UPES

**Internship - High Level Design**

**on**

**Cloud-based Customer Relationship Management**

**BACHELOR OF TECHNOLOGY**

CLOUD COMPUTING & VIRTUALIZATION TECHNOLOGY(CCVT)

SUBMITTED BY:

|  |  |  |
| --- | --- | --- |
| **Specialization** | **Roll Number** | **Name** |
| CCVT | R2142211066 | Rohan Bakshi |
| CCVT | R2142211034 | Vilish Kumar |
| CCVT | R2142211041 | Harsh Chaudhary |
| CCVT | R2142211045 | Lavanya Raj |
| CCVT | R2142211073 | Sidharth Bansal |

GUIDED BY:

Dr. Lalit Kumar

**Table of Contents**

Introduction 3

Scope of the document 3

Intended Audience 3

System overview 3

System Design 4

Application Design 4

Process Flow 5

Information Flow 6

Components Design. 6

Key Design Considerations 6

API Catalogue 7

Data Design 8

Data Model 8

Data Access Mechanism 9

Data Retention Policies 9

Data Migration. 9

Interfaces 10

State and Session Management 10

Caching 11

Non-Functional Requirements 12

Security Aspects 12

Performance Aspects 12

References 13

# Introduction

* 1. **Scope of the document**

**Cloud-based Customer Relationship Management (CRM)** system developed using PHP, MySQL, Bootstrap, DataTables, JavaScript, and jQuery. It provides comprehensive guidance on installing, configuring, and utilizing the system for both administrators and customers.

The CRM system includes features for administrators such as logging in and out, accessing a dashboard with data summaries and visitor graphs, managing tickets and customers' quotation requests, viewing users' access logs, and managing user accounts. For customers, the system offers functionalities like registration, logging in, accessing a personal dashboard, creating and viewing tickets, requesting quotations, and managing profiles.

* + - **Registration and Login**: Allows customers to create an account and log in to access the system.
    - **Dashboard Access**: Provides a summary view for both admins and customers with relevant data and graphs.
    - **Ticket Management**: Enables customers to create tickets and view their details, while admins can manage and resolve these tickets.
    - **Quotation Request**: Customers can request quotations, and admins can manage and respond to these requests.
    - **Profile Management**: Allows customers to view and update their personal information and change their passwords.
    - **User Management(Admin)**: Admins can manage registered users' accounts, including updating user information and changing passwords.
    - **Access Logs(Admin)**: Admins can view logs of all users' access activities for monitoring and security purposes.

# System overview

# This cloud-based Customer Relationship Management (CRM) system, built with PHP, MySQL, Bootstrap, DataTables, JavaScript, and jQuery, streamlines customer and ticket management. Administrators can log in, access a data summary dashboard, manage tickets and customer quotation requests, view access logs, and handle user accounts. Customers can register, log in, access their dashboard, create and view tickets, request quotations, and manage their profiles. The system is installed by cloning the repository, importing the SQL file into MySQL, updating `config.php` with database details, and starting a local server, making it accessible via a web browser.

# System Design

* 1. **Application Design Frontend:**

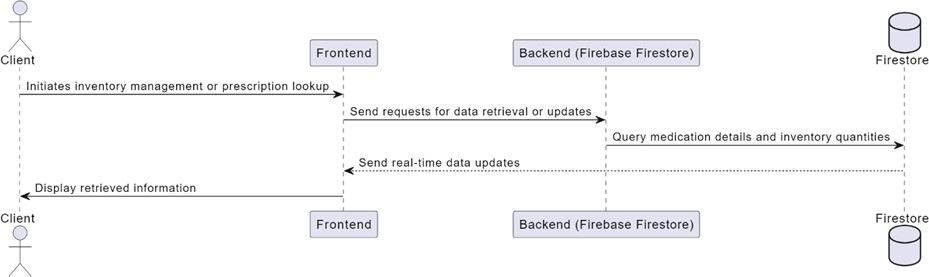
The frontend of the application is designed to provide an engaging and responsive user experience. The technologies used are:

* **HTML:** Structures the web pages, providing the skeleton for content. It is used to create the essential elements of the web pages, ensuring that the content is well-organized and accessible.
* **CSS:** Utilizes custom stylesheets to style the user interface, ensuring a visually appealing and cohesive design.
* **JavaScript:** Adds dynamic functionality and interactivity to the web pages through scripts. JavaScript frameworks and libraries. These scripts handle tasks such as form validation, data fetching, and DOM manipulation, providing a smooth user experience.
* **BootStrap:** Bootstrap is used in this CRM system to ensure a responsive and visually appealing user interface. It provides pre-designed components and a grid system, simplifying the design process and enhancing user experience across various devices.

# Backend:

The backend of the application is built to handle data management, business logic, and real-time synchronization. The technologies used are:

* **PHP:** PHP processes server-side logic, handles form submissions, interacts with the MySQL database for CRUD operations, and dynamically generates HTML content for the CRM system, ensuring seamless functionality and data management.
* **MySQL:** CRM system, PHP interacts with MySQL to perform essential operations such as storing and retrieving user data, tickets, and quotations. It processes SQL queries to create, read, update, and delete records, ensuring efficient data management and providing dynamic content to users.



# Process Flow

The interactions between various components of the system from the initial user request to the final response. The main processes include:

1. **User Registration:** Customer registers an account through a registration form. PHP processes the form data and stores the user information in the MySQL database.
2. **User Login:** Customer or admin logs in using their credentials, PHP verifies the credentials against the MySQL database and starts a session if valid.
3. **Dashboard Access:** The dashboard displays relevant data and statistics using PHP and Bootstrap for a responsive interface.
4. **Response Delivery:** Processed data is sent back to the frontend, updating the user interface in real-time.
5. **Ticket Management:** Admins can view and manage these tickets from the admin dashboard.
6. **Quotation Request:** Admins review and respond to these requests from the admin dashboard.
7. **User Management(Admin):** Admins can manage user accounts, including updating information and changing passwords.

The interactions between user and the system. The main processes include:

1. **Authentication and Registration:** New users register by providing necessary credentials via the frontend interface. Returning users log in, and their credentials are validated through Firebase Authentication.
2. **Manage Inventory:** Users can view the current inventory levels through the frontend interface.
3. **Add New Items: Create New Inventory Item:** Users add new items to the inventory by submitting details through the frontend interface. The new item details are sent to Firestore, where they are stored and synchronized in real-time.
4. **Customer Order Prescription:** Customers submit their prescription orders through the frontend interface. The system validates the prescription details, ensuring they meet required criteria.
5. **Order Submission:** Validated orders are processed, and the details are sent to Firestore. Order status is updated in real-time, and users are notified of the successful submission.

# Information Flow

The flow is as follows:

1. **Data Input:** Users input data (e.g., inventory updates, prescriptions) via the frontend interface.
2. **Data Transmission:** Input data is transmitted to Firebase for processing.
3. **Data Storage:** Firestore stores the data, ensuring real-time synchronization across all connected devices.
4. **Data Retrieval:** Requests for data retrieval (e.g., checking inventory levels) are processed by Firestore and sent back to the frontend.
5. **Data Display:** Retrieved data is displayed on the user interface, providing real-time updates to the users.

# Components Design

Key components include:

# User Interface Components:

* + **Login and Authentication:** Ensures secure access to the system.
  + **Dashboard:** Displays key metrics and real-time data.
  + **Inventory Management:** Allows users to manage pharmaceutical stocks.
  + **Customer Management:** Facilitates the handling of prescriptions.

# Backend Components:

* + **Database Schema:** Designed in Firestore for efficient data normalization and retrieval.
  + **API Endpoints:** Facilitates communication between the frontend and backend.

# Key Design Considerations

Several key considerations were taken into account during the design process:

1. **User Experience:** Ensuring the interface is intuitive and easy to navigate for healthcare providers.
2. **Scalability:** Designing the system to handle increased load and data volume as the user base grows.
3. **Security:** Implementing robust security measures to protect sensitive data and comply with regulatory standards.
4. **Real-Time Updates:** Leveraging Firebase's real-time capabilities to ensure timely data synchronization across all devices.
5. **Reliability:** Ensuring the system remains operational with minimal downtime.

# API Catalogue

The API catalogue outlines the available endpoints for interaction between the frontend and backend. Key APIs include:

# User Authentication:

* **Endpoint:** Firebase Authentication API
* **Method:** Manages user authentication (sign-up, sign-in, etc.).
* **Description:** Facilitates secure user authentication and management within your application.

# Add New Stock to Inventory:

* **Endpoint:** Firestore Database API
* **Method:** Allows Create, Read operations for inventory data.
* **Description:** Enables adding new stock items or updating existing ones in the Firestore database.

# Fetching Real-Time Updates:

* **Endpoint:** Firestore Database API
* **Method:** Utilizes real-time listeners.
* **Description:** Enables real-time synchronization of data updates from Firestore to client applications, ensuring live updates without the need for manual refreshes.

# Data Design

The data design for PharmaELEVATE revolves around efficient storage, retrieval, and management of pharmaceutical inventory, prescriptions, and user data. The system leverages Firebase Firestore, a NoSQL database, to support real-time synchronization and offline capabilities, ensuring seamless operation and data consistency across all devices.

# Data Model

The data model in Firestore is designed to reflect the entities and their relationships within the healthcare logistics management system. Key collections and documents include:

* + - **Users:** Stores information about pharmacists, healthcare providers, administrators, and customers.

Fields: userId, name, email, role, credentials, contactInfo

* + - **Medications**: Stores data about each medication available in the inventory. Fields: medicationId, name, description, quantity, expiryDate, supplierInfo
    - **Prescriptions:** Manages prescription details for customers.

Fields: prescriptionId, customerId, medicationList, issueDate, dueDate, status

* + - **Orders:** Tracks orders placed by customers for medications.

Fields: orderId, customerId, orderDate, pickupDate, orderStatus, medicationsOrdered

# Data Access Mechanism

PharmaELEVATE uses Firebase's Firestore to manage data access. The interaction with the Firestore database is primarily handled through Firebase's SDK, which provides APIs for CRUD (Create, Read, Update, Delete) operations. Key mechanisms include:

* + - **Real-time Listeners**: Firestore supports real-time listeners that update the client-side application whenever data changes. This ensures that users always have access to the latest information without needing to refresh the application manually.
    - **Queries**: Complex queries can be executed to retrieve specific data sets, such as filtering medications by expiry date or searching for prescriptions by customer ID.
    - **Transactions**: For operations that require consistency, such as updating inventory levels, Firestore transactions ensure that multiple read and write operations are executed atomically.

# Data Retention Policies

Key policies include:

* + - **User Data:** Retained for the duration of the user's active engagement with the system. Data is archived upon account deactivation and purged after a specified period.
    - **Medication Records:** Maintained for as long as the medication is available in inventory and required by regulatory standards. Expired or obsolete medications are archived.
    - **Prescriptions and Orders:** Retained for a period specified by healthcare regulations (e.g., 5-7 years) to ensure compliance and support auditing and reporting requirements.

# Data Migration

Key considerations include:

* + - **Schema Mapping**: Mapping existing data schemas to Firestore's document-based model. This involves defining collections, documents, and fields that correspond to existing database tables and records.
    - **Data Import**: Using Firebase's data import tools or custom scripts to transfer data into Firestore. This process ensures data integrity and consistency.
    - **Validation and Testing**: Ensuring that the migrated data is accurate and complete. This involves running validation checks and conducting thorough testing to verify that the new system functions correctly with the imported data.
    - **Rollback Plan**: Having a contingency plan in place to revert to the previous system in case of migration failures or data discrepancies.

# Interfaces

**User Interfaces (UI)**

1. **Login Page:** Allows users (pharmacists, healthcare providers, administrators) to authenticate and access the system securely.
2. **Dashboard:** Provides an overview of key metrics such as inventory levels, prescription status, and alerts. Allows users to navigate to different sections like Inventory Management and Prescription Handling.
3. **Inventory Management:** Enables pharmacists and administrators to:
   1. View current stock levels of medications.
   2. Add new medications or edit existing ones.

# Backend Interfaces

**a) Firebase Firestore:** Interface for storing and managing data related to users, medications, prescriptions, and transactions. Supports real-time data synchronization for seamless updates across the application.

* PharmaELEVATE integrates a user-friendly interface for pharmacists and healthcare providers to manage medications and prescriptions efficiently. Powered by Firebase, it ensures real-time data updates and secure interactions. With interfaces like the dashboard, inventory management, and prescription handling, it enhances operational oversight and patient care across healthcare facilities

.

# State and Session Management

* **State Management:** In PharmaELEVATE, state management ensures consistent and synchronized data across its components. Using Firebase Firestore's real-time database capabilities, the application maintains a single source of truth for medication inventory, prescriptions, and user data. Changes made by users in the frontend interfaces, such as adding medications or updating inventory quantities, trigger immediate updates across all connected clients. This approach ensures that all users, whether accessing the dashboard, managing inventory, or handling prescriptions, see the most current and accurate information. Firebase's robust state management capabilities thus optimize data integrity and operational efficiency throughout the healthcare logistics management process.
* **Session Management:** In PharmaELEVATE, session management leverages Firebase Authentication's built-in mechanisms. Users authenticate using their email and password, which Firebase verifies against its user database securely. Once authenticated, Firebase maintains the user's session state internally, allowing them seamless access to authorized parts of the application without frequent re-authentication. Sessions are managed securely within Firebase's infrastructure, ensuring that user interactions with sensitive data like inventory management, prescription handling, and user settings are protected. Firebase Authentication automatically handles session expiration and renewal, based on configured settings, to enhance security and user experience without the explicit use of tokens.

# Caching

Types of Caching in PharmaELEVATE

1. Client-Side Caching:
   * Browser Storage: Use local storage, session storage, or IndexedDB to store frequently accessed data like user session details and configuration settings.
   * Service Workers: Cache static assets (HTML, CSS, JavaScript) and enable offline functionality, improving load times and user experience.
2. Server-Side Caching:
   * In-Memory Databases: Utilize caching solutions like Redis or Memcached to store frequently accessed data in memory, reducing the number of reads from Firestore.
   * Firebase Caching: Leverage Firebase’s built-in caching capabilities to cache query results and minimize database reads.
3. Edge Caching:
   * Content Delivery Networks (CDNs): Use CDNs to cache static assets closer to end-users, reducing latency and improving load times by serving content from the nearest edge server.

Implementing Caching in PharmaELEVATE

* Client-Side Caching: Implement service workers to cache static assets and enable offline access, and utilize browser storage for user session data.
* Server-Side Caching: Use Firebase’s caching features and integrate an in-memory caching layer for frequently accessed data.

Benefits of Caching

* Improved Performance: Faster data retrieval and reduced latency result in a more responsive application.
* Reduced Server Load: Decreased number of requests to the backend database improves scalability and reduces costs.
* Enhanced User Experience: Quicker load times and offline capabilities provide a smoother and more reliable experience for users.

# Non-Functional Requirements Security Aspects

1. **Data Protection**:
   * **Encryption**: Implement SSL/TLS for data in transit to encrypt data between the client and server. Use Firestore’s built-in encryption for data at rest to protect sensitive information.
   * **Access Control**: Utilize Firebase Firestore Security Rules to define and enforce access controls, ensuring only authorized users can access or modify data.
   * **Authentication**: Leverage Firebase Authentication to securely manage user authentication and authorization. Ensure strong password policies and support multi-factor authentication (MFA) for added security.

# Compliance:

* + **Regulatory Standards**: Ensure the application complies with relevant healthcare regulations, such as HIPAA in the US, for handling medical information. Implement data protection and privacy measures to meet these standards.
  + **Audit Trails**: Maintain detailed logs of user activities, including data access and modifications, to support auditing and compliance requirements.

# Performance Aspects

1. **Scalability**:
   * **Horizontal Scaling**: Design the system to support horizontal scaling, allowing it to handle increased loads by adding more instances of services. Leverage Firebase’s scalable infrastructure to manage growing data volumes and user traffic.
   * **Load Balancing**: Use load balancing techniques to distribute incoming requests evenly across multiple servers, ensuring efficient resource utilization and preventing overload on any single server.

# Response Time:

* + **Optimized Queries**: Optimize database queries to reduce response times and improve data retrieval efficiency. Use indexed queries and avoid complex joins to speed up database operations.
  + **Asynchronous Processing**: Implement asynchronous processing for time- consuming tasks to prevent blocking the main application flow, ensuring quicker response times for users.

# Reliability:

* + **High Availability**: Design the system to ensure high availability, aiming for an uptime of 99.9% or higher. Use Firebase Hosting’s reliable infrastructure and implement redundancy and failover mechanisms.
  + **Error Handling and Recovery**: Implement robust error handling to manage and recover from failures gracefully. Ensure the application can handle unexpected issues without significant downtime or data loss.

# Efficiency:

* + **Resource Optimization**: Optimize resource usage to ensure efficient performance. This includes minimizing the use of memory and processing power, and ensuring efficient use of network bandwidth.
  + **Caching**: Implement effective caching strategies, as discussed, to reduce load times and improve data retrieval speeds.

# References

1. Smith, J., Brown, A., & Lee, L. (2021). Cloud-based predictive analytics for demand forecasting in supply chain management.
2. Yang Liu, Feng Wang, Jianjun Li, and Bo Yang(2022). Proposes a cloud-based digital twin technology for predictive maintenance and optimization in smart logistics.
3. Temjanovski, R. (2021). Integration of cloud-based solutions in supply chain management.
4. <https://www.scribd.com/document/256924692/Online-Medical-Store-synopsis> [5] <https://journals.sagepub.com/doi/full/10.1177/21582440221149717>
5. <https://www.sciencedirect.com/science/article/abs/pii/S0360835219305108>
6. <https://www.sciencedirect.com/science/article/abs/pii/S0925527310000320?via%3Dihub>