Design Document

Individual Track Assignment

25/03/2022 Eindhoven
Version: 5.0

Esther Wolfs: 3329984

Tutor:
Tim Kurvers
Márcio Paixão Dantas

Version history

Version	Date	Author(s)	Changes	State
0.1	25/03/2022	Esther Wolfs	Start design document and research questions	Started
0.2	30/03/2022	Esther Wolfs	Make changes after receiving feedback	Started
0.3	05/04/2022	Esther Wolfs	Update research questions	Finished
1.0	16/05/2022	Esther Wolfs	Answer research questions	Started
1.1	16/05/2022	Esther Wolfs	Move everything research related from this document to specific applied research document	Finished
2.0	02/06/2022	Esther Wolfs	Add CICD Diagram with explanation	Started
2.1	13/06/2022	Esther Wolfs	Update C4 Diagram	Finished
3.0	13/06/2022	Esther Wolfs	Add security report structure	Started
3.1	14/06/2022	Esther Wolfs	Fill in paragraphs analysing the security risks.	Finished
4.0	16/06/2022	Esther Wolfs	Remove security report from this document	Finished
5.0	17/06/2022	Esther Wolfs	Add CD pipeline to diagram	Finished

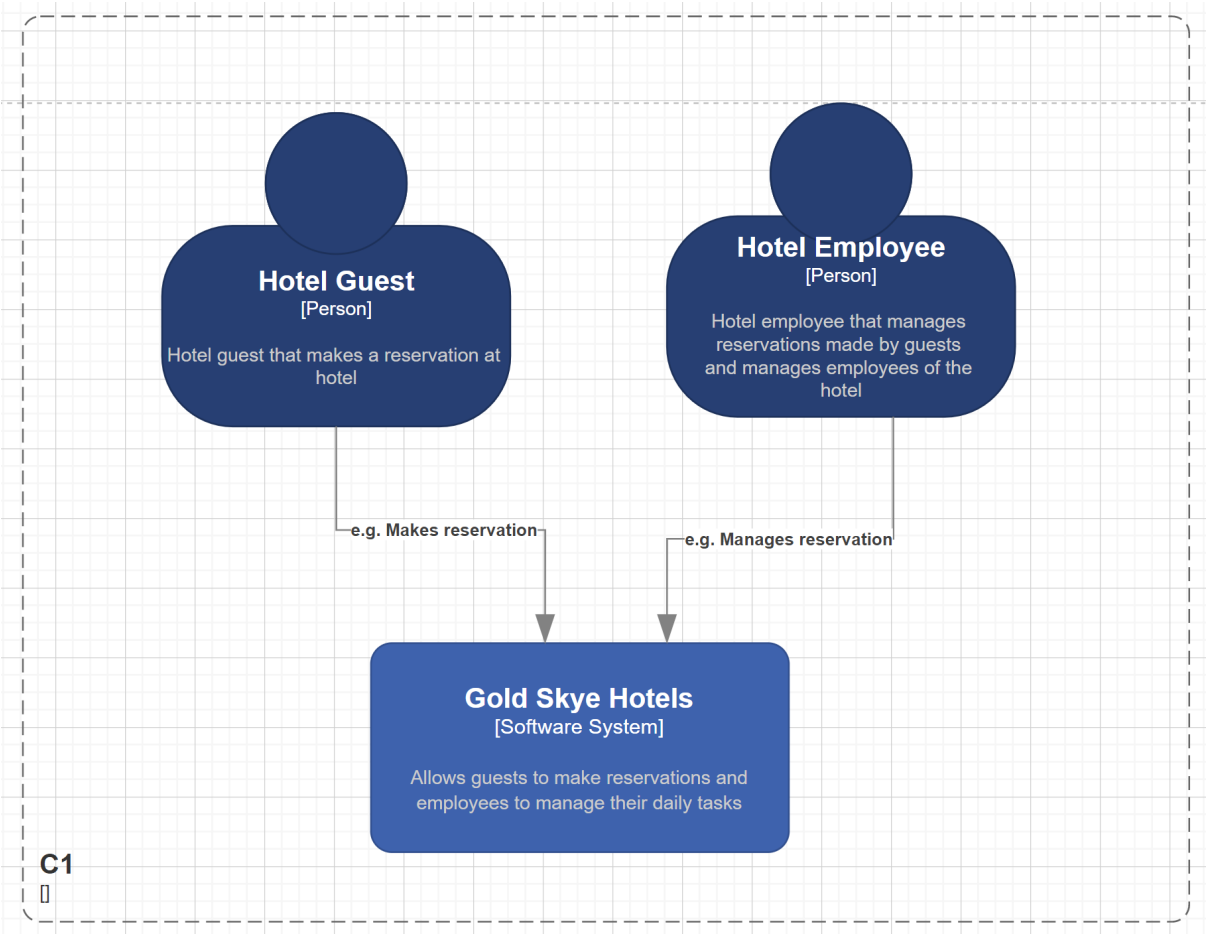
1. Introduction

In this document I will explain the decisions I have made for my project based on the research I have done. I will explain the architecture of my application.

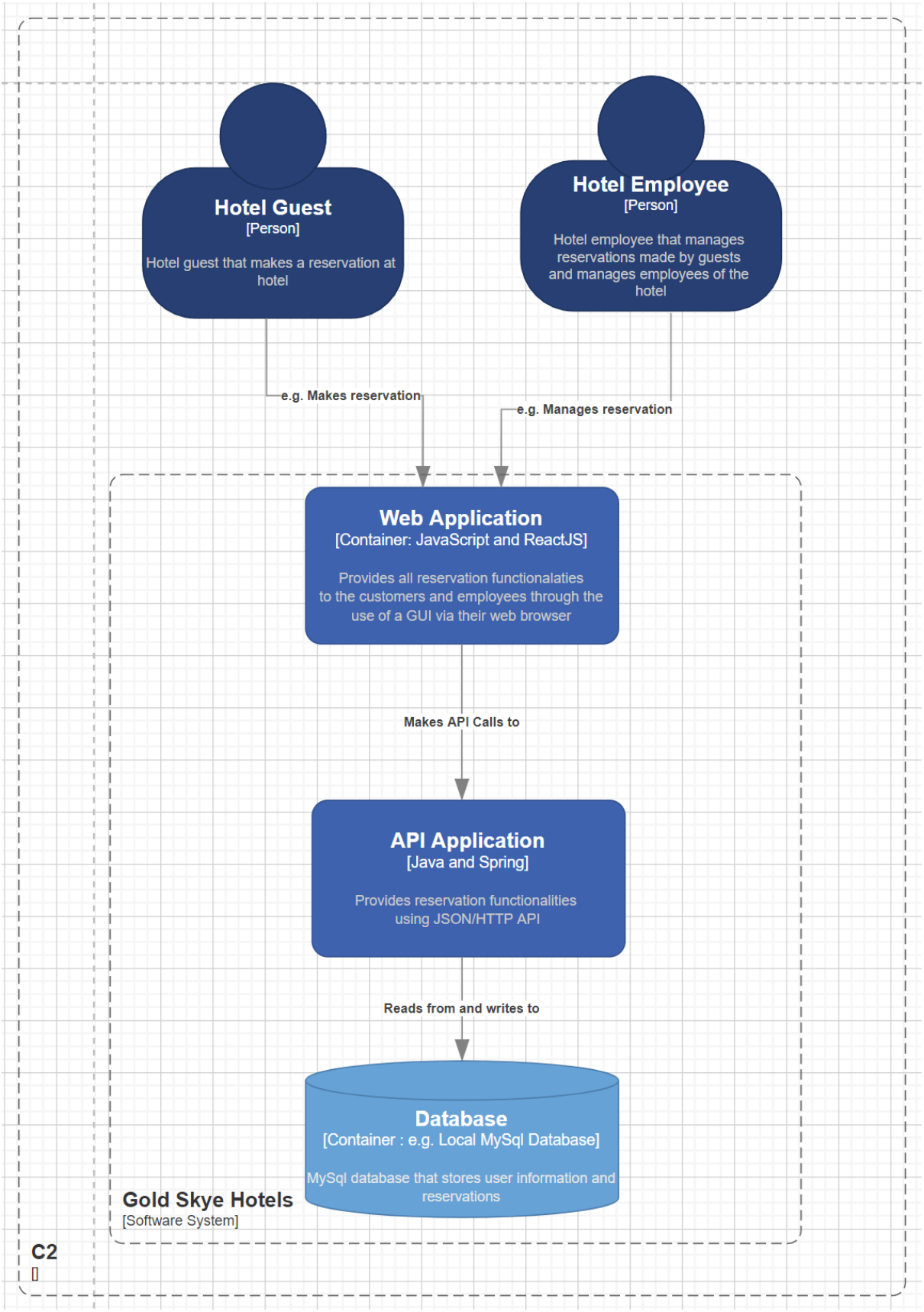
2. Architecture

2.1 C4 Diagram

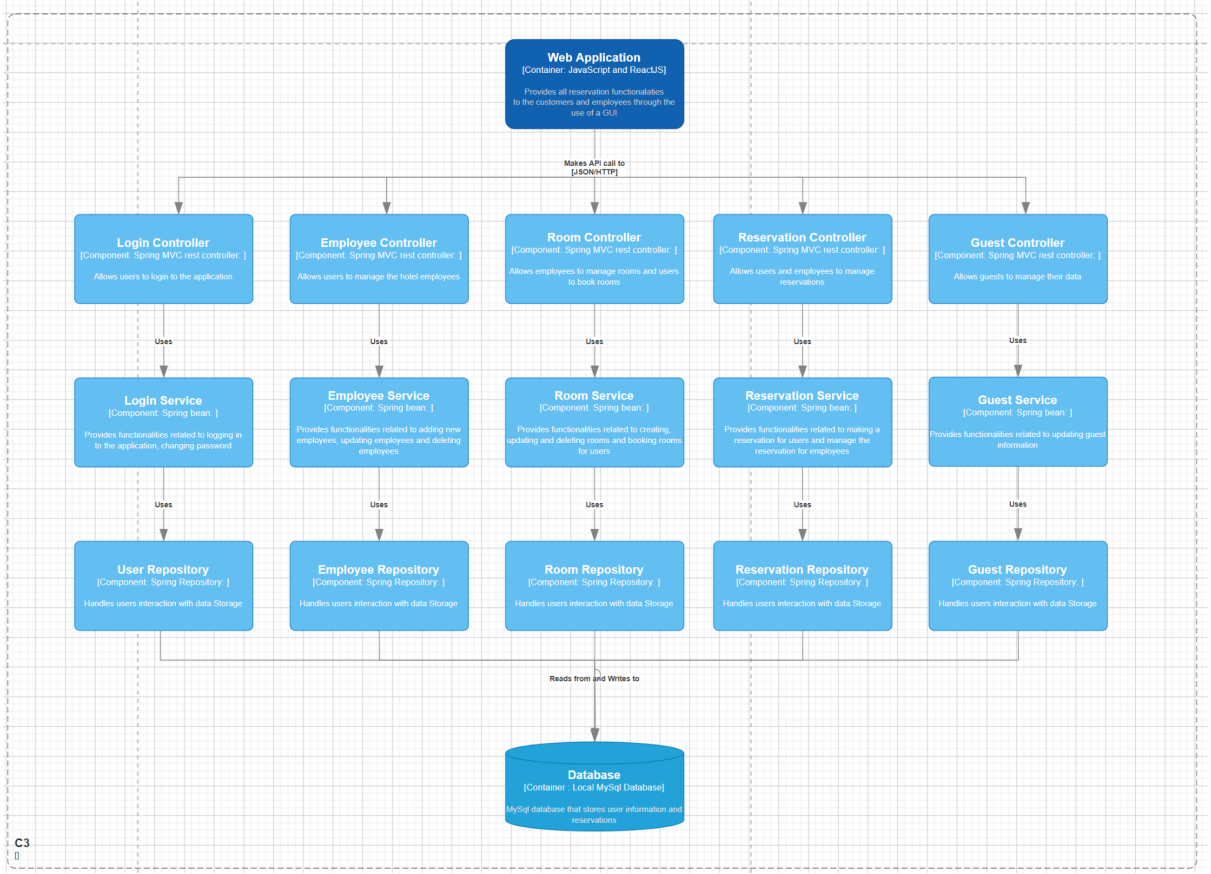
C1: There are two types of users for my application, the guests of the hotel who can manage their bookings and the employees who work at the hotel, who can manage their daily work tasks.



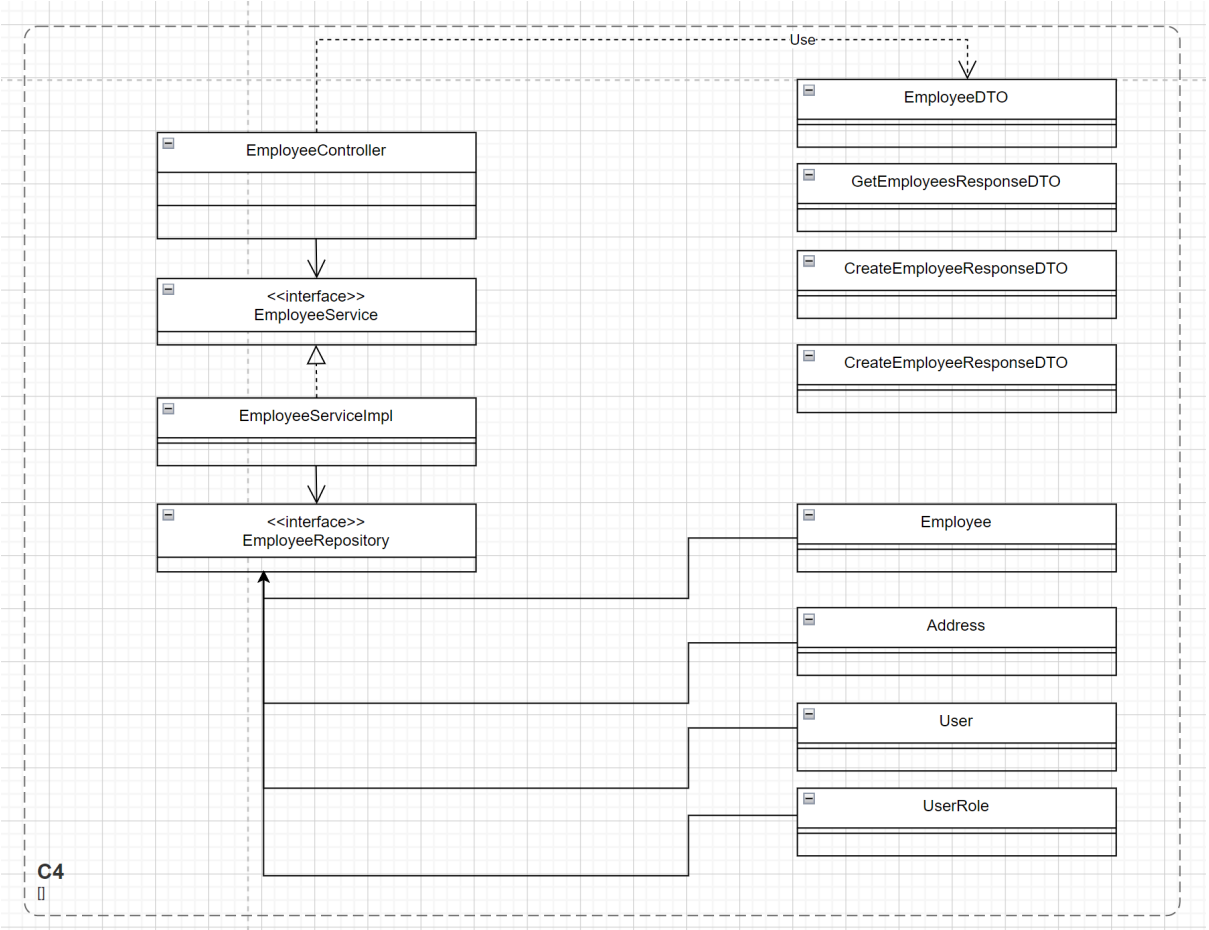
C2: The system is divided in two parts, the web application that is built using the front end JavaScript library React and the backend Java APIs.



C3: For now, the API is divided into 5 sections, a section for rooms, reservations, accounts, employees and login. These 5 sections are separated into 3 layers, the Controller, Service and Repository layer, to guarantee the three layer design.



C4: This is one part of the C4. It only shows the employee part. The DTOs that are used by the EmployeeController are the CreateEmployeeResponseDTO, GetEmployeesResponseDTO and the EmployeeDTO. It is connected to the Business Layer through the EmployeeService interface. EmployeeServiceImpl implements this interface. This is connected to the repository layer by using the EmployeeRepository. This layer uses the Employee, Address, User and UserRole entities.



2.2 How is SOLID guaranteed

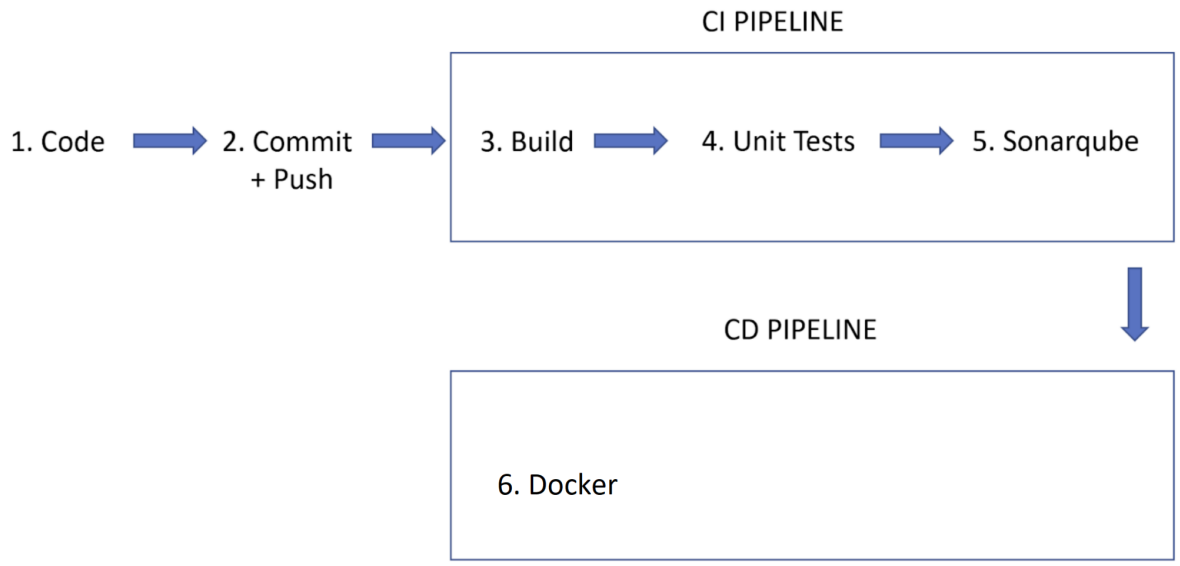
The solid principles are used in Object Oriented Design, to help create a design that is easily maintainable and extendable.

- **S**ingle Responsibility:
  Every single class created is responsible for only one task. This is possible by dividing the controllers by feature, so there are no controllers responsible for example both the employees and the reservations at the same time.
- **O**pen/Closed Principle:
  To keep the code extendable, the open/closed principle is used. Using interfaces will help with this, because this means you don't have to change the existing code if you want to add a new feature. The objects are open for extension but closed for modification. An example for this in my application is the payment method. If the company decides to add another payment method, they can add it without having to modify the current payment methods.
- **L**iskov substitution:
  The liskov substitution means that every subclass should be able to be substituted by the base class. For my application I use inheritance in the Person class. The person class is the main class and Guest and Employee are subclasses. For the different Employee types I will add more subclasses in the future, to separate the functionalities.

- **I**nterface Segregation Principle:
  I will create multiple interfaces to segregate functionalities between users, so that for example a front desk employee doesn't have access to the statistics meant for the manager.
- **D**ependency Inversion Principle:
  The layers are separated by interfaces, the classes depend on the interfaces and are given them when created. For example if you want to switch between using a fake database and a real database you can give the class another instance of the database, without getting any errors or having to modify the existing code.

3. CI/CD Diagram

This diagram shows the CI/CD Pipelines. The CD pipeline will be added later.



3.1 CI Pipeline

1. The first stage is the code stage. In this stage the new code will be created.
2. After the new code is added, the code needs to be committed and then pushed to git. This triggers the pipeline.
3. Stage 3 is the first stage of the CI Pipeline. It will build the application, if there are no errors in the build it will pass, otherwise it will fail.
4. Stage 4 triggers the unit tests, all the unit tests will run and if they all pass this stage will pass, otherwise it will fail.
5. The last part of the CI Pipeline is the sonarqube stage. This triggers the sonarqube build, so you get a sonarqube report.

3.2 CD Pipeline

6. The first stage of the CD pipeline, and also the final stage of the entire CI/CD pipeline, is docker. This stage triggers the docker compose, which will run the docker containers for the frontend, backend and database.