BAHRIA UNIVERSITY (KARACHI CAMPUS)

ASSIGNMENT # 1 - SPRING 2023

Cloud Computing (SEN-325)

[CLO 2]

Class: **BSE-6 (A/B)** Max Marks: **5**

Course Instructor: **Engr. Muhammad Faisal**

[The marks of this assignment may increase or decrease]

Read Carefully:

- The deadline for this assignment is *before* or *on* **Thursday, 5th May, 2023.**

WARNINGS:

- This is an individual assignment; you must implement it by yourself. Any form of plagiarism will result in receiving zero in the assignment.
- Late submission will not be accepted. Any assignment submitted after the cutoff time will receive zero.

This assignment is divided into two parts

1) Compare the following architecture patterns
   a) MVC
   b) MVVM
   c) MVP

2) Write the entire process of creating Web API using
   a) MVVM
   b) MVP

MVC (Model View Controller):

MVC architectural pattern provide a clean and clear separation between database models, User interfaces & controller's business logics. Its extensively used with ASP.NET, Php & Ruby n Rails programing language. It's a triangular structure makes a difference from Client-Server architecture in which each component (Model, View and Controller) can directly interact with each other.

Model: Data & Storage

In Model component we perform database storage operation and apply all type of data constraints. Mapping our database ad constraint as a server objects.

View: User interface Presentation

View component basically provide the visualization of Data objects such that in a grid, lists , charts , graph bars etc.

Controller: Business logic

controller component is responsible for making communication between data objects and view object and control view events and model changes.

Advantages:

- Extensively used approach for building scalable websites & portal services.
- Provides clean coding techniques in which each layer is separated.
- Easily provide support and maintains to a huge application.
- Extension can be done without disturbing other running components
- Provide cohesion of modularity
- Can make your application as loosely coupled

Disadvantages:

- Not really useful for small or a simple application.
- Because of loosely couple may be Author of UI and Business logic is different team so may cause of delays

MVVM (Model View View Model):

MVVM architectural pattern provide bi-directional dataflow as a decoupling User interfaces and backend business logics. Its extensively used with angular, android programing language. It's does not allows/restrict the direct access of Database model layer it can only be call via observable.

Model: Data & Business logic

In Model component we perform database storage operation and apply all type of data constraints. Mapping our database ad constraint as a server objects.

View: User interface Presentation

View component basically provide the visualization of Data objects such that in a grid, lists , charts , graph bars etc.

ViewModel: Middleware Observable

ViewModel is an intermediate component between data objects and view object having responsibility provides observe response to view and observe model changes as callback.

Advantages:

- For event driven code accelerate there unit testing feature.
- Extensively used approach for building component based websites & mobile applications.
- Provides clean coding techniques in which each layer is separated.
- Easily provide support and maintains to a huge application.
- Provide cohesion of modularity

Disadvantages:

- Debug is little bit difficult because of it complex databinding.
- Not really useful for small or a simple application.
- For a huge complex logic its difficult to maintain and design ViewModel which cause delays.

MVP (Model View Presenter):

MVP architectural pattern is not triangular as MVC it has ability for two way dispatching of data also provides decoupling User interfaces and backend business logics. Its extensively used with android mobile applications. It's does not allows/restrict the direct access of Data model layer it can only be call via Presenter component.

Model: Entities & Services

In Model component all type of CRUD operations performed cab use pattern such as Repository etc apply all type of data constraints.

View: dumb User interaction
View component just a dumb with almost zero logic just input output controls which through user interaction to presenter view the response to user.
Presenter: Middleware View Interface
Presenter is an intermediate between Model and view object , view need subscription for changes notified, In Passive View ,view expose setter properties where all databinding done with state management call as View interface.
Advantages:

- Provide cohesion of modularity
- Extensively used for mobile applications UI component for native applications.
- Provides clean coding techniques in which each layer is separated.
- Easily provide support and maintains to a huge application.
- For event driven code accelerate there unit testing feature.

Disadvantages:

- Debug of Presenter is little bit difficult.
- For a huge complex logic its difficult to design Presenter which cause development delays.

	MVC	MVVM	MVP
Component	Service that send/receives response via routing	set of services with flowing data on each component	Two way data dispatching via route
Data Element	Parameter of object model passes return view response	object model passes return bind response on view	Parameter passes between layers
Connector	Procedure calling router call action by passing object	Method invocation, observable reducers	Procedure calling params/arguments passes
Topology	Triangular ,each component can call each other	Bi-directional, if model change view update via notify	Linear , Model pass to presenter->view interface->view (vice versa)
Constraint			
Quality yield	separation concern, clean & clear coding	Decoupling logic & UI layer	Loosely couple UI fully independent can change UI controls
Typical uses	scalable application with huge application with frequent extension	event driven	separate UI framework or native controls independent
Cautions		Complexity when response heavy cost increases	View model interaction prohibited
Relations to programming languages	asp.net with razor code	E6 JavaScript, Php , .net	android and iphone native application JAVA based

Creating a Web API using either MVVM (Model-View-ViewModel) or MVP (Model-View-Presenter) involves several steps, which are as follows:

a) Creating a Web API using MVVM:

- Define the Model: The Model represents the data and business logic of the application. Define the classes and methods that will interact with the database or other data sources.

- Define the ViewModel: The ViewModel acts as an intermediary between the Model and the View. It exposes data and commands to the View and communicates with the Model. Define the classes and methods that will handle data binding and expose data to the View.

- Define the View: The View displays the data to the user and handles user input. Define the HTML, CSS, and JavaScript files that will render the user interface.

- Implement the Web API: Create an ASP.NET Web API project using Visual Studio. Define the controllers that will handle HTTP requests and communicate with the ViewModel.

- Connect the ViewModel to the Web API: Use the HttpClient class to send HTTP requests to the Web API from the ViewModel. Deserialize the JSON response and update the ViewModel properties accordingly.

- Bind the View to the ViewModel: Use data binding techniques (such as AngularJS, KnockoutJS, or ReactJS) to connect the View to the ViewModel. The View should automatically update as the ViewModel properties change.

- Test and deploy the Web API: Test the Web API using a tool like Postman or Swagger. Once you're satisfied with the results, deploy the Web API to a hosting service like Azure or AWS.

b) Creating a Web API using MVP:

- Define the Model: Same as in MVVM, define the classes and methods that will interact with the database or other data sources.

- Define the View: Define the HTML, CSS, and JavaScript files that will render the user interface. Also, define the methods and events that will handle user input and communicate with the Presenter.

- Define the Presenter: The Presenter acts as an intermediary between the View and the Model. It handles user input and communicates with the Model. Define the classes and methods that will handle data binding and expose data to the View.

- Implement the Web API: Same as in MVVM, create an ASP.NET Web API project using Visual Studio. Define the controllers that will handle HTTP requests and communicate with the Presenter.

- Connect the Presenter to the Web API: Use the HttpClient class to send HTTP requests to the Web API from the Presenter. Deserialize the JSON response and update the Presenter properties accordingly.

- Update the View using the Presenter: The Presenter should update the View based on changes to the Model. Use JavaScript or other front-end technologies to update the HTML, CSS, and user interface elements accordingly.

- Test and deploy the Web API: Same as in MVVM, test and deploy the Web API to a hosting service like Azure or AWS.