ASSIGNMENT 3

Student: Andrei Branga

**Group: 30432**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

# Use Java/C# API to design and implement a client-server application for a news agency. The application has 2 types of users: the readers and the writers.

# Functional Requirements

The application has 2 types of users: the readers and the writers. The readers can view a list of articles, read an article and do not need to login in order the use the application. The writers need to authenticate in order to create, update or delete articles. So the writer accounts are preset by the application developer and cannot be altered.

An article has the following components:

● Title

● Abstract

● Author

● Body

The application must support multiple concurrent users. If a writer posts a new article, the readers must see it in the list of articles in real time, without performing any refresh operation.

● The application must be client-server.

● Use the Observer design pattern for updating the list of articles in real time

● For sending data from the client to the server use JSON serialization.

● When writing an article, show a list that supports multi-select for choosing the related articles.

# Non-functional Requirements

* Usability – the application should be easy to use for the end-user
* Testability – the application should be very well tested

2. Use-Case Model

Use case: Publish article

Level: user-goal level

Primary actor: Writer

Main success scenario:

1. The writer is successfully logged in
2. The writer fills the form
3. The writer presses the publish button which will make the article go live

Extensions: The server is not running at the moment.

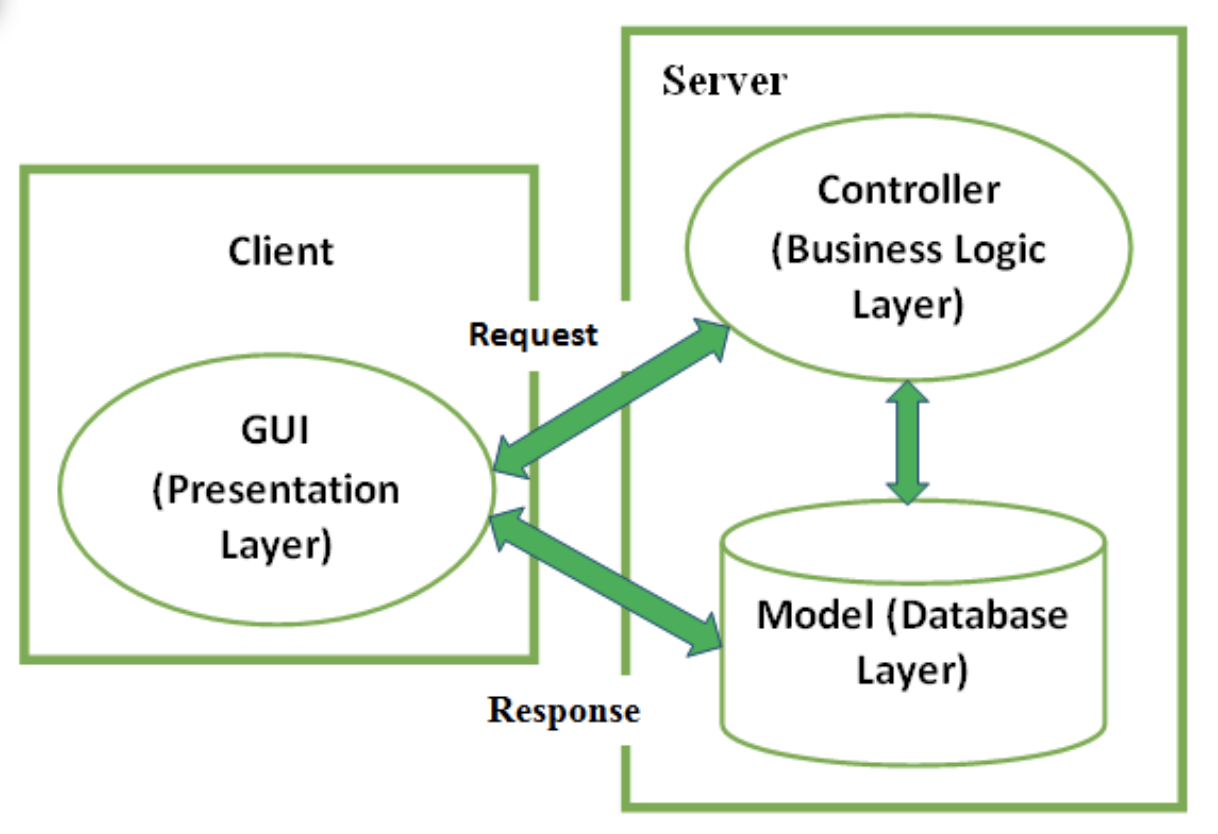
3. System Architectural Design

**3.1 Architectural Pattern Description**

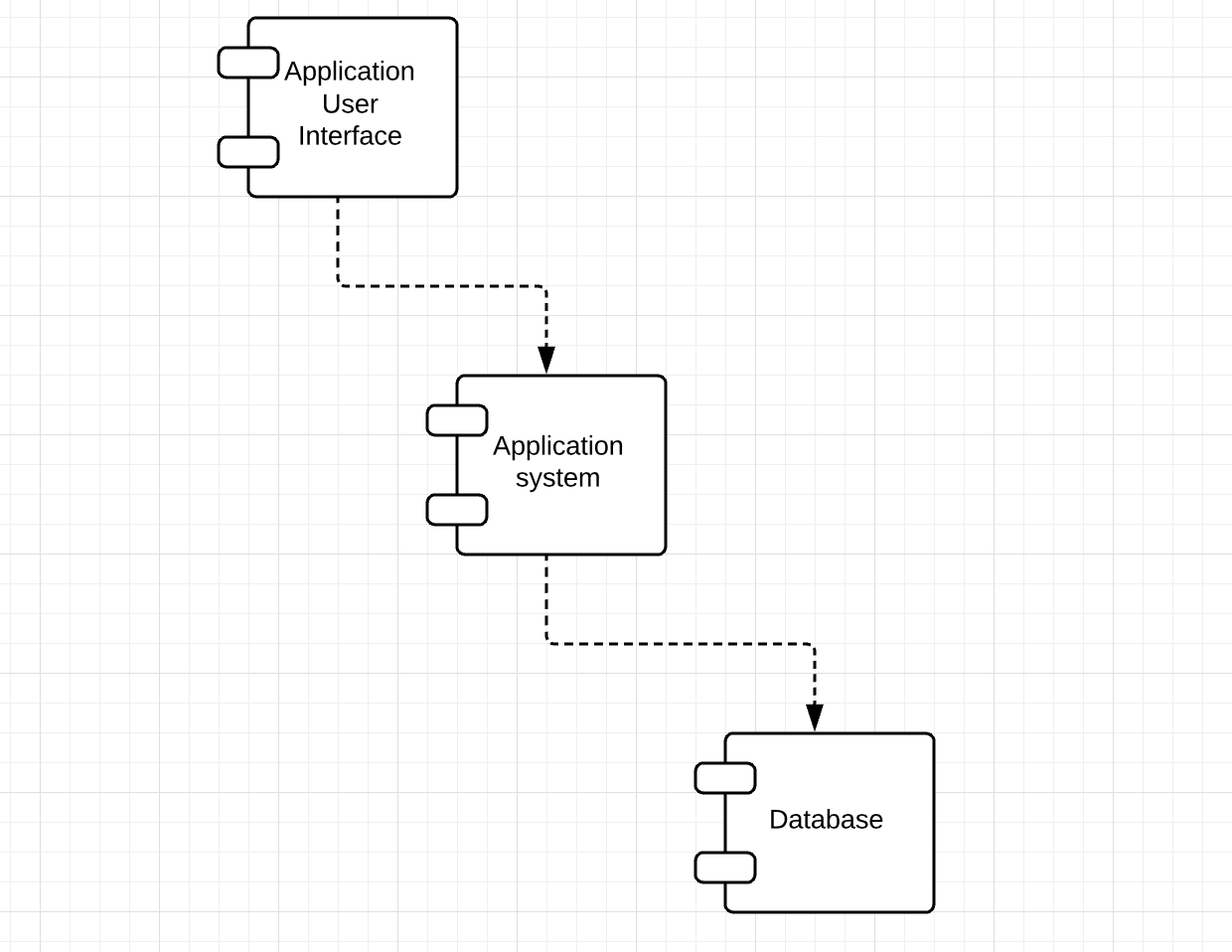
The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.

**3.2 Diagrams**

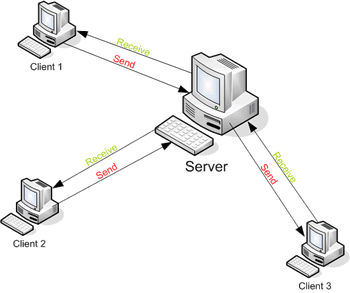
**Package Diagram:**

****

**Component Diagram:**

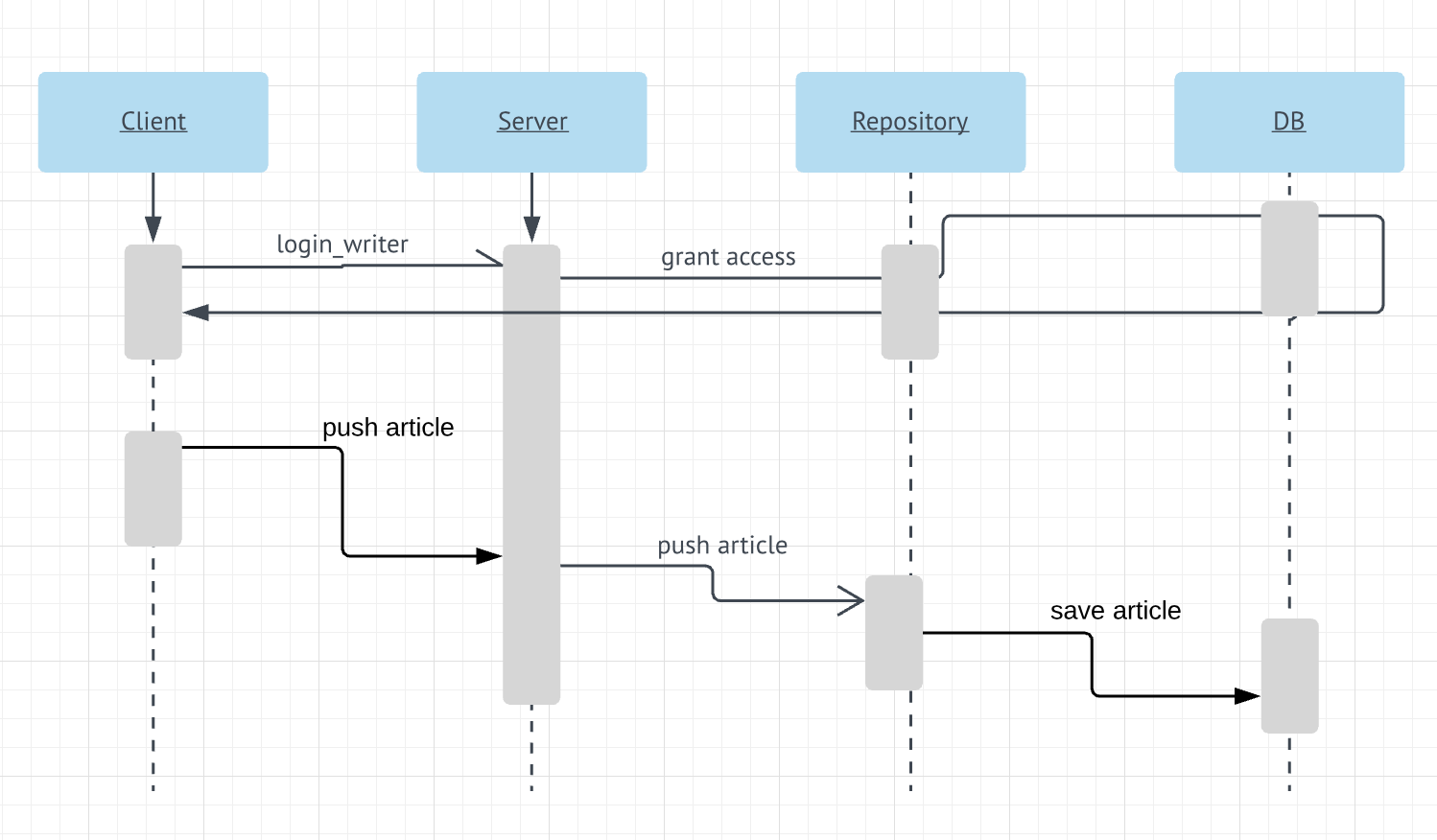
****

**Deployment diagram:**



4. UML Sequence Diagrams

Teacher adds a mark for a student:



5. Class Design

**5.1 Design Patterns Description**

Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific business rules associated with the request. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request.

**OBSERVABLE DESIGN PATTERN:**

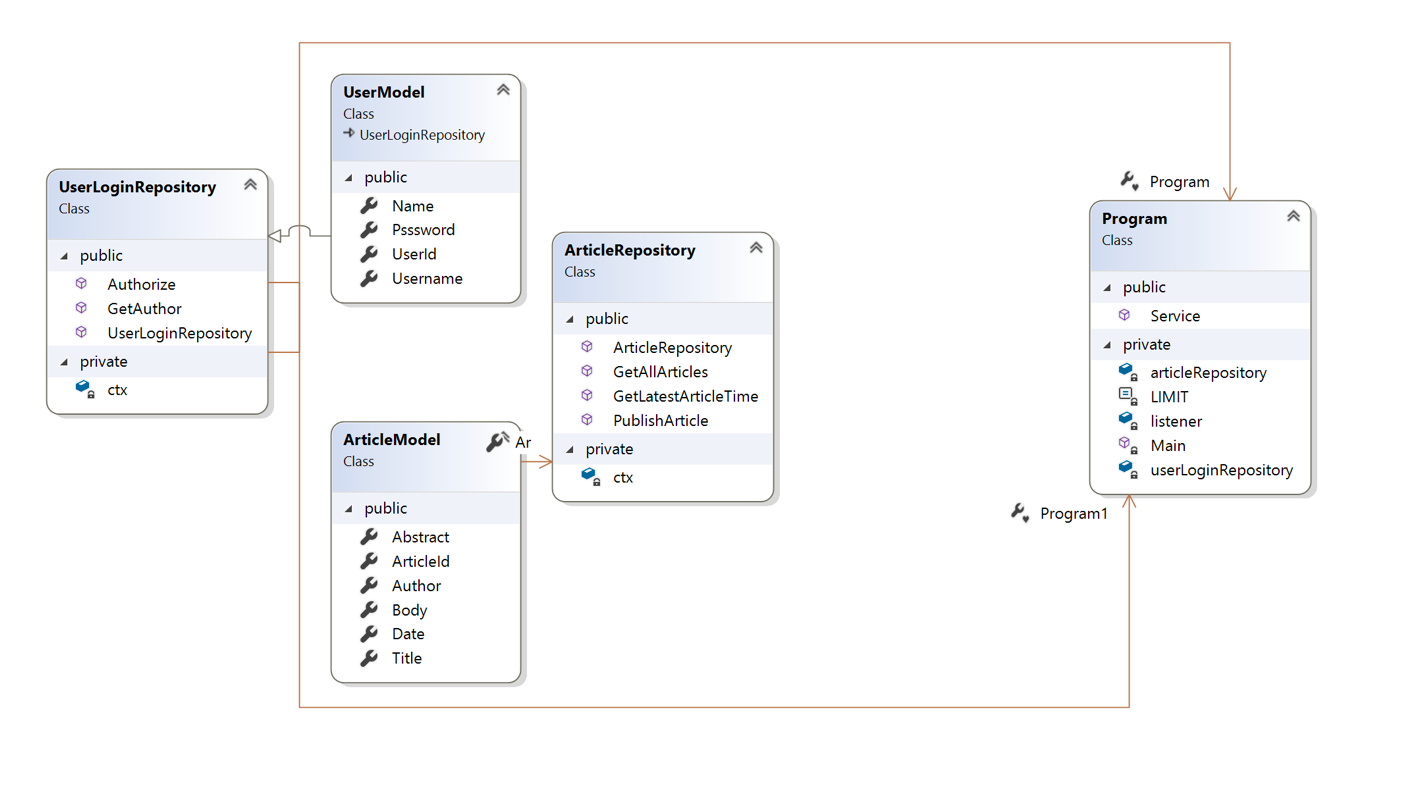
The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

It is mainly used to implement distributed event handling systems, in "event driven" software. Most modern languages such as C# have built in "event" constructs which implement the observer pattern components, for easy programming and short code.

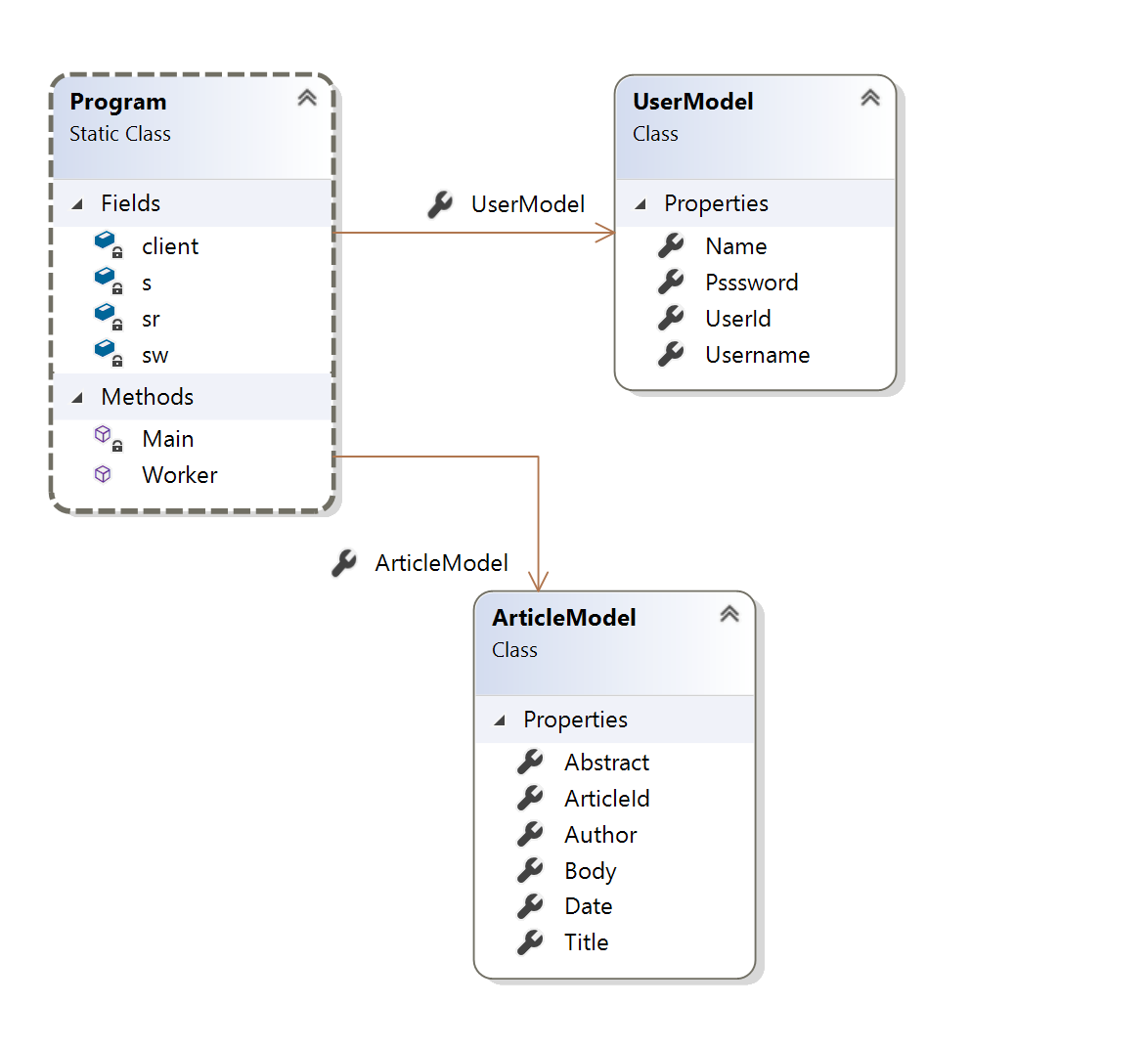
The observer pattern is also a key part in the familiar model–view–controller (MVC) architectural pattern. The observer pattern is implemented in numerous programming libraries and systems, including almost all GUI toolkits.

**5.2 UML Class Diagram**

Server:



Client:



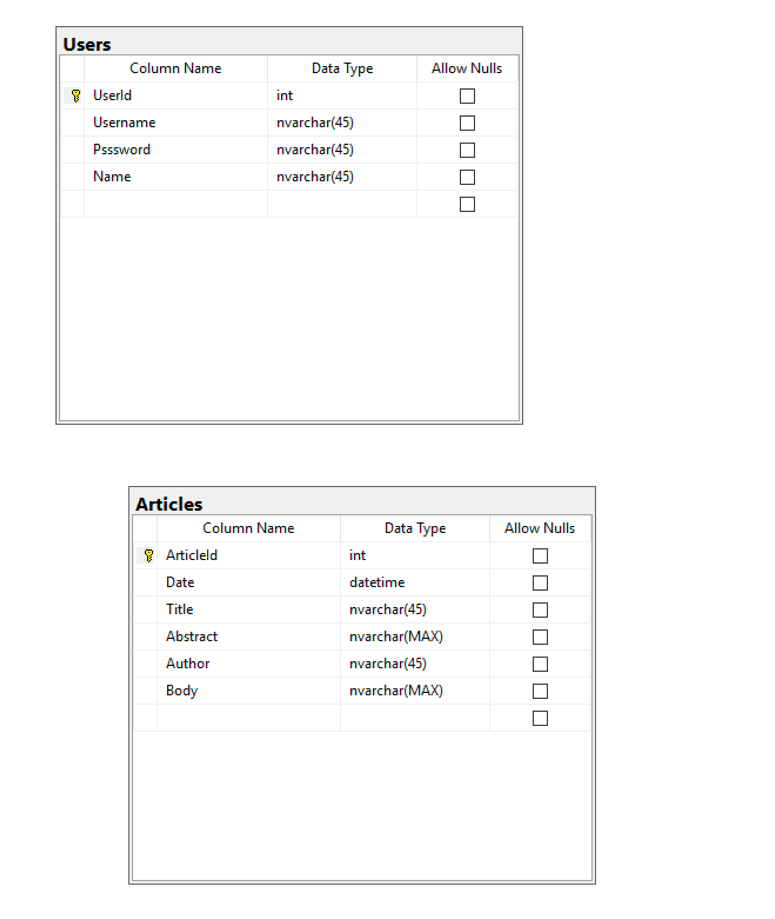
6. Data Model

An ORM class model is part of a data access layer first and maps the data object’s properties to a class(entity). If any conflict of interests arises between DAL and BLL, we will make changes to the BLL entities.

I will use the Entity Framework to model the data access layer:

Entity Framework (EF) is an object-relational mapper that enables .NET developers to work with relational data using domain-specific objects. It eliminates the need for most of the data-access code that developers usually need to write.

Entity Framework allows you to create a model by writing code or using boxes and lines in the EF Designer. Both of these approaches can be used to target an existing database or create a new database. This short video explains the differences and how to find the one that is right for you.



7. System Testing

The application will be covered by UnitTests.

Check that your code is working as expected by creating and running unit tests. It’s called unit testing because you break down the functionality of your program into discrete testable behaviors that you can test as individual units. Visual Studio Test Explorer provides a flexible and efficient way to run your unit tests and view their results in Visual Studio. Visual Studio installs the Microsoft unit testing frameworks for managed and native code. Use a unit testing framework to create unit tests, run them, and report the results of these tests. Rerun unit tests when you make changes to test that your code is still working correctly. When you use Visual Studio Enterprise, you can run tests automatically after every build.

Unit testing has the greatest effect on the quality of your code when it’s an integral part of your software development workflow. As soon as you write a function or other block of application code, create unit tests that verify the behavior of the code in response to standard, boundary, and incorrect cases of input data, and that check any explicit or implicit assumptions made by the code. With test driven development, you create the unit tests before you write the code, so you use the unit tests as both design documentation and functional specifications.

8. Bibliography

https://msdn.microsoft.com/en-us/library/hh694602.aspx