**1.Project Background and Description:**

The goal of this project is to develop a comprehensive software solution for a medical store that facilitates inventory management, medicine tracking, and invoice generation. The final vision of the project is to create a user-friendly system that streamlines the daily operations of a medical store by enabling salespersons to manage medicine stock, track customer purchases, and generate accurate invoices efficiently. This will reduce manual work, minimize errors, and ensure faster service for customers**.**

**Commercial Use**: This software will be beneficial to any medical store by automating several key processes. It will help optimize medicine stock levels, speed up billing, and enhance customer service by providing a history of purchases. The software will also minimize human errors in managing inventory, ensuring better accuracy in stock and financial records.

**Database Description: The database will include five primary entities:**

1. Company: This entity will store details of medicine manufacturers, including their license numbers and names.
2. Medicine: This will store the details of each medicine, such as its name, manufacturing license number, manufacturing and expiry dates, price, batch number, and quantity in stock.
3. Customer: Stores customer data, including phone numbers, names, and addresses.
4. Invoice: Records information about sales transactions, including the invoice date, customer phone number, and total amount.
5. Invoice\_Medicine: A junction table that links invoices to the medicines purchased, including the quantity of each medicine bought.

🡪So OUR base code is built using **Java** with a focus on implementing key object-oriented principles. The Base code includes six main classes: Customer, Medicine, Invoice, MedicineController, InvoiceController, and DatabaseAccess.

* **Class Structure:**
* **Customer and Medicine Classes**: These classes are independent of each other, meaning they don’t directly interact. The Customer class handles customer-related information (e.g., phone number, address), while the Medicine class manages medicine details (e.g., name, price, expiration date).
* **Invoice Class**: The Invoice class is dependent on both Customer and Medicine classes. It combines customer data and medicine data to create an invoice, which is used to track purchases and calculate totals.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

* **Controller Classes:**
* **MedicineController Class**: This controller manages operations related to medicine, such as adding or deleting a medicine. It connects to the server to manage requests and updates the medicine information as needed.

A screenshot of a computer program

Description automatically generated

* **InvoiceController Class**: This class is responsible for adding and generating invoices. It collects customer and medicine data, creates an invoice, and passes it to the database for storage.

A screenshot of a computer program

Description automatically generated

* **DatabaseAccess Class:**

The DatabaseAccess class handles the interaction with the database. It is responsible for performing **CRUD operations** (Create, Read, Update, Delete) based on requests from the controllers. For example, it adds new medicine records, updates customer details, or saves invoices to the database as required.

A screenshot of a computer program

Description automatically generated

* **We are Going to follow the following Patterns**:
* Controller-Service structure with MedicineController and InvoiceController handling requests, while DatabaseAccess manages database tasks.
* Dependency injection using @Autowired for flexible class management.
* DatabaseAccess handles CRUD operations.

🡪The code follows Java conventions, using camelCase for names, private fields with public getters/setters, and simple in-line comments to keep things clear..

* (FOR THE DETAILED BASECODE YOU CAN REFER TO OUR GITHUB URL: <https://github.com/Jay-Ser-dev/MedicalStore.git>).

**2. Project Scope:**

**Team Members and Roles:**

* Katyayani Trivedi: As the project leader, I am responsible for submitting all deliverables on time. I also outline the database structure, ensuring that the code is user-friendly and styled according to best practices. Additionally, I check the code frequently to ensure it meets the necessary quality standards before submission.
* Jay Patel: Jay is responsible for writing the core source code as per the outlined structure. He implements the functionality discussed during team meetings and ensures that the code is delivered to me on time for submission. Jay also collaborates closely with me to make sure all functionalities are working as intended.

**Technical Scope:**

The project will involve the following technical components:

* **User Interface**: A web-based application built using HTML and CSS for the front-end, with a clean and user-friendly layout to ensure ease of use for medical store employees. The interface will allow users to manage medicines, check stock, generate invoices, and store customer data efficiently.
* **Back-End**: The back-end is developed using Java and Spring Boot. It handles database operations, business logic, and API requests. The back-end ensures seamless communication between the user interface and the database.
* **Database**: We use the H2 Database to store information about medicines, customers, and invoices. The database design is centered around five key entities: Company, Medicine, Customer, Invoice, and Invoice Medicine.

**Project Completion:**

We will consider the project complete when the following functionalities are fully implemented and working as expected:

* Medicine stock can be added, updated, and checked in the system.
* The system generates invoices for customers.
* Customer data is stored and retrieved for future purchases.
* The interface is intuitive and responsive, allowing users to easily navigate and perform tasks.
* All components pass testing and meet the requirements outlined in our project documentation.

**3. High-Level Requirements**

The new medical store software must include the following features:

* Medicine Inventory Management: Ability for the salesperson to add new medicine stock and update quantities.
* Medicine Search: Ability for the salesperson to check whether a specific medicine is available in stock.
* Invoice Generation: Ability to generate invoices for customers including the purchased medicines, price, and total amount.
* Customer Data Management: Ability to store customer details (phone number, address) and retrieve past purchases.

**4. Implementation Plan**

* GitHub Repository: <https://github.com/Jay-Ser-dev/MedicalStore.git>
* We decided to check our code at least once a week. This ensures that the latest changes are shared with the entire, allowing for effective collaboration and minimizing conflicts. It will also help in tracking progress and maintaininga clear version history.

The project will have a well-organized directory structure to keep all files accessible:

* **/src**: Contains all Java source code files.
* **/assets**:
  + **/text**: Stores any text files related to project documentation, such as notes and guidelines.
  + **/uml**: Holds UML diagrams created using Visual Paradigm (VP) for visualizing system design and architecture.
* **/resources**: For configuration files like application.properties.
* **/templates**: Stores Thymeleaf HTML templates.
* **/static**: Contains static files like CSS and images.

We were going to use following coding standard:

1**. Java Naming Conventions**: Adhere to standard naming conventions:

* + Class names should be in Pascal Case (e.g., Medicine, Customer).
  + Method and variable names should be in camelCase (e.g., getMedicineName, totalAmount).

2. **Consistent Formatting**: Ensuring consistent indentation, spacing, and line breaks to enhance code readability. Use Eclipse’s built-in formatting tools to maintain consistency.

3. **Version Control Best Practices**: Writing clear and concise commit messages that describe the changes made. Using branches for new features or bug fixes to keep the main branch stable.

4. **Commenting Strategy:** In our project, We will ensure that comments are used consistently throughout the codebase to describe the purpose and functionality of classes, methods, and critical logic. Comments will be updated as needed to reflect any changes in the code to ensure they remain accurate and relevant.

We are going to use following Tools in our group project:

1. Use Eclipse for development, which provides robust support for Java and the Spring Boot framework.

2. Utilize the H2 Database for development and testing, allowing for quick setups and integration.

3. Dependency Management: Use Maven to manage project dependencies, including:

* Spring web
* Thymeleaf
* Spring data JDBC
* Spring DevTools
* H2
* Lombok

4. Version Control System: Employ Git for version control, with GitHub as the hosting platform for collaboration and code sharing.

5. UML Tools: Use Visual Paradigm (VP) for creating UML diagrams to represent system design and architecture visually.

**5.Design Considerations:**

Here’s how your code reflects three key object-oriented principles:

1. Encapsulation

Encapsulation ensures that the internal details of a class are hidden, and only necessary information is exposed through public methods or properties. The Invoice class demonstrates this principle as follows:

Private Fields:

The fields date, customer, medicineList, and totalAmount in the Invoice class are marked as private, meaning they cannot be directly accessed from outside the class. This restricts external classes from modifying these fields directly.

Controlled Access via Methods:

The getTotalAmount() method is used to calculate the total amount from the medicineList. By keeping it private, only the class itself can use this method, ensuring that the calculation logic remains e

ncapsulated.

2. Delegation

Delegation refers to passing responsibilities from one class to another to achieve a specific task, promoting separation of concerns.

* Database Access Delegation:

In the MedicineController and InvoiceController, when a new medicine is added or an invoice is generated, the actual database operations (such as adding medicine or customer details) are delegated to the DatabaseAccess class. This is done using the da.addMedicine(), da.addCustomer(), and da.generateInvoice() methods. By delegating these tasks, the controllers are focused on handling web requests, while the database-related tasks are handled by the DatabaseAccess class.

* Invoice Generation Delegation:

In InvoiceController, the task of generating the invoice details is delegated to the Invoice class. The controller creates an Invoice object, sets its customer and medicine list via setters, and calls da.generateInvoice(invoice) to handle the actual saving of the invoice in the database.

3. Flexibility/Maintainability

Our code is structured in a way that allows easy modifications and adaptability to future changes. This flexibility is supported by several design decisions:

* Use of Dependency Injection:

The use of @Autowired in the controllers (MedicineController and InvoiceController) to inject the DatabaseAccess class provides flexibili

Separation of Concerns:

The code separates the responsibilities between controllers, models, and data access layers, making the system more maintainable. For instance, the Invoice class handles invoice-related details, while DatabaseAccess focuses on database interactions. This modular approach ensures that changes in one part of the system (e.g., database schema) don't affect other parts directly.

Modular Method Design:

In the Invoice class, the getTotalAmount() method is designed to calculate the total based on the medicineList. If the calculation logic needs to change (e.g., adding taxes), user can modify this method without affecting the rest of the class. Similarly, the InvoiceController separates tasks like adding medicine and generating invoices into distinct methods (addMedicine, generateInvoice), making the code more manageable.