Add, Edit, Delete Lab

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;
}
```

Create Post/Insert Method

Open the **CustomerRouter.cs** file and **add** an Insert() method to look like the code shown below.

```
protected virtual IResult Insert(Customer entity,
IRepository<Customer> repo)
{
  IResult 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 =
Results.Created($"/{UrlFragment}/{entity.CustomerID}",
entity);
    }
    else {
      InfoMessage = $"Customer object passed to POST
method is empty.";
      // Return a '400 Bad Request'
      ret = Results.BadRequest(InfoMessage);
      // Log an informational message
      _Logger.LogInformation("{InfoMessage}",
InfoMessage);
    }
  }
  catch (Exception ex) {
    // Return generic message for the user
    InfoMessage = _Settings.InfoMessageDefault
      .Replace("{Verb}", "POST")
      .Replace("{ClassName}", "Customer");

    // Log the exception and return a '500' status
    ErrorLogMessage = $"CustomerRouter.Post() -
Exception trying to insert a new customer:
{EntityAsJson}";

    ret = HandleException(ex);
  }

  return ret;
}
```

Add a app.MapPost() method call in the AddRoutes() method

```
app.MapPost($"/{UrlFragment}", (Customer entity,
IRepository<Customer> repo) => Insert(entity, repo))
   .WithTags(TagName)
   .Produces(201)
   .Produces<Customer>()
   .Produces(400)
   .Produces(500);
```

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 the customer.

Lab 3: Finish Update() Method

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

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

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

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

  return entity;
}
```

Open the **CustomerRouter.cs** file and add an Update() method.

```
protected virtual IResult Update(int id, Customer
entity, IRepository<Customer> repo)
{
  IResult 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 = Results.Ok(current);
      }
      else {
        InfoMessage = $"Can't find Customer Id '{id}' to
update.";
        // Did not find data, return '404 Not Found'
        ret = Results.NotFound(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 = Results.BadRequest(InfoMessage);
      // Log an informational message
      _Logger.LogInformation("{InfoMessage}",
InfoMessage);
    }
  }
  catch (Exception ex) {
    // Return generic message for the user
    InfoMessage = _Settings.InfoMessageDefault
      .Replace("{Verb}", "PUT")
```

```
      .Replace("{ClassName}", "Customer");

    // Log the exception and return a '500' status
    ErrorLogMessage = $"CustomerRouter.Put() - Exception
trying to update customer: {EntityAsJson}";

    ret = HandleException(ex);
  }

  return ret;
}
```

Add a call to app.MapPut() in the AddRoutes() method

```
app.MapPut($"/{UrlFragment}/{{id:int}}", (int id,
Customer entity, IRepository<Customer> repo) =>
Update(id, entity, repo))
   .WithTags(TagName)
   .Produces(200)
   .Produces<Customer>()
   .Produces(400)
   .Produces(404)
   .Produces(500);
```

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: Finish Delete() Method

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

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

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

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

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

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

```
protected virtual IResult Delete(int id,
IRepository<Customer> repo)
{
  IResult ret;

  try {
    // Attempt to delete from the database
    if (repo.Delete(id)) {
      // Return '204 No Content'
      ret = Results.NoContent();
    }
    else {
      InfoMessage = $"Can't find Customer Id '{id}' to
delete.";
      // Did not find data, return '404 Not Found'
      ret = Results.NotFound(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");

    // Log the exception and return a '500' status
    ErrorLogMessage = $"CustomerRouter.Delete() -
Exception trying to delete CustomerID: '{id}'.";

    ret = HandleException(ex);
  }

  return ret;
}
```

Add a call to app.MapDelete() in the AddRoutes() method

```
app.MapDelete($"/{UrlFragment}/{{id:int}}", (int id,
IRepository<Customer> repo) => Delete(id, repo))
   .WithTags(TagName)
   .Produces(204)
   .Produces<Customer>()
   .Produces(404)
   .Produces(500);
```

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