### ****1. Demonstrate creation of an Action method to perform data create, update & delete operation****

In this project, the goal was to implement action methods in an ASP.NET Core Web API to perform update and delete operations on employee data. The employee data was stored in a hardcoded in-memory list, simulating a basic backend without a database. To perform the **update operation**, an action method decorated with [HttpPut("{id}")] was used. The method receives the employee's ID from the URL and the updated employee object from the request body using [FromBody]. It validates the ID, and if the employee exists in the list, it updates the relevant properties and returns the updated employee with a 200 OK response. If the ID is invalid or not found, it returns a 400 Bad Request with an appropriate message. Additionally, a **delete operation** was implemented using an [HttpDelete("{id}")] action. This method receives the employee ID from the URL, checks if it’s valid and exists in the list, and then removes the corresponding employee. Upon successful deletion, a confirmation message is returned with a 200 OK response. If the ID is invalid or does not exist in the list, a 400 Bad Request is returned. These methods demonstrate the complete handling of **update and delete** operations in a stateless Web API environment, and both were thoroughly tested via Swagger and Postman.

### ****Use FromBody attribute and extract data to custom model class****

### The [FromBody] attribute plays a key role in the update action by allowing the API to bind the JSON request body to a complex Employee object. The Employee model includes scalar properties like Id, Name, Salary, and Permanent, along with complex nested objects such as Department and a list of Skill objects. When a client sends a PUT request with the updated employee details in JSON format, ASP.NET Core automatically deserializes the data into the Employee object via [FromBody]. This promotes clean code and allows direct access to structured input, simplifying the update logic and maintaining strong typing.

### ****Use hardcoded data to update & delete data, Use Swagger and POSTMAN to test****

Instead of using a database, this project used a hardcoded in-memory list to simulate a basic data layer. This allowed fast prototyping of business logic without additional setup. The list was pre-populated with two sample employees, and operations like **update (PUT)** and **delete (DELETE)** modified this list at runtime. The API was tested using both Swagger and Postman. Swagger provided an easy interface to send requests with sample JSON bodies, while Postman helped simulate real-world API calls with custom headers and body formatting. Testing validated that the application correctly handled both success and failure scenarios — such as updating an existing employee, attempting to update or delete with invalid IDs, and receiving appropriate status codes (200 OK, 400 Bad Request) along with clear messages. These tests ensured that the Web API behaves reliably and follows RESTful standards.