

Talat Mahmud 1813080642

Md Mahdi Hasan Tazelly 2212193042

Sadia Islam 2212030042

Sanjida Amin Nadia 2122319642

Contents

[PROJECT OVERVIEW 3](#_Toc192703672)

[1. Introduction 3](#_Toc192703673)

[2. System Overview 3](#_Toc192703674)

[3. Functional Requirements 4](#_Toc192703675)

[4. Non-Functional Requirements 5](#_Toc192703676)

[5. User Role & Permissions 5](#_Toc192703677)

[6. Security Consideration 6](#_Toc192703678)

[7. Payment Methods 6](#_Toc192703679)

[8. Technology Stack 6](#_Toc192703680)

[9. Deployment Strategy 7](#_Toc192703681)

[ENTITY RELATIONSHIP DIAGRAM 8](#_Toc192703682)

[USE-CASE DIAGRAM 10](#_Toc192703683)

[CLASS DIAGRAM 11](#_Toc192703684)

[SEQUENCE DIAGRAM 12](#_Toc192703685)

[GITHUB REPOSITORY 14](#_Toc192703686)

# PROJECT OVERVIEW

## 1. Introduction

* This document sheds light on overall flow for different roles of the food ordering and delivery project.
* The goal is to simplify and make a food vendor easy to access. Primary audience are the existing and future customers of the food vendor which was operating via social media.

## 2. System Overview

2.1 System Reasoning

* This document sheds light on overall flow for different roles of the food ordering and delivery project.
* The goal is to simplify and make a food vendor easy to access. Primary audience are the existing and future customers of the food vendor which was operating via social media.

2.2 User Requirements

* Serve customers with available menu for food items, so users don't have to scroll unrelated posts from FB Page to get the menu and message the page to know about availability.
* Customers can pay digitally without going through manual verification process or pay cash on delivery if they do not prefer it due to change amount issue.
* Customers can track order progress live so that they don't have to ask every time by calling or messaging.
* Customers can maintain account, information etc. via an account system, so that they don't have to give address and other info every time they place order via call or message.
* Owner can keep track of different orders and print receipts.
* Owner can manage dishes and add new ones through his own portal as many times as he wants instead of making posts on FB multiple times for multiple dishes over the span of several days.
* Owner can keep track of his expenses and profits and ingredients.
* Owner can know ahead of time which ingredients are running out of stock and affected dishes for those ingredients.
* Waiter or chiefs or delivery man can update the status of order tracking, such as Preparing, picked up, rider is on the way and rider is around the corner and etc.

## 3. Functional Requirements

3.1 Menu Management

* Ability to add, update, and remove food items.
* Display available menu items to customers in real time.

3.2 Order Placement & Tracking

* Customers can place orders online.
* Order status lifecycle (e.g., Ordering, Preparing, Picked Up, On the Way, Delivered).

3.3 Order Management

* Refund or cancel order.
* Option for manager, chief or delivery person to update status of order.
* Order assignment to chief and delivery person

3.4 Payment Processing

* Support for both digital payments and cash on delivery.
* Secure handling of transactions.
* Update payment status.

3.5 User Account Management

* Create, update, and manage user profiles.
* Option to save delivery addresses and personal information.

3.6 Inventory Management

* Track ingredients and stock levels.
* Notify the owner when ingredients are running low.
* Create, update and manage food item and their availability.

3.7 Notifications

* Personalized notifications on user basis for different events related to that user.

## 4. Non-Functional Requirements

* **Performance:** System should response within 2 to 5 seconds after each API call and should be able to support 1M concurrent users without degradation. Orders should be processed and confirmed within 5 seconds after payment completion.
* **Scalability:** The system should scale horizontally to accommodate up to 0.1M user concurrently.
* **Reliability:** Uptime should be 99% with failsafe mechanisms in place. Errors should be handled gracefully. Data should be backed up around a routine with retention period of 14 days with a disaster recovery plan.
* **Maintainability:** Code structure should follow latest code pattern and libraries and follow a full stack monolithic architecture with a separated database. Utilize GitHub for version control and automated deployment.

## 5. User Role & Permissions

5.1 Customer

* **Capabilities:** Browse menu, place orders, track order status, manage account details.

5.2 Delivery Person

* **Capabilities:** Update order delivery status, view delivery details.

5.3 Admin

* **Capabilities:** Manage dishes/menu, view and manage orders, print order receipts, handle inventory, view users and update user roles, initiate order refunds, assign order to chief and delivery person.

## 6. Security Consideration

6.1 User Authentication

* Credential and OAuth Cookie based authentication.
* Passwords are hashed and stored in database securely. User data is exchanged via encrypted cookies.

6.2 Role-Based Access Control (RBAC)

* Permissions and access levels for each role are customized and limited to that role.
* RBAC is enforced throughout the application through checking user role during each API request and Route Access.

6.3 Other Security Measures

* Data and type validation and error handling.
* Protection against common vulnerabilities (e.g., SQL injection, XSS, CSRF).

## 7. Payment Methods

7.1 Digital Payments

* Explain integration with digital payment gateways.
* Process flow for successful and failed transactions.

7.2 Cash on Delivery (COD)

* Process flow and order confirmation for cash on delivery.
* Handling of order verification and finalization.

## 8. Technology Stack

* **Frontend:** NextJs, Tailwind CSS, shadcn/ui
* **Backend/Server:** NextJs Server Actions, NextJs API Route
* **Database:** Neon PostgreSQL
* **APIs and Queries:** Axios, TanStack Query, Zustand
* **Authentication:** Better Auth
* **ORM:** Prisma
* **Validation:** Zod
* **Deployment Platform:** Vercel

## 9. Deployment Strategy

9.1 Deployment Environment

* Production and Development build. Separate URL for both environments.
* Vercel Edge function and managed infrastructure.

9.2 Deployment Steps

* Code changes will be pushed to dev branch.
* After review, code will be merged to main branch.
* Each push to main branch will initiate a production build on Vercel through GitHub actions.
* Include rollback and disaster recovery procedures.

9.3 Rollback & Disaster Recovery

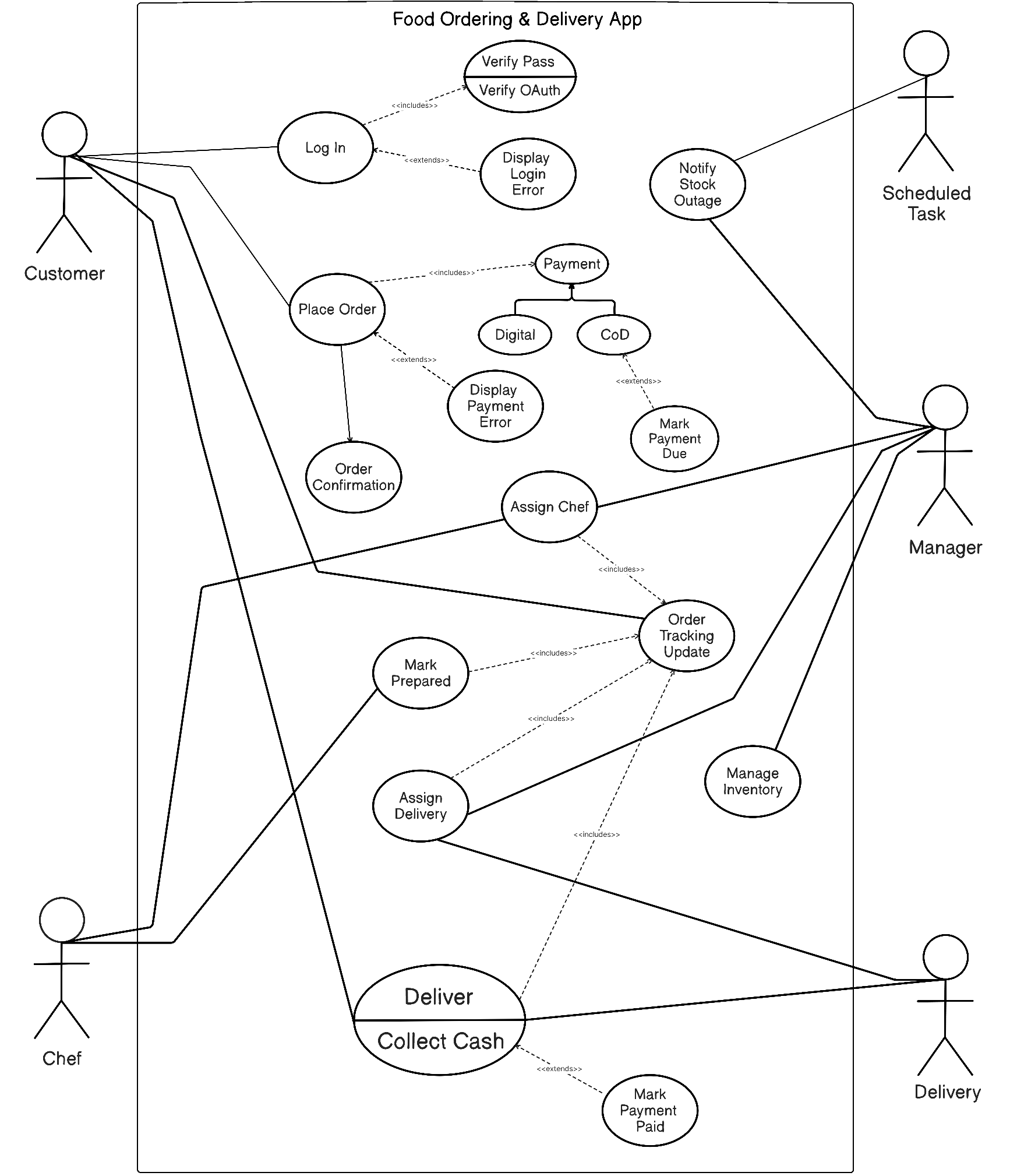
* A track of each deployment is kept using GitHub commit ID.
* In case of any issue or disaster, production can be reversed back to any of the previous deployment.

# ENTITY RELATIONSHIP DIAGRAM



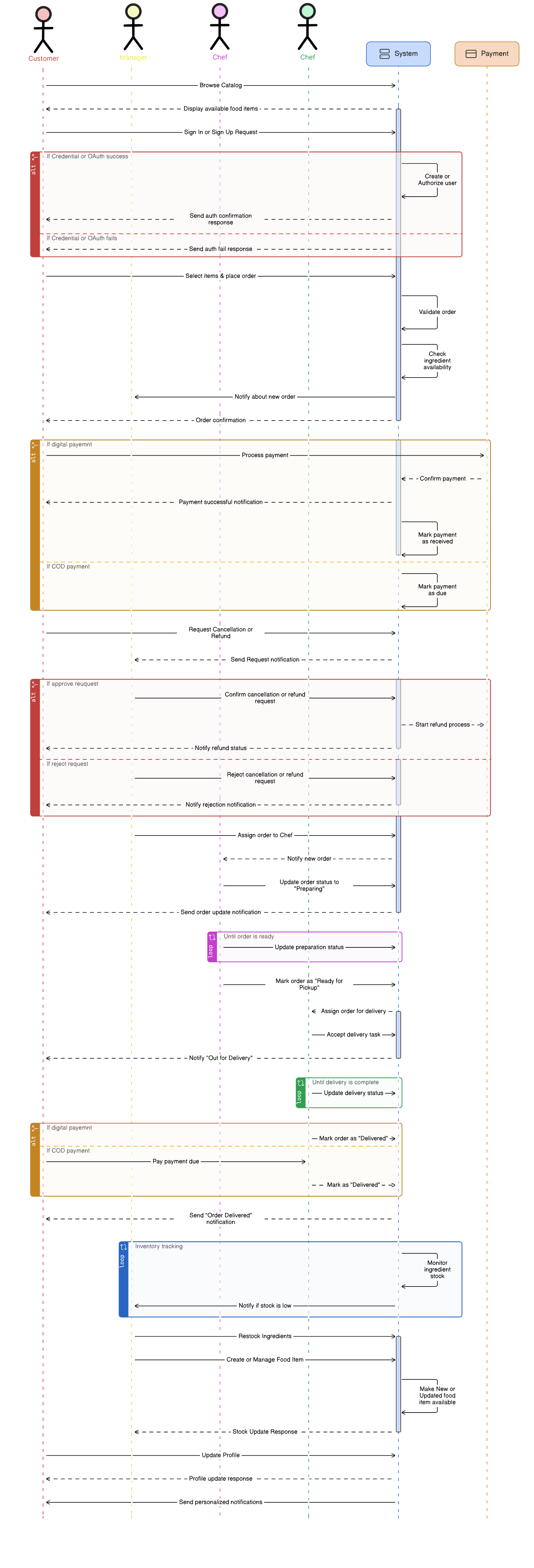


# USE-CASE DIAGRAM



# CLASS DIAGRAM

# SEQUENCE DIAGRAM



# GITHUB REPOSITORY

* <https://github.com/FalconiZzare/crunch-time>