Sean Johnson Q5040916 ICA1 Documentation

Requirements Satisfied:

See videos included with submission.

Security Issues:

User passwords are hashed using SHA1 before being saved to the database. This adds some security to the system as potential intruders can not simply look at the password field and take the password. Another addition to security with passwords is that they are always hashed before being sent over to the database. If the passwords were sent over and then hashed a man in the middle attack could easily intercept and obtain them.

Most of the system requires a user to be logged in to view/access which stops general public coming onto the system and attempting to intrude without having to invest a slight amount of time, this would put a lot of them off.

Software Development frameworks/Tools

EntityFramework:

Object-relational mapper. Returns data from database as an object or collection of objects. User doesn't ever have to write SQL. When the user makes changes to the objects and saves changes, EntityFramework will automatically generate all the queries based on what was done for you.

-EntityFramework can auto generate a lot of files/code that will cause errors for the user and possibly cause you to spend more time on your project rather than saving you time.

-EF is limited when a huger domain model is in question - this puts the scalability of EF into consideration.

-If a schema is changed within the database, the solution must also be updated to match this or errors will occur.

MVC:

Model/View/Controller - Allows the user to easily represent the data in a more real-life scenario for better understanding. Model represents how we can represent our data in objects. Controller includes what we are doing to the data by understanding what needs to be carried out. Views show the data that has been minipulated by controllers.

Downsides:

- MVC inefficiently accesses data inside the views.

-Knowledge of multiple technologies is required

-Developer must have knowledge of client side code and html code

AutoMapper:

Tool used to convert DTOs constructed from data taken in by API calls and converting them to VMs to show on the web application. This method can be done by creating new VM objects and setting each variable one by one, but the automapper makes the process much quicker and easier.

Nunit + Nunit.TestAdapter

I chose to use Nunit over MsTest for unit testing because:

- Allows implementation of parametrized tests

- Assertion API is fully implemented and much easier to use/cleaner to read

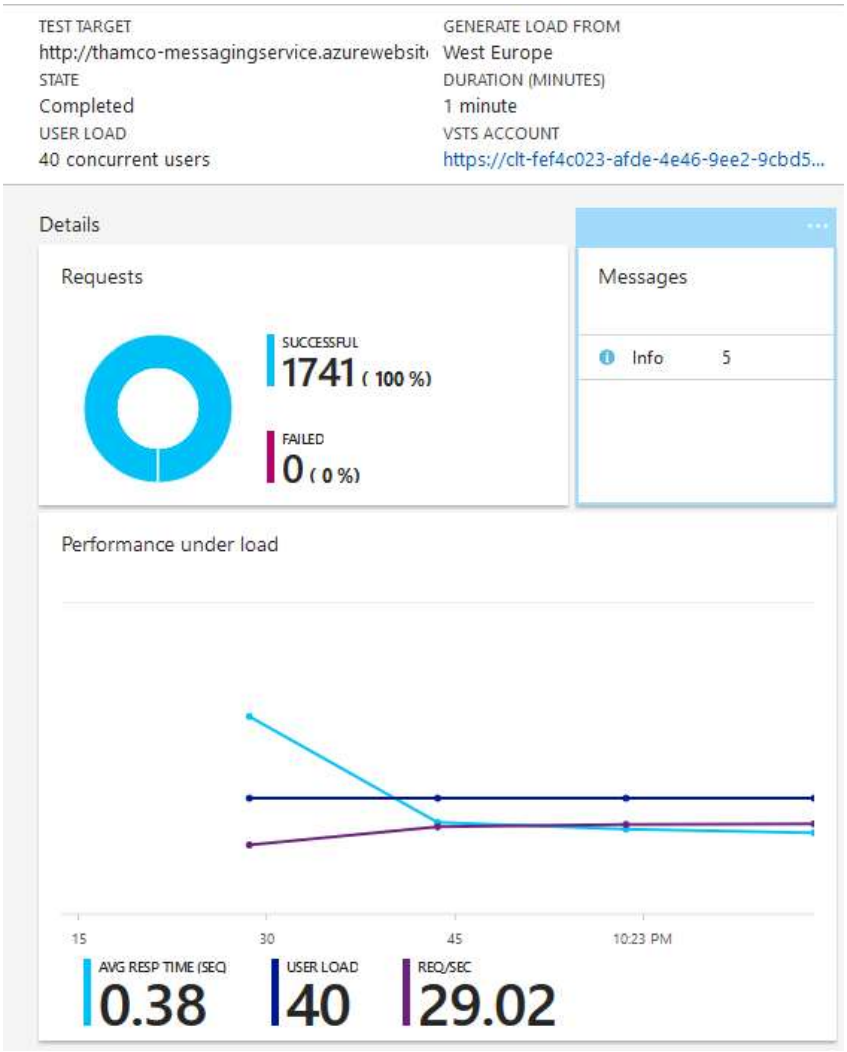
- Allows non-public classes to be test fixtures.

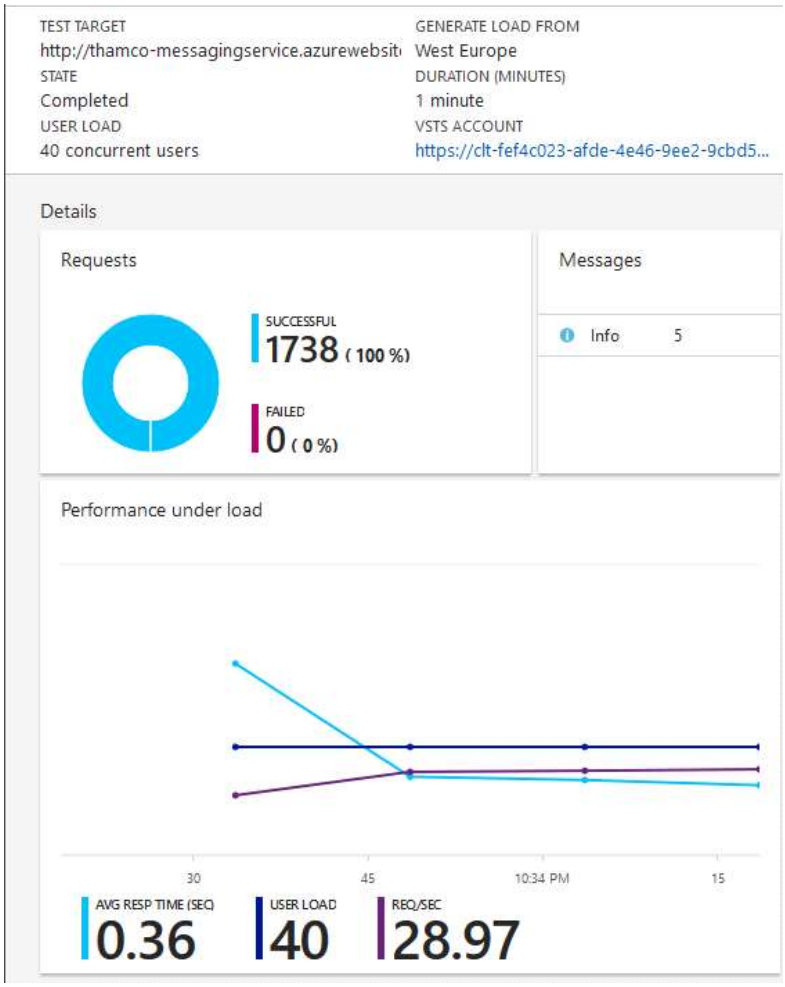
- Quicker and more reliable.

VisualStudio 15 + Sql Server Management Tool

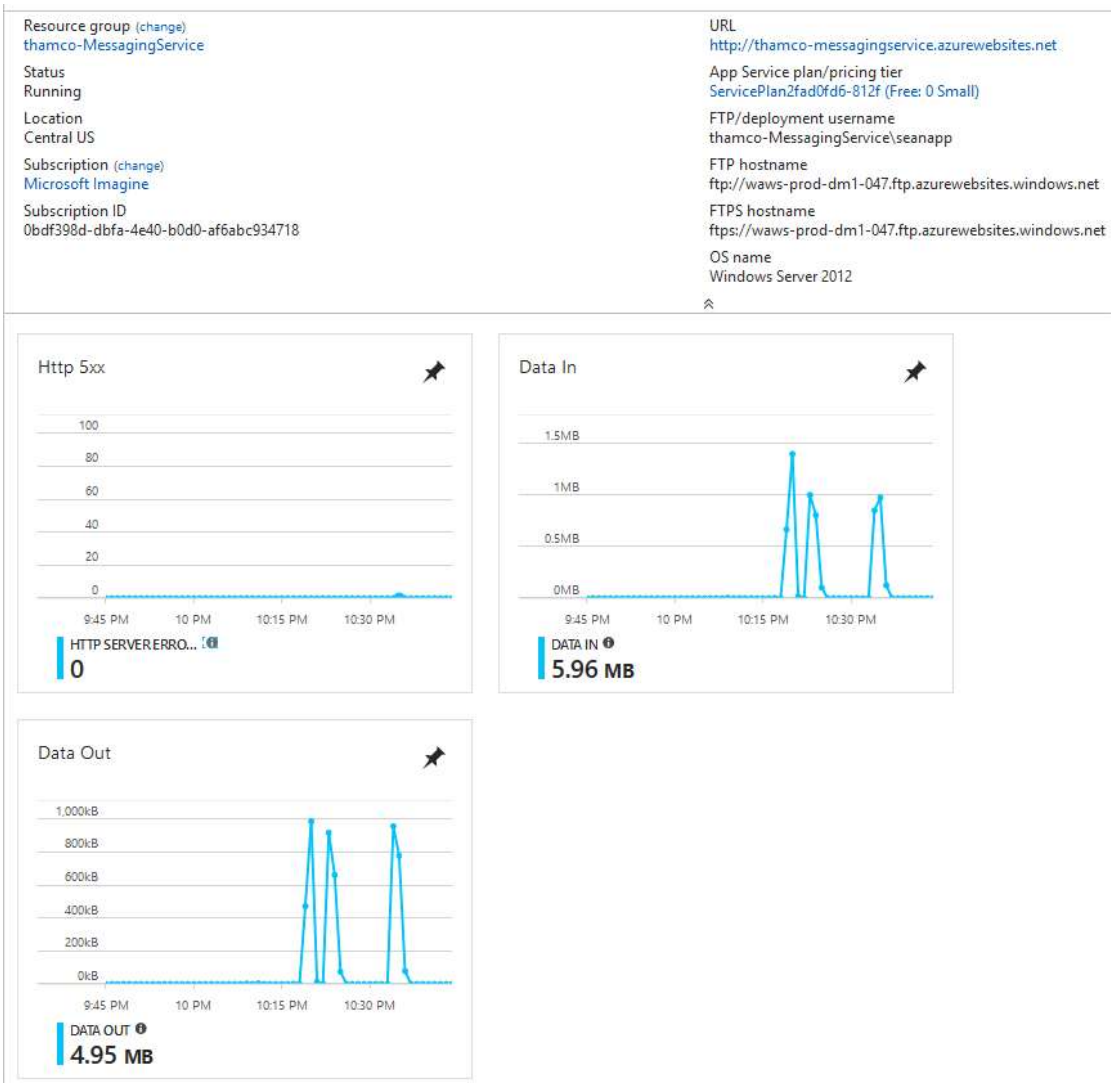
Azure:

To intergrate with my group members without having to have all API services running on the same machine at the same time. Also used Azure load testing slightly to test my API service.


TEST TARGET
http://thamco-messagingservice.azurewebsit
STATE
Completed
USER LOAD
40 concurrent users
GENERATE LOAD FROM
West Europe
DURATION (MINUTES)
1 minute
VSTS ACCOUNT
https://clt-fef4c023-afde-4e46-9ee2-9cbd5...
Details
Requests
SUCCESSFUL
1741 (100 %)
FAILED
0 (0 %)
Messages
Info 5
Performance under load
15
30
45
10:23 PM
AVG RESP TIME (SEC)
0.38
USER LOAD
40
REQ/SEC
29.02



The first test is my API to get messages by customerID and the second test is to return messages by sender (staffID).



Here is my azure portal which shows the URL for my API service (http://thamco-messagingservice.azurewebsites.net). Azure allows the user to view all kinds of information about their service as well as manage different aspects of it.

Testing Performed

Within each facade mocks have been created to test upon so that no live data is being put at risk while changes are being made.

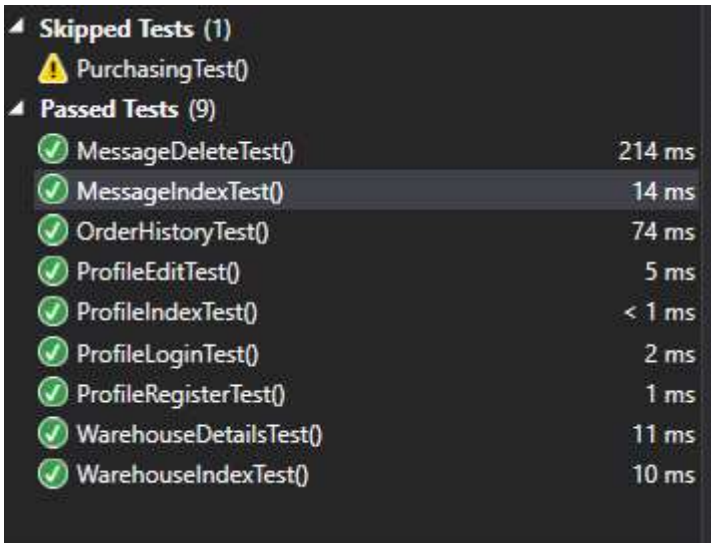
```
public MessageController()
{
    this.messageFacade = new MessageFacade();
    var mapConfig = new MapperConfiguration(cfg =>
    {
        cfg.CreateMap<MessagesDTO, ViewModels.MessagesVM>();
    });
    mapper = mapConfig.CreateMapper();
}

public MessageController(IMessageFacade messageFacade)
{
    this.messageFacade = messageFacade;
    var mapConfig = new MapperConfiguration(cfg =>
    {
        cfg.CreateMap<MessagesDTO, ViewModels.MessagesVM>();
    });
    mapper = mapConfig.CreateMapper();
}
```

When the MessagController is spun up by the web app for general purpose it will create an instance of the standard MessageFacade that accesses the APIs and live data. When I am performing Unit tests and I spin up an instance of the MessageController I have already made an instance of the MessageFacade_test to pass into it's parameters so that the message controller will never talk to the live facade while running tests:

```
[TestCase]
public void MessageIndexTest()
{
    MessageFacade_test messageFacade = new MessageFacade_test();
    MessageController messageController = new MessageController(messageFacade);
```

These test doubles apply when I am runnin unit tests via Nunit:



My tests normally include one accepted piece of data and how the application reacts as well as one unaccepted piece of data and how the system reacts. The general layout of my unit tests is as follows:

```
[TestCase]
public void ProfileLoginTest()
{
    ProfileFacade_test profileFacadeTest = new ProfileFacade_test();
    UserController userController = new UserController(profileFacadeTest);

    var user1 = new User()
    {
        UserName = "C00000001",
        Password = "pw123",
        RememberMe = true
    };

    var user2 = new User()
    {
        UserName = "C0003455",
        Password = "pw123",
        RememberMe = true
    };

    var actual = (userController.Login(user1));
    Assert.IsNotNull(actual);
    Assert.IsInstanceOf(typeof(RedirectToRouteResult), actual);

    var actual2 = (userController.Login(user2) as ViewResult).ViewData.Model;
    Assert.IsNotNull(actual2);
    Assert.IsInstanceOf(typeof(User), actual2);
}
```

Assert.IsInstanceOf(typeof(RedirecttoRouteResult) allows me to check if the return of the method was a RedirectToAction. IsNotNull allows me to check that the return actually contains some information. IsInstanceOf(typeof(User) allows me to check the view result of the Login method by using "as ViewResult).ViewData.Model".

My test doubles include lists of data generated locally which matches that of which is in the database so that the application can simulate talking to a database via a facade without calling any APIs and risking any live information. A problem using these doubles is that every time the test facade is instantiated the information within them is set back to the original set and so testing methods like registering customes which require adding information to these lists is quite useless.