Linnéuniversitetet
Kalmar Växjö

Project - Farmers Market

Software Architecture

Course code: 2DT902
Names: William Abrahamsson, Khaled Matar, Martin Fontin, Theo Danvert
Emails: wa222dt@student.lnu.se
km222uq@student.lnu.se ,
mf223ub@student.lnu.se ,
td222gp@student.lnu.se

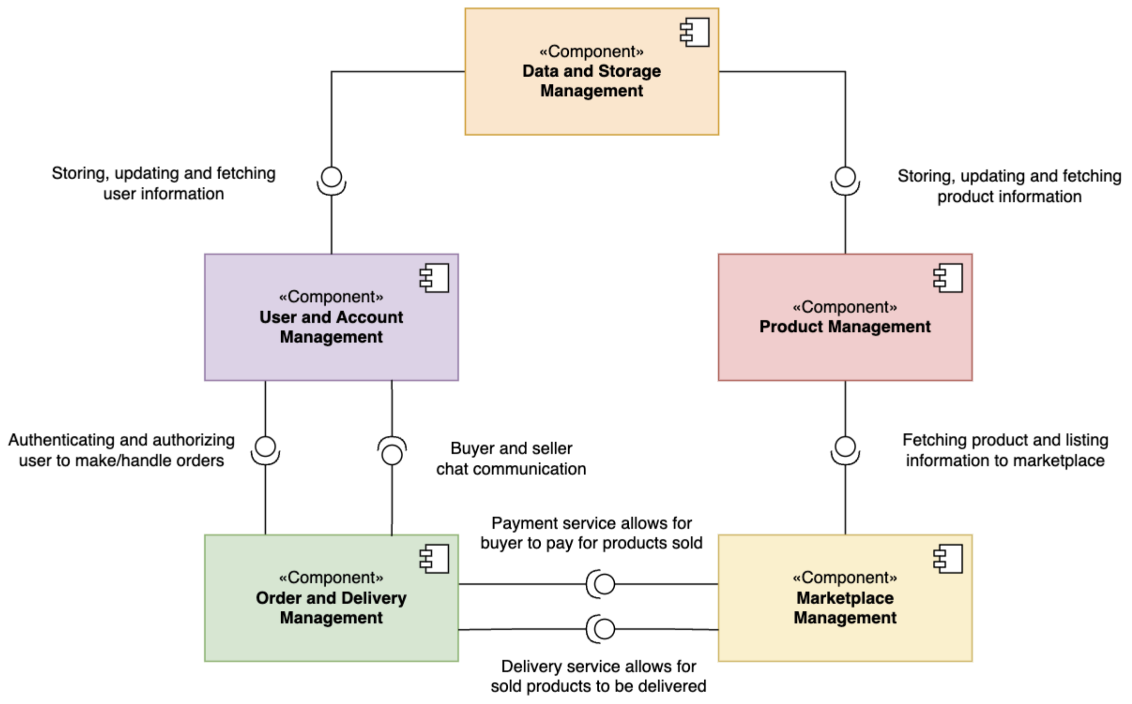
Task 1

The system combines features for online shopping with five components building out the system. The data and storage management component stores, updates and fetches product data to the product management system, and user data to the user and account management system.

The product management system then provides product data to the marketplace system. This could for example be specific product information, listings, dates etc. Anything that has to do with the marketplace and products.

The user and account management system provides authentication and authorization to the order and delivery manager. This allows the user to be authorized access before making any orders or managing their own orders. The order management also provides chat communication to the user and account manager so that buyers and sellers can communicate via direct chat messages.

Finally the marketplace manager provides a payment service to the order and delivery management, along with a delivery service. This allows products to be sold, paid for and delivered. Both the delivery and payment services send feedback back to the marketplace manager allowing it to see whether a product has been paid for and whether it has been delivered.


«Component»
Data and Storage Management
Storing, updating and fetching user information
Storing, updating and fetching product information
«Component»
User and Account Management
«Component»
Product Management
Authenticating and authorizing user to make/handle orders
Buyer and seller chat communication
Fetching product and listing information to marketplace
Payment service allows for buyer to pay for products sold
«Component»
Order and Delivery Management
«Component»
Marketplace Management
Delivery service allows for sold products to be delivered

Task 2

Software Security

Security in software is all about keeping all parts of a software safe from bad actors but also making sure that the users of your software are not affected by the security measures put in place. When making a software secure you have to make sure users are who they say they are (Authentication), deciding what type of access a user has (Authorization), keeping sensitive data safe (Confidentiality), Making sure the data doesn't get manipulated (Integrity) and keeping the system up and running with ease (Availability).

Combined Quality Attribute Scenarios

Quality attribute scenario 1: Unauthorized access to admin dashboard

- **Source**: An unregistered user from a known IP address.
- **Stimulus**: Attempts to access admin-only resources.
- **Environment**: Online, Normal operation.
- **Artifact**: Admin dashboard.
- **Response**: System checks if the user is authorized to view the artifact.
- **Response Measure**: Redirect to "Permission Denied" page in 1 second or less.

Quality attribute scenario 2: User attempts to edit data inside another users profile

- **Source**: A known user.
- **Stimulus**: Attempts to edit data inside other users profiles.
- **Environment**: Online, Normal operation
- **Artifact**: Profile page
- **Response**: System validates if user is authorized to edit that data, And denies access.
- **Response Measure**: Redirect to "Permission Denied" page in 1 second or less.

Quality attribute scenario 3: An unauthenticated users attempt to access an account

- **Source**: An unauthenticated user
- **Stimulus**: Attempts to login to an account.
- **Environment**: Online, Normal operations.
- **Artifact**: Login page.
- **Response**: The system validates if the users provide the correct username/email and respond accordingly.
- **Response Measure**: Logs the login attempt, if unsuccessful..

Quality attribute scenario 4: Unauthorized attempt to access user data during payment

- **Source**: Unknown entity.
- **Stimulus**: Attempts to snoop on user payment data.
- **Environment**: Online, Normal operation
- **Artifact**: Payment page
- **Response**: Web Page encryption for data before transit hinder unauthorized access.

- **Response Measure**: None

Quality attribute scenario 5: Normal users tried to give 5 star rating for product

- **Source**: A known user.
- **Stimulus**: Attempts to give a 5 star rating for a product that they have not bought.
- **Environment**: Product page.
- **Artifact**: Rating system.
- **Response**: The system checks if the user has previously bought the item and detects that they are not allowed to give any feedback on that product.
- **Response Measure**: Sends users an error.

Quality attribute scenario 6: Unauthorized Bulk Creation of Seller Accounts

- **Source**: An unregistered user from a known IP address.
- **Stimulus**: Creation of various seller accounts in a short time period.
- **Environment**: Account creation page.
- **Artifact**: Account database.
- **Response**: Requires social media verification after two accounts are created.
- **Response Measure**: Social media verification prompt appears immediately after second account creation.

Task 3 - Design Alternatives

Here we have two strategies on how to solve the problems from Task 1 and 2. The strategies solve the same problem with different solutions.

Strategy 1 (Preferred)

Preventing unauthorized access

Implementing a robust user authentication mechanism. Where users are required to have a secure password. We can also provide the option for users to have 2-step authentication when logging in from a new device or ip address. This makes it a lot harder for unauthorized users trying to access another user account. And by allowing users to use a social media account to login to our service we can make it easier for users to login to our site, This also reduces some of the liability for the company since we don't have to store those users passwords on our servers.

Proper permissions

Attribute-Based Access Control (ABAC) grants access rights dynamically based on attributes such as user characteristics, resource types, actions, and contextual factors. For instance, in ABAC, a rule could let only 'seller' users change product listings, or allow access to money records just for 'finance department' users during work hours. This detailed approach makes ABAC very adaptable and secure for different and changing access needs.

Securing payment data

Third parties could be Stripe, Paypal or Klarna. This lays the responsibility on third parties to handle payment information safely and securely. Since payment providers specialize in payment processing they invest heavily in making sure that the payment data is secure. These providers operate using secure and resilient infrastructure, that has data backups, and a plan if something fatal goes wrong. Third parties also provide a scalable solution that ensures that security measures remain effective even as transaction volumes increase. This also reduces the liability for the company if a data breach were to happen.

Preventing bulk account creation

We can use bot prevention tactics, such as ReCapcha, and monitor traffic in real-time to be able to analyze it. This way we can know if an IP is creating multiple accounts fast and can catch it early.

Strategy 2

Preventing Unauthorized access

Firewalls can be configured to block traffic from certain IP addresses, restrict access to specific ports, and monitor and report suspicious activities. This is particularly useful in protecting sensitive data and functions of the Farmers Market platform from external threats such as hackers, malware, and other cyber attacks adding an extra layer of protection for sensitive data and important functions of the platform.

Proper Permissions

Role-Based Access Control (RBAC) assigns access rights based on predefined roles within an organization, unlike Attribute-Based Access Control (ABAC) which grants access dynamically based on multiple attributes like user characteristics and contextual factors. RBAC simplifies permission management by linking access to specific roles, but lacks the granularity and adaptability of ABAC, which can consider a broader range of attributes for more nuanced access control.

Securing payment data

We can make sure to use HTTPS to encrypt all traffic that is sent to and from the server, this will ensure that the payment data is encrypted and safe when sent. This also ensures that any man-in-the-middle attempt is unsuccessful. By encrypting the data the risk of data breaches is reduced.

Prevent bulk creation of accounts.

The system can use IP limiting /rate limiting to prevent users from creating multiple accounts. When the system detects that a user is creating more accounts from the same IP address. We can apply a threshold for the number of accounts a user is allowed to create from the same IP address. This helps prevent automated scripts from creating large numbers of accounts quickly.

Task 4

Strategy 1

This is our preferred strategy, and to implement this strategy we need the following three components.

Attribute-based access control System (ABAC) component
Unlike RBAC, which assigns permissions based on predefined roles, ABAC uses a set of policies that evaluate attributes (or characteristics) of users, resources, and the environment to make access decisions.

attributes can include a wide range of factors, such as user attributes (e.g., age, job role, location), resource attributes (e.g., file types, product categories), and environmental attributes (e.g., time of day, current transaction volume)

ABAC handles the following scenarios:
"Normal users tried to give a 5-star rating for product" By checking if the user in question has permission to give the 5-star rating.
"An unauthenticated users attempt to access an account" Checks if the user provides the correct login info.
"User attempts to edit data inside another users profile" By Checking if the user in question has permission to edit the data in question.
"Unauthorized access to admin dashboard" Checks if the users should be allowed to view the dashboard.

Recaptcha System component
This system component is responsible for preventing mass creation of user accounts. Since a bot can't successfully do re-captchas, this will limit the ability to create fake accounts at scale. This component provides its services directly to the user and account management system.

The Recaptcha component handles the scenario
"Unauthorized Bulk Creation of Seller Accounts" The Recamptcha component applies the Recaptcha to prevent an automated system from creating multiple seller accounts.

3rd party payment verification system component
This component is used to handle payments in the system. The component is connected to the marketplace management component which then provides payment services to the order manager. By using a 3rd party payment system we make sure that the payments are safe and secure.

This component handles the scenario "Unauthorized attempt to access user data during payment" This scenario is handled by the third-party payment providers.

Pros:
- Utilizing third-party payment providers boosts security in the payment management system. It allows the system to handle payments without having to manually give the payment information to the bank to make transactions.

- ABAC offers greater flexibility and security than RBAC by allowing for access decisions based on a variety of attributes, such as user characteristics, environment, and resource types. RBAC used pre-defined roles within the organization to grant access. Since we will be doing a lot of authorization to different parts of the system we need the dynamic aspects of the ABAC method.
- Recaptcha systems are generally good at preventing the majority of bot attacks. This greatly increases the ratio of real accounts to fake bot accounts on the system.

Cons:

- AI can work around the re-captcha in certain situations. Especially in todays age when LLMs are growing exponentially.
- Adding third-party payment providers can add complexity to the system.
- ABAC is complex in policy management and performance overhead. Due to ABAC's granular and dynamic nature, designing and maintaining a comprehensive set of policies can be challenging, potentially leading to administrative overhead and increased chances of misconfiguration. Additionally, the real-time evaluation of multiple attributes can result in higher computational load and slower access decision times compared to simpler models like RBAC.

Strategy 2

Role-based access control system (RBAC) component
This component is used for authorizing the correct things for the specific type of user account that the user is logged in to. This component provides its services directly to the user and account management system. Since this component handles authorization in the system. This also meets the criteria for preventing unauthorized access.

RBAC handles the following scenarios:
“Normal users tried to give a 5-star rating for a product” By checking the users role and seeing if that user is allowed to give the 5-star rating.
”User attempts to edit data inside another users profile” Checking if the user is actually editing their own data or not.

IP limiting / Rate limiting component
This component is used to limit certain IP addresses from certain actions on the server. This component can be used for multiple reasons, for example when a user has failed multiple login attempts. Or when the system detects a DDoS attempt. This component is in general responsible for handling the blockage of IP addresses.

IP limiting handles the scenarios
“Unauthorized access to admin dashboard” By blocking the IP address from attempting to access the resource again.
“Unauthorized Bulk Creation of Seller Accounts” By applying a rate limit on the IP address attempting to create multiple accounts.

Security component
This component is responsible for securing both data on the server and data that are being transferred between the server and users, The component uses both the firewall and secure HTTP to prevent unauthorized access to certain resources on the server. And encrypting all data sent to and from the server.

”Unauthorized attempt to access user data during payment” Ensuring the data is encrypted, prevents the attacker from reading the data that is being sent.
“Unauthorized access to admin dashboard” By blocking all IP addresses trying to access the dashboard except the admins IP address.

Pros:
- Using the firewall to only allow a specific IP address to access important resources, is a very efficient way to prevent attacks from the outside.
- IP rate limiting is an efficient way of preventing the automation of account creation.
- RBAC is a very efficient way of giving access depending on predefined roles.

Cons:
- Users could use a VPN/proxy to avoid the IP rate limiting for user account creation.
- Managing RBAC definitions could grow complex as the system grows, requiring ongoing management and updates
- RBAC can be difficult to implement in a large and complicated system where permissions are not always static and specific to organizational roles and more dynamic and changing depending on the specific permissions granted for that user.

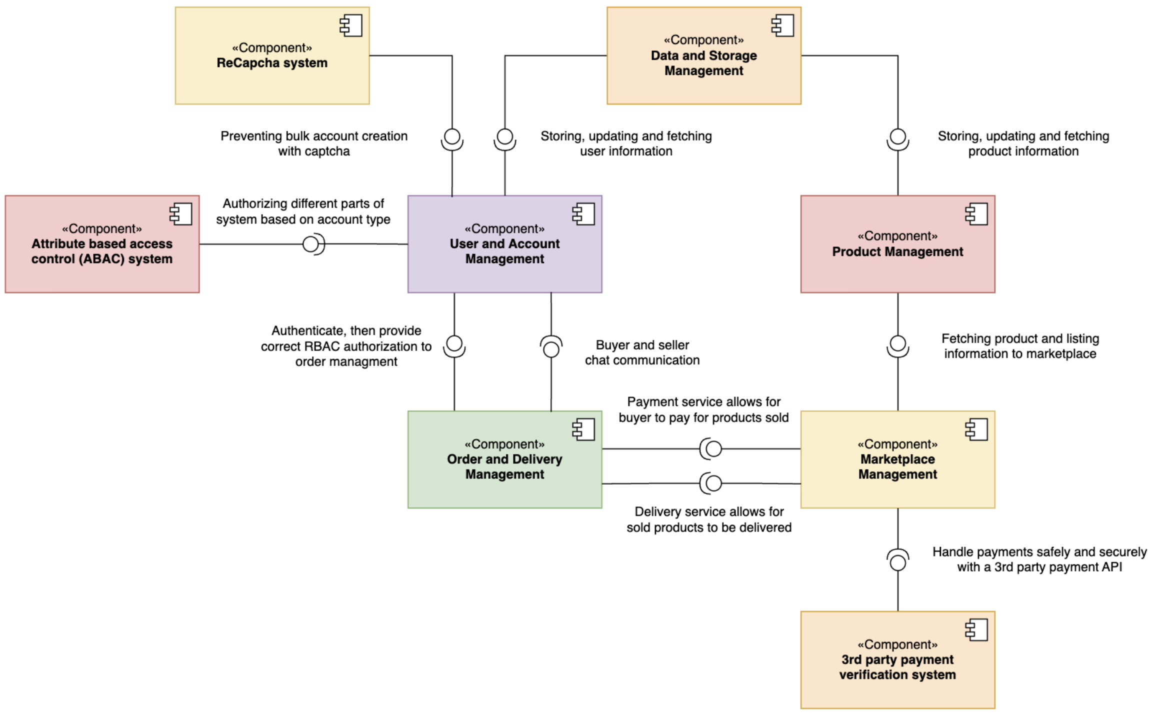
Preferred Strategy: Strategy 1

Strategy 1 was our preferred strategy mainly because ABAC is a lot more dynamic compared to Strategy 2's RBAC component. Instead of granting system access based on roles in the organization ABAC is attribute-based instead of role-based. This means that more attributes than just a users role in the Farmers market system is used to decide whether they will be authorized or not for a specific part of the system. The other reason is that strategy 1 has a third-party payment provider that handles payments in our system in a safe and secure way. Not only don’t we have to do this ourselves, but third-party payment providers also provide multiple different payment methods for the customers. Lastly the ReCaptcha system is easy to implement and provides an extra layer of security.

Task 5

Changes made to the original diagram according to Strategy 1:

- Provide an Attribute-based access control system (ABAC) to handle authorization to accounts. This makes sure that each account has the correct level of access.
- Provide a ReCaptcha system to the user and an account management system to prevent mass account creation.
- Provide 3rd party payment verification to the marketplace management system that then provides payment services to the order and delivery management. This allows for secure and verified payments through a provider like Stripe or Klarna.


«Component» ReCapcha system
«Component» Data and Storage Management
Preventing bulk account creation with captcha
Storing, updating and fetching user information
Storing, updating and fetching product information
«Component» Attribute based access control (ABAC) system
Authorizing different parts of system based on account type
«Component» User and Account Management
«Component» Product Management
Authenticate, then provide correct RBAC authorization to order managment
Buyer and seller chat communication
Fetching product and listing information to marketplace
Payment service allows for buyer to pay for products sold
«Component» Order and Delivery Management
«Component» Marketplace Management
Delivery service allows for sold products to be delivered
Handle payments safely and securely with a 3rd party payment API
«Component» 3rd party payment verification system

Task 6

User (Seller)
A logged-in user needs the following kinds of logging information.
Info that highlights user certain actions such as a successful sale or receiving feedback from buyers, Warning that alerts the user of potential issues such as attempt to sell a product without inventory, Also errors showing for example failed attempts to list a product or unsuccessful changes to existing listings. Lastly they receive logging of fatal stuff for example if a seller is unable to access their dashboard or other major issues.

User (Buyer)
A logged-in buyer needs the following logging information. Info that a successful purchase has been made or feedback given to the seller. Warning that payment information is missing, product out of stock and delay of delivery. Error that transaction failed, issues with order placement and inability to access orders.

Guest
A guest can get the following logging information
info when a guest attempts to sign up and info logging displays what type of information the users has to provide. Warning logs can be displayed when a user types in the wrong password when attempting to log in. Errors can be logged when a bigger problem occurs, for example, a failed attempt to check out as a guest. Lastly fatal logs are generated, when a major part of the webpage is not working, for example if the webpage is not loading in at all.

Developer
A developer gets the following logging information.
Trace as a record of all code modifications or system configurations. Debug for making updates to a test environment and troubleshooting individual components in software. Error when problems such as failed code compilations or system crashes post-deployment occur. Fatal when issues like database corruption and the entire computer system or network has a complete failure.

Administrator
As an administrator, you require comprehensive logging information to effectively manage and optimize our platform. The logs for an administrator include a trace that is used to log every small operation to be able to find bugs during the development phase, The next logging type is debug which is used in the development process to log critical actions like user privilege adjustments and system backups. Normal info logging is also used to display normal activities like administrator changes to the system. Warning logs are used to display warnings of potential site overloads like, to high site load. Error logging is used to log potential errors that the system is experiencing. Lastly, fatal logging is used for major errors such as webpage downtime, and potential attacks.

Architecturally significant requirements

1. The system must log all user activities, including seller actions and buyer interactions, as well as guest usage, ensuring effective oversight and user experience analysis.
   - This will enable administrators and developers to troubleshoot parts of the system in case of failure and allow them to make improvements to the system.
   - Logging also allows administrators to track different quality attributes like performance and make improvements based on that metric.

2. The logging system should track all changes made by system administrators.
   - Since administrators access parts of the system that are not used as often as other parts, it's especially important to log everything.
   - These logs provide valuable information if a corrupt administrator changes something trivial on the server.

3. Stakeholders should be presented with correct and easy-to-understand logs.
   - This allows developers and stakeholders to better understand the problems at hand and make more informed decisions.

4. I/O of the log should not hinder or affect the performance of the system.
   - The logging system will utilize system resources and affect the system. The main point here is to implement the logging in such a way that this won't be noticeable to the users.

5. The logs should be backed up in case of a system failure.
   - This avoids a single point of failure of the logging system.

Task 7

Strategy 1 (Chosen method)

1) **Centralized logging method** (1, 2)
A centralized logging system collects logs from all parts of the system including seller actions, buyer interactions, and guest usage into a single, centralized repository. This approach simplifies management and analysis of logs, making it easier to monitor system health, track user activities, and assess overall performance.

2) **Log visualisation page** (3)
Having a page that can sort logs based on multiple factors such as different log levels, makes it easy for not only developers but also for all stakeholders involved, especially for stakeholders that are less tech-savvy.

3) **Asynchronous logging** (4)
Asynchronous logging refers to the process of logging in a non-blocking manner. In this approach, the logging tasks are executed in the background, independently of the main operational threads of the application. This is achieved by using separate threads or processes dedicated to logging, which can run on their own CPU cores. The primary objective is to reduce the impact of logging on the performance of the core application functions.

4) **Local backup server** (5)
By making sure all logs are backed up on a local server, we prevent any data loss of the logs in the event of data loss, corruption or a system failure. By having the backup locally allows for quick and efficient data recovery and restoration of the logs. It also allows for offline access to the backups, This can be useful in situations where the network or server is compromised.

Strategy 2

1) **Decentralized logging method** (1, 2, 4)
In a decentralized logging system, each component or service within the application maintains its own logging mechanism. This approach allows each part of the system to operate and log events independently, which can be particularly effective in distributed or microservices architectures.

2) **Terminal-based visualisation tool** (3)
A terminal-based approach to viewing system logs means that the system doesn't have a graphical user interface, and all logs are text based in the cmd of the server. This is great for tech-savvy stakeholders but is a real challenge for stakeholders that are not developers.

3) **Cloud-based backup server** (5)
All logging data is sent via a pipeline to a third-party cloud service like AWS or Azure. By backing up the data this way we prevent any data loss of the logs in the event of data loss,

corruption or a system failure. By having the backup in the cloud it becomes more difficult to access, but you don't have to build your own backup server.

Implementation Strategy 1 (Chosen method)

1) **Centralized logging management component**
   This is a centralized component that handles logging in the system. It deploys a system like LOG4j, SLF4J or Winston to establish a structure of the logs. Each level is configured to capture specific types of events: TRACE for finest-grained informational events, DEBUG for general debugging, INFO for routine operations, WARN for potentially harmful situations, ERROR for error events, and FATAL for very severe error events. This helps in filtering logs for relevance and conserves storage by recording only necessary information. This is all done asynchronously from the main parts of the system, allowing logging to happen without blocking higher-priority tasks or processes in the system.

   This component handles the following requirements:
   - ASR 1: By providing full logs of all actions made by seller users, and buyers users.
   - ASR 2: By making sure logs are also stores logs when administrators change trivial stuff inside the system.
   - ASR 4: By making sure the logs are logged asynchronously, and having priority for other tasks make it not block any trivial tasks.

2) **Log visualization management component**
   The Log Visualization Management Component provides a page designed to monitor and analyze system logs. It features a graphical user interface (GUI) that allows stakeholders to easily navigate and manage logs. This component is accessible through a dedicated page, where users can view a comprehensive array of logs in the system. It includes a real-time log display, options to filter logs, analysis tools and a dashboard to combine it all.

   - ASR 3: By providing a page that makes it easy to search, and filter the logs based on different variables.

3) **Local backup server for logs component**
   This component contains a locally hosted backup server on the same network as the main server. Its goal is to keep backups of the logging happening in the system in case of a system failure. This server is backed up automatically and constantly keeps a copy of the logs stored in the main server. This gives the system a new level of security and resilience against data loss.

   - ASR 5: By providing a proper backup system that the system logs are stored in.

Pros:
- A centralized logging management system is easy to expand when new parts of the system is implemented since the base functionality is in place. You just need to add endpoints to the new parts.

- A GUI for viewing logs allows all stakeholders to read the logs compared to the terminal-based alternative.
    - A local backup server allows the data to be accessed quickly in case of a system malfunction.

Cons:

- In a centralized logging system the base functionality is the same for all parts of the logging system. Therefore it’s difficult to create tailor-made logging solutions for specific parts of the system.
- Many advanced developers would want a terminal-based logging system instead of a GUI since it’s faster to use.
- A local backup server is hosted on the same network as the main server. In case of a security breach on the network, both servers are at risk of being compromised.

Implementation Strategy 2

1) **Decentralized logging management component**
   This component allows different services to provide their own unique logs for each system. This allows for more detailed logs for every system, And a more accurate visualisation of what service is causing a potential problem. This kind of decentralized logging mechanism makes it easy to scale if future services get added to the system.

   This component addresses the following ASRs
    - ASR 1: By providing each service with the ability to log every user-based event that happens.
    - ASR 2: By providing the services that administrators use to log every trivial administrator action.
    - ASR 4: By letting the services log on their own make it so that the services can decide when to creator their logs, this prevents any other service from having to wait for a service to be done logging.

2) **Terminal-based logging tool component**
   The terminal-based logging tool, unlike it’s GUI counterpart is used directly in the servers terminal and is controlled via a command line interface. This allows developers to quickly view and manage logs without having to load a web page. It includes a real-time log display, options to filter logs and general analysis tools.

   This component addresses the following ASRs
    - ASR 3: By offering a terminal-based tool to view logs in the system.

3) **Cloud-based backup server component**
   This module contains a cloud-hosted backup server, positioned on a distinct network separate from the primary server, such as an AWS or Azure platform. Its primary function is to maintain backup copies of the system's logs, thereby providing a backup in the event of a system failure. The server is configured to automatically and continuously replicate the log data from the main server. This enhances the system's security framework and protects it against potential data loss, offering an additional layer of protection and reliability.

This component addresses the following ASRs:

- ASR 5: By offering a cloud based backup server solution.

Pros:

- Utilizing a cloud-based backup server can cut down on direct costs since we don't have to buy and integrate a private server. Cloud-based backup servers also has many tools available for handling the data directly in the cloud. This would have to be done manually on a locally hosted server.
- Decentralizing the logging components allows for separate functionality in different parts of the logging system. Allowing for more tailored and sophisticated logs.
- The terminal-based logging tool can be quicker to use for developers who are used to working directly in the command prompt. This could boost developer productivity.

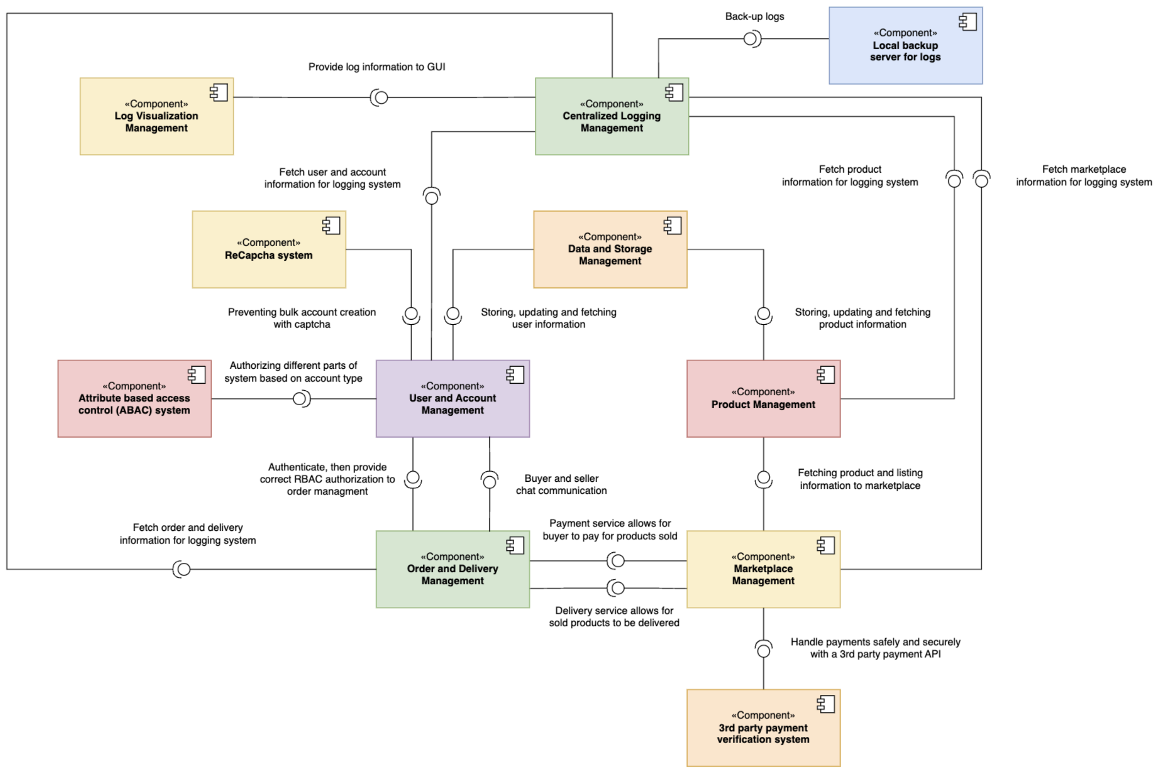
Cons:

- Since a cloud-based server is hosted on another network it takes more time to upload and fetch data from the backup server. This could cause problems in case of a system failure when you quickly need to access the backup logs.
- Since the logging components are de-centralized you have to create whole new components for each part of the system you want to implement logging for. This could be an un efficient use of developer resources.
- A terminal-based logging tool might be quicker to access for a tech-savvy developer, but impossible to access for a normal stakeholder without software engineering knowledge. This drastically reduces the functionality of the logging system since not everyone knows how to use it.

Preferred Strategy: Strategy 1

While Strategy 2 offers robust features such as Cloud-based backup systems, we feel like **Strategy 1** offers better components for our system such as a local-based backup system that benefits Farmers market more. Especially when it comes to having a fast way of restoring the logs in the event of a system failure. Strategy 1 also provides a Log visualization page that compared to a terminal-based logging tool, provides a visual representation of all logs and an easy way to sort the data. And Lastly a Centralized makes it a lot easier since we have a standard for logging that is used every where in our server.

Diagram:


Back-up logs
«Component» Local backup server for logs
Provide log information to GUI
«Component» Log Visualization Management
«Component» Centralized Logging Management
Fetch user and account information for logging system
Fetch product information for logging system
Fetch marketplace information for logging system
«Component» ReCapcha system
«Component» Data and Storage Management
Preventing bulk account creation with captcha
Storing, updating and fetching user information
Storing, updating and fetching product information
Authorizing different parts of system based on account type
«Component» Attribute based access control (ABAC) system
«Component» User and Account Management
«Component» Product Management
Authenticate, then provide correct RBAC authorization to order managment
Buyer and seller chat communication
Fetching product and listing information to marketplace
Fetch order and delivery information for logging system
Payment service allows for buyer to pay for products sold
«Component» Order and Delivery Management
«Component» Marketplace Management
Delivery service allows for sold products to be delivered
Handle payments safely and securely with a 3rd party payment API
«Component» 3rd party payment verification system

Task 8

Roles (What each group member has worked on):

William Abrahamsson:

- Task 1: Order and delivery management and marketplace management.
- Task 2: Quality attribute scenario 3: An unauthenticated users attempt to access an account.
- Task 3: Strategy 1: Preventing unauthorized access, Securing payment data.
- Task 4: Attribute-based access control System (ABAC) component. And Attribute-based access control System (ABAC) component
- Task 5: ABAC system.
- Task 6: Architecturally significant requirements, User (Guest), User (Buyer)
- Task 7: Strategies, implementation of strategies, diagram implementation.

Martin Fontin:

- Task 1: User and account management.
- Task 2: Quality attribute scenario 1: Unauthorized access to admin dashboard, 2 User attempts to edit data inside another users profile 5: Normal users tried to give 5 star rating for product.
- Task 3: Strategy 1: proper permissions, Preventing bulk account creation.
- Task 4: Strategy 1: Recaptcha system, and 3rd party payment system,
- Task 5: Recapcha system.
- Task 6: Architecturally significant requirements, User (Seller), Administrator , Developer.
- Task 7: Strategi 1, implementation of strategie 1. and Preferred Strategy

Khaled Matar:

- Task 1: Data and storage management.
- Task 2: Quality attribute scenario 4: Unauthorized attempt to access user data during payment, Quality attribute scenario 6: Unauthorized Bulk Creation of Seller Accounts
- Task 3: Strategy 2: Preventing Unauthorized access, Proper Permissions
- Task 4: Strategy 1: Pros and cons: strategy 2 por and cons
- Task 5: 3rd party payment verification system.
- Task 6: User (Buyer), Architecturally significant requirements.
- Task 7: Strategi 2, implementation of strategie 2.

Theo Davnert

- Task 1: Product management.
- Task 2: Quality attribute scenario 1: Unauthorized access to admin dashboard
- Task 3: Strategy 2: Securing payment data, Preventing bulk creation of accounts.
- Task 4: Strategy 2: IP limiting / Rate limiting component, and Preferd strategy
- Task 5: 3rd party payment verification system.
- Task 6: User (Guest), User (Buyer), Architecturally significant requirements.
- Task 7: Pros and cons for Strategy 1 and 2.

Project Summary:

This project was not without its challenges. Sometimes, we misunderstood certain aspects of the project, and we had to seek clarification from our teacher. However, once we had the right information, our team worked well together. We divided tasks effectively and managed our time wisely, even while juggling other coursework. This allowed us to create a strong, secure, and scalable software system.

Task 1 - System Overview:
In the first task, we built a comprehensive software system that included user management, data storage, and more. We succeeded by dividing tasks among team members and keeping communication open. This approach ensured that our system met both security and scalability requirements, making it robust.

Task 2 - Software Security:
Task 2 focused on software security. We understood the need for multiple layers of security, addressing unauthorized access attempts and data protection while keeping the system available. We used techniques like IP blocking and permission denial to balance security and user-friendliness, creating a safe and accessible system.

Task 3 - Security Features:
In Task 3, we implemented security features like IP rate limiting, and Re-captcha systems. We chose advanced tools to enhance security without compromising user experience. Our goal was to deter threats and adapt to changing security challenges, resulting in a system that's both secure and user-friendly.

Task 4 - Security Architecture:
Task 4 involved comparing two different strategies for implementing system tactics. We gave pros and cons for each and then chose one of the strategies for implementation.

Task 5 - UML Diagram:
Creating the UML diagram in Task 5 required teamwork. We started with Lucidchart but later switched to draw.io for better collaboration. This task highlighted the importance of effective teamwork and communication.

Task 6 - Logging and Auditing:
Task 6 focused on defining logging and auditing requirements for different stakeholders. We considered the needs of users, developers, and administrators, ensuring they have access to relevant logs. WE implemented ASR that has to do with presenting the logs in a correct and easy-to-understand way. We also learned about trace, debug, info, warn, error, and fatal logs and how they help manage system events.

Task 7 - Logging Strategy:
In Task 7, we created two logging and monitoring strategies tailored to different stakeholders' needs. We stressed the importance of granular logging levels and asynchronous logging. and the different ways of managing logs, both for visualising the logs and for how system can collaborate on logging or that each system manages logging for them self.

Linnéuniversitetet
Kalmar Växjö

Extra Assignment

Strategy
A strategy is a high level plan and set of choices designed for long term goals. It refers to a high-level plan or framework comprising various choices and approaches aligned with the system's long-term goals.
The choices can for instance be what patterns (that implement certain tactics) and mechanisms that shall be used in the system for ensuring accomplishment of its long term goals.

An example of a strategy may be a scheduling strategy that implements some performance tactic and that tactic may include mechanisms like round robin.

Mechanism
Whereas an architectural pattern provides a high level blueprint for solving a specific recurring design problem, mechanisms are actual methods used like protocols, algorithms or components like a security or verification mechanism. Examples of mechanisms are algorithms that perform tasks like encryption, authentication and intrusion detection.

Pattern
A pattern is a reusable and valid solution for a common and specific problem, for example the Model-View-Controller is a design pattern for separating logic, interface and data. A tactic can be seen as a collection of many tactics that together solve the problem. Because of that a pattern satisfies not only one but often various amount quality attribute scenarios. The solution handles relationships between elements, their responsibilities, their roles and how they work together. Because a pattern affects quality attribute requirements it will be a tradeoff between the different QA requirements and then it's important to analyze what QA requirements are ASR:s and give them prioritization.

Tactics
A tactic is a design decision that affects how a quality attribute is done. It can change how a system responds to a QAS stimulus. A tactic can also affect the architectural design in multiple ways.
For example if a tactic makes the availability of a system better then, the system may have worse testability, or better testability. So there can always be a tradeoff when implementing certain tactics.

Change Log

- Task 1: Changed from a list of components to an actual diagram. This new diagram has text explaining what each component provides to another. Also added text explaining the whole diagram.

- Task 2: Modified QAS 2 to relate more to information and resources.

- Task 3: Completely changed what we had. To have two strategies that solve Task 1 and 2, with different approaches.

- Task 4: Hade to change the components based on the new strategies in Task 3, we also redid the pros and cons and what scenarios they are responsible for.

- Task 5: Added new diagram based on diagram from task 1 with the changes from our new preferred strategy in assignment 4, based on our new implementations for tactics in assignment 3.

- Task 6: rewrote the ASR with extra explanations on what exactly the ARS include.

- Task 7: Rewrote both strategies from the new ASR. remade the graph, and added new pros and cons.