2025

Software Architecture Document

DRIESSEN

SANCHEZ, FRANCISCO; THOMAS, GILTON; PESTANA, CRISTIANO; WERNECKROALE, MIGUEL; LE, MINH

FONTYS UNIVERSITY OF APPLIED SCIENCE | Rachelsmolen 1, 5612 MA Eindhoven

Table of Contents

Architecture Constraints (AC)

The curriculum provides the learning material for the semester. Our team has decided to follow this material as outlined in the curriculum to develop the software application for the client. The tools and concepts acquired from the learning material are listed below, each accompanied by a brief explanation.

Front-End

AC: React Framework

React.js is a free and open-source front-end JavaScript library for building user interfaces based on components.

React.js can be used to develop single-page, mobile, or server-rendered applications with frameworks like Next.js. Because React is only concerned with the user interface and rendering components to the DOM, React.js applications often rely on libraries for routing and other client-side functionality. A key advantage of React.js is that it only re-renders those parts of the page that have changed, avoiding unnecessary re rendering of unchanged DOM elements.

AC: JavaScript Programming Language

JavaScript is a popular programming language used primarily for creating dynamic and interactive elements on web pages. It is an essential component of modern web development alongside HTML and CSS.

JavaScript is a versatile language that can be used for a wide range of applications, from simple website enhancements to complex web applications and server-side development using frameworks like Node.js.

Back-End

AC: Spring Framework

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.

A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

AC: Java Programming Language

Java is a multiplatform, object-oriented programming language that runs on billions of devices worldwide. It powers applications, smartphone operating systems, enterprise

software, and many well-known programs. Despite having been invented over 20 years ago, Java is currently the most popular programming language for app developers.

AC: Project Lombok Library

Project Lombok is a java library tool that is used to minimize/remove the boilerplate code and save the precious time of developers during development by just using some annotations. In addition to it, it also increases the readability of the source code and saves space.

Database

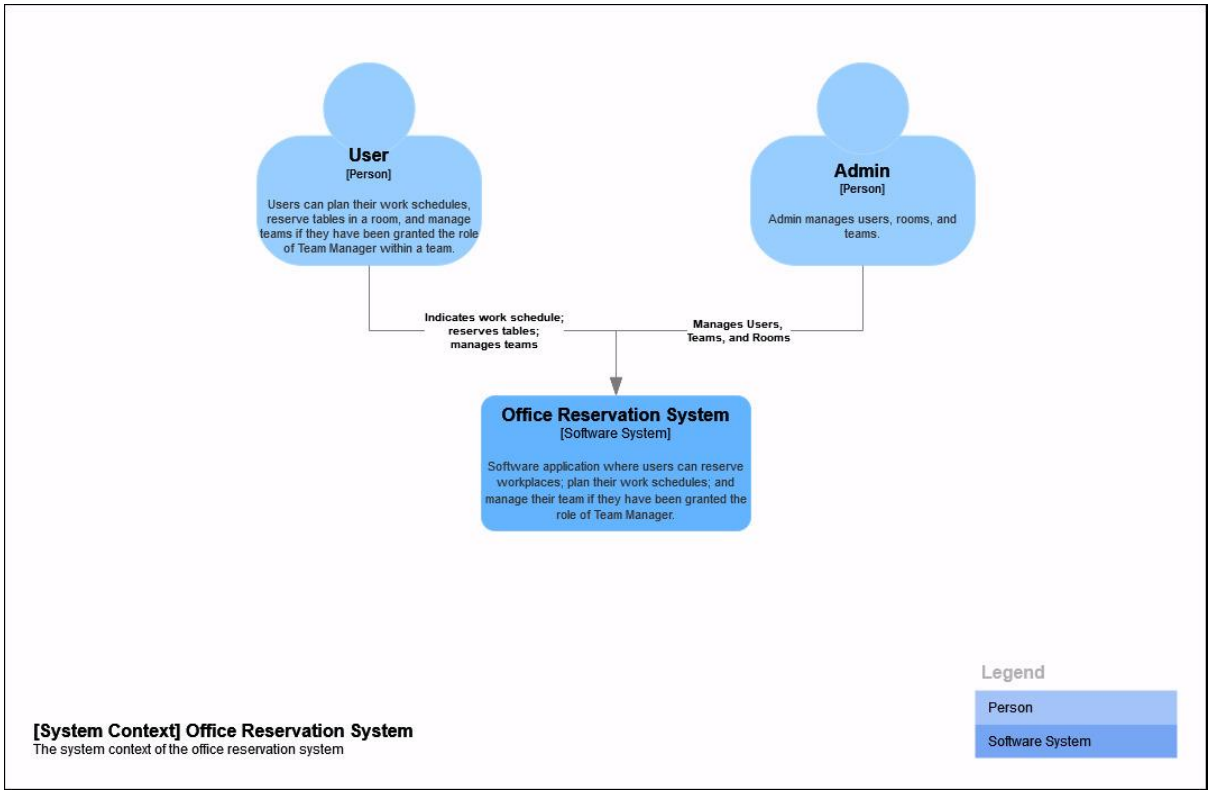
AC: MYSQL RDBMS

MySQL is a popular relational database management system (RDBMS) that is widely used for managing structured data. RDBMS refers to a type of database management system that organizes data into tables with rows and columns and establishes relationships between these tables.

C4 Model Diagrams

Level 1: System Context Diagram

A System Context diagram provides a starting point, showing how the software system in scope fits into the world around it.



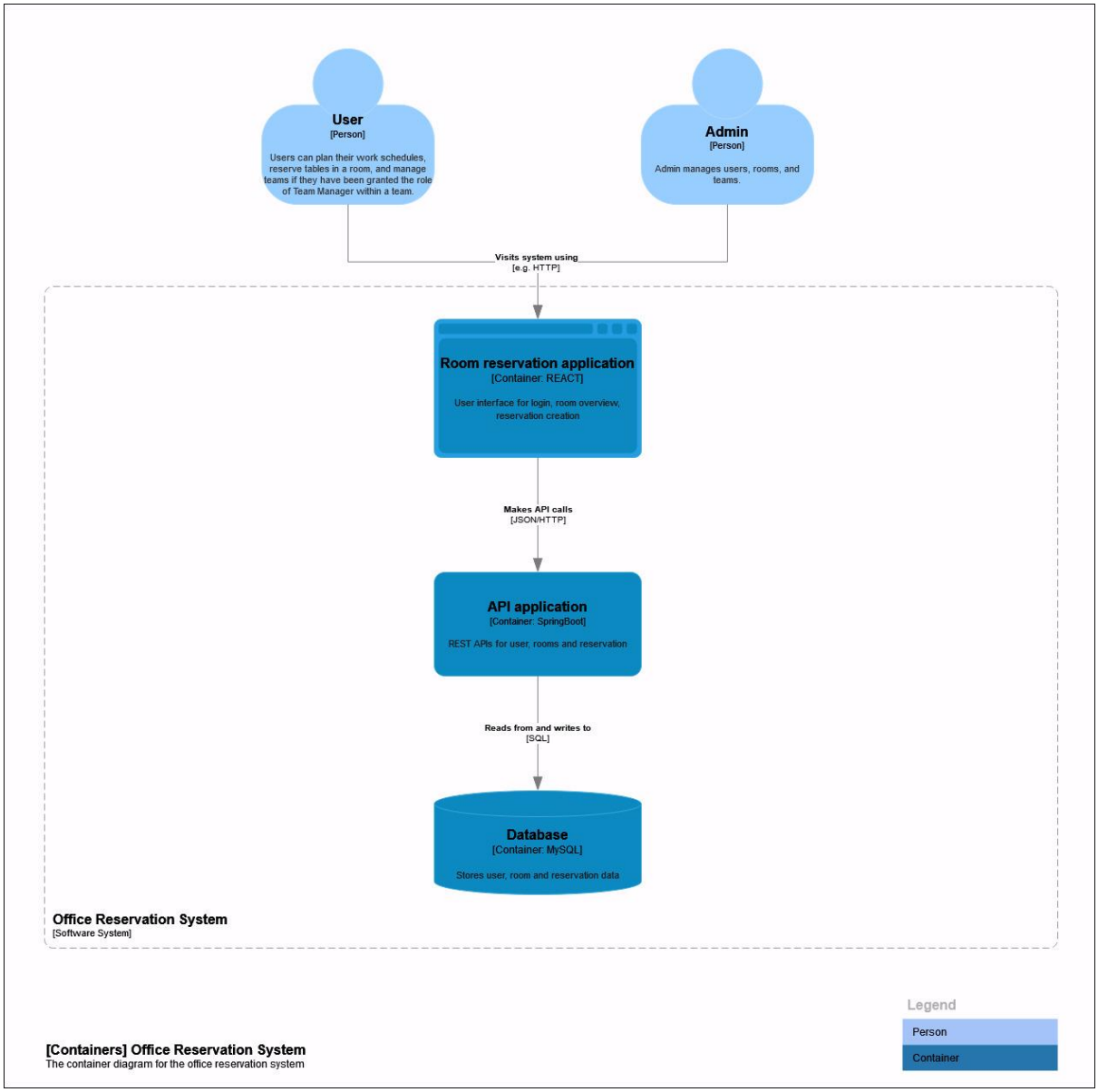
The system context diagram depicts the interactions between the users and the Office Reservation System. The system primarily interacts with two types of users:

1. Users: Regular users can plan their work schedules, reserve tables in rooms, and manage teams if granted the 'Team Manager' role within their team.
2. Admins: Admins are responsible for managing users, teams, and rooms.

The Office Reservation System is an application that facilitates these functionalities, serving as the main point of interaction for both user types.

Level 2: Container Diagram

A Container diagram zooms into the software system in scope, showing the high-level technical building blocks.



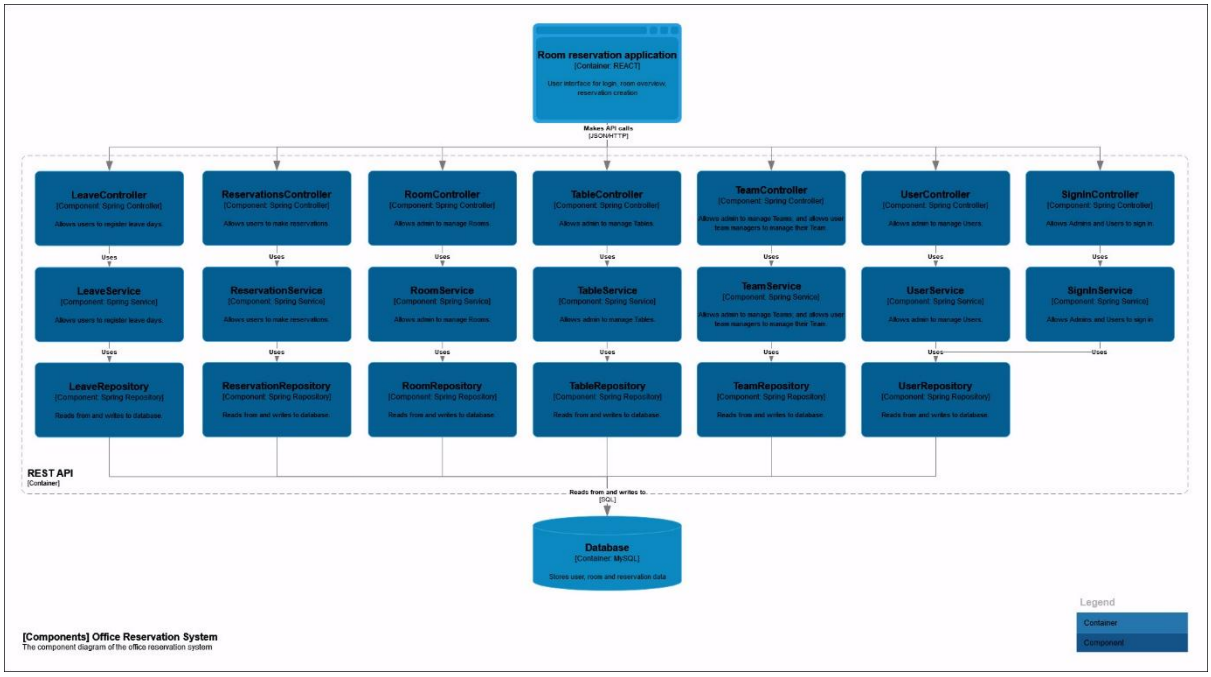
The container diagram illustrates the internal structure of the Office Reservation System, showing how various components interact. Key containers include:

1. Room Reservation Application (React): This is the user interface where users log in, view room availability, and make reservations.
2. API Application (Spring Boot): Serves as the backend, providing RESTful APIs for user, room, and reservation management. It handles requests from the Room Reservation Application.
3. Database (MySQL): Stores critical data, including user profiles, team information, room details, and reservation records.

The flow of data begins with users accessing the system through the React-based interface, which communicates with the API for all operations, with the database storing and retrieving persistent data.

Level 3: Component Diagram

A Component diagram zooms into an individual container, showing the components inside it.



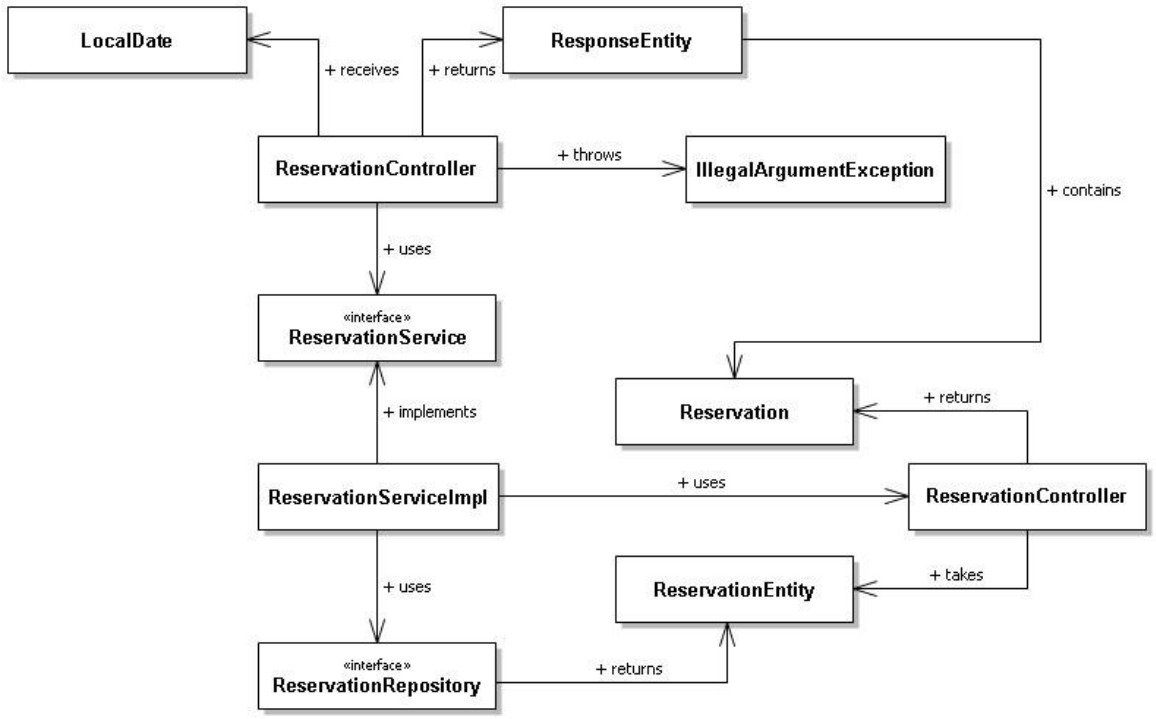
The component diagram breaks down the API application into smaller, focused components, showing their responsibilities and interactions:

1. Controllers: These handle incoming HTTP requests:
    a. LeaveController: Manages leave days.
    b. ReservationsController: Manages reservations.
    c. RoomController: Manages rooms.
    d. TableController: Manages tables.
    e. TeamController: Manages teams.
    f. UserController: Manages users.
    g. SignInController: Allows users to sign in.
2. Services: These components implement business logic:
    a. LeaveService: Manages leave days-related operations.
    b. ReservationService: Processes reservation-related operations.
    c. RoomService: Manages room-related tasks.
    d. TableService: Handles table-related logic.
    e. TeamService: Manages teams and roles.
    f. UserService: Administers user operations.
    g. SignInService: Checks credentials for users and returns jwt.
3. Repositories: Provide data access to and from the database:
    a. LeaveRepository, ReservationRepository, RoomRepository, TableRepository, TeamRepository, UserRepository.

The database remains the core data store, connected to repositories for seamless data persistence and retrieval. The REST API serves as the backend, implementing business logic and database interactions to provide services for the Room Reservation Application (frontend).

Level 4: Code Diagram

A code diagram (e.g. UML class) can be used to zoom into an individual component, showing how that component is implemented at the code level.



The code diagram zooms into an individual component, detailing how the system implements the `getWeeklyReservations` method from the ReservationService to handle retrieval of reservations based on the date. Here's an explanation of the flow:

1. Entry point:
    a. The ReservationController serves as the entry point for incoming HTTP requests. It receives and validates the incoming request.
2. Request Formation:
    a. The ReservationController processes the incoming HTTP request data and forms a LocalDate.
3. Dependency Injection:
    a. The ReservationService is injected at runtime by Spring Boot's dependency injection mechanism. The ReservationController passes the LocalDate to this implementation.
4. Reservation Retrieval Logic:

   a. The ReservationServiceImpl processes the LocalDate to filter the search of reservations based on the week the date is a part of.
   b. The ReservationConverter is then used to convert the Reservation entities into domain objects.

5. Database Interaction:
   a. The ReservationRepository handles the retrieval of all reservations in the system.
   b. The ReservationRepository returns the entities.
6. Response Formation:
   a. The ReservationServiceImpl returns the converted list of domain objects to the ReservationController.
7. Response Delivery:
   a. The ReservationController sends the list of domain objects back to the Single-Page Application (SPA) as an HTTP response.
8. Error Handling:
   a. If the data in the HTTP request is invalid, the ReservationController throws an IllegalArgumentException, ensuring an appropriate error response.

Solid Principles

Single Responsibility Principle (SRP)

The codebase adheres to the Single Responsibility Principle by being responsible for their respective functionality. The project is organized in separate packages, such as controller, business, repository, and configuration., which promotes structural clarity. Inside these packages, these classes are further divided into their specific tasks:

- Controller: These classes manage incoming HTTP requests and outgoing responses. They don't handle any business logic or data storage, only how data flows in and out of the system.
- Business: This contains classes that handle business tasks for specific domain object (e.g., LeaveServiceImpl, SignInServiceImpl). Each class here focuses on just one domain object, instead of combining all the logic in one big service class. Moreover, the business package contains nested packages that contains classes that are responsible for their respective functionality:
  - Converter: These classes handle changes between different formats (like turning an entity into a domain object) so the use case classes don't have to.
  - Exception: These are custom error classes (like EmailAlreadyExistsException) that deal with specific problems related to the business logic.
  - Validator: These classes handle the validation of domain objects by ensuring that their variables adhere to specific constraints. If any of the constraints are violated, an exception from the nested exception package will be thrown.
- Persistence: This contains JPA repositories (like ReservationRepository), which are responsible for interacting with the database.

Open-Closed Principle (OCP)

The codebase adheres to the Open-Closed Principle by defining interfaces, allowing the system to be extended without modifying existing code. This principle is applied through a design where both controllers and business logic depend on interfaces rather than their concrete implementations.

- Interfaces for Business Logic: Interfaces such as LeaveService, SignInService, and ReservationService define contracts for specific functionalities. Controllers interact with these interfaces instead of their implementations, enabling new modifications to be added without altering the controller code.
- Interfaces for Repositories: The business logic classes depend on interfaces in the persistence package, such as RoomRepository from Spring Data JPA.

- Interfaces for Repositories: The business logic classes depend on interfaces in the persistence package, such as TableRepository from Spring Data JPA.

Liskov Substitution Principle (LSP)

The codebase adheres to the Liskov Substitution Principle (LSP) because implementations of interfaces such as LeaveService, SignInService, and ReservationService can be substituted seamlessly without altering the program's behaviour. This ensures that all implementations conform to the expectations set by their interfaces, maintaining consistent functionality and predictable outcomes when replacements or new implementations are introduced.

Interface Segregation Principle (ISP)

The codebase adheres to the Interface Segregation Principle by designing interfaces such as LeaveService, SignInService, and ReservationService to focus on specific functionalities. Each interface is tailored to a particular use case, defining only the methods required to perform that functionality. This approach prevents the creation of service classes overloaded with multiple unrelated functionalities, ensuring that implementing classes are not forced to include irrelevant methods.

Dependency Inversion Principle (DIP)

The codebase adheres to the Dependency Inversion Principle because controllers and business logic depend on abstractions (e.g., RoomRepository, TableRepository, UserRepository, ReservationRepository) rather than concrete implementations. This design ensures that high-level modules (such as controllers and business logic) are not tightly coupled to low-level modules (such as business logic implementations and repositories).