HTL Leonding
next level

HÖHERE TECHNISCHE
BUNDESLEHRANSTALT
LEONDING

SeatGen - The Seating Plan Generation Tool For Stadiums

DIPLOMA THESIS

submitted for the

Reife- und Diplomprüfung

at the

Department of IT-Medientechnik

Submitted by:
Michael Ruep
Michael Stenz

Supervisor:
Prof. Mag. Martin Huemer

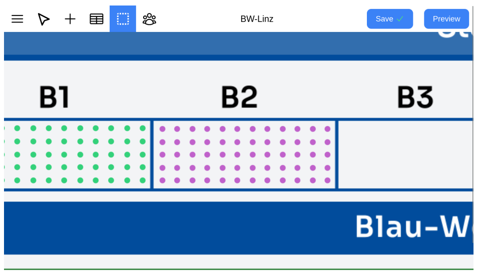
Project Partner:
Solvistas GmbH

Leonding, April 4, 2025

I hereby declare that I have composed the presented paper independently and on my own, without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such. This paper has neither been previously submitted to another authority nor has it been published yet. The presented paper is identical to the electronically transmitted document.

Leonding, April 4, 2025 Michael Ruep & Michael Stenz

Abstract



Seatgen is a internal tool for the company Solvistas. We cooperated with Solvistas to help them with their organization and management of their product, which sells tickets for sport-events which take place in stadiums. Seatgen allows the members of Solvistas to create and edit stadium plans in a fraction of the time that it used to take. With a selection of our handy tools, the workflow to create an entire seating plan gets very efficient and allows the people to create, move and edit seats, areas and more. The tool is designed to be user-friendly and intuitive so that the people at Solvistas don't have to spend a lot of time learning new software.

Inhaltsverzeichnis

1 Introduction

1.1 Initial Situation

The company Solvistas GmbH is a software development company, and one of their main products is the Ticketing project. Ticketing is a software solution that enables customers to purchase tickets for seats or sections in stadiums and other venues hosting events. The software is used by various sports clubs and event organizers to manage ticket sales for their events.

1.2 Problem Statement

The as just mentioned stadiums and venues have a lot of seats and different areas, and therefore the Ticketing software needs to know the layout of the seats. These layouts can have lots of complex shapes like curves and other irregular shapes. The current process of creating these so-called seat plans is done manually by editing text files. There are many problems, and it's a very tedious process when editing seat plans within a text editor. To name a few: When changing the layout of a stadium, all the text files have to be reworked by a schooled developer. This costs the customer a lot of money, and the developer a lot of time. Also, it's very hard to imagine how the rendered plan looks, when staring at text files.

Uploading the plan image is another tedious task when creating new plans. To convert the given SVG file into a functional map compatible with their system, the developer must manually upscale and slice the SVG into tiles, repeating this process for each zoom level—typically 5 to 7 times. Additionally, since each tile is divided into four smaller tiles at every zoom level, the number of tiles increases exponentially. As a result, a massive number of files must be uploaded to an AWS S3 bucket, making the process even more time-consuming.

1.3 Goal

The goal of the diploma thesis was to develop a custom solution for the company Solvistas and solve all these aforementioned problems with a tool that's intuitive to use and easy to learn, saving time and costs. We wanted to create a visual editor to create and manage seat plans for events in a stadium. This editor allows customers to create and edit new seating plans themselves, making the process so accessible and easy to use that no more schooled developers are required to make changes in a seating plan.

@Huemer Milestomes ????

2 Umfeldanalyse

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula. Citing [1] properly.

Was ist eine GUID? Eine GUID kollidiert nicht gerne.

Kabellose Technologien sind in abgelegenen Gebieten wichtig [2].

3 Technologies

3.1 React

3.2 Spring Boot and Kotlin

For the backend logic, we had to use Spring Boot, because Spring boot is in the main Stack of Solvistas, and the project should later be maintainable by Solvistas developers. The problems, that our backend had to solve were: handling the storing of the Seatplan metadata, converting the SVG's into image tiles, uploading the converted tiles to an S3 bucket, and serving all of this data to our frontend via REST. For the converting, resizing and slicing of the SVG's and PNG's, we considered Python as an alternative, because there are lots of easy to use and well documented image manipulation libraries like CairoSVG or the OpenCV python wrapper, but the processing of the images is also possible within Java/Kotlin with libraries like Batik and ImageIO, but it's not as straight forward as in python because it's not as popular and therefore there's a lot less documentation, and there are other limitations like heap size always have to be kept in mind. As for the uploading the files to an S3 Bucket, Amazaon provides good support for Java and Kotlin with their S3 SDK for Java/Kotlin and also has lots of documentation and examples all the operations with the S3 buckets. In the end we went with Spring boot and Kotlin because of our proficiency and expertise with the language, and all the other components of the Ticketing software were also written in Spring Boot.

As for the language we used Kotlin in Spring boot even though it's not used in many of Solvistas projects. We still decided that Kotlin was the better option because it is a modern language that is fully interoperable with Java and has many features that make it easier to write clean and concise code, thus reducing errors, improving readability and maintainability. It eliminates much of the boilerplate code required in Java and provides a rich standard library with many built-in utility functions, significantly reducing development time. Kotlin has no essential functionalities that java couldn't provide, but it is more modern and has a more concise syntax. TODO: Da a bissi mehr schreibi so kotlin vs java ka

3.3 Database

We chose PostgreSQL as our database for several key reasons. First, we needed a relational database because our data follows a structured design that is best represented through classic relations. Additionally, we wanted to simplify the process of exporting generated data into the Ticketing database, which is also relational.

Our system is designed to work with multiple relational databases, not just PostgreSQL, as we use JPA as our ORM. To maintain database flexibility, we deliberately avoided PostgreSQL-specific commands. While using PL/pgSQL for business logic could have improved security, performance, and data consistency, we prioritized keeping our database easily interchangeable.

For the database connection in our Spring Boot application, we used the Spring Data JPA library. This library simplifies the process of connecting to a database and executing queries. It also provides a repository pattern that allows us to define custom queries in an interface, which Spring Boot automatically implements at runtime. This pattern makes it easy to write and maintain queries, as we can use the repository methods directly in our code. We also use the Flyway library to manage database migrations. Flyway allows us to define database changes as SQL scripts, which are executed automatically when the application starts. This ensures that our database schema is always up-to-date with the latest changes. Doing migrations this way simplifies the deployment process drastically and helps avoid conflicts in the deployment and on the local production environment. Another advantage of using Flyway database migrations instead of migrating the database by hand, is that other developers in the team don't run into unexpected errors because of different database versions, and because migrations consist of whole SQL scripts, we have the possibility to not only execute DDL (Data Definition Language) commands but also DML (Data Manipulation Language) commands, which can be useful in a lot of scenarios, for example when migrating data from one table to another, changing the data type of column, and applying other business logic on data in the tables. We had to decide between Liquibase and Flyway for the database migrations. Both tools are very similar in a lot of aspects. They are both open source, both provide support for Spring Boot and other Java frameworks. The key differences are very subtle for out use case which does not need any complex features. We decided to go with Flyway because we do not need the more features and flexibility that Liquibase provides. Because we are a small team and changes on the database don't happen often in parallel, we have no problem with using Flyway's linear database migration approach, even though it could lead to problems in bigger teams with more parallel changes on the database. Flyway's versioning is also cleaner because the filenames of the migrations have to start with a specific prefix that contains the version of the migration in the following form `VX.X.X__migration_name.sql` (where X.X.X is the version of the migration). On the other hand Liquibase stores its changes in changelog files, this would allow a lot more features but also comes with a lot of unnecessary complexity. A changelog file can be written in SQL, XML, YAML, or JSON formats, but comments there is a lot of overhead and unnecessary metadata that has to be specified as seen in

3.4 AWS

3.5 Leaflet

4 Implementation

4.1 Leaflet

4.1.1 Writing Extensions

4.2 Map Generation

4.2.1 AWS

4.2.2 Image Processing

4.3 Add-Tool

4.4 Multiselect-Tool

4.5 Grid-Tool

4.6 Standing-Area-Tool

4.6.1 Frontend

4.6.2 Backend

4.7 Optimizations

4.8 Design-Patterns

5 Summary

Literaturverzeichnis

[1] P. Rechenberg, G. Pomberger *et al.*, *Informatik Handbuch*, 4. Aufl. München – Wien: Hanser Verlag, 2006.

[2] Association for Progressive Communications, „Wireless technology is irreplaceable for providing access in remote and scarcely populated regions,” 2006, letzter Zugriff am 23.05.2021. Online verfügbar: http://www.apc.org/en/news/strategic/world/wireless-technology-irreplaceable-providing-access

Abbildungsverzeichnis

Tabellenverzeichnis

List of Listings

Appendix