Low Level Design (LLD)

Project Name

Calculator App

Your Name: sachin

Document Version Control

Date Issued	Version	Description	Author
Date	1.2	Added Workflow chart	sachin
Date	1.3	Added Exception Scenarios Overall, Constraints	sachin
Date	1.1	First Draft	sachin
Date	1.4	Added KPIs	sachin
Date	1.5	Added user I/O flowchart	sachin
Date	1.6	Added EHR, LSTM model diagrams	sachin
Date	1.7	Added dataset overview and updated user I/O flowchart.	sachin
Date	1.8	Restructure and reformat LLD	sachin

Contents

Abstract

The Low-Level Design (LLD) for the calculator system provides a comprehensive blueprint for its implementation. This design focuses on the detailed architecture of the components required to build a functional calculator, addressing both the user interface and the underlying computation logic.

I Introduction

The Low-Level Design (LLD) phase of software development translates the high-level architecture of a system into detailed, actionable design components. For a calculator application, the LLD outlines the intricate workings of the system, specifying how each part of the application will be implemented to meet functional requirements. This document serves as a bridge between the conceptual design and the actual coding, providing clear instructions for developers to follow.

1.2 Scope

The scope of the Low-Level Design (LLD) for a calculator defines the boundaries and detailed specifications of the system components and their interactions. It outlines what is included in the design, providing a clear understanding of the functionalities, constraints, and design considerations. Here's a structured outline of the scope for the LLD of a calculator:

1.3 Constraints

When designing a calculator at a low level, several constraints must be considered to ensure that the system is functional, efficient, and meets the requirements of the intended use case. These constraints shape the design decisions and implementation strategies. Here are some common constraints for LLD in a calculator application: **Performance Constraints:**

- **Response Time:** The calculator should provide quick feedback to user inputs, ensuring minimal delay in calculations and updates to the display.
- **Computation Speed:** Arithmetic operations, especially complex expressions, should be processed efficiently to maintain responsiveness.
- **Resource Utilization:** The system should use memory and CPU resources efficiently, avoiding excessive consumption that could impact performance.

Platform Constraints:

- **Compatibility:** The design must be compatible with the target platform(s), such as desktop, web, or mobile. This includes adhering to platform-specific guidelines and constraints.

- **Resolution and Screen Size:** The user interface should be adaptable to various screen sizes and resolutions, providing a consistent experience across devices.
- **Input Methods:** The design must support the input methods available on the target platform (e.g., touchscreen, keyboard, mouse).

1.4 Risks

Identifying and mitigating risks during the Low-Level Design (LLD) phase is crucial to ensure the successful development of a calculator application. Risks can impact various aspects of the system, from functionality and performance to maintainability and user experience. Here are some common risks associated with the LLD for a calculator and strategies for mitigating them:

Complexity and Overengineering:

- **Risk:** Designing overly complex components or implementing features that exceed the requirements can lead to increased development time, higher maintenance costs, and potential bugs.
- **Mitigation:**
  - Adhere to the principle of simplicity in design. o Implement only the required features and functionalities.
  - Regularly review the design to ensure it aligns with requirements and avoids unnecessary complexity.

1.5 Out of Scope

Defining what is out of scope in the Low-Level Design (LLD) for a calculator is crucial for setting clear boundaries and ensuring that the design effort is focused on the core functionalities required. Out-of-scope items help prevent scope creep and ensure that the design remains manageable and aligned with project goals. Here's a structured list of potential out-of-scope elements for an LLD of a calculator:

Advanced Mathematical Functions Beyond Requirements

- **Examples:**
  - Complex number calculations. o Symbolic mathematics or algebraic simplifications.
  - Advanced calculus functions such as integrals or differential equations.
- **Reasoning:**

- The calculator is designed for basic and possibly some intermediate functions. Advanced mathematical functions are not part of the core functionality for this scope.

2 Technical specifications

2.1 Dataset

In the context of a calculator application, the term "dataset" typically refers to the data structures and internal representations used to handle calculations, manage state, and facilitate interactions. Unlike applications dealing with large-scale data storage or complex data management, a calculator's datasets are usually simple but critical to its functionality. Here's an outline of the datasets you might encounter in the LLD for a calculator:

1. User Input Data:

- **Description:** Data representing the current input from the user, including numbers and operators.
- **Data Structure:**
  - **Input Buffer:** A string or list that temporarily holds the user input as they enter numbers and operators. o **Example:** ["5", "+", "3", "*", "2"]
- **Purpose:** To capture and process user input before performing calculations.

2. Calculation Data:

- **Description:** Data used to perform calculations and store intermediate results.
- **Data Structures:**
  - **Expression Tree:** A tree structure representing mathematical expressions in a hierarchical form. Nodes represent operators and operands.
  - **Example:** For the expression (5 + 3) * 2, the tree would have a root node with *, left child with +, and leaf nodes with 5, 3, and 2.
  - **Stack:** Used for operations such as parsing expressions (e.g., Shunting Yard algorithm) and evaluating them.
- **Purpose:** To manage and evaluate mathematical expressions efficiently.

3. Result Data:

- **Description:** Data representing the result of a calculation.
- **Data Structure:**

  Result Variable: A variable or field where the result of the current calculation is stored.

  - **Example:** result = 16 for the expression (5 + 3) * 2.
- **Purpose:** To store and display the final result of the calculations to the user.

4. History Data (if applicable):

- **Description:** Data that keeps track of past calculations and results.
- **Data Structures:**
  - **History List:** A list or array that stores previous calculations and results. o**Example:** ["5 + 3 = 8", "8 * 2 = 16"]
- **Purpose:** To allow users to view and possibly reuse past calculations. This might be a feature if included in the design.

2.1.1 dataset overview

n the Low-Level Design (LLD) for a calculator, datasets are essential to managing inputs, calculations, results, and states effectively. Understanding these datasets and their structures helps ensure that the calculator functions as expected, performs calculations accurately, and maintains a smooth user experience. Here's an overview of the key datasets used in the LLD for a calculator:

---

1. Key Operations:

- **Parsing:** Convert raw input into a format suitable for computation.
- **Validation:** Check for valid input formats and handle errors if necessary.

2. Calculation Data:

- **Purpose:** Manages the data needed to perform calculations and process mathematical expressions.

- **Data Structures:**

  Expression Tree: A hierarchical structure that represents the mathematical expression. Nodes are operators and operands.

  - **Example:** For 7 + (2 / 3), the tree has + as the root, with 7 and (2 / 3) as children. The (2 / 3) subtree represents division.

  - **Stack:** Utilized in algorithms like Shunting Yard to convert infix expressions to postfix notation and evaluate them.

    - **Example:** Stack = [2, 3, "/"] during the evaluation phase.

2.2 Predicting

In the context of a Low-Level Design (LLD) for a calculator, "predicting" generally refers to anticipating and addressing potential challenges, behaviors, and outcomes associated with the design and implementation of the system. This involves forecasting how various components of the calculator will interact, how they might fail, and how they will handle different types of input and usage scenarios.

2.4 Logging

In the context of a Low-Level Design (LLD) for a calculator, "predicting" generally refers to anticipating and addressing potential challenges, behaviors, and outcomes associated with the design and implementation of the system. This involves forecasting how various components of the calculator will interact, how they might fail, and how they will handle different types of input and usage scenarios.

Predicting User Behavior:

- **Input Patterns:**
  - **Prediction:** Users will input sequences of numbers and operators that the calculator must handle correctly.
  - **Design Consideration:** Implement robust input parsing and validation to manage common and edge-case input patterns (e.g., multiple operators, large numbers).
- **Error Scenarios:**
  - **Prediction:** Users may enter invalid or unexpected inputs (e.g., 5 / 0 or * 3).
  - **Design Consideration:** Ensure comprehensive error handling and userfriendly error messages to guide users in correcting their inputs.

2. Predicting Calculation Complexity:

- **Expression Complexity:**

- **Prediction:** Calculations will range from simple arithmetic to more complex expressions involving multiple operators and parentheses.
- **Design Consideration:** Use appropriate algorithms for parsing and evaluating expressions, such as the Shunting Yard algorithm for converting infix expressions to postfix.

- **Performance:**
  - **Prediction:** Complex expressions may impact performance, especially if they involve multiple nested operations.
  - **Design Consideration:** Optimize computation methods and manage resources efficiently to maintain responsiveness.

3. Predicting System Interactions:

- **Component Integration:**
  - **Prediction:** The interaction between UI elements and the calculation engine needs to be seamless.
  - **Design Consideration:** Define clear interfaces and communication protocols between the UI and backend components to ensure smooth data flow and state management.
- **State Management:**
  - **Prediction:** The calculator will need to maintain and update its state based on user interactions and previous calculations.
  - **Design Consideration:** Implement a state management system to track current inputs, operations, and results effectively.

.

2.5 Database

For most calculator applications, especially basic ones, the use of a traditional database is not typically required due to their straightforward functionality and limited data management needs. However, if a calculator application includes features that necessitate data storage, such as saving history, user preferences, or advanced features, then designing a database schema can be relevant.

Here's an overview of how a database might be utilized and structured in the LLD for a calculator:

1. Purpose of Database in Calculator

a. History Tracking:

- **Purpose: To store past calculations and results for user reference.**

- **Use Case: Users might want to view their calculation history or reuse previous results.**

b. User Preferences:

- **Purpose: To save user settings and preferences such as precision, mode (basic, scientific), and UI customizations.**
- **Use Case: Preserving user preferences between sessions or device uses.**

c. Advanced Features (if applicable):

- **Purpose: To support features that require persistent data storage, like custom formulas or scientific constants.**
- **Use Case: Allowing users to define and store custom functions or constants for future use.**

2.6 Deployment

Deployment in the Low-Level Design (LLD) phase for a calculator application involves outlining how the application will be delivered and made available to end-users. This includes specifying the deployment environment, configurations, and processes required to ensure the application runs smoothly. Here's a structured approach to deploying a calculator application: **Deployment Environment** **a. **Platform:**

- **Web-based Calculator:** Hosted on a web server accessible via a browser.
- **Desktop Calculator:** Installed on users' local machines, typically through an installer.
- **Mobile Calculator:** Deployed through app stores (e.g., Google Play, Apple App Store).

b. **Infrastructure:

- **Web Server:** For web-based calculators (e.g., Apache, Nginx).
- **Application Server:** Handles application logic and user interactions.
- **Database Server:** If the application requires data storage (e.g., MySQL, PostgreSQL).

c. **System Requirements:

- **Hardware:** Specify minimum hardware requirements (e.g., CPU, RAM) for the application to run efficiently.
- **Software:** Specify required software dependencies (e.g., operating systems, libraries, frameworks).

3 Technology stack

Front End	HTML/CSS/JS/React

4 Proposed Solution

- The **Proposed Solution** section in a Low-Level Design (LLD) for a calculator details the specific design and implementation strategies for building the calculator application. This includes architectural decisions, component designs, data handling, user interface considerations, and more. Below is a structured outline of the proposed solution for a calculator application:

Architectural Design

a. **Overall Architecture:

- **Client-Server Model:** For web-based calculators, the architecture involves a client (browser) and a server (hosting the application and handling computations).
- **Standalone Application:** For desktop or mobile calculators, the application runs locally on the user's device.

b. **Components:

- **User Interface (UI):** The frontend component where users interact with the calculator.
- **Calculation Engine:** The backend component responsible for processing calculations and handling business logic.
- **Data Storage (if applicable):** Manages persistent data such as history and user preferences.
- **API Layer (for web-based calculators):** Facilitates communication between the frontend and backend.

5 Model training/validation workflow

In the context of a calculator application, **Model Training** is typically not a standard requirement since calculators traditionally perform

deterministic arithmetic operations based on fixed algorithms rather than predictive or machine-learning-based tasks. However, if the calculator application includes advanced features that involve predictive modeling, machine learning, or adaptive functionalities, such as personalized suggestions or predictive text, then model training becomes relevant.

1. Purpose of Model Training

a. **Predictive Features:

- **Personalized Recommendations:** Suggest frequently used operations or functions based on user behavior.
- **Adaptive Interfaces:** Adjust the calculator's layout or features based on user preferences or usage patterns.

b. **Error Prediction:

- **Handling Common Errors:** Predict and mitigate common user errors or missteps in calculations.

c. **Enhanced Functionality:

- **Advanced Features:** Integrate complex functionalities like predicting future values or trends based on historical data.

2. Data Collection

a. **User Interaction Data:

- **Data Sources:** Collect data on user interactions, input patterns, and calculation history.
- **Types of Data:** User inputs, results, frequency of operations, and timestamps.

b. **Feedback Data:

- **User Feedback:** Collect feedback on incorrect results or suggestions to improve the model.

6 User I/O workflow

he User Input/Output (I/O) workflow in a Low-Level Design (LLD) for a calculator outlines how the application handles user interactions, including input processing and output presentation. This workflow ensures that the calculator operates intuitively and efficiently, providing accurate results and a smooth user experience.

1. Overview of User I/O Workflow **a.

User Input:

- **Types of Input:** User input can include numbers, operators, functions, and commands.
- **Input Methods:** Users interact with the calculator via buttons, keyboard inputs, or touch gestures (depending on the platform).

b. **Processing:

- **Input Handling:** The application processes the user's input, validates it, and prepares it for computation.
- **Calculation:** The calculator performs the requested operation or computation based on the processed input.

c. **Output:

- **Display Results:** The result of the calculation is displayed to the user.
- **Error Handling:** Display error messages or prompts if there are issues with the input or computation.

2. Detailed Workflow Steps

a. **User Input Handling:

1. **Capture Input:**
    - **Event Detection:** Detect user actions such as button clicks, key presses, or touch events.

- **Input Extraction:** Extract the input from the event (e.g., button label, key value).

2. **Update Input Buffer:**

- **Buffer Management:** Append the input to the current expression or command buffer.

- **Display Update:** Reflect the updated input on the display area in real-time.

3. **Input Validation:**

- **Syntax Check:** Validate the syntax of the input (e.g., ensure correct operator placement).

- **Pre-processing:** Handle special cases such as function calls or complex expressions.

7 Exceptional scenarios

Exceptional scenarios refer to the conditions where the calculator may encounter errors or unusual situations that deviate from normal operation. Handling these scenarios effectively ensures the application remains robust, user-friendly, and reliable. Here's a detailed breakdown of exceptional scenarios in the Low-Level Design (LLD) for a calculator application:

1. Input Errors

a. **Invalid Input Format:

- **Scenario:** User enters invalid characters or malformed expressions.
- **Example:** User types 5 ++ 3 or 3 * & 2.
- **Handling:**
  - **Validation:** Implement syntax checks to detect invalid inputs.

- **Feedback:** Display an error message such as "Invalid input format" and prompt the user to correct the input. **b. **Incomplete Expressions:**

- **Scenario:** User inputs an incomplete expression and attempts to calculate.
- **Example:** User types 5 + or * 3.
- **Handling:**
  - **Validation:** Check for missing operands or operators.
  - **Feedback:** Display an error message like "Incomplete expression" and highlight the issue in the input buffer.

2. Calculation Errors

a. **Division by Zero:

- **Scenario:** User attempts to divide by zero.
- **Example:** User types 5 / 0.
- **Handling:**
  - **Error Detection:** Implement checks to catch division by zero before computation.
  - **Feedback:** Display an error message such as "Division by zero is not allowed."

8 Test cases

est cases are essential to validate the functionality, performance, and reliability of a calculator application. They help ensure that all features work as expected and that the application handles various scenarios correctly. In the Low-Level Design (LLD) for a calculator, test cases are detailed to cover different aspects of the application, including basic operations, edge cases, error handling, and user interface elements.

Here's a structured approach to test cases for a calculator application: **1. Basic Arithmetic Operations** **a. **Addition:**

- **Test Case 1.1: Simple Addition** o **Description:** Test the addition of two positive numbers.
  - **Input:** 5 + 3 o **Expected Output:** 8
- **Test Case 1.2: Addition with Negative Numbers** o **Description:** Test addition involving negative numbers.
  - **Input:** -2 + 4 o **Expected Output:** 2
- **Test Case 1.3: Addition with Zero** o **Description:** Test addition where one of the operands is zero.

- **Input:** 7 + 0 ○
  Expected Output: 7
  b. Subtraction:
- **Test Case 2.1: Simple Subtraction** ○ **Description:** Test the subtraction of two positive numbers.
  - **Input:** 10 - 4 ○
    Expected Output: 6
- **Test Case 2.2: Subtraction with Negative Result** ○ **Description:** Test subtraction resulting in a negative number.
  - **Input:** 3 - 5 ○
    Expected Output: -2
- **Test Case 2.3: Subtraction with Zero** ○ **Description:** Test subtraction where the second operand is zero.
  - **Input:** 9 - 0 ○
    Expected Output: 9
    c. Multiplication:
- **Test Case 3.1: Simple Multiplication** ○ **Description:** Test the multiplication of two positive numbers.
  - **Input:** 4 * 6 ○
    Expected Output: 24
- **Test Case 3.2: Multiplication with Zero** ○ **Description:** Test multiplication where one of the operands is zero.
  - **Input:** 5 * 0 ○
    Expected Output: 0

9 Key performance indicators (KPI)

- Time and workload reduction using the EHR model.
- Comparison of accuracy of model prediction and doctor's prediction.
- Number of times a patient visits the hospital.
- Time between symptom onset and detection of illness/visit to hospital.
- Immunity of patient (based on previous illnesses).
- Vaccines the patient has taken. ● Length of stays in hospital.