UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Kustu Künnapas

Orienteering Event Registration Software: From Prototype To Web Application

Bachelor's Thesis (9 ECTS)

Supervisor: Dietmar Pfahl, PhD

Tartu 2023

Orienteering Event Registration Software: From Prototype To Web Application

Abstract:

This Bachelor's thesis aimed to develop a user-friendly web application for managing orienteering event registration. The application was designed using the hexagonal architecture, which promotes modularity and adaptability. Java was used for the back-end, React for the front-end, and PostgreSQL as the database, and the application was developed in collaboration with the orienteering event organizer Seiklushunt to meet user needs and provide a reliable system.

The thesis describes the software development process and real-life testing during an orienteering event. As a result robust and adaptable system suitable for modern orienteering events was created. The thesis also discusses potential future development.

Keywords: Software development, Java, React, PostgreSQL, Orienteering,

CERCS: P175 Informatics

Orienteerumispäevakute registreerimise tarkvara: prototüübist veebirakenduseni

Lühikokkuvõte:

Selle bakalaureusetöö eesmärk oli arendada kasutajasõbralik veebirakendus orienteerumispäevakute registreerimise haldamiseks. Rakenduse loomisel lähtuti kuusnurksest arhitektuurist, mis toetab modulaarsust ja kohandatavust. Java’t kasutati tagarakenduse, React’i esirakenduse ja PostgreSQL’i andmebaasina ning rakendust arendati koostöös orienteerumisürituste korraldaja Seiklushunt'iga, et rahuldada kasutajate vajadusi ja pakkuda usaldusväärset süsteemi.

Töös käsitletakse tarkvaraarenduse protsessi ning rakenduse testimist orienteerumispäevaku ajal. Töö tulemusena loodi paindlik süsteem, mis sobib tänapäevastele orienteerumis -päevakutele. Lisaks arutatakse töös potentsiaalseid edasisi arendusvõimalusi.

Võtmesõnad: Tarkvaraarendus, Java, React, PostgreSQL, Orienteerumine

CERCS: P175 Informaatika

1. Introduction

Orienteering is a popular outdoor activity that involves navigating through a terrain using a map and a compass. It is often used for fitness, recreation, and competitive purposes. Orienteering events attract many participants, ranging from casual walkers to serious athletes. To manage these events, organizers require a registration system that can handle participant registration, payment processing or checking, and timekeeping devices management. The idea came from the author's own problems while organizing orienteering events.

In recent years, there has been a growing trend toward using digital technologies to streamline the registration process for orienteering events. However, most orienteering clubs still manage participants using paper and pen. As a result, there is a need for modern and customizable orienteering event registration software.

This bachelor thesis aims to develop orienteering event registration software that is scalable and accessible through a web application. The software will be designed to replace an existing prototype developed for a local orienteering event organizer, Seiklushunt. The new software will improve upon the prototype by adding new features, enhancing usability, and ensuring compatibility with modern web browsers.

The work is divided into six chapters. The first chapter is the introduction. The second introduces the general background of orienteering and the problem by describing the prototype. The third chapter describes the methods used to solve the problem. The fourth chapter presents the results, including the technologies used, the final solution, and the validation process of the completed program, including testing. Chapter five contains the discussion, including lessons learned and suggestions for further improvements. And sixth chapter is the conclusion.

2. Background

This chapter discusses the background of the problem and examines the existing prototype.

2.1 Background of orienteering

Orienteering is a sport in which the participant, the orienteer, navigates a given route using a map and compass [1]. Competitions and events can take place in forests or cities, presenting a unique challenge for the orienteer. Every course and event features different terrains and checkpoints.

Estonia has two kinds of orienteering events: competitions and evening orienteering events (EOEs). Competitions, primarily targeting competitive runners, generally take place on weekends from the end of April until the end of October and can be 1 to 3 days long [2]. EOEs, intended for a broader audience, typically occur weekly on Monday to Thursday, from April to October, in various locations across Estonia [1,3]. Some clubs hold EOEs all year round.

The primary distinction between competitions and EOEs lies in the registration process. Competitions require participants to register in advance, while EOEs allow individuals to attend the event during the designated start time without prior registration [1].

According to the Estonian orienteering federation's (EOF) 2022 annual report, there were 24 different EOE series with a total of 342 events, more than 6000 unique participants, and a total of more than 57000 participations [4]. Every orienteer in Estonia has a unique identification code called the EOF code.

2.1.1 Timekeeping

Each participant has a timekeeping device to record their result, and every checkpoint has a station that saves time information to the device. There are currently two major timekeeping systems in the market. The Emit eCard system is used mainly in Scandinavia countries, Finland, and partly in Denmark, Belgium, and the UK [5]. In contrast, other countries like Estonia utilize the SPORTident timing system, which relies on a timekeeping device known as the "SI-Card" [6]. A device called an "SI station" is used to read information from SI-Cards and transfer it to a computer.

A wide variety of event administration and timekeeping software is available[1], with most of them designed for competitions that require additional functionality. Consequently, these software solutions can be more complex to use and often necessitate the expertise of a professional timekeeper. In Estonia, RaceManager[2] by Tak-Soft is the most popular software for orienteering competitions, as the developer is Estonian. Tak-Soft also offers SPORTident Mini Events (SIME)[3], the most widely used software for managing small orienteering events and EOEs, including all the EOEs in Estonia.

SIME streamlines the process of reading out all competitors' information, particularly for those who have an EOF code or own an SI-Card associated with their EOF code. The software is designed to efficiently manage competitor data by accepting files containing the necessary information. This feature allows users to make modifications and establish new associations between SI-Cards and runners, ultimately simplifying the overall event management process [7].

2.1.2 Stebby

In Estonia, the government has introduced a tax exception for companies, allowing them to allocate up to 100 euros per employee per quarter for expenses related to improving employee health without incurring fringe benefit taxes [8]. To help large companies manage this benefit, Stebby has stepped in. According to Stebby OÜ (formerly SportID) webpage [9], this Estonian company, established in 2012, offers an online health and wellness services platform. The platform provides various services, including online coaching, personalized nutrition plans, fitness plans, and wellness challenges. Stebby's mission is to make health and wellness more accessible and affordable for individuals while enabling businesses to support employee well-being. The company partners with numerous gyms and wellness centers in Estonia, allowing users to access a wide array of services through the platform. Stebby has grown in popularity in Estonia and expanded its services to other European countries [9].

Orienteering is a popular activity on the Stebby platform. Around half of the orienteering series already have their events listed in Stebby, and as more of the participants use that, more series are expected to be listed there.

[1] https://orienteering.sport/iof/it/list-of-software-for-orienteering/
[2] https://www.tak-soft.com/tooted/sport/rm/index.php
[3] https://www.tak-soft.com/tooted/sport/sime/index.php

2.2 Challenges during event organizing

Evening orienteering events are designed to be straightforward, with participants paying for a map, running the chosen course, and checking out to receive their results. However, several challenges complicate the organization of these events.

One challenge involves SI-Card ownership. Many orienteers do not have their own SI-Cards, leading them to rent cards from event organizers. One SI-Card costs around 30 to 70 euros, depending on the model [10], so organizers should keep track of all rentable cards. Currently, organizers rely on paper records to monitor who has rented each card and when it was returned, if at all.

Another challenge comes from the use of multi-use tickets. Many orienteering clubs offer these tickets to reduce costs for frequent participants and streamline the payment process. However, tracking these tickets can be difficult, with most clubs resorting to paper records or trust-based systems.

Additionally, some companies form agreements with EOEs organizers, allowing their employees to use company benefits to attend events. This partnership creates an added layer of complexity in managing event participation and attendance.

Lastly, many first-time participants lack an EOF code, which slows down the SI-Card rental and checkout processes. Organizers must repeatedly collect and record the names of these participants across multiple systems and paper records, increasing the likelihood of errors. Moreover, using only a name as an identifier can lead to confusion and duplication, as it may not be unique.

2.3 Current prototype

The SIE[4] software, created by Kustu Künnapas in collaboration with Toomas Roosma, was developed to tackle the challenges identified in the orienteering event management process. This project was undertaken during the LTAT.03.003 Object-Oriented Programming[5] course in their first year of studies. The project competed in a Student project contest at the University of Tartu in 2021 and got first place in First-year Bachelor's category [11].

[4] https://www.kunnapas.com/sie/
[5] https://ois2.ut.ee/#/courses/LTAT.03.003/details version 20/21 K PÕ LT

The technology stack used for the application includes Java for the back-end, JavaFX for the front-end, and Google Sheets as the database, with additional text files serving as data sources.

2.3.1 Features of the prototype

The prototype for the orienteering event management application was created in collaboration with the non-profit organization Seiklushunt and incorporating domain knowledge, the prototype's features were tested during Seiklushunt's EOEs. Client expectations for the project were collected through interviews and text conversations conducted before and during the development of the prototype.

The main features of this prototype include:

- Communication with the SI station using GecoSI[6]
- Registration view
    - Prefilling forms based on SI-Card number and available data
    - Name/EOF code changing with data matching and prefilling
    - Displaying ticket options and preselecting. (multi-use tickets, company agreement)
    - Choosing a number of tickets to use
- Reading/writing data from/to Google Sheets[7] and information files
- Preparing a file for SIME with runners' information

The testing process led to numerous changes and improvements in the prototype. However, some issues could not be resolved due to the technical complexity of testing the program and the difficulty of obtaining the required physical hardware.

2.3.2 Issues

The existing SIE software prototype faces several issues that prevent it from attaining complete functionality.

Firstly, the development process was constrained by a limited understanding of programming concepts, resulting in a codebase that is difficult to read, extend, and maintain. This impacted the overall efficiency and adaptability of the software, making it challenging to address evolving requirements and implement necessary improvements.

[6] https://github.com/sdenier/GecoSI
[7] https://www.google.com/sheets/about/

Secondly, the software was designed as a monolithic Java application, which is not optimal for front-end desktop implementations. This architecture choice led to scalability and maintainability challenges, as well as hindered the user experience in desktop environments.

Thirdly, the process of updating the software is cumbersome, as it requires clients to obtain a new version of the application after each update. The delivery of updated executables is complex and time-consuming for both the developers and the end-users.

Lastly, the program lacks comprehensive automated testing coverage, impeding the addition of new code and the detection of issues, especially considering the numerous edge cases that may arise. As a result, the consequences of incorporating new code remain uncertain.

2.3.3 Functional limitations

The SIE software prototype demonstrates limited functionality in several aspects.

Firstly, the configuration of the program's parameters relied on a text file, increasing the possibility of errors. Similar problems occur with data scattering from Google Sheets, as any input can be written there, which may lead to errors and inconsistencies.

Secondly, the program offered only two views. The initial view facilitated the selection of a folder to store the data produced by the software. The subsequent view focused on competitor registration, in which related files were modified and, if required, data updates were reflected in Google Sheets.

2.3.4 Planned improvements

Based on the issues with the current SIE software prototype, several changes are necessary to develop a new version called SOE (Streamline Orienteering Events) to make it modern and user-friendly.

The initial step involves refactoring, restructuring, or rewriting the code to improve readability, extensibility, and maintainability. Subsequently, the software architecture should be modified to integrate a modern front-end web application framework, enhancing the user interface and experience. Lastly, comprehensive and automated testing coverage must be added to facilitate early issue detection and expedite bug fixes.

To achieve these modifications, the SOE software should be developed using contemporary web technologies for the front-end web application framework and an appropriate database

management system. JavaScript or TypeScript programming languages could be utilized for the front-end development, as they are modern and supported by a vast developer community.

To address the technical challenges related to software updates and Java version compatibility, a modern deployment methodology, such as containerization, should be adopted using popular tools like Docker or Kubernetes. This approach simplifies software deployment and reduces complexity, making the SOE more accessible to users.

Automated testing coverage should be implemented to ensure the new SOE software is fully functional by using modern testing frameworks, which are efficient and provide comprehensive testing coverage.

In conclusion, the SOE should employ modern web technologies, an up-to-date deployment methodology, and comprehensive automated testing coverage to overcome the issues associated with the existing SIE software. These improvements are expected to enhance the user interface and experience, support software maintainability, and shorten the time required for bug fixes.

3. Methodology

Developing the SOE from prototype to web application involves several steps to achieve the project's objectives. The following steps can be taken to implement the proposed solution.

3.1 Steps to solution

Step 1 - Identify features for the new application:

The goal of this step is to identify features for the new application using the prototype's features. This involves understanding the limitations and areas for improvement that need to be addressed in the new version of the software. This step also includes gathering user feedback on the prototype and prioritizing features based on their importance and impact.

Step 2 - Choose the technology stack:

The goal of this step is to find the appropriate technology stack for the web application. This includes selecting the programming language, web server, database management system, and other technologies required for the application's development.

Step 3 - Develop the web application:

The goal of this step is to develop the web application using the chosen technology stack and design specifications. This involves coding, testing, and debugging the software components to ensure they function correctly. Regular progress updates and communication with stakeholders should be maintained throughout this process.

Step 4 - Deploy the web application:

The goal of this step is to deploy the integrated system to a production environment. This involves setting up the necessary hardware and software infrastructure, configuring the system, and making it available to users. Additionally, ensure proper security measures are in place to protect the system and user data.

Step 5 - Validate the web application in a real-life scenario:

The goal of this step is to validate the created application in an EOE. This ensures that the created application is usable and the client validates the functionality. In addition, all the yet necessary functionality can be communicated so that when the applications going to be used in one day, it is fully functional and satisfies the customer. Additionally, problems that occurred during validating are analyzed.

3.2 Writing methodology

For the purpose of reviewing and enhancing the linguistic quality of this thesis, Grammarly and ChatGPT by OpenAI are used. These tools helped to ensure that the writing was more coherent and understandable, ultimately contributing to well-presented and polished work. It is important to note that all suggestions provided by these tools were critically reviewed.

4. Results

The results chapter of this thesis presents the outcomes of each step in the development process of the new version of the SOE. The chapter is divided into sections corresponding to the steps outlined in the methodology chapter. Each section highlights the key findings and improvements identified during the respective step. By presenting these results, this chapter provides a comprehensive overview of the project's progress and the factors that contributed to the successful development and implementation of the web application.

4.1 Identifying features for the new application

This section focuses on identifying features for the new SOE application. The evaluation process involved examining the prototype's features, understanding its limitations, and identifying areas for improvement that need to be addressed in the new web application. The knowledge gained from this assessment was essential in determining the functional requirements and guiding the following stages of the project.

4.1.1 Lifted features

The majority of the functional requirements for the new web application remain consistent with those of the prototype, especially from the client-side perspective. Some features have been lifted due to low popularity or time constraints.

First, the ability to use multiple tickets per registration has been removed. Users who wish to use two or more tickets must now register separately for each ticket. This change was made due to the low popularity of multiple ticket purchases. An analysis of Seiklushunt's multi-use ticket usage statistics revealed that only 5 out of 513 instances during the first four EOEs of spring 2023 involved more than one ticket being used at once. As registering separately for each ticket does not significantly increase the registration time, this adjustment was deemed acceptable.

Secondly, the implementation of company contracts functionality has been excluded from the current version, as this feature has limited appeal with the growing popularity of Stebby. According to the same dataset mentioned earlier, only three companies have used the company agreement system, accounting for just 20 instances during the first four weeks. Since excluding this feature does not create an excessive amount of manual work, it has not been developed in the current version.

Thirdly, multi-use ticket buyers without an EOF code will still need to be counted manually due to the error-prone nature of the process. This issue is not considered significant, as only a few individuals fall into this category. The dataset indicated that 22 multi-use tickets were purchased without an EOF code. As the number of such cases is expected to decrease over time with the creation of new EOF codes, this functionality is not prioritized in the current version.

Lastly, integrations with Stebby were initially planned but were removed due to time constraints. Currently, the Stebby API does not support this specific use case, meaning that direct communication with Stebby is necessary and might require some development on their end. This can be time-consuming and would require effort from both parties.

Nevertheless, additional requirements have been identified to address the distinct challenges of transitioning to a web application.

4.1.2 Functional requirements

The functional requirements outlined in this section will serve as a basis for the development of the new SOE application, with the goal of providing an efficient and user-friendly platform for orienteering event management. The following functional requirements have been chosen:

a) Front-end tasks:

- Login view
- Registration view
    - Communication with the SI station
    - Prefilling forms based on SI-Card number and available data
    - Name/EOF code changing with data matching and prefilling
    - Displaying ad preselecting multi-use ticket options
    - Showing ticket usage confirmation
    - Viewing multi-use tickets
- Administration view
    - Adding and viewing new events and event series
    - Adding new multi-use tickets
    - Adding new and viewing rental SI-Cards
    - Checking information about rental SI-Cards, including current use
    - Viewing program usage statistics

b) Back-end tasks:

- Providing filtered data exchange between the database and front-end
- Adding new data to the database from the front-end
- Preparing runners file for SIME
- Updating information about used tickets and payments

c) SIME script tasks:

- Fetch initial data from database when starting the script
- Updating the runners file periodically fetching new data on top of the initial file.

4.2 Used technologies

The development of the web application utilized a variety of technologies to enable its functionality. These technologies were selected based on their ability to support the project requirements and their compatibility with one another. The following section provides an overview of the technologies used in the development of the web application, including those used on the front-end, back-end, and database.

4.2.1 Back-end technologies

The back-end technology stack for the web application consists of Java with Spring Boot and jOOQ. According to Axon's article by Mykhailo Spirich [12], Java is a popular and widely used programming language for developing robust and scalable back-end systems. Its platform independence and wide library ecosystem make it suitable for building complex web applications [12].

Spring Boot, a popular Java-based framework, was chosen for its ability to accelerate web application development by providing a set of preconfigured modules and libraries that allow developers to create web applications without worrying about the underlying component configuration [13]. Spring Boot's easy-to-use configuration system simplifies configuring web applications and allows developers to customize the preconfigured defaults as necessary [13].

jOOQ [14], a framework that provides an API for accessing SQL databases, was selected for its ability to generate type-safe queries at compile time, which reduces the risk of runtime errors and improves code quality and maintainability. jOOQ's API allows developers to write complex SQL queries in Java code, which is intuitive and easy to read. Additionally, jOOQ is

vendor-neutral and supports multiple SQL dialects and database vendors, allowing developers to use the same API to access different databases and making it easier to switch between or support multiple databases in the same application [14].

4.2.2 Front-end technologies

React.js[8] with Material UI[9] was selected as the front-end technology stack for the web application. React.js is a widely used JavaScript library created by Meta (formerly Facebook), initially released in 2013, for building user interfaces, recognized for its efficiency, flexibility, and reusability [15]. Its component-based architecture allows developers to manage and maintain code for complex user interfaces [15]. React.js was preferred for its extensive community of developers and resources available for developers to access, contributing to the development of new libraries, components, and tools that simplify web application development [16].

Material UI is a popular open-source design framework that offers pre-built components and styling that can be customized to meet the application's design requirements [17]. Material UI is known for its high-quality design and meticulous attention to detail, making it a reliable choice for building user interfaces [17].

The combination of React.js and Material UI offers a powerful and flexible front-end technology stack, facilitating the development of high-quality, responsive, and intuitive user interfaces for the web application.

Initially, the npm package *sportident-react*[10] was chosen for communication between the web application and the SI station, influencing the selection of technology for the project. However, the package did not function as expected during testing due to a lack of technical documentation and examples.

4.2.3 Database

PostgreSQL, commonly referred to as Postgres, was chosen as the database management system for the web application. Postgres [18] is a powerful, open-source, and highly regarded relational database management system known for its advanced features, scalability, performance, stability, and security. It supports advanced features such as complex queries, indexing, and much more, making it flexible for storing and retrieving various types of data.

[8] https://react.dev/
[9] https://mui.com/
[10] https://www.npmjs.com/package/sportident-react

Postgres is also highly scalable and can easily handle large datasets, making it a suitable option for both small-scale and large-scale applications. The selection of Postgres as the database management system ensures efficient and secure data storage and retrieval for the web application [18].

4.3 Web application development

This section discusses the web application development process results, following the methodology outlined in Section 3. The discussion covers architectural decisions, front-end, back-end, and database designs, as well as code samples and testing strategies employed. Instructions for accessing the source code are provided in Appendix I.

4.3.1 Design pattern

Initially, the architecture of the SIE software was analyzed. It was quickly determined that the best course of action would be to rewrite the entire application, as SIE did not use a relational database, and a significant portion of the business logic was intertwined with the front-end code. This made the code unmaintainable, and it would have been difficult to identify any domain logic that could be reused.

An appropriate architecture for the SOE was chosen to build a scalable and maintainable web application. The hexagonal architecture was selected for the back-end system, as it aligns with the principles of Clean Architecture, separating concerns and minimizing dependencies, as described in Tom Hombergs' book, "Get Your Hands Dirty on Clean Architecture" [19].

As outlined in Hombergs' book, the pattern divides the application into three components: domain, ports, and adapters. The domain component comprises core business logic and models; the ports component establishes interfaces for communication between layers; and the adapters component implements these interfaces, connecting the infrastructure layer to the domain and application layers. The hexagonal design focuses on creating clear boundaries between components, fostering testability and adaptability [19].

The developed web application complies with the requirements presented in Section 4.2, centering on meeting user needs and constructing a dependable and efficient system. The application's architecture encourages modularity and facilitates effortless adaptation to future enhancements or modifications.

4.3.2 Database design and relations

As mentioned earlier, PostgreSQL was chosen as the database engine. The Entity-Relationship (ER) diagram, which illustrates the relationships among entities in the database, is provided in Appendix II. As depicted in the diagram, each table includes fields for "created," "created by," "modified," "modified by," and "version," ensuring audibility and enabling the tracking of changes.

The database is designed to accommodate multiple clubs using the software simultaneously, with the "club" entity serving as a central component connecting all other entities. Apart from login information and user entities, two primary circles of functionality are evident within the ER diagram.

The first circle centers around the "si_card" entity, where an individual can possess multiple SI-Cards and rental cards are assigned to a specific club. Based on this information, files can be packaged and sent to SIME.

The second circle encompasses the logic of event series, events, tickets, and multi-use tickets. A multi-use ticket contains references to both the "series_id" and "club_id," as different clubs operate unique multi-use ticket systems. Some clubs issue multi-use tickets for specific event series with an expiration date, while others allow the use of tickets for all club events, regardless of the series.

By utilizing a well-designed database schema, the web application effectively supports multiple clubs' needs and ticketing systems, ensuring smooth operation and flexibility.

4.3.3 Programm logic diagram

This section demonstrates how the main logic's user side operates.

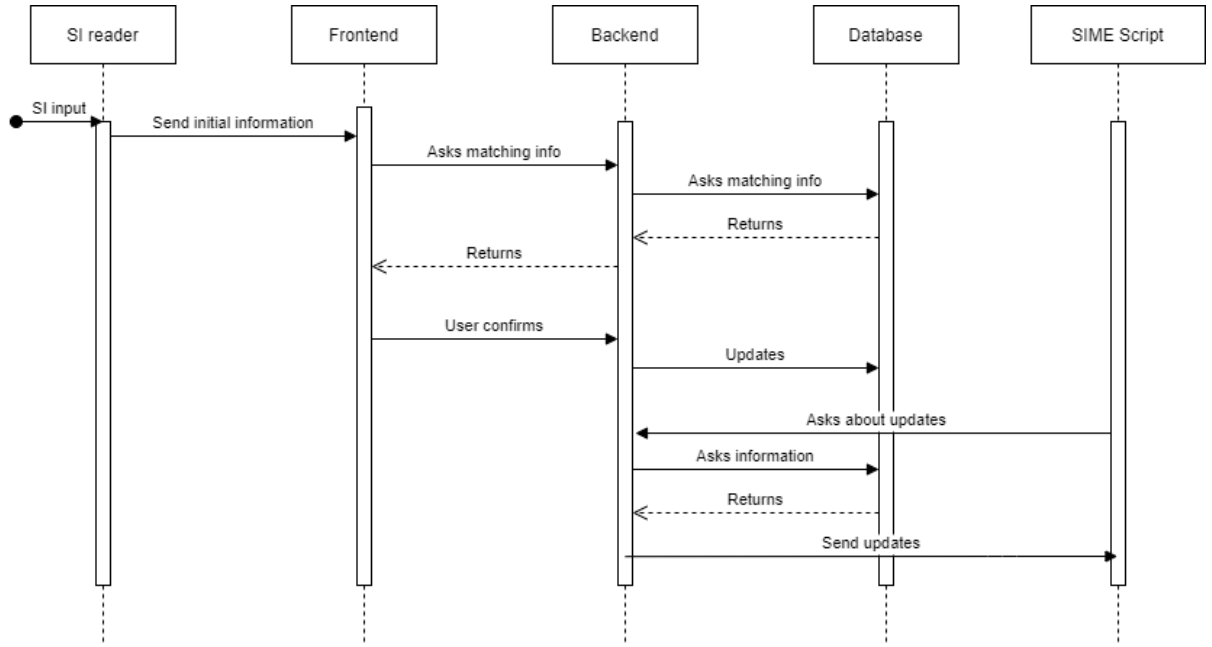


Figure 1: Sequence diagram

Fig. 1 illustrates the typical workflow and interactions between various program components. The process begins with SI-Card input to the SI station, which sends the information to the front-end application. Next, a request is sent to the back-end, which retrieves the relevant data from the database. Users can modify the data before sending another request to the back-end, if needed, following the same procedure. Once the information is verified and confirmed, it is sent to the back-end for storage. Periodically, the SIME script sends a request to the back-end to inquire about updates. The back-end then retrieves these updates from the database and sends them back as a response.

4.3.4 Technical description

The backend application comprises six distinct modules, each responsible for a specific function:

1. Database Module: This module consists of Liquibase[11] scripts, enabling programmatic database creation. This ensures that the database always contains the correct tables and configurations.
2. Web Adapter Module: the Web Adapter manages incoming traffic via HTTP requests and returns HTTP responses to the client. As a web application, this is the first point of entry.
3. Domain Module: this module is called by the Web Adapter and manages all business logic, use cases, services, and entities. It serves as the core of the application.
4. Jooq Module: located within the adapter package, the Jooq Module is called by the Domain Module for all database communication.
5. Common Module: this module manages classes used across multiple modules, such as authentication context, utilities, and exceptions.
6. Config Module: this final module is responsible for running the application and houses security configurations along with other settings.

A crucial aspect of the hexagonal design is the implementation of adapters through interfaces. This approach ensures that if a component like jOOQ becomes deprecated, only its implementation needs to be replaced while the rest of the system continues to function seamlessly.

Classes

All domain entities are organized within a domain class, as illustrated in Fig. 2, except authentication as it is also needed by the configuration module.

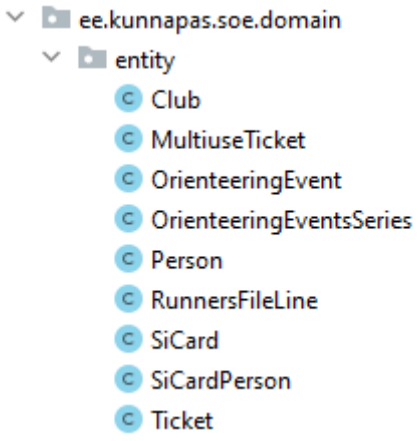


Figure 2: Entities in domain module

[11] https://www.liquibase.org/

The domain class also comprises loading and storing ports utilized by services and implemented in the jOOQ modules through gateways, as shown in Fig. 3.

At the heart of the domain logic are use cases and services. These components follow the single-responsibility principle and execute specific tasks using ports. This principle ensures that the classes are easy to test and use. If additional logic is required, new use cases can be added seamlessly, maintaining the simplicity and clarity of the domain logic. This approach allows for the incorporation of new functionality on the fly without risking disruption to the existing system.

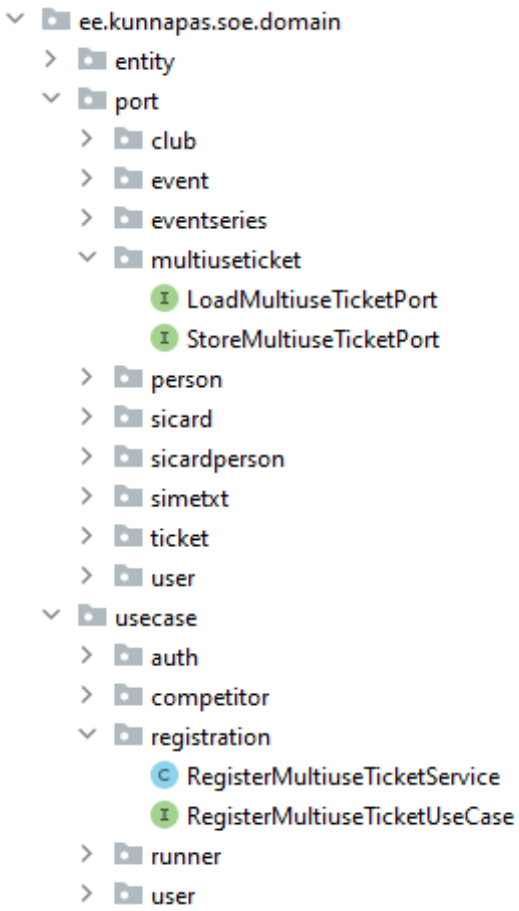


Figure 3: Ports and use cases in Domain module

The jOOQ module contains all of the ports implementations via gateways and mappers with utils that are used to implement communication between the domain and database.

The web adapter plays a crucial role in processing HTTP requests. Data transfer objects (DTOs) were used to convert web input objects into domain entities and vice versa. This

approach ensures efficient communication between the web layer and the domain layer, streamlining the exchange of information within the application.

Code examples

In the following section, the use case of registering the usage of a multi-use ticket is analyzed with code snippets.

The *createTicket* method within a controller class represents the entry point for this use case. This method is mapped to the "/tickets/multiuse" endpoint, as depicted in Appendix III.

The method accepts a *TicketDto* object as input from an HTTP request containing PersonId, MultiuseTicketId, EventId, SeriesId, and ClubId. The method then proceeds to validate the following:

1. The authenticated user has the appropriate rights to perform the action for the given club.
2. The referenced *Person*, *MultiuseTicket*, and *Event* objects are present and have the correct references.

If the validation fails, an error is thrown, and an exception handler returns an appropriate error code in the HTTP response.

If all validations pass, the method calls the *registerMultiuseTicketUseCase* with the collected and validated objects. This use case processes the ticket registration, and upon successful registration, the method returns an HTTP response with a 200 status code and a "Ticket created" message.

The *createTicket* method ensures that only valid tickets are created, maintaining data integrity and providing a secure way for users to register multi-use ticket usage.

Fig. 4 shows an example where *RegisterMultiuseTicketService* receives the mentioned objects as input. The service creates a new *Ticket* object and associates it with the given *MultiuseTicket* and *OrienteeringEvent*. The purchase time is set to the current time.

If there is only one ticket left in the *MultiuseTicket*, it is marked as used by setting the isUsed property to true, and the *MultiuseTicket* is then stored using the *storeMultiuseTicketPort*. Finally, the newly created *Ticket* object is stored using the *storeTicketPort*.

```java
@Component
@RequiredArgsConstructor
public class RegisterMultiuseTicketService implements RegisterMultiuseTicketUseCase {

    1 usage
    private final StoreTicketPort storeTicketPort;
    1 usage
    private final StoreMultiuseTicketPort storeMultiuseTicketPort;


    @Override
    public void execute(Request r) {

        Ticket ticket = Ticket.builder()
                .multiuseTicketId(r.getMultiuseTicket().getId())
                .event(r.getEvent())
                .purchaseTime(LocalDateTime.now())
                .build();

        if (r.getMultiuseTicket().getNumberOfTicketsLeft() == 1) {
            r.getMultiuseTicket().setIsUsed(true);
            storeMultiuseTicketPort.store(r.getMultiuseTicket());
        }

        storeTicketPort.store(ticket);
    }
}
```

Figure 4: *RegisterMultuseTicketService* code example

The *StoreTicketGateway* class uses the jOOQ library, a popular Java-based database access framework that simplifies data access and object-relational mapping. As seen in Fig. 5, the adapter class contains minimal code, primarily focusing on handling the store operation and mapping between domain entities and jOOQ database entities.

The store method in the *StoreTicketGateway* class accepts a *Ticket* domain entity as input. The method first attempts to load the *TicketRecord* from the database or create a new one if it does not exist. The *recordFactory.loadOrCreate* method handles this operation.

Next, the *setRecord* utility method maps the properties from the domain *Ticket* entity to the jOOQ *TicketRecord* entity. After the mapping is complete, the *r.store* method is called to save the *TicketRecord* in the database.

Finally, the method updates the *Ticket* domain entity's id with the value from the *TicketRecord* entity and returns the updated *Ticket* domain entity. This allows the caller to access the persisted ticket data, which can be useful in some use cases.

```java
@Component
@RequiredArgsConstructor
public class StoreTicketGateway implements StoreTicketPort {

    1 usage
    private final RecordFactory<TicketRecord> recordFactory;

    @Override
    public Ticket store(Ticket ticket) {
        var r : TicketRecord = recordFactory.loadOrCreate(TicketRecord.class, ticket.getId());
        setRecord(ticket, r);
        r.store();
        ticket.setId(r.getId());
        return ticket;
    }
}
```

Figure 5: *StoreTicketGateway* code example

SIME script

The SIME script, as shown in Appendix IV, plays a critical role in the SOE application, ensuring backward compatibility with the existing SIME application for checkout. It is responsible for maintaining and updating the runners' file, which contains essential information about the participants. Written in Python, the script efficiently fetches the initial data and subsequently retrieves new data from specified sources at regular intervals. This continuous updating process ensures that the SIME application always has access to the most recent and accurate information about the runners, enhancing its efficiency and effectiveness.

The script consists of a main function called *main_cycle*, which executes the following steps:

1. Download the initial file: The script first downloads an initial file from the given URL (initial_file_url) and saves it with a specified name (initial_output_filename).
2. Download and append new data: The script enters an infinite loop where it periodically downloads new data from a specified URL (new_file_url). The loop runs every 5 minutes, ensuring the data in the output file remains up-to-date.
3. Process and merge the data: When the new data is successfully downloaded, it is temporarily stored in a buffer. The script then reads the content of the initial_output_filename and buffer, merging them together before writing the combined data to the output_filename.
4. Handle exceptions: In case of any errors while downloading the new data, the script logs an error message and continues the loop, attempting to download the new data again in the next iteration.

The SIME script is designed to ensure that the runners file for the SIME application stays current by continuously monitoring and updating the data. This automated process guarantees that the SIME application always has access to the most recent information about the runners, enabling it to operate efficiently and effectively.

Front-end

The React application is organized into three primary sections: components, pages, and shared parts. Pages use components to create the desired layouts for each web page, and components are designed to be reusable, promoting efficient code usage across the application. React hooks and state management are employed to handle data changes in the front-end. Material UI streamlines component design, while TypeScript is used to reduce JavaScript type-related errors. Security measures between front-end and back-end were implemented using JSON Web Tokens[12] (JWTs).

The *SessionHistory* component serves as an example in Fig. 6. This component accepts a list of registered people as a prop, named history, and returns a container containing a table. Each row of the table presents information about a different registration, ensuring a clear and organized display of user data in the front-end application.

[12] https://jwt.io/

```tsx
const SessionHistory: React.FC<SessionHistoryProps> = ({ history : RegisteredPerson[] }) => {
    const { t } = useTranslation();
    return (
        <div className={styles.container}>
            <Typography variant="h6">Session History</Typography>
            <TableContainer className={styles.tableContainer}>
                <Table>
                    <TableHead>
                        <TableRow>
                            <TableCell>{t( key: "history.si_card")}</TableCell>
                            <TableCell>{t( key: "history.eol_code")}</TableCell>
                            <TableCell>{t( key: "history.name")}</TableCell>
                            <TableCell>{t( key: "history.tickets_left")}</TableCell>
                        </TableRow>
                    </TableHead>
                    <TableBody>
                        {history.map((person : RegisteredPerson , index : number ) => (
                            <TableRow key={index}>
                                <TableCell>{person.siCard}</TableCell>
                                <TableCell>{person.eolCode}</TableCell>
                                <TableCell>{person.firstName + " " + person.lastName}</TableCell>
                                <TableCell>{person.multiuseTicket?.numberOfTickets}</TableCell>
                            </TableRow>
                        ))}
                    </TableBody>
                </Table>
            </TableContainer>
        </div>
    );
};
```

Figure 6: *SessionHistory* code example

In the code snippet, the *SessionHistory* component utilizes Material UI components such as *TableContainer*, *Table*, *TableHead*, *TableBody*, *TableRow*, and *TableCell* to create a visually appealing and functional table. The *useTranslation* hook is employed to access translations, facilitating internationalization.

Within the *TableBody*, the component iterates over the history array and creates a *TableRow* for each person object. The table cells display the person's SI card number, EOF code, full name, and the number of tickets left in their multi-use ticket (if applicable).

This example demonstrates how the React and Material UI libraries, along with TypeScript, can be used to develop a clean, efficient, and visually appealing front-end application that provides a user-friendly interface for the SOE system.

Web Serial API

Web Serial API[13] was used to connect the SI station with the web application via the serial port, enhancing the efficiency of the multi-use ticket-checking process in the SOE application. This integration is particularly beneficial for frequent multi-use ticket owners with their own SI-Cards.

When a user inserts their SI-Card into the SI station, the Web Serial API fetches their information and automatically fills the relevant fields in the web application. The user's multi-use tickets are also retrieved in the process. Once the event organizer verifies the data, they simply need to press enter to create a new ticket, allowing the next runner to proceed with registration.

For runners without a personal SI-Card, a rental SI-Card can be issued using the SI station. Inserting the rental card into the station retrieves only the SI-Card number, leaving the remaining fields empty, as rental SI-Cards are not assigned to specific individuals. The event organizer can then manually enter the person's EOF code or their first and last name into the corresponding fields. This process enables the SOE application to connect the person with the rental SI-Card and transfer the data to the SIME application using the previously mentioned Python script. As a result, the checkout process becomes more streamlined, facilitating a smooth user experience for both event organizers and participants.

Initially, the *sportident-react* package was intended for use in communication between the web application and the SI station, as it implements the Web Serial API and most of the necessary functionality. However, due to a lack of technical documentation and examples, some issues arose, making it unfeasible to use the package. As a result, the Web Serial API was used directly. Fortunately, the examples provided in the references document were sufficient, and with the knowledge gained from writing the prototype's communication with the SI station, the implementation was successful.

4.3.5 Unit testing

Unit testing is a crucial aspect of software development, ensuring that each component of the application works as expected. In this project, unit tests are employed to verify the correct functioning of all domain classes. Adhering to the single-responsibility principle allows each domain service class to be tested independently, simplifying the testing process. As new features are added, this approach usually requires the creation of new tests rather than

[13] https://developer.mozilla.org/en-US/docs/Web/API/Web_Serial_API

modifications to existing ones. The concise nature of the services results in smaller test cases, improving readability and making the validation process more manageable. Consequently, this methodology helps maintain the application's quality and reliability as it evolves.

ChatGPT by OpenAI was utilized to generate some of the unit test templates, which were then refined and supplemented with more domain logic when needed.

To create effective unit tests, JUnit[14] and Mockito[15] are employed to mock outbound requests to the database and other services to create effective unit tests. This ensures that tests remain lightweight and straightforward.

For instance, consider the *RegisterPersonActiveSiCardService* class depicted in Fig. 7. By mocking the dependencies of this class using Mockito, the test cases can focus on validating the service's logic without worrying about external factors such as database calls or other service interactions. This approach allows for the efficient and reliable testing of the application's domain classes and their respective functionalities.

```
@Override
public void execute(Request r) {
    var personId :Long = r.getPersonId();
    var siCardNumber :Long = r.getSiCardNumber();

    if (personId != null && siCardNumber != null && siCardNumber != 0) {
        var card :SiCard = loadSiCardPort.bySiCardNumber(siCardNumber)
                .orElseGet(() -> storeSiCardPort.store(SiCard.builder().siNumber(siCardNumber).build()));

        if (!loadSiCardPersonPort.isExistingActiveSiCardPerson(personId, card.getSiNumber())) {
            storeSiCardPersonPort.store(SiCardPerson.builder()
                    .siCard(card)
                    .isCurrentlyRented(true)
                    .isInitial(r.isInitial())
                    .personId(personId)
                    .build());
        }
    }
}
```

Figure 7: *RegisterPersonActiveSiCardService* code example

This class takes in three arguments within the request: personId, siCardNumber, and isInitial. Since isInitial is only used to pass a boolean from the request to the database, it will not be covered further in this discussion.

[14] https://junit.org/
[15] https://site.mockito.org/

The focus is on handling the *SiCard* entity and creating new relations with a person if necessary. If the *SiCard* entity is already present, it is retrieved from the database. If it is not present, a new card is created. Next, the class checks whether the current person already has a relationship with a *SiCard*. If they do not, a new relationship is created. Otherwise, the flow is finished.

Four simple tests can be written to cover all the functionality of the *RegisterPersonActiveSiCardService*, as illustrated in Fig. 8. These tests aim to validate the various aspects of the class, ensuring it operates correctly under different scenarios.

```
@Test
public void test_createNewSiAndActiveSiPersonRelation_ifNotExisting() {...}

@Test
public void test_createNewActiveSiPersonRelationCreated_ifSiExists() {...}

@Test
public void test_verifyNewActiveSiPersonRelationNotCreated_ifAlreadyExists() {...}

@Test
public void test_doNothing_ifSiCardNumberIsNotProvided() {...}
```

Figure 8: RegisterPersonActiveSiCardService unit test code example

Using descriptive names for the tests is crucial, as it simplifies understanding what has failed if tests stop working in the future. Descriptive test names provide clear insights into the specific functionality being tested, making it easier to pinpoint issues and fix them promptly. This practice contributes to a more effective testing strategy and overall better code quality.

4.3.6 Result screenshots

As mentioned in section 4.1.2, three primary pages were created due to functional requirements: login page, registration page, and administration page. To enhance the user experience, a start page was also incorporated, as depicted in Appendix V, which is displayed when a user logs into the application. A landing page preceding the login page is not within the scope of this application, as communication with clubs and training occurs externally to the application.

The login page contains only username and password fields, as shown in the Fig. 9. Users for the application are currently added directly to the database, as there are not too many users, and clients do not need to or should not create users themselves.

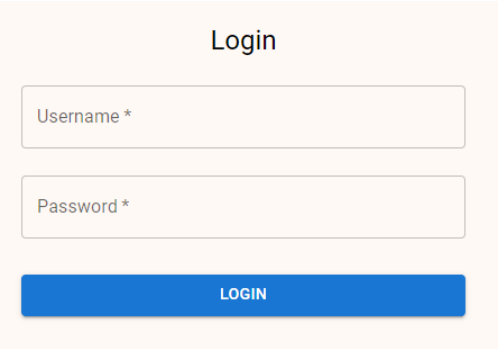


Figure 9: Login component

The registration page, as depicted in Appendix VI, functions as the main page for the application, where users can register all competitors participating in the EOE. Additionally, users can search for a person's EOF code using their name. Competitors can be registered with a multi-use ticket if they possess one, or a new ticket can be added if purchased. Moreover, users can create a new association between a (rental) SI-Card and a person and review previously registered individuals in case of any issues.

The registration page also features a second tab, shown in Appendix VII, where all relevant multi-use tickets can be examined in case of any problems or concerns. This functionality is included here to enable users to swiftly modify and verify information from the table during the registration process, ensuring a smooth and efficient experience for both event organizers and competitors.

The administration page, as depicted in Appendix VIII, serves as a platform to view or add event series, events, and club rental SI-Cards, as well as to facilitate the addition of multi-use tickets. The events section displays ticket usage statistics for each event, providing valuable insights. A button is available for updating persons from the EOF database, allowing for a quick and efficient download of new person data and information from the central database. This feature ensures that the administration page is a comprehensive and user-friendly hub for managing all aspects of orienteering event organization.

4.4 Deploying the web application

Following the methodology described in earlier sections, the integrated system was successfully deployed to a production environment. This process involved setting up the necessary software infrastructure, configuring the system, and making it accessible to users,

while also implementing appropriate security measures to protect both the system and user data.

For the deployment of the web application, Docker was employed to create containerized images of the application, ensuring consistent execution and simplified deployment. Docker Compose[16] was used to manage the multi-container Docker application.

Utilizing Docker images and Docker Compose facilitated the deployment process, making managing the application's lifecycle more straightforward and allowing for seamless updates and maintenance. As a result, the production environment can be established with just a single command.

4.5 Web application testing

The web application was tested in a real-life event on April 12, 2023, in collaboration with Seiklushunt during the Ujula evening orienteering event in Tartu. The testing process involved several steps, including setting up all related database relations, configuring the SIME side and client registration computers, and instructing users on navigating the application.

The real-life testing provided valuable insights into the application's performance under actual event conditions, allowing for the identification and resolution of potential issues. This process contributed to the refinement and optimization of the application, ensuring a seamless and efficient user experience.

During the testing phase, some critical issues were discovered:

1. Connection Security: The application ran locally, and when connected using a local IP, the connection was not secure (i.e., it did not use HTTPS). This limitation posed a problem when using two computers simultaneously with SI stations, as only one could have a localhost connection. While this issue did not prevent the application from functioning without the SI station communication, it slowed down the registration process, requiring event organizers to enter all the SI numbers.
2. Front-end Limitations: The front-end limitations for checking first and last names were too broad, resulting in some empty-named persons being added to the person

[16] https://docs.docker.com/compose/

table. These empty-named individuals were also transferred to the SIME file and required manual entry when the person approached the checkout.

Despite these issues, the testing process offered a valuable opportunity to identify areas for improvement and further optimize the application. The insights gained from this real-life testing scenario will help enhance the overall user experience and ensure the system is better prepared for future events.

5. Discussion

5.1 Technical challenges

Technical challenges emerged throughout the development of the orienteering event management application. These challenges needed to be addressed to ensure the application could function effectively in various real-life scenarios.

The first challenge was to choose correctly which parts of the program should go as back-end and which should go as front-end as a big part of the data should be ready for display. However, holding the whole dataset in the cache is not practical. The current design assumes that the back-end is always accessible, and network issues will not occur. In real-life scenarios, this may not be the case, and redundancy measures may need to be considered.

The second challenge was modularity. The program needed to be modular enough to facilitate the quick addition of new features. The hexagonal architecture used in the application's design greatly contributed to addressing this challenge, as it inherently promotes modularity.

The third challenge is that multiple orienteering clubs must be able to use the new application simultaneously. This means that problems like security, different user data in one database, and user authentication are also relevant. Security measures were implemented using JWTs, and a club-based approach was employed for data management. All requests are protected by authentication, and user permissions for accessing data are carefully verified.

Ensuring compatibility with the existing SIME system was essential to provide a seamless experience for event organizers. This compatibility required the automatic updating of SI-Card numbers and owner information in the SIME system. A SIME script was developed to ensure that the checkout computer consistently had access to the most recent data.

5.2 Suggestions for further improvements

While the developed application has successfully met its initial objectives and demonstrated its effectiveness in real-world scenarios, there is always room for improvement and expansion. This section will explore potential enhancements and new features that could be introduced to elevate the application's capabilities.

5.2.1 Planned features

Several features are planned to be completed before offering the application to potential clients, but they were not finished in time for the thesis. These features aim to enhance the functionality and ease of use of the SOE application.

1. Integration with Stebby:

   Stebby is a ticket-selling platform, and it is essential to explore potential collaboration opportunities with the Stebby team. This partnership would ideally involve creating or using an API interface that would enable seamless communication between the SOE and Stebby systems. The integration would allow the automatic transfer of information about newly purchased one-time or multi-use tickets to the SOE application, eliminating the need for manual updates.

   Benefits of integrating with Stebby include:

   - Automated ticket data synchronization between Stebby and SOE systems
   - Reduced manual workload for event organizers and club administrators
   - Improved data accuracy and real-time updates on ticket information
   - Enhanced user experience for participants purchasing tickets through Stebby

   The integration with Stebby is an essential feature to implement, as it will greatly simplify the ticket management process and ensure a seamless and efficient registration experience for both event organizers and participants. It wasn't completed within the scope of this thesis because integrating with external systems can be time-consuming and potentially present numerous issues.

2. Moving to the cloud:

   As previously mentioned, the application is currently running in a Kubernetes cluster but has not yet been deployed to the cloud. Deploying the application to the cloud is essential because it enables even non-tech-savvy users to utilize the application for their events without needing to run it in a local environment.

   There may still be use cases where a local environment is required, such as deep-forest locations with no internet connection. However, these instances are relatively rare. By deploying the application to the cloud, it will become more accessible and user-friendly for a wider range of users, including event organizers and club administrators who may not possess technical expertise.

5.2.2 Additional features

While the application successfully addresses the requirements outlined in the thesis, there are additional features that could be implemented to further enhance its usability and appeal to clubs for weekly use.

1. EOF code creation:

   Although there are relatively few competitors each week who do not have an EOF code, simplifying the process of creating new EOF codes would facilitate the management of larger events. Integrating this feature within the application would streamline the registration process and improve overall efficiency.

2. Handling competitors without EOF codes using Stebby for multi-use or one-time tickets:

   Some competitors might not have an EOF code or may prefer not to use one but still want to purchase multi-use or one-time tickets through Stebby. Currently, these individuals must be managed manually due to challenges with the application's functionality. To address this, the application could include:

   - logic for connecting a person with and without an EOF code, ensuring proper allocation of tickets to the correct individual,
   - a mechanism to transfer multi-use tickets from one person to another when necessary,
   - enhanced security measures, as relying solely on names for ticket usage may not be secure enough, given that names are not unique identifiers.

3. Landing page for application:

   While the number of clubs using SOE as their EOE registration manager may not be substantial, creating a landing page for the application could provide multiple benefits. The landing page could serve as a point of reference for users to verify that the application is working and operational. Moreover, it could function as an informative space to showcase the application's features, benefits, and potential for growth.

   Having a project overview readily available would also be helpful when seeking to engage potential collaborators, clients, or employees. Providing a clear, concise, and

visually appealing summary of the project on the landing page would generate interest and facilitate a better understanding of the application's capabilities.

4. Enhanced Reporting and Analytics:

   As clubs and event organizers begin to use SOE for their EOE registration management, it is important to provide them with enhanced reporting and analytics features. These features would allow users to gain insights into event participation, competitor performance, and other relevant metrics, helping them make informed decisions for future events.

5. Company Agreement with Clubs:

   Before implementing the company agreement feature in the SOE application, it is essential to gather feedback from potential clients, including clubs and event organizers. While Seiklushunt may not find this feature particularly useful, other clubs may have different needs.

   A company agreement feature would enable clubs to arrange payment for their employees' participation in EOE events through a centralized system. This could streamline the registration and payment process for both the clubs and the participants. Features that could be incorporated into a company agreement system include:

   - bulk registration of employees for events,
   - centralized invoicing and payment management,
   - customizable payment rules and schedules,
   - reporting and analytics specific to company-sponsored participants.

   By understanding the needs and preferences of potential clients, the SOE application can incorporate features that cater to a broader range of clubs, ultimately expanding its user base and increasing its value as an EOE registration management tool.

5.2.3 Maintain and update the web application

After deployment, it is crucial to consistently monitor, maintain, and update the web application to tackle any bugs or performance issues that may emerge. Updating the application with new features and improvements based on user feedback and changing needs will guarantee its long-term success and relevance. Ongoing maintenance and adaptation will

enhance the user experience and keep the application in line with advancements in technology and industry best practices. The web application will continue to serve as an invaluable tool for efficient event management by staying responsive to the evolving landscape of orienteering events and user expectations.

6. Conclusion

This thesis aimed to develop a modern, efficient, and user-friendly web application for orienteering event management that streamlines registration and ticketing processes while ensuring compatibility with existing systems. The project was driven by the difficulties and limitations encountered by orienteering clubs utilizing the current SIE software, which served as a foundation for the new application.

Throughout the thesis, various software development approaches were examined to build a web application based on the initial prototype. The hexagonal architecture was selected due to its ability to segregate responsibilities and minimize dependencies, fostering modularity and flexibility. The developed application employed Java for the back-end, React for the front-end, and Postgres as the database system.

The application's features were designed in close collaboration with the orienteering event organizer Seiklushunt, prioritizing user requirements and delivering a dependable and efficient system. The testing phase, carried out during an actual orienteering event, yielded valuable insights into the application's performance, leading to numerous refinements and optimizations.

Although several technical challenges were encountered, such as data management, modularity, multi-club usage, and backward compatibility, the application successfully addressed these concerns, resulting in a resilient and adaptable system that caters to the needs of modern orienteering events. The thesis also discusses potential avenues for further development of the completed software solution, which could eventually be offered as a service to other orienteering clubs.

References

[1] Eesti Orienteerumisliit, Orienteerumise ABC.

[2] Eesti Orienteerumisliit, Kalender. https://orienteerumine.ee/kalender/kuu-kalendervaade/ (accessed April 20, 2023).

[3] RMK Orienteerumispäevakud, Kalender. https://paevakud.ee/orienteerumispaevakute-kalender/ (accessed April 20, 2023).

[4] Eesti Orienteerumisliit, EOL majandusaruanne 2022, (2023).

[5] EMIT ECard system. https://www.emit-uk.com/emit-ecard-system/ (accessed September 12, 2022).

[6] SPORTident GmbH Germany, About Us. https://www.sportident.com/about-us.html (accessed September 12, 2022).

[7] T. Klaar, SPORTident Mini Events Manual, (2013). https://www.tak-soft.com/dokuwiki/doku.php?id=osport:sime (accessed April 20, 2023).

[8] Estonian Tax and Customs Board, Health and Sport Expenses. https://www.emta.ee/en/business-client/taxes-and-payment/income-and-social-taxes/fringe-benefits#health-and-sport-exp (accessed April 17, 2023).

[9] Stebby, About Us, (n.d.). https://stebby.eu/about-us/ (accessed April 17, 2023).

[10] Tak-Soft, SI-kaartide hinnad. https://www.tak-soft.com/sportident/kaart/tellimine/index.php (accessed April 15, 2023).

[11] University of Tartu, Student Project Results: May 2021 winners, (2021). https://courses.cs.ut.ee/t/student_projects_results/Main/2021-05 (accessed April 28, 2023).

[12] M. Spirich, Is Java still relevant in 2023, (2023). https://www.axon.dev/blog/is-java-still-relevant-in-2022 (accessed April 17, 2023).

[13] S. Bos, Java Basics: What Is Spring Boot?, (2020). https://www.jrebel.com/blog/what-is-spring-boot (accessed April 17, 2023).

[14] jOOQ, Home page. https://www.jooq.org/ (accessed April 17, 2023).

[15] HubSpot, What is React.js?, (2022). https://blog.hubspot.com/website/react-js (accessed April 17, 2023).

[16] Simpli Learn, The Best Guide to Know What Is React, (2023). https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs (accessed April 17, 2023).

[17] Material UI - Overview. https://mui.com/material-ui/getting-started/overview/

(accessed April 17, 2023).

[18] About Postgresql. https://www.postgresql.org/about/ (accessed April 17, 2023).

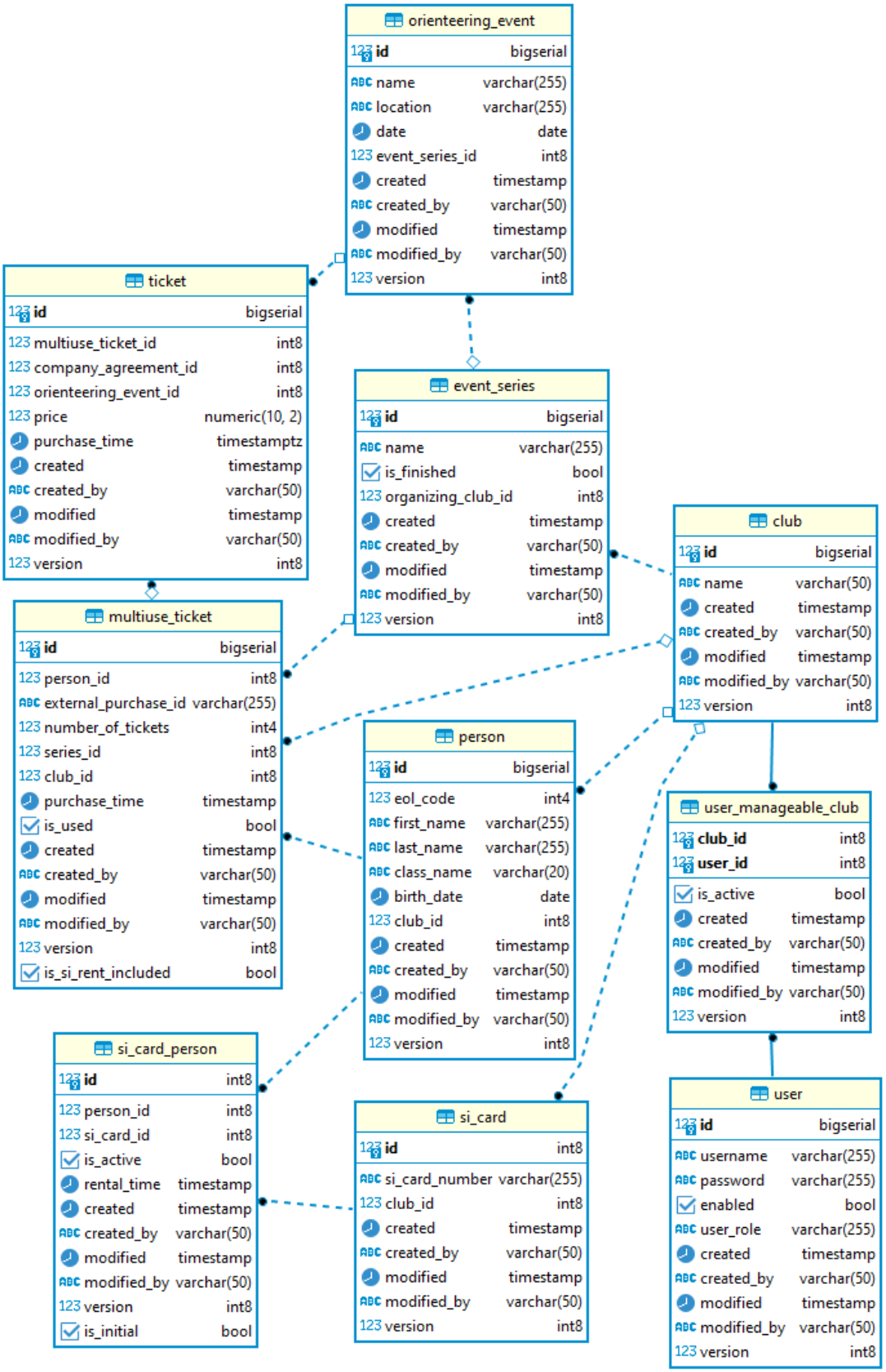
[19] T. Hombergs, Get Your Hands Dirty on Clean Architecture: A Hands-On Guide to Creating Clean Web Applications with Examples in Java, 1st ed., Packt Publishing Ltd, 2019.

Appendix

I. Source code

The source code and associated files for this project can be found in a private repository at https://github.com/Kustu11/SOE. To gain access to the source code, please contact the author via email at kynnapas.kustu@gmail.com.

II. Database design



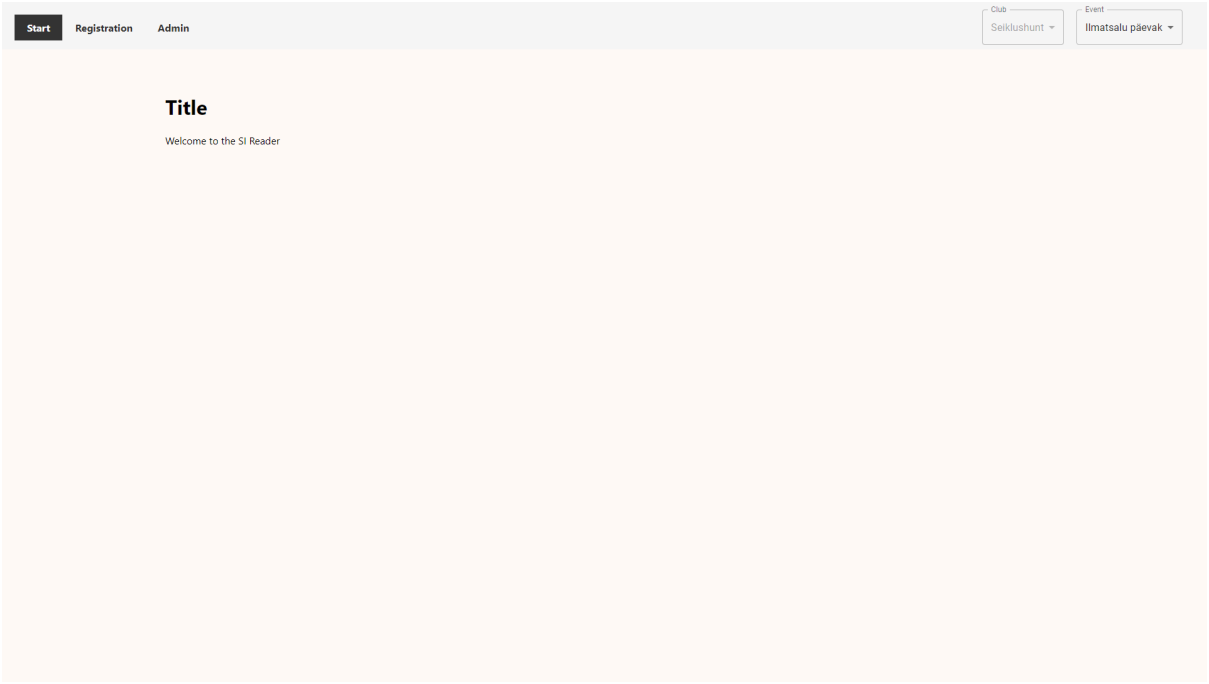
III. *createTicket* method

```
@PostMapping("/tickets/multiuse")
@Operation(summary = "Create a multi-use ticket",
        description = "Creates a multi-use ticket for the specified event and person.")
@ApiResponse(responseCode = "200", description = "Ticket created")
@ApiResponse(responseCode = "400", description = "Multi-use ticket not found or not enough tickets left",
        content = @Content(schema = @Schema(implementation = String.class)))
ResponseEntity<String> createTicket(@RequestBody TicketDto dto){

    AuthUtil.isClubAccessAllowed(dto.getClubId());

    Person person = loadPersonPort.byPersonId(dto.getPersonId())
            .orElseThrow(RequestNotAllowedException::new);

    if (!dto.isPersonValidAndUnedited(person)) {
        return ResponseEntity.badRequest().body("Person is invalid or edited");
    }

    var multiuseTicket : MultiuseTicket = loadMultiUseTicketPort.byId(dto.getMultiuseTicket().getId())
            .orElseThrow(RequestNotAllowedException::new);

    if (!person.equals(multiuseTicket.getPerson())) {
        throw new RequestNotAllowedException();
    }

    var event : OrienteeringEvent = loadEventPort.byId(dto.getEventId())
            .orElseThrow(RequestNotAllowedException::new);

    var eventSeries : OrienteeringEventsSeries = loadEventSeriesPort.byId(event.getSeriesId())
            .orElseThrow(RequestNotAllowedException::new);

    if (isInvalidMultiuseTicket(dto, multiuseTicket, eventSeries, dto.getClubId())) {
        return ResponseEntity.badRequest().body("Multiuse ticket not found or not enough tickets left");
    }

    registerMultiuseTicketUseCase.execute(RegisterMultiuseTicketUseCase.Request.from(
            person,
            multiuseTicket,
            event
    ));
```

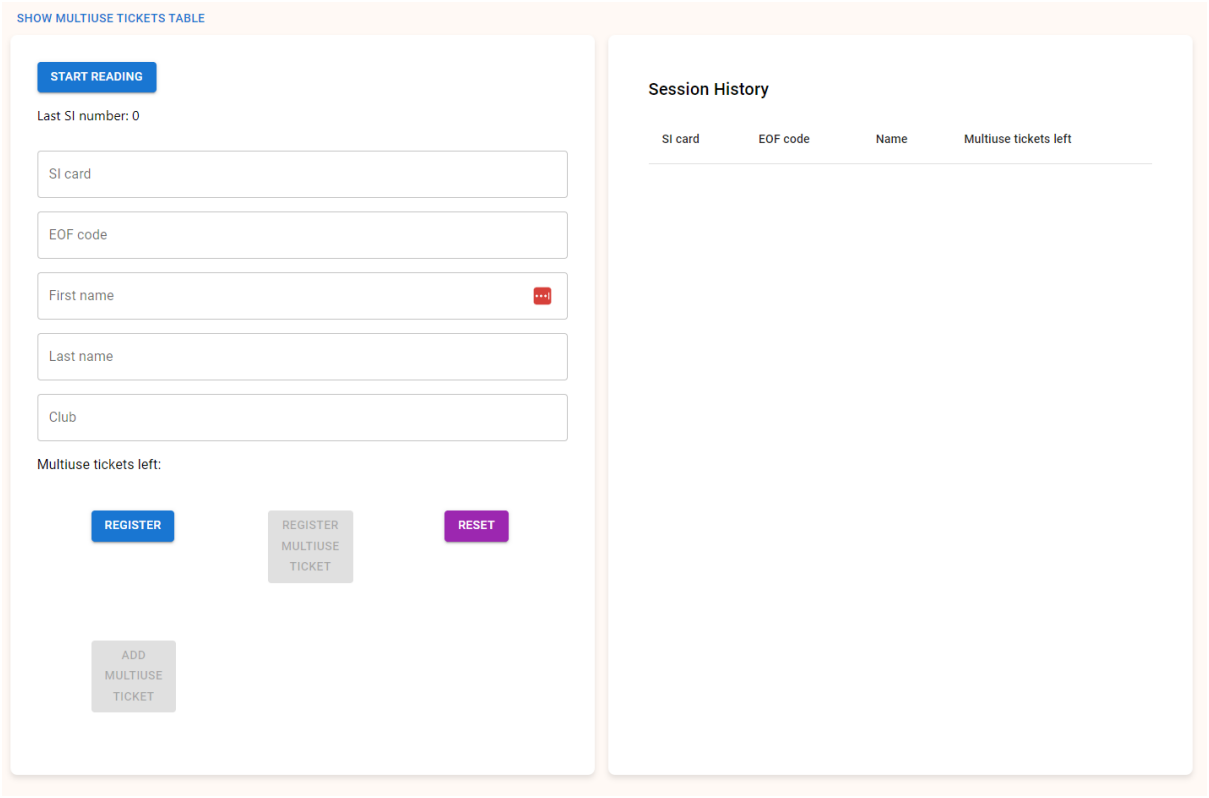
IV. SIME script

```
def main_cycle():

    # Download the initial file
    download_file(initial_file_url, initial_output_filename)

    # Copy the initial file content to the runners.txt file
    with open(initial_output_filename, 'r', encoding='cp1252', errors='replace') as initial_file:
        initial_content = initial_file.read()

    with open(output_filename, 'w', encoding='cp1252', errors='replace') as output_file:
        output_file.write(initial_content)

    # Download and append the new file every 5 minutes
    while True:
        info('Downloading new data...')
        # Download new data to a temporary buffer
        buffer = StringIO()
        try:
            response = get(new_file_url)

            if response.status_code == 200:
                buffer.write(response.text)

                # Prepend the buffer content to the eolrunners.txt content and write it to runners.txt
                with open(initial_output_filename, 'r', encoding='cp1252', errors='replace') as initial_file, \
                        open(output_filename, 'w', encoding='cp1252', errors='replace') as output_file:
                    buffer.seek(0)  # Move the buffer pointer to the beginning
                    new_data = buffer.read()
                    output_file.write(new_data)
                    output_file.write(initial_file.read())
                info('Done!')
            else:
                info('Error downloading new data')
        except:
            info('Error downloading new data')

        sleep(5 * 60)  # 5 minutes in seconds
```

V. Start page


Start
Registration
Admin
Club
Seiklushunt
Event
Ilmatsalu päevak
Title
Welcome to the SI Reader

VI. Registration components



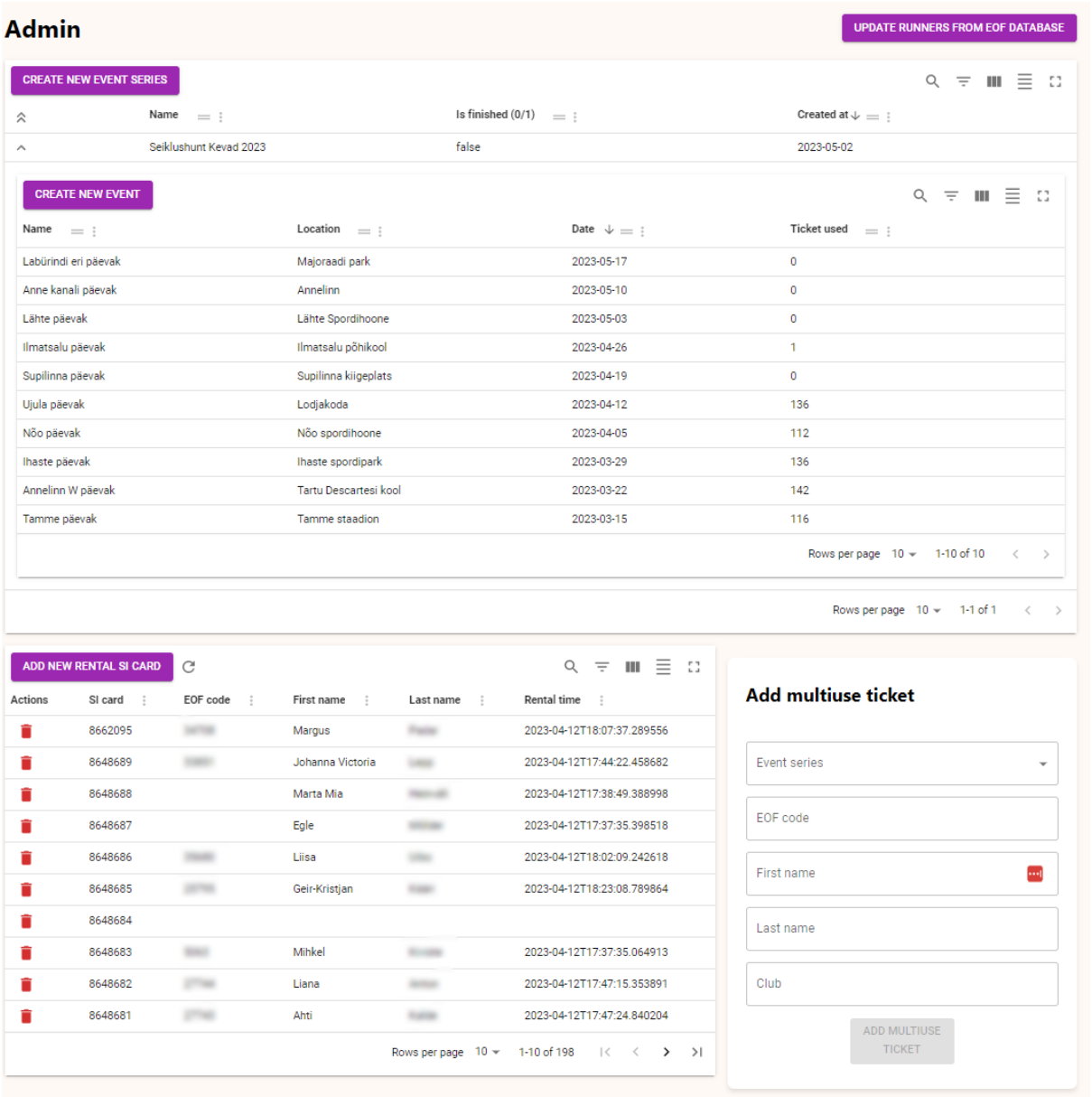
VII. Multi-use ticket table

BACK TO REGISTRATION

EOF code	First name	Last name	SI included	Number of tickets	Tickets left	2023-04-26	2023-05-03	2023-05-10	202
[illegible]	Tiiu	[illegible]	No	5	4	-	-	-	-
[illegible]	Udo	[illegible]	No	10	5	-	-	-	-
[illegible]	Marika	[illegible]	Yes	5	2	-	-	-	-
[illegible]	Jüri	[illegible]	No	10	7	-	-	-	-
[illegible]	Maie	[illegible]	No	5	1	-	-	-	-
[illegible]	Ülle	[illegible]	No	5	1	-	-	-	-
[illegible]	Tõnu	[illegible]	No	5	2	-	-	-	-
[illegible]	Riina	[illegible]	No	5	1	-	-	-	-
[illegible]	Priit	[illegible]	No	5	2	-	-	-	-
[illegible]	Merike	[illegible]	No	5	2	-	-	-	-

Rows per page 10 1-10 of 238

VIII. Admin components



License

Non-exclusive license to reproduce the thesis and make the thesis public

I, Kustu Künnapas,

1. grant the University of Tartu a free permit (non-exclusive license) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

   Orienteering Event Registration Software: From Prototype To Web Application,
   supervised by Dietmar Pfahl,

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kustu Künnapas
08/05/2023