RMIT UNIVERSITY

EEET2580 – Enterprise Application Development

Assignment 2 - Build a backend

Design document

Table of Contents

1. Introduction

The purpose of this report is to document a design of a backend component of an enterprise application to automate business processes. The report includes business requirements, all functionalities of the system with use case diagram, technologies and architect that have been applied for this project. The class diagram is provided to see the project's structure and the classes' relationships. The report states the project's achievements and implementation status with full instructions on the constraints when working with the application. Testing is also conducted and documented to ensure the quality of the system which 100% code and APIs coverage. Some limitations of the project are listed at the end of this report as well. The report does not provide actual code and future development plan or integration plan between backend and front end components.

2. Business requirement

The system is a backend component of an enterprise application with the purpose of providing an automation tool for business processes of a trading company.

The company buys products from providers and sells them to customers. For each activity, we have different data that needs to be saved and managed. To be more specific, there are 4 types of activities including ordering products from providers, receiving shipped products to warehouse, delivering products out of warehouse and shipping them to the customers. Hence, we also have 4 types of documenting acknowledgements for all the activities including order, inventory receiving note, inventory delivery note and sales invoice respectively. The activities involve different stakeholders such as customers, providers or staff.

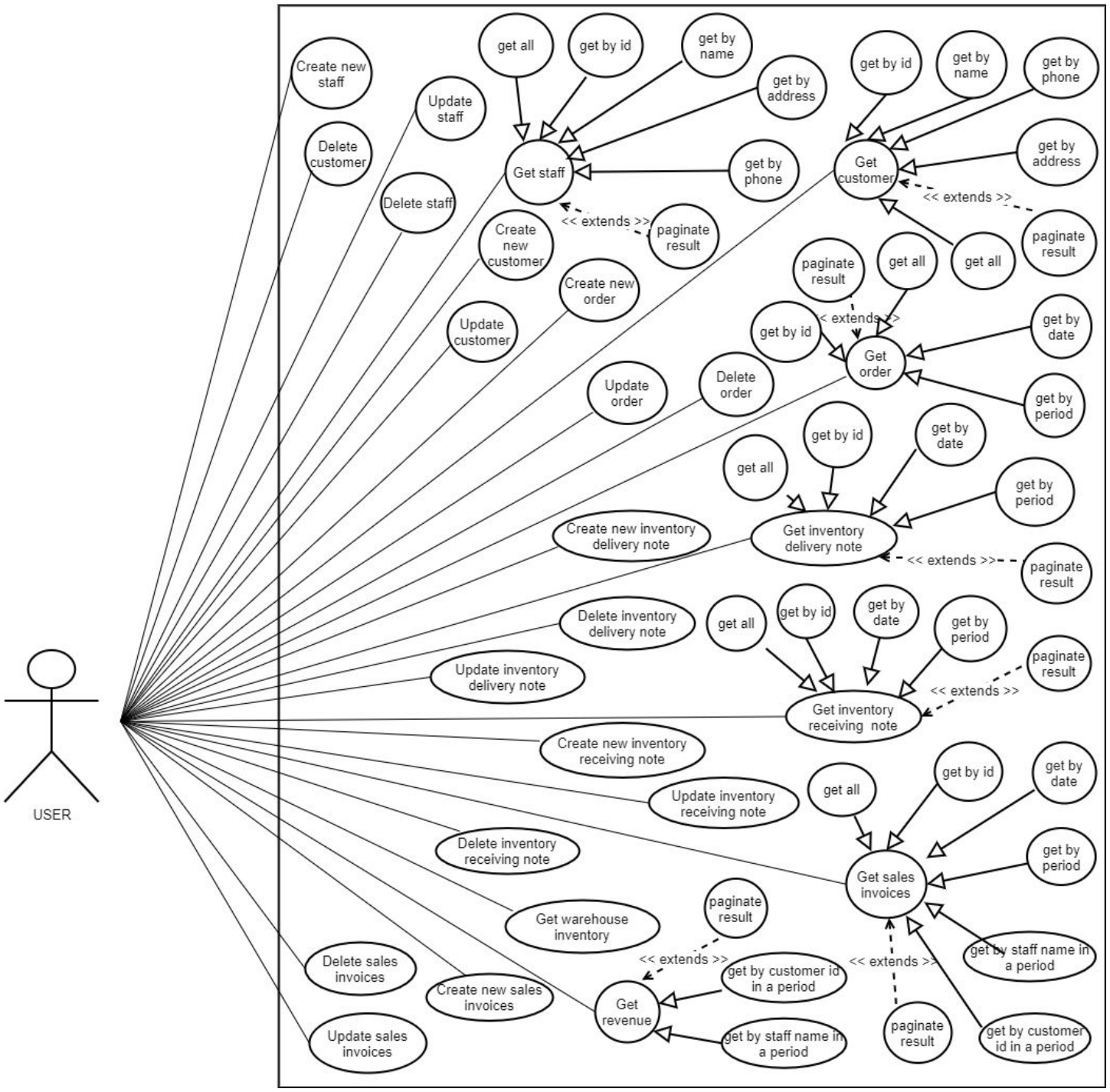
In order to manage the mentioned entities, we have some data requirements that needed to be managed and maintained for each entity:

- Staff: id, name, address, phone, email.
- Customer: id, name, address, phone, fax, email, contact person.
- Provider: id, name, address, phone, fax, email, contact person
- Product: id, name, model, brand, company, description, category, and selling price
- Category: id and name
- Order: id, date, staff who makes the order, and provider
- Order detail: product, quantity and price
- Inventory receiving note: id, date, staff who makes the order
- Inventory receiving note detail: product, quantity
- Inventory delivery note: id, date, staff who makes the order
- Inventory delivery note detail: product, quantity
- Sales invoice: id, date, name of sales staff, customer, total value
- Sales invoice detail: product, quantity, price

The system implements different technologies and provides RESTful Web services to manipulate the above entities. For this system, it is assumed that all provider, product and category data will be provided. The system only provides features to directly manipulate the following entity: staff, customer, order, inventory receiving note, inventory delivery note, sales invoice.

The system is helpful to companies in terms of storing and tracking business processes and activities, managing internal and external human resources as well as its warehouse and products. The system allows companies to easily add, edit, delete all the entities' information. The information can be filtered by different criterias. Furthermore, the system provides some essential statistical APIs for the companies to track their revenue and warehouse products. While working with the system, some data automated generation work will be done in order to give a more efficient and optimal operation.

3. Use case diagram



Full list of the system's main functions including:

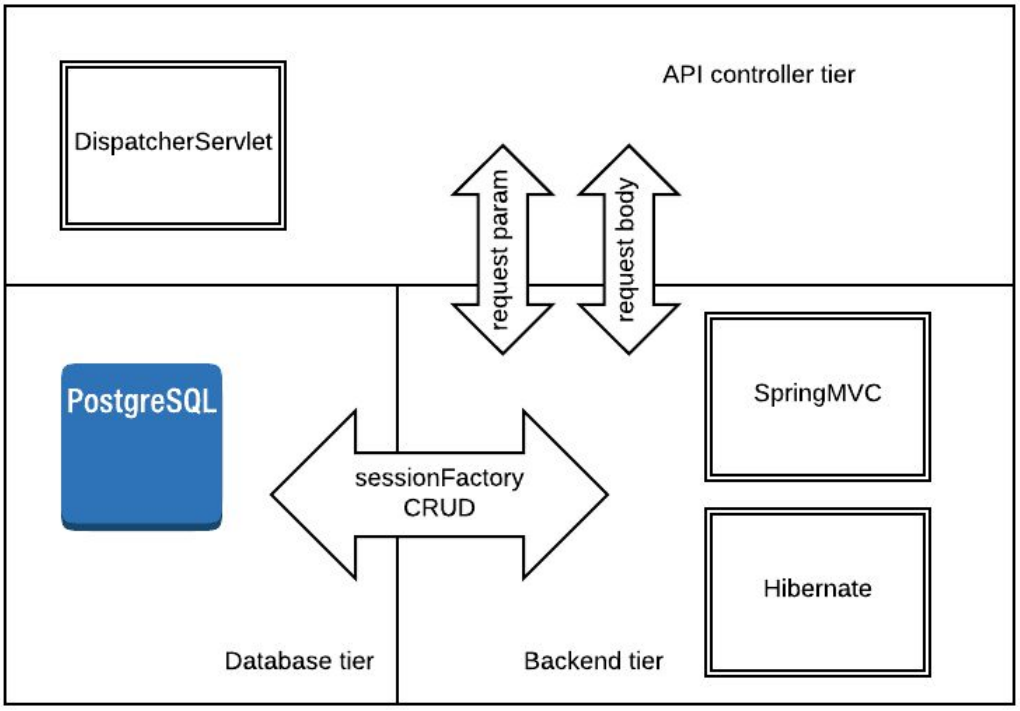
- Create, delete, update, get all data: customer, staff, inventory delivery note, inventory receiving note, sales invoice, order
- Get and filter by id, name, phone, address: staff, customer
- Get and filter by id, date, period: inventory delivery note, inventory receiving note, sales invoice, order
- Get and filter by sales staff name or customer id in a period: sales invoice
- Get warehouse inventory
- Get revenue filter by customer id or sales staff name in a period
- All get data functions that return a list of records providing result pagination.
- Besides the main functions, product and provider records can be created and fetched by id for testing purposes.

4. Technology use

During the process of developing the system, multiple technologies have been applied.

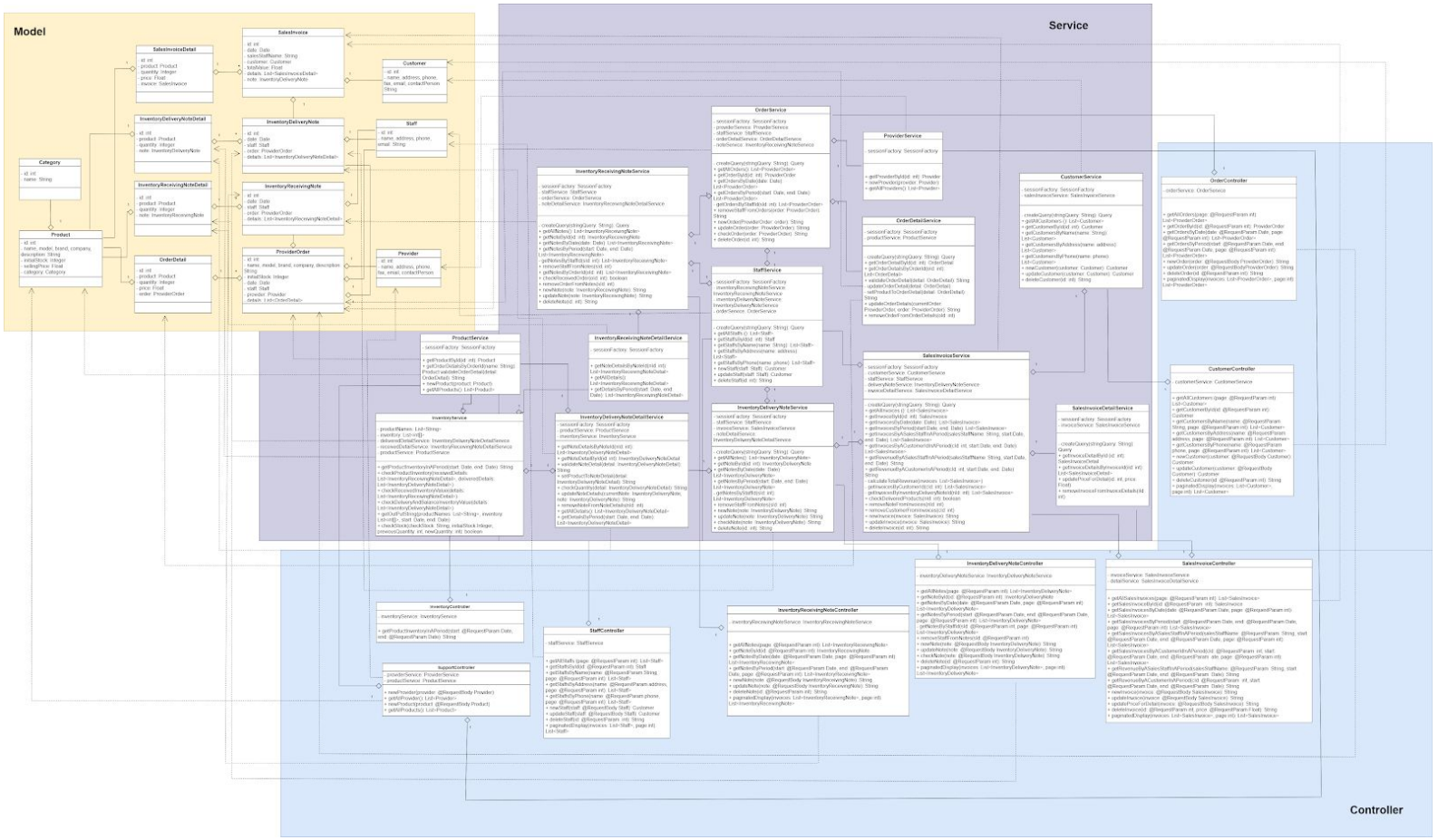
- Databases: postgresql
- Programming languages: Java and HQL
- Framework: Hibernate, SpringMVC, Spring
- Libraries: spring-context, spring-webmvc, spring-tx, spring-orm, hibernate-core, postgresql, persistence-api, javax.servlet-api, jackson-databind, annotations-java5, commons-lang3, junit
- JDK: 11.0.2
- Tools: Maven
- IDE: IntelliJ

5. Architecture



The project is developed with Java language using IntelliJ IDE. Build automation tool Maven and maven jetty plugin is used to run the system. Spring Web MVC DispatcherServlet is used to handle all HTTP requests and return responses. The system is a backend component, so we focus on using SpringMVC, Hibernate framework to process the request from the API tier and use Hibernate sessionFactory and HQL to work with the database tier and finally return some responses when appropriate. For the relational database management system, we use PostgreSQL for this project. Essential libraries are used to achieve the desired work including Junit to test all the APIs.

6. Class diagram



Designing the structure of our application, we have come up with a class diagram including 3 packages (model, service and controller) with the total of 34 classes.

The model package provides classes that are database entities with all required data fields, getters, setters and data automated generation methods. All the detail classes and its parent classes have bidirectional one to many aggregation relationships. Product and Category, details and Product, SalesInvoice and Customer, Order and Provider, Order, InventoryDeliveryNote, InventoryReceivingNote and Staff has an aggregation one to one relationship as the former class(es) has objects of later class. Besides most of the classes in model package have dependency relationships with classes in controller and service package where a reference to these classes in model package are passed as a method argument to classes in controller and service package.

The service package provides classes that work with hibernate SessionFactory to create, save, update and delete entities' records. Lots of checking activities are performed in order to catch and return appropriate errors to the users and to maintain some special constraint for data integrity. That's why there is a lot of one to one aggregation relationships between the services.

Finally is the controller classes. These classes provide APIs to perform CRUD on desired classes. They have one to one aggregation relation with the respective services and provide pagination as well. Some update methods receive model classes' objects as arguments after being converted from json, hence, the classes have a dependency relationship with model classes for this.

7. Implementation result

Together with the initial business requirements, some extra constraints have been included to optimize the automation process and data integrity assurance.

Firstly, to improve search features, search fields are not case sensitive for string information and for search by phone, the system ignores spaces so they don't affect the result. Dates follow the format as "YYYY-MM-DD". The pagination feature allows users to view the results by page or view all results in one page with page number as "-1". When displaying the records, only selected fields are displayed for better reading.

Secondly, whenever there is an attempt to delete a non-existent record, there is an appropriate error message returned. When successfully delete an record that has reference in another entity's records, the references are set to null and that entity's records are free to assign new value for that field.

Thirdly, for creating records, different constraints are applied to each entity:

- Product: Name is unique, initial stock value is added in order to check current stock for delivery note. If no initial stock value is assigned, then it is counted as 0 stock.
- Customer and staff: Name is compulsory, name and address are saved as lower case and spaces in phone are ignored for better search features.
- Order: Provider and details are compulsory. For order detail, product and quantity are compulsory. Quantity has to be greater than 0 and the product's selling price has to have value. Otherwise, the product is counted as not-for-sell product and can not be ordered. Same with price, if no value is assigned for price, price's value is auto transferred from product's selling price.
- Inventory receiving note: Order is compulsory. One order can be assigned to one note only. And all details of receiving notes are transferred from order's details including product, quantity and also staff in case no new staff is assigned when creating notes.
- Inventory delivery note: Staff and details are compulsory. For inventory delivery note details, product and quantity are compulsory. One product appears in one detail for each note only. And in order to successfully create a delivery note, the available stock (initial stock + received inventory - delivered inventory) should be sufficient for the intended quantity.
- Sales invoice: Customer and delivery note are compulsory. One delivery note can be assigned to one sales invoice only. And all details of the sales invoice are transferred from delivery note's details including product, quantity, price, total value and also sales staff name in case no new staff is assigned when creating sales invoice.

For all the records that have a date, if there is no value assigned for date, the current date at the time a record is created is auto generated. All details are counted as new details and given auto generated ids.

And lastly, for updating records, different constraints are also applied to each entity:

- All fields that are unchanged are kept as before.
- Order: Quantity has to be equal or greater than 0. Already received orders can not be updated. Order detail's product can be changed as long as the product is not-for-sell product with an unavailable selling price. When a new product is set to the detail, the price is also regenerated.
- Inventory receiving note: When a new order is assigned to a receiving note, details and relevant data are regenerated as long as the new order does not belong to any other note.
- Inventory delivery note: Already delivered note that has a sales invoice can not be updated. Details can not change or have the same products. Quantity can be set to equal or greater than 0 as long as current stock is sufficient.
- Sales invoice: When a new delivery note is assigned to a sales invoice, details and relevant data are regenerated as long as the new delivery note does not belong to any other invoice.

For all details, unchanged details are kept, changed details are updated and new details are created. But one detail can belong to one owner only so detail can not be changed between owners except when the owner record is deleted and the detail doesn't belong to any owner any more. Only the details can be updated via the owner record. All other records are not affected when some records having relationships with them are being updated.

With all the above constraints, error handling operations are developed to catch all inappropriate behaviours violating the constraints and return suitable error messages to users. Only new details can be created via the owner records. All the other objects have to be created before performing some activity with them. Whenever an non-existent object is assigned to another object, there should be a suitable error message as well.

At the end, a complete Java maven project using SpringMVC and Hibernate, working with postgresql, is developed and tested. The application provides qualified APIs to perform CRUD on desired entities following the above constraints and fulfill all the listed use cases.

8. Testing

8.1. Objectives

- Functions of the system to be tested:
    - Create, delete, update, get all data: customer, staff, inventory delivery note, inventory receiving note, sales invoice, order
    - Get and filter by id, name, phone, address: staff, customer
    - Get and filter by id, date, period: inventory delivery note, inventory receiving note, sales invoice, order
    - Get and filter by sales staff name or customer id in a period: sales invoice
    - Get warehouse inventory
    - Get revenue filter by customer id or sales staff name in a period
    - All get data functions that return a list of records providing result pagination.

- Create and get product and provider
- Error handling

By fulfilling the above functions, the testing plan aims to ensure that the application meets the desired requirements for an automation business process.

8.2. Testing strategy

- Test methods: Specification-based approach is applied to test the application to meet all the requirements. Black box testing method is applied to test all APIs. In order to cover all positive and negative test cases, Decision-table and branch coverage techniques are applied. For example, when updating an order, we have to check product selling price, order detail id, quantity,... By applying the above mentioned techniques and methods, 100% of APIs have been tested with 100% code coverage.

8.3. Test cases and results

ID	API to be tested	Test cases	Assertions
1	http:/localhost:8080/customers/create	testNewCustomerWithNameAndAddressShouldBeLowerCaseAndNoSpaceInPhone testNewCustomerWithoutName	5
2	http:/localhost:8080/customers/update	testUpdateCustomerWithoutIdOrWithNonExistentCustomer testUpdateCustomerWithNameAndAddressShouldBeLowerCaseAndNoSpaceInPhoneNonUpdatedFieldStaysStill	6
3	http:/localhost:8080/customers/delete/id	testDeleteExistentAndNonExistentCustomer	2
4	http:/localhost:8080/customers/read/all	testGetAllCustomersWithoutPaging testGetCustomersWithMoreThanOnePage	4
5	http:/localhost:8080/customers/read/id	testGetCustomerById	2
6	http:/localhost:8080/customers/read/name	testGetCustomerByName	3
7	http:/localhost:8080/customers/read/address	testGetCustomerByAddress	3
8	http:/localhost:8080/customers/read/phone	testGetCustomerByPhone	3
9	http:/localhost:8080/staffs/create	testNewStaffWithNameAndAddressShouldBeLowerCaseAndNoSpaceInPhone testNewStaffWithoutName	5
10	http:/localhost:8080/staffs/update	testUpdateStaffWithoutIdOrWithNonExistentStaff testUpdateStaffWithNameAndAddressShouldBeLowerCa	6

		seAndNoSpaceInPhoneNonUpdatedFieldsStaysStill	
11	http:/localhost:8080/staffs/delete/id	testDeleteExistentAndNonExistentStaff	2
12	http:/localhost:8080/staffs/read/all	testGetAllStaffsWithoutPaging testGetAllStaffsWithMoreThanOnePage	4
13	http:/localhost:8080/staffs/read/id	testGetStaffById	2
14	http:/localhost:8080/staffs/read/name	testGetStaffByName	3
15	http:/localhost:8080/staffs/read/address	testGetStaffByAddress	3
16	http:/localhost:8080/staffs/read/phone	testGetStaffByPhone	3
17	http:/localhost:8080/orders/create	testNewOrderWithoutProvider testNewOrderWithoutDetails testNewOrderWithoutQuantity testNewOrderWithoutProduct testNewOrderNotForSellProduct testNewOrderAutoGenerateDateAndPrice	7
18	http:/localhost:8080/orders/update	testUpdateOrderWithToHaveDetailsWithTheSameId testUpdateOrderWithDetailBelongedToAnotherOrder testUpdateOrderWithNonExistentOrder testUpdateOrderWithAlreadyReceivedOrder testUpdateOrderWithNonExistentStaff testUpdateOrderWithNonExistentProvider testUpdateOrderWithNotForSellProduct testUpdateOrderWithNonExistentProduct testUpdateOrderWithInvalidQuantity testUpdateOrderNonUpdatedFieldsStaysStillNewProductGeneratePrice	13
19	http:/localhost:8080/orders/delete/id	testDeleteExistentAndNonExistentOrder	1
20	http:/localhost:8080/orders/read/all	testGetAllOrdersWithoutPaging testGetOrdersWithMoreThanOnePage	4
21	http:/localhost:8080/orders/read/id	testGetOrderById	3
22	http:/localhost:8080/orders/read/period	testGetOrdersByPeriod	4
23	http:/localhost:8080/orders/read/date	testGetOrdersByDate	3
24	http:/localhost:8080/inventoryReceivingNotes/create	testNewNoteWithNonExistentStaff testNewNoteWithNonExistentOrder	7

		testNewNoteWithOrderBelongedToAnotherNote testNewNoteAutoGenerateData	
25	http:/localhost:8080/inventoryReceivingNotes/update	testUpdateNoteWithNonExistentNote testUpdateNoteWithNonExistentStaff testUpdateNoteWithOrderBelongedToAnotherNote testUpdateNoteWithNonExistentOrder testUpdateNoteAutoGenerateDataAndNonUpdatedFieldsStaysStill	9
26	http:/localhost:8080/inventoryReceivingNotes/delete/id	testDeleteExistentAndNonExistentNote	1
27	http:/localhost:8080/inventoryReceivingNotes/read/all	testGetAllNotesWithoutPaging testGetNotesWithMoreThanOnePage	4
28	http:/localhost:8080/inventoryReceivingNotes/read/id	testGetNoteById	2
29	http:/localhost:8080/inventoryReceivingNotes/read/period	testGetNotesByPeriod	4
30	http:/localhost:8080/inventoryReceivingNotes/read/date	testGetNotesByDate	3
31	http:/localhost:8080/inventoryDeliveryNotes/create	testNewNoteWithoutStaff testNewNoteWithoutDetails testNewNoteWithoutQuantity testNewNoteWithoutProduct testNewNoteWithShortageOfProduct testNewNoteWithDetailsHavingTheSameProduct testNewNoteAutoGenerateDate	8
32	http:/localhost:8080/inventoryDeliveryNotes/update	testUpdateNoteWithDetailBelongedToAnotherNote testUpdateNoteWithNonExistentNote testUpdateNoteWithAlreadyDeliveredNote testUpdateNoteWithNonExistentStaff testUpdateNoteWithDetailsHavingTheSameProduct testUpdateNoteWithShortageOfProduct testUpdateNoteWithDetailChangingProduct testUpdateNoteWithInvalidQuantity testUpdateNoteNonUpdatedFieldsStaysStill	11
33	http:/localhost:8080/inventoryDeliveryNotes/delete/id	testDeleteExistentAndNonExistentNote	1
34	http:/localhost:8080/inventoryDeliveryNotes/read/all	testGetAllNotesWithoutPaging testGetNotesWithMoreThanOnePage	4
35	http:/localhost:8080/inventoryDeliveryNotes/read/id	testGetNoteById	2

36	http:/localhost:8080/inventoryDeliveryNotes/read/period	testGetNotesByPeriod	4
37	http:/localhost:8080/inventoryDeliveryNotes/read/date	testGetNotesByDate	3
38	http:/localhost:8080/salesInvoices/create	testNewInvoiceWithNonExistentCustomer testNewInvoiceWithNonExistentStaffName testNewInvoiceWithNonExistentDeliveryNote testNewInvoiceWithDeliveryNoteBelongedToAnotherInvoice testNewInvoiceAutoGenerateData	8
39	http:/localhost:8080/salesInvoices/update	testUpdateInvoiceWithNonExistentInvoice testUpdateInvoiceWithNonExistentCustomer testUpdateInvoiceWithNonExistentStaffName testUpdateInvoiceWithDeliveryNoteBelongedToAnotherInvoice testUpdateInvoiceAutoGenerateDataAndNonUpdatedFieldsStaysStill	10
40	http:/localhost:8080/salesInvoices/update/detail/price	testUpdatePriceForDetail testUpdatePriceForNonExistentDetail	1
41	http:/localhost:8080/salesInvoices/delete/id	testDeleteExistentAndNonExistentInvoice	1
42	http:/localhost:8080/salesInvoices/read/all	testGetAllInvoicesWithoutPaging testGetInvoicesWithMoreThanOnePage	4
43	http:/localhost:8080/salesInvoices/read/id	testGetInvoiceById	2
44	http:/localhost:8080/salesInvoices/read/date	testGetInvoicesByDate	3
45	http:/localhost:8080/salesInvoices/read/period	testGetInvoicesAndRevenueByPeriod	3
46	http:/localhost:8080/salesInvoices/read/staff-period		2
47	http:/localhost:8080/salesInvoices/read/customer-period		2
48	http:/localhost:8080/salesInvoices/revenue/read/staff-period		1
49	http:/localhost:8080/salesInvoices/revenue/read/customer-period		1
50	http:/localhost:8080/inventory/read/period	testGetProductInventoryInAPeriod	1

51	http:/localhost:8080/providers/create	testProvider	1
52	http:/localhost:8080/providers/read/all/?page=-1		
53	http:/localhost:8080/products/create	testProduct	1
54	http:/localhost:8080/products/read/all/?page=-1		
Test result: PASSED 100%			

9. Limitation, known bugs

The system manages to fulfil all the requirements. However, the time given is not enough to optimize the system performance to make it more flexible. For warehouse inventory checking, only received, delivered inventory and balance values are presented, it does not reflect the remaining inventory in the warehouse at that period. SQL injection hasn't been tested. There is no security feature for confidential information as well.