MeowStory App

Problem

Develop a web application that allows authors to publish stories. The app is called MeowStory and publishes stories about Cats.

Analysis

System has been analysed based on my personal assumptions. Authors can create, edit, delete, and submit stories. Story was assumed to be short story about Cats. When author submit a story, system assign it to a reviewer to review then decide to approve or reject. When story is approved, it goes public and when it is rejected, it goes back to the author with a proper comment. Users can see only public stories which were approved. Users can vote and add comments to stories. Draft stories are stories that not submitted yet. System should notify authors when their submission is reviewed via email. And authors are notified when users add comments. Reviewers are notified by email when they get new assigned stories to review.

Design

System is composed of two main apps which are backend web API and frontend app.

The API solution architecture is built based on Domain-Driven Design principles following structure of Clean Architecture (Onion Architecture).

The CQRS pattern is used to communicate between Presentation layer and core application layer. The solid principles are applied to have flexibility in some directions.

The solution has only the implementation of Authentication and Story use cases.

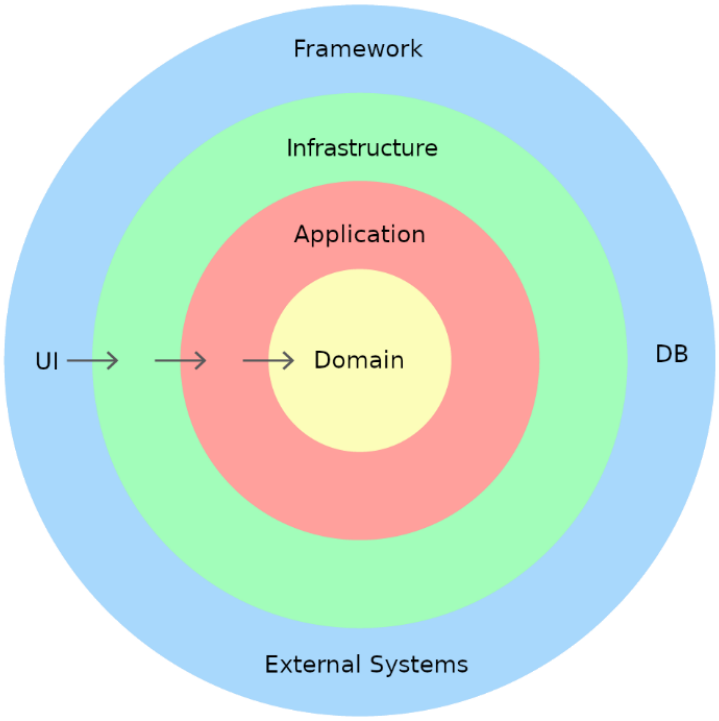


Figure 1 Clean Architecture

Domain

Domain layer consists of classes represents domain contexts and related entities. Domain Entities are the main business entities. Each entity encapsulates required general business rules and has public interface for each business scenario.

Story entity has properties such title, description, content, ...etc, and it has operations such as submit, approve, reject, vote, ...etc. The other entities are supposed to have similar structure.

Domain also has abstract events that are triggered with some rules such as “Story Published Event” that is triggered when story is approved.

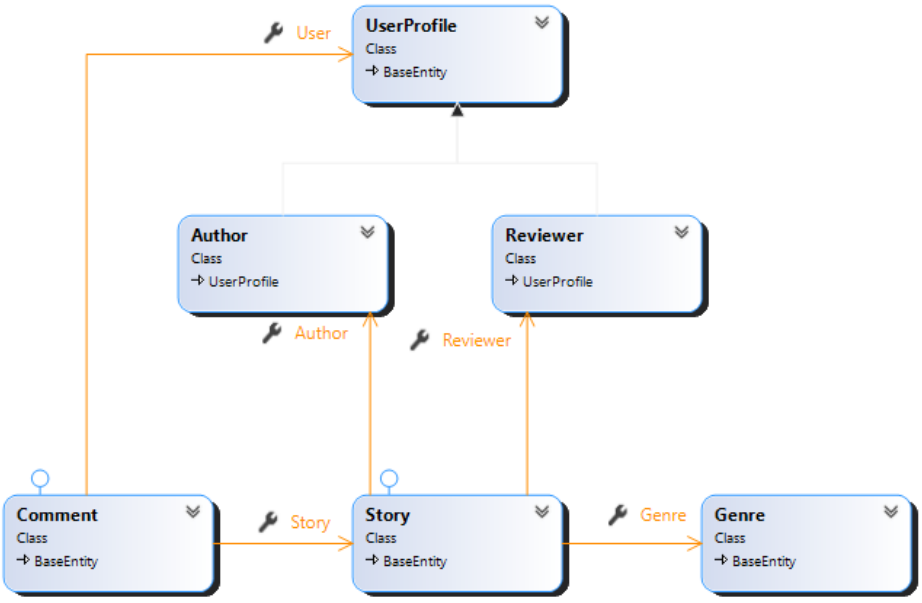
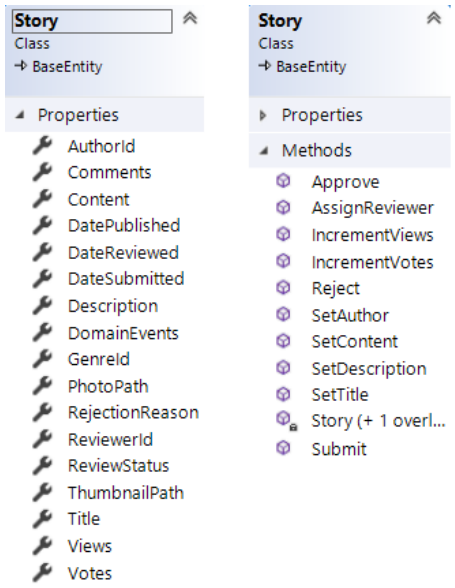


Figure 2 Domain Entities



Application

Application layer encapsulates the use cases and control the data flow from and to domain entities. Each use case has commands and queries. It has one use case for user authentication and use cases for each business entity. Each command has its own required data attributes and represents a specific use case and the same for queries. Queries doesn't expose domain entities, but it returns the response encapsulated in DTOs (data transfer objects) or view models objects.

Application layer has the implementation of domain event handlers.

It also has some common interfaces used to return system datetime, Id of current user, build pdf document. In addition to custom exception and Mediator pipeline objects and some other common types.

Infrastructure

This layer has the implementation of details that depends on frameworks and data persistence.

The data is persisted using MS Sql server and Entity Framework with code first approach. Each entity has one configuration class to describe the mapping details.

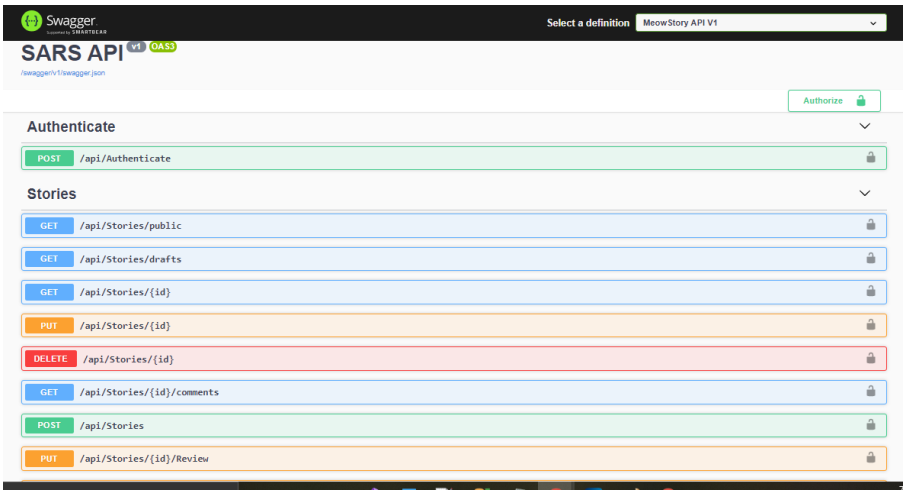
The authentication and authorization are implemented using Microsoft Identity framework. The authentication action takes username and password returns JWT token encapsulated with username, email, id, and role in object.

A custom authorization handler was created to handle problem in old asp core version related to role-based authorization but later found that issue was resolved in newer versions.

Presentation

Here is where user interacts interact with the system. MeowStory has WebApi built on top of ASP Core framework. It currently has two endpoints (Authenticate, Stories) the action handlers receive the request and delegate it to the application layer using Mediator pattern (CQRS). The API has friendly test UI built using Swagger library.

Swagger Url: https://localhost:44312/swagger/



Typical scenario

- User sends http request to a specific endpoint with payload via MeowStory frontend app or swagger or any other client.
- WebApi handles the request and check if authentication and authorization are required.
- WebApi sends the command to the application layer via Mediator Sender object
- The command goes into mediator pipeline which first logs the action, check user authorization, then send the command to the handler.
- Command handler executes the command and return proper response or failure.
    - Command handler validates the command input data
    - Call a method in the underline domain object to execute the business rule in case of command, but with queries it does query the data store using abstract version of Database context then project the data into DTOs.
- If exception was thrown during command execution, it gets handled by exception filter in the WebApi, filter logs the exception and return an error message with suitable http response code

Testing

Solution has unit and integration tests, but they cover only story use cases which are total of 54 tests.

NUnit framework and fluent assertions are used to write tests.

In unit test, we test the domain general business rules implemented in domain entities.

In integration test, we test the use cases with real data persistence.

Technologies

- .NET5
- ASP.NET Core 5
- ASP.NET Identity2, OAuth, JWT
- MS Sql Server
- EntityFramework
- CQRS(Mediator Pattern)
- FluentValidation
- AutoMapper
- NUnit
- Fluentassertions
- Swagger

Frontend App(Proposal)

Frontend app has not implemented yet but there is a conceptual design for it.

The app will be built using Angular Framework and NGRX state management engine (Redux implementation for Angular) for state management.

The project will consist of main parts:

- App Store: consist of Redux building blocks(states, reducers, actions, and selectors).
    - State:
        - Story[stories array, drafts array, selectedStory object, loaded flag]

        - Authentication [loggedInUserInfo]
    - Selectors:
        - Story[SelectStories, SelectDrafts, SelectStoryById]
        - Authentication[SelectToken, SelectUserRole, SelectUserName]
    - Actions
        - Story[LoadPublishedStories, LoadDraftStories, LoadStory, Create, Update, Review, Vote, Submit, AddComment, LoadPublishedStoriesSuccess, LoadDraftStoriesSuccess, LoadStorySuccess, CreateSuccess, UpdateSuccess, ReviewSuccess, VoteSuccess, SubmitSuccess, AddCommentSuccess]
        - Authentication[Login, Logout, LoginSuccess, LogoutSuccess]
    - Effects: make external calls to the WebApi and trigger another action with response payload
        - Story[LoadPublishedStories, LoadDraftStories, LoadStory, Create, Update, Review, Vote, Submit]
        - Authentication[Login]
    - Reducers: mostly handles success actions and update state by payload
- Components:
    - Login Component
    - Admin Module
        - Layout Component (for Admin, Author, Reviewer)
        - Stories List Component
        - Story Create Component
        - Story Edit Component
    - Layout Component(for normal users)
        - Stories List Component
        - Story Component
- Core:
    - Models[User, Author, Story, StoryItem, Comment]

Technologies

- Typescript
- Angular 11
- RxJs
- NGRX
- HTML5
- SASS

References

- https://github.com/ardalis/CleanArchitecture