# PROJECT TITLE:
# ILIR SUBASHI ECONOMIST MANAGEMENT SYSTEM

## PHASE I: GROUP DIVISION

### 1.1 Team Members and Responsibilities

1. **KLEDIA BOKA** [LEADER] - GitHub Username [kboka]
2. **SORINA HASTOCI** [MEMBER] - GitHub Username [SorinaHastoci]
3. **SONIA SOTIRI** [MEMBER] - GitHub Username [soniasotiri]
4. **JADA MECE** [MEMBER] - GitHub Username [jadaaaaaaaaaaa]
5. **FRIONA POCARI** [MEMBER] - GitHub Username [FrionaPo]

### 1.2 Problem Statement

Our team met with the management team of Ilir Subashi Metal Supplier small company in Vore. The thorough need analysis which was performed by our team revealed that the company continuously faced difficulties managing its inventory and financial transactions by manual record keeping. As the company has grown in the number of transactions and inventory, the issues with manual record keeping have continued increasing. Thus, the need for a centralized inventory economist management system has become crucial in order for the company to minimize these inefficiencies in their inventory management of the product list. Thus, this centralized inventory economist management system will be able to manage the administrative tasks, inventory tasks, and economist tasks effectively and efficiently.

### 1.3 Solution Proposal

The Ilir Subashi Management System will offer the solution of a user-friendly designed interface through which economists, administrators, and workers can successfully achieve an efficient performance for each of their respective, required tasks within the company, no longer through manual record keeping, and as a result continuously updated information simultaneously so there are no inefficiencies and a real-time update of product inventory.

### 1.4 Features and Technologies

**Features**:

*Interface That's Easy to Use:*

-User-friendly design that makes it simple to navigate and operate, guaranteeing rapid user adoption.

*Centralized dashboard*

-A unified dashboard with rapid access to pertinent capabilities and individualized overviews for workers, economists, and administrators.

*Access Control Based on Roles (RBAC):*

-Distinct user roles for workers, economists, and administrators to guarantee safe access and features tailored to their respective tasks.

*Inventory Management Done Automatically:*

-Reduced human error and offered a real-time updating of the product inventory through streamlined procedures for adding, removing, and searching goods.

*Automating Financial Transactions:*

-Automated financial transaction recording that guarantees accuracy and makes it easier to observe financial activity in real time.

*Generation of Receipts:*

-Productive generation of receipts, accurately recording key information including the product ID, name, quantity, cost, total, and balance. The technology makes processing payments or printing simple.

*Analytics & Reporting:*

-Reporting capabilities that enable data-driven decision-making by allowing economists to examine financial performance and inventory patterns.

*Architecture that is Scalable:*

-A system that can be expanded to accommodate the company's future needs in terms of features, users, and goods.

**Technologies**:

*Backend and Frontend:*

We plan to use the NetBeans program with Java Programming Language. On the backend side, we create files, the connection to the database and develop the features.

Whilst on frontend, NetBeans offers a pallet of elements from textboxes, buttons, labels, dates etc, that are to be used with formatting the register files, login, dashboard and adding color to them.

*Database:*

For the database of the project we use MySql from Xampp. The tables of admin, employee, product etc, are connected to the forms from the application.

## 1.5 Project Scope

The Ilir Subashi Management system is a software solution for the challenges that the business faces in manual inventory management and offers a more efficient and centralized operation. We aim to create a user friendly interface for administrators, economists and workers, and subsequently enhance their performance and efficiency.

*Interface That's Easy to Use:*

-User-friendly design that makes it simple to navigate and operate, guaranteeing rapid user adoption.

*Centralized dashboard*

-A unified dashboard with rapid access to pertinent capabilities and individualized overviews for workers, economists, and administrators.

*Access Control Based on Roles (RBAC):*

-Distinct user roles for workers, economists, and administrators to guarantee safe access and features tailored to their respective tasks.

*Inventory Management Done Automatically:*

-Reduced human error and offered a real-time updating of the product inventory through streamlined procedures for adding, removing, and searching goods.

*Automating Financial Transactions:*

-Automated financial transaction recording that guarantees accuracy and makes it easier to observe financial activity in real time.

*Generation of Receipts:*

-Productive generation of receipts, accurately recording key information including the product ID, name, quantity, cost, total, and balance. The technology makes processing payments or printing simple.

*Analytics & Reporting:*

-Reporting capabilities that enable data-driven decision-making by allowing economists to examine financial performance and inventory patterns.

*Architecture that is Scalable:*

-A system that can be expanded to accommodate the company's future needs in terms of features, users, and goods.

## 1.6 Limitations and Boundaries

**Limitations**

*1. Dependency on Performance:*

The performance of the system is dependent upon the capabilities of the current hardware infrastructure. Hardware constraints or bottlenecks may affect the Ilir Subashi Management System's responsiveness and effectiveness.

*b. Maintenance and Assistance:*

After implementation, there will be a certain amount of ongoing maintenance and support offered. After this time, nevertheless, system upkeep and updates become the client's duty. Problems that result from improper maintenance or system updates are not the project team's responsibility.

*c. System Integration:*

The Ilir Subashi Management System is developed as a standalone solution. Integrations with third-party systems or additional features beyond the proposed scope may incur additional costs and require a separate agreement.

**Boundaries**:

*a. Infrastructure in Hardware:*

Only the software implementation falls within the project's purview. The hardware infrastructure that is already in place cannot be improved or modified. It is the client's obligation to make any necessary hardware upgrades or modifications for the best system performance.

*b. Customization*

The characteristics and functions listed below will form the basis for the development of the Ilir Subashi Management System. The project scope might not involve customization that goes beyond these predetermined boundaries. Specific discussions and agreements may apply to requests for major changes or extra features.

*c. User Expertise:*

The project's boundaries are predicated on the idea that users would actively engage in the offered training sessions in order to get the requisite competence with the Ilir Subashi Management System. Users' mistakes or inefficiencies as a result of non-participation in training are not the project team's fault.

*d. Compliance to regulations:*

Although every attempt will be made to match the system with industry best practices, it is the client's obligation to comply with any industry legislation or standards that are outside the purview of the project. Separate discussions and agreements must be held on any further requirements pertaining to regulatory compliance.

## 1.7 Aims and Objectives

**Overall Aim:** The overall aim of the project is to develop a comprehensive management system for Ilir Subashi Metal Company, which will be focused towards enhancing and increasing efficiency of the inventory management and record keeping for economists, administrators, and workers of the company.

**Main Objectives:**

a. Develop a user-friendly interface for administrators, economists, and workers.

b. Centralize inventory and financial management tasks to eliminate manual record-keeping.

c. Provide real-time updates on product inventory and financial transactions.

d. Implement role-based access control to ensure secure data access and task-specific functionalities.

e. Enable automated inventory management, financial transaction recording, and receipt generation.

f. Incorporate reporting and analytics tools for economists to make data-driven decisions.

g. Design a scalable architecture to accommodate future growth and additional functionalities.

## 1.8 Application Description

The Ilir Subashi Economist Management system will be a comprehensive software solution which will be specifically designed from our team in order to streamline the key processes of the company's economists, workers, and administrators, for all their distinct functionalities tailored, while all the information of inventory is updated simultaneously and in coherence for all users of different roles.

To summarize all the descriptions of the roles of the administrators, economists, and workers:

- The admin will be responsible for the overall management of the system, inventory control, and user registrations

-The economist will focus on transaction management, accessing inventory, and the generation of receipts

-The workers will be involved within the everyday operations, which mainly consist of searching what products are in stock

**ADMIN:**

-The administrators will securely login within the system through their username and password

-Option of administrators changing their password for extra layer of security added

-New users able to be registered as administrators with information of id, name, last name, phone number, email, phone number, safety question (for extra security if forgot password)

-Admins will consist of a Main Page which will include a dashboard with the key functionalities of an administrator within the system: Inventory Management and Receipt Management

-Inventory Management: able to add new products within the inventory, registering a product by: ID, name, product type, delete products from the inventory, and search any product by its ID or name

-Receipts Management: able to create receipts with details such as receipt number, product ID, name, amount, price, total, balance, and options to print or pay

**ECONOMIST:**

-Economists can log in the system using their credentials of username and password

-New economists can be registered through the details of id, name, last name, phone number, email, password, safety question (for extra security if forgot password)

-Similar to the Admins, the economist can generate receipts including details of purchased items

-Economist also has the functionality/ability to search the inventory for a specific product by id or name

**WORKER:**

-Workers can login in the system using their credentials of username and password

-New workers will be able to register through the details of id, name, last name, phone number, email, password, safety question (for extra security if forgot password)

-The ability to search for products in the inventory stock by id or name

**ROLES AND TASKS DISTRIBUTION:**

1. **KLEDIA BOKA** [LEADER] - Responsible for the Economist Interface
2. **SORINA HASTOCI** [MEMBER] - Responsible for the Register and Login Interface
3. **SONIA SOTIRI** [MEMBER] - Responsible for the Admin Interface
4. **JADA MECE** [MEMBER] - Responsible for the Workers Interface and Database

5. **FRIONA POCARI** [MEMBER] - Responsible for Inventory management (part of Admin)

## PHASE II: USER REQUIREMENTS & APPLICATION SPECIFICATIONS

### 2.1 CHOSEN DEVELOPMENT MODEL

*State the chosen development model, such as **Agile, Waterfall, or Scrum**.*
*Justification: Briefly explain the rationale behind selecting this model.*

-Our chosen development model for this project is the **Waterfall** model. The Waterfall development model has been selected for the development of the Ilir Subashi Economist Management System.
-First of all, this model provides a **sequential approach** to software development of the project, ideal as each phase of the project will be completed before going towards the next. The nature of our project aligns perfectly with this methodology, providing us with systematic progression from the gathering of requirements from user to the final implementation of the project.
-Waterfall offers predictability and **stability** in project planning and execution. By defining requirements upfront and following a linear progression through development phases, stakeholders can have a clear understanding of project timelines, costs, and deliverables.
-The Waterfall method divides the project into **well-defined, distinct phases**, including requirements analysis, design, implementation, testing, and maintenance. This clarity within all of the project phases will ensure the clear definition of each stage, the specific required objectives and deliverables, leading towards better control and management of our project.
-Waterfall also **minimizes changes** to requirements once the project has started, as each phase is completed sequentially. Since in this project we expect the requirements to remain stable throughout the development lifecycle, this model is advantageous and suitable. In our case, the requirements are unlikely to change significantly and are well-understood. In the case of our project Ilir Subashi Economist Management System, we have thoroughly analyzed the scope and requirements during Phase 1, thus Waterfall provides a suitable framework for development.

## 2.2 USER REQUIREMENTS

End-users, including administrators, economists, and workers, seek a user-friendly system that streamlines their tasks. Clients aim to improve operational efficiency and decision-making capabilities, while developers focus on technical implementation aspects. Regulatory authorities and external partners indirectly influence project requirements and functionalities.

### a. STAKEHOLDERS:

*Identify key stakeholders, including end-users, **clients, developers, and any other relevant parties.** Provide a brief discussion on their roles and interests in the project.*

#### 1. End-users

**Roles:** End-users consist of administrators, economists, and workers within the Ilir Subashi Metal Company.

**Interests:** End-users are interested in a user-friendly interface which will simplify their tasks, such as inventory management, record-keeping, and financial transactions. They are seeking to find features such as quick access to relevant functionalities of the system, easy navigation, and intuitive workflows for enhancing efficiency and productivity in work.

#### 2. Clients

**Roles:** The management team of the Ilir Subashi Metal Company serves as the client, initiating the project and benefiting from its successful implementation.

**Interests:** The clients aim to improve the accuracy and the efficiency of the inventory management and financial transactions within Ilir Subashi company. They are interested in having access to a centralized system which streamlines administrative tasks and ultimately contributes towards the profitability and continuous growth of the company.

#### 3. Developers

**Roles:** Developers include software engineers, programmers, and technical personnel which will be responsible for designing, implementing, and maintaining the Ilir Subashi Economist Management System.

**Interests:** Developers focus on understanding the requirements and expectations of end-users and clients to design and implement a system that meets their needs appropriately. They prioritize technical aspects such as system architecture, database design, user interface design, and ensuring reliability, security, and scalability of the software. This is what we will also achieve in this project.

### 4. Regulatory Authorities

**Roles:** Regulatory authorities oversee how much the management system is adhering to regulations and industry standards regarding the data privacy of workers and their financial transactions.
**Interests:** The main interest of regulatory authorities includes ensuring that the Ilir Subashi Economist Management System is in compliance with industry and legal standards. This includes requirements related to secure handling of financial data, protecting personal information, and adhering to the regulatory frameworks of financial transactions and business operations. This regulatory compliance is key in order to mitigate any legal risks and also building and maintaining stakeholders' trust.

### 5. Suppliers

**Roles:** Suppliers of the Ilir Subashi Metal Company play a fundamental role in the company's supply chain operations and its business ecosystem.

**Interests:** A well-functioning inventory management system can enhance supply chain visibility of our company, optimize the inventory levels, and also improve the collaboration between the company and its suppliers. Aligning the project's requirements and functionalities with the needs of suppliers can enhance collaboration and contribute to overall supply chain performance and competitiveness.

## <u>b. USER STORIES:</u>

*Present a few detailed user stories. Include the user type (e.g.,* ***administrator, end-user****).*

*Clearly outline the corresponding requirement and the benefit it brings to the user or project.*

### 1. User Story - ADMINISTRATOR

**Requirement:** As an administrator, I want to be able to register new users in the system, including administrators, economists, and workers, by providing their relevant details such as ID, name, last name, phone number, email, and a safety question for password retrieval.

**Benefit:** This feature allows administrators to manage user access to the system efficiently, ensuring that all authorized personnel have the necessary credentials to perform their roles. It streamlines the onboarding process for new employees and enhances overall system security by implementing password retrieval mechanisms.

2. **User Story - ECONOMIST**

**Requirement:** As an economist, I need to generate receipts for transactions, including details such as receipt number, product ID, name, amount, price, total, and balance.

**Benefit:** This functionality enables economists to accurately record and document financial transactions within the system. It facilitates transparent and auditable financial reporting, enhances accountability in transaction management, and ensures accurate documentation for regulatory compliance purposes.

3. **User Story - WORKER**

**Requirement:** As a worker, I want to be able to search for products in the inventory by their ID or name, so I can quickly locate and retrieve the necessary items for my tasks.

**Benefit:** This feature improves efficiency in inventory management for workers, allowing them to easily locate and access products required for their daily operations. It reduces time spent on manual inventory searches, minimizes errors in product retrieval, and enhances overall productivity in warehouse or operational environments.

## 2.3 FUNCTIONAL REQUIREMENTS

### a. BRIEF DESCRIPTION

*Write a short and clear sentence for each thing your system should do. Pretend you're explaining it to a friend who doesn't know much about computers.*

**1. User Authentication:** Users should be able to log in securely using their username and password.
**2. User Registration:** New users should be able to register with the system by providing necessary information.
**3. Inventory Management:** Users should be able to add, delete, and search for products within the inventory.
**4. Receipt Generation:** Users should be able to generate receipts for transactions, including essential details.

**5. Role-Based Access Control:** Different user roles should have access to specific functionalities based on their roles.
**6. Dashboard:** Users should have access to a centralized dashboard providing an overview of relevant functionalities and data.

## b. ACCEPTANCE CRITERIA

*Make a simple checklist for each thing. What specific things must happen so you can say, "Yep, this part is finished!"? Imagine you're making a to-do list for each feature.*

**1. User Authentication:**
- Username and password fields are provided on the login page.
- Successful login redirects the user to the system's dashboard.
- Invalid login attempts result in appropriate error messages.

**2. User Registration:**
- Registration form includes fields for necessary information such as name, email, and password.
- Upon submission, new user accounts are created and stored in the system database.
- Validation checks ensure that no duplicate usernames or email addresses are registered.

**3. Inventory Management:**
- Users can add new products by entering details such as product name and quantity.
- Delete functionality removes selected products from the inventory.
- Search functionality allows users to find products by name or ID.

**4. Receipt Generation:**
- Users can generate receipts for transactions by selecting purchased items and entering transaction details.
- Receipts include essential information such as receipt number, product details, total cost, and payment information.
- Generated receipts are saved and accessible for future reference.

**5. Role-Based Access Control:**
- Different user roles (administrators, economists, workers) have access to specific features based on their roles.
- Access control is enforced throughout the system, ensuring that users cannot perform unauthorized actions.

**6. Dashboard:**
- The dashboard provides quick access to key functionalities such as inventory management, receipt generation, and reporting.
- Relevant data and statistics are displayed in a user-friendly format for easy understanding.

## 2.4 NON-FUNCTIONAL REQUIREMENTS

### a. BRIEF DESCRIPTION

*Describe how well your whole system should work. Is it about being fast, easy to use, or something else? Use simple words to explain each idea.*

1. **Performance:** The system should respond quickly to user actions and maintain responsiveness even during peak usage periods.
2. **Usability:** The system should be intuitive and easy to use, requiring minimal training for end-users to perform tasks efficiently.
3. **Reliability:** The system should operate consistently without frequent failures or downtime, ensuring uninterrupted access to essential functionalities.
4. **Security:** The system should employ robust security measures to protect sensitive data and prevent unauthorized access or data breaches.
5. **Scalability:** The system should be able to handle increased workload and user traffic without significant degradation in performance or functionality.
6. **Compatibility:** The system should be compatible with various devices and web browsers to ensure accessibility for all users.
7. **Maintainability:** The system should be easy to maintain and update, with clear documentation and modular architecture facilitating future enhancements and bug fixes.

### b. ACCEPTANCE CRITERIA

*For each idea, set a goal. If it's about speed, say exactly how fast it should be. If it's about being easy to use, describe what makes it easy. These are like rules to make sure your whole system is great.*

**1. Performance:**
- The system should load pages within 2 seconds on average.
- Response time for user interactions (e.g., clicking buttons, entering data) should be less than 0.5 seconds.

**2. Usability:**
- Navigation within the system should be intuitive, with clearly labeled menus and buttons.

- Common tasks should be accessible within 2-3 clicks from the dashboard.

**3. Reliability:**
- The system should have an uptime of at least 99.9%.
- Failures or errors should occur less than once per week, with prompt resolution by the technical team.

**4. Security:**
- User authentication should use strong encryption and secure protocols (e.g., HTTPS).
- Access controls should be enforced to ensure that users can only access data and functionalities relevant to their roles.

**5. Scalability:**
- The system should be able to handle a concurrent user load of at least 100 users without degradation in performance.
- Database performance should scale linearly with an increase in the volume of data.

**6. Compatibility:**
- The system should be compatible with the latest versions of major web browsers (e.g., Chrome, Firefox, Safari).
- Responsive design principles should be implemented to ensure optimal user experience across devices of different screen sizes.

**7. Maintainability:**
- Code should be well-documented, following established coding standards and best practices.
- Updates and enhancements should be deployed with minimal disruption to the system's operation, utilizing version control and automated testing to ensure reliability.

## 2.4 APPLICATION SPECIFICATIONS

### a. ARCHITECTURE

The system's architecture defines its key elements, their connections, and interactions. In software design, considerations like business strategy, individual roles, interactions, design, and the IT environment are crucial. Software architecture provides the structure for coordinating interactions and managing complexity. It ensures that technical and operational needs are addressed, optimizing quality features such as performance and security. Decisions about software development organization significantly impact the final product's effectiveness, accessibility, efficiency, and quality.

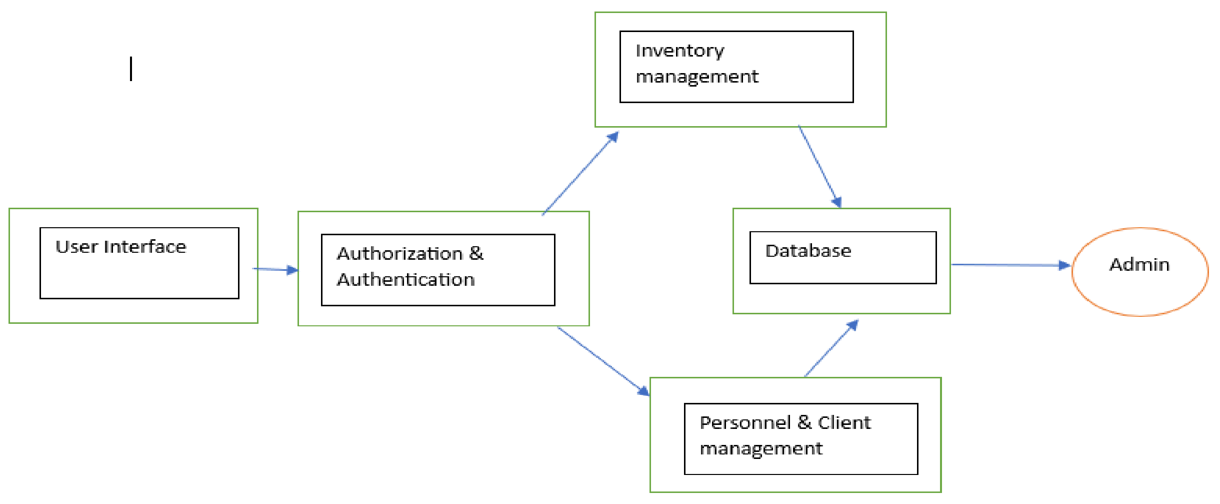
1. **Model**: Extend the model to encompass entities for inventory, clients, personnel, invoices, and bills, each containing relevant data and logic.

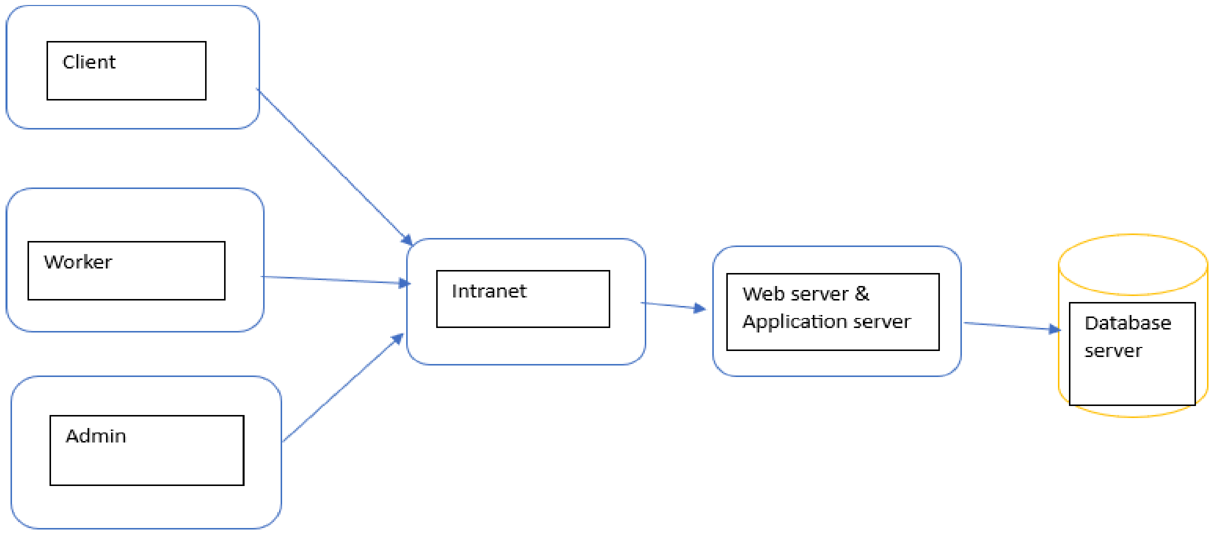
2. **View**: Update the presentation layer to include interfaces for inventory, client, and personnel management, as well as for handling invoices and bills, with role-specific views for admins, economists, and workers.

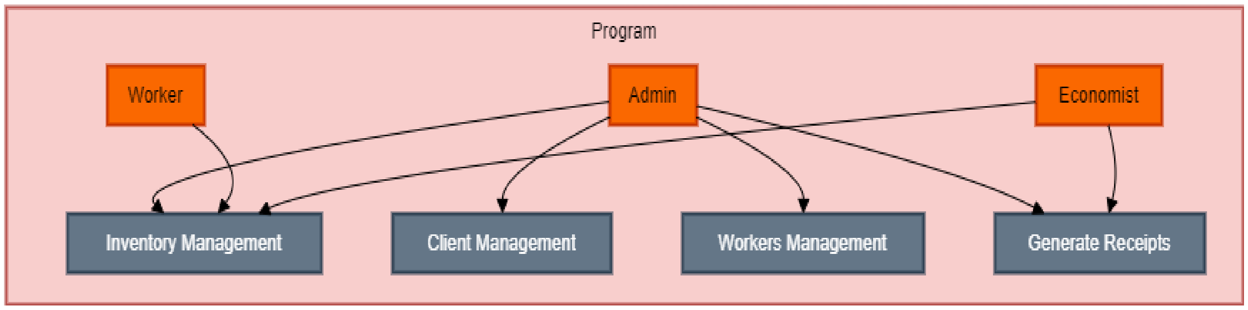
3. **Controller**: Adjust controllers to manage requests based on user roles, such as admins handling inventory, client, and personnel management, economists dealing with invoices and bills, and workers accessing inventory data.

4. **Interactions**: Define user-system interactions according to roles, allowing admins to perform various management tasks, economists to handle financial processes, and workers to view inventory data only.

***Include high-level diagrams** or descriptions of system components and their interactions.*

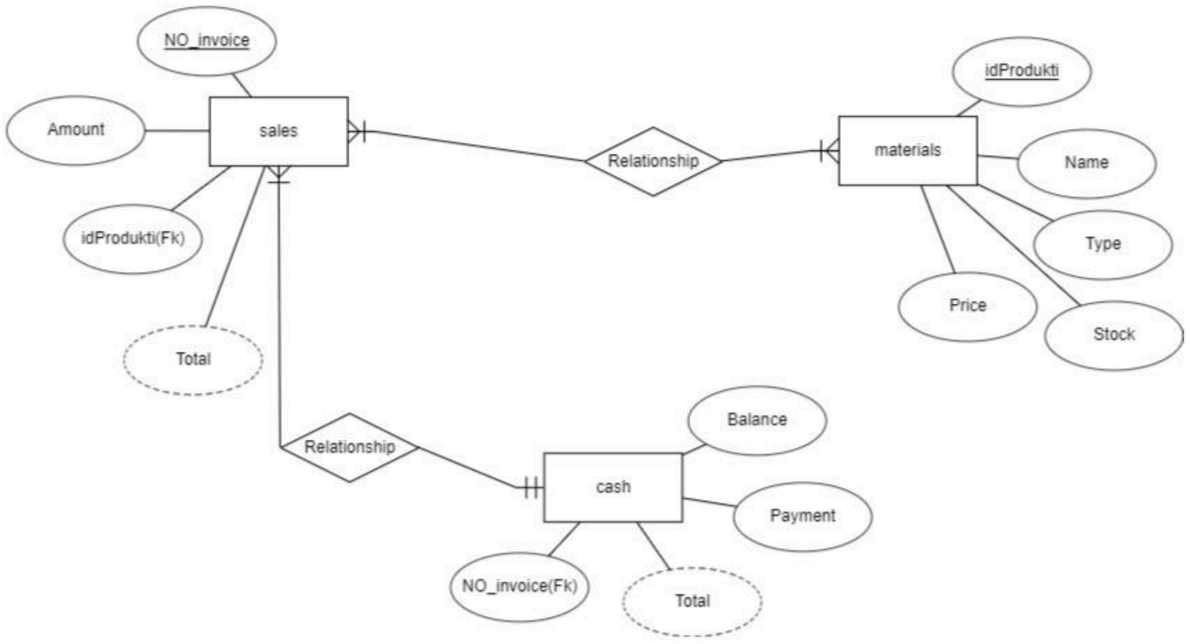


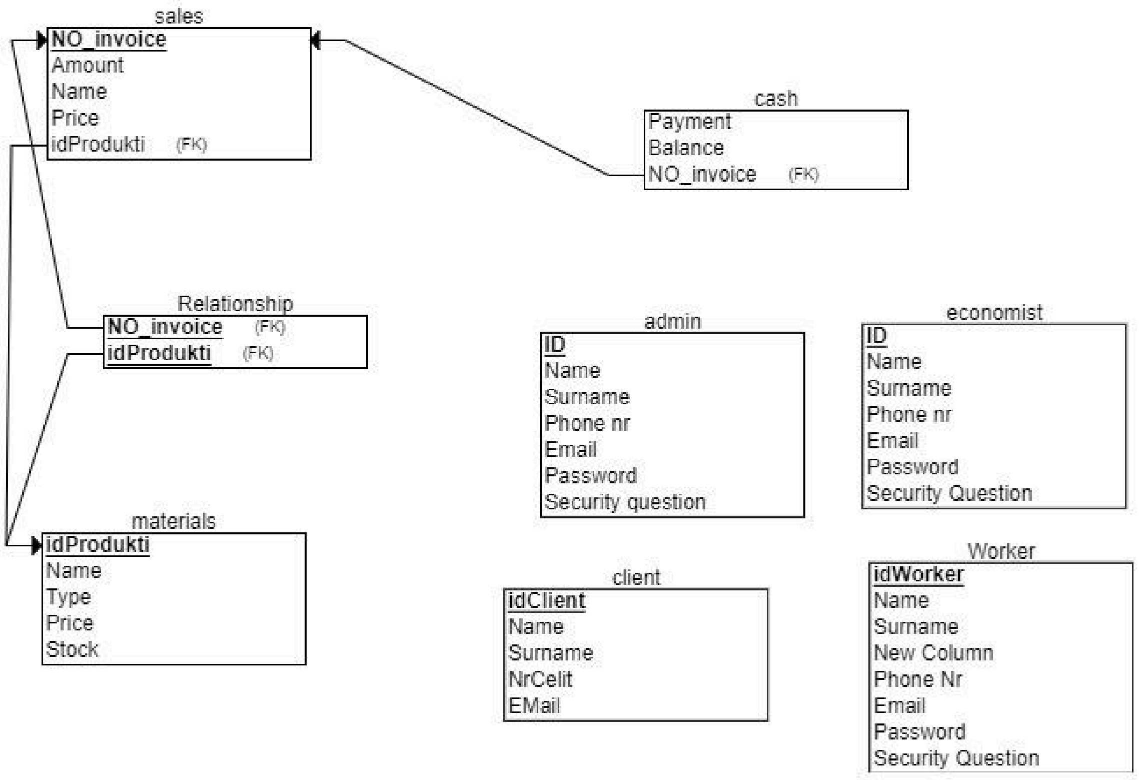




## b. DATABASE MODEL

*Detail the database model. Include information on tables, relationships, constraints, and any other relevant details.*


NO_invoice
Amount
sales
idProdukti(Fk)
Total
Relationship
idProdukti
materials
Name
Type
Price
Stock
Balance
Relationship
cash
Payment
NO_invoice(Fk)
Total


sales
NO_invoice
Amount
Name
Price
idProdukti (FK)
cash
Payment
Balance
NO_invoice (FK)
Relationship
NO_invoice (FK)
idProdukti (FK)
admin
ID
Name
Surname
Phone nr
Email
Password
Security question
economist
ID
Name
Surname
Phone nr
Email
Password
Security Question
materials
idProdukti
Name
Type
Price
Stock
client
idClient
Name
Surname
NrCelit
EMail
Worker
idWorker
Name
Surname
New Column
Phone Nr
Email
Password
Security Question

1. Cash Table:

- Primary Key: NO_invoice

- Attributes: Balance (type: numeric), payment (type: numeric), total (type: numeric, derived attribute)

- Relationship: One-to-One with Sales table on the sales side.

2. Sales Table

- Primary Key: NO_invoice

- Attributes: NO_invoice (type: integer, primary key), amount (type: numeric), id_product (type: integer, foreign key), total (type: numeric, derived attribute)

- Relationship: One-to-Many with Materials table (one Sales can have multiple Materials), One-to-One with Cash table (each Sales has a corresponding Cash entry).

3. Materials Table:

- Primary Key: idProdukti

- Attributes: idProdukti (type: integer, primary key), Name (type: string), Type (type: string), Stock (type: integer), Price (type: integer)

- Relationship: One-to-Many with Sales table (many Materials can be associated with one Sales).

Tables Included in the Relational schema but not part of the entity relationship diagram :

4.Admin Table:

Primary Key: ID

Attributes: ID (type: integer, primary key), Name (type: string), Surname (type: string), Phone_nr (type: string), Email (type: string), Password (type: string), Security_question (type: string)

5.Economist Table:

Primary Key: ID

Attributes: ID  (type: integer, primary key), Name (type: string), Surname (type: string), Phone_nr (type: string), Email (type: string), Password (type: string), Security_question (type: string)

6. Client Table:

Primary Key: idClient

Attributes: idClient (type: integer, primary key),Name (type: string), Surname (type: string), Phone_nr (type: string), Email (type: string)

7.Worker Table

Primary Key: idWorker

Attributes: idWorker (type: integer, primary key), Name (type: string), Surname (type: string), Phone_nr (type: string), Email (type: string), Password (type: string), Security_question (type: string)

## <u>c. TECHNOLOGIES USED</u>

*List and briefly explain the technologies, frameworks, and languages chosen for development: Include any reasoning behind the choices, such as scalability or compatibility.*

**1. Java Programming Language**

**-**Java is chosen for its versatility, robustness, and platform independence. It allows for the development of scalable and reliable enterprise applications. Java's extensive libraries and frameworks support various functionalities required for the Ilir Subashi Economist Management System, including server-side programming, database connectivity, and web application development.

**2. NetBeans IDE**

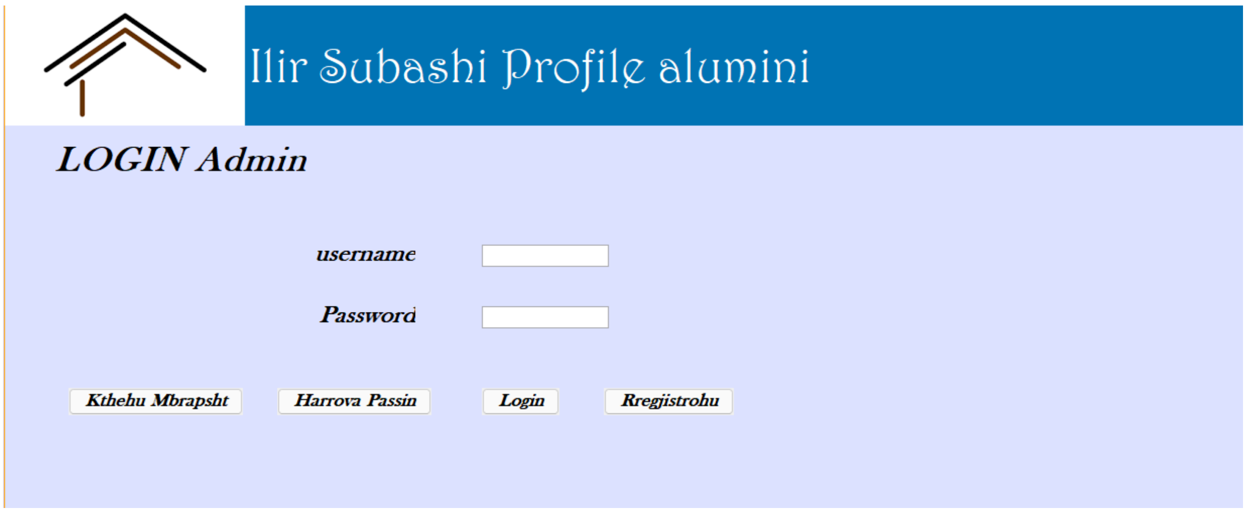
-NetBeans is selected as the integrated development environment (IDE) for Java development. It provides a user-friendly interface and a comprehensive set of tools for Java development, including code editing, debugging, and project management. NetBeans simplifies the development process and enhances productivity, contributing to faster development cycles.

## d. USER INTERFACE DESIGN

*Showcase wireframes, mockups, or describe the user interface. Provide a visual representation of how users will interact with the system.*

### 1. Login Page

- The login page will feature input fields for username and password.
- Below the login fields, there might be an option for users to register if they are new to the system.
- Upon successful login, users will be redirected to the dashboard.

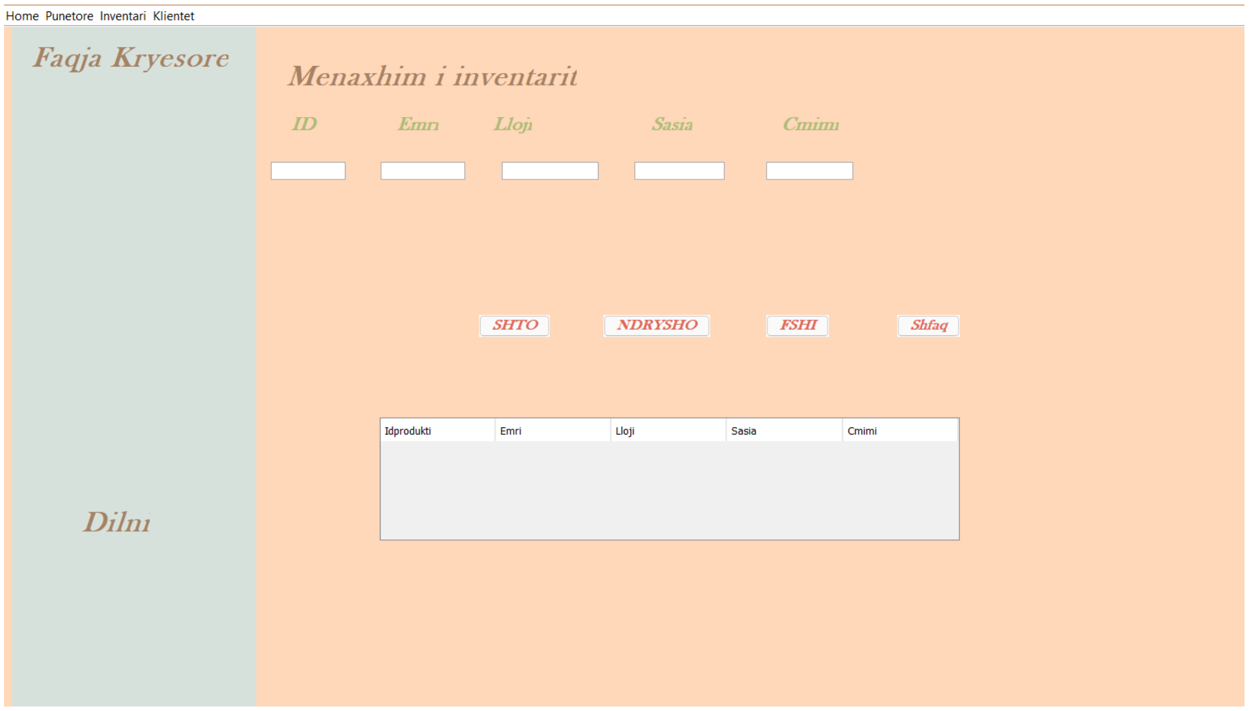


### 2. Dashboard

- The dashboard will serve as the central hub for users, providing quick access to essential functionalities based on their roles (e.g., administrators, economists, workers).
- It may include widgets or cards displaying summary information such as inventory status, recent transactions, and important notifications.
- Navigation menus or buttons will allow users to access different modules of the system, such as inventory management, receipt generation, reporting, and settings.

### 3. Inventory Management

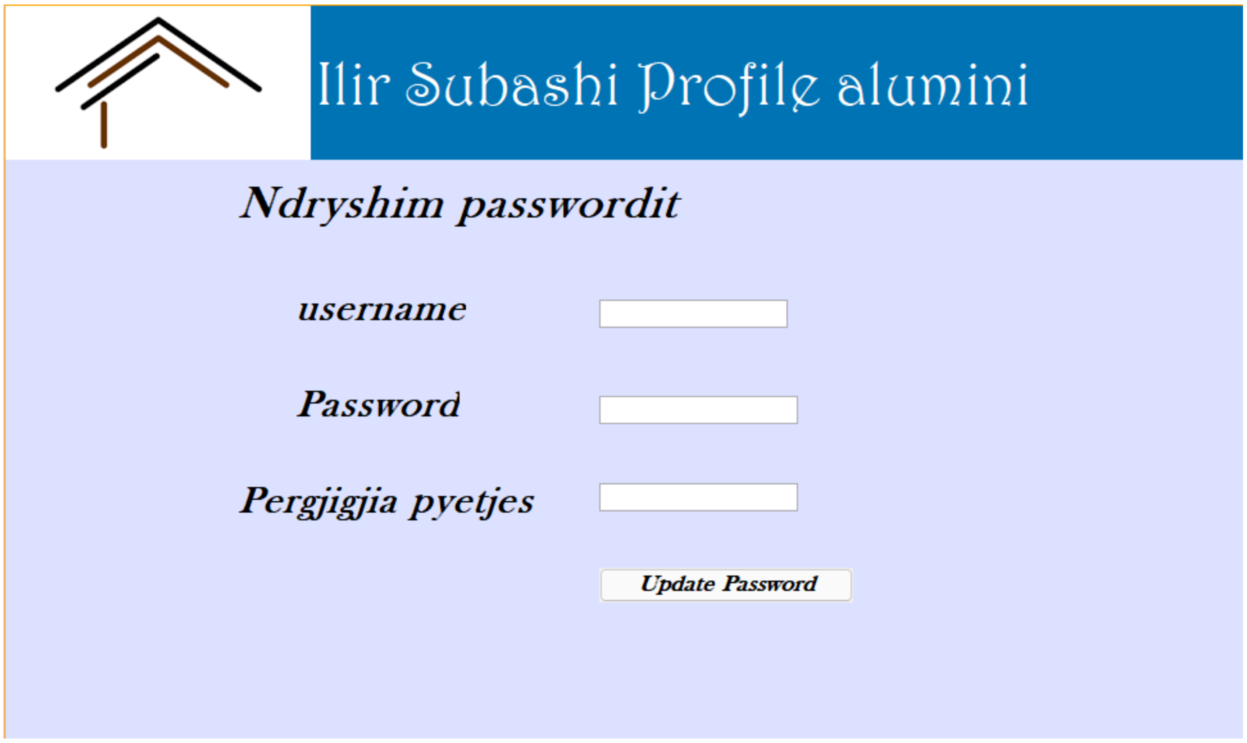
- The inventory management interface will feature forms or tables for adding, deleting, and searching for products.
- Users may be able to input details such as product name, quantity, and price, with options to submit changes or cancel.
- Search functionality might include filters or search bars to quickly locate specific products within the inventory.



| Idprodukti | Emri | Lloji | Sasia | Cmimi |
|---|---|---|---|---|

**4. Receipt Generation**
- The receipt generation interface will allow users to select products from the inventory and enter transaction details such as quantity, price, and payment method.
- Generated receipts will display essential information including receipt number, product details, total cost, and payment information.
- Users may have options to print or save receipts for future reference.

**5. Settings**
- The settings interface may provide options for users to customize their preferences, such as changing passwords or updating personal information.
- Administrators may have additional settings for managing user accounts, permissions, and system configurations.



**e. SECURITY MEASURES**

*Briefly discuss the security measures and protocols implemented. Touch upon encryption, authentication, and any other security features.*

**1. User Authentication**

- Users are required to authenticate themselves using a username and password before accessing the system.
- Strong password policies are enforced, requiring users to create passwords with a combination of uppercase and lowercase letters, numbers, and special characters.
- Passwords are securely stored using cryptographic hashing algorithms to prevent unauthorized access in case of a data breach.

**2. Role-Based Access Control (RBAC):**

- Access control mechanisms are implemented based on user roles (e.g., administrators, economists, workers) to restrict access to specific functionalities and data.
- Each user is assigned a role with predefined permissions, ensuring that they can only perform actions relevant to their job responsibilities.

**3. Data Encryption:**

- Data transmitted between the client and server is encrypted using secure communication

protocols such as HTTPS (HTTP over SSL/TLS).
- Encryption algorithms are used to encrypt sensitive data stored in the database, ensuring that it remains protected even if unauthorized access occurs.

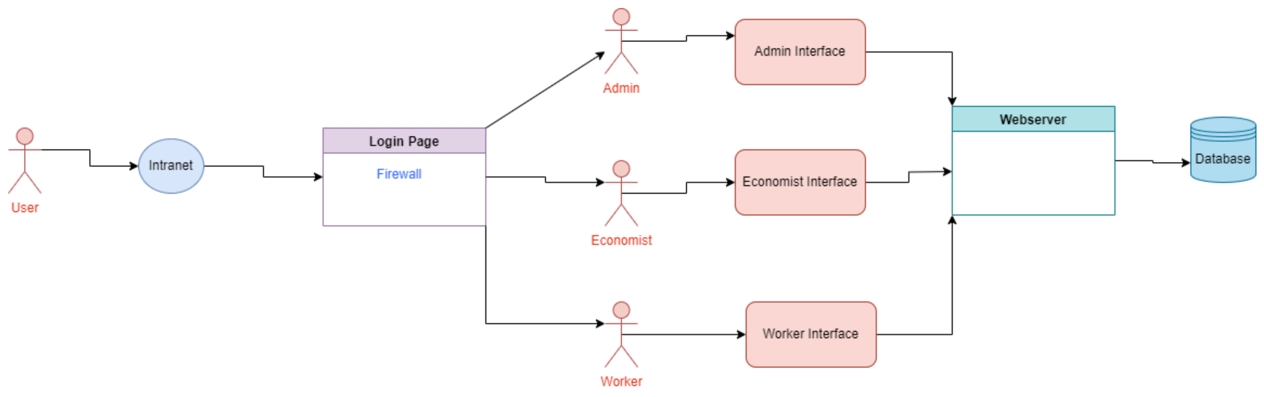
**4. Session Management:**
- Sessions are managed securely to prevent session hijacking or fixation attacks.
- Session tokens are generated using strong random number generators and are invalidated after a predefined period of inactivity or upon logout.

## PHASE III: SOFTWARE DESIGN & MODELING

### Software Design and Modeling

**System Architecture:**

*Explain how different parts of the system work together. Think of it as describing the big picture of your application - what it does and how it does it.*



1. Model Layer: This layer encompasses the entities and their relevant data and logic. Each entity, such as inventory, clients, personnel, invoices, and bills, would have its own set of attributes and methods to interact with and manipulate the data. For example:

- Inventory: Contains information about products, stock levels.

- Clients: Stores details about the customers of the business.

- Personnel: Holds data related to employees, including their roles, contact information, etc.
- Invoices and Bills: Records financial transactions, payments.

2. View Layer: This layer is responsible for the presentation of data and user interfaces. It includes interfaces for managing inventory, clients, personnel, invoices, and bills. Additionally, it provides role-specific views for different types of users:

   - Admin: Have access to all management interfaces and functionalities.
   - Economists: Focus on financial processes such as invoicing and billing.
   - Workers: Typically have limited access, mainly for viewing inventory data.

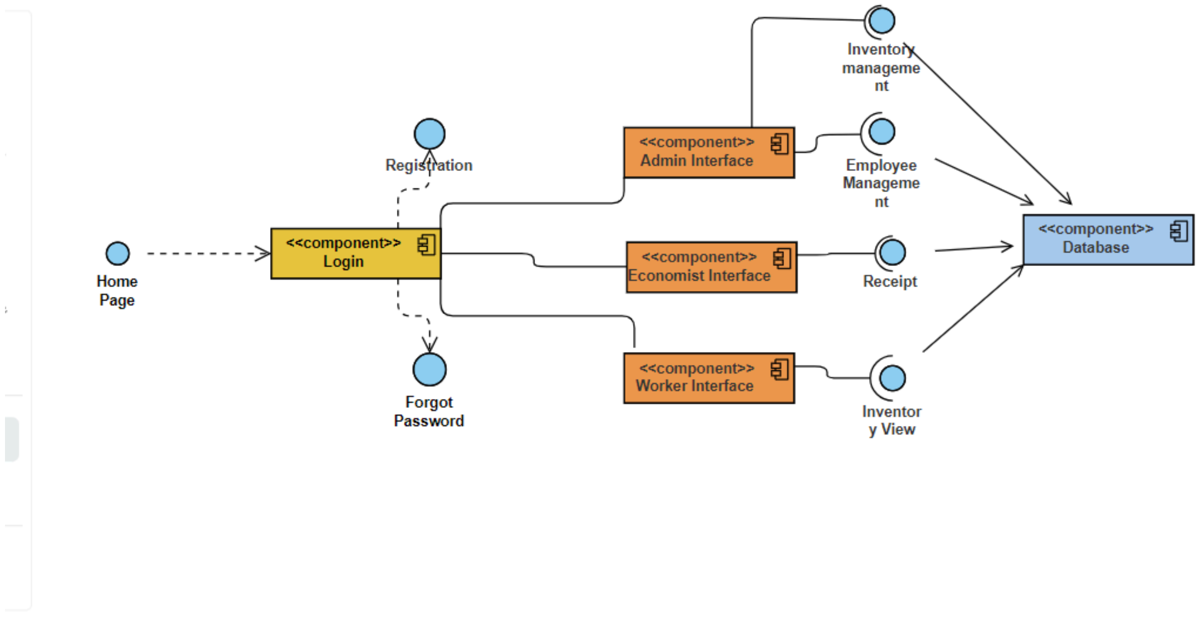
3. Controller Layer: The controllers manage the flow of data between the model and view layers based on user requests. They handle user authentication, authorization, and route requests to the appropriate parts of the system. Controllers are adjusted to enforce role-based access control, ensuring that users can only perform actions permitted by their roles:

   - Admin Controllers: Handle requests related to inventory, client, and personnel management.
   - Economist Controllers: Manage invoicing and billing processes.
   - Worker Controllers: Provide access to inventory data for viewing purposes.

4. Interactions: User interactions with the system are defined according to their roles. Admins can perform various management tasks such as adding new inventory items, managing client accounts, and updating employee information. Economists can generate invoices, track payments, and manage billing processes. Workers, on the other hand, have restricted access and can only view inventory data to carry out their tasks efficiently.

**Component Diagram:**

*Draw a picture showing the different parts (components) of your application and how they interact with each other. For example, if your application has a login feature, a component diagram would show how the login component talks to other parts of the system.*

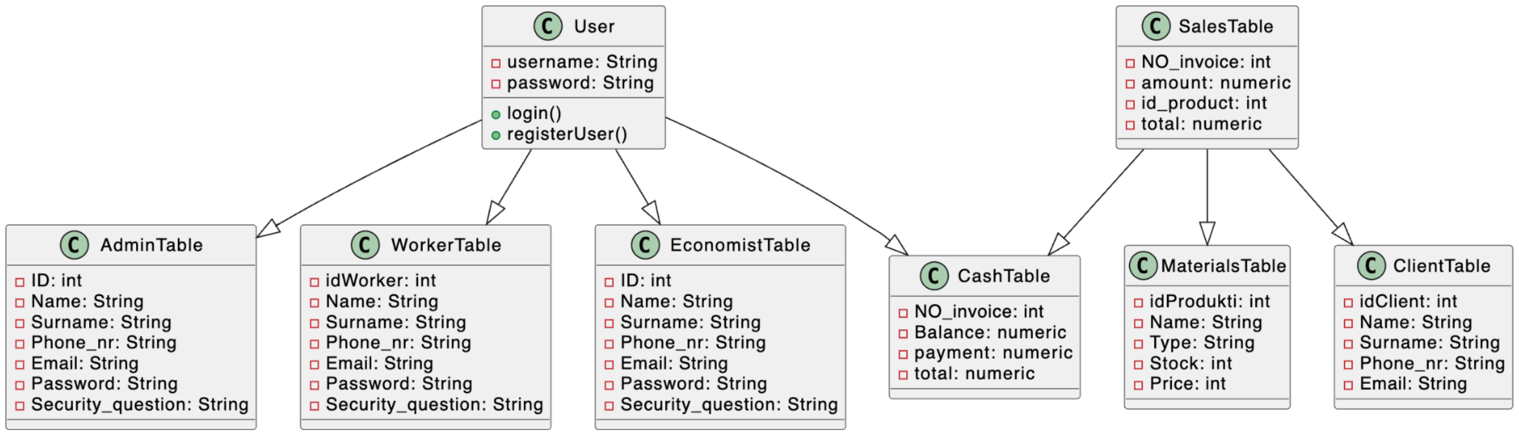


The component is the login page besides the login interface it also has the option to change into registration and forgot password interface.After login a different interface will appear based on your credentials.Meaning if you are an admin you will deal with the admin interface and so on.For admin there are different interfaces.We have inventory management,employment and client management they also have access to receipts. After finishing the changes are then saved in a database.

**Detailed Design**

**Class Diagram:**

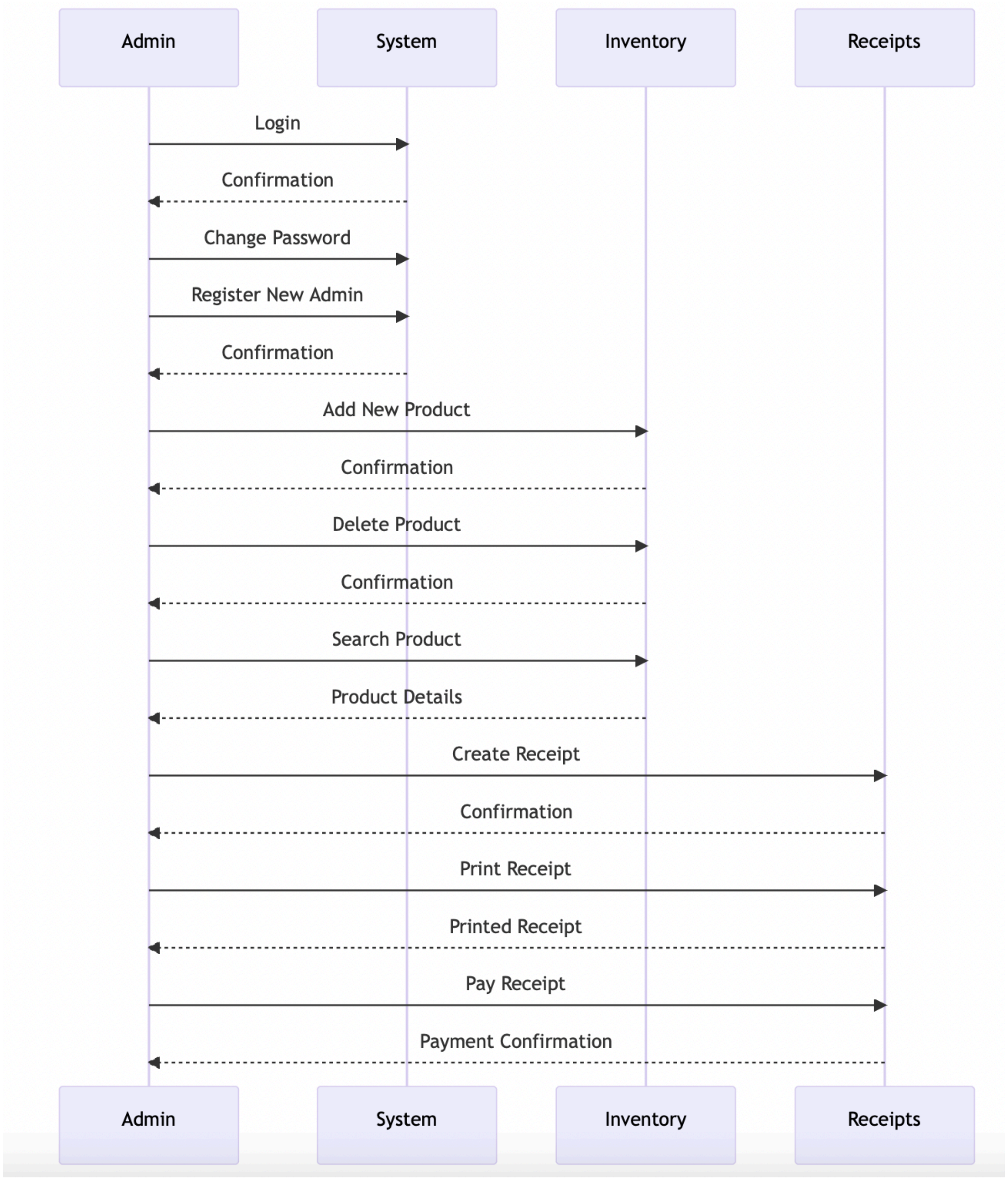
*Think of a class diagram as a family tree for your application. It shows the different types of "things" in your application (called classes) and how they relate to each other. For example, if your application deals with cars, a class diagram would show that a Car class might have attributes like color and model, and methods like drive() and park().*



**Sequence Diagrams:**

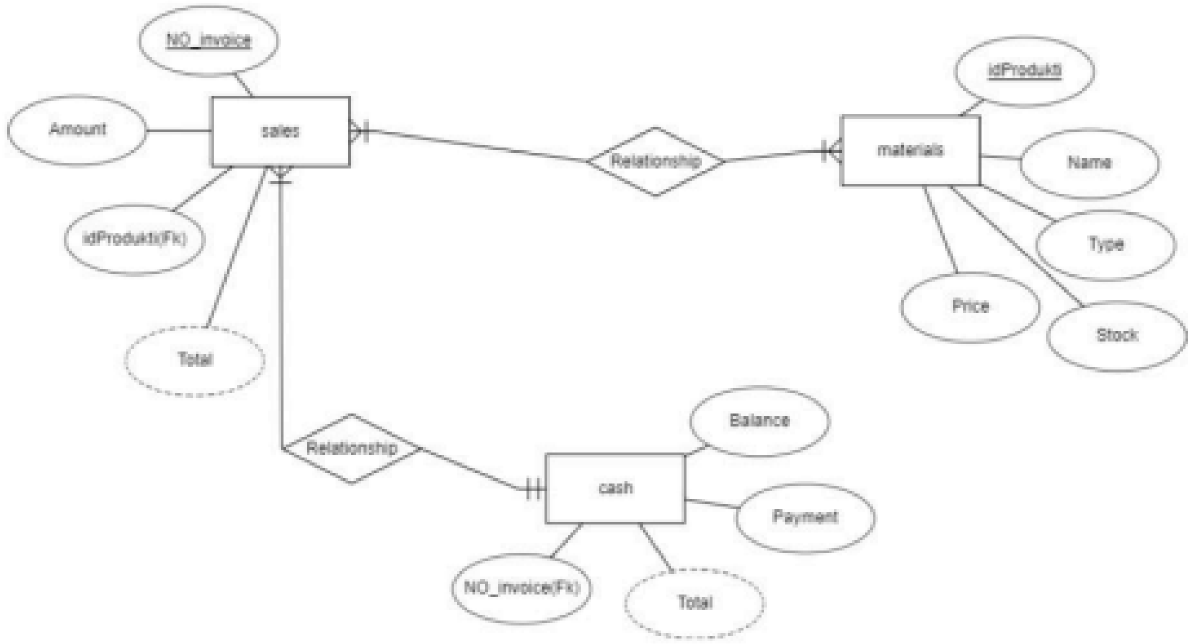
Sequence diagrams show the order in which things happen in your application. They're like step-by-step instructions for how different parts of your application interact with each other to accomplish a task. For example, a sequence diagram for ordering food online would show the steps involved, like selecting items, adding them to the cart, and checking out.

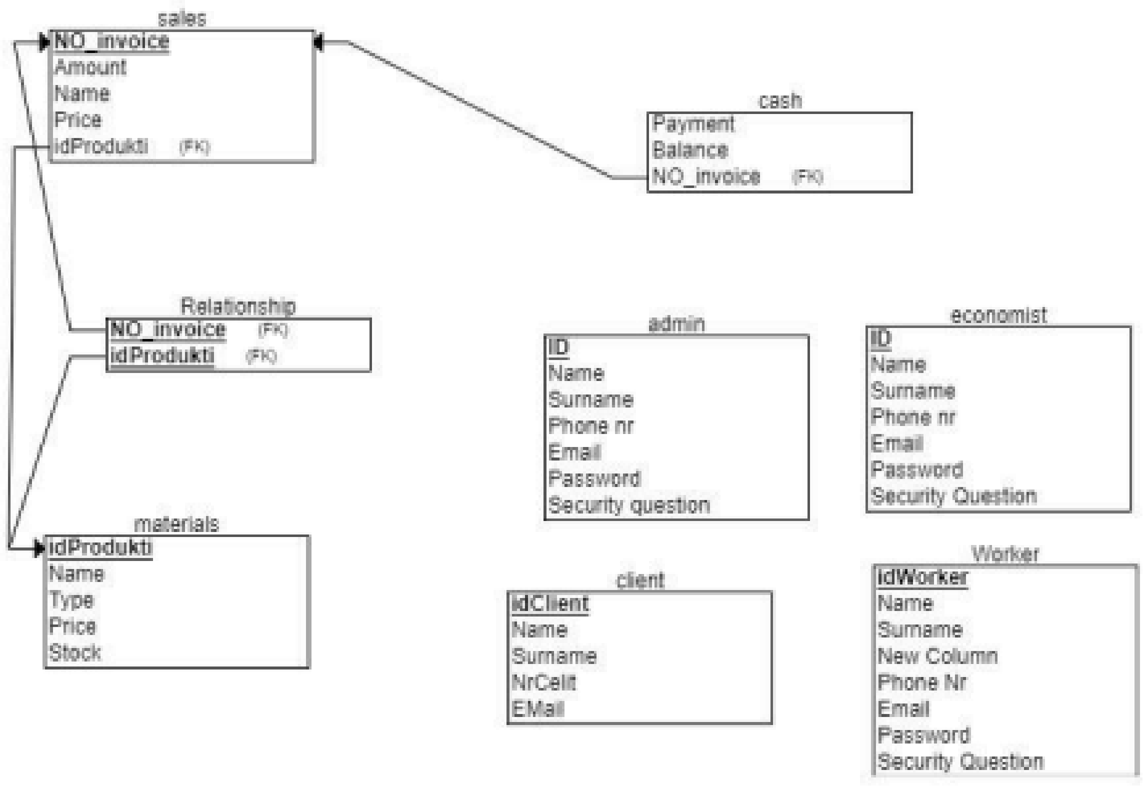
**Database Design:**



**Database Design:**

*Explain how you've organized the data in your application. This includes things like what tables you have in your database, how they're related to each other, and how you've made sure your data is organized efficiently.*





**1. Cash Table:**

- Primary Key: NO_invoice

- Attributes: Balance (type: numeric), payment (type: numeric), total (type: numeric, derived attribute)

- Relationship: One-to-One with Sales table on the sales side.

**2. Sales Table**

- Primary Key: NO_invoice

- Attributes: NO_invoice (type: integer, primary key), amount (type: numeric), id_product (type: integer, foreign key), total (type: numeric, derived attribute)

- Relationship: One-to-Many with Materials table (one Sales can have multiple Materials), One-to-One with Cash table (each Sales has a corresponding Cash entry).

**3. Materials Table:**

- Primary Key: idProdukti

- Attributes: idProdukti (type: integer, primary key), Name (type: string), Type (type: string), Stock (type: integer), Price (type: integer)

- Relationship: One-to-Many with Sales table (many Materials can be associated with one Sales).

Tables Included in the Relational schema but not part of the entity relationship diagram :

**4.Admin Table:**

Primary Key: ID

Attributes: ID (type: integer, primary key), Name (type: string), Surname (type: string), Phone_nr (type: string), Email (type: string), Password (type: string), Security_question (type: string)

5.Economist Table:

Primary Key: ID

Attributes: ID (type: integer, primary key), Name (type: string), Surname (type: string), Phone_nr (type: string), Email (type: string), Password (type: string), Security_question
(type: string)

**6. Client Table:**

Primary Key: idClient

Attributes: idClient (type: integer, primary key),Name (type: string), Surname (type: string), Phone_nr (type: string), Email (type: string)

**7.Worker Table**

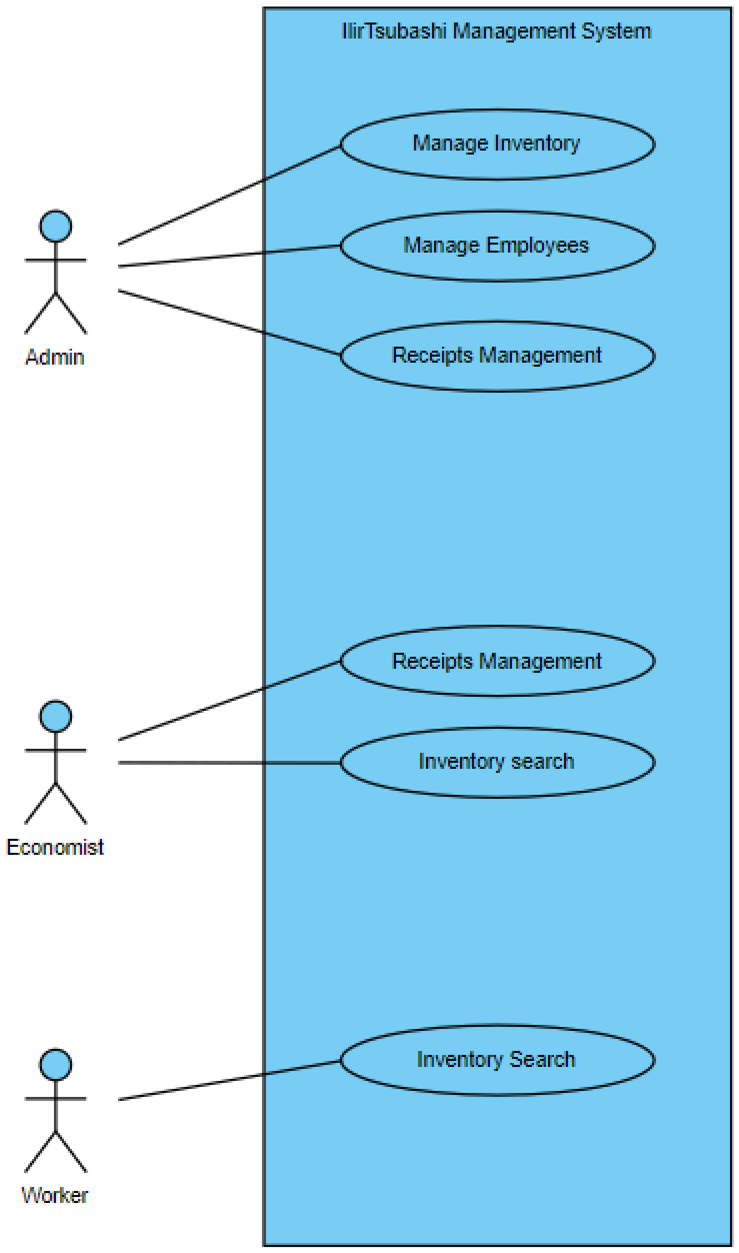
Primary Key: idWorker

Attributes: idWorker (type: integer, primary key), Name (type: string), Surname (type: string), Phone_nr (type: string), Email (type: string), Password (type: string), Security_question (type: string)

**Modeling**

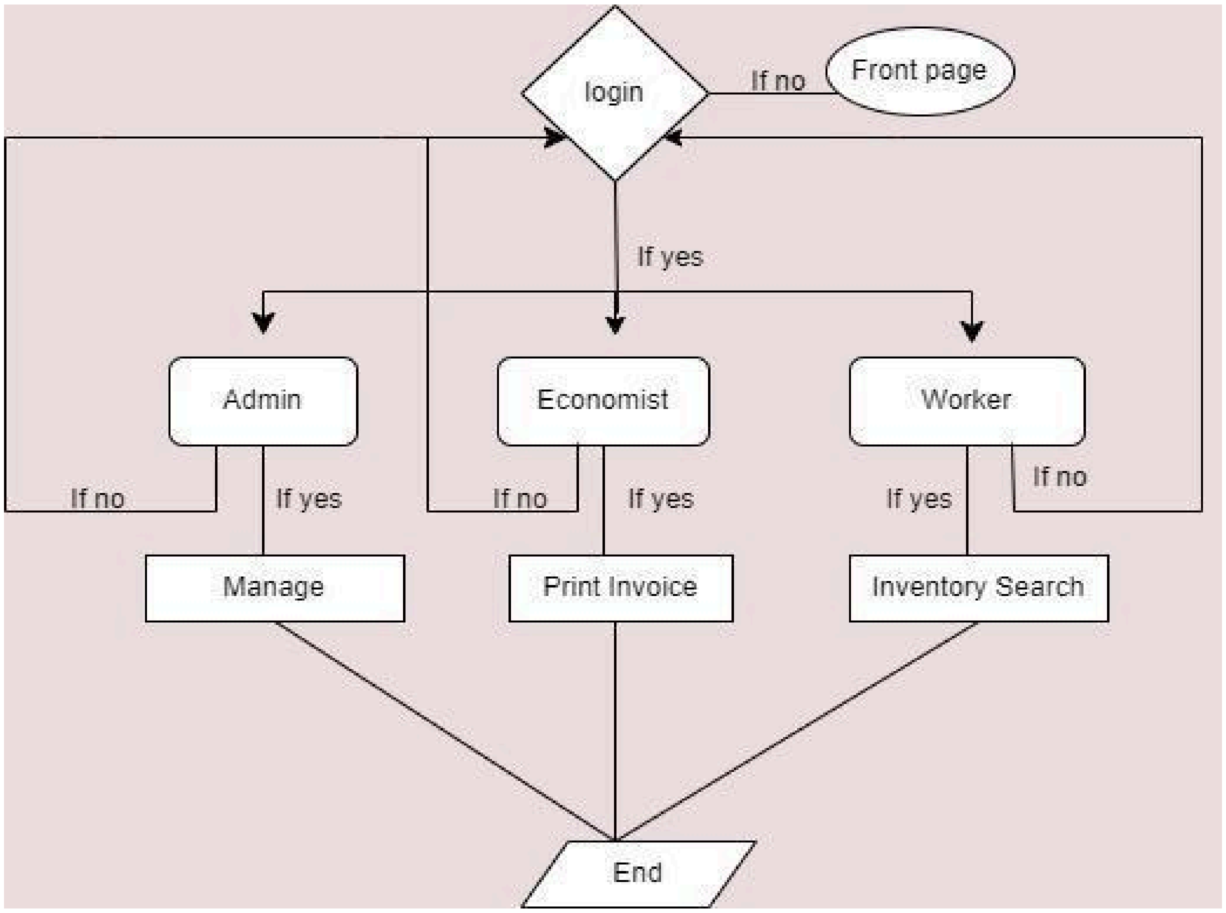
Use Case Diagram:

A use case diagram shows the different ways people (or other systems) can use your application. It's like a map of all the different things your application can do. For example, a use case diagram for a music streaming app might show that users can search for songs, create playlists, and listen to music.



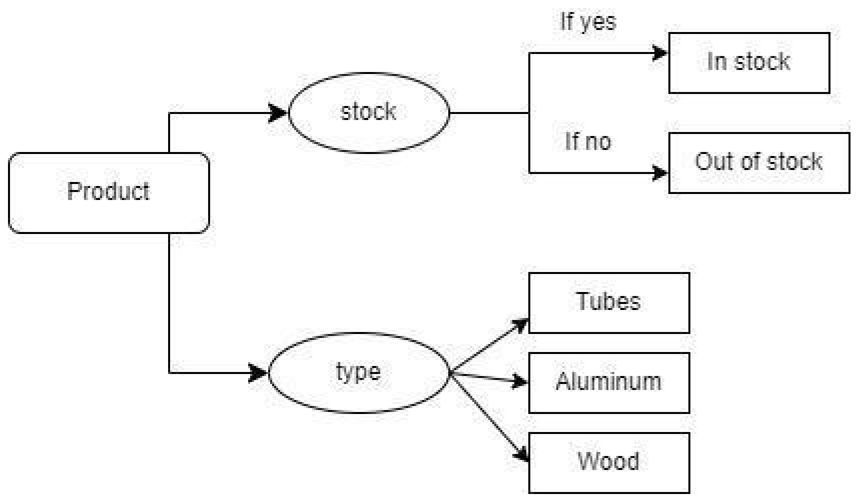
**Activity Diagrams:**

*Activity diagrams show the flow of activities in your application. They're like flowcharts that show the steps involved in completing a task. For example, an activity diagram for booking a* *flight might show the steps involved, like searching for flights, selecting one, and entering passenger information.*



**State Diagrams:**

*State diagrams show the different states that an object in your application can be in, and how it transitions between those states. They're like maps of all the possible "statuses" your application can be in. For example, a state diagram for a light switch might show that the switch can be in the "on" or "off" state, and how it transitions between them.*



## PHASE IV: SOFTWARE DEVELOPMENT

### 1. INTRODUCTION TO TESTING

Software testing is a critical phase in the software development lifecycle aimed at uncovering defects or bugs that could potentially impact the performance, reliability, and usability of an application. It involves a systematic and methodical examination of the software to ensure that it behaves as expected and meets the requirements outlined during the development process. Testing is not just about finding bugs; it's also about validating that the software functions correctly under various scenarios and conditions, including normal usage, edge cases, and unexpected inputs.

There are various types of software testing, ranging from unit testing, where individual components or modules are tested in isolation, to system testing, where the entire application is evaluated as a whole. Testing methodologies can also vary, including manual testing, where testers manually execute test cases, and automated testing, which involves using tools and scripts to automate the testing process. Each type of testing serves a specific purpose and contributes to ensuring the overall quality of the software.

Effective software testing requires a combination of technical expertise, domain knowledge, and thorough understanding of the project requirements. It is an iterative process that starts early in the development cycle and continues throughout the software's lifecycle. By identifying and fixing defects early on, testing helps reduce the cost and time involved in software development, enhances user satisfaction, and ultimately contributes to the success of the software product in the market.

## 2. PURPOSE OF TESTING

Testing the code for the Ilir Subashi Management System is crucial for guaranteeing the software's quality, dependability, and functioning. Here are the main reasons why testing is necessary:

**Early Detection of Faults:** Developers can identify faults and problems in code through testing. By performing tests on multiple portions of the system in a systematic manner, errors may be found and resolved quickly, minimizing the risk of more serious problems arising later in the development lifecycle or after deployment.

**Verification of Software Components:** Testing ensures that functions, modules, and features meet requirements and standards. This procedure guarantees that the developed code matches the intended behavior described in the project's design and functional requirements.
Ensuring functional correctness:

During testing, engineers ensure that the product works properly under all situations and user interactions. This comprises checking for boundary conditions, error handling, input validations, and intended outputs. It verifies that the software fits the functional criteria specified for inventory management, financial transactions, and administrative operations.
Testing improves software quality by identifying and addressing flaws early on. Developers may enhance and optimize software by testing it on a continual basis, resulting in a more robust and dependable solution.

Thorough testing ensures the Ilir Subashi Management System satisfies user expectations and delivers a smooth experience. By discovering and resolving issues before deployment, the system is less likely to have interruptions or usability difficulties that might affect user satisfaction.

Testing reduces hazards connected with software deployment and usage. Identifying and fixing problems during the development phase reduces the chance of significant failures or security vulnerabilities in the production environment, increasing the system's overall dependability.

Testing promotes continual improvement through its iterative nature. Testing feedback allows developers to improve their code, follow best practices, and optimize performance, resulting in a more efficient and maintainable software solution over time.

## 3. FOCUS ON TESTING A SINGLE COMPONENT

**1. DATABASE CONNECTION**

- **Component:** The database connection class/module/function is crucial as it handles the interaction between the software application and the underlying database. It's responsible for establishing, maintaining, and closing connections to the database, as well as executing queries and handling results.
- **Importance of Testing:** Testing this component is vital because any issues with the database connection can lead to data loss, corruption, or security vulnerabilities. Moreover, errors in database interaction can cause the entire application to malfunction, affecting its usability and reliability.
- **Complexity:** The complexity of this component lies in handling various database management systems (e.g., MySQL, PostgreSQL, MongoDB), connection pooling, transaction management, and error handling.
- **Impact on the System:** A faulty database connection can result in data inconsistency, application crashes, or even security breaches. Therefore, thoroughly testing this component ensures the stability, performance, and security of the entire system.

**2. LOGIN WINDOW**

- **Component:** The login window class/module/function is responsible for authenticating users and granting access to the system. It validates user credentials, handles session management, and controls user permissions.
- **Importance of Testing:** Testing this component is essential as it forms the first line of defense against unauthorized access to the system. Any vulnerabilities or flaws in the login process can compromise the system's security and integrity.
- **Complexity:** The complexity of this component lies in handling various authentication methods (e.g., username/password, OAuth), implementing security features (e.g., CAPTCHA, two-factor authentication), and managing user sessions securely.
- **Impact on the System:** A weak or flawed login mechanism can lead to unauthorized access, data breaches, and loss of sensitive information. Thorough testing ensures that the login process is robust, reliable, and resistant to security threats.

**3. ECONOMIST FRONT PAGE EXIT LABEL**

- **Component:** The economist front page exit label represents a specific feature or functionality on the front page of an application, possibly related to navigation or user interaction.
- **Importance of Testing:** Testing this component ensures that users can navigate or interact with the front page of the application seamlessly. Any issues with this component could result in a poor user experience or hinder users from accessing important features or content.

**- Complexity:** The complexity of this component depends on its functionality, such as handling user inputs, triggering actions, or navigating to different sections of the application. It may also involve integration with other parts of the application.

**- Impact on the System:** A malfunctioning or poorly implemented exit label can frustrate users and lead to dissatisfaction with the application. Thorough testing helps identify and address any usability issues or bugs, ensuring a smooth user experience.

**4. INSERTING NEW ADMIN**

**- Component:** The functionality responsible for inserting a new admin user into the system's database.

**- Importance of Testing:** Testing this component is crucial as it involves handling sensitive user data and granting administrative privileges. Any vulnerabilities or flaws in this process could lead to unauthorized access or privilege escalation.

**- Complexity:** The complexity of this component lies in validating user inputs, enforcing security measures (e.g., password hashing, input sanitization), and ensuring proper authorization for administrative actions.

**- Impact on the System:** A security breach or unauthorized access resulting from flaws in the new admin insertion process can have severe consequences, including data breaches, system compromise, and legal liabilities. Thorough testing helps mitigate these risks by identifying and addressing security vulnerabilities.

**5. INSERTING NEW PRODUCT**

**- Component:** The functionality responsible for adding a new product to the system's inventory or database.

**- Importance of Testing:** Testing this component is important as it directly impacts the accuracy and reliability of the system's inventory management. Any errors or inconsistencies in adding new products could lead to inventory discrepancies or disruptions in supply chain operations.

**- Complexity:** The complexity of this component lies in validating product information, handling inventory updates, and ensuring data integrity across the system.

**- Impact on the System:** Inaccurate or incomplete product data resulting from flaws in the insertion process can lead to mismanagement of inventory, stockouts, or overstock situations. Thorough testing helps ensure that the system's inventory remains accurate and up-to-date, facilitating smooth business operations.

## 4. PREPARING TEST CASES

**1. DATABASE CONNECTION**

- **Normal Input:** Test connecting to the database using valid credentials and ensure that the connection is successful.
- **Edge Cases:** Test connecting to the database with maximum allowed connections and verify that it handles connection pooling efficiently. Test connecting to the database with incorrect but recoverable credentials and ensure appropriate error handling.
- **Invalid Input:** Test connecting to the database with incorrect credentials and ensure that it fails gracefully with an informative error message. Test connecting to a non-existent database and verify that it handles the error appropriately.

**2. LOGIN WINDOW**

- **Normal Input:** Test logging in with valid username and password and ensure that the user gains access to the system.
- **Edge Cases:** Test logging in with a username containing special characters and verify that it handles them correctly. Test logging in with a password exceeding the maximum length and ensure that it truncates or rejects the input appropriately.
- **Invalid Input:** Test logging in with an incorrect password and ensure that it fails with an appropriate error message. Test logging in with a non-existent username and verify that it handles the error gracefully.

**3. ECONOMIST FRONT PAGE EXIT LABEL**

- **Normal Input:** Test clicking on the exit label and ensure that it navigates the user to the intended page or performs the intended action.
- **Edge Cases:** Test clicking on the exit label rapidly multiple times and verify that it doesn't cause any unexpected behavior or errors. Test clicking on the exit label while other asynchronous processes are ongoing and ensure that it doesn't interfere with them.
- **Invalid Input:** Test clicking on the exit label when the application is in a state where navigation is not allowed and verify that it handles the action appropriately without crashing or freezing.

**4. INSERTING NEW ADMIN**

- **Normal Input:** Test inserting a new admin with valid username, password, and permissions and ensure that the new admin is added to the system successfully.
- **Edge Cases:** Test inserting a new admin with the maximum allowed username length and ensure that it handles it correctly. Test inserting a new admin with special characters in the password and verify that it handles them securely.
- **Invalid Input:** Test inserting a new admin with a username that already exists in the system and ensure that it fails with an appropriate error message. Test inserting a new admin with a blank password and verify that it rejects the input and prompts for a valid password.

**5. INSERTING NEW PRODUCT**

**- Normal Input:** Test inserting a new product with valid information (e.g., name, price, quantity) and ensure that it is added to the inventory correctly.

**- Edge Cases:** Test inserting a new product with the maximum allowed length for the product name and ensure that it handles it correctly. Test inserting a new product with a price of zero or a negative value and verify that it rejects the input.

**- Invalid Input:** Test inserting a new product with a blank name or missing required fields and ensure that it fails with an appropriate error message. Test inserting a new product with a quantity exceeding the maximum allowed value and verify that it handles the input appropriately.

## 5. CHOOSING TESTING FRAMEWORKS

For Java development in NetBeans, JUnit is a highly recommended testing framework for unit testing. It provides a simple and effective way to write and execute tests for your Java classes. Here's how you can set up your testing environment with JUnit in NetBeans:

**1. Install JUnit:**

- JUnit is typically bundled with most Java development environments, including NetBeans. However, if you need to install it manually, you can download the JUnit JAR file from the official website (https://junit.org/junit5/) and add it to your project's classpath.

**2. Create Test Classes:**

- In NetBeans, right-click on your project in the Project Explorer.
- Go to New > Other.
- Choose the "JUnit" category and select "JUnit Test".
- Click "Next" and follow the prompts to create a new test class.
- You can also manually create test classes by creating Java classes and annotating them with `@Test` annotations from the JUnit framework.

**3. Write Test Methods:**

- Inside your test class, write methods to test different aspects of your code.
- Use JUnit's assertion methods like `assertEquals()`, `assertTrue()`, `assertFalse()`, etc., to validate the expected behavior of your code.

**4. Run Tests:**

- Right-click on your test class or individual test methods.
- Choose "Test File" or "Test Method" from the context menu.
- Alternatively, you can run tests using the keyboard shortcut (usually Shift+F6).

**5. View Test Results:**
- The results of the test execution will be displayed in the NetBeans output console.
- You'll see a summary of the tests executed, along with any failures or errors encountered.

**6. Analyze Results:**
- NetBeans provides detailed information about test failures, including stack traces and assertion failures.
- Use this information to diagnose and fix issues in your code.

**7. Repeat and Refactor:**
- Continuously write new tests as you develop new features or refactor existing ones.
- Run tests frequently to ensure that your code remains functional and regression-free.

## 6. WRITING TEST CODE

For the `testInsertProduct` method in the `ProduktiTest` class, we're testing the functionality of inserting a product. Here's how we can structure the test code:

**1. Test Environment Setup:** Begin by importing necessary libraries, including JUnit and `java.lang.reflect.Method`.

**2. Instantiate the Class Under Test:** Create an instance of the `Produkti` class, which represents the GUI frame for managing products.

**3. Set Sample Data:** Set up sample data within the text fields of the `Produkti` frame. This data simulates user input for inserting a product.

**4. Invoke the Private Method:** Use reflection to access the private method `jButton1ActionPerformed`, which represents the action performed when the insert button is clicked. By setting its accessibility to true and invoking it, we simulate the button click event programmatically.

**5. Assertions:** Add assertions to verify the expected behavior. For example, after insertion, we can assert that the text fields are cleared. Additionally, we might want to check if the table displaying products is updated correctly.

**6. Exception Handling:** Wrap the test logic in a try-catch block to handle any exceptions that might occur during the test execution.

## 7. RUNNING TESTS

**1. Run Tests:**

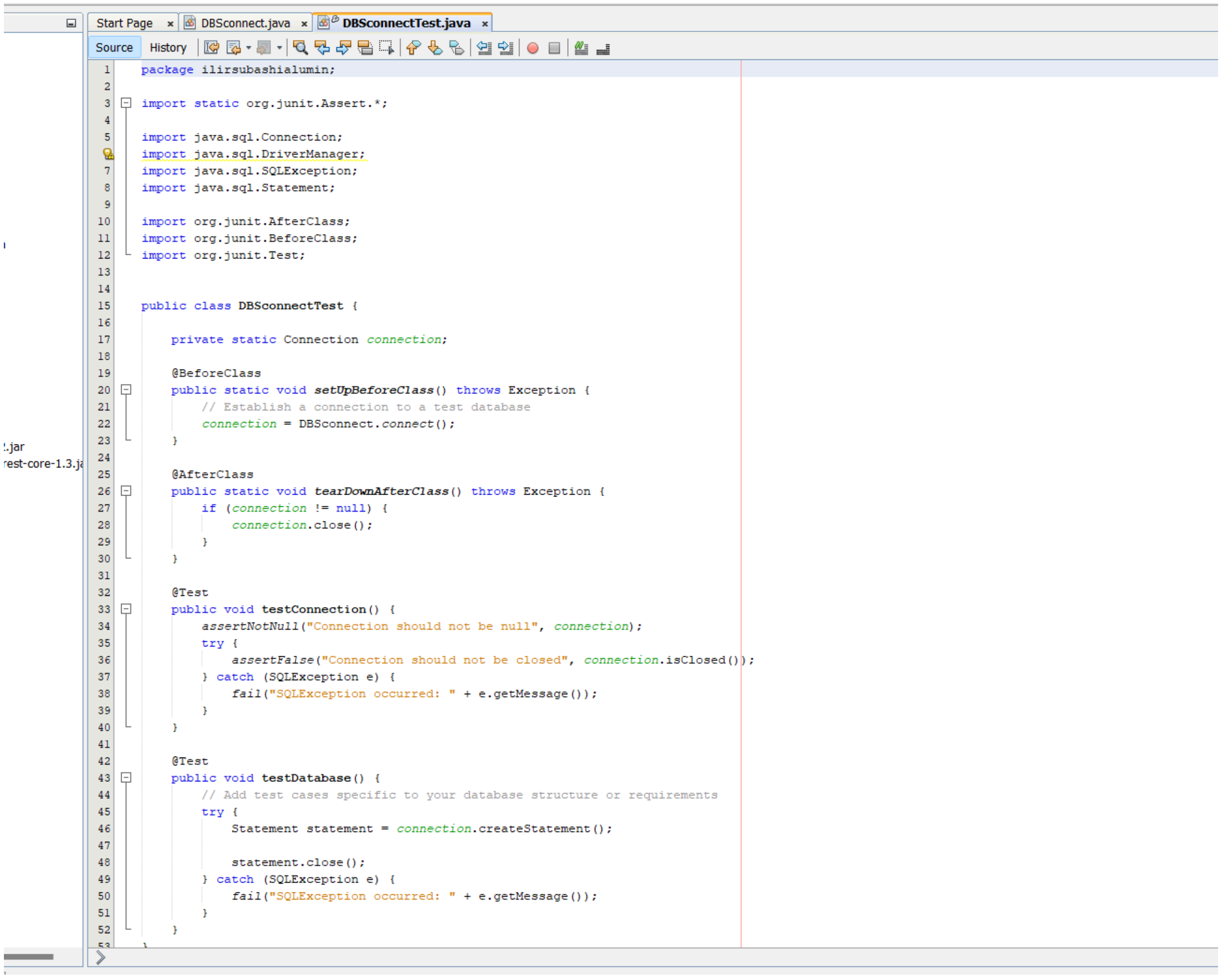
- Open your project in NetBeans.
- Navigate to the test class or test suite you want to execute.
- Right-click on the test class or suite file.
- Choose the "Run File" option from the context menu.
- NetBeans will execute the selected tests and display the results in the Output window.

**2. Interpret Results:**

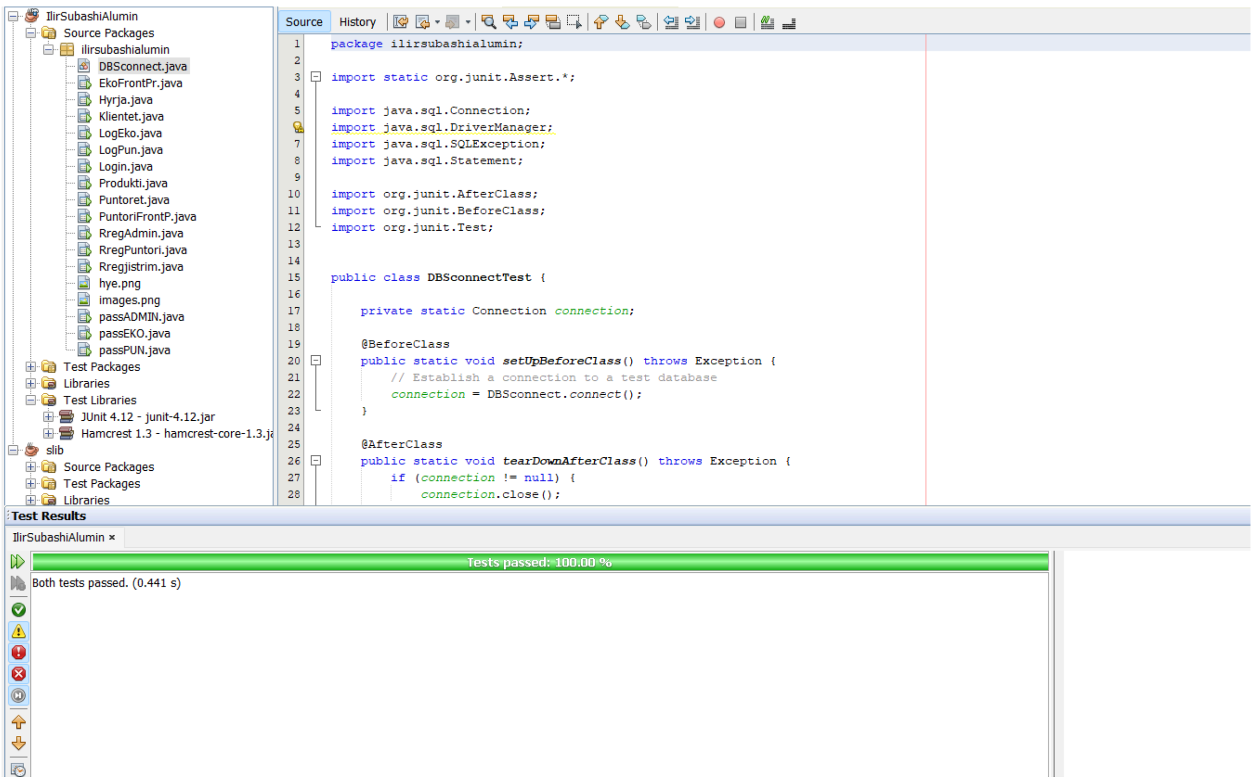
- Passing Tests: Successful tests indicate that the functionality behaves as expected.
- Failing Tests: Tests encountering unexpected behavior fail, highlighting discrepancies between expected and actual behavior.most of the problems happened with assertions
- Error Scenarios: Errors occur due to unexpected exceptions or runtime issues during test execution.This happened because the GUI components have private access to the main code.
- Debugging: Use NetBeans' built-in debugging tools to diagnose test failures and errors.
- Fixing Issues: Modify application or test code as necessary to address identified issues.
- Regression Testing: Re-run affected tests and broader regression test suites to ensure fixes don't introduce regressions.

**3. Reporting:**

- NetBeans provides detailed reports summarizing test execution results, including passed, failed, and skipped tests, along with any errors encountered.
- Utilize these reports to communicate testing outcomes to stakeholders.

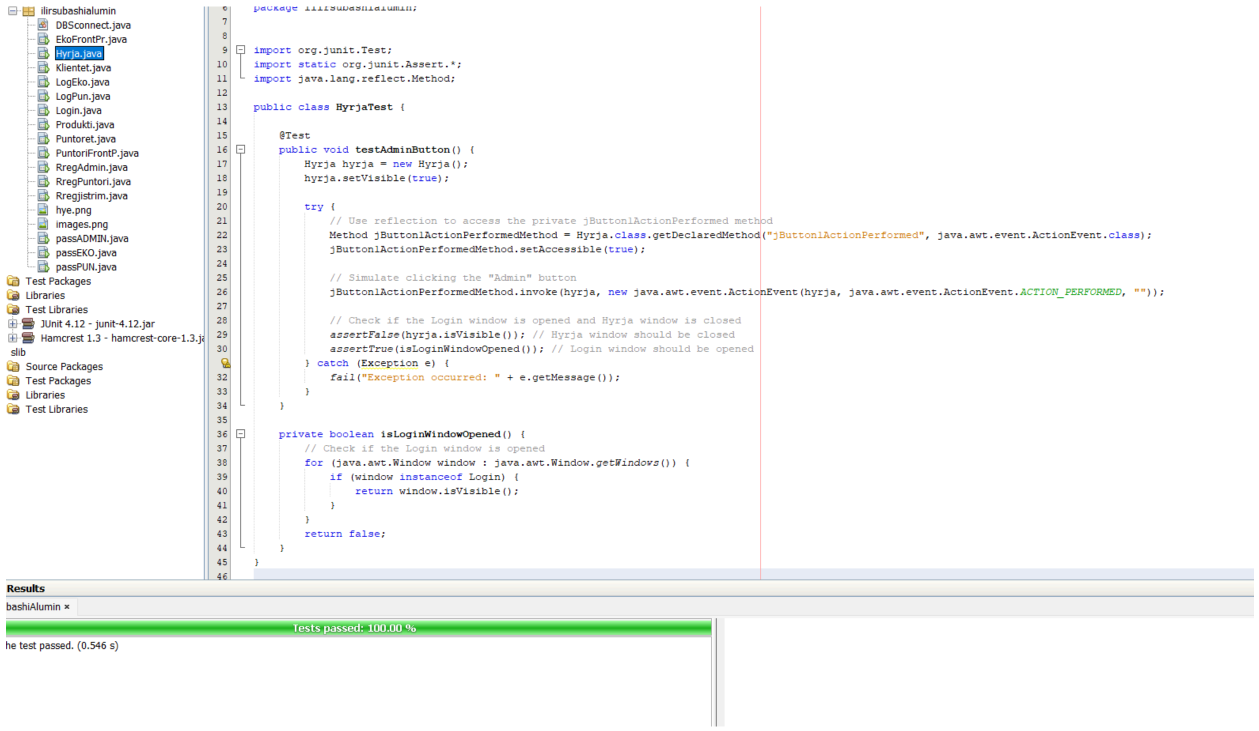


```java
package ilirsubashialumin;

import static org.junit.Assert.*;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;


public class DBSconnectTest {

    private static Connection connection;

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
        // Establish a connection to a test database
        connection = DBSconnect.connect();
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
        if (connection != null) {
            connection.close();
        }
    }

    @Test
    public void testConnection() {
        assertNotNull("Connection should not be null", connection);
        try {
            assertFalse("Connection should not be closed", connection.isClosed());
        } catch (SQLException e) {
            fail("SQLException occurred: " + e.getMessage());
        }
    }

    @Test
    public void testDatabase() {
        // Add test cases specific to your database structure or requirements
        try {
            Statement statement = connection.createStatement();

            statement.close();
        } catch (SQLException e) {
            fail("SQLException occurred: " + e.getMessage());
        }
    }
}
```



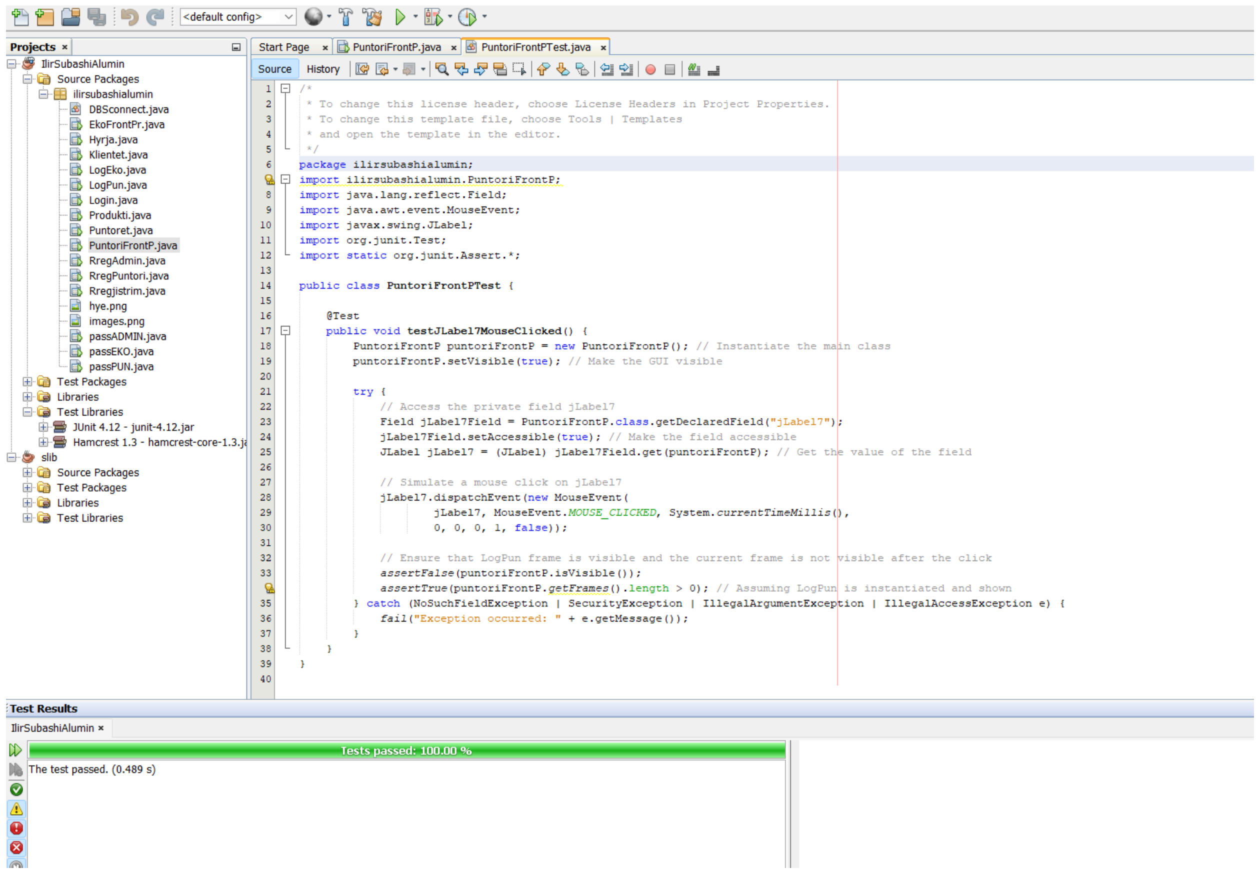
```java
package ilirsubashialumin;

import static org.junit.Assert.*;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;


public class DBSconnectTest {

    private static Connection connection;

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
        // Establish a connection to a test database
        connection = DBSconnect.connect();
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
        if (connection != null) {
            connection.close();
```

```
Both tests passed. (0.441 s)
```



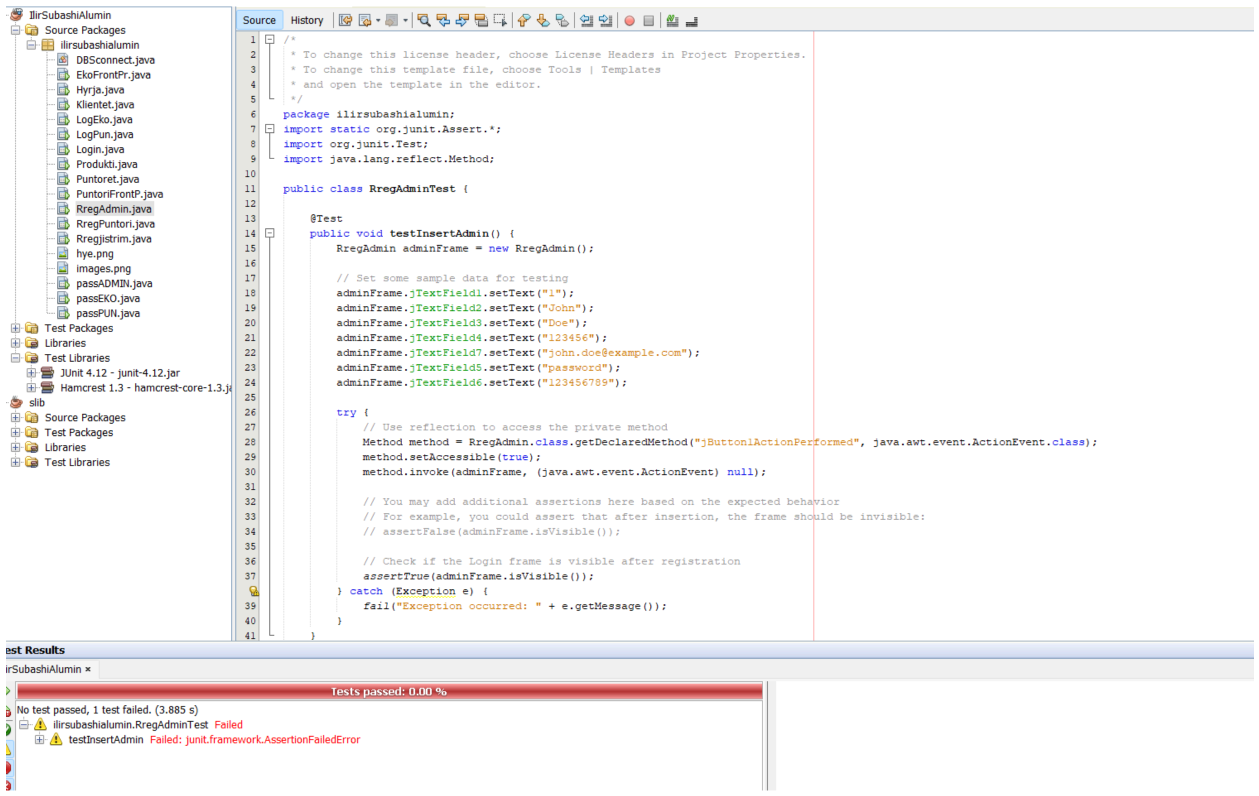
```java
package ilirsubashialumin;


import org.junit.Test;
import static org.junit.Assert.*;
import java.lang.reflect.Method;

public class HyrjaTest {

    @Test
    public void testAdminButton() {
        Hyrja hyrja = new Hyrja();
        hyrja.setVisible(true);

        try {
            // Use reflection to access the private jButton1ActionPerformed method
            Method jButton1ActionPerformedMethod = Hyrja.class.getDeclaredMethod("jButton1ActionPerformed", java.awt.event.ActionEvent.class);
            jButton1ActionPerformedMethod.setAccessible(true);

            // Simulate clicking the "Admin" button
            jButton1ActionPerformedMethod.invoke(hyrja, new java.awt.event.ActionEvent(hyrja, java.awt.event.ActionEvent.ACTION_PERFORMED, ""));

            // Check if the Login window is opened and Hyrja window is closed
            assertFalse(hyrja.isVisible()); // Hyrja window should be closed
            assertTrue(isLoginWindowOpened()); // Login window should be opened
        } catch (Exception e) {
            fail("Exception occurred: " + e.getMessage());
        }
    }

    private boolean isLoginWindowOpened() {
        // Check if the Login window is opened
        for (java.awt.Window window : java.awt.Window.getWindows()) {
            if (window instanceof Login) {
                return window.isVisible();
            }
        }
        return false;
    }
}
```

```
Tests passed: 100.00 %
he test passed. (0.546 s)
```



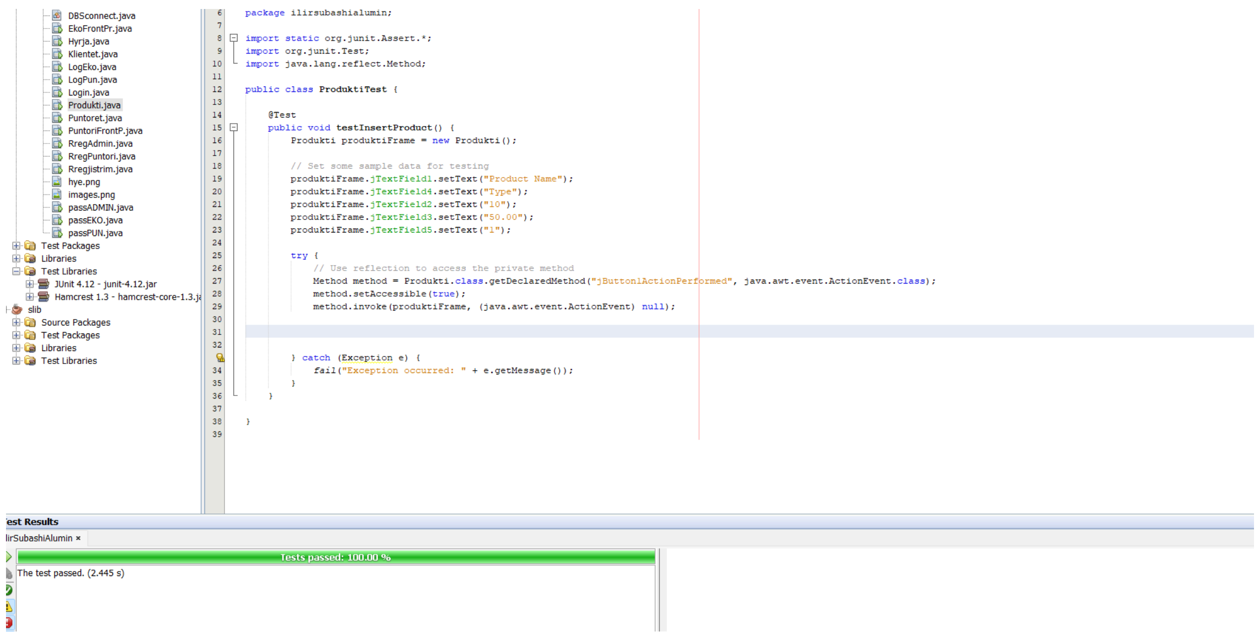
```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ilirsubashialumin;
import ilirsubashialumin.PuntoriFrontP;
import java.lang.reflect.Field;
import java.awt.event.MouseEvent;
import javax.swing.JLabel;
import org.junit.Test;
import static org.junit.Assert.*;

public class PuntoriFrontPTest {

    @Test
    public void testJLabel7MouseClicked() {
        PuntoriFrontP puntoriFrontP = new PuntoriFrontP(); // Instantiate the main class
        puntoriFrontP.setVisible(true); // Make the GUI visible

        try {
            // Access the private field jLabel7
            Field jLabel7Field = PuntoriFrontP.class.getDeclaredField("jLabel7");
            jLabel7Field.setAccessible(true); // Make the field accessible
            JLabel jLabel7 = (JLabel) jLabel7Field.get(puntoriFrontP); // Get the value of the field

            // Simulate a mouse click on jLabel7
            jLabel7.dispatchEvent(new MouseEvent(
                    jLabel7, MouseEvent.MOUSE_CLICKED, System.currentTimeMillis(),
                    0, 0, 0, 1, false));

            // Ensure that LogPun frame is visible and the current frame is not visible after the click
            assertFalse(puntoriFrontP.isVisible());
            assertTrue(puntoriFrontP.getFrames().length > 0); // Assuming LogPun is instantiated and shown
        } catch (NoSuchFieldException | SecurityException | IllegalArgumentException | IllegalAccessException e) {
            fail("Exception occurred: " + e.getMessage());
        }
    }
}
```

```
Tests passed: 100.00 %
The test passed. (0.489 s)
```



```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ilirsubashialumin;
import static org.junit.Assert.*;
import org.junit.Test;
import java.lang.reflect.Method;

public class RregAdminTest {

    @Test
    public void testInsertAdmin() {
        RregAdmin adminFrame = new RregAdmin();

        // Set some sample data for testing
        adminFrame.jTextField1.setText("1");
        adminFrame.jTextField2.setText("John");
        adminFrame.jTextField3.setText("Doe");
        adminFrame.jTextField4.setText("123456");
        adminFrame.jTextField7.setText("john.doe@example.com");
        adminFrame.jTextField5.setText("password");
        adminFrame.jTextField6.setText("123456789");

        try {
            // Use reflection to access the private method
            Method method = RregAdmin.class.getDeclaredMethod("jButton1ActionPerformed", java.awt.event.ActionEvent.class);
            method.setAccessible(true);
            method.invoke(adminFrame, (java.awt.event.ActionEvent) null);

            // You may add additional assertions here based on the expected behavior
            // For example, you could assert that after insertion, the frame should be invisible:
            // assertFalse(adminFrame.isVisible());

            // Check if the Login frame is visible after registration
            assertTrue(adminFrame.isVisible());
        } catch (Exception e) {
            fail("Exception occurred: " + e.getMessage());
        }
    }
```

```
No test passed, 1 test failed. (3.885 s)
ilirsubashialumin.RregAdminTest Failed
    testInsertAdmin Failed: junit.framework.AssertionFailedError
```



```java
package ilirsubashialumin;

import static org.junit.Assert.*;
import org.junit.Test;
import java.lang.reflect.Method;

public class ProduktiTest {

    @Test
    public void testInsertProduct() {
        Produkti produktiFrame = new Produkti();

        // Set some sample data for testing
        produktiFrame.jTextField1.setText("Product Name");
        produktiFrame.jTextField4.setText("Type");
        produktiFrame.jTextField2.setText("10");
        produktiFrame.jTextField3.setText("50.00");
        produktiFrame.jTextField5.setText("1");

        try {
            // Use reflection to access the private method
            Method method = Produkti.class.getDeclaredMethod("jButton1ActionPerformed", java.awt.event.ActionEvent.class);
            method.setAccessible(true);
            method.invoke(produktiFrame, (java.awt.event.ActionEvent) null);


        } catch (Exception e) {
            fail("Exception occurred: " + e.getMessage());
        }
    }

}
```

```
The test passed. (2.445 s)
```

## 8. TEST COVERAGE

Achieving high test coverage is crucial for ensuring thorough testing of software. Test coverage refers to the extent to which the code and functionalities of a software application are exercised by test cases. Here's why high test coverage is important:

1. **Identification of Uncovered Areas**: High test coverage helps in identifying areas of the codebase that have not been adequately tested. Uncovered areas are potential breeding grounds for bugs and defects that may remain undetected until the software is in use. By striving for comprehensive coverage, developers and testers can ensure that all parts of the application are rigorously tested, reducing the likelihood of critical issues slipping through.

2. **Improved Software Quality**: Thorough testing contributes to higher software quality. When a significant portion of the codebase is covered by tests, it increases confidence in the software's reliability and functionality. High test coverage means that more scenarios and edge cases are considered during testing, leading to fewer bugs and defects in the final product. This, in turn, enhances the overall user experience and satisfaction with the software.

3. **Effective Maintenance and Refactoring:** High test coverage makes software maintenance and refactoring easier and safer. When developers need to make changes to the codebase, whether it's adding new features or fixing bugs, having a comprehensive suite of tests provides a safety net. Tests act as a form of documentation, helping developers understand the intended behavior of the code and ensuring that modifications do not inadvertently introduce regressions or break existing functionality.

4. **Facilitates Continuous Integration and Delivery**: In modern software development practices such as continuous integration and delivery (CI/CD), having high test coverage is essential. Automated testing is a cornerstone of CI/CD pipelines, allowing teams to rapidly deploy changes with confidence. High test coverage ensures that automated tests provide meaningful feedback on the quality of each code change, enabling teams to release software more frequently and reliably.

In summary, achieving high test coverage is vital for thorough testing of software because it helps identify uncovered areas, improves software quality, facilitates maintenance and refactoring, and enables effective CI/CD practices. By prioritizing comprehensive testing, software development teams can deliver more reliable and robust applications to their users.