table of contents

Template :revnumber: 8.2 EN :revdate: January 2023 :revremark: (based upon AsciiDoc version) :toc-title: Table of Contents

About arc42

arc42, the template for documentation of software and system architecture.

Template Version {revnumber}. {revremark}, {revdate}

1. Introduction and Goals

Housing system is an administrative system for a landlord that rents houses. It’s main goals are to facilitate the landlord to administrate the followings: * rental objects * tennants * service charge billings The project should also give the tennants a portal to check their daily consumptions of heating energy and water. …

This is a project of Alexander Rothschild, Hantao Zhou, Marco Campione.

1.1. Requirements Overview

Contents

The main purpose of Housing system is keeping track of all the rental objects, tennants informations, billings, consumptions and sensor management so that the landlord has all the informations updated. Another purpose is to create a portal for the tennants to pat for the rent and the services

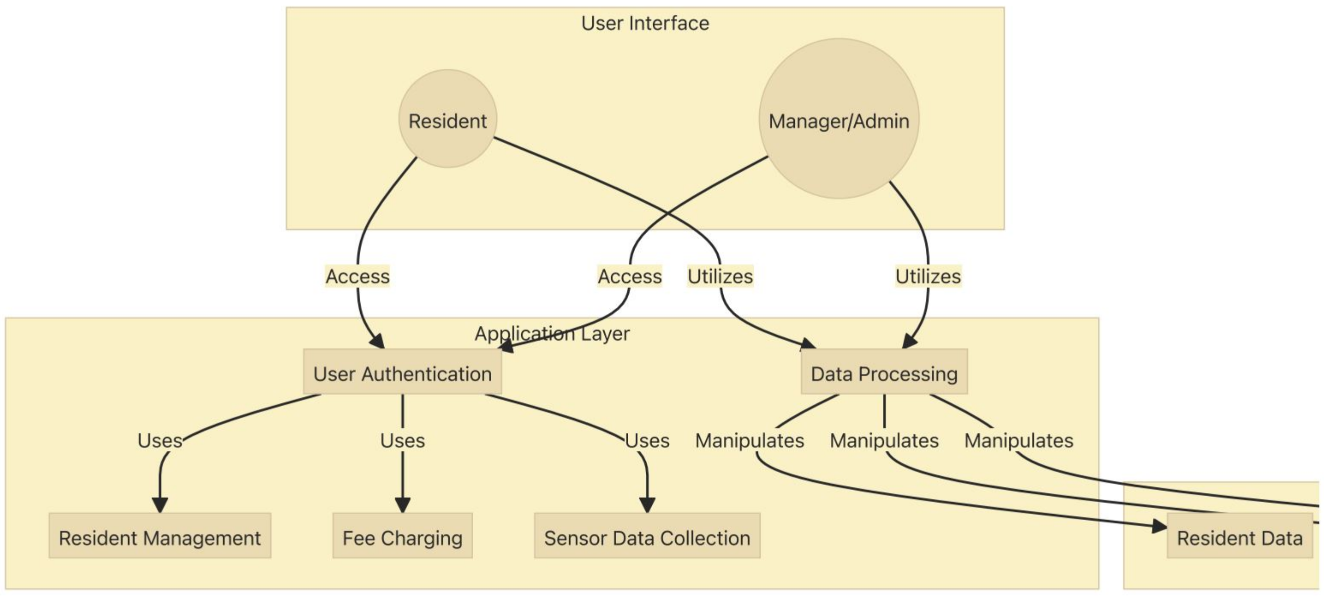
Motivation

The landlord asked for a system that helps him to keep track of everything he think it’s important for the administration of his rental objects. He asked for this project because right now he is working with Excel sheets and he is keeping every information updated "manually". This project aims to shift this manual work to an automated system.

1.2. Basic ideas and Brainstorming

A software design architecture to process all the info related to heat fee, including functions of collecting info from sensors, managing when the resident moves out, charging fees etc. Interact mainly with two types of user, resident as normal users and managers and system maintainers as admins.

We used mermaid to create a diagram Create a coarse boxes and arrows diagram (only most basic components) to give yourself a better first overview of the system’s requirements.



Use Case ID and Name	**UC001 - Record and Charge Heating Fees**
Actors	Tenant, Property Manager
Description	This use case involves recording and charging heating fees for rented houses. The system tracks the usage of heating resources in a house and calculates the corresponding fees for the tenant.
Preconditions	1. Tenant has an active rental agreement. 2. The heating system is installed and operational.
Postconditions	1. Heating fees are recorded for the specified period. 2. Tenant receives a bill for the heating charges.
Invariants	None
Basic Workflow	1. Tenant adjusts the heating preferences. 2. System records heating resource consumption. 3. System calculates heating fees based on usage. 4. Tenant receives a bill for the heating charges.
Alternative Workflow	- If the heating system malfunctions, notify property manager and suspend fee calculation until the issue is resolved. - If tenant disputes the heating charges, initiate a review process and adjust fees if necessary.
Risks	1. Malfunction of heating system. 2. Disputes between tenant and property manager regarding fee calculation.
Quality Goals	1. Accuracy in calculating heating fees. 2. Timely generation and delivery of heating fee bills.
Special Requirements	1. The system should support multiple methods of heating (e.g., central heating, electric heating) for accurate fee calculation. 2. Heating fee bills should be clear and itemized for easy understanding by the tenant.

Use Case ID and Name	UC001 - Record and Charge Heating Fees
Assumptions	1. Tenants are responsible for their heating charges as per the rental agreement. 2. The heating system is equipped with meters or sensors to measure consumption accurately.

Use Case ID and Name	UC002 - Register New Tenant
Actors	Landlord, New Tenant
Description	This use case involves the landlord registering a new tenant in the system when a property is rented out to a new occupant.
Preconditions	1. The landlord is logged into the property management system. 2. The property is available for rent.
Postconditions	1. The new tenant’s information is added to the system. 2. The rental agreement is generated and stored.
Invariants	None
Basic Workflow	1. Landlord logs into the property management system. 2. Landlord accesses the ``New Tenant Registration'' feature. 3. Landlord enters the new tenant’s information (name, contact details, etc.). 4. Landlord specifies the terms of the rental agreement (duration, rent amount, etc.). 5. System validates the information provided. 6. System generates a unique identifier for the new tenant and stores their details. 7. System generates a rental agreement document. 8. Landlord reviews and confirms the registration.
Alternative Workflow	- If the information provided is incomplete or invalid, the system prompts the landlord to correct the details. - If the rental agreement terms need negotiation, the landlord can enter a negotiation phase with the new tenant.
Risks	1. Incomplete or inaccurate tenant information. 2. Miscommunication regarding rental agreement terms.
Quality Goals	1. Accurate and complete recording of tenant information. 2. Efficient generation and storage of rental agreements.

Use Case ID and Name	UC002 - Register New Tenant
Special Requirements	1. The system should support the attachment of legal documents related to the rental agreement. 2. Notifications should be sent to both the landlord and the new tenant upon successful registration.
Assumptions	1. The landlord has the legal authority to rent out the property. 2. The property is in a suitable condition for occupancy.

Use Case ID and Name	UC003 - Handle Tenant Rent Delay
Actors	Property Manager, Tenant
Description	This use case involves the property manager addressing a situation where a tenant is delaying on paying rent. The goal is to initiate communication and resolve the issue in a timely and fair manner.
Preconditions	1. Tenant has not paid the rent on the due date. 2. Property manager is logged into the property management system.
Postconditions	1. Communication with the tenant regarding the rent delay is documented. 2. A plan for resolution is agreed upon, which may include late fees or a revised payment schedule.
Invariants	None
Basic Workflow	1. Property manager receives a notification or identifies that a tenant’s rent is overdue. 2. Property manager accesses tenant information in the property management system. 3. Property manager initiates communication with the tenant through the system, inquiring about the reason for the delay. 4. Tenant responds with the reason for the delay. 5. Property manager reviews the situation and determines appropriate actions, which may include imposing late fees or negotiating a new payment schedule. 6. Property manager updates the system with the details of the communication and any agreed-upon resolution. 7. If the issue persists, the property manager may escalate the matter following the established protocol.

Use Case ID and Name	UC003 - Handle Tenant Rent Delay
Alternative Workflow	- If the tenant provides a valid reason for the delay (e.g., unexpected financial hardship), the property manager may work with the tenant to establish a temporary solution. - If the tenant is unresponsive, the property manager may escalate the issue by sending formal notices or involving legal channels as per the rental agreement.
Risks	1. Miscommunication between the property manager and tenant. 2. Tenant disputes regarding late fees or resolution terms.
Quality Goals	1. Timely and clear communication regarding rent delays. 2. Fair and consistent application of policies for resolving rent delays.
Special Requirements	1. The system should support the documentation of all communication related to rent delays. 2. Notifications to both parties should be clear and provide relevant information.
Assumptions	1. The rental agreement includes terms and policies regarding rent payments and late fees. 2. Both parties are expected to communicate through the system for transparency.

Use Case ID and Name	UC004 - Update System Security
Actors	System Administrator
Description	This use case involves the system administrator updating the security measures of the system to mitigate potential risks and ensure the protection of sensitive information and resources.
Preconditions	1. The system administrator has proper access rights. 2. Identified security vulnerabilities or a routine security update schedule.
Postconditions	1. The system’s security measures are updated. 2. Documentation of the security update is recorded.
Invariants	None

Use Case ID and Name	UC004 - Update System Security
Basic Workflow	1. System administrator identifies the need for a security update, either through routine checks or the discovery of vulnerabilities. 2. System administrator logs into the system with appropriate credentials. 3. System administrator accesses the security settings and configurations. 4. System administrator applies the necessary updates, patches, or configuration changes to address identified vulnerabilities or enhance security. 5. System administrator tests the updated security measures to ensure they do not disrupt system functionality. 6. System administrator documents the details of the security update, including the changes made and any testing outcomes. 7. If the update is successful, the system administrator notifies relevant stakeholders about the security enhancement.
Alternative Workflow	- If the security update requires system downtime, the system administrator coordinates with relevant parties to minimize disruption. - If the update reveals unforeseen issues or conflicts, the system administrator may need to roll back the changes and investigate the cause before reapplying the update.
Risks	1. Potential system downtime during the update. 2. Unforeseen issues or conflicts arising from the security update.
Quality Goals	1. Minimize system downtime during security updates. 2. Ensure that security updates do not introduce new vulnerabilities.
Special Requirements	1. The system should support rollback mechanisms in case of issues with the security update. 2. Detailed documentation of security updates should be maintained for audit and compliance purposes.
Assumptions	1. The system administrator has a thorough understanding of the system’s architecture and security requirements. 2. Relevant stakeholders are informed about the scheduled security update.

Use Case ID and Name	UC005 - Handle Broken Heat Sensor
Actors	Tenant, Maintenance Personnel
Description	This use case involves the process of identifying and resolving a broken heat sensor in a rented home to ensure the proper functioning of the heating system.

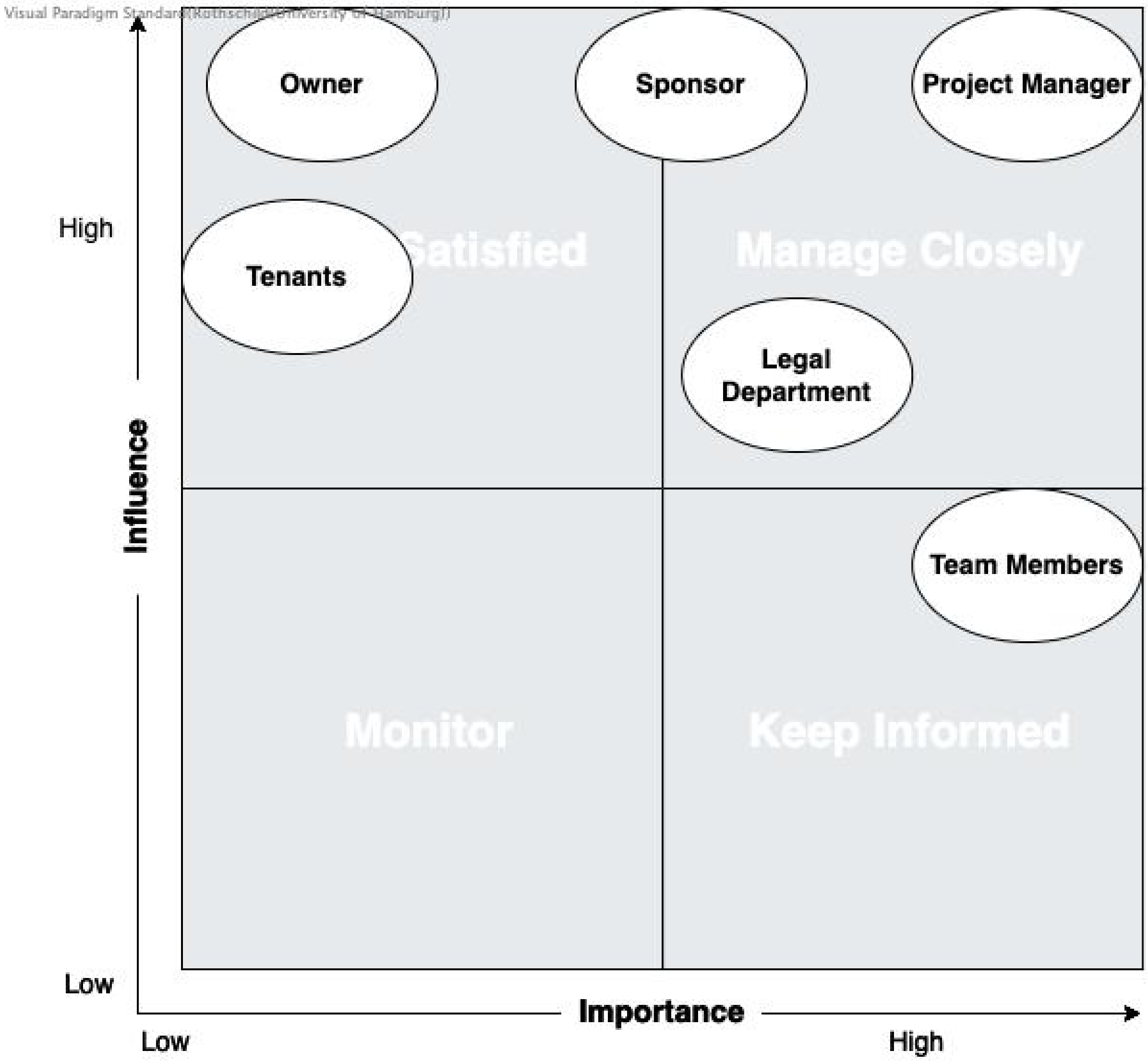
Use Case ID and Name	UC005 - Handle Broken Heat Sensor
Preconditions	1. Tenant notices an issue with the heating system or reports a lack of accurate temperature readings. 2. Maintenance personnel have access to the home and the necessary tools for sensor replacement.
Postconditions	1. The broken heat sensor is replaced. 2. The heating system is functioning correctly.
Invariants	The temperature readings from the heat sensor must be accurate and within an acceptable range.
Basic Workflow	1. Tenant notices a discrepancy in the temperature readings or experiences issues with the heating system. 2. Tenant reports the issue to the property management system. 3. Property management system logs a maintenance request and notifies maintenance personnel. 4. Maintenance personnel schedule a visit to the home. 5. Maintenance personnel assess the heat sensor and confirm it is malfunctioning. 6. Maintenance personnel replace the broken heat sensor with a new one. 7. Maintenance personnel test the heating system to ensure it is functioning correctly. 8. Maintenance personnel update the property management system with details of the resolution. 9. Tenant is notified that the issue has been resolved.
Alternative Workflow	- If the broken heat sensor is under warranty, maintenance personnel may contact the sensor manufacturer for a replacement. - If the replacement of the heat sensor requires a specialized technician, maintenance personnel may coordinate with external service providers.
Risks	1. Delay in addressing the issue may lead to discomfort for the tenant. 2. Unavailability of the required replacement parts or sensors.
Quality Goals	1. Timely resolution of heating system issues. 2. Accurate and reliable temperature readings after sensor replacement.
Special Requirements	1. The property management system should efficiently log and track maintenance requests. 2. Maintenance personnel should have access to replacement parts and sensors as needed.

Use Case ID and Name	**UC005 - Handle Broken Heat Sensor**
Assumptions	1. Tenants are prompt in reporting issues with the heating system. 2. Maintenance personnel are adequately trained to handle sensor replacements.

1.3. Stakeholders

Role/Name	Contact	Expectations
Customer	*<Landlord>*	1. What is your business? What are you doing? Answer: landlord/house owner 2. What challenges you want to solve throughout the project? Answer: Automation of the processes, integrate billing system, management of the sensor system, etc … 3. Are there any existing systems that need to be integrated? Answer: excel sheets (not sure) 4. What is your product and for whom is it? Who is your target audience? Answer: the product is mainly for him, there is no need to generalize to a wider target audience 5. Is it important for you what technologies are going to be used by us? Answer: Didn’t seem that the customer cared about the technologies 6. What are the project risks for your company? Answer: He hasn’t said anything about it 7. What are the deadlines? Till what time what should be ready? Answer: He hasn’t said anything about it 8. If it is a REST app do your have the REST endpoints or should we develop them on your side? Answer: He hasn’t said anything about it 9. What is the projects budget? Answer: He hasn’t said anything about it 10. How many people are there in the project and how many you need? Answer: He hasn’t said anything about it 11. Are there any software architecture constraints that we should know about (architectural pattern or design pattern)? Answer: He hasn’t said anything about it 12. Ask about that: the building block view shows the static decomposition of the system into building blocks (modules, components, subsystems, classes, interfaces, packages, libraries, frameworks, layers, partitions, tiers, functions, macros, operations, data structures, …) as well as their dependencies (relationships, associations, …) Answer: He hasn’t said anything about it 13. Make sure that we know: * important use cases or features: how do building blocks execute them? * interactions at critical external

1.3.1. Stakeholder Influence and Importance Matrix



1.3.2. Stakeholder Interest and Impact Table

Stakeholder	Interests	Estimated Project Impact	Estimated Priority
Owner	*Achieve targets* *Avoid liability*	*Med +* *High-*	*1*
Sponsor	*Save money through an all-in-one service instead of filling excel sheets and processing payments separately*	*Med +*	*4*
Project Manager (Lecturer)	*Keeping owner/sponsor and team members satisfied with the project in general through providing a connection between them*	*High +*	*2*
Team Members	*New product excitement* *Pass an exam in the semester end*	*Med +* *Med +*	*5*
Tenants	*Want to have a comfortable online service*	*Low-*	*6*

Stakeholder	Interests	Estimated Project Impact	Estimated Priority
Legal Department	*Wants everything to be according to the law and/or regulations*	*High-*	*3*

1.3.3. Interest-Influence Classification

Stakeholder	Estimated Project Influence	Estimated Project Importance	Assumptions and Risks
Owner	*High (10)*	*Low (2)*	_Providing all the resources, but his requirements a bit vague _
Sponsor	*High (10)*	*Medium (6)*	*Assuming we have one for this particular project, we are not sure if additional funding will be provided, when and if needed*
Project Manager (Lecturer)	*High (10)*	*High (10)*	*Likes the new project. Risks tied to explaining the requirements to team members*
Team Members	*Medium (6)*	*High (10)*	*Almost all the members are glad to work on a new project. Though one member dropped it almost at the start. Additional training required*
Tenants	*High (8)*	*Low (1)*	*Gladly use the comfortable new service. Financial risks if payment or other service fails*
Legal Department	*High (7)*	*High (7)*	*Different possible legal risks*

2. Architecture Constraints

Technical constraints

- TODO

Nr.	Constraint	Background and/or motivation
TC1	*insert_constraint_here*	*insert_motivation_here*
TC2	*insert_constraint_here*	*insert_motivation_here*
TC3	*insert_constraint_here*	*insert_motivation_here*
TC4	*insert_constraint_here*	*insert_motivation_here*
TC5	*insert_constraint_here*	*insert_motivation_here*

Organizational constraints

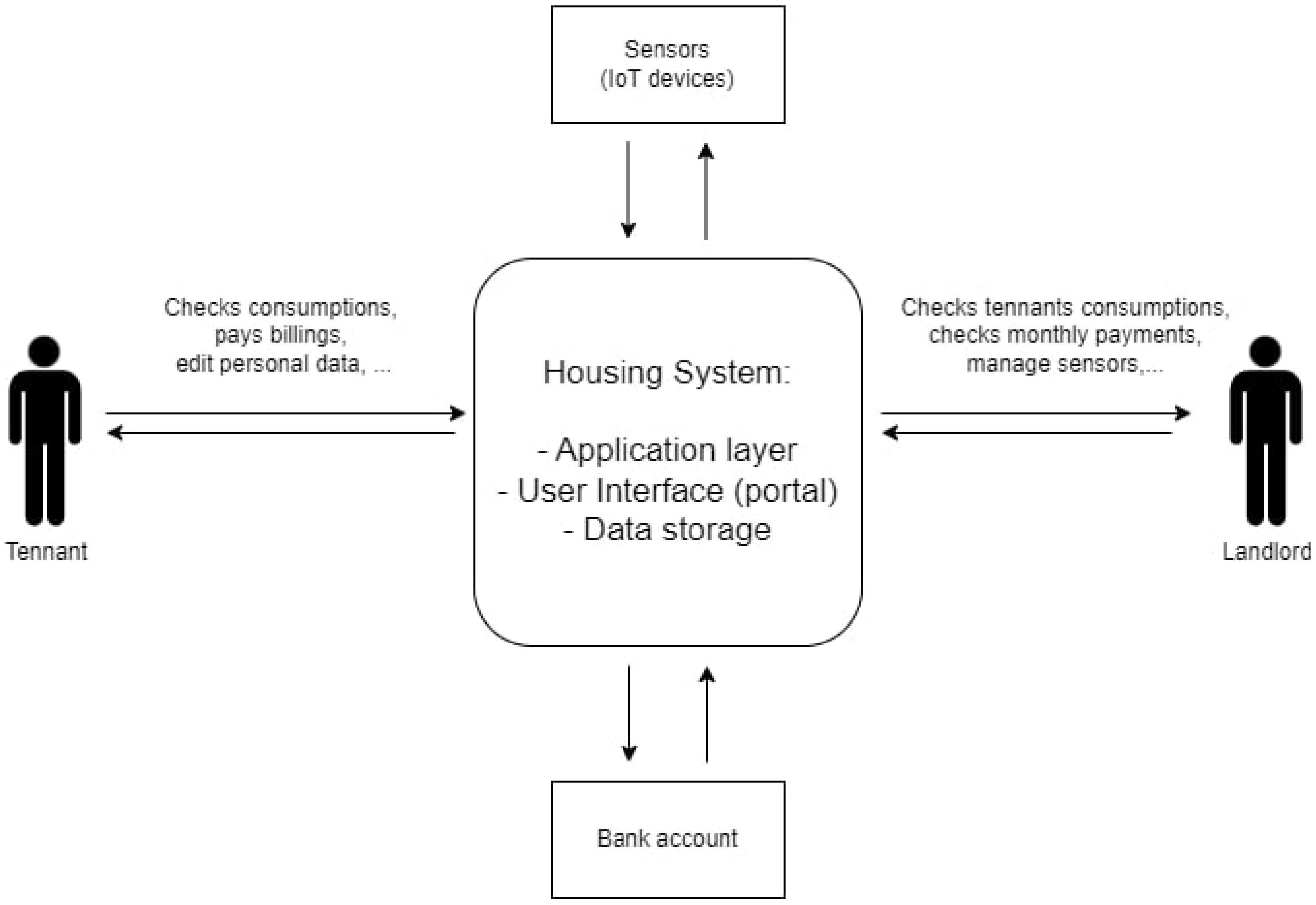
- TOFINISH

Nr.	Constraint	Background and/or motivation
OC1	User data location	All the personal data about users have to be stored in Germany or at least in a EU country, for privacy reasons.
OC2	Use of AWS	All the datas have to stored in cloud like AWS.
OC3	*insert_constraint_here*	*insert_motivation_here*
OC4	*insert_constraint_here*	*insert_motivation_here*
OC5	*insert_constraint_here*	*insert_motivation_here*

3. Context and Scope

3.1. Business Context

This section describes the environment and context of the Housing system at a high level: Who uses the system and on which other system does the it depend.



<Diagram or Table>

<optionally: Explanation of external domain interfaces>

3.2. Technical Context

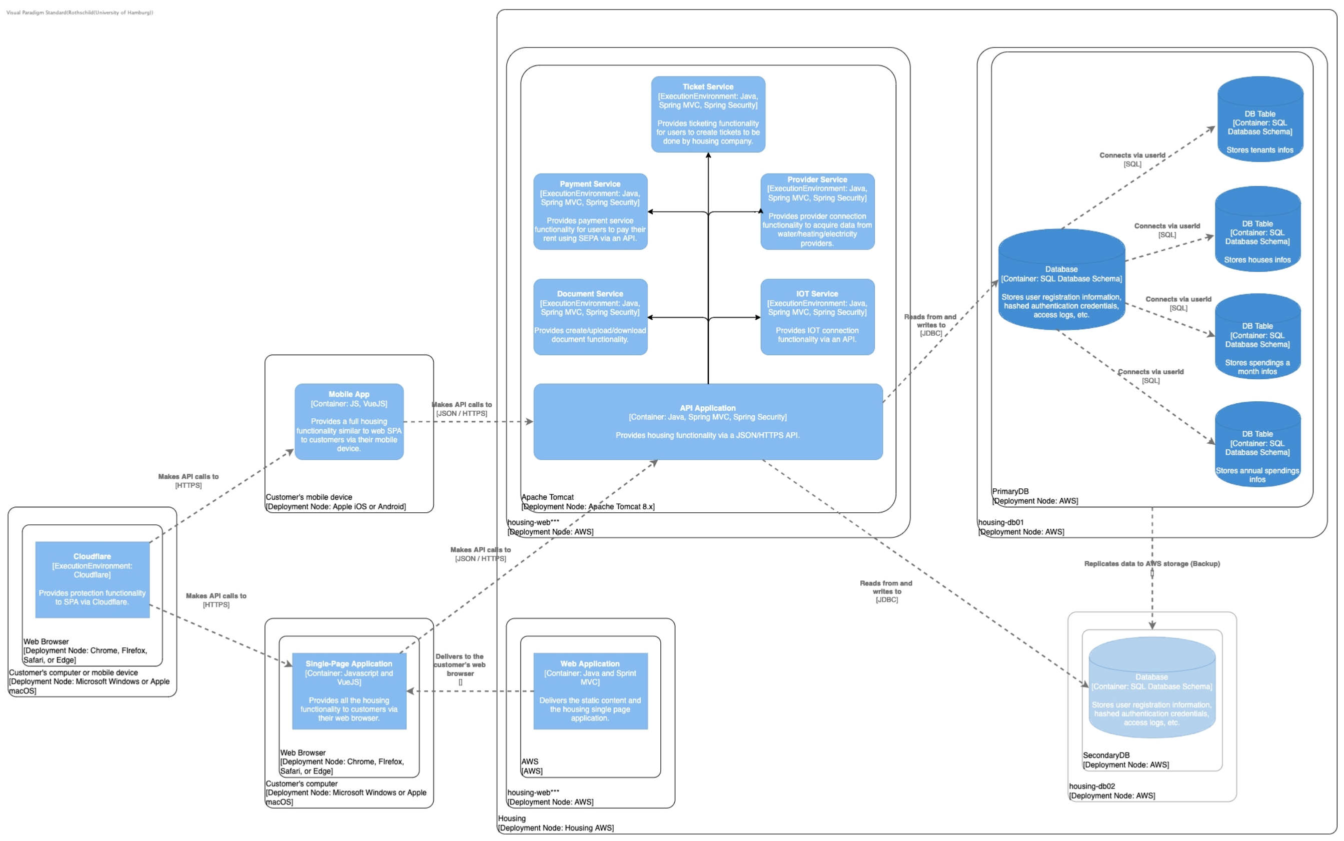
<Diagram or Table>

<optionally: Explanation of technical interfaces>

<Mapping Input/Output to Channels>

4. Solution Strategy

4.1. Microservice architecture


Visual Paradigm Standard(Rothschild(University of Hamburg))
Ticket Service
[ExecutionEnvironment: Java, Spring MVC, Spring Security]
Provides ticketing functionality for users to create tickets to be done by housing company.
Payment Service
[ExecutionEnvironment: Java, Spring MVC, Spring Security]
Provides payment service functionality for users to pay their rent using SEPA via an API.
Provider Service
[ExecutionEnvironment: Java, Spring MVC, Spring Security]
Provides provider connection functionality to acquire data from water/heating/electricity providers.
Document Service
[ExecutionEnvironment: Java, Spring MVC, Spring Security]
Provides create/upload/download document functionality.
IOT Service
[ExecutionEnvironment: Java, Spring MVC, Spring Security]
Provides IOT connection functionality via an API.
API Application
[Container: Java, Spring MVC, Spring Security]
Provides housing functionality via a JSON/HTTPS API.
Apache Tomcat
[Deployment Node: Apache Tomcat 8.x]
housing-web***
[Deployment Node: AWS]
Mobile App
[Container: JS, VueJS]
Provides a full housing functionality similar to web SPA to customers via their mobile device.
Customer's mobile device
[Deployment Node: Apple iOS or Android]
Makes API calls to
[JSON / HTTPS]
Makes API calls to
[HTTPS]
Cloudflare
[ExecutionEnvironment: Cloudflare]
Provides protection functionality to SPA via Cloudflare.
Web Browser
[Deployment Node: Chrome, Firefox, Safari, or Edge]
Customer's computer or mobile device
[Deployment Node: Microsoft Windows or Apple macOS]
Makes API calls to
[HTTPS]
Makes API calls to
[JSON / HTTPS]
Single-Page Application
[Container: Javascript and VueJS]
Provides all the housing functionality to customers via their web browser.
Web Browser
[Deployment Node: Chrome, Firefox, Safari, or Edge]
Customer's computer
[Deployment Node: Microsoft Windows or Apple macOS]
Delivers to the customer's web browser
[]
Web Application
[Container: Java and Sprint MVC]
Delivers the static content and the housing single page application.
AWS
[AWS]
housing-web***
[Deployment Node: AWS]
Housing
[Deployment Node: Housing AWS]
Reads from and writes to
[JDBC]
Database
[Container: SQL Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Connects via userId
[SQL]
DB Table
[Container: SQL Database Schema]
Stores tenants infos
Connects via userId
[SQL]
DB Table
[Container: SQL Database Schema]
Stores houses infos
Connects via userId
[SQL]
DB Table
[Container: SQL Database Schema]
Stores spendings a month infos
Connects via userId
[SQL]
DB Table
[Container: SQL Database Schema]
Stores annual spendings infos
PrimaryDB
[Deployment Node: AWS]
housing-db01
[Deployment Node: AWS]
Replicates data to AWS storage (Backup)
[]
Reads from and writes to
[JDBC]
Database
[Container: SQL Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
SecondaryDB
[Deployment Node: AWS]
housing-db02
[Deployment Node: AWS]

Microservices architecture is a design approach where a software application is structured as a collection of small, independent services that communicate with each other through well-defined APIs. While microservices based architecture is known for its scalability and flexibility, it also comes with its own set of pros and cons.

4.1.1. Benefits:

1. **Scalability:** Microservices can be independently scaled, allowing specific services to handle varying workloads.
2. **Flexibility:** Teams can develop and deploy services independently, enabling faster development cycles and updates.
3. **Technological Diversity:** Each microservice can be developed using different technologies, allowing teams to choose the best tools for each specific task.

4.1.2. Drawbacks:

1. **Complexity:** Managing a distributed system with multiple services introduces complexity in terms of deployment, monitoring, and debugging.
2. **Latency:** Communication between microservices can introduce network latency compared to monolithic architectures.
3. **Data Consistency:** Maintaining consistency across multiple databases can be challenging,

leading to potential data synchronization issues.

4.1.3. Risks:

1. **Data Management:**
    - **Issue:** Microservices often have their own databases, which can lead to challenges in maintaining data consistency across services.
    - **Mitigation:** Implement strategies like eventual consistency, use distributed transactions carefully, and employ techniques such as API gateways or event-driven architectures to handle data synchronization.
2. **Service Discovery and Communication:**
    - **Issue:** Dynamic service discovery and communication between microservices can introduce latency and network-related issues.
    - **Mitigation:** Use service mesh solutions, API gateways, and appropriate protocols to manage service discovery and communication. Implement retries and timeouts to handle transient failures.
3. **Security Concerns:**
    - **Issue:** Microservices architecture introduces additional attack vectors, and securing communication between services is crucial.
    - **Mitigation:** Implement proper authentication and authorization mechanisms. Use secure communication protocols (e.g., HTTPS) and regularly update security practices and policies.

4.2. Service-Oriented Architecture (Alternative Solution)

Service-Oriented Architecture (SOA) is an architectural style that involves organizing software systems as a set of loosely coupled, independently deployable services. It has a better maintainability compared to a full monolithic architecture, and also is not completely separated compared to a microservices architecture. While SOA has several benefits for our solution, it also comes with its set of challenges.

4.2.1. Benefits:

1. **Reusability and Modularity:** SOA promotes the development of modular and reusable services. Each service encapsulates a specific functionality and can be reused in different parts of an application or across different applications. This improves development efficiency and reduces redundancy.
2. **Interoperability:** SOA facilitates interoperability between different software systems and technologies. Services communicate using standard protocols and can be implemented in various programming languages. This enables organizations to integrate diverse systems.
3. **Flexibility:** The flexibility is particularly valuable in dynamic environments where the demand for certain services may vary. It also facilitates easier adaptation to changes and updates in

technology or business requirements.

4.2.2. Drawbacks:

1. **Complexity and Overhead:** Implementing and managing a service-oriented architecture can introduce complexity. Services need to be designed, deployed, and orchestrated properly. Additionally, the overhead of managing service discovery, communication, and security can be challenging, particularly in large and distributed systems.
2. **Performance Concerns:** The communication between services, often over a network, introduces latency compared to in-process calls. This can impact the overall performance of the system, especially if poorly designed or if services are distributed across geographically dispersed locations. Efficient communication and data transfer mechanisms are crucial to mitigating this drawback.
3. **Security Challenges:** Managing security in a SOA environment can be complex. Services may need to authenticate and authorize requests, and securing communication between services is essential. Additionally, ensuring compliance with governance policies, such as version control and service contracts, requires careful planning and implementation.

4.2.3. Risks:

1. **Dependency Management:**
    - **Issue:** Changes in one service may impact other dependent services, leading to potential disruptions.
    - **Mitigation:** Implement version control for services and use service registries to manage dependencies. Define clear service contracts and communication protocols. Establish change management processes to assess the impact of modifications before deployment. Employ automated testing and continuous integration practices to catch integration issues early.
2. **Service Availability and Reliability:**
    - **Issue:** Services may experience downtime, affecting the overall system availability.
    - **Mitigation:** Implement redundancy and failover mechanisms for critical services. Use load balancing to distribute traffic across multiple instances of a service. Regularly monitor and perform testing to identify and address potential points of failure.
3. **Data Security and Privacy:**
    - **Issue:** Sharing data between services may lead to security vulnerabilities or unauthorized access.
    - **Mitigation:** Implement strong authentication and authorization mechanisms. Encrypt sensitive data during transmission and storage. Conduct regular security audits and penetration testing to identify and address vulnerabilities. Define and enforce data access policies.
4. **Performance Bottlenecks:**
    - **Issue:** Increased complexity in a distributed system may lead to performance bottlenecks and degraded response times.

    - **Mitigation:** Conduct performance testing to identify and optimize bottlenecks. Implement caching mechanisms for frequently accessed data. Utilize content delivery networks (CDNs) for static content. Monitor system performance in real-time and scale infrastructure as needed.

4.3. Payment module

4.3.1. Existing Payment Service:

1. **Benefits:**
    - **Time and Cost Efficiency:** Utilizing an existing payment module can significantly reduce development time and costs. Ready-made solutions are often well-tested and come with documentation, saving resources that would be spent on coding, testing, and troubleshooting.
    - **Familiarity:** Users would be much more familiar using an existing Payment gateway (eg. PayPal) compared to a custom payment system.
2. **Drawback:**
    - **Limited Customization:** While existing modules offer convenience, they may have limitations in terms of customization. If your project requires highly specialized features or a unique user experience, you might find it challenging to tailor the module to fit your exact needs.
3. **Risk:**
    - **Dependency on Third-Party:** Relying on an external payment module means your project is dependent on a third-party service. If the external provider faces issues like downtime, security breaches, or discontinuation of support, your payment functionality may be adversely affected.

4.3.2. Custom Payment Service:

1. **Benefit:**
    - **Customization Control:** Building your own payment module provides full control over customization. You can tailor the solution to precisely match your business requirements, ensuring a unique and seamless integration with your application.
2. **Drawback:**
    - **Resource Intensive:** Developing a payment module from scratch demands significant time, effort, and resources. It requires expertise in security, compliance, and various payment processing protocols. This can extend the development timeline and increase initial costs.
3. **Risk:**
    - **Security Concerns:** Creating a payment module involves handling sensitive financial information. If not implemented securely, it can pose a risk of data breaches and compromise user trust. Ensuring compliance with industry standards and best practices is crucial to mitigate these risks.

4.4. IOT module

4.4.1. Cloud-Based Solution:

1. **Benefit:**
    - **Scalability:** Cloud solutions offer easy scalability, allowing you to adjust resources based on demand. This is crucial for handling fluctuating transaction volumes in payment processing.
2. **Drawback:**
    - **Latency:** Transactions may experience latency due to data transfer between the local system and the cloud server. This can impact the real-time nature of payment processing.
3. **Risk:**
    - **Security Concerns:** Storing sensitive payment data in the cloud raises security concerns. A breach could lead to unauthorized access and potential data leaks, compromising user information.

4.4.2. Edge Computing Solution:

1. **Benefits:**
    - **Low Latency:** Edge computing processes data locally, reducing the latency associated with round-trip data transfer to a centralized server. This is advantageous for real-time payment processing.
2. **Drawbacks:**
    - **Limited Scalability:** Edge computing may face challenges when it comes to scaling resources, especially in comparison to the virtually limitless scalability offered by cloud solutions.
3. **Risks:**
    - **Maintenance Complexity:** Edge devices may be distributed across various locations, making maintenance more complex. Ensuring consistent updates and security patches can be challenging.

4.5. Ticket module

4.5.1. Existing Ticketing Solution:

1. **Benefits:**
    - **Time Efficiency:** Implementing an existing ticketing solution can save a significant amount of time compared to developing one from scratch. This is because these solutions are often feature-rich and ready to use.
2. **Drawbacks:**
    - **Cost:** Some established ticketing solutions may come with a substantial upfront cost or ongoing subscription fees. This can be a financial burden, especially for small businesses or startups.

3. **Risks:**
    - **Customization Limitations:** Existing solutions might not fully align with unique business processes, and customization options could be limited. This might lead to compromises in workflow efficiency.

4.5.2. Develop a Ticketing Solution In-House:

1. **Benefits:**
    - **Customization:** Building your own ticketing system allows for tailoring the solution to fit the specific needs and workflows of your organization. This can enhance overall efficiency and user satisfaction.
2. **Drawbacks:**
    - **Time and Resources:** Developing a ticketing system in-house requires time, skilled personnel, and resources. This could potentially divert attention and resources from other critical projects or daily operations.
3. **Risks:**
    - **Maintenance Challenges:** Ongoing maintenance and updates can be challenging and time-consuming. Ensuring the system remains up-to-date with changing requirements and technologies poses a risk of creating a resource-intensive burden.

ASRs	**Basic architecture style**		**Payment Service**		**IOT Service**		**Ticket Service**	
	Microservices architecture	Service Oriented Architecture	Existing Payment Service	Custom Payment Service	Cloud based	Edge computing	Existing Ticket Service	Custom Ticket Service
ASR1	++	o	++	o	++	o	+	-
ASR2	o	++	o	++	o	++	o	++
ASR3	-	++	++	++	o	++	-	++
ASR4	++	+	+	o	++	+	++	+
ASR5	++	+	++	+	++	+	++	+
ASR6	o	o	+	+	-	o	o	o
ASR7	++	+	++	+	++	+	+	o
UC1	+	+	+	+	+	+	+	+
UC2	+	+	+	+	o	o	+	+
UC3	+	+	+	+	+	+	++	+
UC4	+	++	o	+	+	++	+	++
UC5	++	o	++	o	+	-	++	o

5. Building Block View

5.1. Whitebox Overall System

<Overview Diagram>

Motivation

<text explanation>

Contained Building Blocks

<Description of contained building block (black boxes)>

Important Interfaces

<Description of important interfaces>

5.1.1. <Name black box 1>

<Purpose/Responsibility>

<Interface(s)>

<(Optional) Quality/Performance Characteristics>

<(Optional) Directory/File Location>

<(Optional) Fulfilled Requirements>

<(optional) Open Issues/Problems/Risks>

5.1.2. <Name black box 2>

<black box template>

5.1.3. <Name black box n>

<black box template>

5.1.4. <Name interface 1>

…

5.1.5. <Name interface m>

5.2. Level 2

5.2.1. White Box *<building block 1>*

<white box template>

5.2.2. White Box <*building block 2*>

<white box template>

…

5.2.3. White Box <*building block m*>

<white box template>

5.3. Level 3

5.3.1. White Box <_building block x.1_>

<white box template>

5.3.2. White Box <_building block x.2_>

<white box template>

5.3.3. White Box <_building block y.1_>

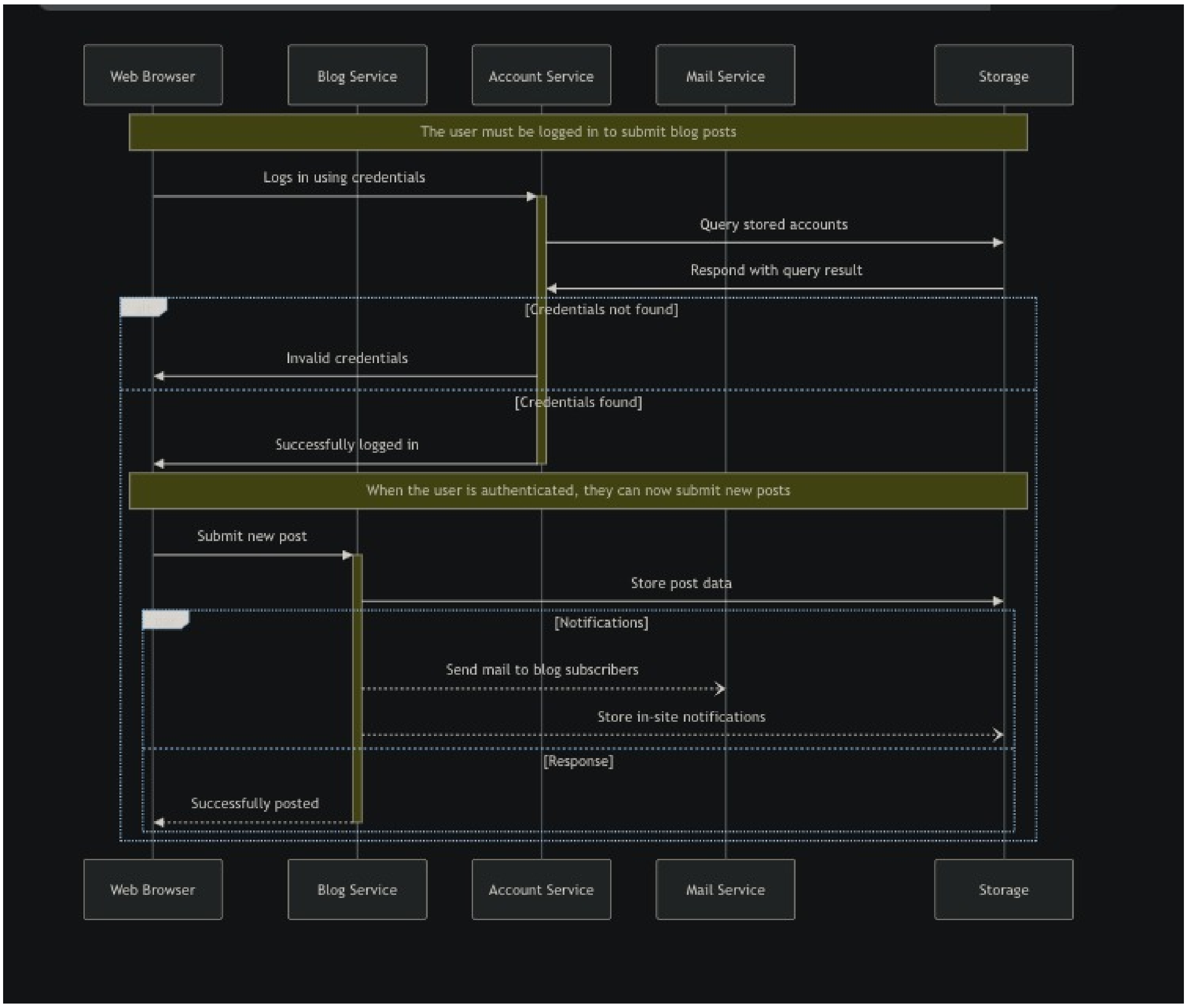
<white box template>

6. Runtime View

User interaction with *project_name* and error handling is pretty basic and simple.

6.1. <Runtime Scenario 1 (Login procedure)>

The following is the use case of the login procedure.



- *<insert runtime diagram or textual description of the scenario>*
- *<insert description of the notable aspects of the interactions between the building block instances depicted in this diagram.>*
    - TODO: add other runtime scenarios.

6.2. <Runtime Scenario 2>

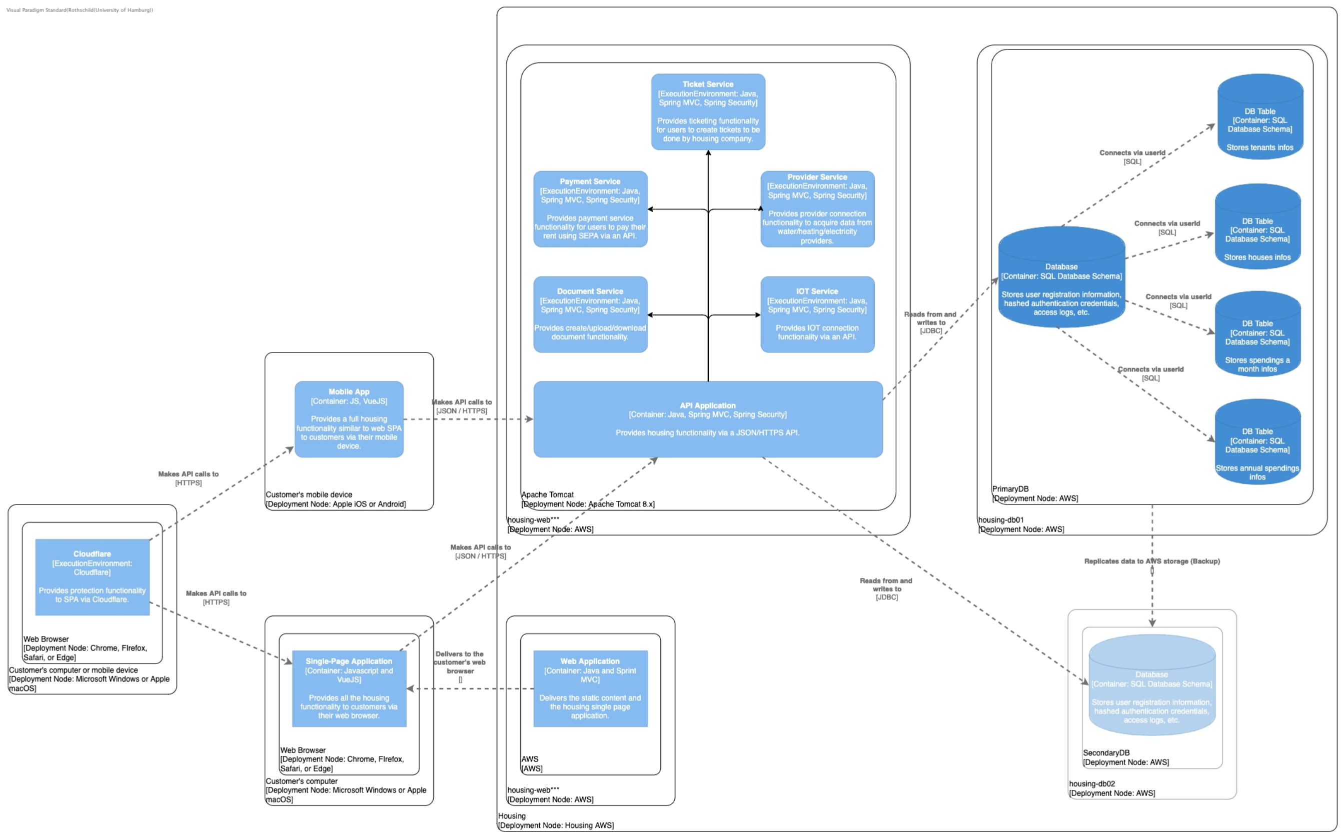
6.3. …

6.4. <Runtime Scenario n>

7. Deployment View

7.1. Infrastructure Level 1

Main deployment diagram for the Housing System.



Visual Paradigm Standard(Rothschild(University of Hamburg))
Ticket Service [ExecutionEnvironment: Java, Spring MVC, Spring Security] Provides ticketing functionality for users to create tickets to be done by housing company.
Payment Service [ExecutionEnvironment: Java, Spring MVC, Spring Security] Provides payment service functionality for users to pay their rent using SEPA via an API.
Provider Service [ExecutionEnvironment: Java, Spring MVC, Spring Security] Provides provider connection functionality to acquire data from water/heating/electricity providers.
Document Service [ExecutionEnvironment: Java, Spring MVC, Spring Security] Provides create/upload/download document functionality.
IOT Service [ExecutionEnvironment: Java, Spring MVC, Spring Security] Provides IOT connection functionality via an API.
API Application [Container: Java, Spring MVC, Spring Security] Provides housing functionality via a JSON/HTTPS API.
Apache Tomcat [Deployment Node: Apache Tomcat 8.x]
housing-web*** [Deployment Node: AWS]
Mobile App [Container: JS, VueJS] Provides a full housing functionality similar to web SPA to customers via their mobile device.
Customer's mobile device [Deployment Node: Apple iOS or Android]
Makes API calls to [JSON / HTTPS]
Makes API calls to [HTTPS]
Cloudflare [ExecutionEnvironment: Cloudflare] Provides protection functionality to SPA via Cloudflare.
Web Browser [Deployment Node: Chrome, Firefox, Safari, or Edge]
Customer's computer or mobile device [Deployment Node: Microsoft Windows or Apple macOS]
Makes API calls to [HTTPS]
Makes API calls to [JSON / HTTPS]
Single-Page Application [Container: Javascript and VueJS] Provides all the housing functionality to customers via their web browser.
Web Browser [Deployment Node: Chrome, Firefox, Safari, or Edge]
Customer's computer [Deployment Node: Microsoft Windows or Apple macOS]
Delivers to the customer's web browser []
Web Application [Container: Java and Sprint MVC] Delivers the static content and the housing single page application.
AWS [AWS]
housing-web*** [Deployment Node: AWS]
Housing [Deployment Node: Housing AWS]
Reads from and writes to [JDBC]
Database [Container: SQL Database Schema] Stores user registration information, hashed authentication credentials, access logs, etc.
Connects via userId [SQL]
DB Table [Container: SQL Database Schema] Stores tenants infos
Connects via userId [SQL]
DB Table [Container: SQL Database Schema] Stores houses infos
Connects via userId [SQL]
DB Table [Container: SQL Database Schema] Stores spendings a month infos
Connects via userId [SQL]
DB Table [Container: SQL Database Schema] Stores annual spendings infos
PrimaryDB [Deployment Node: AWS]
housing-db01 [Deployment Node: AWS]
Replicates data to AWS storage (Backup) []
Reads from and writes to [JDBC]
Database [Container: SQL Database Schema] Stores user registration information, hashed authentication credentials, access logs, etc.
SecondaryDB [Deployment Node: AWS]
housing-db02 [Deployment Node: AWS]

<Overview Diagram>

Motivation

<explanation in text form>

Quality and/or Performance Features

<explanation in text form>

Mapping of Building Blocks to Infrastructure

<description of the mapping>

7.2. Infrastructure Level 2

- TODO: add other level of infrastructure if needed.

7.2.1. *<Infrastructure Element 1>*

<diagram + explanation>

7.2.2. *<Infrastructure Element 2>*

<diagram + explanation>

…

7.2.3. *<Infrastructure Element n>*

<diagram + explanation>

8. Cross-cutting Concepts

8.1. *<Concept 1>*

<explanation>

8.2. *<Concept 2>*

<explanation>

…

8.3. *<Concept n>*

<explanation>

9. Architecture Decisions

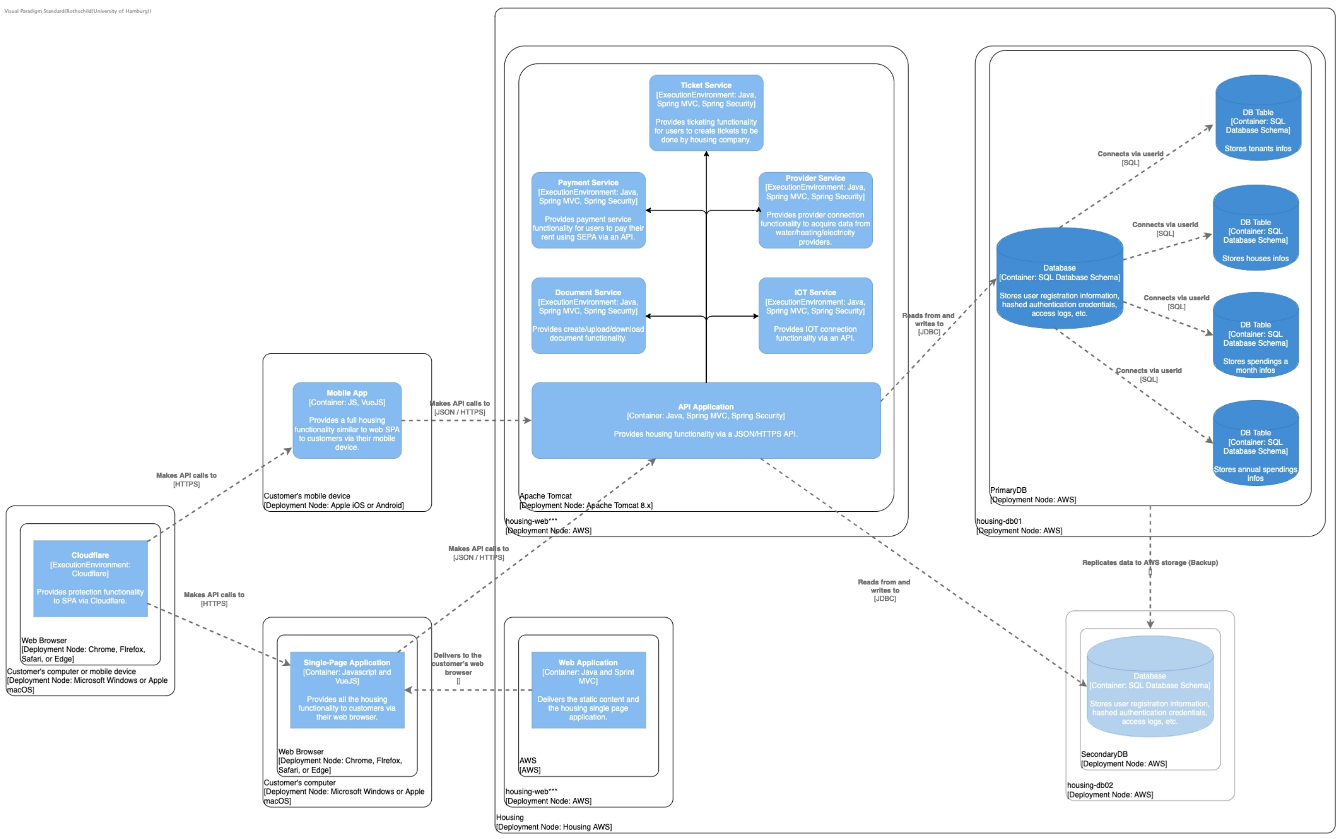
9.1. Table of involved stakeholders

Name	**Reason**	**Requirements**	**Diagram Type**
Owner	1. The owner needs to know mainly how the system will be used by the final users and also by him. 2. His main goal is to have a system with a clear and straightforward use from both sides (him and the users) and he also needs the system to address all the possible edge cases that can arise by using the system.	1. He doesn't really need to know the system general architecture and all the design decision taken. 2. He mainly need to know usage of the system and maybe how it interacts with the third party services.	The owner needs behavioral (Mockup/Wireframe, Sequence and Deployement) diagrams.
Sponsor	-	-	-
Project Manager (Lecturer) & Team Members (Developers)	PM/Developers part of the team must know what and how to implement the system.	1. PM/Developers need to know about the system general architecture. 2. They also need to know how the system components interact between them and also how they interact with the third party services. 3. To better understand the final product behaviour they also need to know how the final system will be used by the tenants.	PM/Developers need both structural and behavioral diagrams.
Tenants (Users)	The users must know about some diagrams since their final UX can benefit from them.	1. The users must know how to interact with the system. 2. They don't need to know how the system architecture or any design decision about it. 3. They also don't need to know how the system communicates with the sensors, payment method and other thirs party services.	Users needs behavioural (Mockup/Wireframe) diagrams.

Name	Reason	Requirements	Diagram Type
Legal Department	1. Some diagrams can help in representing and validating that the software behaves in a legally compliant manner during the interactions with other systems. 2. Legal requirements may specify certain orderings or conditions for some interactions.	1. Legal Department needs to know about interactions between objects or components in the system. 2. Legal Department needs to know about interactions between system and third party components.	The legal department needs both structural (Deployment diagram) and behavioral (Sequence Diagrams and Mockup/Wireframe) diagrams.

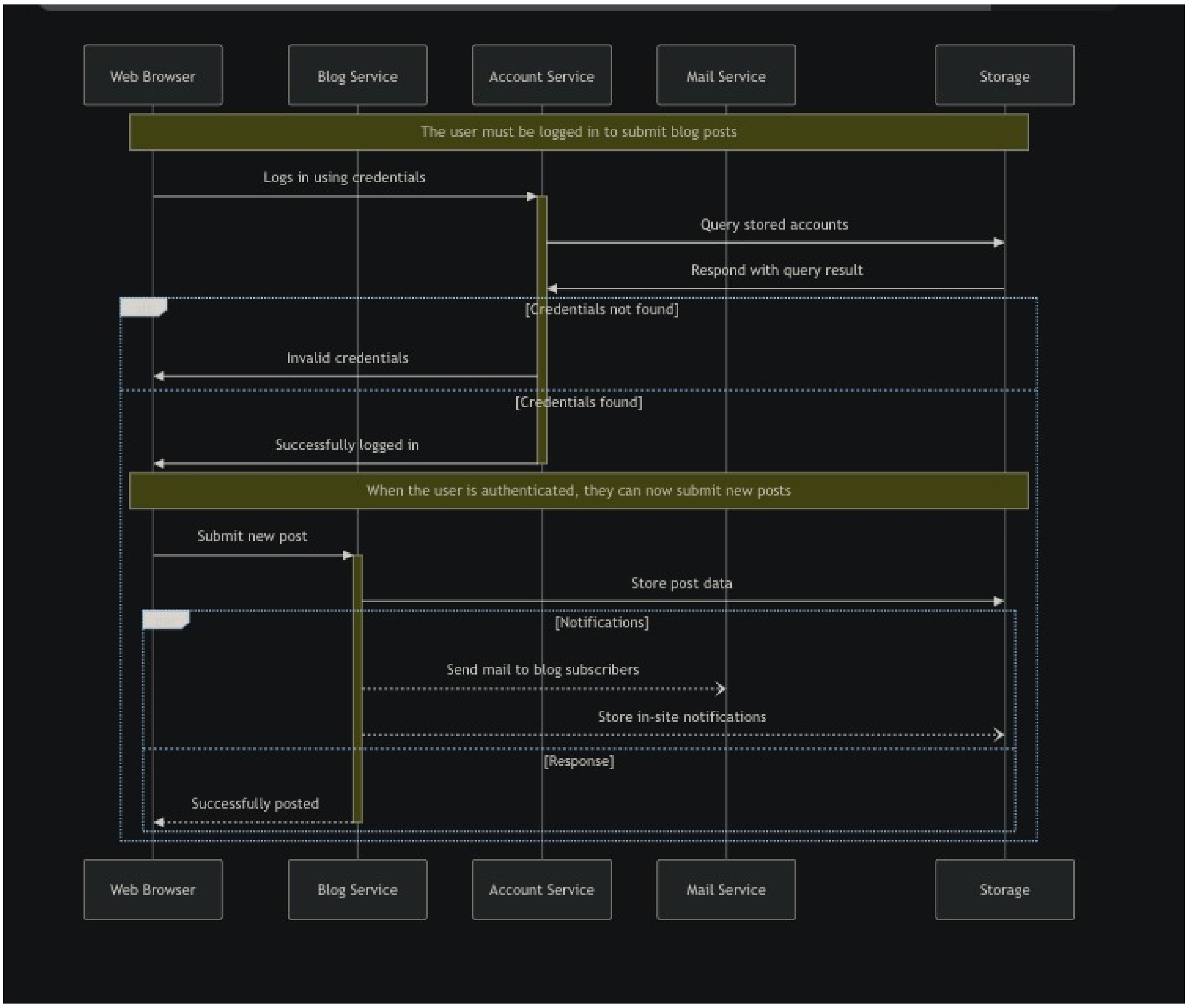
9.2. UML Diagrams

9.2.1. Component diagram

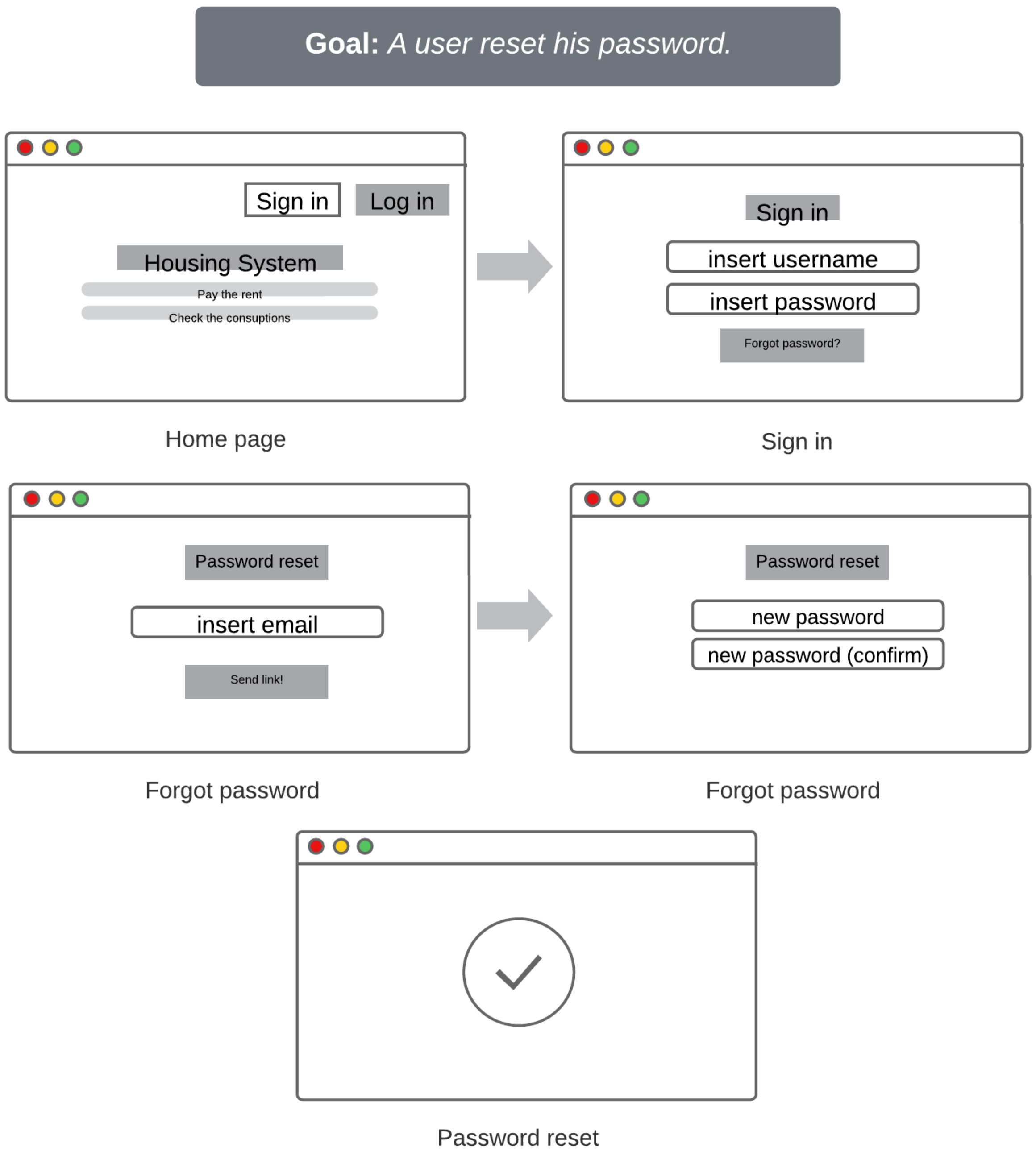


Visual Paradigm Standard(Rothschild(University of Hamburg))
Ticket Service [ExecutionEnvironment: Java, Spring MVC, Spring Security] Provides ticketing functionality for users to create tickets to be done by housing company.
Payment Service [ExecutionEnvironment: Java, Spring MVC, Spring Security] Provides payment service functionality for users to pay their rent using SEPA via an API.
Provider Service [ExecutionEnvironment: Java, Spring MVC, Spring Security] Provides provider connection functionality to acquire data from water/heating/electricity providers.
Document Service [ExecutionEnvironment: Java, Spring MVC, Spring Security] Provides create/upload/download document functionality.
IOT Service [ExecutionEnvironment: Java, Spring MVC, Spring Security] Provides IOT connection functionality via an API.
API Application [Container: Java, Spring MVC, Spring Security] Provides housing functionality via a JSON/HTTPS API.
Apache Tomcat [Deployment Node: Apache Tomcat 8.x]
housing-web*** [Deployment Node: AWS]
Mobile App [Container: JS, VueJS] Provides a full housing functionality similar to web SPA to customers via their mobile device.
Customer's mobile device [Deployment Node: Apple iOS or Android]
Makes API calls to [JSON / HTTPS]
Makes API calls to [HTTPS]
Cloudflare [ExecutionEnvironment: Cloudflare] Provides protection functionality to SPA via Cloudflare.
Web Browser [Deployment Node: Chrome, Firefox, Safari, or Edge]
Customer's computer or mobile device [Deployment Node: Microsoft Windows or Apple macOS]
Makes API calls to [HTTPS]
Makes API calls to [JSON / HTTPS]
Single-Page Application [Container: Javascript and VueJS] Provides all the housing functionality to customers via their web browser.
Web Browser [Deployment Node: Chrome, Firefox, Safari, or Edge]
Customer's computer [Deployment Node: Microsoft Windows or Apple macOS]
Delivers to the customer's web browser []
Web Application [Container: Java and Sprint MVC] Delivers the static content and the housing single page application.
AWS [AWS]
housing-web*** [Deployment Node: AWS]
Housing [Deployment Node: Housing AWS]
Reads from and writes to [JDBC]
Database [Container: SQL Database Schema] Stores user registration information, hashed authentication credentials, access logs, etc.
Connects via userId [SQL]
DB Table [Container: SQL Database Schema] Stores tenants infos
Connects via userId [SQL]
DB Table [Container: SQL Database Schema] Stores houses infos
Connects via userId [SQL]
DB Table [Container: SQL Database Schema] Stores spendings a month infos
Connects via userId [SQL]
DB Table [Container: SQL Database Schema] Stores annual spendings infos
PrimaryDB [Deployment Node: AWS]
housing-db01 [Deployment Node: AWS]
Replicates data to AWS storage (Backup) []
Reads from and writes to [JDBC]
Database [Container: SQL Database Schema] Stores user registration information, hashed authentication credentials, access logs, etc.
SecondaryDB [Deployment Node: AWS]
housing-db02 [Deployment Node: AWS]

9.2.2. Sequence diagram



9.2.3. Mockup/Wireframe for password reset



9.2.4. Deployment diagram (insert here)

10. Quality Requirements

Table 1. Architectural sensitive requirements

Quality requirement	**Description**	**Changeability**	**Flexibility**
Integrability	The ability of a system to seamlessly integrate with other systems or components, ensuring smooth communication and data exchange.	Medium	Medium
Security	Measures and features implemented to protect a system from unauthorized access, data breaches, and other security threats.	Low	Low
Availability	Ensuring that a system is consistently accessible and operational, minimizing downtime and disruptions.	Medium	High
Deployability	TThe ease with which a system can be installed, configured, and deployed in various environments, ensuring a smooth and efficient deployment process.	High	High
Testability	The degree to which a system facilitates testing, making it easier to identify and fix bugs or issues during development and maintenance.	Medium	Medium
Usability	The extent to which a system is user-friendly and provides a positive user experience, considering factors such as ease of use, learnability, and efficiency.	High	High
Maintainability	The ease with which a system can be maintained and updated over time, including aspects like code readability, modularity, and the ability to make changes without causing disruptions.	High	Medium
Performance	The efficiency and responsiveness of a system under various conditions, including factors such as response time, throughput, and resource utilization.	High	Medium

Quality requirement	**Description**	**Changeability**	**Flexibility**
Modifiability	The ease with which a system can be modified or adapted to accommodate changes in requirements, evolving technology, or business needs. This includes aspects like code flexibility, scalability, and the ability to introduce new features without extensive rework.	Medium	Medium

10.1. Quality Scenarios

10.1.1. Availability

Portion of Scenario	**Description with possible values**
ID	ASR1
Source	webservice unavailable
Stimulus	timing
Artifact	communication channel
Environment	normal operation/overload with requests
Response	log failure and notify operator via alarm alarm_
Response Measure	no downtime

Portion of Scenario	**Description with possible values**
ID	ASR2
Source	scheduled maintenance
Stimulus	causes downtime
Artifact	system
Environment	under normal maintenance
Response	quick maintenance
Response Measure	no downtime

10.1.2. Performance

Portion of Scenario	**Description with possible values**
ID	ASR3

Portion of Scenario	Description with possible values
Source	external, internal
Stimulus	event arrival pattern
Artifact	system services
Environment	normal, overload
Response	change operation mode
Response Measure	latency, deadline, throughput, jitter, miss rate, data loss

10.1.3. Security

Portion of Scenario	Description with possible values
ID	ASR4
Source	user/system, known/unknown, unauthorized user
Stimulus	attack to display info, change info, access services and info, deny services, tries to disable system
Artifact	services, data
Environment	online/offline, connected or disconnected
Response	authentication, authorization, encryption, logging, demand monitoring
Response Measure	time, probability of detection, recovery, service is available within 1 minute

10.1.4. Modifiability

Portion of Scenario	Description with possible values
ID	ASR5
Source	developer, system administrator, user
Stimulus	add/delete/modify function or quality, wishes to change fields of a downloadable document
Artifact	UI, platform, environment, external system
Environment	design, compile, build, runtime
Response	make change, test it, deploy it
Response Measure	effort, time, cost, risk, field change and download occurs without error logs

10.1.5. Testability

Portion of Scenario	**Description and possible Value**
ID	ASR6
Source	developer, tester, user
Stimulus	project milestone completed
Artifact	design, code component, system
Environment	design, development, compile, deployment, or run time
Response	can be controlled to perform the desired test and results observed
Response Measure	coverage, probability of finding additional faults given a fault, time to test, entire regression test suite completed in less than 24 hours

Portion of Scenario	**Description with possible values**
ID	ASR7
Source	developer, tester
Stimulus	a change in one module unintentionally affects functionality of another
Artifact	code component, system
Environment	under normal testing
Response	rollback
Response Measure	no downtime

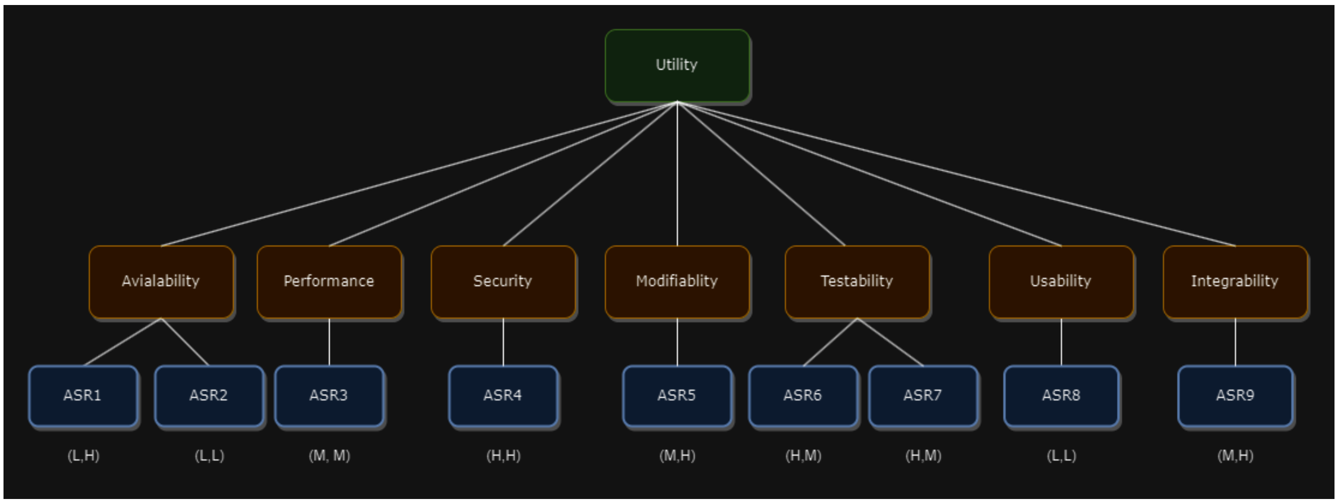
10.1.6. Usability

Portion of Scenario	**Description and possible Value**
ID	ASR8
Source	user
Stimulus	the tenant wants to set up a recurring payment for monthly rent on the rental payment portal
Artifact	interface, system
Environment	normal use of the system
Response	the system set the recurring payment
Response Measure	quick time response, value changed or not

10.1.7. Integrability

Portion of Scenario	**Description and possible Value**
ID	ASR9
Source	external system
Stimulus	a third-party payment gateway needs to be integrated with the system
Artifact	system, api
Environment	normal development
Response	all the other parts of the system should be perfectly integrated with the payment gateway
Response Measure	effort and time spent to integrate the gateway

10.2. Quality Tree



11. Risks and Technical Debts

12. Glossary

Term	**Definition**
<Term-1>	*<definition-1>*
<Term-2>	*<definition-2>*