Modify and Validate Data Lab

Perform these labs on your own computer using Visual Studio 2022 to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1: Add Modification Methods to IRepository Interface

Open the **IRepository.cs** file and add a few more template methods as shown in **bold** below.

```
public interface IRepository<T>
{
  List<T> Get();
  T? Get(int id);

  T Insert(T entity);
  T Update(T entity);
  T SetValues(T current, T changes);
  bool Delete(int id);
}
```

Modify Customer Repository Class

Just so we can keep compiling, open the **CustomerRepository.cs** file and add some new methods to implement the interface methods.

```
#region Insert Method
public Customer Insert(Customer entity)
{
  throw new NotImplementedException();
}
#endregion

#region Update Method
public Customer Update(Customer entity)
{
  throw new NotImplementedException();
}
#endregion

#region SetValues Method
public Customer SetValues(Customer current, Customer
changes)
{
  // Since we don't necessarily pass in all the data,
  // overwrite the changed properties in the one
  // read from the database
  // TODO: Make this a little more bullet-proof
  current.NameStyle = changes.NameStyle;
  current.Title =
string.IsNullOrWhiteSpace(changes.Title) ? current.Title
: changes.Title;
  current.FirstName =
string.IsNullOrWhiteSpace(changes.FirstName) ?
current.FirstName : changes.FirstName;
  current.MiddleName =
string.IsNullOrWhiteSpace(changes.MiddleName) ?
current.MiddleName : changes.MiddleName;
  current.LastName =
string.IsNullOrWhiteSpace(changes.LastName) ?
current.LastName : changes.LastName;
  current.Suffix =
string.IsNullOrWhiteSpace(changes.Suffix) ?
current.Suffix : changes.Suffix;
  current.CompanyName =
string.IsNullOrWhiteSpace(changes.CompanyName) ?
current.CompanyName : changes.CompanyName;
  current.SalesPerson =
string.IsNullOrWhiteSpace(changes.SalesPerson) ?
current.SalesPerson : changes.SalesPerson;
  current.EmailAddress =
string.IsNullOrWhiteSpace(changes.EmailAddress) ?
current.EmailAddress : changes.EmailAddress;
```

```
  current.Phone =
string.IsNullOrWhiteSpace(changes.Phone) ? current.Phone
: changes.Phone;
  current.PasswordHash =
string.IsNullOrWhiteSpace(changes.PasswordHash) ?
current.PasswordHash : changes.PasswordHash;
  current.PasswordSalt =
string.IsNullOrWhiteSpace(changes.PasswordSalt) ?
current.PasswordSalt : changes.PasswordSalt;
  current.Rowguid = changes.Rowguid == Guid.Empty ?
current.Rowguid : Guid.NewGuid();
  current.ModifiedDate = DateTime.Now;

  return current;
}
#endregion

#region Delete Method
public bool Delete(int id)
{
  throw new NotImplementedException();
}
#endregion
```

Try it Out

Build the solution to ensure everything still compiles.

Lab 2: Finish Insert() Method

Open the **CustomerRepository.cs** file and modify the Insert() method.

```
public Customer Insert(Customer entity)
{
  // Fill in required fields not passed by client
  entity.Rowguid = Guid.NewGuid();
  entity.ModifiedDate = DateTime.Now;

  // Add new entity to Customers DbSet
  _DbContext.Customers.Add(entity);

  // Save changes in database
  _DbContext.SaveChanges();

  return entity;
}
```

Open the **CustomerController.cs** file and **replace** the Post() method you added earlier with the code shown below.

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServe
rError)]
public ActionResult<Customer> Post([FromBody] Customer
entity)
{
  ActionResult<Customer> ret;

  // Serialize entity
  SerializeEntity<Customer>(entity);

  try {
    if (entity != null) {
      // Attempt to update the database
      entity = _Repo.Insert(entity);

      // Return a '201 Created' with the new entity
      ret = StatusCode(StatusCodes.Status201Created,
entity);
    }
    else {
      InfoMessage = "Customer object passed to POST
method is empty.";
      // Return a '400 Bad Request'
      ret = StatusCode(StatusCodes.Status400BadRequest,
InfoMessage);
      // Log an informational message
      _Logger.LogInformation("{InfoMessage}",
InfoMessage);
    }
  }
  catch (Exception ex) {
    InfoMessage =
_Settings.InfoMessageDefault.Replace("{Verb}",
"POST").Replace("{ClassName}", "Customer");

    // Return a '500 Internal Server Error'
    ErrorLogMessage = $"CustomerController.Post() -
Exception trying to insert a new customer:
{EntityAsJson}";
    ret = HandleException<Customer>(ex);
  }

  return ret;
```

```
}
```

The **[FromBody]** attribute in the above code is optional. It is the default. I just wanted to show you that you will sometime see this.

Try it Out

Run the application and click on the **POST /api/Customer** button.

In the **Request body** add the following:

```
{
  "NameStyle": true,
  "Title": "Mrs.",
  "FirstName": "Amy",
  "MiddleName": "B",
  "LastName": "Smythe",
  "Suffix": "",
  "CompanyName": "Smythe Motors",
  "SalesPerson": "Gene",
  "EmailAddress": "Amy.Smythe@smythemotors.com",
  "Phone": "(977) 333-9938",
  "PasswordHash": "123bbdeic3332",
  "PasswordSalt": "235asdf"
}
```

Look at the resulting JSON passed back

Save the new **CustomerID** value that was generated as you will need it for updating and deleting the product.

Lab 3: Finish Update() Method

Open the **CustomerRepository.cs** file and modify the Update() method with the code shown in **bold** below.

```
public Customer Update(Customer entity)
{
  // Update last date updated
  entity.ModifiedDate = DateTime.Now;

  // Update entity in Customers DbSet
  _DbContext.Customers.Update(entity);

  // Save changes in database
  _DbContext.SaveChanges();

  return entity;
}
```

Open the **CustomerController.cs** file and add a Put() method.

```
[HttpPut("{id}")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServe
rError)]
public ActionResult<Customer> Put(int id, [FromBody]
Customer entity)
{
  ActionResult<Customer> ret;

  // Serialize entity
  SerializeEntity<Customer>(entity);

  try {
    if (entity != null) {
      // Attempt to locate the data to update
      Customer? current = _Repo.Get(id);

      if (current != null) {
        // Combine changes into current record
        entity = _Repo.SetValues(current, entity);

        // Attempt to update the database
        current = _Repo.Update(current);

        // Pass back a '200 Ok'
        ret = StatusCode(StatusCodes.Status200OK,
current);
      }
      else {
        InfoMessage = $"Can't find Customer Id '{id}' to
update.";
        // Did not find data, return '404 Not Found'
        ret = StatusCode(StatusCodes.Status404NotFound,
InfoMessage);
        // Log an informational message
        _Logger.LogInformation("{InfoMessage}",
InfoMessage);
      }
    }
    else {
      InfoMessage = "Customer object passed to PUT
method is empty.";
      // Return a '400 Bad Request'
      ret = StatusCode(StatusCodes.Status400BadRequest,
InfoMessage);
```

```
      // Log an informational message
      _Logger.LogInformation("{InfoMessage}",
InfoMessage);
    }
  }
  catch (Exception ex) {
    InfoMessage =
_Settings.InfoMessageDefault.Replace("{Verb}",
"PUT").Replace("{ClassName}", "Customer");

    // Return a '500 Internal Server Error'
    ErrorLogMessage = $"CustomerController.Put() -
Exception trying to update Customer: {EntityAsJson}";
    ret = HandleException<Customer>(ex);
  }

  return ret;
}
```

The **[FromBody]** attribute in the above code is optional. It is the default. I just wanted to show you that you will sometime see this.

NOTE:	Do not use [FromBody] when doing an HttpGet. The HTTP specs do not recommend this. It can cause problems with caching.

Try it Out

Run the application and click on the **PUT /api/Customer/{id}** button.

Add the **CustomerID** from the post you did in the last lab into the ID field

Add to the **Request Body**:

```
{
  "CustomerID": CUSTOMER_ID_FROM_POST,
  "NameStyle": true,
  "FirstName": "Amy - CHANGE",
  "LastName": "Smythe - CHANGE",
  "PasswordHash": "123bbdeic3332",
  "PasswordSalt": "235asdf"
}
```

Click the **Execute** button to see the results

Lab 4: Add Delete() Method to Customer Repository Class

Open the **CustomerRepository.cs** file and modify the Delete() method with the code shown in **bold** below.

```
public bool Delete(int id)
{
  Customer? entity = _DbContext.Customers.Find(id);

  if (entity != null) {
    // Locate entity to delete in the Customers DbSet
    _DbContext.Customers.Remove(entity);

    // Save changes in database
    _DbContext.SaveChanges();

    return true;
  }
  else {
    return false;
  }
}
```

Open the **CustomerController.cs** file and add a Delete() method.

```
[HttpDelete("{id}")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status500InternalServe
rError)]
public ActionResult<Customer> Delete(int id)
{
  ActionResult<Customer> ret;

  try {
    // Attempt to delete from the database
    if (_Repo.Delete(id)) {
      // Return '204 No Content'
      ret = StatusCode(StatusCodes.Status204NoContent);
    }
    else {
      InfoMessage = $"Can't find Customer Id '{id}' to
delete.";
      // Did not find data, return '404 Not Found'
      ret = StatusCode(StatusCodes.Status404NotFound,
InfoMessage);
      // Log an informational message
      _Logger.LogInformation("{InfoMessage}",
InfoMessage);
    }
  }
  catch (Exception ex) {
    // Return generic message for the user
    InfoMessage = _Settings.InfoMessageDefault
      .Replace("{Verb}", "DELETE")
      .Replace("{ClassName}", "Customer");

    ErrorLogMessage = $"CustomerController.Delete() -
Exception trying to delete CustomerID: '{id}'.";
    ret = HandleException<Customer>(ex);
  }

  return ret;
}
```

Try it Out

Run the application and click on the **DELETE /api/Customer/{id}** button.

Enter the **CustomerID** from the Post into the ID field

Click the **Execute** button

You should get a **204 – No Content** status code returned. This means the customer was deleted correctly.

Lab 5: Add Validation to Customer Class Using Data Annotations

Open the **Customer.cs** file and add some validation data annotations. Add the items in bold to the appropriate places within the Customer class.

```
[Table("Customer", Schema = "SalesLT")]
public partial class Customer
{
  // CONSTRUCTOR CODE HERE

  [Key]
  [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
  [Required()]
  public int CustomerID { get; set; }

  [Required()]
  public bool NameStyle { get; set; }

  [StringLength(8, MinimumLength = 2, ErrorMessage =
"{0} must be between {2} and {1} characters long.")]
  public string? Title { get; set; }

  [Required()]
  [StringLength(50, MinimumLength = 2, ErrorMessage =
"{0} must be between {2} and {1} characters long.")]
  public string FirstName { get; set; }

  [StringLength(50, MinimumLength = 0, ErrorMessage =
"{0} must be between {2} and {1} characters long.")]
  public string? MiddleName { get; set; }

  [Required()]
[StringLength(50, MinimumLength = 2, ErrorMessage = "{0}
must be between {2} and {1} characters long.")]
  public string LastName { get; set; }

  // REST OF THE CODE HERE
}
```

Try it Out

Run the application and click on the **POST /api/Customer** button.

Add the following input into the **Request Body**

```
{
  "NameStyle": true,
  "Title": "M",
  "FirstName": "A",
  "MiddleName": "B",
  "LastName": "S",
  "Suffix": "",
  "CompanyName": "Smythe Motors",
  "SalesPerson": "Gene",
  "EmailAddress": "Amy.Smythe@smythemotors.com",
  "Phone": "(977) 333-9938",
  "PasswordHash": "123bbdeic3332",
  "PasswordSalt": "235asdf"
}
```

Click the **Execute** button.

You should see the following errors appear.

Model binding and validation is built-in to the MVC controller.