Software Design Document

Introduction

This app is a practical tool that helps students take control of their finances, develop better money management habits, and prepare for a better financial future.

In today's digital world, managing personal finances has become an important skill, especially for university students. Many students face problems like overspending and lack of proper financial planning, often ending up with little money at the end of each month. The "Student Budget Tracker" app, developed using Android Studio, is designed to help students manage their finances more effectively.

This app provides a simple and useful tool for students to record their daily expenses and income, check their financial status, and make better spending plans. With features like personalized theme customization, intuitive chart, and customizable categories of expense and income, it can meet the different needs of users and manage their data in a humanized way.

The main goal of this project is to help university students improve their financial management skills. By using this app, students can build good habits in managing money, avoid unnecessary spending, and plan for their future more effectively. With an easy-to-use interface and fast data processing, the app also ensures a smooth and convenient user experience. This app is also a practical tool that helps students take control of their finances, develop better money management habits, and prepare for a better financial future.

Requirements Specification

Target Users

University students who face challenges in managing their personal finances.

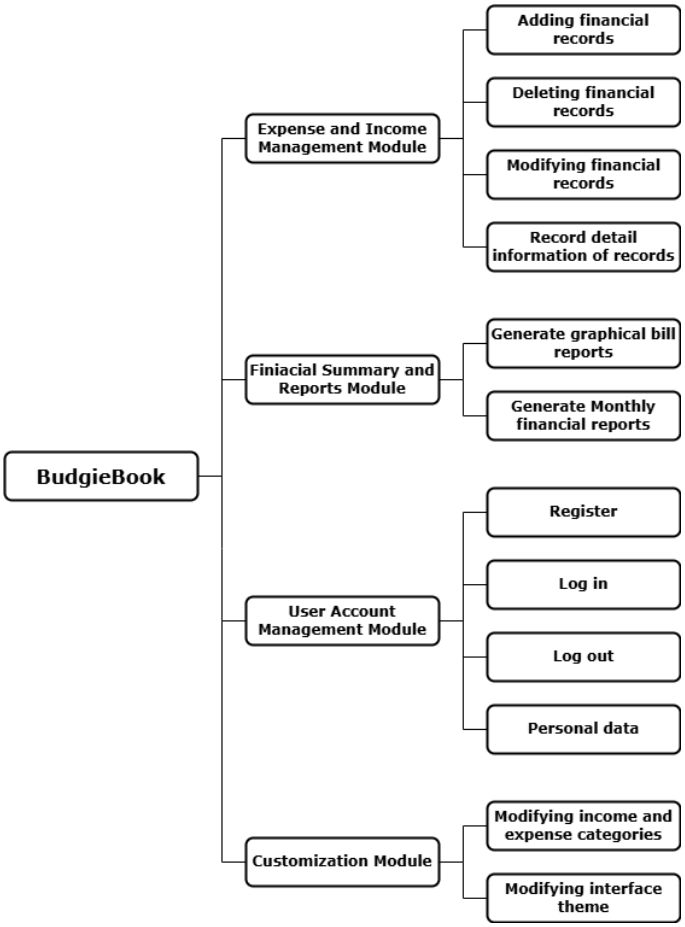
User Requirements:

1. Record daily expenses
2. Record daily incomes.
3. Set categories of every expense and income.
4. View intuitive chart of expense and income.
5. View clear summaries and detailed financial reports.
6. Plan personal budgets effectively and avoid unnecessary spending.
7. A simple and intuitive user interface.
8. Customizable modules to meet individual needs.
9. Features for secure data synchronization and backup.
10. Bill reminders for fixed expenses, such as water bills and electricity bills.

Functional Requirements

1. **Expense and Income Tracking:** Record expenditures and earnings with category classification.
2. **Category Management:** Support for adding and deleting expense/income categories.
3. **Financial Summary and Reports:** Generate income and expense reports monthly
4. **Graphical Charts for Bills:** Provide insights into spending patterns over specific periods.
5. **User Account Management:** Allow users to register, log in, and manage personal information.
6. **Data security:** Ensure secure by login with validation mechanisms and backup to prevent data loss.
7. **Customization:** Allow users to personalize categories and interfaces. Extendable modules to meet specific user requirements.
8. **Bill reminder:** Allow users to set fixed expenses and incomes without manually recording them every month.

Overall Design



Structure of the App

1.Expense and Income Management Module:

a) Enables users to record detailed information about their daily expenses and income.

b) Supports adding, modifying, and deleting financial records.

c) Allows users to categorize transactions (e.g., food, travel, tuition).

2.Financial Summary and Reports Module:

a) Provides visual representations of financial data, such as pie charts and bar graphs, for a clearer understanding of income and expenditure trends.

b) Provides monthly income and expenditure reports.

3.User Account Management Module:

a) Includes features for user registration, login, and profile management.

b) Ensures data security through user authentication and password management.

4.Customization Module:

a) Offers personalized settings for income and expense categories.

b) Allows users to modify the app's interface theme for a personized experience.

Main function

1.Record Keeping:

a) Users can record income and expenses with details such as category, description, and date.

b) Supports adding custom categories to meet individual needs.

2.Reporting and Analysis:

a) Generates dynamic financial reports, including summaries of monthly income and expenses.

b) Provides graphical data representation for quick insights into spending habits and proportion of different type of income and expenditure..

3.User Management:

a) Securely handles user account creation and login.

b) Allows updates to personal information, such as username and password.

4.Ease of Use:

a) Simple, intuitive interface for smooth navigation, and various theme provided to select.

b) Quick data entry options and efficient error handling to enhance user experience.

User Interface Design

1.Registration & Login page: In keeping with our name 'BudgieBook', we included a small parrot on the login screen. Moreover, the parrot will cover its eyes when the user enters the password, increasing the user's sense of interaction.

2.Main page: The top of the main page, from left to right, shows this month's expenses, this month's income, and this month's summary balance. The bottom half of the main page is the specific classification of the amount of each income and expenditure, click on each income and expenditure to view the detailed information. The plus sign at the bottom of the page is used to create a new account. Click on the three bars in the upper left corner to activate the sidebar, and you can change the month in the upper right corner to view the income and expenditure of the previous months. Overall, the homepage is simple, intuitive, and fully functional.

3.**Chart page:** In order to allow users to see the proportion of different types of expenditure and income more intuitively, the software provides pie charts to visually display different types of income and expenditure. Click in the middle of the pie chart to switch between income and expenditure.

4.**Billing page:** This page provides a variety of options to describe an income and expenditure. You can set the type, payment method, payment time, and add a note. In addition, we provide a variety of categories and ICONS. Can help the user more intuitively categorize his income and expenses. Which is very humanized.

5.Category management page: In order to better meet the needs of all users, users can also customize the income and expenditure categories, users can adjust the order according to the frequency and importance of the categories, easy to use. The UI here is designed as three bars, like handles. Users can drag to sort categories.

Key Technologies

Core code (Bill add, delete edit operation):

```
public class BillAddActivity extends BaseMVPActivity<BillContract.Presenter>
        implements BillContract.View, View.OnClickListener {

    private TextView incomeTv;      //income button
    private TextView outcomeTv;     //outcome button
    private TextView sortTv;        //shou selected category
    private TextView moneyTv;       //money
    private TextView dateTv;        //time select
    private TextView cashTv;
```

```
    private ImageView remarkIv;
    private ViewPager viewpagerItem;
    private LinearLayout layoutIcon;

    //calculator
    protected boolean isDot;
    protected String num = "0";                    //integer part
    protected String dotNum = ".00";               //decimal part
    protected final int MAX_NUM = 9999999;       //biggest integer
    protected final int DOT_NUM = 2;               // the largest number of decimal
places
    protected int count = 0;
    //selector
    protected List<String> cardItems;
    protected int selectedPayinfoIndex = 0;          //choose payment method
    //viewpager data
    protected int page;
    protected boolean isTotalPage;
    protected int sortPage = -1;
    protected List<BSort> mDatas;
    protected List<BSort> tempList;
    protected List<View> viewList;
    protected ImageView[] icons;

    //Record the category after last click
    public BSort lastBean;

    public boolean isOutcome = true;
    public boolean isEdit = false;
    //old Bill
    private Bundle bundle;

    //select time
    protected int mYear;
    protected int mMonth;
    protected int mDay;
    protected String days;

    //description
    protected String remarkInput = "";
    protected NoteBean noteBean = null;


/*****************************************************************
```

```
***/
    @Override
    protected int getLayoutId() {
        return R.layout.activity_add;
    }

    @Override
    protected void initData(Bundle savedInstanceState) {
        super.initData(savedInstanceState);

        // set date selector initial date
        mYear = Integer.parseInt(DateUtils.getCurYear(FORMAT_Y));
        mMonth = Integer.parseInt(DateUtils.getCurMonth(FORMAT_M));
        mDay = Integer.parseInt(DateUtils.getCurDay(FORMAT_D));
        //set current date
        days = DateUtils.getCurDateStr("yyyy-MM-dd");

        bundle = getIntent().getBundleExtra("bundle");

        if (bundle != null) {       //edit
            isEdit = true;
            //set bill date
            days = DateUtils.long2Str(bundle.getLong("date"), FORMAT_YMD);
            isOutcome = !bundle.getBoolean("income");
            remarkInput = bundle.getString("content");
            DecimalFormat df = new DecimalFormat("######0.00");
            String money = df.format(bundle.getDouble("cost"));
            //rounding decimal
            num = money.split("\\.")[0];
            //obtain decimal part
            dotNum = "." + money.split("\\.")[1];
        }
    }

    @Override
    protected void initWidget() {
        super.initWidget();
        incomeTv = findViewById(R.id.tb_note_income);
        outcomeTv = findViewById(R.id.tb_note_outcome);
        sortTv = findViewById(R.id.item_tb_type_tv);
        moneyTv = findViewById(R.id.tb_note_money);
        dateTv = findViewById(R.id.tb_note_date);
        cashTv = findViewById(R.id.tb_note_cash);
        remarkIv = findViewById(R.id.tb_note_remark);
```

```
        viewpagerItem = findViewById(R.id.viewpager_item);
        layoutIcon = findViewById(R.id.layout_icon);

        //set bill date
        dateTv.setText(days);
        //set amount of bill
        moneyTv.setText(num + dotNum);
    }
```

Difficulties during development:

1. **Dynamic Addition of Categories:**
   Challenge: Allowing users to add and modify income or expense categories dynamically while ensuring the app's stability.
   Solution: Utilized a flexible database schema that supports the addition of new categories without requiring schema changes. Error-handling routines were added to prevent duplicate or invalid category entries.
2. **User Registration and Login:**
   Challenge: Creating a secure and user-friendly registration and login system, including password recovery and input validation.
   Solution: Developed a robust authentication system with proper encryption for storing passwords. Input validation was enforced to avoid weak credentials and improve security.

Testing and User Experience Analysis

By using the WeTest platform, it was found that there were no high risk and medium risk vulnerabilities after testing, and there were two low risk vulnerabilities and one security risk

The first low-risk vulnerability:

The following node Settings can be invoked externally to expose sensitive information.

android:allowBackup=true

Explanation: android:allowBackup property controls whether the application's data is allowed to be backed up. If this property is set to true, application data (such as SharedPreferences, database files, application files, etc.) may be backed up to external storage or accessed by other applications. Without proper encryption or protection, this can lead to sensitive data being compromised.

Problem: Malicious users can obtain sensitive information by backing up application data.

Without proper encryption, backup data can be restored to unauthorized devices, exposing the user's **private** information.

Solution:

Set android:allowBackup to false to disable application data backup.

The second low-risk vulnerability:

The following node Settings can be invoked externally to expose sensitive information.

android:debuggable=true

Explanation: android:debuggable property controls whether the application allows debugging. If this property is set to true, the developer has enabled debugging in the application, which means that the developer can use the debugger to debug the running of the application. However, if this feature is enabled in a production environment, it exposes the internal implementation of the application, allowing attackers to obtain sensitive information through debugging tools or exploit vulnerabilities in the application.

Problem: A malicious user can reverse engineer the application through a debugging tool such as Android Studio or adb to view the application code, extract sensitive information such as hard-coded passwords, API keys, etc., or modify the application behavior. In debug mode, the attacker can also access more debugging interfaces (such as logcat output) to further understand the inner workings of the application.

Solution: In the release version, be sure to set android:debuggable to false to ensure that debugging is disabled in production.

Security risk

Risk of exposure of BroadcastReceiver components

Explanation: Broadcasting is a communication mechanism in the Android system that allows data transfer and event notification between different applications or components. When one component sends a broadcast, other components can register and receive that broadcast. If the broadcast does not have proper access controls, external applications can receive the broadcast content, exposing sensitive information about the application.

In tests, AppCompatDelegateImpl $AutoNightModeManager smali this line of code that could be exposed to the external access radio, may reveal that the application of internal status, or other sensitive information.

Problem: Exposed broadcast: An external app can send or receive this broadcast if

the correct permission limits are not set, which can lead to sensitive data leakage.

Solution: Set android:exported="false". By default, all broadcast receiver components are accessed by other applications. If you do not intend for an external application to receive the broadcast, you should set android:exported="false" to prevent the broadcast receiver from being externally called.

Experience analysis

User feedback can be grouped into three categories:

1. **Better user interface design:**
   - **a)** Add more dynamic effects.
   - **b)** Add the function to customize themes
   - **c)** Design a more user-friendly interface layout, so that users can be more convenient accounting
2. **Better interactive design:**
   - **a)** Add a function to automatically record the user's commonly used payment/income categories, intelligent recommendation next time input, reduce the operation steps.
   - **b)** Develop a date smart tip function: The date of the current day is selected by default to avoid repeated operations. You can also change the date to supplement records.
3. **Better feedback mechanism:**
   - **a)** Add the ability to automatically generate summary statements such as "Savings grew 5% this month, significant progress!" "Or" Spend more, see where you can save."
   - **b)** Add achievement systems, encourage users to use the app, and motivate users to maintain good billing habits.

Conclusion

The development of the "BudgieBook" app was a meaningful and challenging experience for our team. This project aimed to provide university students with a simple and practical tool to manage their daily finances. Through our efforts, we successfully developed an app with essential features like expense and income tracking, customizable categories, and financial summaries. The app has a clean and intuitive interface, making it easy for users to record their spending and understand their financial habits.

During the development process, we encountered some difficulties. Designing a system that allowed users to dynamically add or edit categories required careful planning and testing to ensure stability. Another challenge was creating financial reports that were both visually clear and easy to generate. Balancing functionality and user experience was a key focus throughout the project.

For future improvements, we plan to add more features, such as subcategories and spending alerts, to make the app even more helpful. Enhancing the design and providing

more options for customization could also improve user satisfaction.

This project not only helped us apply what we learned in class but also taught us how to work together to solve real-world problems. We believe this app can make financial management easier for students and help them build better financial habits.