

Devlog - MeetingPlanner

By Laurin Gobber

Contents

Used technologies and frameworks	1
Patterns used.....	1
Architecture overview	2
Class diagrams	3
Unit tests	4
UI-Design	4
Business Layer	5
Models.....	5
Data Access Layer	5
Report generation	5
Logging	6

Used technologies and frameworks

Technology	Use
VisualStudio 2022	IDE
C# 11 (for .NET 8)	Programming Language
WPF (Windows Presentation Foundation)	Used for building the UI
EF Core (Entity Framework Core)	Data Access + in-memory DB for UnitTests
SQLite	Database
QuestPDF	Report Generation
Log4Net	Logging
NUnit	UnitTests
Draw.io	UML Diagrams
MS Word	Protocol

Patterns used

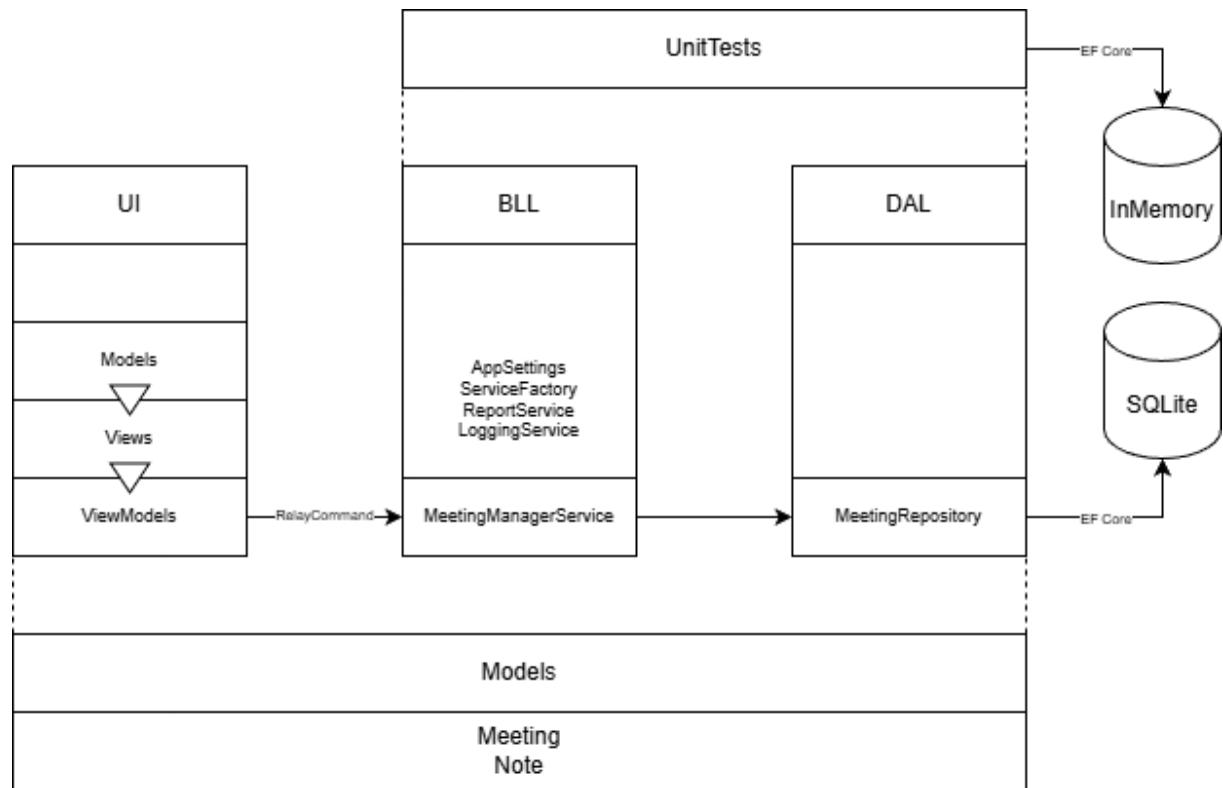
MVVM-Pattern to decouple views and business logic.

Factory-Pattern to provide Services (ServiceFactory Class).

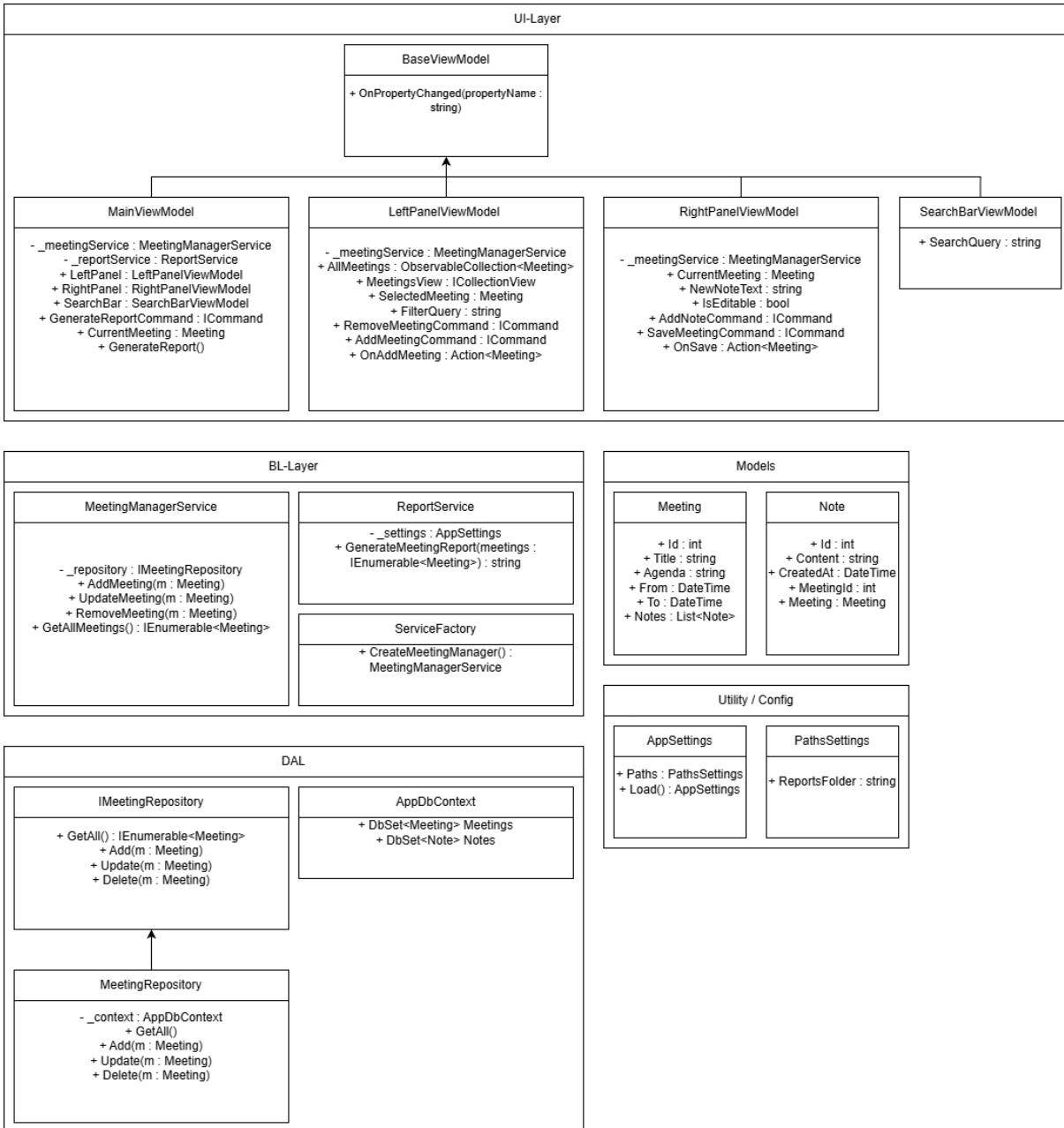
Repository-Pattern to define CRUD-Operations for EFCore.

AAA-Pattern for unit-tests.

Architecture overview

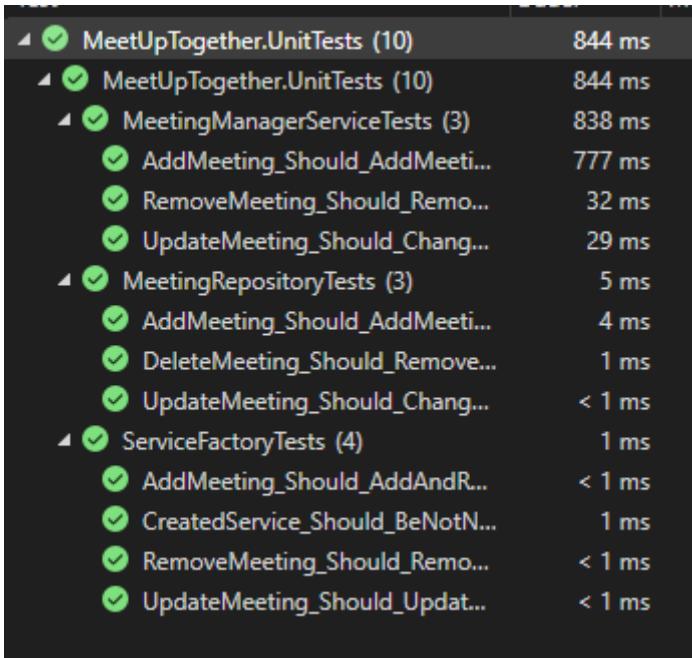


Class diagrams



Unit tests

The unit tests have been chosen to ensure the core feature, the CRUD-Operations, in each layer stay functional:



```
▲ ✓ MeetUpTogether.UnitTests (10) 844 ms
  ▲ ✓ MeetUpTogether.UnitTests (10) 844 ms
    ▲ ✓ MeetingManagerServiceTests (3) 838 ms
      ✓ AddMeeting_Should_AddMeeti... 777 ms
      ✓ RemoveMeeting_Should_Remo... 32 ms
      ✓ UpdateMeeting_Should_Chang... 29 ms
    ▲ ✓ MeetingRepositoryTests (3) 5 ms
      ✓ AddMeeting_Should_AddMeeti... 4 ms
      ✓ DeleteMeeting_Should_Remove... 1 ms
      ✓ UpdateMeeting_Should_Chang... < 1 ms
  ▲ ✓ ServiceFactoryTests (4) 1 ms
    ✓ AddMeeting_Should_AddAndR... < 1 ms
    ✓ CreatedService_Should_BeNotN... 1 ms
    ✓ RemoveMeeting_Should_Remo... < 1 ms
    ✓ UpdateMeeting_Should_Updater... < 1 ms
```

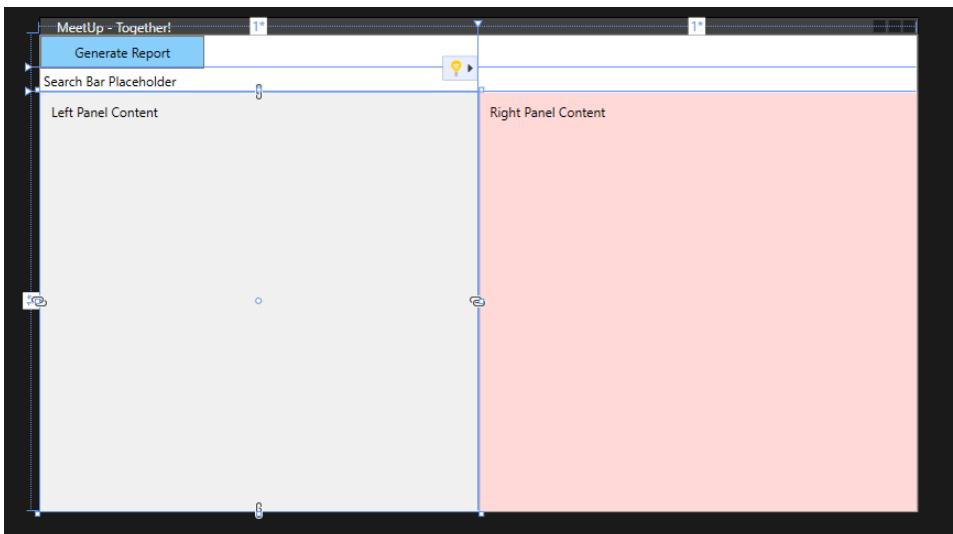
All unit tests are executed on an inMemory database, provided by EFCore. It is executed in the setup-step of each unittest-file.

```
var options = new DbContextOptionsBuilder<AppDbContext>()
    .UseInMemoryDatabase(databaseName: "TestDb_" + Guid.NewGuid())
    .Build();
```

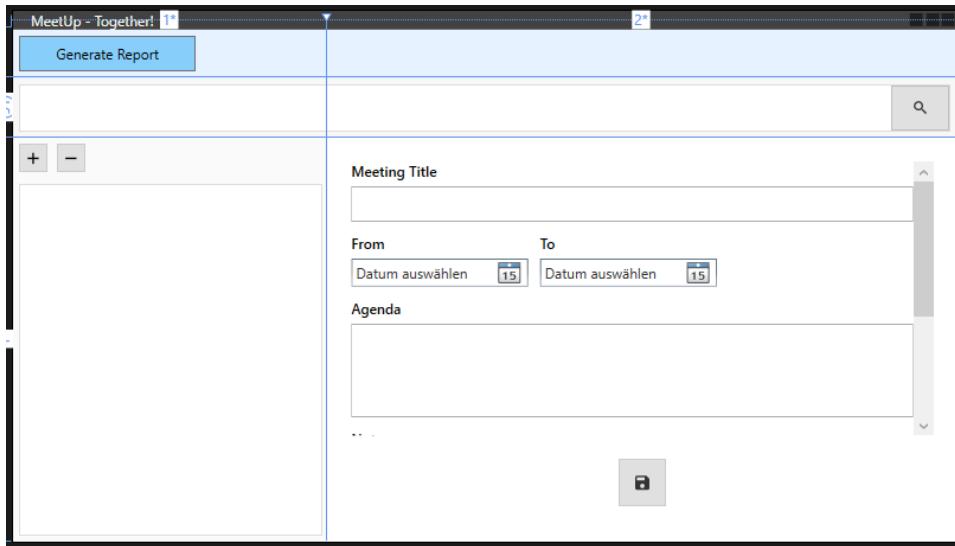
One extra unit-test has been added to ensure the ServiceFactory is providing each service correctly.

UI-Design

The UI-Design was done according to the MeetingPlaner-Specification and designed layer by layer:



These areas correspond to the different usercontrols found in .UI/Controls/.



Final View.

Business Layer

A new project with a class library template was added, in which the services were defined:

MeetingManagerService – Executes CRUD Operations in UI via Two-Way-Binding, validates data and calls DAL-Classes.

LoggingService – Provides class to write into log.

ReportService – Provides class to generate report.

ServiceFactory – Provides Services

Models

The models were added in a separate Class Library to decouple the UI, DAL and Business Layer while allowing all projects to access them. They are POCOs:

Meetings – holds all meeting data (including list of notes)

Note – holds note data

Data Access Layer

For persistence a SQLite-Database was used. It was accessed via the EntityFrameworkCore.

Report generation

Report generation was done via the NuGet-Package QuestPDF.

Logging

Logs are generated via Log4Net-framework in key areas, such as on startup and shutdown. Each user action is created aswell.