

Accounting softwARE

LOW LEVEL DESIGN DOCUMENT



DXC TECHNOLOGY

**1. Introduction**

**1.1 Purpose**

Accounting software is a type of application software designed to manage and streamline financial transactions, record-keeping, and other accounting-related tasks within an organization. It plays a crucial role in automating various financial processes, providing accurate and timely financial information, and facilitating compliance with regulatory requirements.

**1.2 OVERVIEW:**

The LLD document provides a detailed roadmap for the development of the accounting software platform, covering both frontend and backend aspects, validation strategies, component design, database schema, and integration with external components. The emphasis on user experience, scalability, and security reflects a comprehensive approach to building a robust accounting software application.

**1.3 SCOPE:**

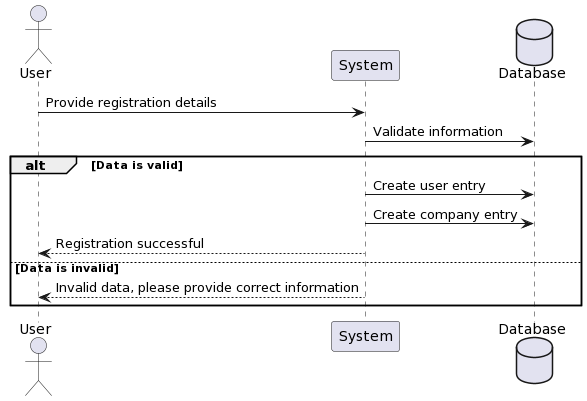
Low-Level Design describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer directly codes the program from the document.

**2. Sequence Diagram**

A sequence diagram provides a visual representation of the interactions and order of execution among various components or actors in a system.

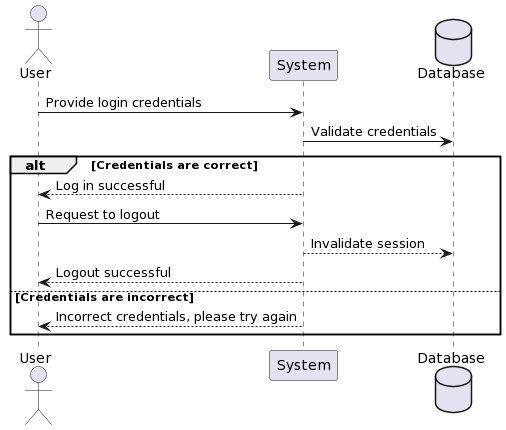
**2.1 USER REGISTRATION:**

The diagram below shows all the processes required for the register feature. The user must enter the URL of the website and click register. The user is then prompted to enter the fields required for registration.



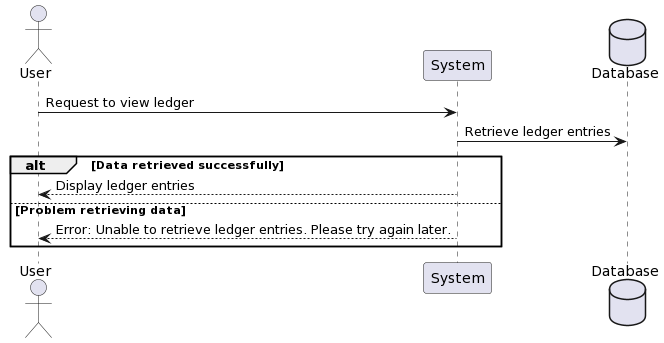
**2.2 LOGIN/LOGOUT:**

The diagram below shows all the processes required for the login feature. The user has to enter the URL of the website and click login. The user is then prompted to enter the fields required for login.



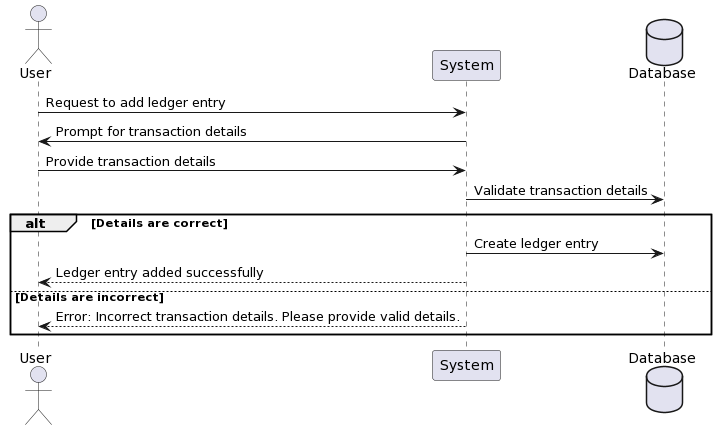
**2.3 LEDGER DISPLAY:**

The diagram below shows the processes required for the ledger display. The user as soon as logins and must click the ledger display. The user will then be displayed the list of ledger entries.



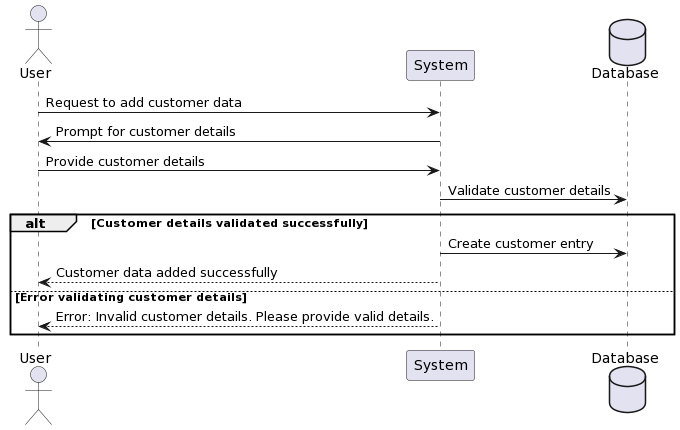
**2.4 LEDGER ENTRY:**

The diagram below shows the process of ledger entry. The user as soon as clicks the add entry, the form will be populated. The user will then need to fill the required details to add the ledger entries.



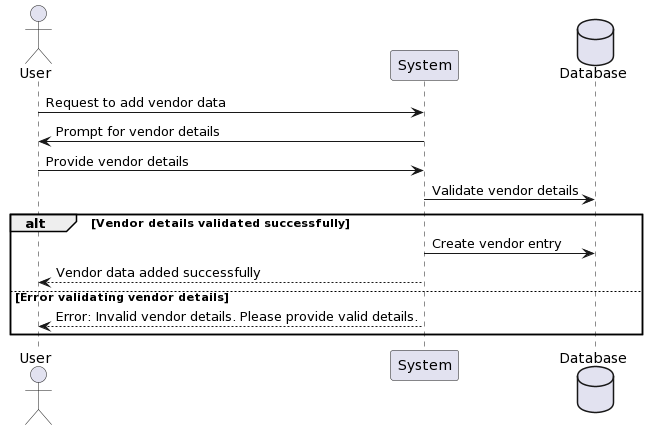
**2.5 CUSTOMER MANAGEMENT:**

The diagram below shows the process of the customer management. The user will be required to fill the customer data and if requested, the customer details should be displayed.



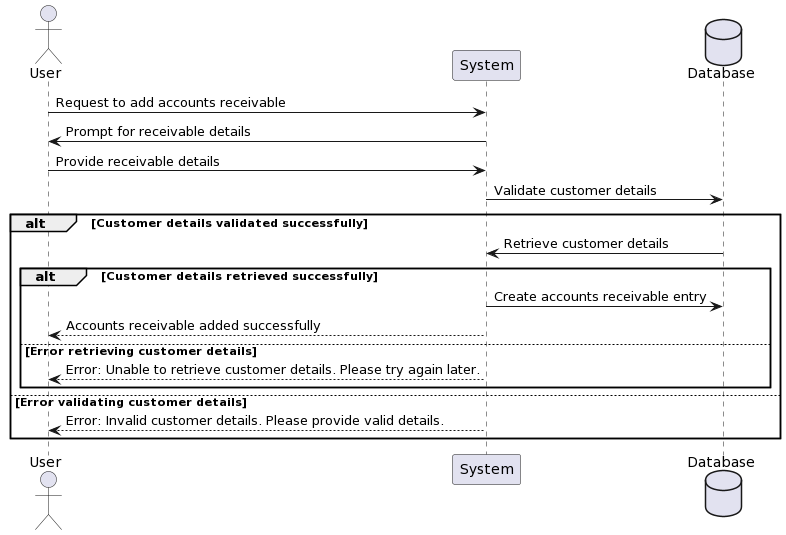
**2.6 VENDOR MANAGEMENT:**

The diagram below shows the process of the vendor management. The user will be required to fill the vendor data and if requested, the vendor details should be displayed.



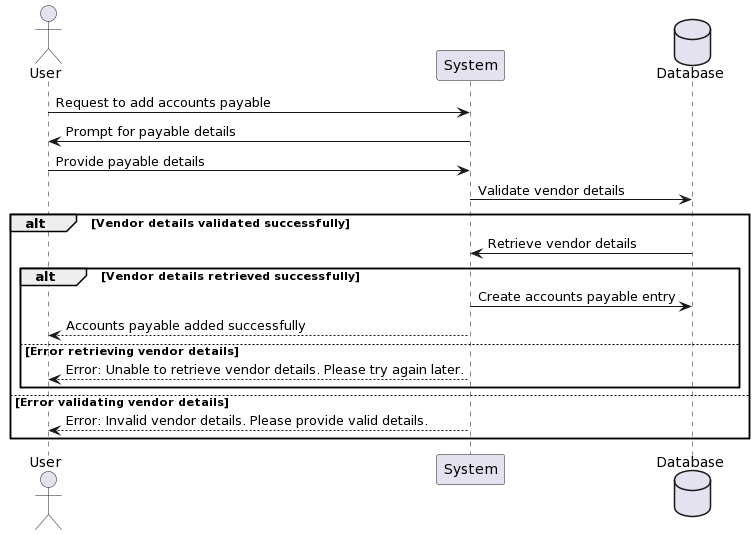
**2.7 ACCOUNTS RECEIVABLE:**

The diagram below shows the process of the accounts receivable. The user will be requested to fill the accounts receivable data and if requested, the details should be prompted.



**2.8 ACCOUNTS PAYABLE:**

The diagram below shows the process of the accounts payable. The user will be requested to fill the accounts payable data and if requested, the details should be prompted.



**3.COMPONENT DESIGN:**

Frontend UI Components

• User Interface: Developed using HTML, CSS, and JavaScript framework like Angular.

• Components: Modular components for user authentication, ledger, accounts payable and receivable, customer, and vendor management.

• UI Design: Intuitive and user-friendly design with responsive layouts for various devices.

Backend API Components

• Routing Layer: Handles incoming HTTP requests and routes them to the appropriate controllers.

• Controllers: Contains business logic for user authentication, ledger management, vendor and customer management, and accounts payable and receivable.

• Middleware: Implements middleware functions for request parsing, authentication, error handling, and logging.

• Data Access Layer: Interacts with the database to perform CRUD operations on user data, product data, and order data.

**4.** **DATA DESIGN**

**4.1 List of Key Schemas/Tables in the Database:**

In the Accounting software Platform, the database schema is meticulously designed to accommodate various entities and relationships essential for the platform's functionalities. The key schemas/tables within the database include:

1. Users: Stores information about registered users, including their profile details, authentication credentials, and preferences.

2. Companies: Stores information regarding the company’s details, transactions, accounts payable and receivable, and vendor management.

3. General Ledger:General ledger (G/L) is a complete record of all of your accounting transactions. Whether you use accounting software or handle your accounting manually, you will use a G/L, which represents a repository of all the financial transactions made by your business.

4. Accounts Payable: Accounts payable keeps track of how much money a business is making as well as how much they are spending. Accounts payable also tracks the money a business owes to any creditors or suppliers, and ensures the numbers are accurate and can be paid off on schedule.

5. Accounts Receivable: Account receivable represents money others owe to the business. It belongs under assets on the firm's balance sheet and is a source of short-term cash for a company. For example, a payment for a software service the company produces is part of accounts receivable.

6. Vendor: Stores information’s regarding the vendors including their communication details.

7. Customer: Stores information regarding the customers details.

**4.2 Key Design Considerations in Data Design:**

Several key design considerations are considered during the data design phase of the accounting software Platform:

1. Normalization: Ensuring the database schema is normalized to minimize redundancy and improve data integrity.

2. Scalability: Designing the database schema to scale efficiently with increasing user base and data volume.

3. Performance Optimization: Indexing key fields, optimizing queries, and caching frequently accessed data to enhance database performance.

4. Data Security: Implementing encryption, access control mechanisms, and data masking to protect sensitive user information.

5. Data Consistency: Enforcing referential integrity, transaction management, and concurrency control to maintain data consistency and reliability.

6. Flexibility: Designing the database schema to accommodate future enhancements, new features, and evolving business requirements.

7. Backup and Recovery: Implementing robust backup and recovery strategies to safeguard against data loss and ensure business continuity.

**5. ENTITY RELATIONSHIP DIAGRAM:**

The ERD illustrates the structure of the application's classes and their relationships. It includes classes and essential entities such as users, company, ledger, accounts and so on. Based on these entities and attributes, we can now define the relationships between them:

**A diagram of a company table

Description automatically generated**

**6. SECURITY CONSIDERATION**

• Encryption: Utilizes encryption algorithms to secure sensitive data such as user passwords and payment information.

• Authorization: Enforces role-based access control (RBAC) to restrict user access to authorized functionalities.

• Input Validation: Validates user input to prevent injection attacks and other security vulnerabilities.

• HTTPS: Ensures secure communication between the client and server using HTTPS protocol.

**7. ERROR HANDLING**

• Exception Handling: Implements robust error handling mechanisms to gracefully handle exceptions and errors.

• Error Logging: Logs errors and exceptions to facilitate troubleshooting and debugging.

• User Feedback: Provides informative error messages to users in case of invalid input or unexpected errors.

**8. PERFORMANCE OPTIMIZATION**

• Asynchronous Processing: Utilizes asynchronous processing techniques to offload time-consuming tasks and improve system responsiveness.

• Database Optimization: Optimizes database queries and indexing strategies to minimize query execution time and improve overall database performance.

**9. DEPLOYMENT ARCHITECTURE**

• Cloud Deployment: Hosts the application on cloud platforms such as AWS, Azure, or Google Cloud for scalability and reliability.

• Containerization: Uses containerization technologies like Docker for packaging and deploying the application components.

• Load Balancing: Implements load balancing mechanisms to distribute incoming traffic across multiple servers and improve scalability and fault tolerance.

**10. CONCLUSION**

This Low-Level Design (LLD) document provides a detailed overview of the e-commerce web application's design and implementation considerations. It covers various aspects such as class diagrams, sequence diagrams, database schema, component design, security considerations, error handling, performance optimization and deployment architecture.