**📝 Project Overview: Order Checkout System**

A robust backend system simulating an e-commerce order & checkout flow using **Flask**, **PostgreSQL**, **Redis**, and **Celery**, all containerized with **Docker**. This document explains the project structure, components, and their roles with a focus on functionality.

**📁 Folder Structure**

order-checkout-system/

├── app/

│ ├── \_\_init\_\_.py # App factory and DB setup

│ ├── models.py # SQLAlchemy models (User, Item, CartItem, Checkout)

│ ├── routes.py # Flask routes / API endpoints

│ └── tasks.py # Celery background task (checkout processing)

├── migrations/ # Alembic migration files

├── worker.py # Celery entrypoint

├── run.py # Flask app runner

├── populate\_and\_check.py # Script to populate data and test metrics

├── Dockerfile # Image setup for backend

├── docker-compose.yml # Multi-service setup for Redis, PostgreSQL, backend

├── .env # Environment variables (ignored in Git)

├── .gitignore # Git ignored files and directories

├── README.md # Project description and usage

**🔧 Core Files and Their Responsibilities**

**1. run.py**

* Entry point for running Flask server.

from app import create\_app

app = create\_app()

**2. app/\_\_init\_\_.py**

* Initializes Flask app, SQLAlchemy, Alembic, and environment variables.

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

**3. app/models.py**

Defines SQLAlchemy models:

* User: represents users.
* Item: represents store items.
* CartItem: connects users to items with quantities.
* Checkout: represents checkout sessions with status enum.

**4. app/routes.py**

Exposes REST APIs:

* POST /user: Add a user
* POST /item: Add an item
* POST /cart/add: Add items to cart
* POST /checkout: Initiate checkout (push to queue)
* GET /checkout/<id>/status: Check a checkout's status
* GET /metrics: View processed checkout stats

**5. app/tasks.py**

* Contains the Celery background task process\_checkout():
  + Updates status to Processing
  + Waits 1–5 seconds (simulates work)
  + Marks checkout Completed

**6. worker.py**

* Runs Celery worker with app context.

celery = Celery(\_\_name\_\_)

celery.config\_from\_object(app.config)

**7. docker-compose.yml**

* Brings up all services:
  + PostgreSQL (db)
  + Redis (redis)
  + Flask API + Celery worker (backend)

**📊 Sample Metrics Output**

**Route: GET /metrics**

{

"average\_processing\_time\_seconds": 2.5,

"checkout\_status\_counts": {

"Completed": 5,

"Processing": 2,

"Pending": 3

},

"total\_checkouts\_processed": 10

}

**🧪 Test Script: populate\_and\_check.py**

Performs:

1. Adds 1 user
2. Adds 50+ items
3. Populates cart
4. Triggers multiple checkouts
5. Queries metrics after short delay

**🐳 Dockerized Environment**

**Services:**

| **Service** | **Role** | **Port** |
| --- | --- | --- |
| Redis | Task queue broker | 6379 |
| Postgres | Data storage | 5432 |
| Backend | Flask + Celery | 5000 |

Start everything with:

docker-compose up --build

**📷 Visual Diagram (Suggested)**

+------------+ +-----------+ +-------------+

| User/API +--POST----> Flask App +--enqueue--> Redis Queue |

+------------+ +-----------+ +-------------+

↓

Celery Worker

↓

PostgreSQL DB

**✅ Conclusion**

This project demonstrates a scalable, asynchronous system handling checkout processes. Celery ensures background execution while Flask serves the API. Metrics give operational visibility. Containerization simplifies deployment and testing.

*For detailed command examples, see README.md.*