Clean Coding and Concurrent Programming

COMP63038

Assignment 2

Gampala Waduge Shemeera Siyumi Fonseka

CB013452

Submitted to

The School of Computing

Of

Asia Pacific Institute of Information Technology

Sri Lanka

Table of Contents

Abstract

SYOS Billing System is a web based solution that is designed to automate a traditional billing process and enhance the efficiency through a structured modern software architecture and it enables the cashiers to process sales using unique item codes and quantities, automatically calculating totals, managing stock reductions and generating bills. The system supports both instore and online sales maintaining separate inventories and generating detailed reports for each. SYOS follows a clean architectural design with a multi-tier client server model containing presentation, application and data layers. It uses Model View Controller (MVC) pattern by using React for the frontend, Java Servlets for the controllers and MySQL for persistent data storage. And also the use of SOLID principles and key design patterns ensures modularity, maintainability and scalability of the system. Concurrency is managed effectively on both server and client sides and the server uses Java EE's built in multithreading and transaction handling to process simultaneous client requests securely while the react based frontend utilizes asynchronous operations and state management to maintain a responsive user interface. Even though it is a strong architectural foundation, the system's design highlights the need for careful boundary management, scalability planning and continuous optimization to support long term growth and maintain system robustness under high concurrency scenarios.

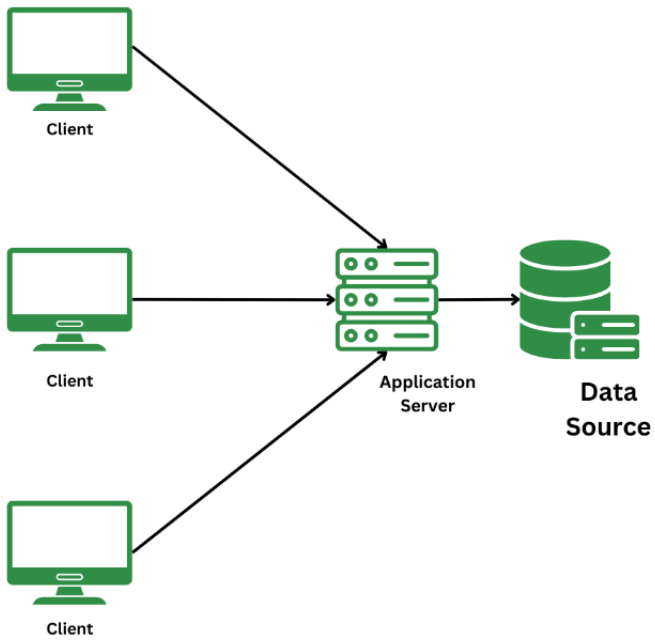
Chapter 1 - Introduction

In today's fast paced retail environment, grocery stores require efficient and accurate billing systems to streamline sales and inventory management. The SYOS billing system is developed to address these needs by replacing manual billing method with a modern automated solution. This web based solution enables cashiers to process transactions using unique item codes and quantities while automatically updating inventory and generating bills. And also it is designed to support both instore and online operations by ensuring seamless inventory tracking and sales reporting across multiple mediums and its structured architecture and the use of modern technologies make it a scalable and maintainable solution for grocery retailers aiming to enhance customer service and operational efficiency.

Chapter 2 – Clean Architecture

2.1 Multi-tier client-server Architecture

SYOS automated billing system follows a multi-tier client-server architecture specifically a three-tier architecture where the system is divided into 3 main layers such as presentation, application and data. The presentation layer is the client side web interface running in the browser where users interact with the system and this layer sends requests to the server to perform actions like saving a bill or registering a user. The application layer is the middle tier which acts as the brain of the system which receives requests from the client, applies business rules, process data and coordinates operations such as generating unique bill numbers or calculating totals and this layer ensures that the logic and processes of the system are centralized and managed efficiently. The data layer is responsible for storing and retrieving information in the database and it handles tasks like saving bill information, accessing customer records and updating inventory. By using this type of architecture in the SYOS system, improves maintainability, scalability and security and also each layer can be developed, tested and updated independently by making the system more flexible and robust.



2.2 MVC Architecture

The SYOS Billing system is built on a layered architecture with a strict separation of concerns. The Model Layer using the entity classes like Batch, Bill, Item, User and Order take care of data management and business logic. The Model Layer interacts with the MySQL database via DAOs that encapsulate the SQL calls. The View Layer is in React to provide a dynamic user interface which handles user interactions and performs updates based on responses from the backend. The Controller Layer uses Java Servlets to handle HTTP requests from the user interface that validate inputs and call the model methods. Service Layer contains core business logic for order processing and stock management and acts as a bridge between the controller layer and DAO layer.

2.3 Allocation of Clean Architecture components

The allocation of clean architecture components within the SYOS automated billing system's multi-tier client-server structure identifies a well considered effort to balance modularity and practical implementation but some critical observations can be seen when examining the approach in depth. While the clear separation between the presentation layer, controller layer, service layer and data access objects aligns well with the clean architecture's principles, some boundaries can be seen between theses layers by risking unintended tight coupling. For example, the controller layer's dual role of parsing input and handling HTTP responses sometimes can lead to the mixing of concerns by violating the single responsibility principle if not carefully managed. Although the service layer effectively centralizes the business logic, the system's reliance on concrete DAO implementations may reduce flexibility in swapping out data sources or mocking dependencies for testing.

Integrating frontend react components with backend java servlets using JSON and Gson adapters presents a practical yet tightly coupled solution and this approach allows efficient data exchange but it introduces a strong dependency on specific data formats. Both frontend and the backend must agree on the structure and naming of data fields which means that any changes in the data model on one side require corresponding updates on the other. Without clear API versioning or abstraction layers such as defined interfaces or schema management evolving the system becomes

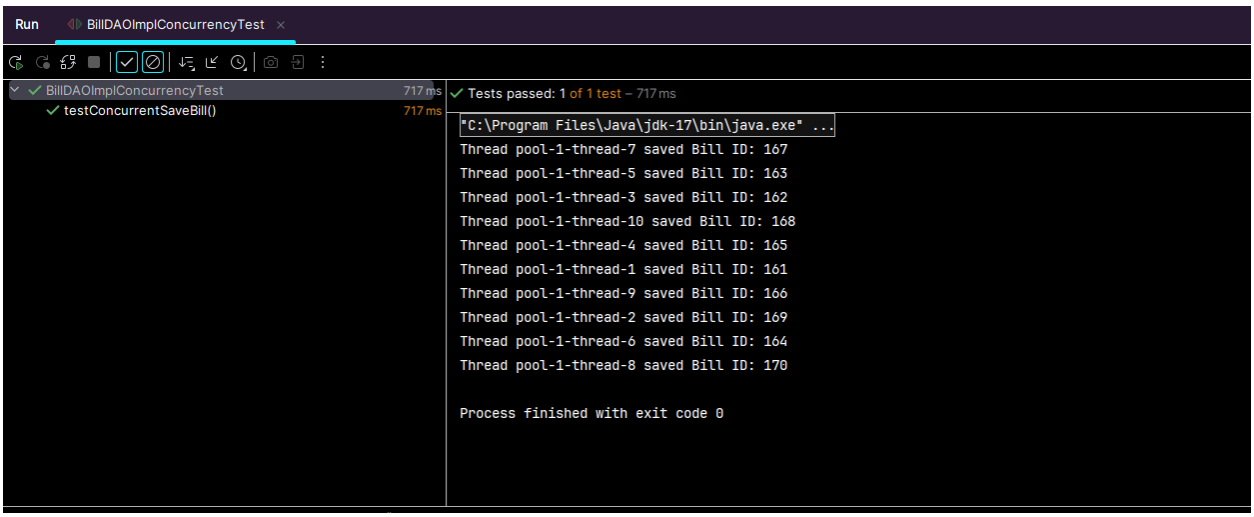
increasingly difficult and the absence of these things can lead to fragile communication, reduce flexibility and scalability as the application grows or requirements change.

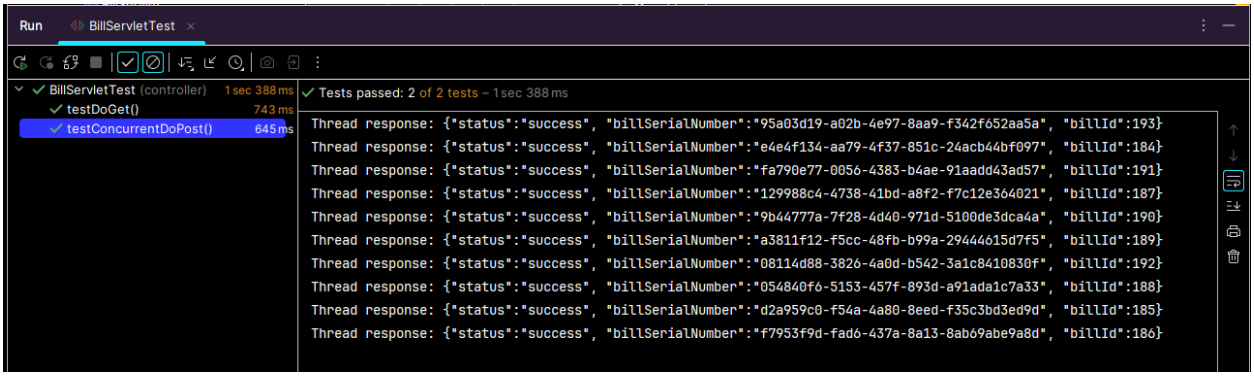
The use of several design patterns like DAO, Factory and Strategy, the overall complexity of the system may increase which could pose a learning curve for new developers and affect maintainability if the documentation or coding standards are not properly followed. Lastly, while SOLID principles are mostly respected, the system's scalability and adaptability depend heavily on ongoing discipline in maintaining strict layering and avoiding the leakage of infrastructure concerns into business logic or UI code. In summary, the allocation of clean architecture components in SYOS demonstrates a strong foundational design with many benefits but it also requires a proper management of layer boundaries, dependencies and complexity to fully realize the theoretical advantages of clean architecture in practice.

Chapter 3 - Concurrency

3.1 Server Side Concurrency

SYOS automated billing system has been implemented as a servlet based application that relies on the built in concurrency mechanisms of the Java EE servlet container such as Apache Tomcat to handle multiple simultaneous client requests efficiently through a thread per request model. When multiple clients send requests to save a bill, log in, register or insert a batch, the server uses a thread pool to process them concurrently by creating a separate thread invoking methods like doPost() or doGet() independently for each incoming request and this thread per request model ensures that one request does not block or interfere with another by maintaining high performance and responsiveness even under heavy load. Thread safety is maintained by using local variables and avoiding shared mutable state and additionally, each database operation creates a new PreparedStatement and obtains its own database connection by ensuring isolated transactions and also the underlying database management system handles concurrency at the data level through locking and transaction isolation by guaranteeing data integrity even when multiple inserts or updates happen concurrently. This combination of server side multi threading and database level concurrency control enables the application to safely and efficiently support multiple client interactions at the same time.





While the SYOS automated billing system depends on the Java EE servlet container's built in multithreading and database transaction mechanisms to handle concurrent client requests efficiently, this approach has several limitations. The use of thread per request and local variables promotes thread safety and prevents shared state issues but sometimes scalability can be challenged under very high loads due to thread pool exhaustion. And also by relying entirely database locking and transaction isolation can lead to bottlenecks or deadlocks if not carefully managed especially with complex or long running transactions. The system's performance and reliability also depend heavily on the servlet container's configuration and the database's concurrency capabilities. Overall while the server side follows standard best practices for concurrency, continuous monitoring and optimization are essential to ensure consistent performance and data integrity under real world high concurrency scenarios.

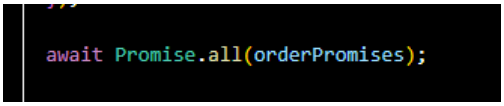
3.2 Client Side Concurrency

The client side application included in the SYOS system uses concurrency mainly by using state management and asynchronous operations to keep the UI as smooth and responsive as possible. And also it uses reactive state updates that schedules changes very efficiently by executing multiple state changes without locking the main thread preventing the UI from freezing up during rapid user interactions. For example, when a user adds items to the cart it executes the setItems([...items, newItems]) function and react schedules the state change without blocking the main thread so the user can smoothly interact with other modals and update input fields.

Asynchronous side effects are employed for the operations like data fetching or computations based on state changes run in the background so the UI can continue rendering and responding to inputs without waiting for each to complete synchronously by ensuring the application reacts immediately to user actions. As an example, operations such as saving a bill or order items use asynchronous fetch calls and this non blocking behavior ensures the UI to stay responsive which means the user can still scroll or interact with other components while the request is processed in the background.

```
try {
  // First, save the bill
  const billResponse = await fetch('http://localhost:8080/SYOS%20Backend/bill', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(billData)
  });

  if (!billResponse.ok) {
    throw new Error('Failed to save bill');
  }

  const billResult = await billResponse.json();
  const billSerialNumber = billResult.billSerialNumber;
  const billId = billResult.billId;

  // Then save each order item
  const orderPromises = billData.orders.map(order => {
    const orderData = new URLSearchParams();
    orderData.append('orderID', order.orderID);
    orderData.append('itemCode', order.itemCode);
    orderData.append('quantity', order.quantity);
    orderData.append('amount', order.amount);
    orderData.append('billSerialNumber', billSerialNumber);

    return fetch('http://localhost:8080/SYOS%20Backend/orders', {
      method: 'POST',
      headers: {
```

For longer running operations such as submitting orders or fetching data, asynchronous processing avoids locking up the interface by running these tasks independently and updating the UI only once results are ready. As an example, when the user clicks the “Checkout” button, both the bill and all related order items are submitted and this parallel execution ensures fast completion without freezing the UI since order submission won’t wait for one another sequentially.



Routing is also handled concurrently by enabling smooth page transitions and view updates without interrupting ongoing user activities. By coordinating theses mechanisms, the application achieves efficient concurrency, improving performance and user experience by allowing multiple processes to run and update the interface seamlessly.

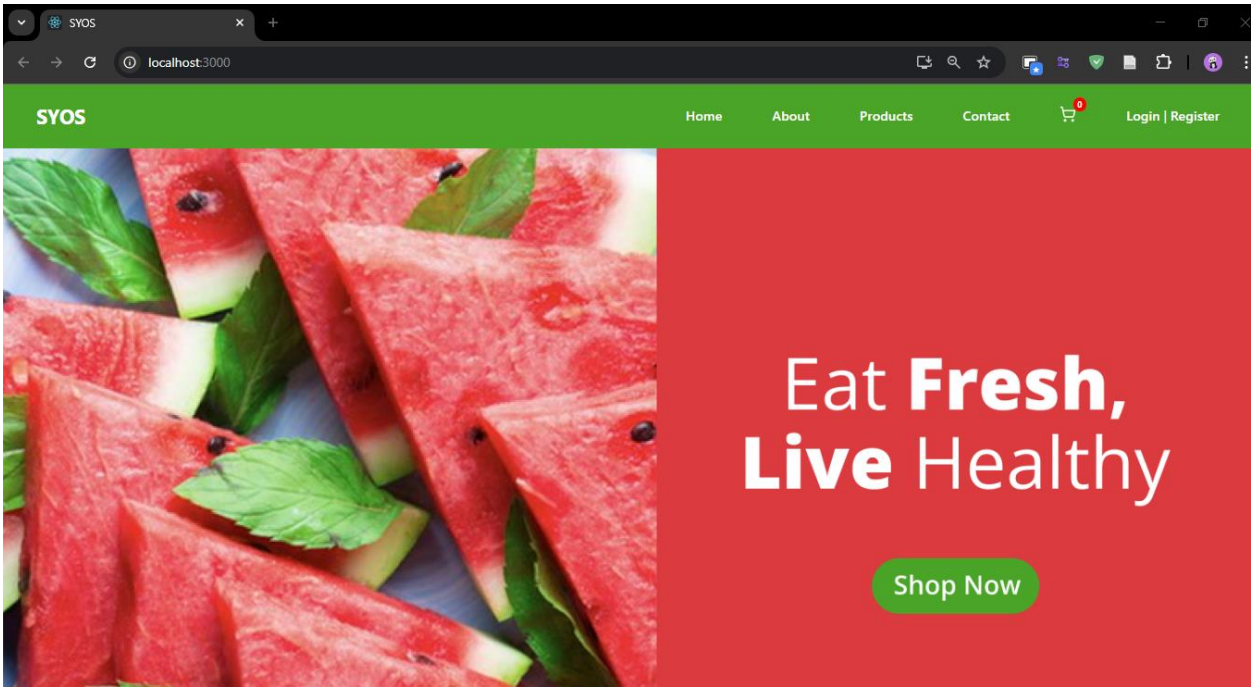
While the SYOS system effectively handle client side concurrency through React’s state management and asynchronous operations to ensure a smooth user experience, there are several architectural limitations that needed to pay attention. The system assumes that all asynchronous tasks such as adding items to a cart or submitting orders will execute reliably and in the expected order which may not hold true under real world conditions like poor network connectivity or high server load. This could lead to delayed UI updates, failed requests or data inconsistencies if not properly managed with error handling and fallback mechanisms. And also while the system supports multiple clients sending asynchronous requests simultaneously, it relies on client side state which can become stale in multi user scenarios without real time synchronization technologies like WebSockets. For example, one client may update a shared resource while another remains unaware which will lead to conflicts or incorrect data rendering. Therefore, while the SYOS shows client side responsiveness and concurrency handling, it would benefit from enhanced real time synchronization, better error recovery and robust backend concurrency control to maintain integrity and reliability in high traffic multi user environments.

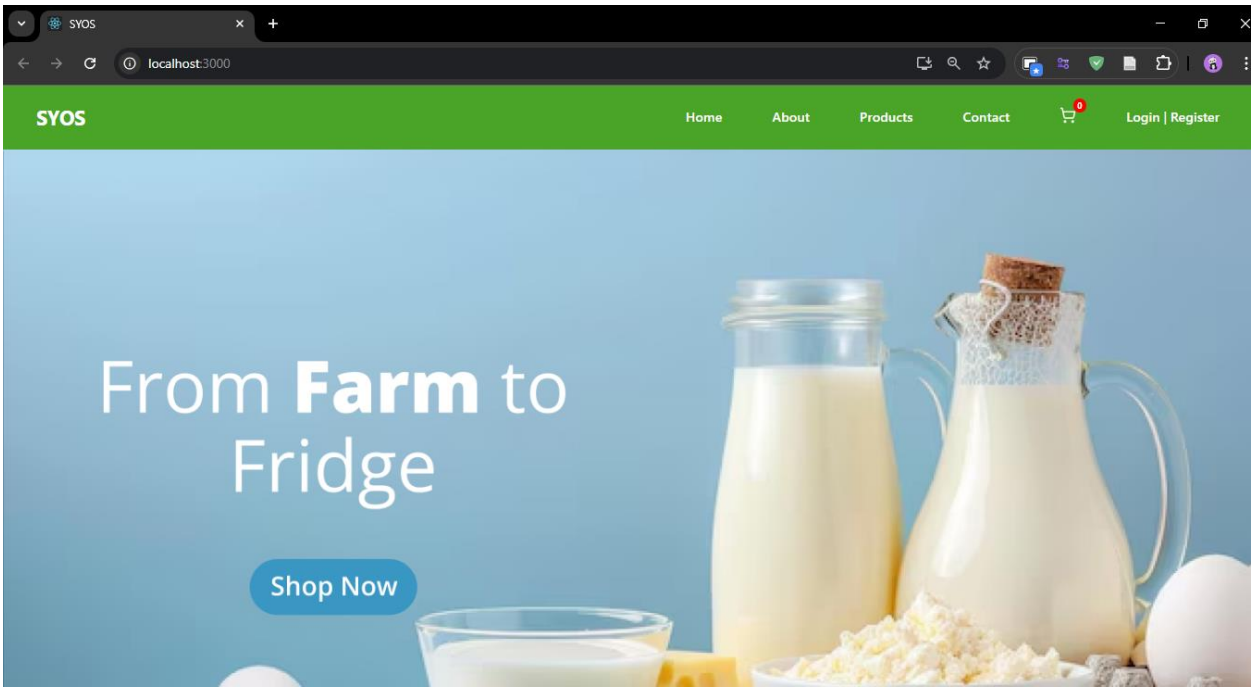
3.3 Summary

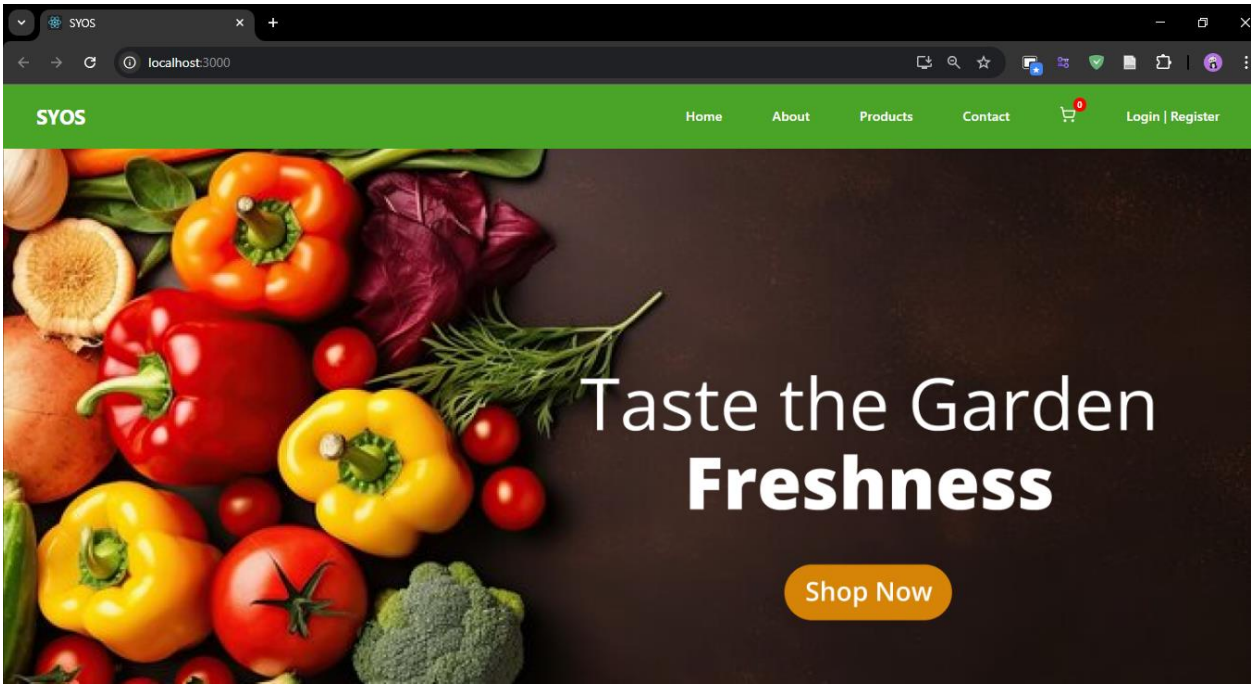
By using the SYOS system, more clients can simultaneously send asynchronous requests in rapid succession and the system is designed to handle such concurrency efficiently. On the client side the react application uses asynchronous operations (async/await, fetch) which allow multiple requests to be happen without blocking the user interface by ensuring a smooth and responsive experience even during heavy interaction. For example, while one client is submitting an order, another can also perform checkout operations without any delay or interference. On the backend, servlets handle each incoming requests from different clients in parallel and the MySQL database will manage these operations by maintaining data integrity and consistency even when multiple clients interact with the system simultaneously. This coordination between client side asynchronous behavior and server side multi threaded processing ensures that the SYOS system can handle concurrent actions from different users without performance issues or data conflicts.

Chapter 4 - GUI

Image Carousel

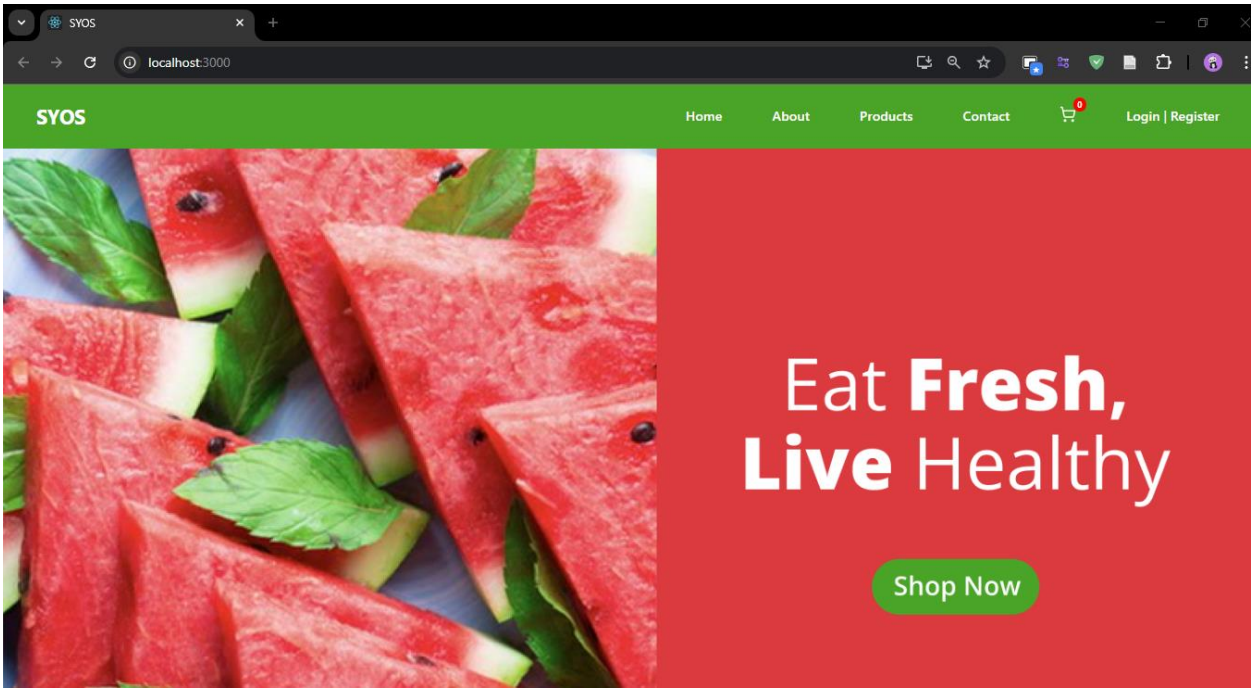




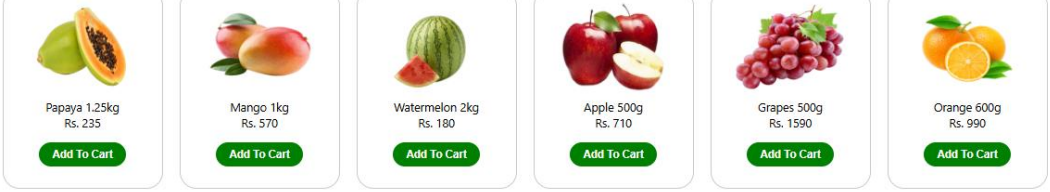


Navigation Bar

Home Page

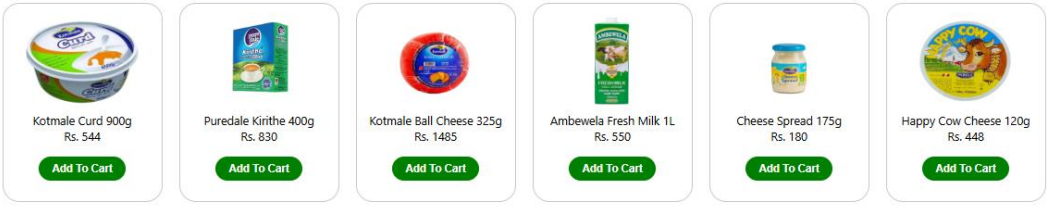


Fruits

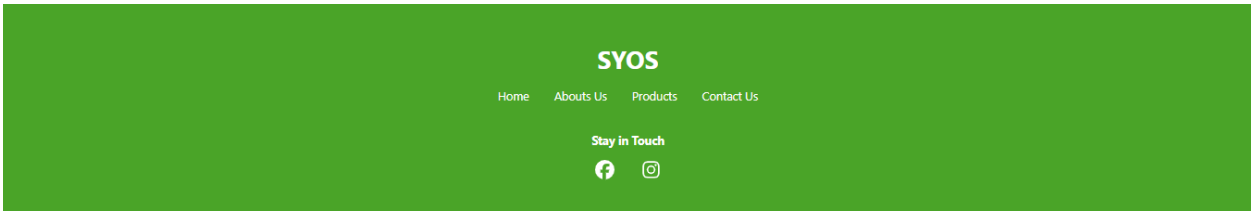


Vegetables

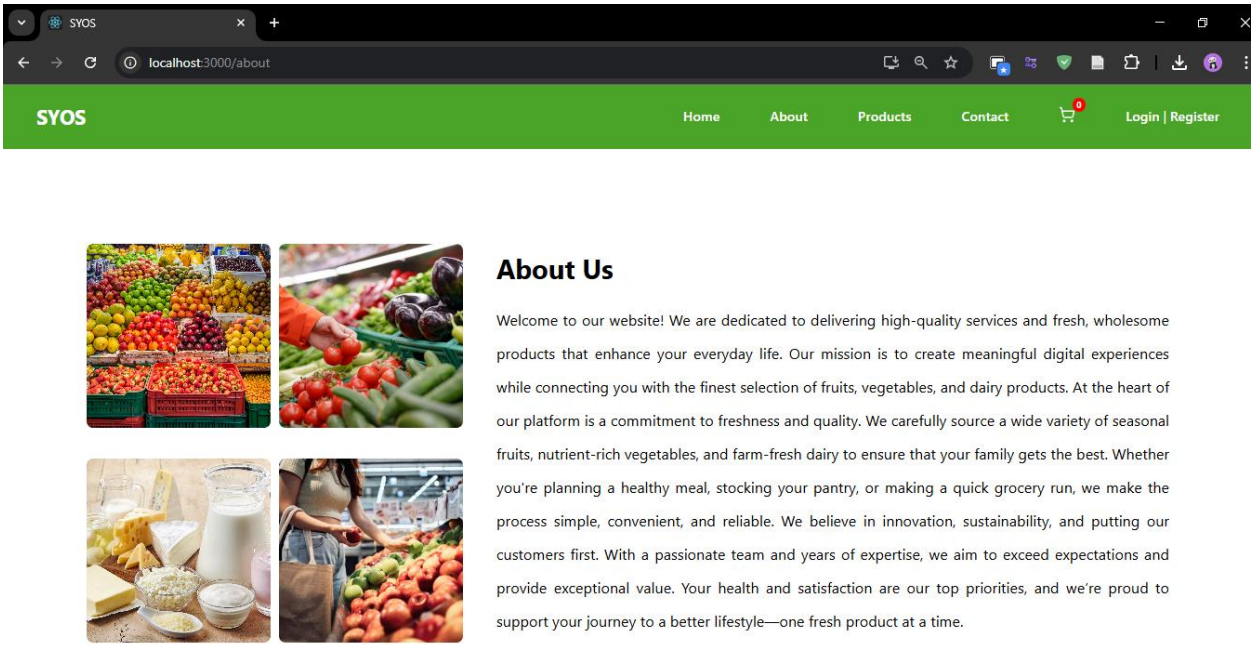
Dairy



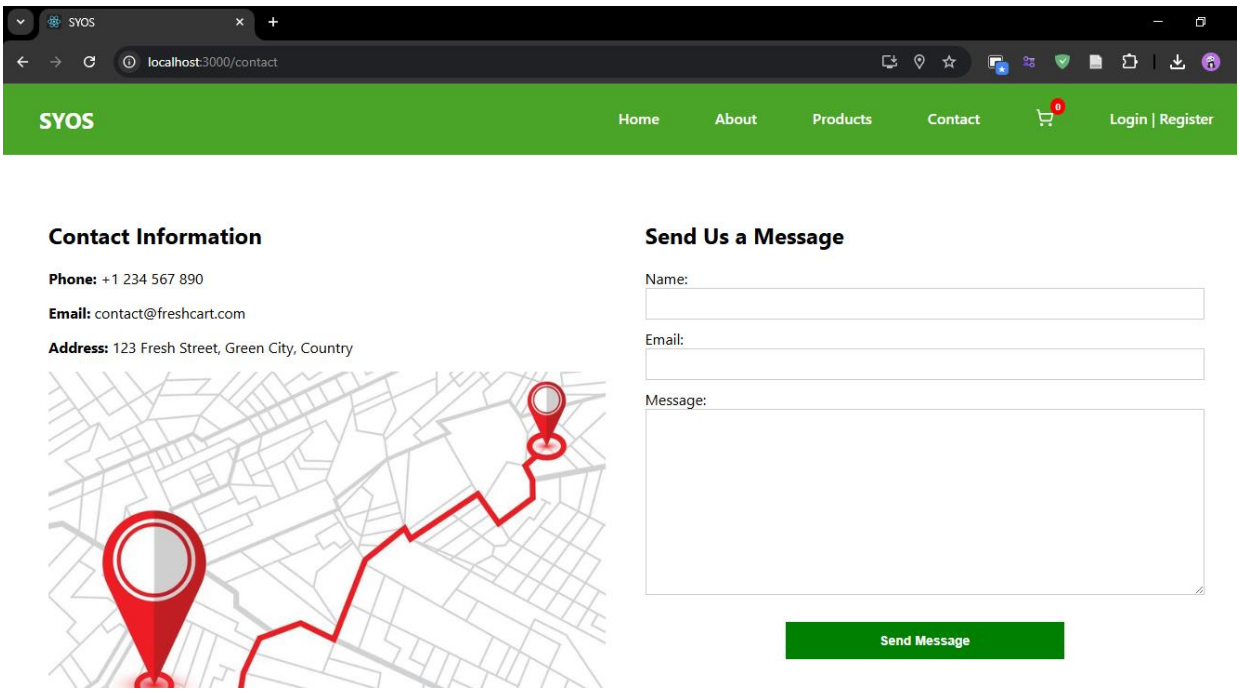
Footer



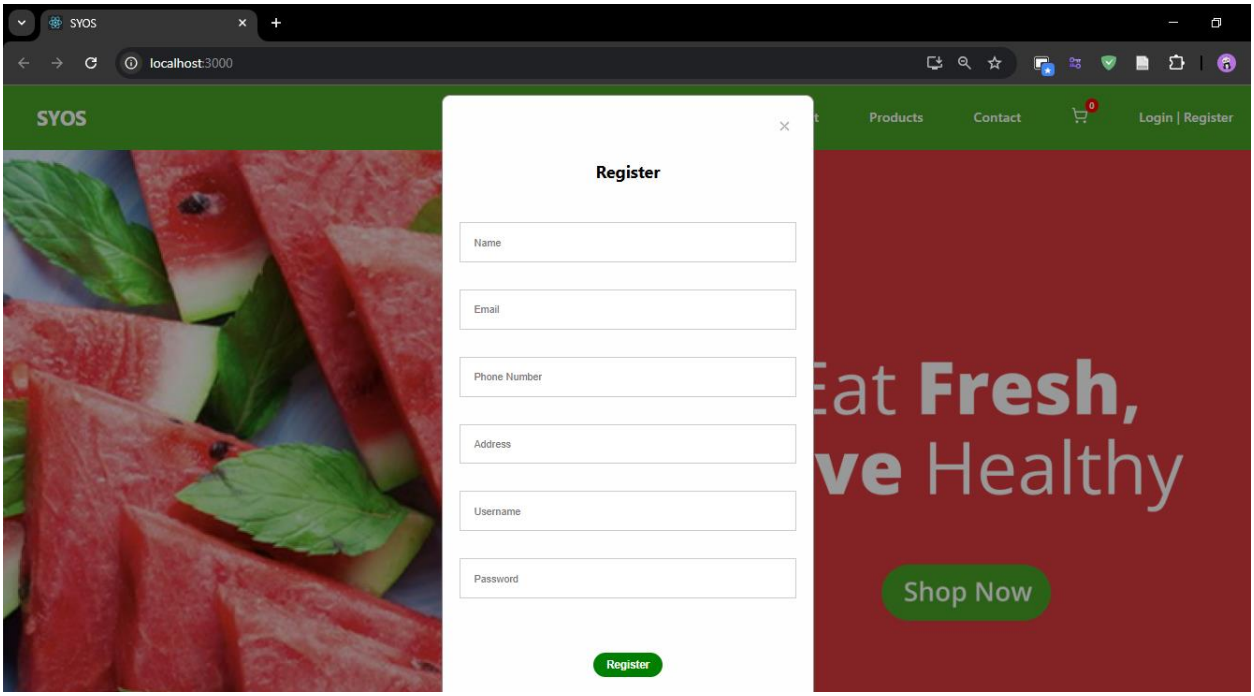
About Page



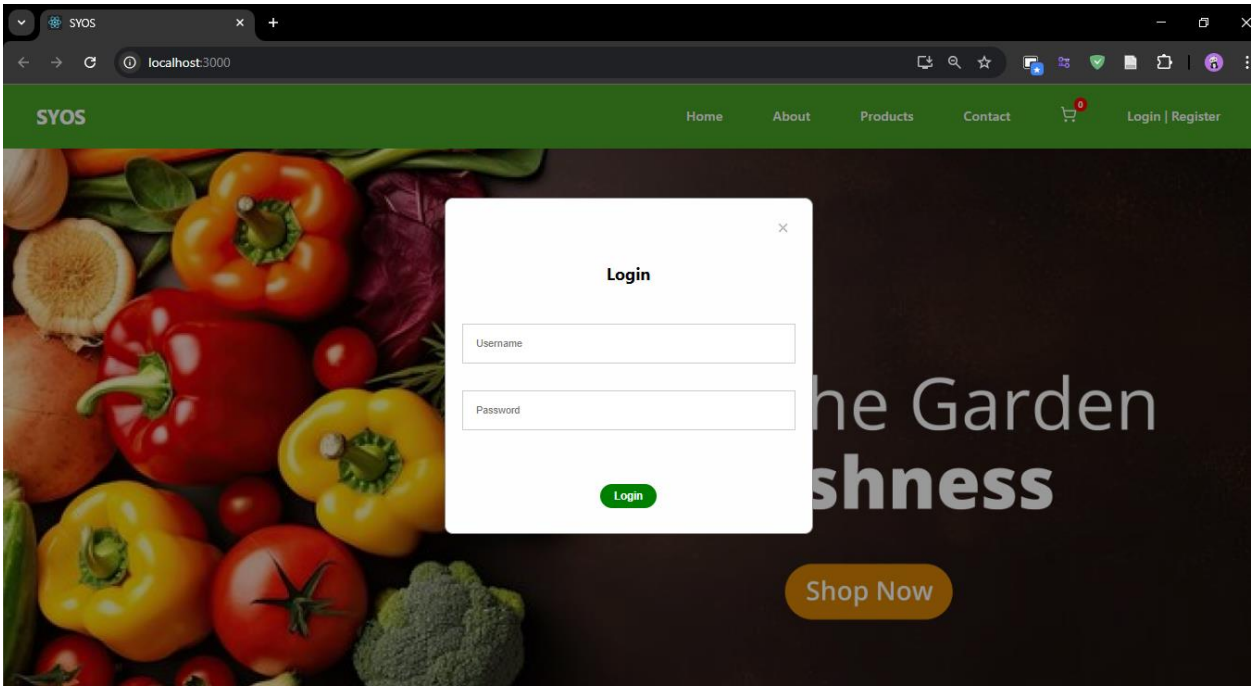
Contact Page



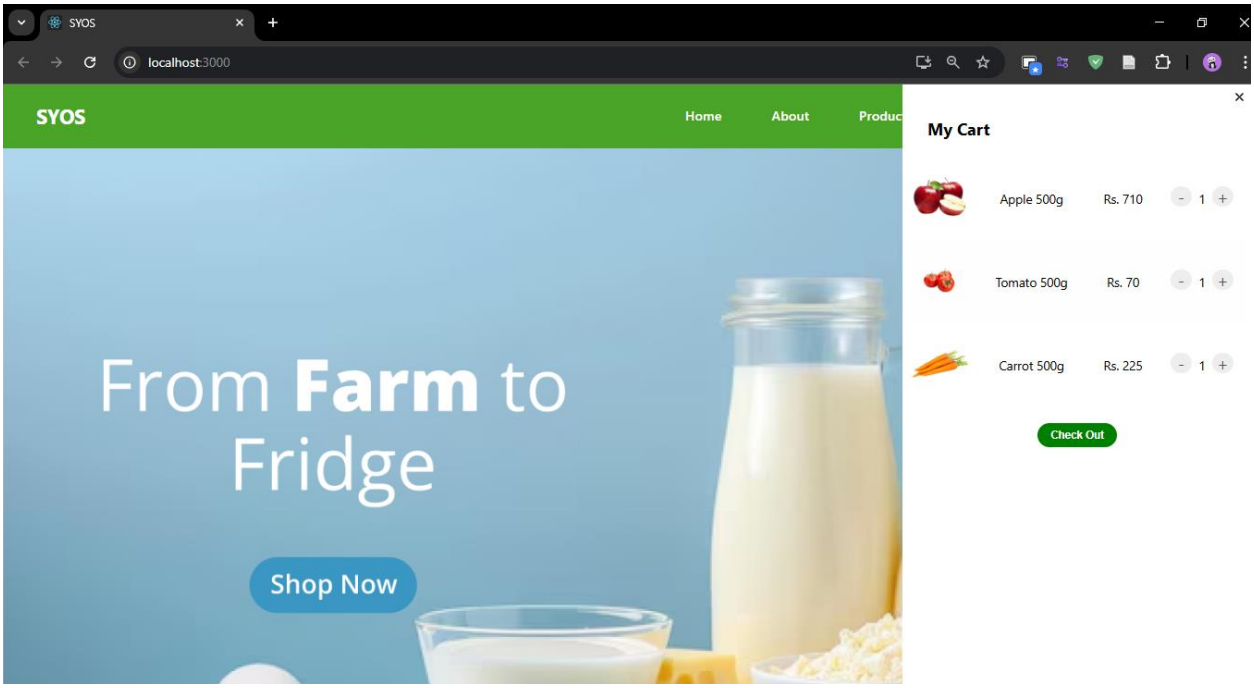
Register Modal



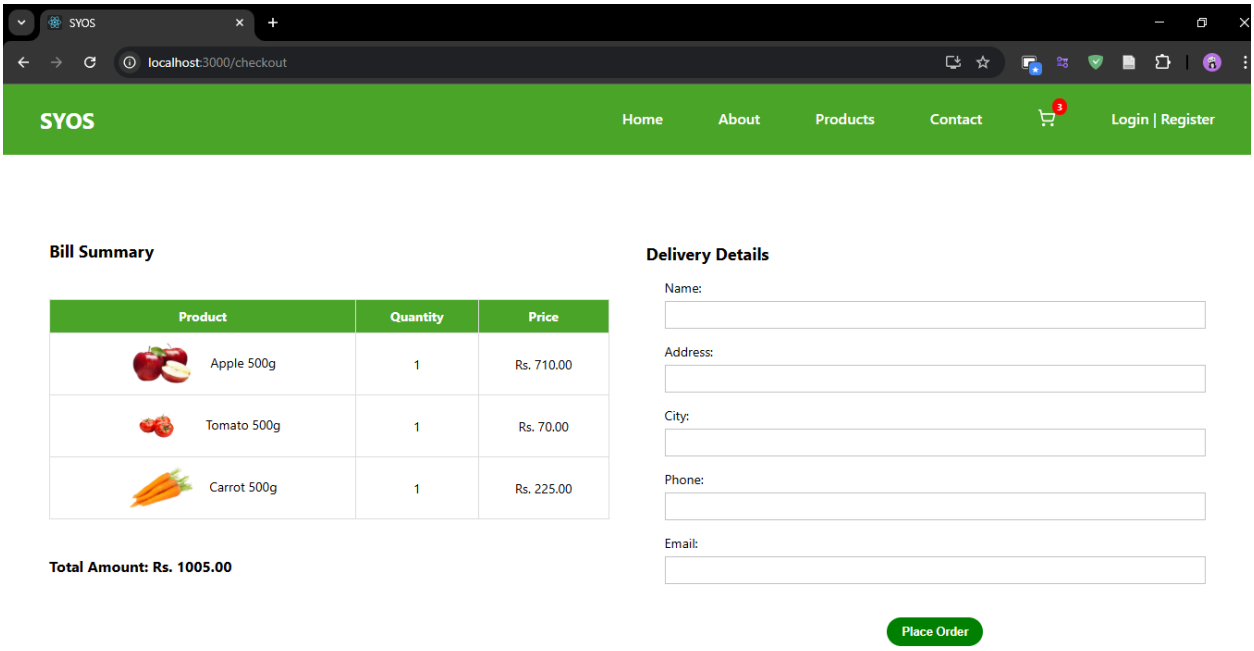
Login Modal



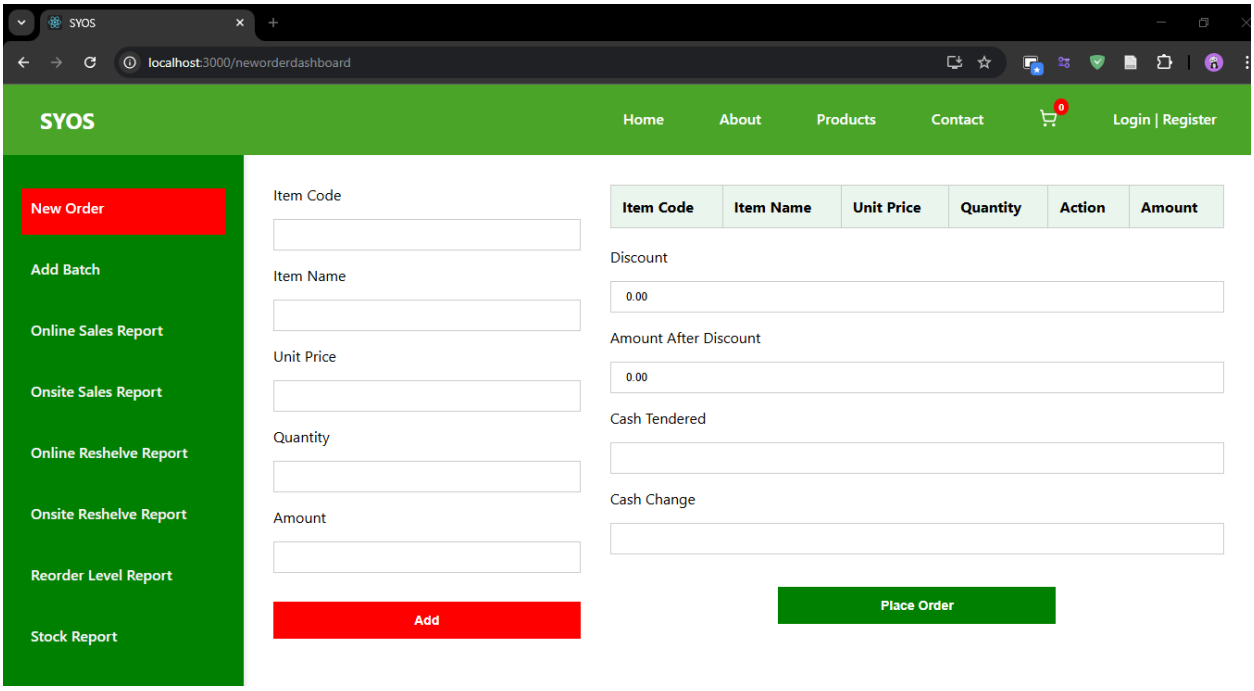
My Cart – Sidebar



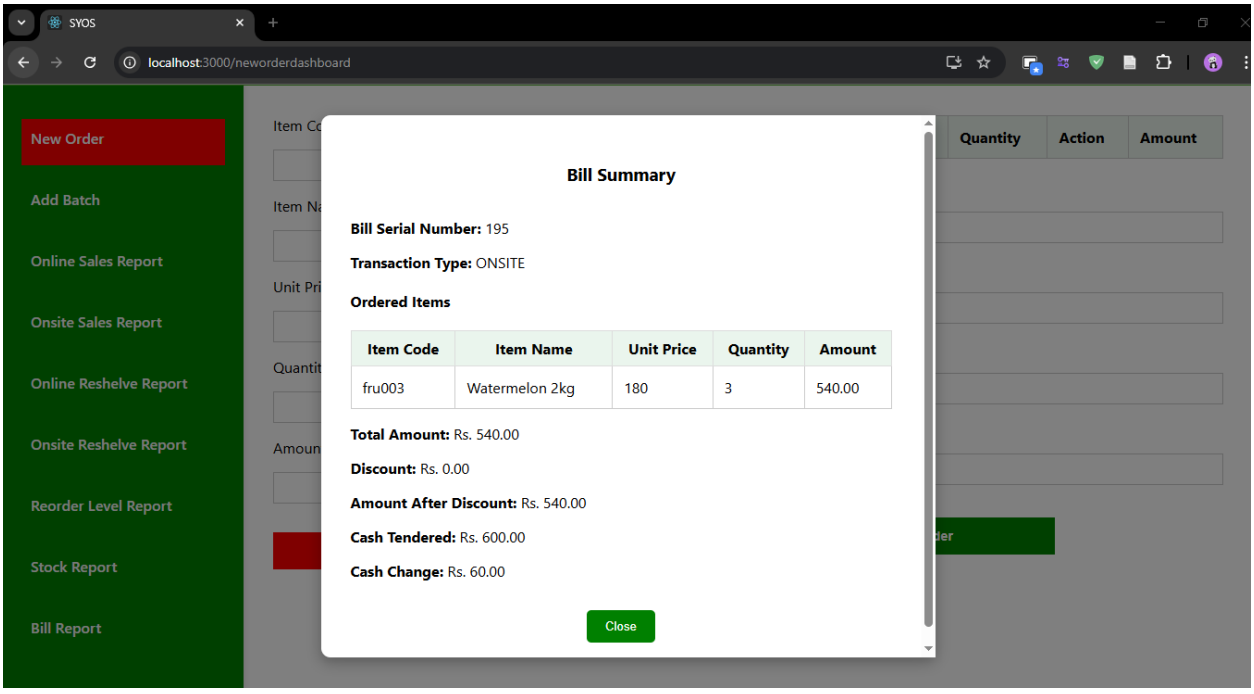
Checkout Page



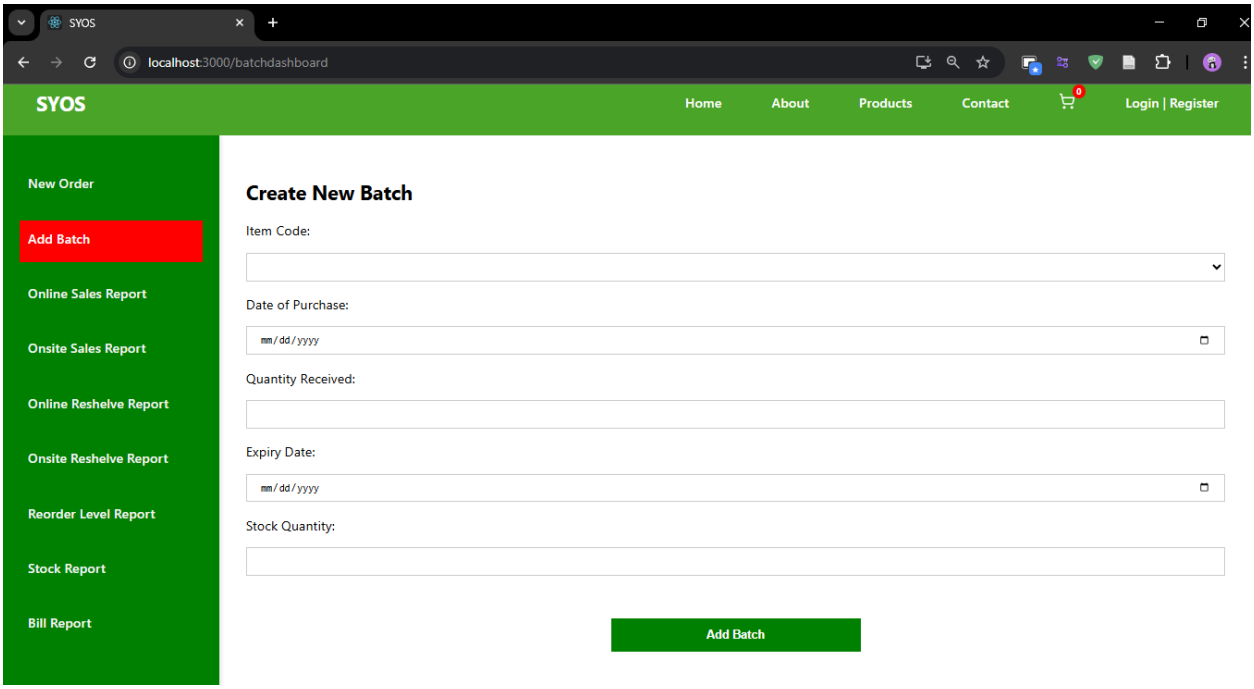
Place Order Page



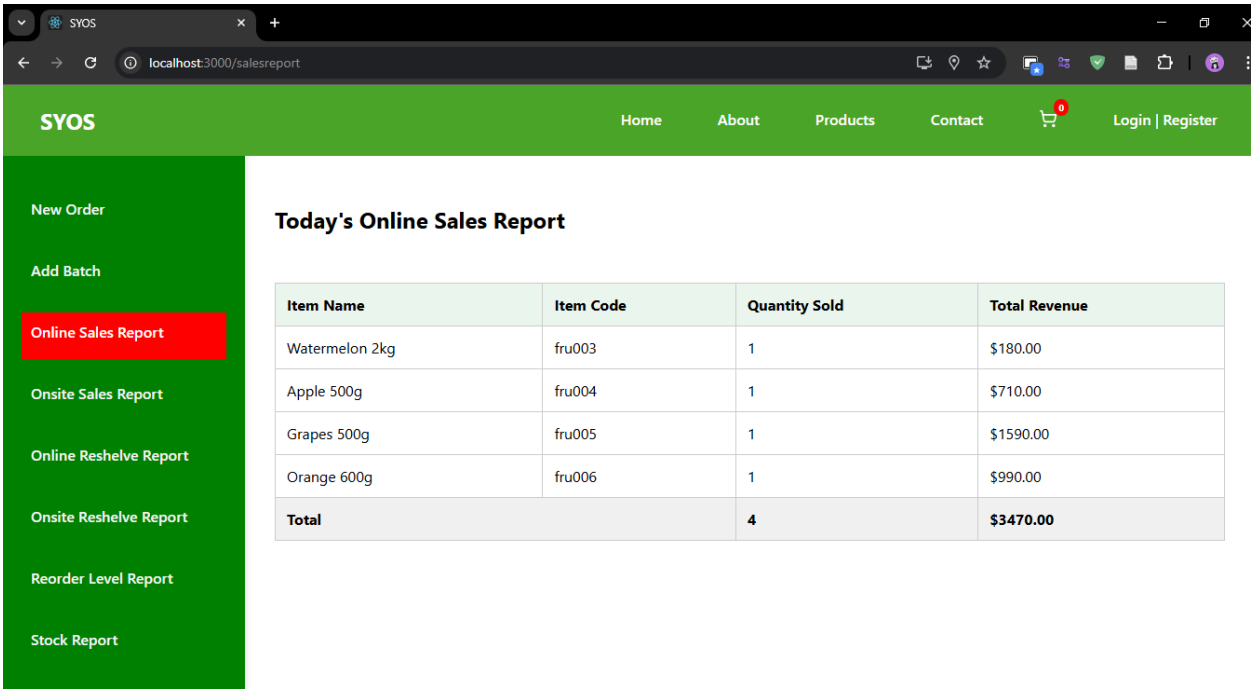
Bill Summary



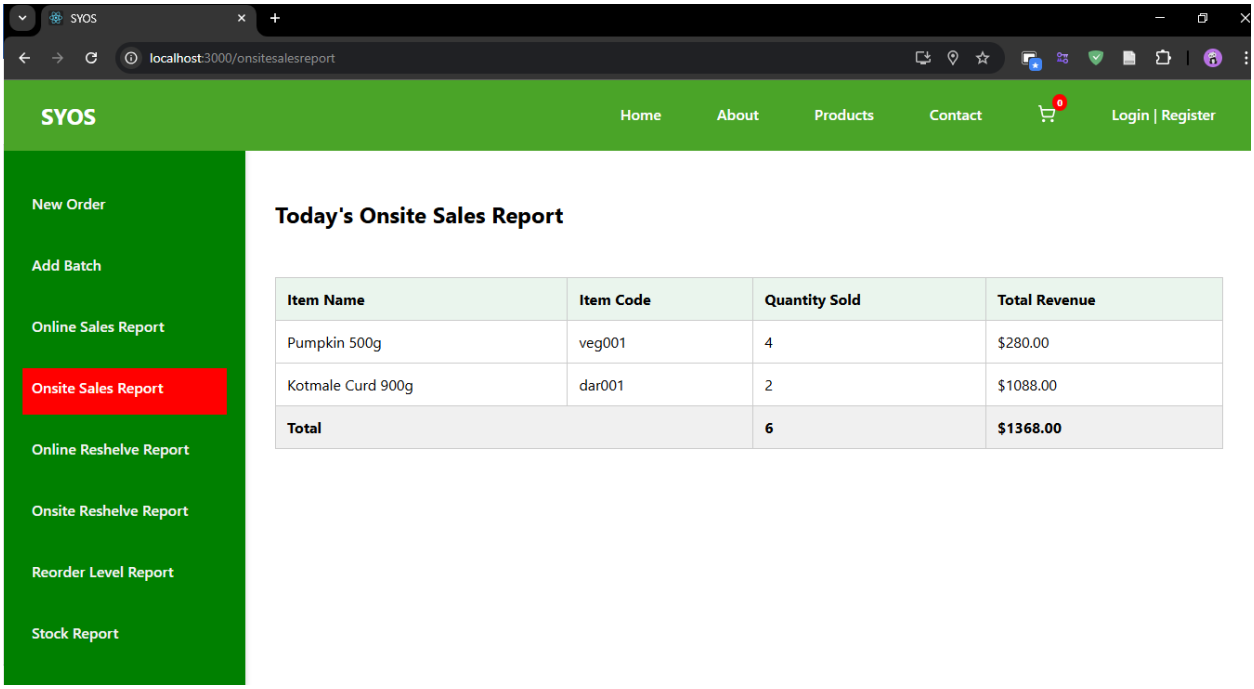
Create New Batch Page



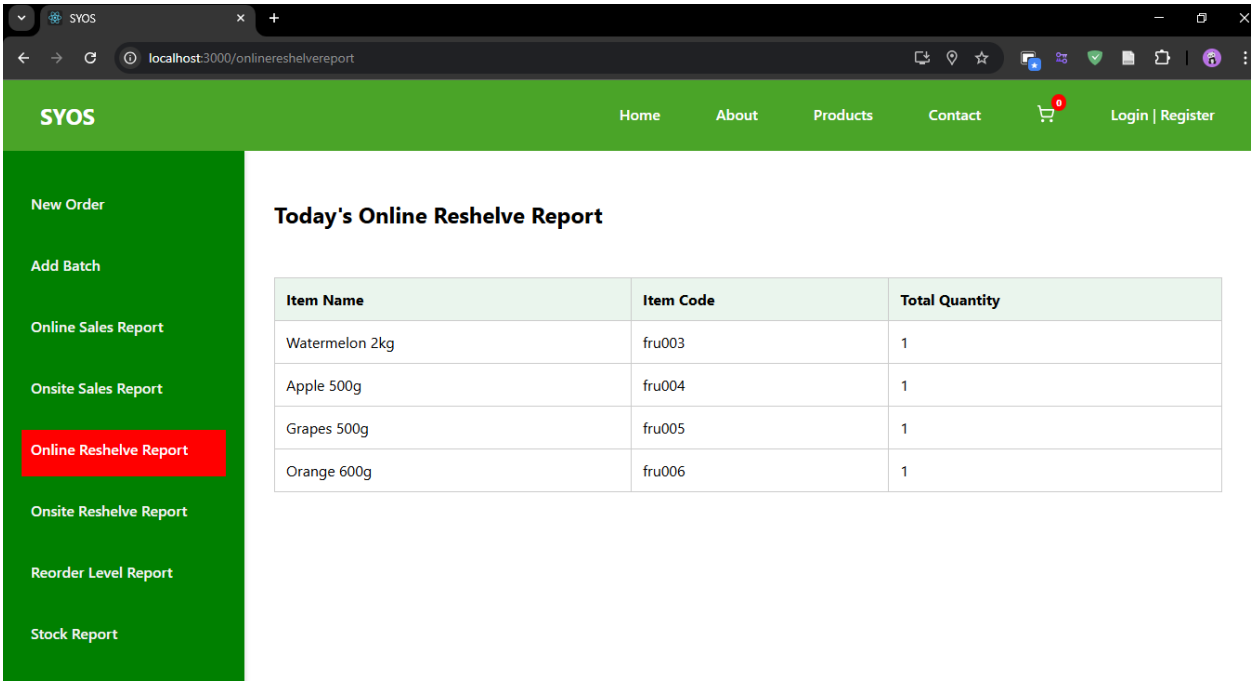
Online Sales Report



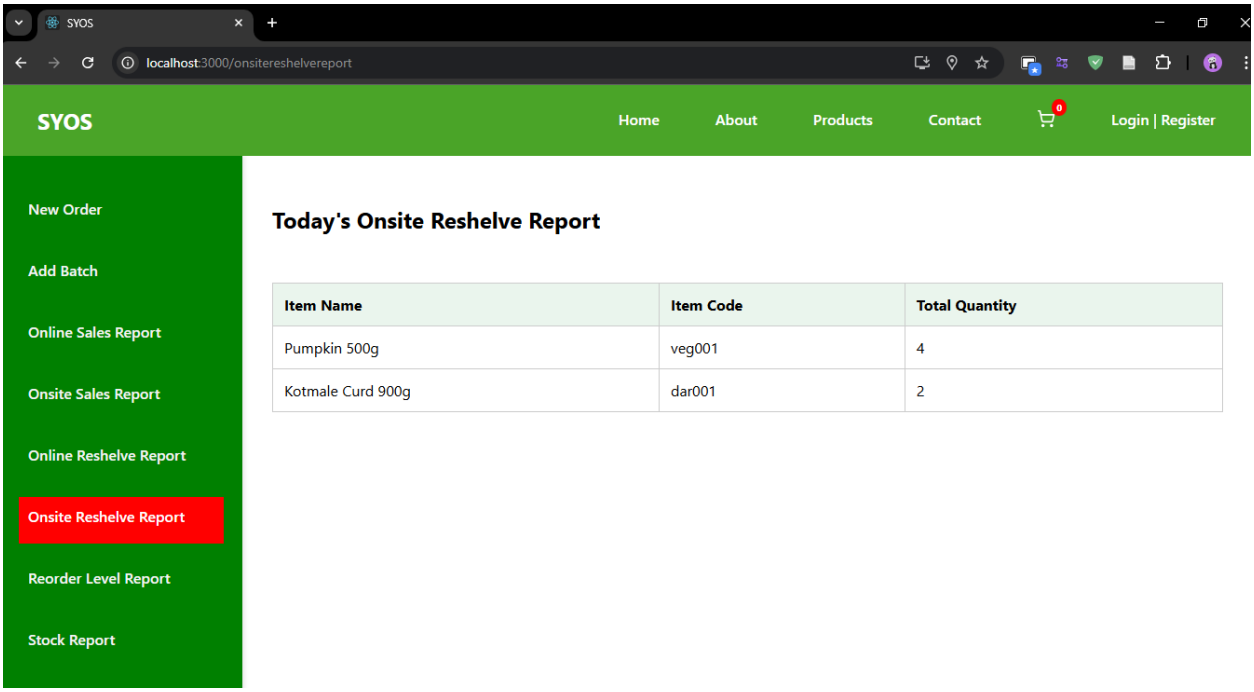
Onsite Sales Report



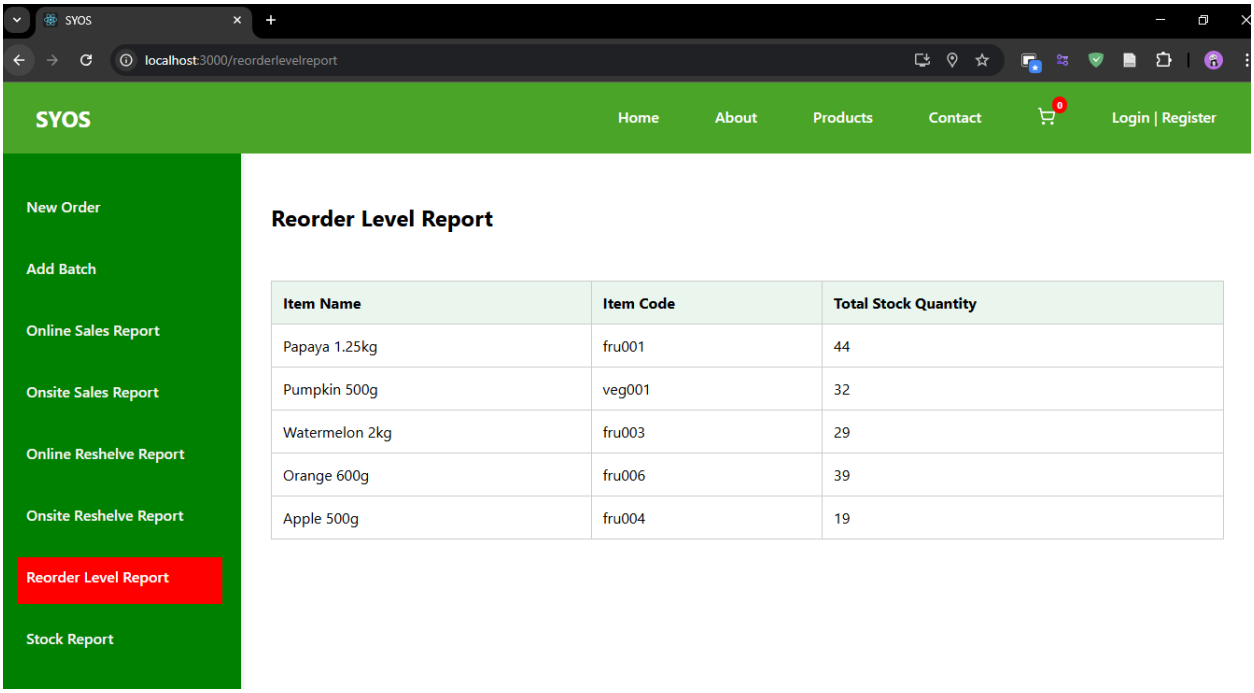
Online Reshelve Report



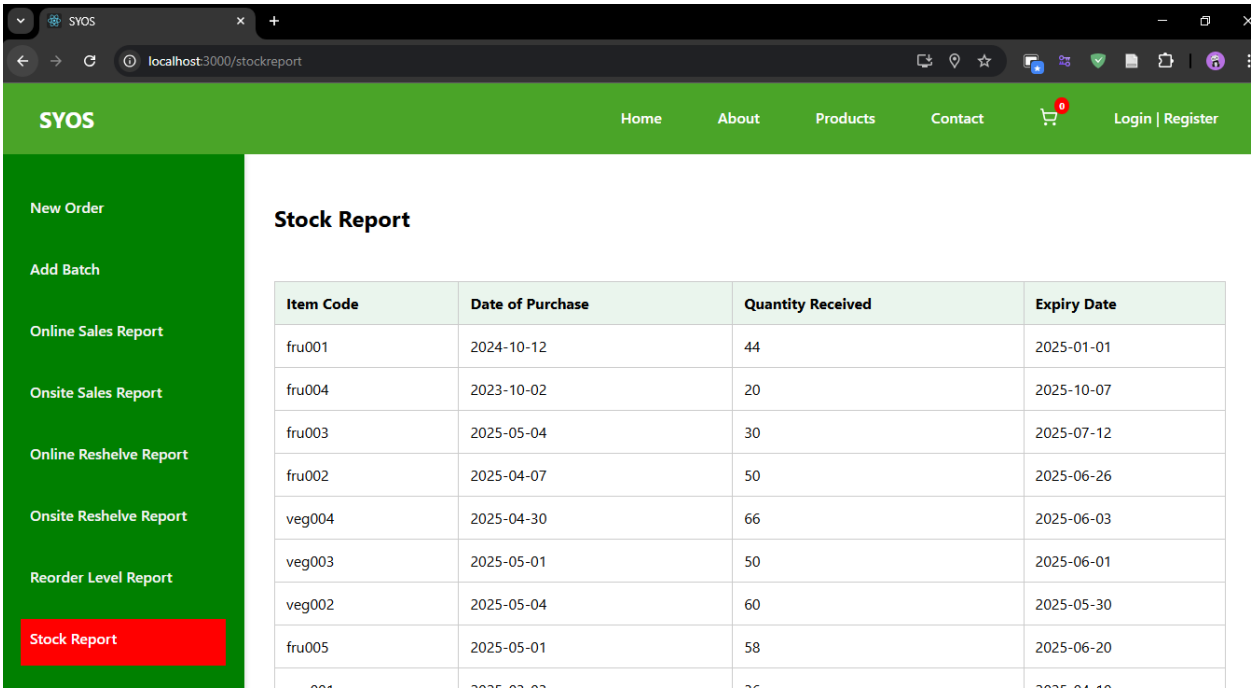
Onsite Reshelve Report



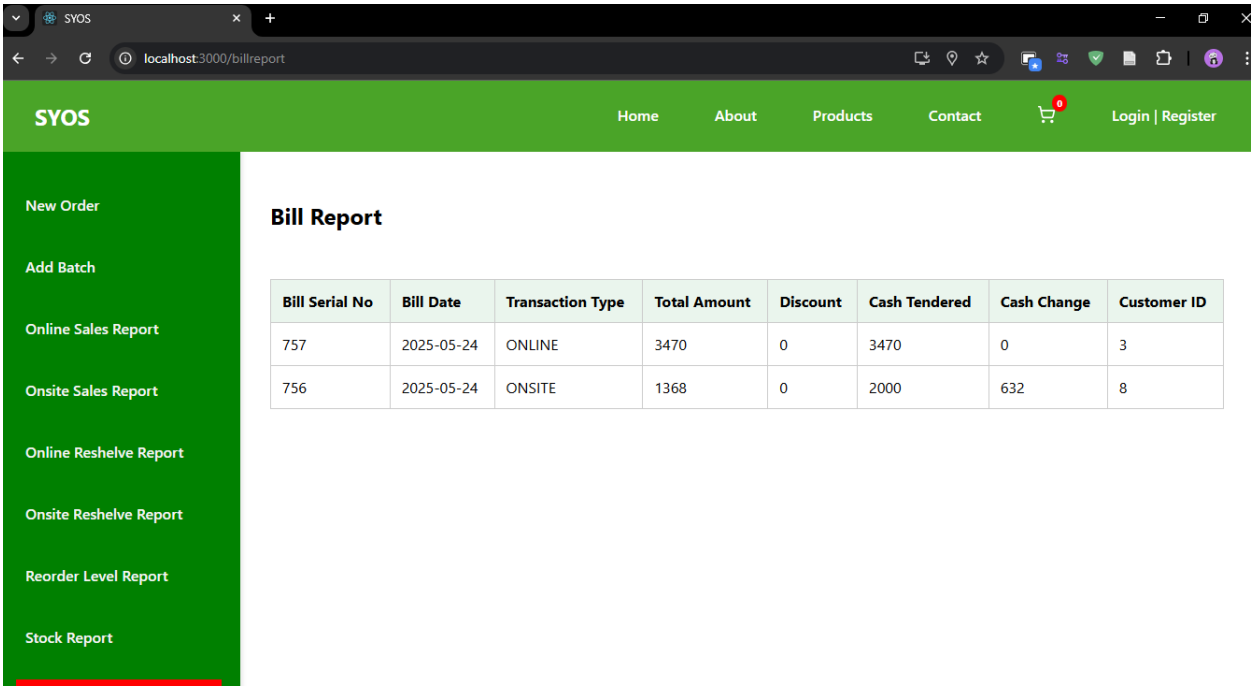
Reorder Level Report



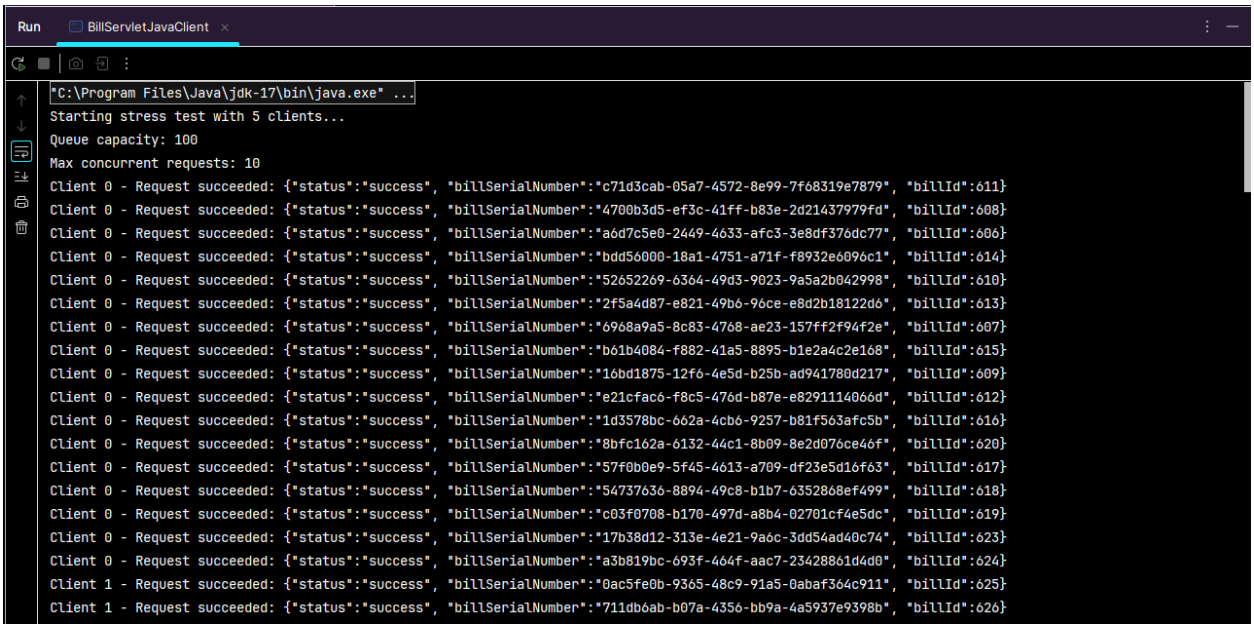
Stock Report

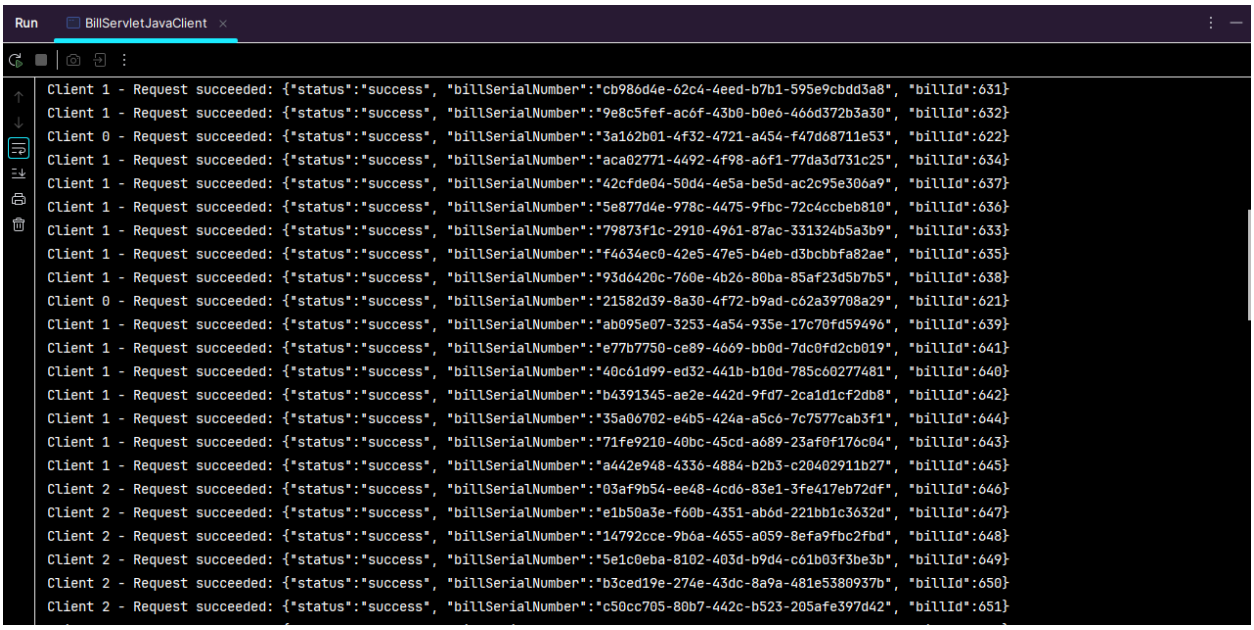


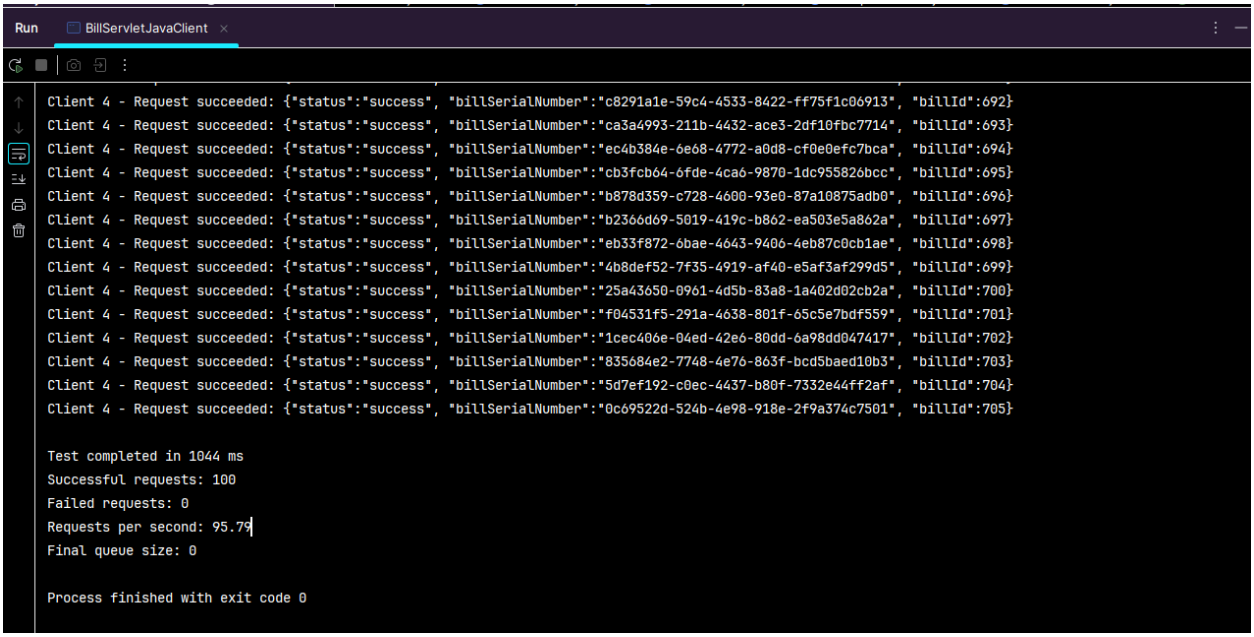
Bill Report



Chapter 5 - Automatic Test Clients

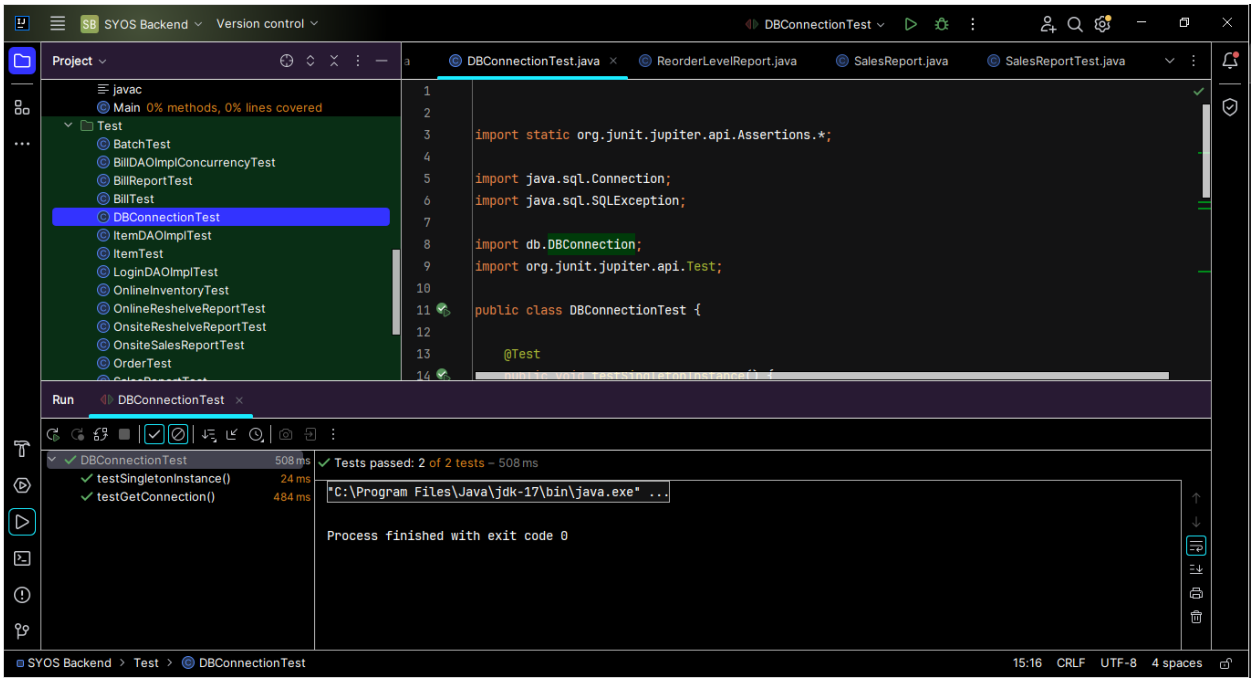


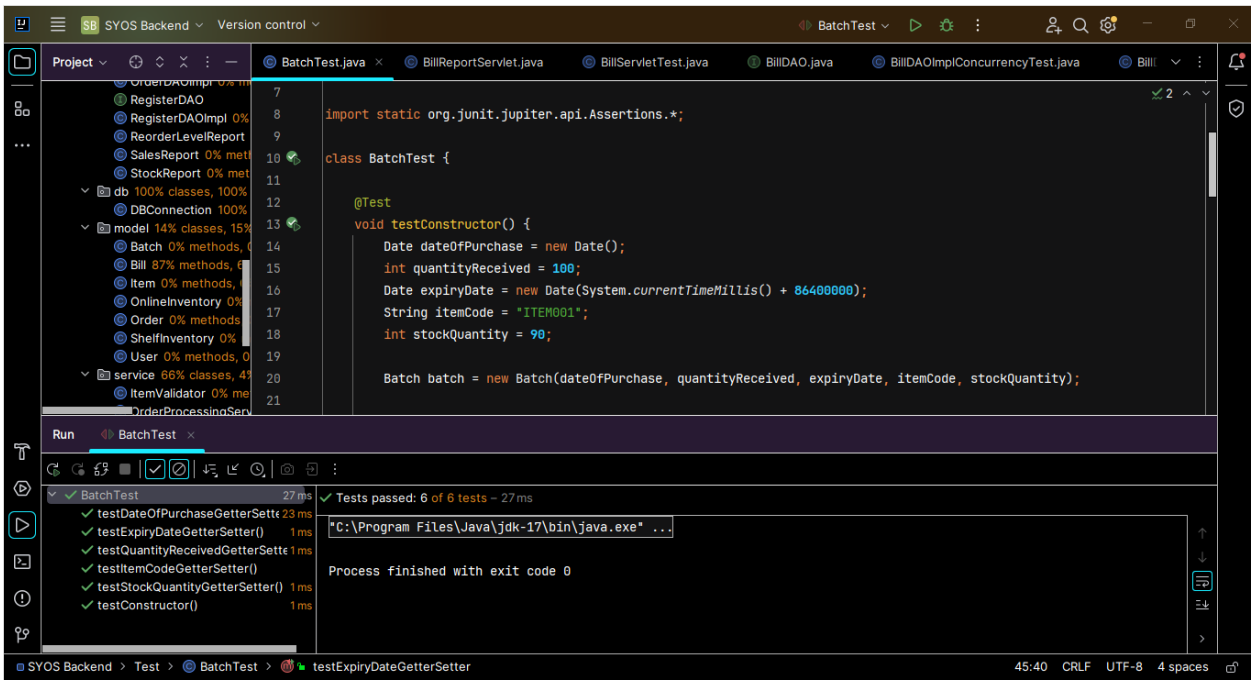


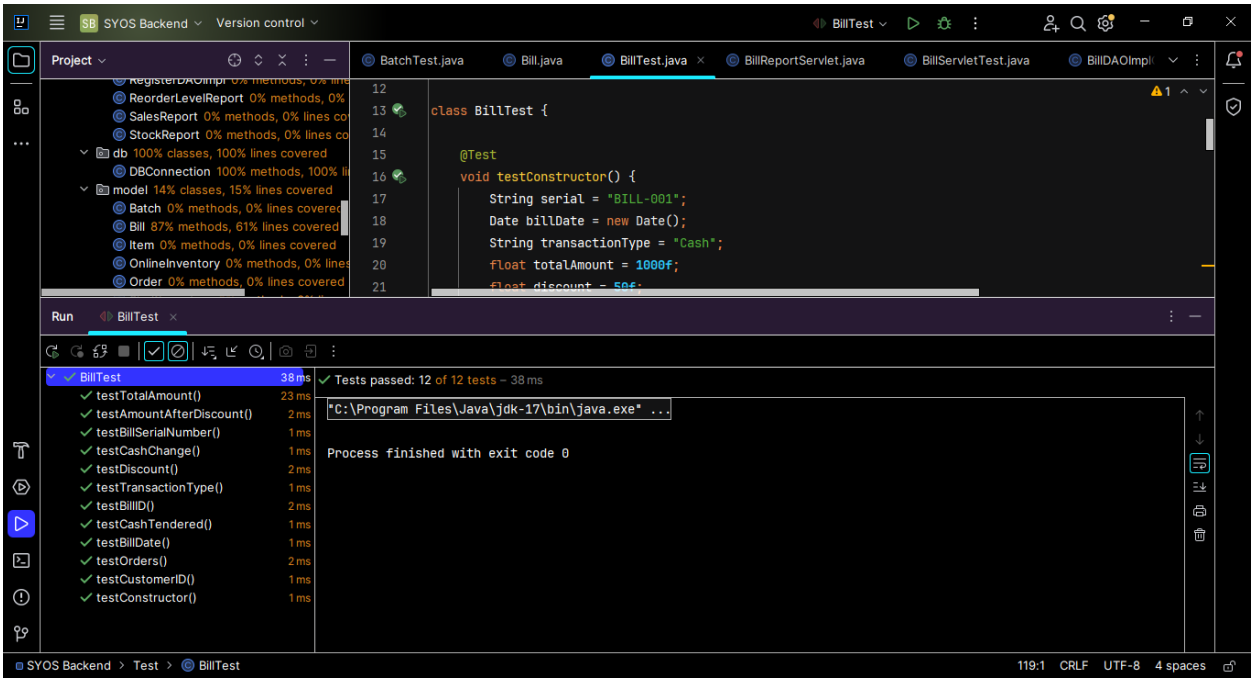

Run
BillServletJavaClient
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"c8291a1e-59c4-4533-8422-ff75f1c06913", "billId":692}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"ca3a4993-211b-4432-ace3-2df10fbc7714", "billId":693}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"ec4b384e-6e68-4772-a0d8-cf0e0efc7bca", "billId":694}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"cb3fcb64-6fde-4ca6-9870-1dc955826bcc", "billId":695}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"b878d359-c728-4600-93e0-87a10875adb0", "billId":696}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"b2366d69-5019-419c-b862-ea503e5a862a", "billId":697}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"eb33f872-6bae-4643-9406-4eb87c0cb1ae", "billId":698}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"4b8def52-7f35-4919-af40-e5af3af299d5", "billId":699}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"25a43650-0961-4d5b-83a8-1a402d02cb2a", "billId":700}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"f04531f5-291a-4638-801f-65c5e7bdf559", "billId":701}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"1cec406e-04ed-42e6-80dd-6a98dd047417", "billId":702}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"835684e2-7748-4e76-863f-bcd5baed10b3", "billId":703}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"5d7ef192-c0ec-4437-b80f-7332e44ff2af", "billId":704}
Client 4 - Request succeeded: {"status":"success", "billSerialNumber":"0c69522d-524b-4e98-918e-2f9a374c7501", "billId":705}
Test completed in 1044 ms
Successful requests: 100
Failed requests: 0
Requests per second: 95.79
Final queue size: 0
Process finished with exit code 0

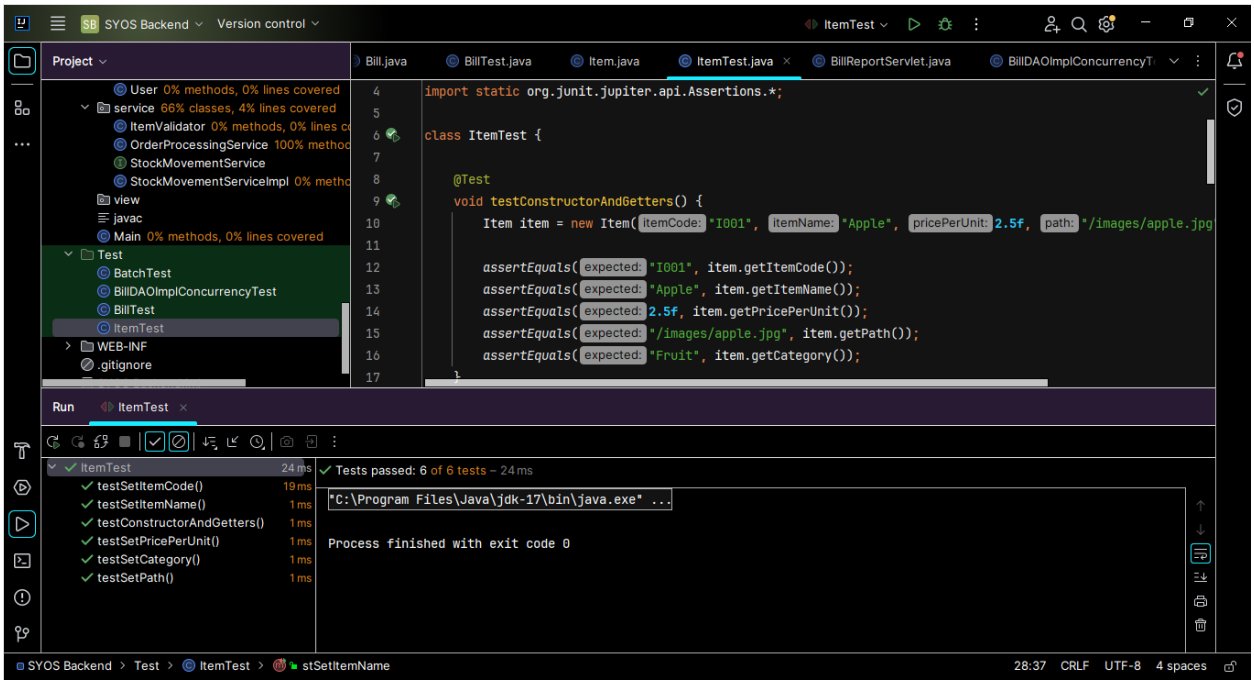
Chapter 6 - Test Cases

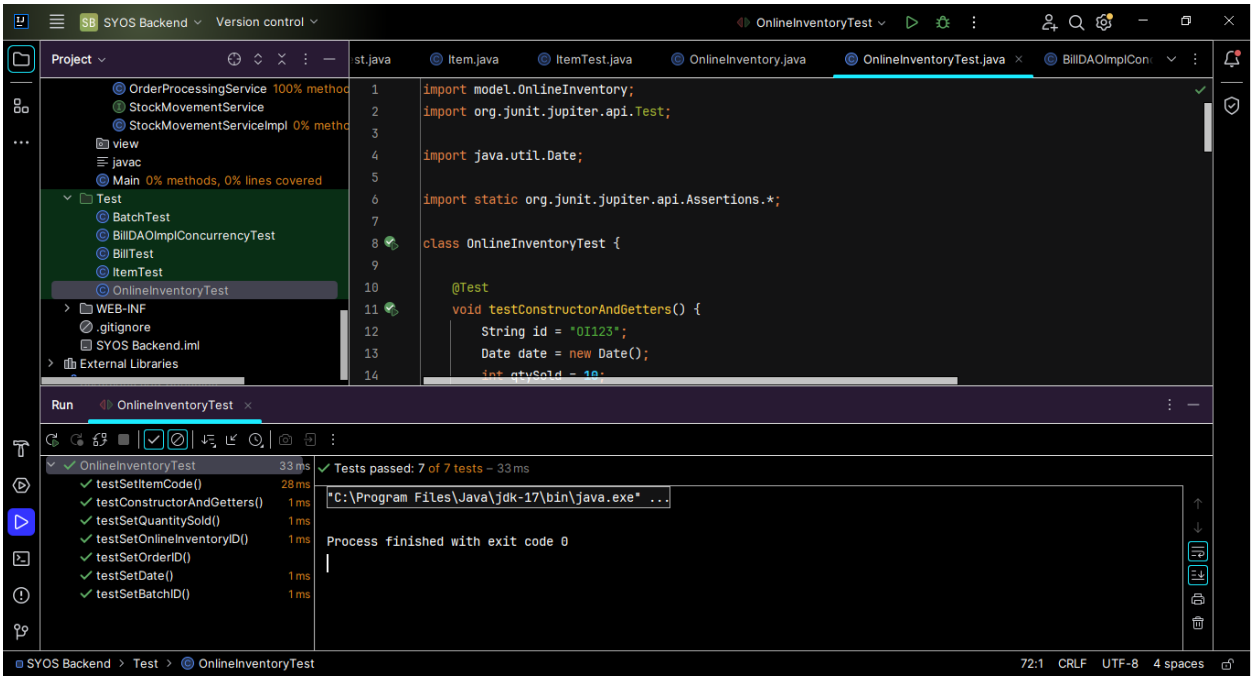
6.1 Junit Test Cases

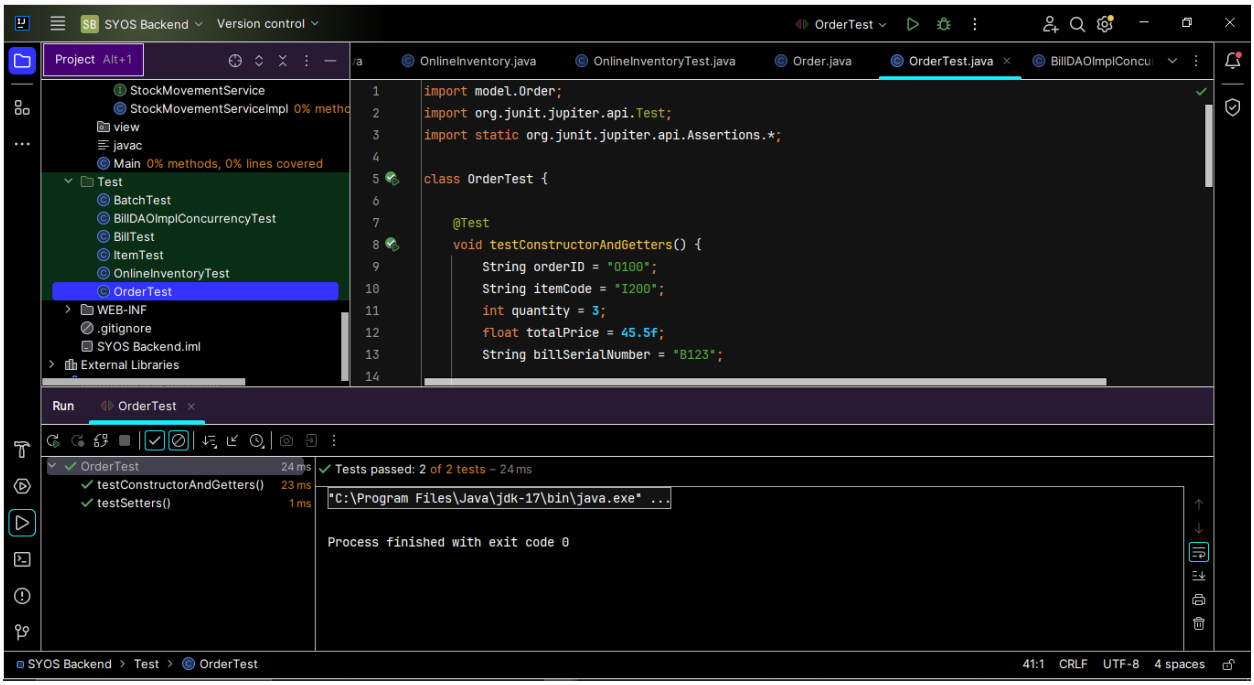


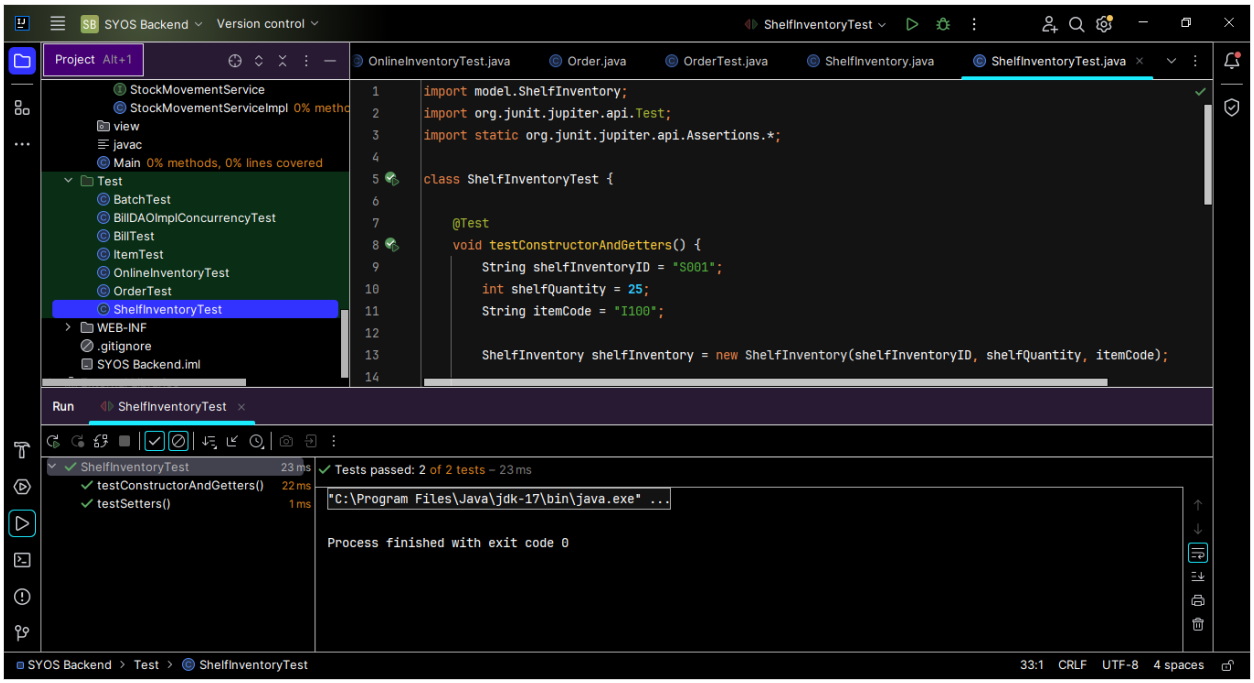


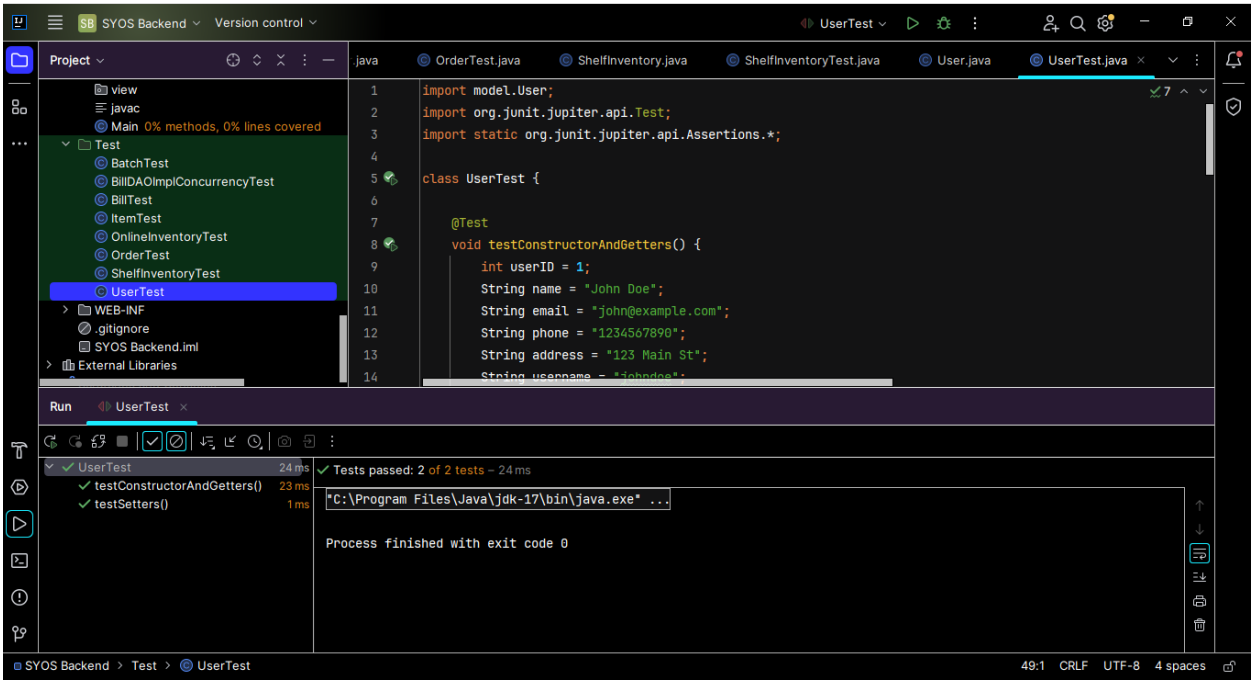

SYOS Backend
Version control
BillTest
Project
BatchTest.java
Bill.java
BillTest.java
BillReportServlet.java
BillServletTest.java
ReorderLevelReport 0% methods, 0%
SalesReport 0% methods, 0% lines co
StockReport 0% methods, 0% lines co
db 100% classes, 100% lines covered
DBConnection 100% methods, 100% li
model 14% classes, 15% lines covered
Batch 0% methods, 0% lines covered
Bill 87% methods, 61% lines covered
Item 0% methods, 0% lines covered
OnlineInventory 0% methods, 0% lines
Order 0% methods, 0% lines covered
class BillTest {
@Test
void testConstructor() {
String serial = "BILL-001";
Date billDate = new Date();
String transactionType = "Cash";
float totalAmount = 1000f;
Run
BillTest
Tests passed: 12 of 12 tests – 38 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
testTotalAmount() 23 ms
testAmountAfterDiscount() 2 ms
testBillSerialNumber() 1 ms
testCashChange() 1 ms
testDiscount() 2 ms
testTransactionType() 1 ms
testBillID() 2 ms
testCashTendered() 1 ms
testBillDate() 1 ms
testOrders() 2 ms
testCustomerID() 1 ms
testConstructor() 1 ms
SYOS Backend > Test > BillTest
119:1 CRLF UTF-8 4 spaces

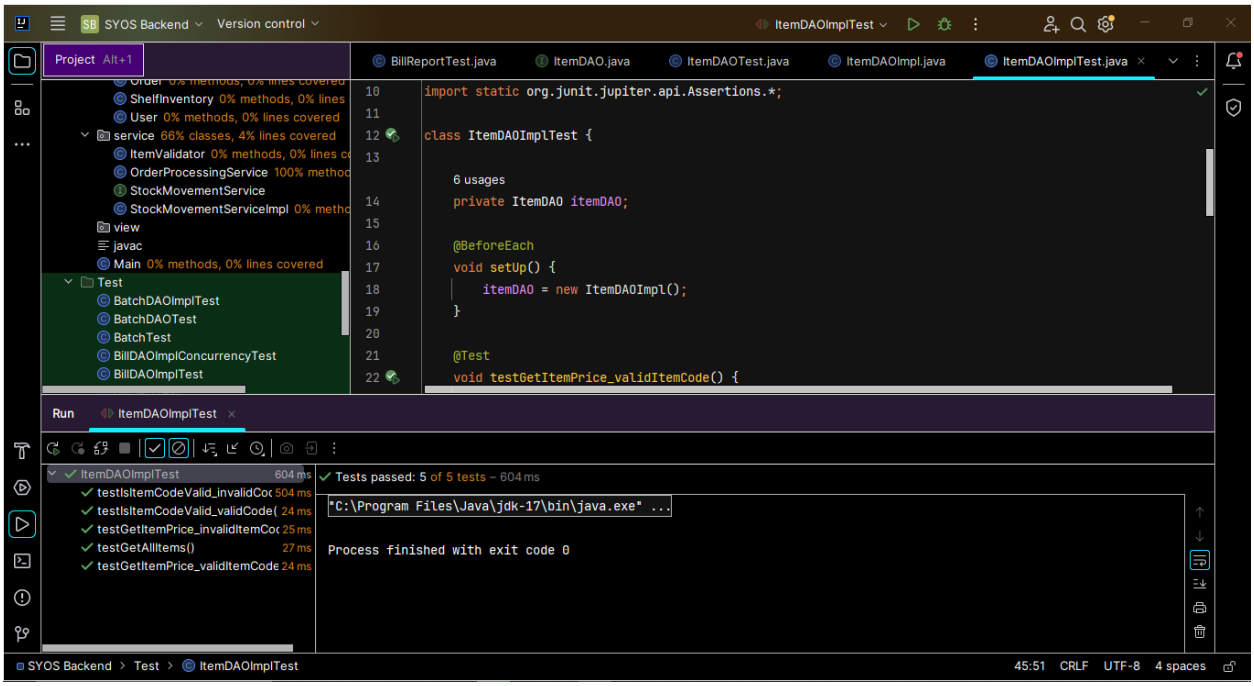

SYOS Backend
Version control
ItemTest
Project
Bill.java
BillTest.java
Item.java
ItemTest.java
BillReportServlet.java
User 0% methods, 0% lines covered
service 66% classes, 4% lines covered
ItemValidator 0% methods, 0% lines c
OrderProcessingService 100% metho
StockMovementService
StockMovementServiceImpl 0% metho
view
javac
Main 0% methods, 0% lines covered
Test
BatchTest
BillDAOImplConcurrencyTest
BillTest
ItemTest
WEB-INF
.gitignore
import static org.junit.jupiter.api.Assertions.*;
class ItemTest {
@Test
void testConstructorAndGetters() {
Item item = new Item(itemCode: "I001", itemName: "Apple", pricePerUnit: 2.5f, path: "/images/apple.jpg
assertEquals(expected: "I001", item.getItemCode());
assertEquals(expected: "Apple", item.getItemName());
assertEquals(expected: 2.5f, item.getPricePerUnit());
assertEquals(expected: "/images/apple.jpg", item.getPath());
assertEquals(expected: "Fruit", item.getCategory());
Run
ItemTest
Tests passed: 6 of 6 tests – 24 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
testSetItemCode() 19 ms
testSetItemName() 1 ms
testConstructorAndGetters() 1 ms
testSetPricePerUnit() 1 ms
testSetCategory() 1 ms
testSetPath() 1 ms
SYOS Backend > Test > ItemTest > stSetItemName
28:37 CRLF UTF-8 4 spaces

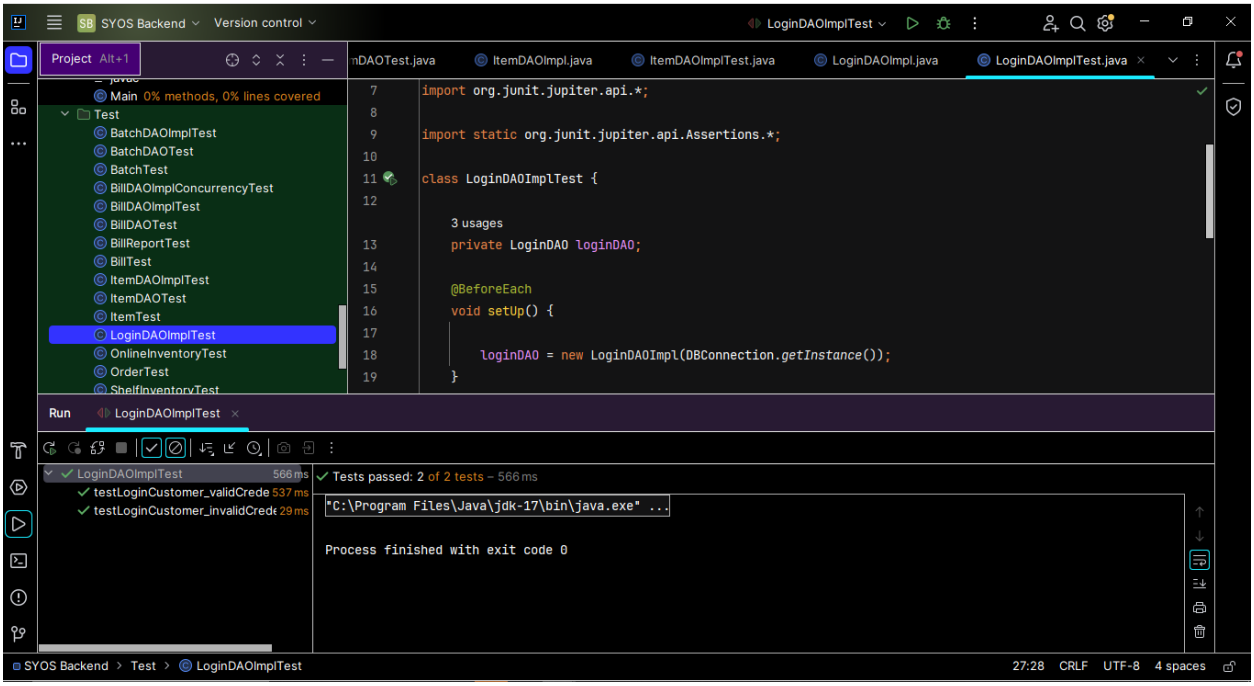

SYOS Backend
Version control
OnlineInventoryTest
Project
OrderProcessingService 100% methoc
StockMovementService
StockMovementServiceImpl 0% metho
view
javac
Main 0% methods, 0% lines covered
Test
BatchTest
BillDAOImplConcurrencyTest
BillTest
ItemTest
OnlineInventoryTest
WEB-INF
.gitignore
SYOS Backend.iml
External Libraries
Item.java
ItemTest.java
OnlineInventory.java
OnlineInventoryTest.java
import model.OnlineInventory;
import org.junit.jupiter.api.Test;
import java.util.Date;
import static org.junit.jupiter.api.Assertions.*;
class OnlineInventoryTest {
@Test
void testConstructorAndGetters() {
String id = "OI123";
Date date = new Date();
Run
OnlineInventoryTest
Tests passed: 7 of 7 tests – 33 ms
OnlineInventoryTest 33 ms
testSetItemCode() 28 ms
testConstructorAndGetters() 1 ms
testSetQuantitySold() 1 ms
testSetOnlineInventoryID() 1 ms
testSetOrderID()
testSetDate() 1 ms
testSetBatchID() 1 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > OnlineInventoryTest
72:1 CRLF UTF-8 4 spaces

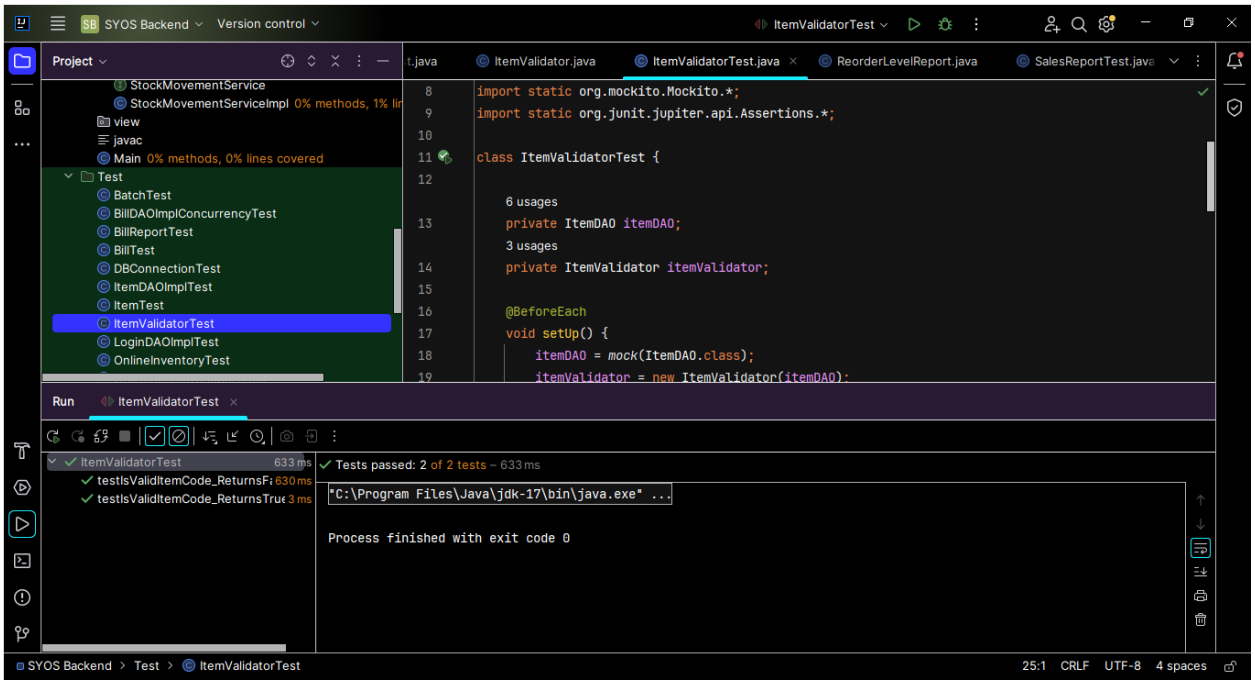

SYOS Backend
Version control
OrderTest
Project Alt+1
StockMovementService
StockMovementServiceImpl 0% metho
view
javac
Main 0% methods, 0% lines covered
Test
BatchTest
BillDAOImplConcurrencyTest
BillTest
ItemTest
OnlineInventoryTest
OrderTest
WEB-INF
.gitignore
SYOS Backend.iml
External Libraries
OnlineInventory.java
OnlineInventoryTest.java
Order.java
OrderTest.java
import model.Order;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
class OrderTest {
@Test
void testConstructorAndGetters() {
String orderID = "O100";
String itemCode = "I200";
int quantity = 3;
float totalPrice = 45.5f;
String billSerialNumber = "B123";
Run
OrderTest
Tests passed: 2 of 2 tests – 24 ms
OrderTest 24 ms
testConstructorAndGetters() 23 ms
testSetters() 1 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > OrderTest
41:1 CRLF UTF-8 4 spaces

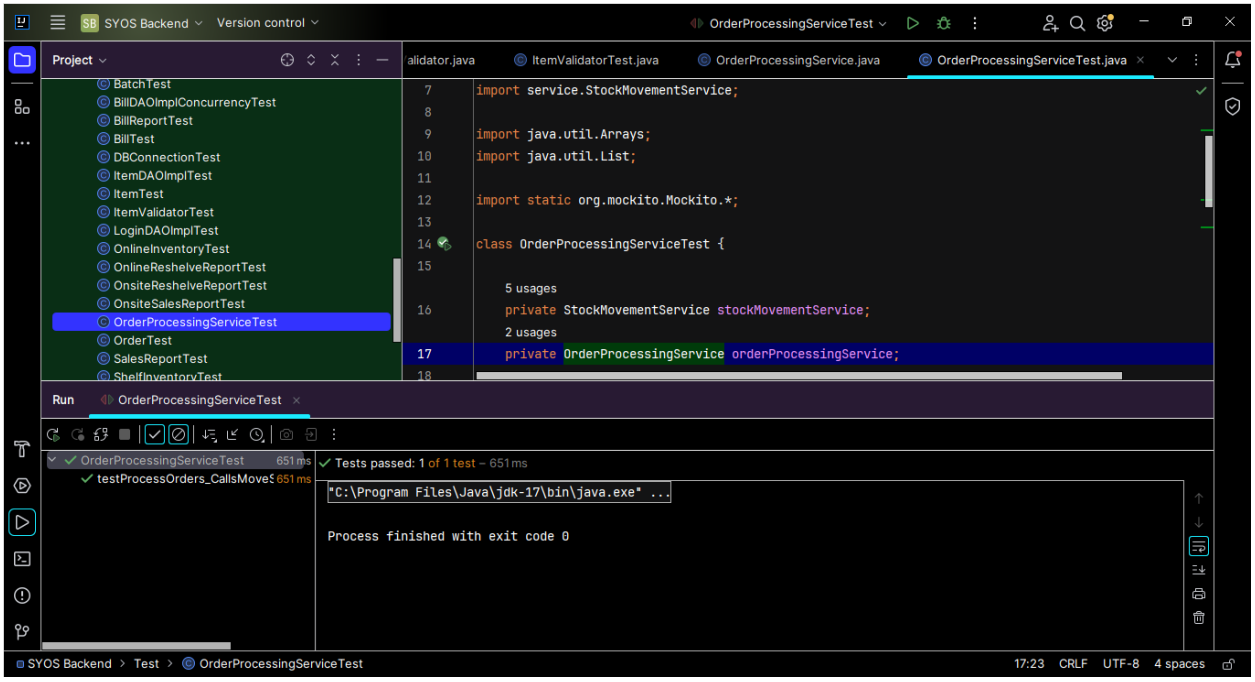

SYOS Backend
Version control
ShelfInventoryTest
Project Alt+1
OnlineInventoryTest.java
Order.java
OrderTest.java
ShelfInventory.java
ShelfInventoryTest.java
StockMovementService
StockMovementServiceImpl 0% methc
view
javac
Main 0% methods, 0% lines covered
Test
BatchTest
BillDAOImplConcurrencyTest
BillTest
ItemTest
OnlineInventoryTest
OrderTest
ShelfInventoryTest
WEB-INF
.gitignore
SYOS Backend.iml
import model.ShelfInventory;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
class ShelfInventoryTest {
@Test
void testConstructorAndGetters() {
String shelfInventoryID = "S001";
int shelfQuantity = 25;
String itemCode = "I100";
ShelfInventory shelfInventory = new ShelfInventory(shelfInventoryID, shelfQuantity, itemCode);
Run
ShelfInventoryTest
ShelfInventoryTest 23 ms
testConstructorAndGetters() 22 ms
testSetters() 1 ms
Tests passed: 2 of 2 tests – 23 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > ShelfInventoryTest
33:1 CRLF UTF-8 4 spaces

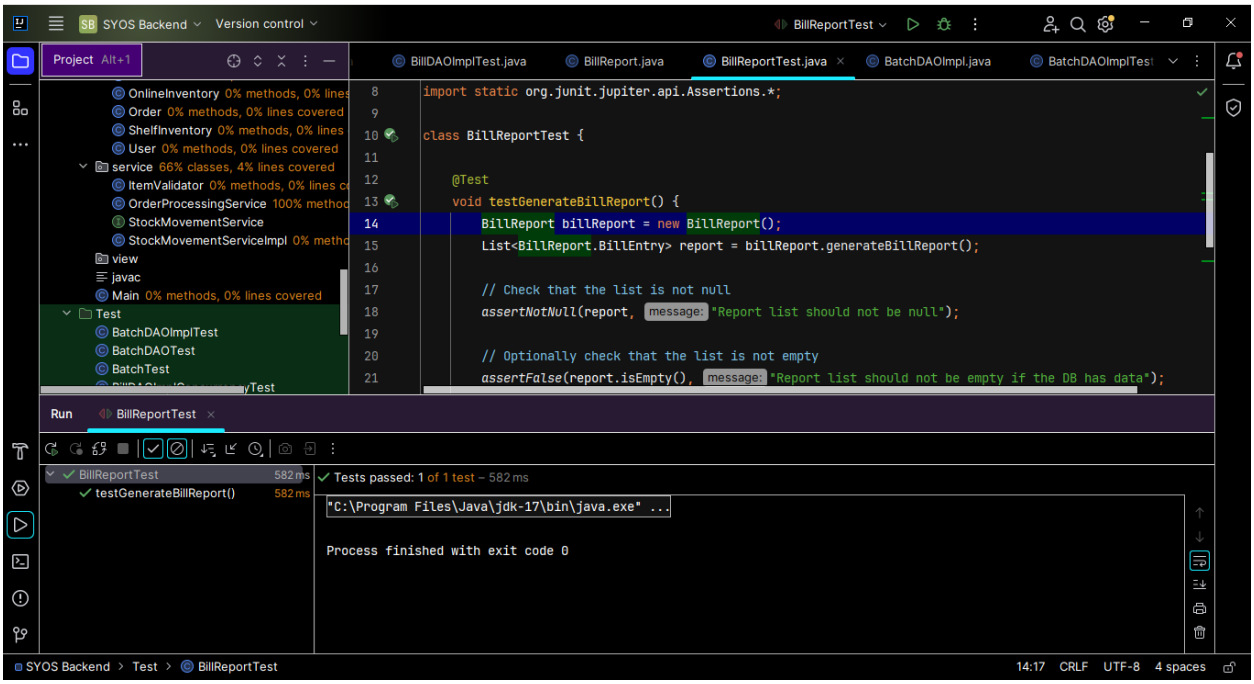

SYOS Backend
Version control
UserTest
Project
java
OrderTest.java
ShelfInventory.java
ShelfInventoryTest.java
User.java
UserTest.java
view
javac
Main 0% methods, 0% lines covered
Test
BatchTest
BillDAOImplConcurrencyTest
BillTest
ItemTest
OnlineInventoryTest
OrderTest
ShelfInventoryTest
UserTest
WEB-INF
.gitignore
SYOS Backend.iml
External Libraries
import model.User;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
class UserTest {
@Test
void testConstructorAndGetters() {
int userID = 1;
String name = "John Doe";
String email = "john@example.com";
String phone = "1234567890";
String address = "123 Main St";
Run
UserTest
UserTest 24 ms
testConstructorAndGetters() 23 ms
testSetters() 1 ms
Tests passed: 2 of 2 tests – 24 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > UserTest
49:1 CRLF UTF-8 4 spaces

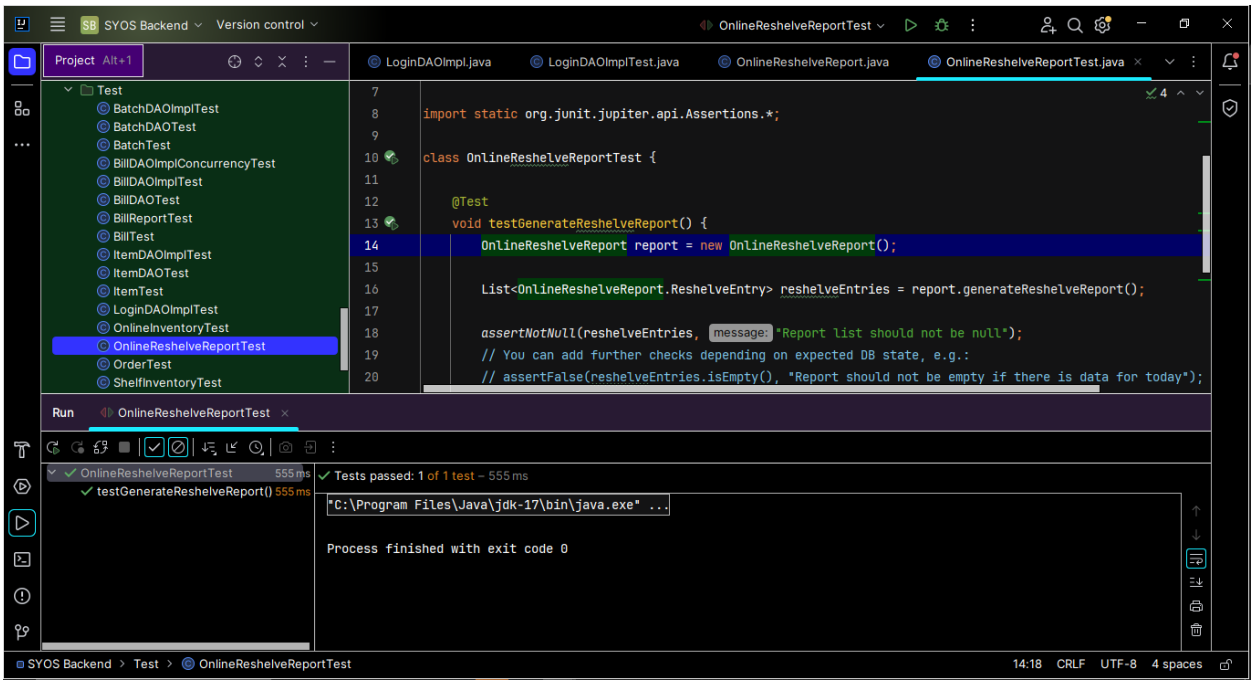

SYOS Backend
Version control
ItemDAOImplTest
Project Alt+1
BillReportTest.java
ItemDAO.java
ItemDAOTest.java
ItemDAOImpl.java
ItemDAOImplTest.java
ShelfInventory 0% methods, 0% lines
User 0% methods, 0% lines covered
service 66% classes, 4% lines covered
ItemValidator 0% methods, 0% lines c
OrderProcessingService 100% metho
StockMovementService
StockMovementServiceImpl 0% metho
view
javac
Main 0% methods, 0% lines covered
Test
BatchDAOImplTest
BatchDAOTest
BatchTest
BillDAOImplConcurrencyTest
BillDAOImplTest
import static org.junit.jupiter.api.Assertions.*;
class ItemDAOImplTest {
6 usages
private ItemDAO itemDAO;
@BeforeEach
void setUp() {
itemDAO = new ItemDAOImpl();
}
@Test
void testGetItemPrice_validItemCode() {
Run
ItemDAOImplTest
ItemDAOImplTest 604 ms
testIsItemCodeValid_invalidCo 504 ms
testIsItemCodeValid_validCode(24 ms
testGetItemPrice_invalidItemCo 25 ms
testGetAllItems() 27 ms
testGetItemPrice_validItemCode 24 ms
Tests passed: 5 of 5 tests – 604 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend › Test › ItemDAOImplTest
45:51 CRLF UTF-8 4 spaces

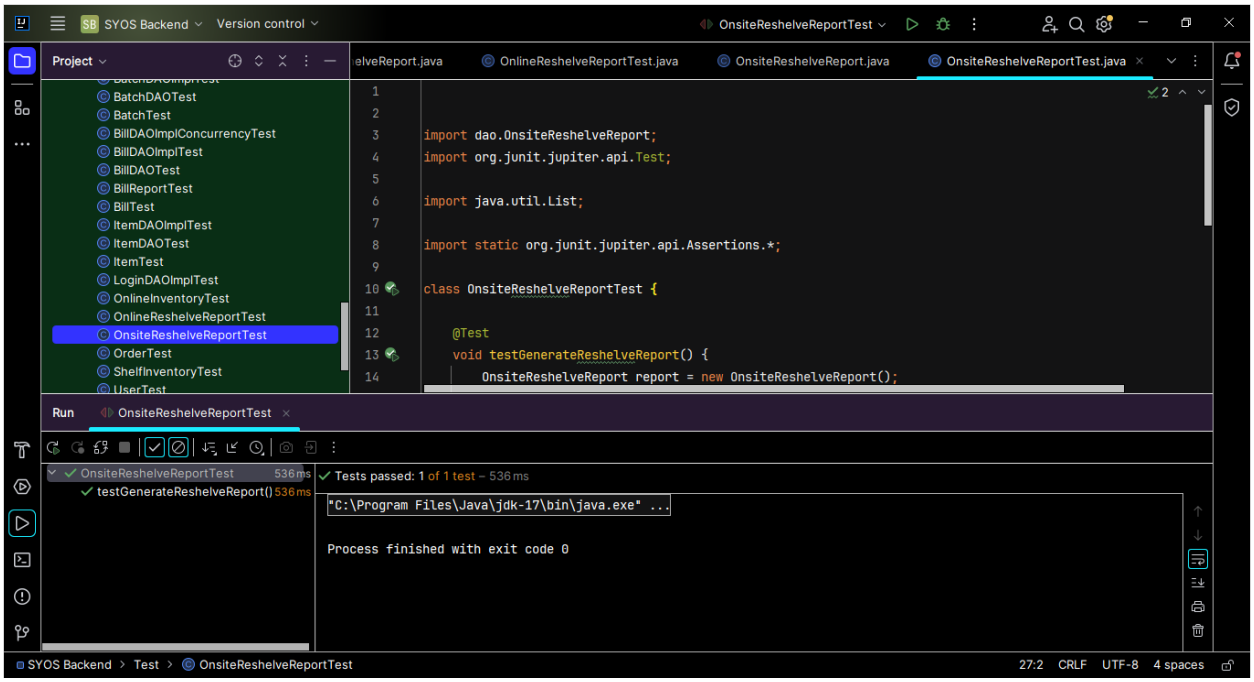

SYOS Backend
Version control
LoginDAOImplTest
Project Alt+1
ItemDAOImpl.java
ItemDAOImplTest.java
LoginDAOImpl.java
LoginDAOImplTest.java
Main 0% methods, 0% lines covered
Test
BatchDAOImplTest
BatchDAOTest
BatchTest
BillDAOImplConcurrencyTest
BillDAOImplTest
BillDAOTest
BillReportTest
BillTest
ItemDAOImplTest
ItemDAOTest
ItemTest
LoginDAOImplTest
OnlineInventoryTest
OrderTest
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
class LoginDAOImplTest {
3 usages
private LoginDAO loginDAO;
@BeforeEach
void setUp() {
loginDAO = new LoginDAOImpl(DBConnection.getInstance());
}
Run
LoginDAOImplTest
LoginDAOImplTest 566 ms
testLoginCustomer_validCrede 537 ms
testLoginCustomer_invalidCred 29 ms
Tests passed: 2 of 2 tests – 566 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend › Test › LoginDAOImplTest
27:28 CRLF UTF-8 4 spaces

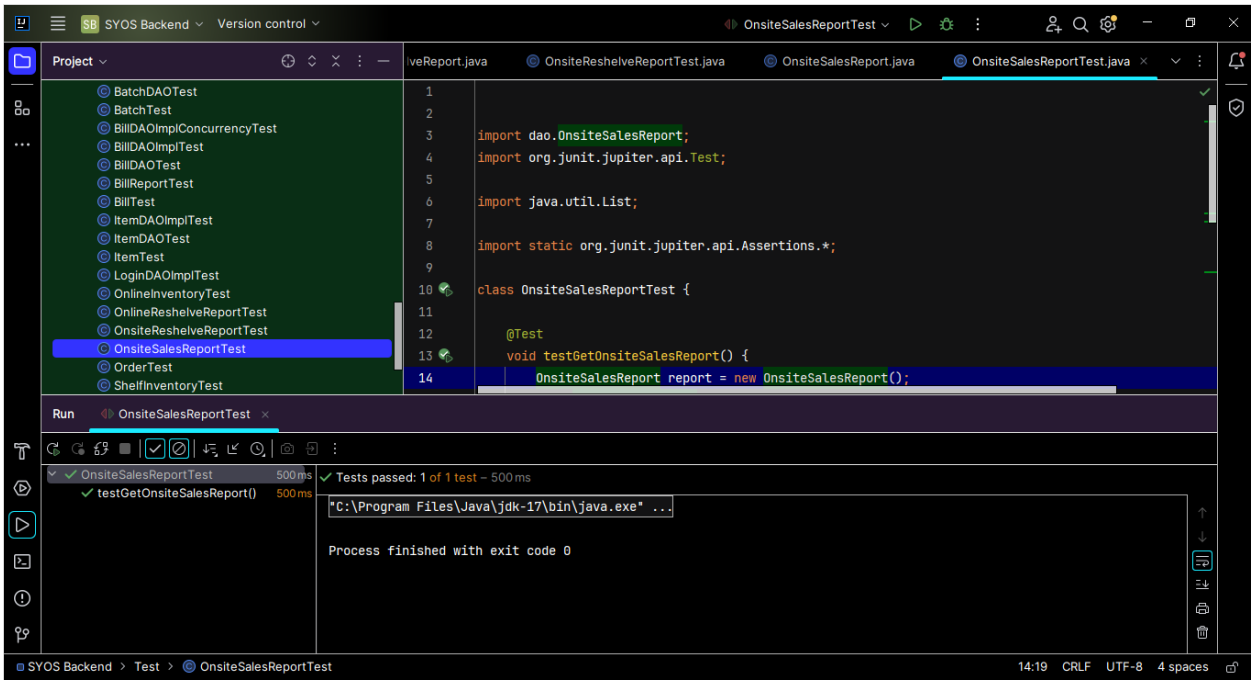

SYOS Backend
Version control
ItemValidatorTest
Project
StockMovementService
StockMovementServiceImpl 0% methods, 1% lir
view
javac
Main 0% methods, 0% lines covered
Test
BatchTest
BillDAOImplConcurrencyTest
BillReportTest
BillTest
DBConnectionTest
ItemDAOImplTest
ItemTest
ItemValidatorTest
LoginDAOImplTest
OnlineInventoryTest
t.java
ItemValidator.java
ItemValidatorTest.java
ReorderLevelReport.java
SalesReportTest.java
8 import static org.mockito.Mockito.*;
9 import static org.junit.jupiter.api.Assertions.*;
10
11 class ItemValidatorTest {
12
6 usages
13 private ItemDAO itemDAO;
3 usages
14 private ItemValidator itemValidator;
15
16 @BeforeEach
17 void setUp() {
18 itemDAO = mock(ItemDAO.class);
19 itemValidator = new ItemValidator(itemDAO);
Run
ItemValidatorTest
ItemValidatorTest 633 ms
testIsValidItemCode_ReturnsF 630 ms
testIsValidItemCode_ReturnsTrue 3 ms
Tests passed: 2 of 2 tests – 633 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > ItemValidatorTest
25:1 CRLF UTF-8 4 spaces

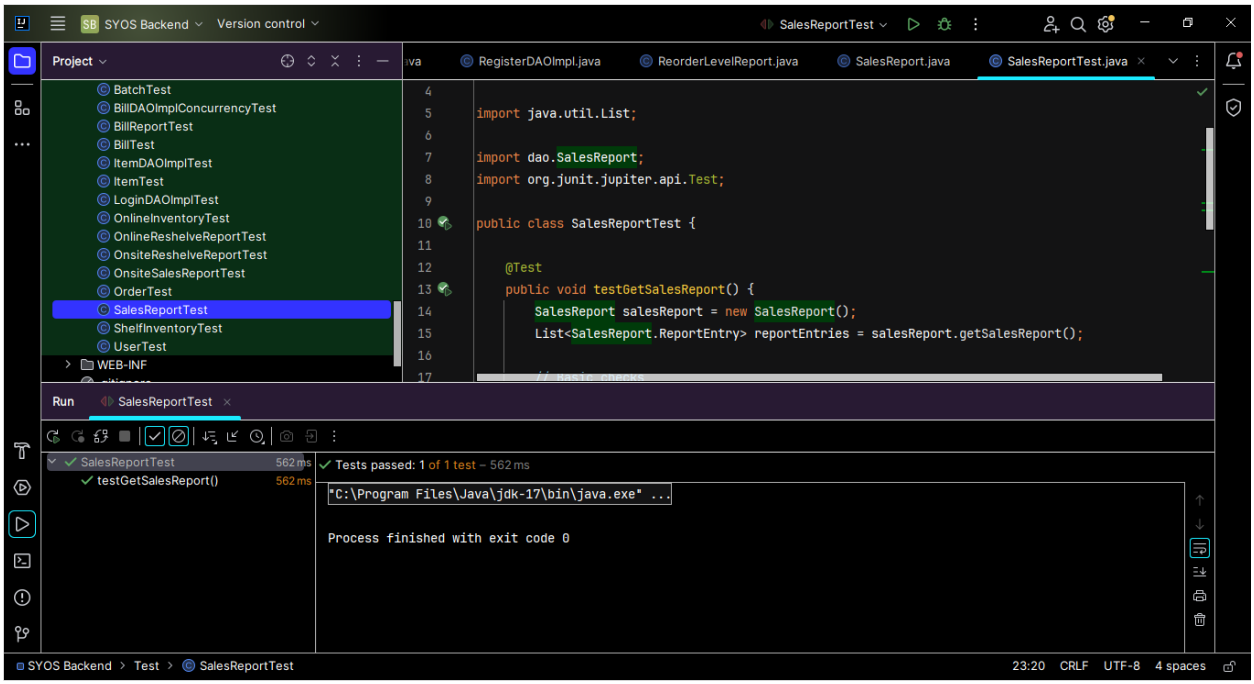

SYOS Backend
Version control
OrderProcessingServiceTest
Project
BatchTest
BillDAOImplConcurrencyTest
BillReportTest
BillTest
DBConnectionTest
ItemDAOImplTest
ItemTest
ItemValidatorTest
LoginDAOImplTest
OnlineInventoryTest
OnlineReshelveReportTest
OnsiteReshelveReportTest
OnsiteSalesReportTest
OrderProcessingServiceTest
OrderTest
SalesReportTest
alidator.java
ItemValidatorTest.java
OrderProcessingService.java
OrderProcessingServiceTest.java
7 import service.StockMovementService;
8
9 import java.util.Arrays;
10 import java.util.List;
11
12 import static org.mockito.Mockito.*;
13
14 class OrderProcessingServiceTest {
15
5 usages
16 private StockMovementService stockMovementService;
2 usages
17 private OrderProcessingService orderProcessingService;
Run
OrderProcessingServiceTest
OrderProcessingServiceTest 651 ms
testProcessOrders_CallsMove 651 ms
Tests passed: 1 of 1 test – 651 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > OrderProcessingServiceTest
17:23 CRLF UTF-8 4 spaces

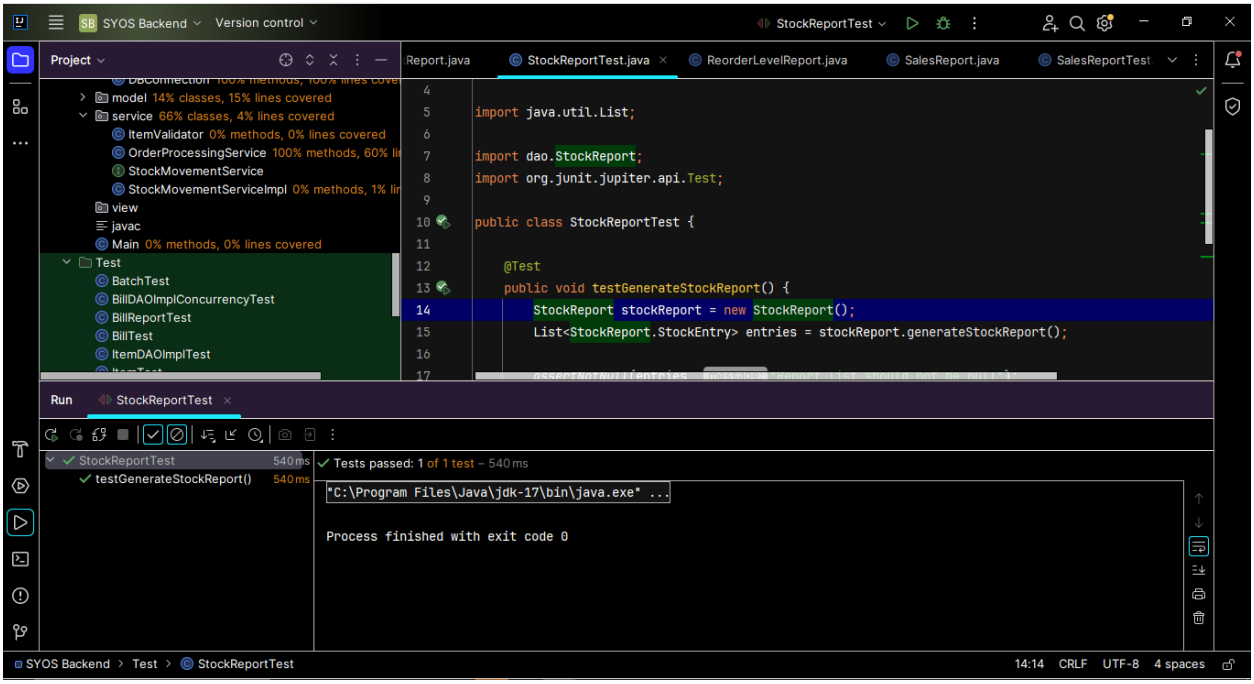

SYOS Backend
Version control
BillReportTest
Project Alt+1
BillDAOImplTest.java
BillReport.java
BillReportTest.java
BatchDAOImpl.java
OnlineInventory 0% methods, 0% lines
Order 0% methods, 0% lines covered
ShelfInventory 0% methods, 0% lines
User 0% methods, 0% lines covered
service 66% classes, 4% lines covered
ItemValidator 0% methods, 0% lines c
OrderProcessingService 100% metho
StockMovementService
StockMovementServiceImpl 0% metho
view
javac
Main 0% methods, 0% lines covered
Test
BatchDAOImplTest
BatchDAOTest
BatchTest
import static org.junit.jupiter.api.Assertions.*;
class BillReportTest {
@Test
void testGenerateBillReport() {
BillReport billReport = new BillReport();
List<BillReport.BillEntry> report = billReport.generateBillReport();
// Check that the list is not null
assertNotNull(report, message: "Report list should not be null");
// Optionally check that the list is not empty
assertFalse(report.isEmpty(), message: "Report list should not be empty if the DB has data");
Run
BillReportTest
BillReportTest 582 ms
testGenerateBillReport() 582 ms
Tests passed: 1 of 1 test – 582 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > BillReportTest
14:17 CRLF UTF-8 4 spaces

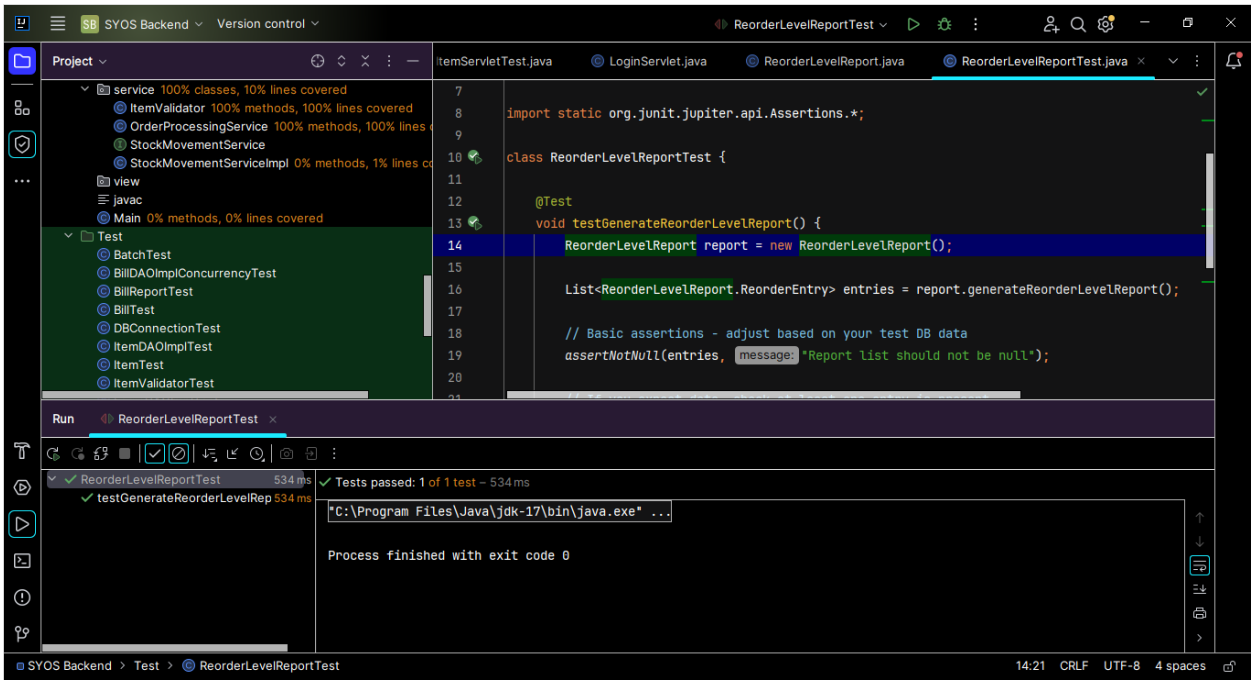

SYOS Backend
Version control
OnlineReshelveReportTest
Project Alt+1
LoginDAOImpl.java
LoginDAOImplTest.java
OnlineReshelveReport.java
OnlineReshelveReportTest.java
Test
BatchDAOImplTest
BatchDAOTest
BatchTest
BillDAOImplConcurrencyTest
BillDAOImplTest
BillDAOTest
BillReportTest
BillTest
ItemDAOImplTest
ItemDAOTest
ItemTest
LoginDAOImplTest
OnlineInventoryTest
OnlineReshelveReportTest
OrderTest
ShelfInventoryTest
import static org.junit.jupiter.api.Assertions.*;
class OnlineReshelveReportTest {
@Test
void testGenerateReshelveReport() {
OnlineReshelveReport report = new OnlineReshelveReport();
List<OnlineReshelveReport.ReshelveEntry> reshelveEntries = report.generateReshelveReport();
assertNotNull(reshelveEntries, message: "Report list should not be null");
// You can add further checks depending on expected DB state, e.g.:
// assertFalse(reshelveEntries.isEmpty(), "Report should not be empty if there is data for today");
Run
OnlineReshelveReportTest
OnlineReshelveReportTest 555 ms
testGenerateReshelveReport() 555 ms
Tests passed: 1 of 1 test – 555 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > OnlineReshelveReportTest
14:18 CRLF UTF-8 4 spaces

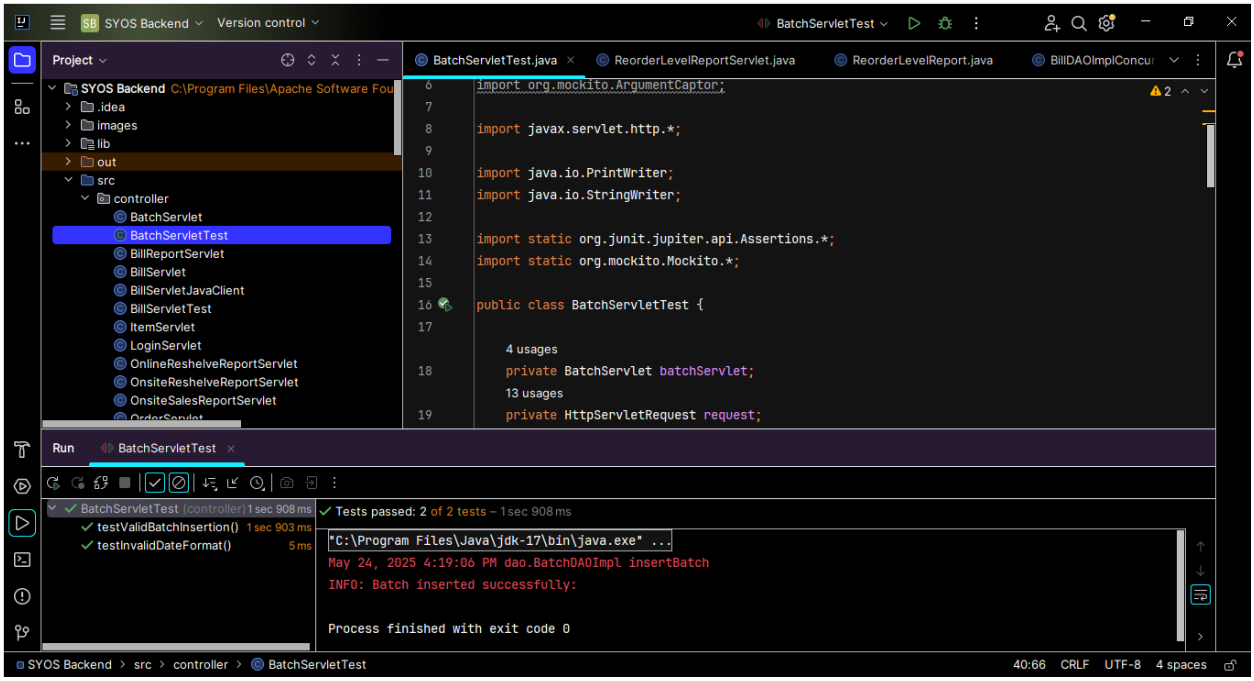

SYOS Backend
Version control
OnsiteReshelveReportTest
Project
OnlineReshelveReportTest.java
OnsiteReshelveReport.java
OnsiteReshelveReportTest.java
BatchDAOTest
BatchTest
BillDAOImplConcurrencyTest
BillDAOImplTest
BillDAOTest
BillReportTest
BillTest
ItemDAOImplTest
ItemDAOTest
ItemTest
LoginDAOImplTest
OnlineInventoryTest
OnlineReshelveReportTest
OnsiteReshelveReportTest
OrderTest
ShelfInventoryTest
UserTest
import dao.OnsiteReshelveReport;
import org.junit.jupiter.api.Test;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;
class OnsiteReshelveReportTest {
@Test
void testGenerateReshelveReport() {
OnsiteReshelveReport report = new OnsiteReshelveReport();
Run
OnsiteReshelveReportTest
OnsiteReshelveReportTest 536 ms
testGenerateReshelveReport() 536 ms
Tests passed: 1 of 1 test – 536 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > OnsiteReshelveReportTest
27:2 CRLF UTF-8 4 spaces

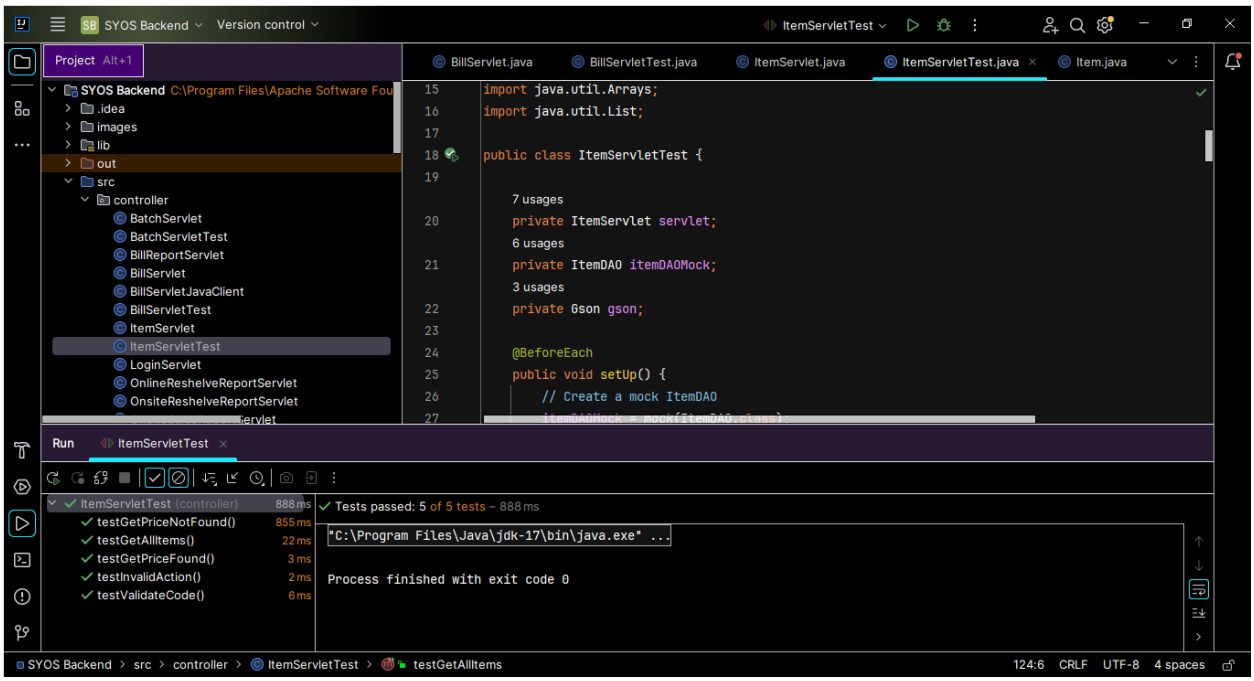

SYOS Backend
Version control
OnsiteSalesReportTest
Project
OnlineReshelveReportTest.java
OnsiteSalesReport.java
OnsiteSalesReportTest.java
BatchDAOTest
BatchTest
BillDAOImplConcurrencyTest
BillDAOImplTest
BillDAOTest
BillReportTest
BillTest
ItemDAOImplTest
ItemDAOTest
ItemTest
LoginDAOImplTest
OnlineInventoryTest
OnlineReshelveReportTest
OnsiteReshelveReportTest
OnsiteSalesReportTest
OrderTest
ShelfInventoryTest
import dao.OnsiteSalesReport;
import org.junit.jupiter.api.Test;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;
class OnsiteSalesReportTest {
@Test
void testGetOnsiteSalesReport() {
OnsiteSalesReport report = new OnsiteSalesReport();
Run
OnsiteSalesReportTest
OnsiteSalesReportTest 500 ms
testGetOnsiteSalesReport() 500 ms
Tests passed: 1 of 1 test – 500 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > OnsiteSalesReportTest
14:19 CRLF UTF-8 4 spaces

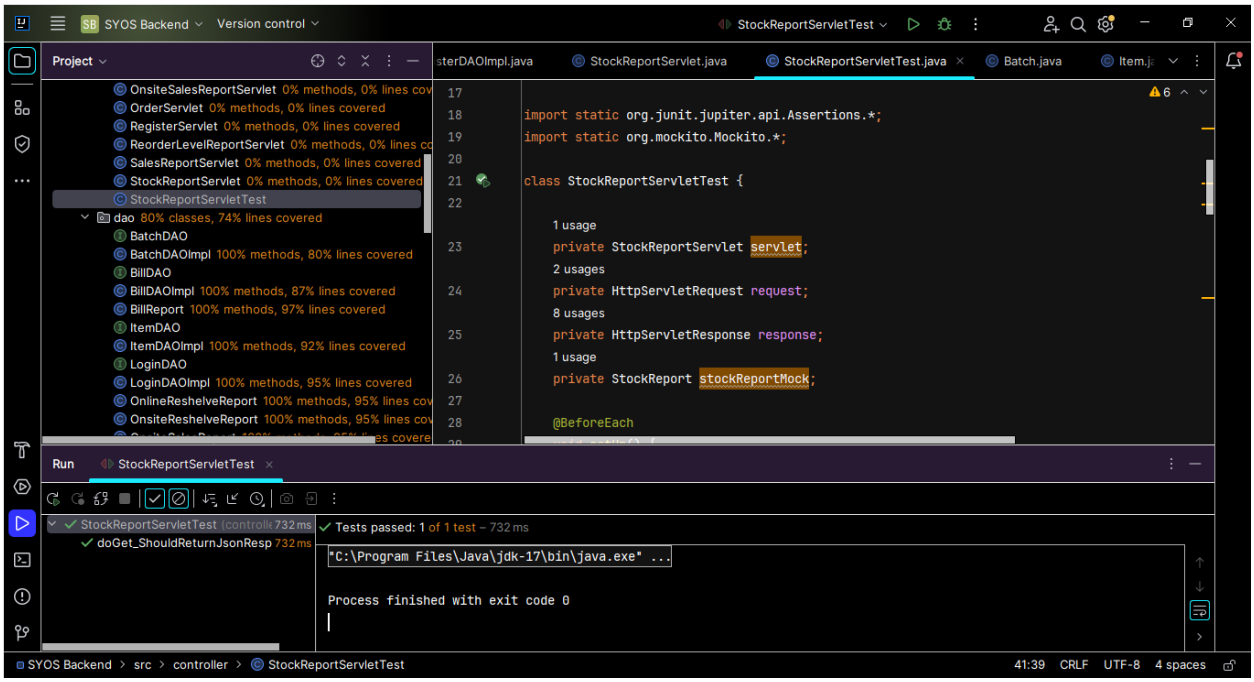

SYOS Backend
Version control
SalesReportTest
Project
RegisterDAOImpl.java
ReorderLevelReport.java
SalesReport.java
SalesReportTest.java
BatchTest
BillDAOImplConcurrencyTest
BillReportTest
BillTest
ItemDAOImplTest
ItemTest
LoginDAOImplTest
OnlineInventoryTest
OnlineReshelveReportTest
OnsiteReshelveReportTest
OnsiteSalesReportTest
OrderTest
SalesReportTest
ShelfInventoryTest
UserTest
WEB-INF
import java.util.List;
import dao.SalesReport;
import org.junit.jupiter.api.Test;
public class SalesReportTest {
@Test
public void testGetSalesReport() {
SalesReport salesReport = new SalesReport();
List<SalesReport.ReportEntry> reportEntries = salesReport.getSalesReport();
Run
SalesReportTest
SalesReportTest 562 ms
testGetSalesReport() 562 ms
Tests passed: 1 of 1 test – 562 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > SalesReportTest
23:20 CRLF UTF-8 4 spaces

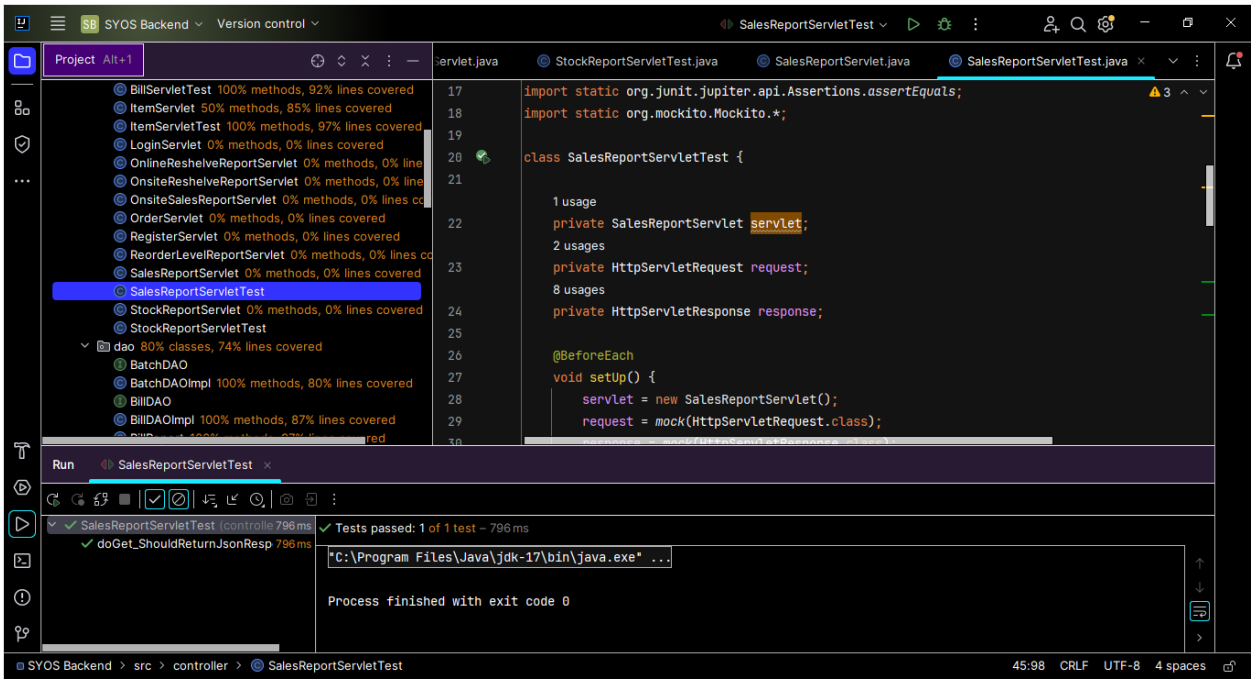

SYOS Backend
Version control
StockReportTest
Project
StockReportTest.java
ReorderLevelReport.java
SalesReport.java
model 14% classes, 15% lines covered
service 66% classes, 4% lines covered
ItemValidator 0% methods, 0% lines covered
OrderProcessingService 100% methods, 60%
StockMovementService
StockMovementServiceImpl 0% methods, 1%
view
javac
Main 0% methods, 0% lines covered
Test
BatchTest
BillDAOImplConcurrencyTest
BillReportTest
BillTest
ItemDAOImplTest
import java.util.List;
import dao.StockReport;
import org.junit.jupiter.api.Test;
public class StockReportTest {
@Test
public void testGenerateStockReport() {
StockReport stockReport = new StockReport();
List<StockReport.StockEntry> entries = stockReport.generateStockReport();
Run
StockReportTest
StockReportTest 540 ms
testGenerateStockReport() 540 ms
Tests passed: 1 of 1 test – 540 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > StockReportTest
14:14 CRLF UTF-8 4 spaces

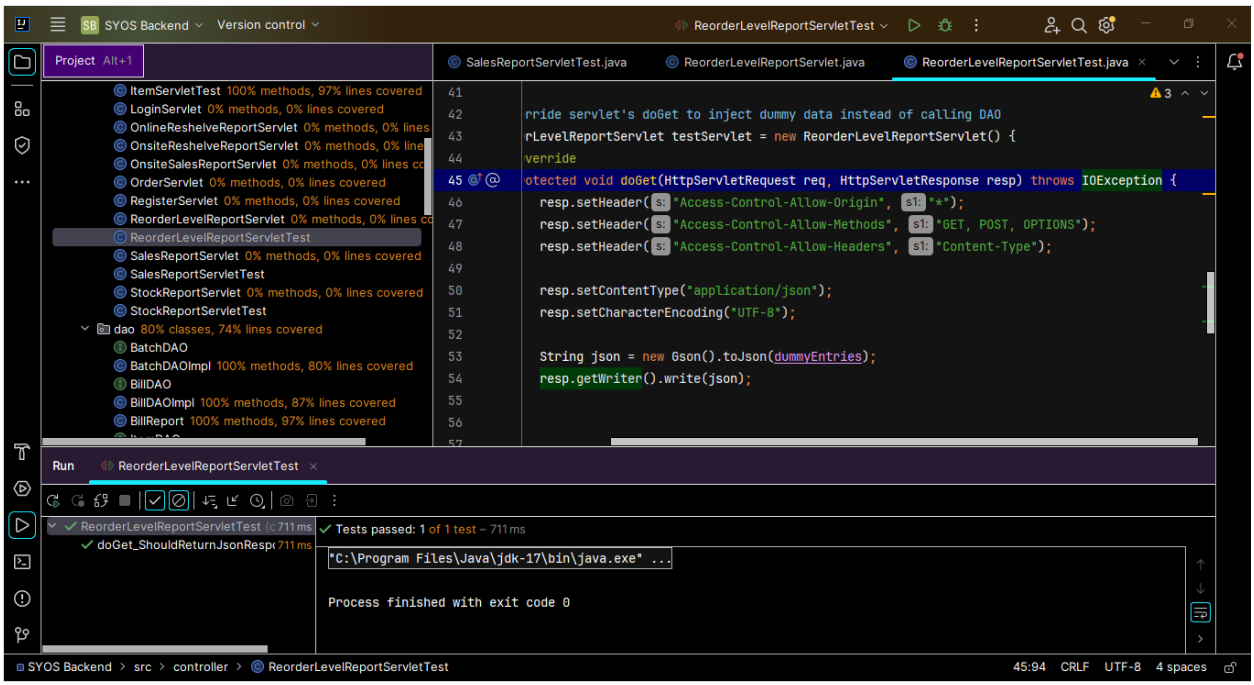

SYOS Backend
Version control
ReorderLevelReportTest
Project
ItemServletTest.java
LoginServlet.java
ReorderLevelReport.java
ReorderLevelReportTest.java
service 100% classes, 10% lines covered
ItemValidator 100% methods, 100% lines covered
OrderProcessingService 100% methods, 100% lines
StockMovementService
StockMovementServiceImpl 0% methods, 1% lines c
view
javac
Main 0% methods, 0% lines covered
Test
BatchTest
BillDAOImplConcurrencyTest
BillReportTest
BillTest
DBConnectionTest
ItemDAOImplTest
ItemTest
ItemValidatorTest
import static org.junit.jupiter.api.Assertions.*;
class ReorderLevelReportTest {
@Test
void testGenerateReorderLevelReport() {
ReorderLevelReport report = new ReorderLevelReport();
List<ReorderLevelReport.ReorderEntry> entries = report.generateReorderLevelReport();
// Basic assertions - adjust based on your test DB data
assertNotNull(entries, message: "Report list should not be null");
Run
ReorderLevelReportTest
ReorderLevelReportTest 534 ms
testGenerateReorderLevelRep 534 ms
Tests passed: 1 of 1 test – 534 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > Test > ReorderLevelReportTest
14:21 CRLF UTF-8 4 spaces


SYOS Backend
Version control
BatchServletTest
Project
BatchServletTest.java
ReorderLevelReportServlet.java
ReorderLevelReport.java
SYOS Backend C:\Program Files\Apache Software Fou
.idea
images
lib
out
src
controller
BatchServlet
BatchServletTest
BillReportServlet
BillServlet
BillServletJavaClient
BillServletTest
ItemServlet
LoginServlet
OnlineReshelveReportServlet
OnsiteReshelveReportServlet
OnsiteSalesReportServlet
import org.mockito.ArgumentCaptor;
import javax.servlet.http.*;
import java.io.PrintWriter;
import java.io.StringWriter;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
public class BatchServletTest {
4 usages
private BatchServlet batchServlet;
13 usages
private HttpServletRequest request;
Run
BatchServletTest
BatchServletTest (controller) 1 sec 908 ms
testValidBatchInsertion() 1 sec 903 ms
testInvalidDateFormat() 5 ms
Tests passed: 2 of 2 tests – 1 sec 908 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
May 24, 2025 4:19:06 PM dao.BatchDAOImpl insertBatch
INFO: Batch inserted successfully:
Process finished with exit code 0
SYOS Backend > src > controller > BatchServletTest
40:66 CRLF UTF-8 4 spaces

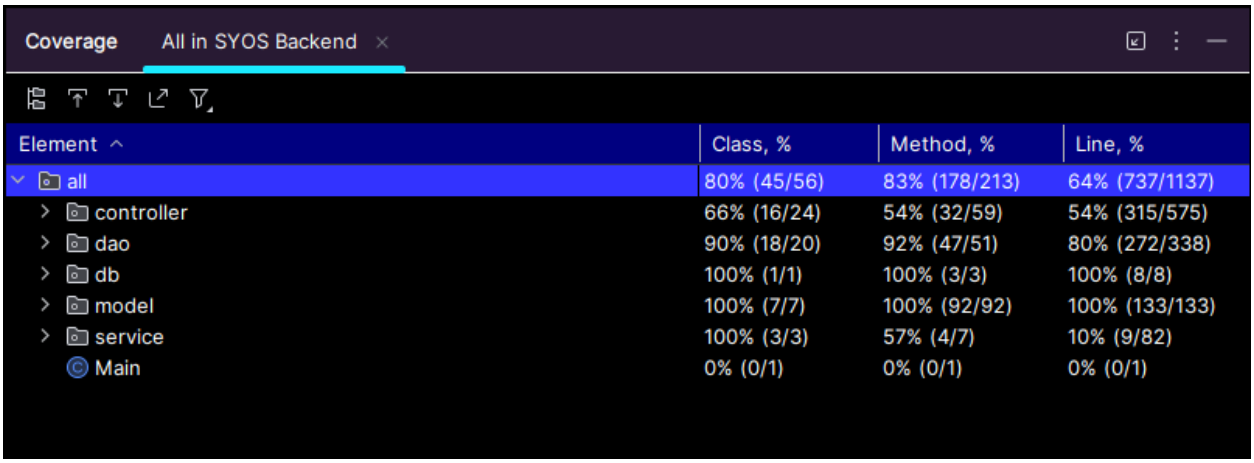

SYOS Backend
Version control
ItemServletTest
Project Alt+1
BillServlet.java
BillServletTest.java
ItemServlet.java
ItemServletTest.java
Item.java
SYOS Backend C:\Program Files\Apache Software Fou
.idea
images
lib
out
src
controller
BatchServlet
BatchServletTest
BillReportServlet
BillServlet
BillServletJavaClient
BillServletTest
ItemServlet
ItemServletTest
LoginServlet
OnlineReshelveReportServlet
OnsiteReshelveReportServlet
import java.util.Arrays;
import java.util.List;
public class ItemServletTest {
7 usages
private ItemServlet servlet;
6 usages
private ItemDAO itemDAOMock;
3 usages
private Gson gson;
@BeforeEach
public void setUp() {
// Create a mock ItemDAO
Run
ItemServletTest
ItemServletTest (controller) 888 ms
testGetPriceNotFound() 855 ms
testGetAllItems() 22 ms
testGetPriceFound() 3 ms
testInvalidAction() 2 ms
testValidateCode() 6 ms
Tests passed: 5 of 5 tests – 888 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > src > controller > ItemServletTest > testGetAllItems
124:6 CRLF UTF-8 4 spaces

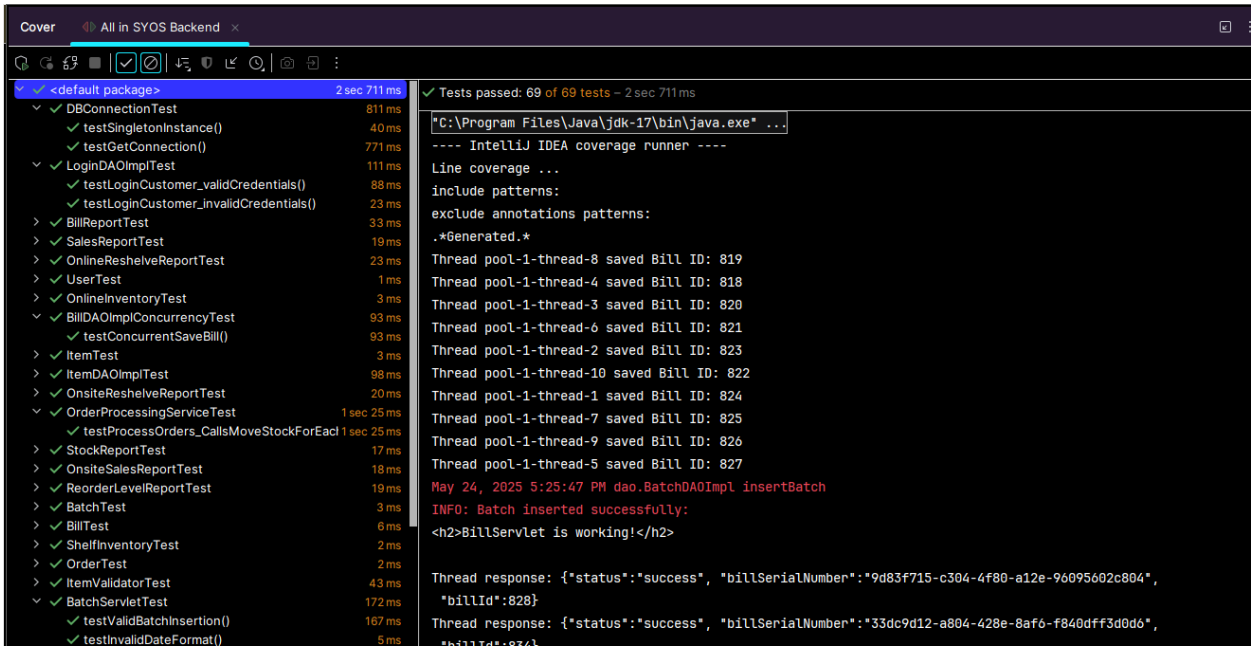

SYOS Backend
Version control
StockReportServletTest
Project
OnsiteSalesReportServlet 0% methods, 0% lines cov
OrderServlet 0% methods, 0% lines covered
RegisterServlet 0% methods, 0% lines covered
ReorderLevelReportServlet 0% methods, 0% lines co
SalesReportServlet 0% methods, 0% lines covered
StockReportServlet 0% methods, 0% lines covered
StockReportServletTest
dao 80% classes, 74% lines covered
BatchDAO
BatchDAOImpl 100% methods, 80% lines covered
BillDAO
BillDAOImpl 100% methods, 87% lines covered
BillReport 100% methods, 97% lines covered
ItemDAO
ItemDAOImpl 100% methods, 92% lines covered
LoginDAO
LoginDAOImpl 100% methods, 95% lines covered
OnlineReshelveReport 100% methods, 95% lines cov
OnsiteReshelveReport 100% methods, 95% lines cov
sterDAOImpl.java
StockReportServlet.java
StockReportServletTest.java
Batch.java
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
class StockReportServletTest {
1 usage
private StockReportServlet servlet;
2 usages
private HttpServletRequest request;
8 usages
private HttpServletResponse response;
1 usage
private StockReport stockReportMock;
@BeforeEach
Run
StockReportServletTest
StockReportServletTest (controll 732 ms
doGet_ShouldReturnJsonResp 732 ms
Tests passed: 1 of 1 test – 732 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend > src > controller > StockReportServletTest
41:39 CRLF UTF-8 4 spaces


SYOS Backend
Version control
SalesReportServletTest
Project Alt+1
StockReportServletTest.java
SalesReportServlet.java
SalesReportServletTest.java
BillServletTest 100% methods, 92% lines covered
ItemServlet 50% methods, 85% lines covered
ItemServletTest 100% methods, 97% lines covered
LoginServlet 0% methods, 0% lines covered
OrderServlet 0% methods, 0% lines covered
RegisterServlet 0% methods, 0% lines covered
SalesReportServlet 0% methods, 0% lines covered
SalesReportServletTest
StockReportServlet 0% methods, 0% lines covered
StockReportServletTest
dao 80% classes, 74% lines covered
BatchDAO
BatchDAOImpl 100% methods, 80% lines covered
BillDAO
BillDAOImpl 100% methods, 87% lines covered
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.*;
class SalesReportServletTest {
1 usage
private SalesReportServlet servlet;
2 usages
private HttpServletRequest request;
8 usages
private HttpServletResponse response;
@BeforeEach
void setUp() {
servlet = new SalesReportServlet();
request = mock(HttpServletRequest.class);
Run
SalesReportServletTest
Tests passed: 1 of 1 test – 796 ms
doGet_ShouldReturnJsonResp 796 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend › src › controller › SalesReportServletTest
45:98 CRLF UTF-8 4 spaces

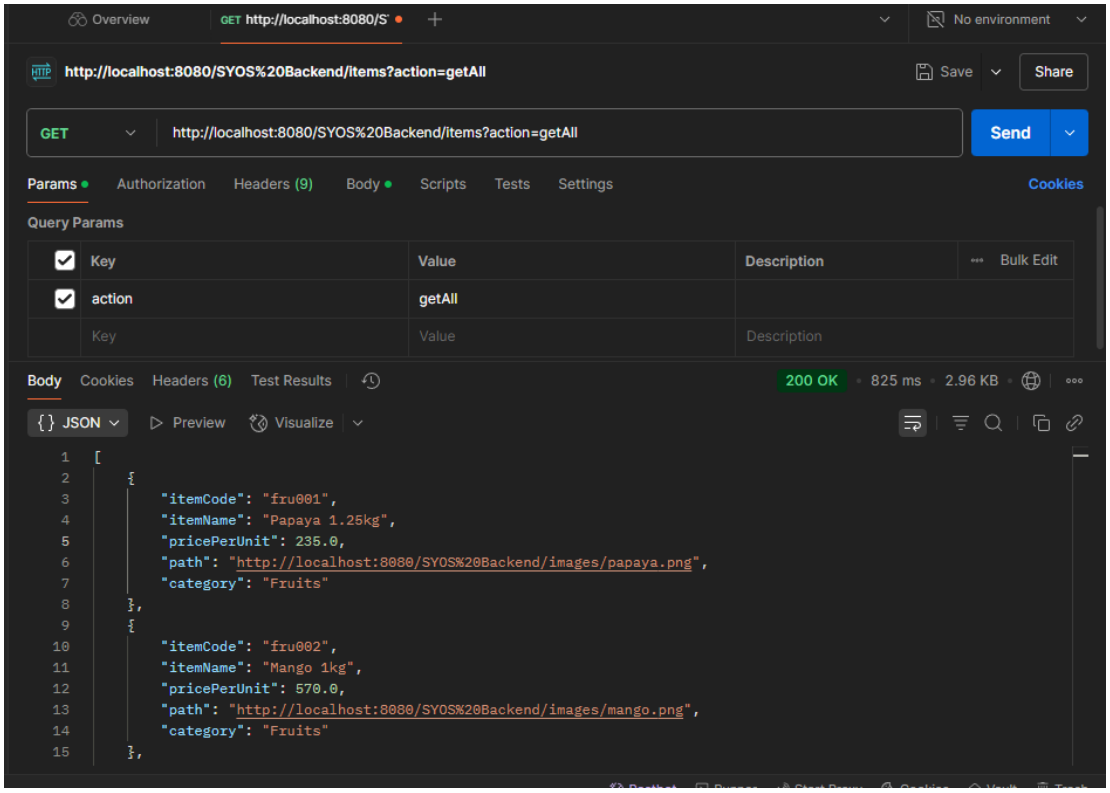

SYOS Backend
Version control
ReorderLevelReportServletTest
Project Alt+1
SalesReportServletTest.java
ReorderLevelReportServlet.java
ReorderLevelReportServletTest.java
ItemServletTest 100% methods, 97% lines covered
LoginServlet 0% methods, 0% lines covered
OrderServlet 0% methods, 0% lines covered
RegisterServlet 0% methods, 0% lines covered
ReorderLevelReportServlet 0% methods, 0% lines covered
ReorderLevelReportServletTest
SalesReportServlet 0% methods, 0% lines covered
SalesReportServletTest
StockReportServlet 0% methods, 0% lines covered
StockReportServletTest
dao 80% classes, 74% lines covered
BatchDAO
BatchDAOImpl 100% methods, 80% lines covered
BillDAO
BillDAOImpl 100% methods, 87% lines covered
BillReport 100% methods, 97% lines covered
rride servlet's doGet to inject dummy data instead of calling DAO
rLevelReportServlet testServlet = new ReorderLevelReportServlet() {
verride
otected void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
resp.setHeader(s: "Access-Control-Allow-Origin", s1: "*");
resp.setHeader(s: "Access-Control-Allow-Methods", s1: "GET, POST, OPTIONS");
resp.setHeader(s: "Access-Control-Allow-Headers", s1: "Content-Type");
resp.setContentType("application/json");
resp.setCharacterEncoding("UTF-8");
String json = new Gson().toJson(dummyEntries);
resp.getWriter().write(json);
Run
ReorderLevelReportServletTest
Tests passed: 1 of 1 test – 711 ms
doGet_ShouldReturnJsonResp 711 ms
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Process finished with exit code 0
SYOS Backend › src › controller › ReorderLevelReportServletTest
45:94 CRLF UTF-8 4 spaces

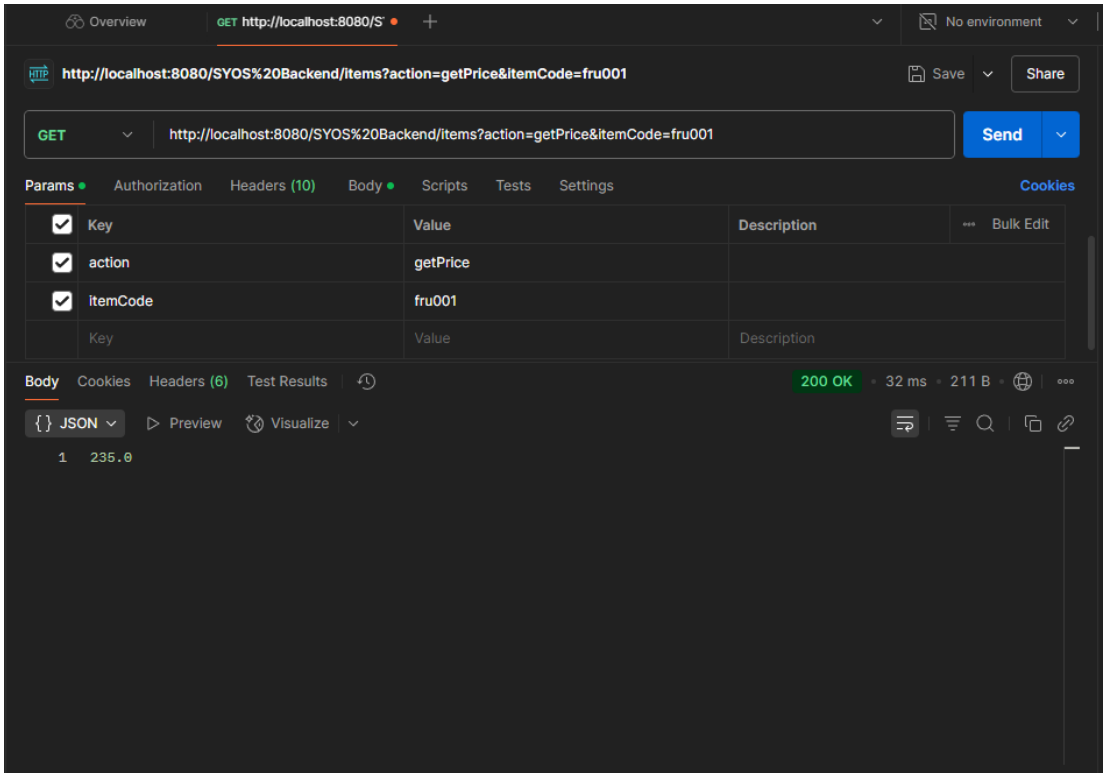
6.2 Test Coverage

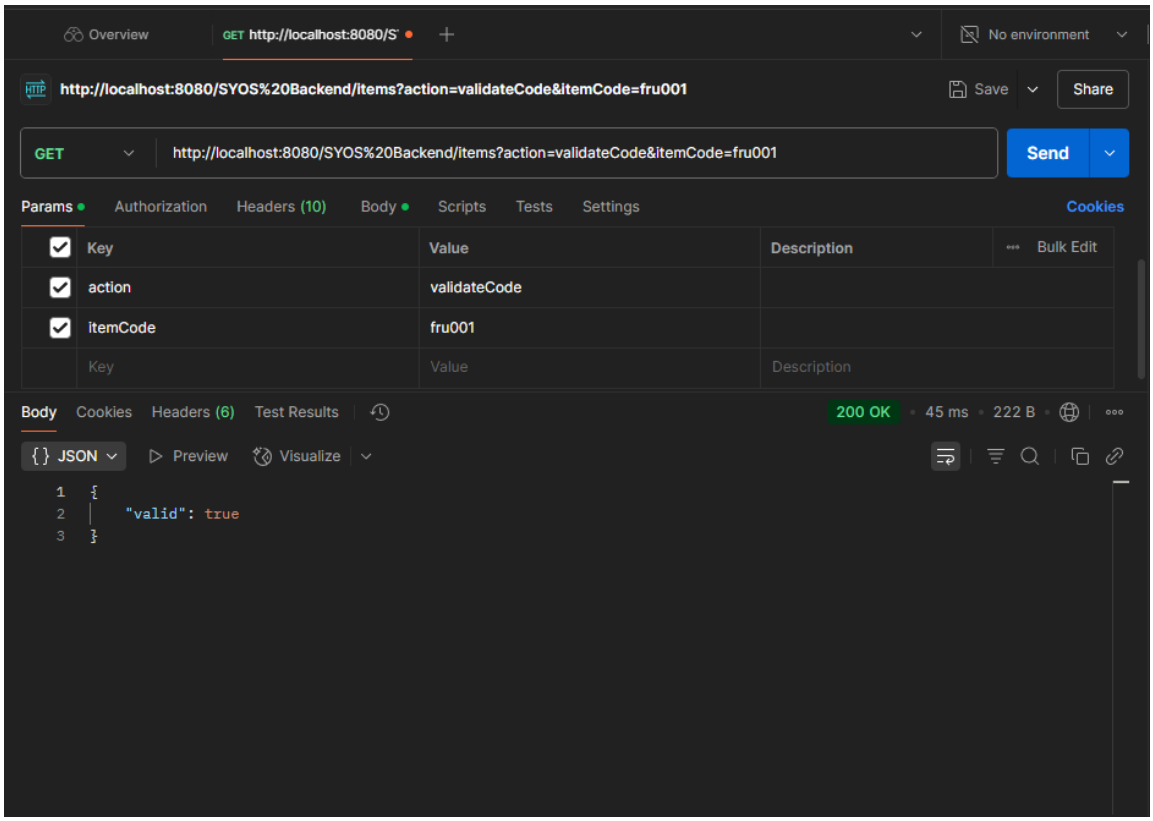


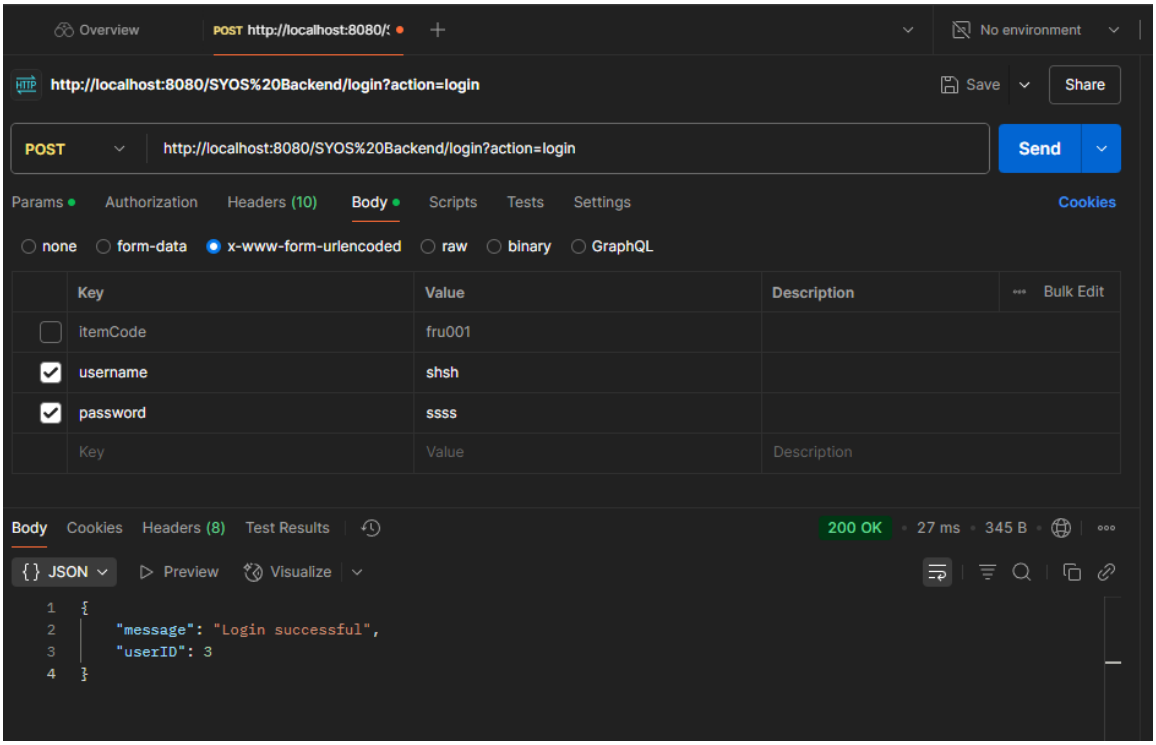


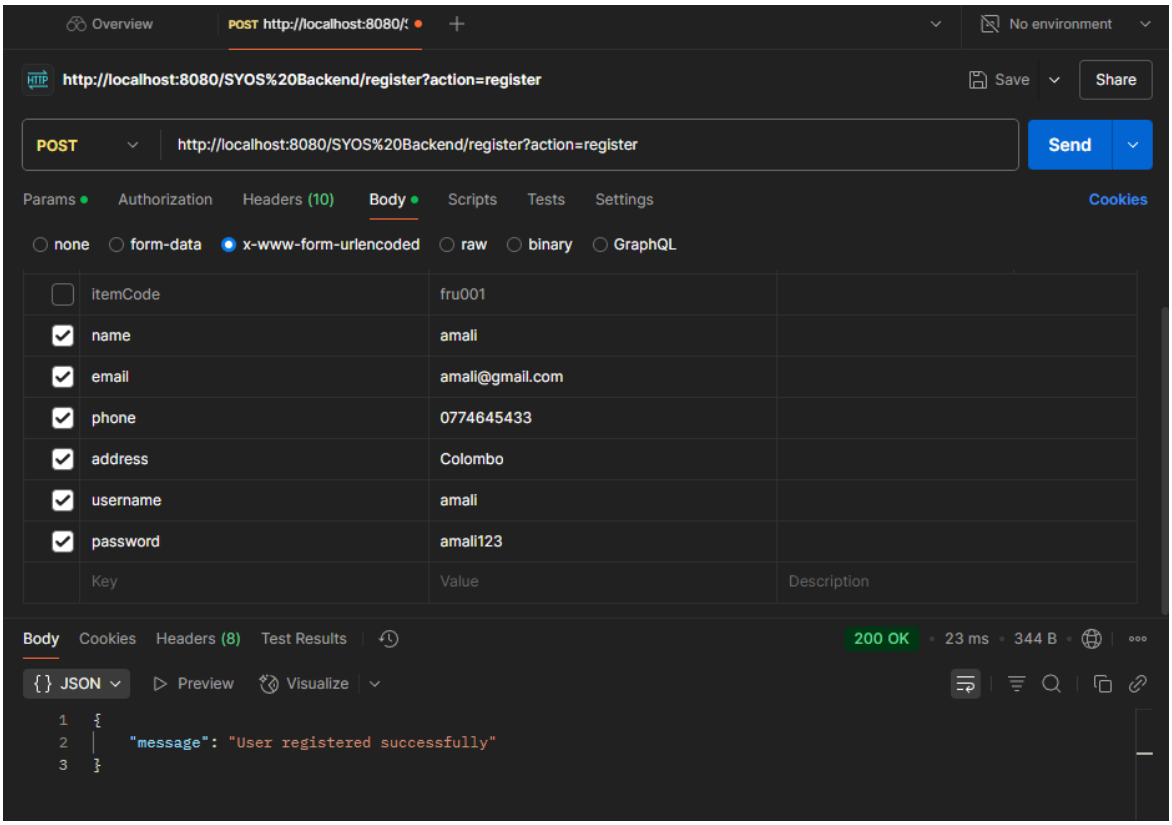
Chapter 7 – Postman Tests

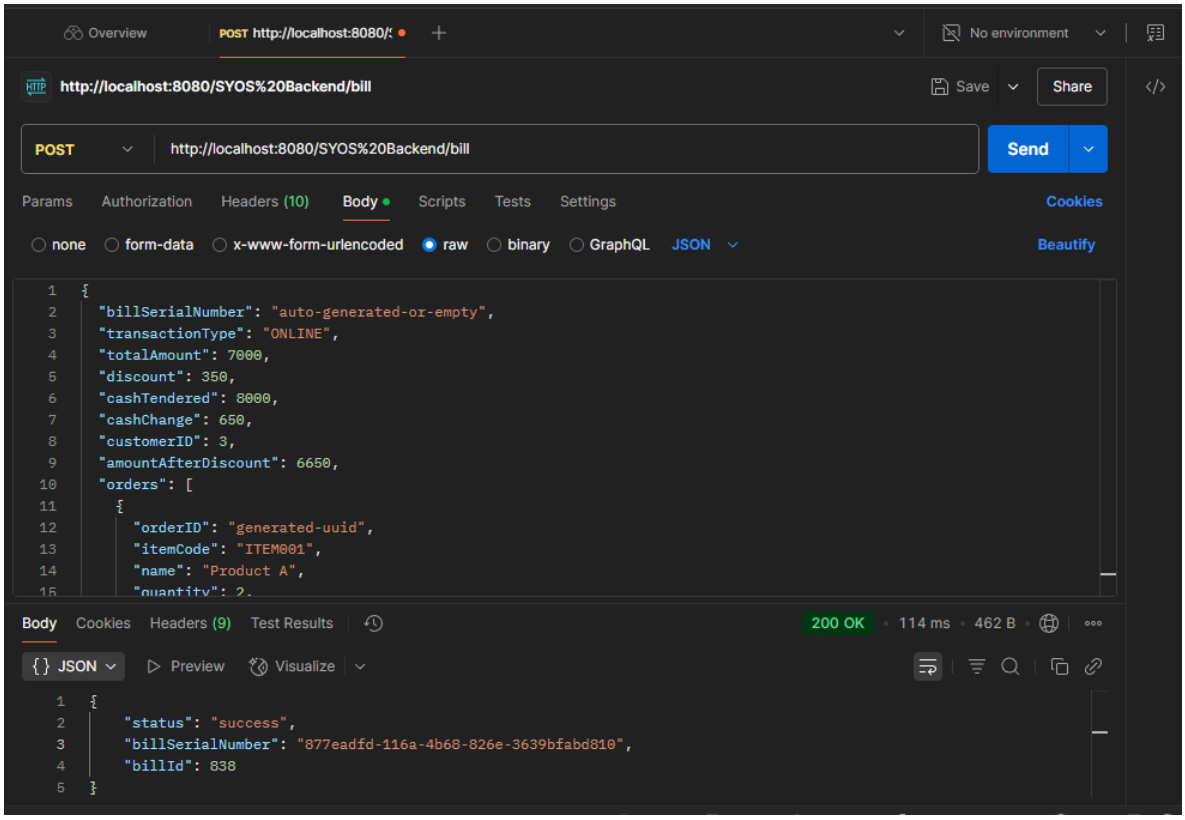





Overview
GET http://localhost:8080/S
No environment
http://localhost:8080/SYOS%20Backend/items?action=validateCode&itemCode=fru001
Save
Share
GET
http://localhost:8080/SYOS%20Backend/items?action=validateCode&itemCode=fru001
Send
Params
Authorization
Headers (10)
Body
Scripts
Tests
Settings
Cookies
Key
Value
Description
Bulk Edit
action
validateCode
itemCode
fru001
Key
Value
Description
Body
Cookies
Headers (6)
Test Results
200 OK
45 ms
222 B
JSON
Preview
Visualize
{
"valid": true
}


Overview
POST http://localhost:8080/
No environment
http://localhost:8080/SYOS%20Backend/login?action=login
Save
Share
POST
http://localhost:8080/SYOS%20Backend/login?action=login
Send
Params
Authorization
Headers (10)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
Description
Bulk Edit
itemCode
fru001
username
shsh
password
ssss
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
200 OK
27 ms
345 B
JSON
Preview
Visualize
{
"message": "Login successful",
"userID": 3
}


Overview
POST http://localhost:8080/
No environment
http://localhost:8080/SYOS%20Backend/register?action=register
Save
Share
POST
http://localhost:8080/SYOS%20Backend/register?action=register
Send
Params
Authorization
Headers (10)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
itemCode
fru001
name
amali
email
amali@gmail.com
phone
0774645433
address
Colombo
username
amali
password
amali123
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
200 OK
23 ms
344 B
JSON
Preview
Visualize
{
"message": "User registered successfully"
}


Overview
POST http://localhost:8080/
No environment
http://localhost:8080/SYOS%20Backend/bill
Save
Share
POST
http://localhost:8080/SYOS%20Backend/bill
Send
Params
Authorization
Headers (10)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"billSerialNumber": "auto-generated-or-empty",
"transactionType": "ONLINE",
"totalAmount": 7000,
"discount": 350,
"cashTendered": 8000,
"cashChange": 650,
"customerID": 3,
"amountAfterDiscount": 6650,
"orders": [
{
"orderID": "generated-uuid",
"itemCode": "ITEM001",
"name": "Product A",
"quantity": 2,
Body
Cookies
Headers (9)
Test Results
200 OK
114 ms
462 B
JSON
Preview
Visualize
{
"status": "success",
"billSerialNumber": "877eadfd-116a-4b68-826e-3639bfabd810",
"billId": 838
}

Chapter 8 - Conclusion

The SYOS Billing System successfully automates the traditional billing process by integrating efficient stock management, cash handling and transaction recording within a well structured software architecture. By adopting a three-tier client-server model combined with MVC principles and SOLID design guidelines, the system achieves a clear separation of concerns that enhances maintainability, flexibility and scalability. The implementation of concurrency control at both server and client sides ensures that the system can handle multiple simultaneous users effectively by providing a smooth and responsive experience. Although the system demonstrates solid architectural foundations and practical design patterns, ongoing attention to managing layer boundaries, improving real-time synchronization and optimizing concurrency handling will be critical to maintain performance and reliability as usage scales. Overall, SYOS lays a strong groundwork for a robust billing solution that supports both instore and online sales with potential for further enhancements to meet evolving business and technical requirements.

Chapter 9 – Acknowledgement

I would like to express my heartfelt gratitude to everyone who supported and encouraged me throughout the successful completion of this project. A special thank you goes to my lecturer for the continuous guidance, insightful feedback and motivation that helped me stay focused and improve at every stage. I am also deeply grateful to my class members for their support. Furthermore, I appreciate my family and friends for their unwavering support, patience and encouragement, which played a vital role in helping me stay positive and driven. Their contributions, both direct and indirect, were truly invaluable in making this project a success.