htblakaindorf
Leistung mit Menschlichkeit

informatik

Höhere Technische Bundeslehranstalt Kaindorf an der Sulm

Abteilung Informatik

Diplomarbeit

im Rahmen der Reife- und Diplomprüfung

Königskarte

Leon Edlinger
Paul Gigler
Andreas Weissl

5BHIF
2024/2025

Betreuer: Prof. DI Johannes Loibner, BSc
Projektpartner: Prof. DI Robert Müllerferli
Datum: 18.03.2025

Statutory declaration

I declare under oath that I have written the present diploma thesis independently and without outside help, have not used sources and aids other than those indicated and have identified the passages taken from the sources used literally and in terms of content as such.

Ort, Datum	Leon Edlinger
Ort, Datum	Paul Gigler
Ort, Datum	Andreas Weissl

Abstract

The ’Sternsinger Aktion’ is a customary fundraising event in which children perform door-to-door while soliciting donations. In Lieboch, it traditionally relied on manual, paper-based planning. While functional, it lacked the convenience and adaptability of digital solutions. This diploma thesis aimed to improve the process by digitalizing the planning and execution, making it more efficient and manageable.

Three tools were developed: an Admin Panel for address assignment, a mobile app for participants to view their assigned addresses, and a software component for managing data exchange between the applications and the database. The software adapts to changes, such as last-minute adjustments to group assignments.

This thesis introduces the technologies used and key features of the tools, including adaptive algorithms and real-time data integration, area border consideration for group assignments, and a focus on usability. Instructions for using the tools are also provided.

The developed solution was successfully tested with four participants in Lieboch. Overall, the digitalization of the ’Sternsinger Aktion’ was successful. This diploma thesis presents a reliable system that highlights the advantages of digital tools in event planning and lays a solid foundation for future improvements and expansion.

Zusammenfassung

Die Sternsingeraktion ist eine traditionelle Spendenaktion, bei der Kinder singend von Tür zu Tür gehen und Spenden sammeln. In Lieboch basierte sie traditionell auf einer manuellen, papierbasierten Planung. Diese war zwar funktional, bot aber nicht den Komfort und die Anpassungsfähigkeit von digitalen Lösungen. Ziel dieser Diplomarbeit war es, den Prozess zu verbessern, indem die Planung und Durchführung digitalisiert und damit effizienter und überschaubarer wird.

Es wurden drei Tools entwickelt: ein Admin-Panel für die Adresszuweisung, eine mobile App, mit der die Teilnehmer ihre zugewiesenen Adressen einsehen können, und eine Softwarekomponente für die Verwaltung des Datenaustauschs zwischen den Anwendungen und der Datenbank. Die Software passt sich an Änderungen an, so wie kurzfristige Anpassungen der Gruppenzuweisungen.

In dieser Arbeit werden die verwendeten Technologien und die wichtigsten Merkmale der Tools vorgestellt, darunter adaptive Algorithmen und Datenintegration in Echtzeit, die Berücksichtigung von Gebietsgrenzen bei der Gruppenzuweisung und ein Schwerpunkt auf der Benutzerfreundlichkeit. Darüber hinaus werden Anleitungen zur Nutzung der Tools bereitgestellt.

Die entwickelte Lösung wurde mit vier Teilnehmern in Lieboch erfolgreich getestet. Insgesamt war die Digitalisierung der Sternsingeraktion erfolgreich. Mit dieser Diplomarbeit wird ein zuverlässiges System vorgestellt, das die Vorteile digitaler Tools in der Veranstaltungsplanung aufzeigt und eine solide Grundlage für zukünftige Verbesserungen und Erweiterungen schafft.

Thanks

It would not have been possible to carry out this thesis to this extent without the active support of a number of people. We would therefore like to thank everyone who supported us in the implementation of this thesis, especially our supervisor Prof DI Johannes Loibner, BSc for the intense cooperation, our project partner Prof. DU Robert Müllerferli that provided us the initial idea and the test users, Lisa and Kathrin Müllerferli, Jasmin Kormann and Eva Reßmann for the constructive feedback.
…

Table of Contents

1 Introduction

Mobile apps are utilized for virtually all aspects of daily life in the modern world. So after we noticed that there is no application that allows the efficient planning of campaigns like the "Sternsinger-Aktion", we asked ourselves why, and furthermore, how hard it is to create an app with intuitive usability with the main purpose of simplifying the process of managing such a campaign and gaining a general overview of the progress made by the groups.

The app needs to comply with specific criteria we defined in cooperation with Prof. DI Robert Müllerferli. He is the main organizer of the campaign in the parish of Lieboch and helped us to work out the key aspects our project should implement. In the finished product, every user should be able to scan a QR-Code, through which the area of this group gets assigned to the device. These areas must be dynamically adjustable, so an admin can coordinate the workload of each area more efficiently. The areas also need to be clearly visible by an outline that gets drawn through "Border" addresses. We implement an algorithm to calculate these border addresses. It should be visible at a glance if there is a "specification", which can be assigned by admins and set for an address. We should achieve this by employing distinct icons rather than the default one. Apart from the app itself, we also implemented a web portal through which administrators can manage and supervise the campaign.

The investigative aspect of this thesis will focus on how components should behave and appear, so that new users can use this tool without requiring a long "onboarding" phase. Interacting with elements should feel familiar, and the limits of what users can and cannot do need to be clearly defined. Our application needs a reliable data source to ensure the consistency and accuracy of marked addresses, so we researched ways to keep our database up to date with minimal manual intervention. After defining the project requirements, we identified a need to determine which addresses qualify as border addresses.

In our context, an area consists of multiple addresses, each with a defined location represented by latitude and longitude coordinates. Border addresses are the ones that form the outer boundary of an area. For example, given five addresses with the following coordinates:

- A (0,0)
- B (2,0)
- C (0,2)
- D (-2,0)
- E (0,-2)

In this case, addresses B, C, D, and E are border addresses because they outline the area, enclosing A at the center. Thus, we explored different algorithms for this task, compared them in terms of efficiency, selected the most suitable one, and implemented it.

This thesis contains an in-depth description of our thought and development process, as well as the steps we took to achieve our goal of a functional mobile application that can be used by volunteers during the "Sternsinger-Aktion 2025," which took place in the parish of Lieboch in January 2025.

The result of this thesis should be a mobile app that provides users with the addresses that they need to visit on this day. They then should be able to easily mark the houses they already visited. If something unusual occurs at this address, the user should be able to record it, ensuring that the organizers are aware of it and can account for it in the future.

1.1 Team

Leon Edlinger
The Admin Panel was developed by Leon. He researched how adaptive algorithms and real-time data integration can be used to continuously update address databases and ensure accuracy in an app designed for mobile response teams.

Paul Gigler
Paul designed and developed the mobile application and was responsible for the deployment of our software. Additionally, he acted as the team-leader and was responsible for maintaining the contact with the end users. He researched on how user experience principles can contribute to an intuitive map display for charitable activities involving people with different technical expertise.

Andreas Weissl
The task of Andreas was developing the backend server. He researched how modern algorithms for geodata analysis and route optimization can be used to improve the efficiency and coverage of the area distribution in the caroling campaign.

2 Management of the project

In this section, the managing process and structure will be described.

2.1 Overview

We managed our project using the scrum methodology. After all requirements were defined, we came together as a team and decided on a rough timeline for our project. We defined the following milestones:

30.06.2024: designed database relations

30.07.2024: designed UI of mobile app

30.07.2024: designed UI of admin panel

30.10.2024: implemented backend

30.10.2024: implemented mobile app

30.11.2024: implemented admin panel

15.12.2024: first tests through end users

Following this plan, we had enough time to implement all features and were able to still make changes in the final days before deployment.

2.2 User stories

We also defined multiple different user stories that we implemented over the course of multiple sprints, some of them we will list in the following.

- As a user, I want to see an input address that I need to visit highlighted on a map based on OpenStreetMap.
- As a user, I want to be able to select an address and write a comment for it.
- As a user, I want to be able to check addresses that I visited.

- As a user, I want to be able to differentiate special addresses from normal ones I need to visit on the map. (Homes that should be left out, Place to eat for the day)

We made use of a "Scrum-Lite", which was more suitable for this project, as we often discussed the current progress in breaks between lessons, rather than having a dedicated meeting every day. Not strictly following all scrum guidelines enabled us to benefit from scrums' advantages without adding organizational overhead. We also had regular meetings with our supervisor to also keep him up-to-date with on the progress we made. This was also beneficial to us because of the slight pressure it provided.

3 Technologies

Development would not have been possible without making use of many tools, frameworks and environments. This chapter will provide a brief description of each tool that we used to create our software.

3.1 Frontend

3.1.1 Dart

Dart is a programming language initially designed for web development, with the goal of replacing JavaScript. Today, it is used in a variety of software products, mainly because of the Flutter framework. Dart can be compiled for many platforms and architectures, including ARM, x64, RISC-V, JavaScript or WebAssembly and is highly popular for its combination of high-level features and practical language features like garbage collection and optional type annotation. It was developed by Google and is now an open-source project. [Bie19]

Fig. 1: Dart Logo [25m]

3.1.2 Flutter

Flutter is an open-source software development framework. It allows programmers to compile their applications for different platforms including Web, macOS and iOS, as well as Windows and any type of Linux-based systems, all from a single codebase, written in Dart. This allows for more efficient and faster cross-platform development. Another benefit of Google's toolkit are the highly customizable, predefined UI components. Developers can mix and match these components as needed, making them an applicable choice. [25n][Dag19]
We chose Flutter mainly for these reasons, but also because of our previous experience with Java, to which Dart is quite similar. Through it, we were able to get started quickly and learn what we needed along the way. Having a design through the components was also very helpful and saved us some time.

Fig. 2: Flutter Logo [25c]

3.2 Backend

3.2.1 Java Spring

Java on its own is an object-oriented programming language that is designed to be platform-independent. It allows programs to run on any device that has a Java Virtual Machine installed. Java is particularly useful for its strong security features and extensive community support.

Fig. 3: Java Logo [17]

Java Spring is an open-source framework based on Java. The Java Spring Framework is an in-depth programming and configuration language that simplifies the development of Java applications. The most used Spring Framework is Spring Boot. Spring Boot simplifies the application process with not much need for preconfigured logic. Other Spring Frameworks that have a huge impact on Java programming are Spring Security for authentication/authorization and Spring Data for database injection.
We chose Java Spring as our backend development tool because of its simplicity and flexibility. As a framework, Spring allowed us to develop a well-scalable and maintainable backend to generate APIs and connect to the database.

3.2.2 PostgreSQL

Fig. 4: PostgreSQL Logo [23a]

PostgreSQL is an open-source relational database management system. It is known for its reliability and scalability. It complies with the SQL standard. Databases, which use PostgreSQL, are designed to handle a huge amount of data efficiently. These databases support more advanced features like full-text search and JSON data storage. One of PostgreSQL's strengths is its flexibility. Developers are able to create custom data types and functions to customize the database to their specific needs. It also provides security mechanisms, which include role-based access control and encryption.

3.3 Version Control

3.3.1 Git

Git is a distributed VCS that was developed by Linus Torvalds in 2005. The main benefit of a VCS is to easily keep track of different versions of files. Git is the most widely used option concerning this area of application. [25ah] This is due to the fact it is open-source, therefore free, as well as reliable and easy to learn. It gets used in almost every, but not only, development project—not just for tracking the history of files but also for developing cooperatively, making use of its branching feature. This allows for development while maintaining a stable version on the main branch. [25p]

Fig. 5: Git Logo [25o]

3.3.2 GitHub

GitHub is a platform maintained by Microsoft. As the name implies, it is based on Git and provides the opportunity to easily share your Git repositories with other users. It is free to use for non-commercial applications and a very popular option when it comes to developing in a team. [25b] Furthermore, it also implements handy extensions that integrate tightly with Git, for example GitHub-Actions, which is their solution for CI/CD pipelines. There are many alternatives to GitHub, like GitLab, which is open-source and can be self-hosted, or BitBucket, a solution by Atlassian. [25ad]

Fig. 6: GitHub Logo [25e]

3.4 Map Data

3.4.1 OpenStreetMap

Fig. 7: OSM Logo [25x]

The project OpenStreetMap (OSM) is an open-source initiative that aims to make high-quality map data available to everyone for free. It was created in 2004 by Steve Coast. It was later widely adopted, with the main push being the pricing Google introduced to the embedding of Google Maps, so that today it can be found in all kinds of applications. These applications range from basic embedded web components to video games such as Microsoft Flight Simulator, which utilizes OSM for its building models. The GIS behind OSM gets maintained by volunteers and is fed not only using GPS-Data but also image data and other different formats.
OSM is the backbone of this thesis and without it, it wouldn't have been possible to develop this tool. [25z] [Wik24]

3.4.2 GraphHopper

Fig. 8: GraphHopper Logo [25d]

GraphHopper (GH) in its core is an open-source routing engine written in Java. It can be deployed as a web server and is used to calculate the optimal route, distance or time between multiple different points. It supports multiple modes of transportation as well as "snap to road" technology. [25q] GH also provides a ready-to-go API that you can access for free, but we choose to deploy our own instance, since the requests we needed to make exceed the limits of the free version. [24c]

3.5 Development Tools

3.5.1 VS Code

Visual Studio Code (VS Code) is a version of the Code-OSS project with Microsoft-specific customizations, making it a closed-source product licensed by Microsoft. VS Code is highly customizable, with support for community-made plugins and many options for the user to define how things should act and look. [25am] VS Code comes with numerous useful features, like native support for VCS. As an Electron-based app, VS Code can be used on a desktop and in any device's browser. [25an] [25al]

Fig. 9: VS Code Logo [25ak]

3.5.2 IntelliJ

IntelliJ IDEA is an IDE primarily designed for Java and Kotlin developers, published by JetBrains. In comparison to VS Code, it is more resource-intensive due to included tools like the direct integration of a database connection via the embedded DataGrip version. IntelliJ is the base framework used for other, more specific IDEs by JetBrains, like Android Studio, WebStorm or DataGrip. Its functionality can also be extended through plugins. [25t] [Con25e]

Fig. 10: IntelliJ IDEA Logo [25v]

3.5.3 Android Studio

Android Studio is based on the IntelliJ framework, but, as the name suggests, with the specification to developing Android apps. Aside from Java and Kotlin-based pure Android apps, it can also be used for developing cross-platform applications using the Flutter framework. The IDE introduced by Google also features an emulator for Pixel devices, which simplifies testing. Lastly, it provides an intuitive way to compile the app and flash it to a physical Android device using ADB and USB debugging.

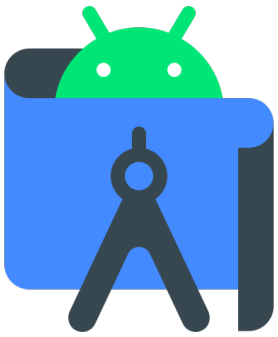

Fig. 11: Android Studio Logo [25a]

3.5.4 Postman

Fig. 12: Postman Logo [25w]

Postman is an API platform, primarily used to test APIs. It features collaboration and automatic code generation, allowing users to design requests directly in Postman and export them into any of the supported languages. We used it to develop and test our API without a frontend application to minimize potential points of failure.

3.5.5 Docker

Fig. 13: Docker Logo [23b]

Docker is a tool for containerization and one of the most popular options in this field. Through containerization, it is possible to isolate a process from the host machine in a virtual environment. Furthermore, Docker is portable, which makes it a great choice for software projects that many developers with different environments work on. It is used to create a controlled infrastructure that guarantees the correct configuration for the production server. We used Docker to host our PostgreSQL database across multiple different systems throughout the development process.

4 Spring Framework

This chapter concentrates on exploring the Spring ecosystem and covers core components like Spring Framework, Spring Boot and Spring Data JPA. It highlights the advantages of this ecosystem in simplifying Java development and improving efficiency.

4.1 Spring Boot

Spring Boot is a tool that makes developing web applications and microservices with the Java Spring Framework faster and easier. As powerful as the Spring Framework on its own is, it still requires much time and knowledge to configure, set up and deploy Spring apps. Spring Boot tries to mitigate this effort with three features.

Autoconfigure:

- Autoconfigure initializes Spring apps with a preset of dependencies so that the developer does not have to configure those manually. Spring Boot comes with this feature to automatically configure the Spring Framework and third-party packages based on the project requirements.
- Even though the developer can override the default configuration after the initialization, the initial setup makes the development process faster and more efficient.
- Meanwhile, the autoconfiguration also reduces the possibility of many human errors, which can occur during the configuration of a Java application.

Opinionated Approach:

- When adding and configuring startup dependencies, Spring Boot takes a subjective approach, customizing them to the requirements of the project. Spring Boot automatically selects the appropriate packages and default values for each configuration, removing the need for human setup and decision-making.
- During initialization, project requirements can be specified, allowing selection from a vast collection of starter dependencies, known as "Spring Starters," which cover most common use cases. For instance, the "Spring Web" dependency streamlines the creation of Spring-based web applications by incorporating all essential dependencies into a

minimal configuration. Likewise, "Spring Security" provides built-in authentication and access control features.

- More than 50 official Spring Starters are offered by Spring Boot, and there are numerous other third-party starters accessible.

Stand-Alone Application:

- Spring Boot enables the development of applications that operate independently of an external web server. This is accomplished by immediately integrating a web server, such as Tomcat, with the application as it is initializing. As a result, you can launch the application on any platform with the appropriate command.

[25af; Ibm24]

Setting up a Spring Boot app:

A Spring Boot project can be initialized quickly using the **Spring Initializr**. For a new Spring Boot-based project, the **Spring Initializr** can be opened, where the project details are filled in and a packaged project is downloaded as a .zip file. While initializing, several factors have to be chosen that determine the project structure, such as the programming language, the version of Spring Boot, and any other dependencies required to support development. Depending on the development environment, you can directly develop a Spring Boot project within the IDE itself. Even this method supports the same amount of configuration to simplify the process of development. Not all IDEs are capable of creating a Spring Boot project out of the box, and sometimes a separate plugin needs to be installed. [Cod25]

Difference between Spring Boot and Spring Framework:

Spring Boot's greatest strengths over the standard Spring Framework are its simplicity and quicker development. Theoretically, this advantage is at the expense of the higher degree of flexibility that direct Spring Framework usage would provide. Practically, the compromise is more than worth it, as the Spring Framework's annotation system can still be utilized to inject other dependencies effectively. Besides, all Spring Framework functionalities, like easy event handling and native security, continue to be accessible. [Ibm24]

4.2 Spring Data JPA

Spring Data JPA stands for Java Persistence API and provides a specification for persisting, reading and managing data from the object in the program, which are called entities, to project's tables in the database. JPA specifies a set of rules for developing interfaces that follow specific standards. Therefore, JPA serves as a set of guidelines for implementing Object Relational Mapping (ORM). ORM is the process of persisting an object in Java directly into a database table.

The goal of Spring Data JPA is to create classes called "repositories", which significantly reduce the amount of unnecessary code needed to access and manipulate data from a project's database.

Entities:
As already mentioned, the objects with which Spring Data JPA manages the data are called entities. Every table of a database is an entity, with each attribute being a column in the respective table. The `@Entity` annotation can be used to define the entity, guaranteeing that the class is understood as a component of the database structure and handled appropriately. However, JPA supports other annotations as well. By using annotations such as `@Id` or `@GeneratedValue`, different custom features of the entity, and therefore the database table, can be defined. Those annotations are a way to make the development of the project much faster and even more efficient. [25ab; Tri23; 24a]

Data access object layer:
Although there are many annotations that are usable for an entity, there is one annotation which is a core feature in JPA. The `@Repository` annotation is a marker for a class that fulfills the role of a repository, which is also known as a Data Access Object. However, using the `@Repository` annotation requires an additional attribute. When this annotation is used, the repository class must extend the `JpaRepository` class, which provides many built-in methods for managing, manipulating, sorting, and filtering data in the respective database table. If a required operation is not available by default, custom queries can also be defined within the repository class. [De 24c]

Service Layer:

Classes that provide data manipulation methods, like a repository class, use the `@Service` annotation. By implementing such a class, the project structure is better understood and clearer to other developers who may work on this project later on. [Mee24; De 24a]

Controller Layer:

This layer provides the applications with the routes with which the data can get manipulated through a GUI. By using the `@RestController` annotation, an actual controller class can be defined. The controller uses the service classes from the service layer to get the data or manipulate it in any way. It also specifies the routes that the code can then access in any form of user interface so that the user can actually use or see the data. [De 24b]

Mappings:

There are numerous annotations that define routes along with all their features and how they can be accessed. The `@GetMapping` gives the user data in any form. This can include all the data from a table or a specific entry in a table based on any filter. The `@PostMapping` is used if a user would want to create a new entry in a specific table.
For example, in a company's management system, the `@PostMapping` annotation can be used to build a route for adding a new employee to the database. If the employee's address or surname were to change, the `@PutMapping` annotation would be used to update the stored data for that employee. On the other side, if the employee were to resign, the `@DeleteMapping` annotation would be used to enable the manager to remove the employee from the company's database.
The user can then access these routes through a GUI, enabling them to manage, update, or delete data as required. [Cim23]

4.3 Lombok

In the modern days of Java development, one big challenge developers face is writing boilerplate code, which are code segments that repeat themselves over and over again and get used often in a project. This is especially the case in frameworks like Spring Boot, where we use classes like the service layer that involve a significant amount of repetitive code.

"Project Lombok" is a Java library that has the aim to reduce this boilerplate code by automatically generating the code for commonly used patterns. By integrating Lombok in your Spring Boot project, you can not only simplify your code but also make it easier to read, maintain and write. Lombok works great with the whole Spring framework. If you wanted to use a project with Spring Data JPA, you would not get that far without using any annotations provided by Project Lombok. To flag an entity class for Spring Data JPA, you need to use the `@Entity` annotation. This annotation comes from the Lombok library and adds a couple of features to this class so that JPA recognizes it as an entity for the database.

Lombok injects the needed methods directly into the compiled class files when the program is getting built, which reduces the need to manually write common functions like constructors, getters and setters. When an annotation like `@Getter`, `@Setter`, `@AllArgsConstructor` is used, Lombok modifies the code before the compilation is done. This ensures that the generated methods are available at runtime while keeping the source code clean and minimal. For example, using `@AllArgsConstructor` tells Lombok to generate a constructor with all the variables of a class. Another useful annotation is `@Data`, which combines many methods like the `@Getter` and `@Setter`. Using this annotation further reduces repetitive code.

Since Lombok creates these methods during the compilation process, developers can keep their code clean while still having fully functional classes. [25ag; 25u; Men24]

4.4 Advantages

The Spring Framework has a range of advantages that make it a popular choice among developers. These benefits shorten the development process and help in designing scalable and maintainable apps.

Reduced Boilerplate Code:

A huge factor of the Spring framework is its ability to reduce repetitive code segments. Mainly through the Lombok library, the Spring framework gives the developer many ways to exchange repetitive code with annotations.

Enhanced Testability:

With Spring's huge library of dependencies, you can not only get dependencies that make the coding easier but the testing too. Some dependencies gave us mock data to test the backend endpoints easier.

Flexibility:

The same thing that helped us with testing gave us and other projects the flexibility with which you can, for example, expand or change certain things in your project. This was extremely helpful for us because a certain requirement changed while we were developing the backend.

Consistency:

Spring provides consistent programming and configuration models across many different types of applications. Whether you would want to develop a web application or a microservice, Spring offers a unified approach that improves developer productivity.

Improved Productivity:

Tools like Spring Boot, which are part of the Spring ecosystem, significantly enhance developer productivity by providing better approaches to different problems and embedded servers. [Ibm24]

5 Structure of the Backend

A well-structured backend is essential for building a scalable and maintainable application. This chapter explores the different layers of our backend. Each layer has its own purpose in handling client requests, processing logic in the code or managing database operations. Furthermore, this chapter provides an overview of the applied design principles we used in our backend.

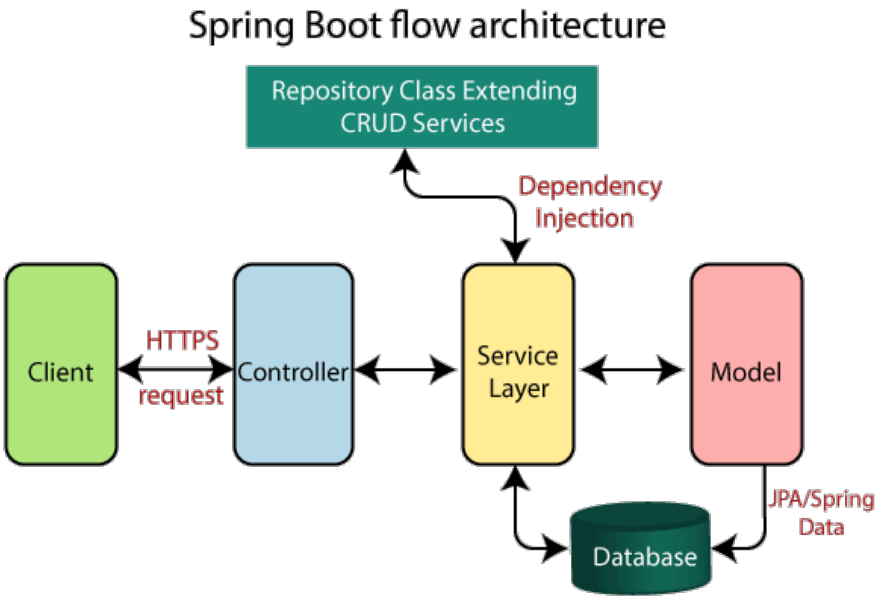

Fig. 14: Layers of Spring Boot [Pat24]

5.1 Controller Layer

The controller layer of a Spring Boot application is a significant layer that takes care of incoming HTTP requests and decides the appropriate response. It serves as the interface between the client and the backend, which receives requests, hands over the tasks to the service layer, and provides the responses accordingly. It ensures that the data, which is used by the logic in the application, is properly and effectively processed.

In a standard configuration, controllers handle a collection of HTTP methods. Amongst those HTTP methods are GET, POST, PUT and DELETE. Each method has a certain function or action that is then performed by the application. Furthermore, every type of method corresponds to different operations on the data, such as retrieving, creating, updating or deleting information. Each controller belongs to a unique endpoint, which serves as the interface for interacting with the application. The controller then uses service methods to execute the logic and return

an appropriate response.

To ensure effective management of routing, Spring Boot uses annotations like `@GetMapping`, `@PostMapping`, `@PutMapping` and `@DeleteMapping`. These annotations identify the mapping between some of the HTTP methods and their respective handler methods in the controller. For example, the `@GetMapping` annotation is used for reading data, whereas the `@PostMapping` is applied for adding new data. The `@PutMapping` is utilized for updating existing data and the `@DeleteMapping` for deleting data.

One of the characteristics of the controller layer is the way it communicates with DTOs and entity classes. While the controller gets HTTP requests, it typically utilizes DTOs to enable data transfer between the client and the backend in a secure and effective manner.

5.2 Service Layer

The service layer plays a significant role in the backend organization to process the logic and serve as a buffer layer between the repository and controller layers. It makes the application modular, maintainable and scalable since it decouples request handling and data access. This separation is necessary to provide flexibility in changes and additions to the application without impacting other parts of the project.

One of the primary advantages of having a dedicated service layer is that it can be reused. Rather than duplicating the logic all over the system, controllers and other code snippets can use service methods when they need them. Not only does this save code duplication, but it also simplifies future changes, since logic changes can be implemented in a single location without having an impact on the application overall.

Another significant aspect of this layer is transaction management. When an operation has more than one step and consists of a sequence of actions, a guarantee that either all or none of those steps are executed is an absolute necessity. By defining such transactional boundaries at the service level, the application ensures data consistency and prohibits incomplete operations from triggering undesired actions.

Wrapping complex logic within the service layer keeps the backend organized and clean. Rather than adding several conditions and operations directly into controllers or repositories, the service layer offers an organized area for processing the data.

5.3 Repository Layer

The repository layer is in charge of managing interactions with the database by exposing efficient methods for storing, reading, updating and deleting data. By using this layer, the backend is kept modular, which results in the application being more maintainable and scalable.

Spring Data JPA comes with numerous in-built methods supporting most of the common database operations. By utilizing the repository layer, operations like saving new data, getting all existing data sets, updating data and deleting data are automatically available. This reduces the amount of code needed for database operations. Furthermore, it also accelerates the development process by providing many methods.

Although built-in repository methods cover most scenarios, there are times when more complicated queries are required. To handle such cases, Spring Data JPA supports defining custom queries through annotations. This allows developers to get certain data efficiently without performing unnecessary database operations.

5.4 Persistence Layer (Entity Classes)

The Spring Boot application's persistence layer handles database communication, entity classes and how different entities are related, along with ensuring data consistency. With JPA in Spring Boot, developers can create a well-structured and optimized data model that enhances the performance and scalability of the application.

Entity classes are used to represent database tables and need to be defined with an annotation to be scanned by Spring Boot JPA. If the table name is different from the class name, it can be annotated explicitly. In addition, unique identifiers need to be defined to ensure that every entity is unique.

Cascading operations automatically persistence and deletion but must be dealt with cautiously in order not to lose data. Validation maintains the integrity of data by placing constraints like mandatory fields and length limitations.

Auditing enables tracking when and by whom entities are modified, which provides more accountability. Restricting field values to predefined choices guarantees information consistency and using DTOs optimizes queries by retrieving only necessary fields.

5.5 Applied Design Principles (Data Transfer Objects)

During the development of a Spring Boot application, having an efficient data communication between different layers is crucial. And exactly this is where DTOs are needed.
DTOs are objects that carry data between layers in an application. They are used to encapsulate and transport data mostly between a server and a client.

Why Use DTOs?

DTOs are crucial for a backend architecture due to their abilities to enhance the security and maintainability, as well as the performance of the data. Their ability to hide data is among their major strengths because they support exposing only the necessary information to clients without showing sensitive data.
From a performance perspective, DTOs maximize network efficiency by only sending the data that is required for the current action performed by the backend. This advantage is particularly valuable in systems where bandwidth and response time are essential.
Lastly, DTOs are adaptable, which allows developers to customize responses to specific use cases. They can support data combination from different entities or the exclusion of unnecessary fields, which then results in more efficient responses and easier processing.
By utilizing DTOs as they are intended, applications can achieve better modularity, security and overall performance.

Example for Using DTOs

A common scenario where DTOs are useful is when managing user information in an application.
Imagine a system that stores user details such as first name, last name, email and password. If the application exposes the user data to external clients, it could lead to security risks, as sensitive information like passwords should never be shared. Instead of exposing the entire user object, a DTO can be used to filter out unnecessary or sensitive data and only transfer the relevant information.
Therefore, when a user requests a list of other users in an application, the system does not need to send private details like passwords. As an alternative, the application can create a DTO that only contains essential information, such as first name, last name and email. This would ensure that the system remains secure while still providing necessary data for the application

to function properly.
DTOs also help to optimize performance by limiting the amount of data transferred. Without DTOs, the application would send the passwords between a client and a server every time the backend needs to send data. However, by using DTOs, the application is limited to sending the first name, last name and email but not the password.
To ensure DTOs are as efficient as possible within a backend application, certain best practices should be followed.

Keeping DTOs Simple:

- DTOs should only contain the necessary fields required for the specific use case that it is needed for. Having too much data makes them harder to maintain and can lead to unnecessary complexity. By keeping DTOs organized, they remain efficient and can be used as they are intended.

Using Validation:

- Data validation in DTOs is required to maintain data integrity and prevent processing invalid or incomplete data. By applying validation rules, such as not allowing a field to be empty or limiting the length of different fields, errors are caught early. This prevents incorrect data from being passed on to the logic or inserted into the database.

Using Automation Tools:

- Manually constructing a DTO based off of an entity is known to create errors and to be time-consuming. To simplify this, there are tools that can be used to map objects automatically. They reduce the usage of unnecessary code and speed up the development process by getting DTOs properly filled without writing manual code.

Documenting the DTOs:

- Proper documentation of every DTO created in the application is crucial. Frameworks like Swagger or SpringFox provide the possibility of auto-generated API documentation, which is easily understandable by other developers concerning the structure and expected data of each DTO. This makes collaboration easier and more consistent within the project.

While DTOs provide numerous benefits in organizing data transfer between the layers of an application, there are some challenges that come with them. As a project grows in complexity, it may take more effort to keep the DTO approach organized. It is wise to know the possible disadvantages so that DTOs can be utilized effectively without introducing unnecessary overhead.

Maintenance Overhead:

- As an application evolves, the number of DTOs can grow extensively. Monitoring these DTOs, modifying them as new requirements emerge and maintaining them synchronized with the logic can prove to be a daunting task. Without proper structuring and an explicit DTO strategy, their maintenance can introduce excessive complexity.

Performance Impact:

- While DTOs are used to reduce the amount of data transferred over the network, mapping entities to DTOs and vice versa is extra processing. This process will be a bottleneck in performance in high-traffic applications. Using optimized mapping strategies and automated tools for this purpose can help this problem. This still needs to be taken into consideration when implementing DTOs.

Consistency:

- Keeping DTOs and entity models in sync over time is challenging. When there are any changes in the underlying data structure, it is essential that the impacted DTOs are also adjusted accordingly. If these changes are not carefully handled, inconsistencies may arise, which lead to potential data mismatches and errors in API responses.

Complexity:

- In use cases involving simple data interactions, DTOs may not be required. Adding DTOs to every operation will probably add a layer of abstraction that will not be as valuable. In this scenario, working with entities directly could be a simpler and cleaner approach.

[Lal23]

6 Area Borders

This chapter describes the use of a convex hull algorithm to define the boundaries of an area. It is a process of determine the outermost points of an area and joining them to create a border. This chapter also explains the fundamentals of the convex hull algorithm, other possible ways of calculating such a boundary, and why we used this method in our project.

6.1 Purpose of Area Borders in the App

The convex hull algorithm is used in the app to define certain addresses that form the boundaries of an area. These boundaries are crucial to better organize and manage all areas which are assigned to all groups within the application. Below is a screenshot of the app that shows the highlighted borders of an area to visually demonstrate how these areas are shown within the app.

6.1.1 What are Border Addresses?

Border addresses are certain points that define the outer boundaries of an area. Furthermore, an area is defined by a set of addresses which get added to a certain area prior to the caroler campaign (Königskarten Aktion). Once the area is created, the convex hull algorithm can be applied to the list of addresses to find the border addresses. These border addresses just go around the perimeter of the area. Therefore, the outcome is a precise, convex shape that covers the entire area.

6.1.2 Why Do We Use Border Addresses?

Border addresses are important for a number of reasons. They give the visual definition of each area's boundaries, making it easy to manage and navigate through the different areas for admins and making it easier for each group to see the boundaries of their respective area. The use of the convex hull algorithm ensures that the boundary defined is the smallest convex shape possible that incorporates all the addresses in this area. The importance of the border addresses is outlined in the following points:

For the Admin:

1. **Clear Visual Representation:** In the app interface, each area is shown in a different color and border addresses are highlighted by a line which goes through all the border addresses of an area. This makes it easy for the admin to see where each area is and how big they are.

2. **Efficient Area management:** By visualizing the areas with the border addresses highlighted, the admin can rapidly evaluate the boundaries of each area. Based on the result of each area's boundaries, the admin can remove or add addresses if necessary. The management process is therefore improved by its flexibility.

3. **Better Decision Making:** The admin can make a well-informed decision about adding or removing addresses from an area by visually seeing how big an area is or how many addresses are in an area. Besides other factors, this information guarantees better decision-making from the admin on how to distribute the addresses to each area.

For Each Group:

1. **Clear Understanding of Area Boundaries:** The border addresses give guides of each group a clear idea of the addresses they are in charge of. Each group can also more easily see the boundaries of their area, which helps to avoid misunderstanding of where each group has to go and where they must not go.

2. **Improved Navigation and Accuracy:** Guides are able to navigate better in their areas when they have a visual representation of their area. By keeping them within their area, this visual aid helps them avoid errors, for example, visiting addresses from other areas.

3. **Increased Efficiency:** The guides are able to finish their work more quickly because the boundaries of each area are clearly marked. They can focus on their designated area and plan their routes to be as efficient as possible and to not cross any other areas in the process.

6.2 Overview of the Convex Hull Algorithm

The convex hull algorithm is a fundamental concept in computational geometry. On a two-dimensional system, it is used to find the smallest convex polygon that can hold a specific set of points. Simply put, it determines the outer border or edge that surrounds each of the specified points. The convex hull, which is the outer boundary, is the tightest shape that can enclose every point without any spaces in between.

To better understand the concept, consider a set of points randomly placed on a flat surface. If a rubber band were stretched around those points, it would form a shape that encloses the outermost points. This resulting shape is the convex hull, which is the smallest convex polygon that fully encloses all points. The image below shows exactly this process, with the points being surrounded by the convex hull. This gives a visual representation of the convex hull algorithm.

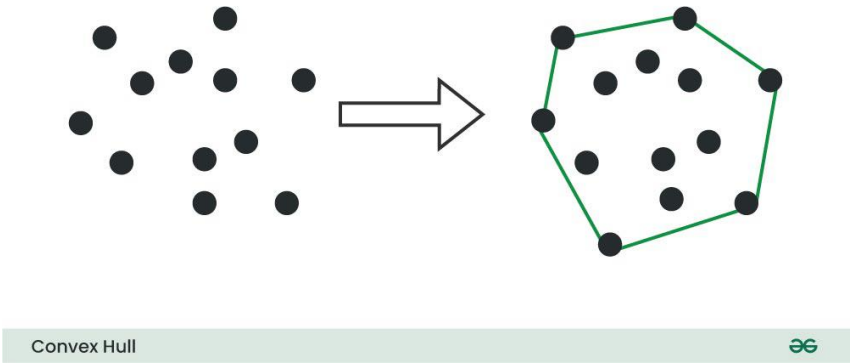

Fig. 15: Process of the Convex Hull Algorithm [Gee24a]

A convex polygon is a shape where all the corners point outward and there are no inward curves or sharp angles. To put it simply, if you draw a straight line between any two points inside the shape, it will always remain inside or on the shape's edge. This indicates that there are never any inward corners on the convex hull. In contrast, a concave polygon has at least one or more inward curves where the interior angle is greater than 180 degrees.

To better visualize the difference, the image below shows that all the angles of the convex polygon have less than 180 degrees, therefore they are "pointy". However, on the concave polygon, it is visible that the angles on top has more than 180 degrees, therefore the polygon has an inwards curve [25l]

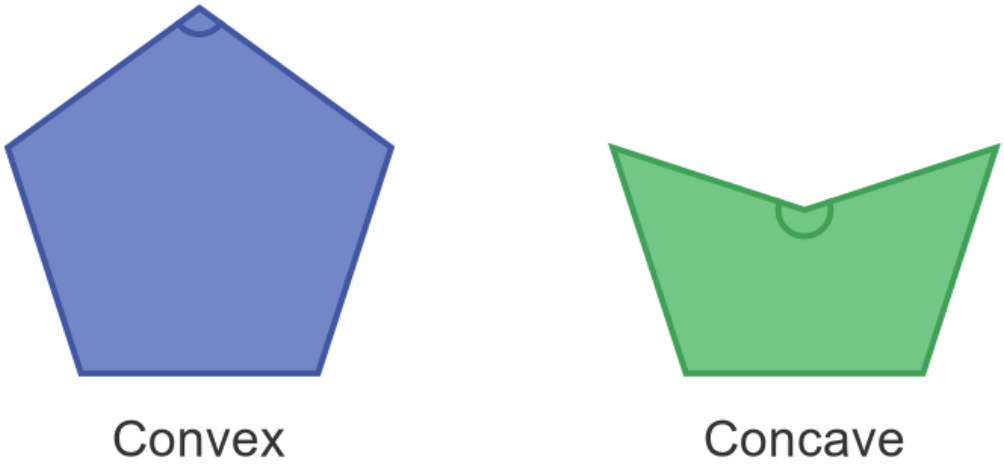

Fig. 16: Difference between a convex/concave polygon [McB25]

6.2.1 Why Use the Convex Hull?

The convex hull algorithm is especially helpful for identifying a set of points' boundary and eliminating any extra points that fall inside it. In the case of the addresses of each area, for example, the convex hull algorithm helps in defining the outermost boundaries that enclose these points in the smallest possible area. Whether in pathfinding, geographical mapping or other spatial analysis applications that demand distinct boundary limits, this procedure is crucial.

6.2.2 Types of Convex Hull Algorithms

The convex hull can be calculated using a variety of algorithms, with the most popular being **Graham's Scan** and the **Monotone Chain algorithm**. Because of their effectiveness and capacity to manage sizable datasets, these algorithms have been widely adopted.

Graham's Scan Algorithm:

- Graham's Scan is a method, which was introduced in 1972 to find the convex hull of a set of points. The process starts by sorting the points based on the "polar angle" of each point relative to the bottom-most point. Simply speaking, the polar angle is the angle between a point and a reference direction, which is often the horizontal axis (the x-axis).
- As soon as the points are sorted, the algorithm builds the convex hull by looking at each point in the sorted list. It moves from one point to the next, and while going through each point, it checks if the previous three points form a right or a left turn. As soon as a right turn is detected, the last point is removed because it would create an inward curve in the shape, which would then result in a concave hull and not a convex hull. The algorithm continues with this until the hull is fully formed.
- The time it takes to run the Graham's Scan is **O(N log N)**, with N as the total number of points. Therefore, the required time grows reasonably with a larger dataset, which makes it efficient for moderately large sets of points.

[25j; Gee24b]

Monotone Chain Algorithm:

- The Monotone Chain algorithm was introduced in 1979 and is another efficient way to calculate a convex hull. It begins by sorting the points based on their horizontal position (x-coordinate), from the leftmost to the rightmost.
- The algorithm then makes two steps to construct the convex hull. First is the upper hull and then the lower hull. The algorithm goes through the sorted list of points, adding them to the hull and ensures that each new point does not cause an inward curve by checking whether the last three points form a right or left turn, just like in Graham's Scan.
- The run time is the same as the Graham's Scan with **O(N log N)**, which makes it well-suited for larger datasets.

[25j; Gee24b]

6.2.3 Key concepts in the Convex Hull Algorithm

The convex hull algorithm has a number of key concepts, all of which are essential to guaranteeing the algorithm's accuracy and effectiveness.

Orientation:

- This is a reference to the relative orientation of three consecutive points. The convex hull algorithm determines whether to add the next point to the hull based on the orientation of a triplet of points. There are three possible orientations for the points: clockwise, anticlockwise and collinear. Only if the points are oriented counterclockwise, should the point be part of the hull. Otherwise, it should be discarded.

Polar Angle Sorting:

- A key step in convex hull algorithms is sorting the points according to their polar angle relative to a fixed point, which is usually the lowest or leftmost point. By sorting the points, this process guarantees that they are processed in a way that minimizes computing resources during hull construction.

Stack Operations:

- Convex hull algorithms often use a stack data structure to keep track of all points that are currently part of the hull. To guarantee that only the outermost points stay in the stack, points are added and removed according to their orientation relative to their earlier points.

Convexity:

- The convexity of the polygon that is formed by a convex hull algorithm ensures that there are no inward curves. This is crucial because the convex hull represents the smallest possible polygon that does not have any concave edges or inward curves.

[25k]

6.3 Use Cases of the Convex Hull in Industry

The ability of the convex hull algorithm to define boundaries of a complex dataset in an efficient way makes it widely used across a variety of fields. Convex hulls offer crucial tools for examining spatial relationships, visualizing data, and resolving optimization issues in a variety of fields. Here are some fields that use this algorithm.

Geographical Mapping

- In geographic mapping, convex hulls are used to define the borders of regions. For example, while mapping a region with multiple data points, such as cities or landmarks, the convex hull helps outline the area that encloses all the points. This makes it easier to visualize and analyze the overall shape of a region, which helps with urban planning, resource management and environmental studies.

Image Processing and Computer Vision

- Convex hulls are used in image processing to determine the shape of an object. This is helpful for tasks like object recognition, where the convex hull helps define and object's boundaries, which improves detection. It makes it easier for algorithms to process and categorize objects in an image by reducing complex shapes to convex polygons.

Computational Geometry

- Convex hulls are crucial in computational geometry for solving issues such as determining the smallest polygon possible that contains a set of points. This has uses in fields like pattern recognition. By providing this convex hull, algorithms can concentrate on the outermost points and ignore the inner one, which lowers processing costs.

Animal Behavior

- In ethology, the study of animal behavior, convex hulls are used to identify an animal's home range. The convex hull gives the estimated area an animal needs to be provided with for its habitat. In ecological studies, this approach is helpful for tracking animal movement patterns, determining territories and keeping an eye on endangered species.

Astronomy and Astrophysics

- Astronomers utilize convex hulls to define the limits of star clusters and other celestial bodies. Scientists can determine a cluster's outer limit by looking at the convex hull of stars, which helps them study things like galactic formation, star density and gravitational impacts in space.

[Con25b]

6.4 Alternate Methods for Area Border Calculation

The convex hull algorithm is a widely used approach for calculating area boundaries. However, there are other methods that may be more suitable depending on the application requirements. These alternative methods offer different ways to calculate area boundaries with varying levels of complexity and accuracy.

Minimum Bounding Box (MBB)

- The simplest technique for calculating area boundaries is the MBB. It determines the smallest rectangle that completely encloses a specified set of points. The edges of the box are normally aligned with the coordinate axes, which makes it efficient with the computing resources and easy to implement. Applications like spatial indexing in databases and image processing use this technique. However, it does not consider things like concavities and irregularities in the point distribution, which may lead to the MBB not always correctly reflecting the data. The picture below shows a figure that visualizes the difference between the MBB and the convex hull. The convex hull is the gray inner area and the outer line is the MBB. This shows that the convex hull is more precise. [Con24b]

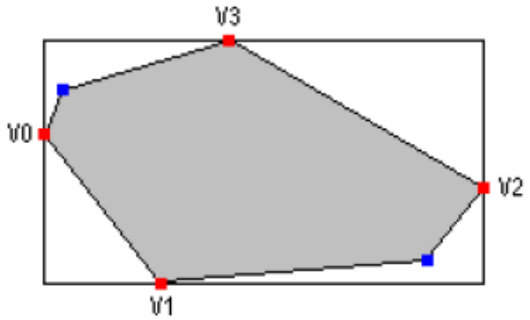

Fig. 17: Difference between MBB and Convex Hull [24b]

Alpha Shapes

- By adding a new parameter, α, that controls how closely the boundary should attach to the specified point, alpha shapes offer a more flexible method of defining area bounds. This approach is more accurate than the convex hull for irregular shapes because it may capture concave features by decreasing this new parameter α. This method is especially helpful in fields like shape recognition, molecular modeling and terrain analysis, where precisely recording complex boundaries is crucial. Choosing an appropriate α value is the main challenge with Alpha Shapes, as different datasets may require different levels of precision. In the picture below, there is the difference between convex hull and alpha shapes. It shows that in this case the alpha shapes actually do a better job than the convex hull. [Con25a]

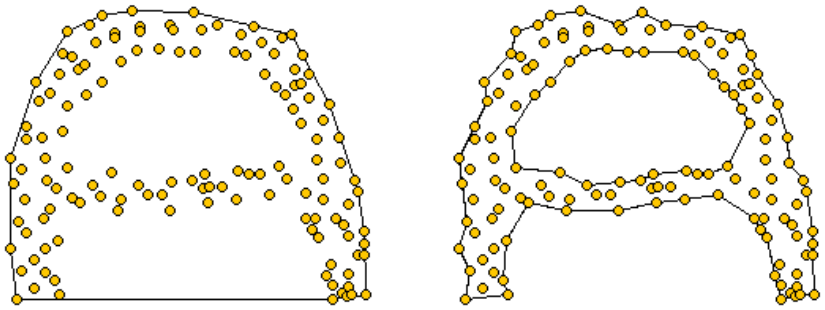

Fig. 18: Difference between Alpha Shape and Convex Hull [13]

Delaunay Triangulation

- Delaunay Triangulation is a method that makes a mesh of triangles that connect a set of points while maximizing the minimum angle of each triangle. This ensures a well-formed structure. A border can be roughly approximated by using the triangulation's outside edges, especially when pairing this method with techniques like Alpha Shapes. This method is mostly used in geographic information systems (GIS). While it provides a more structured representation of an area, it often requires post-processing to extract a meaningful boundary. [Con25d]

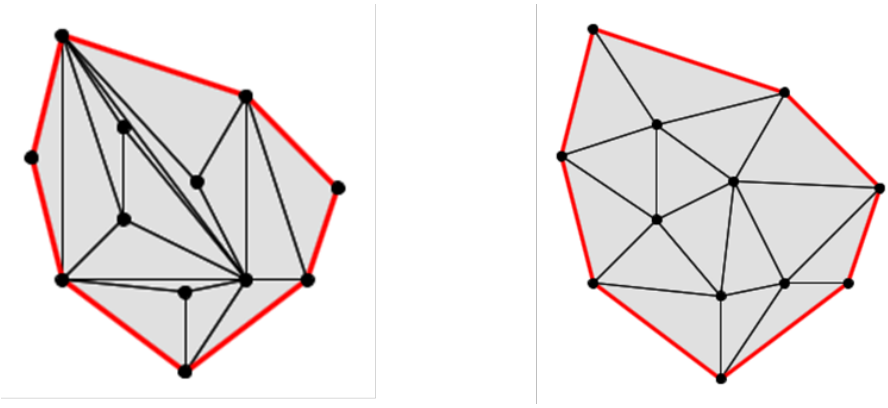

Fig. 19: Mesh of Triangles from Delaunay Triangulation [Luc25]

6.5 Rationale for Choosing the Convex Hull Method

Accuracy, computational efficiency and implementation ease had to be balanced when choosing a method for identifying area borders. The convex hull was selected as the best strategy for our application after a number of approaches were taken into consideration.

One major reason for this choice was that the convex hull method was precise enough in comparison with more complex methods like the alpha shapes. Since our application did not need such fine control, the additional complexity of alpha shapes was unnecessary.

Furthermore, the convex hull approach offers a significant advantage in terms of speed. It enables quick computing in comparison to other methods, which makes it ideal for our application as our resources were somewhat limited for the border addresses.

Finally, our choice was significantly influenced by ease of implementation. The convex hull approach does not require a lot of parameter adjusting, is well-documented, and is simple to incorporate. This made it a practical choice, which allows for a reliable and efficient solution without adding unnecessary complexity.

A fair balance between accuracy and efficiency was achieved by selecting the convex hull method, which allowed area borders to be calculated quickly while still providing a clear visual representation of each area.

6.6 Integration of the Algorithm into the Backend

The convex hull algorithm is implemented within the backend as a part of the service class, which analyzes an area's allocated addresses to identify its boundaries. The frontend can request and use the calculated border addresses thanks to this functionality, which is made available through a dedicated route.

As soon as a request is made to the route, the service class gets the area for which the border addresses are needed. The program then retrieves all the addresses in the respective area, which then get processed by the convex hull algorithm. This then determines the smallest convex shape that encloses all points. The result list of all border addresses is then returned as a response to the frontend, where they are getting used for visualization and management. The server-side calculation load is effectively managed by implementing the convex hull calculation directly in the backend, guaranteeing reliable and consistent results on all client devices.

7 Defining usability

Since my research question, "How can user-experience principles add to an intuitive map displayment for nonprofit activities in which people of different technical know-how levels collaborate?" is all about usability, I want to introduce you to its basic concepts and challenges but also provide some examples of how usability can impact a software's revenue and perception.

Usability is a critical aspect of software and interface design, ensuring that users can efficiently and effectively interact with a product or system. Its job is to provide clear feedback and "experiences" to the user so interactions between software and humans feel smooth and straightforward. Because each human being is different in its emotional experiences, it is difficult to design a kind of "one size fits all" solution. Due to this circumstance, many studies and experiments were conducted. [Nie24a]

7.1 Why it is important

Usability ensures that users can accomplish their goals with minimal frustration and maximum efficiency. With the increasing reliance on digital tools, usability plays a key role, not only in shaping user experiences but also in the accessibility of software for diverse user groups. A well-designed and thought-out usability concept can go a long way from refining a once tedious and complicated-to-use product, to one that can be operated even by non-familiar users or disabled people. This plays a big part in the inclusion of all age and knowledge groups as well as the general market share through mass adoption because of the easiness.

7.2 Components of Usability

According to Jakob Nielsen, usability consists of five core components. To achieve the best possible usability, each of the factors must be taken into account and be improved to its maximum.

- Learnability

  How **easy** it is to accomplish basic tasks the first time

- Efficiency

  How **quickly** tasks can be accomplished after an initial learning period

- Memorability

  How **memorable** actions are to users so, after an extended period of not using software

- Error handling

  How **many** errors users make while using the design and how **severe** they are

- Satisfaction

  How **pleasant** the overall experience of using the product is

[Nie24a]

Now that we are aware of these key points, what measures can we take to reach the goal of great usability? According to Nasrullah Hamidli, human-computer interaction relies on consistency, visibility, feedback, and simplicity. Consistency ensures users do not need to learn new interactions for each task. For example, buttons should look alike and be in a similar location. This makes for a more natural navigation across the product and an overall familiar feel. Simplicity connects directly to this. Its goal is to minimize clutter and make user interfaces easy to understand and provide one clear way to accomplish a task, not many possible but complicated and unintuitive ways. It also aims to reduce distractions. Visibility allows users to clearly understand their options at any given moment, this is most often achieved through visual cues, like grayed-out buttons. This goes hand in hand with the feedback aspect, which

provides immediate confirmation of actions. Loading indicators, color changes and alike get used most often.

Another important part of designing a good UI is typography and colors. These can act as parameters for the attention and emotions of users, as well as establish visual hierarchies, which in turn, contribute again to a simpler-to-navigate interface. [Ham23]

7.3 Fundamental concepts

In this section, we will look further into these basic concepts and examine what designers can concretely do to improve usability. To make it more descriptive of what impacts usability, we will, along the way, modify a simple website.

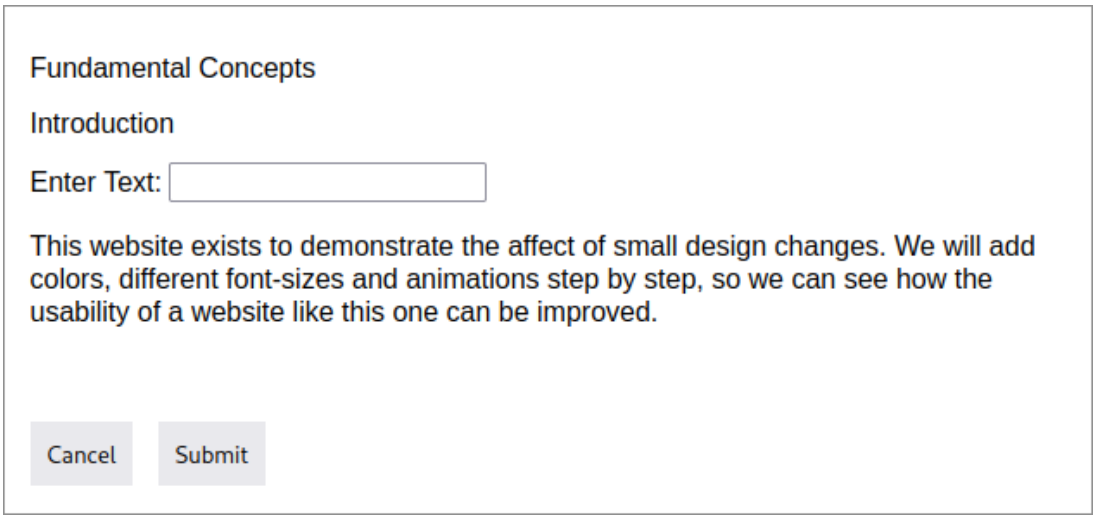

Fig. 20: Starting point of usability example website

Visual Hierarchy

Starting off with visual hierarchy. It is the most basic principle in UI/UX design and dictates how users perceive and navigate content. A strong visual hierarchy contributes to simplicity and ensures that important elements are more visually prominent, guiding users toward essential actions and information. Factors include the text size, weight and spacing.

In the initial version, visitors of our website are not able to clearly and promptly differentiate between headings and actual text. This is due to the use of only a single font size and weight.

If these attributes get tuned just a little, it becomes suddenly much easier to distinguish primary content from secondary information. Headings are bold and larger, while body text is appropriately sized for readability. Proper spacing and alignment create a structured and pleasant reading experience.

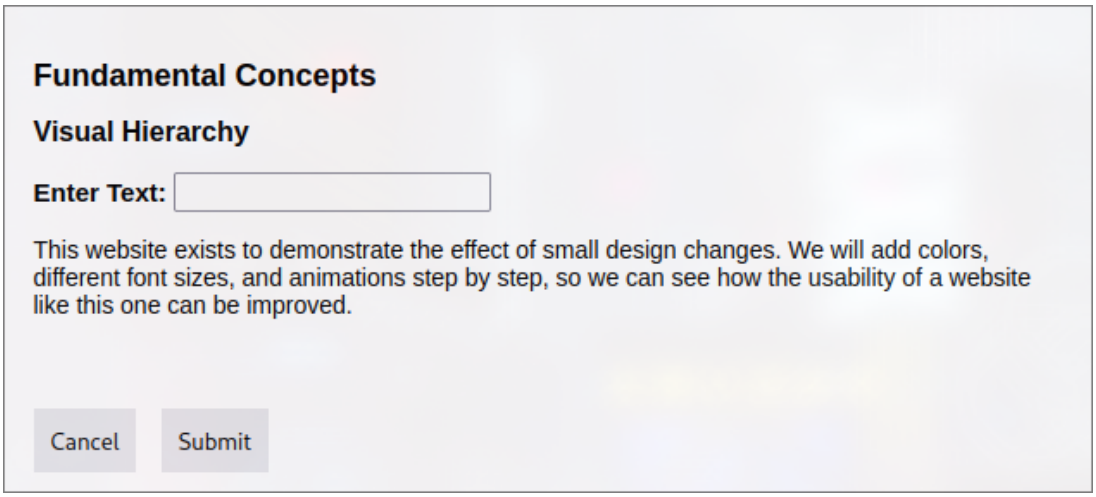

Fig. 21: Example with better visual hierarchy through different font size and weight

Color Theory & Contrast

Color selection plays a critical role in usability and aesthetics. Well-chosen colors draw attention to important elements, while poor color choices can make content inaccessible, especially for users with visual impairments. The advantages of attention through colors can also be used in reverse, for example, a button to delete something important may be colored red to indicate a fatal action but could also be gray, so the user must consciously decide to press it. This acts as a safety measure.

To help designers with their color choices and visualize what colors work together, the color wheel was invented. On it, relations between colors can be shown easily. It consists of three groups of colors.

- **Primary**

  These are the fundamental colors that cannot be created by mixing other colors. They vary, depending on the media format they will be used on. On screens the *RGB model* is in use, while when printing color, cyan, magenta and yellow (CMY model) get used.

- **Secondary**

  Secondary colors arise by mixing two primary colors in equal amounts. In designing applications, they are often used for buttons or highlighting important information.

- **Tertiary** They are often associated with a calm and refined aesthetic. They are not quite the same eye-catchers as primary colors, rather they have their value in creating appealing combinations.

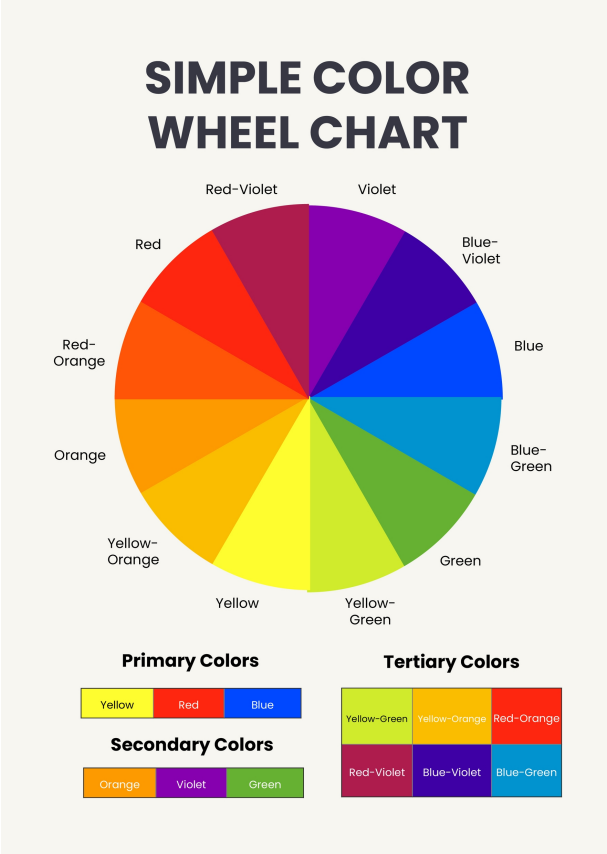

Fig. 22: Color wheel with primary, secondary and tertiary colors sorted [25ae]

Relationship Type	Description	Example Colors	Common Uses
Complementary	Colors opposite each other	Red & Green	Creating bold contrast
Analogous	Colors next to each other	Blue, Blue-green, Green	Harmonious, natural looks
Triadic	Three evenly spaced colors	Red, Yellow, Blue	Balanced, vibrant designs
Split Complementary	One color + two adjacent to its complement	Blue, Yellow-Orange, Red-Orange	Softer contrast than complementary

Fig. 23: Key color combinations with effects [25g]

The current design doesn't feature any color at all. To highlight the buttons, we can give them a color. Now we also need to change the text color, because, if left as is, the contrast would not be great, and therefore the text would be harder to read. [25h]

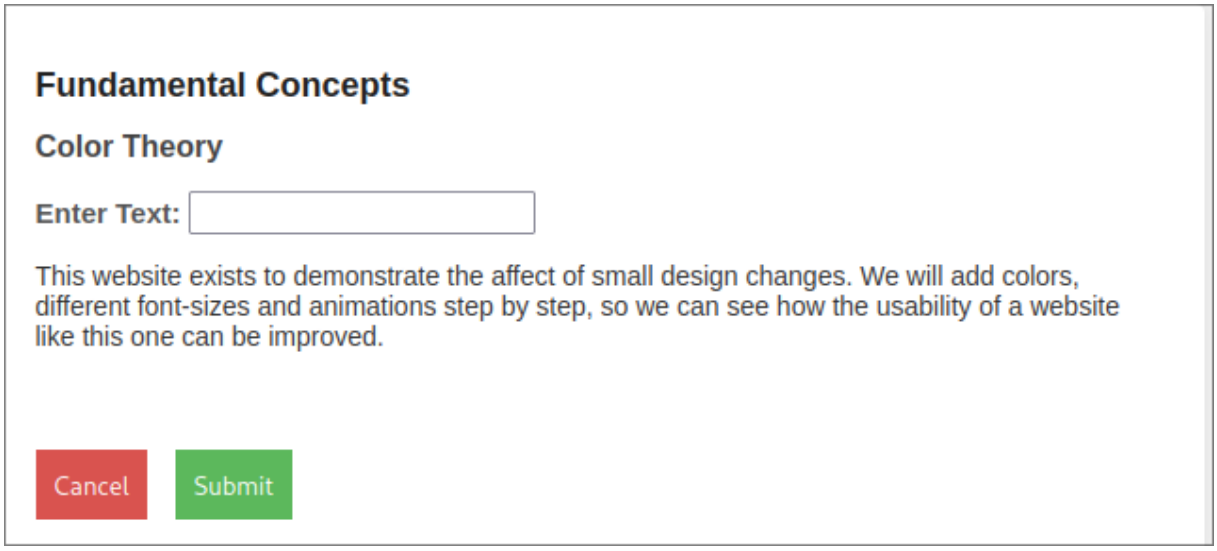

Fig. 24: improved usability through colors

Consistency

Consistency in design enhances user experience by ensuring that UI elements and interactions are predictable across different states of the application. Also, a consistent aesthetic can create familiarity, brand recognition and improve the overall visual appeal.

Button Placement & Behavior

Buttons are a critical and one of the most common interactive elements in UI design. Their placement, size and responsiveness affect usability and efficiency. Since the first graphical UIs, the actions to move forward were always on the right side. In contrast, actions to cancel are most commonly placed somewhere on the left. If designers choose to deliberately change this universal agreement, it leads to potential user errors. Also, a color change on hover got implemented, so the user now gets clearly informed that all required fields are populated, and he can move on.

This design highlights the "Submit" button, and places it in the correct location. Through this, the simplicity to use is improved because users do not have to search for the button that lets them continue.

Feedback

Providing immediate and clear feedback enhances user confidence and prevents frustration. There are many forms in which such feedback can be provided, for example, vibrations, sounds or pop-up messages. Feedback can also be supplied by something like a grayed-out button to signal that this action is not currently available. Success messages or error messages also add to the usability of a website, as well as loading indicators and different animations.

7.4 Challenges in designing for a broad user spectrum

Designing for a diverse user base requires the addressing of varying levels of experience, prior knowledge, cognitive abilities, and accessibility needs. Failure to account for these differences can lead to usability issues, preventing certain groups from effectively using a system. Designers must implement features such as adjustable text sizes, screen reader compatibility, and intuitive navigation to ensure accessibility for all users.

Interfaces should always be tailored to the needs and expectations of the end user. This leads to challenges when the user group is not clearly defined or consists of people with widely different backgrounds. For example, elderly people often need further guidance when interacting with digital solutions than members of younger generations. This leads back to the core components of usability design, interfaces need to be simple and unmistakable in their functionality. Failing to provide these core concepts will sooner or later result in a frustrated and shrinking user base. [Ham23]

8 Usability in context the of maps

In this section we will further inspect the usability of mapping solutions like Google Maps. We will identify some flaws of Google's design choices and how they could influence specific user groups.

8.1 Basic Analysis of the Google Maps Interface

Google Maps is one of the most widely used mapping and navigation applications globally. Its feature set includes real-time traffic updates, route planning, and location discovery. Generally speaking, it is quite difficult to design a simultaneously user-friendly and functional mapping application. Maps get overloaded and confusing quite easily. They are bloated with information like street names, house numbers, borders, and geographical features like rivers or lakes. Due to this fact, maps are not easy to design according to the principles of usability. But Google developed a very good and intuitive concept, on which we now will take a look.

The interface of Google Maps consists of a map, a search bar, and a menu for additional settings. The search function is prominently displayed as it is the most common tool used. This is a good use of *visual hierarchy*, as the initial focus when opening the app immediately gets drawn to the bigger hint text in the search bar on top. Below it, there is a list of buttons, so users can quickly search for local places that match a specific category, like restaurants, cafés, or gas stations. Notably, there are no buttons for movement actions such as zooming or panning; all this is controlled through swipe and pinch gestures directly on the map. [BG08]

To start navigating to a specific area, you can search for the street name and house number or the name of a company or other details. Maps gives you recommendations and tries to provide auto-suggestions for your target. When the user selects a destination, the navigation can be started through a big blue button. This is an example of applied *color theory*. The route gets marked by an again bright blue line, which creates a good contrast to the other colors used on the map and captures the attention of the user. This line also has multiple purposes other than displaying the route; for example, if a part is orange, that means the traffic at this point is beginning to jam. If there is then a full stop traffic jam, the line turns red at this section. Also, icons for speed cameras or accidents that other users reported get displayed along it.

One of the core strengths of Google Maps is its interactive and responsive design. Users can zoom in and out using intuitive pinch gestures on mobile devices or scroll actions on desktops. The transition between zoom levels is smooth, preserving context and avoiding disorientation through too big scaling steps. Additionally, the map changes depending on the zoom level. Street names and buildings get displayed only if the user has zoomed in enough. The same happens with markers for businesses and other map data. Through this concept, Google ensures that users do not get overwhelmed. [BG08]

8.2 Identifying Flaws in Google's Design

While Maps is a well-polished product, it is not perfect. One significant flaw is the cognitive overload caused by excessive information. The inclusion of business listings, other suggested routes, live traffic data, and user-generated content can make it difficult for users to focus on their primary navigation tasks. [BG08]

8.3 How Could Specific User Groups Struggle with This Design

Google Maps caters to a broad spectrum of users, but its design can pose difficulties for certain demographics:

Elderly Users: Many elderly individuals may find the interface overwhelming due to small text sizes, densely packed information, and complex menus. Their unfamiliarity with modern digital navigation tools may lead to confusion, especially when trying to search for locations or adjust route preferences. A lack of prominent, simplified navigation options tailored to this group amplifies the issue. [All22]

Users with Low Digital Literacy: People who are not well-versed in digital technology could struggle with Google Maps' multitude of features. They may have difficulty understanding icons, switching between different map modes, or using advanced functionalities like saved locations and street view. A more guided and *simplified* mode could enhance their experience.

Users with Disabilities: Visually impaired users may struggle with *insufficient contrast*, small icons, and the *lack of tactile feedback*. While screen readers can assist, Google Maps does not always provide clear, structured data for these tools. Additionally, users with motor impairments may find it hard to navigate menus and interact with small buttons, particularly on touch screens. [Fro+19]

9 Adaptive Algorithms and Real-Time Data Integration

Mobile field teams need address databases that stay up-to-date in real time. However, traditional methods cannot keep up with fast changes, like road closures, new buildings, or street name changes. Manual updates take time, and fixed validation rules do not account for regional differences or typos. These issues can lead to wasted time or even serious risks in critical situations.

This chapter explains how adaptive algorithms and real-time data integration can help solve these problems. Adaptive algorithms improve address databases by learning from new data, while real-time systems process live updates instantly. Together, they help mobile apps keep up with changing environments.

9.1 Traditional Methods for Address Database Management

Traditional address database management relies on manual or semi-automated processes that lack real-time adaptability. These methods were designed for stable environments with occasional data changes, which makes them less effective for modern mobile applications requiring constant updates. This chapter outlines common challenges faced by traditional systems and their limitations.

9.1.1 Manual Entry and Batch Updates

In the past, address databases were updated through periodic batch uploads, such as monthly CSV files, which store data in a table format with each value separated by commas. Organizations often relied on printed address lists given to field teams, with corrections submitted on paper. This method caused delays. For example, if a street name changed due to construction, mobile teams could work with outdated data for extended periods. [25i]

Manual data entry by administrators was another common practice. Human errors, such as typos, worsen the problem, making the system less reliable. A study conducted by the Journal of Accountancy found that human error rates in manual data entry can range from 1% to 5%. [25ai]

9.1.2 Static Validation Rules

Address database management systems often rely on strict validation rules, such as regular expressions, to verify address formats. Regular expressions define patterns for strings, making them useful for enforcing specific formats. For example, a regex pattern might require the full word "Street" instead of abbreviations like "St." or "Str.". [Aut02]

```
r'^-?[0-9.]*'
```

Fig. 25: Regular Expression Example

9.1.3 Third-Party Data Purchasing

Third-party data purchasing involves obtaining address datasets from external providers to update and maintain address information. These datasets are usually updated on a regular schedule and cover a wide area. However, the data can quickly become outdated, especially in areas where changes occur rapidly.

Temporary road closures or infrastructure damage might not be reflected in these datasets if updates are infrequent. As a result, teams relying on outdated maps may encounter roads that are blocked or no longer accessible. This lack of real-time updates can cause delays in reaching critical locations or reduce the efficiency of field teams.

9.2 Adaptive Algorithms

Adaptive algorithms are methods that continuously adjust their parameters in response to new data and changing environmental conditions. These techniques are vital for applications where real-time adaptation is necessary. The following sections describe several core approaches within this domain. [Con24a]

9.2.1 Fuzzy Matching

Fuzzy matching, also known as approximate string-matching or fuzzy logic name matching, is a string-matching algorithm that helps identify duplicate strings, words, or entries that closely resemble each other, even when there are abbreviations or misspellings. The technique works by applying edit operations such as insertion, deletion, substitution, and transposition to adjust the strings. These operations are measured in terms of an "editing distance," which influences the match score.

Fuzzy matching is widely applied in various fields, such as document data extraction, spell-checking suggestions, deduplication, and even genome sequencing. It helps achieve high data accuracy by identifying and matching errors. [Nie24b]

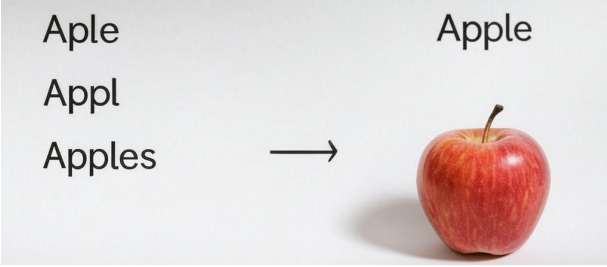

Fig. 26: Fuzzy Matching Example [25r]

9.2.2 Machine Learning Model

Machine learning models can be trained to identify patterns in incoming data and update the database dynamically, eliminating the need for manual intervention. These models are particularly effective in rapidly changing environments, as they continuously learn from new data and provide insights to refine address information in real time.

For example, an ML model can detect and predict changes in street names, new construction sites, or temporary roadblocks based on GPS data, social media updates, or traffic reports. By integrating these data streams, the model can anticipate issues not yet reflected in official datasets and make real-time adjustments to the address database. Moreover, as new information becomes available, the machine learning model continuously improves its predictions, adapting to evolving patterns and ensuring the database remains accurate and up-to-date. [Enc23]

9.2.3 Rule-Based Filters

Rule-based filters enforce logical constraints on address data to maintain quality and relevance in dynamic environments. A filter could flag addresses lacking GPS coordinates during emergency responses, enforcing manual verification before navigation. [25s]

Structure of Rule-Based Filters

A rule-based filter consists of logical conditions applied to address attributes:

- **AND**: All rules must be true.
- **OR**: At least one rule must be true.

Types of Rule-Based Filters

- **Threshold-Based Rules**: Automatically reject addresses with missing or implausible values.
- **Geospatial Rules**: Flag addresses that fall outside predefined operational zones.
- **Historical Consistency Checks**: Detect frequently modified addresses and prompt for manual review.

Integration with Adaptive Algorithms

While rule-based filters provide a structured approach to data validation, they can be enhanced with adaptive algorithms. Machine learning models can analyze user corrections over time and adjust rules dynamically. For instance, if multiple users correct the same address format, the system can learn from these patterns and refine its filtering criteria automatically. [25s]

9.3 Real-Time Data Integration Frameworks

Real-time data integration frameworks are systems designed to collect, process, and deliver data instantly as it is generated. Unlike traditional batch processing, which handles data in chunks, these frameworks enable continuous updates. This makes them ideal for applications where delays are unacceptable.

9.3.1 Challenges

Real-time data integration faces several challenges, particularly in dynamic environments such as mobile field operations:

- **Data Quality:** Errors such as typos or GPS drift require adaptive algorithms for correction.
- **Scalability:** Systems must handle thousands of users and sudden data spikes, especially during disasters or high-demand situations.
- **Fault Tolerance:** Data loss due to network failures must be prevented.
- **Consistency:** Conflicting address data from multiple sources needs resolution.

[hel24]

9.3.2 Core Components

A robust real-time data integration framework consists of four key components:

- **Data Sources:** These include GPS sensors, external databases, and other input streams.
- **Data Ingestion:** Tools and frameworks collect and efficiently route data streams.
- **Data Processing:** Engines process, clean, analyze, and enhance the data.
- **Storage & Delivery:** Processed data is stored in databases and accessed via APIs.

[Lim25]

9.3.3 Popular Frameworks

Three frameworks dominate real-time integration:

Apache Kafka: A high-throughput streaming platform. It could process GPS and traffic data to reroute teams around road closures.

Apache Flink: A low-latency engine for complex workflows. Flink could power ML models predicting address changes.

AWS Kinesis: A cloud-native service for scalability. Kinesis merges regional address updates into a global view for international teams.

Choosing between them depends on needs: Kafka for volume, Flink for processing and Kinesis for cloud scalability. [Ran24]

9.3.4 Evaluation Metrics

To assess the effectiveness of adaptive algorithms and real-time data integration, two critical metrics are analyzed: accuracy and latency.

Accuracy

Accuracy measures how closely the addresses in the database match real-world locations, ensuring reliable address information. Traditional methods often suffer from decay over time due to manual updates or infrequent batch imports, leading to outdated or incorrect data.

Inaccurate data can lead to inefficiencies, delays, or even costly mistakes. Furthermore, achieving high accuracy often involves continuous updates and validation of data, particularly when using multiple data sources.
Accuracy is calculated as:

$$\text{Accuracy} = \frac{\text{Correct Addresses}}{\text{Total Addresses}} \times 100$$

[Gee24c]

Latency

Latency measures the delay between data generation and its availability in the app. For mobile teams, latency above 30 seconds can render updates useless during fast-moving operations. Reducing latency is critical for ensuring timely decision-making and operational efficiency. Key factors influencing latency include:

- **Data Ingestion Speed**: Fast ingestion ensures data is quickly captured, minimizing delays and speeding up availability.
- **Edge Processing**: Local data processing speeds up decision-making by filtering data before transmission, reducing network load.
- **Network Bandwidth**: Latency is affected by network bandwidth; poor conditions, like low signal or congestion, increase delays.
- **Data Prioritization**: Algorithms that prioritize critical data reduce perceived latency by processing urgent updates first.

[Gee24c]

10 Implementation of the Backend

This program's backend, which offers a reliable and scalable system for handling all addresses, streets, areas and special features, is the essential layer that makes sure the admin panel and the mobile app for the guides run well.
This backend serves as the center for data processing and communication, which supports the admin and all guides through a unified API.
It effectively manages all requests from the mobile app, where guides interact with addresses that are in their area. Furthermore, it manages all necessary requests from the admin panel, which enables accurate management and control over anything relevant for the guides, such as addresses, areas, streets and special features an address may have.

One big function of the backend is to enable the admins to perform a wide range of CRUD (Create, Read, Update, Delete) operations on all necessary data. Nevertheless, the other function of the backend is to provide guides with their relevant area and all the addresses in this area. These functionalities are crucial for managing the caroler campaign by the admin and to ensure that the guides have access in the mobile app to accurate and up-to-date addresses and areas.

We use Spring Boot in our backend which ensures seamless data transfer between the user interfaces (admin panel and mobile app) and the database of this project. By having an organized structure with the different layers (configuration, entities, repositories, services and controllers), we ensure that the system remains flexible, scalable and easy to maintain.

Now, this chapter will show the backend's architecture and therefore all its components while also describing how each component functions and contributes to the whole system of the backend.

10.1 Config of Spring Boot (application.properties)

The `application.properties` file in our Spring Boot application is the main configuration file where all the necessary settings which are related to the application's behavior are defined. This behavior ranges from the database connection to server properties. In our

project, this file ensures that the backend is appropriately configured, so it connects to the database, exposes the API on the correct network address and port, and it makes sure that security settings such as CORS policies are configured right.

10.1.1 Database Configuration

One of the main functions of the `application.properties` is to set the database connection. In our case, we utilize the PostgreSQL database and the following properties set the necessary connection details:

```
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=postgres
```

Listing 1: Database Configuration

- `spring.datasource.driver-class-name` defines the driver which is required for the PostgreSQL.
- `spring.datasource.url` sets the connection string to the local database.
- `spring.datasource.username` and `spring.datasource.password` provide the credentials for the database access on the local computer.

This configuration ensures that the backend can interact with the local database to store and retrieve all the data efficiently.

10.1.2 JPA and Hibernate Settings

Spring Boot uses Hibernate for the default JPA implementation. We used the following configuration for this:

Listing 2: JPA/Hibernate Settings

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.dialect=
↪ org.hibernate.dialect.PostgreSQLDialect
```

- *spring.jpa.show-sql=true* and *spring.jpa.properties.hibernate.format_sql=true* enable and format the logging of SQL statements that get executed in the backend.
- *spring.jpa.hibernate.ddl-auto=none* ensures that Hibernate does not automatically generate or modify the database tables based on the schemes from the entity classes.
- *spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect* just tells Hibernate to use SQL optimizations which are PostgreSQL-specific.

10.1.3 Server Configuration

The backend application needs to be accessible over a network, and the following configuration sets its address and port:

```
1 server.address=0.0.0.0
2 server.port=46380
3 server.servlet.context-path=/addresses
```

Listing 3: Server Configuration

- `server.address=0.0.0.0` and `server.port=46380` Set the address and the port of the server.
- `server.servlet.context-path=/addresses` ensures that all API endpoints have the `/address` prefix.

10.1.4 CORS Configuration

To allow secure communication between the frontend interfaces (Admin Panel and mobile app), we needed to configure Cross-Origin Resource Sharing (CORS)

```
1 spring.web.cors.allowed-origins=https://projektlr.htl-kaindorf.at
2 spring.web.cors.allowed-methods=GET,POST,PUT,DELETE
```

Listing 4: CORS Configuration

This configuration solved an error we got during the development process.

10.2 Entity Classes (Structure/Purpose)

Entity classes use JPA annotations to transfer Java objects to the database as tables. These entity classes are necessary for database interaction to ensure structured data storage and retrieval.
Each entity, such as addresses, areas, streets and special features of addresses, corresponds to a different concept in the system. However, together they build a relation model that is well-structured and easy to manage.

Annotations:

There are several annotations we used in our project to make the entities easier to understand. For example, the area entity looks like this:

```
1  @Data
2  @Entity
3  @Table(name = "gebiet")
4  public class Area {
5      @Id
6      @GeneratedValue(strategy = GenerationType.IDENTITY)
7      @Column(name = "gebietid")
8      private Long areaId;
9
10     @Column(name = "bezeichnung")
11     private String desc;
12 }
```

Listing 5: Area Entity

In this class, the `@Data` annotation is used to generate necessary methods like `Getters` and `Setters`. `@Entity` ensures that JPA actually recognizes the class as an entity and `@Table` is used to give the table of this class a custom name. Furthermore, `@Id` sets an identifier for the entity. In this case the ID is getting generated automatically with the `@GeneratedValue` annotation.

Unique identifier:

The most important aspect of the backend is handling all the addresses. Every address has a unique identifier, which is defined by a `house number` and a `street`. Therefore, we cannot use a single primary key as an identifier for each address.

The `AddressId` class is used to manage this composite key. which ensures that each address has its own unique identifier without requiring the generation of an artificial identifier.

```
1  @Data
2  public class AddressId implements Serializable {
3      private String houseNumber;
4      private Long streetId;
5  }
```

Listing 6: AddressId

It is shown in the code snippet from the AddressId class that it uses the mentioned variables as a combination of identifiers, which then results in us not needing to make an artificial identifier like a number that goes up every time the admin adds a new address.

These composite keys are the best practice if it is not possible to use only one variable as the primary key. This is how the `AddressId` class is added to the address entity:

```
1  @IdClass(AddressId.class)
2  public class Address {
3      @Id
4      @Column(name = "Hausnummer")
5      private String houseNumber;
6
7      @Id
8      @Column(name = "Strasseid")
9      private Long streetId;
10 }
```

Listing 7: IDs in Address-Entity

The annotation `@IdClass` has to be used to define what class is used to define the variables of the primary key. Afterwards, the corresponding variables, which are in our case

`houseNumber` and `streetId`, have to be defined as variables in the entity class. Also, the annotation `@Id` has to be used over the variables in the entity class to set them as actual IDs.

Variables in Entities:

There are several types of variable types in an entity class. First, the "normal" variable, which has no connection to other entities or is in the ID in any way. These get used for data that is not related to any other entities. This is an example from the address entity of such a variable:

```
1  @Column(name = "latitude")
2  private double latitude;
3
4  @Column(name = "longitude")
5  private double longitude;
6
7  @Column(name = "schonbesucht")
8  private Boolean alreadyVisited;
```

Listing 8: "Normal" Variable

In this case, these variables save the location data of each address and whether the address had already been visited. It uses the `@Column` annotation, which is needed because sometimes the name of the column in the database table and the name of the variable in the code may not match for better handling in the code. Otherwise, the variables are just defined with their respective data types.

The other type of variables are related variables that have a connection to another entity in some way.

Listing 9: ManyToOne Variable

```
1  @ManyToOne
2  @JoinColumn(name = "gebietid")
3  private Area area;
4
5  @ManyToOne
6  @JoinColumn(name = "besonderheitid")
7  private SpecialFeature specialFeature;
```

This code snippet defines two variables in our address entity that use the `@ManyToOne` annotation. This is crucial because each address belongs to an area and can have a special feature. We use the IDs from the other entities to make sure that the relationship of the two entities are always right. In this case, because it is a ManyToOne relationship, we did not need to define the address entity in the other classes.

Sometimes the variables have more complex relationships than just with a simple identifier that gets compared. We had such a case in our project, and it came out like this:

```
1 @ManyToOne(cascade = CascadeType.PERSIST)
2 @JoinColumns({
3 @JoinColumn(name = "Strasseid", referencedColumnName = "Strasseid",
  ↪  insertable = false, updatable = false),
4 @JoinColumn(name = "Postleitzahl", referencedColumnName = "Postleitzahl",
  ↪  insertable = false, updatable = false)
5 })
6 private Street street;
```

Listing 10: Complex ManyToOne Variable

This code snippet defines a variable which is related to the street entity. The street entity also has a composite key as its ID so we had to use the `@JoinColumns` annotation to have a reference to the composite key of the street entity.

10.3 JPA-Repositories (DB Access and CRUD Operations)

JPA repositories are a crucial part of the backend to simplify database access by providing methods for many different CRUD (Create, Read, Update, Delete) operations. We utilize repository interfaces that extend `JpaRepository` instead of manually implementing all these methods that is given to us by the `JpaRepository`. For example, our `AddressRepo` interface gets defined like this:

```
1 public interface AddressRepo extends JpaRepository<Address, Integer>{}
```

Listing 11: AddressRepo Interface

The interface extends `JpaRepository` with the Address being the entity that gets influenced by the CRUD (Create, Read, Update, Delete) operations and `Integer` being an id.

A JPA repository is an interface rather than a normal Java class. We extend an interface with the `JpaRepository`. By that, we inherit many built-in database methods that make data management significantly easier. Almost any entity class has its own repository interface, which allows us to retrieve, check and manipulate database entries with simple methods calls instead of writing the methods ourselves.

Such a repository can also set custom methods based on variable names. For example, this code snippet finds a street based on its name and postal code:

```
1 Street findStreetByNameAndPostalCode(String name, String postalCode);
```

Listing 12: Find Street By Name And Postal Code

As JPA automatically recognizes the method names, these queries do not require any explicit SQL statements.

Custom Queries with @Query

Although JPA simplifies database interactions in many ways, there are some specific cases where automatic query generation does not work. In our project, we encountered an issue where Hibernate had problems correctly inserting a new address into the table. To solve this problem, we had to create this custom SQL query in the repository:

```
1 @Modifying
2 @Transactional
3 @Query(value = "INSERT INTO adresse (hausnummer, strasseid, postleitzahl,
  ↪  latitude, longitude, besonderheitid, gebietid, schonbesucht,
  ↪  kommentar)" +
4        "VALUES (:houseNumber, :streetId, :postalCode, :latitude,
         ↪  :longitude, :specialFeatureId, :areaId, :alreadyVisited,
         ↪  :comment);",
5        nativeQuery = true)
6 int insertAddress(@Param("houseNumber") String houseNumber,
```

```
7                   @Param("streetId") Long streetId,
8                   @Param("postalCode") String postalCode,
9                   ...);
```

Listing 13: Insert Address with Custom Query

This method inserts a new address with all its variables into the database, but it requires additional annotations.

- `@Query:` Specifies the custom SQL query that has to be executed.
- `@Modifying:` Indicates that this particular query modifies the database, which would not be needed if it were only a `SELECT` statement.
- `@Transactional:` Ensures that the query runs in a transaction, which prevents issues like incomplete data insertion.
- `@Param:` Is needed so the parameters are placeholders in the SQL statement.

10.4 Service Classes

Service classes contain the logic for the database interaction and serve as a bridge between the controllers and repositories. Before passing data to controllers or repositories, they handle data retrieval and manipulation of the data.

10.4.1 @Service Annotation

The `@Service` annotation is used to define a class as an actual service component. It originates from the `@Component` annotation, which makes it eligible for Spring's component scanning and dependency injection.
Every service class interacts with at least one repository, which gets injected through a constructor-based dependency injection using the `@Autowired` annotation. This injection looks like this in the service class:

```
1  @Service
2  public class AddressService {
3      private final AddressRepo addressRepo;
4      private final AreaRepo areaRepo;
```

```
5       ...
6
7       @Autowired
8       public AddressService(AddressRepo addressRepo, AreaRepo areaRepo,
        ↪  StreetRepo streetRepo, SpecialFeatureRepo specialFeatureRepo) {
9           this.addressRepo = addressRepo;
10          this.areaRepo = areaRepo;
11          ...
12      }
13  }
```

Listing 14: Repository Injection and @Service

Here `AddressService` depends on many repositories, and Spring automatically injects the required interface.

10.4.2 Repository Usage in Service Methods

Now that the service class has all the necessary repositories, it has to use the methods too. Some common operations look like this:

This method returns all addresses in the database:

```
1   public List<Address> getAllAddresses() {
2       return addressRepo.findAll();
3   }
```

Listing 15: Fetching all Addresses

In this case, it toggles if the address has already been visited or not:

```
1   public Address toggleAlreadyVisited(String houseNumber, Long streetId) {
2       Address address =
        ↪  addressRepo.findAddressByHouseNumberAndStreetId(houseNumber,
        ↪  streetId);
3
4       address.setAlreadyVisited(address.getAlreadyVisited() == null ||
        ↪  !address.getAlreadyVisited());
5
6       log.info(address.toString());
```

```
7
8       return addressRepo.save(address);
9   }
```

Listing 16: Toggle Already Visited

And that inserts a new address with the custom query from the `AddressRepo`:

```
1   @Transactional
2   public void createNewAddress(
3       String houseNumber,
4       double lat,
5       double lng,
6       ...
7   ) {
8       int insert = addressRepo.insertAddress(
9                       address.getHouseNumber(),
10                      address.getStreetId(),
11                      address.getStreet().getPostalCode(),
12                      ...
13      );
14  }
```

Listing 17: Insert new Address

These methods from the repositories are used in almost every function of the service class. From retrieving data to creating a new address, there are many methods that are built-in or can be easily created in the repository classes that get used by the service classes.

10.4.3 Computing Border Addresses Using Convex Hull

An important function of the backend is to calculate the border addresses that are around a given area. This function is implemented in the `AddressService` class. In this case, the `getBorderAddresses` method calculates the convex hull from this given area. The main method uses many helper methods and an extra class, so the calculation does not use that much resources.

Point Class

To calculate the convex hull, we use the longitude and latitude of the addresses in an area. Processing every address with all its fields would cost much more resources than only using the longitude and latitude of every address. That is the purpose of the `Point` class. It looks like this:

```
1  @AllArgsConstructor
2  @Data
3  public class Point {
4      private double longitude;
5      private double latitude;
6  }
```

Listing 18: Point Class

This class only saves the longitude and latitude of an address to save a bit more resources.

Helper Methods

There are some methods we created that get used in the main method to get the border addresses to make it easier to understand what each method does. **PolarAngle** The `polarAngle` method is used to calculate the polar angle between two points so that it can be used to find out which points are the nearest and can be compared to each other next. This method is implemented like this:

```
1  public double polarAngle(Point origin, Point point) {
2      return Math.atan2(point.getLongitude() - origin.getLongitude(),
       ↪  point.getLatitude() - origin.getLatitude());
3  }
```

Listing 19: Polar Angle Method

The method gets two points and compares them to each other with the `Math.atan2` method, which is provided by the `Math` package.

IsCounterClockwise

The `isCounterClockwise` method is used to check if three points are counterclockwise. This is important to make sure the convex hull is actually convex and not concave. The method to calculate this looks like this:

```
1 public boolean isCounterClockwise(Point p1, Point p2, Point p3) {
2     return (p2.getLatitude() - p1.getLatitude()) * (p3.getLongitude() -
      ↪  p1.getLongitude()) - (p2.getLongitude() - p1.getLongitude()) *
      ↪  (p3.getLatitude() - p1.getLatitude()) > 0;
3 }
```

Listing 20: is Counter Clockwise Method

Convex Hull

The `convexHull` method uses the helper methods and other logic to return the convex hull. The method gets a list of points.

The `convexHull` method starts with a clause that may directly stop the calculation. To build a convex hull, there have to be at least four points, so if there are less than four points, the method returns an empty list.

Listing 21: if Clause for Point Size

```
1 if (points.size() < 3) {
2     return Collections.emptyList();
3 }
```

If there are enough points, the actual `convexHull` can start. First, it finds the lowest point based on the latitude or the leftmost point if the latitudes of the two lowest points are the same. It goes through all the points given to the method and compares the latitude.

```
1 Point lowest = points.stream()
2     .min(Comparator.comparing(
3             Point::getLatitude
4         ).thenComparing(Point::getLongitude))
5     .orElse(null);
```

Listing 22: Lowest Point Calculation

Then, using one of the helper methods, the program sorts the points based on their polar angles to each other. This is relatively easy to do by given a comparator provided by Java.

```
points.sort(Comparator.comparingDouble(p -> polarAngle(lowest, p)));
```

Listing 23: Sort Points Based on Polar Angle

Afterwards, the method creates a stack and adds the first two points of the sorted list to the stack.

```
Stack<Point> stack = new Stack<>();
stack.push(points.get(0));
stack.push(points.get(1));
```

Listing 24: Create Stack

After the stack is created, the program goes through a loop that goes through every point in the list. In the loop, it creates a new variable that always gets the last added point of the stack and uses it as a reference point. Then the other helper method is used to look if the next point of the list is in the convex hull. If the next point is in the convex hull, then it gets added to the stack. This stack is then returned as a list for later processing.

```
for (int i = 2; i < points.size(); i++) {
    Point top = stack.pop();

    while (!stack.isEmpty() && !isCounterClockwise(stack.peek(), top,
    ↪  points.get(i))) {
        log.info("Popping point: " + top);
            top = stack.pop(); // Removes top until the border makes a
            ↪  left turn
        }

    log.info("Pushing point: " + points.get(i));

    stack.push(top);
    stack.push(points.get(i));
}

return new ArrayList<>(stack);
```

Listing 25: Convex Hull Calculation

In the `getBorderAddresses` method, the `convexHull` method is used, and the con-

vex hull points are stored in a list. Furthermore, this list gets used to add a tolerance to every point. This is the case because we encountered some issues while creating this convex hull, and we found out by adding a bit of tolerance that this issue was fixed. This is because the locations of the addresses are not precise enough. After adding this tolerance, the list of points that build the convex hull is returned.

```
1  List<Point> hull = convexHull(points);
2
3  List<Address> borderAddresses = new ArrayList<>();
4
5  for (Point p : hull) {
6      for (Address address : addresses) {
7          double latDiff = Math.abs(address.getLatitude() -
           ↪  p.getLatitude());
8          double lonDiff = Math.abs(address.getLongitude() -
           ↪  p.getLongitude());
9
10         if (latDiff < TOLERANCE && lonDiff < TOLERANCE) {
11             borderAddresses.add(address);
12             break;
13         }
14     }
15 }
16
17 return borderAddresses;
```

Listing 26: Lowest Point Calculation

10.5 Rest Controller (API Endpoints and their Functions)

REST controllers expose API endpoints that handle HTTP requests and then return the appropriate responses. These controllers allow the frontend interfaces (Admin Panel and mobile app) to interact in any way with the backend to perform operations like retrieving, updating and deleting data. In our project, two main REST controllers are responsible for handling all the tasks: the `AddressController` and the `AdminController`.

10.5.1 AddressController

The `AddressController` class is for handling endpoints for the guides and the mobile app and provides the necesssary functions to retrieve and update certain information. Each method in this controller corresponds to a certain operation the guides can do. These are the endpoints the `AddressController` exposes to the network:

GET Routes

1. **`GET /` - Get All Addresses**
    - This endpoint gets a list of all addresses stored in the system.
    - It is mostly used to display the addresses in the mobile app.

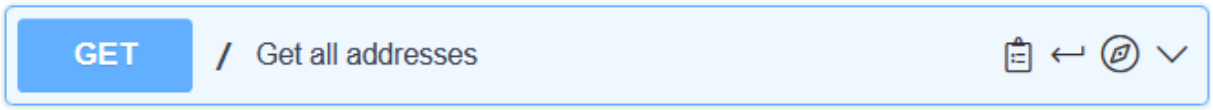

Fig. 27: Get Route for all Addresses

2. **`GET /area` - Get All Addresses Of An Area**
    - This endpoint allows users to get a list of addresses from a specific area.
    - The area is provided as a path variable

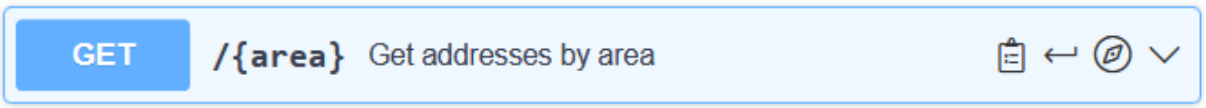

Fig. 28: Get Route for all Addresses by Area

3. **`GET /area/borderAddress` - Get Border Addresses**
    - This endpoint gives the frontend the border addresses of the convex hull of an area.

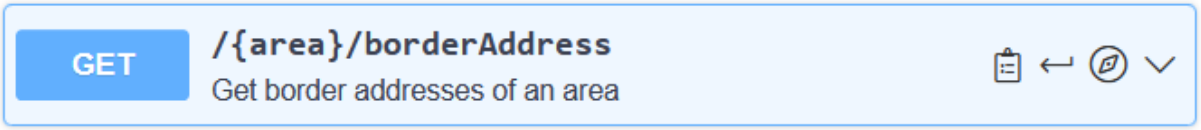

Fig. 29: Get Route for Border Addresses by Area

PUT Routes

1. `PUT /streetId/houseNumber/toggleVisited` **- Toggle Visited Status**
    - This endpoint allows toggling the "visited" status of a specific address.

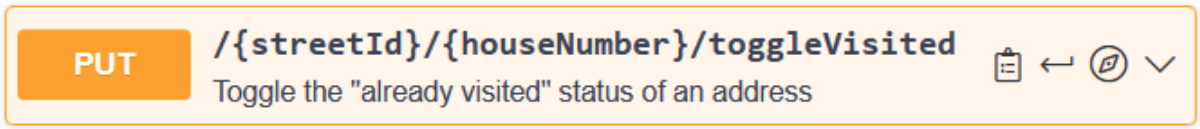

Fig. 30: Put Route to Toggle Visited

2. `PUT /streetName/postalCode/toggleAlreadyVisitedOnStreet`
    - This endpoint updates the "visited" status for all addresses on a particular street, marking them as either visited or not.

Fig. 31: Put Route to Toggle Whole Street Visited

10.5.2 Admin Controller

This controller has all the endpoints relevant to managing the data. These routes are not meant for the guides from the mobile app, they are here for the admin in the Admin Panel. These routes are for the admin to use:

GET Routes

1. `GET /getAllStreets` **- Get All Streets**
    - This endpoint retrieves a list of all streets in the system.

Fig. 32: Get Route for all Streets

2. `GET /getAllAreas` - **Get All Areas**

    - This endpoint returns a list of all areas in the system.

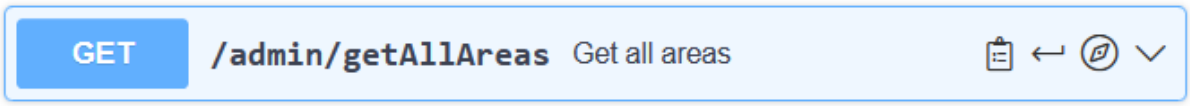

Fig. 33: Get Route for all Areas

3. `GET /getAllSpecialFeatures` - **Get All Special Features**

    - This endpoint retrieves all special features present in the system.

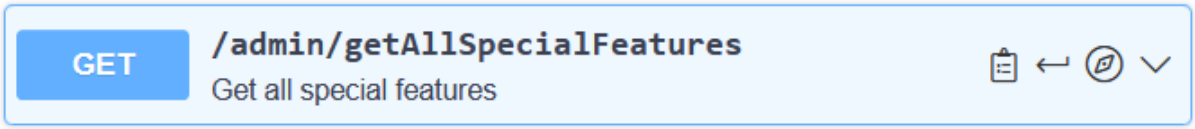

Fig. 34: Get Route for all Special Features

PUT Routes

1. `PUT /updateComment` - **Update Address Comment**

    - This endpoint allows updating the comment associated with an address.

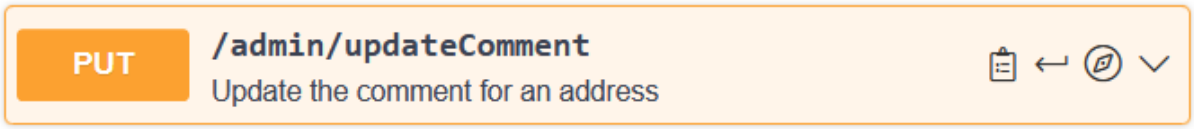

Fig. 35: Put Route to Update Comment on Address

2. `PUT /updateAddressesByRange` - **Update Addresses by Range**

    - This endpoint updates the area of addresses based on a range of house numbers within a street

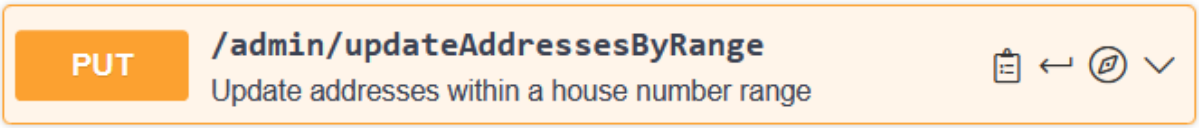

Fig. 36: Put Route to Update Addresses by Range

3. `PUT /updateAddressesByParity` - **Update Addresses by Parity (Odd/Even)**
    - This endpoint updates addresses based on their house number parity (odd or even).

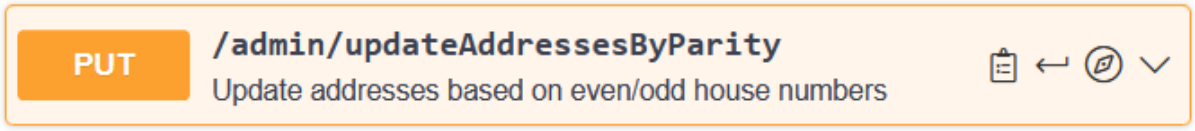

Fig. 37: Put Route to Update Address by Odd/Even

4. `PUT /updateAddressesByList/areaDesc` - **Update Addresses by List**
    - This endpoint allows bulk updates to a list of addresses, changing the any variable of each address in the provided list.

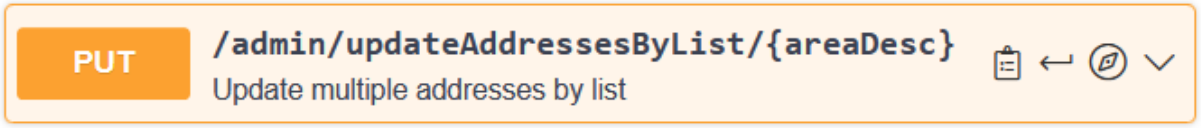

Fig. 38: Put Route to Update all Addresses in a Custom List

5. `PUT /editArea` - **Edit Area Information**
    - This endpoint allows modification of an area's description and associated details.

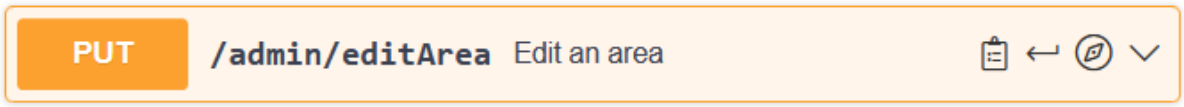

Fig. 39: Put Route to Edit an Area

6. `PUT /setSpecialFeature` - **Set Special Feature for Address**
    - This endpoint assigns a special feature to an address.

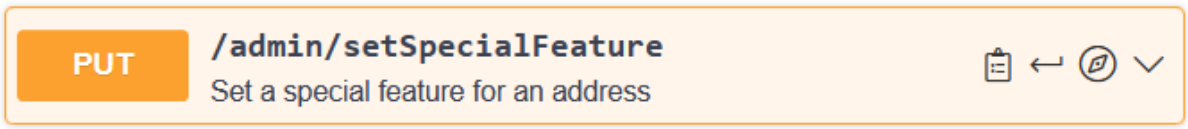

Fig. 40: Put Route to Set Special Feature on Address

7. **`PUT /editAddress` - Edit an Address**

    - This endpoint allows the modification of an existing address, such as updating its house number, street name, or additional details.

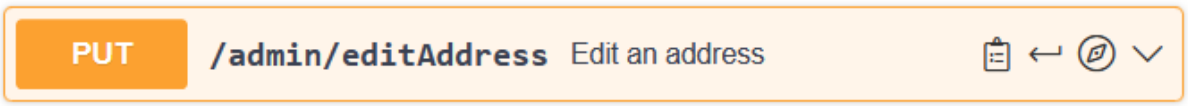

Fig. 41: Put Route to Edit an Address

8. **`PUT /editAddresses` - Edit Multiple Addresses**

    - This endpoint allows bulk edits to multiple addresses at once, facilitating large-scale updates.

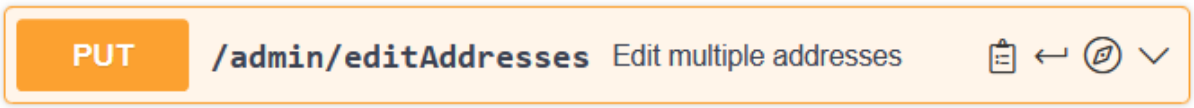

Fig. 42: Put Route to Edit Multiple Addresses

9. **`PUT /editSpecialFeature` - Edit Special Feature**

    - This endpoint allows editing a special feature that is already existing in the database.

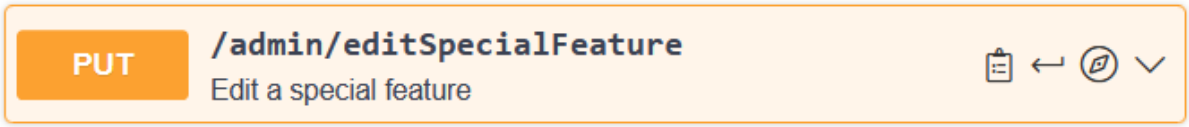

Fig. 43: Put Route to Edit Special Feature

POST Routes

1. `POST /addAddress` **- Add a New Address**
    - This endpoint allows the creation of a new address record.
    - All relevant details such as house number, street name, postal code, and coordinates must be provided.

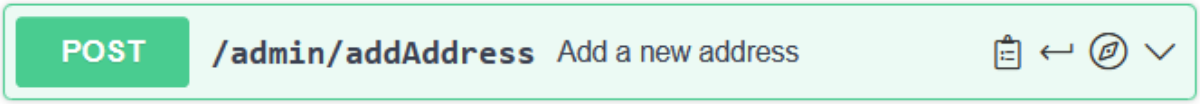

Fig. 44: Post Route to Create an Address

2. `POST /addStreet` **- Add a New Street**
    - This endpoint creates a new street record, specifying its name and postal code.

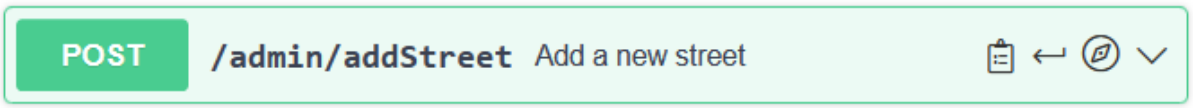

Fig. 45: Post Route to Create a Street

3. `POST /addArea/areaDesc` **- Add a New Area**
    - This endpoint allows the addition of a new area to the system.

Fig. 46: Post Route to Create an Area

4. `POST /addSpecialFeature` **- Add a New Special Feature**
    - This endpoint creates a new special feature.

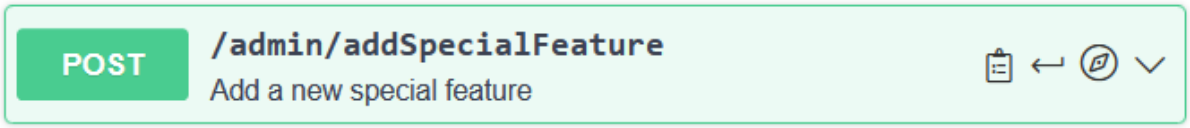

Fig. 47: Post Route to Create a Special Feature

DELETE Routes

1. `DELETE /deleteStreet` **- Delete a Street**
    - This endpoint deletes a street from the system.

Fig. 48: Delete Route to Delete a Street

2. `DELETE /deleteArea` **- Delete an Area**
    - This endpoint deletes an area from the system.
    - All addresses in that area may need to be reassigned or deleted as well.

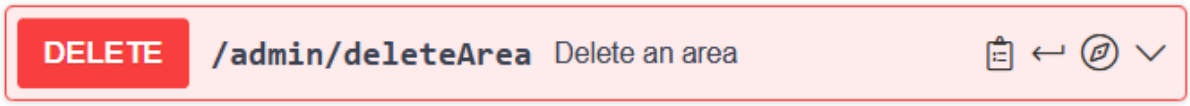

Fig. 49: Delete Route to Delete an Area

3. `DELETE /deleteSpecialFeature` **- Delete a Special Feature**
    - This endpoint removes a special feature from the system.

Fig. 50: Delete Route to Delete a Special Feature

4. `DELETE /deleteAddress` **- Delete an Address**
    - This endpoint deletes an address from the system.

Fig. 51: Delete Route to Delete an Address

11 GraphHopper Setup

For the feature of marking a street that a group needs to visit, we researched a way to calculate a route between multiple different coordinates. But because this route needed to follow the street and could not be just a straight line, we ended up using the routing engine GraphHopper.

11.1 Why use GraphHopper?

First, GraphHopper, since it is an open-source tool, has decent documentation, so it is easy to get started with. It met all our requirements, so we are able to request a route between multiple and not only two coordinates, and it could be self-hosted on our own server, to avoid having to pay fees for API-Tokens. Additional, it was great for development to not be restricted in how many points to route between. The API-Interface is well documented, and there are many example requests provided on their official website. Lastly, GraphHopper provides a simple setup, so getting your own server up and running can be a matter of minutes.

11.2 Local hosting

GraphHopper comes as a .jar file. This is a Java executable and can therefore be run and used on every computer with a recent Java version installed. To start the routing engine, you first need a configuration file in which you can specify where the data can be found on your server, as well as other settings for your server, like the port it should be run on.

11.2.1 Configuration

For the configuration of our GraphHopper instance, we created the following file:

```
graphhopper:
    datareader.file: austria-latest.osm.pbf
    graph.location: austria-gh
    profiles:
      - name: foot
        weighting: custom
        custom_model:
          speed:
            - if: true
```

```
10              limit_to: 5
11    import.osm.ignored_highways: motorway,trunk
12
13 server:
14    application_connectors:
15      - type: http
16        port: 46381
17        bind_host: localhost
18    root_path: /streets/*
19    request_log:
20      appenders: []
```

This file defines the location of the file that contains the initial data, as well as the directory where the calculated information gets stored, as well as the foot profile. In the *server* section, parameters like the request path or port get defined. This will be needed later for making the instance accessible through a reverse proxy managed by NGINX.

11.2.2 Deployment

To deploy our instance and open the API for requests, we need to copy the map data, the .jar executable, and our configuration file onto the server. Afterwards, we need to execute the jar file to start our GraphHopper server. We need to tell GraphHopper in which mode to start, here we use the *server mode* and where our config.yml file is located. Additionally, we need the & to run it in the background so, once the executing terminal is closed, the server will not shut down. We end up with this command:

```
java -jar graphhopper-web.jar server config.yml &
```

12 Working out the Wireframes

Before even starting the development of our app, we decided to mock up the user interface. This turned out to be very helpful during implementation because we had an idea of how things should look and did not need to think about design choices while writing code. We used Figma to create these wireframes.

12.1 Map View

This map view is the center point of our app. Here, the addresses that still need to be visited get displayed, and comments can be added. The user can move, zoom, or rotate the map as well as click on markers to see details and edit the comment field. There are also visual indicators for addresses that are already commented, as well as different icons depending on the specification of an address. On the bottom of our app is a navigation bar that stays consistent across the different screens and is used to switch between them. Our primary goal was to keep the map as simple as possible to avoid distraction or overwhelming the user.

Fig. 52: Initial wireframe

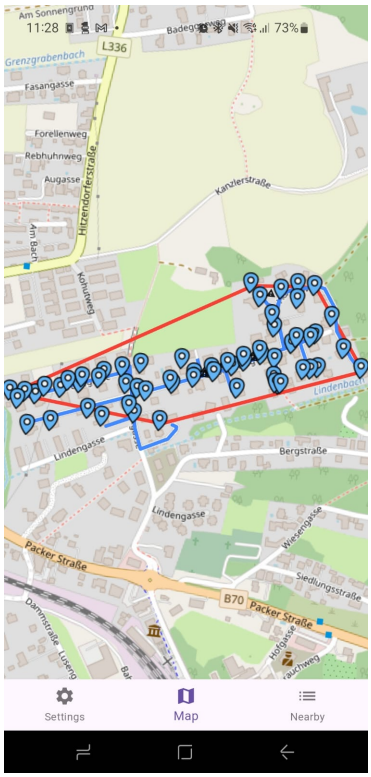

Fig. 53: Final screen in app

12.2 List View

This screen consists of a list in which all addresses that are in the selected area get displayed. They are grouped by the street and are sorted ascending by the house number. The user can swipe on either individual house items or the street name to change the `alreadyVisited` field. When swiping on a whole street, the user has to secondly approve that they really want to toggle all the addresses of this street. Initially it was planned to also incorporate the distance to a specific address into this list. This feature was later abandoned because of time constraints.

There were two different ideas to integrate this view into the app. First, using a panel that can be pulled out of the right side of the screen, but during development, we chose to just make it its own page for a simpler-to-navigate experience. Also, the swipe gesture was added later, after the initial planning. In the wireframe, there were checkboxes that indicated the state.

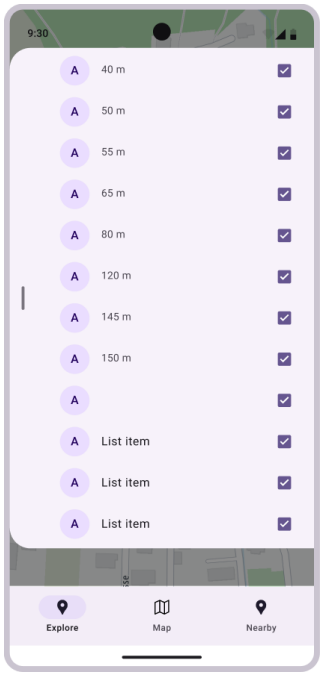

Fig. 54: Initial idea of List view with pull-out panel

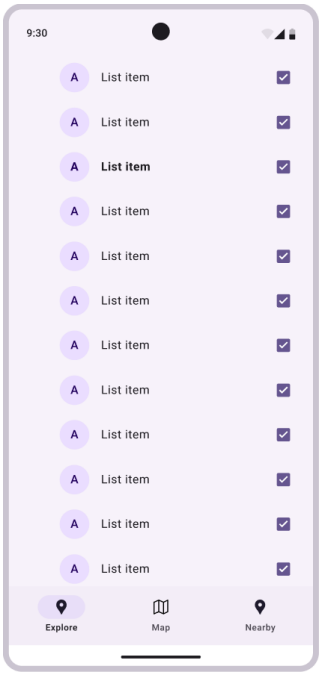

Fig. 55: List view with different opening concept

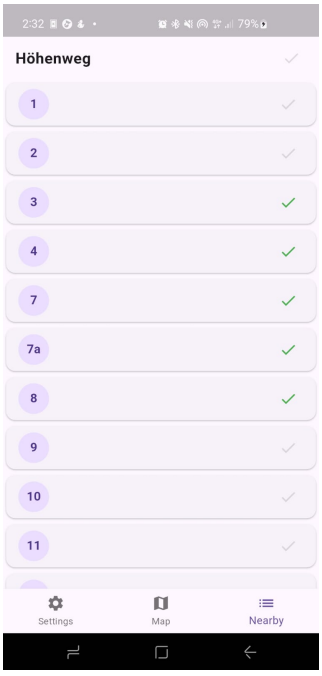

Fig. 56: Final implementation of List view with swiping

In the end, a few things changed, but the basic concept has stood since the project started.

12.3 Details View

This view was designed to not be a whole separate screen from the beginning, but it was basically completely reworked during development. At first, it was planned to be a sliding panel, just like the list view. It ended up being just a small pop-up that gets displayed when a user taps on a marker. In this dialog, the house number, street name, postal code and a text field for comments get displayed. Through two color-coded buttons, the comment can be either saved or discarded.

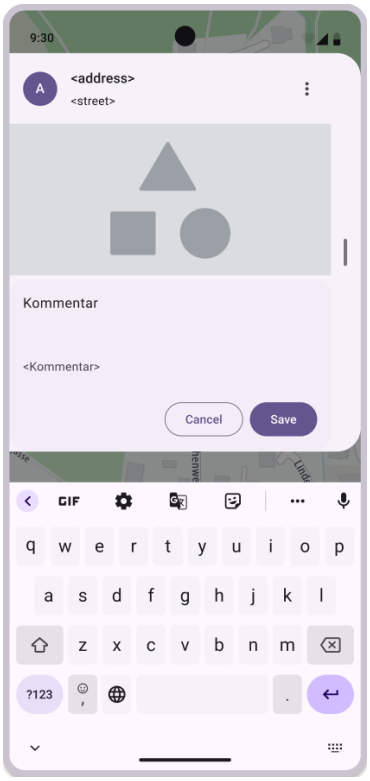

Fig. 57: Wireframe of details slider

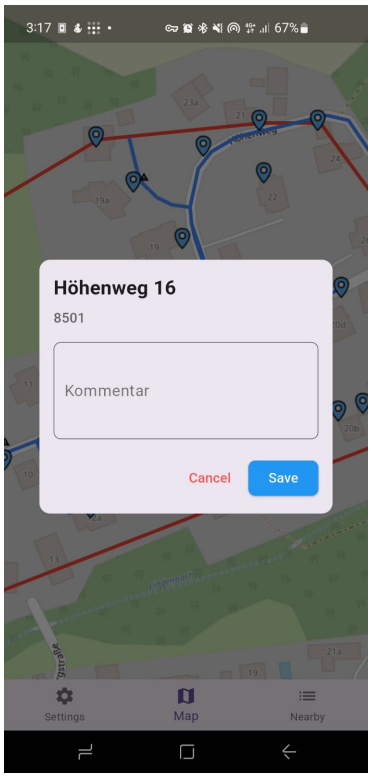

Fig. 58: Final details pop-up

12.4 QR-Code Scanner

This page was initially not planned or mocked up, since it was still unclear how a user would get their area assigned. Eventually we decided on QR-Codes for their ease of use. But to be able to scan the codes, which can be printed from the Admin Panel, we needed to implement a simple scanning mechanism. In the end, we created a single new page with just a button that starts the scanner. This page can and will be expanded in future releases to feature additional settings like an optional dark mode or user-changeable colors.

12.5 Possible improvements for future versions

In a future iteration of the app, the details' dialog should be accessible from the list view, also, according to the feedback we got from legitimate users of our app, it would be nice to have more states than just visited or not. For example, visited, but nobody was home, will not visit again this year, or visited, somebody was at home, but wishes to not be visited anymore. To implement this, the entire list screen needs to be completely redesigned. Additionally, the feature that the user can see which address is closest to them would probably be useful. Furthermore, reliably displaying the user's current location would be appreciated.

13 Functional implementation behind the application

In this section of our thesis, the functional implementation of the mobile app will be described. We will show how the FlutterMap component is built and what logic works in the background.

13.1 Address-Provider

Since without data, our app would be useless, we will start with describing the Address-Provider class. It is implemented using the `provider package` by Flutter. With its help, we can easily notify all screens and widgets when the data was updated.

The main objective of the Address-Provider is to communicate with the backend and Graph-Hopper instance, but also to bring the received data into the correct format for our frontend. For this, we defined four different instance variables. A list that stores all addresses, one that only holds the coordinates of the `border addresses` returned by the backend, a map that is used for coordinates of the routing points for a specific street (*Map<String[street name]>, List<LatLng>[coordinates]*) and finally a map for storing the addresses grouped by their street. Every component that owns an instance of this Address Provider can access these variables, which makes it easy to display the data.

Due to the fact that the backend returns the addresses unsorted, the Address Provider also takes on this task. For the grouping by the street name, we can just use the map from before, but to then also sort the addresses ascending by their house number, we needed to write a function to compare them.

The *compareHouseNumbers* function performs natural sorting of house numbers by splitting them into alternating numeric and non-numeric components. It achieves this using a regular expression that extracts sequences of digits and non-digit characters separately. The function then iterates through these parts, comparing numeric segments as integers to ensure correct numerical order and comparing non-numeric segments lexicographically. This approach ensures that house numbers such as "10a" and "10c" are sorted in an intuitive order, with "10a" preceding "10c". If all corresponding parts are equal, the function resolves ties by comparing the lengths of the extracted parts lists. This method improves the readability and

usability of address lists by maintaining a logical, human-friendly order.

```
1   int compareHouseNumbers(String a, String b) {
2     final regex = RegExp(r'(\d+|\D+)');
3
4     final partsA = regex.allMatches(a).map((match) =>
      ↪ match.group(0)!).toList();
5     final partsB = regex.allMatches(b).map((match) =>
      ↪ match.group(0)!).toList();
6
7     for (int i = 0; i < partsA.length && i < partsB.length; i++) {
8       final partA = partsA[i];
9       final partB = partsB[i];
10
11      //compare numeric part
12      if (int.tryParse(partA) != null && int.tryParse(partB) != null)
        ↪ {
13        final numA = int.parse(partA);
14        final numB = int.parse(partB);
15        if (numA != numB) return numA.compareTo(numB);
16      } else {
17        // Compare alphabetic part
18        final comparison = partA.compareTo(partB);
19        if (comparison != 0) return comparison;
20      }
21    }
22
23    return partsA.length.compareTo(partsB.length);
24  }
```

Listing 27: compareHouseNumbers function (AddressProvider.dart)

13.2 HTTP-Requests

To actually access the data from our database, we built our backend API. It can be accessed through different HTTP-Requests. The mobile application makes use of the following routes:

- **fetch all addresses of area**

  As the name suggests, it fetches and saves all addresses from the area that is set through the QR-Code. Here the *compareHouseNumbers* function gets used.

```
1        addressMap = groupBy(addresses, (Address address) =>
         ↪  address.street.name);
2
3        addressMap = addressMap.map((key, value) {
4          value.sort((a, b) => compareHouseNumbers(a.houseNumber,
           ↪  b.houseNumber));
5          return MapEntry(key, value);
6        });
```

Listing 28: use of compareHouseNumbers function (AddressProvider.dart)

- **fetch border addresses of area**

  Fetches and extracts the coordinates of the border addresses returned by the backend.

```
1        borderAddresses = decodedJSON.map((json) =>
         ↪  Address.fromJson(json)).toList().map((point) =>
         ↪  LatLng(point.latitude, point.longitude)).toList();
```

Listing 29: extraction of coordinates from border addresses (AddressProvider.dart)

- **toggle address visited state**

  Used by the list screen, on swipe of an individual address-item.

- **toggle street visited state**

  Virtually the same as the toggle address, but now for the whole street. Gets called when a user swipes and confirms the change on a whole street.

- **update comment of address**

  Used by the *details pop-up* on save with a new comment in the text field.

- **fetch route information for streets**

  Call to our GraphHopper instance. Iterates over every key of the *addressMap*, requests the routing data, and calls the function *extractPoints* to extract only valid data out of the whole JSON response.

13.2.1 Performance

To improve our performance, the first thing to change would be to move away from HTTP-Request to Web-Sockets. This would be a more suitable technology since only when the data on the server updates do we get an HTTP-Frame. Through this change, we could drastically minimize the traffic our app produces, as well as the performance, since the widgets do not need to re-render constantly.

13.3 Implementation of the FlutterMap Component

The FlutterMap widget is built up in layers, which all can be individually ordered and configured. In this part, the layers that we utilized and the configuration options we set will be described.

- **Tile-Layer**

  This layer imports the tiles, so the actual map of OpenStreetMap's tile server. In future iterations, a different server or tile style (vector tiles) could be used to improve performance and usability further.

- **Polygon-Layer**

  In this layer, we draw the polygon that marks the border of the selected area. To achieve this, we just create a new Polygon object with the coordinates saved in the *borderAddresses* list of the *AddressProvider* and set a few styling options, like the stroke width and border color.

- **Polyline-Layer**

  This layer was utilized to display the streets that correspond to the specified area. Here we draw different polylines based on the routing data we fetched from our GraphHopper server. In the future, this feature could be further improved so different streets get drawn in varying colors. For now, every street polyline gets displayed in blue.

- **Marker-Layer**

  This is the final and arguably most important layer because here we set a marker for every address. This marker can vary, depending on the specification that the admin assigned to an address in the Admin Panel. This was implemented by decoding the Base64 string, which is saved in the address object and stores the marker image. Additionally, if a

comment is saved in the address, a little warning indicator gets displayed on the upper right-hand side of this encoded image. This got introduced so that users could see at a glance if there was something special to know about an address. We solved this by using a Stack widget. Additionally, we added a GestureDetector on top of this stack. This is used to open the details' dialog in which the user can modify the comment field.

13.4 Implementation of the List Screen

Now, we will take a look through our list component. The interactive elements here are the two *Dismissible* containers as well as the icon that indicates if an address has been visited. The logic behind the color-coded *Dismissibles* is just based on the *alreadyVisited* attribute of the address. There is also a function (*isWholeStreetVisited*) that checks if every address of a street has already been visited. When the users swipe the street item, a dialog gets displayed that prompts the user to confirm their action to avoid accidental swipes.

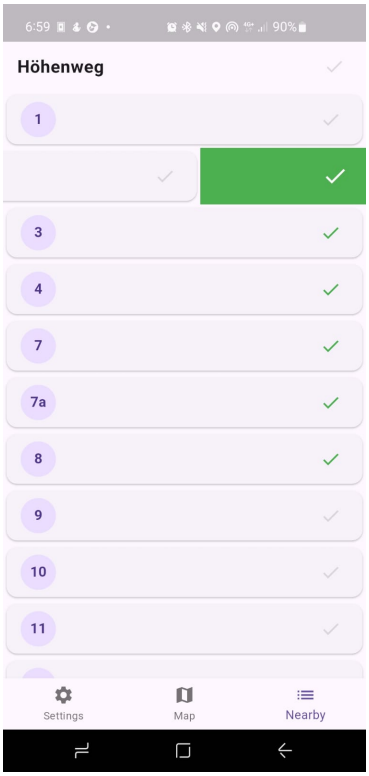

Fig. 59: Setting *alreadyVisited* true

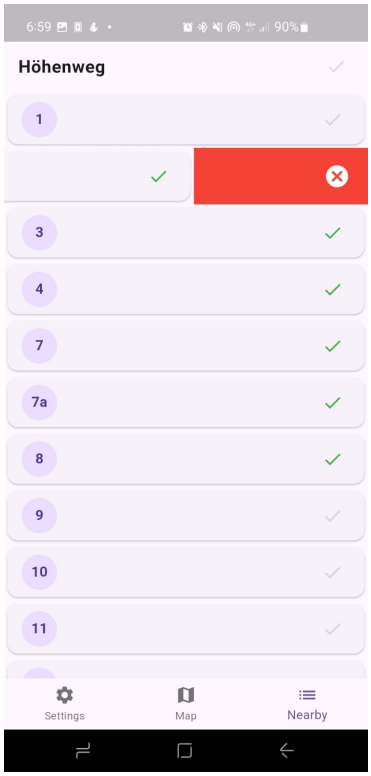

Fig. 60: Same for setting it false

13.4.1 Future outlook for List Screen

In the future, this component will likely be completely redesigned. This is due to the changed requirement that users want to be able to finer select the state of each address. This could

be resolved by replacing the sliding gesture with a Dropbox. Then also a filter should be implemented so addresses that have a specific state, like *visited, but nobody was home, will visit again*. Also, an *onClick* event will be defined that opens the details' dialog.

14 The app in use

In this section, the process of how the app got introduced to the users and their feedback will be examined.

14.1 Introducing new users

Exactly as agreed upon, in the year 2025 only four groups actually used our app during the campaign. We had a personal meeting with the supervisors of the mentioned groups. All of them could be categorized as individuals with an average understanding of technology. We demonstrated the functionality of the app briefly and explained what our goals were. Also, the administrator was at this meeting, so we quickly showed her around the Admin Panel. This was a very short rundown of the functionality, but nevertheless, she figured it all out. After this short meeting, we helped them with the installation and sent them on their way. During the campaign, we sometimes took a look into our Admin Panel and could see that each user reliably checked off her visited houses.

14.2 User Feedback

After the campaign, we once again came together for a short recap on the experience of using the app. There were a few key points that every one of the testers mentioned. First of all, the inclusion of more states for if an address was visited. This was the main talking point and will definitely be improved in a future version. Secondly. the inclusion of a live location was wished for, as well as the perception of long loading times. Overall, the feedback was very positive, and therefore the project will definitely be continued after the finalization of this thesis.

15 Implementation Admin Panel

The Admin Panel is an administrative control center to efficiently manage all addresses within a single environment. It is used for planning upcoming "Sternsinger" events, ensuring that all required addresses are covered and effectively distributed among participating groups. Additionally, it provides a way to assign areas containing addresses that a group needs to visit. This zoning feature ensures that each group is responsible for only the addresses within their designated area.

This tool enables authorized users to perform CRUD operations (Create, Read, Update, Delete) on addresses, streets, and areas. These features make it easy to address issues quickly and make changes to the areas that participants need to visit. For example, when a new street is added to the neighborhood, the administrator can update the system to include it and assign it to the appropriate area. Similarly, if a group withdraws from the event, the administrator can reassign any unvisited addresses to other groups. This ensures that the data remains up-to-date, allowing quick reactions to special cases, helping with the planning and execution of "Sternsinger" events.

This chapter will outline the implementation of the Admin Panel, detailing its components, functionalities, and widgets, and providing guidance on how to use them.

15.1 Navigation

To navigate between pages, a sidebar on the left is used, which can be toggled with a button in the top-left corner of the page. It displays a list of all pages, allowing users to switch between them with a click. The navigation is implemented in the `AdminNavigation` component. The menu contains a list of `ScreenItems` (see 15.5.2), which store the title with the corresponding page. These titles are displayed at the top of the screen above the page. To keep track of the currently selected page, an internal state (`indexState`) is used. Whenever a page is selected in the sidebar, the `indexState` is updated, and the corresponding page is displayed. This widget is the main component. It ensures that all pages are properly displayed.

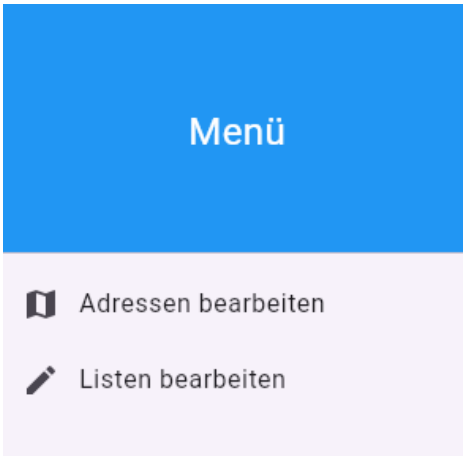

Fig. 61: Navigation in Admin Panel

15.2 AddressPage

The first page, called `AddressPage` provides an interface for managing address data, including creation, modification, and deletion. It uses a form with various input fields and a visualization component that displays addresses on a map or in a database view. The layout is divided into two parts:

- On the right side, all addresses are shown in either the `AdminMapComponent` (15.4.1), displaying addresses on a map, or the `DatabaseViewComponent` (15.4.2), showing them in a table.
- On the left side of the page, there are `InputFields` (15.6.1), which are used to enter new information about a new address or edit existing ones.

The following elements overlay the `AdminMapComponent`:

- A field to filter the addresses that are displayed.
- A button with a dropdown menu to select and edit a street.
- A switch to toggle between the `AdminMapComponent` and the `DatabaseViewComponent`.
- An information box in the bottom left corner to display `Notifications` (15.2.5) about the completed operations.
- A field in the bottom right corner to display the coordinates of the mouse pointer on the map.

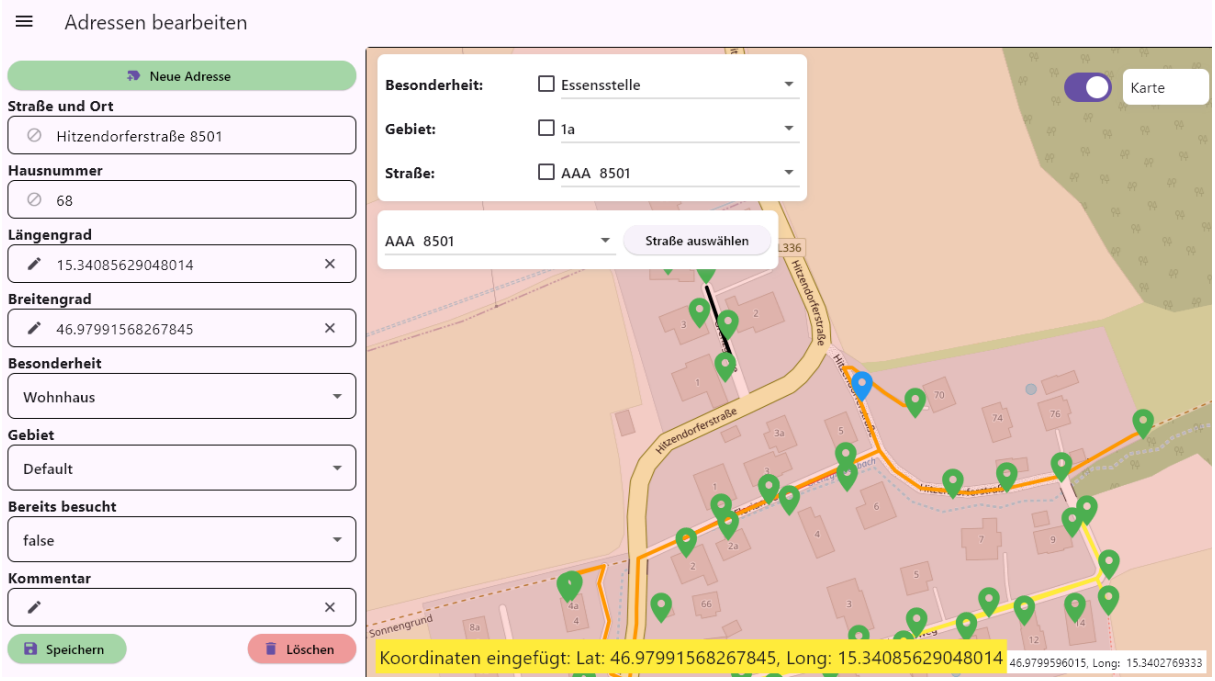

Fig. 62: AddressPage

15.2.1 Add Address

To add a new address, the "Neue Adresse" button is pressed. This triggers the `onClickNewAddress` method, which performs several key operations:

- All `InputFields` are cleared.
- The boolean variable `isNewAddress` is set to `true`, indicating that a new address is being created.

The cleared fields can then be filled with the new information. When the "Speichern" button is pressed, the `saveAddress` method is called. This method performs several validation checks, as shown in 15.2.4. If the address is valid, the `AdminAddressProvider` adds it to the database, a `notification` is displayed, and the newly added address appears.

When the map is clicked, the `onClickNewAddress` method is also invoked within the `AdminMapComponent`. The method receives the click coordinates as parameters to create the address at the selected location.

15.2.2 Edit Address

Existing addresses can be edited by selecting them in either the `AdminMapComponent` or the `DatabaseViewComponent`. The selection fills the `InputFields` with the information of the address, and in this case, the boolean variable `isNewAddress` is set to `false` to indicate that an existing address is being updated.

The administrator can then edit the information and press the "Speichern" button. This triggers the `saveAddress` method, which performs the same validation checks as when adding a new address. The `AdminAddressProvider` updates the selected address in the database, and the `notification` is displayed.

15.2.3 Delete Address

After selecting an address, the administrator can press the "Löschen" button to delete it. This triggers the `deleteAddress` method, which calls the `AdminAddressProvider` to delete the selected addresses.

The method also triggers the `showDeleteDialog` method, which displays an `AlertDialog` to confirm the action and prevent accidental deletions.

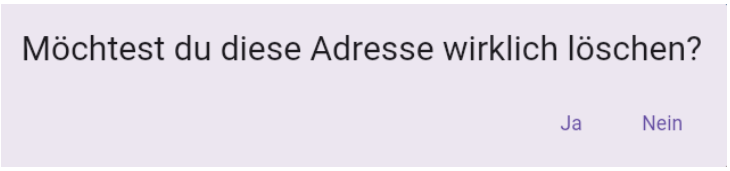

Fig. 63: Dialog to confirm deletion

15.2.4 Validation

Validation is the process of checking that data meets specific criteria before it is accepted and added to a system or database. This ensures that the data is accurate, consistent, and conforms to required standards. Validation prevents errors and inconsistencies that could arise from invalid or incomplete data, maintaining the integrity and reliability of the system. It helps catch mistakes early, such as missing fields or incorrect data inputs. By ensuring that only valid data is processed, validation plays a crucial role in preserving data quality. [Con25c]

isDuplicateAddress

To determine whether an edited or newly added address already exists in the system, the new address is compared against all existing addresses to detect potential duplicates before the address is saved. The comparison takes place within the `saveAddress` method, which handles the saving operation. If an address is found to be identical to an existing one, the method returns `true`, indicating that the address is a duplicate. If no match is found, it returns `false`, confirming that the address is unique and can be safely added to the database.

A duplicate address is identified by the following criteria: **street name, postal code, and house number**

InputField filled Validation

To verify that all required `InputFields` are properly filled, the `validateAddressFields` method is called.
To start the process, a new `Address` object is created and passed to the `validateAddressFields` method. The method checks each field within the `Address`. If a field is missing or incomplete, a `notification` (see 15.2.5) is displayed to inform the user of the missing data, such as "Strasse fehlt" or "Koordinaten fehlen." If all fields are correctly filled out, the method returns `true`, signaling that the address can be saved. However, if any field is incomplete, the method returns `false`, preventing the address from being saved until all required information is provided.

InputField Coordinates Validation

The `InputField` validates **latitude** and **longitude** inputs, ensuring correct coordinates. Validation is applied only when the `isCoordinateInput` parameter is set to `true`. In that case, the `inputFormatter` is passed to the `textfield`. This `inputFormatter` guarantees that only valid inputs are accepted. These are the three validators used:

- A `FilteringTextInputFormatter` with a regular expression is used to restrict the input to digits, decimal points, and an optional minus sign at the beginning.

```
r'^-?[0-9.]*'
```

Fig. 64: Regular expression for input validation

- A `TextInputFormatter` prevents multiple decimal points in a single number. If a user attempts to insert a second decimal point, the input is automatically rejected.
- The final `TextInputFormatter` ensures that the number of digits before the decimal point is limited to a maximum of 3 and the value itself does not exceed 180.

15.2.5 Additional Functionalities

This section outlines several functionalities that have been integrated to improve the overall usability of the application. These enhancements are specifically designed to simplify tasks while ensuring an intuitive user experience.

Notification

To notify the administrator about the success or failure of an operation, a `notification` is displayed at the bottom left, overlaying the `AdminMapComponent`. This `notification` appears when:

- An address has been added, edited, or deleted.
- Validation has failed.
- Coordinates are selected on the `AdminMapComponent` (15.4.1).

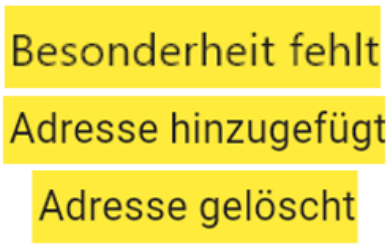

Fig. 65: Notification examples

To display this notification, the `showNotification` method is called. This method sets the `notificationVisible` variable to `true` and starts a `timer` to reset it to `false` after three seconds, causing the notification to disappear shortly after. The method accepts a message as a parameter, which is stored in the `notificationText` variable. When the `notificationVisible` variable is set to `true`, the UI component that shows the notification is rendered.

15.2.6 Edit multiple Addresses

To make it easier to edit multiple addresses at once, the control|command key on the keyboard is listened to, meaning the system detects and responds to when the key is pressed. When this key is pressed, the `isCtrlPressed` variable is set to `true`. This variable is passed to the `DatabaseViewComponent` and the `AdminMapComponent` to inform them that multiple addresses are to be selected.

To detect when the control|command key is pressed or released, the predefined `RawKeyboardListener` widget is used. This widget listens for keyboard events and updates the state of the `isCtrlPressed` variable accordingly. When the key is pressed, the widget sets `isCtrlPressed` to `true`, indicating that it is being held down. Alternatively, when the key is released, `isCtrlPressed` is set to `false`, signaling that the key is no longer pressed.
To handle repeated key presses effectively, especially when the CTRL key is held down, a condition is implemented to prevent continuously setting `isCtrlPressed` to `true`. This is achieved by checking the `event.repeat` property, ensuring that the variable is only updated when `event.repeat` is false.
The `markerSelected` method also updates the `InputField` by listing all house numbers of the selected addresses. This ensures that the user is presented with a clear overview of all the selected addresses, making it easier to manage multiple selections.
The `InputField` looks like this:

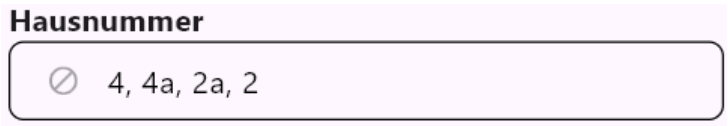

Fig. 66: House numbers of multiple selected Addresses

The selected addresses are saved in the `selectedAddresses` variable in the `AddressPage`. If multiple addresses are selected, certain `InputFields`, such as house numbers, coordinates, or comments, are disabled, as changing them for all selected addresses would not make sense. This is achieved by setting the `editable` parameter of these `InputFields` to `selectedAddresses.length <= 1`, ensuring that they are only editable when a single address is selected.

15.2.7 Edit all Addresses from a Street

All addresses on a street can be edited at once, simplifying bulk modifications. This process is similar to editing multiple addresses (15.2.6), but instead of manually selecting each address, the entire street is selected at once. This allows for quicker adjustments, ensuring consistency across all addresses on the selected street. Once a street is chosen, all associated addresses are automatically included in the selection.

A street can be selected in two ways:

Select a Street via AdminMapComponent

Every click on the `AdminMapComponent` is checked to see if it is near a street. Since there is no predefined method to check if a point is on a street, the `isPointNearPolyline` method was implemented. This method checks whether the click was on the street. More information about this method can be found in the `AdminMapComponent` section 15.4.1.

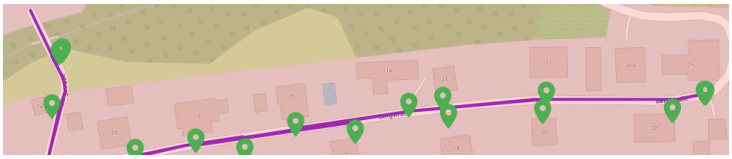

Fig. 67: Street visualization example in AdminMapComponent

Select a Street via Button

This was implemented because selecting the street directly on the map could be a bit spotty in its working. Through this feature, a reliable way to select a street was available, even though in the final deployment, the street selection via map worked flawlessly.

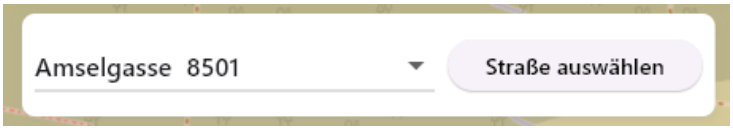

Fig. 68: Field to select a Street

15.2.8 Edit Odd / Even Streets

One requirement was that addresses from a street could be automatically assigned to two areas based on whether the house number is even or odd. This is because it is common for addresses with even house numbers to be on one side of the street and those with odd house numbers on the other side. This division makes it easier to assign the street sides to different areas so that "Sternsinger" participants don't have to cross the street as often. To make this possible, the administrator first selects a street in the `AdminMapComponent` (15.4.1), and then a blue button appears beneath the `InputFields`.

Fig. 69: Button to split Street

After pressing this button, a dialog appears, where the administrator can select the areas for addresses with even and odd house numbers on this street. The "Speichern" button triggers the `AdminAddressProvider` and displays a `notification` (15.2.5) indicating whether the operation was successful or not. Afterward, the dialog is closed.

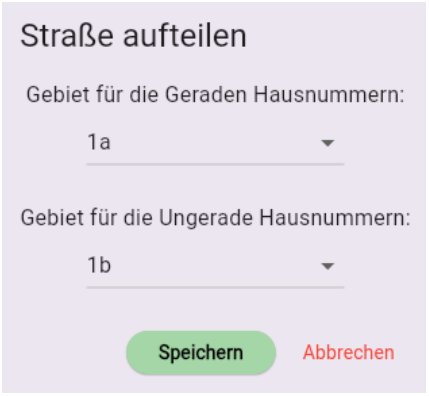

Fig. 70: Dialog to split street

15.2.9 Filter

With the Filter field, the administrator can filter the addresses displayed. It contains three dropdown menus to set the filter criteria, with one checkbox for each to toggle them. These filters can be combined as desired.

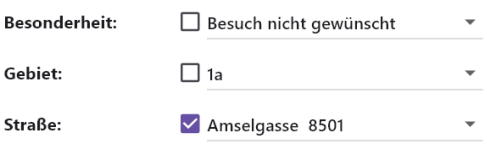

Fig. 71: Filter in AddressPage

The filter is passed and applied to both the `AdminMapComponent` and the `DatabaseViewComponent`. The criteria and their enabled/disabled state are managed by a series of variables within the `AddressPage` class. These variables control which filters are active and store the selected filter values. They are defined as follows:

The filter is then applied to the addresses within the components. The filtering process is done step-by-step, starting with the area filter, followed by the special feature filter, and finally the street filter. If any of the filter conditions are met, the addresses are filtered accordingly. Only the `filteredAddresses` are displayed in the `AdminMapComponent` and the `DatabaseViewComponent`.

15.3 ListEditPage

The `ListEditPage` is used to manage **streets**, **special features**, and **areas**. It allows the administrator to add, edit, and delete these entities. The page is divided into two sections. On the right side, all entities are displayed in a table and can be selected. On the left, there is a dropdown menu for selecting between the three options. The information of the selected item is shown in `InputFields`, where it can be edited, saved, or deleted using the "Speichern" or "Löschen" button.

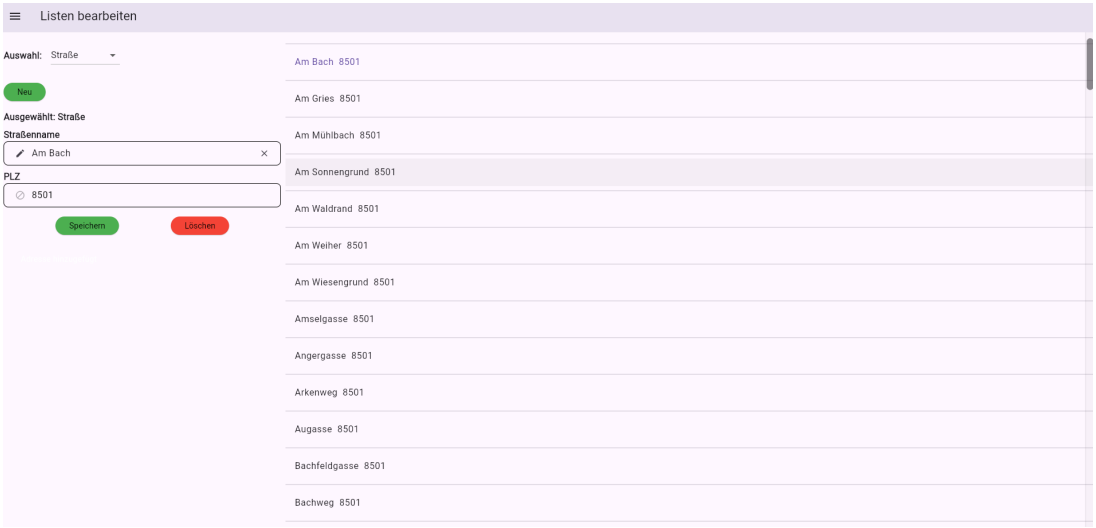

Fig. 72: ListEditPage

15.3.1 QR-Code Visualization for Areas

To make it easier for users of the user application to access information about their area, a QR-Code is generated for the selected area, which can be scanned to get all information. The QR-Code contains the name of the area. The currently selected area is saved in the `selectedItem` variable. To display the QR-Code, the predefined `QrImageView` is used.

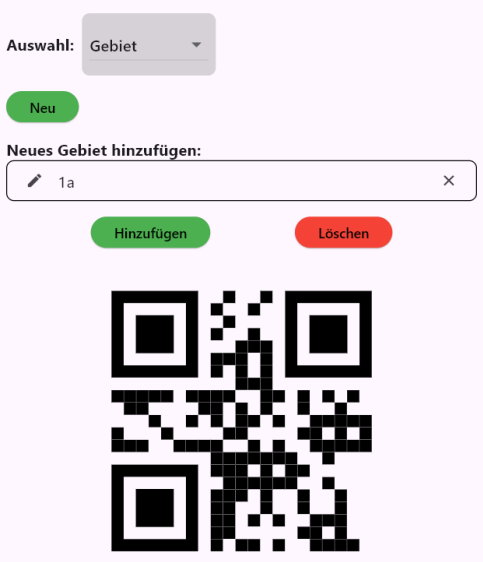

Fig. 73: QR-Code for Area

15.3.2 QR-Code-PDF Download

A PDF containing QR-codes for all areas can be downloaded, making it easier to distribute the information to users. The PDF is generated using the `savePDF` method of the `PDFSaver` class.

To generate the QR-codes, the `generateQRCode` method in the `ListEditPage` class is used. This method takes a `String` as a parameter and converts it into a QR-code using the predefined `QrPainter`, which helps in rendering it [25ac]. It then generates an image of the QR-code, converts it into PNG byte data, and transforms the byte data into a list of bytes (`Uint8List`) to be used in the PDF.

Then, the `savePdf` method of the `PDFSaver` class creates a PDF document containing the QR-codes for all areas. It uses the predefined `pdf` package (imported as `pw`) to manage the PDF document [25aa]. The function starts by initializing a new PDF using the

`pw.Document()` class and an empty list to store the generated QR-codes. The method then iterates over the list of areas, generating a code for each description of all areas using `generateQRCode` and adding it to the list.

Afterward, a page is added to the PDF document using the `pw.MultiPage` widget. This page includes a title and the description of each area along with its corresponding QR-code. Once the page content is built, the PDF data is saved by calling `pdf.save()`. Finally, the `PdfSaver.savePdf` method is called to save the PDF as "Gebiete.pdf."

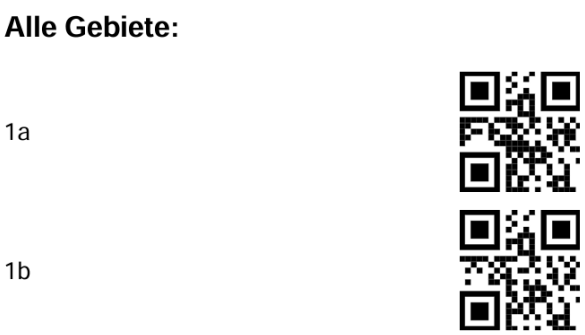

Fig. 74: PDF with QR-Codes for all Areas

15.3.3 Icon for Special Features

When a special feature is selected, an icon is displayed to visually represent it. The administrator has the option to upload a custom image to use as the icon for the selected special feature. Once the image is uploaded and the "Speicher" button is pressed, it is saved in the database and used as the icon, allowing for easy visual identification of the special feature. This icon is displayed in the user application.

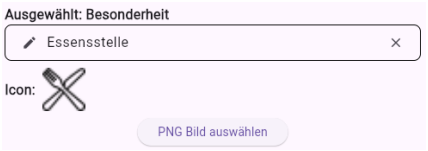

Fig. 75: choose Icon for Special Feature

15.4 Components

Components play a key role in structuring the Admin Panel. Each one is designed to handle specific tasks or display certain UI elements, making them reusable throughout the application.

15.4.1 AdminMapComponent

This component displays the addresses on a geographic map. The map used is OpenStreetMap, which is free to use for everyone. Although the Admin Panel is not used commercially, OpenStreetMap's attribution rules are followed. The map shows the required copyright notice "© OpenStreetMap contributors" in the bottom-left corner. The word "OpenStreetMap" is clickable and links directly to `https://www.openstreetmap.org/copyright`, opening the official license page in a new tab. This meets OpenStreetMap's license requirements by clearly acknowledging the data source and linking to their terms. [25aj]

Fig. 76: OpenStreetMap attribution

Select a Street

To select a street, the `isPointNearPolyline` method was implemented. This method checks whether the click was on a street. It takes the coordinates of the click as a `LatLng` object and a street as a list of `LatLngs`. After each click on the map, a loop is triggered that calls the `isPointNearPolyline` method for every street, passing the click coordinates during each iteration.

The `isPointNearPolyline` method uses another loop that iterates over all points of the street. Since the street is represented as a list of `LatLng` objects, each pair of consecutive points is connected to form a line segment. During each iteration of the loop, a helper method `isNear` is called to calculate the distance between the click and the currently iterated line segment.

The `isNear` method takes the click point, the two points for the line and the tolerance. If the distance is smaller than the predefined tolerance, the method returns `true`, indicating that the click was on the street.

The method uses the Manhattan distance to determine the distance between the click and the line. The Manhattan distance, also known as taxicab or city block distance, measures the distance between two points in a grid-based system. It is calculated as the sum of the absolute differences of the x- and y-coordinates. Why the absolutes? The absolute value ensures that

the distance is always positive, regardless of the direction. If you were to walk 5 blocks north or 5 blocks south, the distance is the same. It's 5 blocks. This distance is then compared to the tolerance. [Alg25]

$$d_{\text{Manhattan}} = |x_2 - x_1| + |y_2 - y_1|$$

```
1 bool _isNear(LatLng p, LatLng a, LatLng b, double tol) {
2     double dx = b.latitude - a.latitude, dy = b.longitude - a.longitude;
3     double t = ((p.latitude - a.latitude) * dx + (p.longitude -
      ↪  a.longitude) * dy) / (dx * dx + dy * dy);
4     t = t.clamp(0, 1);
5     return (p.latitude - (a.latitude + t * dx)).abs() + (p.longitude -
      ↪  (a.longitude + t * dy)).abs() <= tol;
6 }
```

Select Coordinates on Map

To select coordinates to the currently selected address, a long press can be done on the map. The method `onCoordinateSelected` is then called via a callback from the `AdminMapComponent`. This sets the coordinates of the address to the coordinates of the click. The `InputFields` are then filled with the new coordinates.
Because it would not make any sense to select coordinates for multiple addresses at once, the length of the `selectedAddresses` list is checked. Only if it is smaller than 2, the coordinates can be selected.

15.4.2 DatabaseViewComponent

To display the addresses in a table, this component was implemented. It takes the `selectedAddresses`, the filter variables, and the `isCtrlPressed` variable from the `AddressPage` as parameters. The `isCtrlPressed` variable is used to track whether multiple addresses are being selected. The `selectedAddresses` are shown in the table and can be selected by clicking on them.

Above the table, two fields are displayed: one showing the number of selected addresses and the other showing the total number of found addresses. The table is implemented using the `ListView` widget, a predefined widget in Flutter.

2072 Adressen gefunden 3 Adressen ausgewählt

Straße	Hausnummer	Ort	Gebiet	Besonderheit	Bereits besucht	Kommentar
Ahorngasse	4	8501	Default	Wohnhaus	false	
Ahorngasse	5	8501	Default	Wohnhaus	false	
Ahorngasse	6	8501	Default	Wohnhaus	false	
Ahorngasse	8	8501	Default	Wohnhaus	false	
Ahorngasse	10	8501	Default	Wohnhaus	false	
Ahorngasse	12	8501	Default	Wohnhaus	false	
Ahorngasse	14	8501	Default	Wohnhaus	false	

Fig. 77: DatabaseViewComponent

15.4.3 PDFSaver

The `PdfSaver` class provides functionality for saving PDF files from a byte array. The static method `savePdf` handles the saving process depending on the platform.

For web applications, it creates a `Blob` containing the PDF data, generates a temporary URL, and triggers a download via an `AnchorElement`. After the download starts, the temporary URL is revoked to free resources.

On non-web platforms, the method utilizes the `FileSaver` package to store the PDF file. The `saveFile` function is used, specifying the file name, byte data, and MIME type to ensure proper file handling.

15.4.4 AdminAddressProvider

This class serves as a bridge between the Admin Panel and the backend. It is responsible for all CRUD operations on addresses, streets, special features, and areas. It is used in the `AddressPage` and the `ListEditPage`.

The `AdminAddressProvider` includes the `ChangeNotifier` mixin. A mixin is a way to reuse code across multiple classes without using inheritance, as in Java. [25y] The `ChangeNotifier` mixin is used to notify the UI when the data changes. [25f]

This notification is triggered by calling the `notifyListeners` method. Below is a typical method in the `AdminAddressProvider` class, which demonstrates these functionalities:

- `async`: enables non-blocking operations and ensures that the UI remains responsive while waiting for tasks like network requests to complete.
- `await http.get`: sends a GET request to the server to fetch all streets and waits for the response.
- `jsonDecode`: processes the JSON response from the server.
- `utf8.decode`: transforms the UTF-8 encoded response body into readable text.
- `map`: converts the decoded JSON to a list of `Street` objects.
- `sort`: sorts the list alphabetically.
- `notifyListeners`: notifies the listeners that the data has changed.
- `catch`: catches any errors that occur during the operation.

15.5 Models

Models represent and manage data structures within the Admin Panel. They are designed to encapsulate data and provide a structured approach for interacting with it. Various models have been developed to ensure maintainability and efficiency.

15.5.1 AreaWithBorder

To associate an area with its border coordinates, the `AreaWithBorder` model was introduced. It stores all border points as a list of `LatLng` objects, which are then used in the `AdminMapComponent` to draw the boundary. This visualization helps to clearly distinguish which addresses belong to which area.

15.5.2 ScreenItem

The `ScreenItem` was created to store data related to a menu option in the `AdminNavigation`. It is used to save the title, associated screen, and icon for each navigation item.

15.6 Widgets

Widgets are used to build and reuse UI components in Flutter. They help maintain a consistent look and feel across the entire interface, ensuring a seamless user experience. Several custom widgets were implemented for the Admin Panel.

15.6.1 InputField

This widget defines a customizable input field. The `InputField` requires a label and a `TextEditingController` to manage the input. Optional parameters include a boolean `editable`, a list of `Strings` called `dropDownOptions`, and a boolean `isCoordinateInput` to specify whether the input should be numeric.

To indicate whether the `InputField` is editable, icons are displayed on the left side of the field. If the field is not editable, a blocked symbol appears. If it is editable, a pencil icon is shown. When the `InputField` is editable and not a dropdown, a cross icon appears on the right side. Pressing this icon clears the content.

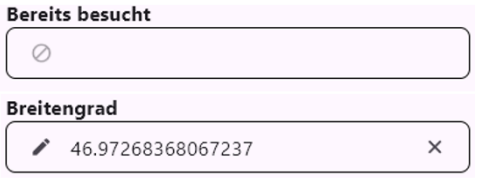

Fig. 78: InputField examples

If the `dropDownOptions` parameter is provided with a non-empty list of `Strings`, the input field is rendered as a dropdown menu. The selected value is displayed, and users can choose from the available options. The selected dropdown value is automatically synchronized with the `TextEditingController`.

To ensure valid inputs, the `InputField` uses an `inputFormatter` for validation. This formatter is applied only when the `isNumberInput` parameter is set to true. For more details about the validation, refer to 15.2.4.

15.6.2 FilterRow

The `FilterRow` is a custom widget which combines a label, a toggleable checkbox, and a dropdown menu to enable dynamic data filtering. It requires several parameters, including callbacks. A callback is a method passed as an argument to another function and executed after a specific event. These callbacks are used to handle activating and deactivating the filter, as well as changes to the filter's state and the selected dropdown value. The following parameters are used:

- `label`: Identification string for the filter.
- `tooltipMessage`: Message displayed when hovering over the checkbox.
- `filterValue`: Boolean value tracking the filter's active state.
- `items`: List of strings for the dropdown options.
- `selectedValue`: Manages the selected dropdown value.
- `onFilterChanged`: Callback for updating the filter state.
- `onDropdownChanged`: Callback for updating the selected dropdown value.

Fig. 79: FilterRow

16 Final Thoughts

16.1 Leon Edlinger

I am happy and proud about the results of this project. Because of this thesis, I have learned a lot of new interesting things and gained a lot of experience. I am very grateful to my colleagues and my supervisor for their support and help. Hopefully our software will be useful in the future for the Sternsinger-Aktion and help to make this event modern and more efficient.

16.2 Paul Gigler

I think that through the course of this thesis I got a pretty good impression of what is like to cooperatively develop a real software-product. Despite some organizational troubles, I think we did a great job of developing a software that meets all the important criteria.

16.3 Andreas Weissl

This project offered a chance to deepen my knowledge of Spring Boot while getting actual hands-on experience. I enjoyed overcoming challenges, which encouraged me to create some creative solutions. The research on the convex hull algorithm was particularly interesting, as I have always enjoyed working with theoretical mathematical ideas. I am grateful to my supervisor and colleagues for their support whenever I needed help. I got a lot of valuable knowledge throughout this project and hope that our application will contribute to making the caroler campaign more modern and efficient.

17 Working Hours

17.1 Paul Gigler

Beschreibung	Dauer
Research Navigation Bar	1 h
Implementation of Navigation Bar	3 h
Research of Map Component	3 h
Implementation of Map Component	12 h
Implementation of List Component	10 h
implementation of HTTP Requests and AddressProvider	11 h
Researching for street marking	5 h
Implementation of street marking	8 h
Implementation of QR-Scanner	5 h
Server set up	9 h
Initialize Database	7 h
Researching Usability	10 h
Meetings	6 h
General tasks (Deployment, Git administration...)	13 h
Writing of the diploma thesis	72 h
Summe	**175 h**

Table 1: Arbeitszeitnachweis Paul Gigler

17.2 Leon Edlinger

Beschreibung	Dauer
Design and Project setup	8 h
implementation Admin Panel	80 h
Bug fixes and Optimazation Admin Panel	35 h
Writing	40 h
Summe	**187 h**

Table 2: Arbeitszeitnachweis Leon Edlinger

17.3 Andreas Weissl

Beschreibung	Dauer
Researching different backend architectures	5 h
Researching algorithms for border addresses	10 h
Researching algorithms for convex hull algorithm	10 h
Defining necessary API endpoints	6 h
Defining necessary entities and relationships	10 h
Implementing logic for processing data	16 h
Convex hull algorithm implementation	21 h
Optimizing and debugging border address detection	11 h
Implementation of all API routes	15 h
Testing the API endpoints	7 h
Debbuging and fixing error	10 h
Documenting the API endpoints with Swagger	5 h
Writing of the diploma thesis	56 h
Summe	**182 h**

Table 3: Arbeitszeitnachweis Andreas Weissl

18 List of listings

19 List of figures

20 List of tables

21 Bibliography

[13] *A Closer Look at Alpha Shapes in pgRouting*. [Online; accessed 9. Mar. 2025]. May 2013. URL: https://anitagraser.com/2011/09/25/a-closer-look-at-alpha-shapes-in-pgrouting.

[17] *Java Logo – PNG e Vetor – Download de Logo*. [Online; accessed 12. Mar. 2025]. Apr. 2017. URL: https://logodownload.org/java-logo.

[23a] *Logo - PostgreSQL wiki*. [Online; accessed 12. Mar. 2025]. Nov. 2023. URL: https://wiki.postgresql.org/wiki/Logo.

[23b] *Logo, Icon, and Brand Guidelines | Docker*. [Online; accessed 12. Mar. 2025]. Oct. 2023. URL: https://www.docker.com/company/newsroom/media-resources.

[24a] *Defining JPA Entities | Baeldung*. [Online; accessed 6. Mar. 2025]. May 2024. URL: https://www.baeldung.com/jpa-entities.

[24b] *Figure(5): Minimum Bounding Box*. [Online; accessed 9. Mar. 2025]. Oct. 2024. URL: https://www.researchgate.net/figure/Figure5-Minimum-Bounding-Box_fig1_354380006?__cf_chl_rt_tk=2cm8jQw.EyF0OgDYrFNdEBrnvCwT54nUX_GTrS7TdVI-1741475168-1.0.1.1-48mwHWEjdeyHeTJwbeGswcjO5vSDIo8qMxaJz.QPMAc.

[24c] *Host Your Own Worldwide Route Calculator With GraphHopper - GraphHopper Directions API*. [Online; accessed 30. Jan. 2025]. Apr. 2024. URL: https://www.graphhopper.com/blog/2022/06/27/host-your-own-worldwide-route-calculator-with-graphhopper.

[25a] *(PNG Image, 100 × 100 pixels)*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://img.icons8.com/?size=100&id=04OFrkjznvcd&format=png&color=000000.

[25b] *About GitHub and Git - GitHub Docs*. [Online; accessed 16. Feb. 2025]. Feb. 2025. URL: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git.

[25c] *Brand*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://flutter.dev/brand.

[25d] *Brandfetch*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://brandfetch.com/graphhopper.com.

[25e] *Build software better, together*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://github.com/logos.

[25f] *ChangeNotifier class - foundation library - Dart API*. [Online; accessed 25. Feb. 2025]. Feb. 2025. URL: https://api.flutter.dev/flutter/foundation/ChangeNotifier-class.html.

[25g] *Color Theory for Beginners: Essential Design Tips*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://colorpage.ai/blog/color-theory-for-beginners.

[25h] *Color Theory for Beginners: Essential Design Tips*. [Online; accessed 4. Mar. 2025]. Mar. 2025. URL: https://colorpage.ai/blog/color-theory-for-beginners.

[25i] *Common Challenges In Batch Processing And How To Overcome Them - FasterCapital*. [Online; accessed 10. Mar. 2025]. Mar. 2025. URL: https://fastercapital.com/topics/common-challenges-in-batch-processing-and-how-to-overcome-them.html.

[25j] *Convex Hull*. [Online; accessed 8. Mar. 2025]. Mar. 2025. URL: https://usaco.guide/plat/convex-hull?lang=cpp.

[25k] *Convex Hull | Brilliant Math & Science Wiki*. [Online; accessed 8. Mar. 2025]. Mar. 2025. URL: https://brilliant.org/wiki/convex-hull.

[25l] *Convex Polygon - Definition, Formulas, Properties, Examples*. [Online; accessed 8. Mar. 2025]. Mar. 2025. URL: https://www.cuemath.com/geometry/convex.

[25m] *Dart brand guidelines*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://dart.dev/brand.

[25n] *flutter/README.md at master · flutter/flutter*. [Online; accessed 23. Jan. 2025]. Jan. 2025. URL: https://github.com/flutter/flutter/blob/master/README.md.

[25o] *Git*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://git-scm.com.

[25p] *Git - Branching and Merging*. [Online; accessed 16. Feb. 2025]. Feb. 2025. URL: https://git-scm.com/about/branching-and-merging.

[25q] *graphhopper*. [Online; accessed 30. Jan. 2025]. Jan. 2025. URL: https://github.com/graphhopper/graphhopper.

[25r] *Grok*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://grok.com/chat/79f8bcd9-99c9-4a3d-8464-4865f27672d5%7D.

[25s] *Help | About Rule-Based Filters | Autodesk*. [Online; accessed 11. Mar. 2025]. Jan. 2025. URL: https://help.autodesk.com/view/RVT/2025/ENU/?guid=GUID-400FD74B-00E0-4573-B3AC-3965E65CBBDB.

[25t] *IntelliJ IDEA – the Leading Java and Kotlin IDE*. [Online; accessed 30. Jan. 2025]. Jan. 2025. URL: https://www.jetbrains.com/idea.

[25u] *Introduction to Project Lombok | Baeldung*. [Online; accessed 6. Mar. 2025]. Jan. 2025. URL: https://www.baeldung.com/intro-to-project-lombok.

[25v] *JetBrains Brand Assets - JetBrains*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://www.jetbrains.com/company/brand.

[25w] *Logo approval process | Postman*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://www.postman.com/legal/logo-usage.

[25x] *Logos - OpenStreetMap Wiki*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://wiki.openstreetmap.org/wiki/Logos.

[25y] *Mixins*. [Online; accessed 25. Feb. 2025]. Feb. 2025. URL: https://dart.dev/language/mixins.

[25z] *OSM: The simple map that became a global movement*. [Online; accessed 28. Jan. 2025]. Jan. 2025. URL: https://web.archive.org/web/20240420144212/https://www.directionsmag.com/article/1163.

[25aa] *pdf | Dart package*. [Online; accessed 2. Mar. 2025]. Mar. 2025. URL: https://pub.dev/packages/pdf.

[25ab] *Persisting Entities :: Spring Data JPA*. [Online; accessed 6. Mar. 2025]. Mar. 2025. URL: https://docs.spring.io/spring-data/jpa/reference/jpa/entity-persistence.html.

[25ac] *QrPainter class - qr_flutter library - Dart API*. [Online; accessed 2. Mar. 2025]. Mar. 2025. URL: https://pub.dev/documentation/qr_flutter/latest/qr_flutter/QrPainter-class.html.

[25ad] *Quickstart for GitHub Actions - GitHub Docs*. [Online; accessed 16. Feb. 2025]. Feb. 2025. URL: https://docs.github.com/en/actions/writing-workflows/quickstart.

[25ae] *Simple Color Wheel Chart in Illustrator, PDF - Download | Template.net*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://www.template.net/editable/114132/simple-color-wheel-chart.

[25af] *Spring Boot*. [Online; accessed 6. Mar. 2025]. Mar. 2025. URL: https://spring.io/projects/spring-boot.

[25ag] *Stable*. [Online; accessed 6. Mar. 2025]. Mar. 2025. URL: https://projectlombok.org/features.

[25ah] *Stack Overflow Developer Survey 2022*. [Online; accessed 16. Feb. 2025]. Feb. 2025. URL: https://survey.stackoverflow.co/2022/#version-control-version-control-system.

[25ai] *The impact of human error rates in manual data entry across multiple systems*. [Online; accessed 10. Mar. 2025]. Mar. 2025. URL: https://integrationmadeeasy.com/resources/impact-of-human-error-rates.

[25aj] *Urheberrecht und Lizenz*. [Online; accessed 27. Feb. 2025]. Feb. 2025. URL: https://www.openstreetmap.org/copyright.

[25ak] *Visual Studio Code and VS Code icons and names usage guidelines*. [Online; accessed 12. Mar. 2025]. Mar. 2025. URL: https://code.visualstudio.com/brand.

[25al] *Visual Studio Code for the Web*. [Online; accessed 30. Jan. 2025]. Jan. 2025. URL: https://code.visualstudio.com/docs/editor/vscode-web.

[25am] *vscode*. [Online; accessed 30. Jan. 2025]. Jan. 2025. URL: https://github.com/microsoft/vscode.

[25an] *What is the Visual Studio Code editor built on*. [Online; accessed 30. Jan. 2025]. Jan. 2025. URL: https://stackoverflow.com/questions/29966093/what-is-the-visual-studio-code-editor-built-on.

[Alg25] AlgoDaily. *AlgoDaily - What Is The Manhattan Distance?* [Online; accessed 4. Mar. 2025]. Mar. 2025. URL: https://algodaily.com/lessons/what-is-the-manhattan-distance.

[All22] Allyant. "Is Google Maps Accessible?" In: *Allyant* (2022). URL: https://allyant.com/is-google-maps-accessible/.

[Aut02] Autoren der Wikimedia-Projekte. *Regulärer Ausdruck – Wikipedia*. [Online; accessed 10. Mar. 2025]. July 2002. URL: https://de.wikipedia.org/w/index.php?title=Regul%C3%A4rer_Ausdruck&oldid=251213016.

[BG08] Sarah E Battersby and Kirk Goldsberry. "User-centered design for digital map navigation tools". In: *Cartography and Geographic Information Science* 35.4 (2008), pp. 233–244.

[Bie19] Alessandro Biessek. *Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2*. Packt Publishing Ltd, 2019.

[Cim23] Mustafa Ciminli. "Simplifying Database Access with Spring Data JPA: A Comprehensive Overview of its Powerful Features". In: *Medium* (May 2023). URL: https://medium.com/@mustafa_ciminli/simplifying-database-access-with-spring-data-jpa-a-comprehensive-overview-of-its-powerful-features-620c0de4cb91.

[Cod25] CodingNomads. *Spring Boot Tutorial: Build an App with Spring Initializr*. [Online; accessed 6. Mar. 2025]. Mar. 2025. URL: https://codingnomads.com/spring-boot-tutorial-build-an-app-with-spring-initializr.

[Con24a] Contributors to Wikimedia projects. *Adaptive algorithm - Wikipedia*. [Online; accessed 10. Mar. 2025]. Aug. 2024. URL: https://en.wikipedia.org/w/index.php?title=Adaptive_algorithm&oldid=1242668518.

[Con24b] Contributors to Wikimedia projects. *Minimum bounding box - Wikipedia*. [Online; accessed 8. Mar. 2025]. Oct. 2024. URL: https://en.wikipedia.org/w/index.php?title=Minimum_bounding_box&oldid=1249919681.

[Con25a] Contributors to Wikimedia projects. *Alpha shape - Wikipedia*. [Online; accessed 8. Mar. 2025]. Mar. 2025. URL: https://en.wikipedia.org/w/index.php?title=Alpha_shape&oldid=1278461386.

[Con25b] Contributors to Wikimedia projects. *Convex hull - Wikipedia*. [Online; accessed 8. Mar. 2025]. Mar. 2025. URL: https://en.wikipedia.org/w/index.php?title=Convex_hull&oldid=1278659713.

[Con25c] Contributors to Wikimedia projects. *Data validation - Wikipedia*. [Online; accessed 24. Feb. 2025]. Feb. 2025. URL: https://en.wikipedia.org/w/index.php?title=Data_validation&oldid=1276518123.

[Con25d] Contributors to Wikimedia projects. *Delaunay triangulation - Wikipedia*. [Online; accessed 8. Mar. 2025]. Mar. 2025. URL: https://en.wikipedia.org/w/index.php?title=Delaunay_triangulation&oldid=1278482810.

[Con25e] Contributors to Wikimedia projects. *IntelliJ IDEA - Wikipedia*. [Online; accessed 30. Jan. 2025]. Jan. 2025. URL: https://en.wikipedia.org/w/index.php?title=IntelliJ_IDEA&oldid=1272820016.

[Dag19] Lukas Dagne. "Flutter for cross-platform App and SDK development". In: (2019).

[De 24a] Kalana De Silva. "Building the Service Layer in Spring Boot - Kalana De Silva - Medium". In: *Medium* (Nov. 2024). URL: https://medium.com/@kalanamalshan98/building-the-service-layer-in-spring-boot-dbbc900b0853.

[De 24b] Kalana De Silva. "Implementing the Controller Layer in Spring Boot - Kalana De Silva - Medium". In: *Medium* (Nov. 2024). URL: https://medium.com/@kalanamalshan98/implementing-the-controller-layer-in-spring-boot-8c4e5928d888.

[De 24c] Kalana De Silva. "Repository Layer and Custom Queries in Spring Boot - Kalana De Silva - Medium". In: *Medium* (Nov. 2024). ISSN: 1263-5458. URL: https://medium.com/@kalanamalshan98/repository-layer-and-custom-queries-in-spring-boot-1b26d3545c8c.

[Enc23] Encora. "Machine Learning: What is Adaptive ML?" In: *Encora* (Nov. 2023). URL: https://insights.encora.com/insights/machine-learning-what-is-adaptive-ml.

[Fro+19] Jon E Froehlich et al. "Grand challenges in accessible maps". In: *Interactions* 26.2 (2019), pp. 78–81.

[Gee24a] GeeksforGeeks. "Convex Hull Algorithm". In: *GeeksforGeeks* (Aug. 2024). URL: https://www.geeksforgeeks.org/convex-hull-algorithm.

[Gee24b] GeeksforGeeks. "Convex Hull Algorithm". In: *GeeksforGeeks* (Aug. 2024). URL: https://www.geeksforgeeks.org/convex-hull-algorithm.

[Gee24c] GeeksforGeeks. "Latency vs. Accuracy in System Design". In: *GeeksforGeeks* (Oct. 2024). URL: https://www.geeksforgeeks.org/latency-vs-accuracy-in-system-design.

[Ham23] Nasrullah Hamidli. "Introduction to UI/UX design: key concepts and principles". In: *Academia. URL: https://www.academia.edu/98036432/Introduction_to_UI_UX_-Design_Key_Concepts_and_Principles [accessed 2024-04-27]* (2023).

[hel24] hello. "Real-time Data Integration Challenges: Explained - Vexdata". In: *Vexdata* (June 2024). URL: https://vexdata.io/real-time-data-integration-challenges-explained.

[Ibm24] Ibm. *Java Spring Boot*. [Online; accessed 6. Mar. 2025]. Dec. 2024. URL: https://www.ibm.com/think/topics/java-spring-boot.

[Lal23] Vishamber Lal. "Understanding Data Transfer Objects (DTO) in Spring Boot". In: *Medium* (Dec. 2023). URL: https://medium.com/@vishamberlal/understanding-data-transfer-objects-dto-in-spring-boot-ac06b575a1d5.

[Lim25] Infosys Limited. *The key components of Data integration – glossary | Infosys BPM*. [Online; accessed 10. Mar. 2025]. Mar. 2025. URL: https://www.infosysbpm.com/glossary/data-integration.html.

[Luc25] G. W. Lucas. *Delaunay Triangulation*. [Online; accessed 9. Mar. 2025]. Feb. 2025. URL: https://gwlucastrig.github.io/TinfourDocs/DelaunayIntro/index.html.

[McB25] Martin McBride. *GraphicMaths - Other types of polygon*. [Online; accessed 8. Mar. 2025]. Feb. 2025. URL: https://graphicmaths.com/gcse/geometry/other-polygons.

[Mee24] Meet2sudhakar. "Why we need a service layer in Spring boot rest API application". In: *Medium* (Nov. 2024). URL: https://medium.com/@meet2sudhakar/why-we-need-a-service-layer-in-spring-boot-rest-api-application-db20e4b2a027.

[Men24] Mendes. "Dependency Injection In Java made easy (With Lombok)". In: *Medium* (Jan. 2024). URL: https://medium.com/%40mendesskrillacc/dependency-injection-in-java-made-easy-with-lombok-0dfc0fc34248.

[Nie24a] Jakob Nielsen. "Usability 101: Introduction to Usability". In: *Nielsen Norman Group* (Jan. 2024). URL: https://www.nngroup.com/articles/usability-101-introduction-to-usability.

[Nie24b] Rick Nieters. "Was ist Fuzzy Matching? Eine nicht so fuzzy Erklärung". In: *Klippa* (Dec. 2024). URL: https://www.klippa.com/de/blog/informativ/fuzzy-matching-de.

[Pat24] Uday Patil. "Spring Boot Architecture - Uday Patil - Medium". In: *Medium* (Nov. 2024). URL: https://medium.com/@udaypatil318/spring-boot-architecture-39935654ce5c.

[Ran24] Shahab Ranjbary. "Choosing the Right Real-Time Stream Processing Framework". In: *DEV Community* (Sept. 2024). URL: https://dev.to/ranjbaryshahab/comparing-top-real-time-stream-processing-frameworks-ek8.

[Tri23] Bubu Tripathy. "Best Practices: Entity Class Design with JPA and Spring Boot". In: *Medium* (Aug. 2023). ISSN: 6703-3393. URL: https://medium.com/@bubu.tripathy/best-practices-entity-class-design-with-jpa-and-spring-boot-6f703339ab3d.

[Wik24] OpenStreetMap Wiki. *Main Page — OpenStreetMap Wiki*. [Online; accessed 28-January-2025]. 2024. URL: https://wiki.openstreetmap.org/w/index.php?title=Main_Page&oldid=2752444.

22 Abbreviation

ADB	Android Debugging Bridge
API	Application Programming Interface
CI|CD	Continuous Integration & Continuous Deployment
CMY	Cyan Magenta Yellow
CRUD	Create Read Update Delete
CSV	Comma Separated Values
DTO	Data Transfer Object
GH	GraphHopper
GIS	Geo Information System
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
JPA	Java Persistence API
JVM	Java Virtual Machine
ORM	Object Relational Mapping
OSM	Open Street Map
PDF	Portable Document Format
RGB	Red Green Blue
REST	REpresentational State Transfer
SQL	Structured Query Language
UI	User Interface
VCS	Version Control System
XML	eXtensible Markup Language