UNIVERSITY OF STAVANGER, NORWAY

DAT240 Project Report (SmaHauJenHoaVij): Campus Eats

Mio Småland, Vegar Hauge, Sanjai Vijeyaratnam, Rich Hoang

December 6, 2024

CONTENTS

1 INTRODUCTION

Campus Eats is a food delivery platform designed specifically for campus communities, aiming to make food delivery easier for students and staff. This report covers the development journey, how we applied Domain-Driven Design (DDD), event-driven architecture, and tools like Docker, GitHub Actions, and external payment services. Our platform supports three main user roles: customers, couriers, and admins, and includes a variety of business workflows to meet the needs of these users.

The system uses containing to ensure consistency and scalability, with tools like SQLite for database management and MediatR to handle event-based communication. We've also integrated external services like Stripe for payment processing, making order payments smooth and secure.

This report gives an overview of the system's design and architecture, discusses some of the challenges we faced along the way, and explains the solutions we implemented to overcome them.

2 Design

The design of Campus Eats follows the principles of Domain-Driven Design (DDD), which helps structure the system in a way that keeps each part focused on its specific responsibility. By breaking the system into different "bounded contexts," we ensure that each part can evolve independently, making the whole system modular, easier to maintain, and scalable.

2.1 Domains and Contexts

Campus Eats is divided into several key domains, each designed to handle a specific set of business rules. Each domain encapsulates a portion of the system's logic, allowing changes to be made in isolation without affecting the rest of the application. This modular design allows us to evolve each domain according to its specific needs.

- **Cart Domain:** This domain is responsible for managing the shopping cart experience. It allows customers to add and remove items, adjust quantities, and proceed to checkout. The Cart Domain ensures that customer selections are tracked and properly handled during the purchasing process.
- **Ordering Domain:** The Ordering Domain manages the entire lifecycle of an order. This includes creating new orders, updating their statuses, and triggering necessary events when an order moves from one stage to another (like from "Placed" to "Shipped"). It communicates with other domains, such as Fulfillment, to keep the order status updated.
- **Products Domain:** This domain handles the catalog of food items. It manages the details of each product, including pricing, descriptions, and availability. This domain ensures the product data is kept accurate and can be easily updated as needed, while also providing any necessary data for the ordering process.
- **Fulfillment Domain:** The Fulfillment Domain is responsible for managing the logistics around order fulfillment. This includes assigning couriers to orders, tracking deliveries, and updating the order status as it progresses through the fulfillment process. It communicates with the Ordering Domain to ensure that when an order is shipped, it reflects the correct status.
- **Users Domain:** The Users Domain handles all user-related functionality, including authentication and authorization. It manages the registration process, login/logout, and access control for different roles (customers, couriers, and admins). It ensures that users can only access the parts of the system that match their role and permissions.

2.2 User Pages

The platform offers distinct pages and functionalities depending on the user role. These pages are tailored to ensure that each type of user can perform the necessary actions related to their role.

- **Admin Pages:** Admins have a set of tools that allow them to manage orders, products, and view system-wide data. They can also oversee the actions of other users and ensure that everything is running smoothly across the platform.
- **Courier Pages:** Couriers have a interface where they can view orders assigned to them, update delivery statuses, and communicate relevant delivery information. The pages are designed to be simple and easy to navigate, enabling couriers to track and manage their deliveries efficiently.
- **Customer Pages:** Customers can register, browse products, place orders, and track the status of their deliveries. They can also view past orders and manage their account details. The customer experience is optimized for simplicity, allowing users to place orders and track them with minimal steps.

2.3 Infrastructure

The backend infrastructure is designed to support the modular nature of the system and ensure smooth communication between different domains.

- **Database:** The system uses a relational database managed by Entity Framework Core, which makes it easy to interact with and manage data. SQLite is used as the database provider for simplicity and ease of development. This allows us to store information like product details, orders, and user data securely and efficiently.
- **MediatR:** MediatR is used for event-driven communication between the various bounded contexts. This ensures that when important changes occur in one part of the system (e.g., an order is placed or a product is updated), other parts of the system can react and update accordingly. This decoupling of systems allows for easier maintenance and scalability.
- **Services:** We have placed the EmailService in this folder, which consists of the emailservice, it's interface and a template for the different events that are raised.
  The emailservice retrieves a users email upon one of the raised events, and makes the email be sent out, which is also something configured by the help of Gmail's SMTP server.

3 Backend

The backend of Campus Eats is built using .NET Core, following Domain-Driven Design (DDD) principles. This approach ensures clear separation of concerns between business logic and infrastructure, providing a structured framework for the system's development. Each domain is designed to manage its specific responsibilities, with communication between domains facilitated through events published via MediatR, promoting a decoupled and scalable architecture.

3.1 Domains and Their Responsibilities

3.1.1 Users Context

The Users Context handles all aspects of user management, including authentication, registration, and role-based access control. It ensures that only authorized users (customers, couriers, and administrators) can perform specific actions in the system, providing a secure access control mechanism.

3.1.2 Cart Context

The Cart Context is responsible for managing the user's shopping cart. It supports operations like adding and removing food items, adjusting item quantities, and processing the checkout. The context ensures a seamless flow from cart creation to order submission, interacting with other domains like Ordering and Products.

3.1.3 Ordering Context

The Ordering Context manages the entire order lifecycle, from creating a new order to updating its status as it progresses through various stages (e.g., "Placed", "Shipped", "Delivered"). This context also triggers domain events using MediatR when important state changes occur, ensuring other parts of the system are notified and can react accordingly.

3.1.4 Products Context

The Products Context is responsible for maintaining the product catalog, which includes the details of available food items, their pricing, and inventory. This context ensures product data is accurate and up-to-date, and provides functionality for adjusting product attributes, such as availability and price, through specialized pipelines.

3.1.5 Fulfillment Context

The Fulfillment Context oversees the logistics and delivery of orders. It handles the assignment of couriers, tracks the status of deliveries, and ensures that the order's fulfillment status is properly maintained. This context also communicates with the Ordering Context to update the order status when it has been shipped or delivered.

3.2 Technical Stack

The backend relies on a robust set of technologies to support the system's functionality and scalability:

- **C# and .NET Core:** These core technologies provide a fast, reliable, and cross-platform environment for building the application, including RESTful APIs and background services.
- **Entity Framework Core:** This ORM simplifies database interactions, enabling easy queries and updates with SQLite as the database provider.
- **MediatR:** Used for event-driven communication between domains, MediatR facilitates asynchronous operations and allows different parts of the system to react to events without tight coupling.
- **Docker:** Ensures a consistent and reproducible development environment by containerizing the application, making it easier to manage dependencies and deployments.
- **Facebook Development Tool:** Integrated for user authentication through Facebook, providing an additional login option that enhances the user experience.

3.3 Challenges and Solutions

Throughout the development process, several challenges were encountered, each addressed with targeted solutions:

- **Integration with Stripe:** The Stripe API was integrated to handle payment processing securely. This involved configuring the API for various payment methods.
- **Team Coordination:** To minimize code conflicts, collaboration tools such as GitHub and project management software were implemented, along with more detailed sprint planning.
- **SMTP Integration:** The implementation of email notifications simplified communication with users, ensuring they receive timely updates for events like registration and order status changes, etc.
- **Database Schema Issues:** Compatibility issues with SQLite, especially regarding the handling of decimal data types, were addressed by adjusting the schema and ensuring accurate data persistence.
- **Unit Testing and Database Seeding:** Creating meaningful unit tests while avoiding redundant tests was a challenge. Pipelines were designed to ensure test coverage without unnecessary duplication.

4 Technologies Used

The Campus Eats project utilizes a range of technologies and tools to meet both the functional and non-functional requirements of the system.

4.1 Framework and Libraries

- **.NET Core:** The primary framework for building the backend, known for its high performance and cross-platform capabilities. It enables the development of scalable and maintainable RESTful APIs.
- **Entity Framework Core:** An Object-Relational Mapping (ORM) tool used to interact with the SQLite database, simplifying data management and eliminating the need for manual SQL queries.
- **MediatR:** A library that implements the Mediator pattern, used to handle asynchronous communication and events between different parts of the system, ensuring loose coupling and scalability.
- **Stripe API:** Integrated for secure payment processing, Stripe handles customer transactions, supporting multiple payment methods and ensuring PCI-DSS compliance.
- **Docker:** Used to containerize the application, ensuring a consistent development environment and simplifying deployment across different platforms.

4.2 External Services

- **SMTP Server:** Used to send automated email notifications to users, such as order confirmations, delivery updates, courier application updates and welcome emails.
- **GitHub Actions:** Facilitates continuous integration and deployment (CI/CD) pipelines, automating the testing, building, and deployment processes to ensure high-quality code and seamless updates.

5 Conclusion

The Campus Eats project successfully demonstrates the application of Domain-Driven Design principles in a real-world scenario. By using an event-driven approach, the system is scalable, allowing for easy addition of new features and changes in business requirements.

5.1 Achievements

- Successful implementation of DDD for a scalable system architecture.
- Integration with external services such as Stripe and SMTP for payments and notifications.
- Deployment using Docker for easy management and environment consistency.
- Communication, through discord, Github and spending multiple hours together in school.

5.2 References :

:

ChatGpt

Documentation for C

Nuget package Documentation

References