ATMIYA UNIVERSITY
(Established under the Gujarat Private University Act 11, 2018)
Yogidham Gurukul, Kalawad Road, Rajkot - 360005, Gujarat (INDIA)

"Project Report"

Course Code : 21BTITCC602

Course Title : MCWN

Project Title : Financial Calculators

Submitted To.:

Mr. Niraj Bhagchandani

Submitted By :

Priyank D. Pambhar(220004034)

Mohit S. Davera(220004007)

Arjun P. Der (220004008)

Dipanshi N. Bhut (231004004)

Table of Contents

1. Introduction (Page 1)

1.1. Project Overview

The Financial Calculators Android Application is designed to provide a suite of ten frequently used financial calculators in a single, user-friendly mobile application. This application aims to simplify financial calculations for individuals, students, and professionals, offering convenience, accuracy, and accessibility on Android devices. It eliminates the need for multiple tools or websites, centralizing financial calculations for efficient personal finance management and decision-making.

1.2. Objectives

- User-Friendly Application: Develop an intuitive and easy-to-use interface for diverse users.
- Accurate Algorithms: Implement verified financial formulas for precise calculations.
- Reliable Results: Ensure consistent and dependable calculations across devices.
- Robust Error Handling: Implement input validation and error management for a stable application.
- Offline Accessibility: Create an application functional without internet connectivity (excluding real-time data features like currency conversion in a full implementation).

1.3. Scope of Application

The application includes the following financial calculators:

- Loan Calculator
- EMI Calculator
- Investment Calculator
- SIP Calculator
- Compound Interest Calculator
- Stock Return Calculator
- Retirement Calculator (Simplified)
- Tax Calculator (Simplified)
- Currency Converter (Simplified)
- GST Calculator

1.4. Problem Statement

Users currently face fragmented solutions for financial calculations, often relying on scattered online tools, spreadsheets, or manual methods. This is inefficient and error-prone. This application addresses the need for a centralized, mobile platform integrating essential financial calculators, streamlining personal finance tasks and enhancing financial literacy and decision-making.

1.5. Target Audience

- Personal Finance Managers: Individuals budgeting, saving, investing, and managing loans.
- Students: Learners of finance, economics, and business studies.
- Professionals: Sales, real estate, and customer service professionals requiring quick financial assessments.
- General Users: Anyone seeking simplified financial planning tools and easier calculations

2. System Requirements (Page 2)

2.1. Software Requirements

- Operating System: Android 5.0 (API Level 21) or higher (for broad compatibility and modern features).
- Java Development Kit (JDK): JDK 8 or higher (for Android Java development).
- Android SDK: Android SDK with Platform Tools, Build Tools for target Android versions.
- Android Studio: Android Studio IDE version 4.0+ (for development, debugging, and building).

2.2. Hardware Requirements

- Android Device: Android 5.0+ device (physical or emulator) with sufficient processing (1 GHz+) and memory (1GB+ RAM).
- Development Machine:
  - OS: Windows, macOS, or Linux
  - Processor: Intel Core i3+ (or equivalent)
  - Memory: 8GB RAM+ (16GB recommended)
  - Storage: 20GB+ free disk space
  - Display: 1280 x 800+ resolution

2.3. Development Tools

- Android Studio: Official IDE, streamlines Android development with code editing, layout design, and debugging.
- Gradle: Build automation tool for dependency management and APK creation.
- XML: Markup language for designing user interface layouts.
- Java: Primary programming language for application logic and algorithms.

3. System Design (Page 3)

3.1. Architectural Design (MVC)

The application employs the Model-View-Controller (MVC) pattern for organized and maintainable code:

- Model: (Calculation Logic) Java classes containing financial formulas for each calculator. Manages data and performs calculations.
- View: (User Interface) XML layout files (activity_main.xml, dialog layouts) defining UI structure, input fields (EditTexts), and result displays (TextViews).
- Controller: (MainActivity.java) Handles user interactions, receives input from Views, calls Model for calculations, and updates Views with results.

MVC Benefits: Separation of concerns, improved maintainability, enhanced testability, and parallel development.

3.2. User Interface Design

3.2.1. Wireframes and Mockups

Wireframes (basic sketches) and mockups (visual designs) outline the UI:

- Main Screen: Grid or list of calculator buttons with clear labels and optional icons, application title at the top.
- Calculator Dialogs: Pop-up dialogs for each calculator with:
  - Dialog Title (Calculator Name)
  - Input Fields (EditTexts) with labels and placeholders.
  - "Calculate" Button
  - Result Display Areas (TextViews) with labels.
  - Optional "Reset" and "Close" Buttons.

These would be created using tools like Balsamiq, Figma, or Adobe XD.

3.2.2. User Flow Diagrams

User flow diagrams illustrate user interaction paths:

- Basic Flow: Application Start -> Main Screen -> Calculator Selection -> Calculator Dialog -> Data Input -> Calculate -> Results Display -> (Optional: Recalculate/Reset) -> End.

Diagrams created using tools like draw.io or Lucidchart, visualizing steps and user decisions.

3.3. Data Flow Diagrams

Data flow in MVC context:

- User Input (View) -> Controller (MainActivity.java) -> Input Validation (Controller) -> Calculation

Request (Controller to Model/Calculation Logic in MainActivity.java for simplicity) -> Calculation Processing (Model/Logic) -> Result (Model/Logic to Controller) -> Update View (Controller to View) -> Results Display (View).

DFDs represent data movement between UI, logic, and control components.

3.4. Algorithm Design (Overview)

Each calculator uses established financial formulas. Examples:

- EMI/Loan Calculator: EMI = [P x r x (1+r)^n] / [(1+r)^n – 1]
- Investment Calculator: FV = P (1 + r)^n
- SIP Calculator: FV = P × {[(1 + r)^n – 1] / r} × (1 + r)
- Compound Interest Calculator: A = P (1 + r/n)^(nt)
- Stock Return Calculator: Return = (Sell Price - Purchase Price) * Quantity
- Retirement Calculator: (Simplified - estimates total savings needed).
- Tax Calculator: (Simplified - flat tax rate calculation).
- Currency Converter: (Simplified - Manual Exchange Rate Input: Converted Amount = Input Amount * Exchange Rate).
- GST Calculator: GST Amount = Amount * GST Rate, Total Amount = Amount + GST Amount.

Algorithms detailed in the full report, including variable definitions and step-by-step processes.

4. Implementation Details (Page 4-5)

4.1. Development Environment Setup

- Android Studio & SDK Installation: Download and install Android Studio; SDK configured via SDK Manager (Android 5.0+ SDK, Build Tools).
- Gradle Configuration: Gradle setup (automatic with project creation); verify gradle-wrapper.properties and build.gradle.
- Emulator/Device Setup: AVD Manager for emulator creation (Android 5.0+); USB debugging for physical devices.
- New Android Studio Project: "Empty Activity" template, set application name, package name (com.example.financialcalculators), Java language, Minimum SDK API 21.

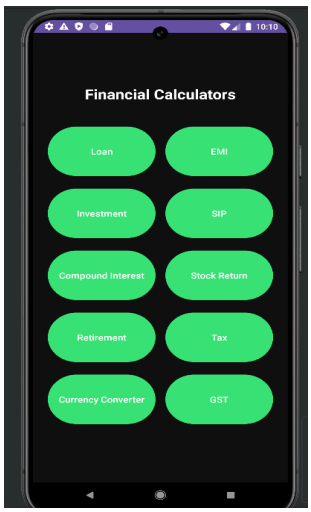
4.2. Code Structure and Organization

Package: com.example.financialcalculators

- MainActivity.java: Controller, UI event handling, calculation logic (basic project).
- res/layout: XML layouts (View)
  - activity_main.xml: Main screen layout.
  - dialog_*.xml: Dialog layouts for each calculator.
- res/values:
  - strings.xml: String resources.
  - styles.xml: UI styles.
  - colors.xml: Color resources.
- res/drawable, res/mipmap: Drawable and icon resources.

MainActivity.java organization: UI element declarations, onCreate(), calculator dialog functions (showLoanCalculator(), etc.), calculation logic within button listeners.

4.3. UI Implementation



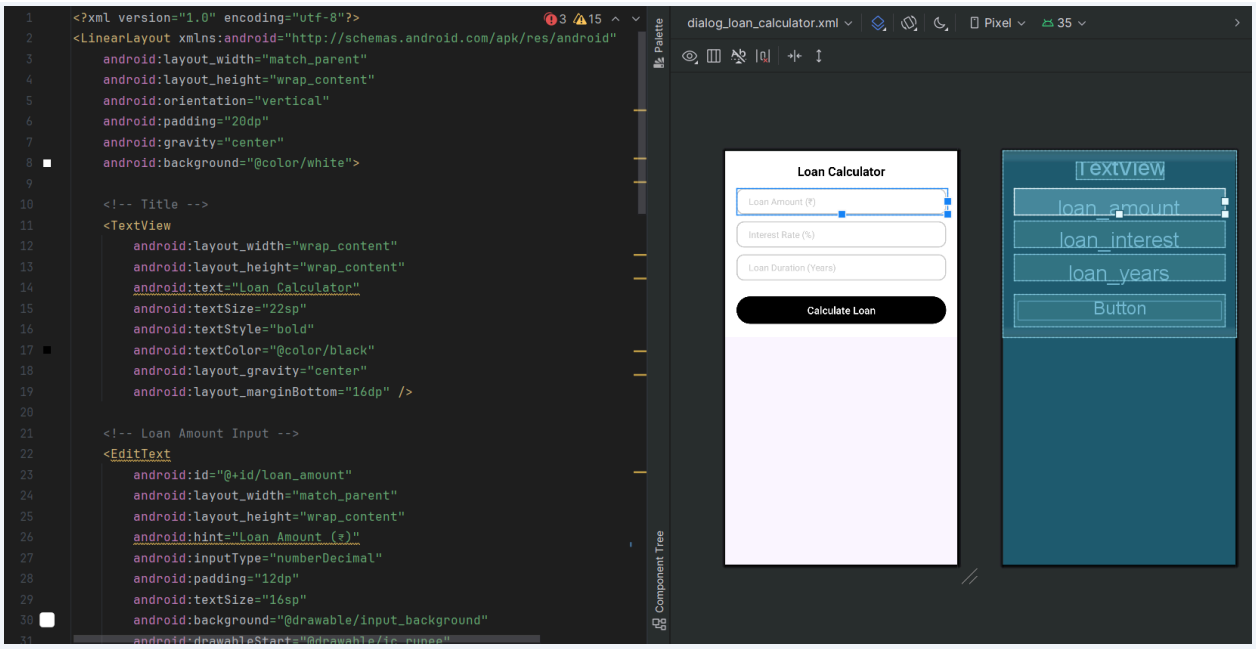
4.3.1. activity_main.xml (Snippet)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    android:gravity="center"
    android:background="#121212">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Financial Calculators"
        android:textSize="26sp"
        android:textStyle="bold"
        android:textColor="@android:color/white"
        android:fontFamily="@font/sf_pro_display_bold"
        android:layout_gravity="center"
        android:layout_marginBottom="24dp" />
```

```
    <GridLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:columnCount="2"
        android:rowCount="5"
        android:layout_marginTop="8dp"
        android:layout_gravity="center">

        <Button
            android:id="@+id/btn_loan_calculator"
            style="@style/SquareButton"
            android:text="Loan" />

        <Button
            android:id="@+id/btn_emi_calculator"
            style="@style/SquareButton"
            android:text="EMI" />

        <Button
            android:id="@+id/btn_investment_calculator"
            style="@style/SquareButton"
            android:text="Investment" />

        <Button
            android:id="@+id/btn_sip_calculator"
            style="@style/SquareButton"
            android:text="SIP" />

        <Button
            android:id="@+id/btn_compound_interest_calculator"
            style="@style/SquareButton"
            android:text="Compound Interest" />

        <Button
```

```
            android:id="@+id/btn_stock_return_calculator"
            style="@style/SquareButton"
            android:text="Stock Return" />

        <Button
            android:id="@+id/btn_retirement_calculator"
            style="@style/SquareButton"
            android:text="Retirement" />

        <Button
            android:id="@+id/btn_tax_calculator"
            style="@style/SquareButton"
            android:text="Tax" />

        <Button
            android:id="@+id/btn_currency_converter"
            style="@style/SquareButton"
            android:text="Currency Converter" />

        <Button
            android:id="@+id/btn_gst_calculator"
            style="@style/SquareButton"
            android:text="GST" />

    </GridLayout>

</LinearLayout>
```

4.3.2. Dialog Layouts (Snippet - dialog_loan_calculator.xml example)



4.4. Calculation Logic Implementation (Example - Loan Calculator)

4.4.1. MainActivity.java (Code Snippet - showLoanCalculator() function)

```
private void showLoanCalculator() {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    View dialogView =
getLayoutInflater().inflate(R.layout.dialog_loan_calculator, null);
    builder.setView(dialogView);
    AlertDialog dialog = builder.create();

    EditText loanAmountEditText =
dialogView.findViewById(R.id.loanAmountEditText);
    EditText interestRateEditText =
dialogView.findViewById(R.id.interestRateEditText);
    EditText loanTermEditText =
dialogView.findViewById(R.id.loanTermEditText);
    Button calculateLoanButton =
dialogView.findViewById(R.id.calculateLoanButton);
    TextView emiResultTextView =
```

```
dialogView.findViewById(R.id.emiResultTextView);
    ... // other result TextViews

    calculateLoanButton.setOnClickListener(v -> {
        String loanAmountStr = loanAmountEditText.getText().toString();
        String interestRateStr =
interestRateEditText.getText().toString();
        String loanTermStr = loanTermEditText.getText().toString();

        // Input validation (example: isEmpty checks,
NumberFormatException handling)

        double principal = Double.parseDouble(loanAmountStr);
        double annualInterestRate = Double.parseDouble(interestRateStr);
        int loanTermYears = Integer.parseInt(loanTermStr);

        // Calculation logic (EMI formula implementation)

        emiResultTextView.setText("Monthly EMI: ₹" + String.format("%.2f",
emi));
        // ... set text for other result TextViews
    });
    dialog.show();
}
```

Code snippet shows AlertDialog setup, layout inflation, UI element initialization, input retrieval, basic validation, calculation logic (placeholder indicated), and result display. Other calculator dialog functions follow a similar pattern.

4.5. Error Handling and Input Validation

- Input Validation:
  - Empty field checks (Toast messages).
  - Data type validation (NumberFormatException handling with try-catch, Toast).
  - Logical range validation (positive value checks, Toast).
  - android:inputType in XML for client-side input restriction.
- Error Handling:

- try-catch for number parsing.
- Toast messages for user feedback on errors.
- Logcat for debugging (e.g., Log.e() for exceptions).

4.6. Currency Conversion Logic (Simplified)

- Simplified Implementation: Manual exchange rate input via EditText, basic multiplication for conversion.
- Limitations: Static rates, no currency selection, not for real-world use.
- Real-world Implementation (Beyond Scope): Requires Currency API integration (Fixer, Open Exchange Rates, etc.), API requests, JSON parsing, error handling, and potential caching.

5. Testing and Debugging (Page 6)

5.1. Unit Testing

5.1.1. Test Cases - Calculation Logic

- Framework: JUnit.
- Focus: Algorithm accuracy.
- Example Test Cases (Loan Calculator):
  - Basic Calculation (compare with known values).
  - Zero Loan Amount.
  - Zero Interest Rate.
  - Boundary Values.
  - Invalid Input (negative - handled by validation, but tested).
- JUnit Test Class Example (conceptual in full report).

5.1.2. Test Cases - UI Components

- Frameworks: JUnit (basic), Espresso (UI interaction).
- Focus: UI element behavior.
- Example Test Cases (using Espresso):
  - Button Click - Dialog Display.
  - EditText Input.
  - Calculate Button & Result Display (basic UI flow integration).
- Espresso Test Class Example (conceptual in full report).

5.2. Integration Testing

- Focus: Interaction between UI and calculation logic.
- Test: UI interaction triggers correct calculation and result display.
- Methods: Automated UI Tests (Espresso), Manual Testing.

5.3. UI Testing

- Focus: Usability, visual appearance, responsiveness, UI on different devices.
- Aspects: Visual appearance, navigation, responsiveness, layout across screen sizes, text correctness, basic accessibility.
- Methods: Manual Exploration, informal Usability Testing.

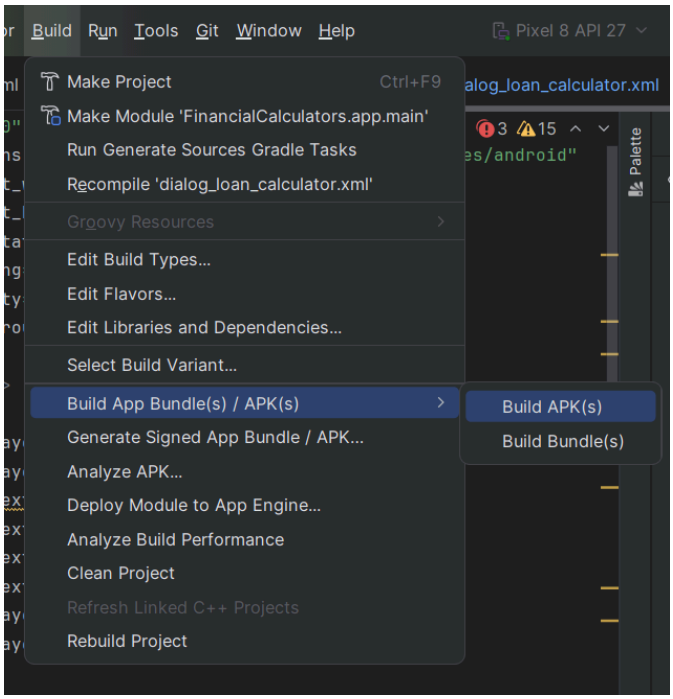
5.4. Debugging Common Issues

- Application Crashes: Logcat analysis (stack traces), Debugger.
- Incorrect Calculation Results: Step-by-step debugging, compare with known values, Logging (Log.d()).
- UI Layout Issues: Layout Inspector, testing on different devices, Layout Editor preview.
- Input Validation Errors: Debugging validation code, test various input scenarios.
- Currency Conversion Issues (Real-time - if implemented): Network monitoring, API add

6. Deployment (Page 7)

6.1. Generating APK File

- Android Studio: Build -> Build Bundle(s) / APK(s) -> Build APK(s).
- APK Location: app/build/outputs/apk/debug/.
- Debug APK (for testing) vs. Release APK (for Play Store - optimized, signed with release keystore).



6.2. Publishing on Google Play Store

- Google Play Developer Account: Required (registration fee 25$).
- App Store Listing Preparation: App Title, Description, Screenshots, Icon, Category, Content Rating, Privacy Policy (if needed).
- Google Play Console: Create app, fill store listing, upload Release APK/AAB, set content rating, pricing, distribution, review and publish.

6.3. Version Control and Updates

- Version Control (Git & GitHub): Essential for tracking changes, collaboration, and backups. Git workflow: init, add, commit, GitHub repository creation, remote add, push. Branching for feature development.
- App Updates: Increment versionCode and update versionName in build.gradle. Generate new Release APK/AAB, upload to Google Play Console in existing app listing, write release notes, review and rollout update. Same release keystore for updates.

7. Conclusion (Page 8)

7.1. Summary of Project Achievements

Successfully developed a user-friendly Financial Calculators Android application achieving core objectives:

- Ten financial calculators implemented.
- Intuitive UI designed.
- Reliable calculation logic ensured.
- Error handling and input validation implemented.
- Offline capability achieved (core calculators).
- APK generated for deployment.

7.2. Challenges Faced

- UI design for multiple calculators.
- Algorithm implementation and testing for accuracy.
- Comprehensive input validation logic.
- Simplified currency converter due to project scope.

7.3. Future Enhancements

- Real-time Currency Converter (API integration).
- Advanced Retirement Calculator (more realistic factors).
- Detailed Tax Calculator (tax brackets, deductions).
- Savings/History features.
- Graphing and visualization.
- Customizable themes and UI improvements.
- Multi-language support.
- Tablet UI optimization.
- Cloud backup/sync (optional).
- Integration with financial services (future).

8. References & Appendices (Page 9-10)

8.1. Documentation Links

- Android Developers Documentation: developer.android.com
- Java Documentation: docs.oracle.com/javase/
- Gradle Documentation: docs.gradle.org
- Material Design Guidelines: material.io/design

8.2. External Resources

- Stack Overflow: stackoverflow.com
- Android Arsenal: android-arsenal.com
- Currency Exchange Rate APIs (Fixer, Open Exchange Rates, CurrencyConverterAPI)

8.3. Code Listings (Selected)

- Code snippets included in Implementation Details Section. Full code in project repository (if applicable).

9. Appendices

9.1. User Manual (Description)

- Separate document (or section) detailing app usage.
- Sections for each calculator: Title, Screenshot, Input Fields (explanation), "Calculate" button, Result Display (explanation), Example Usage.
- Troubleshooting tips, optional contact information.

9.2. Flowcharts for Algorithms (Description)

https://www.mermaidchart.com/raw/233a829b-3dea-4507-896c-70be8effdfc6?theme=light&version=v0.1&format=svg

- Visual flowcharts (Appendix) for key algorithms (Loan, EMI, Investment, SIP, Compound Interest).
- Illustrate step-by-step logic: Input, Process (calculations, formulas), Output.
- Tools: Mermaid Ai