**Faculty of Engineering and Information Technology**

**School of Computer Science**

31927 – Application Development with .NET

**JULY 2024**

**ASSIGNMENT 1 – Code Design Report Template**

|  |  |
| --- | --- |
| **Due date** | Wednesday 22 July 2024, 11:00am |
| **Demonstration** | Required in the lab/tutorial session |
| **Weight** | 35% |
| **Groupwork** | Individual |
| **Submission** | Complete project folder zip |
| **Submit to** | Canvas |

**Summary**

This report template needs to be filled out to receive marks for the “Code Design” & “Database Design” section of Assignment 1. Please include as much justification & detail as you feel is necessary to receive full marks. For full marks screenshots are encouraged alongside code references. This **must be included in your zip file submission**.

|  |  |
| --- | --- |
| Student Name | Oliver Warrick |
| Student ID | 24959254 |
| Lab Time | 7:30 |
| Lab Tutor | Davey |

**Justification**

For each criterion, include a justification as to how you have fulfilled it. Include references to your code & screenshots of how you applied the given code or database design principle.

|  |  |
| --- | --- |
| **Criteria** | LINQ is used in at least 3 different classes. |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | The LINQ methods are used in the data layer to setup database queries and used in the Services layer to manipulate results. |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 1.5/1.5 |

|  |  |
| --- | --- |
| **Criteria** | Nullable Reference Type checking is enabled in all files |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | Nullable reference type checking is enabled on a project level to statically ensure null’s aren’t used in a way which may result in a runtime exception. There are no null check warning in the project. |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 1/1 |

|  |  |
| --- | --- |
| **Criteria** | Unit tests are written for at least 3 database related classes. These should test both success & error scenarios with mocks for the database queries. |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | Comprehensive and passing units tests are written for the DoctorRepository, PatientRepository and AdminRepository. Unit tests were also written for Seeder and TableLayoutService as an aid during development |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 3/3 |

|  |  |
| --- | --- |
| **Criteria** | Helpful comments (At least 1 per method) |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | Comments are used extensively within methods to explain logic, design decisions, unexpected behaviour and summarize behaviour, they are also used to document public functionality of interfaces and classes. Some methods, such as those which are both very simple and private, don’t have comments as I think good naming practices are sufficient to convey intention. |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 0.5/0.5 |

|  |  |
| --- | --- |
| **Criteria** | Appropriate indenting and whitespace |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | Automatic formatting was used during development to ensure formatting in a way that is consistent with other code in the C# ecosystem, for example by using correct indentation, putting braces on their own lines and spaces around operators are some key words. |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 0.5/0.5 |

|  |  |
| --- | --- |
| **Criteria** | Consistent and appropriate C# naming convention used |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | All variables, field, properties, types, methods and interfaces use the correct C# convention, such as ‘I’ prefix on interfaces, ‘Extensions’ suffix on extension classes, correct tenses and plurality, and minimal abbreviations. Pascal case was used on types and public properties and methods, and camel case with a underscore prefix on private fields. |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 0.5/0.5 |

|  |  |
| --- | --- |
| **Criteria** | The unit of work pattern is used in all database classes |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | All methods that write to the database do it through the unit of work pattern, this is done to ensure database consistency and, along with the repositories, provides an abstraction layer above entity framework to allow for easier testing. |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 3/3 |

|  |  |
| --- | --- |
| **Criteria** | Strong OOP principles used.   * At least 1 example of inheritance * At least 1 example of method overloading is used * At least 1 example of method overriding is used * At least 1 example of an extension method * At least 1 example of a generic * At least 1 example of a delegate |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | Inheritance is used in the repository classes to share common data and functionality among all repositories    Method overloading and extension methods are used as part of the menu builder interface to provide default validation, this is done with extension methods so that any menu builder implementation would get the overloads, not just the concrete ConsoleMenuBuilder that I implemented.    Method overriding is used in the repositories to add additional functionality to the Add method when adding a new model.    Generics are used in the TableColumnFactory to generate a column list for any db user model    A custom delegate called Menu is used throughout the program to represent a method that will display a menu a return the next menu the user has navigated to, or null if they wish to exit.    Some of the other advanced features used are:   * the Expression type to allow for inferred column types from lambdas in the table builder * a fluent interface to allow creation of menus while restricting unwanted usages * Implemented IEnumerable to allow key value collection initialization of TableColumns |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 3/3 |

|  |  |
| --- | --- |
| **Criteria** | Use of Low coupling & high cohesion |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | All services are constructed using dependency injection and are behind interfaces, concerns are kept separated and classes follow the single responsibility principle, the interface boundaries were chosen to allow for maximum cohesion. For an example, there is a set of menu builder interfaces which all the modules use to create their menus and there is a concrete ConsoleMenuBuilder that knows how to display a menu using the console, it would be possible to write a new menu builder that displayed via windows forms, or generated html and the business logic in the modules would not have to change at all, the same is true for all other services. By putting interfaces in front of services the code is made more extensibility and easier to test. The ‘modules’ have static calls between their own menu methods, but use the service locator pattern to call into other modules, this allows for them to be written quickly and easy to read and understand while still maintaining loose coupling to other parts of the system.  Some of the other things done to improve the non-functional qualities of the code are   * Using helper classes and extension methods to share behaviour and composition to share data, rather than achieving the same means through highly coupled inheritance. Particularly but using extension methods on interfaces in place of base classes, this allows concrete classes to avoid many layers of inheritance and avoids combinatorial explosion when combining multiple behaviours. * Using the least restrictive parameter types possible (e.g. IEnumerable instead of List) to allow the most flexibility of the caller, and using the most specific return type where possible to give the caller the most information possible, with the exception of interfaces. |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 2/2 |

|  |  |
| --- | --- |
| **Criteria** | At least 3 tables in the database with foreign & primary keys setup correctly. |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | Four tables are used. The first is the table for all users, having a base users table was done instead of having the common fields on each of the user subtypes to ensure uniqueness of ids, make looking up a user of unknown type easier, and increase code reuse through composition. The next three are the user subtype tables with use a single value as both the primary and foreign key to Users, this is to ensure that there aren’t multiple subtypes using the same user, the Patients table also has a nullable FK to doctors. The forth table is appointments which has an FK to both patients and doctors, a new id is used for appointments rather than a composite key to allow for multiple otherwise identical appointments. |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 1/1 |

|  |  |
| --- | --- |
| **Criteria** | Minimum 10 Rows of Seed Data Exists In Each Table |
| **Justification**  Include code, screenshots & any written documentation on how you met this criteria. | Seed data is generated automatically if the database is empty by randomly combining options for parts of each field. E.g. there is a set of first names and last names and when a new name is needed a random one of each is chosen and combined to get the name. |
| **Recommended Mark**  Based on your justification, what mark do you believe you deserve. | 1/1 |