SmartExpense Tracker Application Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive overview of the SmartExpense Tracker application's architecture. It serves as a guide for developers, designers, and stakeholders to understand the application's structure, components, and interactions.

1.2 Scope

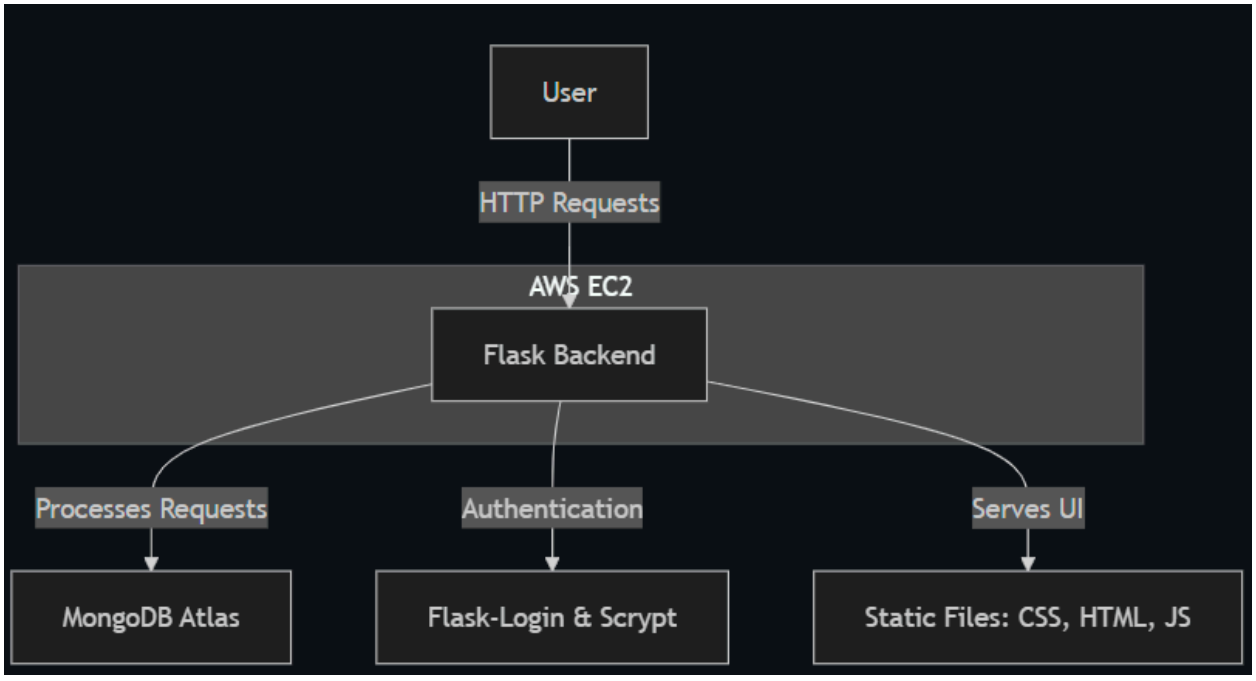
The scope of this document includes the architecture of the SmartExpense Tracker web application, detailing its components, data flow, technology stack, security measures, and deployment strategy.

1.3 Overview

SmartExpense Tracker is a web application designed to help users track their expenses, manage transactions, and categorize spending. The application will be deployed using AWS services to ensure scalability, reliability, and security.

2. Architecture Overview

2.1 High-Level Architecture Diagram



2.2 Architecture Style

The application follows a layered architecture style, with distinct layers for presentation, business logic, and data access.

3. Components and Modules

3.1 Presentation Layer

- **Templates:** HTML files (e.g., base.html) that define the structure of the web pages.
- **Static Files:** CSS, JavaScript, and image files used for styling and client-side functionality.

3.2 Business Logic Layer

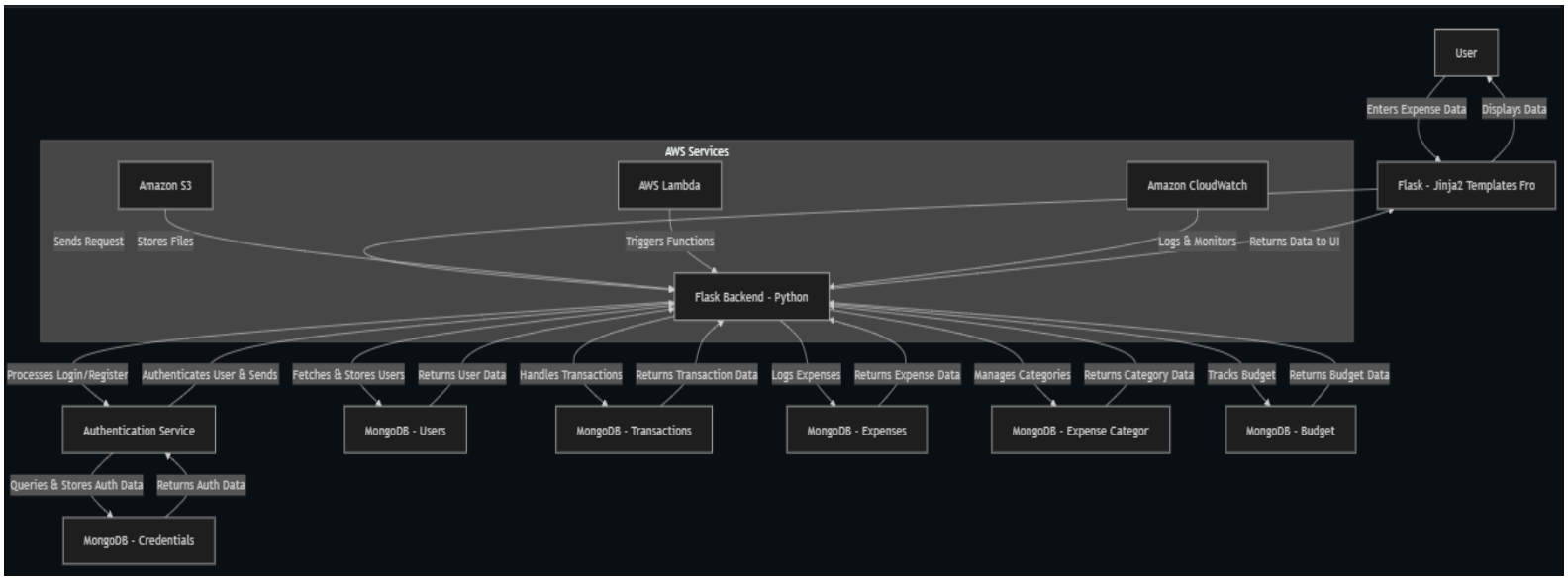
- **Views:** Python functions that handle HTTP requests and responses.
- **Forms:** Python classes that handle form validation and processing.

3.3 Data Access Layer

- **Models**: Python classes that define the structure of the database tables and handle database interactions.

4. Data Flow

4.1 Data Flow Diagrams



4.2 Data Movement

Data flows from the user's browser to the server via HTTP requests. The server processes the requests, interacts with the database, and returns the appropriate responses.

5. Technology Stack

5.1 Frontend

- HTML
- CSS
- JavaScript

5.2 Backend

- Python
- Flask (web framework)

5.3 Database

- MongoDB Atlas (Cloud Deployment)

5.4 Other Tools

- Jinja2 (templating engine)
- Flask-WTF (form handling)

6. Integration Points

6.1 External Systems (next sprint implementation)

- Email service for user notifications
- Payment gateway for handling transactions

6.2 APIs and Protocols

- RESTful APIs for communication between frontend and backend

7. Security

7.1 Security Measures (next sprint implementation)

- HTTPS for secure communication
- CSRF protection for forms

- Input validation and sanitization

7.2 Authentication and Authorization

- Flask-Login for user authentication
- Role-based access control for authorization

8. Deployment

8.1 Deployment Architecture

- Development environment: Local machine (pull git repo locally)
- Testing environment: Staging server
- Production environment: Cloud server AWS, MongoDB Atlas

8.2 Environments

- Development: MongoDB Atlas
- Testing: MongoDB Atlas
- Production: MongoDB Atlas

9. Scalability and Performance

9.1 Scalability Strategies (Future Implementation if needed)

- Horizontal scaling by adding more servers
- Load balancing to distribute traffic

9.2 Performance Considerations

- Caching frequently accessed data
- Optimizing database queries

10. Maintenance and Support

10.1 Maintenance Procedures

- Regular updates and patches
- Monitoring and logging for error detection

10.2 Support and Troubleshooting

- Documentation for common issues
- Support contact information

11. Appendices

11.1 Glossary of Terms

- **HTTP**: Hypertext Transfer Protocol
- **REST**: Representational State Transfer
- **CSRF**: Cross-Site Request Forgery

11.2 References and Additional Resources

- Flask Documentation: https://flask.palletsprojects.com/
- Jinja2 Documentation: https://jinja.palletsprojects.com/