Introduction

In today's fast-paced world, managing personal expenses is crucial for financial stability and planning. To aid individuals in organizing their finances effectively, a console-based expense tracker application has been developed using Java. This application provides various features to record, track, and analyze expenses, making it easier for users to monitor their spending habits and manage their budgets efficiently. This report discusses the design, implementation, and functionalities of the expense tracker application, including a step-by-step procedure of the coding process.

Step 1: Requirement Analysis

The first step in developing the expense tracker application was to conduct a detailed requirement analysis. The development team collaborated with stakeholders to identify the core functionalities and features that the application should offer. The key requirements included expense recording, expense categories, tracking, filtering, modification, data persistence, report generation, and a user-friendly interface.

Step 2: Technology Stack and Architecture Design

After defining the requirements, the development team selected Java as the primary programming language for implementing the application. Java was chosen for its versatility, robustness, and wide adoption in the software development community. Additionally, Java offers excellent support for data manipulation, file handling, and user input processing, making it suitable for a console-based application.

The application follows a layered architecture design, which includes:

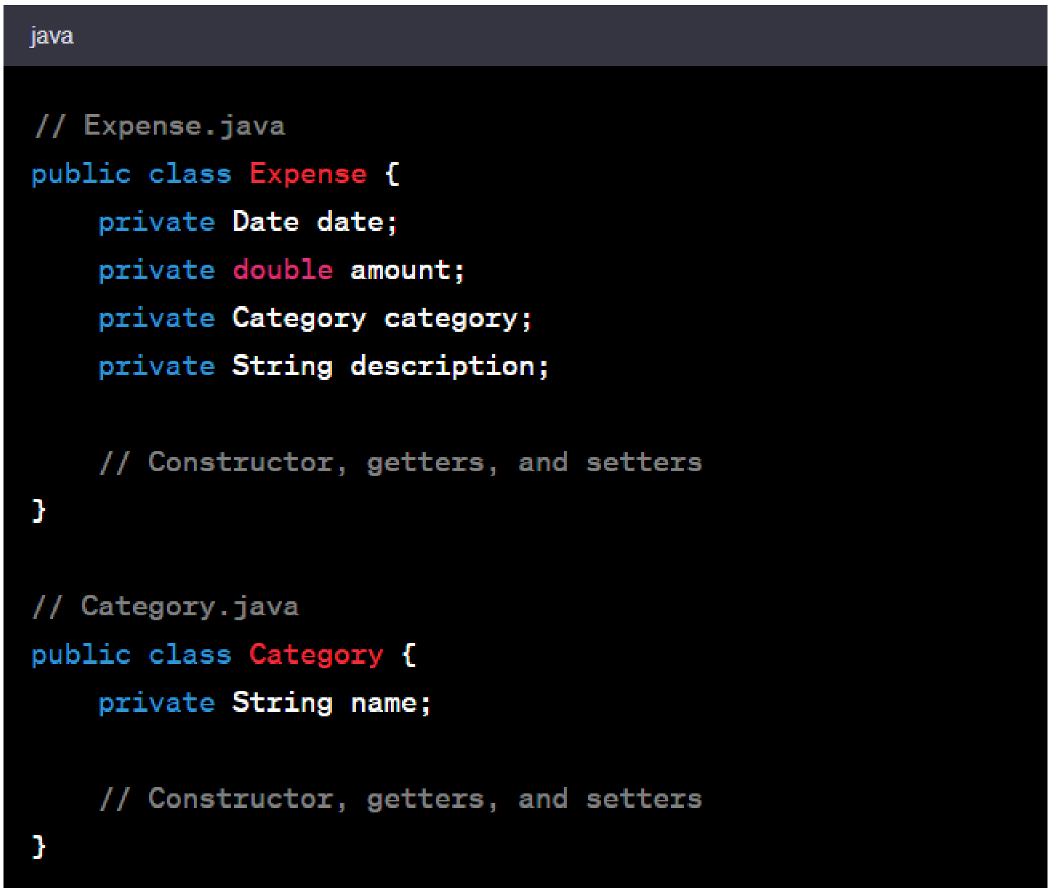
1. **Data Layer**: Responsible for data persistence and handling. It includes classes for reading and writing data to files or a database.

2. **Service Layer**: Contains the business logic and acts as an intermediary between the presentation layer and data layer. It handles expense recording, filtering, modification, and report generation.

3. **Presentation Layer**: Manages the user interface and interaction with the application. It takes user input, displays prompts, and presents the reports.

Step 3: Implementing the Data Model

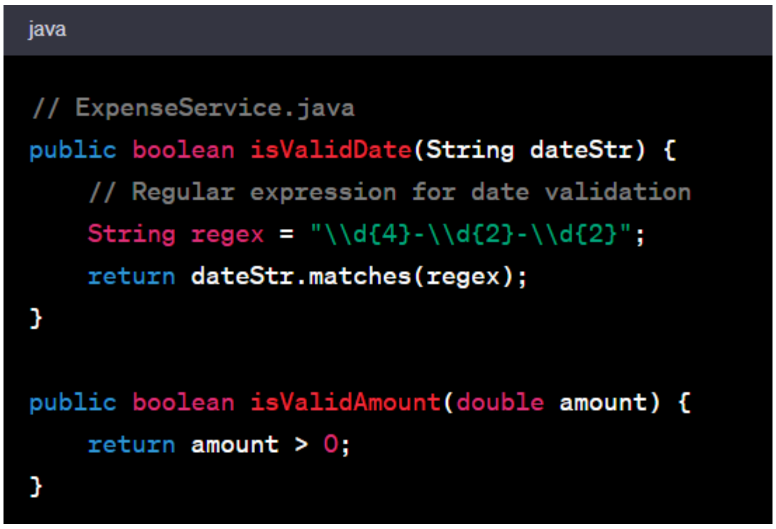
The development team began implementing the data model, starting with the Expense class. This class represents an individual expense and contains attributes such as date, amount, category, and description. Additionally, the Category class was created to represent expense categories.



```java
// Expense.java
public class Expense {
    private Date date;
    private double amount;
    private Category category;
    private String description;

    // Constructor, getters, and setters
}

// Category.java
public class Category {
    private String name;

    // Constructor, getters, and setters
}
```

Step 4: Data Validation

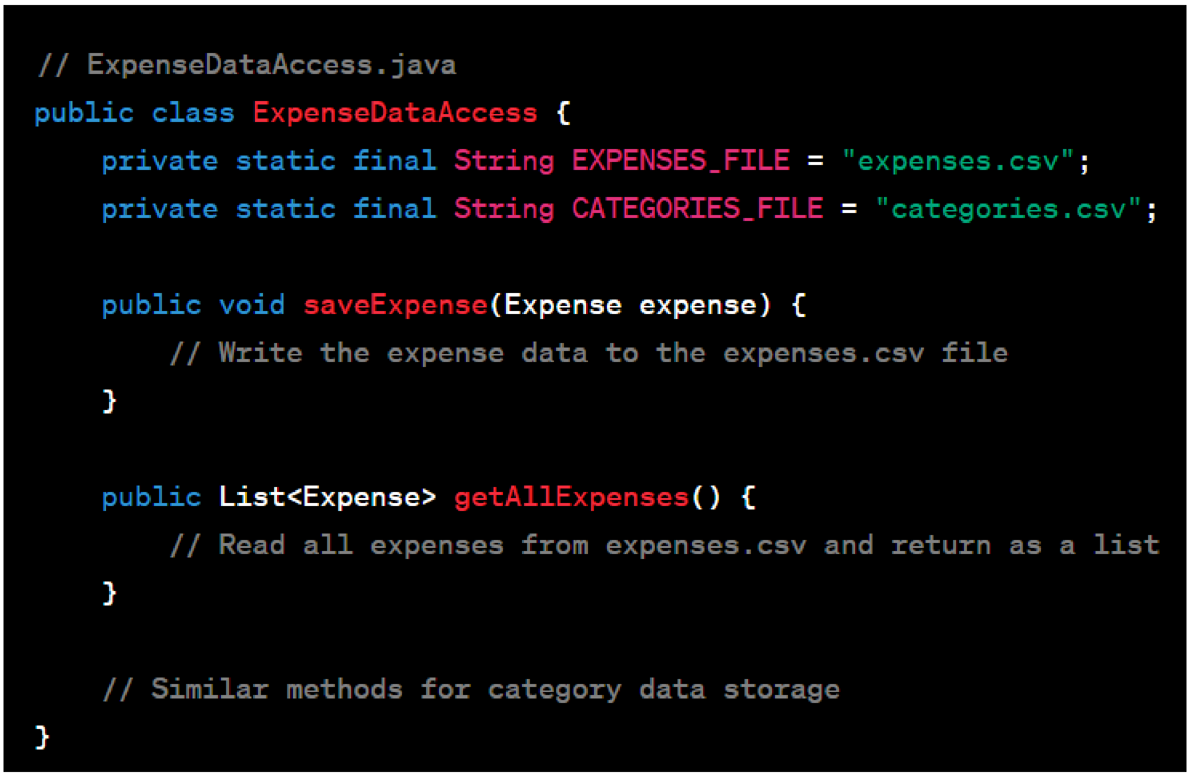
To ensure the correctness of user inputs, data validation was implemented in the application. Regular expressions were used to validate the date format, and Java's Number class was employed to ensure that the amount is a positive numerical value.


java

```java
// ExpenseService.java
public boolean isValidDate(String dateStr) {
    // Regular expression for date validation
    String regex = "\\d{4}-\\d{2}-\\d{2}";
    return dateStr.matches(regex);
}

public boolean isValidAmount(double amount) {
    return amount > 0;
}
```

Step 5: Data Storage and Persistence

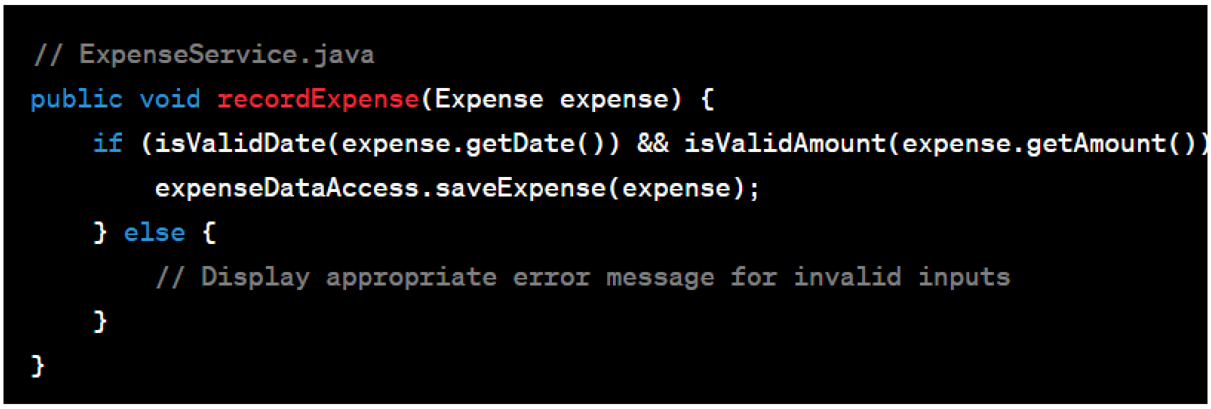
Java's File I/O API was used to implement data storage and persistence. Each expense is saved as a record in a CSV file, making it easy to read and update expense data. The application also stores category data in a separate CSV file.



```java
// ExpenseDataAccess.java
public class ExpenseDataAccess {
    private static final String EXPENSES_FILE = "expenses.csv";
    private static final String CATEGORIES_FILE = "categories.csv";

    public void saveExpense(Expense expense) {
        // Write the expense data to the expenses.csv file
    }

    public List<Expense> getAllExpenses() {
        // Read all expenses from expenses.csv and return as a list
    }

    // Similar methods for category data storage
}
```

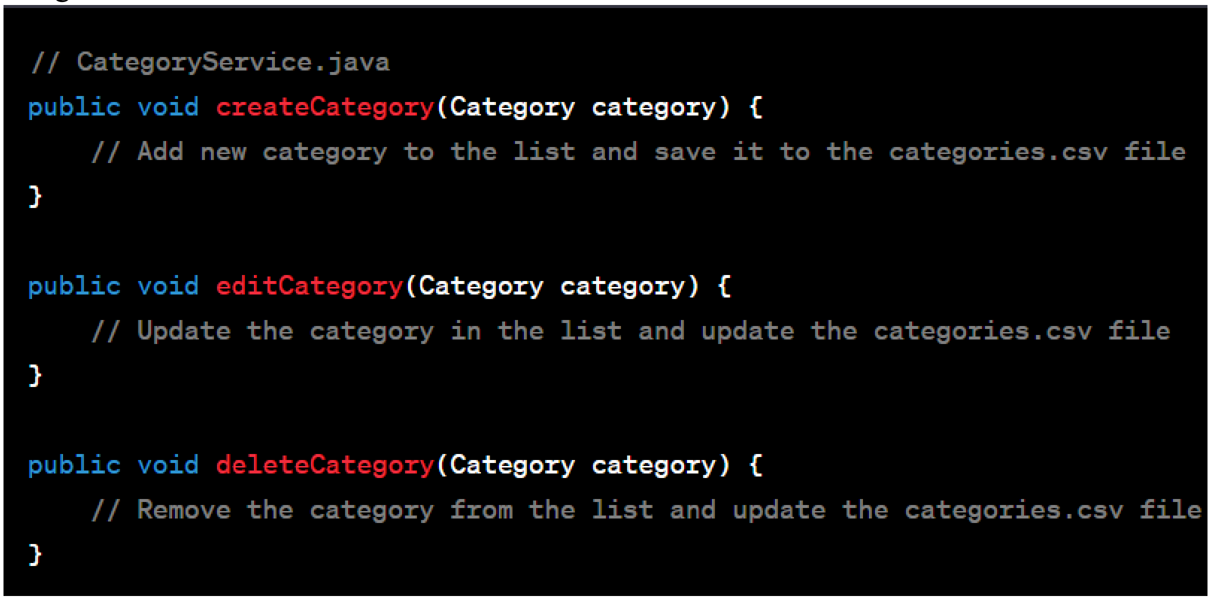
Step 6: Expense Recording

The application's ExpenseService class provides methods for recording new expenses and updating the expense data.


```java
// ExpenseService.java
public void recordExpense(Expense expense) {
    if (isValidDate(expense.getDate()) && isValidAmount(expense.getAmount())
        expenseDataAccess.saveExpense(expense);
    } else {
        // Display appropriate error message for invalid inputs
    }
}
```

Step 7: Expense Categories

The CategoryService class handles expense categories, including creating, editing, and deleting categories.


```java
// CategoryService.java
public void createCategory(Category category) {
    // Add new category to the list and save it to the categories.csv file
}

public void editCategory(Category category) {
    // Update the category in the list and update the categories.csv file
}

public void deleteCategory(Category category) {
    // Remove the category from the list and update the categories.csv file
}
```

Step 8: Expense Tracking and Filtering

The ExpenseService class also provides methods to calculate and display total expenses for a specified time period or by category. Filtering methods are implemented to display relevant expense data to the user.

```java
// ExpenseService.java
public double getTotalExpensesForDateRange(Date startDate, Date endDate) {
    // Calculate total expenses for the specified date range
}

public double getTotalExpensesByCategory(Category category) {
    // Calculate total expenses for the specified category
}

public List<Expense> filterExpenses(Date startDate, Date endDate, Category category, double minimumAmount) {
    // Filter expenses based on the specified criteria and return as a list
}
```

Step 9: Reports Generation

The ReportGenerator class handles the generation of various reports, such as monthly expense reports and category-wise expense reports, based on user input.

```java
// ReportGenerator.java
public String generateMonthlyExpenseReport(int year, int month) {
    // Generate the monthly expense report for the specified year and month
}

public String generateCategoryWiseExpenseReport(Category category) {
    // Generate the category-wise expense report for the specified category
}
```

Step 10: User-Friendly Interface

To ensure a user-friendly experience, the application utilizes Java's Scanner class to receive user input and provide clear instructions and prompts for interactions. The console interface is designed to be intuitive and easy to navigate.

```java
// ExpenseTrackerApp.java
public class ExpenseTrackerApp {
    private Scanner scanner;
    private ExpenseService expenseService;
    private CategoryService categoryService;

    // Constructor and application flow methods
}
```

Conclusion

The Java-based console expense tracker application offers a powerful solution for individuals to manage their personal expenses efficiently. Through the step-by-step coding process, the development team successfully implemented various features, such as expense recording, expense categories, tracking, filtering, modification, data persistence, report generation, and a user-friendly interface.

The application's architecture and implementation provide a robust foundation for financial management and long-term financial stability. By adhering to Java's best practices and employing appropriate data storage mechanisms, the application offers a seamless user experience and valuable insights into spending habits.

Future Enhancements

As the expense tracker application evolves, several future enhancements can be considered:

- **Graphical User Interface (GUI):** Develop a GUI version of the application to enhance the user experience and offer more interactive features.

- **Cloud-Based Storage:** Implement cloud-based data storage and synchronization to enable users to access their expense data across multiple devices.

- **Expense Analysis Algorithms:** Utilize data analysis and machine learning algorithms to provide users with personalized expense management recommendations.

- **Budget Alerts:** Introduce real-time budget alerts to notify users when they approach or exceed their predefined spending limits.

- **Multi-Language Support:** Add support for multiple languages to cater to a broader user base.

- **Expense Splitting:** Enable users to split expenses among multiple categories or users, ideal for shared expenses or group budgets.

References:

1. Java Documentation: https://docs.oracle.com/en/java/

2. Stack Overflow: https://stackoverflow.com/

3. GeeksforGeeks: https://www.geeksforgeeks.org/

4. Java Tutorials at Tutorialspoint: https://www.tutorialspoint.com/java/index.htm

5. File I/O in Java - Baeldung: https://www.baeldung.com/java-write-to-file

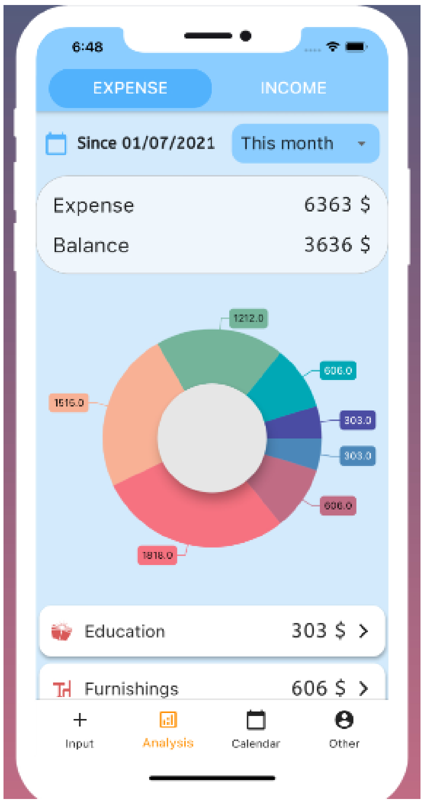
6. Regular Expressions in Java - Oracle: https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/regex/Pattern.html

7. CSV File Handling in Java - JournalDev: https://www.journaldev.com/2462/java-readwrite-csv-file

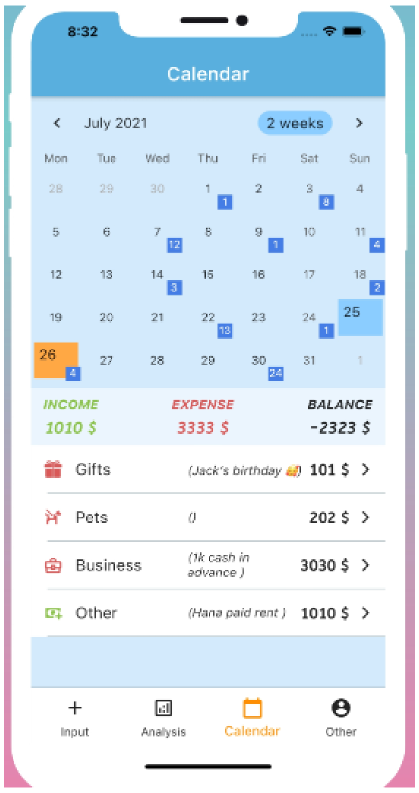
8. Design Patterns in Java - Gang of Four (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides)

9. Clean Code: A Handbook of Agile Software Craftsmanship - Robert C. Martin

Appendix

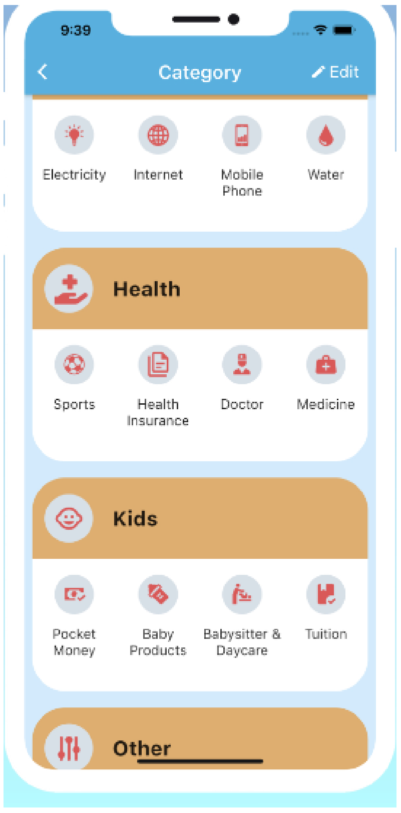


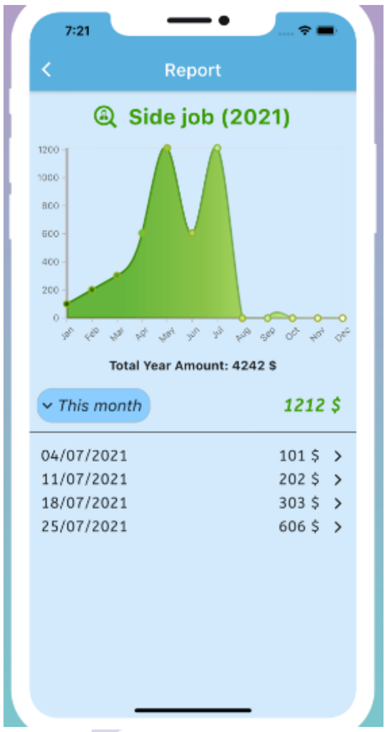
Expense 6363 $
Balance 3636 $
Education 303 $
Furnishings 606 $



INCOME 1010 $ EXPENSE 3333 $ BALANCE -2323 $

Gifts (Jack's birthday) 101 $
Pets () 202 $
Business (1k cash in advance) 3030 $
Other (Hana paid rent) 1010 $




7:21

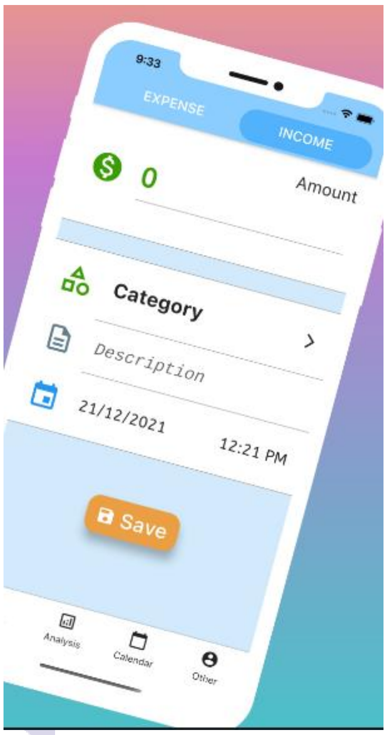
Report

Side job (2021)

Total Year Amount: 4242 $

This month 1212 $

04/07/2021	101 $
11/07/2021	202 $
18/07/2021	303 $
25/07/2021	606 $


9:33

EXPENSE INCOME

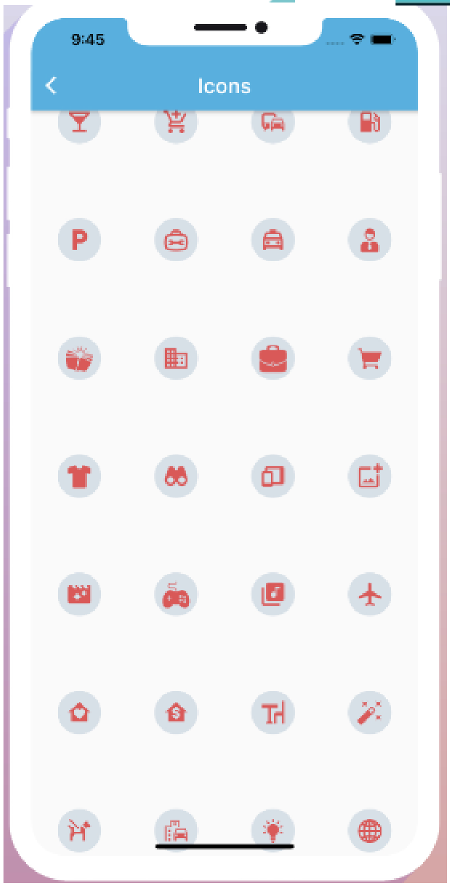
0 Amount

Category

Description

21/12/2021 12:21 PM

Save


9:45

Icons