Dominic Drury

Budget Buddy

Budgeting for The EVERY person

A green robot with a sign on a clipboard

Description automatically generated

Table of Contents

[Document Revision History 2](#_Toc199791119)

[Project Overview 5](#_Toc199791120)

[Version 5](#_Toc199791121)

[Purpose 5](#_Toc199791122)

[Scope 5](#_Toc199791123)

[Requirements 5](#_Toc199791124)

[Functional 5](#_Toc199791125)

[Non-Functional 6](#_Toc199791126)

[Technical 6](#_Toc199791127)

[Design 7](#_Toc199791128)

[System Architecture 7](#_Toc199791129)

[User Interface Design 7](#_Toc199791130)

[Screens 7](#_Toc199791131)

[Layout and Components 8](#_Toc199791132)

[UI Elements Used 8](#_Toc199791133)

[Database Design 8](#_Toc199791134)

[Development 10](#_Toc199791135)

[Development Environment 10](#_Toc199791136)

[Source Control 10](#_Toc199791137)

[Coding Standards 10](#_Toc199791138)

[Testing 11](#_Toc199791139)

[Testing Strategy 11](#_Toc199791140)

[Test Cases 12](#_Toc199791141)

[Test Results 13](#_Toc199791142)

[Deployment 13](#_Toc199791143)

[Deployment Process 13](#_Toc199791144)

[Versioning 14](#_Toc199791145)

[Release Notes 14](#_Toc199791146)

[Maintenance 14](#_Toc199791147)

[Bug Reporting and Tracking 15](#_Toc199791148)

[Appendices 16](#_Toc199791149)

[Glossary 16](#_Toc199791150)

[References 17](#_Toc199791151)

[Additional Notes 18](#_Toc199791152)

# Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 0.0 | 08/04/2024 | Dominic Drury | Initial creation of document and addition of wireframes |
| 0.01 | 08/10/2024 | Dominic Drury | Completion of initial sections using a template |
| 0.02 | 02/24/2025 | Dominic Drury | Created rough draft of home screen that included title, add button, calculate button, income button, settings button, and recycler view. |
| 0.021 | 02/28/2025 | Dominic Drury | Created functionality for recycler view to display preset data and a drop down menu for selecting budget |
| 0.022 | 03/02/2025 | Dominic Drury | Made slight adjustments to the home screen UI that was added on 02/24/25. I added texts for income, expenses, and the money left. I also added a recycler view to display the percentile of the budget types. |
| 0.03 | 03/03/2025 | Dominic Drury | Created a database for budget items and adjusted recycler view to take the database data instead of provided hardcoded data |
| 0.031 | 03/07/2025 | Dominic Drury | Used database to get array list of bill types, fill budget percentile recycler view with list from bill types, and calculate the percentage of bills that each type take up. Calculated total of expenses from displayed budget. |
| 0.032 | 03/10/2025 | Dominic Drury | Added functionality to the add budget item button to allow for adding new items to the budget database, then updating the recycler views for the budget recycler view and budget percentiles. |
| 0.033 | 03/17/2025 | Dominic Drury | Added a variable to budget items for handling which budget is selected from the drop down and only displaying the budget items from that budget.  Added functionality to the Cancel button of add budget item. |
| 0.034 | 03/18/2025 | Dominic Drury | Added an add button for creating a new budget, created functionality for the button, and ensured that database updates appropriately when that new budget is populated, but does not save if that budget is left empty on close. |
| 0.035 | 03/20/2025 | Dominic Drury | Added functionality for a longItemClick on the budget currently being displayed and the option to delete that item is given to the user. If confirmed the database and both the budget and percentile recycler views are updated. Updated the SQLite db to accommodate for an added table of incomes. |
| 0.04 | 03/24/2025 | Dominic Drury | Created UI for income, updated database and validated CRUD functionality of incomes. Moved calculating expenses to happen whenever budget recycle view is loaded instead of being in onCreate. |
| 0.041 | 03/25/2025 | Dominic Drury | Added functionality to the income recycleview so that it displays the incomes for the budget currently being displayed. Added functionality for accessing a fragment to handle adding new incomes to the budget. |
| 0.042 | 03/31/2025 | Dominic Drury | Created UI for adding new incomes. Altered the logic for adding a new budget so that all fragments are removed when a new budget is added. |
| 0.043 | 04/02/2025 | Dominic Drury | Added functionality for deleting income items and updating totals both on the income fragment and the main activity. Added text views to image buttons to label each. |
| 0.05 | 04/03/2025 | Dominic Drury | Added sorting functionality to the budget item recycler view. Added sorting functionality to income recycler view. |
| 0.051 | 04/09/2025 | Dominic Drury | Created a custom class to force whole application to be set to light mode in order to ensure a consistent and clean UI. |
| 0.052 | 04/14/2025 | Dominic Drury | Got feedback on current UI, made adjustments like adding extra labels to buttons and recycler views and adjusting sizes of UI elements to reflect suggestions |
| 0.053 | 04/23/2025 | Dominic Drury | Updates account UI element to adding a new bill so that it displayed a dropdown menu of previously used accounts as well as maintaining its edit text functionality. |
| 0.054 | 04/24/2025 | Dominic Drury | Attempted to fix adding bill issue where it does not move up when keyboard is in use to prevent UI elements from being blocked. Solution to have the fragment move failed, so the solution of having the fragment be at the top center of the fragment container was used. I successfully adjusted the income UI to better display the columns without unnecessary white space in the vertical columns. |
| 0.06 | 04/25/2025 | Dominic Drury | I changed the edit text for bill types to be a drop down menu of preselected types of bills the user might use to make the budget percentile more valuable as well as improving the understanding of what that section is meant to mean. I set up the account part of the UI for adding new bills to display a full list of previously used accounts when the box is tapped and still show the keyboard so the user can easily type in an account if it isn’t a part of the list. I changed the due date UI to display the integer as a date (ex: 1st instead of 1). Restored functionality to the delete long click for the budget recycler view that would break when the list was sorted. |
| 0.061 | 05/05/2025 | Dominic Drury | Added input validation for adding a new bill to a budget.  Altered the adding income UI to have a drop down for the frequency so the user can choose from weekly, biweekly, monthly, or annually and the monthly income that is displayed and used is adjusted accordingly for the varying type of income. |
| 0.07 | 05/06/2025 | Dominic Drury | Added calculate UI, added functionality for a dropdown menu for dates of the next payday and added functionality to get day of the month from the phone system and displaying it as a date (1st instead of 1). |
| 0.071 | 05/08/2025 | Dominic Drury | Added functionality to calculate button that calculates the amount of money the user will have left in their account when the next payday comes around given the amount of money left in the account by iterating through the bills listed in the selected budget. Added a delete budget button and added logic for that button to delete whatever budget is currently selected or display a toast message if no budget is selected. |
| 0.08 | 05/12/2025 | Dominic Drury | Added a black border to all fragments to improve the look of the UI. Increased the height of the fragment container to cover the budget selector, add, and delete buttons. Now the income and settings pages will cover the inappropriate UI elements. Then I added padding to the rest of the fragments, so the buttons are not covered when appropriate. Added functionality to the delete all budgets on the settings page so that all budgets and all budget items can be deleted by user. |
| 0.081 | 06/02/2025 | Dominic Drury | Made some minor UI/UX changes to improve flow and design and make the app more user friendly. Created functionality for tutorial and added a return button to the settings page. |
| 1.0 | 06/03/2025 | Dominic Drury | Final checks, testing, comment checking, and initial release. |
| 3.0 | 07/01/2025 | Dominic Drury | Made changes to button icons, button size, and added a functionality for a savings button that will add a bill to a budget to allow a user to enter a desired savings amount and goal date, and the needed monthly amount will be added to the bill. Added edit functionality to budgets and incomes. |

# Project Overview

## Version

3.0

## Purpose

BudgetBuddy is a mobile application designed to help users effectively manage their personal budgets. It allows users to create and track budgets, categorize expenses, and calculate disposable income by subtracting upcoming bills from their available finances. With an intuitive interface and focused features, BudgetBuddy simplifies day-to-day financial management and helps users stay on top of their spending and saving goals.

## Scope

BudgetBuddy is developed as a mobile application for Android devices aimed at helping users manage their personal finances efficiently. The application allows users to:

* Create and maintain personal budgets by listing bills, expenses, and income sources.
* Categorize budget items and income entries for better organization.
* Calculate available funds by subtracting upcoming expenses from total available finances, specifically factoring in the time until the next payday.
* Delete all budget data through the settings interface.
* Access a built-in tutorial to assist first-time users with app functionality.

The scope of BudgetBuddy focuses primarily on individual and small-group budget tracking and calculation. It is intended for personal use and does not include features such as:

* Automatic bank account integration or real-time financial transactions.
* Complex financial forecasting or investment tracking.
* Cross-platform availability (currently limited to Android devices).
* Cloud-based real-time synchronization (future implementation planned via Google Sign-In integration).

The application utilizes a local SQLite database for data persistence and operates without requiring a continuous internet connection, ensuring user data privacy and offline accessibility.

# Requirements

## Functional

* User can create a new budget by entering a name, associated bills, due dates, and amounts.
* User can add new budget items (bills) with bill name, due date, and amount.
* User can add income sources with details like source name, frequency, and amount.
* User can view a list of budget items and income entries.
* User can calculate leftover finances by subtracting upcoming expenses from total finances based on the next payday.
* User can delete individual budget items or income entries via a long-click interaction.
* User can delete all data through a settings option.
* User can access a tutorial that explains the core functionality.
* RecyclerViews should allow for sorting (headers can sort entries when clicked).
* App should persist data locally even after the app is closed (using SQLite database).
* (Future/Optional) User login and syncing with Google Sign-In (planned for future).

## Non-Functional

* Performance: The app should load and display budget and income lists quickly (under 2 seconds for normal data volumes).
* Usability: Interfaces should be simple, intuitive, and require minimal learning (backed by a built-in tutorial).
* Offline Availability: The app must function without an internet connection.
* Data Integrity: Data entered by the user must be reliably saved and retrieved from the local database.
* Responsiveness: Layouts should adjust appropriately to different Android screen sizes (phones and tablets).
* Security: Sensitive financial data is stored locally; no transmission over the network unless Google Sign-In is added.
* Maintainability: Codebase should be organized and commented for easy maintenance and future updates.
* Scalability: The system should support budgets with 100+ entries without a significant slowdown.
* Battery Efficiency: The app should not perform background tasks that unnecessarily consume battery life.

## Technical

* Platform: Android OS (minimum SDK version 24, Android 7.0 Nougat or higher).
* Language: Java.
* IDE: Android Studio.
* Database: SQLite (local storage).
* UI Components:
  + RecyclerView for displaying budget and income lists.
  + ConstraintLayout and LinearLayout for responsive design.
* Dependencies:
  + AndroidX libraries (RecyclerView, ConstraintLayout, etc.).
  + (Optional/Future) Google Sign-In API for authentication and cloud sync.
* Minimum Device Specs:
  + RAM: 2 GB
  + Storage: At least 50 MB free for app and data.
* Version Control: Git for source control.
* Permissions:
  + Storage access (for local database usage).
* Testing Devices:
  + Emulated and physical devices on Android 8.0 (Oreo) to Android 13 (Tiramisu).

# Design

## System Architecture

BudgetBuddy follows a 3-layer architecture:

1. Presentation Layer (UI Layer):

* Activities (MainActivity, AddBudget, AddIncome, AddBudgetItem) and Fragments (SettingsFragment) handle user interaction.
* RecyclerView Adapters (BudgetRecyclerViewAdapter, IncomeRecyclerViewAdapter, BudgetPercentRecycleViewAdapter) manage how lists of data are displayed.
* Tutorial Activity offers onboarding help for users.

1. Business Logic Layer:

* Calculate.java handles calculation logic for leftover finances after bills.
* Business rules and interaction flows are handled in the Activities (e.g., when to calculate, how to update the view).
* Sorting logic is implemented in the RecyclerView adapters for dynamic list reordering.

1. Data Layer (Persistence Layer):

* SQLite Database is used for local data storage.
* BudgetItemsDatabaseHelper manages database creation and versioning.
* BudgetItemsDataAccessObject, IncomeDataAccessObject handle data CRUD operations (Create, Read, Update, Delete).
* BudgetItem and IncomeItem are model classes representing the data schema.
* No remote data — all data is stored locally on the device.

## User Interface Design

The user interface is made to be as simple and straightforward as possible. With financial issues being one of the biggest stress factors to begin with, this app seeks to help alleviate that stress from the first day of installation.

### Screens

1. MainActivity (Home Screen):

* Displays a list of all budgets and expenses.
* Offers options to add new budget items and view existing ones.
* Includes a "Calculate" button to determine remaining finances after bills.

1. AddBudget Activity:

* Form-based layout for creating a new budget.
* Users input budget title and initial information.

1. AddBudgetItem Activity:

* Allows users to add specific bills/expenses to a budget.
* Supports free text entry and selection from previous bill types.

1. AddIncome Activity:

* Form for entering income sources.
* Inputs include income name, amount, and frequency.

1. SettingsFragment:

* Provides user options such as viewing a tutorial or clearing all budgets.
* Future enhancement: Google Sign-In integration for syncing data.

1. Tutorial Activity:

* Onboarding guide explaining core features.
* Provides screenshots and descriptions of app functionality.

1. Calculate Screen (Result shown inline on Home Screen):

* Displays calculated available funds after subtracting all bills due before the next payday.

### Layout and Components

| **Screen** | **Key UI Components** |
| --- | --- |
| Home | RecyclerView (Budgets/Expenses List), Buttons (Add, Calculate) |
| Add Budget | EditText (Title), Button (Save Budget) |
| Add Budget Item | EditText (Bill Name, Amount), Dropdown (Previous Bill Types), Button (Save) |
| Add Income | EditText (Source Name, Amount), Dropdown (Frequency), Button (Save) |
| Settings | Buttons (View Tutorial, Delete All Data) |
| Tutorial | ImageView (Screenshots), TextView (Descriptions) |

### UI Elements Used

* ConstraintLayout: For flexible and responsive screen designs.
* RecyclerView: For efficiently listing budgets and incomes.
* EditText: For user data input.
* Buttons: For form submission and navigation.
* TextViews: For displaying headings, labels, and calculated results.
* Spinners / Dropdowns: For selecting from previously entered bill names and income frequencies.

## Database Design

BudgetBuddy uses a local SQLite database to store user data persistently on the device. The database design is simple and relational, with two main tables:

1. BudgetItems Table — Stores all budget items (bills/expenses).

| **Field Name** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| id | INTEGER | PRIMARY KEY AUTOINCREMENT | Unique ID for each budget item. |
| name | TEXT | NOT NULL | Name/description of the budget item (bill). |
| amount | REAL | NOT NULL | Amount due for the budget item. |
| dueDate | TEXT | NOT NULL | Due date for the bill (stored as string date). |
| type | TEXT | NULLABLE | Category/type of the budget item (optional). |

1. IncomeItems Table — Stores all income sources.

| **Field Name** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| id | INTEGER | PRIMARY KEY AUTOINCREMENT | Unique ID for each income entry. |
| name | TEXT | NOT NULL | Source of income (e.g., Job, Freelance). |
| amount | REAL | NOT NULL | Amount of income. |
| frequency | TEXT | NOT NULL | Frequency (e.g., Weekly, Monthly). |

Each table has a unique identifier and relevant financial fields.

There is no direct foreign key relationship between BudgetItems and IncomeItems — they are independent tables.

BudgetItems → manages expenses/bills.

IncomeItems → manages income sources.

Calculation logic (remaining balance) happens in the app layer — not via joins or relationships in the database.

# Development

## Development Environment

BudgetBuddy was developed using the following tools and technologies:

| **Tool/Technology** | **Version/Details** |
| --- | --- |
| **Operating System** | Windows 10 (Developer Machine) |
| **IDE** | Android Studio (Giraffe |
| **Programming Language** | Java (Java 8 compatibility) |
| **Android SDK** | Minimum SDK 24 (Android 7.0 Nougat) Target SDK 33 (Android 13) |
| **Build System** | Gradle (Version 8.x) |
| **Database** | SQLite (via SQLiteOpenHelper) |
| **Testing Tools** | Android Emulator, Physical Android Devices |
| **Version Control System** | Git |
| **Libraries** | AndroidX RecyclerView, ConstraintLayout |
| **Device Requirements** | Minimum 2GB RAM, Android 7.0+ devices |

## Source Control

Version control was managed using Git, ensuring consistent tracking of changes and collaboration readiness.

* Repository Initialization: .git folder initialized at project root.
* Ignored Files: Managed via .gitignore (ignoring /build, local settings, caches, etc.).
* Commits: Regular commits with meaningful messages to capture significant changes.
* Backup: Project safely backed up through Git and (optionally) synced to remote repositories (e.g., GitHub, GitLab) if collaboration was required.

## Coding Standards

BudgetBuddy adheres to basic Java coding best practices and Android development standards to ensure code readability, maintainability, and scalability.

* Java Coding Standards:
  + Naming Conventions:
    - Class Names: PascalCase (e.g., BudgetItem, IncomeItem).
    - Method Names: camelCase (e.g., calculateAvailableFunds(), addIncomeEntry()).
    - Constant Variables: UPPER\_SNAKE\_CASE.
  + Code Formatting:
    - Consistent indentation (4 spaces).
    - Braces {} always used for conditional blocks, even if single-line.
    - Proper use of access modifiers (private, public, protected).
  + Commenting:
    - JavaDoc comments for major classes and methods.
    - Inline comments for complex logic blocks (e.g., calculation, data access).
  + File Organization:
    - Logical separation of UI (Activities, Fragments), Business Logic, and Data (DAO, Models).
    - Error Handling:
    - Proper try-catch blocks around database operations to handle exceptions gracefully.
* Android Best Practices:
  + Layout Design:
    - Used ConstraintLayout for responsive UI design.
    - Avoided hard-coded dimensions by using dp and sp units and externalized string and dimension resources.
  + RecyclerView Optimization:
    - ViewHolder pattern used correctly to enhance performance.
  + Database:
    - Local data persisted using SQLite with structured DAO classes.
    - Separation of concerns — no direct SQL in UI classes.
  + Activity Lifecycle Management:
    - Respect for lifecycle methods (onCreate(), onResume(), onPause()) for resource handling.
    - Permissions:
    - Minimal use of runtime permissions (only local storage/database access; no sensitive permissions).

# Testing

## Testing Strategy

BudgetBuddy was primarily tested using manual testing to ensure that all major features worked as expected under normal and edge-case scenarios.

The testing strategy included:

* Functional Testing: Verifying each function (budget creation, income entry, calculation) works as intended.
* UI/UX Testing: Ensuring layouts render correctly across different screen sizes and Android versions.
* Data Persistence Testing: Confirming data is saved, retrieved, and deleted properly via SQLite database.
* Boundary Testing: Checking app behavior with maximum and minimum values (e.g., very large numbers for bills/incomes).
* Device Compatibility Testing: Running the app on Android Emulators (different Android versions) and physical devices.
* Error Handling Testing: Verifying the app does not crash when given invalid input (e.g., empty fields, zero amounts).

No automated unit or UI testing frameworks (e.g., JUnit, Espresso) were used for this project due to scope and resource constraints.

## Test Cases

| **Test Case ID** | **Description** | **Expected Result** | **Status** |
| --- | --- | --- | --- |
| TC-001 | Add a new budget with valid data | Budget appears in the main list | Passed |
| TC-002 | Add a new budget item (bill) with valid data | Bill appears under the correct budget | Passed |
| TC-003 | Add a new income entry with valid data | Income appears in the income list | Passed |
| TC-004 | Calculate remaining finances after adding bills | Correct amount is displayed | Passed |
| TC-005 | Delete individual budget item | Selected budget item is removed from list | Passed |
| TC-006 | Delete all budgets via Settings | All budget and income data is cleared | Passed |
| TC-007 | View tutorial from Settings | Tutorial screens open and are navigable | Passed |
| TC-008 | Add bill/income with empty fields | Proper validation prevents submission | Passed |
| TC-009 | Rotate device during operation | UI remains stable without crashes | Passed |
| TC-010 | Reopen app after closing | Previously saved data loads correctly | Passed |
| TC-011 | Input very large numbers for bills and incomes | App handles large inputs gracefully | Passed |
| TC-012 | Launch on Android 7.0, 9.0, 12.0 | App runs successfully on multiple versions | Passed |

## Test Results

All major features and use cases were manually tested across multiple Android versions (7.0 Nougat to 12.0 Snow Cone).

The app consistently met expectations with no critical bugs or crashes observed.

Key findings:

* User inputs are properly validated.
* Calculations for available finances are accurate.
* Data persistence works reliably between sessions.
* Layouts are responsive and adapt to different screen sizes.
* The app performs smoothly with no noticeable lag, even with larger datasets (100+ entries).

Overall, BudgetBuddy passed all manual test cases successfully and is considered stable for release.

# Deployment

## Deployment Process

The deployment process for BudgetBuddy involves preparing the application for release on Android devices through manual APK generation.

Steps:

1. Code Review and Final Testing:

* Ensured all manual test cases passed.
* Verified UI consistency and app stability on multiple Android versions.

1. APK Generation:

* Built a release APK using Android Studio:
  + Selected Build → Generate Signed Bundle / APK.
  + Chose APK, configured the keystore for app signing.
  + Set build variant to release to optimize and shrink the APK.
* Used ProGuard rules to obfuscate code and minimize APK size.

1. Device Testing:

* Installed the release APK on multiple physical Android devices for final verification.

1. Distribution:

* The release APK is ready for deployment to:
  + Android devices via direct APK installation.
  + Future submission to the Google Play Store after completing store listing details (screenshots, description, privacy policy, etc.).

## Versioning

BudgetBuddy follows **Semantic Versioning (SemVer)** in the format:

MAJOR.MINOR.PATCH

## Release Notes

Version 1.0.0 — Initial Release

Release Date: 06/03/2025

Features:

* Create and manage budgets.
* Add bills/expenses with due dates and amounts.
* Add income sources with frequency and amount.
* Calculate available funds after upcoming bills.
* Delete individual or all budget and income entries.
* View built-in tutorial for onboarding.
* Offline-first app — no internet required for core functions.
* Responsive layouts for multiple device screen sizes.
* Local data storage with SQLite for secure data persistence.

Known Limitations:

* No real-time cloud sync (planned for future release).
* No budget sharing or multi-user collaboration.
* No dark mode UI (planned enhancement).

# Maintenance

BudgetBuddy will be supported and maintained to ensure continued usability, stability, and relevance over time.

**Maintenance Activities Include**:

* **Bug Fixes**:
  + Identify and resolve functional bugs that impact user experience.
  + Provide quick patches for critical issues (e.g., crashes, data loss).
* **Feature Enhancements**:
  + Implement improvements based on user feedback (e.g., new features like Google Sign-In sync or dark mode).
  + Ensure compatibility with new Android OS versions and device form factors.
* **Performance Optimization**:
  + Monitor app performance and optimize memory usage, battery consumption, and UI responsiveness.
* **Security Updates**:
  + Apply security best practices to ensure the safety of local data storage.
* **Database Maintenance**:
  + Upgrade database schemas gracefully without losing user data when necessary (e.g., adding new fields or tables).
* **User Support**:
  + Provide a channel for users to report issues or request features (e.g., email support or GitHub Issues).

**Maintenance Cycle**:

* **Minor Updates**: Every 3–6 months or as needed based on bug reports and user feedback.
* **Major Updates**: Annually or with significant new features.
* **Android OS Compatibility Updates**: Within 2 months of a major Android release.

## Bug Reporting and Tracking

**Bug Reporting**:

* Users can report bugs via:
  + A dedicated support email: *druryllcappdev@gmail.com*
  + A GitHub Issues page: *https://github.com/ZarZar1873/BudgetBuddy*

**Bug Tracking**:

* Internally, bugs and enhancement requests are logged and tracked using:
  + **GitHub Issues**
  + **Manual Bug Tracker** (Excel sheet) each bug logged with:
    - Unique Bug ID
    - Date Reported
    - Summary
    - Steps to Reproduce
    - Severity (Critical, Major, Minor)
    - Status (Open, In Progress, Fixed)

**Bug Prioritization**:

* **Critical Bugs**: Immediate fix and hot-patch release.
* **Major Bugs**: Addressed in the next maintenance update.
* **Minor Bugs**: Logged and addressed in future planned updates.

# Appendices

## Glossary

| **Term** | **Definition** |
| --- | --- |
| **APK** | Android Package — the file format used to distribute and install apps on Android devices. |
| **SQLite** | Lightweight, embedded SQL database engine used for local storage in mobile applications. |
| **RecyclerView** | A flexible Android UI component used to efficiently display large sets of data. |
| **ConstraintLayout** | A powerful Android layout that allows positioning and sizing widgets flexibly. |
| **Semantic Versioning (SemVer)** | A versioning scheme using Major.Minor.Patch format to indicate app versions. |
| **Git** | A distributed version control system for tracking changes in source code. |
| **Manual Testing** | Testing performed manually by a person without using automated tools. |
| **Gradle** | Build automation tool used in Android Studio for compiling and packaging apps. |
| **Google Sign-In** | A service that allows users to authenticate with their Google account in apps. |
| **Android SDK** | Software Development Kit — provides the API libraries and developer tools for building Android apps. |
| **Draw.io** | An online diagramming application used for creating wireframes and diagrams. |

## References

* Android Developers Official Documentation — <https://developer.android.com/>
* SQLite Official Documentation — https://www.sqlite.org/docs.html
* Draw.io Online Diagramming Tool — https://app.diagrams.net/
* Git Official Documentation — <https://git-scm.com/doc>
* Stack Overflow — for code troubleshooting and debugging assistance.
* SNHU Coursework — Prior academic materials and templates used as references.
* General articles and templates from:
  + Atlassian Confluence for Software Documentation Templates
  + Various Android development blogs for best practices

**Wireframes**

Wireframes for BudgetBuddy were designed using **Draw.io** to outline initial screen layouts and navigation flow.  
These wireframes served as the base for the user interface design and user flow during development.

*(Insert link or attached screenshots of wireframes here if submitting digitally)*

**Wireframes Include**:

* MainActivity (Home Screen)
* AddBudget Screen
* AddIncome Screen
* AddBudgetItem Screen
* Settings Screen
* Tutorial Screen

## Additional Notes

* The initial template for this documentation was created by combining **online resources**, **industry templates**, and **previous coursework from SNHU**.
* BudgetBuddy was developed as a solo project with a focus on learning and applying full-stack mobile development practices.
* Future improvements may include:
  + Adding Google Sign-In for cloud data synchronization.
  + Enhancing UI/UX with Material Design Components.
  + Implementing dark mode support.
  + Introducing automated testing for regression detection.
* The project is currently in version **1.0.0**, and will continue to evolve based on user feedback and future goals.