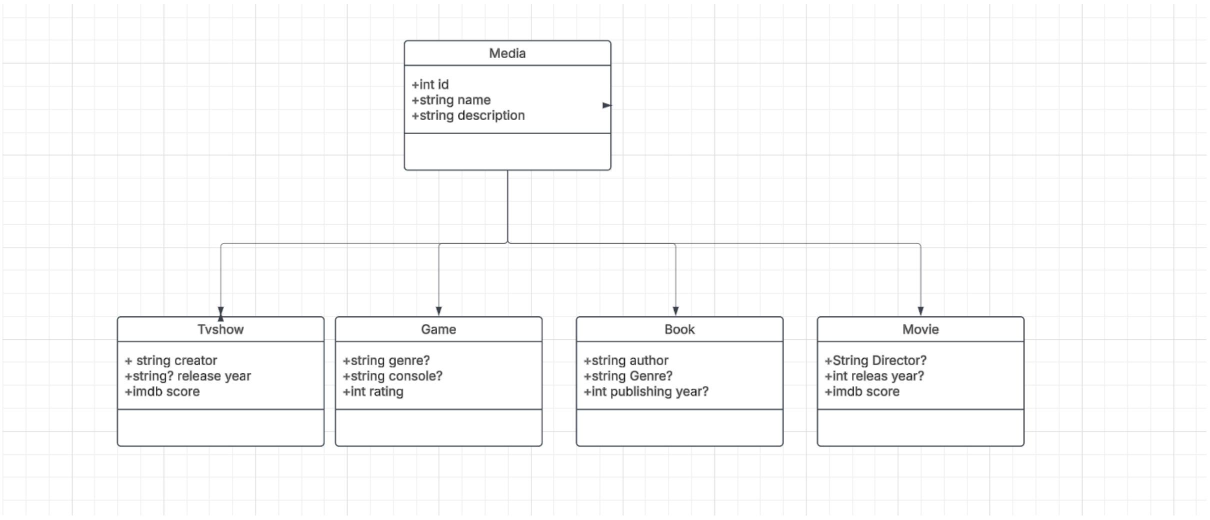
Report for Software Design Exam

1. The process

I started working on this project just after my fifth semester at Kristiania. I knew this was an exam I had to resit from semester three. I had different ideas in my head since then for what this project should be about. But I knew I wanted to make an easy to use application with very limited frontend (a console app) showing my skills and what I learned in the curriculum and since then. After much thought I landed on a Media application that uses all Crud principles and can display information about these types of media, later I decided to add a rating system for more complexity in the application.

So my first draft of the application (just in my head) was a media abstract class that contained data types that I knew could be reused for the different media types. Like int id, string name and string description. And had 4 classes that was going to inherit from this class, movie, tv-show, game and book. With added data Types that are specific to their class of media. There are of course many media types I could have added but I thought these would be a fitting number of classes for the application. Due to time crunch and the other work I had to do in this time the application only has 2 media types. But it would not be too hard to add the other types of media, since the code for each type is pretty similar to each other.

With this Information I made a very simple uml diagram so that I would be easier to at least know where to start on my application. This is what I made.



Then I started to code the application. I only started with a media abstract class and a movie class that was going to inherit from that one. Then I had a decision to make, do I want to create all the classes that are going to inherit from media and their respective logic or finish the entire movie side of the application and then start on the others. I decided to go for the second option, because I knew the logic in the code was going to be very similar for each of the different media types. I did not use test driven development a thing that came back to haunt me somewhat later. But more on that later in the report. After a while when I was done with the movie side of the application, I realized that there was not going to be time for more than two media types. So I made some changes to the abstract class to contain things like year of release and my Appscore. Since I decided to add a rating system for some more complexity to my application. Then I made the shows logic and to wrap it up I made a UI for the application.

2. Curriculum

1. Version Controll

In my project I decided to use git with mostly the cmd line, but also used their desktop app a few times. Git is usually a great tool to use when working in groups, with the possibility to work on different branches and easy to share progress with the group members. Since I was working alone I mainly used git for version control. If you use git right it is a great tool for version control, it makes it easy to revert changes to the last commit you had if your application don't work anymore. Or if changes you have done is something you don't need. Git helped me a lot during this project. But I should have committed and pushed a lot more often then I did.

Here is the link to my git repository:
https://github.com/robinruudW/SoftwareDesignExam_37

2. Uml class Diagram

To create a good application the architecture of the application is very important. It makes it much easier to know what you need to code and how each class should look before starting, if you create a complete uml class diagram for the entire application. As stated earlier I made a very simple uml class diagram when I started just to make it easier for myself when I started coding. Using a uml class diagram and making the entire application architecture would be in my opinion best practice for a huge project. Its also very helpful if you are working in groups, to make sure every team member understands the architecture and what needs to be in every class of the application. Since this is a relatively simple application and I am working alone, I did not make an extensive Uml class diagram. With movie logic classes and UI classes etc. or at least I did not do that on paper, I did go through my application architecture in full in my head before I started to code. So, I had a very good perception on how I wanted the application to look like.

3. Cnet modern features

I have used the ToString method in both my movie.cs and show.cs. this is to overwrite relevant data Tostring in my application. I decided to make no operator overload in this application. I have used an abstract class called media and 2 other classes that inherits from this. I have used datatypes with? Except the id and name in the media class.

4. Solid Principles

The solid principles is a set of five object orientated design principles that help make code more maintainable, Scalable and testable.

S – Single responsibility principle (SRP)

This principle is to ensure that every class in a system should only focus on one single responsibility. An example can be my movie.cs class. It only contains information and methods that are relevant to movies. If I want to add a movie to a database this should be handled in another class, in this case my movieLogic.cs.

Open – Closed principle

This principle is to make your code open for extension but closed for modification. It can make it easier to expand code without changing/rewriting old code. It's usually done through the use of Lists or interfaces. The example in my code is the media class, its an abstract class that two other classes inherit from. I was able to expand my application through 2 different media "type" classes. And makes it easy to add a new type of media.

L- Liskov Substitution principle (LSP)

Subtypes should be replaceable for their base types without breaking the program. Aslong as there are no compiler error in your code this principle should be followed in the code you are making if you are using c#.

I – Interface Segregation principle (ISP)

Classes should not be forced to implement methods they don't use. It means that no part of the project should have a method or properties that it does not use. I don't have any interfaces in the code. But none of my classes uses any methods or properties that it does not need.

D- Dependency Inversion principle (DIP)

High level modules should not depend on low level modules. Instead both should depend on abstractions. In my code my movieLogic and Showlogic depends on MovieDatabaseContext class, which is a concrete class. An improvement could be to make an own repository for each of those classes, that would make them depend on abstractions not concrete classes.

5. Layers

My application consists of 4 layers. An entity layer that contains my data models, it defines the data structures used in my application. It only defines data and does not contain any logic. Then I have a database layer. I decided to use Entity Framework Core with SQLite. It handles database interactions and is easy to use since it creates the database from my data models, so I don't have to set up the database in code. This layer is responsible for data storage and does not contain any business logic. Then we have the logic layer (business logic). This contains all the business logic for movies and shows. This layer interacts with the Database layer. And lastly we have the UI layer. This layer handles everything the user can do in my application. All of the users inputs and outputs, it displays the different menus and calls methods from the logic layer.

6. Data storage

For my data storage I use SQLite and Entity.framework.core (Efcore). I decided to use SQLite because its an embedded database, this means that I don't require a separate database server. The database in my case is just a file Movies.db. that is stored locally on my computer. Since my applications is very small I did not think I would need a large scale database and SQLite works just fine. Entity Framework core is an ORM (object relation mapper) that simplifies database interactions. Instead of writing raw SQL queries manually I can use c# LINQ to perform crud operations. So instead of "Select * FROM movies where name = 'the dark knight';" I can use var movie = _context.Movies.FirstOrDefault(m=> m.name =="the dark knight"). It also makes it so much easier to create the database tables. It maps my C# classes to database tables and saves a lot of time. It also sends in the object in my addmovie() through _context.Movies.Add(movie); and _contextSaveChanges(); instead of having to type out the SQL query manually. Overall it saves a lot of time.

7. Design patterns

Design patterns are patterns are reusable solutions to common software design problems. They help programmers build software that is simpler, more efficient and maintainable. Below, I describe the design patterns used in my application and some that could have been beneficial but were not implemented.

Dependency injection:

my application implements dependency injection by passing the movieDatabaseContext as a parameter to MovieLogic and ShowLogic. This decouples database access from business logic, improving modularity and making testing easier. Instead of creating database connections inside the logic classes, the dependencies are injected when an instance of the class is created. This reduces tight coupling between classes, makes unit testing easier and improves flexibility by enabling different database configurations.

Factory pattern:

My application follows the factory pattern indirectly through Ef core. EF core automatically manages object creation for movie and show, reducing manual instantiation and simplifying database interactions.

Façade pattern:

The UI in my application serves as facades. Simplifying complex operations for the users. It provides an interface for interacting with my logic classes.

MVx:

My application follows a structure that Is similar too MVP. My UI classes acts as views, logic classes acts as Presenters, and the entity classes represents the models.

Singleton pattern:

The application I made does not use singleton pattern, since my application does not need a globally shared instance. Instead I use dependency injection to manage dependencies efficiently. I could have used singleton for maybe a logging system. A single logger that is shared over the entire system. But I did not think of it when starting to build the application.

Decorator pattern:

I did not use the Decorator pattern either, but it could have been useful to dynamically extend the movie and show functionality without modifying the original classes. If I added user class with different privileges I could have used the decorator pattern to make “premium” functionality for premium users.

8. Multithreading /event based.

Multithreading and event based programming is not used in this solution. My application is very simple and follows a simple sequential execution model where tasks are completed before the next one begins. I could have used event based programming for a notification system if some of the crud operations were used. Due to time crunch I did not expend too any of these features in my application.

9. Testing

I have written some test for my application. Since I did not follow test driven development in this project I had some trouble setting up tests. But after some extensive troubleshooting I managed to make some tests. I decided to use xUnit for my test library. Xunit is a great framework for .NET. I have unit tests that focuses on testing individual methods in isolation, ensuring that each function is behaving correctly under different conditions. I have also made some integration tests that checks that multiple components work together as expected.

10. **Refactoring**

I had to do some refactoring during my time developing this application. For example I decided to add the rating system later in project. So all my UI and Logic classes was working without this complexity added. I later had to add a few data types to my models and logic to each of the logic classes to make this happen. And then change the UI to handle one more option. For rating the movie/show and let the UI show the new field of myAppRating. The database file as been recreated a few times when adding new datatypes in the models. When I thought I was done and tested for Buggs I found out that my application crashed if did not imput the right datatype in each of the prompts for adding a movie. Forexample a string in imdbScore or yearOfRelease. So I had to refactor every update and addmovie function to check trough a while loop if the right datatype was provided if it was not, it would prompt a message prompt correcting it. Since I added my xUnit project later in the process. My git did not recognize the testing project so I had to move git one folder out. My folder one out was SoftwareDesign with all my related school projects so I did a lot of refactoring in git and on my computer to make this right.

11. **Other to help improve grade**

I have nothing other too add with the limited time of a resit exam and working alone I think I have managed to made an pretty good application.

3. **Known Buggs**

I have tested the application too the best of my ability. The biggest bug I could manage to find is that Name is not a protected field so you can have multiple inputs in the database with the same name. Since most of the logic uses name to find what movie to edit or delete, it can take a few extra steps to delete or edit the right movie. **I have said this in my readme.md. but in MovieDataBaseContext.css the dbpath has to be set too where your db file is on your computer before testing.**

4. **Sources:**

I have mostly used slides from the Lectures. Everything from Canvas PG3302-1-23h and Canvas PG3302-1-24h. I have also used a lot of my own material from the classes.

Lucid.app/lucidchart for my uml chart.