University for Applied Sciences Informatics Department Applied Informatics

Theme: Small Warehouse Management and Rental Software for Universities

Documentation for the architecture of a Software for Management Warehouse

Manoel Jessica Noumsi Moguem
inf4173@hs-worms.de

Supervisor:	Prof. Dr. Volker Schwarzer
Working Period:	Winter Semester 2023/2024
Due Date:	14. January 2024

Contents

1 Introduction

1.1 Design Purpose

The design purpose section elucidates the key objectives of the project, with a focus on creating an efficient software solution tailored to the unique requirements of universities. Emphasis is placed on optimizing warehouse management and equipment rental while ensuring scalability and integration with existing systems.

ID	Category	Description
DP1	Purpose	• The purpose of the architectural design is to create an optimal software solution that meets the specific needs of universities in warehouse management and equipment rental. • Ensure a robust and scalable architecture that enables efficient management of inventory, rental transactions, and seamless integration with existing university systems. • Optimize the user experience by providing a user-friendly and intuitive interface while ensuring the security of sensitive data.
DP2	Business Goals	• Improve Operational Efficiency: Optimize inventory and equipment management to reduce operational costs associated with equipment loss and poor inventory management. • Increase User Satisfaction: Provide a user-friendly platform that simplifies the equipment rental process, thereby enhancing user satisfaction, including staff and students. • Enhance Data Security: Implement robust security mechanisms to protect user-sensitive information and ensure compliance with privacy standards. • Foster Scalability: Design an architecture that can evolve with the growing needs of universities, especially regarding the expansion of inventory and users.

ID	Category	Description
DP3	Implementation Timeline	• The implementation of the architectural design will commence after the detailed design phase is finalized. • Development will occur in multiple iterations, following an Agile approach, to ensure incremental deliveries and the flexibility to adjust the design based on feedback.

1.2 Primary Functionality

ID	Category	Description	Priority[1-10]
PF1	Inventory Management	Equipment Inventory	8
PF2	Inventory Management	Handle Lost / Broken Equipment	6
PF3	Warehouse Location Management	Chaotic Warehouse Location Management	7
PF4	Transactions and Equipment Rental	Equipment Checkout / Rental Functionality	5
PF5	Transactions and Equipment Rental	Reminder Function for Stakeholder / Equipment Renters	5
PF6	Efficient location management in the warehouse	Logical classification system for organizing items in the warehouse. /Location moving and updating functionalities. /Advanced search features to quickly locate items.	8

1.3 Stakeholders

Stakeholders are integral to the project's success, and this section introduces the key players. Students, administration, warehouse staff, and IT support each have distinct roles and motivations within the project, influencing its development and implementation.

Stakeholders	Description	Motivation
Student	Students are the primary end users of the system as they will use the platform to rent equipment. Their user experience and ease of use are crucial.	Students are motivated by a user-friendly interface, a simple and quick rental process, and the availability of a diverse range of equipment.
Administration	The university administration is responsible for the overall management of the institution, including administrative aspects. In the project context, the administration may be involved in making strategic decisions and defining priorities in inventory management and rental.	The administration is motivated by operational efficiency, cost reduction, and satisfaction of students and staff.
Warehouse Staff	Warehouse staff is responsible for the physical management of stocks, recording inventory movements, and preparing equipment for rental. They will be active users of the system.	Warehouse staff is motivated by the simplification of their daily tasks, reduction of human errors, and improvement of operational efficiency.
IT Support	The IT support team is responsible for the maintenance and technical support of the system. They may be involved from the implementation phase and will play a crucial role in issue resolution and system performance maintenance.	The IT support team is motivated by system stability, downtime reduction, and the ability to provide effective support to end users.

1.4 Constraints

Constraints present challenges and limitations that impact the project. This section delves into constraints at both the business and technical levels, offering insights into factors that need to be navigated during the project's lifecycle.

1.4.1 Constraints in the Business Context

This subsection delves into how the project aligns with the strategic goals of the university administration. It explores the business motivations, challenges, and expected outcomes.

ID	Constraints	Reasonning
CT1	Budgetary Constraints	Limitation on allocated financial resources for the project, impacting technology choices, feature scope
CT2	Academic Calendar	Constraints related to academic periods, such as exam periods and holidays, may influence project development and deployment timelines.
CT3	Compliance with Unuversity Policies	The project must adhere to established university policies, including data security, privacy, and other compliance considerations.

1.4.2 Constraints in the Technical Context

Here, the technical aspects of the project are discussed. This includes the technological infrastructure, programming languages, platforms, and server requirements.

ID	Constraints	Reasonning
CT1	Technological Constraints	Limitations imposed by existing technologies, such as supported operating platforms, database.
CT2	Scalability	The project must be scalable to accommodate future growth in the number of students, equipment.
CT3	Programming Language Constraints	The project may be required to use a specific programming language mandated by the university or aligned with existing technology stacks.
CT4	Platform Constraints	The project might need to run on specific platforms or frameworks dictated by university policies or existing infrastructure.

1.5 Quality Attributes

ID	Quality Attribute	Associated Driver	Priority	Motivation
QA1	Security	PF1	(H,H)	Security is crucial to protect sensitive information related to incidents, ensuring that only authorized personnel can access and manage such data. Robust security measures are essential to prevent unauthorized access and manipulation of the inventory, ensuring data integrity.
QA2-1	Usability	PF3	(M,H)	Usability is improved by providing warehouse staff with a logical and user-friendly system for organizing items, simplifying their daily tasks.
QA2-2	Usability	PF4	(H,M)	Usability is tied to user satisfaction, and efficient search features enhance the overall user experience by making it easy to find items quickly.
QA3	Availability	PF5	(H,H)	Availability is critical to ensure that reminders are sent in a timely manner, facilitating a smooth equipment rental process. Availability of location management functionalities ensures operational efficiency in the warehouse, preventing delays in locating items.
QA4	Performance	PF2	(H,H)	High performance in detailed tracking ensures a quick and accurate update of equipment status, essential for efficient stock management.

ID	Quality Attribute	Associated Driver	Priority	Motivation
QA5	Interoperability	PF4	(M,M)	Interoperability may be critical to ensure that the equipment rental feature can be integrated with other relevant university systems, such as financial systems or student identification systems.
QA6	Maintainability	PF6	(H,H)	This functionality involves managing the logical classification of items in the warehouse, which may evolve over time based on changes in the university's needs and the nature of the stored items. Maintainability ensures that this classification can be modified and updated without disrupting the entire system.

2 Quality Scenarios

ID	source	Stimulus	Artifact	Environment	Response	Response Measure
QA1-SC1	Authorized users, potential intruders.	Attempted unauthorized access, security attack.	User data, sensitive information.	System in regular use.	Identification and rejection of unauthorized access, security alerts.	Monitoring activity logs, success rate of identification.
QA1-SC2	Warehouse staff or university administration initiates the need to track and manage equipment inventory.	A Input of new equipment details, updates, or removals from inventory.	The small warehouse management and rental software.	University warehouse setting.	The system updates the equipment inventory database with a focus on the security of sensitive equipment data.	Accuracy and speed of inventory updates; adherence to security measures for sensitive equipment data.
QA2-SC1	System users.	Interaction with the user interface.	User interface, equipment rental processes.	Daily system usage by staff and students.	Intuitive navigation, user guides.	Transaction completion rates, positive user feedback
QA2-SC2	Warehouse staff struggling with disorganized equipment placement.	Requests for reorganization or difficulty in locating items.	The small warehouse management and rental software.	University warehouse setting.	The system provides tools for logical classification, location moving, and advanced search features to enhance organization and ease of access with a focus on usability.	Usability and efficiency in managing warehouse location chaos.

ID	source	Stimulus	Artifact	Environment	Response	Response Measure
QA3	Students or equipment renters who need reminders for return deadlines.	Requests for reminders or approaching return deadlines.	The small warehouse management and rental software.	University campus and external notifications.	The system sends reminders for equipment returns, ensuring availability for other users.	Availability and effectiveness of reminder notifications.
QA4	Reports from students or warehouse staff about lost or broken equipment.	Notification of lost or broken equipment.	The small warehouse management and rental software.	University warehouse and campus.	The system marks the equipment as lost or broken, triggering necessary actions for replacement or repair with an emphasis on performance.	Performance in promptly handling and updating lost/broken equipment records.
QA5	Integration with other university systems.	Data exchange with other applications.	Interoperability protocols, APIs.	Diverse university systems.	Validation and conversion of data formats.	Success rates of integrations, compatibility with standards.
QA6	University administration department.	Warehouse managers identify a need to restructure the classification of items due to a shift in the nature of stored equipment.	The warehouse management software.	University warehouse and campus.	The system allows for a modification of the logical classification of items, with a seamless update of the database and associated functionalities.	The modification of the classification is carried out without a negative impact on other functionalities, and the system remains operational during the update process.

3 First Overview

3.1 Business Context

The business context of the project is rooted in the growing need for universities to efficiently manage their material resources, including laboratory equipment, educational devices, and other assets. The demand for a warehouse management and rental solution arises from the desire to streamline operations, minimize equipment losses, and enhance accessibility for end-users such as staff and students.

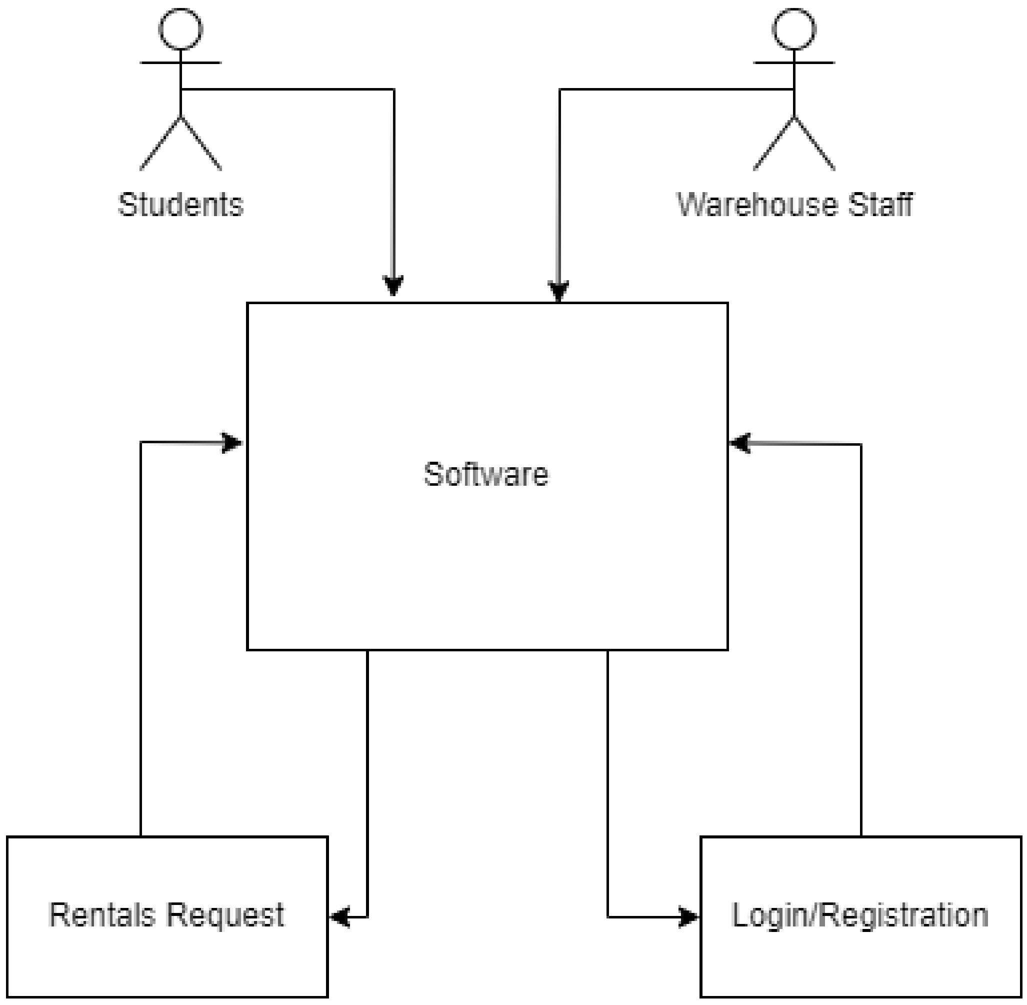


Figure 1: Business Context

3.2 Technical Context

The technical scope of the project encompasses judicious selection of software components, defining interfaces between modules, and establishing an architecture that promotes maintenance, scalability, and future integration. This technical context defines the terrain on which the architecture team will work to bring forth an efficient software solution tailored to the specific needs of universities.

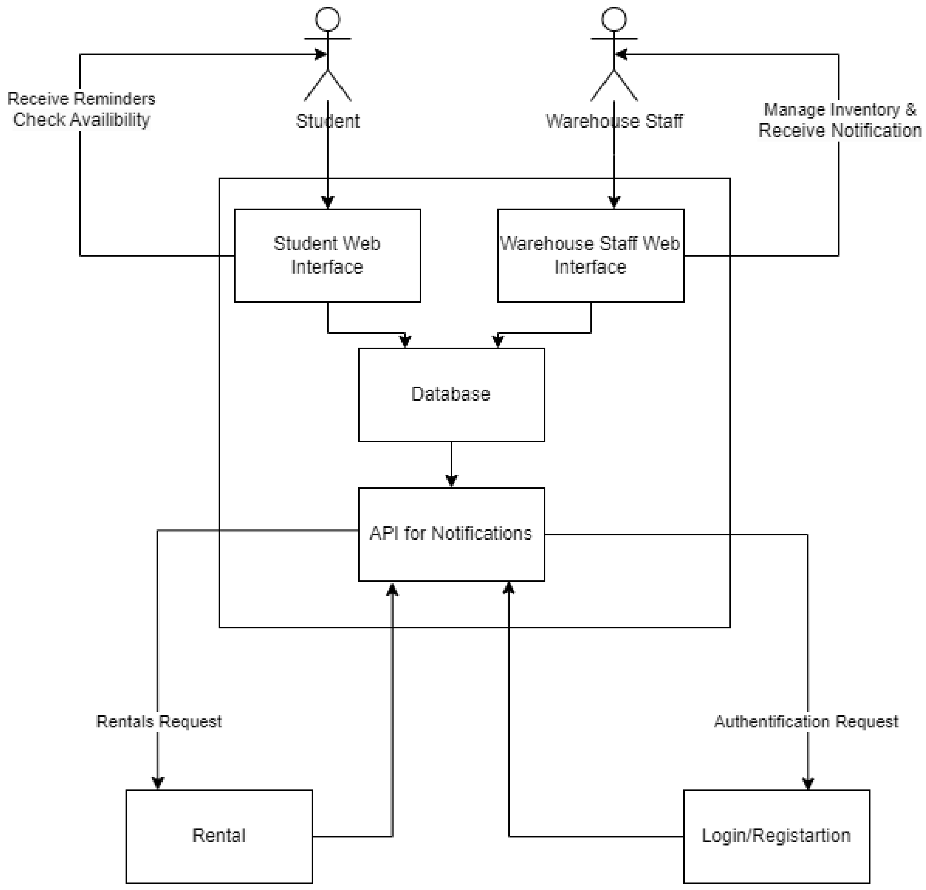


Figure 2: Technical context

4 Crosscutting Concept

The architecture of the project is shaped by carefully chosen tactics and patterns aligned with crucial quality attributes. The basics quality attributes are Performance, availability,maintanability, security and usability. Each quality attribute is addressed with specific strategies to ensure optimal performance, continuous availability, robust security, maintainability and exceptional usability. Let's delve into how these tactics and patterns are integrated to meet the specific requirements of each quality attribute.

4.1 Maintainability

In crafting a resilient and adaptive small warehouse management system for universities, the focal point lies in the art of maintainability. This section unveils the carefully selected tactics and patterns, such as "Increase Cohesion" and "Reduce Coupling," alongside architectural choices like the "Client-Server Pattern" and "Plug-in (Microkernel) Pattern," all strategically aimed at fortifying the software's ability to endure changes and enhancements over time.

Tactics	Patterns	Reasonning
Increase Cohesion	Pattern: Client-Server Pattern	Increasing cohesion is chosen to enhance maintainability by grouping elements of the warehouse management software that are closely related and share similar responsibilities. The client-server model is adopted to separate responsibilities between client and server components, improving system flexibility and scalability.
Reduce Coupling	Pattern: Plug-in (Microkernel) Pattern	Reducing coupling is chosen to minimize dependencies between software components, facilitating changes and updates without affecting the entire system. The plug-in model is chosen to enable easy extension of the system with new features, promoting maintainability.

4.2 Security

Ensuring robust protection against potential threats, our project employs security tactics such as detecting and resisting attacks. Intercepting and Intrusion Prevention System patterns are carefully integrated into our architecture, safeguarding sensitive data and ensuring the integrity of our warehouse management and rental software.

Tactics	Patterns	Reasonning
Detect Attacks	Intercepting	Timely detection of attacks allows for rapid response and mitigation, preventing unauthorized access and potential damage. Intercepting patterns involve intercepting requests or responses to enforce security policies, offering an additional layer of protection.
Resist Attacks	Intrusion Prevention System	Implementing measures to resist attacks, such as encryption and access controls, ensures the system's robustness against various security threats. IPS proactively monitors and analyzes system activities, preventing potential security breaches and safeguarding sensitive data.

4.3 Usability

In enhancing the overall user experience, our project focuses on tactics that support both user and system initiatives. This involves implementing user-friendly features supported by architectural patterns like Model-View-Controller (MVC) and Observer, fostering a design that prioritizes ease of use and responsiveness for stakeholders interacting with the software.

Tactics	Patterns	Reasonning
Support User Initiative	Model-View-Controller (MVC)	Empowering users by supporting their initiatives enhances usability, making the system more intuitive and user-friendly.The Model-View-Controller (MVC) pattern can contribute to usability by separating concerns and providing a clear structure for user interactions.
Support System Initiative	Observer	Providing guided interactions and system-initiated prompts ensures a smooth and user-centric experience, enhancing overall usability. The Observer pattern can be beneficial for real-time updates and notifications, supporting both user and system initiatives seamlessly.

4.4 Availability

For continuous and reliable operation, our project incorporates tactics like detecting and preventing faults. Redundancy patterns and Circuit Breakers are strategically employed to mitigate potential disruptions, ensuring that our warehouse management software remains available to users even in the face of unexpected challenges.

Tactics	Patterns	Reasonning
Detect Faults	Redundancy Patterns	Proactively identifying faults allows for quick response and resolution, minimizing downtime and ensuring continuous availability. Introducing redundancy patterns, such as backup servers or data replication, ensures that if one component fails, there's a backup to maintain system functionality.
Prevent Faults	Circuit Breaker	By implementing measures to prevent faults, the system becomes more resilient, reducing the likelihood of service disruptions and enhancing overall availability. Circuit breakers prevent cascading failures by automatically stopping requests to a failing service, allowing the system to recover and maintain overall availability.

4.5 Performance

In ensuring optimal system responsiveness and resource utilization for our project, we implement performance-centric tactics such as controlling resource demand and managing resources. Complemented by architectural patterns like Service Mesh and Load Balancer, our design aims to provide a seamless user experience even during peak usage.

Tactics	Patterns	Reasonning
Control Resource Demand	Service Mesh	In a university environment with fluctuating demands, controlling resource demand ensures optimal utilization of system resources, preventing bottlenecks during peak usage. Service mesh facilitates communication between microservices, enhancing performance by managing network traffic, load balancing, and providing essential functionalities like circuit breaking and retries.
Manage Resources	Load Balancer	Efficiently managing resources involves prioritizing critical operations, allocating resources based on demand, and ensuring a responsive system even under varying workloads. Load balancers distribute incoming traffic across multiple servers, preventing overloading of any single server, thus improving system responsiveness and resource utilization.

5 Decision Record

Title	Status	Context	Considered Drivers	Decision	Considered Alternatives	Consequences
DR1-Architecture and Component Design	Accepted	The need for a scalable and modular architecture for the small warehouse management and rental software.Requirements include Equipment Inventory Management, Lost/Broken Equipment Handling, Warehouse Location Management, Equipment Checkout/Rental, Reminder Function, and Logical Classification System.	Maintainability, PF6 Constraints, Stakeholders, Design Purpose	Adopt a Microservices Architecture for flexibility, scalability, and maintainability. Design components for each functionality: Inventory Management, Equipment Handling, Location Management, Checkout/Rental, Reminder, and Classification System.	Monolithic Architecture: Considered but deemed less flexible and scalable for evolving requirements.	Microservices enable independent development and deployment of each functionality. Components facilitate modular updates and maintenance. Improved modularity and maintainability.

Title	Status	Context	Considered Drivers	Decision	Considered Alternatives	Consequences
DR2-User Authentication Approach	Accepted	Ensuring secure access to the warehouse management system for students, warehouse staff, and administrators.	Security, user experience, PF1, PF3	Implement Token-Based Authentication using industry-standard protocols (e.g., OAuth 2.0 or JWT).	Session-Based Authentication: Considered but token-based chosen for scalability and statelessness.	Improved security with a centralized approach. Simplified authentication management. Consistent user experience across the system.

Title	Status	Context	Considered Drivers	Decision	Considered Alternatives	Consequences
DR3-Database Technology Selection	Accepted	The need for efficient and reliable data storage for equipment inventory, user information, and system logs.	Performance, availability, PF2, PF5	Choose a relational database for structured data (e.g., PostgreSQL) for equipment inventory. Utilize a NoSQL database (e.g., MongoDB) for flexible storage of user information and logs.	Solely Relational Database: Considered but NoSQL chosen for flexibility in user data and logs.	Structured data benefits from relational database consistency. Leveraged the strengths of PostgreSQL for relational data storage.

Title	Status	Context	Considered Drivers	Decision	Considered Alternatives	Consequences
DR4-Choice of Patterns	Accepted	Selecting architectural patterns to address quality attributes.	Performance, availability, security, usability	For Security: Implement Microservices Security Patterns, including OAuth 2.0 for authentication. For Usability: Apply Model-View-Controller (MVC) for a structured user interface. For Availability: Use Redundancy Patterns for critical microservices. For Performance: Implement Service Mesh and Load Balancer patterns.	Security: Considered API Gateway pattern but chose Microservices Security Patterns for granularity. Usability: Considered Single Page Application (SPA) but opted for MVC for separation of concerns. Availability: Considered relying solely on Load Balancer but chose redundancy for critical services. Performance: Considered Request-Response pattern but Service Mesh chosen for microservices communication.	Enhanced security with Microservices Security Patterns. Improved user interface structure through MVC. Increased system reliability with Redundancy Patterns. Optimized resource utilization and load balancing with Service Mesh and Load Balancer.

6 Building Block View

In this section we will describe the App using some elements of the 4+1 Architectural View Model. With this model we aim to target an understanding of all our main stakeholders.

6.1 Scenario View

This diagram illustrates the interactions between end-users and the system, showcasing the various actions they can perform and the functionalities available to them.

Perspective	Stage	Focus	Concerns	Artefact
End Users	Putting it alltogether	Understandability, Usability	Feature Decomposition	Use-Case Diagram

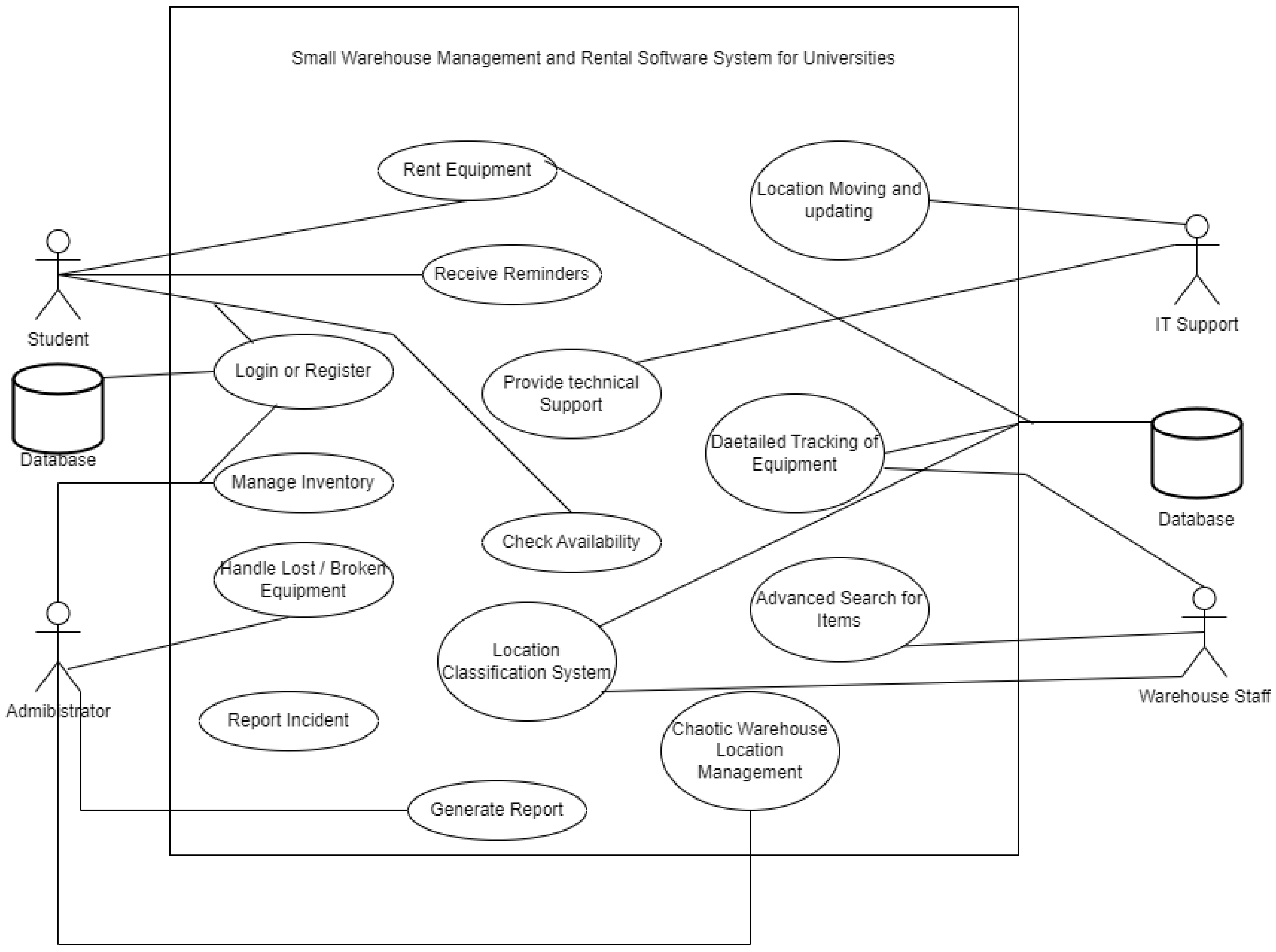

Small Warehouse Management and Rental Software System for Universities
Rent Equipment
Location Moving and updating
Receive Reminders
Student
IT Support
Login or Register
Provide technical Support
Database
Daetailed Tracking of Equipment
Database
Manage Inventory
Check Availability
Handle Lost / Broken Equipment
Advanced Search for Items
Location Classification System
Warehouse Staff
Admibistrator
Report Incident
Chaotic Warehouse Location Management
Generate Report

Figure 3: Scenario View

6.2 Structural View

This diagram provides a detailed representation of the system's structure, highlighting key entities and
their relationships, offering insights for designers into the logical organization of the software.

Perspective	Stage	Focus	Concerns	Artefact
Analysis, Designers	Requirement analysis	Object oriented decomposition	Functionality	Class Diagram

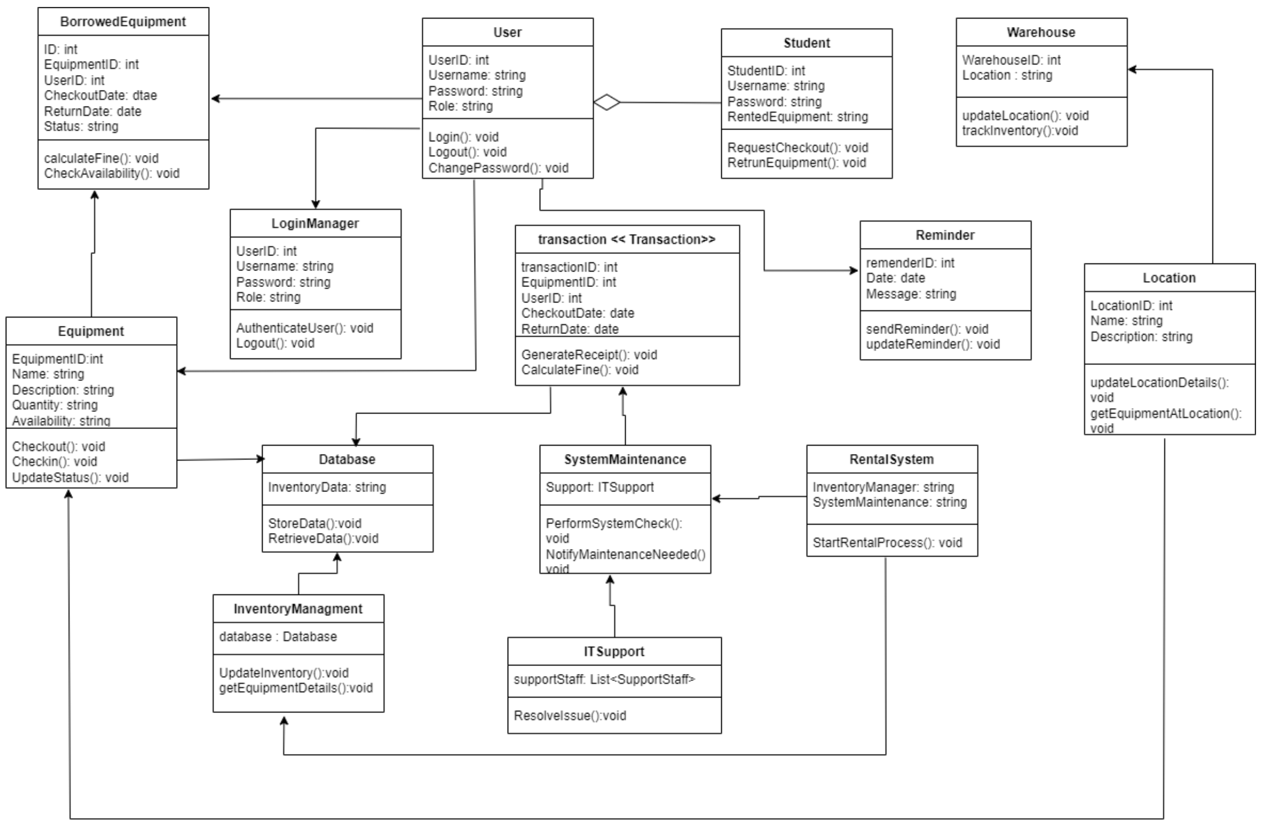


Figure 4: Structural View

6.3 Behavior View

This dynamic diagram illustrates the interactions between different components over time, aiding systems integrators in understanding the flow of activities within the system.

Perspective	Stage	Focus	Concerns	Artefact
System Integrators	Design	Process Decomposition	Performance, Scalability, throughput	Sequence Diagram

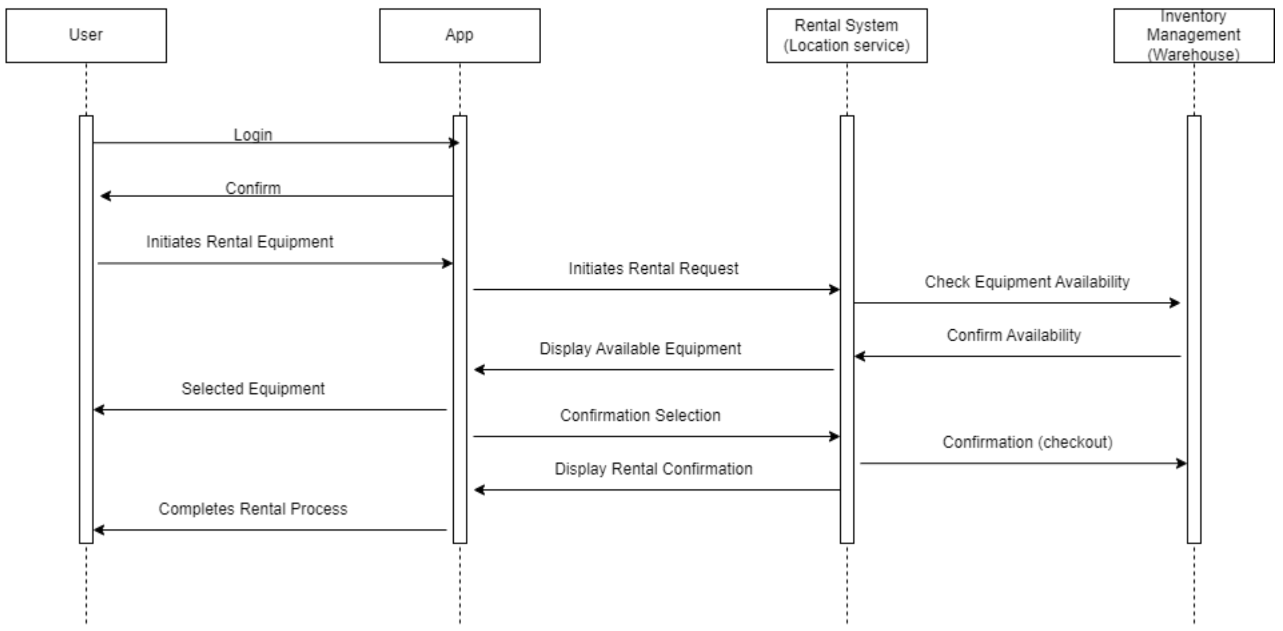

User
App
Rental System (Location service)
Inventory Management (Warehouse)
Login
Confirm
Initiates Rental Equipment
Initiates Rental Request
Check Equipment Availability
Confirm Availability
Display Available Equipment
Selected Equipment
Confirmation Selection
Confirmation (checkout)
Display Rental Confirmation
Completes Rental Process

Figure 5: Behavior View

6.4 Developer View

This diagram delineates the modular structure of the software, guiding developers in organizing and developing individual components for maintainability and flexibility.

Perspective	Stage	Focus	Concerns	Artefact
Developers	Design	Subsystem Decomposition	Software management	Component Diagram

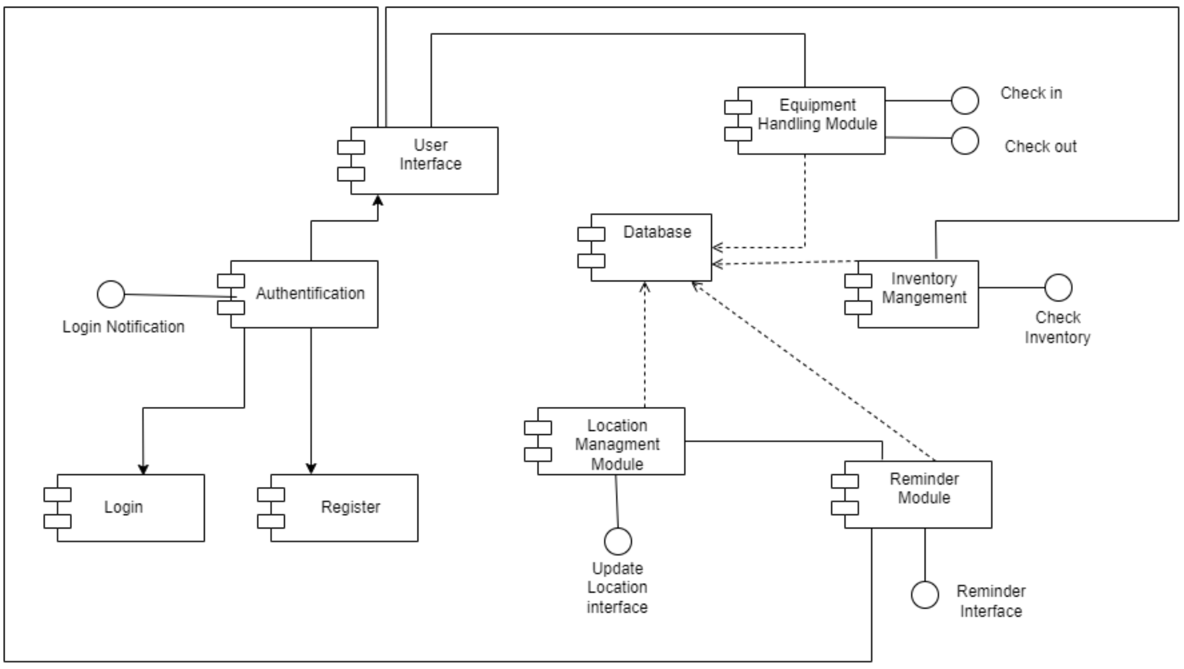
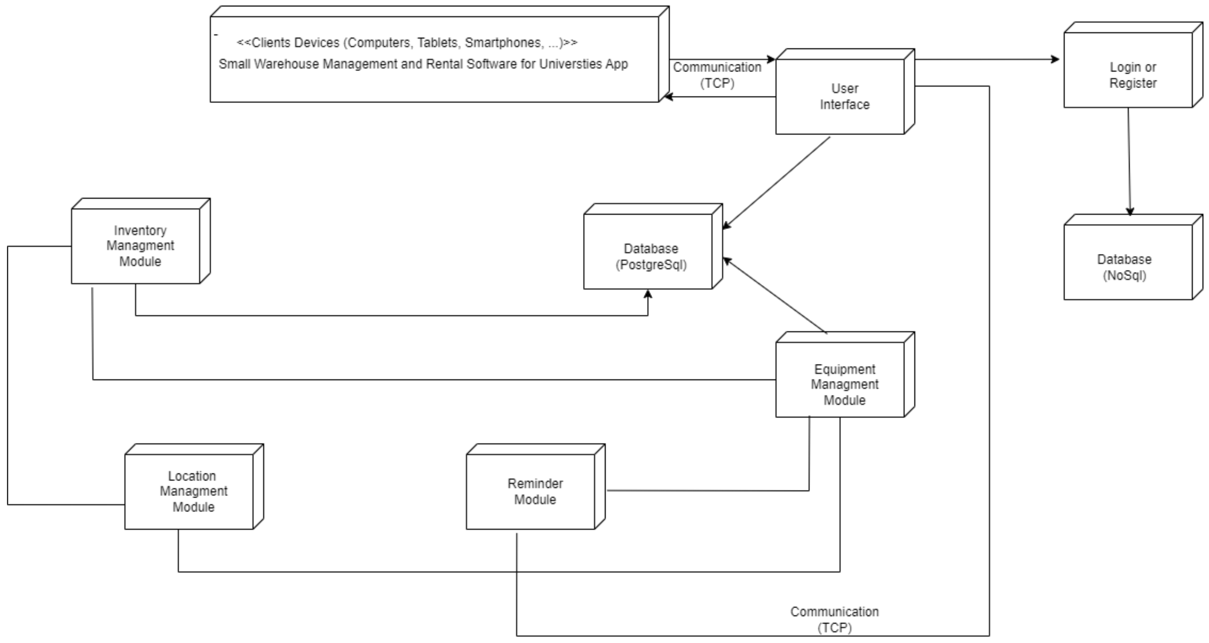


Figure 6: Developer View

6.5 Physical View

This diagram outlines the physical deployment of the software, detailing the distribution of components across different nodes, aiding system engineers in infrastructure planning and optimization.

Perspective	Stage	Focus	Concerns	Artefact
System Engineers	Design	Map software to hardware	System topology, deülivery, installation, communication	Deployment Diagram



<<Clients Devices (Computers, Tablets, Smartphones, ...)>>
Small Warehouse Management and Rental Software for Universties App
Communication (TCP)
User Interface
Login or Register
Database (NoSql)
Inventory Managment Module
Database (PostgreSql)
Equipment Managment Module
Location Managment Module
Reminder Module
Communication (TCP)

Figure 7: Developer View

7 Final Architecture

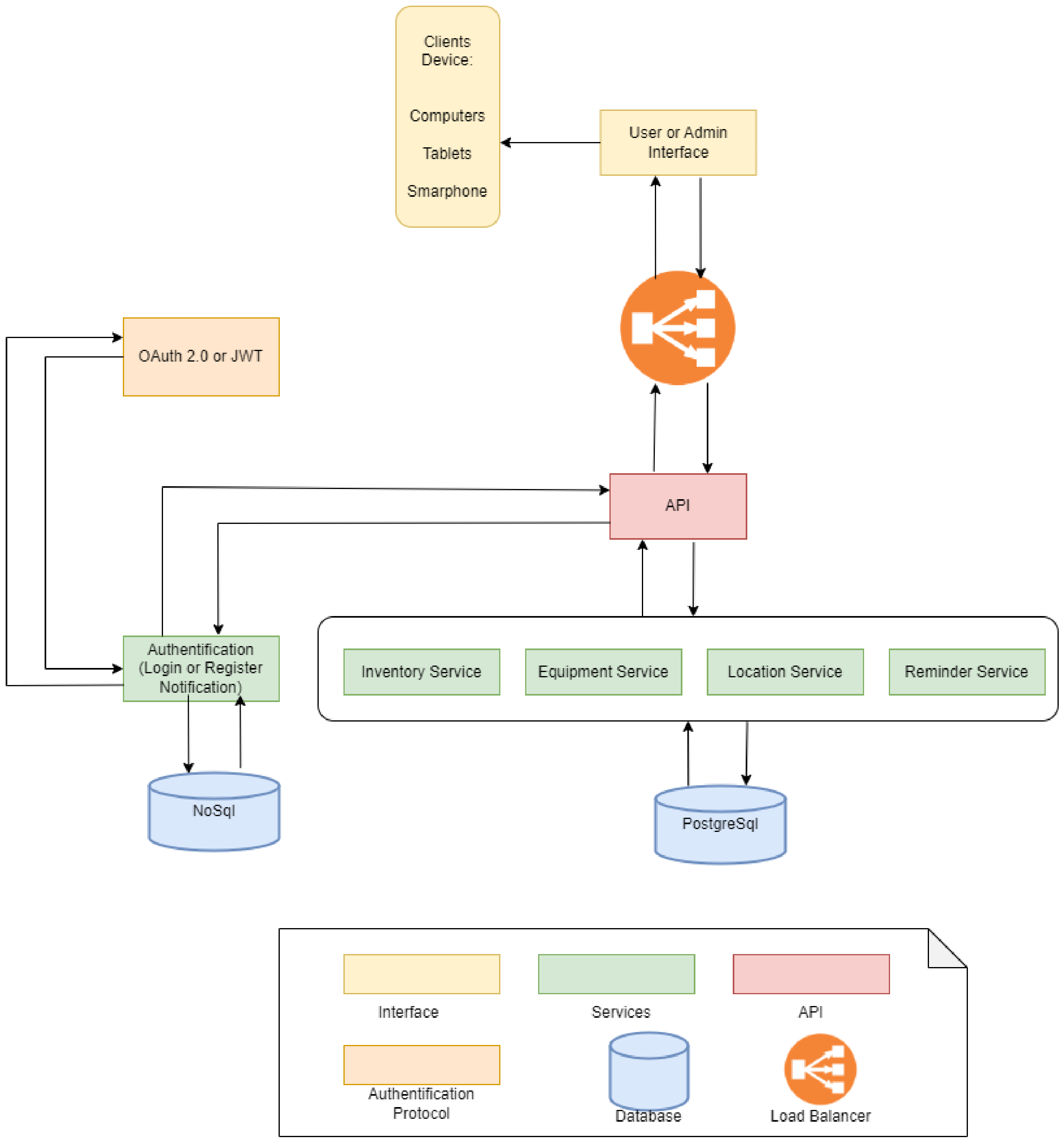


Figure 8: Final Architecture

8 Glossary

Application Programming Interface (API) Software intermediary that promotes the com- munication between different systems/applications/ softwares [MuleSoft (2015)].
Activity Diagram This kind of diagram shows the behavior of a system, it depicts in a graph- ical fashion the logic of a single use case [Baresi (2009)].
Class Diagram This kind of diagram presents the structure of a system with its classes, attributes, methods and relationships [IBM (2004)].
Sequence Diagram This kind of diagram presents the interaction of the elements over time as the communication fows from one direction to the other. It also displays which objects communicate with each other and the request that starts the process [Sparxsystems(2004)].
Use Case Diagram This kind of diagram resents the main requirements and functionalities of a systems. It displays a simplifed overview of core purpose of the application [Waykar (2015)].
A UML deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them.
Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering.

9 Abreviations

API Application Programming Interface.
MVC Model-View-Controller.
QA Quality Attributs.
PF Primary Functionality.
CT Constraints.
DP Design Purpose.
SC Scenario.
H High, **M** Medium, **L** Low.

10 References

- https://de.search.yahoo.com/yhs/search?p=ibm+2004+
- https://en.wikipedia.org/wiki/Architectural$_{pattern}$
- https://developer.ibm.com/articles/an-introduction-to-uml/
- https://microservices.io/
- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/
- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/
- The Courses in Class and the textbook “Software Architecture in Practice”

11 Questions

1- Which structures is used?
Component Connector structures focus on the way the elements interact with each other at runtime to carry out the system's functions. They describe how the system is structured as a set of elements that have runtime behaviour (components) and interactions (connectors).
- Service structures: The units here are services that interoperate through a service coordination mechanism, such as messages.
- PostgreSQL is designed to be extensible, allowing it to scale with the growing needs of your system.
- PostgreSQL is compliant with SQL standards, ensuring consistency in database queries and operations.
- PostgreSQL integrates well with other technologies and programming languages, making it easy to integrate with other components of your system, including the aforementioned microservices.
-One of the key features of microservices architectures is their ability to isolate services from each other, thereby minimizing the impact of a failure on the entire system.
- The use of circuit breakers can be implemented to automatically detect failures in a service and temporarily prevent requests from being directed to it.
- if database pb?
- Backup and Restore Strategies: Implement regular backup strategies for the database. In the event of a major issue, restoring from a backup can be an option for a quick recovery.
- Implementation of Circuit Breakers:
Integrate circuit breaker mechanisms for queries to the database. This can prevent occasional failures in the database from affecting the entire system by avoiding excessive queries during a failure period.