Expense Tracker

Software Engineers:

- *Cody Strange – Software Engineer*

Abstract

The Expense Tracker is a full-stack web application developed using Flask, designed to help users manage and monitor their personal finances. It allows users to record daily expenses, categorize them for better organization, and review their spending patterns over time. Core features include adding expense entries, as well as viewing a list of all recorded transactions with category tags and timestamps. The application supports user authentication for multi-user support and includes a summary dashboard that visualizes expense data, such as category-wise spending charts.

The backend is powered by SQLAlchemy for database operations, while the frontend uses Bootstrap for responsive design and for rendering dynamic content. The project emphasizes clean code practices, modular structure, and ease of deployment using services like AWS Lambda. This project shows key skills in web development, database design, and full-cycle application deployment.

Organization

Coding Standards

Commenting

Functions

- Brief description of function
- Parameters
- Return Values

Classes

- Brief description of class

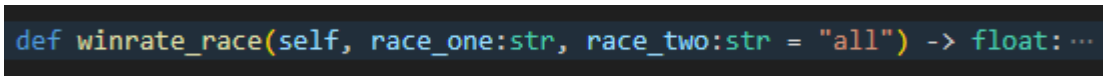
ChatGPT

- I highly recommend letting ChatGPT do most of the commenting
- Double check any comments by ChatGPT

Naming Conventions

- Classes: CapWords
- Functions: snake_case
- Variables: snake_case
- Constants: ALLCAPS
- Files/Folders: snake_case

Type Safety

- Every function should have the parameters types listed and the return type of the function listed
- 

Software Requirements

Programming Languages and Frameworks

- Python 3.12.0
- Flask
- Flask-WTF
- Flask-Login
- WTForms
- Bootstrap
- Chart.js
- Cryptography
- Email validator
- Setuptools

Database Management

- MySQL –
    - Root password=SQLrootpassword
    - User credentials password=SQLuserpassword
    - cody@localhoast=yourpassword
    - SQLAlchemy
- SQLite

Cloud Deployment and Infrastructure

- AWS CLI
    - Access Key = AKIAX464R6FHHIHZCT4D
    - Secret access key = dV2X+6Frh8g2x9dNLU/DzUWFU5gkmyP9ob4rXieo
    - Region name = us-west-2
    - Output format = json
- AWS RDS
    - Database identifier = expenseTracker
    - Master username = admin
    - Master Password = expensetracker
    - Account Id = 543239041358
    - Database name = paymentAPI
    - Database port = 3306
    - Database name = expenseTracker
    - Database endpoint = expensetracker.crm4oyaco2kc.us-west-2.rds.amazonaws.com
- AWS Lambda
- AWS API Gateway
- Zappa

Development Environment

- VS Code

Version Control

- Git
- Github

Virtual Environment

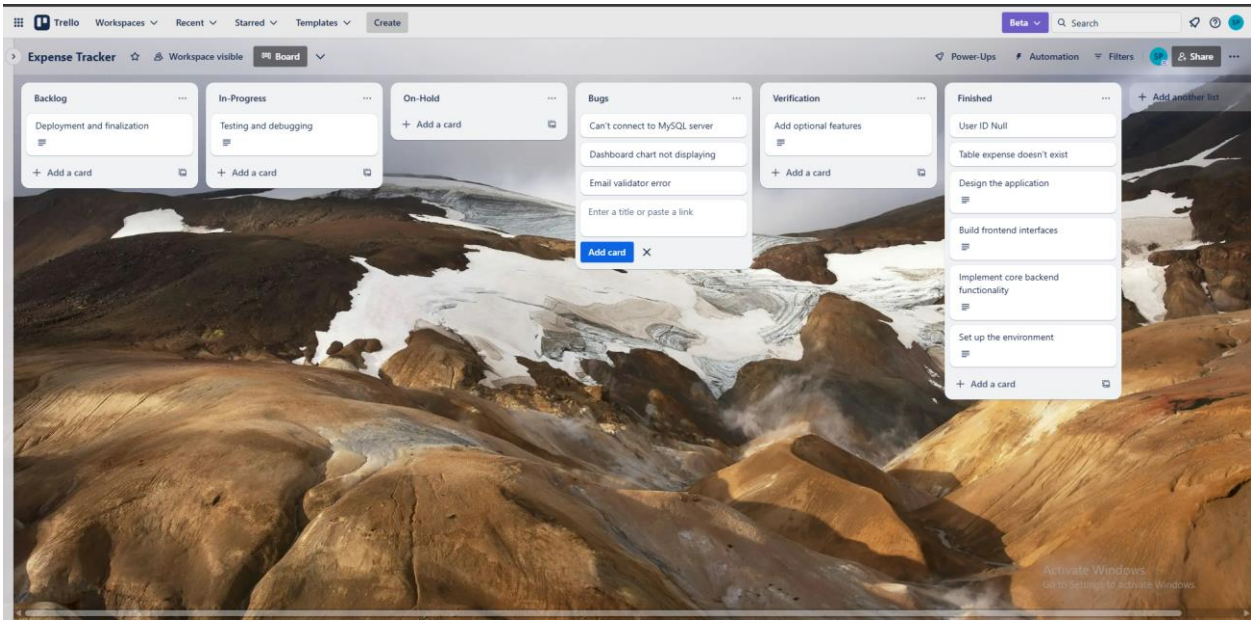
- Venv

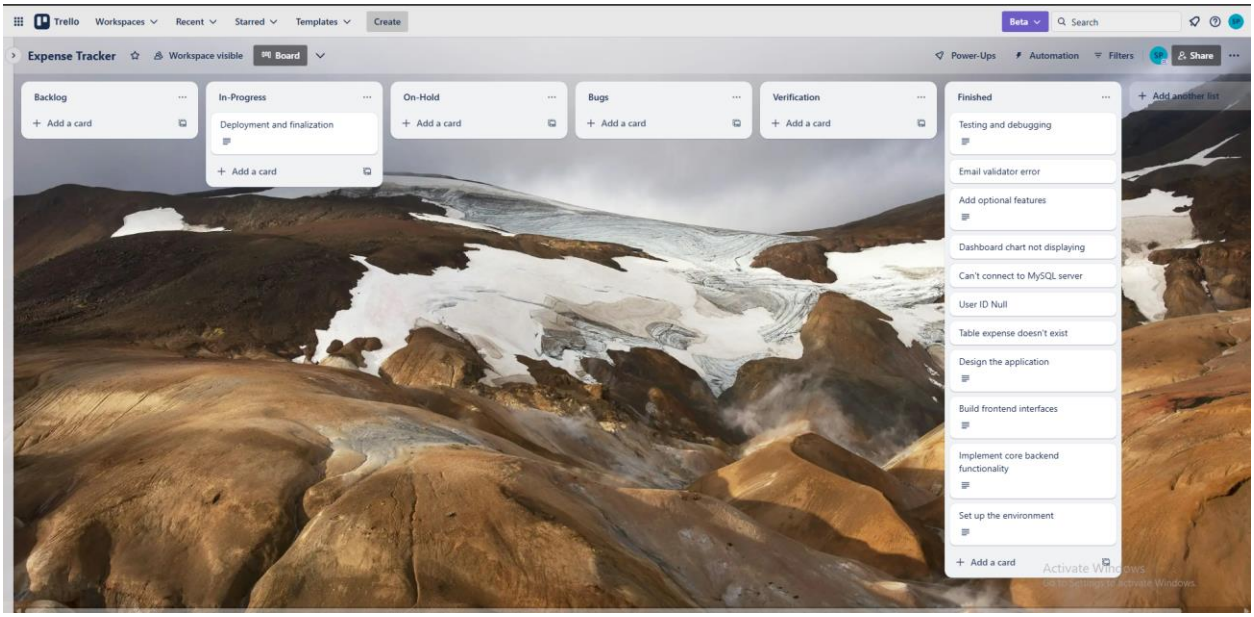
Additional Tools

- Lucidchart

Backlog

Description: This contains the tasks that we have completed each iteration as well as what we are currently working on, what known bugs exist, and what remains to be done. We add to the backlog as we discover new tasks but ignore them until we finish what we are working on.





Requirements Gathering

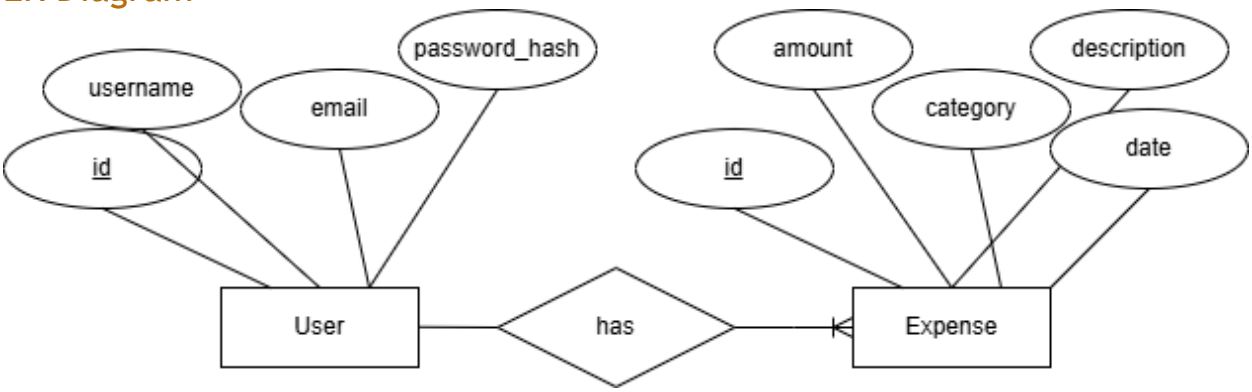
Functional Requirements

Description: These requirements are the general goals that our application should be able to meet.

- Add a new expense (with amount, category, date, and description)
  - Amount – The value of the expense (validated as a number)
  - Category – Chosen from a dropdown (e.g., Food, Transportation, Utilities)
  - Date – defaults to the current date if not selected
  - Description – text to add details (e.g., "Coffee at Starbucks")
- View all expenses in a list
- Show total and category-wise summaries with charts
  - A pie chart showing the proportion of total spending by category
- User authentication
  - User Registration: New users can sign up with a unique username, email, and password. Passwords are hashed before storing using Werkzeug.security.
  - Login/Logout: Users can log in to access their dashboard and securely log out when done.

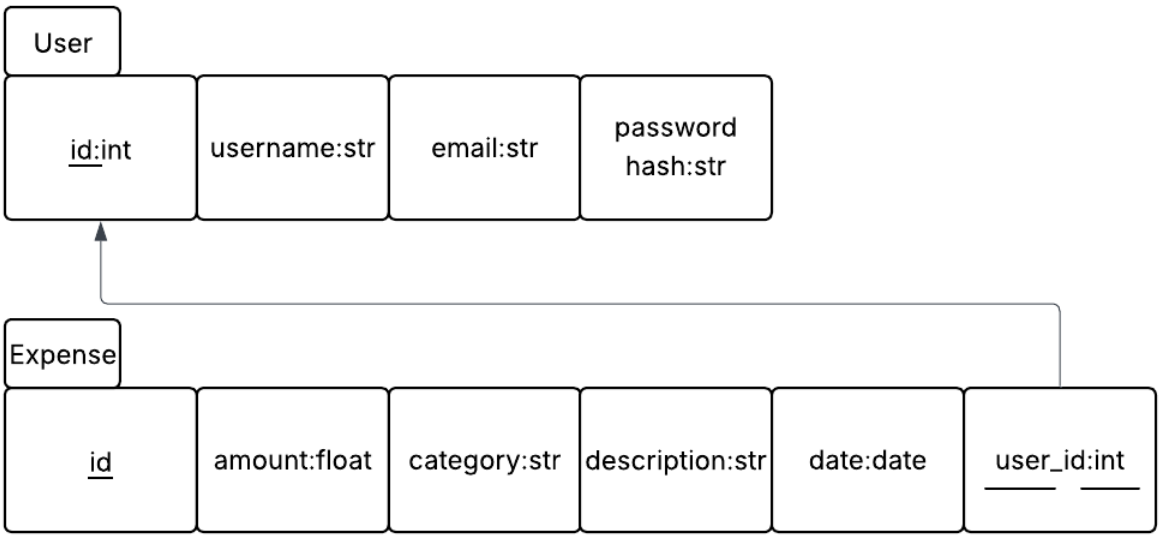
Design

ER Diagram



RTS Analyzer

Schema



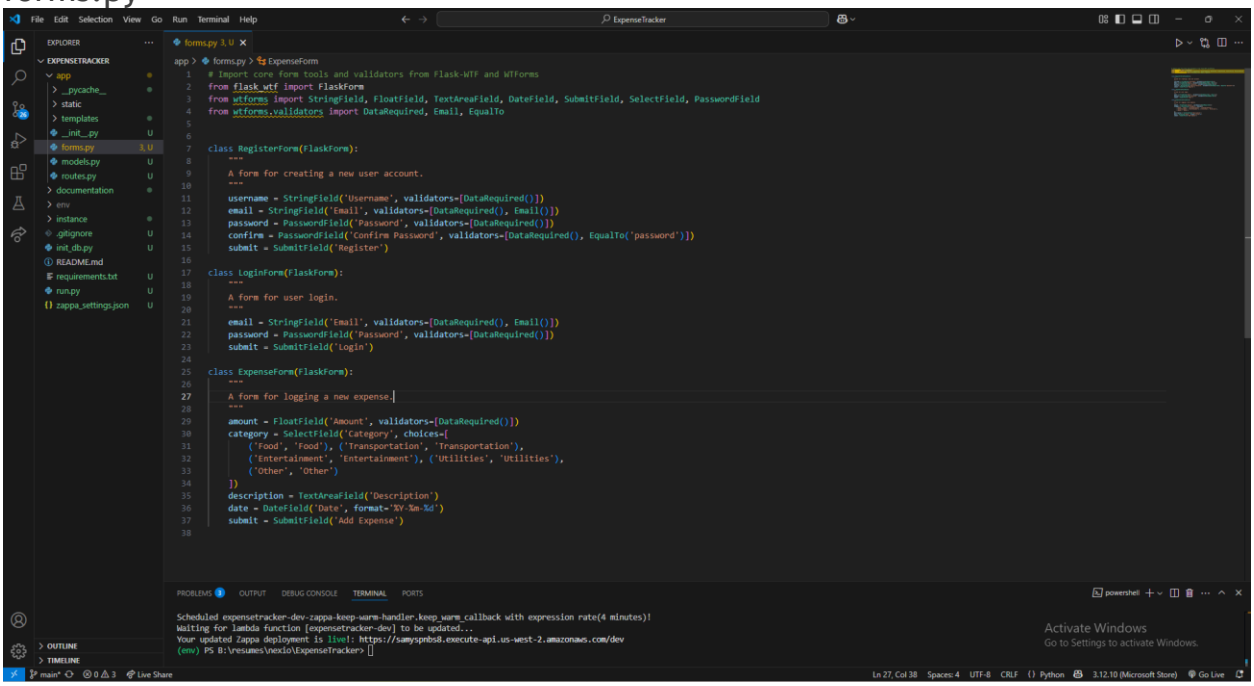
Categories List

- Food
- Transportation Entertainment
- Utilities
- Other

Development

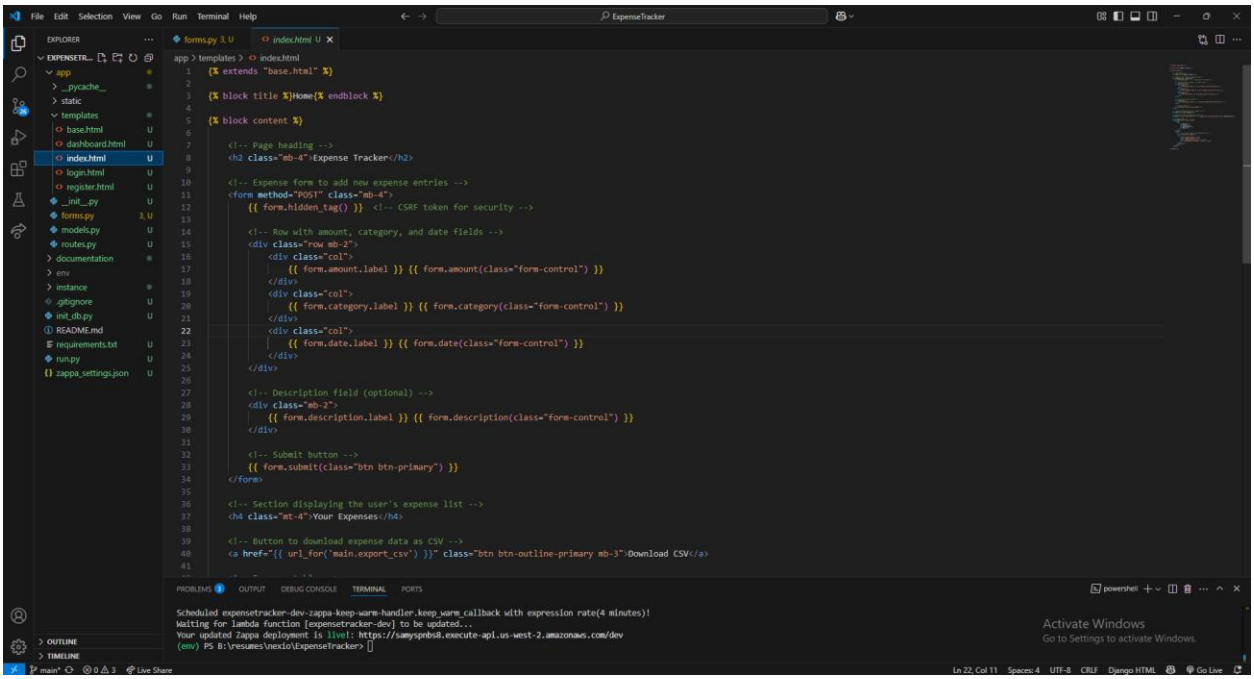
Code Snippets

forms.py



```python
# Import core form tools and validators from Flask-WTF and WTForms
from flask_wtf import FlaskForm
from wtforms import StringField, FloatField, TextAreaField, DateField, SubmitField, SelectField, PasswordField
from wtforms.validators import DataRequired, Email, EqualTo


class RegisterForm(FlaskForm):
    """
    A form for creating a new user account.
    """
    username = StringField('Username', validators=[DataRequired()])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm = PasswordField('Confirm Password', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Register')

class LoginForm(FlaskForm):
    """
    A form for user login.
    """
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')

class ExpenseForm(FlaskForm):
    """
    A form for logging a new expense.
    """
    amount = FloatField('Amount', validators=[DataRequired()])
    category = SelectField('Category', choices=[
        ('Food', 'Food'), ('Transportation', 'Transportation'),
        ('Entertainment', 'Entertainment'), ('Utilities', 'Utilities'),
        ('Other', 'Other')
    ])
    description = TextAreaField('Description')
    date = DateField('Date', format='%Y-%m-%d')
    submit = SubmitField('Add Expense')
```

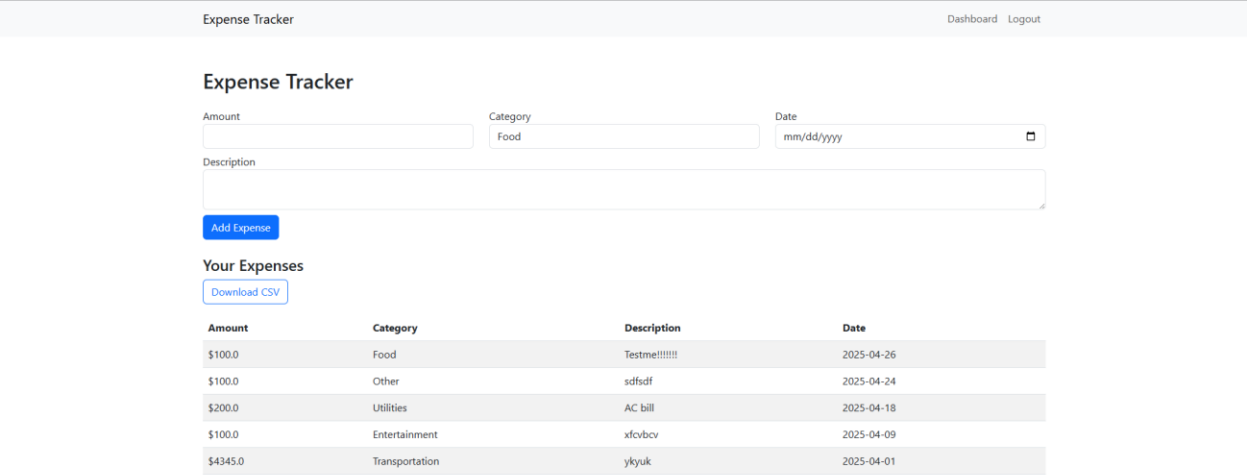
Index.html



```html
{% extends "base.html" %}

{% block title %}Home{% endblock %}

{% block content %}

    <!-- Page heading -->
    <h2 class="mb-4">Expense Tracker</h2>

    <!-- Expense form to add new expense entries -->
    <form method="POST" class="mb-4">
        {{ form.hidden_tag() }}  <!-- CSRF token for security -->

        <!-- Row with amount, category, and date fields -->
        <div class="row mb-2">
            <div class="col">
                {{ form.amount.label }} {{ form.amount(class="form-control") }}
            </div>
            <div class="col">
                {{ form.category.label }} {{ form.category(class="form-control") }}
            </div>
            <div class="col">
                {{ form.date.label }} {{ form.date(class="form-control") }}
            </div>
        </div>

        <!-- Description field (optional) -->
        <div class="mb-2">
            {{ form.description.label }} {{ form.description(class="form-control") }}
        </div>

        <!-- Submit button -->
        {{ form.submit(class="btn btn-primary") }}
    </form>

    <!-- Section displaying the user's expense list -->
    <h4 class="mt-4">Your Expenses</h4>

    <!-- Button to download expense data as CSV -->
    <a href="{{ url_for('main.export_csv') }}" class="btn btn-outline-primary mb-3">Download CSV</a>
```

RTS Analyzer

Expense Tracker UI



https://samyspnbs8.execute-api.us-west-2.amazonaws.com/dev/

Expense Tracker Summary

Review

Overview

This project was a deep dive into full-stack backend development, deploying a serverless payment API using Flask, AWS Lambda (via Zappa), and a MySQL database hosted on AWS RDS. The goal was to create a production-ready, cloud-hosted API capable of processing transactions and exposing them through a paginated history endpoint.

What I Learned

- How to build a modular Flask app.
- Implemented user authentication using Flask-Login and secure password hashing.
- Designed relational database models with user-expense relationships.
- Developed dynamic forms and integrated WTForms validation for clean data entry.
- Created CSV exports and integrated Chart.js for real-time data visualization.
- Deployed a serverless app using Zappa and connected to a MySQL RDS instance.

Struggles I Faced

- Database connection errors when deploying to Lambda:
    - Misconfigured VPC/subnet assignments
    - Missing or incorrect security group rules
    - Troubleshooting user_id NULL issues breaking data filters
- Script errors from missing dependencies, e.g., email-validator for registration
- Challenges with Chart.js not rendering due to script timing and data format issues

Successes

- Built a fully working cloud-hosted expense tracker with registration, login, form validation, and real-time analytics.
- Established a clean UI using Bootstrap and responsive design.
- Implemented user-specific expense tracking, ensuring proper data isolation and privacy.
- Generated dynamic, exportable CSVs and interactive spending summary charts.
- Documented the project thoroughly with a well-commented codebase, consistent form design, and a professional README.md.

Takeaways

This project strengthened my confidence in building full-stack applications and deploying them to the cloud using serverless architecture. I gained hands-on experience integrating many core components of real-world software: form handling, user auth, data visualization, CSV export, database relationships, and cloud deployment.

I'm now more confident in:

- Designing and deploying Flask applications with real users
- Navigating AWS services like RDS, Lambda, IAM, and VPC
- Writing and organizing code in a modular, scalable way
- Debugging live cloud applications and handling production-level issues

This project is a strong portfolio piece that demonstrates full-stack capability, cloud fluency, and real-world software architecture from the ground up.