**Low Level Design Document**

**PHARMACY**

DATE: 29-02-2024

HIMANSHU KAKKAR

WHAT IS LOW LEVEL DESIGN ?

• Low-level design refers to the process of specifying and defining the detailed design of a software system. This type of design focuses on the implementation details of a system and is concerned with how the system will be built and how it will function at a detailed level. It provides the foundation for high-level design, which defines a system's overall architecture and design.

**1. Introduction:** The Pharmacy App is designed to provide a convenient platform for users to browse and purchase pharmaceutical products online. This document outlines the low-level design details of various components of the Pharmacy App.

**2. Architecture Overview:** The Pharmacy App follows a client-server architecture model. The client-side application is developed for both web and mobile platforms, while the server-side components include a database server and application server.

**3. Components:**

**a. Client-side Components:**

* **User Interface (UI):**
  + The UI is developed using HTML, CSS, and JavaScript for the web platform and native UI components for the mobile platform.
  + It provides interfaces for users to register, login, browse products, add products to the cart, and proceed to checkout.
* **Application Logic:**
  + Implemented in JavaScript for web applications and platform-specific languages (e.g., Swift for iOS, Kotlin for Android) for mobile applications.
  + Handles user interactions, validation, and communicates with the server-side components via RESTful APIs.
* **Local Storage:**
  + Used for caching frequently accessed data, such as user authentication tokens, product details, and user preferences.
  + Implemented using browser local storage for web applications and platform-specific mechanisms (e.g., User Defaults for iOS, Shared Preferences for Android) for mobile applications.

**b. Server-side Components:**

* **Application Server:**
  + Developed using a framework like Express.js for Node.js or Flask for Python.
  + Handles client requests, performs business logic, and interacts with the database server.
* **Database Server:**
  + Stores user data, product details, order information, etc.
  + Utilizes a relational database management system (e.g., PostgreSQL, MySQL) for structured data storage.
  + Tables include User, Product, Order, Order Item, etc.
  + Ensures data integrity, consistency, and security through proper indexing, normalization, and access control mechanisms.
* **RESTful APIs:**
  + Exposed by the application server to allow communication with client applications.
  + Endpoints include user authentication, product listing, cart management, order placement, etc.
  + Implemented using HTTP methods (GET, POST, PUT, DELETE) for CRUD operations.

**4. Data Flow:**

* **User Registration/Login:**
  1. User provides registration/login details through the UI.
  2. Client application sends the credentials to the server via the appropriate API endpoint.
  3. Server validates the credentials, generates a JWT (JSON Web Token), and sends it back to the client.
  4. Client stores the JWT securely for subsequent authenticated requests.
* **Product Browsing:**
  1. User requests product listing through the UI.
  2. Client application sends a request to the server's product listing API.
  3. Server retrieves product details from the database and sends them back to the client.
  4. Client displays the product listing to the user.
* **Order Placement:**
  1. User adds products to the cart and proceeds to checkout.
  2. Client sends the cart details along with the user's JWT to the server.
  3. Server verifies the user's identity, processes the order, updates the database accordingly, and sends a confirmation response to the client.
  4. Client displays the order confirmation to the user.

**5. Security Considerations:**

* **Authentication & Authorization:**
  + JWT-based authentication mechanism ensures secure user authentication.
  + Access control lists (ACLs) are enforced on server-side endpoints to restrict unauthorized access.
* **Data Encryption:**
  + User credentials, JWT tokens, and sensitive data are encrypted during transmission using HTTPS.
  + Data stored in the database is encrypted using encryption algorithms to prevent unauthorized access.
* **Input Validation:**
  + Client-side and server-side input validation mechanisms are implemented to mitigate injection attacks and data tampering.

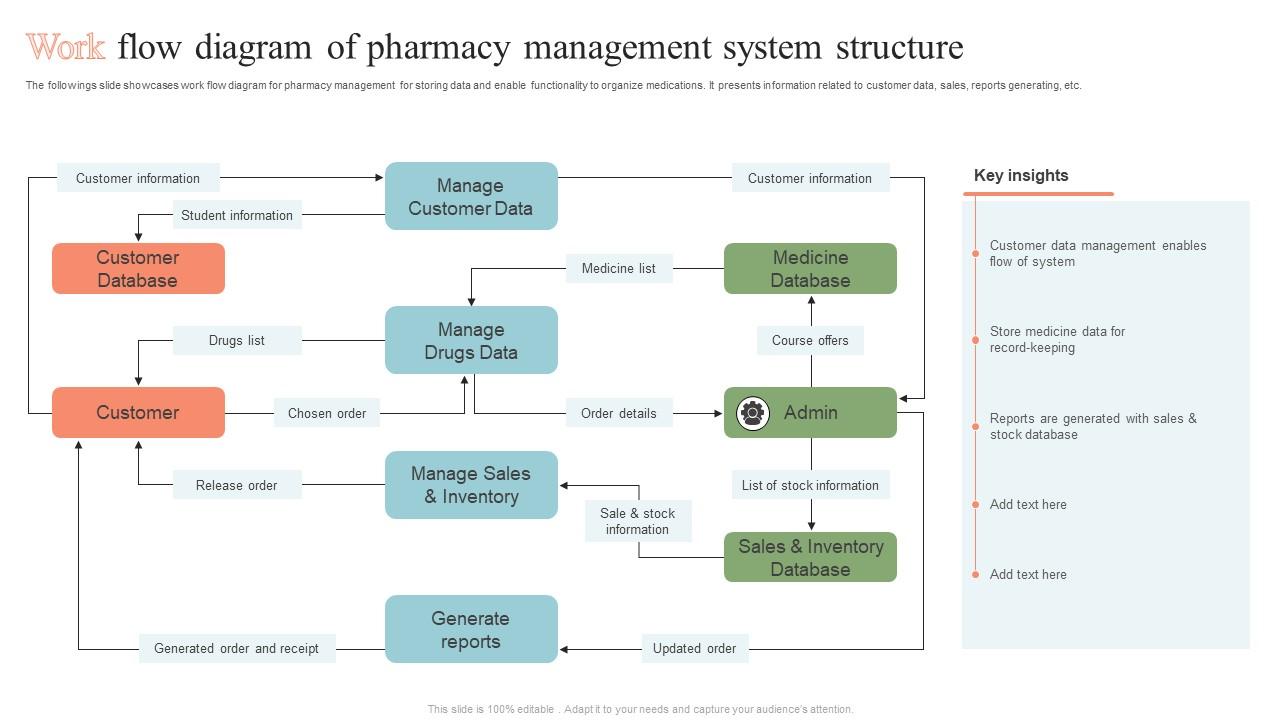
**6. Scalability and Performance:**

* **Horizontal Scaling:**
  + Application servers can be deployed across multiple instances to handle increasing user loads.
  + Load balancers distribute incoming traffic among these instances for improved performance.
* **Caching:**
  + Utilize caching mechanisms (e.g., Redis) to cache frequently accessed data and reduce database load.
  + Implement browser caching for static assets to enhance client-side performance.

**7. Error Handling:**

* **Client-side Errors:**
  + Display user-friendly error messages for common errors (e.g., network issues, validation errors).
* **Server-side Errors:**
  + Implement proper HTTP status codes (e.g., 400 Bad Request, 500 Internal Server Error) along with descriptive error messages.
  + Log server-side errors for troubleshooting and debugging purposes.

**8. Conclusion:** This Low-Level Design Document provides a detailed overview of the architecture, components, data flow, security considerations, scalability, and error handling strategies for the Pharmacy App. By following these design guidelines, the development team can effectively implement and maintain a robust and secure pharmacy application.



THANK YOU