**Design Notes**

This document serves to explain the architecture of ASP.NET MVC and Web API assignment.

**Views**

**Submission List Page**

1. Submission list page contains the list of all player who are already registered
2. A new player can also be added using the Add Player button
3. For Every Single player, there are options like adding an Assignee or deleting the player.

**Player Details Page**

1. This page contains the individual details of a player
2. Using this page, the user can edit the details of the player

**REST Service Call Flow**

1. For getting the list of all players, a get request will be sent to the API controller in the MVC with Web API Project. From there using the Http Client a request will be sent to the WebAPI Project. The WebAPI Project uses dapper to get data from the database. The Players List will be created as response and will be sent to the MVC with WebAPI project which then returns it to the view
2. For viewing the details of a single player, a get request will be sent to the API which contains the Player ID which then returns the appropriate response
3. For adding a new player, a post request will be sent to the API in which player object is passed as a parameter which then returns the status
4. For updating the details of the player, a Put request will be sent to the API with the player list which then returns the status message

**Folders**

1. Controller
2. Entites
3. Model
4. DAL
   1. Repository
   2. Persistence

**Database Details**

|  |  |
| --- | --- |
| **Server** | 10.194.50.77 |
| **Database Name** | TrainingAssignment |
| **Schema Name** | SatheshRangasamy |

**Application Tables**

Following are the tables which are used in the project

1. **Player Details Data Tables**
   1. AssigneeDetails
   2. SectionDetails
   3. PersonalDetails
   4. SubmissionMetaData
   5. Assignees
2. **Meta Tables**
   1. AuthorizedUsers
   2. Registering\_Club\_Input\_Meta
   3. Registration\_type\_input\_Meta
   4. RegistrationDetails
   5. Section\_Status\_Input\_Meta
   6. Status\_Input\_Meta
   7. Status\_List\_meta

**Entites**

1. **Player**
   1. id - integer
   2. firstName - string
   3. lastName - string
   4. preferredName - string
   5. gender - string
   6. dateOfBirth - string
   7. countryOfBirth - string
   8. townOfBirth - string
   9. nationalities - string
   10. nationalInsuranceNumber - string
   11. status - string
   12. registrationType - string
   13. registrationID - string
   14. registeringClub - string
   15. player - string
   16. registration - string
   17. transfer - string
   18. intermediaries - string
   19. ITC - string
   20. GBE - string
   21. registrationDateTime - string
   22. submittedDateTime - string
   23. submittedBy - string
   24. lastUpdatedDateTime - string
   25. updatedBy - string
2. **MetaData**
   1. Assignee - List<Assignees>
   2. status - List<string>
   3. registration\_type - List<string>
   4. registering\_club - List<string>
   5. section\_status - List<string>
   6. section\_status\_name - List<string>
3. **Assignee**
   1. assignee - string
   2. assigneeShort- string
   3. Color - string

**Project - MVC With WebAPI**

**View**

1. Front End Designed using Angular
2. The angular project build serves as a view in the MVC project

**API Controller - Methods**

1. **public async Task<HttpResponseMessage> GetMetaData()**
   1. Returns HttpResponseMessage which contains the metadata needed for the application
   2. Sends an asynchronous request to WebAPI and returns the response
2. **public async Task<HttpResponseMessage> GetPlayers()**
   1. Returns HttpResponseMessage which contains the list of all players
   2. Sends an asynchronous request to WebAPI and returns the response
3. **public async Task<HttpResponseMessage> GetPlayer(int id)**
   1. Gets the Player Id as input and returns the HttpResponseMessage which contains the list of the requested player
   2. Sends an asynchronous request to WebAPI and returns the response
4. **public async Task<HttpResponseMessage> updatePlayer([FromBody]Player player, int id)**
   1. Gets Player Object and Player ID and sends a asynchronous request to WebAPI and returns the HttpResponse
5. **public async Task<HttpResponseMessage> delete(int id)**
   1. Gets Player Id and sends a asynchronous request to WebAPI and returns the HttpResponse
6. **public async Task<HttpResponseMessage> addPlayer([FromBody]Player player)**
   1. Gets Player Object and sends a asynchronous request to WebAPI and returns the HttpResponse
7. **public bool GetUserDetails(String UserName)**
   1. Gets Username sends a asynchronous request to WebAPI and returns the HttpResponse
   2. Used to Check whether a user is available or not

**Project - WebAPI**

**Controllers Methods**

1. **public async Task<HttpResponseMessage> GetMetaData()**
   1. Constructs a HttpResponseMessage using the Metadata object from the Player Model GetMetaData() method.
   2. Returns HttpResponseMessage which contains the metadata needed for the application
2. **public async Task<HttpResponseMessage> GetPlayers()**
   1. Gets the player list from the Player Model GetPlayers() Methods
   2. Constructs and returns HttpResponseMessage which contains the list of all players
3. **public async Task<HttpResponseMessage> GetPlayer(int id)**
   1. Gets the Player Id as input and sends the player id to the Player Model GetPlayer() methods and gets the player list
   2. Returns the HttpResponseMessage which contains the list of the requested player
4. **public async Task<HttpResponseMessage> updatePlayer([FromBody]Player player, int id)**
   1. Gets Player Object and Player ID and sends to player model updatePlayer() method and returns the HttpResponse
5. **public async Task<HttpResponseMessage> addPlayer([FromBody]Player player)**
   1. Gets Player Object and sends to player model updatePlayer() method and returns the HttpResponse
6. **public HttpResponseMessage deletePlayer(int id)**
   * 1. Gets Player ID and sends to player model deletePlayer() method and returns the HttpResponse
7. **public bool GetUserDetails(String UserName)**
   1. Gets Username and sends to player model updatePlayer() method and returns the HttpResponse

**Player Model**

1. **public List<Player> GetPlayers(int? id)**
   1. Calls the Method in the player Repository
   2. Method also works for getting all users and get individual user
2. **public int updatePlayers(Player playerVal,int id)**
   1. Calls the Method in the player Repository
   2. Returns Status code as response to the controller
3. **public int deletePlayer(int id)**
   1. Calls the Method in the player Repository
   2. Returns Status code as response to the controller
4. **public MetaData getMetaData()**
   1. Return MetaData Object as a Response to the controller
5. **public int addPlayer(Player playerVal)**
   1. Calls the Method in the player Repository
   2. Returns Status code as response to the controller
6. **public int authorizeUser(string name)**
   1. Calls the Method in the player Repository
   2. Returns Status code as response to the controller

**DAL - Repository**

It contains the declarations for the following methods

1. public List<Player> GetPlayers(int? id)
2. public int updatePlayers(Player playerVal,int id)
3. public int deletePlayer(int id)
4. public MetaData getMetaData()
5. public int addPlayer(Player playerVal)
6. public int authorizeUser(string name)

**DAL - Persistence**

It contains the definitions for the following methods

1. **public List<Player> GetPlayers(int? id)**
   1. This method works for both getting the details of all players and for a specific player
   2. This method uses dapper and calls the **USP\_getUserList** procedure and returns the values
   3. If the player id is passed to the stored procedure it returns the list of the specific player with that ID else it will return the whole list.
2. **public int updatePlayers(Player playerVal,int id)**
   1. This method is for updating the details of the player
   2. Gets the Player Object as input
   3. Constructs the parameters needed for the procedure
   4. Using dapper, the procedure **USP\_updatePlayer** is called
   5. Returns the status.
3. **public int deletePlayer(int id)**
   1. This method is for deleting an individual user.
   2. Gets the player ID as a input argument
   3. Using dapper, the procedure **USP\_deletePlayer** is called with the player ID as a argument.
   4. Returns the status
4. **public MetaData getMetaData()**
   1. This method is to get all the metadata needed for the project.
   2. Using Dapper, the Procedure **USP\_GetMetaData** is called
   3. The Response from the procedure are constructed as a metadata object and is sent back as a response
5. **public int addPlayer(Player playerVal)**
   1. This method is to add a new Player
   2. Gets the Player Object as input
   3. Constructs the parameters needed for the procedure
   4. Using dapper, the procedure **USP\_addPlayer** is called
   5. Returns the status.
6. **public int authorizeUser(string name)**
   1. This method is to check whether the user is an authorized user or not
   2. Gets the player name as input
   3. Using dapper, the procedure **USP\_authorizeUser** is called
   4. Returns the status