Tasks

In this workshop we will set up our development environment using Docker containers. Our development environment is going to replicate the server environment that our application is going to run on but we want to be able to work without an internet connection.

The tasks will build up. First task you can explore creating a simple authentication service. Task 2 challenges you to discover what you can about Docker compose and to see if you can use that to run Task 1 from a container whilst calling the database container.

In the diagram below, we are exploring how to create the AccountService (used to authenticate users) & how it relates to the Coursework Profile Service.

Diagram

Description automatically generated

**Task 1:**

To create an AccountService in a Docker container that takes a set username and password and returns a valid or invalid message according to the input.

Context

It is a rare web application that does not require you to have some form of account and to authenticate yourself against that account. This workshop will explore how a microservice could provide some of the functionality required to make accounts happen.

In this workshop you will need to have an understanding of C# - something gained from COMP1000.

Please bear in mind that this is only one suggested way of decomposing the architecture. There are many ways to do this and there is current debate in the industry over what is the best method. Every attempt has been made to ensure best practice is followed, but this is a quickly moving field and you may well find better approaches as you develop your skills. Some of the approach shown here has been simplified to provide a better chance of understanding.

This uses Visual Studio Code to illustrate the steps, but you are free to use any Visual Studio and the visual interface to achieve the same goals. Just bear in mind that templates can and do change.

Instructions

Task 1:

Overview

In this task we create an Authenticator Development API in a Docker container that takes a set username and password and returns a valid or invalid message according to the input.

Note: In the Profile Service, the return message received will be the email address and an indicator of verified. If verified is true, the email address can be used to return the profile information, otherwise file not found should be returned.

Steps

1. Using Visual Studio Code open the terminal window.
   1. Make sure you are in a folder where you can create folders.
   2. cd to it
2. Create a new dotnet application based on the webapi template, giving the application a suitable name:

dotnet new webapi -n authenticatorDEV

Text

Description automatically generated

1. Run this on the host (your laptop or desktop). To run:
   1. click 'Run and Debug' icon (in left menu)
   2. click 'Generate C# Assets for Build and Debug' button
   3. select the project
   4. click the run button (green triangle) or press the [F5] key
   5. You are likely to be asked to install a VS code plugin / extension: C# Dev Kit

A screenshot of a computer

Description automatically generated

1. This comes with a built in weatherforecast controller (the port pppp will change):

https://localhost:pppp/weatherforecast

1. Open the editor to view the file explorer.
   1. At this stage you could initialise a GitHub repo (strongly recommended, especially if you are working on your coursework). You can do this by navigating to the Source Control menu and publish to GitHub.
2. Inside the root of the solution, create a new folder called Models.
3. Inside Models folder, create a new file – User.cs
4. Create your user class giving it the properties of a firstname, lastname, email and password. Make sure you use the getter and setter and also set up the defaults for instantiation.
   1. Add a using Microsoft.AspNetCore.Mvc;
   2. Remember to add in the namespace

**User.cs**

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Text.Json.Serialization;

using System.Text.Json;

using Microsoft.AspNetCore.Mvc;

namespace AuthenticatorDEV.Models

{

[Table("UserView")]

public class User

{

[Key]

[Column("User")]

// You should add the other fields.

public string Email { get; set; } = string.Empty;

public string Password { get; set; } = string.Empty;

public bool Valid { get; set;}

}

}

1. Add a UsersController.cs to Controllers folder
   1. Add a using Microsoft.AspNetCore.Mvc;
   2. Add in a using statement for the Models folder for this solution.
   3. Also add in the appropriate namespace.
2. Create the UsersController class – taking note of each of the lines you need.

[Route("api/[controller]")]

[ApiController]

public class UsersController : ControllerBase

{

//GET: api/Users

[HttpPost]

public IActionResult Post([FromBody] User usr)

{

bool Verified = getValidation(usr);

return Ok(new string[] {"Verified", Verified.ToString()});

}

private bool getValidation(User usr)

{

bool validation = false;

if((usr.Email == "testme@test.com")

&& (usr.Password == "insecurePWD"))

{

validation = true;

}

return validation;

}

}

}

1. You should now be able to run your project again, but this time there should be both the weather forecast controller and the users controller.
2. Add a new file to the solution – in the root add a docker file. Enter the following text:

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build-env

WORKDIR /app

COPY \*.csproj ./

RUN dotnet restore

COPY . ./

RUN dotnet publish -c Release -o out

FROM mcr.microsoft.com/dotnet/aspnet:7.0

WORKDIR /app

EXPOSE 80

COPY --from=build-env /app/out .

ENTRYPOINT [ "dotnet", "authenticatorDEV.dll" ]

1. Add a .dockerignore file to the root of the solution. Note the leading dot.

bin/

obj/

1. Before this next step, check you have Docker desktop running. Back to the terminal and run the following command to build the application and spin up the docker image. The tag is important – use your docker hub repo name. Don’t forget the dot at the end.

docker build -t yourname/authenticatordev .

1. In VS Code check the docker symbol at the side to see the image. You can also check the Docker dashboard to see it. Now run the application by typing in the following:

docker run -p 8080:80 --name authenticator yourname/authenticatordev

1. Open up something like postman and test with test with [testme@test.com](mailto:testme@test.com) and insecurePWD

Recommended Reading:

This task has been based from this tutorial here.

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/min-web-api?view=aspnetcore-6.0&tabs=visual-studio-code>

Task 2:

Overview

It is not possible to call one docker container from another directly. This is due to the security permissions. There are a number of mechanisms that can be used, Kubernetes is one but the one you are encouraged to explore for yourself is Docker Compose.

Your challenge is to search online and discover how you would use Docker Compose to run the AccountService in one docker container whilst it works with the database (described by the SQL below) in another docker container (we created a Docker Compose file in week 1).

HINT: Les Jackson YouTube videos are a very useful source.

CREATE TABLE [dbo].[User] (

    [User\_ID] INT IDENTITY(1,1) NOT NULL,

    [Surname] VARCHAR(255),

    [Forename] VARCHAR(255),

    [Email] VARCHAR(255)NOT NULL,

    [Password] VARCHAR(255)NOT NULL,

CONSTRAINT PK\_User PRIMARY KEY (User\_ID]);

GO

INSERT INTO dbo.[User]  (Surname, Forename, Email, Password)

VALUES ('Smith', 'Allen', 'A.Smith@domain.co.uk', 'red1234');

GO

SELECT \* FROM dbo.[User];