Intro

Tuesday, January 10, 2023 10:33 PM

This is a one note that describes the same named project.

Reservation system, which is a part of event ticketing system, was originally created for problem solving purpose. A group of artists needed a way to organize events, basically.

Since the deadline was placed I was in obligation to separate the project into MVP version and After MVP versions.

Having said that, the MVP version will be 1.0 and After MVP versions will move forward as 2.0.

The Goal

Monday, 16 January 2023 23:06

MVP

Wednesday, 11 January 2023 00:25

So a Minimal Viable Product.

Deadline is 01.February.2023.

The application needs to do the following task:

Create a reservation for an event.

Seems pretty simple. Let's review the use case in detail.

Use Case:

- A customer enters a web page where he can see various available events.
- Selects an event.
- Enters the number of tickets to reserve and enters his email, first and last name.
- Clicks a button reserve.
- Receives a response whether his reservation was completed or not.

To make the matter a bit more complicated, we're going to create a mobile web app version as well.

On the other side we have an admin use case. Admin is a role that creates and manages events. This means that this role is able to:

- Manage different events types
- Manage different event questions
- Manage different events
- Manage event's occurrences
- Manage event occurrence's tickets
- Manage ticket's extensions

These actions can be separated in couple of use cases. We are going to create one that covers entire event creation process with following use case in mind - creating an event and all of it's dependants:

- An admin logins into the dashboard
- Clicks a button create an event
- Fills in an one page form that contains:
    - Event basic info
    - Event questions
    - Event occurrences (dates)
    - Number of and price of event tickets
    - Ticket (or event) extensions
- After filling out lengthy one page form the admin clicks save button
- Receives a response whether event creation was successful or not

This is the scope that's defined for the MVP.

This solution will be using the following technologies:

- Database - MySql
- Backend - .Net Core 6
- Frontend - Angular latest version (both web app and mobile app for customer only)

This 3way infrastructure needs to be deployed to the private VPS, host will be decided afterwards.

Authorization

Wednesday, 11 January 2023 00:24

The initial authorization use case for reservation was pretty straight forward: enable a possibility for people which do not have an account to create a reservation for artist's event.

On the other side, admin login will be accessible through a url path that is not intuitively available for customers or admins.

The recommendation will be for admins to bookmark the login path.

_____

After MVP

At this moment I am considering whether to implement a mandatory Google SignIn option.
Perks of this scenario:

- Reduction of potential DDoS attacks
- A form of simple yet effective authorization
- Having necessary info without the user filling in a lot of info

Cons of this approach:

- Implementation that I am not familiar with, which will take additional development time to get familiar with
- Possibility of expanding the database model, which may complicates things a bit more
- User may not have a gmail account, which means that additional implementation for other quick authorizations will be needed (Facebook, Microsoft, etc.)

Before I continue, I think I should define a MVP with different section and their decisions. MVP deadline is in less then a month.

12.01.2023.

It has been decided that Google Authentication implementation will move forward after MVP. Likewise, other sign in options will have to be developed (Facebook, Microsoft).

MVP Use Cases - In detail

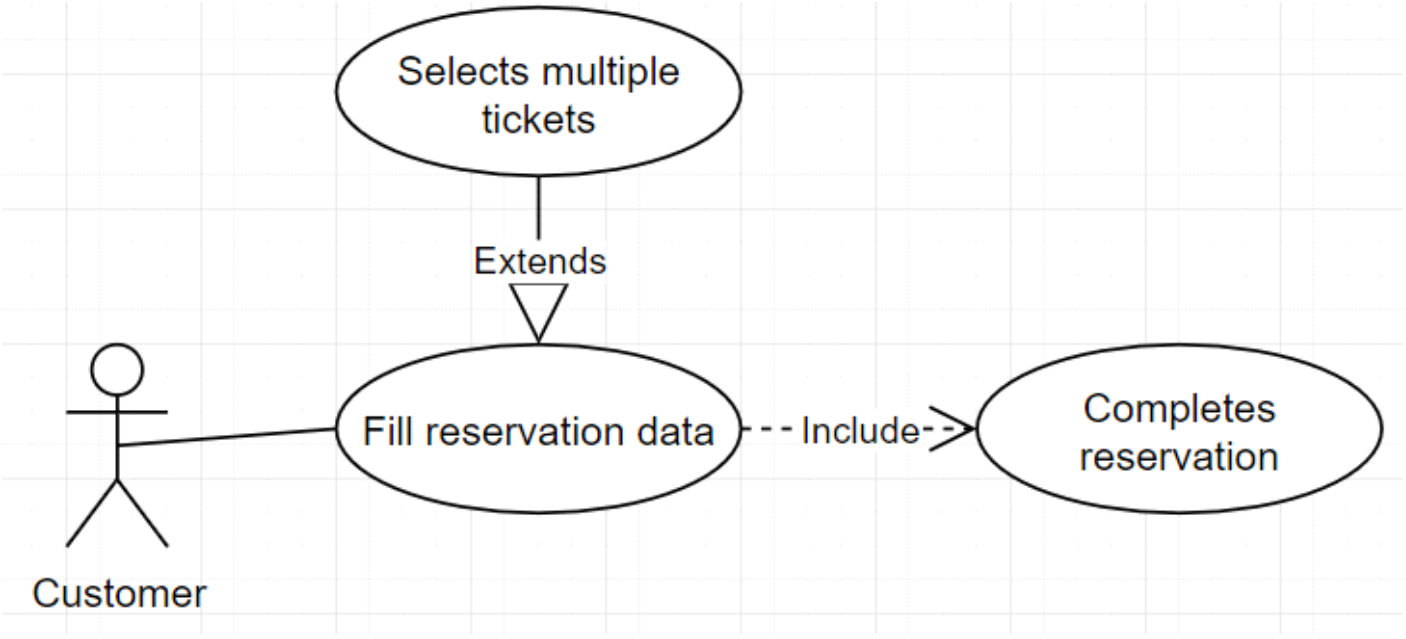
Wednesday, 11 January 2023 10:13

Customer creates a reservation

This use case describes a process of ticket reservation for an event. It is viewed from the Customer's point of view and maps his steps in order to successfully complete the process.

Extends = optional step
Include = mandatory step

Only out of the ordinary thing here is an option to select multiple tickets.



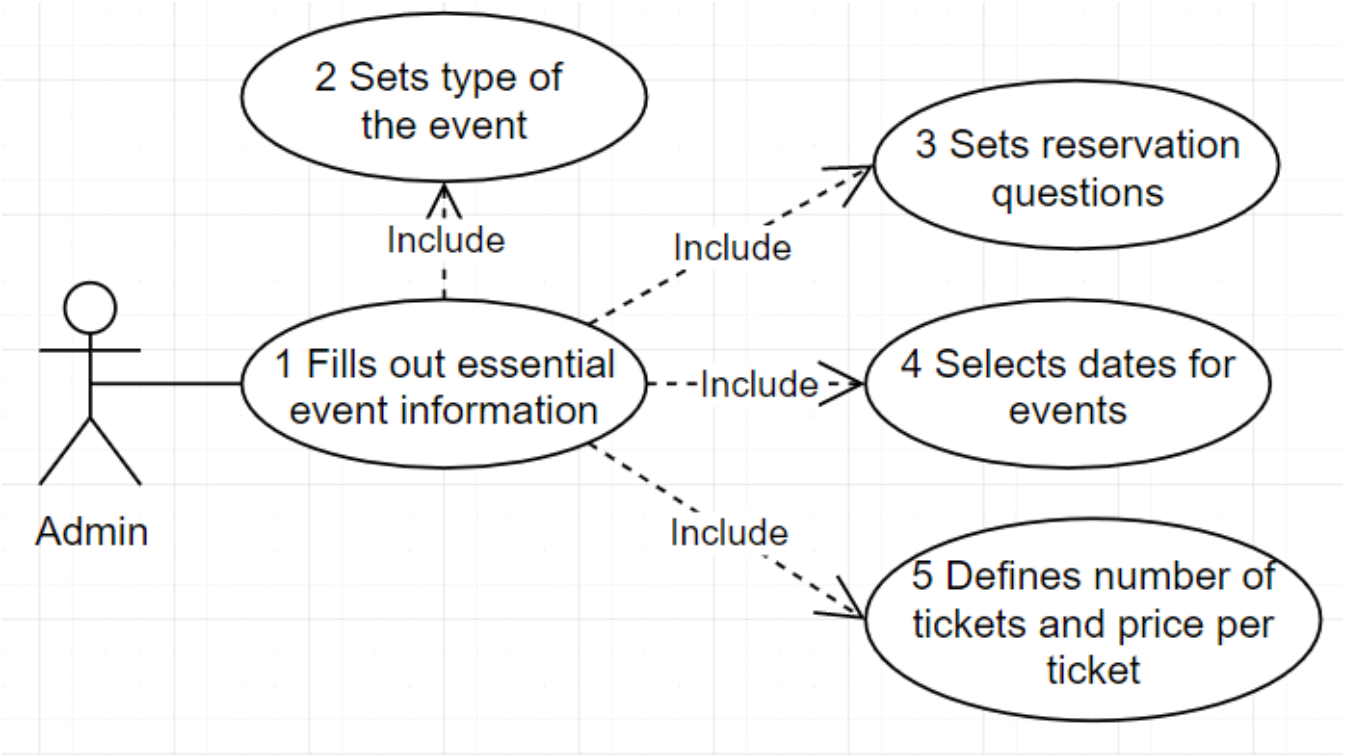
Admin creates an event

This use case is a bit more complicated by the number of activities only. I've pointed out these activities with numbers before their description because I want to map them to classes (and database tables) that will be created afterwards. The activities represented in the diagram:

- Event - 1
- EventType - 2
- EventQuestion - 3
- EventOccurrence - 4
- Ticket - 5

These entities are all related to the main one - event. They either describe or extend the event.

Extends = optional step
Include = mandatory step

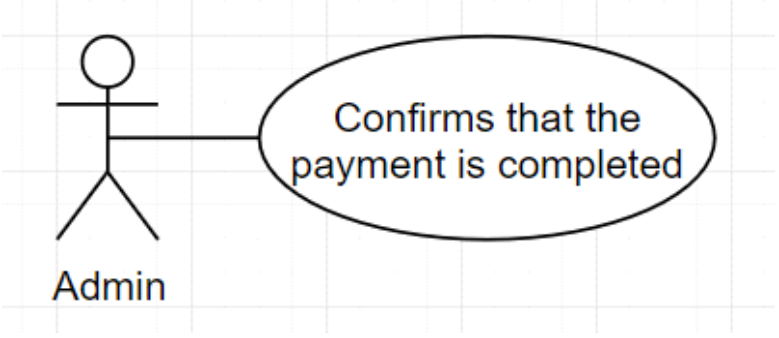


The process itself it pretty straightforward: it is going to be a one page form where an admin will fill out all mandatory fields and saves the event.

Admin confirms that reservation's payment is completed

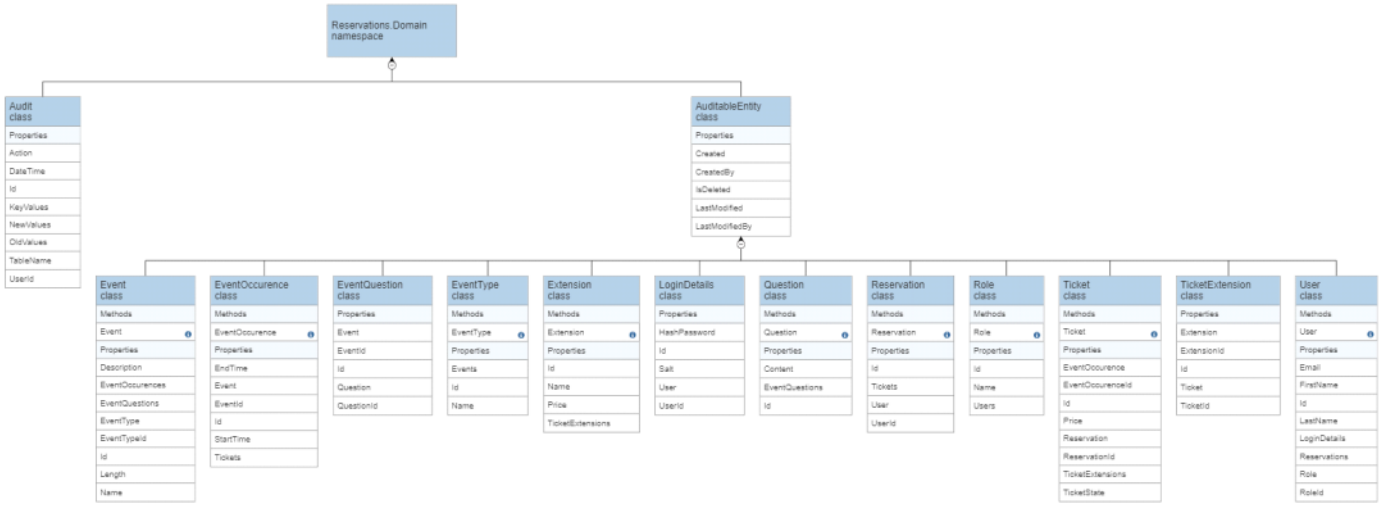
This is the last use case for the MVP. It is the final piece of the puzzle.
Event is created, reservation is made. All that is left is the payment and ticket purchase can be completed.
Activity that occurs before this use case, besides reservation use case, is that the customer sends money to the event organizers. This use case will not be covered in MVP and as such will not be covered in this documentation for the time being as it is out of the scope of this project.

This use case will assume that the payment received and that the only thing left to do is for admin to confirm that in the app.



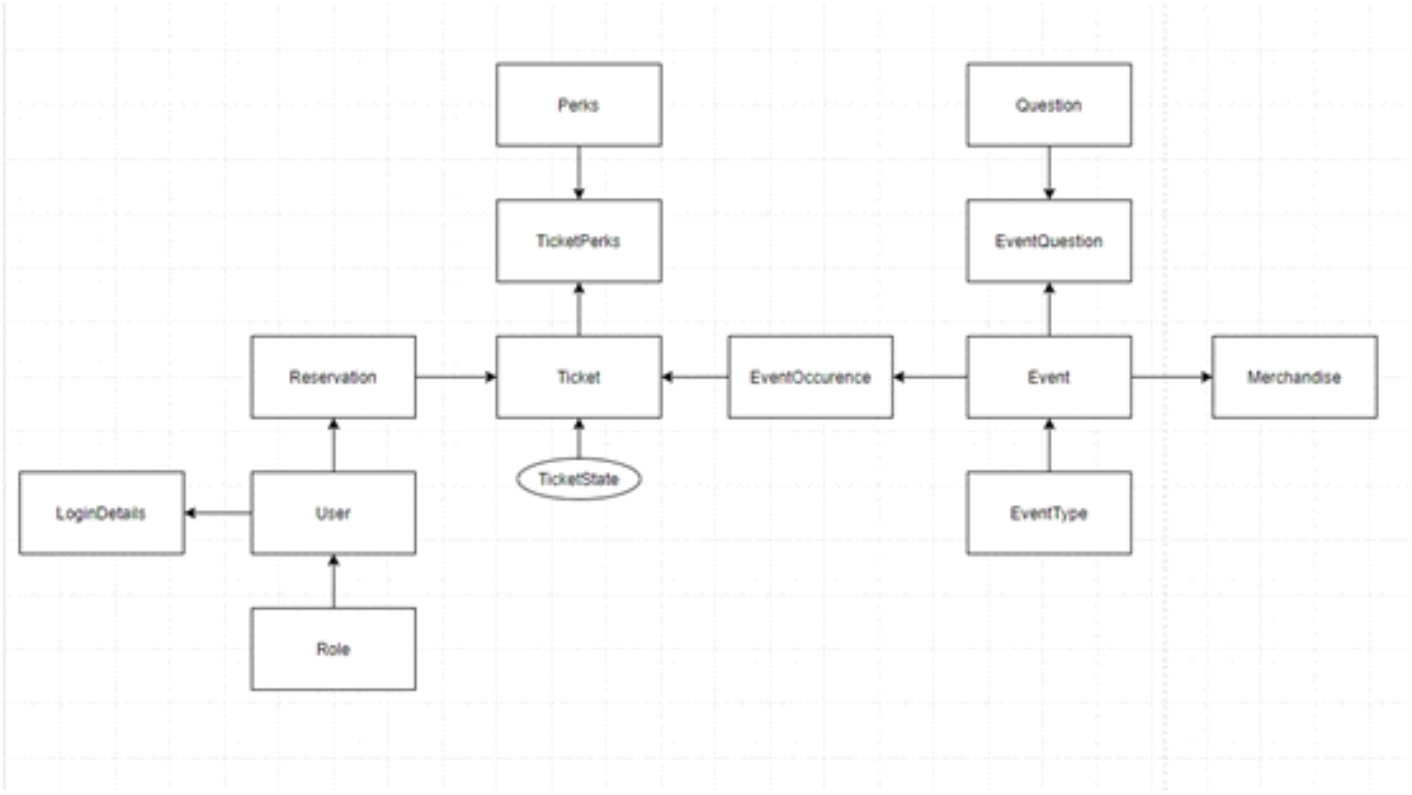
Class Diagram

Wednesday, January 11, 2023 6:14 PM



Entity Relationship Diagram

Wednesday, January 11, 2023 6:15 PM



Backend Architecture

Monday, 16 January 2023 15:26

Backend is architected with Onion Layer architecture. The projects are separated into a couple of different folders:

1. First one is Core. The Core folder consists of Application project and Domain project.
    a. The Application project is where business logic lays.
    b. The Domain project is where the domains or classes are set up.
2. Next one is Infrastructure. In the Infrastructure folder currently only Persistence project is there. The Persistence project's purpose is to establish a communication with a database.
3. Next folder is Tests where Unit Tests are placed. Unit Tests are there to support the development and since we're using Test Driven Development we are creating unit tests first and basing the business logic on them.
4. The last one is not a folder but it is a startup project API. This is where the communication with the backend is initialized and it's the entry point to the backend.

Each of these folders represent a part of the system that serves a certain purpose. Thus, making the system a bit more loosely coupled.

For example, core is where the business logic is placed. Next one infrastructure is where the communication with the external services occurs. The folder tests is where we place unit tests. And the last one the API, which is not a folder, is where we have our entry point to the system. Using this example we can map the data route in the following order:

1. Data enters through the API project. **Presentation layer**.
2. Core layer is called from API project and the business logic is applied. **Business Logic layer.**
3. Infrastructure layer holds the Persistence project which is where the data is passed for database operations. **Persistence layer.**
4. Tests folder is a folder that supports Core layer. **Support layer.**

Using this logic we come to the following conclusion:

1. All data entry related activities should be placed in the API project.
2. All business logic related activities should be placed in Application project, which is inside Core folder. Likewise all new enitity classes and their enums should be placed inside Domain project, which is inside Core folder.
3. All database related activities should be placed in persistence project, which is inside Infrastructure folder.
4. All Unit Test related activities should be placed in unit test project which is inside Tests folder.

By using these rules, we can achieve a loosely coupled system with its components not interfering with projects and classes that do not relate to them, thus making it easier to develop and maintain the codebase.

Design Pattern - CQRS

Monday, January 16, 2023 8:12 PM

Command Query Responsibility Segregation pattern.

Unit Tests

Monday, 16 January 2023 23:06