Software Project Documentation

Team Member:

1.Đinh Công Huy – 10383 (Leader)

2.Hoàng Thái Dương - 11052

3.Lê Khắc Hoàn Vũ - 10211

1.Overview

MyLittleShop is a system which contains a combination of two different web application:

- A single web application serving as a frontend, which shows the graphical interface, requests data from
- A RESTful web application serving as a backend that handles request from the frontend, queries into database and responses with appropriate message

This documentation contains information about requirements of the project, design architecture, the task distribution between member of the group, how to deploy the application, task has been done and possible future implementation.

2.Requirements:

2.1.Functional requirements

2.1.1. Use case 1: Login

2.1.2. Use case 2: List products

2.1.3. Use case 3: List transactions

2.1.4. Use case 4: Create a user

2.1.5. Use case 5: List users

2.1.6. Use case 6: Logout

2.1.7. Use case 7: Update a user

2.1.8. Use case 8: Create a product

2.1.9. Use case 9: Delete a product

2.1.10. Use case 10: Create a transaction

2.1.11. Use case 11: Update a product

2.2 Non-functional requirements

2.2.1 Security

- The password is stored in the database.
- Each user upon logging in will be given a random token. The randomization is implemented using the library UUID. When a user logs out, their token will be deleted in the database.

2.2.2 Cost of server

- Demo is utilized in local server for the intention of cost saving

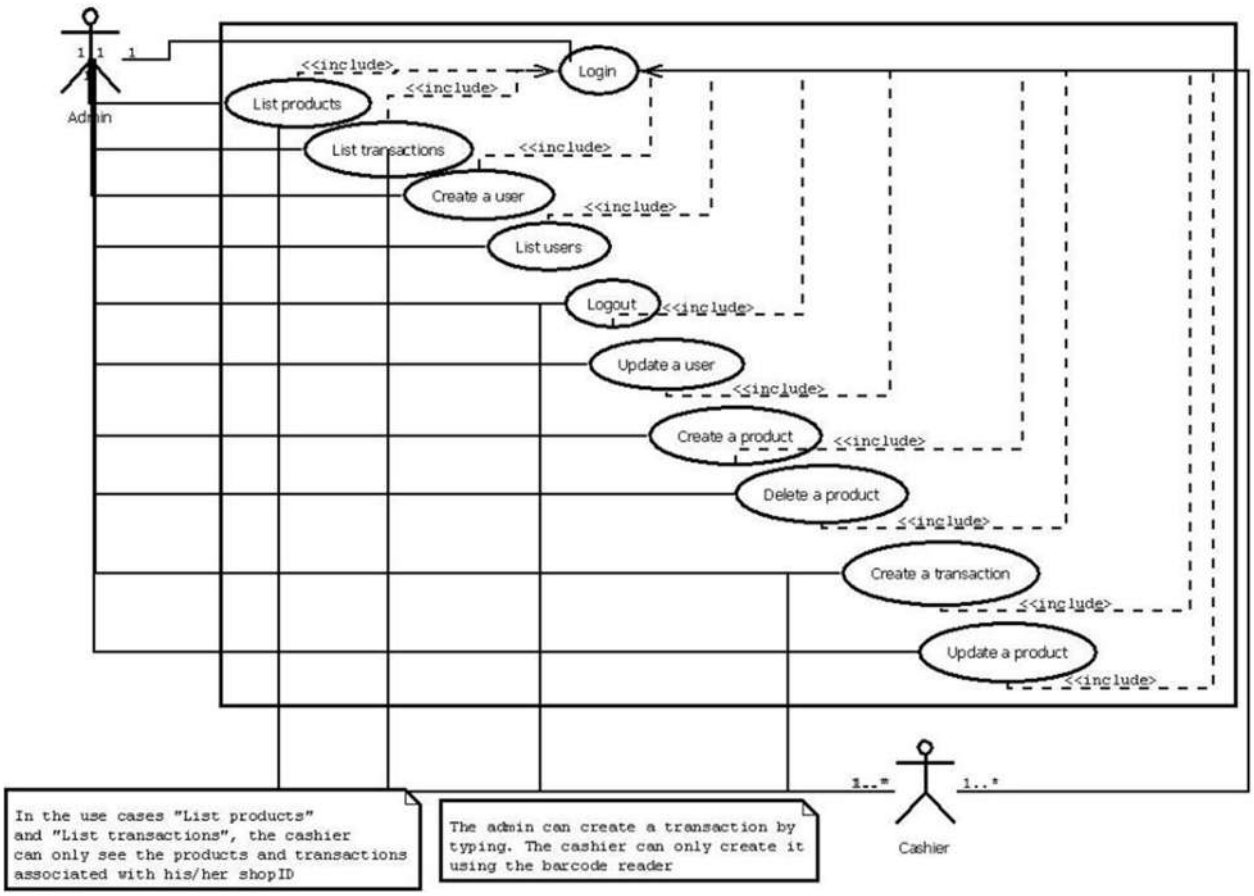
3.Design and Architecture

MyLittleShop is composed of a single page application frontend and a RESTful service backend.

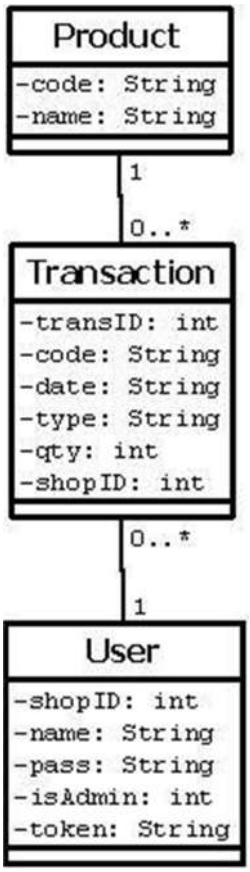
The frontend retrieves all data each time it is booted up. It sends requests to the backend to retrieve information or to modify the database.

The backend acts as a black box, receiving request and responding appropriately to them. The backend follows REST architecture, with Json (JavaScript Object Notation) for the requests and responses. The backend publishes an application program interface (API) for client to interact (client can be a browser, mobile or desktop application, given that they knows the format of the requests). The backend also interacts with the database in order to retrieve and save data persistently.

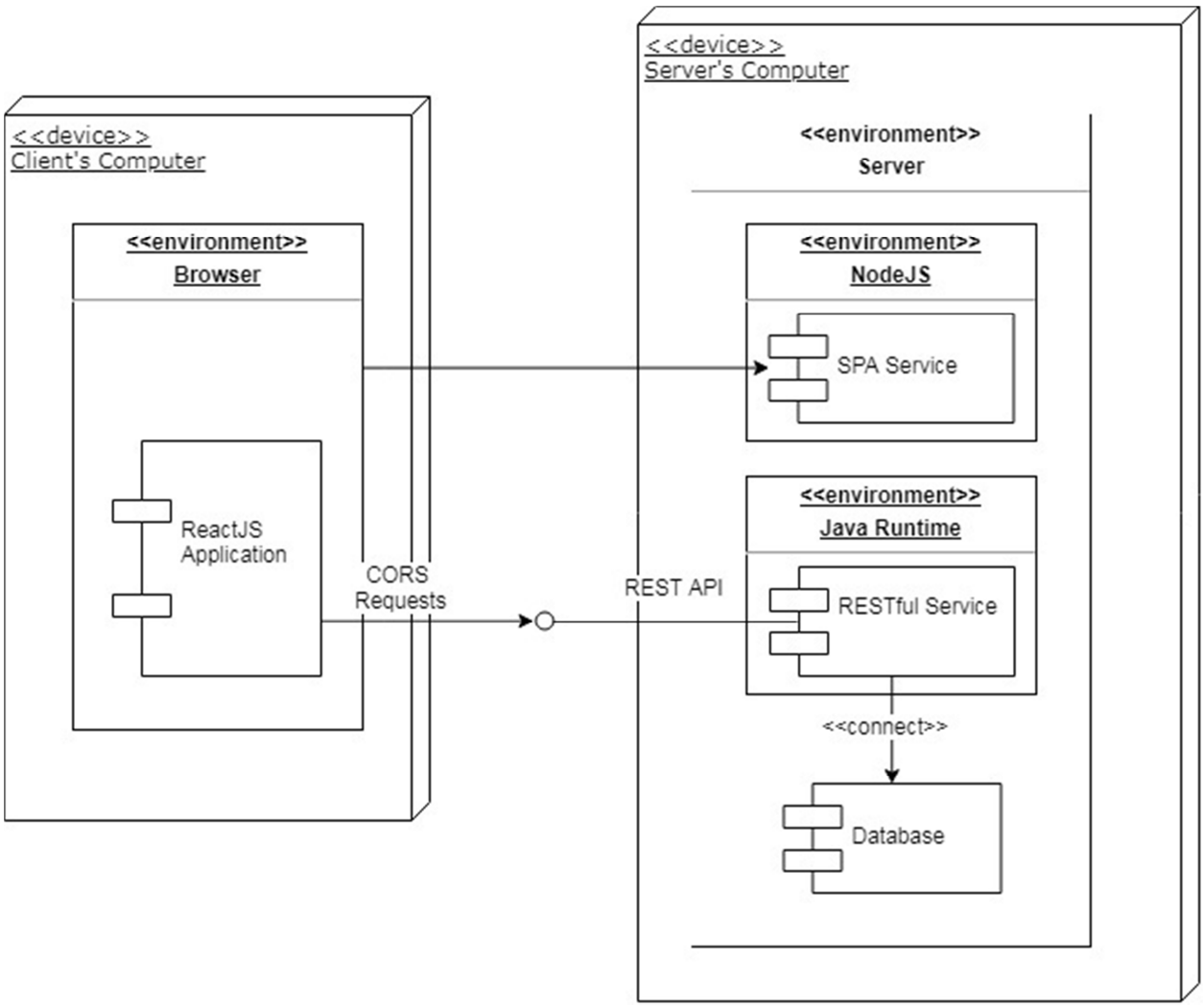
3.1 Use Case Diagram



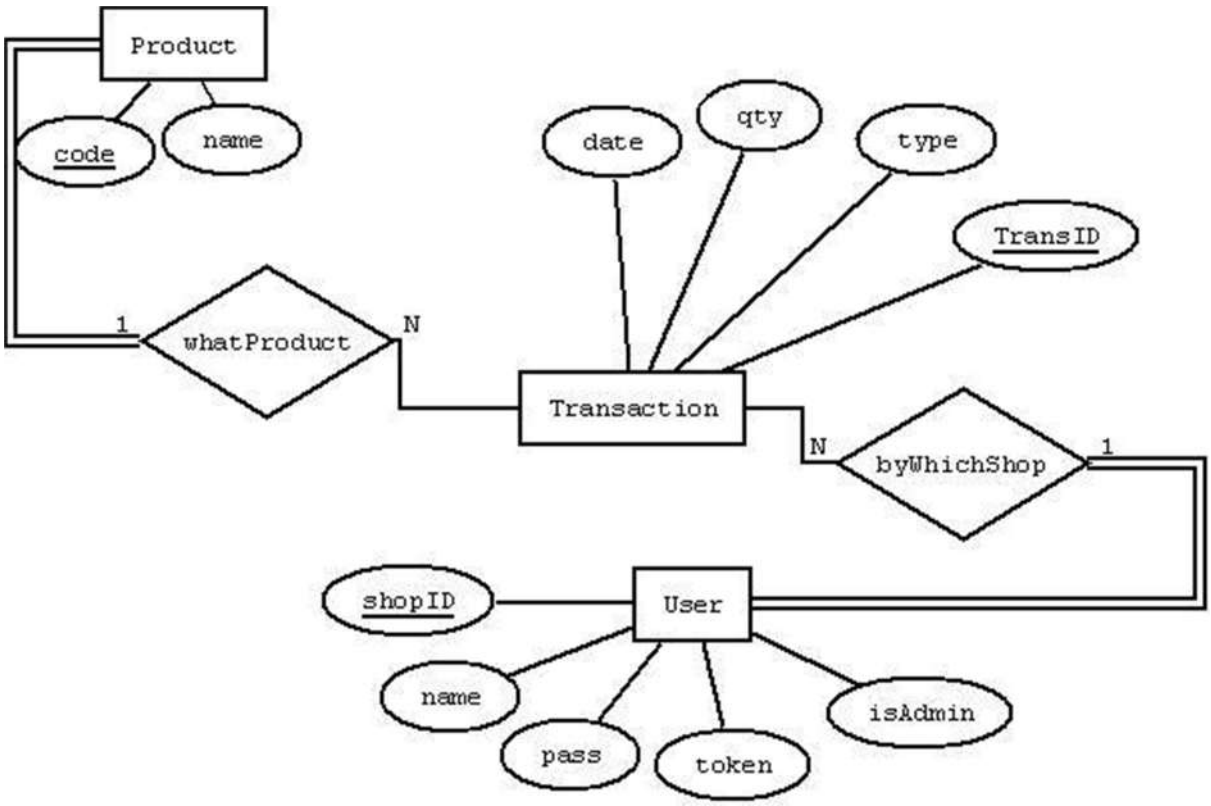
3.2. Class Diagram



3.3. Deployment Diagram



3.4.Database Entity Relation Diagram



4.Technology:

4.1.The frontend:

We use the framework ReactJS in order to build the application. ReactJS is a famous Javascripts framework for building single page application. To manage dependencies, we use NodeJS's npm and yarn, as well as web-pack for hot-reloading (automatically rerun server when code changes). We also uses CSS preprocessors (SASS) and Material-UI, which is a frontend framework similar to Bootstrap but designed specifically for ReactJS and follows Google's material design principle.

To write code, we use Visual Code due to its numerous plugin availability that facilitate development speed.

Language: HTML, CSS(SASS), Javascript (ES6)

4.2.The backend:

The backend is built with Spring, which is a famous Java framework for building web application. Maven is used to generate and manage the project's build.

The chosen IDE to write the backend code is Spring Suite Tool, which is an Eclipse-based distribution that makes it simple and easy to develop Spring application.

Language: Java, XML

5.Management

5.1.Task management

Đinh Công Huy:

- Created the frontend application
- Decided request format (how client should send the request and how backend should handle the request)
- Decided the authentication theme

Hoàng Thái Dương: Restful service

- Designed the database, relationship between entity
- Partially wrote code for the Rest API, implemented the logic for the following use case: update user, create new product, delete product, create transaction, update product, delete user

Lê Khắc Hoàng Vũ:

- Partially wrote code for the Rest API (login, list product, list transaction, create user, list user, logout)
- Wrote the authentication scheme
- Helped the leader to write the documentation (draw use case diagram, class diagram, entity diagram, requirement,api documentation)

Nhân:

- Did not contributed to the final result of the project. We tried to allocate works for him but failed, as he could not finished the given tasks and he refused to communicate with us.

5.2.Version Control

We use Github for version control. Due to the nature of the application being a combination of a single page application and a restful web service, we maintain two different Github repositories for each application. We modify code only on the default master branch, each team member will notify the others before pushing any code on the repository. The two repositories can be found at the following url:

The frontend (SPA): https://github.com/huycong151197/pos-spring

The restful service: https://github.com/huycong151197/point-of-sale-v2

6. Deployment and Usage:

6.1.Deployment:

The system needs to run two different application on the server:

The Single Page Application, whose service provider runs on port 3000

The RESTful Service runs on port 8080

How to run the RESTful Service and the database

1. Go to https://spring.io/tools/sts/all. Download the version running on your operating system.
2. Extract the downloaded file.
3. Go to the folder that you extracted the file
4. Go to sts-bundle/sts-3.9.4.RELEASE
5. Launch the STS application.
6. Create a maven project: File > New > Spring Legacy Project (Project name: MyLittleShop)
7. In the Templates box: select Simple Projects > Simple Spring Maven
8. In the project, right click on src/main/java then select New > Package. (set the name of that package as: demo)
9. Copy and paste all classes into the demo package
10. In the project, open the file pom.xml, then click on the tab pom.xml to see its content. Download the file pom_xml on github, open the available pom.xml of the project and replace the content with the downloaded pom_xml (do not overwrite the files – just copy and replace the content inside)
11. Import jar: right click on the project MyLittleShop > Properties > Java > Build Path: in the Libraries tab, select Add external jar (find your path to the jar file and double click on it) > Apply and close
12. Copy and paste the database file mls into the working-space folder of the project.
13. Run: right click on the project: Run as > Spring Boot App

How to install the and run the SPA:

1. install Nodejs: https://nodejs.org/en/download/

2. install npm: https://www.npmjs.com/get-npm (npm is included in the Nodejs installer in Window)

3. install yarn: https://yarnpkg.com/lang/en/docs/install

Via Terminal:

4. git clone https://github.com/huycong151197/point-of-sale-v2.git

5. cd point-of-sale-v2

6. yarn install

7. yarn start

6.2.Usage:

1.Launch 2 application

2.Go to browser and to localhost:3000

3.Username and password of the admin:

- user: admin

- pass: admin

Username and password for a shop:

- user: shop1

- pass: shop1

4. Barcode usage:

- The barcode scanner use CODE 128

- Admin can detect the product base on barcode.

- For the cashier, each time the product is detected an dialog will open and allow user to enter a quantity, if the user press submit, a new transaction of type IN will be added.

7. RESTful API

This section shows the sample requests and responses of the RESTful service (for future alternative frontend of the system):

Methods
1. Method: Login
-URL: /api/auth

- Request method: POST
- Request body: //A request written in JSON format

```
{
   "name": "" //your username,
   "pass": "" //your password
}
```

- Response success:
+ if the user logged in as admin:

```
{
   "role": "admin",
   "token": "" //the randomized token
}
```

+ if the user logged in as a cashier:

```
{
   "role": "cashier",
   "shopID": "" //the shopID of that user,
   "token": "" //the randomized token
}
```

- Response error:

```
{
   "error": "Login Unsuccessful",
}
```

2. Method: List all products

-URL: /product?token= //your token

- Request method: GET
- Response success:

//success if the token exists in its column in the table User of the database

```
[
   {
      "code": "" //barcode of item 1,
      "name": ""//name of item 1
   },
   {
      "code": ""//barcode of item 2,
      "name": "" // name of item 2
   },
         ....
....
....
      {
```

"code": ""//barcode of last item,
"name": "" // name of item 3
}
]
- Response error:
// error if the token does not exist in its column in the table User of the database, or no token provided in the request parameter.
{
"error": "Authentication failed"
}

3. Method: List all transactions
-URL: /api/transaction?token= //your token
- Request method: GET
- Response success:
+ if the token is of an admin:
[
//a list of all transactions
{
"transID": “” //ID of the transaction,
"code": "" //barcode of the item took place in the transaction,
"date": "" //date of the transaction,
"type": "in" / “out”,
"qty": //quantity of the items, how many of it go in/out in the transaction,
"shopID": //ID of the shop which implemented the transaction
},
…
]

+ if the token is of a cashier
[
//a list of transactions implemented by that cashier.
{
"transID": “” //ID of the transaction,
"code": "" //barcode of the item took place in the transaction,
"date": "" //date of the transaction,
"type": "in" / “out”,
"qty": //quantity of the items, how many of it go in/out in the transaction,
"shopID": //ID of the shop of that cashier
},
…
]

- Response error:
// error if the token does not exist in its column in the table User of the database, or no token provided in the request parameter.
{
  "error": "Authentication failed"
4. Method: Create a user
-URL: /api/user?token= //your token
- Request method: POST
- Request body: //A request written in JSON format
{
  "name": "" //your username,
  "pass": "" //your password
}
- Response success:
//success if your token is of an admin, and the “name” you put in your request body did not exist in the table User of the database:
{
  "success": "Create user successful"
}
- Response error:
+ if your token is of a user, or does not exist, or no token provided
{
  "error": "Authentication failed"
}
+ if the “name” you put in your request body already exists in the table User of the database:
{
  "error": "Create user failed"
}
5. Method: Show the list of users
-URL: /api/user?token= //your token
- Request method: GET
- Response success:
+ success if the token is of an admin
[
//a list of all users
  {
    "name": "" //the username,
    "pass": "" //the password,
    "isAdmin": 1 / 0 //1 if the user is the admin, 0 if the user is the cashier,
    "shop": //the shop id of the shop the user is in charge of. 0 if the user is admin
  },
…

]
- Response error:
+ if your token is of a user, or does not exist, or no token provided
{
  "error": "Authentication failed"
}
6.Method: Log out
-URL: /api/user?token= //your token
- Request method: GET
- Response success:
//if your token is of an admin
{
  "success": "logout successfully"
}

7. Method: Update a user
-URL: /api/user/{shopid}?token=
//Input the shop ID of the user you want to update, and your token
- Request method: PUT
- Request body: //A request written in JSON format
{
 "name" : "" //the new username of the going-to-be-updated user,
 "pass" : "" //the password,
}
- Response success:
//if the token is of an admin
{
 "success" : "Change user successfuly"
}

- Response failed:
// if your token is of a user, or does not exist, or no token provided
{
  "error": "Authentication failed"
}

8. Method: Create a product
-URL: /api/product?token= //your token
- Request method: POST
- Request body: //A request written in JSON format
{

"code" : “” //barcode of the new product
"name" : “” //name of the new product
}
- Response success:
//if the token is of an admin, and the barcode did not exist in its column in the table Product, and the barcode is not null
{
"success" : "Create product successfully"
}
- Response error
//does not satisfy the conditions of success
{
"success" : "Create product failed"
}
9.Method: Delete a product
-URL: /api/product/{code}/?token=
//Input the barcode of the product you want to delete, and your token
- Request method: DELETE
- Response success:
//if the token is of an admin, and the barcode did not exist in its column in the table Product, and the barcode is not null
{
"success" : "Delete product successfully"
}
- Response error
//does not satisfy the conditions of success
{
"success" : "Delete product failed"
}
10. Method: Create a transaction (cần Dương hoàn chỉnh)
-URL: /api/transaction?token= //your token
- Request method: POST
- Response success:
//if the token exists and the parameters are correctly input
{
"success" : "Transaction finished"
}
- Response error:
{
"error" : "Transaction failed"
}
11. Update Product

-URL: /api/product/{code}?token= //your token
- Request method: PUT
- Response success:
//if the token exists and the barcode was input
{
  "success" : "Product updated"
}
- Response error:
{
  "error" : "Update product failed"
}
12. Delete User
-URL: /api/user/{shopID}?token= //your token
- Request method: DELETE
- Response success:
//if the token belongs to admin and the user id exists
{
  "success" : "Product deleted"
}
- Response error:
{
  "error" : "Delete user failed"
}