EXPENSE MANAGER

A MINI PROJECT REPORT

Submitted by

VIGNESH C **220701317**

VIJAI T **220701319**

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023 - 24

BONAFIDE CERTIFICATE

Certified that this project report "**EXPENSE MANAGER**" is the bonafide work of

"VIGNESH C (220701317), VIJAI T (220701319) "

who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Dr.R.SABITHA
Professor and II Year Academic Head
Computer Science and Engineering,
Rajalakshmi Engineering College (Autonomous)
Thandalam, Chennai - 602 105

SIGNATURE

Ms.V.JANANEE
Assistant Professor (SG)
Computer Science and Engineering.
Rajalakshmi Engineering College (Autonomous)
Thandalam, Chennai - 602 105

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

This project involves the creation of an Expense Manager application using Python, with PyQt5 for the graphical user interface (GUI) and Pyrebase for Firebase integration. The application aims to provide users with an efficient and user-friendly platform to track and manage their daily expenses. Firebase's Realtime Database is utilized to store and synchronize expense data in real-time, ensuring users have up-to-date access to their financial information across multiple devices. Key features include secure user authentication, expense tracking with detailed entry fields, and data visualization.

The integration of Pyrebase simplifies interaction with Firebase services, allowing for seamless authentication, real-time data updates, and consistent synchronization. Users can add, edit, delete, search, and filter expenses, with changes instantly reflected across all devices. By combining PyQt5's robust GUI capabilities with Firebase's powerful backend services, this Expense Manager application provides a comprehensive solution for personal finance management.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

1.2 OBJECTIVES

1.3 MODULES

2. SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

2.2 LANGUAGES

2.2.1 PYTHON

2.2.2 FIREBASE REALTIME DATABASE

3. REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

` 3.3 ARCHITECTURE DIAGRAM

3.4 DATA MODELING IN NOSQL

4. PROGRAM CODE

5. RESULTS AND DISCUSSION

6. CONCLUSION

7. REFERENCES

Ch - 1. INTRODUCTION

1.1 INTRODUCTION

The Expense Manager application is a robust and intuitive tool crafted to assist users in efficiently tracking and managing their daily expenses. Developed using Python and PyQt5, it offers a sleek and responsive graphical interface. Integration with Firebase's Realtime Database via Pyrebase ensures secure, real-time data synchronization and storage. Key features include secure user authentication, detailed expense tracking, and insightful data visualization through charts and graphs, making personal finance management straightforward and effective across multiple devices.

1.2 OBJECTIVES

1. User-Friendly Interface:

Design an intuitive and responsive graphical user interface using PyQt5 to ensure a seamless user experience for tracking and managing expenses.

2. Secure Authentication:

Implement secure user authentication using Firebase Authentication to protect user data and provide safe access to the application.

3. Real-Time Data Management:

Utilize Firebase's Realtime Database to store and synchronize expense data in real-time, ensuring users have the most up-to-date information across all devices.

4. Comprehensive Expense Tracking:

Enable users to add, edit, and delete expense entries with detailed information, including date, category, amount, and description.

6. Search and Filter Functionality:

Allow users to efficiently search and filter their expenses based on various criteria such as date range, category, and amount for better data management.

9. Performance and Usability:

Conduct thorough testing to ensure the application performs efficiently and meets user needs effectively, delivering a smooth and reliable user experience.

1.3 Modules:

1. User Interface Module (UI):

PyQt5 Components: Handles the design and layout of the graphical user interface. Includes windows, dialogs, forms, buttons, labels, and input fields.

Navigation: Manages navigation between different sections of the application, such as adding expenses, viewing reports, and accessing settings.

2. Authentication Module:

User Registration: Allows new users to create accounts using Firebase.

User Login: Handles secure user login and session management.

Password Reset: Provides functionality for users to reset their passwords.

3. Expense Management Module:

Add Expense: Interface and logic for adding new expense entries with details like date, category, amount, and description.

Edit Expense: Allows users to modify existing expense entries.

Delete Expense: Provides functionality to remove expense entries.

Expense List: Displays a list of all expenses with options for sorting and filtering.

4. Data Synchronization Module:

Realtime Database Integration: Utilizes Pyrebase to interact with Firebase's Realtime Database for storing and retrieving expense data.

Data Sync: Ensures that changes in expense data are synchronized across all devices in real-time.

Each of these modules interacts seamlessly to provide a comprehensive and efficient expense management solution.

CH - 2. SURVEY OF TECHNOLOGIES

2.1 Software Description

The Expense Manager application integrates various technologies to provide a robust and user-friendly experience for managing personal finances. Key technologies used in the development of this application include Python and Firebase Realtime Database (NoSQL).

2.2 Languages

2.2.1 Python

Python is a high-level, interpreted programming language known for its simplicity and versatility. It is the primary language used for developing the Expense Manager application. Python is chosen due to its numerous advantages:

Ease of Learning and Use: Python's straightforward syntax makes it easy to read, write, and maintain, allowing for rapid development and prototyping.

Extensive Libraries: Python has a rich ecosystem of libraries and frameworks, such as PyQt5 for GUI development, Matplotlib for data visualization, and Pyrebase for Firebase integration, which streamline the development process.

Cross-Platform Compatibility: Python runs on various operating systems, enabling the Expense Manager to be deployed across different platforms without significant modifications.

Community Support: Python boasts a large and active community, providing ample resources, tutorials, and third-party modules that can assist in development and troubleshooting.

Integration Capabilities: Python's ability to integrate with other languages and technologies makes it a flexible choice for creating comprehensive applications.

2.2.2 Firebase Realtime Database (NoSQL)

Firebase Realtime Database is a cloud-hosted NoSQL database provided by Google Firebase. It is utilized in the Expense Manager application for storing and synchronizing expense data in real-time. The Realtime Database is chosen for several reasons:

Real-Time Data Synchronization: Firebase Realtime Database ensures that any changes to the data are instantly synchronized across all connected devices, providing users with up-to-date information at all times.

Scalability: Designed to handle large volumes of data, the Realtime Database can scale automatically to support a growing number of users and increased data loads.

Ease of Use: Firebase offers a developer-friendly API, making it straightforward to store, retrieve, and manage data within the application.

Security: With built-in security rules, Firebase allows for fine-grained control over data access and validation, ensuring that user data is protected and only accessible to authorized users.

Offline Capabilities: Firebase Realtime Database supports offline data persistence, enabling users to access and modify data even when they are offline. Once the connection is restored, changes are automatically synchronized.

Integration with Firebase Ecosystem: The Realtime Database seamlessly integrates with other Firebase services such as Authentication, Cloud Functions, and Analytics, providing a comprehensive backend solution for the Expense Manager application.

CH - 3 REQUIREMENTS AND ANALYSIS

3.1 Requirement Specification

The Expense Manager application is designed to help users efficiently track and manage their personal expenses. The requirements for this application are divided into functional and non-functional requirements.

Functional Requirements:

1. User Authentication:
    1. Users must be able to register an account and log in using Firebase
    2. Password recovery functionality must be available.
2. Expense Management:
    1. Users must be able to add new expenses, including details such as date, category, amount, and description.
    2. Users must be able to edit and delete existing expenses.
    3. Users should be able to view a list of their expenses with options for sorting and filtering.
3. Data Synchronization:
    1. Expense data must be stored and synchronized in real-time using Firebase Realtime Database.
    2. Users should be able to access their data across multiple devices.

Non-Functional Requirements:

1. Usability:

- The application must have an intuitive and user-friendly interface.
- The GUI must be responsive and easy to navigate.

2. Performance:

- The application must perform efficiently, with quick data retrieval and updates.

- Real-time synchronization should be smooth and without noticeable delays.

3. Security:

- User data must be securely stored and transmitted, ensuring privacy and protection against unauthorized access.

4. Scalability:

- The application must be able to handle increasing numbers of users and larger amounts of data without performance degradation.

3.2 Hardware and Software Requirements

Hardware Requirements:

Development Environment:

- Processor: Intel i5 or equivalent
- RAM: 8GB or more
- Storage: 256GB SSD
- Display: 1080p monitor

User Environment:

- Any modern computer or mobile device capable of running the application

Software Requirements:

Development Environment:

- Operating System: Windows, macOS, or Linux
- Python 3.8 or higher
- PyQt5 library
- Pyrebase library
- Firebase SDK

3.3 Architecture Diagram

The architecture diagram illustrates the high-level structure of the Expense Manager application, highlighting the interaction between different components.

Architecture Diagram Description:

1. User Interface (UI):

- Developed using PyQt5, it includes forms for inputting expenses, viewing data.

2. Application Logic:

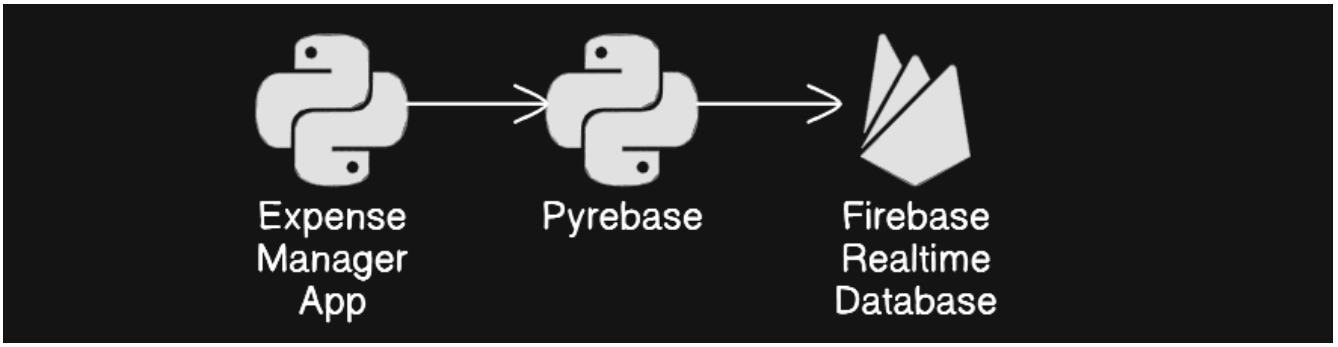
- Handles user authentication, CRUD operations for expenses, and real-time data synchronization.

3. Firebase Integration:

- Pyrebase facilitates communication with Firebase services for authentication and database operations.

4. Database:

- Firebase Realtime Database stores all expense data and ensures real-time synchronization.



3.4 Data Modeling in NoSQL

While NoSQL databases like Firebase Realtime Database do not require traditional normalization as relational databases do, it's still essential to design the data model thoughtfully to ensure efficiency and scalability. Here is a structured approach:

Data Modeling Principles:

1. Atomicity:

- Ensure each data entry is atomic, meaning that each record contains all necessary information to avoid partial updates and ensure data consistency.

2. Denormalization:

- Since NoSQL databases are designed to handle denormalized data, related data can be stored together in a single document to optimize read performance.

3. Nested Data:

- Use nested structures to store related information within a single document, which reduces the number of database reads.

Example Data Structure:

User Collection:

```
{
 "user_id": "1",
 "email": "user1@mail.com",
 "password": "hashed_pw1",
 "name": "John Doe",
 "expenses": [
  {
   "expense_id": "1",
   "date": "2023-05-01",
   "category": "Food",
   "amount": 20.50,
   "description": "Lunch at cafe"
  },
  {
   "expense_id": "2",
   "date": "2023-05-02",
   "category": "Transport",
   "amount": 15.00,
   "description": "Bus fare"
  }
 ]
}
```

In this structure:

- Each user document contains an array of expenses.
- Each expense is a nested object within the user document.

- This denormalized approach allows for quick retrieval of user data along with all their expenses in a single read operation.

By following these data modeling principles, the Expense Manager application ensures efficient and reliable data storage, leveraging the strengths of Firebase Realtime Database for real-time synchronization and scalability.

4. Program Code:

```
import sys
from PyQt5.QtWidgets import (QApplication, QMainWindow, QLabel, QLineEdit, QPushButton, QVBoxLayout, QWidget, QListWidget, QListWidgetItem, QMessageBox, QComboBox, QInputDialog, QHBoxLayout, QDateEdit)
from PyQt5.QtCore import QDate
from firebase_config import db, auth
from qt_material import apply_stylesheet

class ExpenseTrackerApp(QMainWindow):
    def __init__(self, user_id):
        super().__init__()
        self.user_id = user_id
        self.setWindowTitle('Expense Tracker')
        self.setGeometry(100, 100, 480, 600)

        self.expense_list = QListWidget()
        self.expense_list.itemClicked.connect(self.load_selected_expense)

        self.expense_label = QLabel('Title:')
        self.expense_input = QLineEdit()

        self.amount_label = QLabel('Amount:')
        self.amount_input = QLineEdit()

        self.date_label = QLabel('Date:')
        self.date_input = QDateEdit()
        self.date_input.setCalendarPopup(True)
```

```
self.date_input.setDate(QDate.currentDate())

self.category_label = QLabel('Category:')
self.category_dropdown = QComboBox()
self.filter_category_dropdown = QComboBox()
self.filter_category_dropdown.addItem("All Categories")
self.filter_category_dropdown.currentIndexChanged.connect(self.filter_expenses)

self.load_categories()

self.add_category_button = QPushButton('Add Category')
self.add_category_button.clicked.connect(self.add_category)

self.add_button = QPushButton('Add Expense')
self.add_button.clicked.connect(self.add_expense)

self.edit_button = QPushButton('Edit Expense')
self.edit_button.clicked.connect(self.edit_expense)
self.edit_button.setEnabled(False)

self.delete_button = QPushButton('Delete Expense')
self.delete_button.clicked.connect(self.delete_expense)
self.delete_button.setEnabled(False)

self.total_label = QLabel('Total Expenses: $0.00')

form_layout = QVBoxLayout()
form_layout.addWidget(self.expense_label)
```

```
form_layout.addWidget(self.expense_input)
form_layout.addWidget(self.amount_label)
form_layout.addWidget(self.amount_input)
form_layout.addWidget(self.date_label)
form_layout.addWidget(self.date_input)
form_layout.addWidget(self.category_label)
form_layout.addWidget(self.category_dropdown)
form_layout.addWidget(self.add_category_button)
form_layout.addWidget(self.add_button)
form_layout.addWidget(self.edit_button)
form_layout.addWidget(self.delete_button)

filter_layout = QHBoxLayout()
filter_layout.addWidget(QLabel("Filter by Category:"))
filter_layout.addWidget(self.filter_category_dropdown)

layout = QVBoxLayout()
layout.addLayout(form_layout)
layout.addLayout(filter_layout)
layout.addWidget(self.expense_list)
layout.addWidget(self.total_label)

container = QWidget()
container.setLayout(layout)
self.setCentralWidget(container)

self.selected_expense_id = None
self.load_expenses()
```

```
def load_categories(self):
    self.category_dropdown.clear()
    categories = db.child(f"users/{self.user_id}/categories").get()
    if categories.each():
        for category in categories.each():
            self.category_dropdown.addItem(category.val())
            self.filter_category_dropdown.addItem(category.val())
    else:
        self.category_dropdown.addItem("Miscellaneous")


def add_category(self):
    category, ok = QInputDialog.getText(self, 'Add Category', 'Category Name:')
    if ok and category:
        db.child(f"users/{self.user_id}/categories").push(category)
        self.load_categories()


def load_expenses(self):
    self.expense_list.clear()
    self.selected_expense_id = None
    self.edit_button.setEnabled(False)
    self.delete_button.setEnabled(False)
    expenses = db.child(f"users/{self.user_id}/expenses").get()
    total_expense = 0.0
    if expenses.each():
        for expense in expenses.each():
            item = expense.val()
            print(item)
```

```
            list_item = QListWidgetItem(f"{item['expense']}: ${item['amount']} [{item['category']}] on {item['date']}")
            list_item.setData(32, expense.key())
            self.expense_list.addItem(list_item)
            total_expense += float(item['amount'])
        else:
            self.expense_list.addItem("No expenses found.")
        self.total_label.setText(f'Total Expenses: ${total_expense:.2f}')

    def update_expense_list(self, expenses):
        self.expense_list.clear()
        self.selected_expense_id = None
        self.edit_button.setEnabled(False)
        self.delete_button.setEnabled(False)
        total_expense = 0.0
        if expenses.each():
            for expense in expenses.each():
                item = expense.val()
                if self.filter_category_dropdown.currentIndex() > 0 and self.filter_category_dropdown.currentText() != item['category']:
                    continue
                list_item = QListWidgetItem(f"{item['expense']}: ${item['amount']} [{item['category']}] on {item['date']}")
                list_item.setData(32, expense.key())
                self.expense_list.addItem(list_item)
                total_expense += float(item['amount'])
        if self.expense_list.count() == 0:
            self.expense_list.addItem("No expenses found.")
        self.total_label.setText(f'Total Expenses: ${total_expense:.2f}')
```

```
    def filter_expenses(self):
        expenses = db.child(f"users/{self.user_id}/expenses").order_by_child("amount").get()
        self.update_expense_list(expenses)

    def add_expense(self):
        expense = self.expense_input.text()
        amount = self.amount_input.text()
        date = self.date_input.date().toString("yyyy-MM-dd")
        category = self.category_dropdown.currentText()
        if expense and amount and category:
            try:
                amount = float(amount)
                db.child(f"users/{self.user_id}/expenses").push({"expense": expense, "amount": amount,
"date": date, "category": category})
                self.expense_input.clear()
                self.amount_input.clear()
                self.load_expenses()
            except ValueError:
                QMessageBox.warning(self, 'Invalid Input', 'Amount must be a number.')
        else:
            QMessageBox.warning(self, 'Input Error', 'All fields are required.')

    def load_selected_expense(self, item):
        if item.text() == "No expenses found.":
            return
        expense_text = item.text()
        expense_id = item.data(32)
```

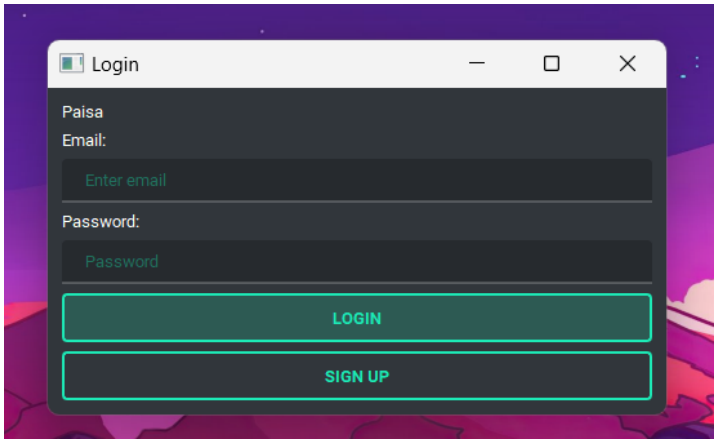
```
        if expense_id:
            self.selected_expense_id = expense_id
            parts = expense_text.split(': $')
            print(parts)
            expense = parts[0]
            amount, date = parts[1].split(' on ')
            amount, category = amount.split('[')
            category = category.replace(']', '')
            self.expense_input.setText(expense)
            self.amount_input.setText(amount)
            self.date_input.setDate(QDate.fromString(date, "yyyy-MM-dd"))
            self.category_dropdown.setCurrentText(category)
            self.edit_button.setEnabled(True)
            self.delete_button.setEnabled(True)
        else:
            QMessageBox.warning(self, 'Format Error', 'The selected item has an unexpected format.')

    def edit_expense(self):
        if self.selected_expense_id:
            expense = self.expense_input.text()
            amount = self.amount_input.text()
            date = self.date_input.date().toString("yyyy-MM-dd")
            category = self.category_dropdown.currentText()
            if expense and amount:
                try:
                    amount = float(amount)

db.child(f"users/{self.user_id}/expenses").child(self.selected_expense_id).update({"expense":
expense, "amount": amount, "date": date, "category": category})
```

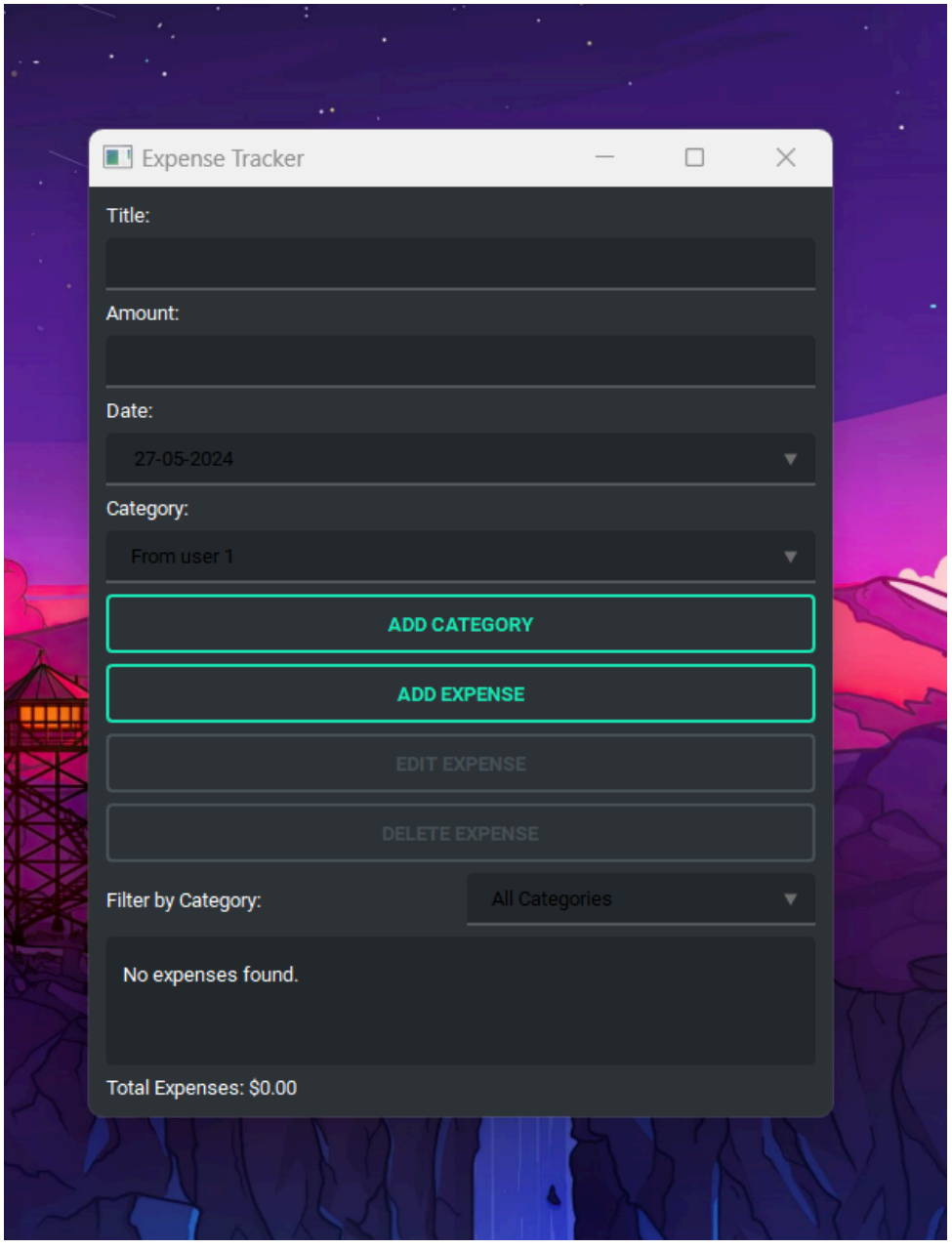
```
                self.load_expenses()
            except ValueError:
                QMessageBox.warning(self, 'Invalid Input', 'Amount must be a number.')
        else:
            QMessageBox.warning(self, 'Input Error', 'All fields are required.')

    def delete_expense(self):
        if self.selected_expense_id:
            db.child(f"users/{self.user_id}/expenses").child(self.selected_expense_id).remove()
            self.expense_input.clear()
            self.amount_input.clear()
            self.load_expenses()

class LoginWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Login')
        self.setGeometry(100, 100, 480, 200)

        self.app_name_label = QLabel('Paisa', self)

        self.email_label = QLabel('Email:', self)
        self.email_input = QLineEdit(self)
        self.email_input.setPlaceholderText('Enter email')

        self.password_label = QLabel('Password:', self)
        self.password_input = QLineEdit(self)
        self.password_input.setPlaceholderText('Password')
```

```
        self.password_input.setEchoMode(QLineEdit.Password)

        self.login_button = QPushButton('Login', self)
        self.login_button.clicked.connect(self.login)

        self.signup_button = QPushButton('Sign Up', self)
        self.signup_button.clicked.connect(self.signup)

        layout = QVBoxLayout()
        layout.addWidget(self.app_name_label)
        layout.addWidget(self.email_label)
        layout.addWidget(self.email_input)
        layout.addWidget(self.password_label)
        layout.addWidget(self.password_input)
        layout.addWidget(self.login_button)
        layout.addWidget(self.signup_button)

        container = QWidget()
        container.setLayout(layout)
        self.setCentralWidget(container)

    def login(self):
        email = self.email_input.text()
        password = self.password_input.text()

        try:
            user = auth.sign_in_with_email_and_password(email, password)
            QMessageBox.information(self, 'Login Successful', f'Welcome {email}!')
```

```
            user_id = user['localId']
            self.redirect_to_main_app(user_id)
        except Exception as e:
            QMessageBox.warning(self, 'Login Failed', str(e))

    def signup(self):
        email = self.email_input.text()
        password = self.password_input.text()

        try:
            auth.create_user_with_email_and_password(email, password)
            QMessageBox.information(self, 'Sign Up Successful', f'Account created for {email}!')
            self.redirect_to_main_app()
        except Exception as e:
            QMessageBox.warning(self, 'Sign Up Failed', str(e))

    def redirect_to_main_app(self, user_id):
        self.main_app_window = ExpenseTrackerApp(user_id)
        self.main_app_window.show()
        self.close()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    apply_stylesheet(app, theme='dark_teal.xml')
    window = LoginWindow()
    window.show()
    sys.exit(app.exec_())
```

Output:


Login
Paisa
Email:
Enter email
Password:
Password
LOGIN
SIGN UP


Expense Tracker
Title:
Amount:
Date:
27-05-2024
Category:
From user 1
ADD CATEGORY
ADD EXPENSE
EDIT EXPENSE
DELETE EXPENSE
Filter by Category:
All Categories
No expenses found.
Total Expenses: $0.00

5. Result and Discussion:

The Expense Manager application delivers a robust and user-friendly experience for tracking and managing personal finances. With an intuitive interface built using PyQt5, users can easily add, edit, and view expenses. Real-time synchronization via Firebase Realtime Database ensures seamless data updates across devices, even supporting offline access. Data visualization features using Matplotlib or PyQtGraph enable users to analyze spending patterns effectively. Overall, the application meets its objectives of providing a reliable, efficient, and insightful tool for personal expense management.

6. Conclusion:

The Expense Manager application successfully meets its goal of providing users with an efficient and intuitive tool for managing personal finances. By leveraging Python and PyQt5 for the user interface, and Firebase Realtime Database for real-time data synchronization and storage, the application ensures a seamless and responsive user experience. The integration of data visualization tools allows users to gain valuable insights into their spending habits, promoting better financial management. Overall, the application stands out as a reliable, scalable, and user-friendly solution for personal expense tracking.

7. References:

1. https://www.pythonguis.com/pyqt5-tutorial/
2. https://github.com/thisbejim/Pyrebase
3. https://github.com/aangelone2/sem-qt
4. https://firebase.google.com/docs/web/setup?authuser=0&hl=en