Course Code/Course Title: ICT 2140 / Introduction to Software Engineering
Group Number: Group14
Project Topic: Finance Tracking App
Link to Github Repository:
https://github.com/Blackysynch/Finance_Tracking_App
Group Leader: Penchia Mediesse Ornelle Davilla

SN	Members	Registration No	Team Role	%participation
1	PENCHIA MEDIESSE ORNELLE DAVILLA	ICTU20234123	Team Leader/ Frontend / Backend Developer / Database designer	30%
2	KENMOGNE MATCHUEKAM SERGINE BARTHE BRES	ICTU20233772	Frontend / Backend Developer	15%
3	ROSEMARY LECHANGWA MBENOH	ICTU20234079	Frontend / Backend Developer	10%
4	PRECIOUS MALOBA	ICTU20233724	Frontend / Backend Developer	10%
5	NKOUH THELMA IKFUI	ICTU20234006	Frontend / Backend Developer	15%
6	TAKEM BRIGHTON	ICTU20233929	Frontend / Backend Developer	10%
7	MBIYDZENYUY WENDY BONGISI	ICTU20233816	Frontend / Backend Developer	10%
8	NKINYAM PRAISES NCHA	ICTU20233776	Frontend / Backend Developer	15%

CHAPTER ONE:

INTRODUCTION

General Introduction

The finance tracking app, Money Pilot, was developed with the primary goal of assisting users in efficiently monitoring and managing their personal financial activities. In today's digital world, keeping track of income, expenses, and savings can be overwhelming for many individuals. Therefore, this app serves as a comprehensive tool that enables users to track and categorise their spending, set budgets, and view detailed reports on their financial health. The platform is designed to be intuitive, enabling users to make informed financial decisions with ease, even without a deep understanding of personal finance. Moreover, the app integrates data visualisation tools to provide clear insights into spending patterns over time.

Aim and Objectives

- ➢ Aim:
  - Providing an efficient and user-friendly platform that simplifies personal financial management.

- ➢ Objectives:
  - Enable real-time tracking of income, expenses, and savings.
  - Offer visualisation tools for easy understanding of financial trends.
  - Support budget management and spending categorization.
  - Ensure data security to maintain user trust.
  - Provide customization options for personalised experiences.

Problem Statement

Managing personal finances can be a daunting task for many individuals. The complexity associated with budgeting, tracking expenses, and staying organised often leads to overspending or failure to save effectively. While several existing tools aim to address these issues, many are either too complicated to use or lack essential features such as proper categorization, reporting, and user-friendly interfaces. This project addresses the gap by developing an app that not only provides a seamless experience but also integrates critical features like budget setting, income/expense tracking, and financial visualisation, all in a user-friendly manner

CHAPTER TWO:
LITERATURE REVIEW

Review of Concepts

Web App Development: This is the creation of application programs which reside on remote servers and are delivered to the user's device over the internet without the need of being downloaded. It involves two major processes: Frontend **and Backend Development.**

Frontend Development: This is the development of visual and interactive elements of a website that users interact directly with. The main languages involved are **HTML, CSS, and Javascript**.

- **HTML**: HTML stands for Hyper Text Markup Language and is used on the frontend and gives the structure of the webpage.
- **CSS**: CSS stands for Cascading Style Sheet is the language used to style the frontend of any website.
- **Javascript** : Allows the addition of interactivity to web pages

Backend Development: This refers to the server-side aspect of web development, focusing on creating and managing the server logic, databases, and APIs. It involves handling user authentication, authorization, and processing user requests, typically using backend development languages such as Python, Java, Ruby, PHP, JavaScript.

- **Database**: It is a structured collection of data organised in a way that allows efficient retrieval and modification of the data. It is typically stored electronically, and can be accessed by multiple users
- **API**: Standing for Application programming Interface is a set of rules and protocols that allow different software applications to communicate with each other
- **Server**: It is a computer program or device that provides a service to another computer program and its user, also known as the client

Version Control: It is the software engineering practice of controlling computer files and versions of files; primarily source code text files, but generally any type of file. It involves the use of version control systems like **git , csv, mercurial**.

Software Development Methodologies

Software development methodology is defined as a set of principles and techniques used to guide the entire software development life cycle toward the successful completion of projects. It benefits both teams and customers by improving efficiency and adaptability to changes.

Types of Software Development Methodologies

The various types of software development methodologies include:

Waterfall model:
It is a Linear approach with sequential flow of development phases like; planning, design, development, testing, deployment, and maintenance. It is most suitable for projects with well-defined requirements and stable environments.

Iterative and incremental model:
It is a software development methodology that emphasises continuous improvement through repeated cycles of development and testing. This approach is often used in situations where requirements may evolve or change over time.

Spiral method:
This is a risk-driven software development methodology that combines elements of the Waterfall Model and iterative development. It emphasises risk management and flexibility to accommodate changing requirements.

V-Model:
an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage or every single phase in the development cycle, there is a directly associated testing phase

Agile Model:
It is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.The methodology is well-suited for projects with uncertain requirements, dynamic environments, and a need for flexibility.

Method Used:

For this project the method chosen was a combination of waterfall and iterative and incremental model. For the following reasons

Well-Defined Requirements: The Waterfall model is particularly effective for projects with clearly defined requirements. In the case of Money Pilot, the project's objectives and features were well-defined from the outset, including income and expense tracking, budget management, and financial reporting.

Flexibility: Agile allows for adaptability to changing requirements and unexpected challenges. This is important knowing the group members were unfamiliar with the different technologies.

Risk Mitigation**:** Waterfall can help identify and address risks upfront, while Agile enables continuous risk management.

Improved Collaboration: Combining the strengths of both methodologies can foster better collaboration between teams and stakeholders.

Predictable Timelines and Milestones: The Waterfall model's sequential nature allows for predictable timelines and milestone tracking. Given the clear and fixed project requirements, the Waterfall methodology provided a straightforward framework for scheduling development phases and meeting deadlines.

Stable Requirements: The Waterfall model is best suited for projects with stable and well-understood requirements. Since the core functionalities of Money Pilot—such as tracking income, managing expenses, and generating financial reports—were well-defined and unlikely to change significantly during development, the Waterfall approach provided a stable framework for delivering the application without frequent changes to the project scope.

Review of Related Literature:

Existing Finance Tracking Applications

Many personal finance apps exist, such as Spending Tracker, YNAB, PocketGuard and so on.
While they offer valuable features, they may have limitations in customization, integration, or advanced analytics But Money Pilot stands out by offering customization, advanced analytics, a user-friendly interface, and strong security measures.

CHAPTER THREE:
METHODOLOGY AND MATERIALS

Research Methodology:

The research methodology for this code involves a combination of data analysis, user input handling, and data visualisation. The following steps outline the research methodology:

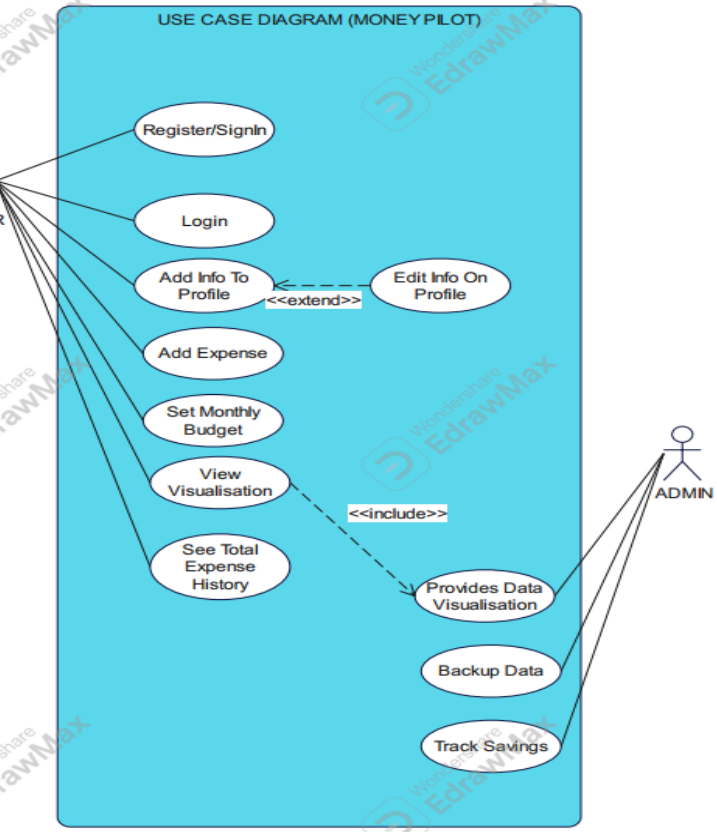
1. Data Collection: The system collects data from user inputs, such as username, password, name, email, telephone, location, country, budget, income sources, and expense details. The system also collects data from the database, such as user details and expense records.
2. Data Validation: The system validates user inputs to ensure they meet specific criteria, such as unique usernames and emails, positive budget amounts, and valid expense amounts. The system also validates data from the database to ensure accuracy and consistency.
3. Data Processing: The system processes user inputs and database data to perform various functions, such as user authentication, profile creation and updates, expense tracking, and data visualisation.
4. Data Analysis: The system analyses expense data to provide insights into user spending habits. The analysis includes calculating total expenses, categorising expenses, and visualising expense trends over time.
5. Data Visualization: The system visualises expense data using various charts, such as pie charts, bar charts, and line charts. The visualisations help users understand their spending habits and identify areas for improvement.
6. Debugging and Testing: The system undergoes rigorous testing and debugging to ensure it works as expected. The testing includes unit testing, system testing, and integration testing.

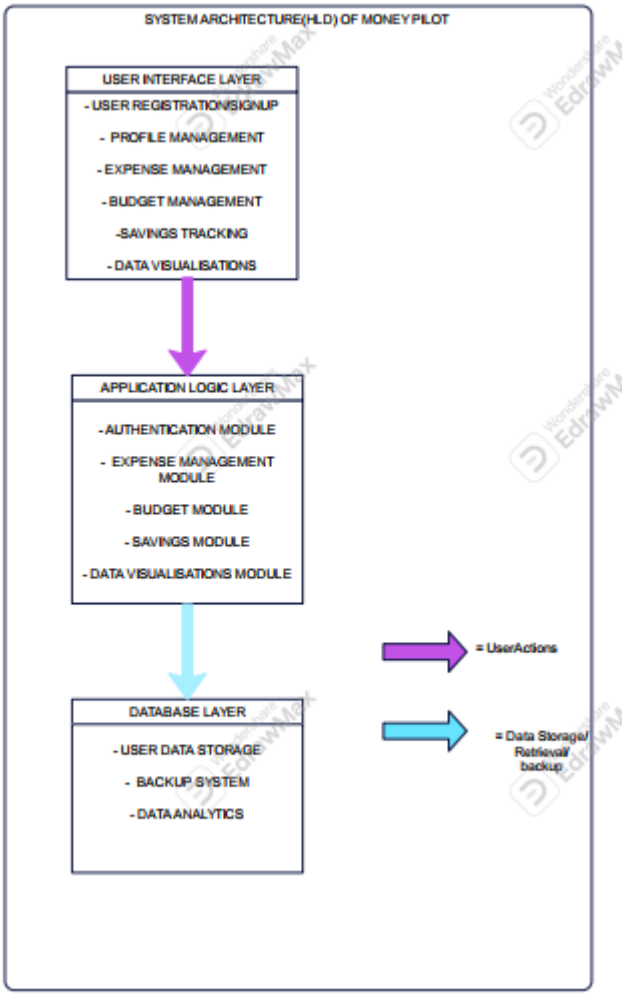
System Requirements:*

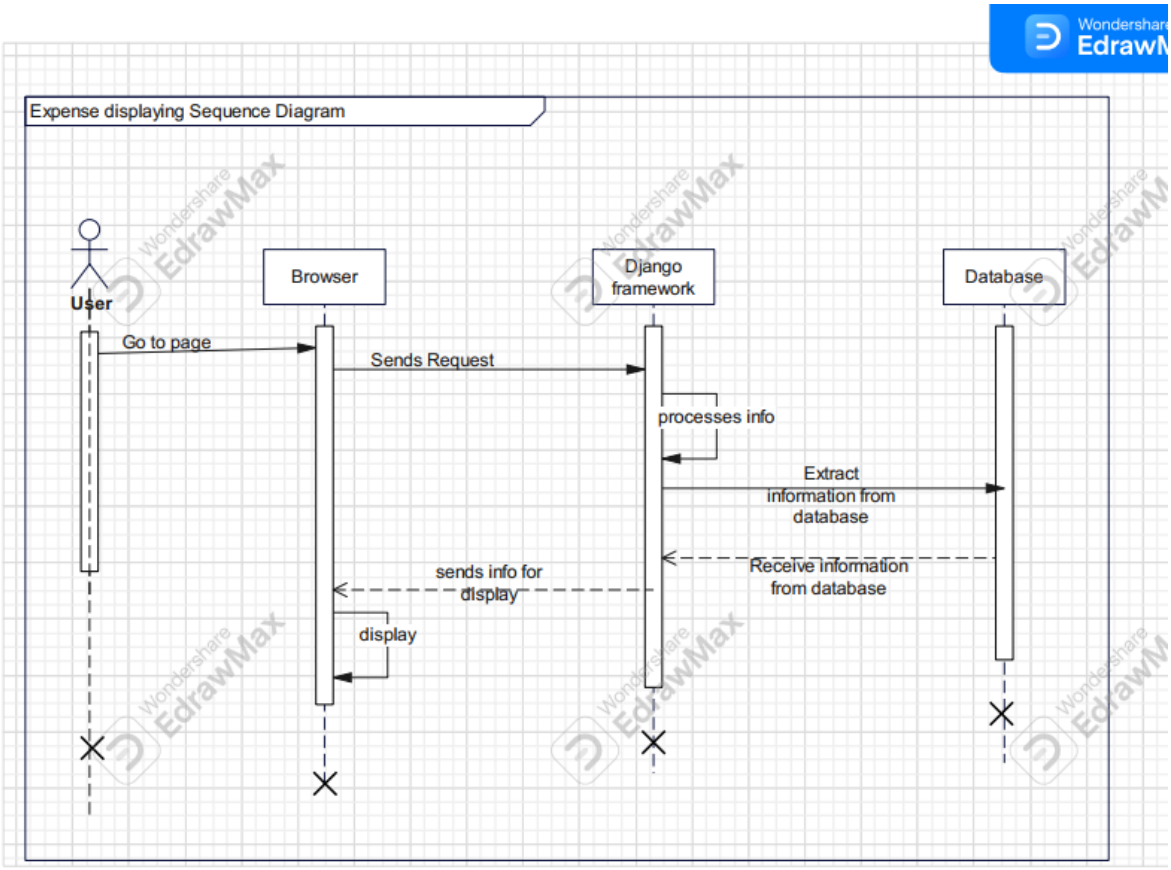
HTML
CSS

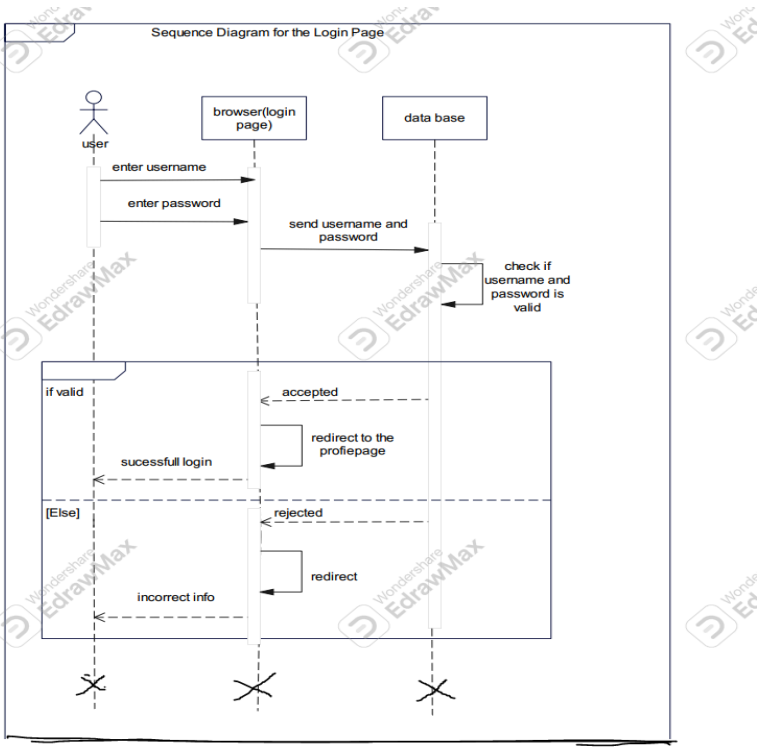
Javascript

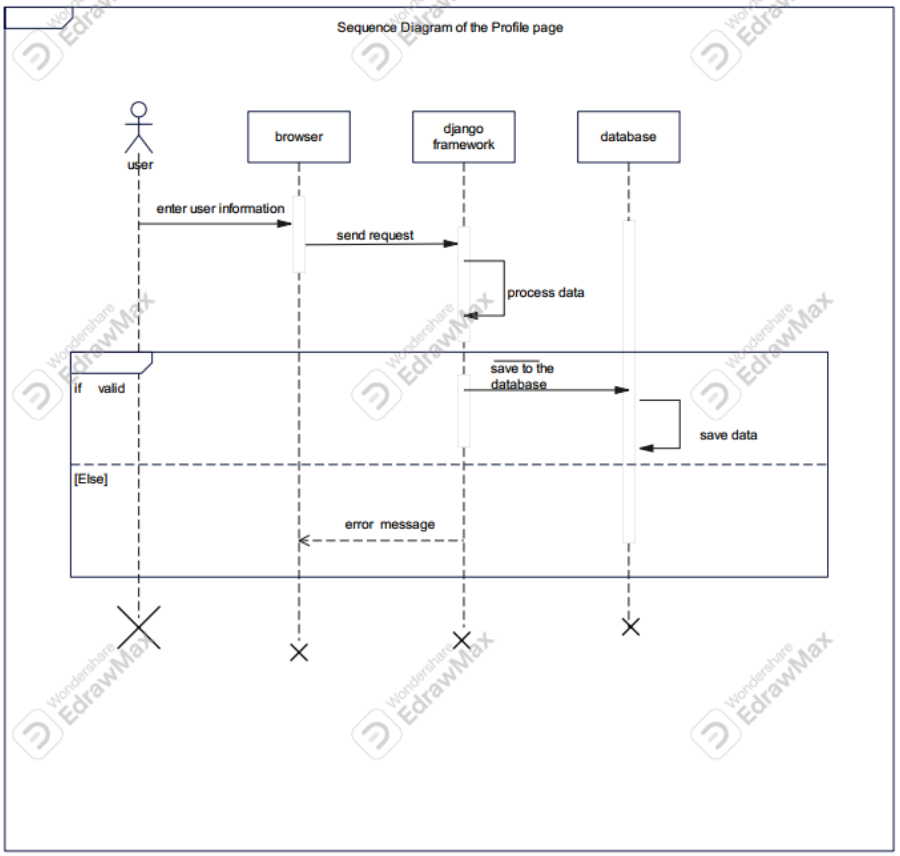
Python
SQLite**System Design:**

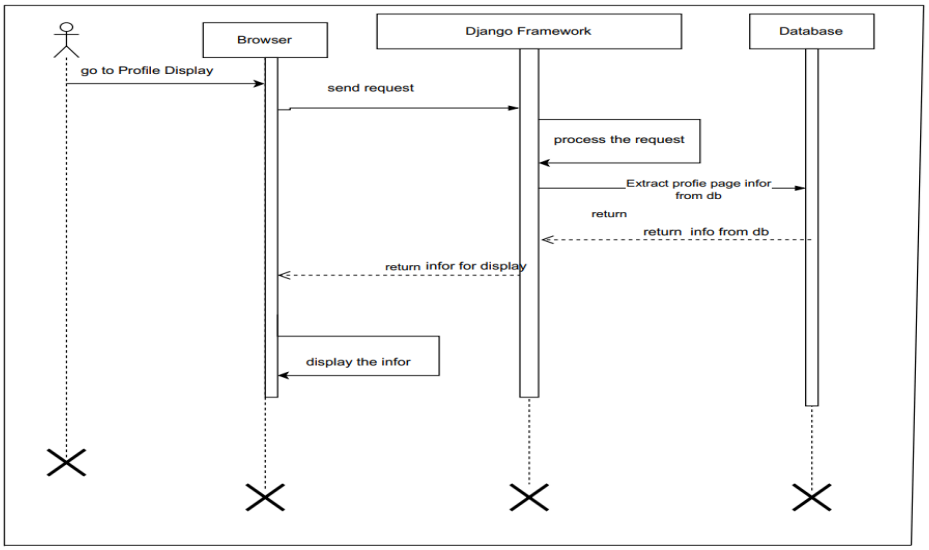


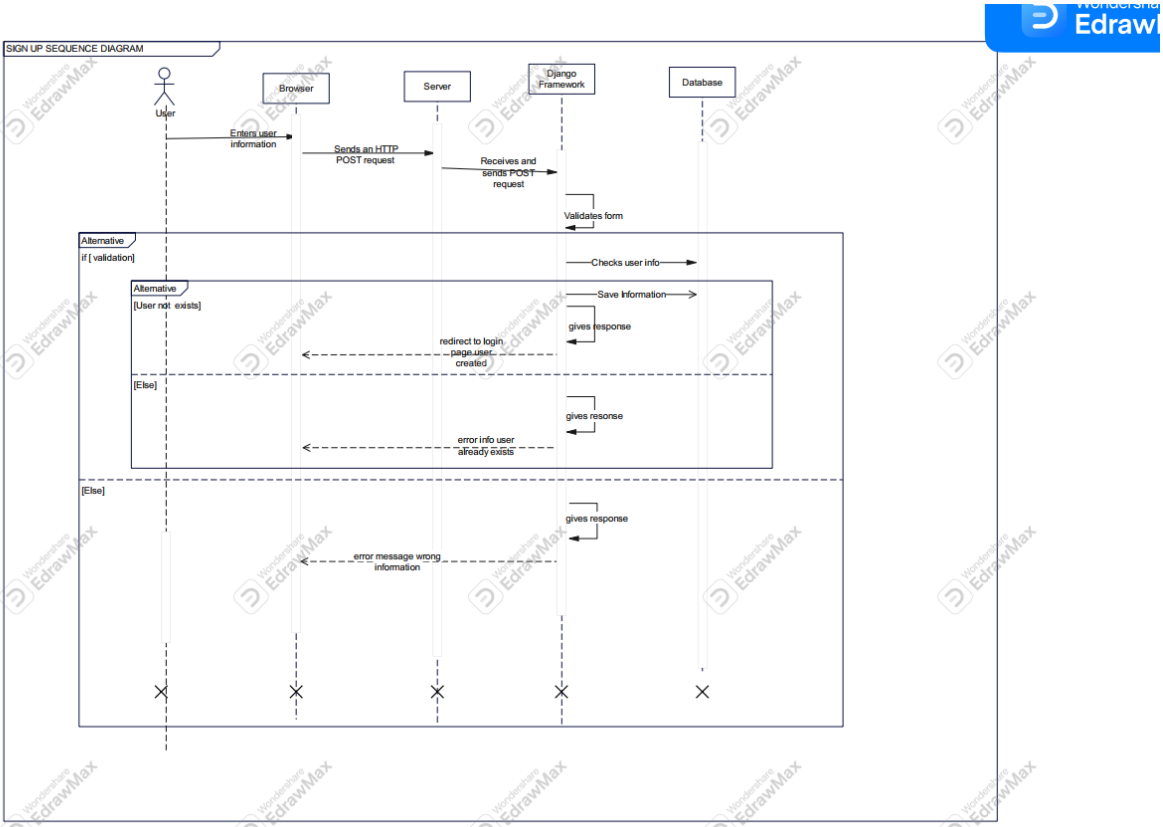



Expense displaying Sequence Diagram
User
Browser
Django framework
Database
Go to page
Sends Request
processes info
Extract information from database
Receive information from database
sends info for display
display


Sequence Diagram for the Login Page
user
browser(login page)
data base
enter username
enter password
send username and password
check if username and password is valid
if valid
accepted
redirect to the profiepage
sucessfull login
[Else]
rejected
redirect
incorrect info


Sequence Diagram of the Profile page
user
browser
django framework
database
enter user information
send request
process data
if valid
save to the database
save data
[Else]
error message


Browser
Django Framework
Database
go to Profile Display
send request
process the request
Extract profie page infor from db
return
return info from db
return infor for display
display the infor



Application of chosen methodology:

- For this project the methodology used was the incremental and iterative model with some elements of the agile model. The initial design for the Web app was first created to get a clearly defined of the final projects
- The project is divided into smaller, manageable tasks(the frontend pages, backend functionalities. Extra functionalities as needed).Each page was developed and completed independently before moving on to the next. This allowed for a more flexible and adaptable approach, as changes could be made to one feature without affecting the others.

Requirements specification:

Functional Requirements

- User Registration and Login:
    - Allow users to create accounts and log in securely.
    - Implement password recovery and reset mechanisms.
- Expense Entry:
    - Enable users to input expenses, including:
        - Date
        - Amount
        - Category (e.g., food, transportation, entertainment)

        - Notes or descriptions
    - Support for multiple currencies.
    - Option to attach receipts or photos.
- Expense Categorization:
    - Provide customizable expense categories and subcategories.
    - Allow users to create custom categories.
- Budget Tracking:
    - Set monthly or yearly budgets for specific categories.
    - Track spending against budgets and provide visual representations (e.g., charts).
- Reporting and Analysis:
    - Generate detailed reports on spending patterns, including:
        - Total expenses by category
        - Spending trends over time
        - Comparison to budget
    - Allow users to export reports in various formats (e.g., CSV, PDF).
- Data Synchronization:
    - Enable users to sync data across multiple devices (e.g., smartphones, tablets, computers).

Non-Functional Requirements

- User Interface:
    - Design an intuitive and user-friendly interface.
    - Ensure the app is responsive and works well on different screen sizes.
    - Provide clear and concise instructions.
- Security:
    - Implement robust security measures to protect user data, including:
        - Data encryption
        - Strong password policies
        - Secure authentication mechanisms

Proposed Algorithms:

Start
Get user expenses

Process expenses:
  Extract total expenses, categories, dates, and amounts
  Convert total expenses to floats
  Create dictionary for expenses by year and month
Prepare chart data:
  Pie chart: total expenses and categories
  Line chart: dates and amounts
  Resume chart: expenses by year and month
Render charts:
  Pie chart, bar chart, line chart, and resume chart

Render template:analysispage.html with chart data
stop

Materials and technologies:

For the realisation of our project the following materials and technologies where used

- ➔ HTML: (Hyper Text Markup Language) as backbone of web pages.
- ➔ CSS: (CASCADING STYLE SHEETS) for styling the webpages.
- ➔ Javascript: for interactive functionalities.
- ➔ Django: backend framework used for the server side implementation of the application.
- ➔ Chartjs: simple javascript library for graph creations for the data visualisation
- ➔ Pythonanywhere: hosting service for the code supports django application and dynamic web app hosting.
- ➔ Git: version control tool to manage
- ➔ Github: online platform for software development collaboration and version control. It's a cloud-based service that provides a repository for storing and managing source code.
- ➔ Google: Used for learning material collection and learning of the different materials and technologies used.

Testing:

We have thoroughly reviewed the provided Django code, which consists of multiple views, templates, and models. Our review approach involved a systematic and incremental evaluation of each component to ensure their individual functionality and integration with the entire web application.

Individual View Review

We started by testing each view individually, verifying that they rendered correctly and performed their intended functions. This included:

- landingPage: Successfully rendered the landing page template.
- loginPage: Tested login functionality with valid and invalid credentials, ensuring correct error messages and redirects.
- signUpPage: Verified that new users could be created, and existing usernames and emails were correctly validated.
- profilePage: Tested profile updates, including validation of required fields and correct error messages.

- addExpense: Successfully added expenses with valid and invalid data, ensuring correct error messages and redirects.
- expenseHistory: Verified that expense data was correctly displayed and filtered by user.
- analysisPage: Tested the rendering of charts and graphs, ensuring that data was correctly aggregated and displayed.

Integration Review

After verifying individual view functionality, we tested each view in conjunction with others to ensure seamless integration. This included:

- login and signUp page integration: Tested that successful sign-up redirected to the login page, and successful login redirected to the profile page.
- profile and addExpense page integration: Verified that profile updates were reflected on the add expense page, and expense additions were correctly linked to the user's profile.
- expenseHistory and analysis page integration: Tested that expense data was correctly displayed on both pages, and filtering on the expense history page affected the analysis page charts.

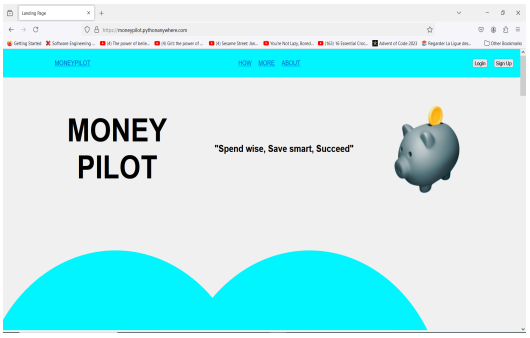
Whole Web App Review

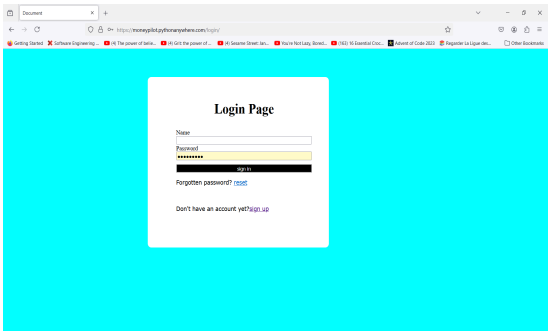
Finally, we reviewed the entire web application together, simulating user interactions and verifying that all components worked in harmony. This included:

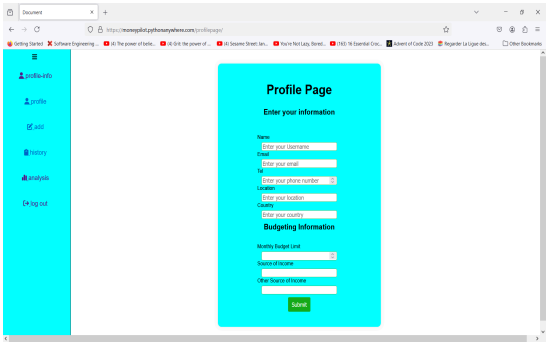
- Testing the entire user workflow, from sign-up to expense addition and analysis.
- Verifying that authentication and authorization mechanisms were correctly implemented, restricting access to authorized users.
- Ensuring that error messages and redirects were correctly handled throughout the application.

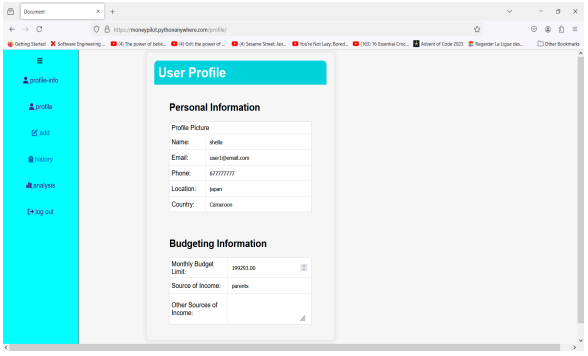
CHAPTER FOUR:

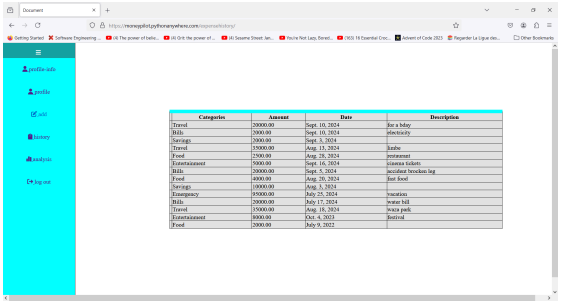
RESULTS AND DISCUSSION

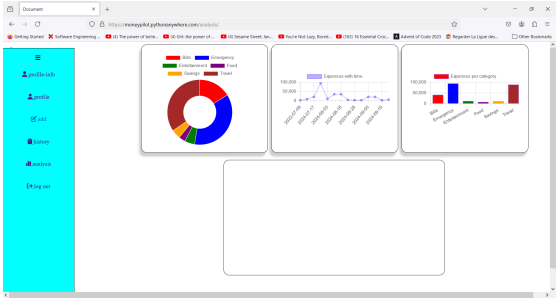


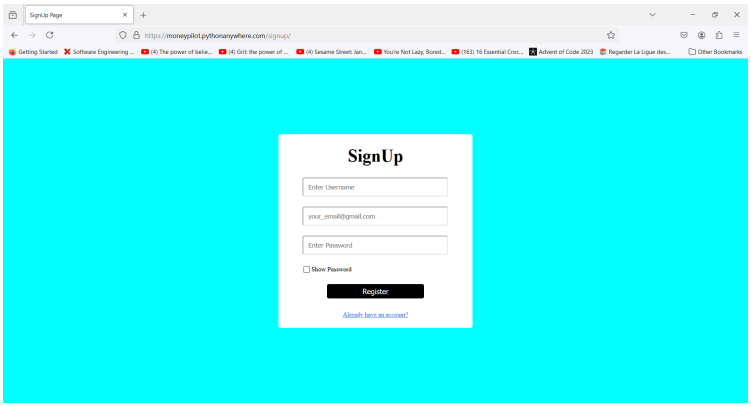












Api and Code

Analysis view

```python
@login_required
def analysisPage(request):
    user = request.user.id


    #for 1st pie chart and bar chart
    user_expenses = Expense.objects.filter(user=user).order_by('date')
    user_expenses_per_category = Expense.objects.filter(user=user).values('category').annotate(total_exp=Sum('amount'))

    total_expenses = [float(expense['total_exp'])  for expense in user_expenses_per_category]
    categories = [expense['category'] for expense in user_expenses_per_category]

    #print (f"{total_expenses} \n'{'dates'}' \n{categories}")



    #for the line chart
    # getdates and amounts from expenses
    dates = [str(expense.date) for expense in user_expenses]
    amounts = [float(expense.amount) for expense in user_expenses]


    # for resume chart
    #months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December']

    expenses_by_year_and_month = list(Expense.objects.annotate(
        month=ExtractMonth(TruncMonth('date')),
        expense_year=ExtractYear(TruncYear('date'))
    ).values('month', 'expense_year').annotate(total_exp=Sum('amount')).order_by('expense_year', 'month'))
```

```python
years = list(range(min_year, max_year + 1))
#trial
expenses_by_y_m = list(Expense.objects.annotate(
    month=ExtractMonth(TruncMonth('date')),
    expense_year=ExtractYear(TruncYear('date'))
).values('month', 'expense_year').annotate(total_exp=Sum('amount')).order_by('expense_year', 'month'))



# Convert the total_exp to float
for expense in expenses_by_year_and_month:
    expense['total_exp'] = float(expense['total_exp'])

for expense in expenses_by_year_and_month:
    print(expense)

#print(f" max: {max_year} min: {min_year} \n expyearmonth: {expenses_by_year_and_month}")
#create

expenses_by_year = {}

# iterate over the expenses
for expense in expenses_by_year_and_month:
    year = expense['expense_year']
    month = expense['month']
    total_exp = expense['total_exp']

# if the year is not already in the dictionary, add it
if year not in expenses_by_year:
    expenses_by_year[year] = {}

# add the month and total expense to the year's dictionary
expenses_by_year[year][month] = total_exp


return render(request, 'analysispage.html', { 'total_expenses': total_expenses,'categories': categories,  'dates': dates, 'amounts': amounts, 'max_year': max_year, 'min_year': min_y
```

Add Expense view

```python
@login_required
def addExpense(request):
    if request.method == 'POST':
        # Retrieve data from the POST request
        user = request.user.id # Assumes user is authenticated
        date = request.POST.get('date')
        category = request.POST.get('category')
        amount = request.POST.get('amount')
        description = request.POST.get('description')
        try:
            amount = float(amount)
            if amount <= 0:
                messages.error('Amount must be a positive number.')
        except ValueError:
            messages.error('Invalid amount format.')


        if not date:
            messages.error(request, 'Date is required.')
        if not category:
            messages.error(request, 'Category is required.')
        if not amount:
            messages.error(request,'Amount is required.')

        try:
            amount = float(amount)
            if amount <= 0:
                messages.error('Amount must be a positive number.')
        except ValueError:
            messages.error('Invalid amount format.')

        if description and len(description) > 255:
            messages.error('Description cannot exceed 255 characters.')


        # Save the data to the Expense model
        Expense.objects.create(
            user_id=request.user.id,
            date=date,
            category=category,
            amount=amount,
            description=description
        )

        return redirect('add-expense')  # Redirect to the same page after adding the expense
```

Login and landingpage

```python
def landingPage(request):
    return render(request, 'landingpage.html')



def loginPage(request):
    if request.method == 'POST':
        username = request.POST.get('Name')
        password = request.POST.get('password')


        print(f" {username}, {password}")

        if not username or not password:
            messages.error(request, 'username and password are required')
            return render(request, 'login.html')

        try:
            user = User.objects.get(username=username)
        except User.DoesNotExist:
            messages.error(request, 'Invalid username or password')
            return render(request, 'login.html')

        user = authenticate(request, username=username, password=password)
        if user is not None:
            if user.is_active:

                login(request, user)
                messages.success(request, 'You have successfully logged in')
                return redirect('profile-page')
            else:
                messages.error(request, 'Account is inactive')
                return render(request, 'login.html')
        else:

            print(f"User is {user} does not exist")
            messages.error(request, 'Invalid username or password')
            return render(request, 'login.html')

    return render(request,'login.html')
```

Expense history

```python
@login_required
def expenseHistory(request):
    username = request.user.username
    user_expenses = Expense.objects.filter(user=request.user)

    # Print expense details
    #for expense in user_expenses:
        #print(f"Date: {expense.date}, Category: {expense.category}, Amount: {expense.amount}, Description: {expense.description}")

    return render(request, 'expensetable.html', {'expenses': user_expenses})
```

Profile display

```python
@login_required
def profileDisplay(request):
    user_detail = UserDetail.objects.get(user_id=request.user.id)
    return render(request, 'profiledisplay.html', {'user_detail': user_detail})
```

Analysis.html code

```
-->
{% include 'nav.html' %}
<script src="https://kit.fontawesome.com/ee9f777706.js" crossorigin="anonymous"></script>
<script>
    document.addEventListener("DOMContentLoaded", function (event) {

        /*pie chart*/
        const pieCtx = document.getElementById('piechart').getContext('2d');
        const pieChart = new Chart(pieCtx, {
            type: "doughnut",
            data: {
                labels: {{ categories | SafeArray }} ,
                datasets: [{
                    label: 'expenses',
                    data: {{ total_expenses | SafeArray}},
                    backgroundColor: ['red', 'blue', 'green', 'purple', 'orange', 'brown'],
                    borderWidth: 1
                }],
                options: {
                    title: {
                    display: true,
                    text: 'Top 10 Expenses'
                },
                legend: {
                    position: 'right',
                    align: 'top',
                    labels: {
                        boxWidth: 10,
                        fontSize: 12,
                        fontColor: '#666'
                    }
                }
            }
        });

        /*line chart*/
        const lineCtx = document.getElementById('linechart').getContext('2d');
        const lineChart = new Chart(lineCtx, {
            type: "line",
            data: {
                labels: {{ dates | SafeArray }} ,
                datasets: [{
                    label: 'Expenses with time',
```

```
144         /*bar chart*/
145         const barCtx = document.getElementById('barchart').getContext('2d');
146         const barChart = new Chart(barCtx, {
147             type: "bar",
148             data: {
149                 labels: {{ categories | SafeArray }} ,
150                 datasets: [{
151                     label: 'Expenses per category',
152                     data: {{ total_expenses | SafeArray}},
153                     backgroundColor: ['red', 'blue', 'green', 'purple', 'orange', 'brown'],
154                     borderColor: 'rgba(86, 0, 255, 0.5)',
155                     borderWidth: 1
156                 }],
157                 options: {
158                     title: {
159                         display: true,
160                         text: 'Total Expenses per Category'
161                     },
162                     legend: {
163                         position: 'right',
164                         align: 'top',
165                         labels: {
166                             boxWidth: 10,
167                             fontSize: 12,
168                             fontColor: '#666'
169                         }
170                     },
171                     scales: {
172                         y: {
173                             beginAtZero: false,
174                             stepSize: 10000
175                         }
176                     }
177                 }
178             }
179         });
180
```

```
/*bar chart*/
const barCtx = document.getElementById('barchart').getContext('2d');
const barChart = new Chart(barCtx, {
    type: "bar",
    data: {
        labels: {{ categories | SafeArray }} ,
        datasets: [{
            label: 'Expenses per category',
            data: {{ total_expenses | SafeArray}},
            backgroundColor: ['red', 'blue', 'green', 'purple', 'orange', 'brown'],
            borderColor: 'rgba(86, 0, 255, 0.5)',
            borderWidth: 1
        }],
        options: {
            title: {
                display: true,
                text: 'Total Expenses per Category'
            },
            legend: {
                position: 'right',
                align: 'top',
                labels: {
                    boxWidth: 10,
                    fontSize: 12,
                    fontColor: '#666'
                }
            },
            scales: {
                y: {
                    beginAtZero: false,
                    stepSize: 10000
                }
            }
        }
    }
});
```

CHAPTER FIVE: RECOMMENDATIONS AND CONCLUSION

Achievements:

Our team has successfully developed a robust web application that effectively addresses the identified user needs. We have successfully integrated various technologies, including user authentication, password encryption, data analysis and visualisation through the different graphs, to create a comprehensive and user-friendly platform. The application's performance is optimized for desktop and mobile devices, ensuring a seamless user experience.

Difficulties Encountered:

One of the primary challenges we faced was integrating the diverse technologies and ensuring smooth compatibility between different components of the application. Especially due to the unfamiliarity of most members to the different technologies.

The greatest conflict encountered by the tea had to do with the app design.
Resolving conflicts and optimizing performance required extensive testing and debugging. Another difficulty arose from the evolving nature of the project requirements. We had to adapt our plans and designs to accommodate changes, which sometimes led to delays and increased complexity.

Recommendations for Further Studies:

To further enhance the application and address potential future challenges, we recommend the following areas for further study:

- **Advanced security techniques:** Explore more sophisticated security measures to protect against emerging threats and ensure data privacy.
- **Performance optimization:** Investigate techniques to further improve the application's speed and responsiveness, especially for mobile devices.
- **Advanced Data visualistions techniques:** The implementation of more advanced graphics while remaining user friendly.
- Flexibility: The possiblity of bettering customer experience with more customisable categories. And user account customisation.
- **Scalability:** Study methods to ensure the application can handle increased user loads and data volumes in the future. Such as transferring to databases such as MySQL.
- **Accessibility:** Research ways to make the application more accessible to users with disabilities, improving inclusivity.