**PROJECT REPORT**

RESTFUL WEB API

Himanshu Singh

B00657143

**ABSTRACT**

This project is a simple web application. Web applications are hosted on web servers which are a container for web applications. This web application conforms to the RESTFUL architecture. RESTFUL refers to representational transfer state, it is an architectural style. The front end of client side of this project is a simple HTML page which interacts with the web API using JavaScript and jQuery. The back end of API is built on the .NET framework using .NET MVC, Entity framework, C# programming language data used is from local MySQL database.

**INTRODUCTION**

This section introduces concepts of several technologies used for the project.

*Web Servers*:

Containers for web applications.

They store resource which can be static/dynamic web pages: website, data: data web service/data service, computing power: set of functions/ client libraries: Web service/Web API.

They receive HTTP request from clients forwards the info included in request to web applications sitting in container and sends response back.

Examples of web servers; Internet Information Server IIS, Apache Tomcat, Node.js, Nginx, Jetty.

*Web Applications*:

Websites, web services (Web API), web data service are all web applications.

Performs jobs of serialization, deserialization and computing actions.

Examples of web applications:

REST web API using C#, ASP.NET, MVC host website on IIS.

Web services using C# and WCF then host in IIS.

Build website, web service, REST web API using spring MVC in Java then host in Tomcat.

Build website and REST web API using PHP and host them in Tomcat or Nginx.

*Remark*:

All web apps don’t need web services. Web applications using Node or ASP.NET core can be self-hosted do not need web server to accept HTTP request and send back response can be done locally. In this project the webApi is hosted on locally.

***REST Architecture****:*

It is an architectural style for distributed systems (single client/server app or multiple apps connected via a network).

Set of constraints that apply to architecture level design and API level design.

If a distributed system applies these constraints, then it is called RESTFUL distributed system.

*Important terms*:

*Resource*: any info on web server that can be named; data, computing power.

*Resource Identifier*: unique name of resource. Uniform Resource Identifier URI implementation of resource identifier in HTTP protocol. URL uniform resource locator web address used to locate reference target resources on web server.

*Representation*: sequence of bytes contains serialized resource and info about serialized resource (representation metadata). RESTFUL components exchange info and perform actions by transferring representations.

***REST architecture constraints***:

1. Client-Server system: must be applied to client server system.
2. Statelessness: server does not cache any client or session related data. Client can cache data for future use. RESTful request never need session data on the server and never need web server to support session data.
3. State in REST is not the session state it is the state of resources carried by representation. The Jason objects being transferred have state information about the resource e.g. it was modified by client. So, it is a transfer of representation which carries serialized resource with its resource state.

***REST API level design constraints***:

1. Identification of resource: every resource has unique URL Manipulation of resources through representations: all data/info needed by servers to action fully included in representation by request.
2. Hypermedia as engine of resource: URLs are embeded that reference the related resource.
3. Layered system: to improve scalability organize components into different functional layers.
4. Code on demand: allows client functionality to extended by downloading and executing code in the form of applets or scripts. Client can fetch these and execute to run predefined logic.

***URLS for RESTful API***:

1. Its supports the uniform interface feature i.e. central to this architecture.
2. Its is a hierarchical relation separated by /.
3. All words lower case and use hyphens.
4. No file extensions.
5. Use plural names for collections.
6. Query params used for filtering, use verb verb phrase for controller names.
7. Create Read Update Delete (CRUD) should not be used in urls.

***HTTP request***:

1. It is a message from client to server follows HTTP protocol for message transfer over the web.
2. Request methods indicates what operation client wants server to take on resource. Actions are namely GET, POST, PUT, Delete, Patch.

***HTTP response:***

1. Serve responds to http request via http response one response per request.
2. Includes status message.

***ASP.NET core web API:***

***MVC:***

1. ***Model abstraction of entities***:

1.1 Domain Model: abstraction of data used by business logic.

1.2 View Model: data rendered by view.

1.3 Data transfer model: data being sent back and forth b/w client and server.

Domain model can play role of view model or data transfer model.

View: not part of RESTful web API it is for client-side application.

1. ***Controller***:

2.1 Holds logic of application driven by user interaction, makes app functional.

2.2Member methods of controller classes are actions which respond and process operations.

***URL routing***:

web clients invoke Web API remotely by sending HTTP request points out which action of which controller of server will be called.

***Routing templates***:

strings representing URL pattern. Controllers have base routes then their actions have routes built on this base rout.

There is explicit mapping between controller names, controller action names, routing templates used for each controller and action name and URLs used clients. This attribute routing is taken care of by Microsoft.AspNetCore.Mvc framework. It makes sure that all request to a specific URL are directed to the corresponding controller and action.

**Implementation Details**

***Models***:

Used Entity Framework which is an Object Relational Mapping framework that maps data from local SQL database to model classes in project.

1. Model classes created:

1.1Cities.cs: class representing cities with attributes same as data store in SQL DB for cities.

1.2Country.cs: class representing countries.

1. Utility class created:

2.1DataAccess.cs:

WorldDbContextFactory: uses factory pattern to provide database connection objects of type dbContext when required. dbContext class is integral part of EntityFramework.

WorldDbContext inherits from DbContext from entity framework. The Dbcontext class manages the entity objects during run time, which includes populating objects with data from a database, change tracking, and persisting data to the database. DbSet<Country>, DbSet<City> are properties of WorldDbContext class that hold data from the sql database which gets mapped to array of Country.cs and City.cs object types.

3. Controller classes created:

3.1 CitiesController.cs: works on the DBSet from Cities.cs Model class. Supports all the HTTP request operations Get, Post, Delete, Put.

3.2 CountriesController.cs: works on DBSet from Countries.cs Model class. Supports all the HTTP request operations Get, Post, Delete, Put.

Returned data is an array of Model objects that gets converted to json string and is passed in the HTTP response packet along with the Status Code.

***Client.html***:

Basic html jQuery client that supports the following operations:

1. Get all cities.
2. Get specific city by city ID
3. Create a new city
4. Update an existing city
5. Delete an existing city.

***Running Application:***

1. Open console go to project directory.
2. Execute command dotnet restore.
3. Execute command dotnet build.
4. Execute command dotnet run.
5. Web page opens. Add /client.html to the URL to access the client code.

***Future Modifications:***

Making the client code richer in abilities by adding map which can show location of the selected city.

***References:***

1. RESTful API knowledge taken from edx.org course.
2. Client.html taken from: <https://courses.edx.org/courses/course-v1:Microsoft+DEV247x+1T2018/course/>
3. MSDN docs referred for all .NET framework technologies used.