SAD assignment

Blekinge Tekniska Högskola, Sep 2023

Hammam Dowah - Karam Kirawan - Abdulkarim Dawalibi - Adnan Altukleh

1 Part 1: Architectural Drivers

1.1 Introduction and Design Purpose

The architecture design is being done for a new system by building native apps for all popular cell phone operating systems. The new system should be integrated with the current system. Knowing that, the architecture design is a bridge between requirements and code[1], The architecture design is being done after acknowledging the customer's requirements. Having a well-defined architectural design is crucial for building a system that not only meets certain quality benchmarks but is also easier to maintain over time. Architecture design provides a clear roadmap for developers, it makes a clear plan that makes the system more understandable and clarifies what technologies are being or will be used. This structured approach facilitates more accurate cost estimation, ensuring the application remains cost-effective throughout its lifecycle. One common risk companies often encounter is poor communication among stakeholders. A strong architectural design solves this by establishing specific standards and protocols for system development. These standards act as a common language, enabling all stakeholders to interact seamlessly with the system. A well-defined architecture design considers performance implications, this means optimizing the way the system uses or utilizes resources like CPU and memory. It will not only improve the overall system performance but will also help in cost savings.

At this time Sagittarius Bank AB is mostly concerned about availability, scalability, security, modifiability, and last but not least time to market. Sagittarius Bank aims to optimize the system's ability to function and be accessible to users whenever they need to use it. It involves ensuring that the software application or system remains operational, responsive, and able to handle user requests without any undue downtime or interruptions. Also when releasing new modifications, updates, or new features to the system it should be cost-effective. Given the intention to expand services, it is crucial that the new system can handle an increasing number of users without compromising on performance or uptime. As a fintech company that deals with financial transactions and sensitive customer data, it is important to make sure that security measures are a top priority. The main goal for the Sagittarius Bank is to have a cross-platform to reach all markets and to have a piece of cake of the market cap.

1.2 Quality Attributes

1.2.1 Selected Quality Attributes

Based on the system, the following quality attributes were chosen:

- **QA1- Security:** Security is an important aspect of the Sagittarius Bank AB system because it deals with financial transactions and sensitive customer data. Ensuring the security of this data is essential to protect it from unauthorized access. Additionally, Authorization and Authentication (AA) are required to control user access and prevent unauthorized modifications to the

[1] *Bass, L., Kazman, R., & Clements, P. (2021)*

system, maintaining the integrity of the system and the confidentiality of customer information.

- **QA2 - Performance:** The need for performance in the Sagittarius Bank AB is evident from the requirement "The service should be able to keep up with the increasing number of customers to provide pristine and instant functionality" This shows the importance of a high-performance system to minimize downtime and deliver a responsive and efficient user experience.

- **QA3 - Modifiability:** As requested in the system requirements, the system should be change-friendly, which means that the system should be easily modifiable when new features are going to be implemented. Also, the fintech industry is fast-evolving, requiring the system to adapt to new technologies. Moreover, considering the company's close engagement with customers, it may need to apply changes based on the customer's preferences/feedback to build a reputation and maintain a competitive edge. A modifiable system ensures that the adaptation and changes are implemented seamlessly without undergoing a complete system overhaul.

- **QA4 - Testability:** Identifying bugs, issues, or vulnerabilities in the system in an early stage is critical to eliminate the chance of encountering hard-to-resolve errors. Testing allows early detection and provides more time to address the errors. In addition, the system's capability to handle an increasing volume of transactions and users. To address that, performance testing can be conducted to verify if the system can maintain low downtime as requested in the requirements.

1.2.2 Quality Attributes Scenarios

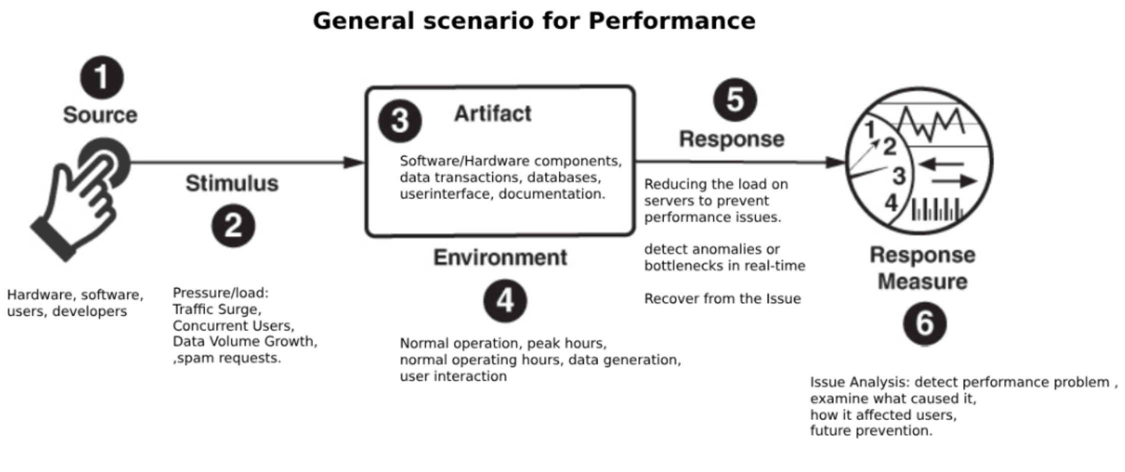

General scenario for Performance
1
Source
Hardware, software, users, developers
Stimulus
2
Pressure/load:
Traffic Surge,
Concurrent Users,
Data Volume Growth,
,spam requests.
3
Artifact
Software/Hardware components, data transactions, databases, userinterface, documentation.
Environment
4
Normal operation, peak hours, normal operating hours, data generation, user interaction
5
Response
Reducing the load on servers to prevent performance issues.
detect anomalies or bottlenecks in real-time
Recover from the Issue
Response Measure
6
Issue Analysis: detect performance problem , examine what caused it, how it affected users, future prevention.

Figure 1: The figure shows a general scenario for performance

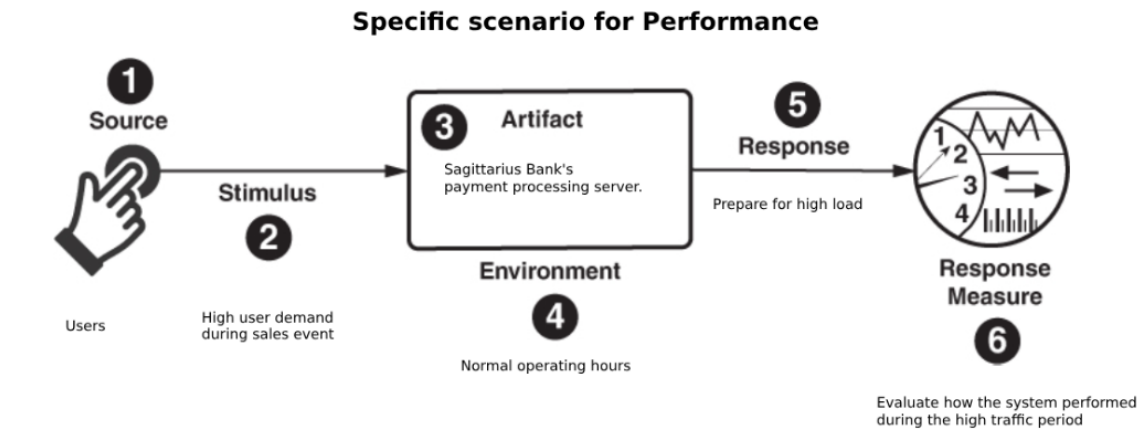

Specific scenario for Performance
1
Source
Users
Stimulus
2
High user demand during sales event
3
Artifact
Sagittarius Bank's payment processing server.
Environment
4
Normal operating hours
5
Response
Prepare for high load
Response Measure
6
Evaluate how the system performed during the high traffic period

Figure 2: The figure shows a specific scenario for performance

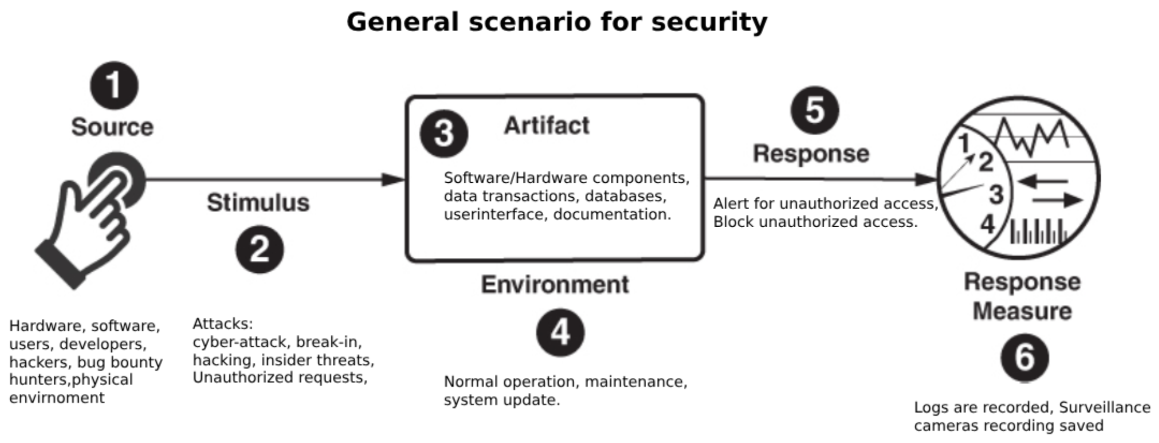

General scenario for security
1
Source
Hardware, software, users, developers, hackers, bug bounty hunters,physical envirnoment
Stimulus
2
Attacks: cyber-attack, break-in, hacking, insider threats, Unauthorized requests,
3
Artifact
Software/Hardware components, data transactions, databases, userinterface, documentation.
Environment
4
Normal operation, maintenance, system update.
5
Response
Alert for unauthorized access, Block unauthorized access.
Response Measure
6
Logs are recorded, Surveillance cameras recording saved

Figure 3: The figure shows a general scenario for security

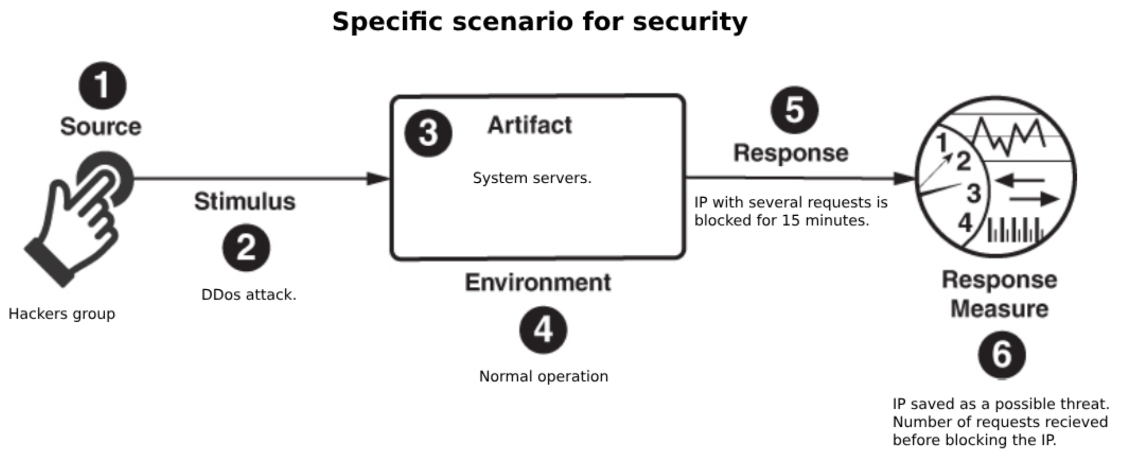

Specific scenario for security
1
Source
Hackers group
Stimulus
2
DDos attack.
3
Artifact
System servers.
Environment
4
Normal operation
5
Response
IP with several requests is blocked for 15 minutes.
Response Measure
6
IP saved as a possible threat. Number of requests recieved before blocking the IP.

Figure 4: The figure shows a specific scenario for security

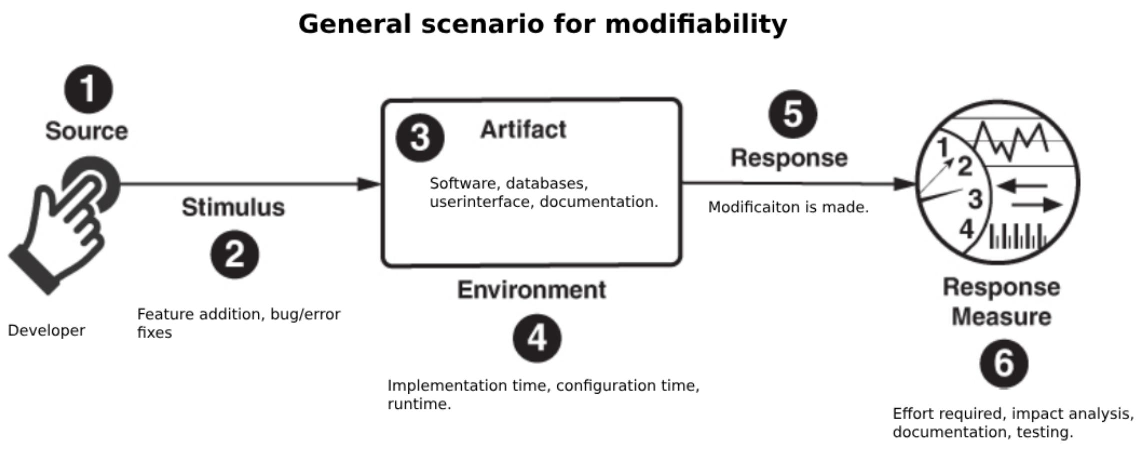

General scenario for modifiability
1
Source
Developer
Stimulus
2
Feature addition, bug/error fixes
3
Artifact
Software, databases, userinterface, documentation.
Environment
4
Implementation time, configuration time, runtime.
5
Response
Modificaiton is made.
Response Measure
6
Effort required, impact analysis, documentation, testing.

Figure 5: The figure shows a general scenario for modifiability

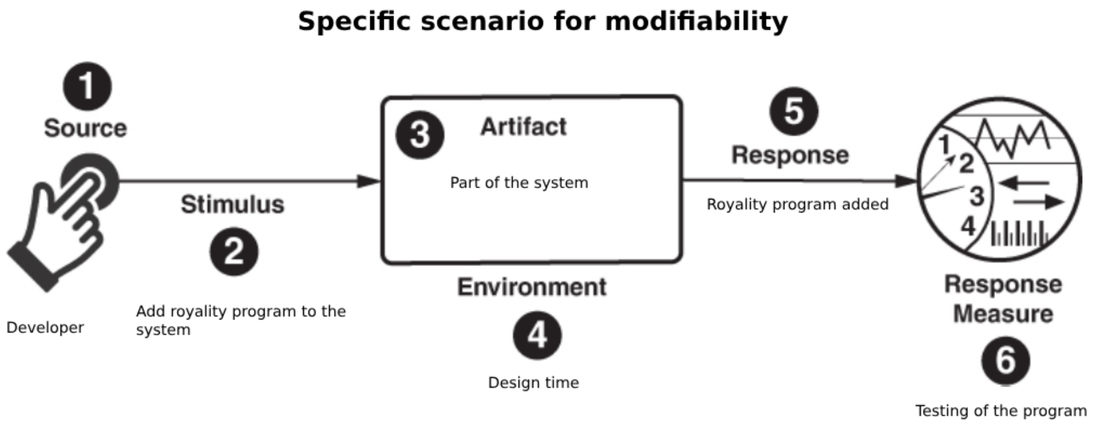


Figure 6: The figure shows a specific scenario for modifiability

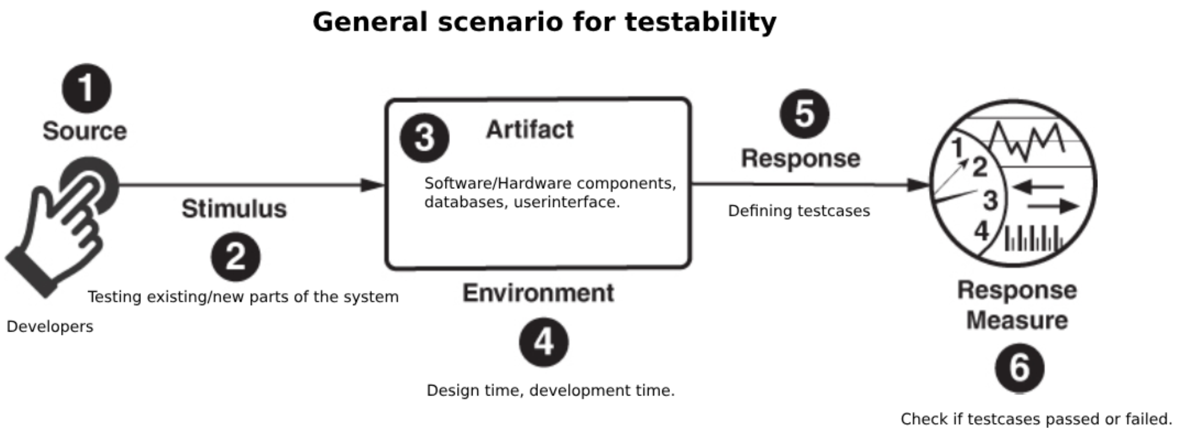

General scenario for testability
1
Source
Developers
Stimulus
2
Testing existing/new parts of the system
3
Artifact
Software/Hardware components, databases, userinterface.
Environment
4
Design time, development time.
5
Response
Defining testcases
Response Measure
6
Check if testcases passed or failed.

Figure 7: The figure shows a general scenario for testability

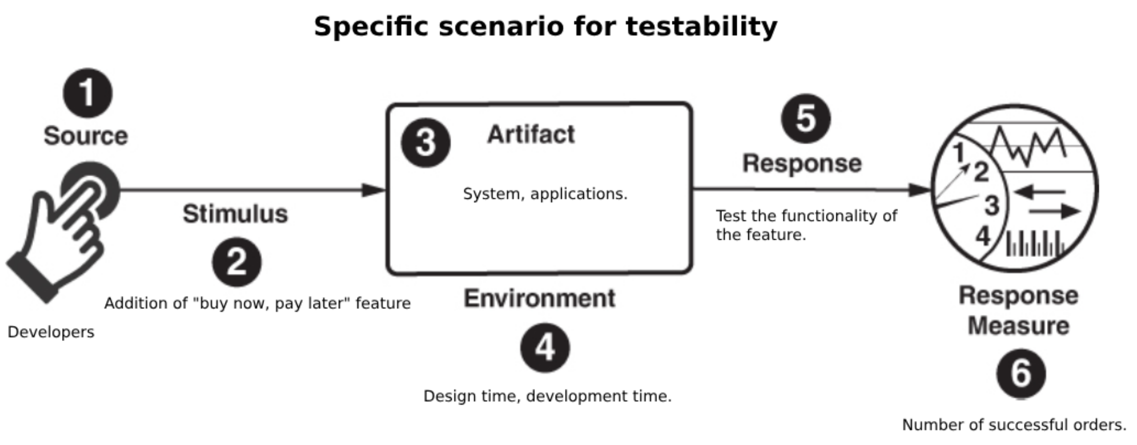

Specific scenario for testability
1 Source
Developers
Stimulus
2
Addition of "buy now, pay later" feature
3 Artifact
System, applications.
Environment
4
Design time, development time.
5
Response
Test the functionality of the feature.
Response Measure
6
Number of successful orders.

Figure 8: The figure shows a specific scenario for testability

1.2.3 Utility tree

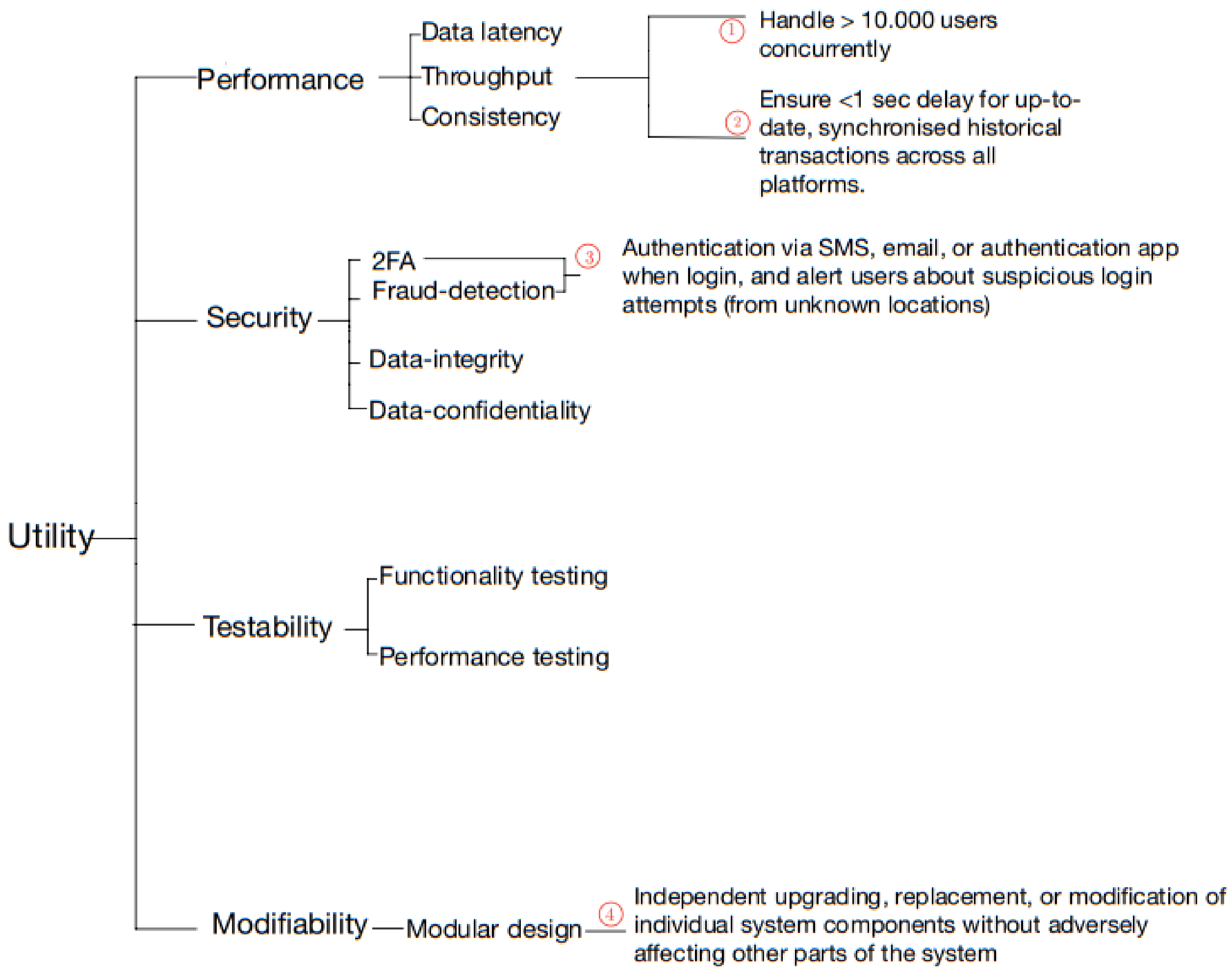
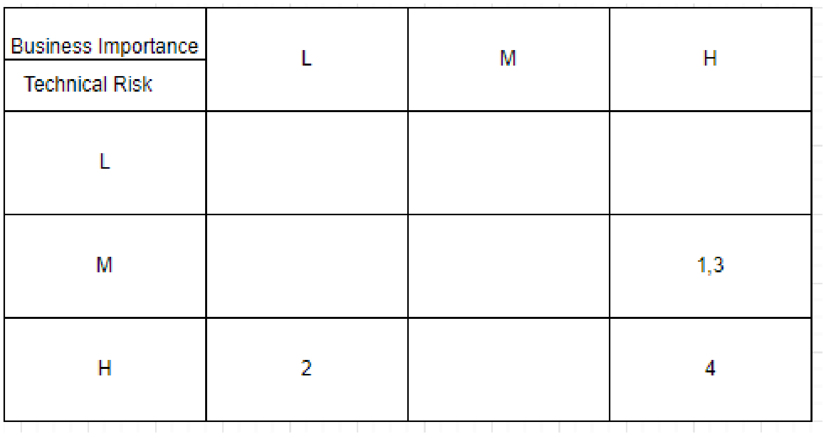

Utility
Performance
Data latency
Throughput
Consistency
1
Handle > 10.000 users concurrently
2
Ensure <1 sec delay for up-to-date, synchronised historical transactions across all platforms.
Security
2FA
Fraud-detection
3
Authentication via SMS, email, or authentication app when login, and alert users about suspicious login attempts (from unknown locations)
Data-integrity
Data-confidentiality
Testability
Functionality testing
Performance testing
Modifiability
Modular design
4
Independent upgrading, replacement, or modification of individual system components without adversely affecting other parts of the system

Figure 9: The figure shows the utility tree of the system



Business Importance / Technical Risk	L	M	H
L			
M			1,3
H	2		4

Table 1: The table shows the business value of the candidate scenario and the technical risk of achieving it

1.3 Primary Functionality

UC1: Shipment tracking
UC2: Alert suspicious login attempts
UC3: Extend due date
UC4: High-capacity
UC5: Dynamic Credit Limit Allocation

1.3.1 Use Case model

Use Case Nr	Use Case Name	Description
UC1	Shipment tracking	The user will be able to track the status of the shipment, and when and where the shipment will arrive.
UC2	Alert suspicious login attempts	Notify the customer through SMS or email in the event of a login attempt from an unfamiliar device or following a few incorrect password entries.
UC3	Extend due date	Possibility to push the due date for payment x days for extra cost.
UC4	High-capacity	The system should be able to handle more than 10,000 customers concurrently.
UC5	Dynamic Credit Limit Allocation	Offering a dynamic credit limit allocation based on users' credit histories or shopping behaviors.

Table 2: The table presents the primary use cases with their description

1.3.2 Use Case Diagram

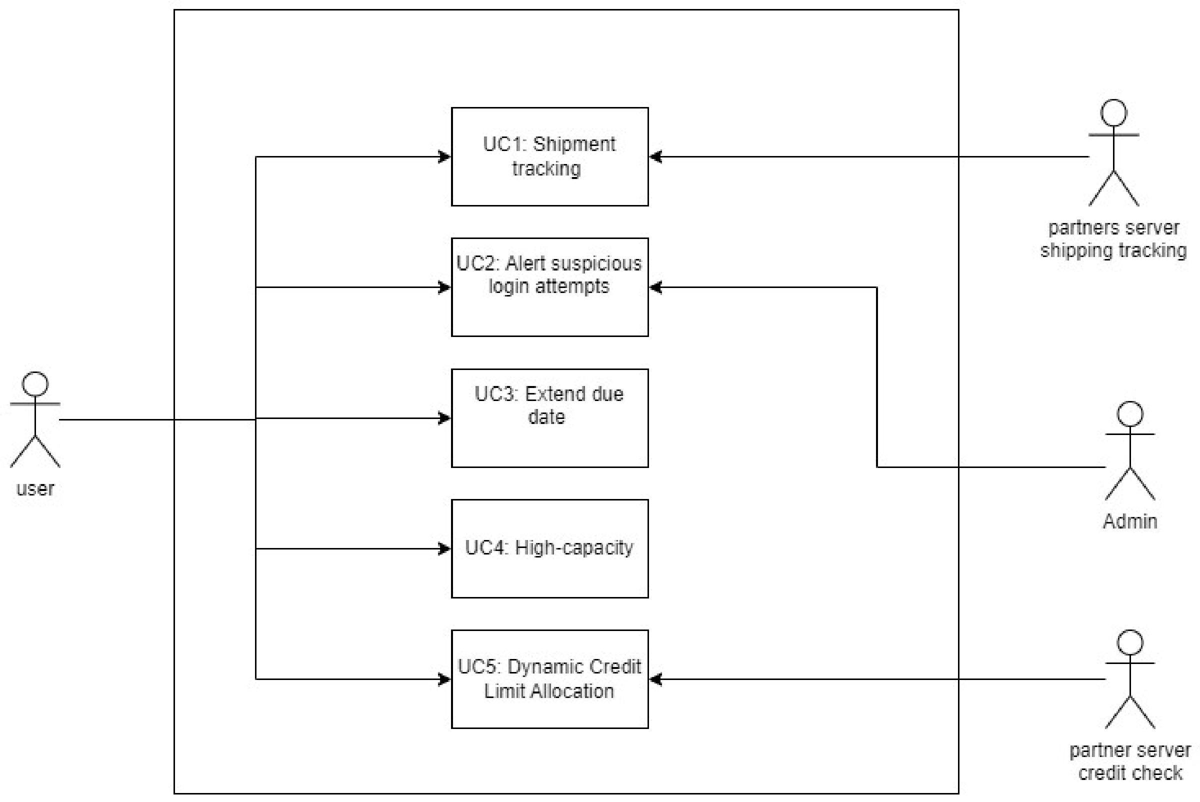

UC1: Shipment tracking
UC2: Alert suspicious login attempts
UC3: Extend due date
UC4: High-capacity
UC5: Dynamic Credit Limit Allocation
user
partners server shipping tracking
Admin
partner server credit check

Figure 10: The figure shows a use case diagram

1.3.3 Use Case Analysis

1.3.3.1 UC1: Shipment tracking

The company wants to add package tracking to its growing list of services. Discussing this addition now is important because it aligns with Sagittarius Bank AB's current goals of enhancing availability and being user-friendly. Talking about this now ensures that the system is designed with a prime focus on user experience, making it a tool that boosts customer satisfaction and aids the bank's success.

Incorporating this service at this point in the planning process also lets the team think ahead about system performance and scalability. They can predict and plan for a potential increase in user activity to guarantee that the system can handle more people using this service without any issues. Bringing this topic to the table now is a step towards recognizing and addressing possible security concerns at the right time, aligning with the bank's dedication to securing additional data that comes with package tracking. Showing when a payment is due based on the shipment status through the "Buy Now Pay Later" option, works towards the bank's vision.

1.3.3.2 UC2: Alert suspicious login attempts

It is important to discuss this use case now as part of the architectural process because Sagittarius Bank AB is taking steps to grow, and it wants to ensure its system is ready to handle new challenges safely and efficiently. Sagittarius Bank AB wants to keep its systems safe from threats and make sure that the system is always up and running. This means putting measures in place to warn users if there's a security risk, to ensure the system operates smoothly at all times, aligning with the bank's goals of strong security and constant availability. UC2 focuses on allowing only verified individuals to access specific data, thereby safeguarding customer information. Addressing these concerns early in the architectural process ensures the bank can build a system that is both secure and able to meet increasing demand.

1.3.3.3 UC3: Extend due date

Discussing this use case during the architectural process is essential for Sagittarius Bank AB, ensuring it aligns with its goals of high availability, user-friendliness, and security. Sagittarius Bank AB is deeply involved in facilitating secure and responsive money transactions and offering flexible credit services. A significant aspect of this is providing extended payment durations. To achieve customer satisfaction and build trust, the bank proposes a small fee for customers who are choosing a later payment date, helping them avoid substantial fines associated with missed payment deadlines.

By exploring this use case now, the bank aims to maintain a competitive edge, emphasizing secure and sustainable growth while prioritizing customer satisfaction in the rapidly evolving FinTech branch.

1.3.3.4 UC4: High-capacity

Discussing this use case in the current architectural process is essential as it directly serves Sagittarius Bank AB’s immediate needs for scalability, quick market entry, and security. This directly addresses Sagittarius Bank AB's requirement of not only being able to service an increasing number of customers with low down-time and instant functionality but also ensuring a timely entry into the market to secure a favorable market share. This move leverages the bank's objective of enhancing source availability and modifiability, promoting a cost-effective approach when releasing updates or new features. This strategy is central as it maintains a responsive and operational system.

1.3.3.5 UC5: Dynamic credit limit allocation

Discussing dynamic credit limit allocation during the current architectural process stage is important for Sagittarius Bank AB. It directly supports the bank's commitment to offering a user-friendly and adaptable service. Facilitating secure and informed credit allocations through well-rounded data analysis supports the bank's strict security measures, protecting both the users and the bank's interests. Introducing this use case now, it's a proactive step towards realizing a service that is not only secure and adaptable but also aligns perfectly with the bank's immediate objectives.

1.4 Architectural Concerns

Figure 11 shows a context diagram that describes the information flow between the main system and the external servers.

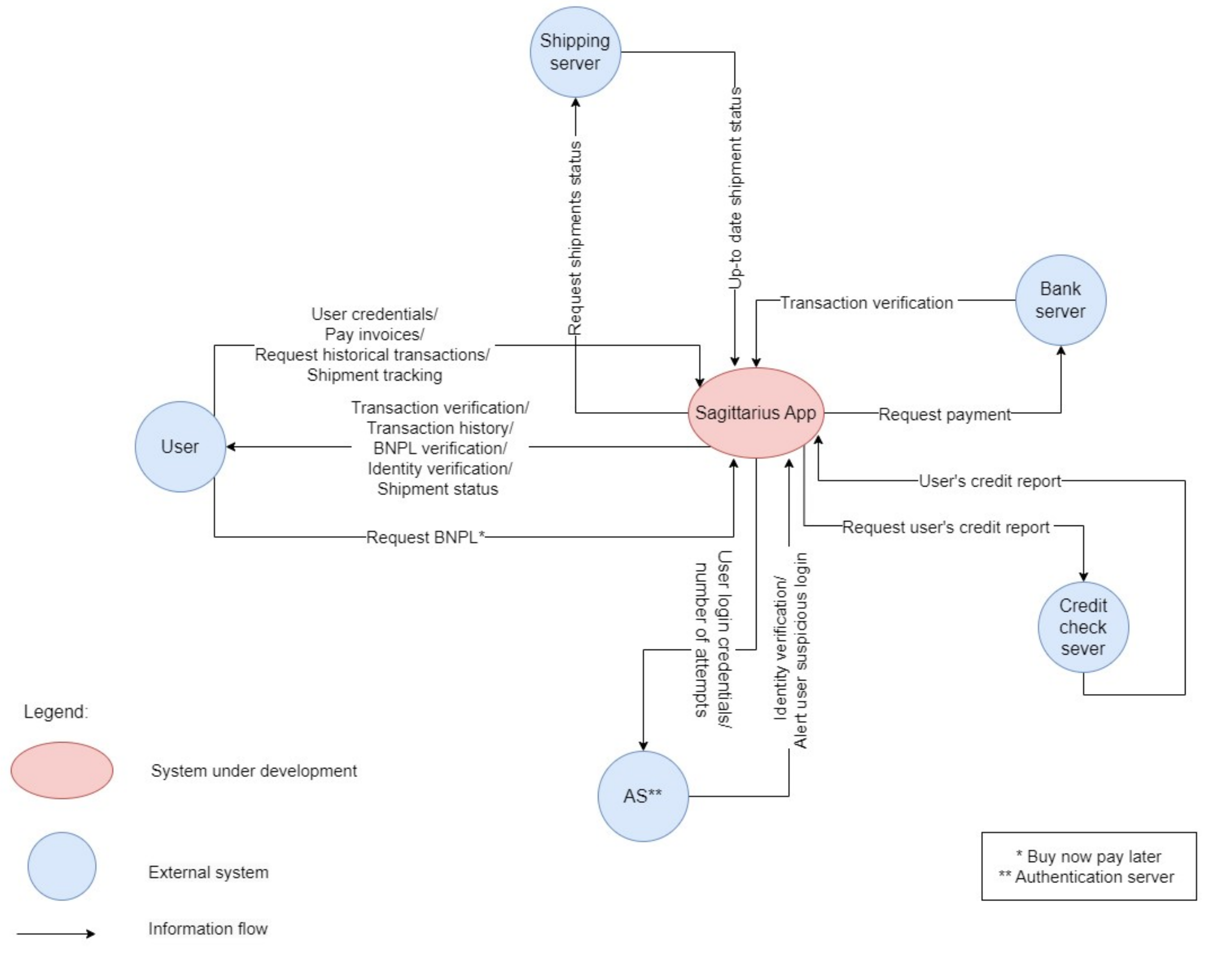

Shipping server
Request shipments status
Up-to date shipment status
Bank server
Transaction verification
User credentials/ Pay invoices/ Request historical transactions/ Shipment tracking
Sagittarius App
Request payment
User
Transaction verification/ Transaction history/ BNPL verification/ Identity verification/ Shipment status
User's credit report
Request user's credit report
Request BNPL*
User login credentials/ number of attempts
Identity verification/ Alert user suspicious login
Credit check sever
AS**
Legend:
System under development
External system
Information flow
* Buy now pay later
** Authentication server

Figure 11: Context diagram that describes the information flow

CRN-1: (Authentication -) Session Management Flaws

After a user logs in, they usually start a session that lets them keep using the system without needing to log in again for a while. Managing these sessions well is very important to keep the system safe. Sometimes there can be problems in how sessions are managed, like "session hijacking," where bad actors can take over someone's session and pretend to be them. This can happen more easily if the session information is sent over unsafe channels, increasing the risk of unauthorized access.

CRN-2: (Authorization -) Detailed Error Messages

In many cases, systems can give away too much information in their error messages, which can accidentally share secure details with the user. This is a big security problem. It is important to create error messages that don't share sensitive information but still help the user understand what the issue is. This means finding the right balance in how we design error messages to keep the system secure while still being helpful to users.

CRN-3: (Communications -) API Security

As the system plans to offer package tracking and shopping suggestions, it will likely involve integrations with external systems via APIs. The security of these APIs is a significant concern because they could potentially be exploited to gain unauthorized access to the system or to leak data.

CRN-4: Compliance with Legal Requirements

The fintech sector is heavily regulated, and communications involving financial transactions and personal data are subject to numerous legal requirements. Non-compliance not only poses a risk of legal repercussions but can also affect the company’s reputation.

CRN-5: Complexity in Managing Multi-platform Releases

Developing native apps for all popular mobile operating systems introduces complexity in maintaining parity across different platforms during updates and the release of new features.

CRN-6: Response Time

Ensuring low downtime and instant functionality under the pressure of increasing user numbers is a primary concern. A delay in response time during transaction processing could result in user dissatisfaction and harm the service’s reputation.

1.5 Constraints

CON-1: Platform Compatibility

The system should ensure compatibility with all major mobile operating systems, in order to reach a wide range of users across various platforms.

CON-2: Security and Data Privacy

The system needs strong security measures to keep transaction data and user information safe. Since it deals with sensitive customer data, ensuring a high level of security. When it keeps this data safe, it builds trust and a good reputation for the service, which in turn attracts more customers.

CON-3: Seamless Updates and Integrations

The system should facilitate seamless implementation of new features and updates without causing disruptions to the user experience.

CON-4: Choice of DBMS and Programming Language

The development team needs to choose the most suitable DBMS and programming language considering the specific context and requirements of the system. In this case, SQL is suitable to store the system data, and when it comes to the programming language the most suitable one is React Native. This programming language allows the developers to build native mobile apps for both iOS and Android using a single codebase[2].

CON-5 : Cost-Effectiveness

The system must be designed to enable the effortless integration of new functionalities and updates while minimizing any adverse impact on the user's overall experience.

CON-6: Open-source Systems/Technologies

The source code of the system must remain closed and not open-source to eliminate any chances of finding potential vulnerabilities that could be used to disrupt the system. In which it leads to an increase in the user trustability in the system.

[2] *Baruah, B. (2023, July 3)*

2 Part 2: Software Architecture Description

2.1 Primary presentation

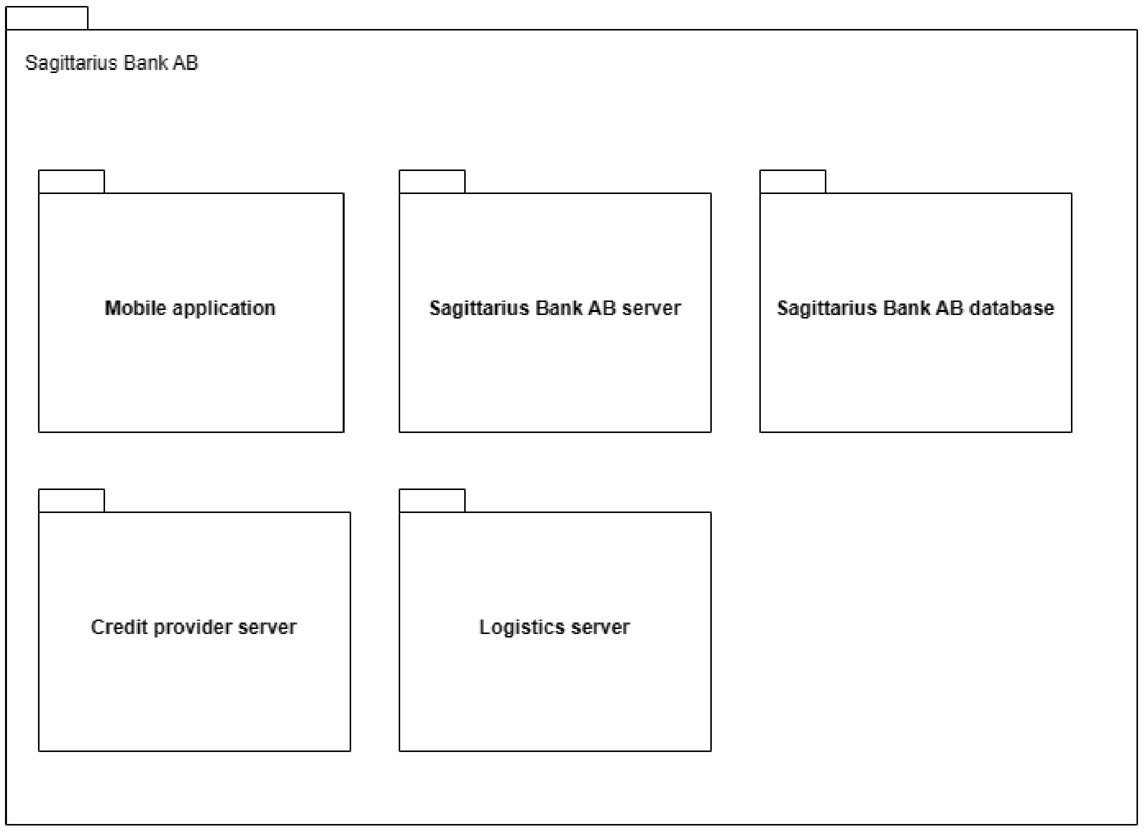


Figure 12: Decomposition view

2.2 Element catalogue

2.2.1 Elements and their properties

Element	properties
Application	- Depending on the platform (Android, Ios, etc...)
Sagittarius Bank AB server	- High-speed multi-core processors to handle a large volume of transactions and concurrent users - High-capacity SSDs to store a large volume of data securely and to facilitate fast data retrieval - A secure and stable operating system optimized for server environments, with regular updates to ensure security and performance - Implement load-balancing solutions to distribute traffic

	evenly across servers, ensuring high availability and reliability
Logistics server	- The server can handle a large number of requests without going down. - Capability to provide real-time updates on the shipment status to maintain data accuracy. - Data in transit and at rest is encrypted to maintain data security.
Credit provider server	- The data transmitted between the system and the credit provider server is encrypted to maintain confidentiality. - The server can quickly respond to requests to maintain a smooth user experience. - The server is operational and accessible at all times, minimizing downtimes.
Sagittarius Banks AB database	- SQL database

Table 3: The table shows elements and their properties

2.2.2 Relations and their properties

Relations	**properties**
Application & Sagittarius Bank AB server	- The application sends/receives requests/responds to/from the Sagittarius server to handle users' accessibility to the service, such as logging in/out, making transactions, extending payments due dates, etc.
Sagittarius Bank AB server & Logistics server	- The Sagittarius server sends/receives requests/responds to/from the logistics server about the status of the shipments.
Sagittarius Bank AB server & Credit provider server	- The Sagittarius server sends/receives requests/responds to/from the Credit provider server about the user's credit score.

Table 4: The table shows the relations between the elements

2.2.3 Element interfaces and behavior

To execute a payment transaction, a series of steps, as shown in the UML sequence diagram see Figure 13, must be undertaken. Initially, the user is required to log into the application. Following this, Sagittarius Bank AB's server will validate the login request by referencing its database. Once authenticated, the user can initiate a payment transaction. Then, the system will communicate with the credit provider's server to ascertain the user's credit status. If the user's credit standing is satisfactory, the transaction will be processed, subsequently triggering the dispatch of a shipment tracking link to the user. The system will ensure all steps have been successfully completed before allowing the user to logout.

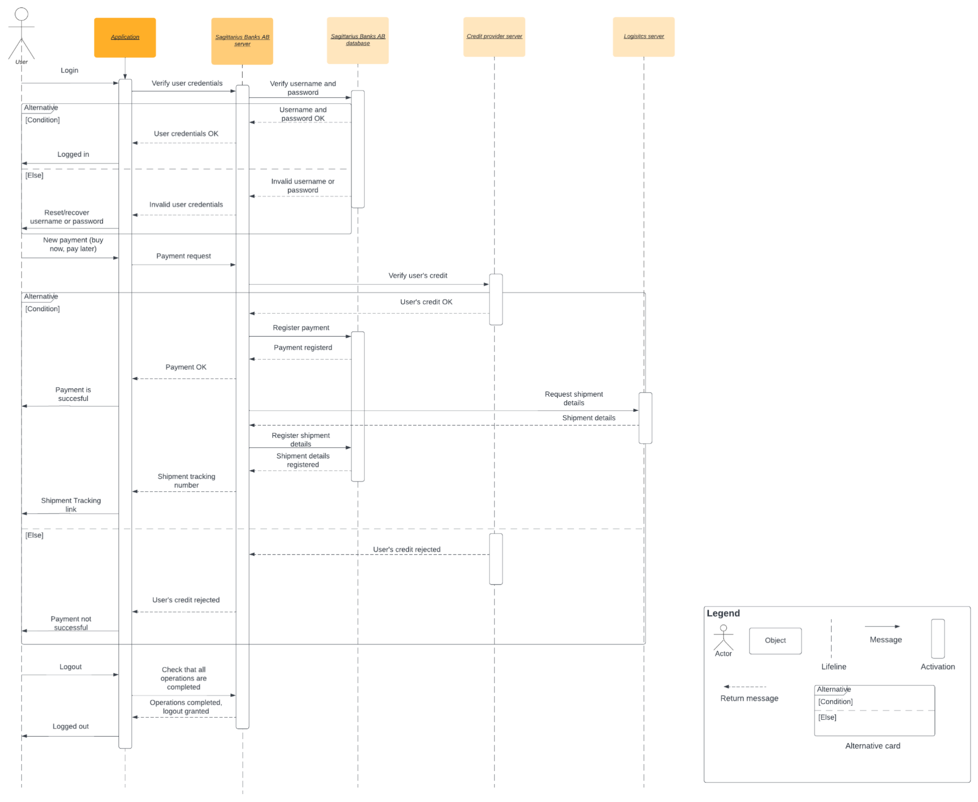


Figure 13: UML sequences diagram

2.3 Context diagram

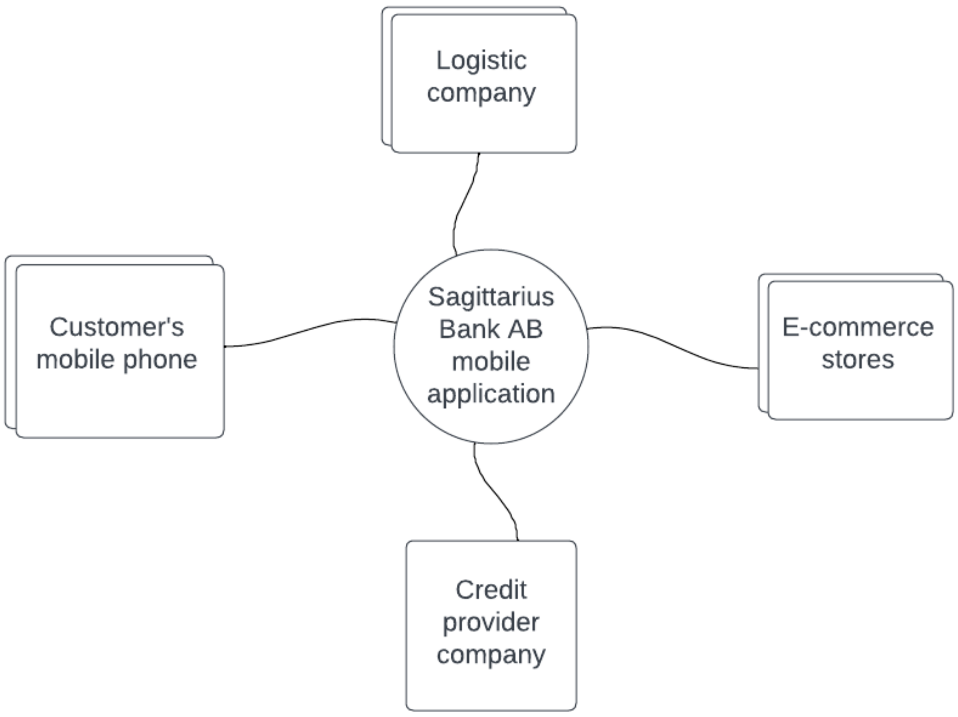


Figure 14: The context diagram of Sagittarius Bank AB application

2.4 Variability guide

- **Adaptability:** As the system collaborates with external servers to offer various features to users, it is essential for the system to be able to interpret different response types from these external servers without necessitating any modifications to the source code, even if changes have been made to the external servers. For instance, if an external server transitions from providing data via HTTP responses to delivering data through API responses, the system should be able to adapt seamlessly to this change without requiring modifications to the source code.

- **Availability:** If an external server, such as a credit server, shuts down for any reason, the system should address this issue seamlessly for the user. It should provide the user with an alternative payment option until the server is operational again.

- **Scalability & Performance:** The system database should be designed with scalability in mind, to ensure that it can expand its capacity to manage higher transaction volumes and increased data storage requirements as user numbers increase. This scalability should guarantee consistent system performance even under heavy loads.

- **Availability, Performance & Scalability:** Implementing load balancers is important, as they help to divide incoming requests including unexpected ones, across multiple servers. By doing so, the system can maintain high availability and prevent any individual server from becoming a performance bottleneck.

2.5 Rationale

A decomposition style was chosen as a module view style. The style provides a clear breakdown of the different modular components that comprise the system[3]. This is useful for Sagittarius Bank AB as it has multiple distinct components such as payment processing, package tracking, authorization, and authentication. This style presents a hierarchical organization of modules, making it clear what sub-modules or components exist within broader modules. For instance, the app development for different platforms will be delineated under a broader “Mobile Application”.

Achieved	**Tactic**
UC2 & QA1	2FA
UC4 & QA2	Powerful components/servers
QA3	Modular system
QA4	Unit tests
UC1	External APIs from logistic companies to request/receive tracking information
UC3	Pay-to-extend strategy to offer flexibility in extending due dates
UC5	External API from a credit provider company to provide us with user credit score

Table 5: The table shows the tactics made to achieve specific use cases/quality attributes

[3] *Clements, P., Bachmann, F., Bass, L., & Garlan, D. (2010).*

References:

1. *Bass, L., Kazman, R., & Clements, P. (2021). Software Architecture in Practice (4th ed.). Addison-Wesley Professional.* [*https://learning.oreilly.com/library/view/software-architecture-in/9780136885979/*](https://learning.oreilly.com/library/view/software-architecture-in/9780136885979/)
2. *Baruah, B. (2023, July 3). Best Programming Languages For Mobile App Development. Taazaa.* [*https://www.taazaa.com/best-mobile-programming-languages/*](https://www.taazaa.com/best-mobile-programming-languages/)
3. *Documenting Software Architectures: Views and Beyond (2nd ed.). Addison-Wesley Professional.* [*https://learning.oreilly.com/library/view/documenting-software-architectures/9780132488617/*](https://learning.oreilly.com/library/view/documenting-software-architectures/9780132488617/)