1.Technologies and Design Patterns

1.1. List of Technologies

- ASP.NET MVC Framework
- NoSQL : MongoDB
- IoC (Dependency Injection) : Ninject
- Unit Testing : Moq

1.2. Design Patterns

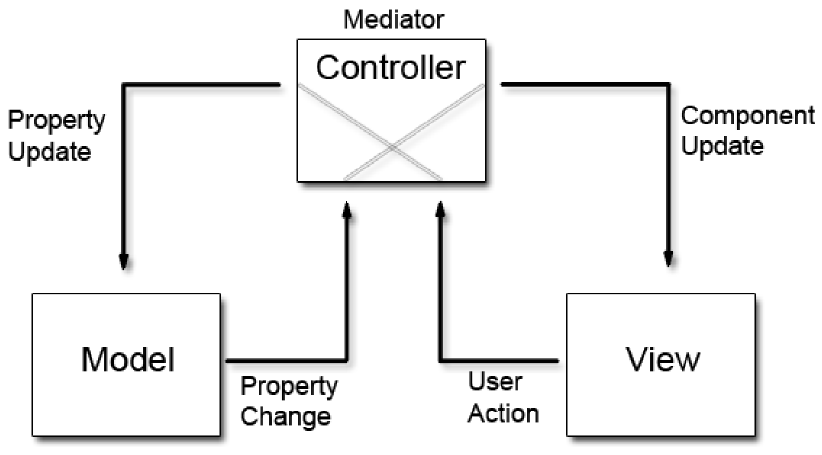
- MVC architectural pattern for ASP.NET MVC
- Repository pattern for data access
- Dependency Injection pattern for Ninject

2. Explanation of Architecture and Design

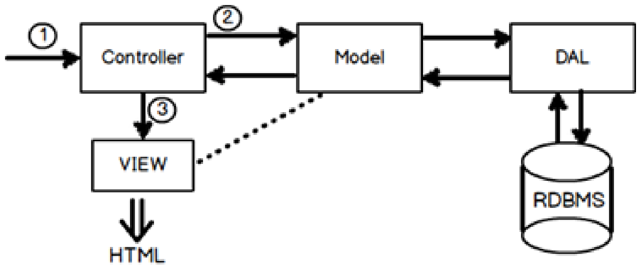
2.1. MVC architectural pattern

1) Definition

   Model-view-controller (MVC) is a pattern used to isolate business logic from the user interface. Using MVC, the Model represents the data of the application and the business rules used to manipulate the data, the View corresponds to elements of the user interface, and the Controller manages details involving the communication between the model and view. The controller handles user actions and pipes them into the model or view as required.



- ASP.NET MVC request typical flow



Step 1: The first hit comes to the controller.
Step 2: Depending on the action controller creates the object of the model. Model in turn calls the data access layer which fetches data in the model.
Step 3: This data filled model is then passed to the view for display purpose.

2) Why
Separation of concerns (SoC)
From a technical standpoint, the organization of code within MVC is very clean, organized and granular, making it easier (hopefully) for a web application to scale in terms of functionality. Promotes great design from a development standpoint.

Easier integration with client side tools (rich user interface tools)
More than ever, web applications are increasingly becoming as rich as the applications you see on your desktops. With MVC, it gives you the ability to integrate with such toolkits (such as jQuery) with greater ease and more seamless than in Web Forms.

Works well with developers who need high degree of control
Many controls in ASP.NET web forms automatically generate much of the raw HTML you see when an page is rendered. This can cause headaches for developers. With MVC, it lends itself better towards having complete control with what is rendered and there are no surprises. Even more important, is that the HTML forms typically are much smaller than the Web forms which can equate to a performance boost - something to seriously consider.

Test Driven Development (TDD)
With MVC, you can more easily create tests for the web side of things. An additional layer of testing will provide yet another layer of defense against unexpected behavior.

2.2. Repository pattern

1) Definition

   A Repository mediates between the domain and data mapping layers, acting like an in-memory domain object collection. Client objects construct query specifications declaratively and submit them to Repository for satisfaction. Objects can be added to and removed from the Repository, as they can from a simple collection of objects, and the mapping code encapsulated by the Repository will carry out the appropriate operations behind the scenes. The repository pattern is an abstraction. It's purpose is to reduce complexity and make the rest of the code persistent ignorant. As a bonus it allows you to write unit tests instead of integration tests.

2) Why

   Unit Testing

   One reason being that I can use Dependency Injection. Basically I create an interface for my repository, and I reference the interface for it when I am making the object. Then I can later make a fake object (using moq for instance) which implements that interface. Using something like ninject I can then bind the proper type to that interface.The idea is to be able to easily swap out implementations of objects for testing purposes.

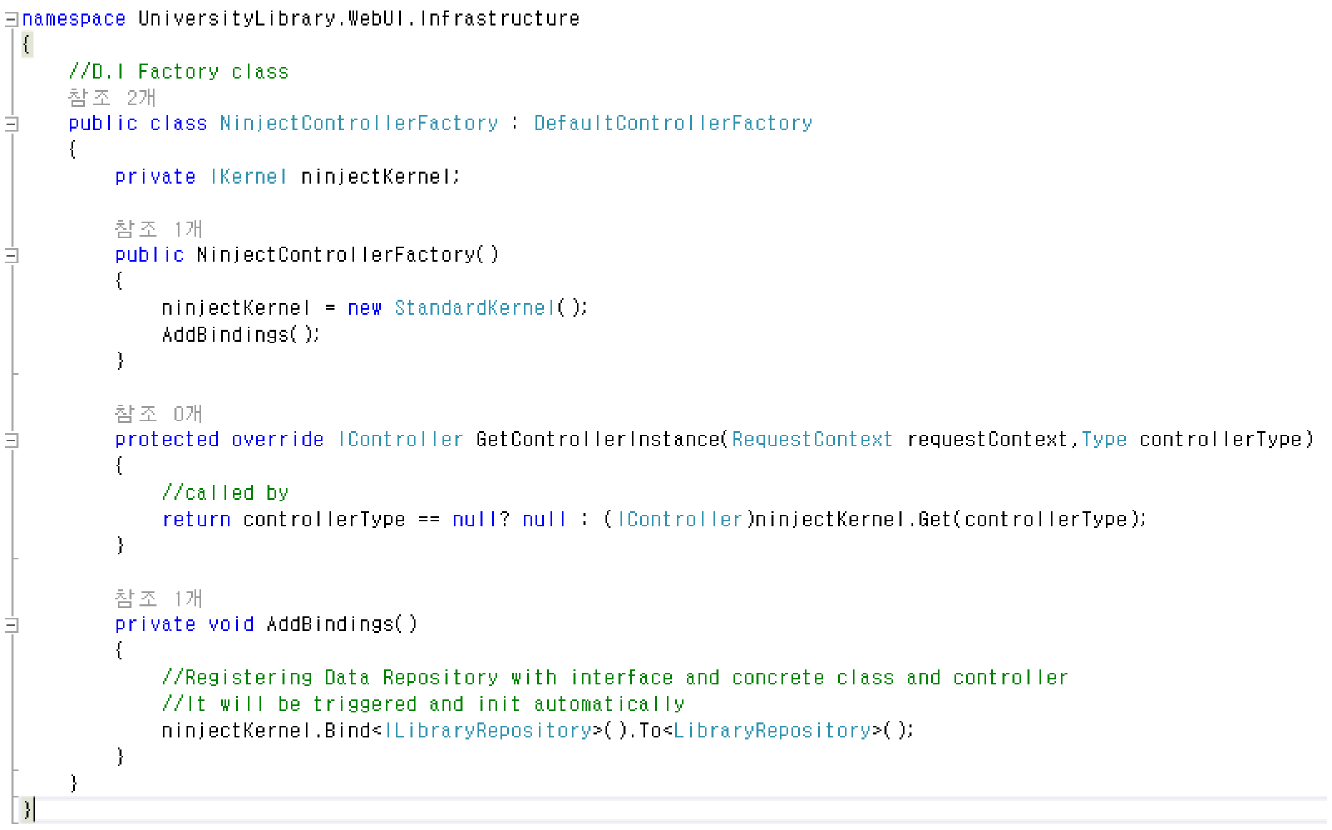
2.3. Dependency Injection pattern

1) Definiton

   DI is a software design pattern which allows the removal of hard-coded dependencies between classes and allows to change them at run-time. In other words, it allows you to load mock classes in test environments v.s real objects in production environments.

2) Why

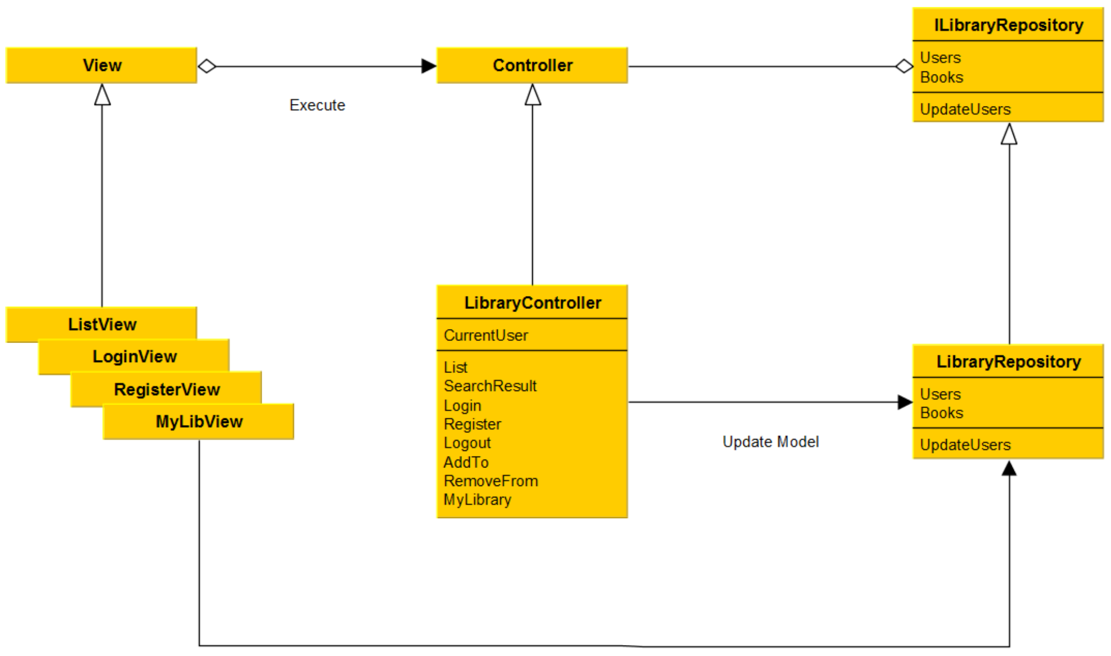
   The below code is fetching an instance of Controller that is responsible for generating all the controllers required in the MVC application. The Ninject container is also injecting the repository into the Controller Factory instance, thus making the Repository available for use in the View layer.This discovery instantiating and injection looks trivial enough, but imagine a project with multiple controllers and repositories and other related dependencies, trying to track all of those manually and wiring them in the container will soon become a nightmare and devs in the team will start trying to hack their way around. This is where Inversion of Control Containers (or IoC Containers) comes into picture. IoC containers when configured to map an interface to a concrete type, can generate a new concrete instance whenever required. This automates the process of dependency wiring. As long as everyone is referring to Interfaces and the IoC container knows how to get a concrete class for that Interface, we are good. Ninject is once of the newer and popular IoC containers. It's an open source project that is actively worked upon and has a vibrant community.



```
namespace UniversityLibrary.WebUI.Infrastructure
{
    //D.I Factory class
    참조 2개
    public class NinjectControllerFactory : DefaultControllerFactory
    {
        private IKernel ninjectKernel;

        참조 1개
        public NinjectControllerFactory()
        {
            ninjectKernel = new StandardKernel();
            AddBindings();
        }

        참조 0개
        protected override IController GetControllerInstance(RequestContext requestContext,Type controllerType)
        {
            //called by
            return controllerType == null? null : (IController)ninjectKernel.Get(controllerType);
        }

        참조 1개
        private void AddBindings()
        {
            //Registering Data Repository with interface and concrete class and controller
            //It will be triggered and init automatically
            ninjectKernel.Bind<ILibraryRepository>().To<LibraryRepository>();
        }
    }
}
```

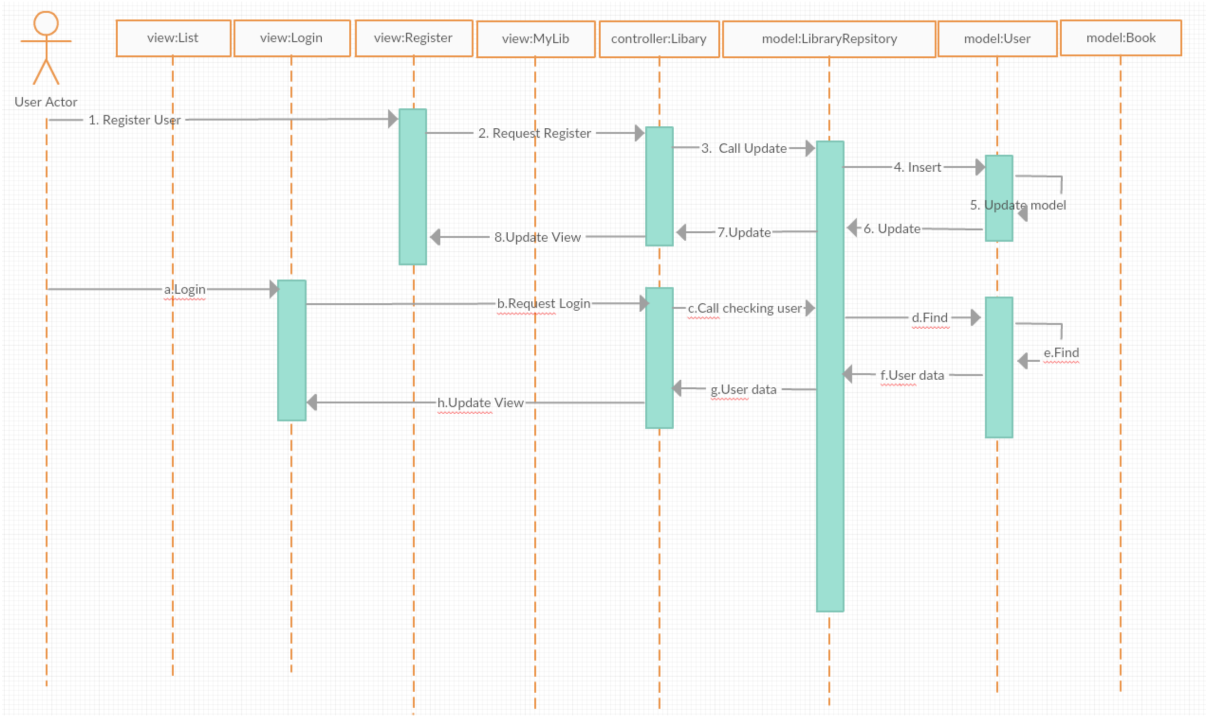
3. Diagrams

3.1. Class Diagram

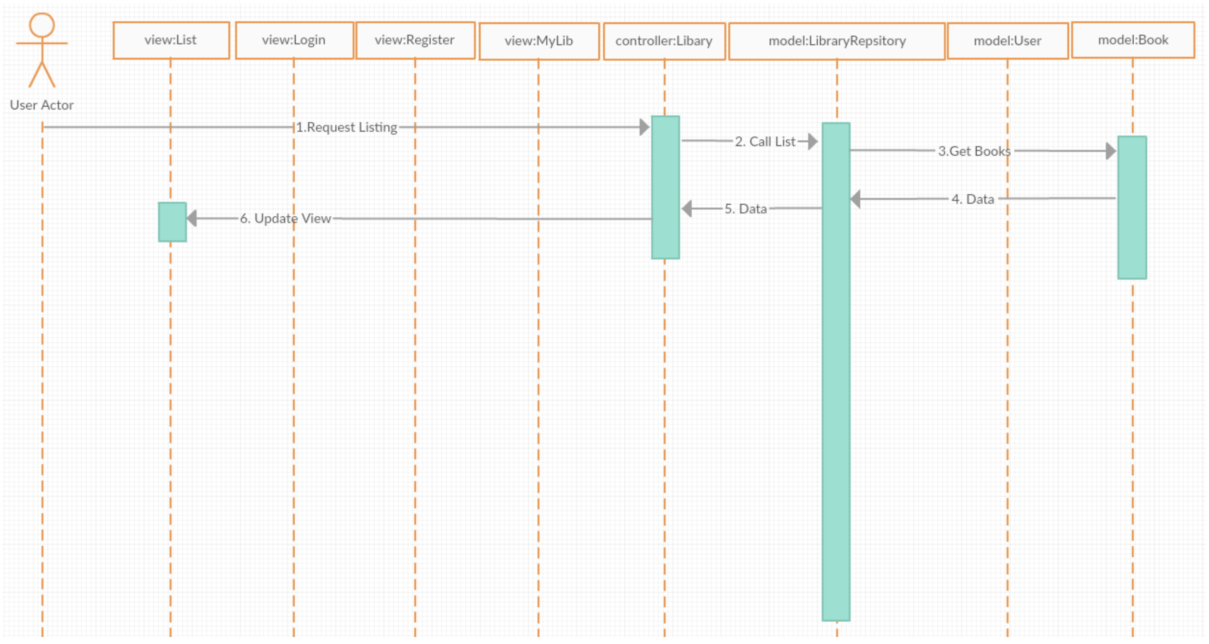


3.2. Sequence Diagram

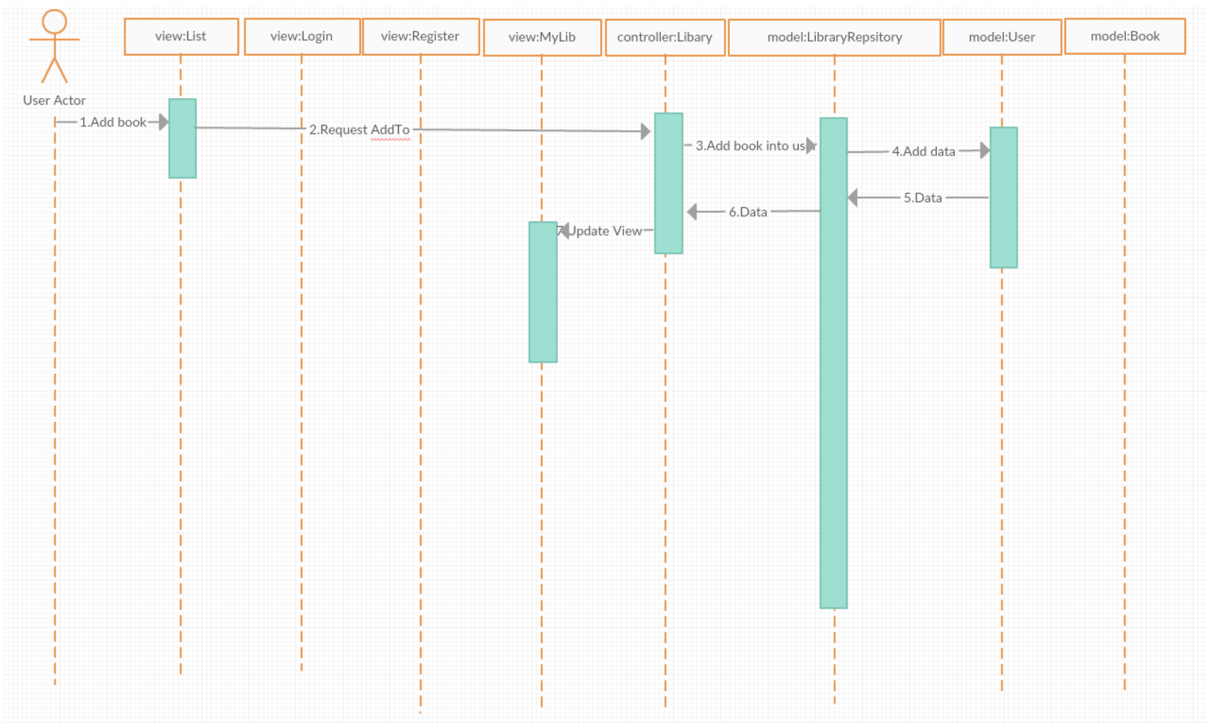
- Login/Register



- Listing Books



- Add /Remove Book into My Library


User Actor
view:List
view:Login
view:Register
view:MyLib
controller:Libary
model:LibraryRepsitory
model:User
model:Book
1.Add book
2.Request AddTo
3.Add book into user
4.Add data
5.Data
6.Data
7.Update View