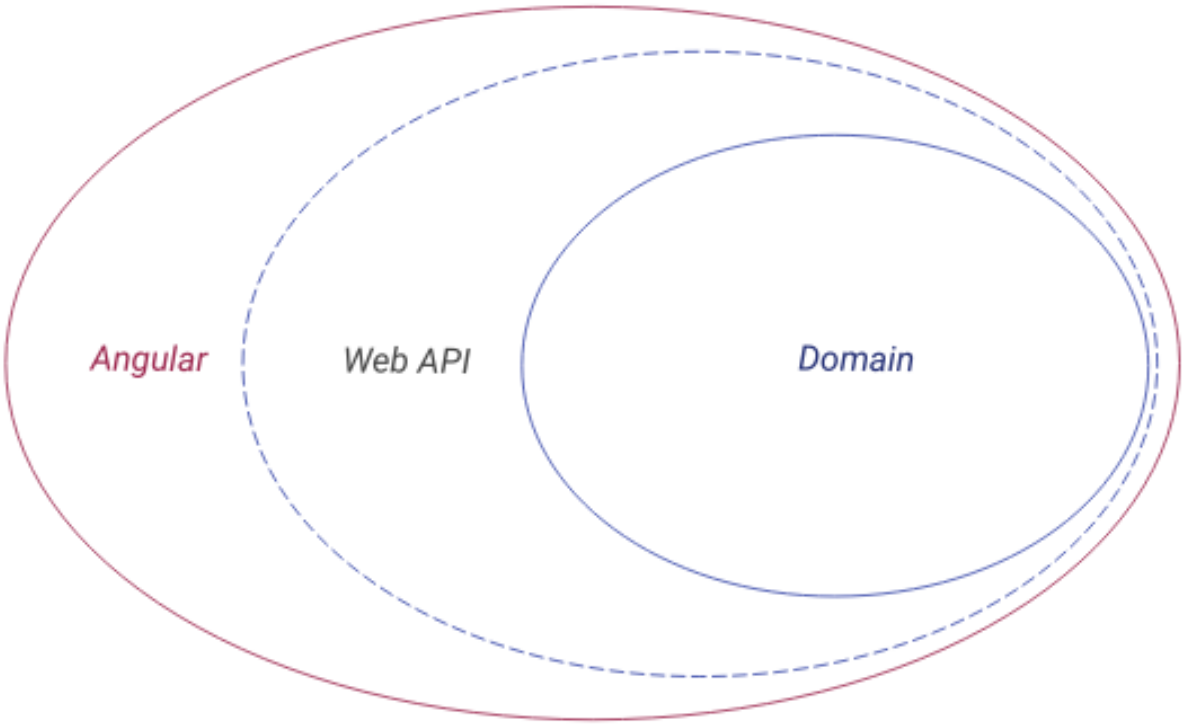
Day 13 and 14 - Angular & Asp in 21 days Handouts

Status

Day 13: Entities



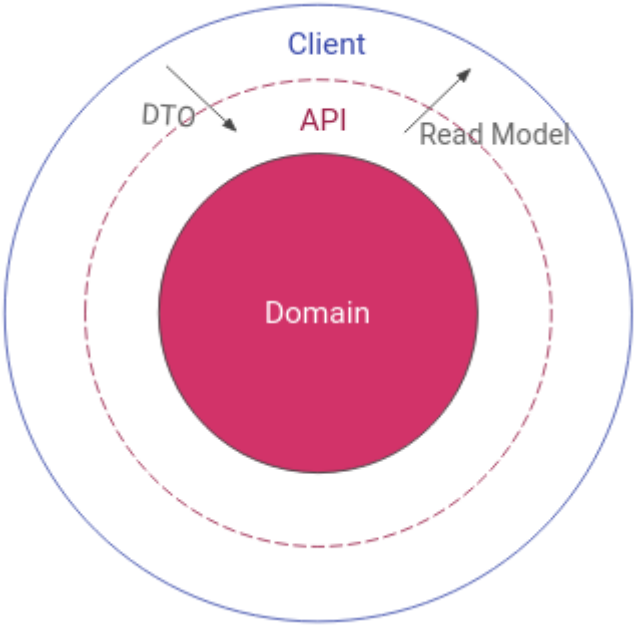
We as developers should keep the most important part of our application as easy to modify as possible. So we separate it from any technology. We avoid the code that validates domain rules to be mixed with the code that handles HTTP requests, or even the code that manages transactions.

REST API, HTTP, and Transactions are not what a stakeholder deals with when a software system does not exist, but keeping an account balanced in an accounting domain, or controlling overbooking of flights in our course project are subjects that are meaningful and important for the stakeholders even when they don't use any software automation. For that reason we try to separate them and avoid mixing them with any technological concern to keep them maintainable.

We used Data Transfer Objects to pass data from our Angular application to the API.

Entities are in the domain layer. They shape a cohesive internal core of our application in the form of a model representing the business processes our application automates and the rules enforced in them.

Data Transfer Objects are small sets of data we pass between layers when necessary.



Domain rules are a set of rules that exist in the business domain even when the rules are not automated by software.

A flight should have the capacity the passenger needs. It’s a domain rule. Domain rule validation is best to be done in the domain layer.

Ideally, entities should not depend on any library or framework. They should be pure and easy to understand. A code that’s easy to understand is easy to maintain and change.

The reason we build applications is to automate some processes in the domain. The most important thing that requires the application to be modified is a change in domain rules or its processes.

Like anything else in this world, the domain is constantly changing.

We, as developers, should keep the most crucial part of our application as easy to modify as possible. So we separate it from any technology. We don’t let the code that validates domain rules be mixed with the code that handles HTTP requests or even the code that manages transactions.

REST API, HTTP, and transactions are not what a stakeholder deals with when a software system does not exist, but keeping an account balanced in an accounting domain or controlling the overbooking of flights in a flight booking domain are subjects that are meaningful and important for the stakeholders even when they don’t use any software automation. Therefore, we try to separate them and avoid mixing their code with the code that handles technological concerns to keep them maintainable.

Domain Rule Example

Canceling a booking should free up the seats.

```
public class Flight
{
  ...
  public object? CancelBooking(string passengerEmail, byte numberOfSeats)
  {
      //Find the booking to remove it from the bookings list:
      var booking = Bookings.FirstOrDefault(b =>
              numberOfSeats == b.NumberOfSeats
           && passengerEmail.ToLower() == b.PassengerEmail.ToLower());

      if (null == booking) return new NotFoundError();

      Bookings.Remove(booking);
      RemainingNumberOfSeats += booking.NumberOfSeats;

      return null;
  }
  ...
}
```