity groups. It facilitates organization through features such as event scheduling, automated invitations, attendance tracking, and integrated payment processing. The platform is particularly useful for team coordinators, coaches, and administrators seeking to streamline communication and event planning. Supporting a wide range of activities, including sports, music, and community events, Spond reduces administrative overhead while improving member engagement.

12 Conclusion

Even though there were setbacks and times of frustration, the project overall was a success. It gave me the opportunity to engage with technologies and systems, ranging from software development to server and database management, that otherwise I wouldn't have had time for. The project was incredible for learning new things and solidifying and enhancing the knowledge I have gathered in school. Furthermore, there is a satisfaction in seeing a project up and running that was worked on for over half a year.

Though, it has to be considered that there were things that I would do differently now. Mainly, thoroughly researching the technologies that are to be used and really thinking about the idea and how it is implemented before just starting to work.

But as said, overall, the project is a success and gave me the opportunity to educate myself in various technical fields.

Listings

List of Figures

Bibliography

[1] Amazon. What is sql (structured query language)? https://aws.amazon.com/what-is/sql/, 2025. [Technology].

[2] MDN. Javascript. https://developer.mozilla.org/de/docs/Web/JavaScript, 2025. [Technology].

[3] Gustavo Henrique Letério Da Silva Leite. Our brand guidlines. https://worldvectorlogo.com/logo/javascript-1, 2025. [Logo].

[4] IBM. What is ajax? https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=page-asynchronous-javascript-xml-ajax-overview, 2025. [Scientific Works].

[5] MDN. Css: Cascading style sheets. https://developer.mozilla.org/de/docs/Web/CSS, 2025. [Technology].

[6] Gustavo Henrique Letério Da Silva Leite. Our brand guidlines. https://worldvectorlogo.com/logo/css-3, 2025. [Logo].

[7] Bootstrap. Bootstrap. https://getbootstrap.com/, 2025. [Product].

[8] Bootstrap. Brand guidlines. https://getbootstrap.com/docs/4.0/about/brand/, 2025. [Logo].

[9] MDN. Html. https://developer.mozilla.org/de/docs/Web/HTML/Element, 2025. [Technology].

[10] Gustavo Henrique Letério Da Silva Leite. Our brand guidlines. https://worldvectorlogo.com/logo/html-1, 2025. [Logo].

[11] The PHP Group. Php. https://www.php.net/docs.php, 2025. [Technology].

[12] PHP Group. Download logos and icons. https://www.php.net/download-logos.php, 2025. [Logo].

[13] Google. Dart. https://dart.dev/, 2025. [Technology].

[14] Google. Our brand guidlines. https://dart.dev/brand, 2025. [Logo].

[15] Google. Flutter. https://flutter.dev/, 2025. [Technology].

[16] Google. Representing the flutter brand. https://flutter.dev/brand, 2025. [Logo].

[17] Flutter. webview_flutter. https://pub.dev/packages/webview_flutter, 2025. [Technology].

[18] dash overlfow. provider. https://pub.dev/packages/provider, 2025. [Technology].

[19] Flutter. image_field. https://pub.dev/packages/image_field, 2025. [Technology].

[20] miguelruivo. file_picker. https://pub.dev/packages/file_picker, 2025. [Technology].

[21] victoreronmosele. image_field. https://pub.dev/packages/web_color_picker, 2025. [Technology].

[22] loki3d. drift_dev. https://pub.dev/packages/archive, 2025. [Technology].

[23] Simon Binder. drift_dev. https://pub.dev/packages/drift_dev, 2025. [Technology].

[24] Dart. build_runner. https://pub.dev/packages/build_runner, 2025. [Technology].

[25] Simon Binder. sqlite3_flutter_libs. https://pub.dev/packages/sqlite3_flutter_libs, 2025. [Technology].

[26] Plesk. Plesk. https://www.plesk.com/, 2025. [Company].

[27] Plesk. Our brand guidlines. https://www.plesk.com/brand/, 2025. [Logo].

[28] SQlite. Sqlite. https://sqlite.org, 2025. [Technology].

[29] Opensource. Sqlite. https://www.sqlite.org/, 2025. [Logo].

[30] Simon Binder. Drift. https://pub.dev/packages/drift, 2025. [Logo].

[31] Jakub Vrána. Adminer. https://www.adminer.org/de/, 2025. [Logo].

[32] Microsoft. Visual studio code. https://code.visualstudio.com/, 2025. [Product].

[33] Visual Studio Code. Icons and names usage guidlines. https://code.visualstudio.com/brand, 2025. [Logo].

[34] Git. Git. https://git-scm.com/, 2025. [Product].

[35] Git. Logos. https://git-scm.com/downloads/logos, 2025. [Logo].

[36] Microsoft. Github. https://github.com/, 2025. [Product].

[37] GitHub. Logos. https://github.com/logos, 2025. [Logo].

[38] Apachefriends. Xampp. https://www.apachefriends.org/de/index.html, 2025. [Product].

[39] XAMPP. Apache friends xampp. https://www.apachefriends.org/index.html, 2025. [Logo].

[40] Google. Androidstudio. https://developer.android.com/studio?hl=en, 2025. [Technology].

[41] Google. Brand guidelines. https://developer.android.com/distribute/marketing-tools/brand-guidelines?hl=en, 2025. [Logo].

[42] Chiradeep BasuMallick. What are emulators? definition, working, types and examples. https://www.spiceworks.com/tech/devops/articles/what-are-emulators/, 2025. [Technology].

[43] Microsoft. Language server protocol. https://microsoft.github.io/language-server-protocol/, 2025. [Technology].

[44] T. Ylonen & C. Lonvick. The secure shell (ssh) transport layer protocol. https://www.rfc-editor.org/rfc/rfc4253, 2025. [Technology].

[45] Wikimedia. Unofficial ssh logo. https://commons.wikimedia.org/wiki/File:Unofficial_SSH_Logo.svg, 2025. [Logo].

[46] MDN. Usage of http-cookies. https://developer.mozilla.org/de/docs/Web/HTTP/Guides/Cookies, 2025. [Technology].

[47] MDN. The websocket api (websockets). https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, 2025. [Technology].

[48] Google. Google calender. https://workspace.google.com/intl/de/products/calendar/, 2025. [Product].

[49] OWASP. Owasp top ten. https://owasp.org/www-project-top-ten/, 2025. [Scientific Works].

[50] OWASP. Secure coding practices. https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/stable-en/, 2025. [Scientific Works].

[51] Flutter Docs. Automatic platform adaptions. https://docs.flutter.dev/ui/adaptive-responsive/platform-adaptations, 2025. [Illustration].

[52] Google. State mangement. https://docs.flutter.dev/get-started/fundamentals/state-management, 2025. [Technology].

[53] John Deacon. Model view controller architecture. http://www.johndeacon.net/john-deacon/articles/model-view-controller-architecture/, 2025. [Scientific Works].

[54] Microsoft. Model view view model. https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm, 2025. [Scientific Works].

[55] Microsoft. Model view presenter. https://learn.microsoft.com/en-us/archive/msdn-magazine/2006/august/design-patterns-model-view-presenter, 2025. [Scientific Works].

[56] Portswigger. What are xml-entities. https://portswigger.net/web-security/xxe/xml-entities, 2025. [Scientific Works].

[57] AWS. Http vs https. https://aws.amazon.com/de/compare/the-difference-between-https-and-http/, 2025. [Scientific Works].

[58] Yahoo! Pure css. https://pure-css.github.io/, 2025. [Product].

[59] Tailwind. Tailwind css. https://tailwindcss.com/, 2025. [Product].

[60] Materialize. Materialize. https://materializecss.com/, 2025. [Product].

[61] Knack. Knack. https://www.knack.com/, 2025. [Company].

[62] Glide. Glide. https://www.glideapps.com/, 2025. [Company].

[63] Vereinsplaner. Vereinsplaner. https://vereinsplaner.at/, 2025. [Company].

[64] Spond. Spond. https://www.spond.com/de/, 2025. [Company].

CV

Lukas Daxecker

Geburtstag, Geburtsort:	17.09.2005, Salzburg
Schulbildung:	VS Neukirchen a.d.E NMS Neukirchen a.d.E HTL Braunau
Praktika:	AMO GmbH, 4 Wochen, Elektronikfertigung Karl Weinhäupl GmbH, 6 Wochen, Metaller Wacker Chemie AG, 6 Wochen, Siliziumverarbeitung
Anschrift:	Sternstraße 2 5145, Neukirchen a.d.E Österreich
E-Mail:	daxecker.lukas@gmail.com