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:	MISSING DATE

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

Abstract in English

Kurzfassung

Kurzfassung in Deutsch

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.
…

Table of Contents

1 Introduction

TODO: Is halt die frage ob ma den anfang einfach so schreiben, war ja eigentlich net ganz so xD

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 which gets drawn through "Border" addresses. These border addresses get calculated by an algorithm implemented by us. It should be visible at a glance if there is an "specification", which can be assigned by admins, set for an address. This should be realized through the use of different icons instead of the default icon. Apart from the app itself, we also implemented a web-portal through which administrators can manage and supervise the campaign.

TODO: vielleicht noch was rein bezüglich der borders und dann unten nurmehr drauf referenzieren?

The research part of this thesis will be dedicated to how components should act and look, so that new users can use this tool without requiring a long "onboarding" phase. It should feel familiar to interact with elements and the borders of what users can and can not do need to be clearly defined. Because our application also needs a reliable data source to guarantee the consistency and accuracy of marked addresses, we researched ways to keep our database up-to-date, without the need of much manual intervention. After defining the project requirements, we noticed that we need to calculate which addresses are border addresses. So we decided to take a look into different algorithms for this task and compare them concerning their efficiency, decide on one of them and implement it.

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

1.1 Team

This thesis was created by three Students attending the BHIF20 at the HTBLA Kaindorf Computer Science Department.

TODO: andis bild anpassen

Leon Edlinger

Database, Admin-Panel

Paul Gigler

Deployment, Mobile App

Andreas Weissl

Backend

2 Technologies

Development would not have been possible without making use of many tools, frameworks and environments. In this chapter each tool used in the creation of our software will be described briefly.

2.1 LaTeX

Hier kommt eine Beschreibung zu Latex hin

2.2 Frontend

2.2.1 Dart

Dart is a programming language initially designed for web development, with the goal, of replacing JavaScript, in mind. Today it gets used in a variety of software products, mainly because of the flutter framework. It can be compiled for many platforms and architectures (ARM, x64, RISC-V, JavaScript or WebAssembly) and is loved for its combination of High-Level Features, with practical language features like Garbage collection and optional Type annotation. It was developed by Google and is now an open-source project.

[]

2.2.2 Flutter

Flutter is an Open-Source software development framework. It allows programmers to compile their application for different platforms including Web, macOS, IOS as well as Windows and any type of Linux-based systems, all from one code-base, 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 however needed which makes them an applicable choice.

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, learn what we need along the way. Having a design through the components was also very helpful and saved us some time.

[25d] [Dag19]

Flutter

2.3 Backend

2.3.1 Java Spring

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

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet,

2.3.2 PostgreSQL

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

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet,

2.4 Version Control

2.4.1 Git

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

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet,

2.4.2 GitHub

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

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet,

2.5 Map Data

2.5.1 OpenStreetMap

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

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet,

2.5.2 Graphhopper

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

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet,

2.6 Development Tools

2.6.1 VS Code

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

2.6.2 IntelliJ

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

2.6.3 Android Studio

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

2.6.4 Postman

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

2.6.5 Figma

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

2.7 Deployment

2.7.1 Docker

2.7.2 Uberspace

2.7.3 Webmin

3 Research Questions

3.1 Leon Edlinger

3.2 Paul Gigler

3.3 Andreas Weissl

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 the efficiency.

4.1 Spring Boot

Spring Boot is a tool which 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 an Java application.

Opinionated Approach:

- When adding and configuring startup dependencies, Spring Boot takes a subjective approach, customizing them to the requirements of the project. The right packages and default values are chosen automatically by Spring Boot, eliminating the need for human setup and decision-making for each configuration.

- 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 example, the "Spring Web" dependency simplifies the development of Spring-based web applications by including all necessary dependencies with 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:

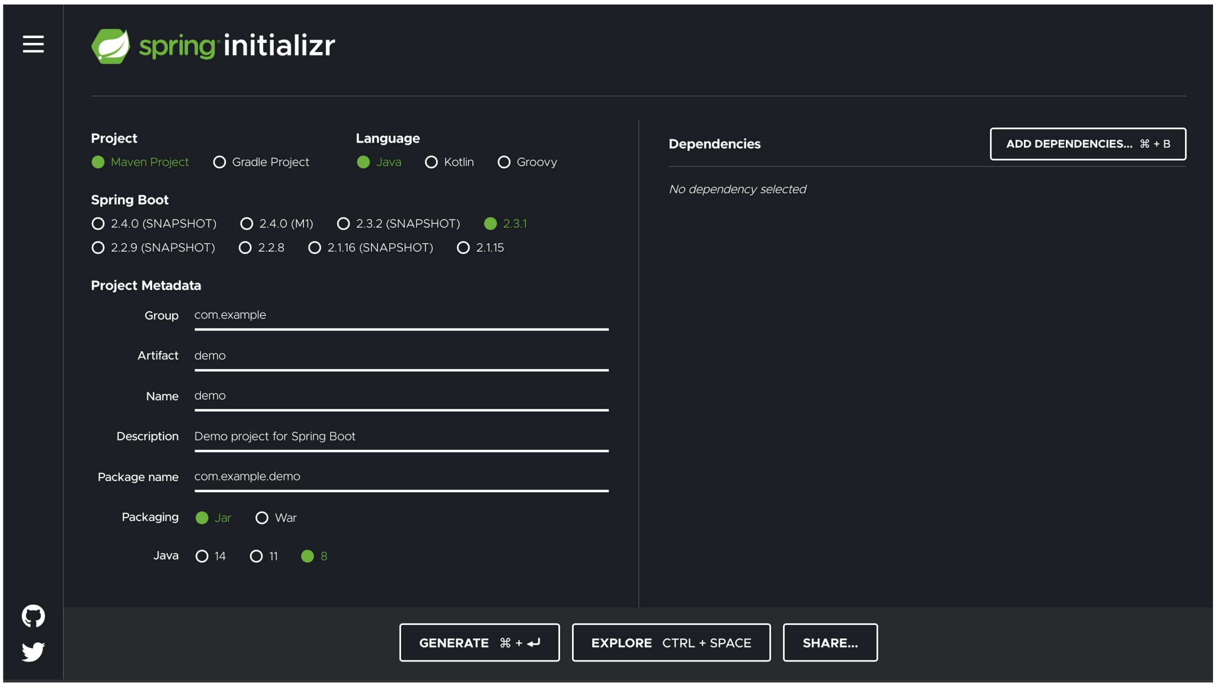
- With the help of Spring Boot, applications can be developed that function without the need for an external web server. This is accomplished by immediately integrating a web server, such as Tomcat, with the application as it is initializing. As a result, the application can be launched with the appropriate command on any platform.

[25g; 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, a number of 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 IDE where the development is being performed, a Spring Boot project can be directly developed inside 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]



spring initializr

Project
● Maven Project ○ Gradle Project

Language
● Java ○ Kotlin ○ Groovy

Spring Boot
○ 2.4.0 (SNAPSHOT) ○ 2.4.0 (M1) ○ 2.3.2 (SNAPSHOT) ● 2.3.1
○ 2.2.9 (SNAPSHOT) ○ 2.2.8 ○ 2.1.16 (SNAPSHOT) ○ 2.1.15

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	● Jar ○ War
Java	○ 14 ○ 11 ● 8

Dependencies

No dependency selected

Abb. 1: GUI of the Spring Initializr

[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 is at the expense of the higher degree of flexibility that direct Spring Framework usage would provide. Practically, the compromis 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. As a result, JPA is just some guidelines to implement 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 to access and manipulate data from the database of a project.

Entities:

As already mentioned, the object from 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, this annotation is not the only one which is supported by JPA. 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. [25f; 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 know as a Data Access Object. However, something else need to be done when using the `@Repository` annotation. 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 use the `@Service` annotation are used with classes that provide some data manipulation methods such as a repository-class. By implementing such a class, the project structure is better understandable and more clear to other developers which 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 which 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 be 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.
These routes can then be accessed through a GUI, allowing the user to manage, update, or delete data as needed. [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 itself 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 make it easier to read maintain and write. Lombok works great with the whole Spring framework. If you would want 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 build, which reduces the need to manually write common functions like constructors, getter 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. [25h; 25e; 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 abilty 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 which 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 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 configurations models across many different types of applications.

Whether you would want to develop a web application or a microservice, Spring offers a unified approach which 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.

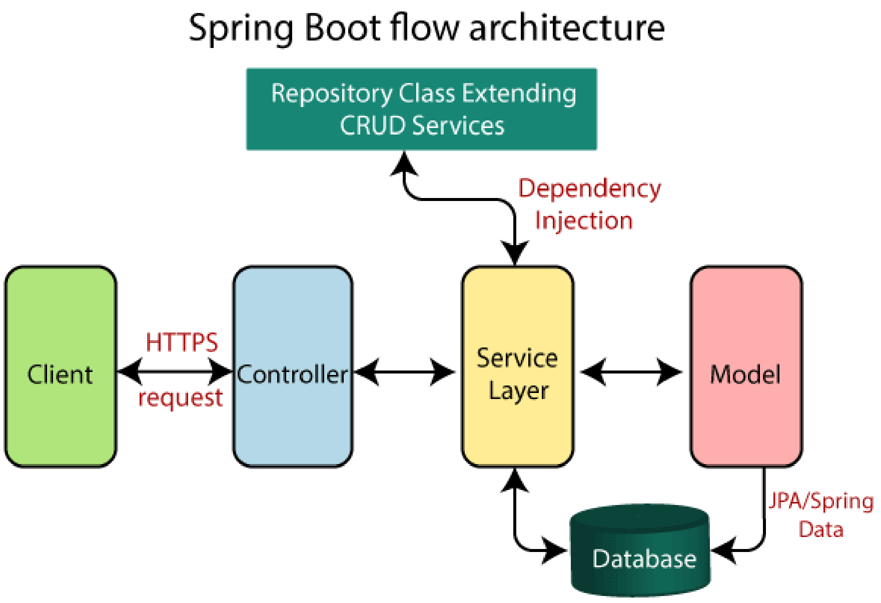


Abb. 2: Layers of Spring Boot

[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 cuts down a lot of unnecessary code required 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 in an efficient way 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 which 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 optimize 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, 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 optimizing 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 by 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 its 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 to the database.

Using Automation Tools:

- Manually constructing an 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 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 leads 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 determin 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 which 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 on the list of addresses to find the border addresses. This 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. The 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 colour 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 rapdily evaluate the boundaries of each area. Based on the result of each area's boundaries, the admin can remove or add addresses if necesssary. 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 factor, 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 in 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 sets 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 would be 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.

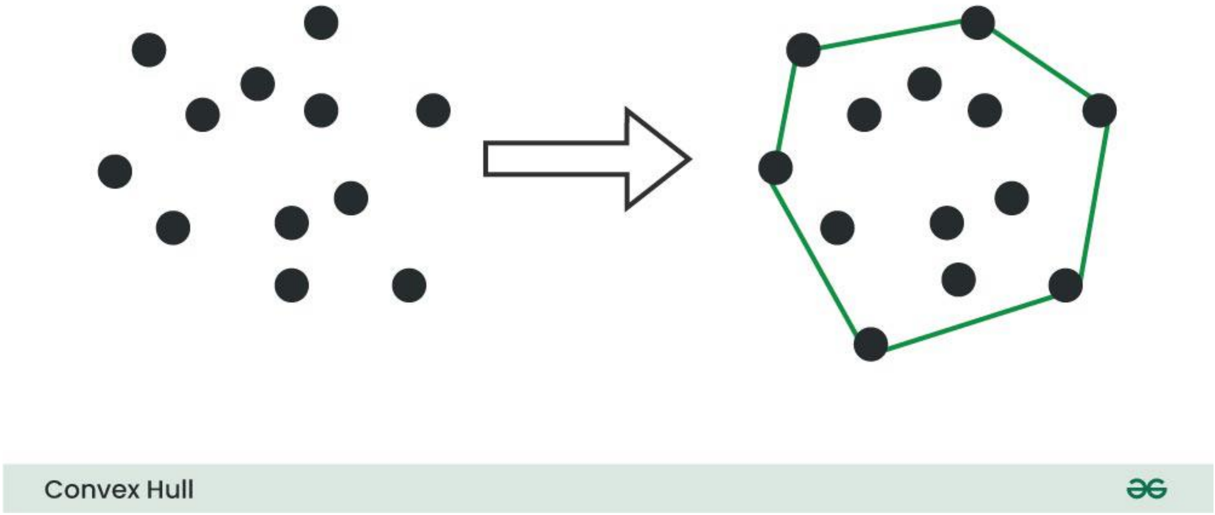


Abb. 3: Process of the Convex Hull Algorithm

[Gee24]

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 atleast 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 [25c]

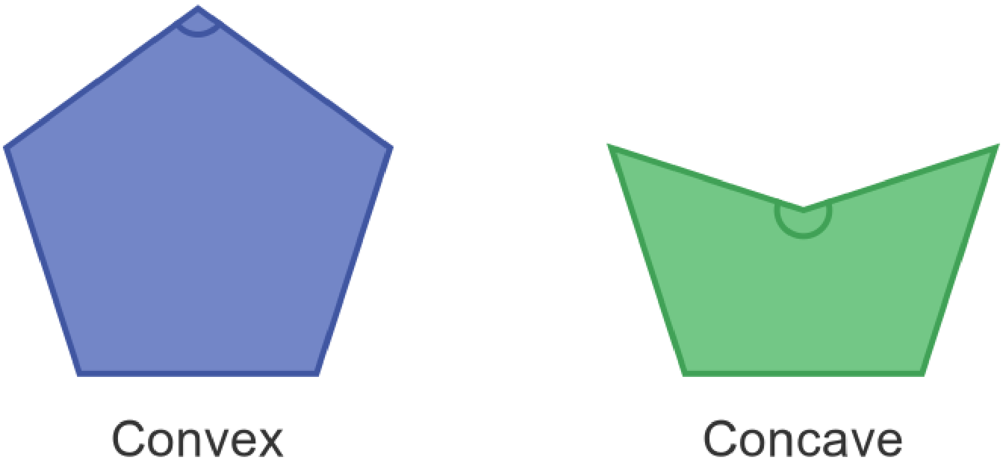


Abb. 4: Difference between a convex/concave polygon

[McB25]

6.2.1 Why Use the Convex Hull?

The convex hull algorithm si especially helpful for identifying a set of points' boundary and eliminating any extra points that fall inside it. In the cas 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 methods 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, 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 curves 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.

[25a; Gee24]

Monotone Chain Algorithm:

- The Monotone Chain algorithm was introduced in 1979 and is antoher efficient way to calculate a convex hull. It begins by soirting 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 hul. The algorithm goes through the sorted list of points, adding them to the hull and ensures that each new point does not cause a 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.

[25a; Gee24]

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 wether 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 minimises 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.

[25b]

6.3 Use Cases of the Convex Hull in Industry

The ability if 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, visualising data and resolving optimisation 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 defining and object’s boundaries which improves detection. It makes it easier for algorithms to process and categorise 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 cost.

Animal Behavior

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

Astronomy and Astrophysics

- Astronomers utilise 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 is normally aligned with the coordiante 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, because 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 which visualizes the difference between the MBB and the convex hull. The convex hull is the gray inner are and the outer line is the MBB. This shows that the convex hull is more precise. [Con24]

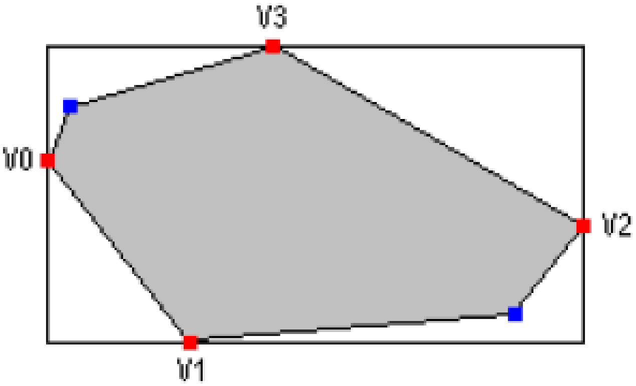


Abb. 5: Difference between MBB and Convex Hull

[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 caputre 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 make a better job than the convex hull. [Con25a]

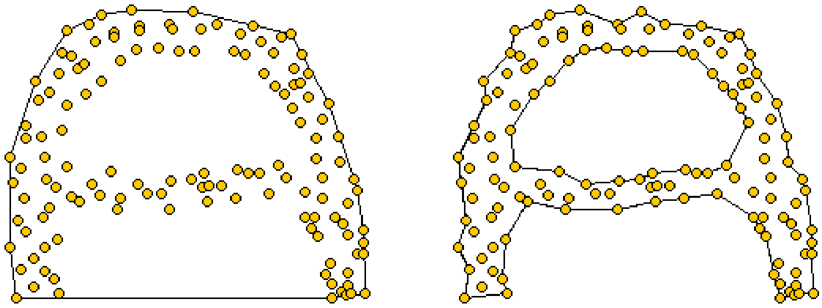


Abb. 6: Difference between Alpha Shape and Convex Hull

[13]

Delaunay Triangulation

- Delaunay Triangulation is a method that makes a mesh of triangles which connect a set of points while maximizing the minimum angle of each traingle. 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.

[Con25c]

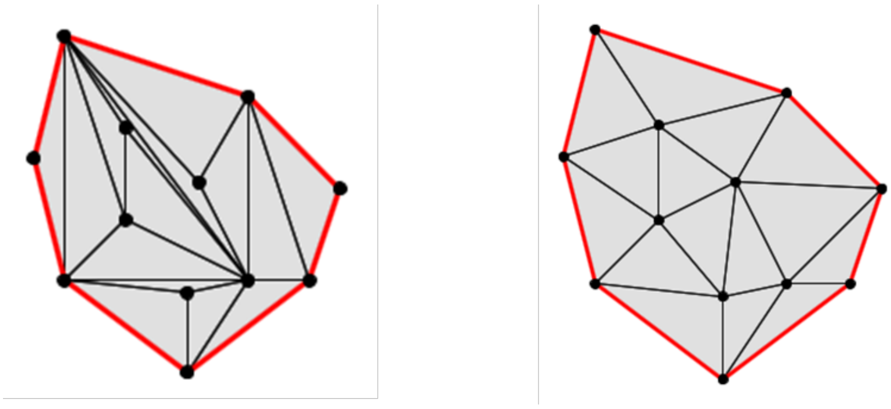


Abb. 7: Mesh of Triangles from Delaunay Triangulation

[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 complecity 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 where 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 whil 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 analyses 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 are then returned as a response to the frontend, where they are getting used for visualization and management.
The server-side calculation laod 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

7.1 Why it is important

7.2 Fundamental concepts of usability

7.3 Challenges in designing for a broad user spectrum

8 Usability in context of maps

8.1 Basic Analysis of the Google Maps interface

8.2 Identifying Flaws in Googles Design

8.3 How could specific user groups struggle with this design

9 Adaptive algorithms and real-time data integration

9.1 Theoretical Framework

9.1.1 Traditional Methods for Address Database Management

9.1.2 Adaptive Algorithms: Concepts and Applications

9.1.3 Real-Time Data Integration Frameworks

9.2 Technical Framework

9.2.1 Data Sources

9.2.1.1 GPS Data

9.2.1.2 External APIs

9.2.1.3 User Inputs

9.2.2 Adaptive Algorithms

9.2.2.1 Fuzzy Matching

9.2.2.2 Machine Learning Model

9.2.2.3 Rule-Based Filters

9.2.2.4 Dynamic Duplicate Resolution

9.2.2.5 Real-Time Address Normalization

9.2.3 Evaluation Metrics

9.2.3.1 Accuracy

9.2.3.2 Latency

10 Traditional Methods for Address Database Management

11 Adaptive Algorithms: Concepts and Applications

12 Real-Time Data Integration Frameworks

13 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 an unified API.
It effectively manages all requests from the mobile app, where guides interact with addresses which are in their area. Furthermore, it manages all necesssary 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 has.

One big function of the backend is to enable the admins to perform a wide range on CRUD (Create, Read, Update, Deleter) operations on all necesssary data. Nevertheless, the other function of the backend is to provide guides with their relevant area and all the addresses in this area. This 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 Sprin 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.

13.1 Config of Spring Boot (application.properties)

The `application.properties` file in our Spring Boot application is the main configuration file where all the necesssary 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.

13.1.1 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
```

Quellcode 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 effieciently.

13.1.2 JPA and Hibernate Settings

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

```
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
```

Quellcode 2: JPA/Hibernate Settings

- `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.

13.1.3 Server Configuration

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

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

Quellcode 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.

13.1.4 CORS Configuration

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

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

Quellcode 4: CORS Configuration

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

13.2 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 entites easier to understand. For example, the area entity looks like this:

```
@Data
@Entity
@Table(name = "gebiet")
public class Area {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "gebietid")
    private Long areaId;

    @Column(name = "bezeichnung")
    private String desc;
}
```

Quellcode 5: Area Entity

In this class, the `@Data` annotation is used to generate necesssary 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 an unique identifier which is defined by a `house number` and a `street`. Therefore, we can not 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 to generate an artificial identifier.

```
@Data
public class AddressId implements Serializable {
    private String houseNumber;
    private Long streetId;
}
```

Quellcode 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 everytime the admin adds a new address.

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

```
@IdClass(AddressId.class)
public class Address {
    @Id
    @Column(name = "Hausnummer")
    private String houseNumber;

    @Id
    @Column(name = "Strasseid")
    private Long streetId;
}
```

Quellcode 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. Aftrwards, 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 is 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 are 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:

```
@Column(name = "latitude")
private double latitude;
```

```

@Column(name = "longitude")
private double longitude;

@Column(name = "schonbesucht")
private Boolean alreadyVisited;
```

Quellcode 8: "Normal" Variable

In this case, these variables safe the location data of each address and if 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 datatype.

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

```
@ManyToOne
@JoinColumn(name = "gebietid")
private Area area;

@ManyToOne
@JoinColumn(name = "besonderheitid")
private SpecialFeature specialFeature;
```

Quellcode 9: ManyToOne Variable

This code snippet defines two variables in our address entity which 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 entites to make sure that the relationship of the two entites 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 variable have more complex relationships than just with a simpl identifier that gets compared. We had such a case in our project and it came out like this:

```
@ManyToOne(cascade = CascadeType.PERSIST)
@JoinColumns({
@JoinColumn(name = "Strasseid", referencedColumnName = "Strasseid", insertable =
    false, updatable = false),
@JoinColumn(name = "Postleitzahl", referencedColumnName = "Postleitzahl",
    insertable = false, updatable = false)
})
private Street street;
```

Quellcode 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.

13.3 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:

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

Quellcode 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:

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

Quellcode 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 any way, there are some specific cases where automatic query generation does not work. In our project, we encountered an issue where Hibernate had problems to correctly insert a new address into the table. To solve this problem, we had to create this custom SQL query in the repository:

```
@Modifying
@Transactional
@Query(value = "INSERT INTO adresse (hausnummer, strasseid, postleitzahl,
    latitude, longitude, besonderheitid, gebietid, schonbesucht, kommentar)" +
        "VALUES (:houseNumber, :streetId, :postalCode, :latitude, :longitude,
            :specialFeatureId, :areaId, :alreadyVisited, :comment);",
        nativeQuery = true)
```

```
int insertAddress(@Param("houseNumber") String houseNumber,
                  @Param("streetId") Long streetId,
                  @Param("postalCode") String postalCode,
                  @Param("latitude") double latitude,
                  @Param("longitude") double longitude,
                  @Param("alreadyVisited") Boolean alreadyVisited,
                  @Param("areaId") Long areaId,
                  @Param("specialFeatureId") Long specialFeatureId,
                  @Param("comment") String comment);
```

Quellcode 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 would only be 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.

13.4 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 retireval and manipulation of the data.

13.4.1 @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 components scanning and dependency injection. Every service clas interacts with atleast one repository which gets injected through a constructor-based dependency injection using the `@Autowired` annotation. This injection looks like this in the service class:

```
@Service
public class AddressService {
    private final AddressRepo addressRepo;
    private final AreaRepo areaRepo;
    private final StreetRepo streetRepo;
    private final SpecialFeatureRepo specialFeatureRepo;

    @Autowired
    public AddressService(AddressRepo addressRepo, AreaRepo areaRepo, StreetRepo
        streetRepo, SpecialFeatureRepo specialFeatureRepo) {
```

```
        this.addressRepo = addressRepo;
        this.areaRepo = areaRepo;
        this.streetRepo = streetRepo;
        this.specialFeatureRepo = specialFeatureRepo;
    }
}
```

Quellcode 14: Repository Injection and @Service

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

13.4.2 Repository Usage in Service Methods

Now that the service class has all the necesssary repositories, it has to use the methods too. Some common operation look like this:
This method returns all addresses in the database:

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

Quellcode 15: Fetching all Addresses

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

```
public Address toggleAlreadyVisited(String houseNumber, Long streetId) {
    Address address = addressRepo.findAddressByHouseNumberAndStreetId(houseNumber,
        streetId);

    address.setAlreadyVisited(address.getAlreadyVisited() == null ||
        !address.getAlreadyVisited());

    log.info(address.toString());

    return addressRepo.save(address);
}
```

Quellcode 16: Toggle Already Visited

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

```
@Transactional
public void createNewAddress(
    String houseNumber,
    double lat,
    double lng,
    String areaDesc,
    String postalCode,
    String streetName,
    String specialFeatureText,
    String comment
) {
    int insert = addressRepo.insertAddress(
                    address.getHouseNumber(),
                    address.getStreetId(),
                    address.getStreet().getPostalCode(),
                    address.getLatitude(),
                    address.getLongitude(),
                    address.getAlreadyVisited(),
                    address.getArea().getAreaId(),
                    address.getSpecialFeature().getSpecialFeatureId(),
                    address.getComment()
    );
}
```

Quellcode 17: Insert new Address

This methods from the repositories are used in almost every methods from 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.

13.4.3 Computing Border Addresses Using Convex Hull

An important function of the backend is to calculate the border addresses which 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 method and an extra class so the calculation does not use so 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:

```
@AllArgsConstructor
@Data
public class Point {
    private double longitude;
    private double latitude;
}
```

Quellcode 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:

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

Quellcode 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 counter clockwise. This is important to make sure the convex hull is actually convex and not concave. The method to calculate this looks like this:

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

Quellcode 20: is Counter Clockwise Method

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

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

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

Quellcode 21: if Clause for Point Size

If there 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.

```
Point lowest = points.stream()
    .min(Comparator.comparing(Point::getLatitude).thenComparing(Point::getLongitude))
    .orElse(null);
```

Quellcode 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 done by using a comparator given by Java.

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

Quellcode 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));
```

Quellcode 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);
```

Quellcode 25: Convex Hull Calculation

In the `getBorderAddresses` method, the `convexHull` method is used and the convex 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 creatin 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 are returned.

```
List<Point> hull = convexHull(points);

List<Address> borderAddresses = new ArrayList<>();

for (Point p : hull) {
    for (Address address : addresses) {
        double latDiff = Math.abs(address.getLatitude() - p.getLatitude());
        double lonDiff = Math.abs(address.getLongitude() - p.getLongitude());

        if (latDiff < TOLERANCE && lonDiff < TOLERANCE) {
            borderAddresses.add(address);
            break;
        }
    }
}

return borderAddresses;
```

Quellcode 26: Lowest Point Calculation

13.5 Rest Controller (API Endpoints and their Functions)

REST controllers expose API endpoints that handle HTTP requests and then return the appropriate repsonses. These controllers allow the frontend-interfaces (Admin Panel and mobile app) to interact in any way with the back-end 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`.

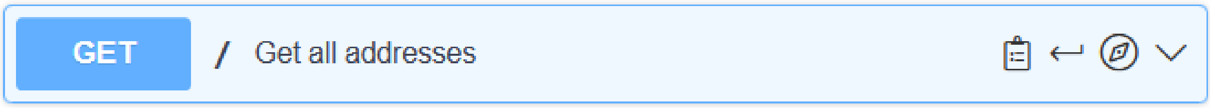
13.5.1 AddressController

The `AddressController` class is for handlin 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.

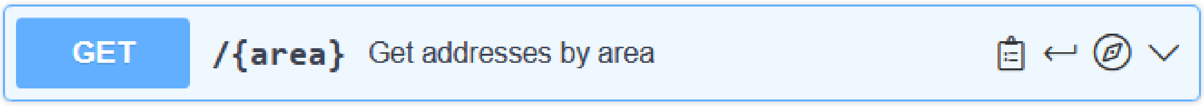


GET / Get all addresses

Abb. 8: 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

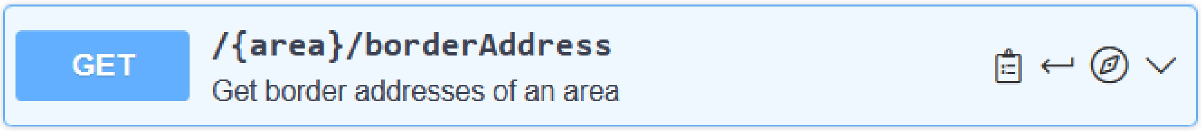


GET /{area} Get addresses by area

Abb. 9: 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.



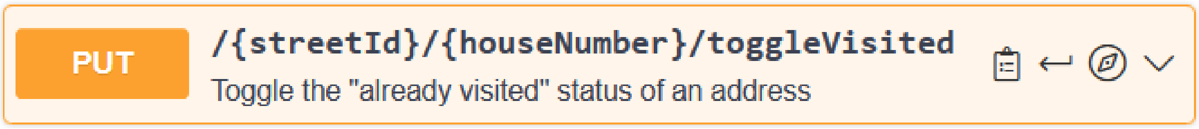
GET /{area}/borderAddress
Get border addresses of an area

Abb. 10: 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.

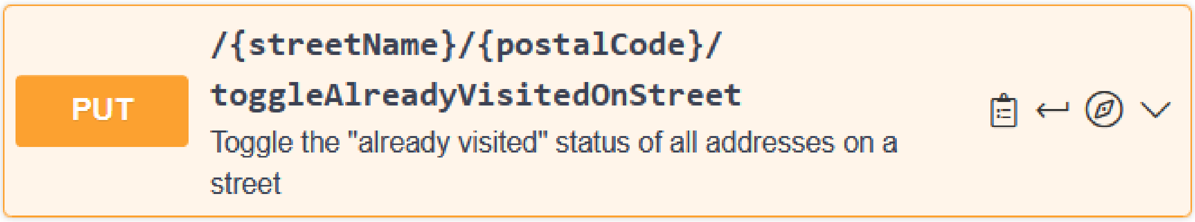


PUT /{streetId}/{houseNumber}/toggleVisited
Toggle the "already visited" status of an address

Abb. 11: Get Route for all Addresses

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.



PUT /{streetName}/{postalCode}/toggleAlreadyVisitedOnStreet

Toggle the "already visited" status of all addresses on a street

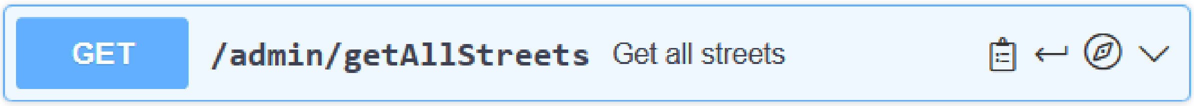
Abb. 12: Get Route for all Addresses

13.5.2 Admin Controller

This controller has all the endpoints relevant to managin the data. This routes are not meant to 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

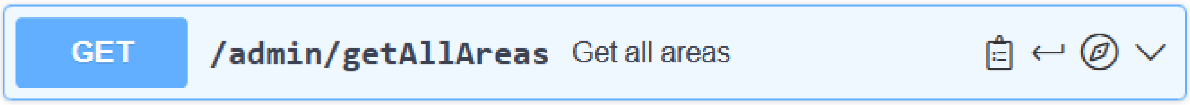
1. `GET /getAllStreets` **- Get All Streets**
   - This endpoint retrieves a list of all streets in the system.



GET /admin/getAllStreets Get all streets

Abb. 13: Get Route for all Addresses

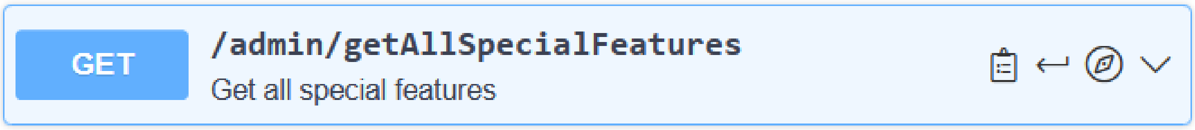
2. `GET /getAllAreas` **- Get All Areas**
   - This endpoint returns a list of all areas in the system.



GET /admin/getAllAreas Get all areas

Abb. 14: Get Route for all Addresses

3. `GET /getAllSpecialFeatures` **- Get All Special Features**
   - This endpoint retrieves all special features present in the system.



GET /admin/getAllSpecialFeatures

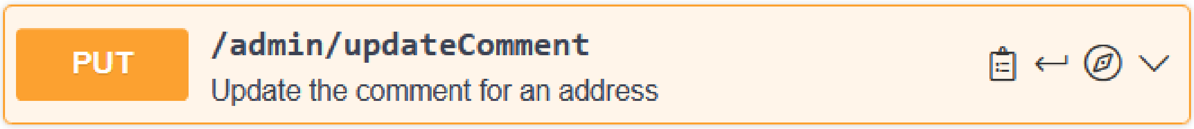
Get all special features

Abb. 15: Get Route for all Addresses

PUT Routes

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

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



PUT

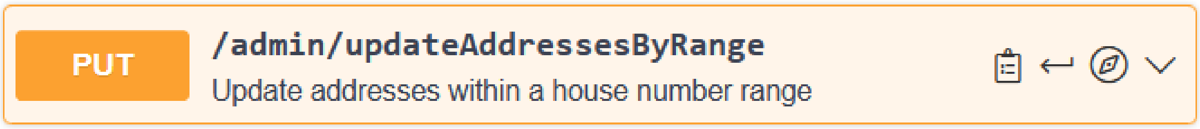
/admin/updateComment

Update the comment for an address

Abb. 16: Get Route for all Addresses

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

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



PUT

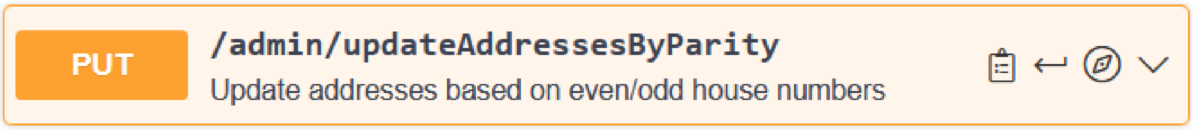
/admin/updateAddressesByRange

Update addresses within a house number range

Abb. 17: Get Route for all Addresses

3. `PUT /updateAddressesByParity` **- Update Addresses by Parity (Odd/Even)**

   - This endpoint updates addresses based on their house number parity (odd or even).



PUT

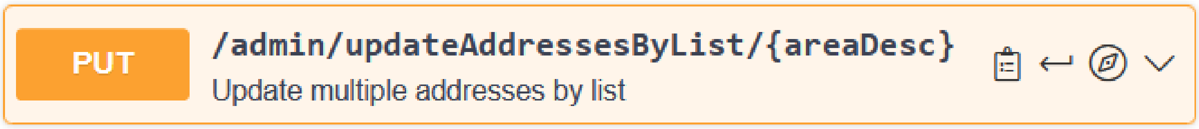
/admin/updateAddressesByParity

Update addresses based on even/odd house numbers

Abb. 18: Get Route for all Addresses

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.



PUT

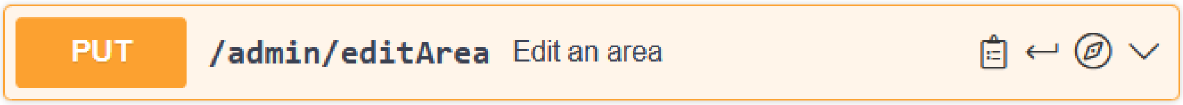
/admin/updateAddressesByList/{areaDesc}

Update multiple addresses by list

Abb. 19: Get Route for all Addresses

5. `PUT /editArea` **- Edit Area Information**

   - This endpoint allows modification of an area's description and associated details.

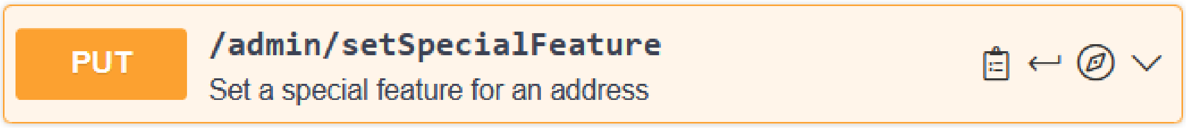


PUT /admin/editArea Edit an area

Abb. 20: Get Route for all Addresses

6. `PUT /setSpecialFeature` - **Set Special Feature for Address**

   - This endpoint assigns a special feature to an address.

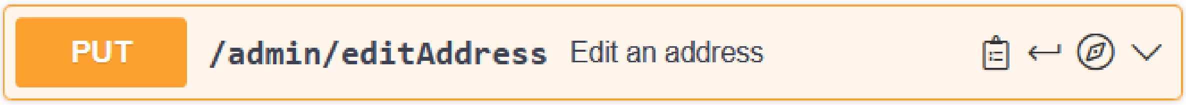


PUT /admin/setSpecialFeature
Set a special feature for an address

Abb. 21: Get Route for all Addresses

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.

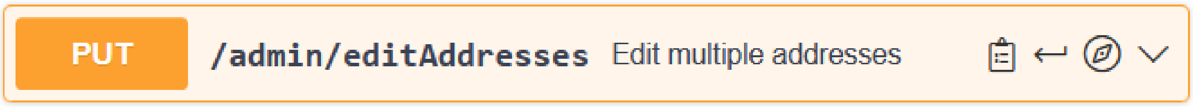


PUT /admin/editAddress Edit an address

Abb. 22: Get Route for all Addresses

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

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

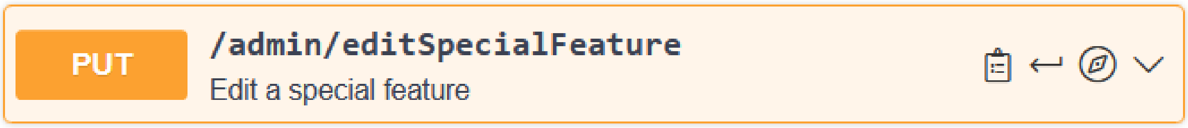


PUT /admin/editAddresses Edit multiple addresses

Abb. 23: Get Route for all Addresses

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

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



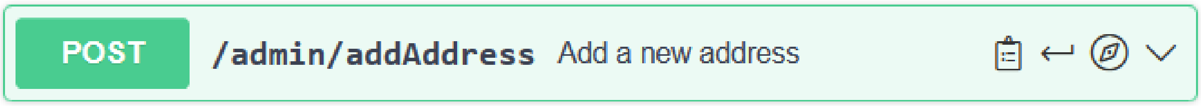
PUT /admin/editSpecialFeature
Edit a special feature

Abb. 24: Get Route for all Addresses

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.

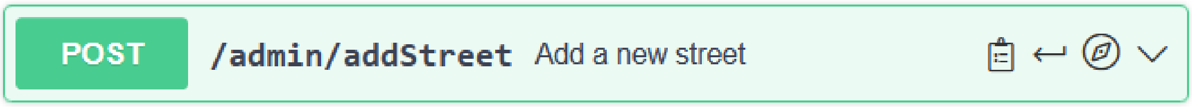


POST /admin/addAddress Add a new address

Abb. 25: Get Route for all Addresses

2. `POST /addStreet` **- Add a New Street**

   - This endpoint creates a new street record, specifying its name and postal code.

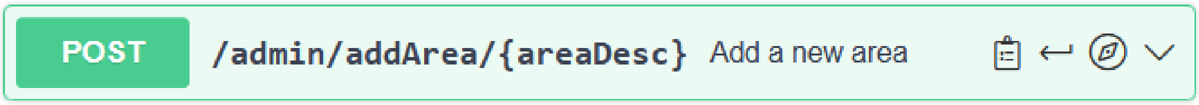


POST /admin/addStreet Add a new street

Abb. 26: Get Route for all Addresses

3. `POST /addArea/areaDesc` **- Add a New Area**

   - This endpoint allows the addition of a new area to the system.

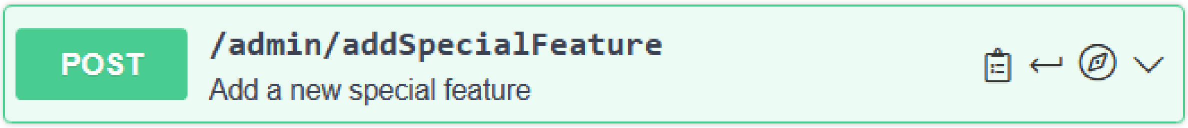


POST /admin/addArea/{areaDesc} Add a new area

Abb. 27: Get Route for all Addresses

4. `POST /addSpecialFeature` **- Add a New Special Feature**

   - This endpoint creates a new special feature.



POST /admin/addSpecialFeature Add a new special feature

Abb. 28: Get Route for all Addresses

DELETE Routes

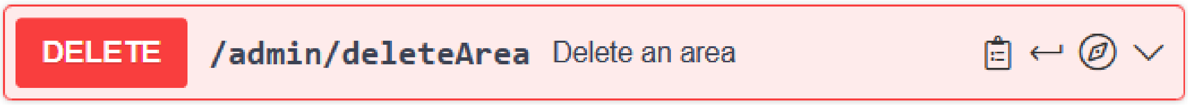
1. `DELETE /deleteStreet` **- Delete a Street**

   - This endpoint deletes a street from the system.

DELETE /admin/deleteStreet Delete a street

Abb. 29: Get Route for all Addresses

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.



DELETE /admin/deleteArea Delete an area

Abb. 30: Get Route for all Addresses

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

DELETE /admin/deleteSpecialFeature Delete a special feature

Abb. 31: Get Route for all Addresses

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

DELETE /admin/deleteAddress Delete an address

Abb. 32: Get Route for all Addresses

14 GraphHopper Setup

14.1 Why use GraphHopper?

14.2 Configuration

14.3 Local hosting

15 Working out the Wireframes

15.1 Map View

15.2 List View

15.3 Possible improvements for future versions

16 Functional implementation behind the application

16.1 Address-Provider

16.2 HTTP-Requests

16.3 Implementation of the Flutter Map Component

17 The app in use

17.1 Introducing new users

17.2 The app in operation

17.3 User Feedback

18 Final Thoughts

18.1 Leon Edlinger

18.2 Paul Gigler

18.3 Andreas Weissl

19 Meetings

Protokolle der Meetings, vielleicht auch ein zeitplan wann immer und wie lang

20 Working Hours

Arbeitspaket-Nr.	Beschreibung	Dauer
1	Einführung und Einarbeitung	8 h
2	Grundkonzept erstellen	8 h
3	Struktur der App festlegen	6 h
5	Wifi-Socket in App implementieren	39 h
6	Write-Funktionalität in App implementieren	14 h
7	Read-Funktionalität in App implementieren	19 h
8	Trim-Funktionalität in App implementieren	10 h
9	Konfigurationsmöglichkeiten für Flug in App implementieren	16 h
10	Höhenregelung-Funktionalität in App implementieren	14 h
12	Graphische Darstellung der Flugdaten	18 h
14	App testen und debuggen	19 h
26	Gesamtkonzept testen und debuggen	16 h
Summe		**187 h**

Table 1: Arbeitszeitnachweis

21 Source code directory

Source Code directory, kein plan was des is

22 List of figures

23 List of tables

24 Bibliography

[] *Flutter for Beginners*. URL: https://books.google.at/books?hl=de&lr=&id=pF6vDwAAQBAJ&oi=fnd&pg=PP1&dq=benefits+dart+language&ots=dZJWUGVs4x&sig=a196WqhXmQzuy23cmcKpEpIqn_k&redir_esc=y#v=onepage&q=benefits%20dart%20language&f=false.

[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.

[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.

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

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

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

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

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

[25f] *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.

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

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

[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.

[Con24] 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. *Delaunay triangulation - Wikipedia*. [Online; accessed 8. Mar. 2025]. Mar. 2025. URL: https://en.wikipedia.org/w/index.php?title=Delaunay_triangulation&oldid=1278482810.

[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.

[Gee24] GeeksforGeeks. “Convex Hull Algorithm”. In: *GeeksforGeeks* (Aug. 2024). URL: https://www.geeksforgeeks.org/convex-hull-algorithm.

[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.

[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.

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

[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.

25 Abbreviation

ADC	Analog Digital Converter
API	Application Programming Interface
BLE	Bluetooth Low Energy
CPU	Central Processing Unit
DAC	Digital Analog Converter
DAVE	Digital Application Virtual Engineer
DSP	Digital Signal Processor
FPU	Floating Point Unit
FPV	First Person View, First Pilot View
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GUI	Graphical User Interface
HDMI	High Definition Multimedia Interface
I^2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IP	Internet Protocol
RPI	Raspberry Pi
SD	Secure Digital
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver Transmitter
WLAN	Wireless Local Area Network
WPA	WiFi Protected Access
XML	Extensible Markup Language