SRI KRISHNA COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

Approved by AICTE | Affiliated to Anna University Chennai|

Accredited by NBA - AICTE| Accredited by NAAC with 'A' Grade

KOVAIPUDUR, COIMBATORE 641042

PROJECT NAME

BUDGET BUDDY-AN EXPENSE TRACKER

A PROJECT REPORT

Submitted by

NIVEDHITHA J	-	**727822TUIT142**
NITHYASHRI DD	-	**727822TUIT141**
PAGALAVAN M	-	**727822TUIT143**
PALANIVEL RAJAN C	-	**727822TUIT144**

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

DEPARTMENT OF INFORMATION TECHNOLOGY

OCT/NOV 2024

BONAFIDE CERTIFICATE

Certified that this project report **BUDGET BUDDY** is the bonafide work of **NITHYASHRI DD 727822TUIT141, NIVEDHITHA J 727822TUIT142, PAGALAVAN M 727822TUIT143, PALANIVEL RAJAN C 727822TUIT144** who carried out the project work under my supervision.

SIGNATURE	*SIGNATURE*
Ms. P. ALAGUVATHANA	**Dr. T. RAJESH KUMAR**
SUPERVISOR	**HEAD OF THE DEPARTMENT**
Assistant Professor,	Associate Professor,
Department of Information Technology,	Department of Information Technology,
Sri Krishna College of Technology,	Sri Krishna College of Technology,
Coimbatore-641042	Coimbatore-641042

Certified that the candidate was examined by me in the Project Work Viva Voce examination held on_______________at Sri Krishna College of Technology, Coimbatore-641042.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

First and foremost, we thank the **Almighty** for being our light and for showering his gracious blessings throughout the course of this project.

We are grateful to our beloved Principal **Dr.M.G. Sumithra M.E., Ph.D**. for her tireless and relentless support.

We are grateful to our beloved Head, Computing Sciences **Dr.T. Senthilnathan,** for his tireless and relentless support.

We extend our sincere thanks to our Head of the Department of Information Technology **Dr. T. Rajesh Kumar** for his encouragement and inspiration.

We sincerely thank our supervisor **Ms. P.Alaguvathana**, Assistant Professor, Department of Information Technology, for her motivation and support.

We are thankful to all the **Teaching and Non-Teaching Staff** of Department of Information Technology and to all those who have directly and indirectly extended their help to us in completing this project work successfully.

We extend our sincere thanks to our **family members** and our beloved **friends**, who had been strongly supporting us in all our endeavour.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF FIGURES

LIST OF ABBREVIATIONS

LIST OF ABBREVIATIONS

ABBREVIATIONS	**ACRONYMS**
HTML	HYPERTEXT MARKUP LANGUAGE
CSS	CASCADING STYLE SHEET
JS	JAVASCRIPT
API	APPLICATION PROGRAMMING INTERFACE
JPA	JAVA PERSISTENCE API
ORM	OBJECT-RELATIONAL MAPPING
JSON	JAVASCRIPT OBJECT NOTATION

CHAPTER 1

INTRODUCTION

The Budget Buddy Project, also known as the Budget Planner, is a comprehensive financial management tool designed to help individuals and households monitor and control their spending habits. In today's fast-paced world, managing personal finances can be a challenging task. With countless transactions and varying expenses, it's easy to lose track of where your money goes. Budget Buddy is an intuitive and user-friendly expense tracker app designed to help individuals take control of their finances. Whether you're trying to save for a big purchase, reduce unnecessary spending, or simply keep a better eye on your budget, Budget Buddy is your go-to solution.Budget Buddy allows users to effortlessly record their daily expenses, categorize them, and monitor their spending patterns over time.

With features like budget setting, spending limits, and insightful financial reports, the app provides a clear picture of your financial health. By making it simple to track and analyze expenses, Budget Buddy empowers users to make informed financial decisions, ensuring they stay on top of their financial goals.This project categorizes expenses into various segments such as food, transportation, entertainment, utilities, and more, allowing users to gain insights into their financial behaviours .By offering a detailed breakdown of expenditures, the system aims to promote better budgeting practices, ultimately leading to improved financial health and stability. With intuitive interfaces and advanced analytical features, the Budget Planner facilitates a seamless and engaging user experience.

1.1 PROBLEM STATEMENT

In today’s fast-paced world, managing personal finances has become increasingly challenging. Many individuals struggle to keep track of their daily expenses, often leading to overspending, debt accumulation, and difficulty in saving money.

1.2 OVERVIEW

An Budget Buddy is a tool, typically in the form of a mobile or web-based application, designed to help individuals and households monitor, categorize, and manage their spending. The primary goal of an expense tracker is to provide users with clear insights into their financial behavior, allowing them to make informed decisions, improve budgeting, and achieve financial goals.

1.3 OBJECTIVE

The objective of an Budget Buddy app is to empower individuals and households to manage their finances effectively by providing them with a comprehensive tool to monitor, categorize, and analyze their spending habits. Through the app, users can gain greater financial awareness, set and adhere to budgets, reduce unnecessary expenses, and work towards achieving their financial goals. The app aims to simplify personal financial management.

CHAPTER 2

LITERATURE SURVEY

2.1 RELATED WORKS

1) K. M. O'Reilly, "User Interface Design for Expense Tracking Applications" (2021)
    - This study explores the design principles of user interfaces for expense tracking apps, emphasizing the importance of simplicity and ease of use. The research highlights how intuitive navigation and clear visualizations improve user satisfaction and engagement.

2) P. H. L. Smith et al., "Designing for Financial Literacy: The Role of UI in Expense Tracking" (2019)
    - This paper examines how the user interface (UI) influences financial literacy. It suggests that well-designed UI elements in expense tracker apps can enhance users' understanding of their spending patterns and budgeting goals.

3) J. L. Miller et al., "Data Privacy Concerns in Personal Finance Apps" (2019)
    - This study addresses the privacy concerns associated with personal finance apps, including expense trackers. It emphasizes the need for robust data protection measures and transparent privacy policies to gain users' trust.

CHAPTER 3
SYSTEM SPECIFICATION

In this chapter, we are gonna see the softwares that we have used to build the website. This chapter gives you a small description about the softwares used in the project.

3.1 VS CODE

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux, and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences.

VS Code is an excellent code editor for React projects. It is lightweight, customizable, and has a wide range of features that make it ideal for React development. It has built-in support for JavaScript, JSX, and TypeScript, and enables developers to quickly move between files and view detailed type definitions. It also has a built-in terminal for running tasks. Additionally, VS Code has an extensive library of extensions that allow developers to quickly add features like code snippets, debugging tools, and linting support to their projects.

3.2. REACT

React is a JavaScript library created by Facebook for building user interfaces. It is a component-based, declarative, and highly efficient library that is used to develop interactive UIs (user interfaces) for single page web applications. React uses a virtual

DOM (Document Object Model) that makes it faster and easier to manipulate the elements. It also provides declarative components that allow developers to write code thatis easy to read and maintain. React also offers an extensive library of tools and components that make it easier to develop complex user interfaces.

3.3 ROUTERS IN REACT

Routers are important components in React applications. They provide the ability to navigate between different views or components of the application. React Router is themost popular library to handle routing in React applications. It provides the ability to define routes, set up links, and render components based on the current route. It also provides features like data fetching, code-splitting, and server-side rendering.

3.4 LOCAL STORAGE

Local storage is a type of web storage for storing data on the client side of a web browser. It allows websites to store data on a user's computer, which can then be accessed by the website again when the user returns. Local storage is a more secure alternative to cookies because it allows websites to store data without having to send it back and forth with each request.

Local storage is a key-value pair storage mechanism, meaning it stores data in the form of a key and corresponding value. It is similar to a database table in that it stores data in columns and rows, except that local storage stores the data in the browser rather than in a database. Local storage is often used to store user information such as preferences and settings, or to store data that is not meant to be shared with other websites. It is also used to cache data to improve the performance of a website. Local storage is supported by all modern web browsers, including Chrome,

3.5 SPRING BOOT

Spring Boot is an open-source framework developed by Pivotal Software that simplifies the creation of stand-alone, production-grade Spring-based applications. It is built on the Spring framework and offers a range of features to streamline development, including auto-configuration, embedded servers, and starter dependencies. Spring Boot eliminates the need for complex XML configuration and provides sensible defaults to help developers get started quickly.

Spring Boot is highly effective for building Java-based web applications and microservices. It supports a variety of data sources and web technologies, integrates seamlessly with popular tools and frameworks, and provides robust support for security, testing, and monitoring. With its convention-over-configuration approach, Spring Boot enables rapid development and deployment while maintaining flexibility and scalability. It includes built-in support for common tasks like database access, messaging, and web services, making it an excellent choice for both small and large-scale applications.

3.6 VITE

Vite is a modern build tool that aims to improve the development experience for web applications. Created by Evan You, the author of Vue.js, Vite is designed to be fast and efficient, addressing some of the limitations of traditional build tools like Webpack. Here's anoverview of Vite and its key features: Vite leverages native ES modules in the browser, allowing for lightning-fast startup times. modules in the browser, which improves development and serves only the parts of the application that are being used.

CHAPTER 4

PROPOSED SYSTEM

This chapter gives a small description about the proposed idea behind the development of our website.

4.1 PROPOSED SYSTEM

The proposed Expense Tracker Project aims to address the limitations of existing systems by offering a robust, user-friendly platform that provides detailed categorization and analysis of expenditures. This system will feature multiple expense categories, including but not limited to food, transportation, entertainment, utilities, healthcare, and miscellaneous expenses. Users will be able to input their expenses through a streamlined interface, which will then automatically categorize and log the data. The platform's design will prioritize ease of use, ensuring that users of all technical proficiencies can effectively manage their finances without unnecessary complexity. The platform will offer real-time tracking and updates, ensuring users are always aware of their current financial status.

4.2 ADVANTAGES:

1.Budget Management: It helps users create and maintain budgets, ensuring they live within their means and avoid overspending. By categorizing expenses (e.g., groceries, entertainment, utilities), users can see where their money is going and identify areas to cut back.

2.**Real-Time Tracking**: The app allows for real-time tracking of expenses, giving users an up-to paying off debt, and monitor their progress towards these goals.

3.**Financial Awareness**: By providing detailed reports and summaries, the app enhances users' awareness of their spending habits, leading to better financial decisions. Notifications and alerts can help users stay on top of bills and avoid late fees or overdraft charges.

4.**Convenience**: The app provides a convenient, centralized platform for managing finances, eliminating the need for manual record-keeping. Many apps offer customizable features, allowing users to tailor the app to their specific financial needs and preferences.

5.**Data Insights:** With data analytics, users can gain insights into their spending patterns over time, helping them adjust their habits for better financial health. Many expense tracker apps offer high-level security features to protect users' financial data, including encryption and secure login methods.

CHAPTER 5

METHODOLOGIES

This chapter gives a small description about how our system works.

5.1 PROCESS FLOWCHART DIAGRAM

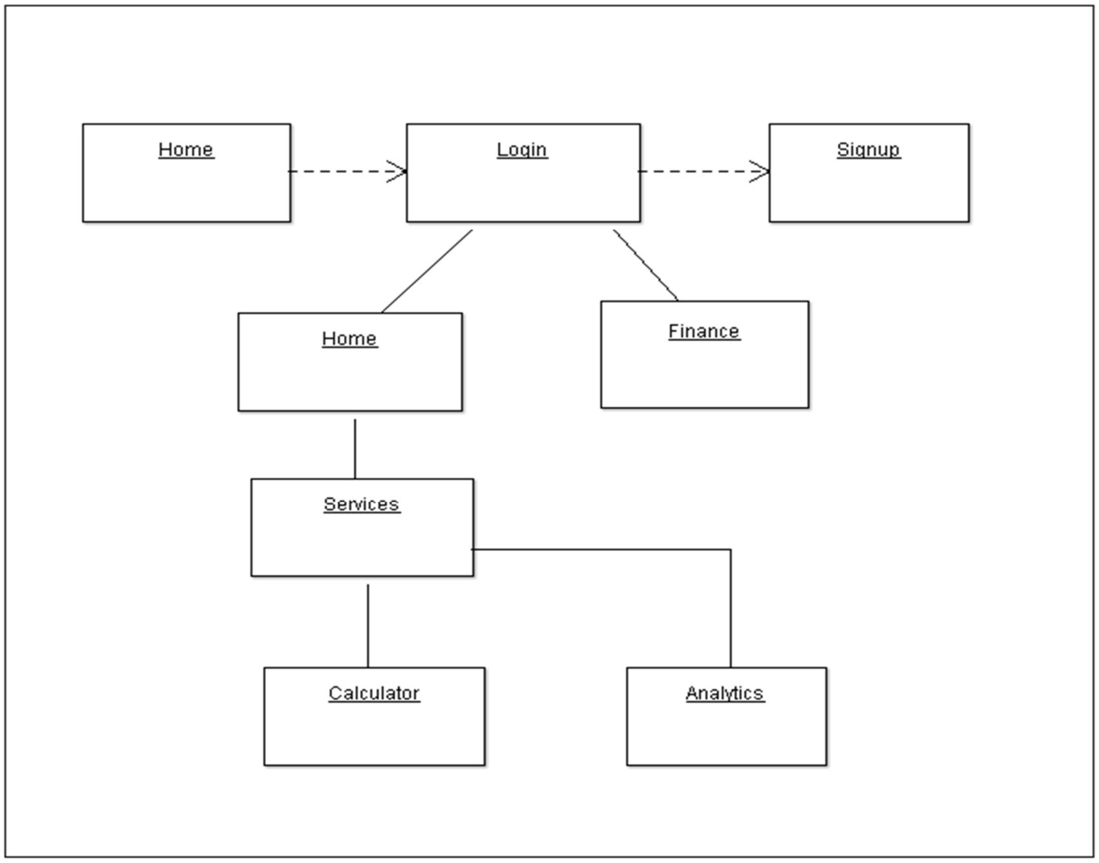


Fig 5.1 PROCESS FLOWCHART

5.2 LOGIN

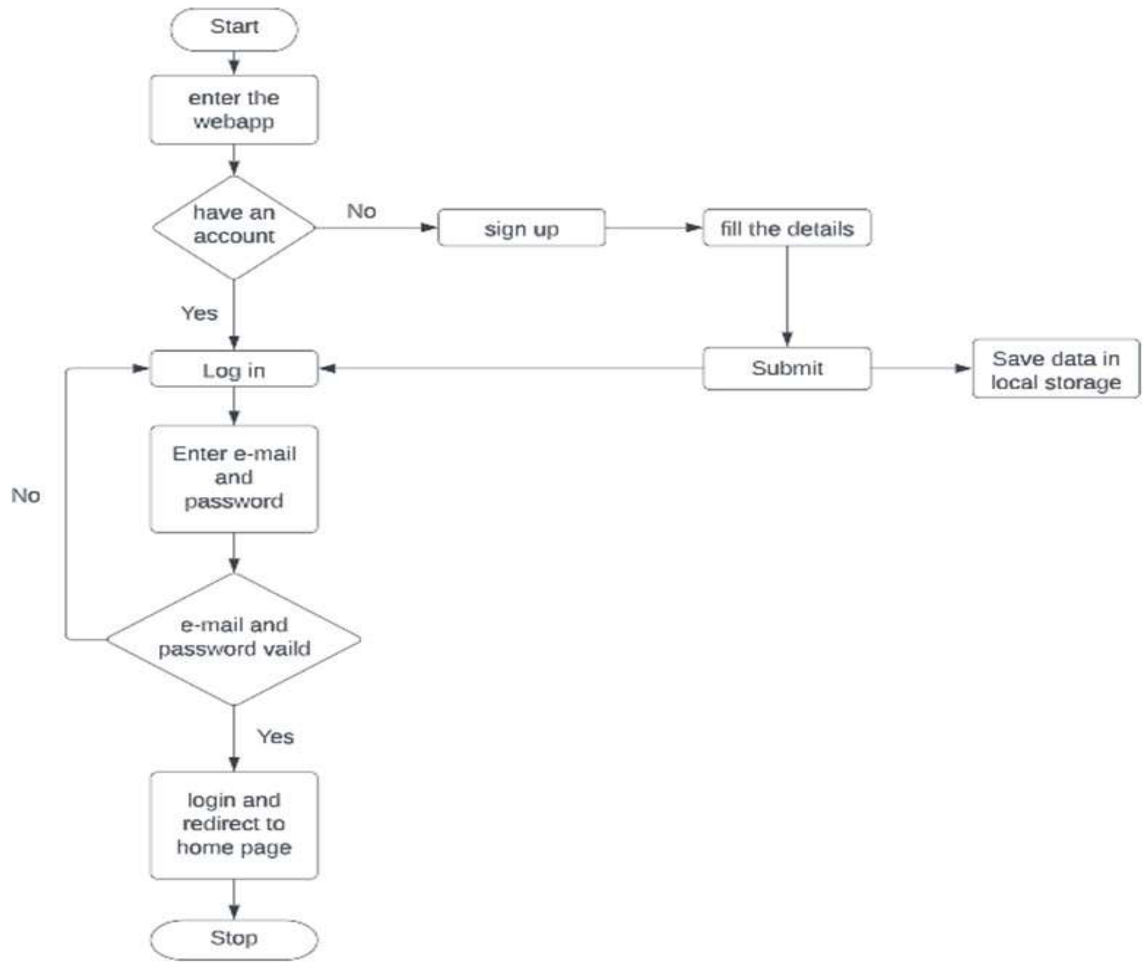

Start
enter the webapp
have an account
No
sign up
fill the details
Yes
Log in
Submit
Save data in local storage
Enter e-mail and password
No
e-mail and password vaild
Yes
login and redirect to home page
Stop

Fig 5.2 LOGIN FLOWCHART

In this page we will be asking about the username and password of the user. Firstly the website validates the user inputs. It verifies the username and password by checking it with the usernames and passwords stored in the local storage when the user creates an account in the website.

5.3 SIGNUP

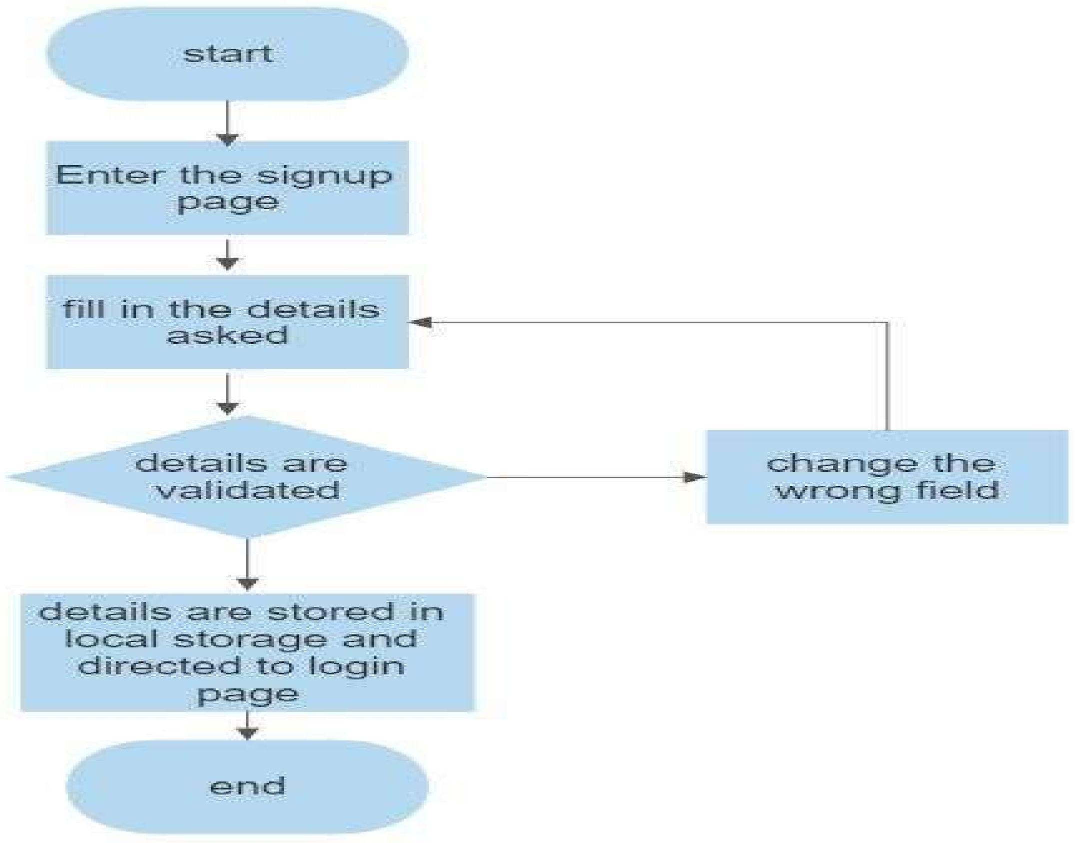

start
Enter the signup page
fill in the details asked
details are validated
change the wrong field
details are stored in local storage and directed to login page
end

Fig 5.3 SIGNUP FLOWCHART

In this page we will be asking about the username and password and mailid of the user. Firstly the user give the details and register it. It verifies password by checking it with confirm password and passwords stored in the local storage when the user creates an account in the website.

5.4 EXPENSE CALCULATOR

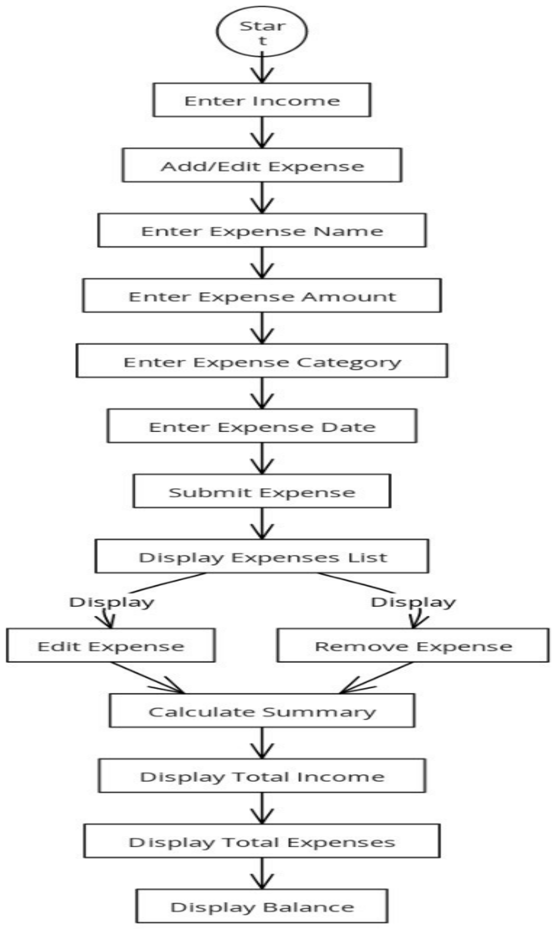


Fig 5.4 EXPENSE CALCULATOR FLOWCHART

In this page An Expense Calculator is a tool designed to help individuals manage their personal finances by tracking and categorizing expenses. It provides a systematic way to record daily expenditures, monitor spending habits, and ensure that one stays within a defined budget.

5.5 SERVICES PAGE:

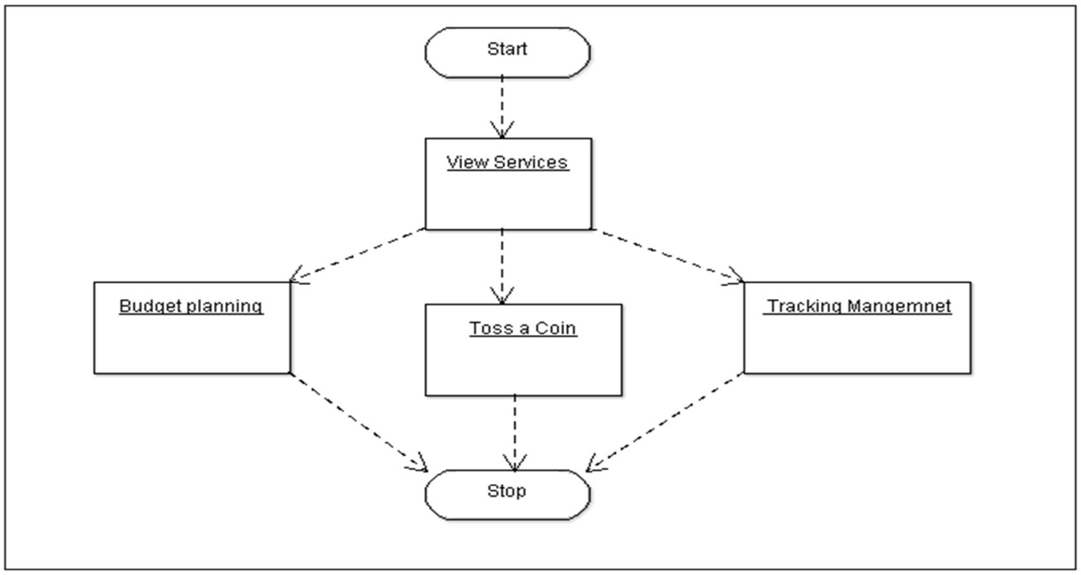


Fig 5.5 SERVICE PAGE FLOWCHART

A Services Page in an expense tracker app typically outlines various features and functionalities available to users, focusing on how these services can enhance their financial management. And also provides category page. And provides Budget planning to plan the budget direct to calculator page.

5.6 ANALYTICS PAGE:

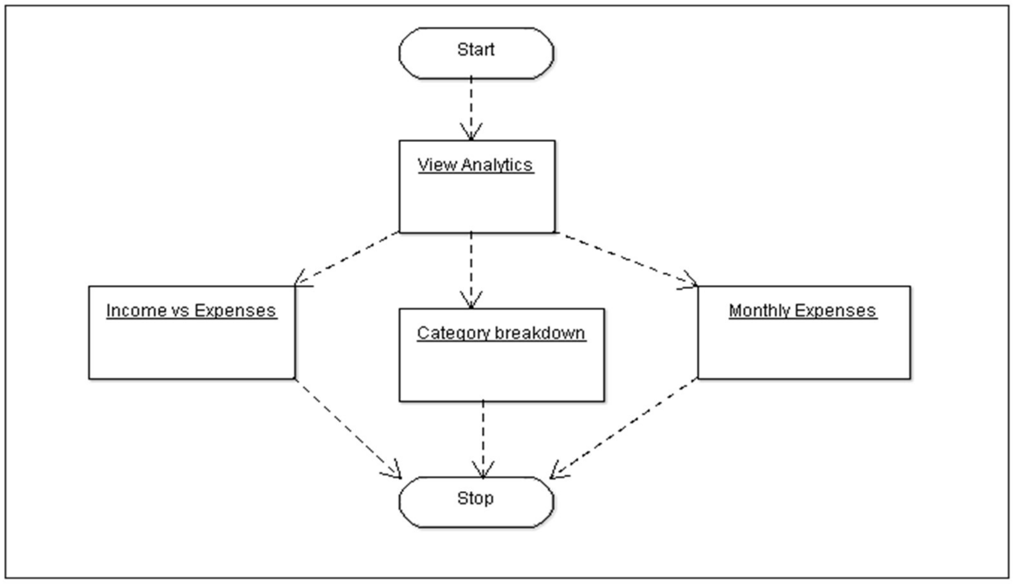


Fig 5.6 ANALYTICS PAGE FLOWCHART

An Analytics Page for an expense tracking app provides users with insights and data visualizations to help them understand their spending patterns, track financial goals, and make informed decisions. And also it gives graph , piechart and analysis graph.

5.7 FINANCE PAGE

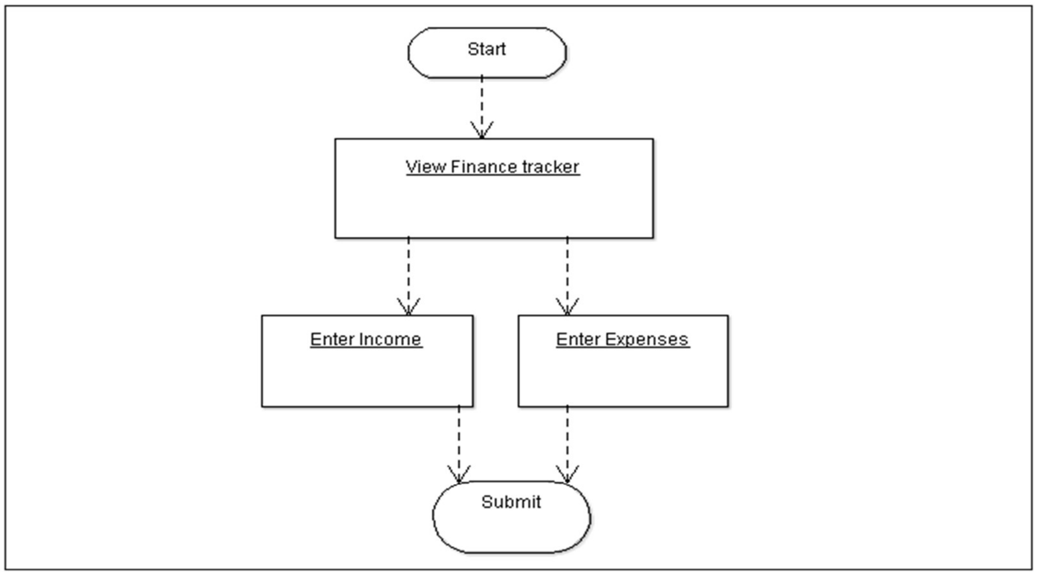


Fig 5.7 FINANCE PAGE FLOWCHART

The Finance page for an expense tracker app is crucial for organizing and managing various types of expenses. This page allows users to categorize their giving, rent, utility, groceries , and make adjustments as needed. And also it have paycheck and Hustle operations.

5.8 PROFILE PAGE

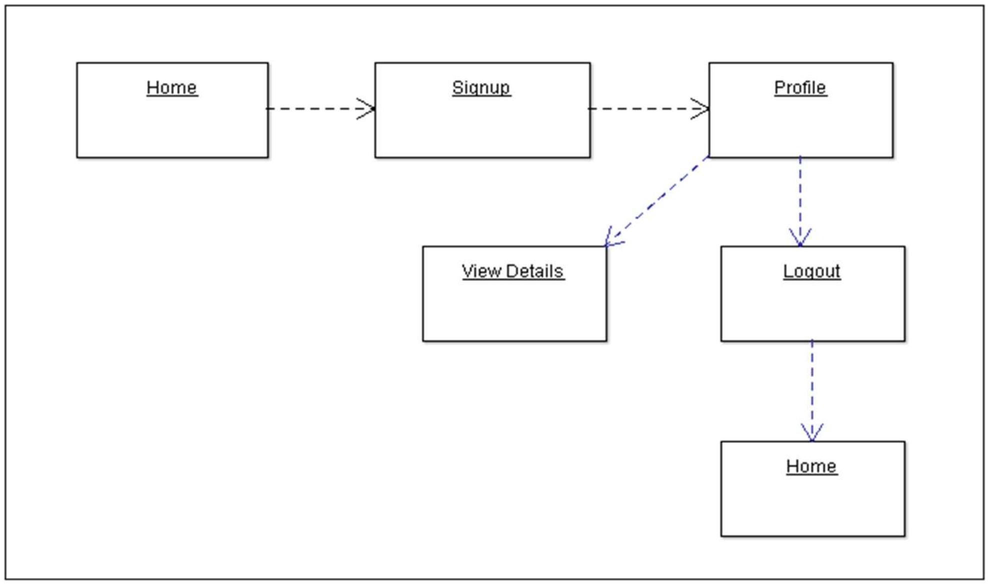


Fig 5.8 PROFILE PAGE

The ProfilePage component in your application serves as a central hub where users can view and manage their personal information. It typically includes features like displaying the user's name and email address, as well as it has a logout functionality.when I logout it navigate to the homepage.

CHAPTER 6
IMPLEMENTATION AND RESULT

This chapter gives a description about the output that we produced by developing the website of our idea.

6.1 LOGIN

When User enters our website he/she will be asked about his/her login details like email id and password. The login details will be verified with the details given while the user creates an account and it redirects login credentials else redirects to home page.

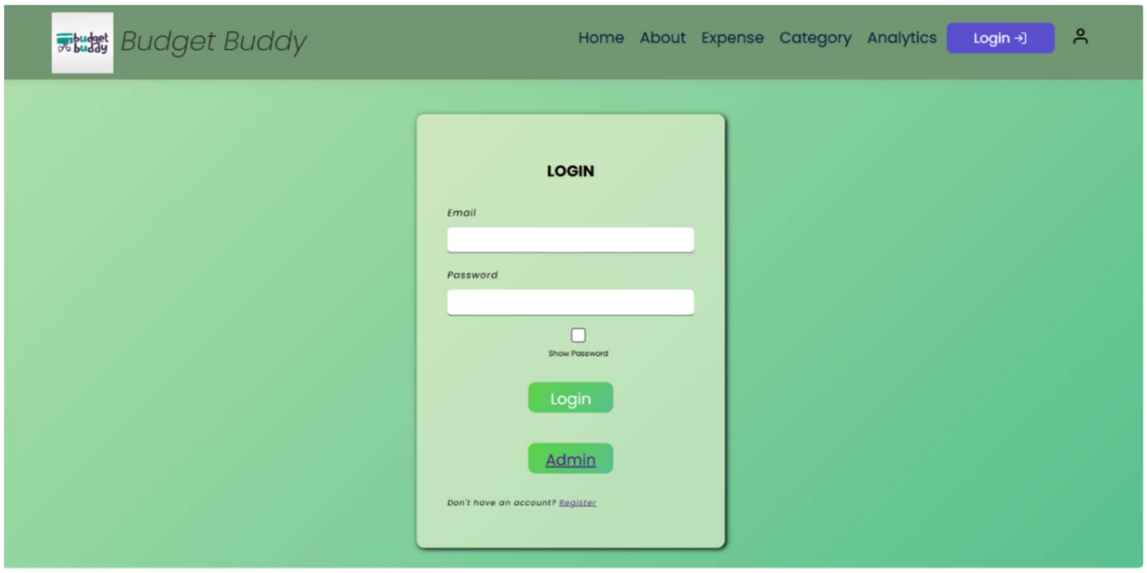


Fig 6.1 LOGIN PAGE

6.2 SIGN UP

If a user doesn't have an account on the website, User can use a component namedsign up in the login page. When the user clicks on that he will be redirected to the signup page. In sign up he/she should fill up his/her name, email id, password and confirm password. These inputs will be validated.

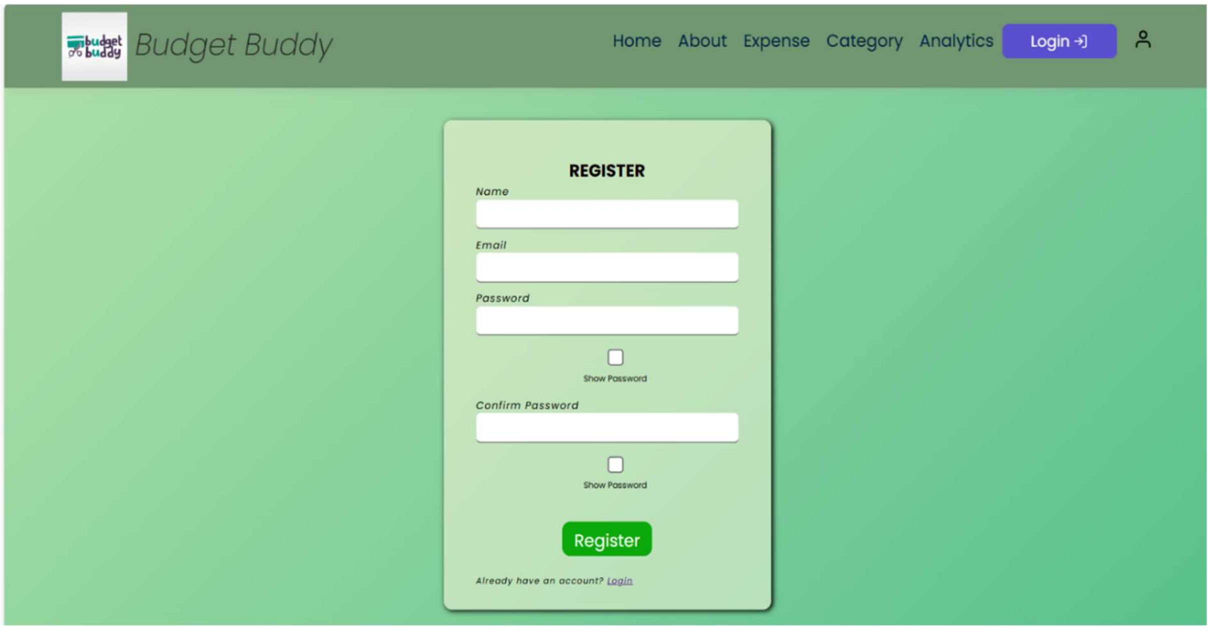
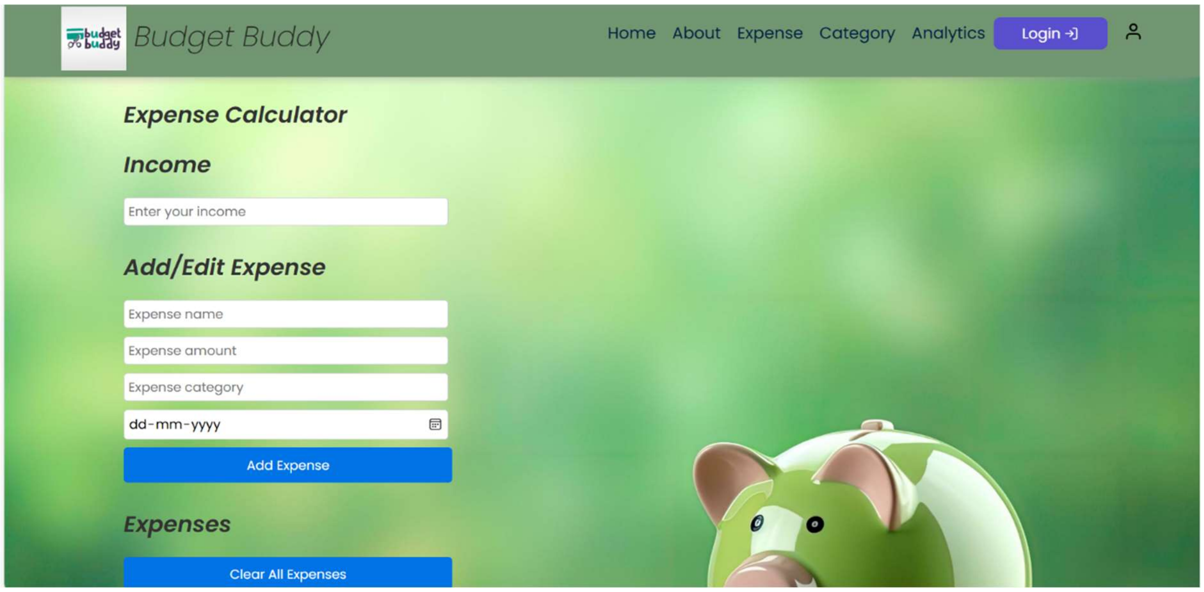
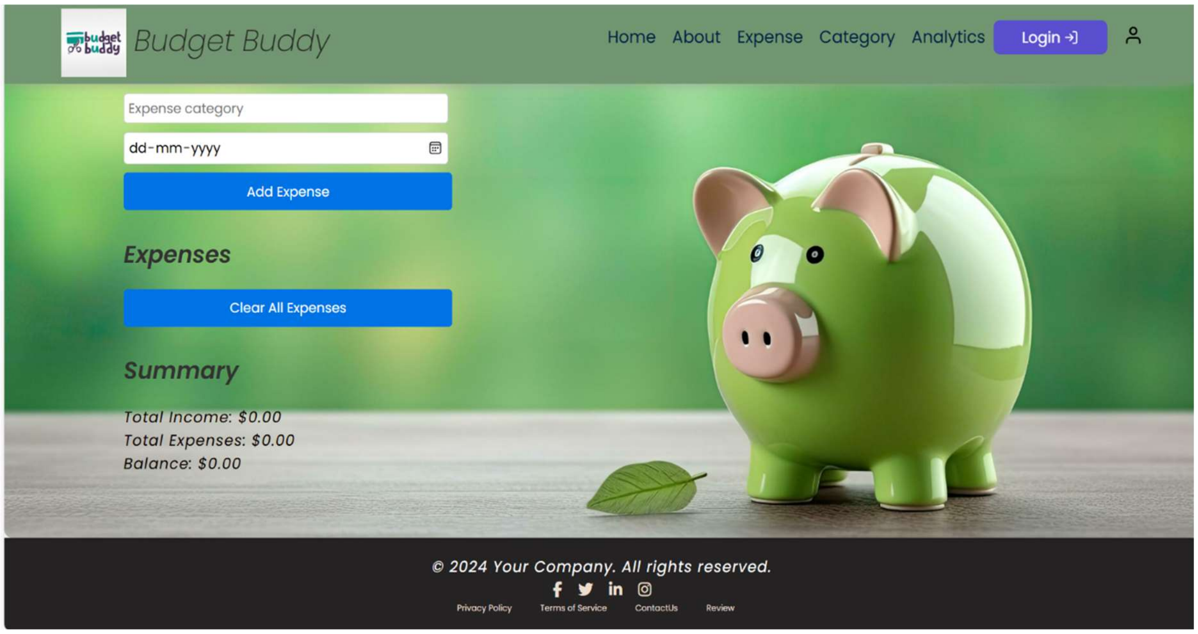


Fig 6.2 SIGNUP PAGE

6.3 EXPENSE CALCULATOR

An Expense Calculator is a tool designed to help individuals manage their personal finances by tracking and categorizing expenses. It provides a systematic way to record daily expenditures, monitor spending habits, and ensure that one stays within a defined budget. With an expense calculator, users can easily input their expenses, categorize them (such as groceries, utilities, entertainment, etc.), and get a clear picture of where their money is going.





Expenses

Summary

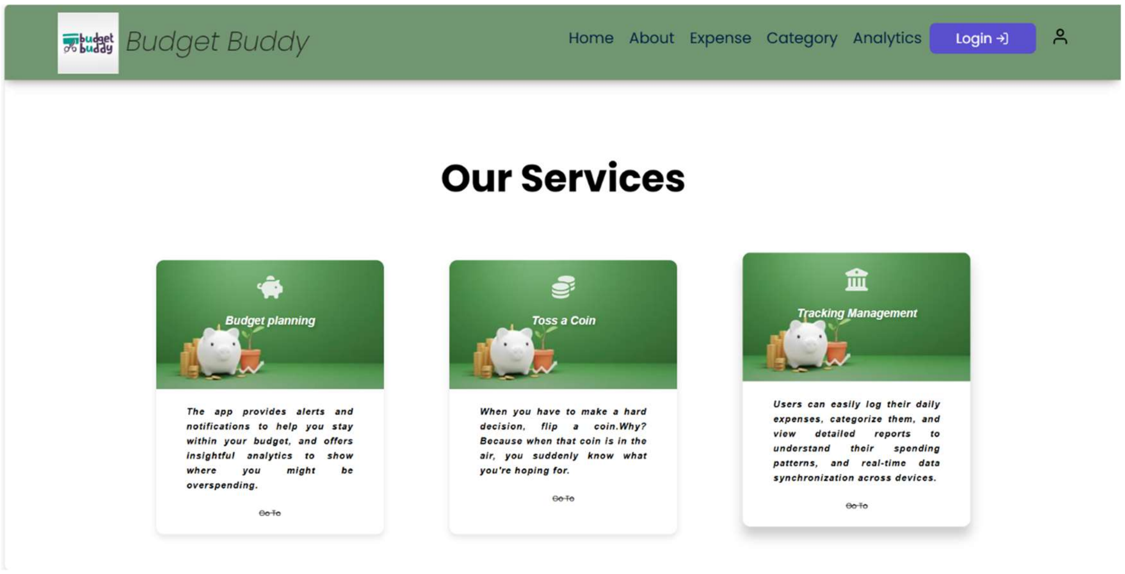
Total Income: $0.00
Total Expenses: $0.00
Balance: $0.00

© 2024 Your Company. All rights reserved.

Fig 6.3 EXPENSE CALCULATOR PAGE

6.4 SERVICE PAGE

A Services Page in an expense tracker app typically outlines various features and functionalities available to users, focusing on how these services can enhance their financial management. And also provides category page.



Our Services

Budget planning

The app provides alerts and notifications to help you stay within your budget, and offers insightful analytics to show where you might be overspending.

~~Go To~~

Toss a Coin

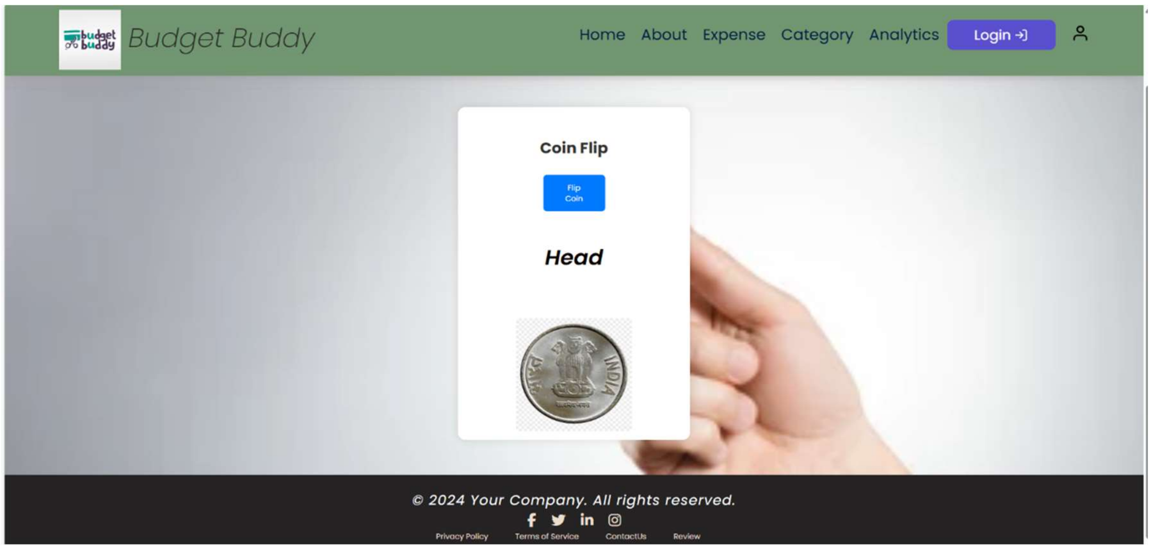
When you have to make a hard decision, flip a coin.Why? Because when that coin is in the air, you suddenly know what you're hoping for.

~~Go To~~

Tracking Management

Users can easily log their daily expenses, categorize them, and view detailed reports to understand their spending patterns, and real-time data synchronization across devices.

~~Go To~~



© 2024 Your Company. All rights reserved.

Fig 6.4 SERVICES PAGE

6.5 FINANCE PAGE

The Finance page for an expense tracker app is crucial for organizing and managing various types of expenses. This page allows users to categorize their giving, rent, utility, groceries , and make adjustments as needed. And also it have paycheck and Hustle operations.

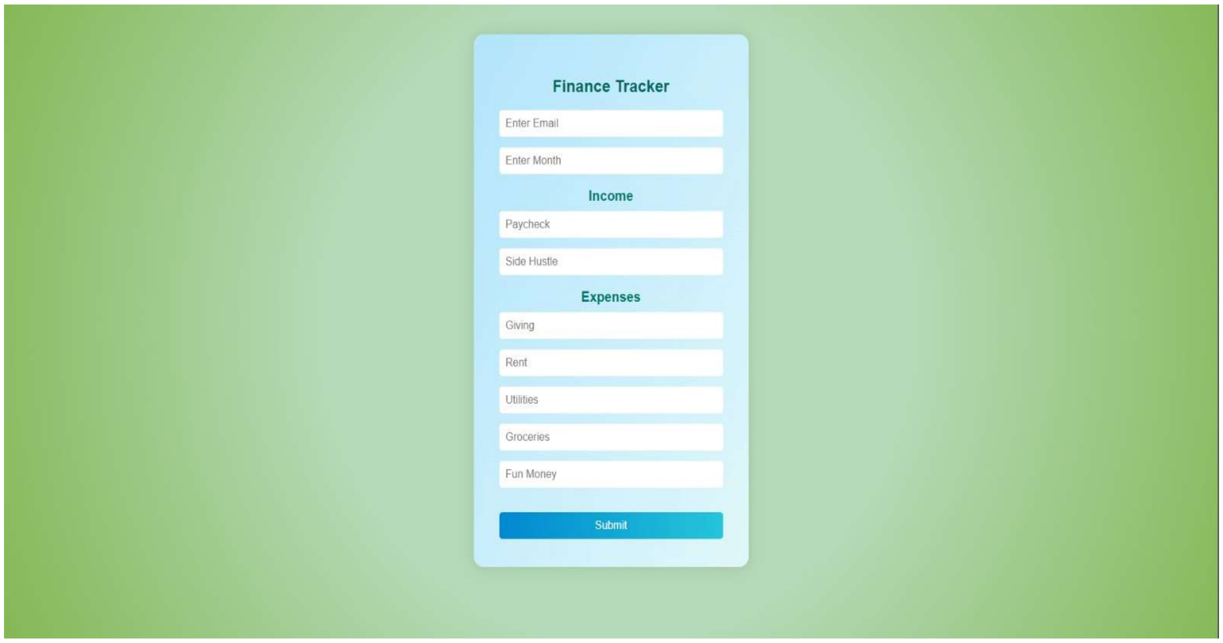
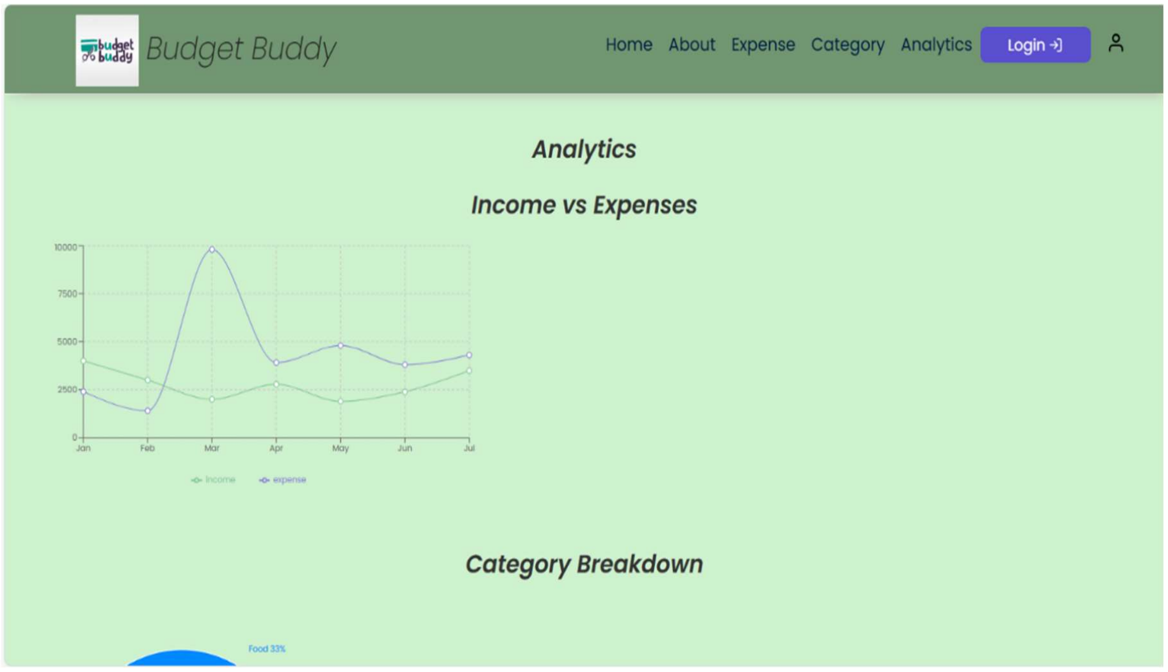
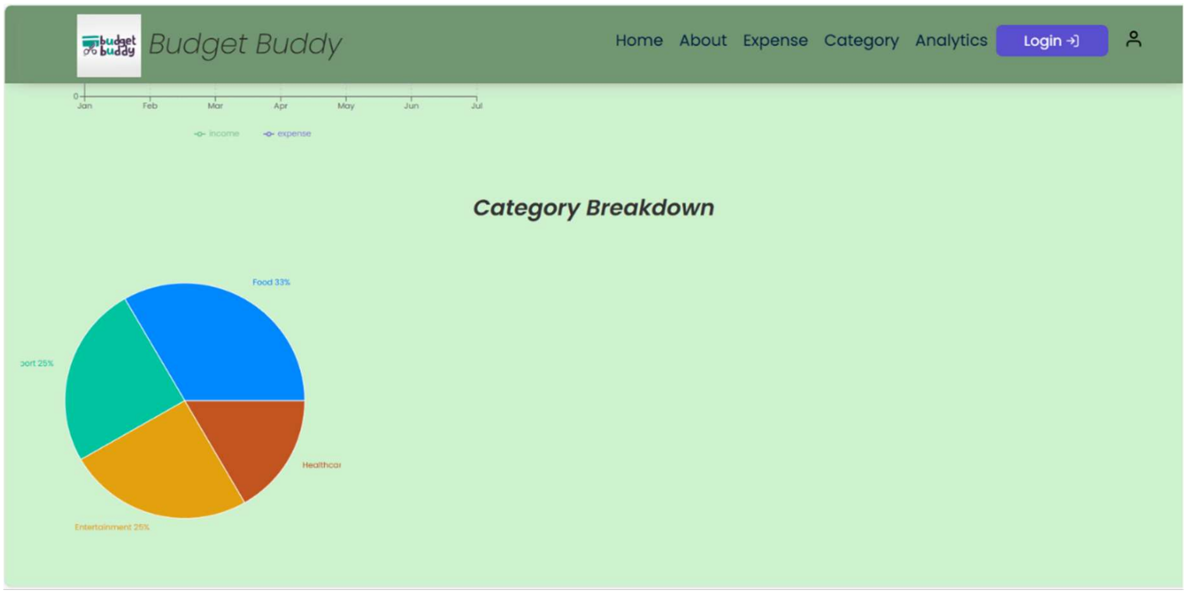


Fig 6.5 FINANCE PAGE

6.6 ANALYTICS PAGE

An Analytics Page for an expense tracking app provides users with insights and data visualizations to help them understand their spending patterns, track financial goals, and make informed decisions. And also it gives graph ,piechart and analysis graph.





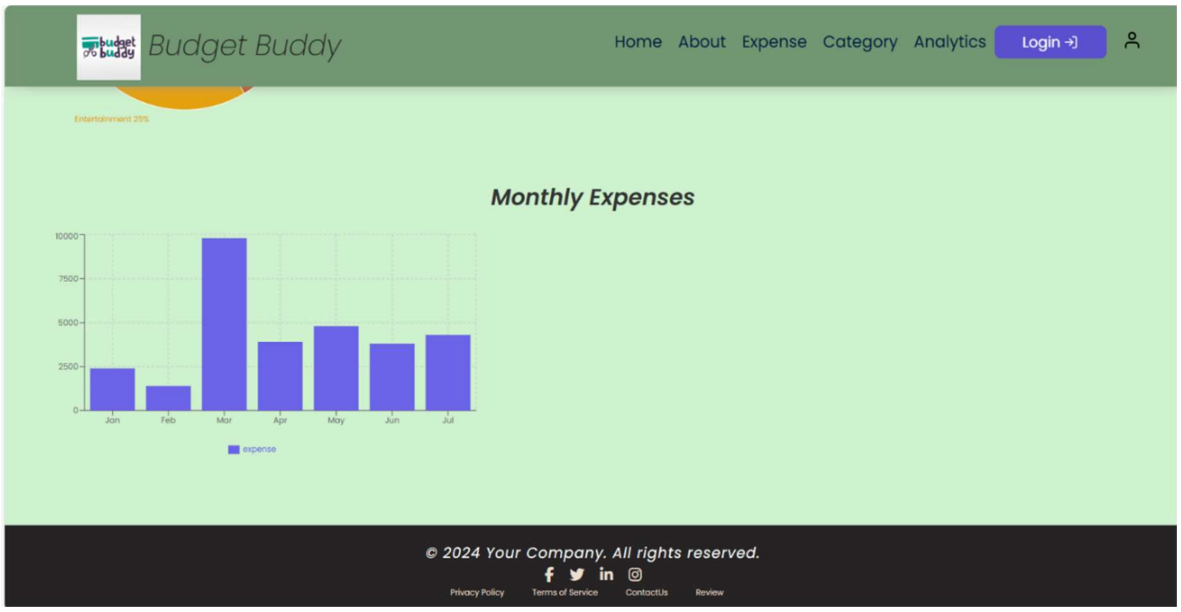
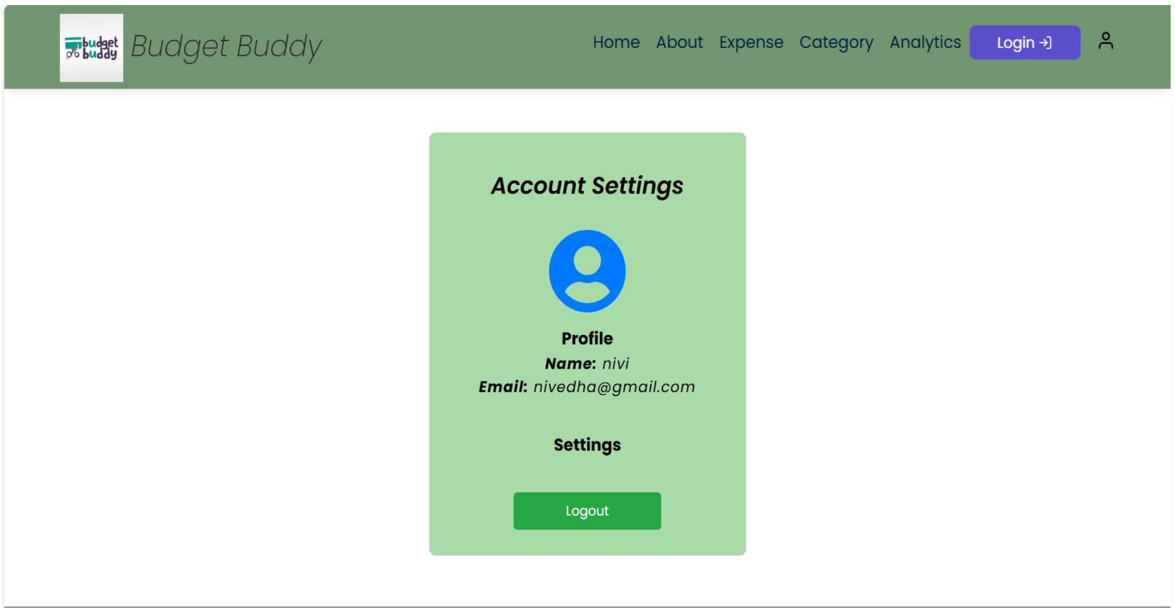


Fig 6.6 ANALYTICS PAGE

6.7 PROFILE PAGE:

The ProfilePage component in your application serves as a central hub where users can view and manage their personal information. It typically includes features like displaying the user's name and email address, as well as it has a logout functionality.

.



Name: nivi
Email: nivedha@gmail.com

Fig 6.7 PROFILE PAGE

6.8 CODING

LOGIN:

```
import axios from 'axios'; // Import axios for making HTTP requests
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import '../Login/Login.css';
function Login() {
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
const [showPassword, setShowPassword] = useState(false);
const [errors, setErrors] = useState({});
const [loading, setLoading] = useState(false);
const navigate = useNavigate();
const handleEmailChange = (event) => {
setEmail(event.target.value);
};
const handlePasswordChange = (event) => {
setPassword(event.target.value);
};
const validateForm = () => {
const newErrors = {};
if (!email) {
newErrors.email = "Email is required.";
}
if (!password) {
```

```
newErrors.password = "Password is required.";
}
setErrors(newErrors);
return Object.keys(newErrors).length === 0;
};
const handleSubmit = async (event) => {
event.preventDefault();
if (validateForm()) {
setLoading(true);
try {
console.log("in")
const response = await axios.post('http://localhost:8080/api/users/login', {
email: email,
password: password,
});
console.log('User logged in:', response.data);
setLoading(false);
navigate('/Homepage1'); // Navigate to the dashboard or home page after successful
login
} catch (error) {
console.error('Error logging in:', error.response ? error.response.data : error.message);
setErrors({ auth: 'Failed to log in. Please try again.' });
setLoading(false);
}
}
};
return (
<div className="login-body">
<div className="login-main">
<h1>LOGIN</h1>
{errors.auth && <p className="errP">{errors.auth}</p>}
<p>Email</p>
<input
type='text'
value={email}
onChange={handleEmailChange}
/>
{errors.email && <p className="errP">{errors.email}</p>}
<p>Password</p>
<input
type={showPassword ? 'text' : 'password'}
value={password}
onChange={handlePasswordChange}
/>
{errors.password && <p className="errP">{errors.password}</p>}
<label className="checkbox-container">
<input
type="checkbox"
checked={showPassword}
onChnge={() => setShowPassword(!showPassword)}
/>
Show Password
</label>
<center><button onClick={handleSubmit}
```

```
disabled={loading}>Login</button></center>
<center><button  disabled={loading}><Link
to={'/AdminPage'}>Admin</Link></button></center>
<p style={{ fontSize: '10px' }}>Don't have an account? <Link
to={'/Signup'}>Register</Link></p>
</div>
</div>
);
}
export default Login;
```

SIGNUP:

```
import axios from 'axios'; // Import axios for making HTTP requests
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import '../Signup/signup.css';

function Register() {
const [username, setUsername] = useState('');
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
const [confirmPassword, setConfirmPassword] = useState('');
const [nameErr, setNameErr] = useState(false);
const [showPassword, setShowPassword] = useState(false);
const [showRPassword, setShowRPassword] = useState(false);
const navigate = useNavigate();
function registration() {
if ((username.trim().length === 0) || (password.trim().length === 0) ||
(email.trim().length === 0)) {
setNameErr(true);
} else if (!email.includes('@') || !email.includes('.') || !email.includes('com')) {
alert('Please enter a valid email address');

} else if (password.length < 5) {
alert('Please enter a password with more than eight characters');
} else if (password !== confirmPassword) {
alert('Passwords do not match');
navigate('/Signup');
}
else {
setNameErr(false);
const user = { name: username, email: email, password: password };
axios.post('http://localhost:8080/api/users/signup', user)
.then(response => {
console.log(response.data);
navigate('/login');
})
.catch(error => {
console.error('There was an error registering the user!', error);
});
}
}
return (
<div className="register-body">
<div className="register-main">
<h1>REGISTER</h1>
{nameErr && <p className="errP">Please fill every input field</p>}
<p>Name</p>
<input
type='text'
value={username}
onChange={(e) => setUsername(e.target.value)}
/>
<p>Email</p>
<input
type='text'
value={email}
onChange={(e) => setEmail(e.target.value)}
/>
<p>Password</p>
<input
type={showPassword ? 'text' : 'password'}
value={password}
onChange={(e) => setPassword(e.target.value)}
/>
<div className="checkbox-container">
```

```
<input
type="checkbox"
checked={showPassword}
onChange={() => setShowPassword(!showPassword)}
/>
<label>Show Password</label>
</div>
<p>Confirm Password</p>
<input
type={showRPassword ? 'text' : 'password'}
value={confirmPassword}
onChange={(e) => setConfirmPassword(e.target.value)}
<div className="checkbox-container">
<input
type="checkbox"
checked={showRPassword}
onChange={() => setShowRPassword(!showRPassword)}
<center><Link to={'/Signup'}><button
onClick={registration}>Register</button></Link></center>
<p style={{ fontSize: '10px' }}>Already have an account? <Link
to={'/login'}>Login</Link></p>
</div>
</div>
);

export default Register;
```

LOGIN & SIGNUP:

Controller:

```
package com.example.demo.controller;
import com.example.demo.model.User;
import com.example.demo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api/users")
@CrossOrigin(origins = "http://localhost:5174")
public class UserController {
@Autowired
  private UserService userService
 @PostMapping("/signup")
    public ResponseEntity<String> registerUser(@RequestBody User user) {
   User registeredUser = userService.registerUser(user);
  return ResponseEntity.ok("User registered successfully with ID: " +
registeredUser.getId());
   }
   @PostMapping("/login")
    public ResponseEntity<?> loginUser(@RequestBody User user) {


User loggedInUser = userService.loginUser(user.getEmail(), user.getPassword());
```

```
 if (loggedInUser != null) {
return ResponseEntity.ok(loggedInUser);
} else {
return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid email or
password.");
    }
  }
}
```

Model:

```
package com.example.demo.model;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
@Entity
@Table(name="up")
public class User {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String name;
private String email;

private String password;
// Constructors, getters, and setters
```

```
public User() {}
public User(String name, String email, String password) {
this.name = name;
this.email = email;
this.password = password;
 }
 public Long getId() {
        return id;
    }
   public void setId(Long id) {
        this.id = id;
    }
   public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {

this.email = email;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

Repository:

```
package com.example.demo.repository;
import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.example.demo.model.User;
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
}
```

Service:

```
package com.example.demo.service;
import com.example.demo.model.User;
import com.example.demo.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class UserService {
```

```
@Autowired
   private UserRepository userRepository;

   public User registerUser(User user) {
      return userRepository.save(user);
   }
   public User findUserByEmail(String email) {
      return userRepository.findByEmail(email).orElse(null);
   }

   public User loginUser(String email, String password) {
      User user = findUserByEmail(email);
      if (user != null && user.getPassword().equals(password)) {
         return user;
      }
      return null; // Return null if authentication fails
   }
}
```

EXPENSE CALCULATOR:

```
import '../Expense/ExpenseCalculator.css';
// src/ExpenseCalculator.jsx
import React, { useState, useEffect } from 'react';
// import './ExpenseCalculator.css';
import Navbar from '../../components/Navbar/Navbar';
const ExpenseCalculator = () => {
const [income, setIncome] = useState('');
const [expenseName, setExpenseName] = useState('');
const [expenseAmount, setExpenseAmount] = useState('');
const [expenseCategory, setExpenseCategory] = useState('');
const [expenseDate, setExpenseDate] = useState('');
const [expenses, setExpenses] = useState([]);
const [editIndex, setEditIndex] = useState(null);
useEffect(() => {
const storedExpenses = JSON.parse(localStorage.getItem('expenses'));
if (storedExpenses) {
setExpenses(storedExpenses);
}
}, []);
useEffect(() => {
localStorage.setItem('expenses', JSON.stringify(expenses));
}, [expenses]);
const addIncome = (e) => {
setIncome(e.target.value);
};
const handleExpenseSubmit = () => {
if (expenseName && expenseAmount && expenseCategory && expenseDate) {
const newExpense = { name: expenseName, amount: parseFloat(expenseAmount),
category: expenseCategory, date: expenseDate };
if (editIndex !== null) {
const updatedExpenses = [...expenses];
updatedExpenses[editIndex] = newExpense;
setExpenses(updatedExpenses);
setEditIndex(null);
} else {
 setExpenses([...expenses, newExpense]);
      }
      setExpenseName('');
      setExpenseAmount('');
      setExpenseCategory('');
      setExpenseDate('');
    }
  };
  const editExpense = (index) => {
    const expense = expenses[index];
    setExpenseName(expense.name);
    setExpenseAmount(expense.amount);
    setExpenseCategory(expense.category);
    setExpenseDate(expense.date);
    setEditIndex(index);


};
const removeExpense = (indexToRemove) => {
 setExpenses(expenses.filter((_, index) => index !== indexToRemove));
  };
  const clearAllExpenses = () => {
    setExpenses([]);
  };
  const totalExpenses = expenses.reduce((acc, expense) => acc + expense.amount, 0);
  const balance = income - totalExpenses;
  console.log('Expenses:', expenses); // Log expenses to console
 return (
<section>
<Navbar/>
<div className="expensecontainer">
<div className='cal'>
<br></br>
<h2 className='exa'>Expense Calculator</h2>
</div>

<div style={{ marginBottom: '20px' }}>
<h2 className='exa'>Income</h2>
<input
type="number"
value={income}
onChange={(e) => addIncome(e)}
placeholder="Enter your income"
className="small-input"

/>
</div>
<div style={{ marginBottom: '20px' }}>
<h2 className='exa'>Add/Edit Expense</h2>
<input
type="text"
value={expenseName}
onChange={(e) => setExpenseName(e.target.value)}
placeholder="Expense name"
className="small-input"
/>
```

```
<input
type="number"
value={expenseAmount}
onChange={(e) => setExpenseAmount(e.target.value)}
placeholder="Expense amount"
className="small-input"
/>
<input
type="text"
value={expenseCategory}
onChange={(e) => setExpenseCategory(e.target.value)}
placeholder="Expense category"
className="small-input"
/>
<input
type="date"

value={expenseDate}
onChange={(e) => setExpenseDate(e.target.value)}
placeholder="Expense date"
className="small-input"
/>
<br></br>
<div className='but'>
<button onClick={handleExpenseSubmit}>
{editIndex !== null ? 'Update Expense' : 'Add Expense'}
</button>
</div>
```

```
</div>
<div style={{ marginBottom: '20px' }}>
<h2 className='exa'>Expenses</h2>
<ul>
{expenses.map((expense, index) => (
<li key={index}>
<span>
{expense.name} (${expense.amount.toFixed(2)}) - {expense.category} on
{expense.date}
</span>
<div>
<button onClick={() => editExpense(index)} style={{ marginLeft: '30px' }}>
Edit
</button>
<button onClick={() => removeExpense(index)}>
Remove
</button>
</div>
</li>
))}
</ul>
<button onClick={clearAllExpenses}>
Clear All Expenses
</button>
</div>
<div className="expensesummary">
<h2 className='exa'>Summary</h2>
<p>Total Income: ${parseFloat(income || 0).toFixed(2)}</p>
```

```
<p>Total Expenses: ${totalExpenses.toFixed(2)}</p>
<p>Balance: ${balance.toFixed(2)}</p>
</div>
</div>
</section>
);
};
export default ExpenseCalculator;
```

Analytics Page:

```
// src/pages/Analytics.js
import React from 'react';
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend, BarChart,
Bar, PieChart, Pie, Cell } from 'recharts';

import '../Analytics/Analytics.css';
import Navbar from '../../components/Navbar/Navbar';
const data = [
  { name: 'Jan', income: 4000, expense: 2400 },
  { name: 'Feb', income: 3000, expense: 1398 },
  { name: 'Mar', income: 2000, expense: 9800 },
  { name: 'Apr', income: 2780, expense: 3908 },
  { name: 'May', income: 1890, expense: 4800 },
{ name: 'Jun', income: 2390, expense: 3800 },
{ name: 'Jul', income: 3490, expense: 4300 },
];
const pieData = [
{ name: 'Food', value: 400 },
{ name: 'Transport', value: 300 },
{ name: 'Entertainment', value: 300 },
{ name: 'Healthcare', value: 200 },
];
const COLORS = ['#0088FE', '#00C49F', '#e4a10f', '#c2551f'];
const Analytics = () => {
return (
<div className="analytics-container">
<h2 className='Analytics-h2'>Analytics</h2>
<div className="chart-container">
<h2 className='Analytics-h2'>Income vs Expenses</h2>
<LineChart
width={600}
height={300}
data={data}
margin={{ top: 5, right: 30, left: 20, bottom: 5 }}
>
<CartesianGrid strokeDasharray="3 3" />
<XAxis dataKey="name" />
<YAxis />
<Tooltip />
<Legend />
<Line type="monotone" dataKey="income" stroke="#82ca9d" />
<Line type="monotone" dataKey="expense" stroke="#8884d8" />
</LineChart>
</div>
<div className="chart-container">
```

```
<h2 className='Analytics-h2'>Category Breakdown</h2>
<PieChart width={400} height={400}>
<Pie
data={pieData}
cx={200}
cy={200}
labelLine={false}
label={({ name, percent }) => `${name} ${(percent * 100).toFixed(0)}%`}
outerRadius={150}
fill="#35317b"
dataKey="value"
>
{pieData.map((entry, index) => (
<Cell key={`cell-${index}`} fill={COLORS[index % COLORS.length]} />
))}
</Pie>
<Tooltip />
</PieChart>
</div>
<div className="chart-container">
<h2 className='Analytics-h2'>Monthly Expenses</h2>
<BarChart
width={600}
height={300}
data={data}
margin={{ top: 5, right: 30, left: 20, bottom: 5 }}
<CartesianGrid strokeDasharray="3 3" />
<XAxis dataKey="name" />
```

```
<YAxis />
<Tooltip />
<Legend />
<Bar dataKey="expense" fill="#6a63e7" />
</BarChart>
<Navbar/>
</div>
</div>
};

export default Analytics;
```

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

This chapter tells about the conclusion that anyone can drive from the project andthe learning we learnt by taking over this project.

7.1.CONCLUSION

In conclusion, an expense tracker app plays a pivotal role in empowering users to take control of their finances. By offering detailed insights into spending habits, helping users set and achieve financial goals, and providing a user-friendly interface, the app can significantly enhance financial literacy and discipline. As the app evolves, integrating advanced features such as AI-powered recommendations, multi-currency support, and secure financial institution integration will further enhance its value. Future expansions into areas like bill management, investment tracking, and community features can transform the app from a simple expense tracker into a comprehensive financial management tool. Ultimately, this will enable users to make informed financial decisions, achieve their savings goals, and enjoy a more secure financial future.

7.2. FUTURE SCOPE

Advanced Analytics and Insights: Implement features that analyze spending patterns over time, offering insights into where users spend most of their money and how their

Spending the money.Use machine learning to predict future expenses based on past data, helping users plan .

Integration with Financial Institutions: Integrate with banks and credit card companies to automatically import transactions and balance information. Expand the app to include tracking of investments, showing users a full picture of their financial health.

AI-Powered Recommendations: Offer personalized tips on how to save more money based on user behavior Provide recommendations for investment opportunities, such as stocks, bonds, or savings accounts, based on the user's financial situation.

Enhanced User Experience: Integrate voice recognition to allow users to input data or generate reports using voice commands. Add elements of gamification, such as rewards for meeting savings goals, to motivate users. Allow multiple users to track expenses together, useful for families or small businesses.

Cross-Platform Accessibility: Ensure seamless synchronization between mobile and desktop versions of the app, allowing users to access their data from any device.Wearable Integration: Expand the app to work with wearable devices, providing quick access to expense tracking and notifications.

CHAPTER 8

REFRENCES

1.GeeksforGeeks, “Types of Online Expense Platforms,” https://www.geeksforgeeks.org/types-of-online-expense-platforms/

2. Demirkan, H., & Delen, D. (2013). *Data, information, and analytics as a competitive advantage*. *Business Horizons*, 56(5), 491-499. doi:10.1016/j.bushor.2013.05.003

3.Kettunen, P. (2019). *Digital expense tracking: A case study of fintech innovations*. *Journal of Financial Technology*, 2(1), 34-49. doi:10.1016/j.jfinte.2018.11.002

4.Statista. (2024). *Expense Tracker Market Size & Share - Global Report 2024*. Retrieved from https://www.statista.com/statistics/624345/expense-tracker-market-size/

5. Hargreaves, R. (2021). *How to Build a Personal Finance App: A Comprehensive Guide*. *TechCrunch*. Retrieved from https://techcrunch.com/2021/06/15/how-to-build-a-personal-finance-app/