**Team Name:** Astrum Distributed Systems
**Project Name:** Hotel Management and Booking System

**Team Number:** 3
**Team Members:**


- Karol Wojcik (18322146)
- Peter O'Riordan (18749619)
- Ross Murphy (20207271)


## Project Synopsis

The hotel management application is a comprehensive system designed to streamline and automate various aspects of hotel operations. It aims to facilitate efficient room management, booking management, customer management, availability tracking, and billing/invoicing processes within the hotel industry.

**Application Domain:**

The application primarily targets hotels and hospitality businesses, providing them with a centralized platform to manage their rooms, bookings, and customer interactions. It caters to the specific needs of hotel administrators, staff, and guests, enhancing overall operational efficiency and guest satisfaction.

**Functionality:**

The application enables users to add, update, and delete room information, including details such as room type, bed size, balcony availability, and more. It offers robust booking management capabilities, allowing users to create and manage room bookings with check-in and check-out dates, customer details, and pricing information. The system also provides customer management features to maintain comprehensive records of guests, including their names, emails, and phone numbers.

Additionally, the application incorporates availability tracking functionality, allowing users to monitor room availability based on check-in and check-out dates. This helps in efficiently managing bookings and ensuring optimal occupancy rates. Furthermore, the system generates invoices for bookings, considering factors such as room type and duration of stay, enabling accurate billing and invoicing processes. Furthermore, it adds ability to scale the application with numerous replicas/hotels that can be added by the admin into the system.

## Project Implementation and Demonstration

*Link to Project GitLab:* ***Access Here***

*Link to Project Video Demonstration (Private YouTube Video):* ***Access Here***

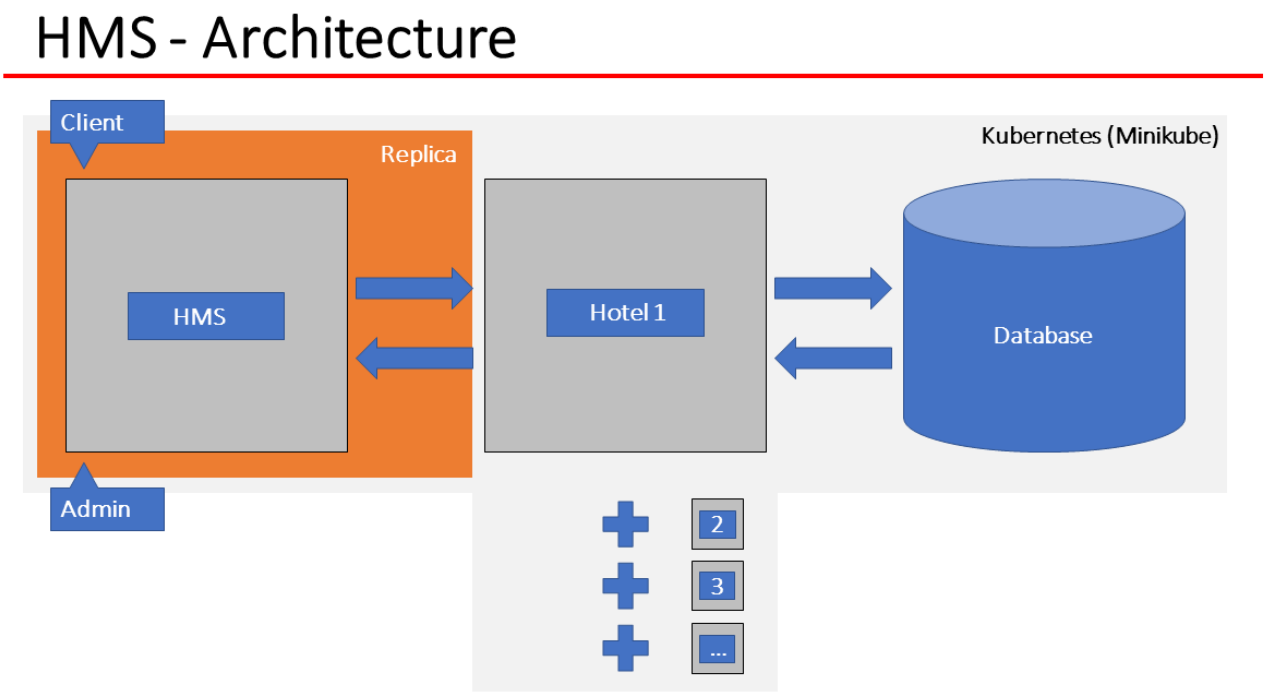
## Technology Stack

The Hotel Management and Booking System Java REST Application using Kubernetes leverages a carefully chosen to set of technologies to provide a robust and efficient solution. The following technologies were used, and each serves a specific purpose:

- **Java:**
  - Used for backend development.
  - Java is a versatile and widely adopted programming language known for its scalability, portability, and extensive ecosystem of libraries and frameworks.
- **REST:**
  - Architectural style that allows for standardized communication between client and server using HTTP methods and resource-based URLs.
  - REST enables the creation of scalable and interoperable web services, providing a flexible and efficient means of communication between different components of the application.
- **Spring Boot:**
  - Framework used for developing the web application.
  - Spring Boot simplifies the development process by providing a comprehensive set of tools, libraries, and conventions for building robust and scalable Java applications.
- **H2 Database:**
  - In-memory database used for storing room and booking data.
  - H2 Database is lightweight, fast, and well-suited for development and testing purposes. It allows for efficient data storage and retrieval without the need for an external database server.
- **Docker:**
  - A platform that enables the creation, deployment, and running of applications in lightweight and isolated containers.
  - Docker provides a consistent and reproducible environment for deploying applications, ensuring that the application runs consistently across different platforms and environments.
- **Kubernetes (Minikube):**
  - A container orchestration tool that automates the management, scaling, and deployment of containerized applications.
  - Kubernetes simplifies the management of distributed applications by providing features such as automated scaling, load balancing, and self-healing capabilities.
- **JavaScript:**
  - A programming language commonly used to add interactivity and dynamic functionality to web pages.
  - JavaScript enhances the user experience by enabling dynamic and responsive features in the application's frontend.
- **HTML/CSS:**
  - Frontend technologies used for creating user interfaces.
  - HTML provides the structure of web pages, while CSS is used for styling and layout, allowing for the creation of visually appealing and user-friendly interfaces.
- **GitLab:**
  - Version control system used for collaboration and code management.
  - GitLab facilitates collaboration among team members, version control management, and continuous integration, ensuring efficient code management and seamless project collaboration.

The chosen set of technologies was carefully considered to meet the requirements of the Hotel Management and Booking System. Java, with its robust ecosystem, provided a solid foundation for backend development. REST architecture enabled standardized communication between the client and server, ensuring interoperability. Spring Boot streamlined development processes and provided a range of useful features. H2 Database offered a lightweight and efficient solution for data storage. Docker and Kubernetes enabled containerization and efficient deployment. JavaScript and HTML/CSS enhanced the frontend user experience. GitLab ensured efficient version control and collaborative code management through AGILE methodology.

## Project System Overview




**Components of System**

There is a plethora of components which make up the HMS Architecture, all working together in a distributed manner to provide the highest level of reliable functionality, namely:

**Client:**

The client component of our system plays a crucial role in enabling users to interact with the hotel management and booking system. Through the client interface, users can browse available rooms, request quotes from hotels, and create bookings based on their desired criteria.

**Admin:**

Administrators have privileged access to functionalities such as user management, room management, booking management, and hotel addition/management. The admin component ensures secure access and control over system resources, allowing administrators to efficiently oversee the operations of the hotel management system.

**HMS:**

The HMS (Hotel Management System) component acts as the central hub of our architecture, integrating all the essential functionalities required for efficient hotel management. It encompasses modules for room management, booking management, customer management, availability tracking, billing, and invoicing. The HMS component ensures seamless coordination between different modules, enabling smooth operations and effective management of hotel resources. It serves as the backbone of our system, providing the necessary infrastructure to handle the core functions of hotel management.

**Hotel(s):**

The hotel component represents the individual hotel entities within our system. Each hotel is associated with a set of rooms, availability, pricing, and other relevant information. The hotel component allows hotels to register their properties, manage room details, update availability, and handle booking requests. It provides a platform for hotels to showcase their offerings and interact with potential customers through the system.

**H2 Database:**

The H2 Database component serves as the in-memory database system for our hotel management and booking system. It stores and manages the data related to rooms, bookings, customers, and other system entities. The H2 Database offers fast and efficient data storage and retrieval, enabling seamless access to critical information. By leveraging an in-memory database, our system achieves high performance and responsiveness. The H2 Database component ensures data integrity, consistency, and persistence, supporting the smooth functioning of the entire system.

**Kubernetes (Minikube):**

The Kubernetes (Minikube) component is a container orchestration tool that plays a crucial role in managing the deployment and scaling of our system's components. It automates the distribution and coordination of containerized services, ensuring optimal resource allocation and fault tolerance. Kubernetes (Minikube) enables us to scale our system horizontally by adding or removing instances of services based on demand. It monitors the health and availability of containers, automatically restarting failed instances, and maintaining system stability. The Kubernetes (Minikube) component enhances the reliability, scalability, and fault tolerance of our hotel management and booking system.

**How the System Functions (Based on Diagram)**

The system functions with the Client providing desired room booking information to the Hotel(s) which through the HMS. The Hotel queries the H2 Database to retrieve relevant room information to the client by matching the quotes presented to the client based on the client provided information. The client the proceeds to select the desired booking which is sent back through the HMS to the Hotel(s) which updates the H2 Database with the relevant booking information in the Booking Table (once the client successfully completes the payment and booking confirmation steps).

On the other side the admin who possesses administrative privileges in managing rooms, clients, bookings, and the hotel(s). Can utilize the functionalities afforded to them in real time which allow the admin to manage the hotel(s) accordingly.

In the background the HMS application which is deployed using Kubernetes (Minikube) provides scalability and fault-tolerance through allowing client, admin, and HMS replicas to scale the application accordingly (in addition to adding multiple hotel admin-side). The application’s resource allocation and up-time is maintained allowing for persistent and reliable usage without any down-time because of queueing issues in the event of multiple clients performing their desired bookings. The problem of having a reliable booking and hotel management system that allows for scaling and management of multiple real-time components benefits greatly from the implementation of the architecture described in the diagram in addition to the technologies utilized.

The system functions by facilitating communication between the client, hotel(s), and HMS, utilizing the H2 Database for data retrieval and storage. The client selects a booking, which is processed by the HMS and updated in the H2 Database. The admin manages the system's functionalities in real-time. The system, deployed using Kubernetes (Minikube), ensures scalability and fault tolerance, enabling multiple replicas and maintaining resource load and uptime. The implemented architecture and technologies contribute to a reliable, scalable, and efficient hotel management and booking system.

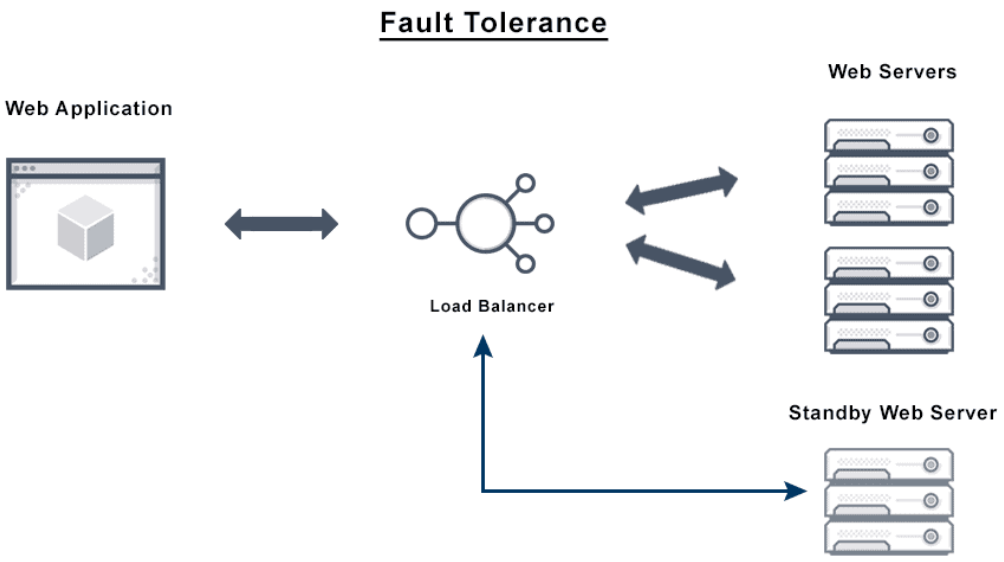
**How the System is Designed to Support Scalability and Fault Tolerance.**

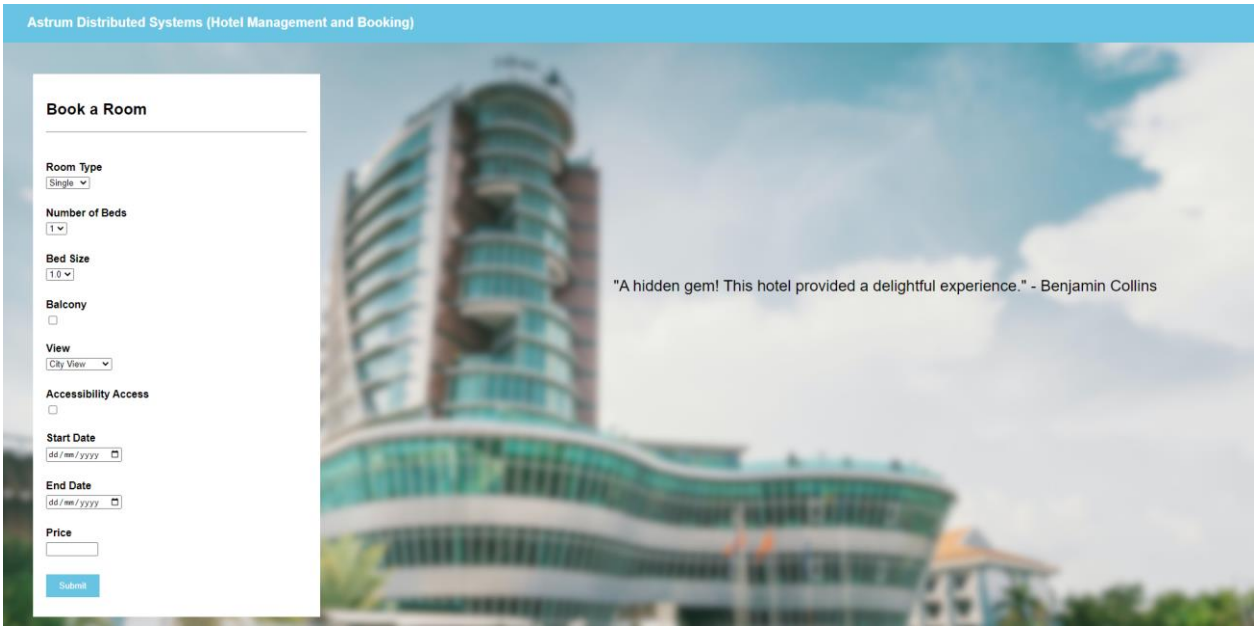
Scalability and fault tolerance are integral aspects of the design of our hotel management and booking system. We have implemented several strategies to ensure the system can handle increasing demands and maintain high availability. Here is an overview of how the system is designed to support scalability and fault tolerance:

We have adopted Docker to containerize our application components. By encapsulating each component into a separate container, we achieve a modular and scalable architecture. Docker containers provide isolation, making it easier to scale individual components independently. Additionally, Docker enables easy deployment and replication of containers across different environments, ensuring consistency and reliability.

To manage and orchestrate our containerized application, we have leveraged Kubernetes, which provides powerful features for automating the deployment, scaling, and management of containerized applications. By defining replica sets, we can scale our application horizontally, allowing for the addition of more instances of a particular component to handle increased traffic or workload. Kubernetes also monitors the health of the system and automatically restarts failed containers, ensuring fault tolerance.

Our system incorporates a load balancer, which plays a crucial role in distributing incoming traffic evenly across multiple instances of our application. By evenly distributing the workload, the load balancer prevents any single component from being overwhelmed and ensures optimal resource utilization. This load balancing mechanism not only improves performance but also enhances fault tolerance. In the event of a component failure, the load balancer automatically redirects traffic to healthy instances, ensuring uninterrupted service availability.



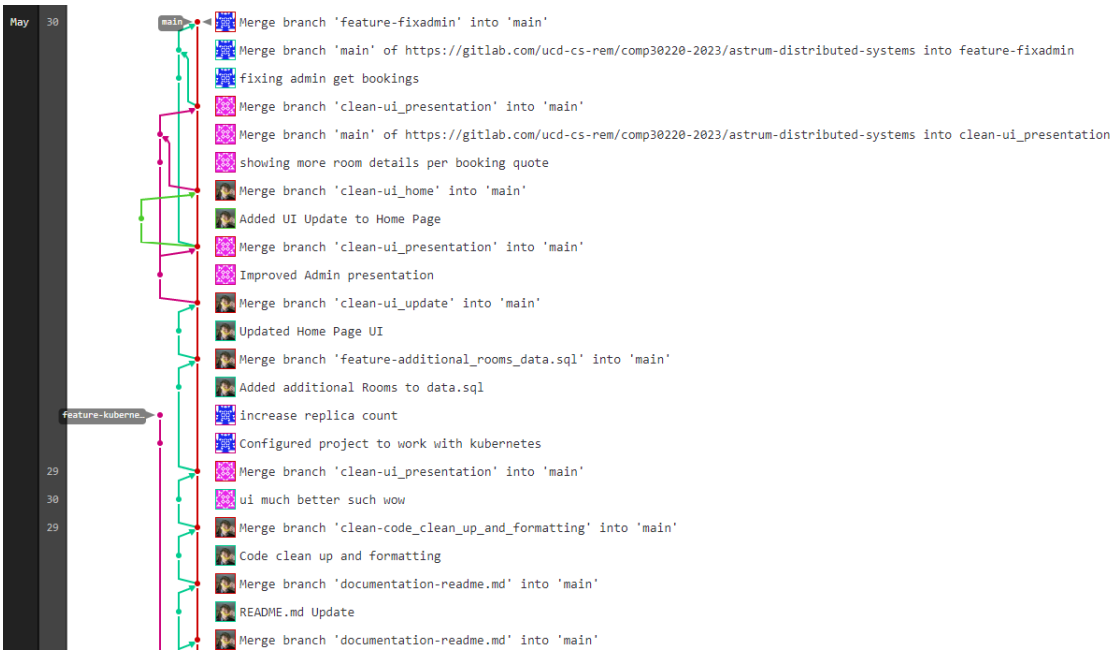



# Project Contributions

- **Karol Wojcik:**
  - **UI (HTML/CSS)**
  - **Hotel Services**
  - **Booking/Payment**
  - **data.sql**
  - **Docker**
  - **README.md**
  - **Report**
  - **Editing Video**
- **Peter O'Riordan:**
  - **UI (HTML/CSS)**
  - **HMTL/CSS**
  - **Spring Boot**
  - **Hotel**
  - **Booking/Payment**
  - **Quotation**
  - **Bash Scripts**
  - **JavaScript**
- **Ross Murphy:**
  - **Admin**
  - **Kubernetes (Minikube)**
  - **H2 Database**
  - **Hotels**
  - **HMS**
  - **Kubernetes Scripts**
  - **Models and Repositories**
  - **Controller Logic**

The implementation of the Hotel Management and Booking System project was a collaborative effort that involved the contributions of all team members. Each team member took responsibility for specific aspects of the project, utilizing their expertise and skills to deliver a high-quality application.

Throughout the project, the team followed AGILE methodologies and employed pair programming practices. This approach facilitated efficient collaboration, ensured organized project management, and fostered effective problem-solving and code review processes.



# Project Reflections

**What were the key challenges you have faced in completing the project? How did you overcome them?**

During this project, we encountered several key challenges that tested our problem-solving skills and required us to adapt our approach.

Developing a comprehensive hotel management and booking system involved dealing with multiple interconnected components and functionalities. To overcome this challenge, we carefully planned the system architecture, breaking it down into manageable modules and assigning specific tasks to team members. Regular communication and coordination ensured a cohesive integration of the different components.

Incorporating distributed technologies like Docker and Kubernetes introduced complexities related to setup, configuration, and deployment. We had to familiarize ourselves with these technologies, invest time in learning best practices, and troubleshoot any issues that arose. Collaboration and knowledge sharing within the team helped us overcome these challenges and successfully implement the distributed aspects of the system.

Ensuring the reliability and accuracy of the system required thorough testing and validation. We faced challenges in designing and executing comprehensive test cases, especially considering the distributed nature of the application. By adopting a systematic approach, writing effective test cases, and conducting extensive testing, we were able to identify and fix bugs, ensuring a stable and robust system.

**What would you have done differently if you could start again?**

While we managed to complete the project within the allocated timeframe, we realized that better time management could have been beneficial. Allocating more time for certain tasks and setting clearer milestones would have helped us maintain a smoother workflow and address any unexpected challenges more effectively.

Although we actively collaborated and utilized pair programming, we believe that further emphasizing collaboration could have improved overall productivity. Regular code reviews, knowledge sharing sessions, and more frequent communication could have facilitated a stronger team dynamic and promoted continuous learning.

Looking back, we could have also explored additional Database Options to experiment with additional ways to make even more efficient data handling with the HMS application.

**What have you learnt about the technologies you have used? Limitations? Benefits?**

Through this project, we gained valuable insights into the technologies we used, including Java, Spring Boot, Docker, Kubernetes, and more.

While powerful and widely adopted, each technology has its own limitations. For example, Docker introduces some overhead in terms of resource utilization, and Kubernetes configuration can be complex for beginners. Understanding these limitations allowed us to make informed decisions and develop workarounds when necessary.

The technologies we utilized provided numerous benefits. Java's robustness and extensive ecosystem made it a reliable choice for backend development. Spring Boot simplified the development process by providing a comprehensive framework with built-in features. Docker allowed us to containerize our application, enabling seamless deployment across different environments. Kubernetes offered scalable and automated container orchestration, ensuring efficient management of our distributed system.