Code Review Document

Grocery Store Feature Branch

Mark Nichols

Step 1 – Setting up the database

Steps taken include:

- dotnet tool install --global dotnet-ef
- dotnet ef migrations add InitialCreate
- examine the migration files (both up and down methods)
- Review the database schema in database schema in *Migrations/GrocertyStoreContextModelSnapshot.cs*.
- dotnet ef database update
- Review the database and check relationships in database Groery_dev

Relationships were setup in the data annotation files. Fluent api methods were added for future database changes were needed to supplement the data annotation files.

Database is seed with the initial migration.

Step 2 – Creating the Unit of Work design pattern

I used the unit of work design pattern rather than the repository pattern. The unit of work incorporates all of the repositories. This allows for the unit of work repository to be injected into the controllers rather than all the repositories separately. This is a clean approach that makes all repositories available.

Using the unit of work allows for all entity framework changes (adds, updates, deletes, etc.) to be built up in memory and the save method is called once at the end. If one transaction fails then the save method will not be successful. This is important for adding records to multiple tables and only using one save.

Step 3 – Customer controller test plan

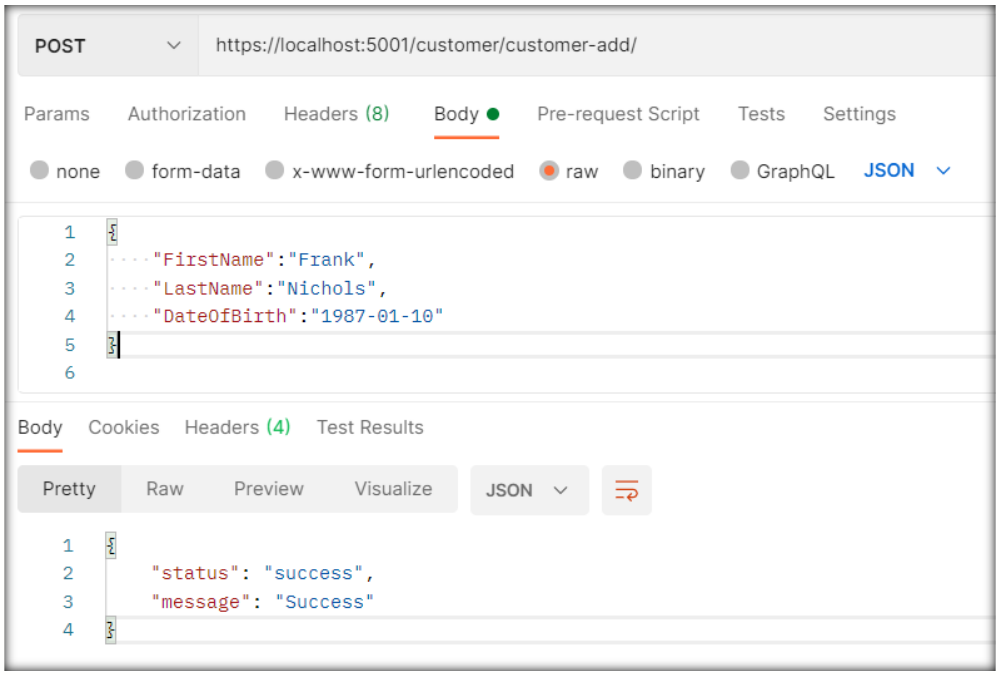
Tests conducted with postman.

Testing the Customer Add:

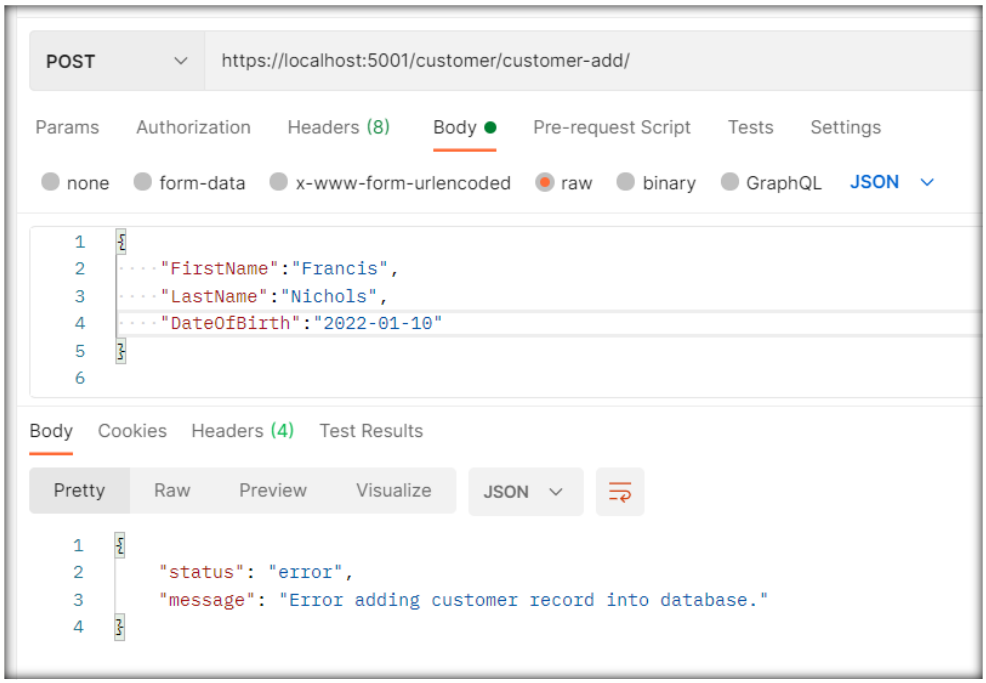
https://localhost:5001/customer/customer-add


```
{
  "FirstName":"Mark",
  "LastName":"Nichols",
  "DateOfBirth":"1932-01-10"
}
```

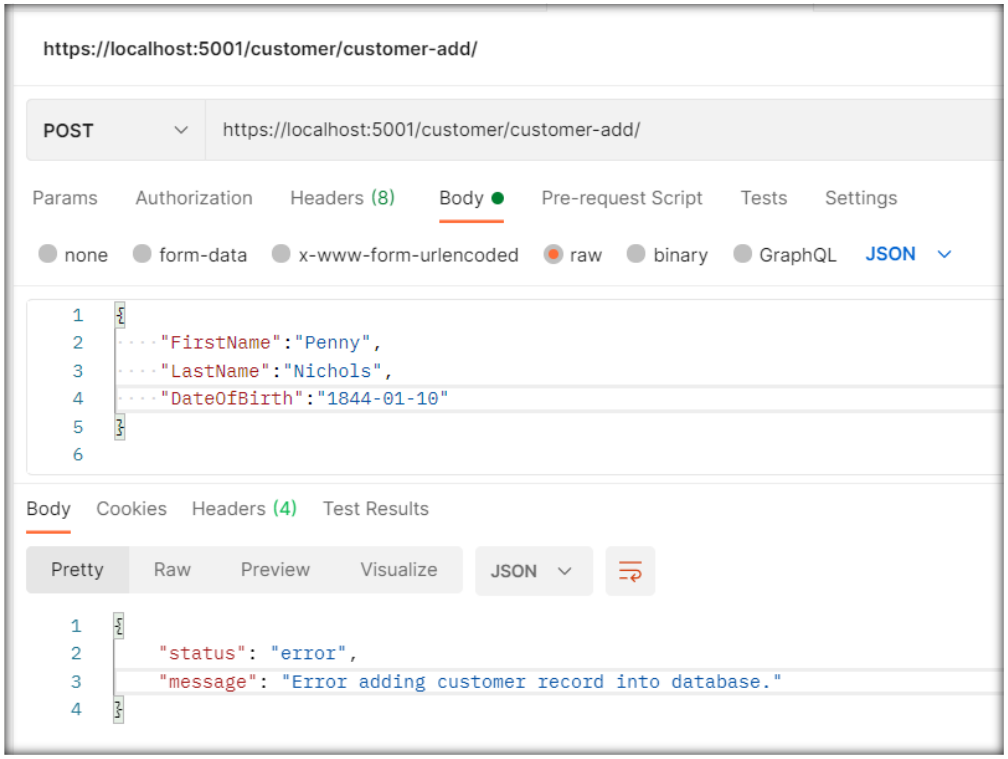
Test of successful customer:



Test of a customer error. Birth date is in the future and not allowed.



Test of a customer error. Birth date is greater than 120 years ago and is not allowed.

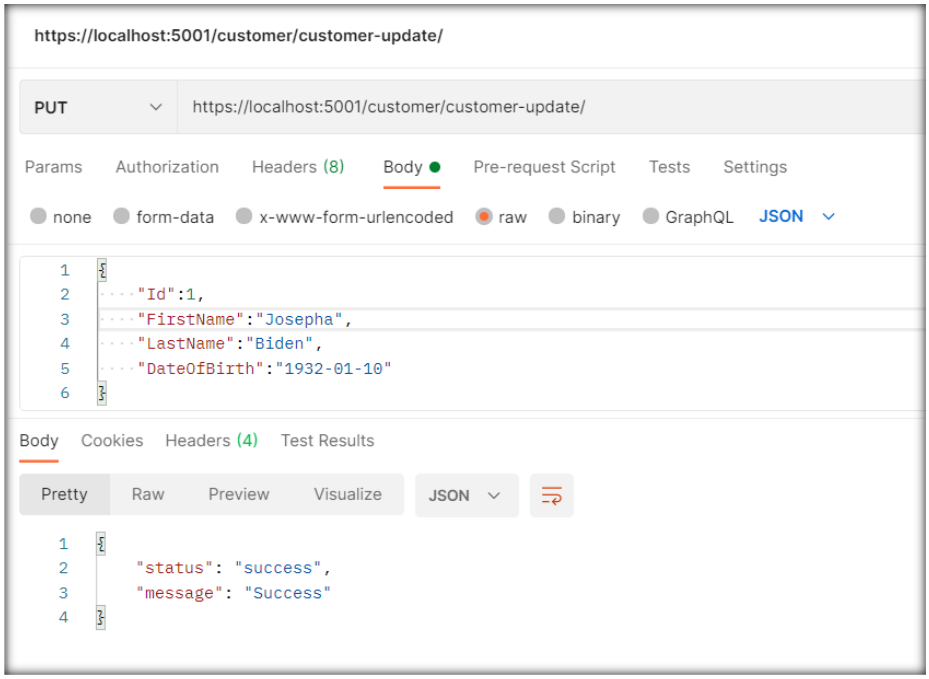


Testing of the customer update

Testing the Customer Update

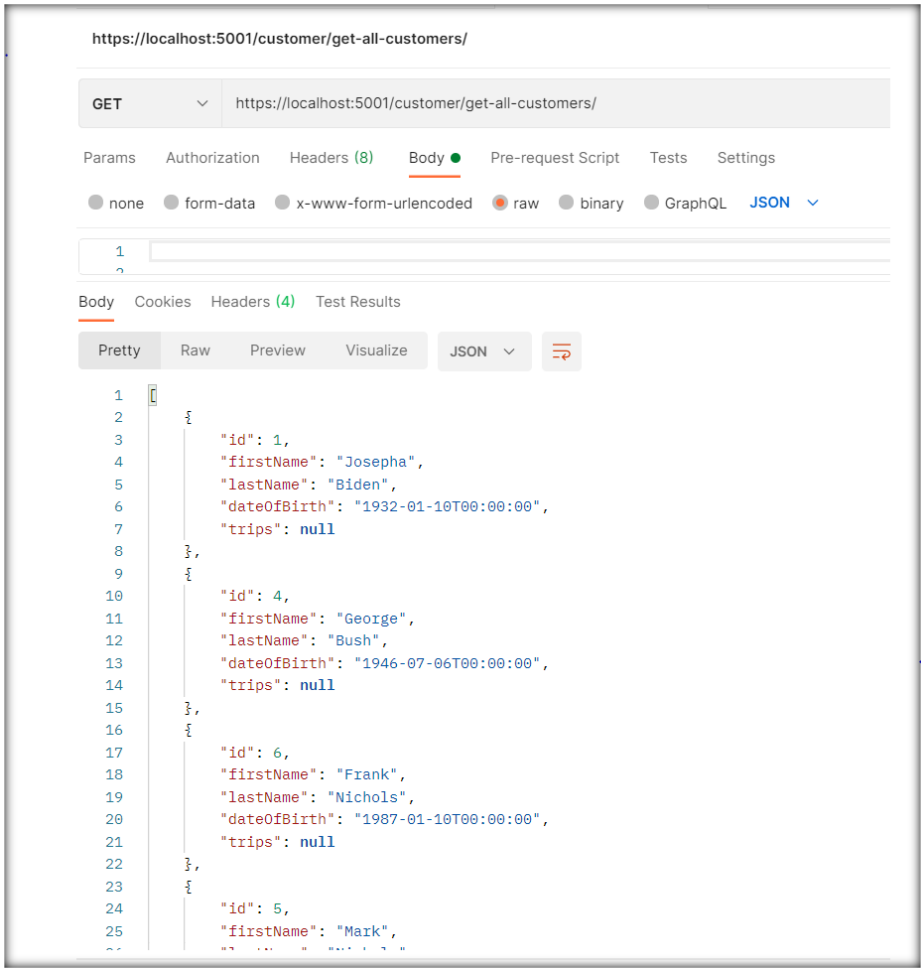
https://localhost:5001/customer/customer-update

```
{
    "Id":1,
    "FirstName":"Josepha",
    "LastName":"Biden",
    "DateOfBirth":"1932-01-10"
}
```



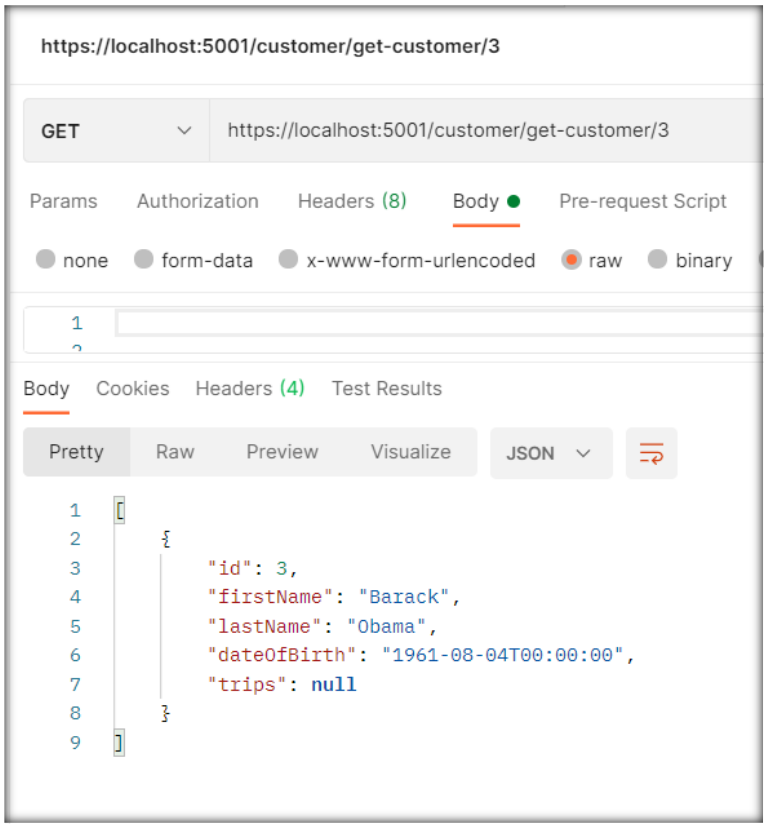
Testing the get all customers

https://localhost:5001/customer/get-all-customers/



Testing get one customer

https://localhost:5001/customer/get-customer/3



Step 4 – Unit Testing

Unit testing is to be completed during each step of the process where appropriate.

The add customer method in the customer controller includes error checking. A customer cannot have a birth date in the future. A customer also cannot be over 120 years old.

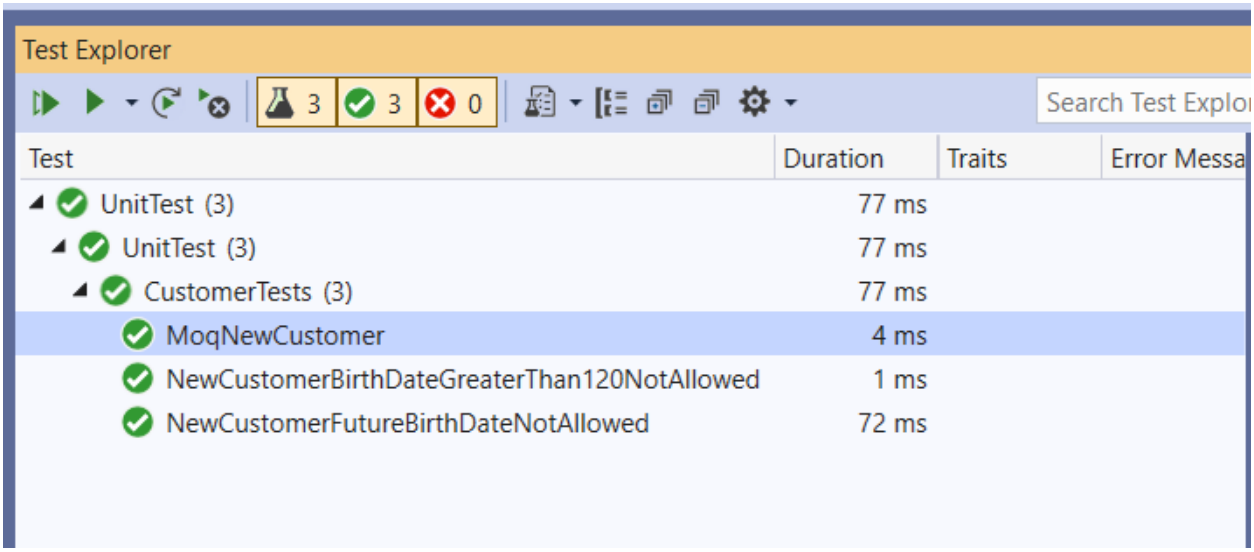
Three unit tests were included using xUnit and Moq.

Unit Test 1: MoqNewCustomer

Unit Test 2: NewCustomerFutureBirthDateNotAllowed

Unit Test 3: NewCustomerBirthDateGreaterThan120NotAllowed

Reasoning for unit tests: If the error checking in the controller is ever modified to allow for birth dates outside of 0-120 years old, then the unit tests will fail which will satisfy the purpose.



Step 5 – Clean Code Standards

At this stage, prior to the final commit, I scanned the code for the following:

- Variables and methods should be readable. They need to be understood by their name.
- No commented out code.
- Only comments where absolutely necessary. When variables and methods have descriptive names, comments are rarely necessary.
- Check for code complexity. Methods should do one thing. Avoid long complex methods that have nested for loops and nested if statements. Break them down into separate methods.
- Scan code with resharper or other automated tool such as sonarqube.