**Error! Unknown document property name.**

**Error! Unknown document property name.**

High Level Design & Low Level Design

The purpose of this document is to provide with a template for documenting both HLD & LLD.

**Document Control :**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Project Revision History** | | | | | | | | |
|  |  |  | |  |  |  |  |  |
| **Date** | **Version** | **Author** | **Brief Description of Changes** | | | | **Approver Signature** | |
| 12-10-24 | 1 | Team Lead – Sneha Garg | Added use case diagram | | | | Sneha | |
| 12-10-24 | 1 | Anusha Chegoni | Chapter 1 | | | | Anusha | |
| 12-10-24 | 1 | Ananya Mallampati | Chapter 2 | | | | Ananya | |
| 13-10-24 | 1.1 | Team Lead – Sneha Garg | Added DFD diagram | | | | Sneha | |
| 15-10-24 | 1.2 | Team Lead – Sneha Garg | Chapter 3  Added sequence diagram | | | | Sneha | |
| 16-10-24 | 1.3 | Usha Rani Kallepalli | Chapter 4 | | | | Usha | |
| 16-10-24 | 1.3 | Ananya Mallampati | Chapter 4 | | | | Ananya | |
| 16-10-24 | 1.4 | Anusha Chegoni | Chapter 5 | | | | Anusha | |
| 17-10-24 | 1.5 | Maheshwari Ponnagani | Chapter 6,7 | | | | Maheshwari | |
| 17-10-24 | 1.6 | Team Lead – Sneha Garg | Modified DFD diagrams | | | | Sneha | |
| 18-10-24 | 1.7 | Team Lead – Sneha Garg | Modified sequence diagram Added activity diagram | | | | Sneha | |

[1. Introduction 3](#_Toc368912248)

[1.1. Intended Audience 3](#_Toc368912249)

[1.2. Acronyms/Abbreviations 3](#_Toc368912250)

[1.3. Project Purpose 3](#_Toc368912251)

[1.4. Key Project Objectives 3](#_Toc368912252)

[1.5. Project Scope and Limitation 3](#_Toc368912253)

[1.5.1. In Scope 3](#_Toc368912254)

[1.5.2. Out of scope 3](#_Toc368912255)

[1.6. Functional Overview 3](#_Toc368912256)

[1.7. Assumptions, Dependencies & Constraints 3](#_Toc368912257)

[1.8. Risks 3](#_Toc368912258)

[2. Design Overview 3](#_Toc368912259)

[2.1. Design Objectives 3](#_Toc368912260)

[2.1.1. Recommended Architecture 3](#_Toc368912261)

[2.2. Architectural Strategies 3](#_Toc368912262)

[2.2.1. Design Alternative 3](#_Toc368912263)

[2.2.2. Reuse of Existing Common Services/Utilities 3](#_Toc368912264)

[2.2.3. Creation of New Common Services/Utilities 3](#_Toc368912265)

[2.2.4. User Interface Paradigms 3](#_Toc368912266)

[2.2.5. System Interface Paradigms 3](#_Toc368912267)

[2.2.6. Error Detection / Exceptional Handling 3](#_Toc368912268)

[2.2.7. Memory Management 3](#_Toc368912269)

[2.2.8. Performance 3](#_Toc368912270)

[2.2.9. Security 3](#_Toc368912271)

[2.2.10. Concurrency and Synchronization 3](#_Toc368912272)

[2.2.11. Housekeeping and Maintenance 3](#_Toc368912273)

[3. System Architecture 3](#_Toc368912274)

[3.1. System Architecture Diagram. (Not Necessary) 3](#_Toc368912275)

[3.2. System Use-Cases 3](#_Toc368912276)

[3.3. Subsystem Architecture 3](#_Toc368912277)

[3.4. System Interfaces 3](#_Toc368912278)

[3.4.1. Internal Interfaces 3](#_Toc368912279)

[3.4.2. External Interfaces 3](#_Toc368912280)

[4. Detailed System Design 3](#_Toc368912281)

[4.1. Key Entities 3](#_Toc368912282)

[4.2. Detailed-Level Database Design 3](#_Toc368912283)

[4.2.1. Data Mapping Information 3](#_Toc368912284)

[4.2.2. Data Conversion 3](#_Toc368912285)

[4.3. Archival and retention requirements 3](#_Toc368912286)

[4.4. Disaster and Failure Recovery 3](#_Toc368912287)

[4.5. Business Process workflow 3](#_Toc368912288)

[4.6. Business Process Modeling and Management (as applicable) 3](#_Toc368912289)

[4.7. Business Logic 3](#_Toc368912290)

[4.8. Variables 3](#_Toc368912291)

[4.9. Activity / Class Diagrams (as applicable) 3](#_Toc368912292)

[4.10. Data Migration 3](#_Toc368912293)

[4.10.1. Architectural Representation 3](#_Toc368912294)

[4.10.2. Architectural Goals and Constraints 3](#_Toc368912295)

[4.10.3. Logical View 3](#_Toc368912296)

[4.10.4. Architecturally Significant Design Packages 3](#_Toc368912297)

[4.10.5. Data model 3](#_Toc368912298)

[4.10.6. Deployment View 3](#_Toc368912299)

[5. Environment Description 3](#_Toc368912300)

[5.1. Time Zone Support 3](#_Toc368912301)

[5.2. Language Support 3](#_Toc368912302)

[5.3. User Desktop Requirements 3](#_Toc368912303)

[5.4. Server-Side Requirements 3](#_Toc368912304)

[5.4.1. Deployment Considerations 3](#_Toc368912305)

[5.4.2. Application Server Disk Space 3](#_Toc368912306)

[5.4.3. Database Server Disk Space 3](#_Toc368912307)

[5.4.4. Integration Requirements 3](#_Toc368912308)

[5.4.5. Jobs 3](#_Toc368912309)

[5.4.6. Network 3](#_Toc368912310)

[5.4.7. Others 3](#_Toc368912311)

[5.5. Configuration 3](#_Toc368912312)

[5.5.1. Operating System 3](#_Toc368912313)

[5.5.2. Database 3](#_Toc368912314)

[5.5.3. Network 3](#_Toc368912315)

[5.5.4. Desktop 3](#_Toc368912316)

[6. References 3](#_Toc368912317)

[7. Appendix 3](#_Toc368912318)

# 

# Introduction

The Call Data Record (CDR) project is a client-server application designed for cellular service providers to efficiently process billing information from CDR files. It features a menu-driven client interface for user registration, login, and CDR processing, while the server handles requests, stores user data, and processes CDR files using multithreading for concurrent billing tasks. The system utilizes a local file system for data storage (CB.txt for Customer Billing and IOSB.txt for Inter operator Settlement Billing) and incorporates a structured logging mechanism with four levels (FATAL, ERROR, WARNING, INFO, DEBUG) for monitoring. Overall, the project aims to enhance billing management and improve responsiveness through a robust architecture and efficient data handling.

## Intended Audience

This HLD and LLD document is intended for key stakeholders in the Call Data Record (CDR) project. Project Managers will use it to track progress and ensure alignment with objectives. Software Developers will refer to it for guidance on coding and integration. System Architects will review architectural decisions for performance and scalability. Quality Assurance Engineers will develop test cases for validation, while Technical Writers will create user documentation based on the specifications. Overall, this document serves as a comprehensive guide to facilitate collaboration among all project participants.

|  |  |
| --- | --- |
| PROJECT MANAGERS | To monitor project progress and alignment with goals. |
| SOFTWARE DEVELOPERS | To guide coding, testing, and integration of the client-server application. |
| SYSTEM ARCHITECTS | To review architectural decisions and ensure performance requirements. |
| QUALITY ASSURANCE ENGINEERS | To develop test cases and validate system functionalities. |
| TECHNICAL WRITERS | To create user documentation based on design specifications. |

## Acronyms/Abbreviations

|  |  |
| --- | --- |
| **ACRONYMS** | **ABBREVIATIONS WITH DESCRIPTION** |
| CDR | Call Data Record – A record containing details of charging events such as voice calls and SMS messages for billing purposes. |
| CLIENT | The application that interacts with the server, allowing users to perform operations like SignUp, Login, and CDR processing. |
| SERVER | The backend application that processes requests from the client, manages user data, and handles CDR file processing. |
| MSISDN | Mobile Station International Subscriber Directory Number – The phone number used to identify a mobile subscriber. |
|  |  |
|  |  |
| MMC/MNC | Mobile Country Code / Mobile Network Code – Identifiers used to distinguish mobile operators and their networks. |
| STL | Standard Template Library – A collection of C++ template classes to provide general-purpose classes and functions. |
| FATAL | A log level indicating critical errors that cause system failure. |
| ERROR | A log level for non-critical issues requiring attention. |
| WARNING | A log level indicating potential issues that may affect performance. |
| INFO | A log level for general operational messages. |
| DEBUG | A log level for detailed diagnostic messages useful during development and troubleshooting. |

## Project Purpose

The Call Data Record (CDR) project aims to establish a robust client-server application designed for cellular service providers to efficiently manage and process CDR files. This system facilitates essential functionalities, including user registration, authentication, and comprehensive billing management by handling various charging events, such as voice calls and SMS messages. By leveraging multithreading, the project will enable simultaneous processing of Customer Billing and Interoperator Settlement Billing, thereby optimizing performance and enhancing responsiveness. Ultimately, this initiative seeks to improve billing accuracy and streamline user interactions through efficient data retrieval and reporting capabilities.

## Key Project Objectives

The key objectives of the Call Data Record (CDR) project are as follows:

* **User Management:** Implement a secure client-server connection that allows users to register and authenticate through a streamlined SignUp and Login process.
* **CDR Processing:** Enable efficient processing of CDR files for both Customer Billing and Interoperator Settlement Billing using multithreading to enhance performance and reduce processing time.
* **Data Storage:** Utilize STL containers to effectively manage and store processed billing information for easy retrieval and reporting.
* **User Interaction:** Provide a user-friendly interface for clients to access, search, and print billing information, including detailed records for specific MSISDNs and operator IDs.
* **Logging and Monitoring:** Integrate a comprehensive logging mechanism with multiple levels (FATAL, ERROR, WARNING, INFO, DEBUG) to facilitate monitoring and troubleshooting.
* **Local File System Integration:** Ensure that all user data and billing information are securely managed within the local file system for accessibility and data integrity.

## Project Scope and Limitation

**Project Scope:**

* **Client-Server Architecture:** Establish a TCP connection for client-server communication.
* **User Management:** Implement sign-up and login functionalities, storing user data in local files.
* **CDR Processing:** Process Call Data Records for Customer Billing and Interoperator Settlement Billing using multithreading.
* **Client Operations:** Provide a menu for processing CDR files and printing/searching billing information.
* **Logging:** Integrate logging with four levels: FATAL, INFO, WARNING, DEBUG.
* **File Management:** Handle input CDR files and output processed data files (CB.txt, IOSB.txt).
* **Graceful Shutdown:** Ensure proper closure of client and server sockets.
* **Documentation:** Provide clear setup and usage instructions.

**Limitations:**

* **Local File Database:** User data stored in local files limits scalability.
* **Single Server Instance:** Only one server instance may lead to performance issues under heavy load.
* **Data Concurrency:** Multithreading may lead to race conditions without proper management.
* **Error Handling:** Limited validation for user input could result in errors.
* **No GUI:** Command-line interface may be less user-friendly.
* **Network Dependency:** Relies on stable network connectivity.
* **Limited Search Functionality:** Simplified search capabilities may not cover all scenarios.
* **CDR File Format:** Assumes specific CDR file formats, risking processing errors.
* **Resource Management:** Multithreading may lead to resource exhaustion if not managed.
* **Logging Overhead:** Extensive logging can affect performance.

### In Scope

The CDR Processing Project includes:

* **Client-Server Architecture**: Establish a connection for seamless communication with a user-friendly client interface.
* **User Management**:
* **SignUp**: Create and securely store user accounts.
* **Login**: Authenticate users to access system features.
* **CDR File Processing**:
* Use multithreading for simultaneous customer and interoperator billing processing.
* Store processed data in STL containers.
* **Billing Information Management**:
* Search and print billing information by MSISDN and Operator MMC/MNC.
* Output processed data to CB.txt and IOSB.txt.
* **Logging**: Implement logging with four levels (FATAL, INFO, WARNING, DEBUG) for monitoring and troubleshooting.
* **Client-Socket Management**: Properly manage and close client sockets post-request and user exit.

### Out of scope

The project excludes the following elements: real-time CDR data collection from network operators, integration with external billing systems, and advanced analytics or data visualization features. It will not support multi-user concurrent access beyond the basic client-server interactions, nor will it develop mobile applications or cloud-based solutions for CDR management. Additionally, enhanced security protocols beyond standard authentication and logging mechanisms are not included. This delineation ensures a focused approach to delivering the core functionalities of the CDR processing system.

## Functional Overview

The CDR Processing System is designed to facilitate charging management for cellular service providers by efficiently handling Call Data Records (CDRs). It establishes a client-server architecture that allows users to sign up, log in, and access billing functionalities. Upon successful authentication, users can process CDR files for customer and interoperator billing through multithreading, optimizing processing time. The system provides options for users to search billing information based on MSISDN or operator identifiers and to retrieve detailed billing data in specified text files (`CB.txt` and `IOSB.txt`). Additionally, the server logs events at multiple severity levels (FATAL, INFO, WARNING, DEBUG) to ensure system monitoring and maintain data integrity. This comprehensive approach ensures robust performance while addressing the operational needs of cellular service providers.

## Assumptions, Dependencies & Constraints

This project assumes that users have basic familiarity with client-server applications and C programming. It relies on a stable local file system for data storage and assumes that the operating environment will support multithreading capabilities. The system is designed for use on standard operating systems, such as Linux or Windows, that support C development. Dependencies include the availability of appropriate libraries for socket programming and STL container usage. Constraints may arise from limitations in data handling capacity and processing speed, particularly with large CDR files. Additionally, any future changes in functionality will require careful consideration of user needs and system performance to ensure seamless integration and operation.

## Risks

Several risks have been identified in relation to the CDR Processing Project:

* **Software Risks**: Potential bugs in the multithreading implementation could lead to race conditions or deadlocks, affecting system stability and data integrity.
* **Hardware Risks**: Insufficient server resources (CPU, memory) may hinder the performance of concurrent processes, especially with large CDR files.
* **Operating System Risks**: Compatibility issues may arise with different operating systems, potentially affecting socket communication and file handling.
* **User Risks**: Users may encounter difficulties with the interface or functionalities due to a lack of familiarity with client-server applications, leading to improper usage.
* **Data Risks**: Corruption or loss of CDR data during file processing could compromise billing accuracy and reliability.
* **Security Risks**: Vulnerabilities in the authentication process or data transmission may expose sensitive user information, necessitating robust security measures.

# Design Overview

The Call Data Record (CDR) project is designed to serve cellular service providers by efficiently managing and processing call data records. This software system provides essential functionalities for user management, CDR processing, and billing information retrieval, all while ensuring robust performance through multithreading and a clear client-server architecture.

**2.1. Design Objectives**

The primary objective of the CDR project is to provide an efficient system for managing call data records for cellular service providers. The system must enable user authentication, process CDR files, and facilitate billing information retrieval while ensuring concurrent processing through multithreading. Key design objectives include:

* **User Management**: Enable user signup, login, and session management.
* **CDR Processing**: Efficiently process CDR files for billing information using multithreading.
* **Data Retrieval**: Provide options for users to search and retrieve billing data.
* **Robust Logging**: Implement a logging mechanism with multiple levels for error tracking and debugging.

**2.1.1. Recommended Architecture**

The system will follow a client-server architecture, where the server handles requests and processes CDR files, while the client provides the user interface. The components of the architecture are:

* **Client**: Responsible for user interactions and displaying menus.
* **Server**: Handles client requests, processes CDR files, and manages user data.
* **Database**: Local file system to store user information and processed CDR data.

**2.2. Architectural Strategies**

The architectural strategies for the Call Data Record (CDR) project encompass key design decisions and abstractions that influence the organization and structure of the system. These strategies aim to ensure the system meets functional requirements, maintains performance, and provides a robust user experience.

**2.2.1. Design Alternatives**

Several design alternatives were considered:

* **Single-threaded vs. Multithreaded Processing**: Multithreading was chosen to improve performance and responsiveness when processing CDR files, allowing customer billing and inter operator billing to be processed simultaneously.
* **In-memory Database vs. File System**: A local file system was selected for its simplicity and ease of implementation, given the project scope.

**2.2.2. Reuse of Existing Common Services/Utilities**

The system will utilize:

* **Logging Utility**: A custom logging module to handle different log levels (FATAL, INFO, WARNING, DEBUG).
* **Networking Libraries**: Standard socket programming libraries for client-server communication.

**2.2.3. Creation of New Common Services/Utilities**

New services to be created include:

* **User Management Service**: Handles user signup, login, and session management.
* **CDR Processing Service**: Manages the logic for processing CDR files and generating billing information.

**2.2.4. User Interface Paradigms**

The user interface will be text-based, presenting users with menus to navigate through options. Each menu will be displayed after a user action, prompting the user to select a choice.

**2.2.5. System Interface Paradigms**

The system will employ a client-server communication model using TCP sockets. The client will send requests to the server, which will respond with processed data or status messages.

**2.2.6. Error Detection / Exception Handling**

* **Error Logging**: All critical errors will be logged using the logging utility.
* **Input Validation**: User inputs will be validated to prevent invalid operations.
* **Graceful Degradation**: The system will handle exceptions gracefully, providing meaningful messages to the user.

**2.2.7. Memory Management**

* **Dynamic Memory Allocation**: Careful use of malloc and free to manage memory for user data and CDR records.
* **Memory Leaks Prevention**: Use of tools like Valgrind to identify and fix memory leaks.

**2.2.8. Performance**

* **Multithreading**: Enables simultaneous processing of billing tasks, improving overall system responsiveness.
* **Efficient Data Structures**: Use of STL containers (like vectors and maps) for fast data access and manipulation.

**2.2.9. Security**

* **User Authentication**: Passwords will be securely stored and checked during login.
* **Data Validation**: Inputs will be sanitized to prevent injection attacks.
* **Secure Communication**: While not implemented in this initial version, consider TLS for encrypting client-server communications in future iterations.

**2.2.10. Concurrency and Synchronization**

* **Mutexes and Condition Variables**: These will be used to manage shared resources and ensure thread safety during CDR processing.

**2.2.11. Housekeeping and Maintenance**

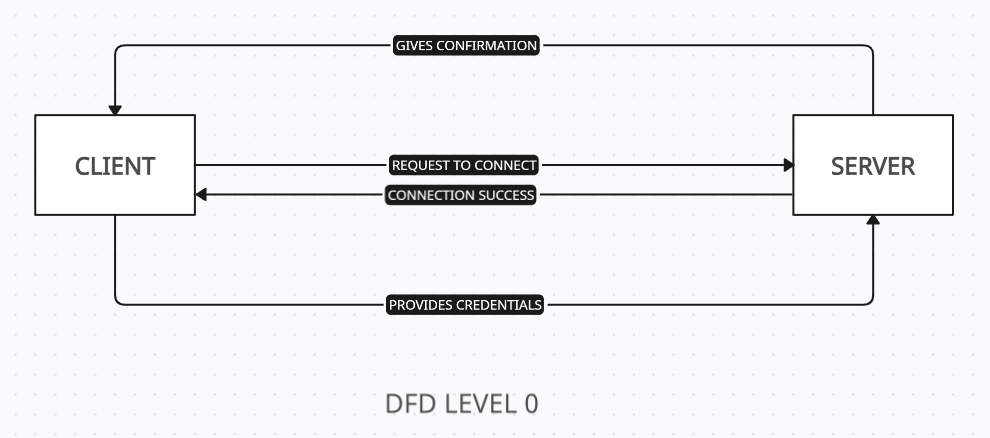
* **Log Management**: Implement periodic log cleanup to prevent excessive disk usage.
* **Data Purging**: Regular purging of outdated records from the user database to maintain performance.

# System Architecture

The CDR Management System is structured to efficiently handle the charging functionality for cellular service providers, focusing on processing CDR files and generating billing outputs. Its primary responsibilities include ingesting CDR files, validating data, calculating charges based on usage, storing processed records, and generating reports for billing and inter-operator accounting.

## System Architecture Diagram

**Data Flow Diagrams:**



A diagram of a server

Description automatically generated

**Sequence Diagram:**

A diagram of a customer data

Description automatically generated

**Use case Diagram:**

A diagram of a person with text

Description automatically generated

## 3.1.1 System Use-Cases

* **Signup**: Clients create a new account by providing necessary information (e.g., email, password) enabling access to the application.
* **Login**: Clients authenticate their identity by entering credentials (e.g., email, password), granting access to their account and personalized features.
* **Display Menu**: The application presents users with a navigation menu, showcasing available options and features for easy access and interaction.
* **Send Request**: Clients initiate actions (e.g., data retrieval, updates) by sending requests to the server, typically through forms or buttons.
* **Process CDR Records**: The server handles Call Detail Records (CDR) by analyzing and storing call data, enabling insights and reporting on call activities.
* **Receive Response**: The server sends back a response to the client after processing requests, which may include success messages, data, or error notifications.
* **Display Data**: The application presents the received data from the server to clients in a structured format (e.g., tables, charts), facilitating easy understanding and interaction.

## Subsystem Architecture

The Client-Server CDR Records Processing subsystem is designed to efficiently handle and manage CDR records in a distributed architecture. In this setup, the **Client** component is responsible for sending CDR files to the **Server**, where the actual processing occurs.  
  
This architecture allows for efficient scalability, as multiple clients can send CDR files simultaneously to a centralized server, ensuring streamlined processing and quick turnaround times for billing and reporting. The interaction between clients and the server promotes a clear separation of responsibilities, enhancing system maintainability and performance.

## System Interfaces

* The CDR Management System comprises several critical interfaces that facilitate the processing of CDR records in a client-server architecture.
* The **Client-Server Interface** allows client applications to securely send CDR files to the server using HTTP/HTTPS protocols, with data formatted in JSON or plain text.
* The **Data Validation Interface** then ensures that each CDR record adheres to the required format and structure.
* Once the calculations are complete, the processed data is sent to the **Data Storage Interface**, which archives the records and calculated charges for future reference.
* Finally, the **Output Generation Interface** compiles and produces billing statements from the stored data.

### Internal Interfaces

* The CDR Management System features several key internal interfaces that facilitate communication between its components.
* The **Data Storage to Output Generation Interface** allows the Output Generation Module to access stored data needed for generating billing reports, typically activated at the end of processing cycles, such as monthly or quarterly.
* Additionally, the **Audit Logging Interface** continuously logs significant actions and calculations performed by the system for compliance purposes, providing real-time monitoring of processes.

### External Interfaces

* The CDR Management System features several crucial external interfaces that enable seamless interaction with outside systems.
* The **Client Interface** allows applications to send CDR files to the server via HTTP/HTTPS, with multiple clients able to transmit data simultaneously, often on an hourly or daily basis.
* The **Billing System Interface** integrates with external billing systems to generate customer invoices, typically invoked at the end of each billing cycle.
* The **Third-Party Operator Interface** facilitates inter-operator accounting by sharing CDR data, usually monthly.
* Additionally, the **Monitoring and Analytics Interface** connects to external tools for performance tracking, while the **Data Backup Interface** transfers archived records to external storage systems, typically scheduled for daily or weekly backups.

These interfaces ensure efficient communication and data management within the telecommunications ecosystem.

# Detailed System Design

The Detailed System Design outlines the architecture, data models, processes, and components required to implement the Call Data Record (CDR) project effectively. This project focuses on processing CDR files for cellular service providers, supporting functionalities such as user management, billing processing, and reporting.

## 4.1. Key Entities

* **User**: Represents the client who can sign up and log in.

Attributes: username, password, user\_id

* **CDRRecord**: Represents a call detail record.

Attributes: MSISDN, call\_type (voice/SMS), duration, timestamp, operator\_id

* **BillingInfo**: Contains billing details for users.

Attributes: user\_id, total\_billed, billing\_period

* **Operator**: Represents cellular service providers.

Attributes: operator\_id, brand\_name, MMC, MNC

## 4.2. Detailed-Level Database Design

## Logical Data Model (LDM) and ERD

**Entity Relationship Diagram (ERD)**: The LDM consists of the following entities and their relationships:

1. **User**:
   * One-to-Many relationship with **CDRRecord** (one user can have multiple call records).
   * One-to-One relationship with **BillingInfo** (each user has one billing record).
2. **CDRRecord**:
   * Many-to-One relationship with **Operator** (multiple records can belong to one operator).
3. **Operator**:
   * Contains details about the service provider.

* **LDM Diagram**:

User (msisdn) <---- CDRRecord ----> Operator (operator mmc/mnc)

|

BillingInfo

#### Physical Data Model (PDM) and PDM ERD

**Table Definitions**:

1. **User**:
   * user\_id: INT, PRIMARY KEY, AUTO\_INCREMENT
   * username: VARCHAR(50), UNIQUE, NOT NULL
   * password: VARCHAR(100), NOT NULL (hashed)
2. **CDRRecord**:
   * msisdn: VARCHAR(15), NOT NULL
   * call\_type: VARCHAR(10), NOT NULL (ENUM: 'voice', 'SMS')
   * duration: INT, NOT NULL
   * timestamp: DATETIME, NOT NULL
   * operator\_id: INT, FOREIGN KEY REFERENCES Operator(operator\_id)
3. **BillingInfo**:
   * billing\_id: INT, PRIMARY KEY, AUTO\_INCREMENT
   * user\_id: INT, FOREIGN KEY REFERENCES User(user\_id)
   * total\_billed: FLOAT, NOT NULL
   * billing\_period: VARCHAR(20), NOT NULL
4. **Operator**:
   * operator\_id: INT, PRIMARY KEY, AUTO\_INCREMENT
   * brand\_name: VARCHAR(50), NOT NULL
   * mmc: VARCHAR(5), NOT NULL
   * mnc: VARCHAR(5), NOT NULL

* **Data Dictionary**
* **User**:
  + user\_id: INT, unique identifier for users.
  + username: STRING, 50 characters, source: signup.
  + password: STRING, 100 characters, hashed.
* **CDRRecord**:
  + msisdn: STRING, 15 characters, from CDR files.
  + call\_type: STRING, 10 characters, from CDR files.
  + duration: INT, seconds.
  + timestamp: DATETIME, CDR record timestamp.
  + operator\_id: INT, maps to Operator.
* **BillingInfo**:
  + billing\_id: INT, unique identifier.
  + user\_id: INT, linked to User.
  + total\_billed: FLOAT, calculated total.
  + billing\_period: STRING, billing period details.
* **Indexes**
* Create indexes on username, msisdn, and operator\_id to improve search performance.
* **Implementation Factors**
* Use flat file storage for CDR records. Consider file locking for safe access during multi-threading.
* Define data access patterns based on expected workload.

### 4.2.1. Data Mapping Information

* **Data Transformation**: CDR records will be transformed from raw file format to structured database entries.
* **Data Lineage**: Track how data flows from CDR files to processed billing data, ensuring integrity and traceability.
* **Data Masking**: Sensitive user information (e.g., usernames, passwords) will be hashed and stored securely.
* **Consolidation**: Redundant data from multiple CDR files will be eliminated during processing.

## 4.2.2. Data Conversion

* **Requirements**: CDR files must be converted from text format to structured database records.
* **Formats**: Expect input CDR files in CSV or TSV format, while outputs (like CB.txt, IOSB.txt) will be plain text.
* **Resources**: Utilize multithreading to expedite file reading and processing.

## 4.3. Archival and Retention Requirements

* **Schedule**: Archive data every 5 years, keeping only the last 10 years of active data.
* **Retention Strategies**: Retain user-related data for 10 years, while CDR records are kept for 5 years before archival.

## 4.4. Disaster and Failure Recovery

* **Disaster Recovery Procedures**: Regular backups of user data and CDR files should be scheduled.
* **Resources**: Local storage for user data and CDR files.
* Thread-safe mechanisms for accessing shared resources (e.g., mutexes for STL containers).
* **Data Restoration Paths**: In the event of a failure, restore from the latest backup.
* Ensure that the system logs are intact to facilitate tracing the last known good state.

**4.5. Business Process Workflow**

* **User Registration/Login**: Users sign up or log in via the client.
* **CDR Processing**: Users choose to process CDR files, utilizing multithreading for efficiency.
* **Billing Information Retrieval**: Users can print or search for billing information based on processed data.

## 4.6. Business Process Modeling and Management

* **Business Process Model**: Use UML diagrams to represent workflows for user registration, CDR processing, and billing information retrieval.

## 4.7. Business Logic

* **Signup**: Validate, hash password, and save user data.
* **Login**: Check credentials and retrieve user-specific data.
* **CDR Processing**: Read and parse CDR files, calculate billing amounts, store results in appropriate tables.

## 4.8. Variables

* **Naming Conventions**: Use camelCase for variable names (e.g., totalBilled).
* **Usage**: Variables will hold user data, CDR records, and processing states.

## 4.9. Activity Diagram

* **Activity Diagram**: Illustrate the steps of user registration, CDR processing, and billing retrieval.

A diagram of a company

Description automatically generated

## 4.10. Data Migration

Data migration is a critical aspect of the CDR project, ensuring that data flows seamlessly between various system components and that historical data is appropriately transitioned to the new system. This section outlines the requirements, processes, and considerations for successful data migration.

### 4.10.1. Architectural Representation

* **Software Architecture**: A client-server architecture is employed, facilitating communication over sockets.
* **View Types**: Use Case, Logical, Process, and Deployment views will be documented for clarity.

### 4.10.2. Architectural Goals and Constraints

* **Goals**: Ensure high availability and responsiveness. Minimize latency in processing requests.
* **Constraints**: Maintain compatibility with existing systems and databases.

### 4.10.3. Logical View

* **Subsystems**: Divide into User Management, CDR Processing, and Billing Management.
* **Classes**: Introduce significant classes and describe their responsibilities.

### 4.10.4. Architecturally Significant Design Packages

* **User Management Package**: Contains User class, responsible for registration and authentication.
* **CDR Processing Package**: Contains CDR Record class and processing functions.

### 4.10.5. Data Model

* **Lightweight Processes**: Each user session and CDR processing occurs in separate threads for concurrency.
* **Communication**: Use message passing for client-server interactions.

### 4.10.6. Deployment View

* **Physical Configuration**: Describe server and client machines, network configurations, and their interactions.

# Environment Description

The system environment is designed to support the application’s diverse functionalities and user requirements effectively. This includes a robust architecture that accommodates user interactions, data processing, and integration with external systems. The environment encompasses all necessary hardware and software configurations, ensuring optimal performance, security, and scalability. Specific requirements regarding time zones, language support, user desktop and server-side configurations, deployment considerations, and integration needs are outlined to facilitate seamless operation.

## 5.1. Time Zone Support

The application provides comprehensive time zone support, enabling users to operate in various time zones such as UTC, EST, PST, and IST. The system automatically adjusts for daylight saving time where applicable, ensuring accurate timekeeping and scheduling for users across different regions. Additionally, timestamps are converted based on user preferences, allowing for consistency in data presentation and reporting.

## 5.2. Language Support

Language support within the system includes English with the user interface and documentation localized to cater to diverse user needs. All interface elements, error messages, and help documentation are translated to ensure clarity and accessibility. Furthermore, the application offers full right-to-left (RTL) support for languages such as Arabic and Hebrew, enhancing usability for speakers of those languages.

## 5.3. User Desktop Requirements

Users are required to have a compatible desktop environment to access the application efficiently. Supported operating systems include Windows 10 or later, macOS 10.14 or later, and recent distributions of Linux. Hardware requirements specify a minimum dual-core processor, 4 GB of RAM (8 GB recommended), and at least 500 MB of free disk space for installation.

## 5.4. Server-Side Requirements

The server-side requirements are crucial for the application’s performance and reliability. Application servers should run on Linux (preferably Ubuntu 20.04 LTS) or Windows Server 2019, with at least 16 GB of RAM and 100 GB of disk space to support application files and logs.

# 5.4.1. Deployment Considerations

Deployment considerations include a structured approach to installing and configuring the application in the production environment. A detailed deployment plan should be established, covering pre-deployment testing, staging, and roll-back procedures in case of issues. It’s essential to ensure that all dependencies and integrations are correctly configured before going live. Additionally, training for users and administrators should be part of the deployment strategy to facilitate a smooth transition.

# 5.4.2. Application Server Disk Space

The application server requires a minimum of 100 GB of disk space for the installation of application files, system logs, and temporary data. Adequate disk space must also be provisioned for future updates, caching, and potential log file growth. Utilizing SSDs is recommended to enhance read and write speeds, thereby improving overall application performance.

# 5.4.3. Database Server Disk Space

The database server should be provisioned with at least 500 GB of disk space to accommodate the database, backups, and transaction logs. It is advisable to use high-speed storage solutions like SSDs for the database server to ensure optimal performance, particularly for read and write operations. Regular monitoring and management of disk space usage are critical to prevent performance degradation due to insufficient storage.

# 5.5. Integration Requirements

Integration requirements specify the need for seamless interaction between various components within the system and external applications. The application should support RESTful APIs for efficient data exchange and communication. Data formats such as JSON and XML must be supported to facilitate compatibility with external systems. Additionally, an integration layer should be implemented to handle data synchronization and ensure consistency across platforms.

# 5.5.1. Jobs

Job management features are essential for the system's operational efficiency. The application must allow for the addition, modification, and deletion of scheduled jobs that handle tasks such as data processing, reporting, and maintenance. A reliable job scheduling tool, such as Cron for Linux or Task Scheduler for Windows, should be utilized to automate these processes, along with monitoring capabilities to track job status and alert administrators of any failures.

# 5.5.2. Network

Network requirements include a minimum bandwidth of 100 Mbps to ensure optimal performance and responsiveness of the application. The network must be configured to support both IPv4 and IPv6 protocols, with proper firewall settings to allow necessary traffic. Load balancing solutions should be implemented to distribute incoming requests across multiple servers, enhancing scalability and reliability. Additionally, VPN support should be available for secure remote access to the system.

# 5.6. Others

Additional considerations may include specific compliance requirements relevant to the system's operation, such as adherence to GDPR, HIPAA, or other industry standards. Security protocols must be established to protect sensitive data during transmission and storage, incorporating encryption and regular audits to ensure compliance and mitigate risks.

## 5.6.1. Configuration

Configuration requirements provide detailed guidelines for setting up the operating system, database, and network components. These requirements ensure that the environment is optimized for performance, security, and reliability. Each component should be configured to meet the minimum specifications and best practices to facilitate efficient operation of the application.

# 5.7. Operating System

The operating system requirements specify a minimum of 16 GB of RAM and a quad-core processor for optimal performance. Supported operating systems include Linux (Ubuntu 20.04 LTS) or Windows Server 2019. Additionally, the system should have a minimum of 100 GB of free disk space for installation and future updates, ensuring that all components run smoothly.

# 5.8. Database

Database configuration requirements emphasize the need for a robust DBMS, such as PostgreSQL 13 or MySQL 8.0. The database server should have at least 32 GB of RAM and a minimum of 500 GB of disk space. Performance settings should be optimized, including adjustments to connection limits and caching parameters, to ensure efficient data processing and retrieval.

# 5.8.1. Network

Network configuration requirements encompass all components necessary for a secure and efficient network setup. This includes routers, switches, and firewalls configured to facilitate secure data flow and communication between servers and clients. Proper configuration ensures high availability, redundancy, and protection against unauthorized access, contributing to the overall stability of the system.

# 5.8.2. Desktop

Desktop configuration requirements outline the necessary application software and hardware needed for users to interact with the system effectively. Users should have access to the latest web browsers, the Java Runtime Environment, and any additional application software required for specific tasks. Standard peripherals, such as monitors with a minimum resolution of 1920x1080, should also be specified to enhance the user experience.

# References

|  |
| --- |
| * <https://docstore.mik.ua/univercd/cc/td/doc/product/wanbu/das/das_1_4/das14/das14apd.htm> |
| * <https://www.gl.com/Presentations/Call-Data-Records-Presentation.pdf> |
| * <https://en.wikipedia.org/wiki/Call_detail_record> |
|  |
| * <https://gist.github.com/kaisesha/bd10fd299a3bed2b12ff031c937cdd4c> |
| * <https://github.com/kaisesha/cdrgraph/blob/master/CDR_Gist3.csv> |
| * <https://anjuchamantha.github.io/cellyzer---CDR-data-analyzer/> |
| * <https://github.com/mayconbordin/cdr-gen> |
| * <https://github.com/deshpandetanmay/cdr-data-generator> |
| * <https://www.sciencedirect.com/topics/computer-science/call-data-record> |
| * <https://www.itu.int/en/ITU-D/Emergency-Telecommunications/Documents/2017/Reports/LB/D012A0000C93301PDFE.pdf> |
| * <https://ijarcce.com/upload/2016/december-16/IJARCCE%2064.pdf> |
| * <https://www.etsi.org/deliver/etsi_ts/122100_122199/122115/03.02.00_60/ts_122115v030200p.pdf> |
| * <https://lawwatch.in/how-to-obtain-call-data-records-cdr/> |
| * <https://www.etsi.org/deliver/etsi_ts/132200_132299/132298/17.03.00_60/ts_132298v170300p.pdf> |

|  |
| --- |
|  |

|  |
| --- |
|  |
|  |

# Appendix

**1. Blogs and Tutorials**

* [How to obtain Call Data Records (CDRs)? - Lawwatch : Resources for Learners](https://lawwatch.in/how-to-obtain-call-data-records-cdr/)
* [Call-Data-Records-Presentation](https://www.gl.com/Presentations/Call-Data-Records-Presentation.pdf)

**2. General Information**

* [Call detail record - Wikipedia](https://en.wikipedia.org/wiki/Call_detail_record)

**3. Technical Specifications**

* [Call Data Record - an overview | ScienceDirect Topics](https://www.sciencedirect.com/topics/computer-science/call-data-record)

**Change Log**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **QMS Template Version Control (Maintained by QA)** | | | | | |
|  |  |  |  |  |  |
| **Date** | **Version** | **Author** | | **Description** | |
| 28-May-2015 | 1.0 | QA Team | | Initial Version | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |