# CCS 6440

# Project Overview

**What I Built**

- Microservices-based expense tracking application.
- **Two-tier architecture** with RESTful API and MariaDB database.
- **Complete CI/CD pipeline** with automated testing.
- **Containerized deployment** using Docker Compose.
- **Application monitoring** with Prometheus metrics.

**Key Technologies**

- **Backend:** Python Flask RESTful API.
- **Database:** MariaDB with normalized schema.
- **Containerization:** Docker & Docker Compose.
- **CI/CD:** GitHub Actions workflow.
- **Monitoring:** Prometheus metrics endpoint.

# System Architecture

## Architecture Components

- **Flask API Service**
  - RESTful endpoints for expense management.
  - User authentication and data validation.
  - JSON responses with proper error handling.
- **MariaDB Database**
  - Three main tables: users, expense_categories, expenses.
  - Foreign key relationships for data integrity.
  - Pre-populated with expense categories.
- **Docker Environment**
  - Multi-container setup with service isolation.
  - Health checks and automatic container recovery.
  - Volume mounting for data persistence.

# Core Functionality & API Endpoints

**Key Features Implemented**

- **User Management:** Registration and account handling.
- **Expense Tracking:** Create, update, delete expenses.
- **Category Filtering:** Filter expenses by type (Food, Entertainment, etc.).
- **Summary Reports:** Calculate totals and transaction counts.

**Main API Endpoints**

- POST /api/users - Create user accounts.
- GET /api/categories - List expense categories.
- POST /api/expenses - Add new expenses.
- GET /api/expenses/by_category/{id} - Filter by category.
- GET /api/expenses/summary/by_category - Generate summaries.
- GET /health - Health monitoring.
- GET /metrics - Prometheus metrics.

# Database Design & Implementation

## Database Schema

- **users:** Account information with secure password hashing.
- **expense_categories:** Predefined categories (Food, Transportation, etc.).
- **expenses:** Individual transactions with foreign key relationships.

## Data Integrity Features

- Foreign key constraints preventing orphaned records.
- Unique constraints on usernames and category names.
- Decimal precision for accurate monetary calculations.
- Timestamp tracking for audit purposes.

# Containerization & Git Workflow

## Docker Implementation

- **Multi-container setup** using Docker Compose.
- **Application container:** Python 3.10 with Flask app.
- **Database container:** MariaDB with initialization scripts.
- **Environment variables** for configuration management.

# CI/CD Pipeline & Testing

**GitHub Actions Workflow**

- **Automated testing** on every code push.
- **Dependency installation** and code quality checks.
- **Multi-branch support** for main and development branches.
- **Error reporting** and build status notifications.

**Comprehensive Test Suite**

- **6 automated tests** covering all major functionality.
- **API endpoint testing** for CRUD operations.
- **Integration tests** for database connectivity.
- **Health check validation** for service availability.
- **100% test pass rate** in final implementation.

# Monitoring & Live Demonstration

**Prometheus Metrics**

- **HTTP request metrics:** Request counts and response times.
- **Database operation metrics:** Query performance tracking.
- **Custom application metrics:** Business logic insights.
- **System metrics:** Memory and CPU utilization.

# Challenges Overcome & Solution

## Major Technical Challenges

- **Port Conflicts**
  - **Problem:** macOS using port 5000.
  - **Solution:** Configured Docker to use port 5001.
- **CI/CD Pipeline Issues**
  - **Problem:** YAML syntax errors in GitHub Actions.
  - **Solution:** Simplified workflow structure.
- **Database Connection Management**
  - **Problem:** Connection leaks under load.
  - **Solution:** Implemented SQLAlchemy connection pooling.
- **Test Environment Setup**
  - **Problem:** Tests failing without database.
  - **Solution:** Environment-specific configuration.

# Learning Outcomes & Skills Developed

**Technical Skills Gained**

- **Microservices Architecture:** Distributed system design.
- **API Development:** RESTful service implementation.
- **Database Design:** Relational modeling and optimization.
- **Containerization:** Docker and container orchestration.
- **CI/CD Automation:** GitHub Actions workflow creation.
- **Application Monitoring:** Metrics and observability.
- **Cloud Engineering Concepts**
- **Infrastructure as Code:** Configuration management.
- **DevOps Practices:** Automated testing and deployment.
- **Service Reliability:** Health checks and monitoring.
- **Version Control:** Professional Git work flow.

# Project Success & Conclusion

- **Completion Status**
- ✅ **Application:** Complete Flask microservice with all endpoints
  ✅ **Database:** MariaDB with proper schema and relationships
  ✅ **Containerization:** Docker multi-container deployment
  ✅ **CI/CD:** GitHub Actions automated pipeline
  ✅ **Testing:** 6/6 tests passing with full coverage
  ✅ **Monitoring:** Prometheus metrics implementation
  ✅ **Documentation:** Technical report and repository README

**Key Achievements**

- **Production-ready application** with proper error handling.
- **Professional development practices** with version control.
- **Modern deployment techniques** using containerization.
- **Operational monitoring** with health checks and metrics.

**Future Enhancements**

- JWT authentication system.
- Horizontal scaling with Kubernetes.
- Advanced analytics and reporting.
- Mobile application integration.