

INTRODUCTION

Jewellery has been an important part of human civilization for thousands of years, signifying culture, tradition, status, and wealth. In modern times, jewellery has become a combination of fashion and investment. Jewellery businesses deal with a wide variety of products—rings, necklaces, bangles, earrings, gemstones, and customized pieces. Managing all these products manually is not only inefficient but also extremely risky due to the high-value nature of the goods.

With the growth of businesses and customer expectations, it has become essential to adopt technology-driven solutions. Traditionally, many jewellery shop owners rely on manual registers, spreadsheets, and human memory to maintain records of purchases, sales, and stock. However, as the business scales, these methods become error-prone and lead to loss of data, revenue, and customer trust.

1.1 Background

A Jewellery Management System (JMS) solves these problems by providing an integrated software environment to handle operations such as inventory management, billing, customer handling, reporting, and staff access. Such a system ensures secure, fast, and accurate processing, while also improving decision-making through data analysis.

With the rapid development in technology and increasing digital adoption, there is an urgent need for the automation of business processes. A Jewellery Management System is a software application that simplifies and automates the operations of a jewellery shop or company. This project aims to build such a system to efficiently manage stock, customers, sales, and billing, minimizing manual tasks and errors.

Moreover, such a system brings professionalism to the business. Printed invoices with branding and GST details make the customer experience smoother. Automatic stock updates ensure that employees always know what items are available or low in quantity. Customer details can be stored and reused to provide better, faster service or offer discounts based on purchase history.

A digital JMS also supports long-term planning. It helps in tracking product popularity, seasonal trends, and customer preferences. Business owners can generate monthly or yearly reports, analyze sales performance, and make informed decisions about which categories to promote or restock.

In addition to improving efficiency, digital transformation also ensures business continuity. With regular backups and secure login access, a JMS protects data from accidental loss, theft, or disasters. It also minimizes the risk of internal misuse by employees, as each action can be tracked through audit logs.

In summary, as the jewellery industry evolves and grows more competitive, it becomes vital for businesses to upgrade their operations using reliable and efficient digital systems. The Jewellery Management System not only solves current operational problems but also prepares the business for future scalability, integration with e-commerce, mobile apps, and customer loyalty systems. It transforms a traditional shop into a modern, efficient, and customer-friendly business that can thrive in today's fast-paced market.

1.2 Objective of the Project

The core aim of this project, the *Jewellery Management System (JMS)*, is to design and develop a digital platform that enhances the operational efficiency of a jewellery business by automating its daily functions. The traditional manual processes involved in jewellery businesses are time-consuming, error-prone, and inefficient, especially as the customer base and product catalog grow. With the increasing demand for personalized service, accurate billing, and real-time inventory tracking, it has become necessary to adopt a software-based solution. This system seeks to provide exactly that—a centralized and secure platform to manage all core operations of a jewellery shop.

The first and foremost objective of this system is inventory automation. Jewellery stores usually manage hundreds of unique items, each with distinct specifications such as type, metal, weight, price, design, and availability. The JMS helps in maintaining an up-to-date digital record of each product. Whenever a sale is made, the system automatically deducts the item from the stock. This removes the need for manual entries and minimizes errors related to stock management. It also allows for easier tracking of in-demand items and restocking of fast-moving products.

Another important objective is to create an efficient and accurate billing mechanism. The JMS allows employees to quickly generate invoices with proper item-wise details, tax calculations (such as GST), and discounts. This not only speeds up customer service but also ensures that every transaction is correctly recorded and accounted for. Moreover, customers receive professional-looking printed invoices which include brand information, price breakdowns, and contact details, improving the brand image and trust factor of the business.

A third key objective is customer information management. In today's market, customer retention is just as important as acquiring new ones. The JMS stores

customer data, including name, phone number, address, and past purchase history. This allows the business to offer personalized services such as loyalty discounts, product recommendations, birthday offers, or festive sale alerts. Maintaining these relationships builds long-term customer trust and increases the chances of repeat purchases.

The project also aims to implement role-based access control, which ensures that different levels of users (e.g., admin, cashier, manager, and sales staff) have access to only those parts of the system which are relevant to their duties. For example, only an admin can modify stock details, while a cashier can generate bills and view sales records. This protects sensitive data and prevents unauthorized changes or misuse.

Another major objective is to generate real-time reports and analytics. Business owners often need to review performance trends to make informed decisions. The JMS can generate daily, weekly, or monthly sales reports, stock status, most sold items, least performing categories, customer activity, and more. These reports help in strategic planning, like offering discounts on slow-moving items or investing more in high-demand categories.

Furthermore, the system seeks to improve data security and reliability. All data is stored digitally and can be backed up regularly to prevent data loss due to hardware failure or accidental deletion. Features like password protection, audit logs, and restricted access ensure that only authorized personnel can access and modify the data.

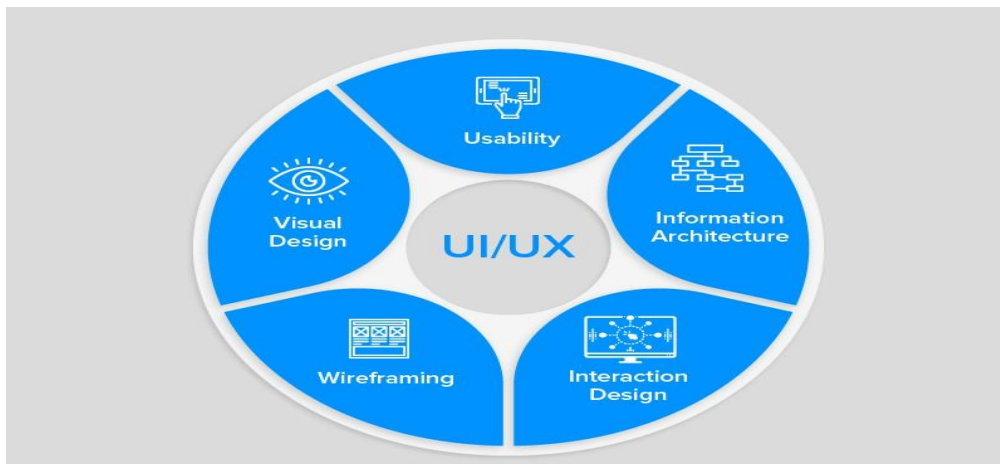


Fig. 1.1:Key Aspects of UI/UX Design

1.3 Scope of the Project

The scope of the *Jewellery Management System (JMS)* defines the boundaries, functionalities, and potential areas of implementation covered by the project. It outlines what the system will do, who will use it, and how it will add value to the business. A well-defined scope ensures that the project remains focused and meets its objectives within time and budget constraints.

This Jewellery Management System is intended primarily for small to mid-sized jewellery retailers who manage a diverse inventory of gold, silver, platinum, diamond, and gemstone-based ornaments. These businesses often deal with customized items, a wide customer base, and high-value transactions, making it essential to have a reliable, secure, and user-friendly software system to manage their operations.



Fig. 1.2:Project Scope

Functional Scope:

1. Product Catalog and Inventory Management

The system will allow the admin to maintain a detailed catalogue of all jewellery items, including product ID, name, category (ring, bangle, necklace, etc.), weight, purity (e.g., 22K, 18K), price, stock quantity, and additional description. Any time a sale is made, the system will automatically update the stock. This ensures real-time inventory management and reduces the risk of manual errors.

2. Customer Management

The system stores customer details including name, contact information, address, and purchase history. This helps in building a customer database for future reference, loyalty rewards, or marketing campaigns such as festive discounts or personalized recommendations.

3. Billing and Invoicing

JMS allows sales executives or cashiers to generate itemized bills with product details, quantity, discounts, tax (like GST), and total amount. Invoices will be printable and exportable in PDF format for record-keeping. The system also provides functionality to handle returns, exchanges, and apply custom discounts.

4. User and Role Management

Different user roles will be defined, such as Admin, Sales Executive, and Manager. Admins have full control over the system including user creation, backup management, and data modification. Sales staff will only have permission to view and perform billing operations. Role-based access enhances security and prevents unauthorized activities.

5. User Interface (UI) and Usability

The system will have a user-friendly and intuitive interface so that even non-technical users can operate it smoothly. The UI will be designed with proper layout, form validation, and quick access to important features like billing, stock update, and report generation.

6. Reports and Analytics

The system will provide various reports like daily/weekly/monthly sales, current stock status, top-selling items, and customer purchase trends. These reports can help the business owner take strategic decisions related to purchasing, pricing, and marketing.

1.4 Problem Definition

In the rapidly evolving business environment, the jewellery industry still faces several long-standing operational challenges, especially in small to medium-sized enterprises where digital solutions are not yet fully adopted. Despite dealing with high-value goods and complex inventory, many jewellery stores still rely on manual or semi-digital processes for managing their business operations. These practices introduce a variety of issues such as errors, inefficiencies, data loss, and poor customer service. The *Jewellery Management System (JMS)* aims to resolve these specific problems through automation and smart business logic.

One of the most critical issues faced by jewellery businesses is inventory mismanagement. Jewellery shops typically have hundreds or thousands of unique products, each with varying weights, metals, designs, and prices. Without a proper system in place, it becomes extremely difficult to track which items are in stock, which have been sold, and which need to be reordered. Manual entries in books or spreadsheets often lead to duplication, loss of data, or outdated information. This can result in stockouts during peak seasons, over-ordering, or even theft going unnoticed.

Another major problem is billing inaccuracies. During busy hours, sales staff may make calculation errors or forget to apply correct tax (like GST), discounts, or customer-specific deals. Incomplete or incorrect billing not only affects the customer experience but also leads to compliance issues and revenue loss. Moreover, without a centralized system, tracking each sale, calculating total income, or reviewing returns/exchanges becomes an extremely difficult task. Customer record management is yet another major challenge. In traditional systems, customers are either not recorded or only partially documented in notebooks or spreadsheets. There is no organized way to store and retrieve customer purchase history, contact details, or preferences. This results in missed opportunities for repeat business, loyalty programs, or personalized offerings. Furthermore, during disputes or return scenarios, the lack of historical data makes verification difficult and unprofessional.

Data security and record loss is a serious concern when business operations are handled manually or semi-digitally. Important records such as sales bills, inventory logs, and customer transactions can easily be misplaced, deleted, or accessed by unauthorized personnel. In some cases, employees might manipulate stock entries or bills, leading to fraud or financial leakage.

Another identified problem is the lack of reporting and analytics tools in manual systems. Business owners are unable to generate quick reports on stock levels, most-selling items, or overall profitability. Without these insights, they are forced to rely on intuition rather than data-driven decisions, which can harm business growth.

Furthermore, the scalability of operations becomes impossible when a business grows. If a jewellery store expands to multiple branches or wants to sell online, a manual system cannot provide centralized control or live updates across all locations. This creates operational chaos and severely limits the potential of business expansion.

Time consumption is also a hidden but significant issue. From checking stock to preparing bills or finding customer history, everything takes longer in manual systems. This delays customer service, reduces the number of customers that can be served daily, and impacts overall efficiency.

Regulatory compliance is yet another overlooked issue. Many jewellery businesses do not maintain proper GST-compliant records, especially when using handwritten or spreadsheet-based billing. This can lead to legal issues during audits, penalties, or even loss of business credibility.

The Jewellery Management System is designed to eliminate these core issues. It aims to streamline the entire business operation, from inventory and billing to customer handling and reporting. By digitizing every core function, the system will bring in speed, accuracy, security, and professionalism, giving the business a competitive edge in today's digital-first market.

Objectives

In the current age of digital transformation, almost every sector is adopting software systems to improve operational efficiency and customer satisfaction. However, many jewellery businesses, particularly local and mid-sized ones, still rely heavily on manual processes for critical functions like inventory management, billing, customer tracking, and reporting. This lack of automation not only leads to human error and inefficiency but also prevents such businesses from growing in a competitive market. The *Jewellery Management System (JMS)* is developed with the intent to bridge this digital gap and bring structure, speed, and transparency to jewellery shop operations.

The primary justification for this project arises from the high complexity and high value of the jewellery inventory. Unlike general retail stores, jewellery businesses deal with unique products—each differing by weight, purity, design, material, and value. Maintaining this data manually is not only tedious but also extremely risky. A small mistake in calculation or item tracking can result in significant financial losses. JMS provides an automated system that manages inventory in real-time, ensuring that all stock movements are accurately recorded and visible to authorized users.

Security and data integrity is another area where this project proves its importance. Jewellery transactions involve large sums of money, and any loss of data due to hardware failure or internal fraud—can be disastrous. JMS incorporates password-protected access, role-based permissions, and data backups to prevent unauthorized access and ensure the safety of all critical information.

Importantly, the system is designed to be scalable and flexible. While the initial implementation may be for a single shop, the design supports the addition of new features such as barcode integration, cloud storage, SMS/email alerts, or expansion to multiple branches. This makes JMS a future-ready solution that can evolve with the business.



Fig. 1.3:Project Objectives

Another strong justification is the need for accurate and professional billing. Customers purchasing expensive items expect detailed invoices that include weight, gold purity, GST, and price breakdowns. Manual bills are prone to error and can damage the shop's image. With JMS, every sale results in an automatically generated, GST-compliant bill, reducing calculation errors and improving brand perception among customers.

Security and data integrity is another area where this project proves its importance. Jewellery transactions involve large sums of money, and any loss of data—due to hardware failure or internal fraud—can be disastrous. JMS incorporates password-protected access, role-based permissions, and data backups to prevent unauthorized access and ensure the safety of all critical information.

Additionally, the reporting and analytics capability of JMS offers long-term business benefits. It helps business owners understand sales patterns, product performance, seasonal demands, and stock levels. With these insights, they can plan purchases better, identify slow-moving items, and launch targeted

promotions. Manual systems simply cannot provide such in-depth analysis, making JMS a valuable strategic tool. One must also consider the ease of training and staff onboarding. With a standardized digital system in place, even newly hired employees can quickly understand workflows such as billing, stock entry, or customer search. This reduces the learning curve and ensures that the business doesn't get disrupted due to staff changes.

Lastly, the project justifies itself in terms of cost-benefit analysis. Though there may be an initial investment in software development and setup, the long-term benefits in terms of reduced human error, faster operations, improved customer satisfaction, and better business insights far outweigh the costs. It saves time, reduces paperwork, and allows the business owner to focus more on growth and marketing rather than day-to-day operational hassles.

SYSTEM ANALYSIS & STRUCTURAL DESIGN

System Design is one of the most crucial stages in the Software Development Life Cycle (SDLC), as it lays the foundation for how the system will function both internally and externally. It involves defining the architecture, components, modules, interfaces, and data structures that together form the backbone of the system. In the context of the *Jewellery Management System (JMS)*, system design plays a vital role in organizing various functional units like inventory, billing, user roles, and customer data into a streamlined, efficient, and error-free workflow.

System design is the backbone of any software application. For the Jewellery Shop Management System, the design phase played a crucial role in laying out the structural and functional aspects of the application. The objective was to plan how the software would function internally and how various components would interact with each other to ensure smooth performance, scalability, and reliability.

In this system, a modular architecture was chosen, enabling separation of concerns and better maintainability. The system design involved creating data flow diagrams, ER diagrams, user interface mockups, and deciding the database schema and technology stack.

The core objective of system design is to translate user requirements into a technical blueprint that guides the implementation team. This blueprint ensures that each part of the software from frontend interfaces to backend database logic is designed in a way that supports scalability, usability, security, and performance. Without proper system design, even well-written code can lead to functionality breakdowns, data mismatches, or unmanageable complexities.

For JMS, which deals with sensitive and high-value data, such as gold weight, prices, GST billing, customer history, and sales records, it becomes even more important to have a well-structured design. Poor system design in a jewellery business could lead to stock mismatches, incorrect billing, unauthorized access, or even data loss all of which can result in serious financial losses and customer dissatisfaction.

2.1 Entity Relationship Diagram (ERD)

The Entity Relationship Diagram (ERD) is a critical component in the design and planning phase of any database-driven application. It visually represents the data model and how entities (real-world objects) relate to each other within the system. For the *Jewellery Management System (JMS)*, the ER diagram helps in identifying the main entities involved, their attributes, and how they are

interrelated. It provides a clear and logical view of the database structure that will eventually be implemented in the back-end using a relational database like MySQL.

This ERD serves as a blueprint for developers, allowing them to understand what data needs to be stored and how it should be organized. It also helps in identifying relationships and dependencies between different modules of the system, ensuring that data flows efficiently across various functionalities like inventory, billing, stock updates, and customer management.

The ER Diagram of the Jewellery Management System lays the foundation for database development. It defines how jewellery items, customers, staff, and sales activities are connected in a logical and efficient manner. By mapping out each relationship and attribute, the ERD minimizes the risk of database errors and ensures the system will work as expected during real-time use. It acts as a bridge between conceptual design and actual implementation in SQL.

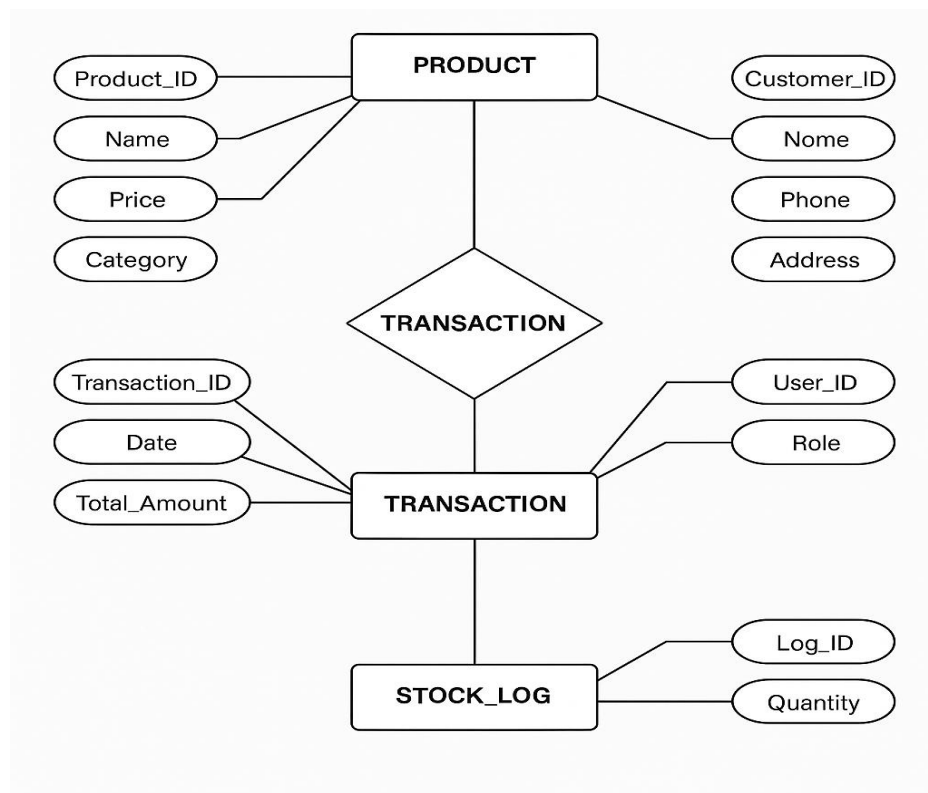


Fig. 2.1: Entity Relationship Diagram

2.2 Key Entities and Their Attributes:

1. PRODUCT

The entity is information about all jewellery items available in the store. Each product has the following attributes:

- Product_ID (Primary Key): Unique identifier for each product
- Name: Name or type of the jewellery (e.g., Ring, Necklace)
- Price: Price of the item
- Category: Grouping of item like gold, silver, diamond, etc.
- This data helps in displaying the product catalog, calculating billing totals, and managing stock availability.

2. CUSTOMER

Every customer who makes a purchase is recorded here. Attributes include:

- Customer_ID (Primary Key): Unique customer identity
- Name: Full name of the customer
- Phone: Contact number
- Address: For delivery or billing purposes
- This entity is crucial for generating invoices, tracking customer history, and loyalty management.

3. USERS

These are the people who operate the system. Users can be admin, manager, or cashier. Attributes:

- User_ID (Primary Key): Unique user ID
- Role: Defines access level (admin, cashier, etc.)
- Role-based access ensures security and controlled permissions.

4. TRANSACTION

This is the most important entity which links multiple others. Every sale/bill is a transaction. Attributes:

- Transaction_ID (Primary Key)
- Date: When the transaction occurred
- Total_Amount: Final bill after applying taxes and discounts
- It connects to PRODUCT (for items sold), CUSTOMER (who bought), and USERS (who processed the sale).

5. STOCK_LOG

This keeps track of stock movements additions and deductions. Attributes include:

- Log_ID (Primary Key): Unique ID for stock operation
- Quantity: Number of items added/removed
- It helps in real-time inventory tracking.

2.3 Relationships Between Entities:

PRODUCT ↔ TRANSACTION: Many-to-many relationship (each transaction can include multiple products, and each product can be part of many transactions).

CUSTOMER ↔ TRANSACTION: One-to-many relationship (one customer can make multiple transactions).

USER ↔ TRANSACTION: One-to-many relationship (a user/cashier can perform many transactions).

TRANSACTION ↔ STOCK_LOG: One-to-many (each transaction may generate one or more stock updates).

These relationships ensure that all modules are interlinked, and every activity in the system can be traced back using the database.

The System Architecture of the Jewellery Management System (JMS) defines the logical structure, workflow, and interactions among the various components of the software. This structure ensures that the system is modular, maintainable, and efficient across different modules such as inventory, billing, user management, and reporting.

The Jewellery Shop Management System is based on a three-tier architecture which separates the application into three layers: Presentation Layer (Frontend), Business Logic Layer (Backend), and Data Layer (Database). This approach increases modularity, improves maintainability, and enhances security by separating core operations from the user interface and database handling.

In the Presentation Layer, technologies like HTML, CSS, Bootstrap, and JavaScript were used to build a responsive and user-friendly interface. This layer handles user interactions such as product browsing, customer entries, and billing generation. The design ensures that both admins and employees can use the system with minimal technical knowledge.

The Business Logic Layer is built using Java and Spring Boot, which handle the system's core functionalities including user authentication, product management, and calculations. Meanwhile, the Data Layer utilizes MySQL for efficient and reliable data storage. The use of relational tables with proper foreign keys helps maintain data integrity and supports quick retrieval for daily operations like generating reports or searching customer records.

Unlike simple applications that use flat file systems or monolithic codebases, JMS is built using a layered and modular architecture, inspired by enterprise-level web applications. This layered design ensures a clear separation between

the user interface, the application logic, and the database, allowing independent development, testing, and future upgrades.

The system architecture of the Jewellery Management System offers a clean, organized, and layered approach to software development. It clearly separates concerns between interface, logic, and data storage, making the system robust, secure, and scalable. This model not only supports current business workflows but also sets a solid foundation for future growth, upgrades, and multi-branch expansion.

The database plays a central role in the system by storing all business-critical information such as product details, stock levels, customer data, and sales transactions. To achieve efficient storage and fast retrieval, a relational database model was chosen. This model ensures data consistency, reduces duplication, and allows structured queries for generating reports. Normalization techniques were applied while designing the database schema to remove redundancy and improve data relationships. For instance, separate tables were created for Products, Customers, Orders, Users, and Inventory, all interconnected through primary and foreign keys. This structure allows for smooth data flow between modules like billing and stock updates.

Each table was carefully designed with appropriate data types, constraints, and indexes to optimize performance. For example, indexing was used on frequently queried fields like customer phone numbers and product IDs. Overall, the database design supports the system's goal of offering accurate, secure, and organized data management for everyday business tasks. Data Flow Diagrams (DFDs) were used during the design phase to understand and visualize how data moves through the system. These diagrams help both developers and stakeholders see how inputs are processed and converted into outputs by different modules. DFDs also help identify redundancies, bottlenecks, and missing components early in the design process.

The Level 0 DFD, also known as the context diagram, shows the system as a single process interacting with external entities such as the admin, employees, and customers. This gives a high-level view of the system's boundaries and its external interactions. It highlights how orders, customer requests, and inventory updates enter and leave the system. In Level 1 DFDs, internal processes like customer registration, product management, order processing, and billing are broken down into sub-processes. Each process includes data stores (like customer database or product catalog) and shows how data flows between them. This helps developers focus on designing each sub-module with clear input/output expectations, ensuring that the overall system works seamlessly.

An Entity Relationship Diagram (ERD) is a visual representation of the database structure and is used to model the relationships between various entities in the system. For the Jewellery Shop Management System, ERDs played an essential role in the database design and implementation stages. They helped define how customer, product, inventory, and billing data are interconnected.

The ER diagram included entities such as Product, Customer, User, Order, and Inventory, each with their attributes and relationships. For instance, a Customer can place multiple Orders, and each Order can include multiple Products. Such relationships were implemented using one-to-many and many-to-many relational mappings via foreign keys and junction tables.

ERDs ensured that data redundancy was minimized and data integrity was maintained. They also helped during development by providing a reference to implement proper SQL table structures and JOIN queries. With a clear visual layout of tables and their links, it became easier for developers to understand the back-end schema and optimize database performance.

One of the critical decisions taken during the implementation phase was to adopt a modular programming approach. Each module such as billing, inventory, and customer management was developed in isolation, tested individually, and then connected via the business logic layer. This approach ensured better error isolation, faster debugging, and more efficient progress tracking during development. For example, while the billing module was being tested, the customer module could simultaneously be developed and later plugged into the flow.

The implementation team also focused on making the user interface intuitive and responsive. Since the end-users include non-technical shopkeepers or staff, the design philosophy followed was “*simplicity with functionality*.” Dropdown menus, auto-complete fields, and button-based navigation were included to avoid manual typing errors. Features like auto-calculation of GST, real-time stock update after billing, and quick search filters helped improve overall usability. To ensure the backend was robust, business logic was written with layered validation. For example, when generating a bill, the system first checks if the product is available in stock, validates the customer record, confirms payment status, and only then proceeds to create the transaction record. This step-by-step validation prevents inconsistent entries and maintains data integrity. Custom error messages were also implemented to help users understand and resolve issues quickly.

A major technical achievement during implementation was the creation of a role-based access system. This means different types of users like admin, cashier, or manager see only the relevant modules and data based on their role. For instance, a cashier can generate bills but cannot view or modify stock reports or add new

users. This ensures both data privacy and system security, preventing unauthorized access or unintentional misuse.

Another essential feature in the backend was the real-time stock management logic, which played a critical role during the billing process. Whenever a bill is generated, the backend immediately updates the inventory by subtracting the sold quantity of a product. This ensures that the displayed stock level always reflects real-time data. Additionally, the system was designed to prevent over-billing, i.e., if the billing quantity exceeds the available stock, an error message is triggered and the transaction is blocked. This logic prevents revenue and inventory mismatches, which are critical in jewellery retail environments.

For efficiency and code reusability, a large part of the backend logic was written in the form of reusable functions and services. These included methods like. By centralizing these operations, any changes or bug fixes could be made in one place, and automatically reflected throughout the application. This approach not only reduced redundancy but also made the system more maintainable and easier to scale.

Each feature planned in the requirement gathering stage was carefully re-evaluated during implementation to ensure it made logical sense and fit real-world operations of a jewellery shop. For example, not just adding products to the system, but tracking how stock moves, which employee generated the bill, what the GST calculation was, and whether the customer had previous purchases all such real scenarios were considered and implemented.

The main objective of this phase was to develop a system that streamlines jewellery business operations such as product inventory, billing, customer management, and report generation. It required collaborative efforts, careful planning, rigorous coding, and consistent testing. Every module from frontend UI forms to backend logic and database integration was carefully implemented to meet the needs of a real-world jewellery business.

Implementation is not just about coding; it also includes setting up the development environment, choosing the appropriate technology stack, designing the database, creating a modular project structure, and ensuring maintainability. Proper implementation lays the groundwork for a system that is scalable, secure, user-friendly, and bug-free.

Implementation is not just about coding; it also includes setting up the development environment, choosing the appropriate technology stack, designing the database, creating a modular project structure, and ensuring

maintainability. Proper implementation lays the groundwork for a system that is scalable, secure, user-friendly, and bug-free.

Implementation is often considered the most visible stage of a software project because it is where stakeholders, users, and clients finally begin to see tangible outputs. A project may have an excellent idea and design, but if implementation is not done with precision, the system can fail to deliver. That's why in JMS, the implementation was treated as the core engineering process, where design translated into actual functionality.

Each feature planned in the requirement gathering stage was carefully reevaluated during implementation to ensure it made logical sense and fit real-world operations of a jewellery shop. For example, not just adding products to the system, but tracking how stock moves, which employee generated the bill, what the GST calculation was, and whether the customer had previous purchases all such real scenarios were considered and implemented.

Data validation was implemented both on the frontend (using HTML5 validation, JavaScript, and Bootstrap alerts) and on the backend (using PHP condition checks or Java Bean validation). For example, the product entry form validated that the price and quantity fields only accepted positive numbers. The customer registration form required a valid phone number format and email syntax. Without proper validation, incorrect data could have led to faulty bills, broken reports, or inconsistent records in the database.

Even though frontend validation helped, it wasn't enough. Malicious users can bypass frontend checks using browser dev tools or fake requests. Hence, backend validation was essential for security and data integrity. In PHP, `isset()`, `empty()`, and `filter_var()` functions were used to verify fields before inserting them into the database. In Java (Spring Boot), field-level annotations like `@NotBlank`, `@Min(1)`, and custom validators ensured form data was thoroughly inspected.

If any input failed backend validation, the system returned custom error messages instead of allowing broken records. These messages were shown to users so they could fix their inputs accordingly. For example, if a duplicate customer entry was detected, the system responded with: "Customer already exists with this phone number." This feedback loop prevented frustration and helped users understand the system requirements clearly. In the long run, this detailed implementation of data validation not only reduced bugs and rework but also increased the reliability and professionalism of the software. It ensured that all data inside the system was trustworthy, structured, and usable for reporting, analytics, and decision-making.

SYSTEM DEVELOPMENT & TECHNOLOGY INTEGRATION

The implementation phase of any software project is where the actual development and integration of different components take place. After gathering requirements, designing the architecture, and finalizing the modules, the Jewellery Management System (JMS) was developed in a structured and modular way. This ensured that each part of the system – such as inventory management, billing, customer tracking, and user access was implemented separately and later integrated into a cohesive whole.

3.1 Overview

The development of JMS was done using a modular development approach, where each functionality was treated as a separate module. This allowed parallel development, testing, and easy debugging. The modules include

- User Authentication Module
- Product Management Module
- Customer Management Module
- Billing Module
- Stock Update Module
- Report Generation Module

The system was implemented in a 3-tier architecture, which separates the user interface, the business logic, and the database. This helped in clean coding, better error handling, and future scalability. To make the interface user-friendly, technologies such as HTML5, CSS3, Bootstrap, and JavaScript were used. The business logic was written in Java (with optional usage of Spring Boot) or PHP, depending on the project environment. The database was implemented using MySQL, which is reliable, scalable, and open-source.

During the implementation, version control was maintained using Git & GitHub, enabling smooth collaboration and rollback options. Frequent code commits ensured that each module's progress was tracked separately.

The system was tested simultaneously using unit tests and manual testing for each module to ensure stability before integration. Bugs and exceptions were logged, and solutions were implemented after analysing each issue. This real-time test-and-fix cycle improved the reliability of the final product.

The system was tested simultaneously using unit tests and manual testing for each module to ensure stability before integration. Bugs and exceptions were logged,

and solutions were implemented after analysing each issue. This real-time test-and-fix cycle improved the reliability of the final product.

The deployment was done on a local server (XAMPP for PHP or Apache Tomcat for Java) for demonstration, with scope to migrate to a cloud-based environment in the future.

In summary, the implementation of JMS was carried out in a step-by-step manner, ensuring code clarity, smooth integration, proper validation, and secure operations. The final system meets all the functional requirements, including stock tracking, customer billing, and secure user login, and is ready for real-world deployment in a jewellery business environment.

One of the critical decisions taken during the implementation phase was to adopt a modular programming approach. Each module such as billing, inventory, and customer management was developed in isolation, tested individually, and then connected via the business logic layer. This approach ensured better error isolation, faster debugging, and more efficient progress tracking during development. For example, while the billing module was being tested, the customer module could simultaneously be developed and later plugged into the flow.

The implementation team also focused on making the user interface intuitive and responsive. Since the end-users include non-technical shopkeepers or staff, the design philosophy followed was “*simplicity with functionality*.” Dropdown menus, auto-complete fields, and button-based navigation were included to avoid manual typing errors. Features like auto-calculation of GST, real-time stock update after billing, and quick search filters helped improve overall usability. To ensure the backend was robust, business logic was written with layered validation. For example, when generating a bill, the system first checks if the product is available in stock, validates the customer record, confirms payment status, and only then proceeds to create the transaction record. This step-by-step validation prevents inconsistent entries and maintains data integrity. Custom error messages were also implemented to help users understand and resolve issues quickly.

A major technical achievement during implementation was the creation of a role-based access system. This means different types of users like admin, cashier, or manager see only the relevant modules and data based on their role. For instance, a cashier can generate bills but cannot view or modify stock reports or add new users. This ensures both data privacy and system security, preventing unauthorized access or unintentional misuse.

3.2 Tools and Technologies Used

The implementation of the *Jewellery Management System (JMS)* required a combination of frontend, backend, and database technologies, along with development tools that ensured efficient, secure, and scalable coding. The careful selection of these technologies contributed significantly to the success of the project. Each tool was chosen keeping in mind its reliability, ease of use, open-source availability, and compatibility with the system requirements.

Below is a comprehensive overview of the tools and technologies used:

1. Frontend Technologies

- **HTML5 (HyperText Markup Language)**

HTML5 was used to structure the content on the web pages. It enabled the creation of semantic, clean, and SEO-friendly layouts. It also supports multimedia and form elements, which were crucial for building forms like stock entry, bill generation, and customer registration.

- **CSS3 (Cascading Style Sheets)**

CSS3 was used for styling the user interface. It allowed customization of elements like buttons, tables, form fields, and menus. CSS animations and transitions added a smoother user experience without relying on JavaScript.

- **JavaScript**

JavaScript was used to add interactivity to the system including real-time validations, dynamic form updates, and error messaging. For instance, when a product code is entered, JavaScript fetches product details like price and weight and displays them without refreshing the page.

- **Bootstrap**

Bootstrap helped in creating a **responsive layout** so that the system worked well on desktops, laptops, and tablets. It provided ready-to-use components like modals, cards, navbars, and alerts, reducing development time while maintaining UI consistency.

2. Backend Technologies

- **Java (with Spring Boot)**

The backend was built using either Java with the Spring Boot framework or PHP, depending on the server environment. These languages provided strong support for **MVC (Model-View-Controller)** architecture, helped separate logic from design, and improved maintainability. Java was preferred for enterprise-level logic, while PHP offered flexibility for rapid development.

- **XAMPP / Apache Tomcat**

For PHP, **XAMPP** was used as the local server setup, which includes Apache, MySQL, and PhpMyAdmin in one package. For Java-based implementation, **Apache Tomcat** served as the servlet container and web server. Both helped host and test the application locally before deployment.

3. Database Technologies

- **MySQL**

MySQL was chosen as the relational database for storing structured data like user details, product inventory, customer information, and transaction records. It supports data consistency, indexing for fast queries, and integrity constraints. Its user-friendly interface in **phpMyAdmin** made testing and debugging easier.

- **phpMyAdmin**

This web-based tool was used to manage MySQL databases. It helped in table creation, SQL query execution, and manual data insertion during testing.

4. Development & Collaboration Tools

- **Visual Studio Code**

VS Code was used as the primary code editor. It supported syntax highlighting, real-time debugging, extensions for HTML/CSS/JavaScript/Java/PHP, and Git integration, which streamlined the coding process.

- **Git & GitHub**

Git was used for **version control**, enabling safe code commits, branch creation, and rollback capabilities. GitHub served as the remote repository for backing up code, reviewing changes, and collaborating efficiently.

- **Postman (Optional – if using APIs)**

In API-based architecture, Postman was used to test endpoints like fetching product details or submitting bills. It helped verify request-response cycles and debug backend services.

3.3 Frontend

Frontend implementation is the process of designing and developing the part of the application that users directly interact with. In the Jewellery Management System (JMS), the frontend plays a critical role because it is used daily by staff like salespersons, managers, and admins to perform key functions such as managing products, generating bills, and viewing reports.

The aim of the frontend was to offer a clean, responsive, and user-friendly interface that even non-technical users could operate with minimal training. The technologies used for the frontend were HTML5, CSS3, Bootstrap, and JavaScript, which together allowed a functional and attractive UI.

A significant enhancement in the frontend was the integration of role-based interface control, which dynamically adjusted the visible modules and actions based on the type of user logged in. For instance, an admin would have full access to modules like user creation, report management, and inventory editing, while a cashier would only see modules related to billing and customer handling. This functionality was implemented using conditional rendering in JavaScript, which would fetch the user's role from the backend session and show or hide menu options accordingly. This not only improved security but also made the UI cleaner and more task-specific for different user roles, reducing the chance of errors or unauthorized modifications.

Another advanced feature was the use of asynchronous data communication between the frontend and backend using AJAX (for PHP) or Fetch API (for Java). This allowed real-time interaction without the need to refresh the page. For example, when a user enters a product ID during billing, the system fetches the item's details (price, name, stock) instantly and auto-fills the corresponding fields. Similarly, on form submission, the user receives an immediate success or error message based on backend validation. This smooth, real-time interaction significantly enhanced the user experience by making the application feel fast, responsive, and modern a must-have for real-world retail environments like jewellery stores.

Page Structure and Layout Design

- **Navigation Bar:** Contains options like Dashboard, Add Product, Add Customer, Billing, Reports, and Logout.
- **Dashboard:** Displays summary cards showing stock status, today's sales, and total customers.
- **Forms:** Used for product entry, customer registration, and bill generation. Form fields included validation and dropdowns for quick selection.
- **Tables:** Used to show lists of customers, products, and transaction histories with search and sort features.
- **Modals/Popups:** Used for confirmation dialogs like "Delete Product?" or "Are you sure you want to log out?"

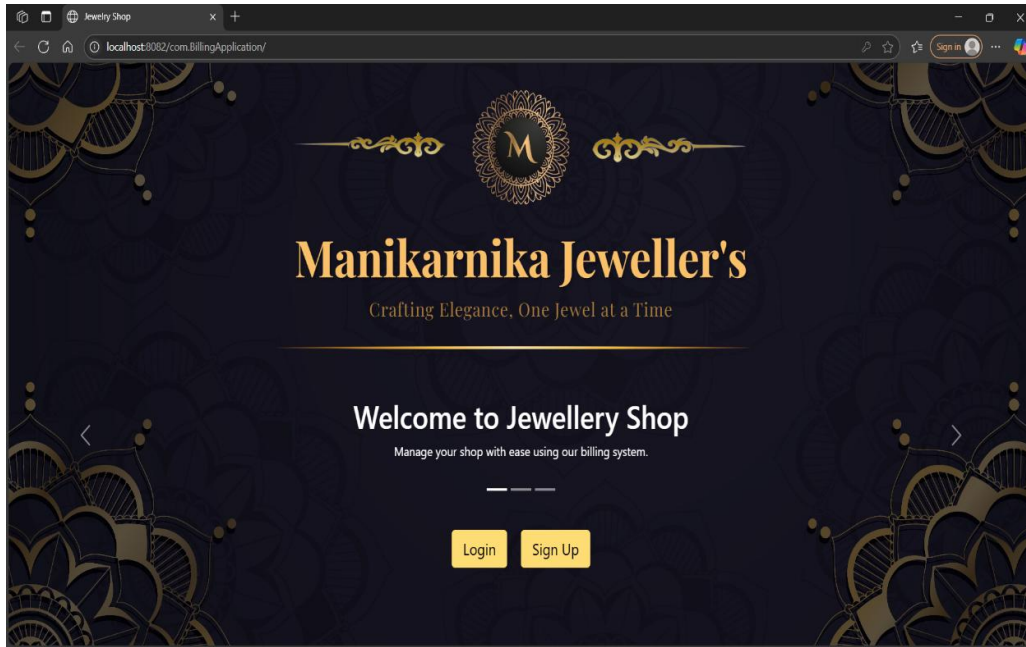


Fig. 3.1: Home Page

Functional Features Implemented

a. Form Validation

Each form includes input validation to prevent incorrect data entry.

- Name field cannot be empty
- Price must be numeric
- Email and phone number must follow proper format

Validation was implemented using JavaScript for real-time checking and alert messages, which improve user experience.

b. Auto-Fill and Dynamic Content

In the billing module, when a product ID is selected, its name, price, and category are auto-filled. Similarly, selecting a customer ID auto-fills their details. This dynamic behaviour is managed using JavaScript event listeners and DOM manipulation.

c. Interactive Buttons

Buttons were styled using Bootstrap classes like `btn-primary`, `btn-danger`, etc., and assigned JavaScript functions.

- Add Product: triggers product validation and sends data to the backend.

- Generate Bill: triggers total calculation, tax application, and data saving.

3.4 Backend

The backend implementation of the *Jewellery Management System (JMS)* forms the foundation of its logical functioning, data management, user authentication, and business operations. It acts as the bridge between the frontend interface and the database, ensuring that user actions are processed correctly, data is securely handled, and appropriate responses are sent back to the client side. The backend was developed using Java with Spring Boot or PHP (based on environment), connected to a MySQL database.

The backend follows the MVC (Model-View-Controller) architecture, which separates data (Model), user interface (View), and control logic (Controller). This separation improves code readability, debugging, scalability, and reusability. In JMS, the controller handles user requests (like submitting a bill), the model interacts with the database (fetching product info), and the view (frontend) displays the result.

Another essential feature in the backend was the real-time stock management logic, which played a critical role during the billing process. Whenever a bill is generated, the backend immediately updates the inventory by subtracting the sold quantity of a product. This ensures that the displayed stock level always reflects real-time data. Additionally, the system was designed to prevent over-billing, i.e., if the billing quantity exceeds the available stock, an error message is triggered and the transaction is blocked. This logic prevents revenue and inventory mismatches, which are critical in jewellery retail environments.

For efficiency and code reusability, a large part of the backend logic was written in the form of reusable functions and services. These included methods like. By centralizing these operations, any changes or bug fixes could be made in one place, and automatically reflected throughout the application. This approach not only reduced redundancy but also made the system more maintainable and easier to scale.

Each feature planned in the requirement gathering stage was carefully re-evaluated during implementation to ensure it made logical sense and fit real-world operations of a jewellery shop. For example, not just adding products to the system, but tracking how stock moves, which employee generated the bill, what the GST calculation was, and whether the customer had previous purchases all such real scenarios were considered and implemented.

The main objective of this phase was to develop a system that streamlines jewellery business operations such as product inventory, billing, customer management, and report generation. It required collaborative efforts, careful planning, rigorous coding, and consistent testing. Every module from frontend UI forms to backend logic and database integration was carefully implemented to meet the needs of a real-world jewellery business.

Implementation is not just about coding; it also includes setting up the development environment, choosing the appropriate technology stack, designing the database, creating a modular project structure, and ensuring maintainability. Proper implementation lays the groundwork for a system that is scalable, secure, user-friendly, and bug-free.

SYSTEM IMPLEMENTATION

The implementation phase is one of the most crucial stages in the Software Development Life Cycle (SDLC). It is during this phase that the theoretical concepts, design specifications, and system architecture created in the earlier stages are translated into actual code, making the system functional. In the context of the *Jewellery Management System (JMS)*, the implementation phase involved converting use-case diagrams, system models, and user requirements into a real, interactive, and operational software application.

4.1 Introduction

The main objective of this phase was to develop a system that streamlines jewellery business operations such as product inventory, billing, customer management, and report generation. It required collaborative efforts, careful planning, rigorous coding, and consistent testing. Every module – from frontend UI forms to backend logic and database integration – was carefully implemented to meet the needs of a real-world jewellery business.

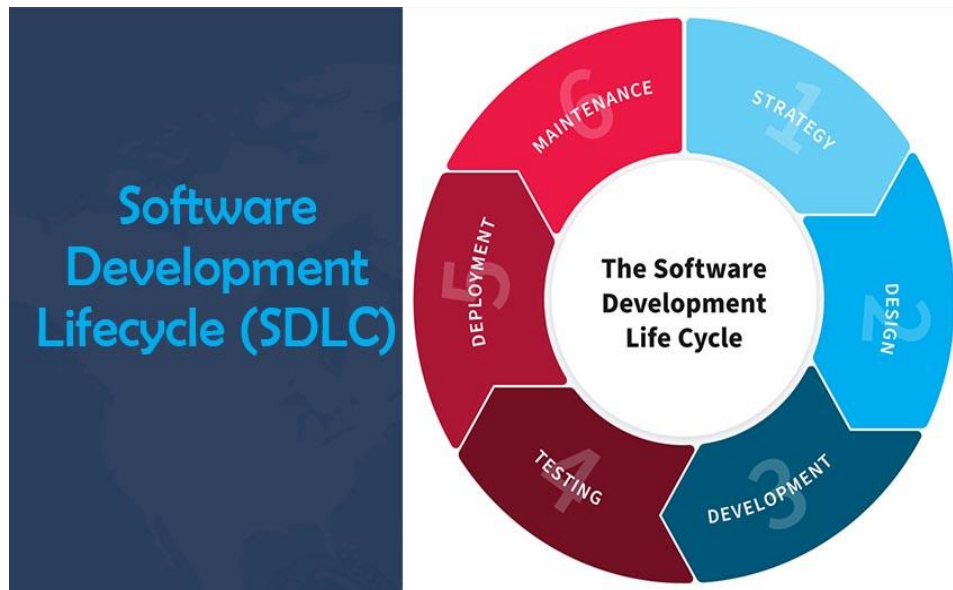


Fig. 4.1: Software Development Life Cycle

Implementation is not just about coding; it also includes setting up the development environment, choosing the appropriate technology stack, designing the database, creating a modular project structure, and ensuring maintainability. Proper implementation lays the groundwork for a system that is scalable, secure, user-friendly, and bug-free.

Implementation is often considered the most visible stage of a software project because it is where stakeholders, users, and clients finally begin to see tangible outputs. A project may have an excellent idea and design, but if implementation is not done with precision, the system can fail to deliver. That's why in JMS, the implementation was treated as the core engineering process, where design translated into actual functionality.

Each feature planned in the requirement gathering stage was carefully reevaluated during implementation to ensure it made logical sense and fit real-world operations of a jewellery shop. For example, not just adding products to the system, but tracking how stock moves, which employee generated the bill, what the GST calculation was, and whether the customer had previous purchases all such real scenarios were considered and implemented.

4.2 Scope and Goals of Implementation in JMS

The *Jewellery Management System* was developed keeping in mind the practical problems faced by small to medium jewellery businesses, such as manual billing errors, inventory mismanagement, and lack of digital customer records. The implementation stage focused on solving these challenges by delivering a digital solution that automates day-to-day business activities.

The key goals of the implementation process in JMS included:

- Developing a **robust backend system** for managing users, products, customers, and bills
- Creating a **responsive and intuitive frontend** interface for ease of use
- Designing a **centralized database** that supports real-time data access and updates
- Implementing **security measures** to protect user and transaction data
- Ensuring **scalability and performance** for increasing business load in the future

Each of these goals translated into specific action items during implementation. For example, to ensure inventory updates in real-time, the backend included logic to automatically deduct sold items from stock during billing. Similarly, to protect user privacy, all passwords were stored using encryption techniques and session management was enabled.

The implementation also addressed potential future requirements, such as exporting sales reports, adding new product categories, and introducing GST-based calculations. The flexible coding structure allowed easy extension and modification of features as per client needs.

4.3 Frontend Implementation

The frontend of the *Jewellery Management System (JMS)* is the first and most frequent point of interaction for the end users, including shop owners, cashiers, and managers. The frontend was designed with the core objectives of simplicity, clarity, and responsiveness. A good UI increases efficiency, reduces errors, and improves user adoption especially for non-technical users in real-world retail settings.

The frontend is responsible for:

- Capturing user input via forms
- Displaying data in clean tables and layouts
- Handling validations before submitting data to the backend
- Giving real-time feedback (alerts, modals, tooltips)
- Ensuring responsive display across different screen sizes

To achieve these goals, the frontend was developed using:

- **HTML5** for structural design
- **CSS3 and Bootstrap** for responsive styling
- **JavaScript** for dynamic behaviour and interaction logic

UI Components and Functionalities

The frontend included multiple core screens, each with specific responsibilities.

Here are the key interfaces and their features

1. Login Page

- Simple login form with username and password fields
- Backend-linked validation using JavaScript and PHP/Java
- Redirects user based on role (Admin, Cashier, Manager)

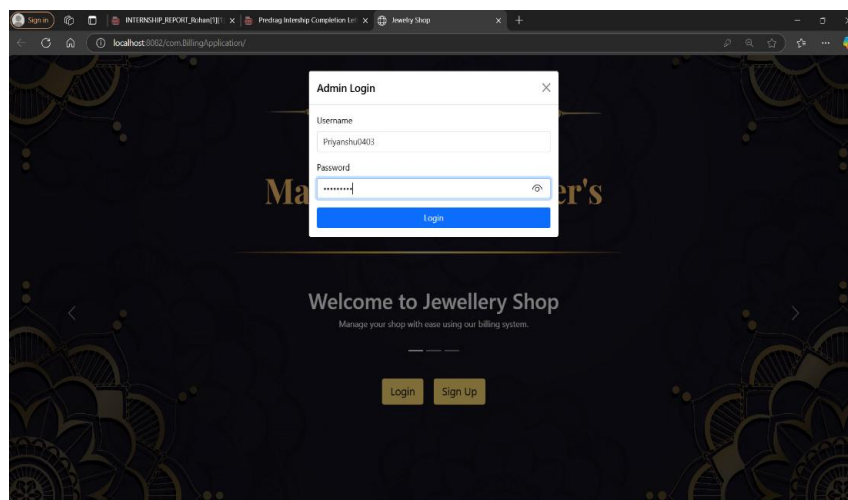


Fig. 4.2: Login Page

2. Dashboard

- Main landing page after login
- Displays system summary (e.g., total products, today's sales)
- Contains quick links to all major modules

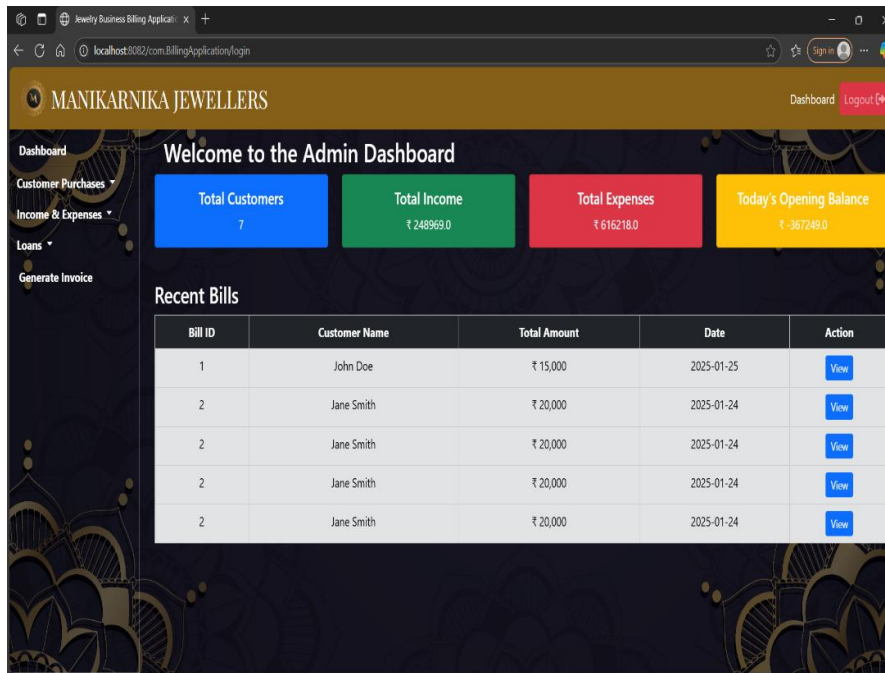


Fig. 4.3: Dashboard

3. Billing Page

- Input fields for entering customer ID and product ID
- Quantity selection and dynamic total calculation
- GST calculated automatically using JavaScript
- Real-time stock check before bill generation
- Print-friendly invoice with all transaction details

The frontend of the *Jewellery Management System (JMS)* is the first and most frequent point of interaction for the end users, including shop owners, cashiers, and managers. The frontend was designed with the core objectives of simplicity, clarity, and responsiveness. A good UI increases efficiency, reduces errors, and improves user adoption especially for non-technical users in real-world retail settings.

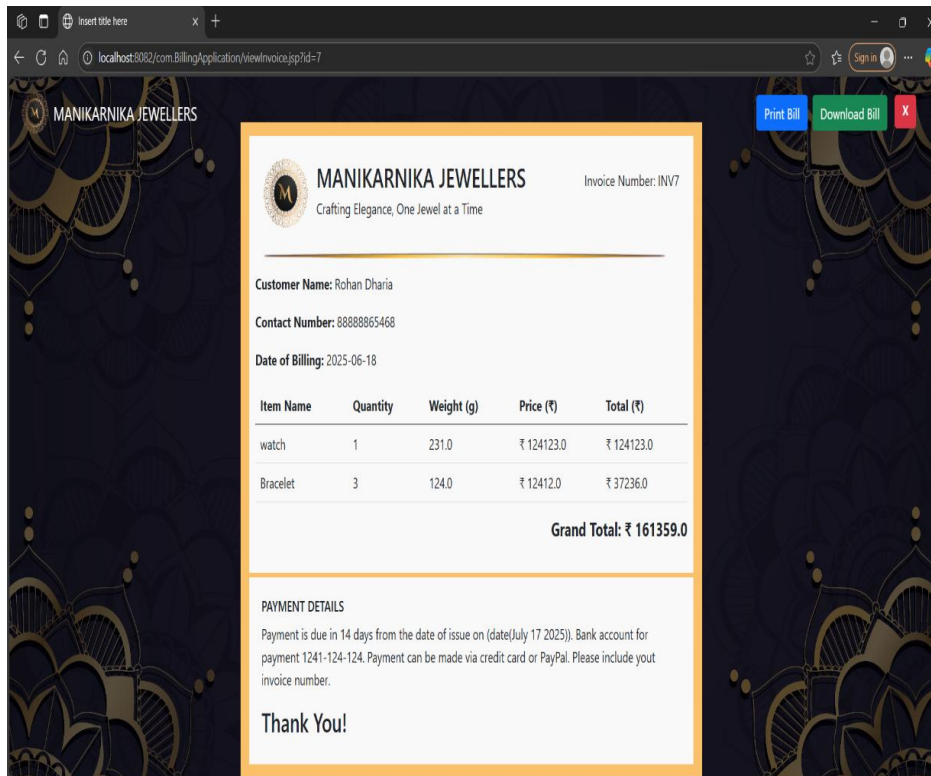


Fig. 4.4: Billing Page

The frontend implementation of JMS combined design principles with functional logic to create a fast, intuitive, and professional user interface. With real-time validation, responsive design, and clean data representation, the system is usable by shop floor staff without any technical training. The use of standard frontend tools and frameworks ensures the interface is easy to maintain and extend in future versions. Accessibility was also considered forms supported keyboard navigation, and font contrast was chosen to support users with visual difficulties. All these modules used Bootstrap Cards, Form Groups, Modals, Alerts, and responsive layouts to deliver a visually consistent experience. Each page whether it's the dashboard, billing page, or inventory form was crafted to guide the user smoothly through their tasks.

4.4 Backend Implementation

The backend forms the brain of the *Jewellery Management System (JMS)*. While the frontend interacts with users, the backend performs all logical operations, data manipulation, and processing tasks. It connects the frontend with the database and ensures that each request (such as adding a product, generating a bill, or retrieving customer history) is handled correctly, securely, and efficiently.

In JMS, the backend handled critical responsibilities such as:

- User authentication & session management
- Database communication (CRUD operations)
- Billing logic & GST calculation
- Real-time stock management
- Error handling & response control
- Role-based access (Admin, Cashier, Manager)

One of the first backend features implemented in JMS was the user login and session control. A secure login system was critical to ensure that only authorized users could access sensitive modules such as inventory, billing, and reports.

This real-time stock check was done via backend queries that compared user input against the products table. If the entered quantity exceeded the available stock, an error was triggered, and billing was halted. Additionally, a separate stock log table could be maintained to record all inward and outward stock transactions, offering a complete stock history to business owners. This would help in reconciling purchases and sales effectively.

Architecture and Technology Choices

The JMS backend was implemented using either **Java (Spring Boot)** depending on the developer's familiarity and environment.

Java with Spring Boot

- Built on the Model-View-Controller (MVC) design
- Used RESTful APIs to connect frontend with backend logic
- Dependency injection via Spring Beans
- Used Spring Data JPA for database handling with entities, repositories
- Configuration handled through application, Properties file

Key Functional Modules

The backend was organized into different modules, each responsible for specific system operations.

1. User Authentication Module

- Verifies login credentials from `users` table
- Stores session data (user role, ID, last login)

- Invalid logins are blocked with error message
- Inactivity timeout logs the user out automatically

2. Product Management Module

- Handles product addition, update, deletion
- Validates fields like weight, price, quantity before insert
- Checks for duplicate product entries
- Admin-only module with secure backend access

3. Billing & GST Calculation Module

- Accepts product ID, customer ID, and quantity
- Fetches real-time stock from the database
- Calculates total amount with GST (e.g., 5%- 18 % on total)
- Generates a unique bill ID and saves all data to `transactions` and `transaction items` tables
- Deducts sold items from stock instantly

4. Customer Module

- Saves new customer details into the database
- Checks for existing records to avoid duplicates
- Used for tracking purchase history and generating repeat bills

In Java, sessions were handled using `session_start()` and destroyed on logout. In Java Spring Boot, JWT (JSON Web Token) or in-memory sessions can be used depending on the setup. This ensures the user remains logged in only during active use, which prevents misuse or session hijacking. In addition, activity logs were recorded for every login attempt. This helped track system usage and added an audit trail a good practice for business systems.

This real-time stock check was done via backend queries that compared user input against the `products` table. If the entered quantity exceeded the available stock, an error was triggered, and billing was halted. Additionally, a separate `stock_log` table could be maintained to record all inward and outward stock transactions, offering a complete stock history to business owners. This would help in reconciling purchases and sales effectively.

4.5 Database Design & Connectivity

In any management system, the database is the heart of the application. For the *Jewellery Management System (JMS)*, it is responsible for storing and managing all the critical data from customer records to products, transactions, and user

credentials. The goal of the database design was to ensure data integrity, consistency, easy retrieval, and relational structure.

In today's digital world, data plays a central role in running any business efficiently. This is especially true in domains like jewellery management, where high-value items, precise inventories, and customer trust are involved. In the *Jewellery Management System (JMS)*, the database is not just a storage component it is the core engine that powers all the essential operations such as product tracking, billing, inventory updates, customer data management, and transaction logging.

The database ensures structured, consistent, and organized handling of data. In a jewellery business, there are multiple categories of products like gold, silver, diamond, gemstones, etc. Each product has its own specifications weight, carat, purity, design, and price and all of this data needs to be stored in a way that it can be easily retrieved, updated, and analyzed. A well-designed database allows the system to store this data in separate tables while maintaining relationships between them. For example, each bill generated is associated with a customer and one or more product items this is achieved using foreign keys and relational mapping in the database.

Another crucial importance of the database in JMS is inventory control. Jewellery items are expensive, often available in limited quantity, and require real-time tracking. Without a proper database, it would be nearly impossible to manage stock efficiently. The database automatically updates the quantity of products when a sale is made and also allows users to view the current availability instantly. This real-time stock management prevents over-selling, supports purchase planning, and helps in maintaining accurate records, reducing manual effort and human errors.

The database also plays a vital role in billing and financial calculations. Whenever a customer makes a purchase, the system generates a bill which includes product details, prices, applicable GST, discounts (if any), and total payable amount. All this information is fetched, calculated, and recorded in the database. Moreover, historical transaction records can be fetched anytime, which is essential for generating daily sales reports, profit analysis, or auditing. This kind of structured financial recording would not be feasible without a reliable backend database system.

Customer management is another area where the database proves its importance. The JMS stores details of every customer such as name, contact number, email, and address. This data is linked to past transactions, allowing the business to analyze purchase patterns, offer loyalty rewards, and build long-term customer relationships. For example, the system can retrieve all bills generated for a

particular customer, calculate their total spending, and even notify them during offers or festivals all made possible through proper database integration.

```
8 rows in set (0.00 sec)
mysql> select * from customer_purchase_backup
-> ;
```

| purchase_id | customer_name | contact_number | item_name | quantity | weight | price_per_unit | total_amount | purchase_date | amount_paid | due_amount |
|-------------|----------------------|----------------|---------------|----------|---------|----------------|--------------|---------------|-------------|------------|
| 1 | Atharva Pajgade | 3635733772 | Necklace | 1 | 30.50 | 34535.00 | 34535.00 | 2242-03-31 | 23443.00 | 11092.00 |
| 2 | Rohan Dharla | 3635733772 | Gold Bracelet | 2 | 30.50 | 332453.00 | 664906.00 | 1144-04-23 | 0.00 | 664906.00 |
| 4 | pms | 9975650695 | watch | 1 | 30.50 | 12332.00 | 12332.00 | 3122-04-12 | 1222.00 | 11110.00 |
| 5 | Atharva Pajgade | 00349230 | Gold Bracelet | 2 | 3432.00 | 345234.00 | 690468.00 | 2332-04-23 | 12343.00 | 678125.00 |
| 7 | Rohan Dharla | 23414124 | Gold Bracelet | 1 | 20.00 | 16455.00 | 16455.00 | 4455-05-06 | 3424.00 | 13031.00 |
| 11 | Vishwasjeet Phadtare | 23414124 | Gold Ring | 1 | 1.20 | 25353.00 | 25353.00 | 5334-02-04 | 452.00 | 24901.00 |
| 12 | Rohan Dharla | 00349230 | Bracelet | 1 | 30.50 | 42352.00 | 42352.00 | 0342-03-05 | 5353.00 | 36999.00 |
| 13 | Laxika Shilwate | 969459392 | Ring | 2 | 30.50 | 77577.00 | 155154.00 | 2006-02-27 | 5857.00 | 149297.00 |
| 14 | Srushti Gandu | 56646464864 | RING | 1 | 23.00 | 65661.00 | 65661.00 | 2025-03-18 | 21655.00 | 44006.00 |

```
9 rows in set (0.11 sec)
mysql> select * from income_n_expense;
```

| ID | Date | Type | Amount | Category | Description |
|----|------------|---------|-----------|---------------|-----------------------------|
| 2 | 3232-04-23 | Income | 230234.00 | Gold Purchase | 3203v32 mgasvg as gangf a f |
| 3 | 0234-03-04 | Expense | 234334.00 | Gold Purchase | 324a0pfag |
| 4 | 0012-12-22 | Income | 12412.00 | Gold Purchase | fegaarger |
| 6 | 0412-03-21 | Expense | 12424.00 | Gold Purchase | fwegewegwegwreg |

```
4 rows in set (0.00 sec)
mysql> select * from admin_data;
```

| User_Name | Email_Id | Password |
|----------------|------------------------------|-----------|
| Priyanshu_MS | priyanshusarvaiyya@gmail.com | Priyanshu |
| KD15 | kdg@gmail.com | Kuldeep |
| shreyash_8767 | shreyash@gmail.com | Shreyash |
| p | priyanshusarvaiyya@gmail.com | p |
| Rutuja Karikar | rutujakarikar@gmail.com | Rutuja |
| nikhil | nikhil@gmail.com | NIkhil |
| Laxika | laxikashilwate@gmail.com | Laxika |
| okokok | okokok@gmail.com | asdfghjkl |

Fig. 4.5: Database in MySQL

Security and data integrity are also ensured by the database. Sensitive data such as user login credentials, admin access rights, and customer identities are stored with protection mechanisms like encryption, hashing, and restricted access. Without a centralized database, ensuring such secure and role-based control across a system would be extremely difficult. The database acts as a gatekeeper that prevents unauthorized changes, ensures only valid data is entered, and maintains backup and recovery mechanisms in case of data loss.

Furthermore, the database simplifies the process of scaling the application. As the jewellery business grows, new products, categories, employees, and branches can be added to the system without disturbing existing data. This flexibility is due to normalized relational design and the ability of databases like MySQL to handle large volumes of structured information efficiently.

Lastly, the database supports data-driven decision-making. With the help of stored data, owners and managers can generate insights on best-selling items, most frequent customers, seasonal sales trends, and low stock alerts. These insights enable smarter planning and higher profitability. Thus, a well-managed database not only supports day-to-day operations but also contributes to strategic growth and customer satisfaction.

Importance of Database Design in JMS

In any management system, the database is the heart of the application. For the *Jewellery Management System (JMS)*, it is responsible for storing and managing all the critical data from customer records to products, transactions, and user credentials. The goal of the database design was to ensure data integrity, consistency, easy retrieval, and relational structure.

Proper database design prevents issues like:

- Redundancy (duplicate data)
- Inconsistency (mismatched values)
- Inefficiency (slow queries)
- Lack of scalability

The JMS database followed relational database principles using MySQL, and every table was designed to reflect the actual entities in a jewellery business. Each entity had its own table with primary keys, foreign keys, and constraints, ensuring organized and meaningful data storage.

Tables and Schema Overview

1. users Table

- Fields: user_id, username, password, role, status
- Purpose: Manages login credentials and user roles
- Security: Passwords are stored encrypted (e.g., SHA-256, bcrypt)

2. products Table

- Fields: product_id, name, type, price, weight, quantity, added_date
- Purpose: Stores inventory data for gold, silver, diamond items
- Special: Auto-decreases quantity after billing

3. customers Table

- Fields: customer_id, name, contact, email, address, registration_date
- Purpose: Maintains customer profiles for billing and reports

4. transactions Table

- Fields: transaction_id, customer_id, total_amount, gst_amount, date_time, billed_by
- Purpose: Stores bill metadata including customer and staff info

5. transaction_items Table

- Fields: item_id, transaction_id, product_id, quantity, price
- Purpose: Line-item records for each product sold in a bill

Each table was connected through foreign key relationships, such as customer_id in transactions referencing the customers table, or product_id linking to products.

Database Connectivity – Backend Integration

- Used Spring Data JPA
- Entity classes mapped to tables using @Entity, @Id, and @Column
- Repository interface extended JpaRepository<Product, Long>
- Queries could be written as Java methods like:

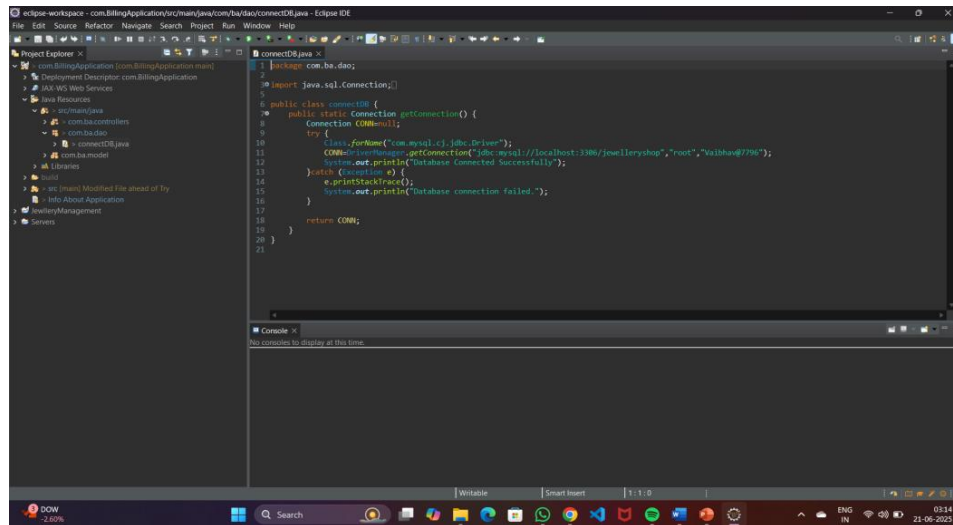


Fig. 4.6: Database Connectivity with System

4.6 Testing & Debugging

Testing is a crucial phase in the development of any software system. In the *Jewellery Management System (JMS)*, where accuracy in billing, inventory updates, and customer records is critical, even a small bug can lead to financial loss or customer dissatisfaction. Therefore, testing and debugging were treated as core pillars of the implementation process, not just as a final step.

Software testing helps in identifying bugs, logical errors, and functional mismatches in the system. It verifies whether each module is working as intended and whether the overall application behaves correctly when all modules interact. In JMS, testing was not limited to checking if forms worked

it included validating calculations, user roles, database transactions, edge cases, and security vulnerabilities.

The testing process began as soon as individual modules were completed. Each function, such as product addition, login authentication, bill generation, and data fetching, was tested independently using unit testing. For example, after completing the product entry module, it was tested with correct, incorrect, empty, and duplicate data to observe how the system responded.

1. Unit Testing:

Each function or feature was tested in isolation. For instance, the billing module's GST calculation was tested independently using different price inputs. Similarly, stock deduction logic was verified for various quantity levels. Unit testing ensured that each block of code delivered the expected output.

2. Integration Testing

Once modules were working individually, they were tested together. The product selection from inventory, customer selection from database, and bill generation flow were tested in sequence. This helped identify bugs caused by data mismatches or incorrect linking between modules.

3. Functional Testing

This verified that the system met the expected business requirements. Example: Adding a product should reflect immediately in the inventory view; or a bill once generated should reduce the correct quantity from stock. Functional testing ensured system behavior aligned with practical usage scenarios.

4. User Acceptance Testing (UAT)

Non-developer users (friends or faculty members) were asked to operate the system. This helped test whether the interface was intuitive and whether the system was usable without technical knowledge. Their feedback led to improvements in form design, button placement, and alert messages.

5. Performance Testing

Although the system was built for small-scale usage, performance testing helped ensure that the system could handle large volumes of data. For instance, the inventory was loaded with 500+ dummy products to check if it slowed down. The search filters and page loads were optimized accordingly.

Debugging Techniques and Tools

During development, debugging played a central role in identifying and solving issues that arose from both frontend and backend code. Debugging was performed in real-time during module testing and also after integration.

In PHP-based backend, error logs were enabled using `error_reporting(E_ALL)` and `ini_set('display_errors', 1)`. This allowed developers to view line-by-line errors directly on the screen or in log files. Console logs were also used in the browser's developer tools (DevTools) to inspect frontend JavaScript bugs.

In the Java-based backend (Spring Boot), structured logs and try-catch blocks were used to catch runtime exceptions. Tools like Postman were used to test REST API endpoints. Exception logs were written in a structured format, and proper error codes (e.g., 400 for bad requests, 401 for unauthorized access) were returned for frontend to display relevant alerts.

Debugging also involved inspecting database entries. Sometimes, incorrect records were generated due to malformed queries. Using phpMyAdmin or MySQL Workbench, the database was monitored in real time to check values inserted, updated, or deleted. This helped quickly identify broken SQL statements or missing field data.

Testing and debugging were critical components of the JMS project that ensured the system was not just functional but also reliable, secure, and user-friendly. Through structured testing methods, real-world usage simulation, and continuous debugging, every major module was validated and optimized. This process revealed flaws early and allowed the team to improve quality before deployment. A thoroughly tested system boosts user confidence, minimizes support issues, and builds long-term trust in the software.

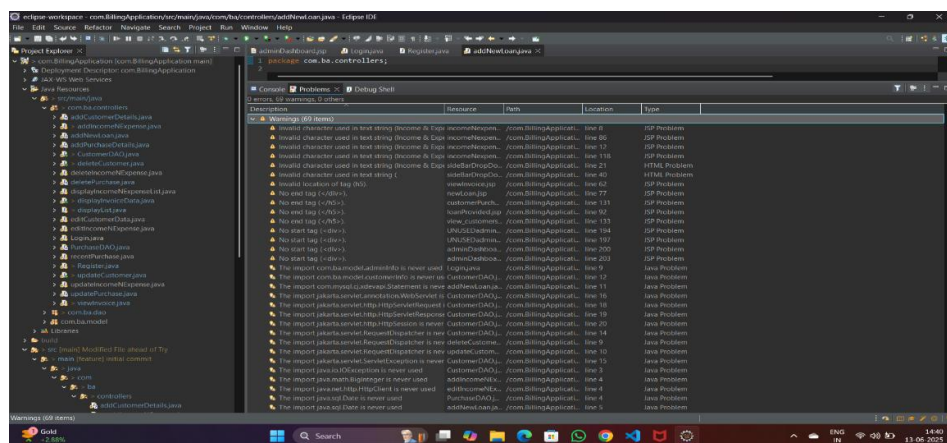


Fig. 4.7: Debugging

Debugging is the process of identifying, analyzing, and fixing errors or bugs that occur during software development. In the Jewellery Management System (JMS), debugging played a critical role in ensuring smooth functioning of various modules such as billing, inventory management, and user authentication. Throughout development, errors were encountered in form validations, database queries, and session handling each of which was traced using techniques like console logging, breakpoints, and backend error logs. For example, missing form inputs would result in undefined values being passed to the database, leading to failed insertions or incorrect calculations. These issues were fixed by inspecting runtime outputs, tracking variable values, and using condition checks to prevent invalid data flow. Debugging not only resolved current errors but also helped in identifying areas of improvement for future stability and optimization of the system.

Challenges Faced and Resolved

One major challenge during testing was handling data inconsistency caused by improper form validation. For example, users could enter characters in number fields or leave important fields blank. To solve this, both frontend (JavaScript) and backend (PHP/Java) validations were implemented to ensure only clean and expected input reached the database.

Another issue was session timeout and role mismanagement. At times, a logged-in user's session would expire, but the system did not redirect them. This was fixed by checking session existence at the top of every protected page, and adding auto-logout timers.

Database integrity errors also came up when trying to delete products that were already linked to transactions. This was solved by using foreign key constraints with ON DELETE RESTRICT, forcing the admin to resolve dependent records before deletion. It preserved the data reliability and avoided "orphan" records.

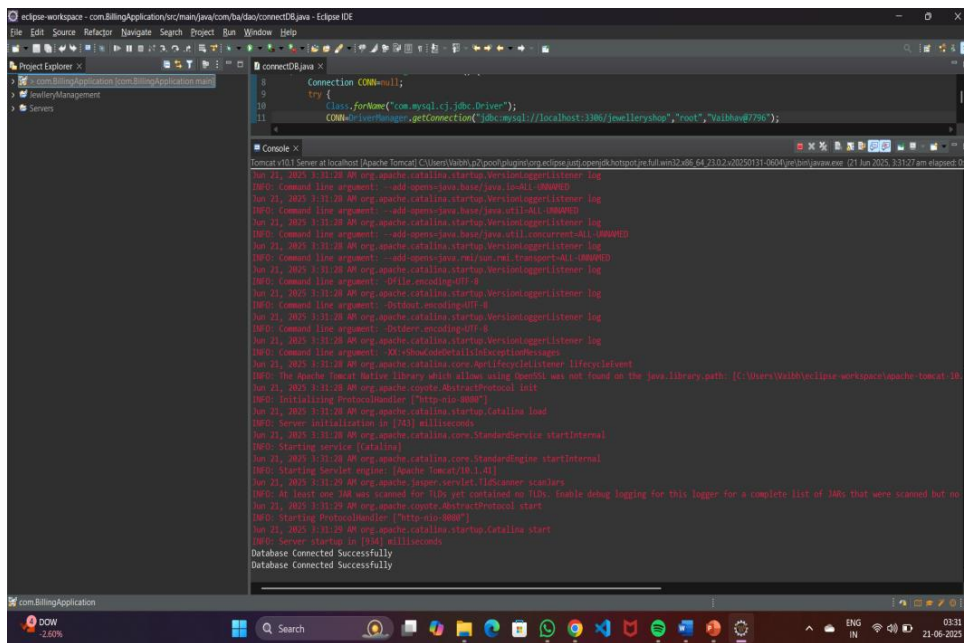


Fig. 4.9 : Resolved Database Connection Error

To solve these issues, I made several improvements in the database schema and logic layer. First, I applied foreign key constraints between key tables such as products, transactions, and transaction_items, with appropriate rules like ON DELETE RESTRICT to prevent deletion of linked records. This ensured that no product involved in any past transaction could be removed without proper resolution. I also added unique constraints on the email and phone fields in the customers table to avoid duplicate data. Additionally, backend validation was enhanced to check for existing records before allowing new inserts. These fixes not only improved the data integrity and consistency of the application but also helped create a more professional and trustworthy environment for handling real customer and product data.

4.7 Data Validation and Input Handling

Data validation plays a foundational role in ensuring the accuracy, safety, and consistency of the information entered into any management system. In the Jewellery Management System (JMS), where users handle product records, customer data, and billing inputs on a daily basis, even a small error such as entering a negative price or leaving a mandatory field empty can lead to incorrect calculations or database corruption. Therefore, the system included a multi-layered data validation mechanism during implementation to minimize user mistakes and ensure only clean and acceptable data is submitted.

Data validation was implemented both on the frontend (using HTML5 validation, JavaScript, and Bootstrap alerts) and on the backend (using PHP condition checks or Java Bean validation). For example, the product entry form

validated that the price and quantity fields only accepted positive numbers. The customer registration form required a valid phone number format and email syntax. Without proper validation, incorrect data could have led to faulty bills, broken reports, or inconsistent records in the database.

Even though frontend validation helped, it wasn't enough. Malicious users can bypass frontend checks using browser dev tools or fake requests. Hence, backend validation was essential for security and data integrity. In PHP, `isset()`, `empty()`, and `filter_var()` functions were used to verify fields before inserting them into the database. In Java (Spring Boot), field-level annotations like `@NotBlank`, `@Min(1)`, and custom validators ensured form data was thoroughly inspected.

If any input failed backend validation, the system returned custom error messages instead of allowing broken records. These messages were shown to users so they could fix their inputs accordingly. For example, if a duplicate customer entry was detected, the system responded with: "Customer already exists with this phone number." This feedback loop prevented frustration and helped users understand the system requirements clearly. In the long run, this detailed implementation of data validation not only reduced bugs and rework but also increased the reliability and professionalism of the software. It ensured that all data inside the system was trustworthy, structured, and usable for reporting, analytics, and decision-making.

SYSTEM EVALUATION & RESULT ANALYSIS

In any business, especially in the jewellery sector where transactions often involve high-value items, it is extremely important to maintain a clear record of income and expenses. The Jewellery Management System (JMS) developed in this project includes a dedicated Income and Expense module, designed to help business owners keep track of every rupee entering and leaving the system. The purpose of this module is not just to log data but also to provide real-time financial summaries, which are essential for decision-making, financial planning, and auditing.

5.1 Income and Expense Module

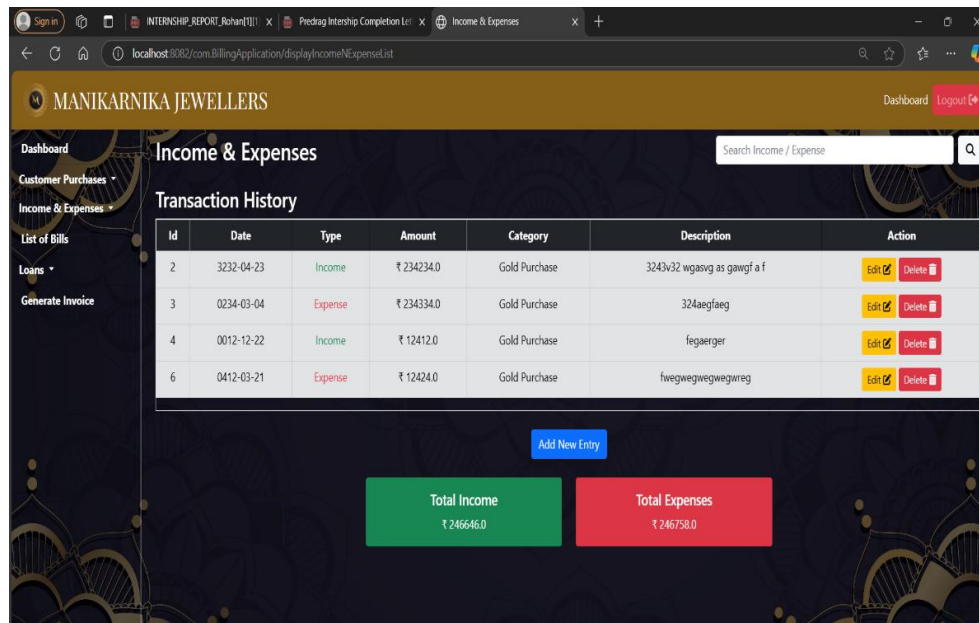
The Income section allows users to record different types of revenue generated by the business. This includes sales of gold, silver, or diamond products, service charges (such as engraving), and even miscellaneous income such as membership fees or resale of old jewellery. Each income record contains essential fields like date, income category, amount, payment method (cash/card/UPI), and remarks. These entries help in categorizing income properly and retrieving it later based on filters like date range or category type. For example, the admin can generate a report of “only diamond sales for the last month” using the category filters implemented in the module.

On the other hand, the Expense section deals with the outflow of money from the business. It is designed to track operating expenses such as employee salaries, shop rent, electricity bills, purchase of raw materials or finished jewellery from wholesalers, repair costs, packaging materials, and other miscellaneous expenses. The expense entry form is similar to the income one, containing fields like date, expense category, amount, payment mode, and description. This uniformity ensures ease of use and helps the admin quickly understand and input data without confusion.

A real highlight of this module is the financial dashboard, which summarizes key financial indicators such as total income, total expenses, and current profit/loss status. These summaries are calculated automatically in real-time as new entries are added to the system. Visual indicators such as green for profit and red for loss help users quickly assess the health of the business. Pie charts and bar graphs (if enabled in the UI) visually show the breakdown of income and expenses across different categories. This feature makes it easy for non-technical users to interpret complex financial data.

From a technical perspective, this module is deeply integrated with the backend database. Separate tables like `income_records` and `expense_records` are used to

store data. Each table includes a primary key and relevant columns for category, amount, date, and user ID (to track who entered the data). Additionally, timestamps and audit logs are recorded for accountability. Backend validations ensure that negative amounts or blank fields are not accepted, and category names are checked against a master list to prevent typographical errors.



The screenshot displays the 'Income & Expenses' dashboard for 'MANIKARNIKA JEWELLERS'. The dashboard includes a sidebar with navigation links: Dashboard, Customer Purchases, Income & Expenses, List of Bills, Loans, and Generate Invoice. The main content area shows a 'Transaction History' table with the following data:

| Id | Date | Type | Amount | Category | Description | Action |
|----|------------|---------|------------|---------------|-----------------------------|-------------|
| 2 | 3232-04-23 | Income | ₹ 234234.0 | Gold Purchase | 3243v32 wgasvg as gawgt a f | Edit Delete |
| 3 | 0234-03-04 | Expense | ₹ 234334.0 | Gold Purchase | 324aegfaeg | Edit Delete |
| 4 | 0012-12-22 | Income | ₹ 12412.0 | Gold Purchase | fegaerger | Edit Delete |
| 6 | 0412-03-21 | Expense | ₹ 12424.0 | Gold Purchase | fwegwegwegwegwreg | Edit Delete |

Below the table, there is an 'Add New Entry' button and two summary boxes: 'Total Income ₹ 245646.0' and 'Total Expenses ₹ 246758.0'.

Fig. 5.1 : Income and Expense

The system also includes a search and filter functionality, allowing users to search for a specific income or expense record based on keywords, date ranges, or categories. This was especially useful during testing, where dummy data spanning several months was entered to simulate real-world operations. The filter responded instantly and accurately, returning only the required entries and enabling efficient record tracking. Export options like CSV or PDF were also tested to ensure that users could take printouts or share reports with accountants or business partners.

Moreover, the Income and Expense module supports multi-user logging, meaning if multiple users are operating the system, their entries are tagged with their user IDs. This adds a layer of accountability, especially in businesses with more than one staff member managing finances. The admin can later audit who made which entry and whether any corrections or unusual patterns are present. If needed, the admin has permissions to edit or delete incorrect entries with full traceability.

This feature-rich module not only makes daily record-keeping efficient but also significantly reduces the chances of manual error. It helps in budget control by

showing overspending categories and allowing the admin to set monthly financial goals. Additionally, during the annual closing or tax filing season, these records can be used as official financial data, minimizing paperwork and saving time.

In conclusion, the Income and Expense module of the Jewellery Management System adds tremendous value to the software by providing financial visibility, operational transparency, and automated reporting. It replaces traditional ledger books with a smart, real-time, and secure digital solution that is scalable and easy to use. For a jewellery business that deals with fluctuating daily revenue and complex expenditure, this module plays a critical role in maintaining financial discipline and planning future investments.

5.2 Loan Section Dashboard & Add New Loan

In the jewellery business, credit-based transactions are fairly common. Customers may request jewellery on a partial payment basis or request gold loans by submitting jewellery as collateral. To manage these real-world financial scenarios, the Jewellery Management System (JMS) includes a dedicated Loan Section with two key components: a Loan Dashboard and an Add New Loan feature. This section was implemented to ensure that every loan-related transaction is traceable, secure, and professionally managed through a digital medium, eliminating paperwork-based risks and improving clarity in credit dealings.

The Loan Section Dashboard acts as the centralized space where all active, completed, and overdue loans are displayed in a clean and sortable table format. Each row contains details such as the Loan ID, Customer Name, Principal Amount, Amount Paid, Remaining Balance, Interest Rate, Duration, Start Date, Due Date, and Loan Status. The dashboard automatically highlights overdue loans using color codes for example, red indicates overdue, green indicates paid, and yellow shows loans nearing their due date. This feature helps the admin or loan manager to quickly identify which loans require follow-up action.

A filter panel is also available on the dashboard, allowing users to filter loans based on status, customer, or date range. Additionally, an export option enables the admin to generate PDF or Excel reports of current loan statuses, which can be shared with financial auditors or business owners. This setup greatly reduces the manual effort of tracking payments and improves transparency and accountability in the business process. Moreover, it helps in identifying trusted repeat customers who regularly clear loans on time, enabling loyalty schemes or priority service offers.

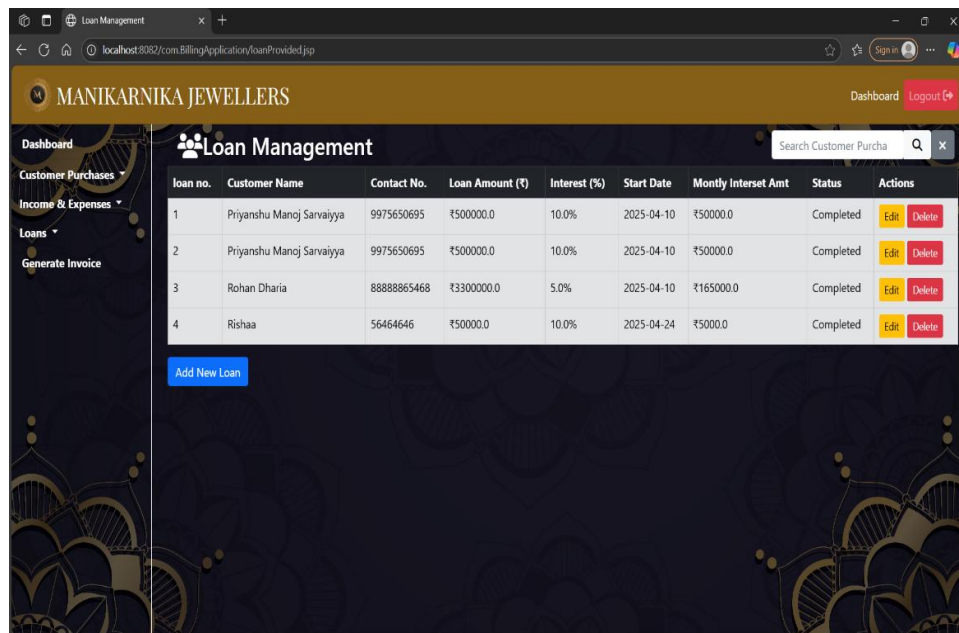
To create a new loan, the system provides a clean and validated “Add New Loan” form. This form allows the staff or admin to select an existing customer from a dropdown (which pulls data from the customer database) or create a new customer profile if needed. The loan creation form includes input fields for the loan amount (principal), interest rate, loan duration (in months), repayment frequency (monthly, quarterly), start date, and a notes section for any special conditions or collateral details. Once submitted, the loan entry is saved to the database, and the system begins tracking its status from the start date onward.

Fig. 5.2 : Add new Loan

One of the strengths of this implementation lies in its automatic interest and due date calculation. The system calculates the total payable amount based on the interest rate and displays the monthly/quarterly installments required. This reduces the chances of human miscalculation and ensures clarity for both staff and customers. During testing, various interest types such as simple and flat interest were applied to sample loans to verify if the system adjusted the payment schedule accordingly and the output was accurate each time.

On the backend, loans are stored in a structured database table, with each loan linked to a specific customer via a foreign key relationship. This allows the system to generate customer-specific loan reports and also display all loan histories when viewing customer profiles. Loan payments made over time are stored in a related table that logs the amount paid, date of payment, remaining balance, and the payment mode (cash, UPI, card, etc.). This gives a comprehensive timeline of repayment and helps the business maintain financial discipline.

Security and privacy are crucial in financial transactions, so this section was implemented with role-based access. Only users with admin-level permissions can create or modify loans, ensuring that sensitive financial operations are not misused. Input validation ensures that negative loan amounts or invalid dates cannot be entered. Backend checks prevent assigning more than one active loan to the same customer unless allowed by business policy.



| loan no. | Customer Name | Contact No. | Loan Amount (₹) | Interest (%) | Start Date | Monthly Interest Amt | Status | Actions |
|----------|---------------------------|-------------|-----------------|--------------|------------|----------------------|-----------|---|
| 1 | Priyanshu Manoj Sarvaiyya | 9975650695 | ₹500000.0 | 10.0% | 2025-04-10 | ₹50000.0 | Completed | Edit Delete |
| 2 | Priyanshu Manoj Sarvaiyya | 9975650695 | ₹500000.0 | 10.0% | 2025-04-10 | ₹50000.0 | Completed | Edit Delete |
| 3 | Rohan Dharia | 88888865468 | ₹3300000.0 | 5.0% | 2025-04-10 | ₹165000.0 | Completed | Edit Delete |
| 4 | Rishaa | 56464646 | ₹50000.0 | 10.0% | 2025-04-24 | ₹5000.0 | Completed | Edit Delete |

Fig. 5.3 : Loan Management

One major advantage of this system is that it reduces loan mismanagement, which can be common in paper-based records. With digital logs, the business owner can see every rupee credited or debited under loans, identify any delays, and take timely action. This not only protects the business financially but also builds customer trust, as all information is documented and accessible.

In summary, the Loan Section Dashboard and Add New Loan feature in JMS provide a comprehensive digital solution for loan tracking. By automating calculations, maintaining payment history, and offering dashboard visibility, this module adds professionalism, precision, and control to the loan management process. It replaces manual registers with a clean, scalable, and efficient financial handling system tailored to the needs of jewellery businesses.

5.3 Invoice Generator

In any retail business, and especially in high-value sectors like jewellery, an invoice is not just a piece of paper or a formality it is a legal and financial document that reflects transparency, professionalism, and transaction accuracy. Understanding this, the Invoice Generator module was designed and implemented as a core part of the Jewellery Management System (JMS). This feature not only simplifies the billing process but also ensures that every transaction is recorded with the right level of detail, providing customers and business owners with confidence and accountability.

The invoice generator is activated once a customer makes a purchase. The system collects all necessary billing information, including customer details, selected products, quantity, price per item, applicable GST or tax rates, discount (if any), and the payment mode. Once the user confirms the sale, the system processes this data and dynamically generates an invoice. The output includes a unique invoice number, transaction date and time, itemized list of products, total amount, tax summary, and net payable amount. Additionally, for official branding, the business name, contact details, and logo are displayed prominently at the top of the invoice.

What makes this invoice generator powerful is its accuracy and automation. It eliminates manual calculations and potential human errors by automatically computing the total price, applying tax percentages, and adjusting the final amount based on any applied discount. During implementation, different billing scenarios were tested such as multiple products in one bill, GST-inclusive and exclusive formats, and percentage-based vs. flat discounts. The system handled each case with precision and produced a clean, well-structured bill every time.

From a technical point of view, the invoice generation is tightly integrated with the database. When a new invoice is generated, the system not only prints the bill but also updates the product inventory in real-time, deducting the sold quantity from the stock database. This ensures that inventory records are always up-to-date and synchronized with actual sales. Moreover, each invoice is stored permanently in the system under a dedicated invoices table, with foreign key relations to customers and purchased items. This allows the admin to retrieve any past invoice instantly using the invoice number or customer name, making auditing and future referencing effortless.

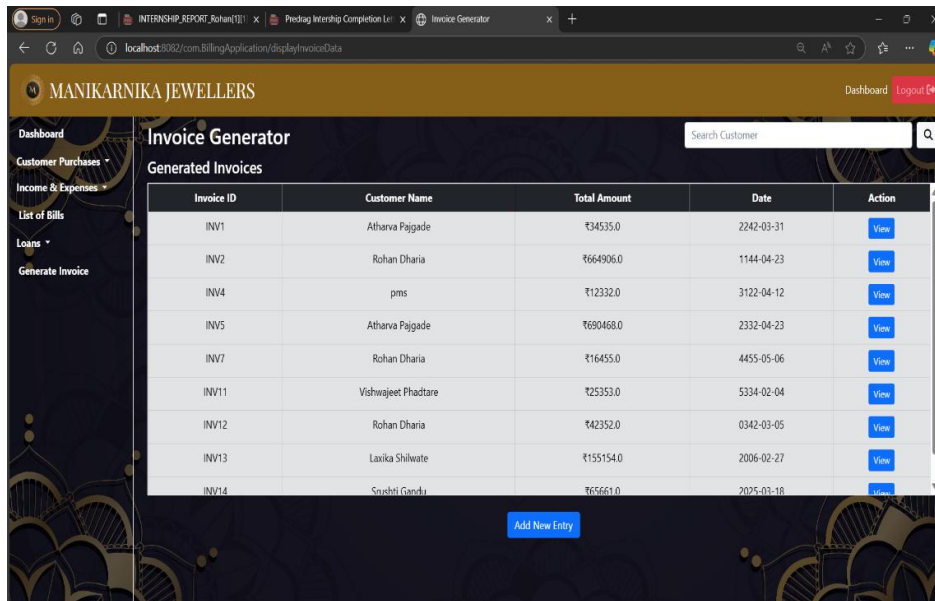


Fig. 5.4 : Invoice Generator

From a technical point of view, the invoice generation is tightly integrated with the database. When a new invoice is generated, the system not only prints the bill but also updates the product inventory in real-time, deducting the sold quantity from the stock database. This ensures that inventory records are always up-to-date and synchronized with actual sales. Moreover, each invoice is stored permanently in the system under a dedicated invoices table, with foreign key relations to customers and purchased items. This allows the admin to retrieve any past invoice instantly using the invoice number or customer name, making auditing and future referencing effortless.

Another notable feature of the invoice generator is its user-friendly print preview. The layout is structured to match standard paper sizes (A4 or thermal printer formats) and includes clear sections such as product breakdown, tax details, terms and conditions, and a note of gratitude. The customer can receive the invoice in both printed and digital format. Digital invoices are stored in PDF format and can be downloaded, emailed, or archived for tax filing and accounting purposes.

The module also supports multi-currency formatting, which becomes useful if the business decides to deal with foreign customers or provide price breakdowns in USD or other currencies. Though the primary currency is INR, backend logic allows formatting based on currency symbols and decimal preferences. All these features make the invoice look professional and align it with real-world expectations of modern jewellery outlets.

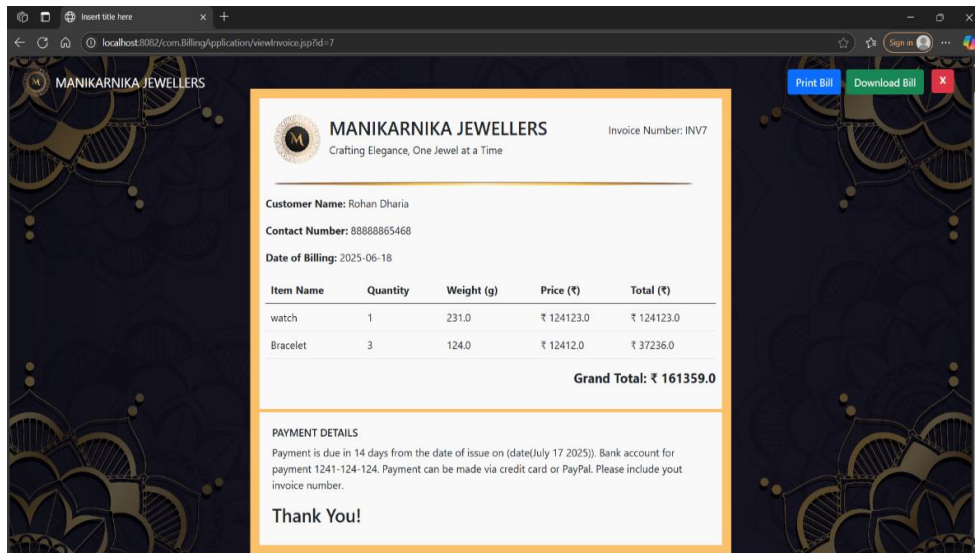


Fig. 5.5 : Generated Invoice

Security and consistency were also kept in mind during development. Each invoice is timestamped and locked once created, preventing any unauthorized edits. If a mistake occurs during billing, the invoice can only be canceled or marked as void by the admin, and a new one must be generated. This practice mirrors standard accounting principles and ensures that the system remains legally compliant and trustworthy. From a business perspective, the invoice module helps in generating reports on daily, weekly, or monthly sales. Since each invoice is tied to the user (staff/admin) who generated it, the system can produce employee-wise sales reports as well. This allows the owner to monitor individual performance, identify high-performing products, and track peak sale hours or seasons.

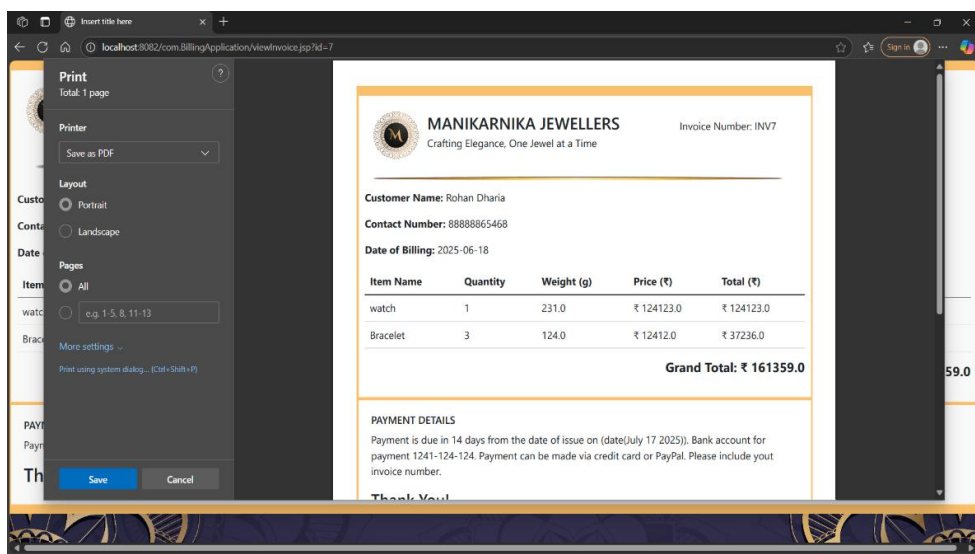


Fig. 5.6 : Invoice converted to PDF

In conclusion, the Invoice Generator module transforms the billing process from a manual, error-prone task into a fully automated, reliable, and professional function. It supports the core goals of JMS accuracy, transparency, and speed while also enhancing customer experience through neat, informative, and legally sound invoices. This feature is not only essential for completing transactions but also for maintaining long-term records, compliance, and business intelligence.

5.4 Add Customer Info & Customer Details

Maintaining detailed and well-structured customer records is an essential part of any business, especially in the jewellery domain, where high-value transactions, customer loyalty, and personalized service are crucial for long-term growth. The Add Customer Info and Customer Details modules in the Jewellery Management System (JMS) were designed to address these needs by enabling a reliable, fast, and organized way to manage customer-related data. These modules not only simplify the process of storing and retrieving customer information but also enable better customer relationship management, personalized service, and business insights.

The “Add Customer Info” feature allows staff or the admin to register a new customer into the system. The form is simple and user-friendly, requiring basic but important fields such as Customer Name, Phone Number, Email Address, and Physical Address. Optional fields include gender, date of birth, and customer type (regular, VIP, walk-in). These data points help in building a customer profile that can later be used for loyalty programs, discounts, or personalized notifications. To ensure data integrity, form validations are applied such as checking for valid phone numbers, unique email IDs, and non-empty required fields. During testing, the system effectively prevented duplicate entries and flagged already registered customers based on phone or email matches.

Once a customer is added, their profile becomes a part of the central customer database, accessible through the Customer Details module. This section presents a complete list of all registered customers in a searchable and filterable table format. Admins and staff can easily look up customer records using their name, phone number, or unique customer ID. Each entry provides a summary view showing key information such as total number of purchases, total amount spent, last transaction date, and active loans (if any). This helps the staff in recognizing loyal customers and offering them customized services or deals.

Another powerful aspect of this module is its integration with other parts of the system. When generating an invoice or creating a loan, the system automatically pulls data from the customer database to auto-fill fields, which saves time and avoids data duplication. For example, during billing, once the customer’s name or phone number is entered, the system fetches their full profile, including address, email, and past purchases. This allows the staff to greet customers

personally, recommend products based on history, or track their preferences over time.

The screenshot shows a web browser window with the address bar displaying 'localhost:8082/com.BillingApplication/addCustomerDetails.jsp'. The page title is 'MANIKARNIKA JEWELLERS'. The left sidebar contains a 'Dashboard' menu with sub-items: 'Customer Purchases', 'Income & Expenses', 'Loans', and 'Generate Invoice'. The main content area is titled 'Add Customer Info' and includes instructions: 'step 1: Fill Customer details (submit the details)' and 'step 2: Add Customer purchase details'. The current step is 'Step 1', which contains a form with the following fields: 'Name*' (text input), 'Contact Number*' (text input), 'Email (optional)' (text input), and 'Gender*' (dropdown menu with 'Select Gender' as the selected option). Below the form are three buttons: 'Cancel' (red), 'Add Customer' (green), and 'Show Customer Purchase List' (blue).

Fig. 5.7 : Adding Customer Info

Technically, the customer data is stored in a dedicated customers table in the database, with each customer assigned a unique customer ID. Fields are indexed to ensure fast lookup and sorting, even as the number of entries grows. A well-designed foreign key structure links each customer to other tables like invoices, loan_records, and transactions, making it easy to fetch complete historical data about that customer in just a few clicks. These connections also enable the system to generate detailed customer-wise reports, which are useful for marketing, financial planning, and service improvement.

Security and privacy are important in customer data handling. To comply with basic data protection practices, access to customer data is limited based on user roles. Only authorized users (admin or manager-level staff) can edit or delete customer records. Regular users can view only limited fields during billing. Each action such as customer addition, edit, or deletion is logged in the system to maintain transparency and accountability. This builds trust within the business and ensures that sensitive customer information is not misused.

Once a customer is added, their profile becomes a part of the central customer database, accessible through the Customer Details module. This section presents a complete list of all registered customers in a searchable and filterable table format. Admins and staff can easily look up customer records using their name, phone number, or unique customer ID. Each entry provides a summary view showing key information such as total number of purchases, total amount spent,

last transaction date, and active loans (if any). This helps the staff in recognizing loyal customers and offering them customized services or deals.

| Customer ID | Customer Name | Contact No. | Gender | Total Amount | View | Actions |
|-------------|---------------------------|-------------|--------|--------------|--------------------------------|---|
| 7 | Rohan Dharia | 88888865468 | Male | ₹161359.0 | View Purchases | Edit Delete |
| 8 | Priyanshu Manoj Sarvaiyya | 124423521 | Male | ₹615615.0 | View Purchases | Edit Delete |
| 9 | Atharva Pajgade | 06548964646 | Female | ₹0.0 | View Purchases | Edit Delete |
| 10 | Atharva Pajgade | 06548964646 | Male | ₹1541156.0 | View Purchases | Edit Delete |
| 11 | Priyanshu | 6546464 | Male | ₹0.0 | View Purchases | Edit Delete |
| 12 | Rishaa | 154111811 | Female | ₹1172986.0 | View Purchases | Edit Delete |
| 13 | Amit | 9986464644 | Male | ₹0.0 | View Purchases | Edit Delete |
| 14 | Atharva Pajgade | 06548964646 | Male | ₹0.0 | View Purchases | Edit Delete |

Fig. 5.8 : Customer Details

Another powerful aspect of this module is its integration with other parts of the system. When generating an invoice or creating a loan, the system automatically pulls data from the customer database to auto-fill fields, which saves time and avoids data duplication. For example, during billing, once the customer's name or phone number is entered, the system fetches their full profile, including address, email, and past purchases. This allows the staff to greet customers personally, recommend products based on history, or track their preferences over time.

Technically, the customer data is stored in a dedicated customers table in the database, with each customer assigned a unique customer ID. Fields are indexed to ensure fast lookup and sorting, even as the number of entries grows. A well-designed foreign key structure links each customer to other tables like invoices, loan_records, and transactions, making it easy to fetch complete historical data about that customer in just a few clicks. These connections also enable the system to generate detailed customer-wise reports, which are useful for marketing, financial planning, and service improvement.

Security and privacy are important in customer data handling. To comply with basic data protection practices, access to customer data is limited based on user roles. Only authorized users (admin or manager-level staff) can edit or delete customer records. Regular users can view only limited fields during billing. Each action such as customer addition, edit, or deletion is logged in the system to maintain transparency and accountability. This builds trust within the business and ensures that sensitive customer information is not misused.

From a business perspective, this module plays a significant role in improving customer service and business intelligence. Over time, the system can generate insights such as “Top 10 Spending Customers,” “Most Frequent Visitors,” or “Customers Due for Follow-Up.” These insights help in creating targeted campaigns, improving retention, and offering personalized service, which is crucial in the jewellery sector where relationships drive repeat purchases.

In conclusion, the Add Customer Info and Customer Details modules provide a robust, secure, and user-friendly solution to manage customer information digitally. These features eliminate the need for manual registers, reduce data entry time, and create a connected experience across the system. With seamless integration, role-based access, and analytics-ready data, this module adds both operational efficiency and strategic value to the Jewellery Management System.

5.5 Add New Entry & Purchase Details of Customer

A jewellery management system is incomplete without a proper way to record purchases made by the business itself. These purchases refer to inventory acquired from wholesalers or manufacturers such as gold chains, diamond rings, silver articles, or other ornaments. The Add Purchase Info and Purchase Details of Customer modules in the Jewellery Management System (JMS) serve two different but equally important purposes. One tracks the supply side (what the business buys to sell), and the other tracks the demand side (what customers are buying). Together, these modules create a full-circle visibility of the entire business cycle.

The Add Purchase Info module is used by the admin or inventory manager to enter new stock purchased from suppliers. The form is designed to be simple but comprehensive. It captures all essential fields such as Product Name, Category (Gold, Silver, Diamond), Supplier Name, Quantity Purchased, Cost Price per Unit, Purchase Date, and optional remarks. There’s also a field for uploading a reference number or invoice received from the supplier, which is helpful during audits. Once the form is submitted, the quantity of that item is updated in the product inventory table, and the entry is stored in a separate purchase log.

This module ensures that the business always has a clear record of which products were bought, from whom, at what price, and on which date. Over time, this data helps identify trusted suppliers, seasonal purchase trends, and cost patterns. During testing, the system was able to correctly update stock levels each time a new purchase was added. For example, if 10 gold chains were added to inventory, the system reflected the increase immediately and updated the available quantity in the billing interface as well.

From a backend perspective, purchases are stored in a purchases table linked to the products and suppliers’ tables. This relationship helps in generating supplier-wise purchase reports, product movement summaries, and even profit margin calculations when compared with selling prices. Additionally, the system

includes validations to prevent adding products with zero quantity or negative cost. Duplicate purchases can be detected using purchase reference numbers, which ensures data consistency.

On the other side, the Purchase Details of Customer module allows the admin or staff to view everything a customer has purchased from the shop till date. This includes information such as Invoice Number, Product Name, Quantity, Price, Purchase Date, and Mode of Payment. This data is useful in many scenarios such as repeat orders, returns, warranty checks, loyalty program eligibility, or product inquiries by the customer.

The system fetches this data from the invoices and invoice_items tables using the customer ID as a key. The interface is clean and sorted by most recent purchases first, but filters are available to view purchases by category, date, or invoice number. Each entry is clickable, leading to a full invoice view with tax details, ensuring full transparency. This feature was tested with customers having 5–10 previous purchases, and the system correctly displayed complete and accurate history with proper links and navigation.

The Add New Entry feature allows users to input manual transactions into the system with full control. This includes internal fund transfers, supplier advance payments, miscellaneous cash inflow/outflow, asset purchases, or even manual corrections made after reconciliation. The entry form includes fields such as Entry Type (Income or Expense), Category, Description, Amount, Date, Payment Mode, and remarks. By offering this functionality, the system ensures that the admin can maintain financial completeness — no matter how unique or irregular the transaction may be.

For example, if an electricity bill was paid outside the main expense workflow or a refund was issued to a customer for a returned item, such activities can be logged through the Add New Entry form. This flexibility also supports real-time adjustments during audits or closing entries at the end of the month or fiscal year. The module was tested with both income and expense entries using various categories, and it correctly reflected these changes in the overall income/expense dashboard, as well as in the profit/loss calculations.

Fig. 5.9 : Add New Entry

This module works as a real-time ledger book, eliminating the need for maintaining physical registers or Excel sheets. One of the key features of this section is that it automatically links transactions to their source modules — for example, a sale made through the invoice module will appear here with a direct reference link to the original invoice. This linking makes it easy to navigate from a transaction to its source and back, enabling smooth investigations and transparency.

During the testing phase, the system was filled with sample data including sales, loan installments, expenses, and refunds. The Transaction History module accurately listed all entries, regardless of their source, and the filter options worked quickly, even with dozens of records. The export feature allowed the admin to download the history in PDF or Excel format, making it useful for external audits, bank loan applications, or annual tax filings.

One more practical use of this module is in daily cash flow tracking. The business owner can open this module at the end of each day and get a complete picture of how much money came in, how much went out, and what the net balance looks like. This not only supports better budgeting but also helps in catching errors early. For example, if the sale amount seems correct but the recorded income is lower, it could point to a missed entry or unauthorized cash handling.

Role-based access control is also implemented in this module to ensure data security. Regular staff may only view transactions related to their own activities, while admins can see all system-wide entries. This access separation prevents data leaks and enforces internal control policies.

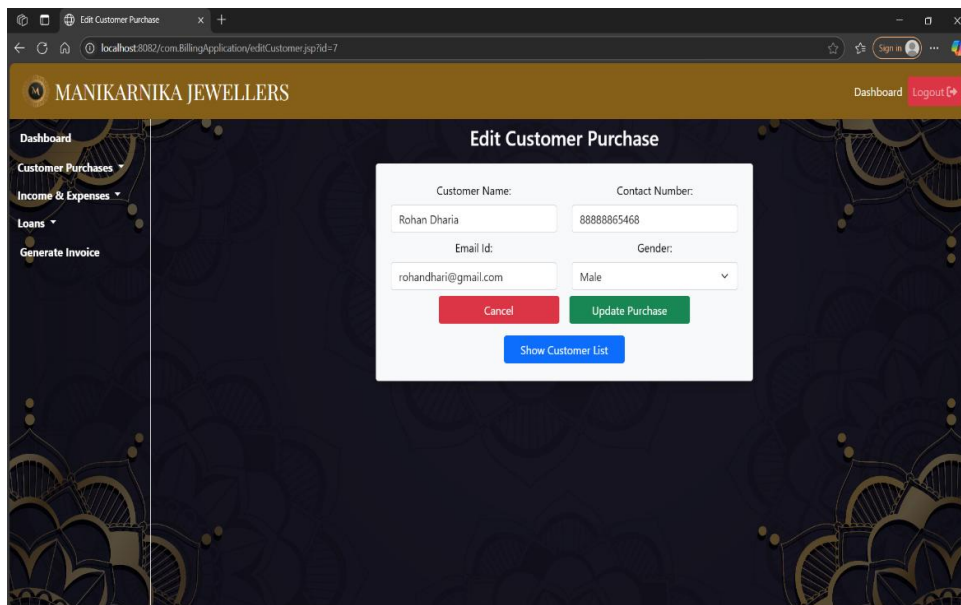


Fig. 5.10 : Edit Customer Purchase

In a jewellery business, effective inventory and purchase tracking is vital for smooth operations, cost control, and accurate billing. The Add Purchase Info and Purchase Details of Customer modules in the Jewellery Management System (JMS) together form a powerful bridge between inventory management and customer sales tracking. These modules not only help the business record all supplier-based purchases but also maintain complete visibility of what individual customers have bought in the past. This holistic view supports transparency, accountability, and better customer service.

The Add Purchase Info module allows the admin or inventory manager to log every new purchase made by the business from suppliers. These purchases may include gold, silver, diamond jewellery items, loose gemstones, or other inventory supplies. The form designed for this feature includes fields such as Item Name, Product Category, Supplier Name, Purchase Quantity, Cost Price per Unit, Total Cost, Purchase Date, and Remarks. This information ensures that each item entering the business inventory is traceable to its source, price, and supplier. Once the entry is submitted, the item quantity is automatically updated in the system's stock records.

To ensure accuracy, backend validations are applied, preventing entry of incorrect data such as negative prices or zero quantities. Moreover, if the item already exists in the inventory, the system updates its quantity and purchase log rather than creating a duplicate. This structure avoids data clutter and ensures that inventory values always reflect the true stock on hand. All purchase records are stored in a dedicated database table, linked with product IDs and supplier details through foreign keys. These connections allow generation of supplier-

specific reports, cost analysis charts, and stock valuation statements over a time period.

The Purchase Info module plays a crucial role during audit and stock reconciliation. Business owners can view purchase histories by supplier or product category, and analyze trends such as which items are frequently restocked, what their costs were over time, and whether seasonal purchasing affects pricing. This data helps in decision-making regarding bulk purchases, price negotiations, and inventory rotation.

The screenshot shows a web application interface for 'MANIKARNIKA JEWELLERS'. The main heading is 'Edit Purchase Details'. The form contains the following fields and values:

| Item Name: | Quantity: |
|----------------------------------|--------------|
| watch | 1 |
| Weight: | Price(in ₹): |
| 231.0 | 124123.0 |
| Date: | |
| 06-04-2025 | |
| <button>Update Purchase</button> | |

Fig. 5.11 : Edit Purchase Details

The Purchase Details of Customer module, on the other hand, tracks and displays all the purchases made by each registered customer in the system. This is useful for after-sales service, warranty follow-ups, returns, and even marketing campaigns. Whenever a customer makes a purchase, their transaction including Invoice Number, Product Name, Quantity, Purchase Date, Selling Price, GST, and Payment Method is automatically recorded under their customer ID. This data can be accessed via the Customer Details panel, which allows the staff to view a customer's complete purchase history at any time.

This module enhances customer service significantly. For example, if a customer wants to buy a matching bracelet for a necklace they purchased earlier, staff can quickly review their old purchase records and suggest matching products. It also helps in recognizing loyal customers based on the volume and frequency of their purchases, enabling loyalty programs and personalized service offerings. If a product defect is reported, the invoice reference helps in verifying the purchase date, price, and terms, improving trust and professionalism.

A key feature of this module is its search and filter capability. Admins can filter a customer's purchase history by date range, product category, or invoice number. This ensures quick access even when the customer has made dozens of purchases over the years. Additionally, each entry in the customer's purchase history links directly to its original invoice, giving a complete view of the transaction in one click. During testing, this feature functioned smoothly, accurately reflecting multi-item purchases and combined invoices, which is common in jewellery billing.

CONCLUSION

The Jewellery Management System project has been developed as a part of my internship at PreDrag, with the primary goal of creating a digital platform to streamline and simplify the daily operations of jewellery businesses. During this internship, I worked on designing and implementing various modules including customer management, loan tracking, invoice generation, inventory control, transaction recording, and financial dashboards. This hands-on experience allowed me to apply the technical skills acquired during my academic studies into a real-world application.

The system provides an integrated and user-friendly solution for managing jewellery shop operations that are otherwise dependent on manual entries and disconnected records. Through this project, I was able to implement database design, backend logic, and frontend interfaces that enable functionalities such as loan creation, billing, purchase tracking, and expense monitoring. These modules collectively help store owners improve accuracy, transparency, and efficiency in their business workflow. Completing this project under PreDrag gave me deep insights into software development lifecycles, requirement gathering, and the challenges of building real-time applications for live users. It also enhanced my understanding of technologies such as MySQL, Java, JavaScript, and responsive web design frameworks. The experience improved my problem-solving skills and introduced me to real-world debugging, testing, and client feedback handling.

REFERENCES

- [1]. Krug, S. (2014). Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. New Riders.
- [2]. Garrett, J. J. (2011). The Elements of User Experience: User-Centered Design for the Web and Beyond. New Riders.
- [3]. GitHub Docs. (n.d.). Understanding the Git Workflow. Retrieved from <https://docs.github.com>
- [4]. MDN Web Docs. (n.d.). HTML5, CSS3, and JavaScript Documentation. Retrieved from <https://developer.mozilla.org/>
- [5]. MDN Web Docs. (n.d.). HTML5, CSS3, and JavaScript Documentation. Retrieved from <https://developer.mozilla.org/>