**Software Project Documentation**

MyLittleShop

Team:

1. Nguyen Hai Duc - 9701
2. Trần Minh Hoàng - 11112
3. Le Hoang Quan - 9307
4. Nguyen The Viet (Team leader) – 9990

Project leader email: cs2015\_viet.nt@student.vgu.edu.vn

**Prepared by:**

Nguyen The Viet

1. **Introduction**

MyLittleShop is a desktop application which is available for PC users. The project is built in JavaEE platform. For our project, we assume that:

* Each shop uses a laptop and has internet connection
* Each laptop has its own default camera (or third-party camera)

Client side provides access to the web services (Axis2) on server side via HTTP/HTTPS connection. For security reason, message exchange is configured to use HTTPS port.

This document provides information of the requirements of the project, the architecture of the application, control management, risk managements and constraints that the project has.

1. **Requirements**
   1. **Functional requirements**
      1. **Use case 1:**

Description: **Login System**

Actor: Client (employee & manager)

Basic Path:

1. Client inputs data (username and password), chooses a role

1.1. If role is employee, client chooses shop ID, then moves to step 2.

1. System sends request to the server side to validate authentication.
2. System prompts success, client access to given functions.
3. Client clicks “Logout”
4. Returns to login page.

Alternate Path:

4. System prompts failure, client re-enters the username and password again  
 5. Back to step 1 of the basic path.

**2.1.2. Use case 2:**

Description: **Barcode Scanner**

Actor: Client (Employee)

Basic Path:

1. Client logs in as employee.
2. Employee clicks “Scan” to start the process.
3. Camera stream appears. Employee scans the barcode.
4. Camera stream shuts down.

4.1. Client goes back to step 2, else moves to step 5.

1. Request is sent to server.
2. Server returns product name based on barcode in database.
3. Employee clicks “Submit”.
4. Server receives request, returns datasets and updates transaction history.
5. Employee clicks “Logout”.
6. Returns to login page.

**2.1.3. Use case 3:**

Description: **Barcode Generator**

Actor: Client (Manager)

Basic Path:

1. Client logs in as manager
2. Manager clicks “Generate Barcode”
3. Manager inputs value of barcode and submits
4. An image of barcode (.jpg) is generated
5. Manager clicks “Logout”
6. Returns to login page

**2.1.4. Use case 4:**

Description: **Shop Balance**

Actor: Client (Manager)

Basic Path:

1. Client logs in as manager.
2. Client sends request to server.
3. Server returns balance view (quantity & price) of products in all shops.
4. Manager clicks “Logout”.
5. Returns to login page.

**2.1.5. Use case 5:**

Description: **See Transactions**

Actor: Client (Manager)

Basic Path:

1. Client logs in as manager
2. Client clicks “See transactions”
3. Manager changes (restock) the quantity of a product in a particular shop by taking items from other shops if available.
4. Manager clicks “Submit”.
5. Server receives request, update the database and return message.
6. Manager clicks “Logout”
7. Returns to login page.

**2.1.6. Use case 6:**

Description: **Restock**

Actor: Client (Manager)

Basic Path:

1. Client logs in as manager.
2. Manager clicks “Restock”.
3. Manager chooses product to input.
4. Manager input quantity and chooses shop ID.
5. Manager clicks “Ok”.
6. Request is sent to the server.
7. Server updates database.
8. Manager click “logout”.
9. Returns to login page.

**2.1.7. Use case 7:**

Description: **Add shop**

Actor: Client (Manager)

Basic Path:

1. Client logs in as manager.
2. Manager clicks “Add Shop/Product”.
3. Manager selects “Shop”.
4. Manager inputs shop name, employee name, password and password confirmation.
5. Manager selects “Ok”.
6. Sever updates database.
7. Manager clicks “logout”.
8. Returns to login page.

**2.1.7. Use case 8:**

Description:  **Add product**

Actor: Client (Manager)

Basic Path:

1. Client logs in as manager.
2. Manager clicks “Add Shop/Product”.
3. Manager selects “Product”.
4. Manager inputs product name, barcode, and price.
5. Manager selects “Ok”.
6. Sever updates database.
7. Manager clicks “logout”.

**2.2 Non-functional requirements**

**2.2.1 Security**

* Data exchange between client and server is encrypted
* Client password is hashed and stored in database

**2.2.2 Cost of server**

* Demo is deployed in local server for cost-saving purpose
* Future hosting on the internet should be economical

**3. Architecture**

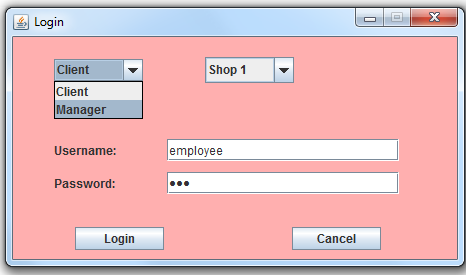
MyLittleShop is a client – server application which is built on JavaEE with Dynamic Web Project and Axis2 Web services. Client is a desktop application and server is a web application that provides Axis2 web services. The back-end of the server connects to the database remotely by using Java and JBDC as connector to MySQL.

* Client is a desktop application which is mainly coded in Java
* Server side provides Axis2 web services for client side which are run on Tomcat server 7.0+.
* Back-end of server is coded in Java, which handles client requests as parameters and connects to Database (MySQL) remotely by JDBC.
* Application makes use of the portable web application phpMyAdmin that provides administrations tool for MySQL (port 3306) that runs on the local server (Apache server port 80).

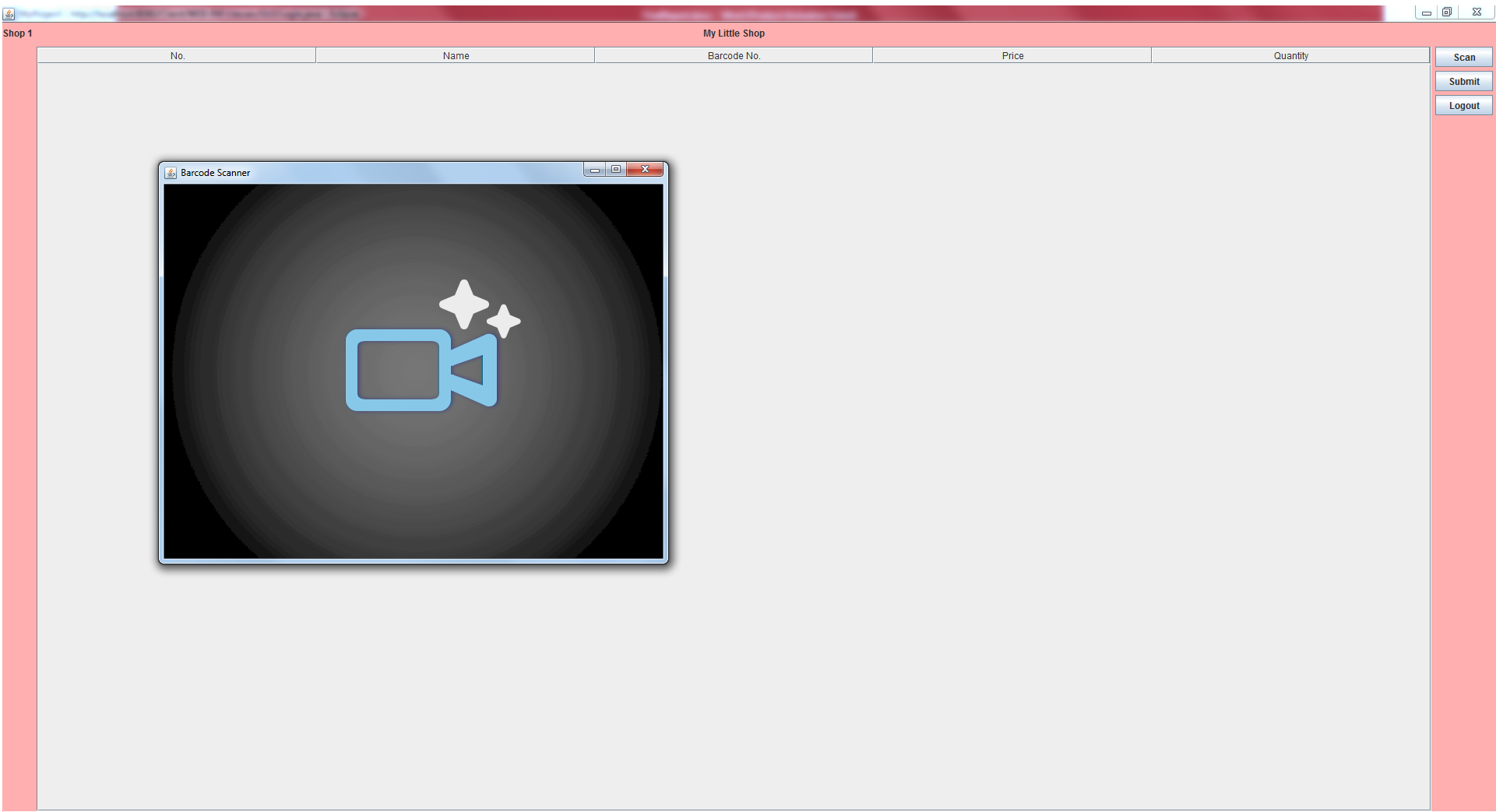
**Client:** Desktop application:

Graphical User Interface is created mainly with Java Swing Library:

* 1. Initial GUI for starting the application: Login system
     1. Role: Client/Manager
     2. Username
     3. Password
     4. Shop: chosen shop

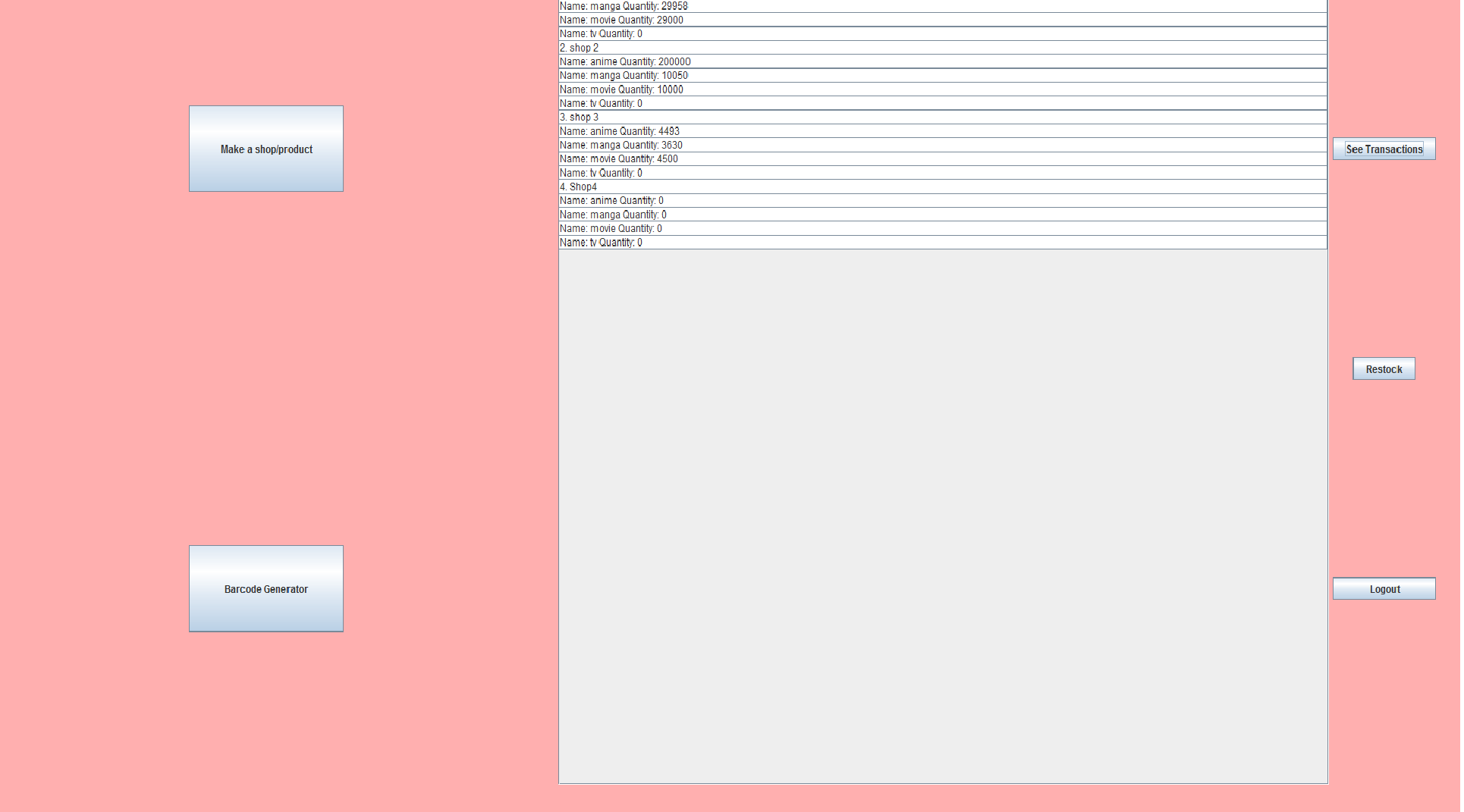
**Figure 3.1 Login interface (Java Swing)**

* 1. GUI of Employee:
     1. Scan: opens camera stream to scan the barcode
     2. Submit: sends request to server side (retrieving product name, storing transactions)
     3. Logout: return the login page



**Figure 3.2 Employee interface (Java Swing)**

* 1. GUI of Manager:
     1. Make a shop/product: create new shop/product
     2. Bacode Generator: produce new barcode
     3. See transactions: view transaction (all and between dates, transactions can be categorized as IN or OUT)
     4. Restock: add product quantities into database



**Figure 3.3 Manager interface (Java Swing)**

**Server:** Web application

1. Run on Tomcat server 7.0+
2. Provides Axis2 web services (Back-end is written in Java):
3. Communicates with client side via HTTP
4. Connects with Database (MySQL) via JDBC (queries)
5. Database: uses phpMyAdmin (run on Apache local web server) tool to run SQL queries. The database is also hosted in a local server which can be accessed from the server side. There are 4 tables in the database. “Transactions” table has two foreign keys: **barcode** reference to **barcode** (primary key) in “product”, **shopid** reference to **id** (primary key) in “shop”. However due to lack of time, database structure has some constraints that will be discussed in the last part.

**5. Management Control**

**5.1 Version Control**

The project source code is uploaded weekly on Github (with support tool Git Bash to clone the local repository, pull, push, commit and merge branch, etc.).

Since there were some unexpected errors that prevented some laptops from creating webs services axis2 that cost too much time to fix, we decided to upload different .rar files with different versions to github in order to keep track of the server errors.

Link to github: <https://github.com/KuroViet97/MyLittleShop/>

KuroViet97: Viet (master)

Hoang19101997: Hoang

nguyenhaiduc1994: Duc

quan: Quan  
The version control is a bit messy, and it is my fault not to control it well for this project since this is my first experience using github and being a manager. Specific tasks are listed in **part 5.3.**

**5.2 Functions**

* **Client:**

1. Barcode scanner and generators (128): <https://github.com/KuroViet97/MyLittleShop/tree/master/Client_Side/Barcode>
2. Login, employee, manager GUI: <https://github.com/KuroViet97/MyLittleShop/tree/master/Client_Side/GUI>
3. Client service:

<https://github.com/KuroViet97/MyLittleShop/tree/master/Client_Side/ClientService>

Explanation for clientGUI.java is clientGUI.txt

Explanation for MylittleStub.java is MylittleStub.java

* **Server :**

1. Service functions and retrieving data from database:

<https://github.com/KuroViet97/MyLittleShop/tree/master/Server_Side/Backend>

1. Database back-up:

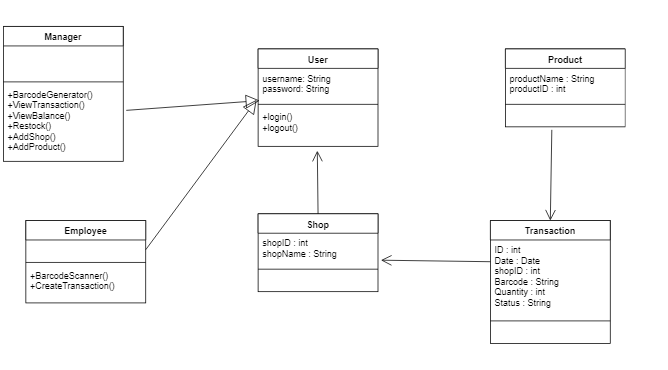
<https://github.com/KuroViet97/MyLittleShop/tree/master/Server_Side/DatabaseCode>

1. Hash function (SHA-256): to hash password into database

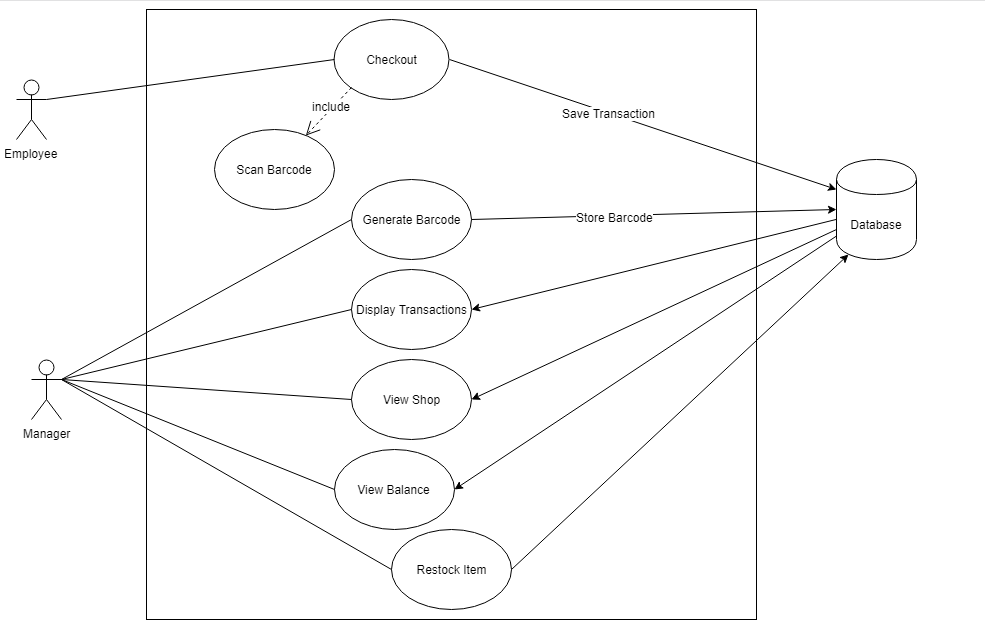
<https://github.com/KuroViet97/MyLittleShop/tree/master/Server_Side/HashFunction>

**Diagram:**

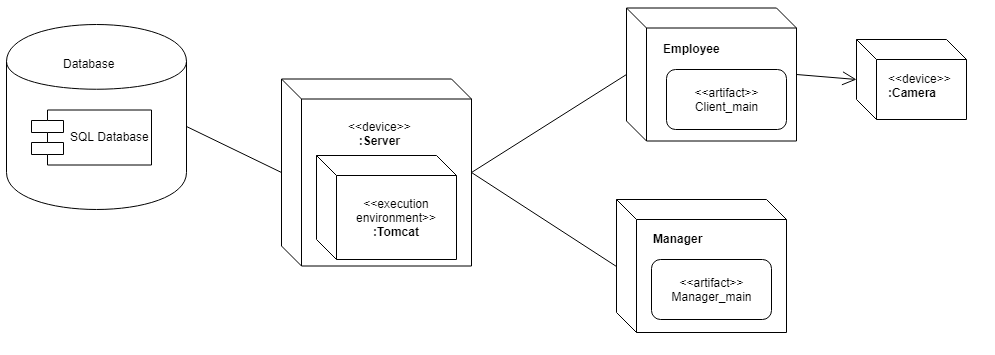
1. **UML class diagram**

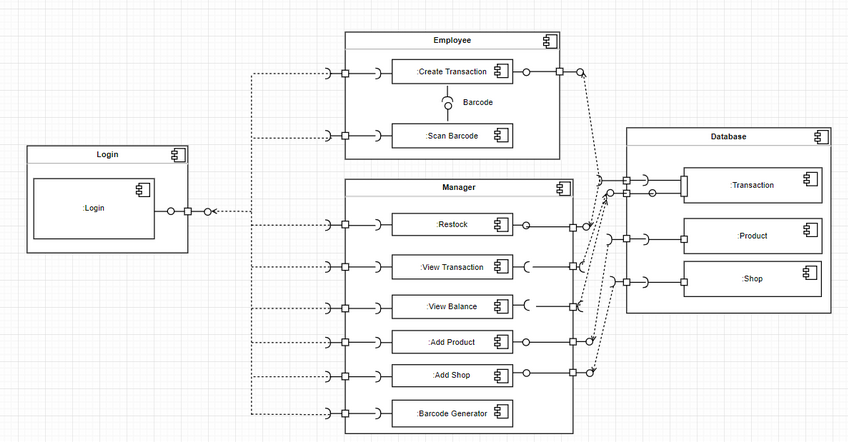
****

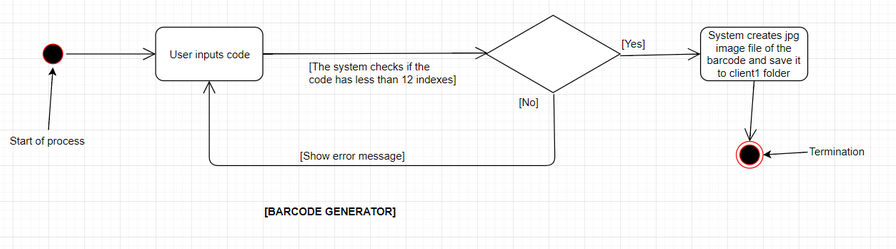
1. **Use case diagram**

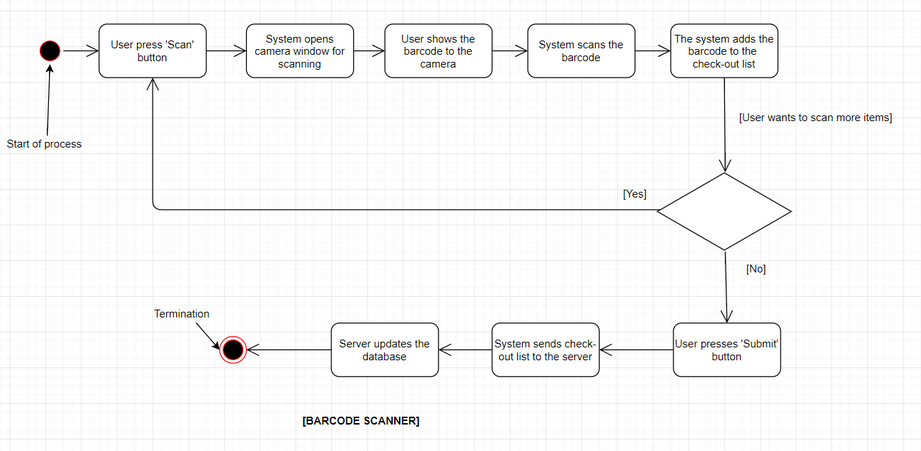
****

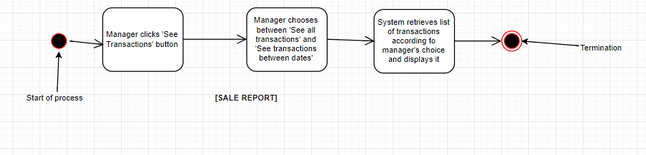
1. **Deployment diagram**

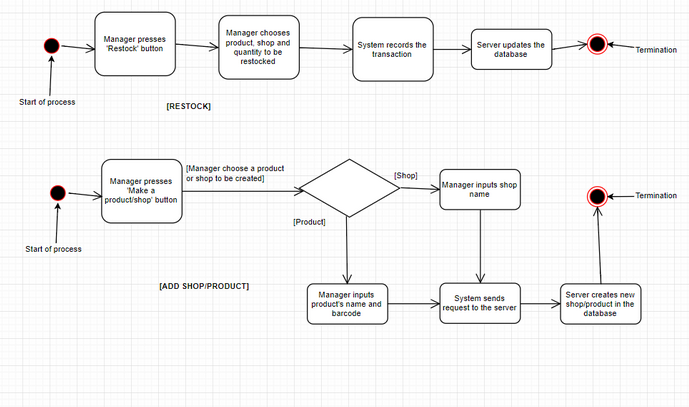
****

1. **Component diagram**
2. **Activity diagrams**

****

****

****

****

**5.3 Tasks**

**5.3.1. Le Hoang Quan – 9307**

* + Design graphical user interface for client side (Java Swing)
  + Integrate client functions into GUI
  + Implement codes that handle all client requests to server, display data on GUI (client-side)
  + Implement codes that receives and responses requests in back-end server

**5.3.2. Nguyen Hai Duc – 9701**

* + Implement Barcode reader & generator in client side
  + Draw diagrams of the design architecture of the projects
  + Assist writing the report about **how axis2 web services in server communicate with client**

**5.3.3. Tran Minh Hoang - 11112**

* + Design the connection between client and server using Axis2 web services (converting parameters into XML messages and parsing java objects)
  + Design the way to connect to database (using JDBC)
  + Implement security parts:
    - Messages (XML) between client and server are exchanged via **HTTPS port 8443** instead of HTTP port 8080

**5.3.4 Nguyen The Viet – 9990**

* + Design project architecture
  + Create relational database (MySQL) and connect it to server (using JDBC)
  + Implement codes for hash function (sha256) in back-end server. (Login passwords are stored as hash value inside the database)
  + Implement codes for connecting server to database and handle all queries for service functions
  + Draw UML Class diagram
  + Write documentation of the project
  + Write installation instruction for the project

**5.3 Risk Management**

**5.3.1. Random people are chosen to form a group**:

* + Risk: People have different strength, experience and weakness.
  + Solution: face-to-face meeting discussion about the common design of the architecture, analyzing potential of each member and assigning appropriate task to each member. Regular communication to each other is highly concerned.

**5.3.2. Changes in requirements:**

* + Risk: Old functions may need to be changed to adapt new requirements, or possible additional functions can be added into the application
  + Solution: spend Q/A times with lecturer regularly, project architecture should be designed in the way that functions are created independently and easy to maintain or change.

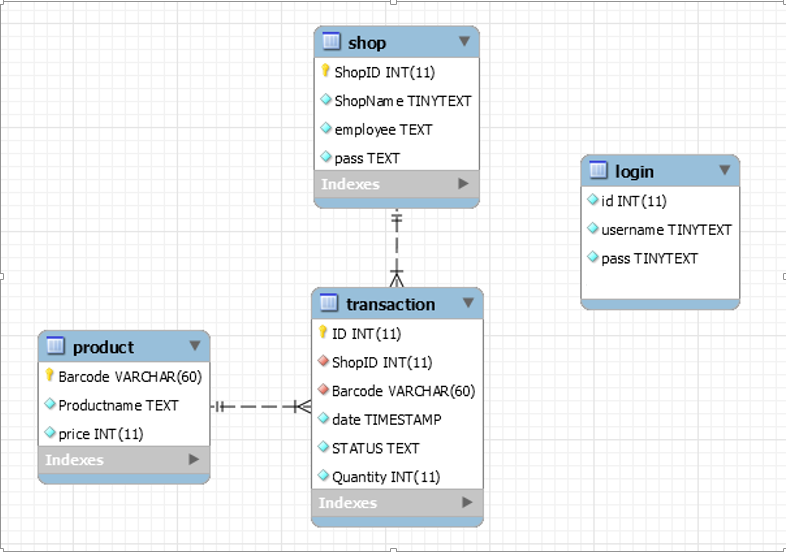
**5.3.3. Project complexity:** Project involves using new technology to create services, provide connection between client – server

* + Risk: Some technologies are quite complex that require deep understanding to implement (time-consuming) and cause confusion among team members
  + Solution: Discuss and analyze the structure and requirement of the project, choose a technology that all team members are able to understand and implement.

1. **Constraints**

**6.1 Database:**

a) We did not think far about the future implementation, so we did not provide access control at first. After the last Q/A time today, we decided to temporarily fix the solution: Whenever a shop is created, only one new employee is created and assigned to that shop. Both share the same ID in the database. The login table only stores manager username and password, and currently there is only one manager.



For future implementation, the database can be reconstructed: Table “login”: username, password, role (employee/manager) and shopID (foreign key) which references to ID in table “shop”.

b) We only store the “price” as a “dirty way” in the product table to display the price to the GUI. This can only allow us to update the new price, but the data of the old price will be lost and it is impossible to provide “discount” section. We made this decision because we did not have enough time to think of a new solution and implement it.

**6.2 Barcode reader:**

Barcode reader depends on the quality of the camera of the laptop used, so the quality of the scanned result could be different sometimes (it could be wrong or may take time to scan because the camera stream is too blur and old). Solution: every shop should have a standard device that can run the application.

**6.3 Security:**

We tried to implement token-based login system by using WS-security in Axis2, but the system failed when we added more services into backend server. Since we have not figured out all the problems and there may be risks of failure running the project, we decided to omit that part.

1. Password is stored as hash value in database. Whenever client logs in, client side will send request to server, the server will then hash the value and compared it with that of the database. If two values match, client can access to services
2. However, server cannot recognize which type of client is sending requests after logging in because there is no token to compare to grant access. A temporary solution: let client enter username, password and role again each time they use the service. This solution is inefficient.
3. A good point of the connection is that messages are transmitted HTTPS, which prevented data from being hacked.

**6.4 Naming:**

Since this is the first time we work randomly as a team and the project is quite long, the file naming (package, class, function) of some members did not follow the common standard rules (lowercase and uppercase). And this is also my fault that I forgot to inform the rules properly at the beginning of the project.