Downtow-App – Backend Overview
Version 1.0

Capstone Project – Team 44
February 19, 2023

Vincent Harvey
Elias Mahamane Sidi
Nabil Wahbi
Younes Anys

Content

Downtown-App – Backend Overview

Technologies

1. Framework: NET 6, a.k.a Asp.Net Core 6
    a. EntityFrameworkCore
    b. **IdentityUser** - User Identity Framework used for managing users, authentication and authorization.
    c. **IdentityDbContext** – Implements database interactions.
    d. **ORM**, a.k.a Object Relational Mapping.
    e. JWT Tokens + Auth0 (JSON Web Key Sets).
2. Language : C# (CSharp)
3. External Packages:
    a. MediatR [See more details here]
        i. MediatR Library is used to reduce dependencies between objects, by implementing a `mediator pattern`. It allows in-process messaging [command and query], but it will not allow direct communication between objects. Instead of this it forces communication via MediatR only, such as classes that don't have dependencies on each other, that's why they are less coupled. [Warade, 2020]
    b. FluentValidation
        i. FluentValidation is a .NET library for building strongly-typed validation rules for the `objects` processed in our api calls. [Skinner, 2021]
    c. AutoMapper
        i. AutoMapper is a simple little library built to solve a deceptively complex problem - getting rid of code that mapped one object to another. [Bogard, 2022]
    d. Swashbuckle (Swagger) and OpenAPI.
        i. OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs. An OpenAPI file allows you to describe your entire API, including: [Swagger, 2022]
            1. Available endpoints (/users) and operations on each endpoint (GET /users, POST /users)
            2. Operation parameters Input and output for each operation
            3. Authentication methods
            4. Contact information, license, terms of use and other information.
    e. NUnit
        i. NUnit is an open-source unit testing framework for the .NET Framework and Mono. It serves the same purpose as JUnit does in the Java world. [Wikipedia, 2023]
4. Development/Testing Database: Sqlite (local)
5. Development Database: PostgreSQL.

    a. **Docker** Container for the local development. (local)
        i. Postgres Database Server is running in the container.
        ii. PostgreSQL Database is created and maintained on the contained server.
6. Production: [foreseen for production]
    a. Database hosted in AWS.
        i. Relational Database Service (RDS) for PostgreSQL.
    b. Api/Backend hosted in AWS.
        i. EC2, a.k.a .Elastic Compute Cloud.
        ii. Elastic Beanstalk, and/or API Gateways.
    c. File Storage hosted in AWS.
        i. Amazon S3.

Architecture

(See Diagram)

1. **Clean Architecture**

   The code is structured in Layers, which have their own responsibility and interact with each other in a certain way. It's also known as Domain Driven Development. (DDD) Often seen has a design pattern that increases "*testability, UI independence, database independence and independence from any external agents [...]*".

    a. **Layers**
        i. **Presentation** – In our case, the API is the entry point for the client-app.
        ii. **Application** – Business Logic, containing everything from the controller handlers, data transfer objects, validation logic, etc.
        iii. **Domain** – Entreprise Business Rules, i.e. Domain entities and domain events.
        iv. **Persistence** – Repository pattern with IdentityDbContext.
        v. **Infrastructure** - External services (i.e. Payment Service, Email Service) and third-party applications.
    b. **Rules**
        i. The Api layer knows the Application layer.
        ii. The Application layer knows the Domain layer.
        iii. The Domain layer is the central piece for the business logic.
        iv. The Persistence layer knows the Application layer, and hence the Domain Layer indirectly.
        v. The Infrastructure knows the Application layer, and hence the Domain Layer indirectly.

2. **CQRS Pattern**, a.k.a Command and Query Responsibility Segregation.
    - **a.** Separating the reads (query) and writes (command) from the database inside the business logic.
    - **b.** Performance, Scalability and Security.
    - **c.** Having a read-optimized database for queries and write-optimized database for commands is an example of scalability and significant increase in performance. – This is not our case for the MVP.

3. **Mediator Pattern**.

    Using an open-source library that implements this pattern wonderfully. [***Mediatr***] The goal is to encapsulate the interactions between a set of objects. In our case, between the Api, the Application and the Persistence.

    a. Workflow Example:
        i. Controller receives a request from the client for the listing of Events on our platform. (Read – Query)
        ii. Mediator dispatches the query object to the Application.
        iii. Application uses the DataContext from Persistence to access the Database and returns a list of events, empty or not, or throws an exception.

4. Backend Overview - **Visual**

    (See next page)

    Notes –
    a. You can see the backend overview diagram, using the ***draw.io*** editor here.
    b. You can see the database class diagram, on github, here.

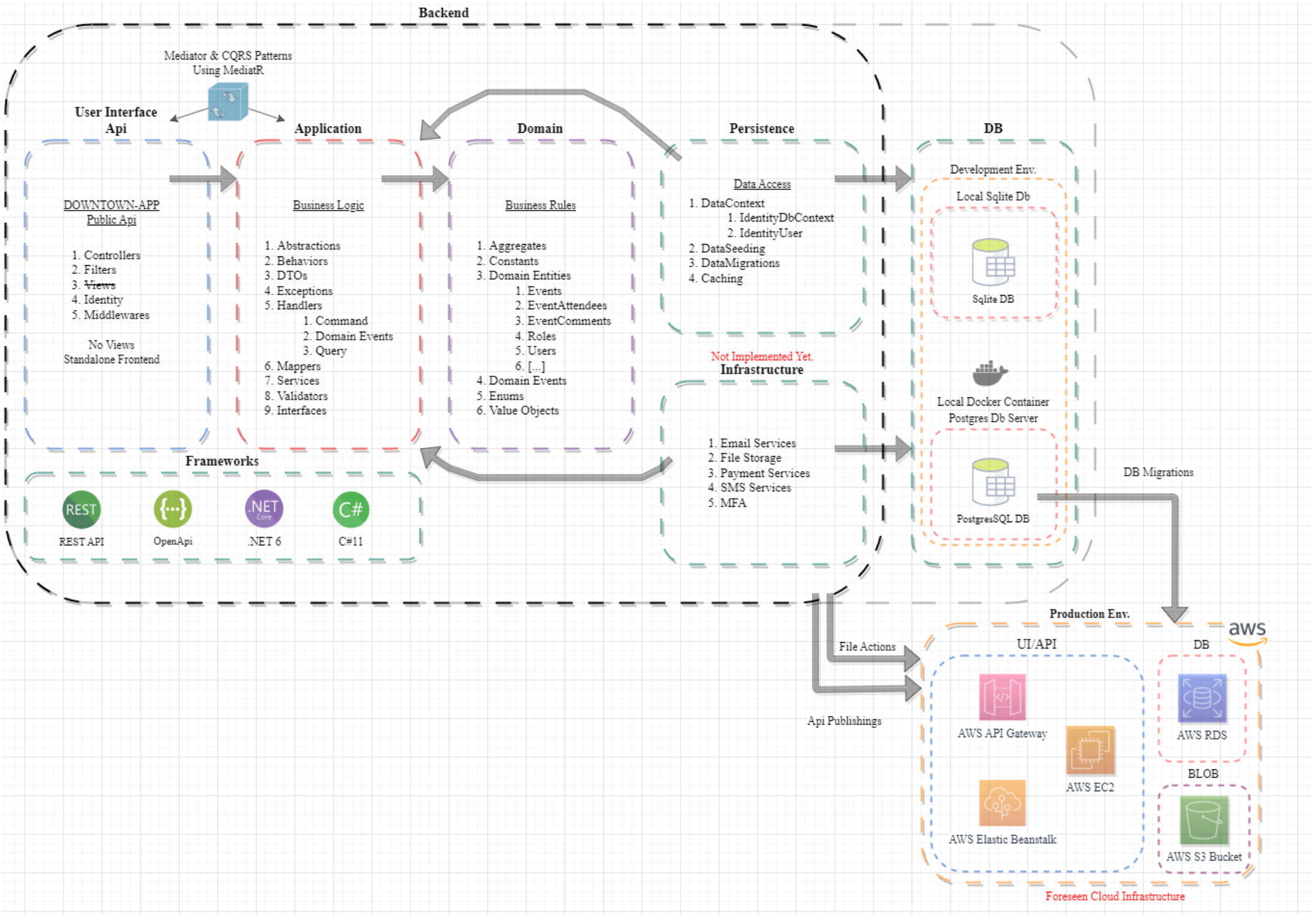

Figure 1: Backend Overview Visual.

References/Useful Links

[1] – Gil Vegliach. (Sep. 2018). Clean Architecture by Uncle Bob: Summary and Review. Retrieved from: https://clevercoder.net/2018/09/08/clean-architecture-summary-review/

[2] – Amish Patel. (Sep. 2021). Clean Architecture with .NET and .NET Core — Overview. Retrieved from: https://medium.com/dotnet-hub/clean-architecture-with-dotnet-and-dotnet-core-aspnetcore-overview-introduction-getting-started-ec922e53bb97

[3] – Mehmet Ozkaya. (Sep. 2021). CQRS Design Pattern in Microservices Architectures. Retrieved from: https://medium.com/design-microservices-architecture-with-patterns/cqrs-design-pattern-in-microservices-architectures-5d41e359768c

[4] – Jason Taylor. (Dec. 2022). Clean Architecture Solution Template for .NET 7. Retrieved from: https://github.com/jasontaylordev/CleanArchitecture

[5] – Mohsen Rajabi. (Feb. 2022). How in MediatR we can have events (Notifications) async and completely real Parallel. Retrieved from: https://medium.com/@mohsen_rajabi/how-in-mediatr-we-can-have-events-notifications-async-and-completely-real-parallel-2068f24912e6

[6] – Atul Warade. (Jul. 2020). Introduction To MediatR Pattern. Retrieved from: https://www.c-sharpcorner.com/article/introduction-to-mediatr-pattern/

[7] – Jeremy Skinner. (2021). Fluent Validation with .NET. Retrieved from: https://docs.fluentvalidation.net/en/latest/

[8] – Jimmy Bogard. (2022). Auto Mapper. Retrieved from: https://automapper.org

[9] – Swagger OpenApi. (2022). What Is OpenAPI? Retrieved from: https://swagger.io/docs/specification/about/

[10] – Wikipedia. (2023) . NUnit. Retrieved from: https://en.wikipedia.org/wiki/NUnit

[11] – Charlie Poole, Rob Prouse. (2023). What Is NUnit? Retrieved from: https://nunit.org