

Development of Human Resource Internal Application

Software Design and Research Report.

Rowy Hardware



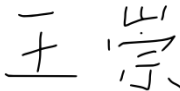


Name	Position	email	phone
Wan Huey Shin	Team Leader	J23039703@student.newinti.edu.my	0122819878
Ismail Mamedov	Team Member	J19031355@student.newinti.edu.my	0173823277
Cheah Ruo Ying	Team Member	J22036361@student.newinti.edu.my	01126204705
Wang Chong	Team Member	J21035099@student.newinti.edu.my	01155077416
Lee En Qi	Team Member	J22036481@student.newinti.edu.my	0124089927

SWE40001, Software Engineering Project A, Semester 4, 27 Oct 2024


DOCUMENT CHANGE CONTROL

Version	Date	Authors	Summary of Changes

DOCUMENT SIGN OFF

Name	Position	Signature	Date
Wan Huey Shin	Team Leader		27 Oct 2024
Cheah Ruo Ying	Team Member		27 Oct 2024
Wang Chong	Team Member		27 Oct 2024
Lee En Qi	Team Member		27 Oct 2024
Ismail Mamedov	Team Member		27 Oct 2024

CLIENT SIGN OFF

Name	Position	Signature	Date
Lew Kok Sin	Director		12 Nov 2024
Organisation			
Rowy Hardware Sdn Bhd			

1. Introduction

The **Employee Performance Management System (EPMS)** is software designed to simplify important HR tasks like tracking employee performance, managing leave requests, giving feedback, and sharing announcements. The goal is to help employees grow, make them more responsible, and improve the overall productivity of the company. With features such as performance tracking, dashboards, leaderboards, and a reward system, the EPMS will make managing tasks easier for both employees and managers.

Purpose

This Software Design and Research Report (SRS) provides a detailed plan for how the EPMS will work, including its features and requirements. It acts as a guide for developers, project managers, HR teams, and quality assurance staff, so everyone knows what the system needs to do. The report ensures that all involved teams stay on the same page throughout the project, helping the final product meet the organization's expectations. Communication between teams will make sure the system is built correctly and works as intended.

Target Reader/Audience:

- Developers understand the system's features and how to build them.
- Project managers to keep the project on track and within scope.
- Quality assurance teams to test the system and make sure it works properly.
- HR staff and managers to use the system effectively for performance management.

The software is developed for organizations to manage employee performance by reducing manual work, tracking progress accurately, and encouraging employee engagement with rewards and feedback.

1.1 Overview

This document gives a complete summary of the design and development of the Employee Performance Management System (EPMS), which helps with tasks like tracking employee performance, handling leave requests, giving feedback, and posting announcements. The system also encourages employee growth and engagement through features like dashboards, leaderboards, and rewards, making it easier to improve productivity. It guides developers, project managers, HR staff, and quality assurance teams to work together smoothly and stay aligned throughout the project. The report covers key areas such as the system's purpose, goals, assumptions, design decisions, and alternative approaches. It also

shows how the design meets requirements and includes research on similar systems and technologies. This document acts as a step-by-step plan to ensure the EPMS is built and implemented successfully, with everyone involved understanding the project's scope and goals.

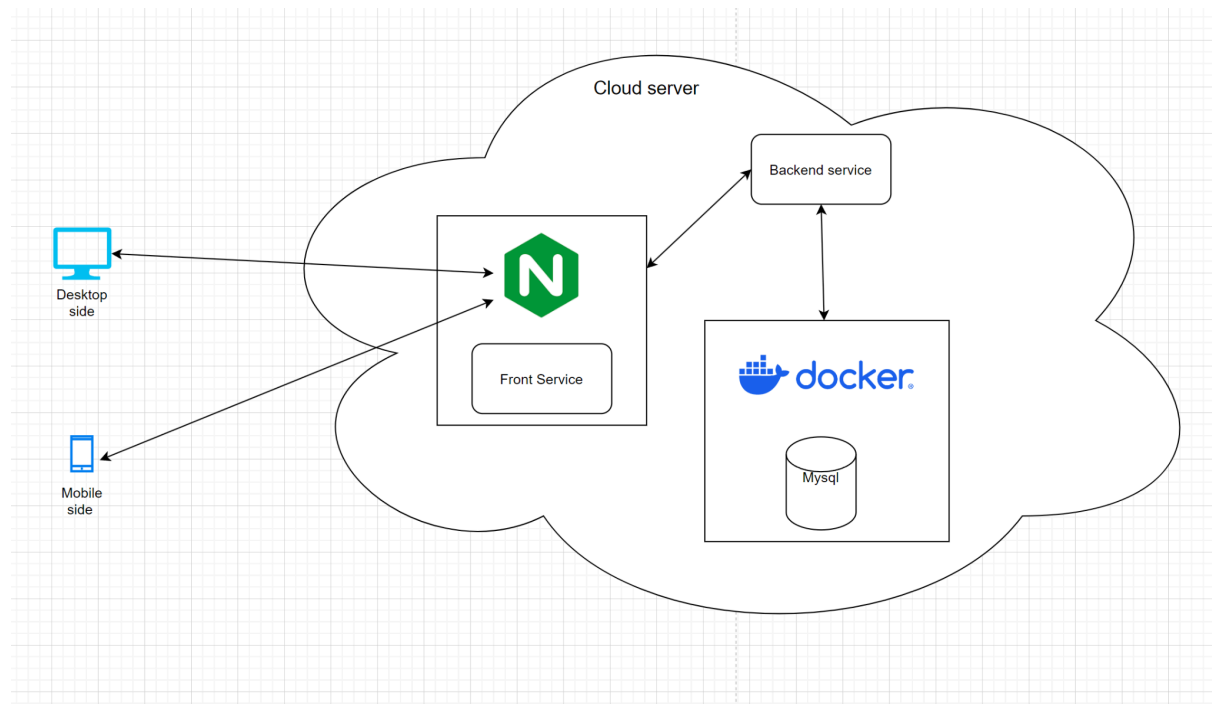
1.2 Definitions, Acronyms and Abbreviations

- **API (Application Programming Interface)** is a set of protocols and tools for building software applications, allowing different systems to communicate.
- **Docker** is an open-source platform for automating the deployment of applications inside lightweight, portable containers.
- **EPMS (Employee Performance Management System)** is a software designed to manage employee performance, handle leave requests, and facilitate feedback and announcements.
- **HR (Human Resources)** is the department within an organization responsible for managing employee-related services like recruitment, performance tracking, and development.
- **HTTP/HTTPS (HyperText Transfer Protocol/Secure)** are protocols used for transmitting data over the web, with HTTPS providing encrypted communication for secure transfers.
- **JWT (JSON Web Token)** is a compact, URL-safe method for securely transmitting information between parties as a JSON object.
- **KPI (Key Performance Indicator)** is a measurable value used to evaluate the performance of employees against predetermined objectives.
- **MySQL** is an open-source relational database management system used for storing and managing structured data.
- **Nginx** is a web server that can also be used as a reverse proxy, load balancer, and HTTP cache, improving the performance and scalability of applications.
- **SPA (Single Page Application)** is a web application that loads a single HTML page and dynamically updates the content as the user interacts with the app.

2. Problem Analysis

2.1. System Goals and Objectives

Mainly the high level design should solve the problem of how each model interacts in this system which includes two main parts in large scale that are user and server side. By analyzing the system requirements and functions, we designed the high level design diagram below and listed down how each small part works .



The high level architecture at the system level can make sure that the system runs stably. To make sure of the reliability of the system they are mainly including some stages.

- **User request stage**

Currently, the system allows users to use mobile phones and desktops to access the website. When the http / https request arrives at the website , the website will, according to the user's screen resolution, redirect the user to a suitable site (mobile side and desktop side). By doing that , it reduced the system complexity.

- **Reversed-proxy stage**

The nginx server can provide the reversed-proxy service, all websites are deployed here. Once a user uploaded a file , the reversed-proxy also can make the user follow the url directly to download it.

- **Back-end processing / service**

The back-end service in charge of all logical functions. Such as KPI tracking and system management. By design a well - authorization can make the system more secure.

- **Data access layer**

In the data access layer , it is better to use container technology such as Docker , as each container running in Docker is like running in a real computer. It can make each service become isolated. Once the service is off-line, it can be easily restarted. When we install it, we can map the physical machine path to the container , so the physical machine can easily back-up the data generated by Mysql.

2.2.Assumptions

- **Stable Internet Connection:** The system assumes that both user devices and the server have stable internet connections, particularly for HTTP/HTTPS requests and file uploads/downloads.
- **Sufficient Server Resources:** The backend and reverse proxy are assumed to have sufficient computational resources (CPU, memory, and storage) to handle incoming traffic and file processing tasks without performance degradation, particularly during peak usage times.
- **Screen Resolution for Device Detection:** It is assumed that users' screen resolution is an accurate and sufficient method to determine whether they are using a mobile or desktop device. The system does not account for potential cases where users may resize their browser windows or use devices with atypical resolutions.
- **Service Isolation via Docker:** The use of Docker containers assumes that container technology provides enough isolation between services to ensure stability and ease of management. It is also assumed that the physical machine is robust enough to handle multiple containers and backups concurrently.
- **Scalability via Docker:** Docker is assumed to provide an easy method for scaling services if needed. However, the system assumes that scaling will not be immediately necessary, and the initial architecture can handle the expected load.
- **Basic Security Measures:** It is assumed that a basic authorization mechanism (e.g., user authentication) will suffice for initial deployment, and more advanced security measures (e.g., encryption, intrusion detection) can be implemented later if needed.
- **Container Persistence:** It is assumed that mapping the physical machine path to Docker containers will sufficiently handle data persistence and backup needs, particularly for database services such as MySQL.
- **Modern Browser Support:** The system assumes that users are accessing the website via modern browsers that fully support the necessary web technologies (e.g., HTML5, CSS3, JavaScript).

2.3. Simplifications

- **Single Reverse Proxy:** The system assumes the use of a single reverse-proxy server (Nginx) to handle all user traffic and manage requests for different versions of the website (mobile vs. desktop). This reduces the complexity of managing multiple proxies but may introduce a single point of failure.
- **Containerized Environment:** By using Docker containers for all services (such as database, backend services, etc.), we have simplified deployment and isolation. Each service is treated as an independent unit, reducing configuration management complexity but abstracting away the detailed environment-specific configurations.
- **User Device Handling:** The system automatically redirects users to the appropriate interface based on screen resolution, rather than maintaining entirely separate services for mobile and desktop. This simplifies the user interface design and reduces development overhead but assumes that screen resolution is the only significant difference between devices.
- **File Upload and Download:** By handling file uploads and downloads through a URL redirection method within the reverse proxy, we simplify the file handling process, relying on Nginx rather than creating a separate file management service. This reduces development complexity but limits advanced file handling capabilities.
- **Authorization Simplification:** The system design assumes a basic level of user authorization for accessing backend services. More complex authorization schemes, such as role-based access control or multi-factor authentication, are not currently implemented to keep the initial system design simpler.

3. High-Level System Architecture and Alternatives

3.1. System Architecture

The system employs a microservices-based architecture with four primary layers:

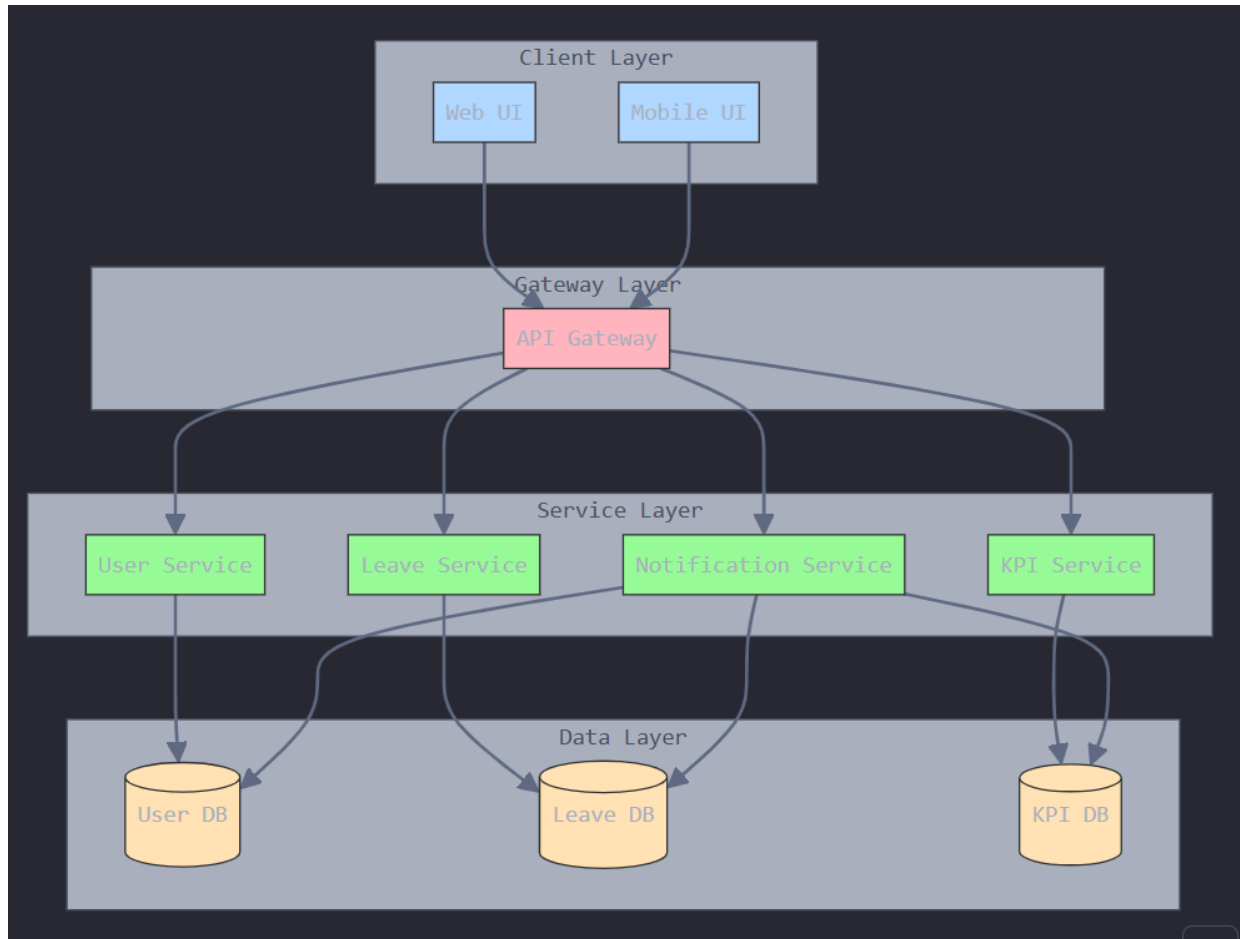


figure: System Architecture Diagram (4 primary layers)

This figure represents the system architecture diagram, illustrating the four primary layers: the Client Layer, Gateway Layer, Service Layer, and Data Layer, each of which plays a crucial role in ensuring efficient communication, processing, and data management within the system.

1. Client Layer

- **Web Client (React Single Page Application)**
 - Provides a responsive user interface that adapts to various screen sizes.
 - Supports offline capabilities to enhance user experience.
 - Utilizes a component-based architecture for modularity and reusability.
- **Mobile Client (Native Applications)**

- Optimized for mobile devices to ensure performance and usability.
- Supports push notifications for real-time updates and engagement.
- Implements local data caching to enhance performance and offline access.

2. Gateway Layer

- **API Gateway**

- Serves as a single entry point for all client requests, simplifying the API structure.
- Manages request routing and load balancing to ensure optimal resource utilization.
- Incorporates authentication and rate limiting to enhance security and control usage.

- **Load Balancer**

- Distributes incoming traffic across multiple servers to maintain performance.
- Performs health monitoring of servers to ensure reliability and availability.
- Provides failover support to minimize downtime in case of server failures.

3. Service Layer

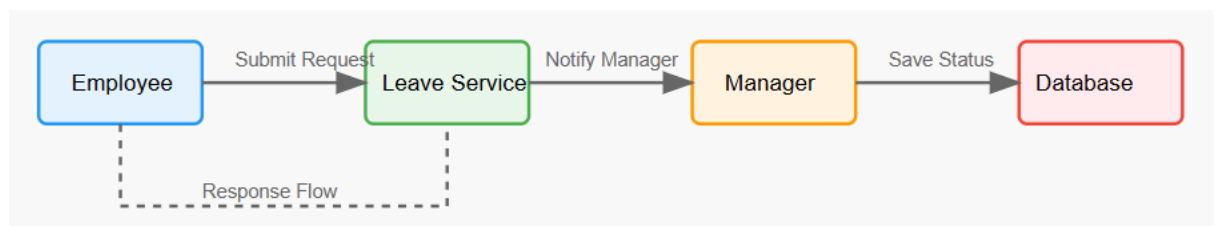


figure: Leave Application Flow in Service Layer.

This figure outlines the steps involved in the leave application process, from user initiation and validation to approval, notification, and final confirmation of leave status.

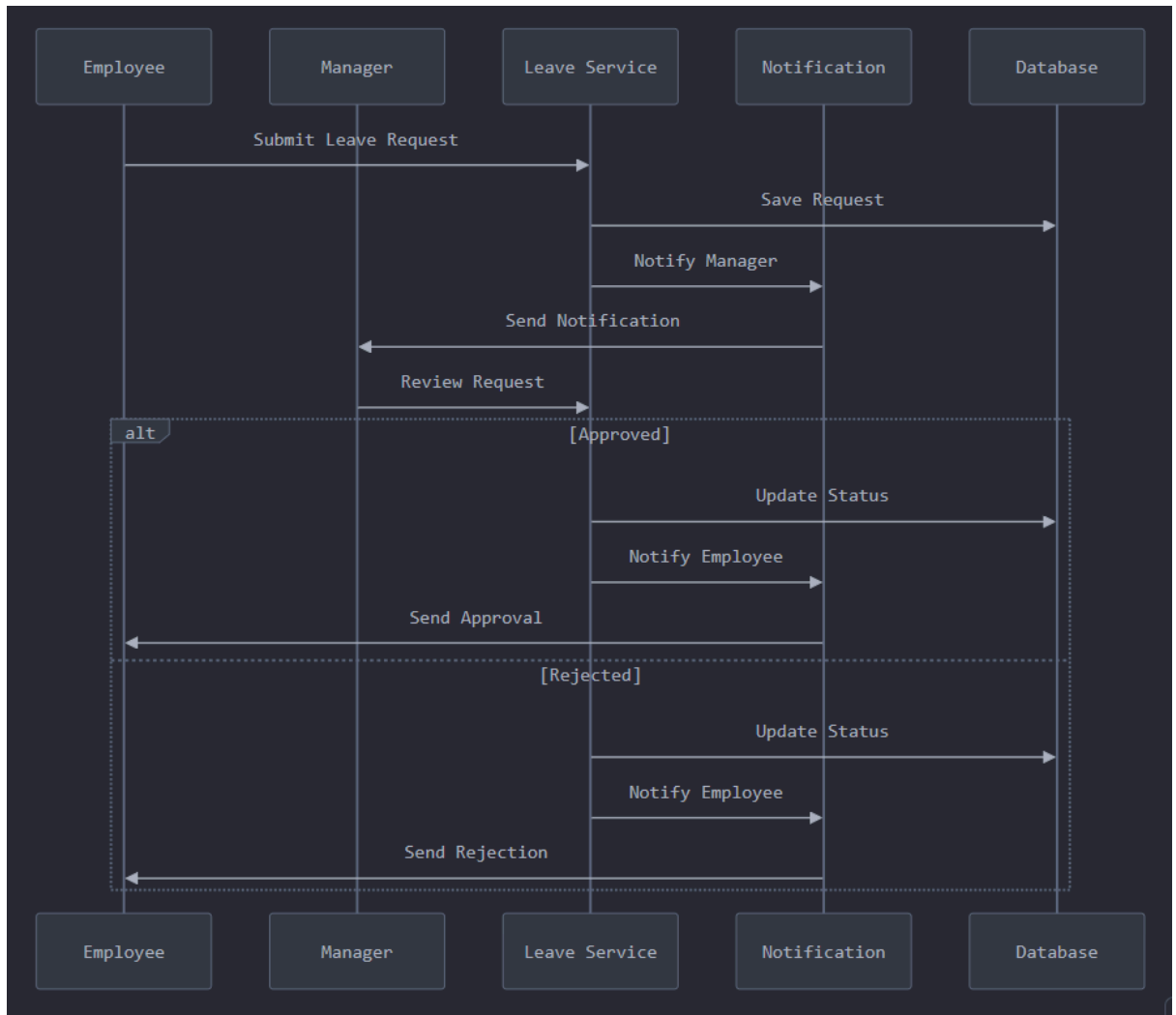


figure: Leave Application Process Details Flow in Service Layer

This figure details the flow of the leave application process within the service layer, highlighting the steps from user submission and validation of the leave request to approval workflows, notifications, and updates to leave balances, ensuring an efficient and organized handling of leave requests.

- **Authentication Service**

- Handles user authentication processes to ensure secure access.
- Manages user sessions for tracking active logins.
- Issues JSON Web Tokens (JWT) for secure communication between services.

- **User Service**

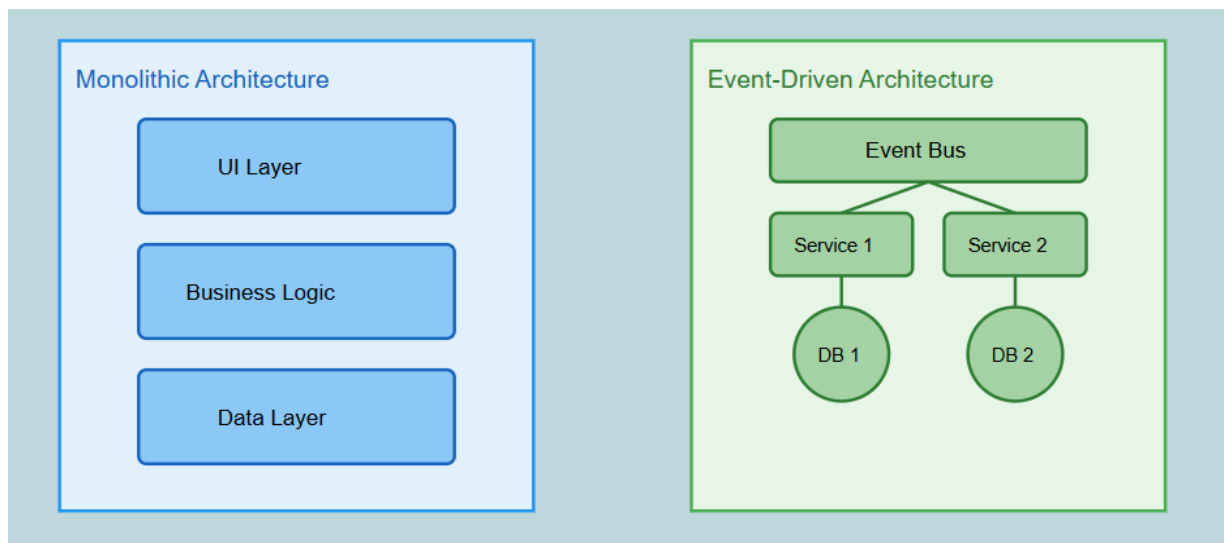
- Manages user profiles, allowing users to update their information.
- Handles role management to define user permissions and access levels.

- Controls access permissions to various system features based on user roles.
- **Leave Service**
 - Processes leave requests from users efficiently.
 - Manages approval workflows to ensure proper oversight and control.
 - Tracks leave balances for accurate user information.
- **KPI Service**
 - Monitors and tracks performance metrics for users and teams.
 - Updates leaderboards to foster competition and engagement.
 - Calculates rewards based on performance metrics for incentive programs.
- **Notification Service**
 - Sends email notifications for important updates and alerts.
 - Manages push notifications to keep users informed of relevant activities.
 - Handles system alerts to notify users of significant events or changes.

4. Data Layer

- **User Database**
 - Stores essential user information for account management.
 - Manages authentication data to facilitate secure login processes.
 - Tracks user roles to enforce access control measures.
- **Leave Database**
 - Stores leave applications and relevant documentation.
 - Tracks leave balances to ensure accurate reporting.
 - Maintains an approval history for auditing purposes.
- **KPI Database**
 - Stores performance metrics to support KPI tracking and analysis.
 - Maintains leaderboard data to reflect user standings.
 - Tracks reward points for performance-based incentives.
- **Cache Layer**
 - Improves response times for user requests by storing frequently accessed data.
 - Reduces the load on databases by caching session data and other relevant information.
 - Enhances overall system performance through efficient data retrieval.

3.2. Other Alternative Architectures Explored



1. Monolithic Architecture

- **Advantages:**
 - Simpler development workflow
 - Easier debugging and testing
 - Lower deployment complexity
 - Better performance for small applications
- **Disadvantages:**
 - Difficult to scale individual components
 - Higher risk of system-wide failures
 - Longer deployment cycles
 - Technology stack limitations

Description: In this design, all components of the system (UI, business logic, and data management) are tightly integrated into a single application. The entire system is deployed as a single unit, with no separation between the presentation, application, and data layers.

Why Not Chosen: While simpler to design and deploy, monolithic architectures have major drawbacks, such as difficulty in scaling, challenges in maintaining codebases, and the risk of system-wide failures. Updating or adding new features would require redeployment of the entire system, reducing flexibility and increasing downtime.

2. Event-Driven Architecture

- **Advantages:**

- Loose coupling between services
- Real-time data processing
- Better scalability for event-heavy systems
- Improved fault isolation
- **Disadvantages:**
 - Complex event handling
 - Challenging to maintain event consistency
 - Difficult to debug and test
 - Higher learning curve

Description: In an event-driven architecture, services communicate through events, allowing for asynchronous processing and real-time updates. This architecture is particularly effective for systems with high variability in workloads and user interactions.

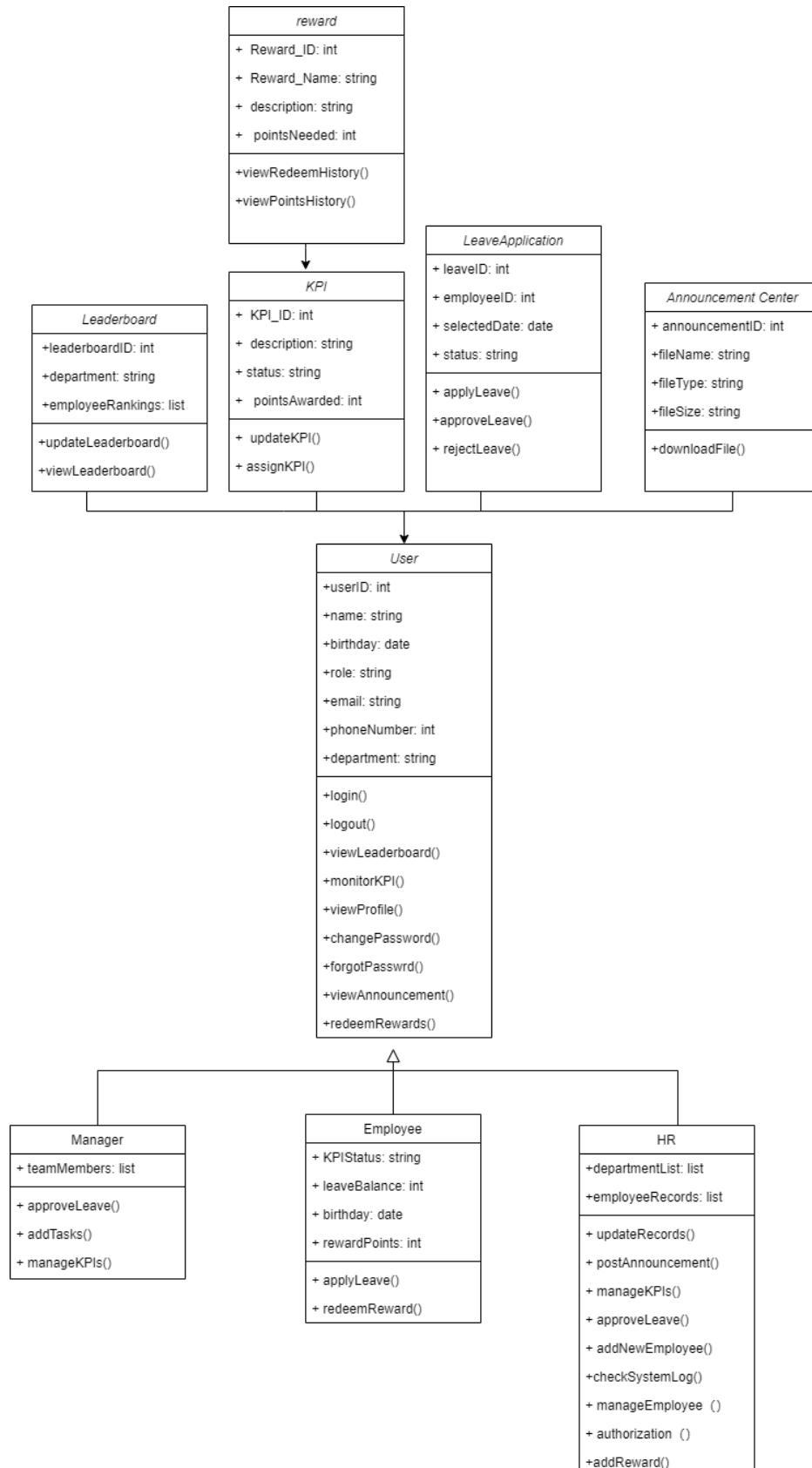
Justification for Chosen Architecture: The event-driven architecture was chosen because it provides:

- **Scalability:**
 - Allows for independent scaling of services based on demand.
 - Efficient handling of varying loads, making it suitable for high-traffic applications.
- **Flexibility:**
 - Services can be developed and deployed independently, enabling rapid feature releases.
 - Supports multiple technology stacks, allowing teams to choose the best tools for their services.
- **Resilience:**
 - Improved fault isolation reduces the impact of service failures on the overall system.
 - Enhances recovery processes, as services can operate independently and be restarted without affecting others.
- **Real-Time Processing:**
 - Facilitates real-time data processing, which is essential for applications requiring immediate feedback.
 - Enables proactive responses to user actions and events, enhancing user experience.

4. Detailed Design

4.1. The Detailed Design and Justification

Below shows the object oriented design of our system.



User (Base Class)

- **Attributes:** userID, name, role, email, phoneNumber, department
- **Methods:** login(), logout(), viewLeaderboard(), monitorKPI(), viewProfile(), changePassword(), forgotPassword(), viewAnnouncement(),redeem reward.
- **Design Role:** The User class is the core entity representing all types of users in the system, like Employees, Managers, and HR. It centralized shared attributes and common methods, such as login, logout, and basic functionalities accessible to all users.
- **Justification:** This class employs the Template Method pattern by providing a basic structure for shared actions while allowing subclasses to implement specific details. It promotes code reuse and consistency while avoiding code duplication across subclasses.

Employee, Manager, and HR Classes (Inherit from User)

- **Design Role:** These subclasses specialize the User class to cater to specific responsibilities. For instance, HR manages employee records, Manager handles KPIs and team tasks, and Employee focuses on rewards and leave balance.
- **Justification:** The use of inheritance here aligns with the Single Responsibility Principle (SRP) by ensuring that each subclass is responsible for a distinct set of responsibilities. Additionally, this design enables flexibility in handling role-specific functions and promotes scalability.

Manager (inherits from User)

- **Attributes:** teamMembers[]
- **Methods:** approveLeave(), addTasks(), manageKPIs()
- **Justification:** The use of **inheritance** here aligns with the **Single Responsibility Principle (SRP)** by ensuring that each subclass is responsible for a distinct set of responsibilities. Additionally, this design enables flexibility in handling role-specific functions and promotes scalability.

Employee (inherits from User)

- **Attributes:** KPIStatus, leaveBalance, birthday, rewardPoints
- **Methods:** applyLeave(), redeemReward()
- **Justification:** The Employee class inherits from User Class while introducing its own attributes such as KPIStatus, leaveBalance, birthday, and rewardPoints. These attributes are essential for tracking an employee's performance, leave eligibility, and rewards points. Employees interact with the

system through methods like `applyLeave()` to submit leave requests and `redeemReward()` to claim rewards based on accumulated points.

HR (inherits from User)

- **Attributes:** `departmentList[]`, `employeeRecords[]`
- **Methods:** : `updateRecords()`, `postAnnouncement()`, `manageKPIs()`, `approveLeave()`, `addNewEmployee()`, `checkSystemLog()`, `authorization()`, `addReward()`
- **Justification:** The HR class is a specialized extension of the User class for Human Resources personnel. It manages a list of departments and employee records while having methods for posting announcements, updating employee records, authorizing employees, managing KPIs, and handling leave requests. This extensive functionality enables HR staff to maintain comprehensive oversight and perform critical administrative tasks.

Leaderboard

- **Attributes:** `leaderboardID`, `department`, `employeeRankings[]`
- **Methods:** `updateLeaderboard()`, `viewLeaderboard()`
- **Design Role:** The Leaderboard class maintains rankings of employees within departments and provides methods to update or view the leaderboard.
- **Justification:** The Leaderboard Class is designed to handle the ranking of the employees based on their KPIs. Attributes such as `leaderboardID` and `department` help separate leaderboards by different departments, and `employeeRankings[]` keeps track of the employee ranking orders. The method `updateLeaderbord()` allows the system to update the rankings as the employees complete tasks and earn points, while `viewLeaderboard()` method enables the users to view their leaderboard standing.

KPI

- **Attributes:** `KPI_ID`, `description`, `status`, `pointAwarded`
- **Methods:** `updateKPI()`, `assignKPI()`
- **Design Role:** The KPI class is responsible for storing and managing KPI details, such as descriptions, status, and points awarded. It provides methods for updating and assigning KPIs.
- **Justification:** This class utilizes the Information Expert principle by consolidating all KPI-related knowledge and behavior, improving maintainability and traceability of performance-related data.

LeaveApplication

- **Attributes:** leaveID, employeeID, selectedDate, status
- **Methods:** applyLeave(), approveLeave(), rejectLeave()
- **Design Role:** This class handles leave applications submitted by employees and offers methods for approving or rejecting leaves.
- **Justification:** The LeaveApplication class focuses on its core task of managing leaves, which adheres to the Single Responsibility Principle (SRP). Additionally, it enforces Encapsulation by keeping leave-related data and logic self-contained.

Announcement Center

- **Attributes:** announcementID, fileName, fileType, fileSize
- **Method:** downloadFile()
- **Design Role:** This class manages announcement details such as file names, types, and sizes, and includes functionality for downloading announcements.
- **Justification:** The design adheres to the Single Responsibility Principle by focusing only on announcements, making it easy to extend or update communication-related features.

Reward Class

- **Attributes:** Reward_ID, Reward_Name, description, pointsNeeded
- **Methods:** viewRedeemHistory(), viewPointsHistory()
- **Design Role:** The Reward class manages reward data, like reward names, descriptions, and point requirements. It also provides methods for viewing reward histories and points accumulated by users.
- **Justification:** This class follows the Encapsulation principle by grouping reward-related data and operations, making it easier to update or extend the reward system without impacting other classes.

4.2. Design Verification

This section presents the approach to verifying the design by illustrating user scenarios proposed in the Software Requirements Specification (SRS) document. These scenarios are supported by the object-oriented design outlined above.

Use Case Scenario 1: User Management Process

Precondition: The HR is logged into the system and the new employee has provided necessary details to HR for account creation.

Steps:

- 1. The HR creates a new account for the new employee in the admin page with the details and the default password.**
 - a. Verification**
 - i. The addNewEmployee() is called, checking that all the details are correctly filled out (name, birthday, role, email, password, phoneNumber, department) and in proper format. If not, the system displays an error message. Upon successful creation, the employee details are saved in the database.
- 2. The HR edits the details of the users because it is wrongly input.**
 - a. Verification**
 - i. The manageEmployee() is called to modify the details of the users. The system validates all entered information, checking for completeness and correct formatting. Once successful, the system saves the updated information, and the changes are reflected on the page.
- 3. The HR deactivates or deletes the user account for employees who have left the company.**
 - a. Verification**
 - i. The manageEmployee() method is called to delete or deactivate the selected user account. The system confirms with the successful deletion or deactivation with a message, and the employee's account no longer appears in the system.
- 4. Once the HR completed creating a new account for the new employee, the new employee login with the email and default password provided by the HR.**
 - a. Verification**
 - i. The login() method is triggered, verifying that the employee enters both email and password. If either is incorrect, the system displays an error message prompting the user to re-enter their details. Upon successful login, the employee gains access to the system.
- 5. After first successful log in, the system prompts the new employee to change the default password for security purposes.**
 - a. Verification**
 - i. The changePassword() is called, checking that the requirements are met (e.g. length, character types). Once the password change is

confirmed, the new password is saved in the database, and the system displays a confirmation message.

6. The existing users login with their email and the updated password on the login page.

a. Verification

- i. The login() is called to ensure that the users enter both email and password correctly. If any information entered incorrectly, the system displays an error message. Upon successful login, users can access the system.

7. The existing users forgot their passwords and clicked on the “Forgot Password” to reset their password.

a. Verification

- i. The forgotPassword() is called, sending a password reset link to the user's registered email address. The system ensures that the user follows the reset process and successfully updates their password, which is then saved in the database.

Use Case Scenario 2: KPI Management Process

Precondition: The manager and HR have successfully logged in and have permission to manage KPIs. The employee has also successfully logged in.

Steps:

1. The manager or HR creates a new KPI by clicking on the “Create New Task” button.

a. Verification:

- i. The assignKPI() method in KPI Class is triggered, ensuring that the required fields (KPI name, description, dates, and employee assignment) are properly filled. If not, an error message will be displayed. Upon successful creation, all the details are saved in the database. The status of the KPI is automatically set to ongoing.

2. The manager or HR updates the details of the existing KPI by clicking on the “Edit” button.

a. Verification:

- i. The updateKPI() method in KPI Class is called to modify the KPI attributes. The system validates that the KPI exists and checks that the updated information is complete and in the correct format. The system saves the changes and reflects the updates.

3. **The manager or HR deletes a wrongly added KPI for an employee. The manager or HR clicks on the “delete” button.**

- a. **Verification:**

- i. The updateKPI() in KPI Class is called to delete the KPI. Upon successful deletion, the system confirms that the KPI is no longer listed in the employee's KPI records.

4. **The employee views and monitors the assigned KPIs in the dashboard.**

- a. **Verification:**

- i. The monitorKPI() is used to retrieve and display the KPIs details to the employees. The system ensures that the view is up-to-date with the latest progress and status of each KPI.

5. **The manager or HR updates the progress on the assigned KPI.**

- a. **Verification:**

- i. The updateKPI() method in the KPI Class is called to modify the progress or status of an existing KPI (e.g., from "ongoing" to "completed" or "delayed"). The system saves the changes and reflects the updated progress.

Use Case Scenario 3: Leaderboard

Precondition: The employee has successfully completed a task and received points.

Steps:

1. **Employees and other users access the leaderboard page to see how they rank against their peers.**

- a. **Verification:**

- i. The viewLeaderboard() method is triggered to fetch each user's total points from the database. The system sorts the users according to their total points in descending order. The sorted rankings are displayed on the leaderboard page.

2. **Each time an employee completes a task and earns points, the system recalculates the leaderboard rankings to reflect the updated scores.**

- a. **Verification:**

- i. The updateLeaderboard() method in Leaderboard Class is called. This method fetches the employee's total points and updates the leaderboard rankings accordingly. The system ensures that the points are recalculated correctly and the leaderboard reflects the new rankings in real time.

Use Case Scenario 4: Reward Management Process

Precondition: HR has accessed the reward management system and the employee has completed a task assigned with award points.

Steps:

- 1. The HR creates a new reward, entering all the necessary details (reward name, description, and point awarded).**
 - a. Verification**
 - i. The addReward() method is called, validating that the information is correctly filled in and meets the format requirements. If any of the information is incorrect or missing, the system displays an error message. Upon successful creation, the reward details are saved in the database and the rewards are reflected in both the admin page and user page.
- 2. Once the employee completes a task, the employee receives the awarded points specified in the tasks.**
 - a. Verification**
 - i. The updateKPI() method is triggered. Once the status is changed from 'ongoing' to 'completed', the system automatically updates the employee's reward points by adding the points earned from the completed task. The changes in total points are reflected in the page.
- 3. The employee views point history by clicking on the points icon on the reward page**
 - a. Verification**
 - i. The viewPointsHistory() method is called, fetching all records of points the employee has received from completed tasks. The system displays the point history details on the designated page
- 4. The employee redeems the rewards in the reward page.**
 - a. Verification**
 - i. The redeemRewards() method is called, ensuring that the employee has sufficient points to redeem the reward. The system deducted the required points from the employee's total point. The system then updates the employee's points balance and saves the changes in the database.
- 5. The employee views the redeemed reward history by clicking the 'View History' button on the reward page.**
 - a. Verification:**

- i. The viewRedeemHistory() method is called, which fetches all records of redeemed rewards. The system displays the redeemed rewards history on the designated page, ensuring that it is accurate and complete.

Use Case Scenarios 5: Leave Management Process

Precondition: The users (employee, manager and HR) have successfully logged into the account.

Steps:

1. **The employee enters the leave details such as leave type, selected dates, and reason for leave. Then, the employee clicks “Submit” to apply for leave.**
 - **Verification:**
 - i. The applyLeave() method in LeaveApplication Class is called to handle the submission. The system validates that all the inputs are in the correct format and the required fields are filled out. The 'status' attribute is automatically changed to “pending” upon successful submission of leave application. Upon successful submission, the LeaveApplication object (leaveID, selectedDate, endDate, status) is stored in the system, linking the leave request to the employee's employee ID (employeeID).
2. **Manager and HR were notified via email. Manager and HR view the pending leave application.**
 - **Verification:**
 - i. Only users with manager or HR roles (determined by the 'role' attribute in the User class) can access the list of leave applications. When reviewing, all the details of the leave application are displayed.
3. **The manager or HR either approves or rejects the leave application after reviewing the details.**
 - **Verification:**
 - i. The approveLeave() or rejectLeave() methods in the LeaveApplication class are called based on the actions taken. If the leave is approved, the approveLeave() method is called and the 'status' attribute changes to "approved". If the leave is rejected, the rejectLeave() method is called and the 'status' attribute changes to "rejected".

Use Case Scenario 6: Announcement

Precondition: There is an upcoming event or important update to be announced to all the users. The HR has logged into the admin account successfully to manage announcements. The employee and manager have also successfully logged into their account.

Steps:

- 1. The HR posts an announcement about an upcoming event in the announcement center. The HR clicks on the “New Announcement” button to create a new announcement post in the admin page.**
 - a. Verification:**
 - i. The postAnnouncement() is triggered, ensuring all the required fields (title, description, file upload) are properly filled. The system displays the error message if the requirements are not met. The system saves the announcement details in the database upon successful creation.
- 2. The HR edits the announcement post by clicking on the “edit” button.**
 - a. Verification:**
 - i. The postAnnouncement() method is called to modify the announcement details in the database. The system validates that the new information is complete and in the correct format before saving the changes. The updated announcement is reflected on the announcement page
- 3. The HR deletes the previous announcement post by clicking on the “delete button”.**
 - a. Verification:**
 - i. The postAnnouncement() method is called to remove the announcement from the database. The system confirms that the announcement is successfully deleted, and it no longer appears in the announcement center for any user.
- 4. After the HR posts a new announcement, all the users receive notification via email and they view the announcement in the announcement page.**
 - a. Verification:**
 - i. The viewAnnouncement() is called to display the announcement in the announcement page. The users can view all the details (title, description, file upload). If the announcement includes files, the downloadFile() is called when the users click to download the file.

5. Research and Investigations

The creation of the Employee Performance Management System (EPMS) involved detailed research across different areas such as business needs, system design, and the best technology to use. This research helped decide how to build the system, what tools to use, and how to implement its features.

5.1. Business Domain Analysis

The EPMS aims to automate HR tasks like tracking employee performance, managing leave, giving feedback, and making announcements. Its goal is to help employees grow, be more responsible, and increase productivity. Research suggests that HR software should be built in parts (modular) and be easy to expand (scalable). Chaieb & Saied (2024) explain that using microservices allows each part of the system to work independently and scale up when needed, making it ideal for large HR systems (Software Design and Res...). Similarly, Souza de Castro & Rigo (2023) found that microservices handle different workloads efficiently, making them suitable for applications like the EPMS(Software Design and Res...).

5.2. System Architecture

The EPMS uses a microservices architecture with four main parts: Client Layer, Gateway Layer, Service Layer, and Data Layer. Each part is designed to support smooth communication and data management. According to Seedat, Abbas & Ahmad (2023), microservices allow each service to work separately, which prevents the entire system from failing if one part has an issue (Software Design and Res...). Heffelfinger (2022) adds that this design makes it easy to update or add new features without affecting the whole system (Software Design and Res...).

Alternative Architectures Considered

Before choosing microservices, two other designs were considered: monolithic and event-driven. Monolithic architecture is easier to build but doesn't scale well and can cause more problems when things go wrong. Event-driven architecture is good for real-time updates but can be hard to manage. Microservices were chosen because they are flexible, can scale independently, and handle high traffic better, as supported by Chaieb & Saied (2024)(Software Design and Res...)(Software Design and Res...).

5.3. Technological Platforms

The technologies used to support microservices in the EPMS include Docker, Nginx, and JSON Web Tokens (JWT):

- **Docker** was selected to run each part of the system in its container, which makes deployment easier and allows for independent scaling. Souza de Castro & Rigo (2023) noted that Docker helps manage resources well and keeps services isolated, making it suitable for microservices(Software Design and Res...). Oracle (2024) also highlighted Docker's widespread use in microservices deployment(Software Design and Res...).
- **Nginx** is used as a reverse proxy and load balancer, managing requests and distributing traffic across servers. This helps the system handle large amounts of traffic efficiently. AppMaster (2022) pointed out that Nginx supports features like routing and load balancing, which improves performance(Software Design and Res...). Shape.host (2023) further emphasized its effectiveness in managing microservices within a Docker environment(Software Design and Res...).
- **JSON Web Tokens (JWT)** are used to secure communication between the client and server, ensuring data safety. JWTs are particularly useful for managing secure, stateless authentication, making them a good fit for the EPMS. Heffelfinger (2022) described how JWTs keep sensitive HR data secure in microservices(Software Design and Res...). Kaper.com (2021) also discussed JWT's role in securing microservices with OAuth2 integration(Software Design and Res...).

5.4. Programming Languages and Frameworks

Django for the backend, Vue.js for the frontend, and MySQL as the database:

- **Django** was chosen for its strong performance, ease of use, and support for secure backend development. It is popular for building scalable web apps and works well with microservices (Reintech 2024)(Software Design and Res...). Coders.dev (2022) also emphasized Django's effectiveness in microservices, making it a good choice for the EPMS backend(Software Design and Res...).
- **Vue.js** was selected for the frontend because it is flexible, easy to use, and helps create interactive user interfaces. Its component-based design aligns well with microservices, as mentioned by Evan You (2024)(Software Design and Res...). Codewithstein.com (2023) also highlighted how Vue.js integrates well with Django, making it suitable for EPMS(Software Design and Res...).
- **MySQL** was chosen as the database for its ability to handle structured data and complex queries efficiently. IEEE (2020) confirmed that MySQL is one of the most effective databases for managing large-scale applications(Software Design and Res...). Vectorsal.org (2023) also noted MySQL's scalability and performance in handling HR data(Software Design and Res...).

Summary

The research and choices for the EPMS focused on creating a system that is scalable, flexible, and secure. Microservices, along with technologies like Docker, Nginx, JWTs, Django, Vue.js, and MySQL, ensure that the system can handle high traffic and provide secure and efficient HR management.

6. References

AppMaster 2022, *Understanding Load Balancing in Microservices with NGINX*, AppMaster, viewed 27 October 2024, <https://appmaster.io/blog/load-balancing-microservices-nginx>.

Chaieb, M & Saied, M A 2024, *Migration to Microservices: A Comparative Study of Decomposition Strategies and Analysis Metrics*, arXiv.org, viewed 27 October 2024, <https://arxiv.org/abs/2402.08481>.

Heffelfinger, D R 2022, *Security with JSON Web Tokens*, Springer, viewed 27 October 2024, https://link.springer.com/chapter/10.1007/978-1-4842-8161-1_11.

Oracle 2024, *Docker Overview*, Docker Inc., viewed 27 October 2024, <https://www.mysql.com/news-and-events/events/mysql-summit-2024.html>.

Reintech 2024, *Django and Progressive Web Apps (PWA): A Beginner's Guide*, Reintech, viewed 27 October 2024, <https://reintech.io/blog/django-and-progressive-web-apps-pwa-beginners-guide>.

Souza de Castro, L F & Rigo, S 2023, *Relating Edge Computing and Microservices by Means of Architecture Approaches and Features, Orchestration, Choreography, and Offloading: A Systematic Literature Review*, arXiv.org, viewed 27 October 2024, <https://arxiv.org/abs/2301.07803>.

You, E 2024, *10 Years of Vue: The Past and the Future*, Vue Mastery, viewed 27 October 2024, <https://www.vuemastery.com/conferences/vuejs-live-2024/10-years-of-vue-the-past-and-the-future/>.

Coders.dev 2022, *Designing Microservices with Django: An Overview of Tools*, Coders.dev, viewed 27 October 2024, <https://www.coders.dev/blog/designing-microservices-with-django.html>.

Codewithstein.com 2023, *Combining Django and Vue.js - Everything You Need to Know*, Codewithstein, viewed 27 October 2024, <https://codewithstein.com/combining-django-and-vuejs-everything-you-need-to-know/>.

Kaper.com 2021, *Micro-services Architecture with OAuth2 and JWT*, Kaper, viewed 27 October 2024, <https://www.kaper.com/cloud/micro-services-architecture-with-oauth2-and-jwt-part-1-overview/>.

Shape.host 2023, *Load Balancing Microservices with Nginx in Docker Environment*, Shape.host, viewed 27 October 2024, <https://shape.host/resources/load-balancing-microservices-with-nginx-in-docker-environment/>.

Vectoral.org 2023, *Integrating AI and Microservices Architecture for Agile HR Operations*, Vectoral, viewed 27 October 2024, <https://vectoral.org/index.php/QJETI/article/view/155>.