Distributed and Parallel System Mid-Term Project

Group	:	4
Name / Student ID	:	Alysa Milano (001202300089) Amanda Putri (001202300078)
Project Title	:	CashTracker (Budgeting/Expenses Website)
Class/Batch	:	IT-2/2023
Lecturer	:	Ahmad Fadhil N

INFORMATICS STUDY PROGRAM
FACULTY OF COMPUTER SCIENCE

PRESIDENT UNIVERSITY

Tables of Content

1. Introduction

Managing personal finances is a common challenge, especially for students and young adults who are just beginning to take control of their spending habits. Many individuals in this stage of life are learning how to handle their own money for the first time, which can be both exciting and daunting. To address this issue, this mid-term project presents the development of a simple budgeting web application. The main objective of this project is to provide users with an easy-to-use tool that helps them track their daily income and expenses, understand their financial behavior, and maintain a balanced budget.

This budgeting application is built as a lightweight, web-based solution that emphasizes simplicity, clarity, and essential functionality. The application allows users to input transactions, view a complete list of their financial activity, and monitor their current balance. By allowing users to record each transaction, the application helps them stay aware of their financial situation on a day-to-day basis. Additionally, it provides a monthly summary that gives users a quick overview of their income and expenses within the current month. This feature enables users to see how much they have earned and spent over a given period, making it easier to identify trends and make informed decisions about their finances.

By focusing on essential features and avoiding overly complex elements, this project ensures a clean and intuitive user experience. The straightforward design makes it accessible for users who may not have prior experience with budgeting tools. It is developed using modern web technologies and structured in a way that supports future scalability and enhancement. As a result, the application not only meets the immediate needs of users but also has the potential to be expanded or improved in the future as requirements evolve.

Overall, this project aims to help students and young adults develop better financial habits by providing a practical and accessible budgeting tool. Through its focus on simplicity and essential features, the application serves as a valuable resource for anyone seeking to gain more control over their personal finances.

2. Features

CashTrack includes a range of essential features designed to help users record, categorize, and monitor their financial activities. Below is a breakdown of each feature along with supporting code excerpts that demonstrate their functionality in the system.

2.1 User Registration and Login (Session-Based Authentication)

The application supports user authentication using Flask-Session, which stores user login state securely in the server.

backend/app.py

```python
@app.route('/register', methods=['POST'])
def register():
    data = request.json
    email = data['email']
    username = data['username']
    password = hash_password(data['password'])

    cur = mysql.connection.cursor()
    cur.execute("SELECT id FROM users WHERE username=%s",(username,))
    if cur.fetchone():
        return jsonify({'message': 'Username already exists'}), 400

    cur.execute("INSERT INTO users (email, username, password_hash) VALUES (%s, %s, %s)",
            (email, username, password))
    mysql.connection.commit()
    cur.close()
    return jsonify({'message': 'Registration successful'})

@app.route('/login', methods=['POST'])
def login():
    data = request.json
    username = data['username']
    password = data['password']

    cur = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
    cur.execute("SELECT * FROM users WHERE username=%s", (username,))
    user = cur.fetchone()
    cur.close()

    if user and check_password(password, user['password_hash']):
        session['user_id'] = user['id']
        return jsonify({'message': 'Login successful'})
    else:
        return jsonify({'message': 'Invalid credentials'}), 401
```

backend/models.py

```
def hash_password(password):
  return hashlib.sha256(password.encode()).hexdigest()

def check_password(password, hashed):
  return hmac.compare_digest(hash_password(password), hashed)
```

- The session["user_id"] is used throughout the application to identify the authenticated user.
- User credentials are securely stored using hashed passwords via werkzeug.security.

2.2 Add Income or Expense Transactions

Users can add a financial transaction by specifying:

- Type (income or expense)
- Category (e.g., Food, Gift)
- Amount
- Description (optional)

backend/app.py

```
@app.route('/transactions', methods=['POST'])
def add_transaction():

  cur.execute('''
    INSERT INTO transactions (user_id, type, amount, category, description, date)
    VALUES (%s, %s, %s, %s, %s, %s)
  ''', (
    user_id,
    data['type'],
    int(data['amount']),
    data['category'],
    data.get('description', ''),
    now
  ))
```

frontend/src/pages/AddTransaction.js

```
const res = await fetch("http://localhost:4000/transactions", {
   method: "POST",
   headers: { "Content-Type": "application/json" },
   credentials: "include",
   body: JSON.stringify(form),
  });
```

2.3 Automatic Timestamping in Waktu Indonesia Barat (WIB)

Every transaction is stored with a timestamp based on Asia/Jakarta timezone using pytz. This ensures consistency across user activity.

backend/app.py

```
jakarta = pytz.timezone('Asia/Jakarta')
    now = datetime.now(jakarta)
```

This value is saved directly to the created_at column in the database.

2.4 Categorization of Transactions

Transactions can be grouped by user-defined or pre-defined categories such as: Food, Transport, Gift, Salary, etc.

backend/models.py

```
category VARCHAR(100),
```

frontend/src/pages/AddTransaction.js

```
const categories = form.type === "Income"
    ? ["Salary", "Gift", "Other"]
    : ["Food", "Transport", "Groceries", "Rent", "Bill", "Other"];
```

2.5 Real-Time Display of Transactions and Monthly Summary

After logging in, users can view:

- A table of all past transactions
- A summary of total income, expenses, and current balance for the selected month

frontend/src/pages/Summary.js

```
const totalIncome = transactions
    .filter((t) => t.type === 'Income')
    .reduce((acc, cur) => acc + Number(cur.amount), 0);

  const totalExpense = transactions
    .filter((t) => t.type === 'Expense')
    .reduce((acc, cur) => acc + Number(cur.amount), 0);
  const balance = totalIncome - totalExpense;
```

2.6 Transaction History

Shows a list of all user transactions with date, type, amount, category, and description.

frontend/src/pages/TransactionList.js

- **Fetch from backend**:

```
useEffect(() => {
    fetch('http://localhost:4000/transactions', {
      credentials: 'include',
    })
      .then((res) => res.json())
      .then((data) => {
        console.log("Dari backend:", data);
        setTransactions(data);
      })
```

- **Render table**:

```
transactions.map((transaction, index) => (
      <tr
       key={index}
       className={getRowClassName(transaction.type)}
      >
       <td>{formatDateTime(transaction.date)}</td>
       <td>{transaction.type}</td>
       <td>{formatRupiah(transaction.amount)}</td>
       <td>{transaction.category}</td>
       <td>{transaction.description}</td>
      </tr>
     ))
```

2.7 Edit and Delete Transactions

Users can edit or delete transactions directly in the table. The **Edit** feature includes dropdown categories based on type and formatted amount input (e.g., 1.000). All changes are saved to the database.

frontend/src/pages/TransactionList.js

```
<select
           value={editForm.type}
           onChange={(e) => {
            const newType = e.target.value;
            const newCategories = getCategories(newType);
            setEditForm((prev) => ({
             ...prev,
             type: newType,
             category: newCategories.includes(prev.category)
              ? prev.category
              : newCategories[0]
            }));
           }}
          >
           <option>Income</option>
           <option>Expense</option>
          </select>
         ) : (
          transaction.type
         )}
        </td>
        <td>
         {editingId === transaction.id ? (
          <input
           type="text"
           value={formatWithDots(editForm.amount)}
           onChange={(e) => {
            const raw = unformatDots(e.target.value);
            if (!isNaN(raw)) {
                  setEditForm({ ...editForm, amount: raw });
                }
              }}
            />
  const res = await fetch(`http://localhost:4000/transactions/${id}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    credentials: 'include',
    body: JSON.stringify({
      ...editForm,
      amount: unformatDots(editForm.amount)
    }),
  });
 const handleDelete = (id) => {
  if (window.confirm("Are you sure you want to delete this transaction?")) {
    fetch(`http://localhost:4000/transactions/${id}`, {
      method: "DELETE",
      credentials: "include",
    })
```

2.8 Logout Functionality
To securely end the session, users can log out. This removes the session data from the server.
backend/app.py

```
@app.route('/logout', methods=['GET'])
def logout():
  session.clear()
  return jsonify({'message': 'Logged out'})
```

LogoutButton.js
The frontend simply calls this endpoint to terminate the session.

```
fetch("http://localhost:4000/logout", {
     credentials: "include",
  })
    .then(() => {
      window.location.href = "/login";
    })
```

3. Tools & Technologies

CashTrack is built using a modern and efficient technology stack that separates the frontend, backend, and database, and is containerized for portability and ease of deployment.

3.1 Frontend: React.js

The frontend is built with React, using react-router-dom for routing and fetch() to call backend APIs. It handles login state, navigation, and renders components like login, register, dashboard, and transaction views.

frontend/src/App.js

```
  fetch("http://localhost:4000/check", {
```

3.2 Backend: Flask (Python)

The backend uses Flask for API endpoints, session management, and MySQL integration. Passwords are securely hashed, and sessions are stored server-side using Flask-Session.

backend/app.py

```
@app.route('/register', methods=['POST'])

@app.route('/login', methods=['POST'])
@app.route('/transactions', methods=['POST'])
@app.route('/logout', methods=['GET'])
```

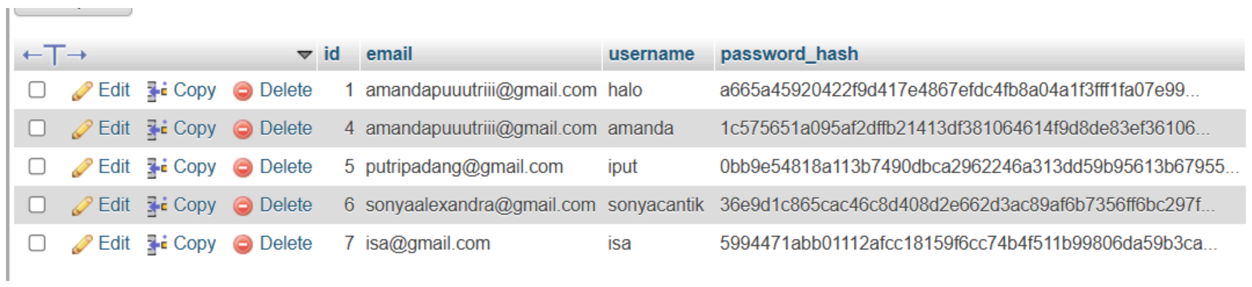
3.3 Database: MySQL

Transaction and user data are stored in a MySQL database. Tables are created programmatically on server start if they don't exist.

backend/models.py

- Users:

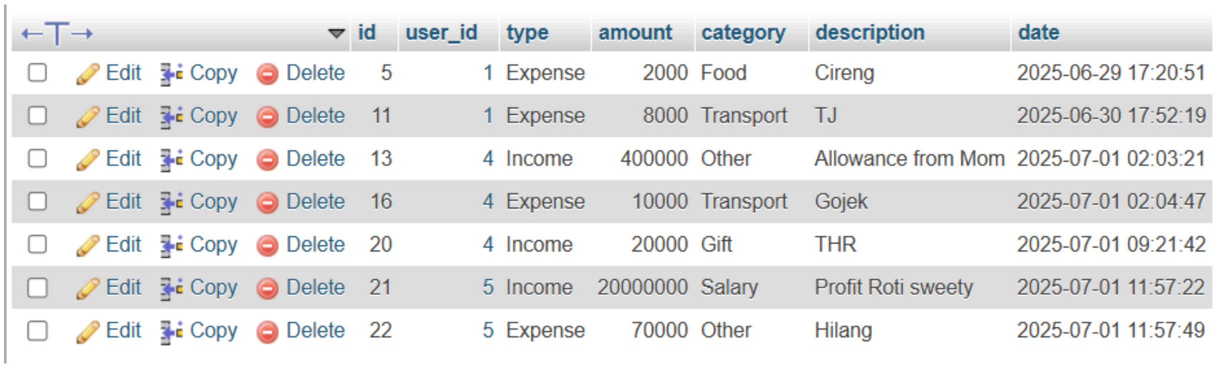
```
def create_user_table(cur):
  cur.execute('''
    CREATE TABLE IF NOT EXISTS users (
      id INT AUTO_INCREMENT PRIMARY KEY,
      email VARCHAR(255),
      username VARCHAR(100) UNIQUE,
      password_hash VARCHAR(255)
    )
  ''')
```



				id	email	username	password_hash
☐	Edit	Copy	Delete	1	amandapuuutriii@gmail.com	halo	a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e99...
☐	Edit	Copy	Delete	4	amandapuuutriii@gmail.com	amanda	1c575651a095af2dffb21413df381064614f9d8de83ef36106...
☐	Edit	Copy	Delete	5	putripadang@gmail.com	iput	0bb9e54818a113b7490dbca2962246a313dd59b95613b67955...
☐	Edit	Copy	Delete	6	sonyaalexandra@gmail.com	sonyacantik	36e9d1c865cac46c8d408d2e662d3ac89af6b7356ff6bc297f...
☐	Edit	Copy	Delete	7	isa@gmail.com	isa	5994471abb01112afcc18159f6cc74b4f511b99806da59b3ca...

- Transactions:

```
def create_transaction_table(cur):
  cur.execute('''
    CREATE TABLE IF NOT EXISTS transactions (
      id INT AUTO_INCREMENT PRIMARY KEY,
      user_id INT,
      type ENUM('Income', 'Expense'),
      amount INT,
      category VARCHAR(100),
      description TEXT,
      date DATETIME,
      FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
    )
  ''')
```



←T→	id	user_id	type	amount	category	description	date
☐ Edit Copy Delete	5	1	Expense	2000	Food	Cireng	2025-06-29 17:20:51
☐ Edit Copy Delete	11	1	Expense	8000	Transport	TJ	2025-06-30 17:52:19
☐ Edit Copy Delete	13	4	Income	400000	Other	Allowance from Mom	2025-07-01 02:03:21
☐ Edit Copy Delete	16	4	Expense	10000	Transport	Gojek	2025-07-01 02:04:47
☐ Edit Copy Delete	20	4	Income	20000	Gift	THR	2025-07-01 09:21:42
☐ Edit Copy Delete	21	5	Income	20000000	Salary	Profit Roti sweety	2025-07-01 11:57:22
☐ Edit Copy Delete	22	5	Expense	70000	Other	Hilang	2025-07-01 11:57:49

3.4 API: RESTful Endpoints

The app communicates between frontend and backend through RESTful API endpoints with JSON in **app.py**. Authentication is session-based and supports GET and POST routes like:

- **POST /login – Login**
- **GET /transactions – Get all transactions**
- **POST /transactions – Add transaction**
- **GET /logout – Logout**

3.5 Docker : Containerization

The project uses Docker for consistent development and deployment across systems. The Python backend is containerized using a Dockerfile and run using docker-compose.

backend/Dockerfile

```
FROM python:3.11-slim

RUN apt-get update && \
  apt-get install -y --no-install-recommends gcc python3-dev default-libmysqlclient-dev pkg-config && \
  rm -rf /var/lib/apt/lists/*

```

```
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 4000
CMD ["python", "app.py"]
```

frontend/Dockerfile

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

EXPOSE 3000

CMD ["npx", "serve", "-s", "build"]
```

docker-compose.yml

```
version: '3'
services:
 frontend:
  build: ./frontend
  ports:
   - "3000:3000"
  depends_on:
   - backend

 backend:
  build: ./backend
  ports:
   - "4000:4000"
```

3.5.1 Deployment using Docker

To simplify development and ensure consistent behavior across different machines, the CashTracker application is fully containerized using Docker and deployed using Docker Compose.

To deploy and run the application locally:

1. **Build the Containers**

This command will build the frontend and backend containers based on their respective Dockerfiles.

```
docker-compose build
```

2. **Start the Services**

   Docker Compose will start all the defined services: frontend, backend, and any dependencies like the database.

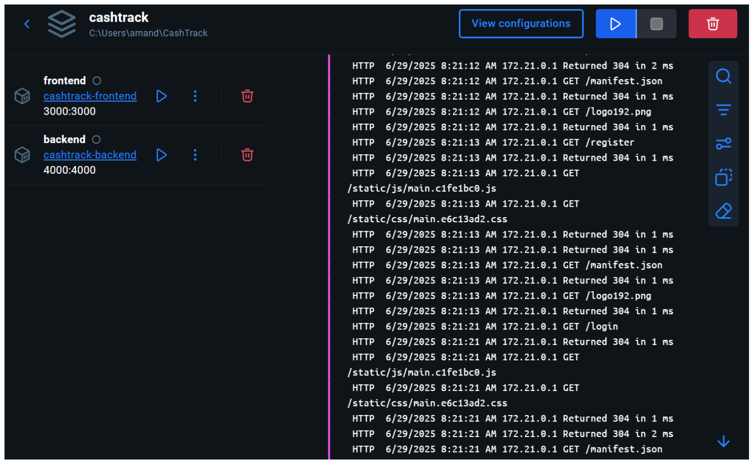
   docker-compose up

3. **Access the Application:**
   - Frontend: http://localhost:3000
   - Backend API: http://localhost:4000

4. **Stop the Application**

   docker-compose down

   This setup allows anyone to run the full application in one command without manually installing Node.js, Python, or MySQL. It encapsulates the entire environment into reproducible containers.



```
 HTTP  6/29/2025 8:21:12 AM 172.21.0.1 Returned 304 in 2 ms
 HTTP  6/29/2025 8:21:12 AM 172.21.0.1 GET /manifest.json
 HTTP  6/29/2025 8:21:12 AM 172.21.0.1 Returned 304 in 1 ms
 HTTP  6/29/2025 8:21:12 AM 172.21.0.1 GET /logo192.png
 HTTP  6/29/2025 8:21:12 AM 172.21.0.1 Returned 304 in 1 ms
 HTTP  6/29/2025 8:21:13 AM 172.21.0.1 GET /register
 HTTP  6/29/2025 8:21:13 AM 172.21.0.1 Returned 304 in 1 ms
 HTTP  6/29/2025 8:21:13 AM 172.21.0.1 GET
/static/js/main.c1fe1bc0.js
 HTTP  6/29/2025 8:21:13 AM 172.21.0.1 GET
/static/css/main.e6c13ad2.css
 HTTP  6/29/2025 8:21:13 AM 172.21.0.1 Returned 304 in 1 ms
 HTTP  6/29/2025 8:21:13 AM 172.21.0.1 Returned 304 in 1 ms
 HTTP  6/29/2025 8:21:13 AM 172.21.0.1 GET /manifest.json
 HTTP  6/29/2025 8:21:13 AM 172.21.0.1 Returned 304 in 1 ms
 HTTP  6/29/2025 8:21:13 AM 172.21.0.1 GET /logo192.png
 HTTP  6/29/2025 8:21:13 AM 172.21.0.1 Returned 304 in 1 ms
 HTTP  6/29/2025 8:21:21 AM 172.21.0.1 GET /login
 HTTP  6/29/2025 8:21:21 AM 172.21.0.1 Returned 304 in 1 ms
 HTTP  6/29/2025 8:21:21 AM 172.21.0.1 GET
/static/js/main.c1fe1bc0.js
 HTTP  6/29/2025 8:21:21 AM 172.21.0.1 GET
/static/css/main.e6c13ad2.css
 HTTP  6/29/2025 8:21:21 AM 172.21.0.1 Returned 304 in 1 ms
 HTTP  6/29/2025 8:21:21 AM 172.21.0.1 Returned 304 in 2 ms
 HTTP  6/29/2025 8:21:21 AM 172.21.0.1 GET /manifest.json
```

4. System Overview

CashTrack follows a classic full-stack web architecture based on the client-server model. It consists of three main layers:

1. **Frontend (React.js)**
2. **Backend (Flask - Python)**
3. **Database (MySQL)**

Each part plays a distinct role and communicates primarily through HTTP (REST API) requests and responses.

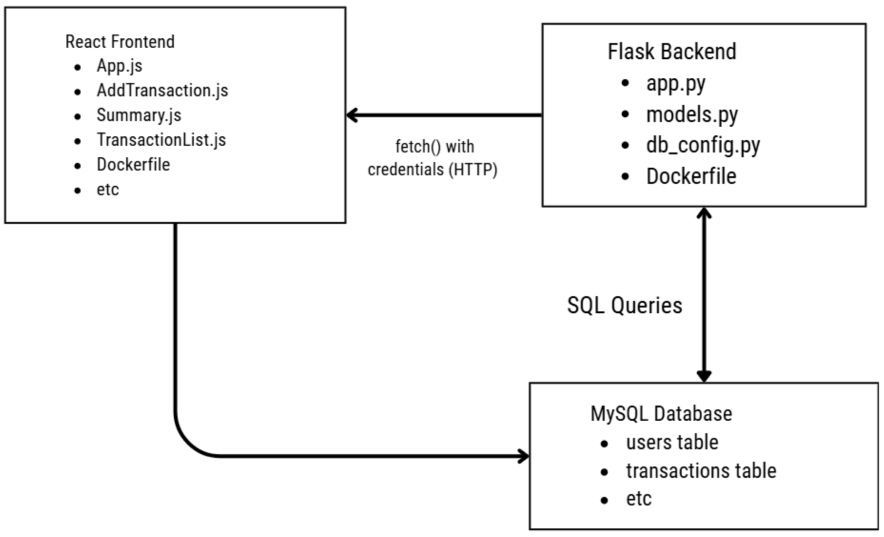
4.1 User Flow:

1. **Authentication**
    - When a user logs in or registers, the frontend sends a POST request to the backend (/login or /register) using fetch() with credentials: 'include'.
    - Flask checks the user credentials, and if valid, stores the user's session using Flask-Session.

2. **Session Management**
    - Session data (user ID) is stored on the server.
    - Every future request from the frontend (e.g., GET /transactions) includes the session cookie to verify the user.

3. **Data Handling**
    - For viewing transactions, React sends GET /transactions.
    - For adding a new transaction, React sends POST /transactions with the transaction data.
    - Flask routes receive the request, validate the session, and interact with the MySQL database to retrieve or store data.

4. **Frontend Display**
    - Once the backend responds with JSON, React parses and renders the data in the UI components: transaction history, monthly summary, etc.



5. Screenshots / UI Preview

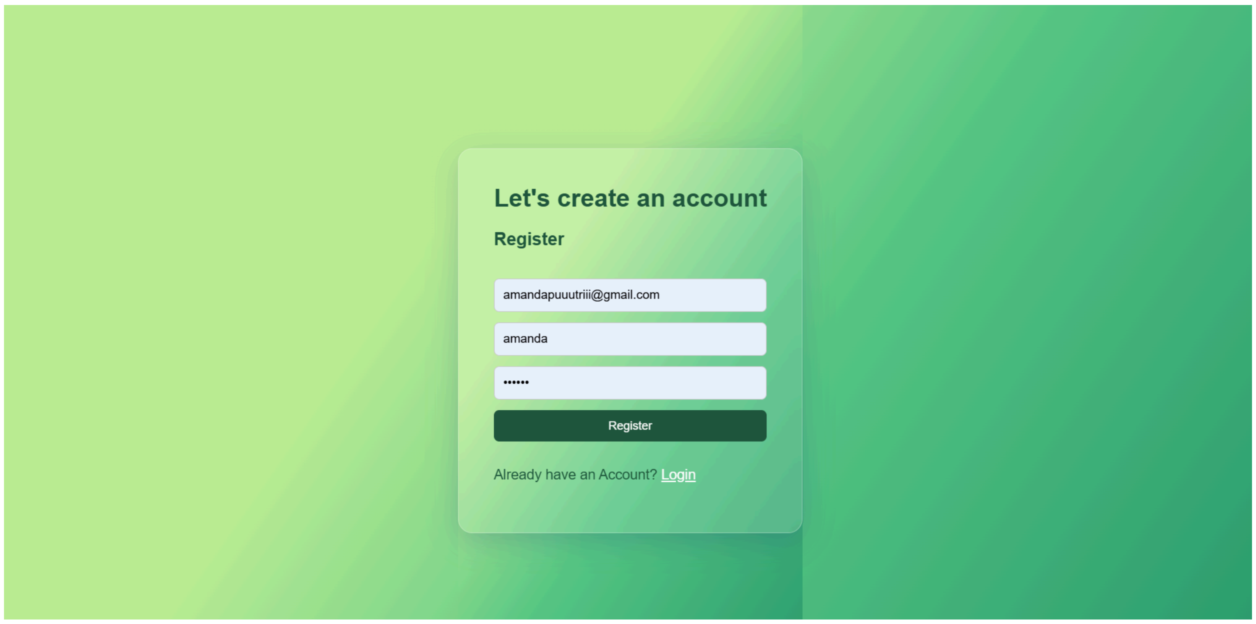
CashTracker features a clean and intuitive user interface (UI) that prioritizes user experience and accessibility. The design philosophy revolves around providing a seamless budgeting experience without overwhelming users with unnecessary complexity. Each page of the application is purposefully structured to guide users through their financial tracking journey.

5.1 Register Page

The registration page serves as the entry point for new users to create an account. It features a simple form where users can input:

- **Email address**
- **Username**
- **Password**

The form includes basic validation such as checking for empty fields and ensuring the username isn't already taken. Error messages are displayed if the input fails validation. This approach ensures a smooth onboarding experience for first-time users.



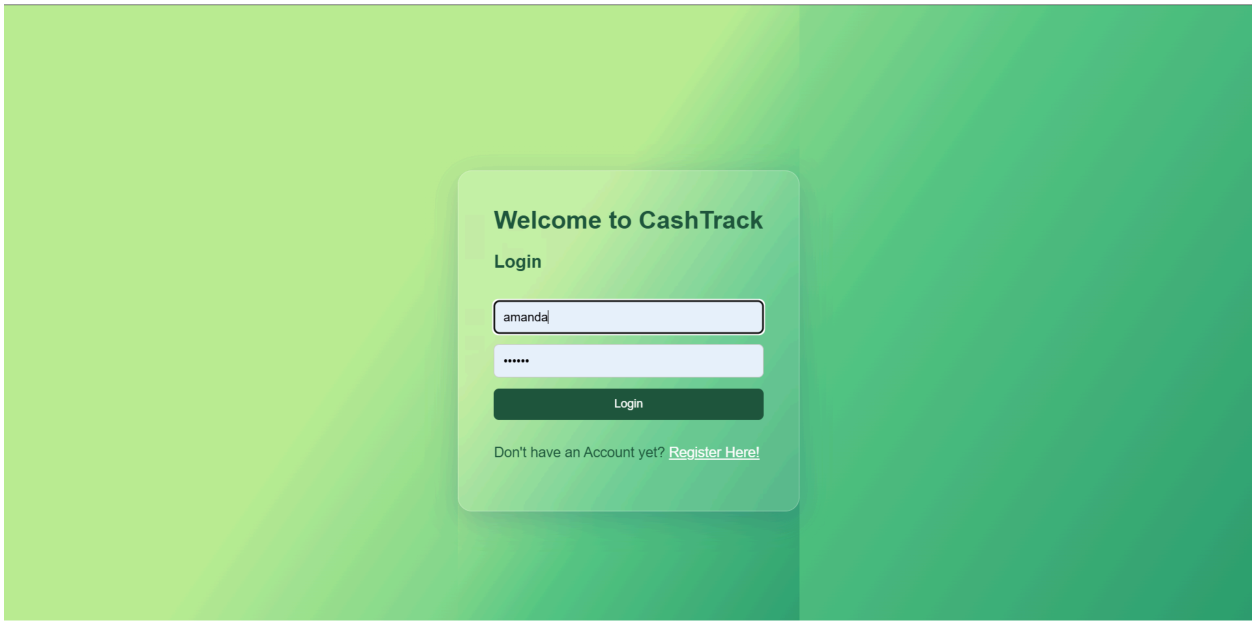
amandapuuutriii@gmail.com

amanda

••••••

5.2 Login Page

The login page allows returning users to access their account using their registered username and password. The credentials are validated by the backend using hashed passwords. When login is successful, the session is initiated and stored securely using Flask-Session. Invalid login attempts are handled gracefully with a clear error message.



5.3 Dashboard Page

The Dashboard is the central hub of the CashTracker application. All major user interactions take place on this single page, which is designed to be simple, informative, and highly interactive. By consolidating multiple functions into one cohesive interface, users can easily track and manage their finances without having to navigate through multiple pages.

5.3.1 Monthly Financial Summary

At the top of the Dashboard, users are presented with a real-time summary of their financial status for the current month. This includes:

- **Total Income**: The sum of all income transactions.
- **Total Expenses**: The total of all recorded expenses.
- **Balance**: The difference between income and expenses.

These values are automatically updated whenever a transaction is added, edited, or deleted. The summary provides users with immediate insight into their financial standing, helping them make smarter budgeting decisions.

5.3.2 Logout Feature

At the top or side of the Dashboard (depending on layout), a Logout button is provided. When clicked, it sends a request to the backend to clear the user's session, ensuring security. The user is then redirected to the login screen.

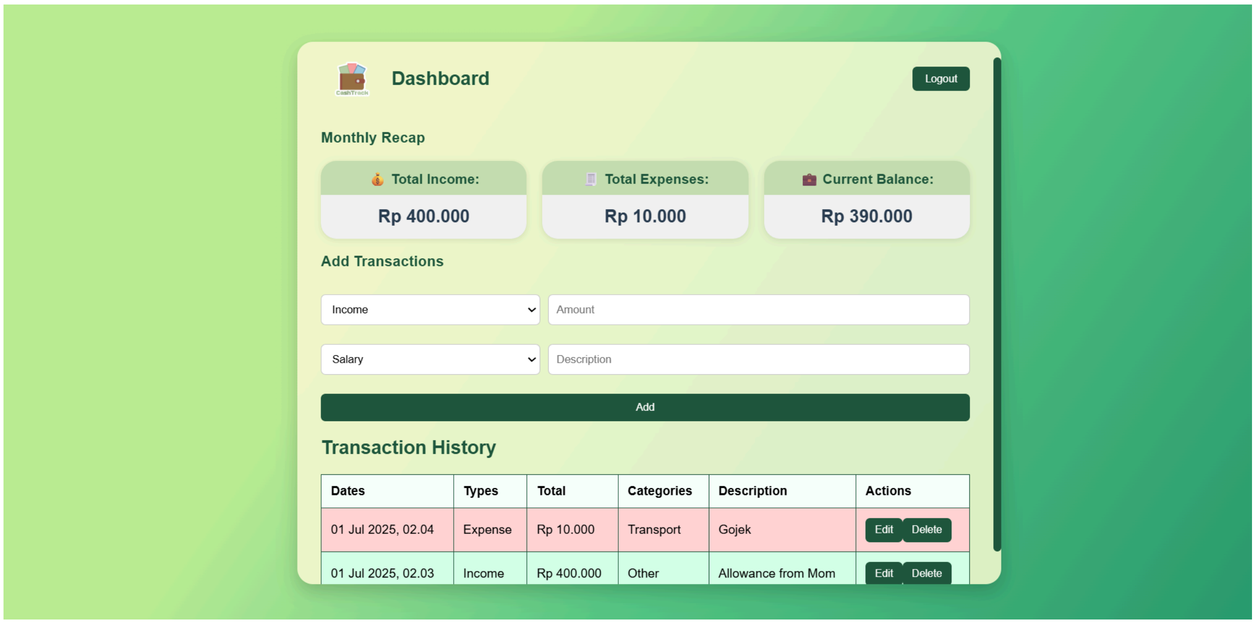
5.3.3 Monthly Financial Summary

At the top of the Dashboard, users are presented with a real-time summary of their financial status for the current month. This includes:

- **Total Income**: The sum of all income transactions.
- **Total Expenses**: The total of all recorded expenses.

- **Balance**: The difference between income and expenses

These values are automatically updated whenever a transaction is added, edited, or deleted. The summary provides users with immediate insight into their financial standing, helping them make smarter budgeting decisions.



Dashboard

Logout

Monthly Recap

Total Income:	Total Expenses:	Current Balance:
Rp 400.000	Rp 10.000	Rp 390.000

Add Transactions

Income | Amount

Salary | Description

Add

Transaction History

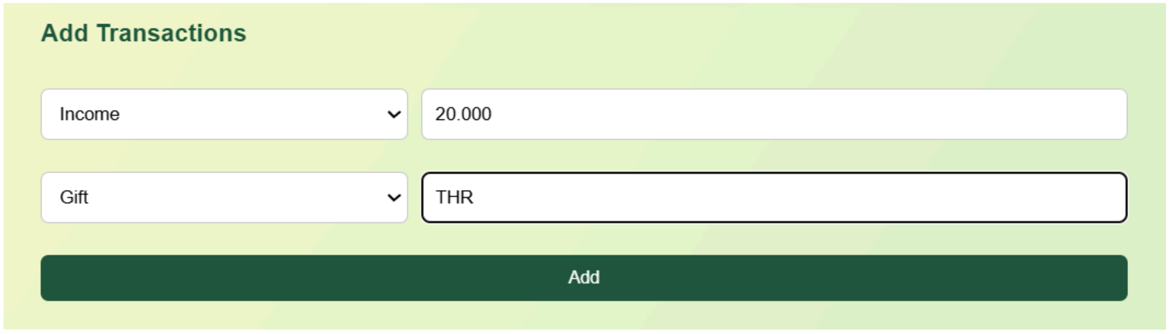
Dates	Types	Total	Categories	Description	Actions
01 Jul 2025, 02.04	Expense	Rp 10.000	Transport	Gojek	Edit Delete
01 Jul 2025, 02.03	Income	Rp 400.000	Other	Allowance from Mom	Edit Delete

5.3.4 Add New Transaction

Below the financial summary is a form where users can input new transactions. This form includes:

- **Transaction Type**: A dropdown to choose either "Income" or "Expense".
- **Category**: Populated based on the selected type. For example:
  - *Income: Salary, Gift, Other*
  - *Expense*: Food, Transport, Rent, Bills, Groceries, etc.
- **Amount**: A numeric input formatted with dots for readability (e.g., 1.000 instead of 1000).
- **Description**: An optional text field for additional notes.

Each transaction is also automatically timestamped using the WIB (Asia/Jakarta) timezone. When submitted, the form sends the data to the backend and immediately reflects it in the Dashboard.



Add Transactions

Income | 20.000

Gift | THR

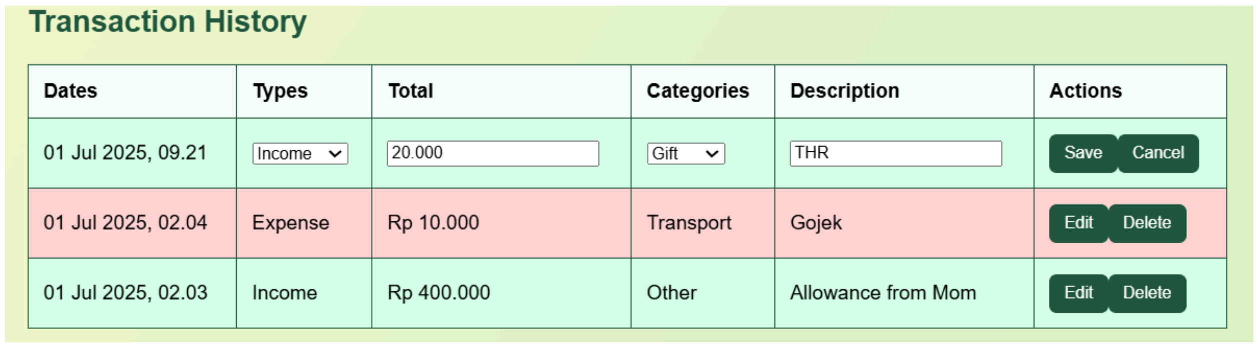
Add

5.3.5 Transaction History Table

Under the input form, all transactions are listed in a table. Each row in the table includes:

- **Date and Time** (formatted)
- **Transaction Type**: Clearly marked as either Income or Expense.
- **Amount**: Displayed in Indonesian Rupiah with proper formatting.
- **Category**: Based on the user's selection.
- **Description**: Optional user-provided note.

The transaction list is dynamically fetched from the backend and re-rendered each time there's a new or updated entry, providing real-time visibility of all financial activities.



Transaction History

Dates	Types	Total	Categories	Description	Actions
01 Jul 2025, 09.21	Income	20.000	Gift	THR	Save Cancel
01 Jul 2025, 02.04	Expense	Rp 10.000	Transport	Gojek	Edit Delete
01 Jul 2025, 02.03	Income	Rp 400.000	Other	Allowance from Mom	Edit Delete

The user interface (UI) of CashTracker is designed with simplicity, clarity, and usability in mind, making it accessible for users of all technical backgrounds. The experience begins with a clean and intuitive registration and login page, allowing users to securely create an account or log in using a straightforward form. Once authenticated, users are directed to a centralized Dashboard page, which serves as the main interface for managing their finances. Here, they can view a real-time summary of their monthly income, expenses, and balance, along with a transaction input form that dynamically adjusts categories based on the selected transaction type (income or expense). The transaction history is displayed in a structured table that lists all past records, complete with timestamps, formatted amounts, categories, and optional descriptions. Users can also edit or delete transactions directly from this table, with responsive forms and confirmation prompts ensuring ease of use and data safety. The UI is responsive, adjusting gracefully across devices, and includes a visible logout button that clears the session and returns users to the login screen. Overall, CashTracker's user interface is designed to be efficient, cohesive, and user-friendly, enabling individuals to monitor and control their financial habits within a single, well-organized environment.

6. Conclusion

Developing CashTracker has provided our team with valuable practical experience in building a full-stack web application using modern development tools and methodologies. This project enabled us to bridge theoretical concepts of distributed and parallel systems with real-world implementation. We explored client-server communication, session-based authentication, database design, and state management in a collaborative software environment.

6.1 Key learnings include:

- Frontend development with React.js, including routing, state management, and API integration.
- Backend development with Flask (Python) for creating RESTful APIs, handling user authentication, and connecting to MySQL databases.
- Database design and management using MySQL, including schema creation, relational modeling, and data querying.
- Session management and security, particularly with hashed passwords and Flask-Session to protect user data.
- Dockerization and deployment, using Docker and Docker Compose to containerize both the frontend and backend, making the application portable and deployment-ready.

This project also taught us how to work as a team, distribute responsibilities, and integrate different parts of a system effectively. We gained experience in debugging, version control, and writing clean, modular code that is easy to maintain.

6.2 Future improvements we would like to explore include:

- Implementing visual analytics such as pie charts and bar graphs to help users understand their spending patterns.
- Adding monthly budgeting goals and real-time alerts when spending nears or exceeds a threshold.
- Providing users with options to export their transaction history as PDF or Excel files.

Overall, CashTracker successfully fulfills its objective of providing a simple, accessible, and functional budgeting tool for students and young adults. It reflects our ability to build practical solutions using distributed systems principles and showcases our readiness to tackle more complex projects in the future.