OCTOBER, 2022

evento

Software Arhitecture Document

For the individual project
of Semester 3, ICT
Software Engineering

Created by
Iulia Toderașcu

Class
S3-CB02

Table of Contents

1. Introduction

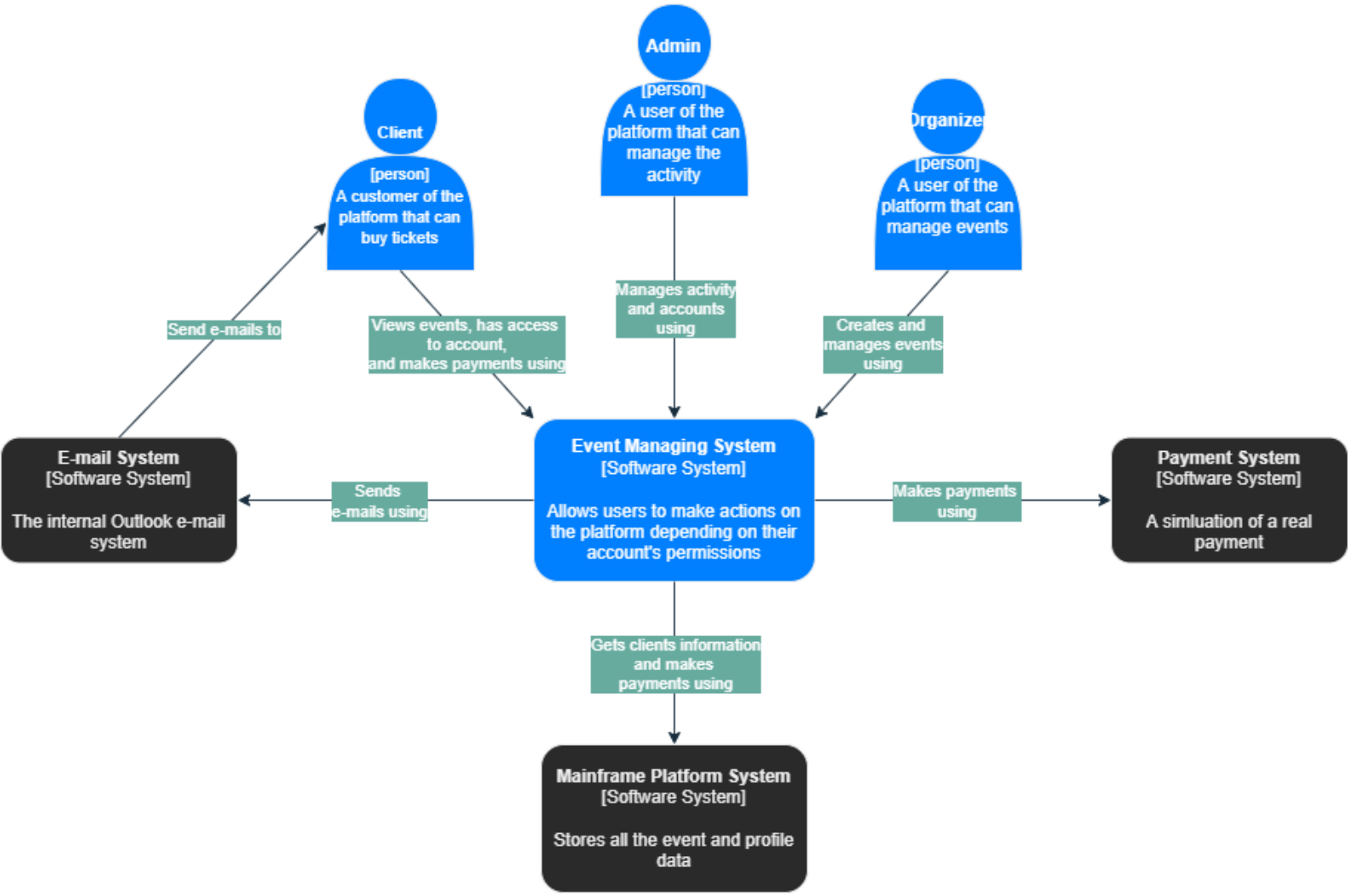
The online market is constantly adapting to people's needs. One of their need is to purchase easily tickets to the desired events. The traditional tickets are almost fully replaced now with the digital ones, making selling and managing them much more easier.

From a digitalized system both organizers and clients can benefit in matters of speed, usability and efficiency.

However, behind the scenes, you must build customer loyalty and attract attention.

2. System context

(C1)



System context

(C1)

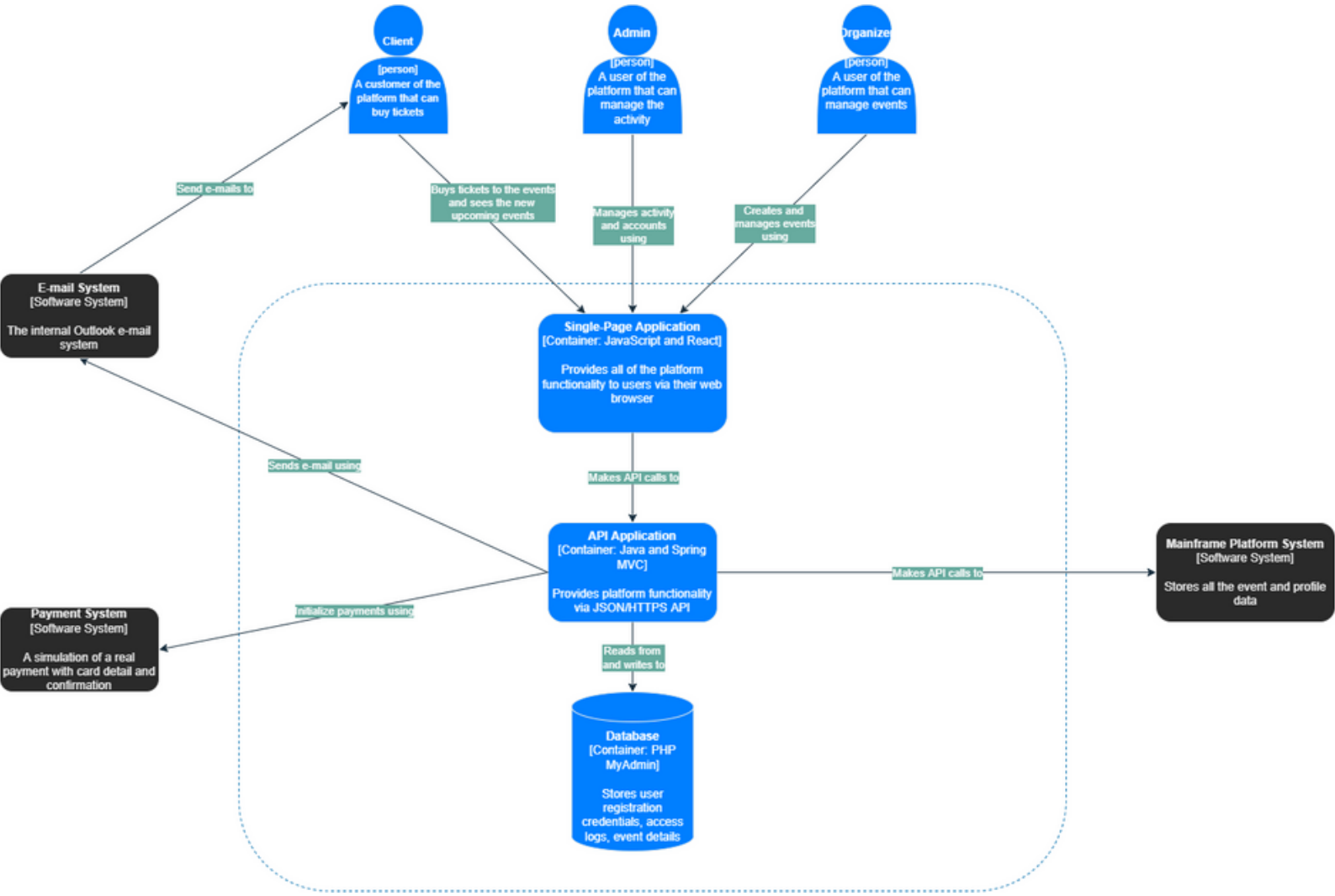
- there will be **3 types of users,** clients, admins and organizers, each account type having its own permissions
- the "Event managing system" represents the part of the application that is responsible for making all the actions of the app available to the users
- the "E-mail System" it's an external system that will help to send e-mails to the clients with the ticket information
- the "Payment System" it's also an external system that will allow the users to simulate the payment for a ticket
- the "Mainframe Platform System" is responsible for storing all client and event data

DECISIONS

1. I will use external systems for the e-mail and payment because the application will have an easy flow and, this way, I can practice API integration
2. I choose to have three types of users because I want to cover all the functionalities I intend to implement in a logical way
3. The data about the clients and events are stored externally for security reasons

3. Containers and technology choices

(C2)



Containers and technology choices

(C2)

- my project will contain two applications: **one for the front end, and one for the back end**
- the Single-Page application represents the front-end and will be built in the IntelliJ environment, using the React library
- the API application represents the back-end and will be built also in the IntelliJ environment

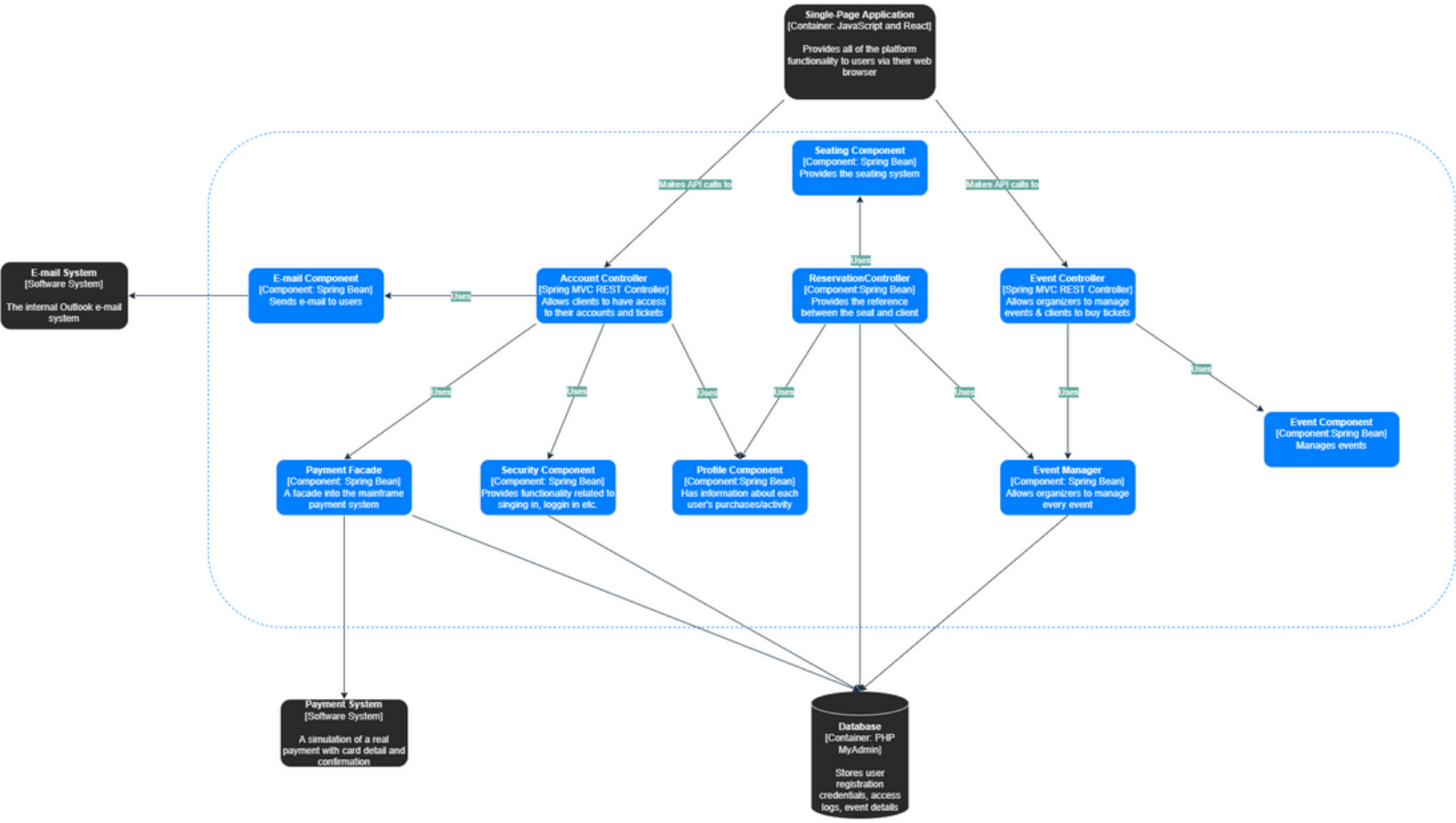
DECISIONS

1. I will use external systems for the e-mail and payment because the application will have an easy flow and, this way, I can practice API integration
2. I choose to have three types of users because I want to cover all the functionalities I intend to implement in a logical way
3. The data about the clients and events are stored externally for security reasons

*Note: the chosen APIs for the e-mail service and payment will be described in the Research Document

4. COMPONENTS

(C3)



COMPONENTS

(C3)

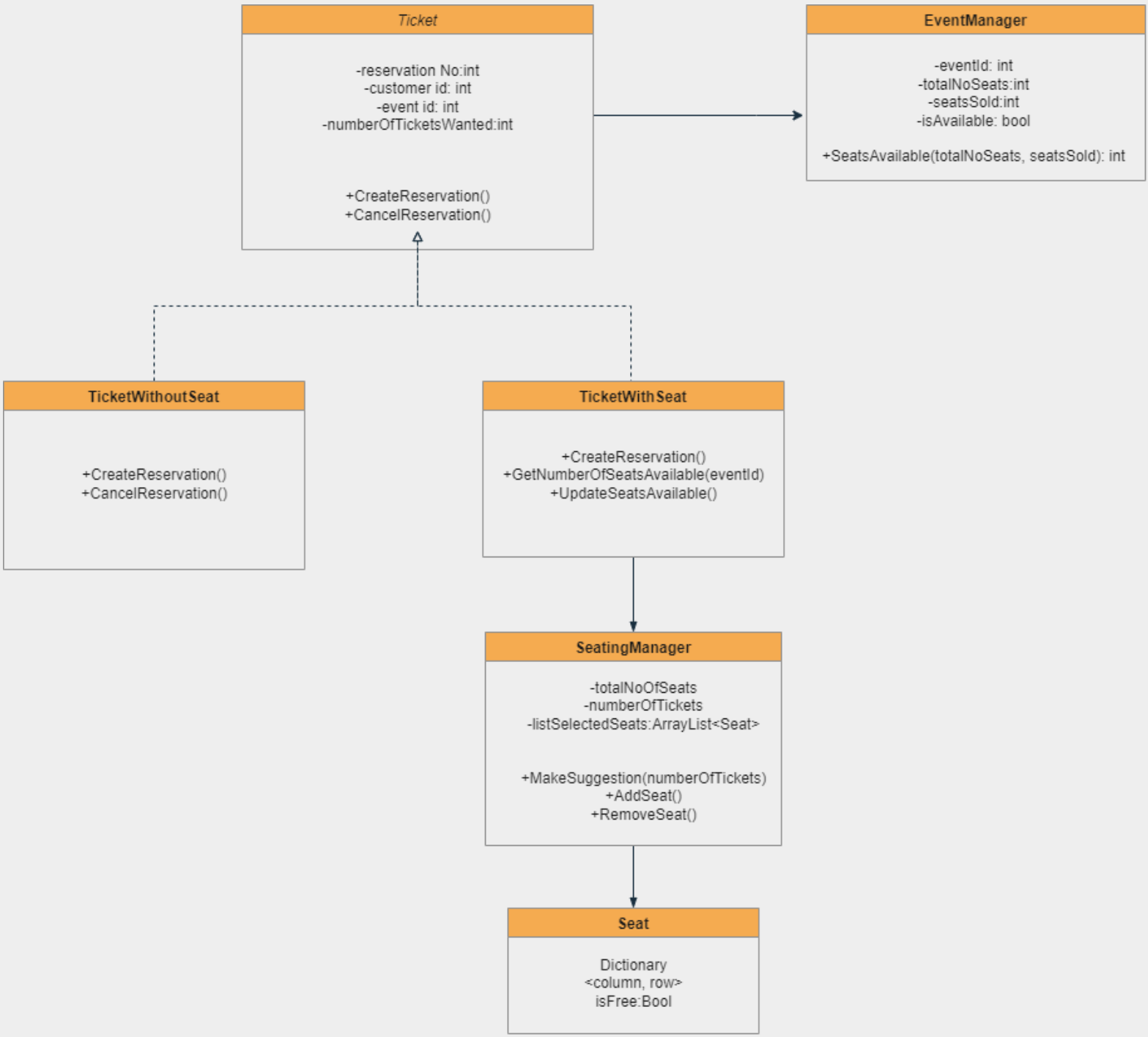
- Zooming on the API Application, I'm planning to have three controllers: one for the **accounts**, one for the **events,** and one for the **reservations**.
- Each controller has multiple components that are also sometimes connected to each other

DECISIONS

1. I choose to divide these controllers in order to manage efficiently all the data
2. Because the events are always changing state(either new ones will be added, some will be deleted or some will be sold out/cancelled) I wanted to think about a system which allows easy interaction for the clients

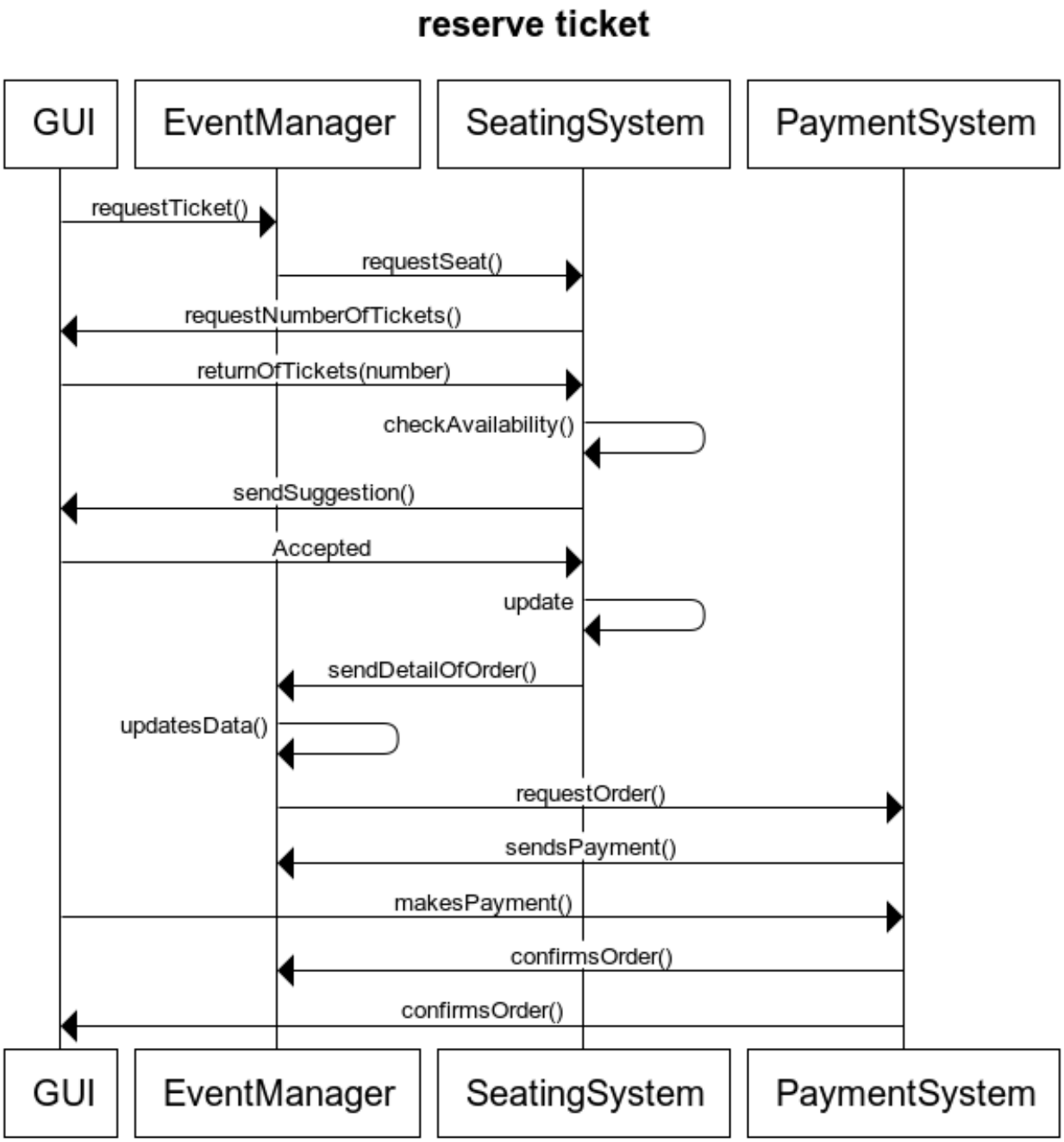
5. CLASS DIAGRAMS AND SEQUENCE DIAGRAMS

(C4)



5. CLASS DIAGRAMS AND SEQUENCE DIAGRAMS

(C4)



CLASS DIAGRAMS AND SEQUENCE DIAGRAMS

(C4)

- The reservation controller contains the seating component which is responsible for managing the sales of the tickets
- The entire process starts with a ticket which can be either an entry ticket (without a seat) or a ticket with a seat. Both types of tickets will need to use data provided by other components about the client and the event in order to update the number of tickets available

DECISIONS

I choose the **manager/service approach** because in my project it is best if I keep the maintainability of the architecture. Although a few classes will be indeed significantly more complex (like the one responsible for the seating suggestion based on the number of tickets wanted by the client), I can avoid over-engineering my system or creating helper classes.

6.PERSISTANCE PER COMPONENT

Link to the diagram

7.SPECIFICATION OF INTERFACES

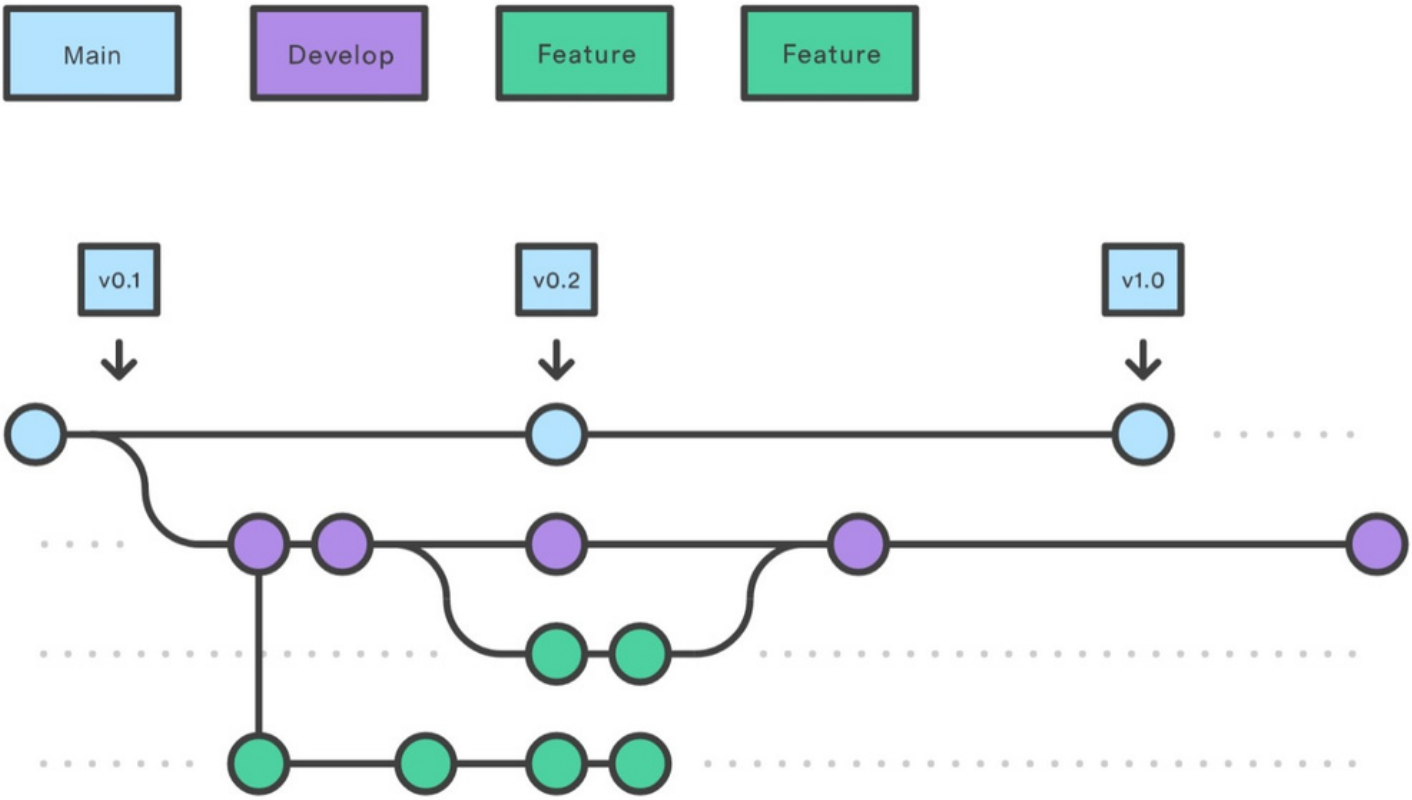
VERB	RESOURCE	ACTION
GET	/users	Retrieve list of users
GET	/events	Retrieve list of events
POST	/createEvent	Create an event
GET	/login	User sing in into the platform
POST	/signup	User register into the platform
GET	/event/{id}	Navigate to a specific event page

8.GIT BRANCHES

Gitflow is an alternative branching model that involves multiple primary branches and feature branches.

Rather than assigning different branches specific roles, it specifies how and when they should interact.

I want to use this branching model because it offers an organized working environment and it is more professional.



9.DECISIONS

- Why WebApp?

1. End-users and businesses both benefit from web applications
2. Updates are applied centrally to web applications, so they are always up-to-date
3. It eliminates compatibility issues since everyone has access to the same version
4. Any web browser can be used to access web applications

- Why Java?

1. Portability
2. Object-oriented
3. Secure
4. Easier network connections
5. Allows multithreading

- Why Springboot?

1. Uses Dependency Injection Pattern
2. Saves memory space
3. Can rely on Spring MVC to handle requests from HTTP servers

- Why React?

1. Developers are authorized to reuse ReactJS components and create new applications using them
2. Code is always written using a mix of JavaScript and HTML syntax, which simplifies the entire process of creating a project

Questions? Contact me.

Student e-mail: 475741@student.fontys.nl

Personal e-mail: toderacuiulia@gmail.com

Phone number: +40757655970