Distributed Systems 600089

Student ID: 201601620

Date: **August 3, 2020**

Deadline: Monday 3rd August 2020 by 2pm

What does an API do?

APIs (application program interface) are routines, protocols and tools put together for building software applications. An API is a guide to software components on how to interact and helps to bridge interaction between different software.

Brief Explanations

Route Mapping

Route mapping is when the framework tries to match the segments in the URI path to the template. Literals in the template must match exactly. A placeholder matches any value unless constraints are set. The framework does not match other parts of the URI, such as the host name or the query parameters. The framework selects the first route in the route table that matches the URI. The URL contains special placeholders"{controller}" and "{action}". The controller is the name of the controller being accessed and action is the name of the action being accessed.

Route mapping is where a URL path is passed into the system and it will try to follow the path along it’s route through the framework. The URL route must be exact as even a small spelling mistake can cause great issue. The URL will contain special placeholders, being the controller and the action and on occasion, if data is to be passed through, parameters may also follow suit.

How WebAPI uses the Id parameter

The webapi gets the value of the id parameter from the URI. To get the value the webapi goes through the route data as well as the query string.

What are Actions?

Actions are the same as functions except that an actions doesn’t return a value. Such as an Action can be used with a method that has a void return type.

API Key use in the Project

The API key was used in multiple instances, to generalise I have listed below a GET use, POST use and a DELETE use.

```
97      static async Task GetUser(string pUser)
98      {
99          var response = client.GetAsync("api/user/new?username=" + pUser);
100         Console.WriteLine(loading);
101         var res = await response.Result.Content.ReadAsStringAsync();
102         Console.WriteLine(res);
103     }
104
105     static async Task PostUser(string pUser)
106     {
107         var json = JsonConvert.SerializeObject(pUser);
108         var data = new StringContent(json, Encoding.UTF8, "application/json");
109         var response = client.PostAsync("api/user/new", data);
110         Console.WriteLine(loading);
111         var res = await response.Result.Content.ReadAsStringAsync();
112         if (response.Result.IsSuccessStatusCode)
113         {
114             mKey = res;
115             client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
116             client.DefaultRequestHeaders.Add("ApiKey", mKey);
117         }
118         else
119         {
120             Console.WriteLine(res);
121         }
122     }
123
124     static async Task Create(string pUser)
125     {
126         mUser = pUser;
127         guid = Guid.NewGuid();
128         mKey = guid.ToString();
129         var values = new Dictionary<string, string>
130         {
131             {mUser,mKey}
132         };
133     }
134
135     static async Task Delete(string pUser)
136     {
137         var response = client.DeleteAsync("api/user/removeuser?username=" + pUser);
138         Console.WriteLine(loading);
139         var res = await response.Result.Content.ReadAsStringAsync();
140         Console.WriteLine(res);
```

```
143     static async Task Update(string pUser, string pRole)
144     {
145         var values = new Dictionary<string, string>
146         {
147             { "username", pUser},
148             { "role", pRole }
149         };
150         var json = JsonConvert.SerializeObject(values);
151         var data = new StringContent(json, Encoding.UTF8, "application/json");
152         var response = client.PostAsync("api/user/changerole", data);
153         Console.WriteLine(loading);
154         var res = await response.Result.Content.ReadAsStringAsync();
155         Console.WriteLine(res);
156     }
157     #endregion
```

RSA Algorithm Steps

The RSA algorithm is used to encrypt and decrypt messages using a private and public key. The private key is used for the server to decrypt the message and the public key is used to

encrypt the message. As stated by their names the key only has 1 private key (hence private) and the public key can be given to anyone who wishes to send a message (hence public).

AES Algorithm Steps

The AES algorithm is used to encrypt and decrypt messages, however, the keys can be either 128 bits in length or 192 bits. A cypher is used to encrypt and decrypt the message using the key provided however both the sender and recipient must have the same key for it to work.

What is the Entity Framework?

Microsoft created an open-source framework for .NET applications know as the entity framework. This framework allows us to work directly with classes and avoid interacting physically/manually with the underlying database.

Using the NuGet Packet Manager in Visual Studio, you can issue some commands in relation to the entity framework. An example of this would be to issue the command to migrate and a migration would then be created using a snapshot.

Code/Database/Model First?

Code first allows for a version controlled database and its contents to be created from business objects. This method is rather good when it comes to eager loading or no serializing at all, however, this method would only really suit smaller applications and not something to support a huge system.

Unfortunately code first must have every little bit of code written down which can lead to some human error or inefficiency. This includes writing the stored procedures as well. The database itself has issues with code first as to update a table you would have to use the packet manager with the command “update-database”.

Database first allows for simplicity when it comes to creating the data model. The database construction is a lot easier requiring little to no code at all to make simple things like relationships. This method is very good for when handling large or data intensive applications, such as a large warehouse stock system.

The downside to using the database first method is you need to use an existing database to build an edmx file which will then generate some code for you, although, for the model to have extra functionality it must be extended to a new class.

The model first method is perfect for a small project as it is simple and easy. The models created will then go on to create the workflow and database template automatically.

The issue with model first though is there is less control over the database and data loss can occur during the defining of the database.

Reflection

Struggling due to a few issues with understanding at the start of the module and other personal things I had a hard time figuring out how to tackle this. Eventually after working through half of the workshops I managed to be able to apply some of what I had done in the workshops to past experiences with programming to work out, albeit in a messy way, how to do most of the tasks. I'm unsure on the ability of the program as it seemed to work some days and other days it would spit issues at me.

It took me a while to figure out how to do the migration using the packet manager and realised there was an issue with the entity framework (this is what the error issued me) and so I had to solve this but it ended up being rather simple.

There were a few other inconsistent issues with things working one day and not working the next even though nothing had been changed. Unfortunately I ran out of time to add in the logging side of things in the database which, in hindsight, should have been done in the task order given. Tackling this in the wrong task order was a bad mistake on my part.