**Online To Do List**

Created a web API, that helps user to maintain a “To Do List”. The user should be able Add, Edit and Delete the contents of this list using this API. Started working on the front-end web application that consumes the API but couldn’t complete this in the allocated time. The code that was done so far for the web app has been included as well in repository for reference.

Design considerations for this development along with the ideas for further development can be found under the section “Design and areas for further development” at the end of this document.

**Components:**

Web API - Completed

Front-end Web Application – Incomplete and in progress

**Tech Stack:**

Web API : .Net Core Web API, EF Core, In Memory Database

Front-end Web Application: .Net Core MVC with JQWidgets ( a modern JQuery based library)

**Git Hub Details**

URL: <https://github.com/vnykprabha/ToDoList>

Web API: OnlineToDoListAPI ( Completed)

Web App: OnlineToDoList

**Web API Functionality**

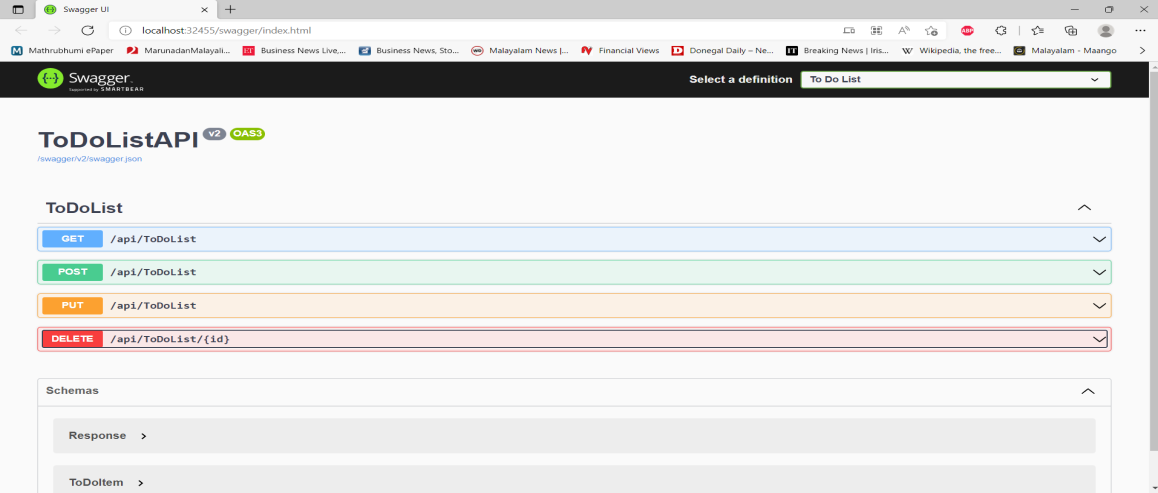
All the below requested minimum functionality has been completed.

* User should be able to view her/his task list.
* User should be able to add/remove task.
* All changes are persisted within an application run.
* Each task should display the last updated date along with description.
* Unit Tests.
* Web API functionality is demonstrable through Swagger UI and Postman.

**Instructions to run the API**

* Open the **OnlineToDoListAPI** Solution in Visual Studio 2019 or later.
* Run the application via visual Studio IIS Express.
* The below URL should list out all the API methods in Swagger. Local host port number in the URL will be different for each user who runs this app.

<http://localhost:32455/swagger/index.html>



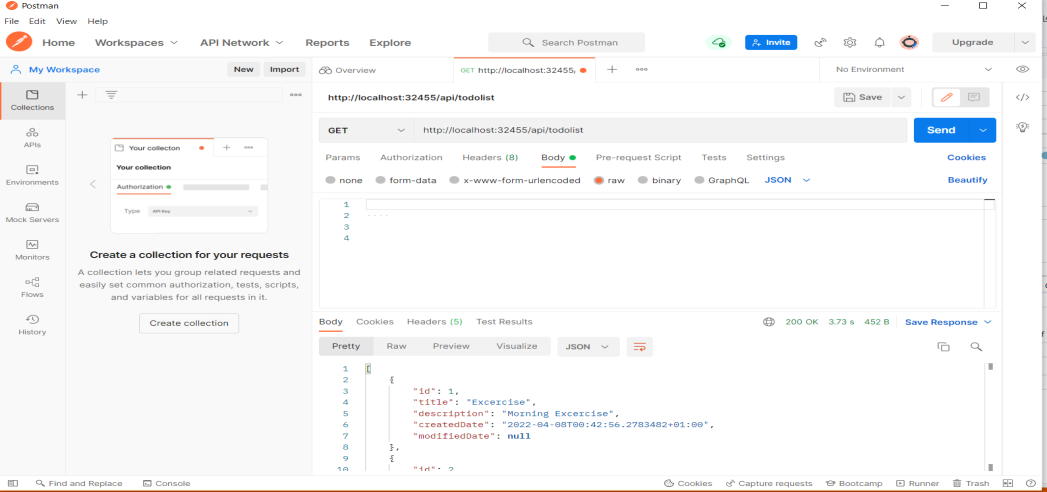
* Use the below URL in Postman to view the results. Local host port number in the URL will be different for each user who runs this app.

<http://localhost:32455/api/todolist/>

* **Web API Methods**

1. Get – Gets the To Do List.

Choose “GET” option along with the url: <http://localhost:32455/api/todolist/> in Postman and click “Send”. User should see the existing To Do List. Some seed data is added to the list on application load and hence user should see two items in the “GET” request when they run the application for the first time.



1. Post - Adds a new item to the to do list.

Choose “POST” option along with the url: <http://localhost:32455/api/todolist/> in Postman. Under “Body” tab, add the below model, choose type as JSON and click “Send”. We don’t need to add values against CreatedDate field as system will automatically sets value for this property via back-end.

{

    "Id": 1000,

    "Title": "Drop Children",

    "Description": "Drop children to school",

    "CreatedDate": **null**,

    "ModifiedDate": **null**

}

On Success, API Should return the below model

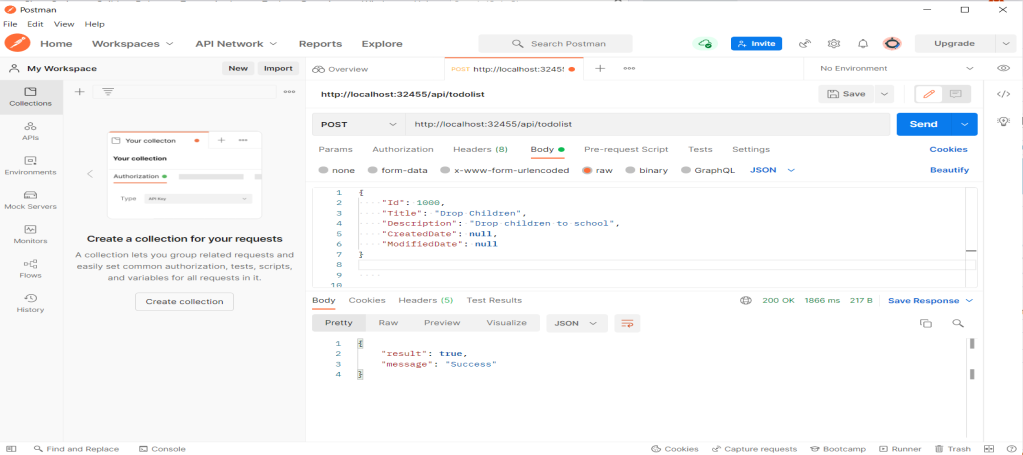
{

    "result": **true**,

    "message": "Success"

}

Use the “Get” Method to confirm that the new item is added to the list.



1. Put - Updates an existing item in the to do list

Choose “PUT” option along with the url: <http://localhost:32455/api/todolist/> in Postman. Under “Body” tab, add the below model, choose type as JSON and click “Send”. We don’t need to add values against “CreatedDate” field as system will automatically sets value for this property via back-end. We are supposed to edit an existing item only and hence the “ID” property here needs to be an existing “ID”. I’ve modified the “Title” and “Description” here for this operation.

{

    "Id": 1000,

     "Title": "Park",

    "Description": "Drop children to park",

    "CreatedDate": **null**,

    "ModifiedDate": **null**

}

On Success, API Should return the below model

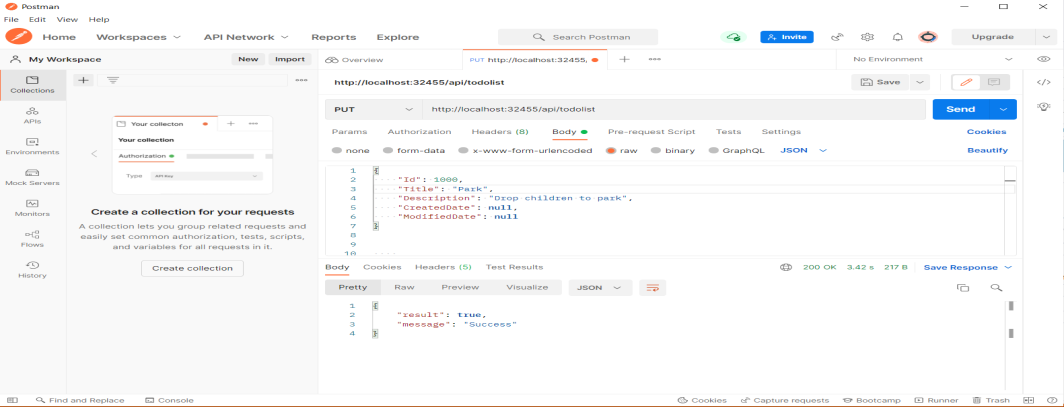
{

    "result": **true**,

    "message": "Success"

}

Use the “Get” Method to confirm that the new item is added to the list. Modified Date will be updated automatically for this to do item.



1. Delete – Deletes an existing in the to do list

Choose “DELETE” option along with the url: <http://localhost:32455/api/todolist/1000> in Postman. Here 1000 in the URL refers to the ID of the item that needs to be deleted from the list. Hence we’re supposed to use only and existing ID for this operation.

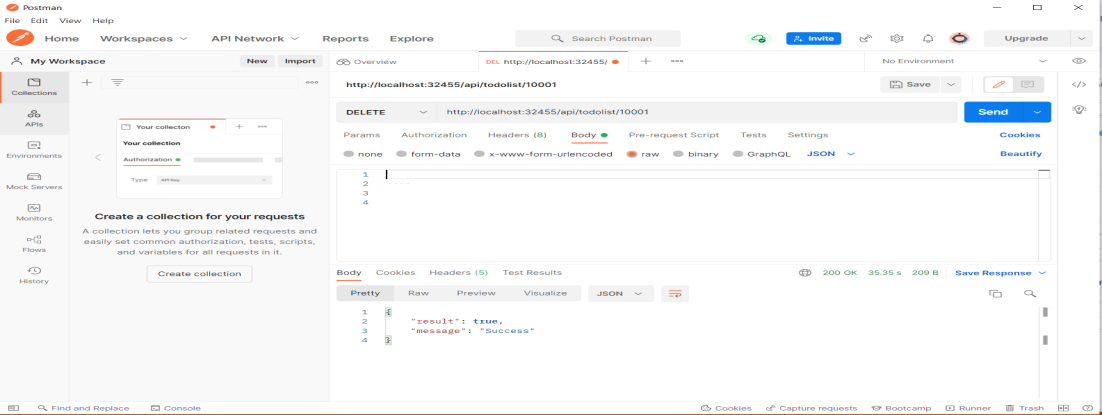
On Success, API Should return the below model

{

    "result": **true**,

    "message": "Success"

}



**Web Application Functionality**

Web application is supposed to consume the API and display the data in a grid. User should be able Add, Edit and Delete a To do item via the user interface. The application is being developed using .Net Core MVC with JQWidgets a modern Jquery based front-end framework. Due to the time constraints for this coding assignment, development for this application couldn’t be completed and is in progress. The code that was done so far has been added to the repository.**Design and areas for further development**

The application uses N-tier architecture with dependency injection design pattern. I’ve used constructor injection for injecting the dependencies where ever needed. The application follows single responsibility principle with each layer handling only its own duty. The application uses Swagger for documentation. Data is persistent and stored in memory using the “UseInMemoryDatabase” option of the DBContext and the configuration for this is defined in Startup.cs class. I’ve used entity framework core as ORM for communicating with the database. A generic repository layer has been designed to handle all CURD operations but due to some reasons it is not working against the in- memory database. Due to the time constraints associated with this assignment, I couldn’t research more on it and had to do move some the repository operations to the business layer. In the mean time, I’ve retained the generic repository class that I’ve created for the reference which should’ve normally worked against a traditional database.

I had started working on the front-end web app that consumes the API but due to the time constraints, couldn’t complete this in the allocated time. The code that was done so far for the web app has been included as well in repository for reference.

If I was given more time, I should have done the following

* Complete Web app development and establish its communication with API
* Middleware based Authorization/Authentication.
* Enhanced error handling via middleware.
* Logging
* Code re-factoring