DAT240 Campus Eats

Project report

Daniel Gaudland Jacobsen

Emma Knorová

Théo Barat

Lilian Breux

András Tarlós

November 23, 2025

Universitetet i Stavanger

DAT 240 - Software Engineering

Norway

Table of Contents

1 Project Summary

The project went overall quite well; the team managed to implement all the base requirements and even some of the more advanced requirements, stated in Campus Eats' project assignment. The result is a functioning, minimalistically designed webpage, where customers can order food, and couriers can deliver the orders to get paid.

2 Organizational

2.1 Project Management Methodology

To track progress, we have decided to create a Trello board with all the basic workflows stated in the project description. We created a user story for each individual workflow, so the project is divided into manageable chunks. The Trello board had the following flow: Backlog, To Do, Doing, Code Review, Testing, and Done. These states helped us to keep track of who was doing what and how much remained to be done. In order to keep track of which specific feature we want to implement in git, we decided to abide by a branch naming convention. Each of our Trello cards were numbered; these numbers worked as unique identifiers, which we used in our branch names for better clarity.

The necessity and advantages of a project management tool have overwhelmingly shown themselves during the project execution.

It has become apparent to us early on, that dividing a project of such magnitude is not a simple task. Most of them were dependent on another task; one could not be started without finishing the other.

2.2 Reflection on Communication Practices

Quite early on after the project preliminaries, we organized a kick-off meeting to set up a basic project structure, so everyone gets the same fundamental overview. Apart from a few disagreements about minor technical implementations - which were mostly irrelevant -, we have successfully set up the foundations of the project. Gathering together to start the project was a very efficient way to get things moving.

A Discord server has been created to keep the communication going and to have a place to exchange relevant pieces of information. Discord served its purpose well.

The primary takeaway in terms of communication from this project is that clarity and knowledge sharing are the key to success; without communication, there is no organization.

3 Technical

3.1 Frontend

3.1.1 Development

In this software engineering course, the focus has primarily been on C# development and different methodologies of building reliable software. It was imperative to use a frontend technology which the entire team is familiar with. Since our team has different educational backgrounds, and almost all of us have different work experience, it was challenging to decide for one frontend technology that whole team can use comfortably and efficiently. Thus, we have decided to stick with the frontend technology used during this course: Razor Pages.

3.1.2 Challenges & Solutions

The first hurdle we had to face was interpreting the requirements. The great majority of them were quite clear and concise, but some of them were a bit ambiguous, and could be interpreted in multiple ways. In a realistic scenario, we would have contacted the client to clarify what they wanted; however, since this is a school project, we interpret the requirements to the best of our ability.

One example is "Registers as a courier on the site. Uses be the same login as a customer, but registering as a courier is a separate action.". It was not clear to us what separate action meant, so we decided to have one registration page for the customer and courier, where they can decide which role they want to register with.

3.1.3 Push notifications

The application uses a real-time push notification system to keep customers notified about changes to the status of their orders, and to notify couriers when a new order has been placed and ready to be accepted. This system uses `SignalR` to provide communication between server and client instantly without page reloads. The notification system uses the `Toastr.js` javascript library for in-browser notifications and the browser's built-in notification API for OS notifications if you have your browser minimized for instance.

3.2 Backend

3.2.1 Development

We adopted a Domain-driven Design (DDD) approach to separate the core business logic (the domain) and the surrounding application infrastructure. This was achieved using patterns like

pipelines, handlers, events, and services. This architecture also boosts maintainability, improves scalability, and makes testing easier by isolating the domain.

3.2.2 External Payment Integration (Stripe)

To handle payments in a secure and reliable way, we integrated the external payment provider Stripe into our application. Stripe was chosen because of its well-documented API, strong security guarantees, and support for advanced workflows such as refunds and tipping. In our system, Stripe manages the entire payment life cycle: creating checkout sessions, processing card payments, and confirming transactions. On the customer side, this allows users to pay for their orders, cancel them with partial or full refunds depending on the status, and add a tip to the courier after delivery. On the courier side, the payment and tip identifiers stored in the `Order` table can be used to query Stripe for the actual amounts received, ensuring transparency and accuracy in earnings calculation. This integration not only simplified our implementation but also ensured compliance with industry standards for payment processing.

3.2.3 Testing

For testing we implemented a number of unit tests using `Xunit` for the testing framework and `Moq` for mocking dependencies. The tests are organized in a separate folder (webapp.Tests) parallel to the main project folder (webapp) mirroring the same domain driven structure of the main application. For instance cart pipelines are located in `webapp/Core/Domain/Cart/Pipelines` and in the and cart pipeline tests are located in `webapp.Tests/Core/Domain/Cart/Pipelines`.

For testing infrastructure we used a helper class that would set up a new in-memory SQLite database for every test and would thus ensure that tests were isolated and repeatable. This also allowed to test against actual database operations without requiring a server connection to the main PostgreSQL database.

All of the tests follow the Arrange, Act, Assert pattern where first the input data is set up, then the function to test is executed and finally the outcome is matched against expectations. The naming convention for the tests generally follow the naming convention of methodName_scenario_expectedOutcome.

We also used continuous integration (CI) via Github Actions for automated testing, any time a pull request was opened, or when someone pushed to the main branch.

3.2.4 Challenges & Solutions

Implementing the **third party login** was a tremendous amount of work and a substantial challenge to get right. Initially, it was decided that Campus Eats will have the option to log in with third party providers such as Google, Microsoft, Twitter, and Facebook. After some initial research - keeping in mind the limitations of our development project -, it was quite clear that we will be unable to use Twitter's API to log in; they enforce a strict `HTTPS` only traffic rule, which the locally running website with `HTTP` could not fulfill. As it later turned out, Facebook had similar issues. This left us only with Google and Microsoft as third party login providers.

Implementing Google's third party login was mildly easier than doing the same with Microsoft's solution. Microsoft caused us a lot of headache. They only allow developers to define a callback URI - the website's URI which gets contacted by the third party provider's API - that starts with `http://localhost`. However, the equivalent IP, `http://127.0.0.1`, did not work and threw a vague error message. Google, however, managed to handle both formats of `localhost`. This took some time to figure out, but now we know that the Microsoft login only works if we open the website by contacting `localhost` and not the equivalent `localhost` IP, `http://127.0.0.1`.

The team chose a simple role-assignment method for managing new admin users. When an admin promotes a customer or courier to an admin role, the user's role is updated instantly, but no automated notification is sent. This decision is based on the reasoning that all new admins must receive mandatory, in-person training on the admin interface. Therefore, the responsible administrator will personally contact the individual, ensuring they are notified of the role change and trained on how to use the system.

4 Contributions

Daniel Gaudland Jacobsen Testing framework; unit tests across the board; Cart and Product context pipelines and handlers; push notifications; admin invites and admin product list management.

Emma Knorová Ordering context; courier pages and pipelines; profile overviews; unit tests for ordering; tipping couriers; setting the delivery fee.

Théo Barat Database planning; implementation of the external payment system with Stripe; customer-side order logic including cancellation rules; delivery fee handling; and tipping workflow.

Lilian Breux Docker setup; admin page for earnings and revenue split; admin product list management page; and admin dashboard with order statistics.

András Tarlós Project management; user & courier & admin registration and login workflow; set up CI workflow in GitHub; courier approval and declining; unit tests; authorization mechanisms; third party login with Microsoft and Google.