**SWE40006 – Deployment Activity 3**

**Student Details**

* **Full name:** Dang Khoa Le
* **Student ID:** 103844421
* **Due date:** 14/09/2025
* **Submission date:** 10/09/2025

**Unit Details**

* **Unit code & name:** SWE40006 – Software Deployment and Evolution
* **Semester:** Semester 2, Year 2025

**Declaration of Task Levels Attempted**

* **Task 3.1 (Pass)** – Create Azure account and deploy calculator Python app to Azure.
* **Task 3.2 (Credit)** – Develop calculator C# app and deploy to Azure. Deactivate app.
* **Task 3.3 (High Distinction)** – Install PHP, develop a webapp and deploy to Azure.

**Prerequisites**

* **VS Code** (preferred IDE for MacOS)
* **Azure CLI**
* **Python 3.11** (for Task 3.1)
* **.NET 8.0 SDK** (for Task 3.2)
* **PHP 8.3** (for Task 3.3)

### **Public URLs**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Azure App | URL (public) | Runtime | Region |
| 3.1 Python | *swe40006-python-calculator-2* | https://*swe40006-python-calculator-2.azurewebsites.net* | Python 3.11 (Linux) | Australia East |
| 3.2 C# | *swe40006-*csharp*-calculator* | https://*swe40006-*csharp*-calculator*.azurewebsites.net | .NET 8.0 (Linux) | Australia East |
| 3.3 PHP | *swe40006-php- calculator* | https://*swe40006-php- calculator*.azurewebsites.net | PHP 8.3 (Linux) | Australia East |

Table of Contents

[3.1 - Pass: Deploy Existing Python App to Azure 3](#_Toc208396150)

[3.1.0 Installations 3](#_Toc208396151)

[3.1.1 Create Azure App Service 6](#_Toc208396196)

[3.1.2 Implement FastAPI Calculator 8](#_Toc208396209)

[3.1.3 Deploy from VS Code (zip deploy) 10](#_Toc208396235)

[3.1.4 Validate Live App 12](#_Toc208396265)

[3.1.5 Inspect App from Public URL 13](#_Toc208396268)

[3.2 - Credit: Deploy C# App to Azure 15](#_Toc208396270)

[3.2.1 Create the ASP.NET Core minimal API 15](#_Toc208396271)

[3.2.2 Create Azure App Service (.NET 8.0) 17](#_Toc208396276)

[3.2.3 Inspect App from Public URL 19](#_Toc208396310)

[3.2.4 Deactivate App 19](#_Toc208396312)

[3.3 – High Distinction: Install & PHP web app Azure deployment 21](#_Toc208396321)

[3.3.1 PHP Installations 21](#_Toc208396322)

[3.3.2 PHP Web App Development 21](#_Toc208396338)

[3.3.3 Create Azure App Service (PHP 8.2) and Deployment 24](#_Toc208396339)

[3.3.4 Validate Live App 24](#_Toc208396340)

[Appendix 26](#_Toc208396341)

**3.1 - Pass: Deploy Existing Python App to Azure**

**3.1.0 Installations**

* **Azure CLI:** Install Azure CLI to MacOS using command:
* **With Brew:**

brew update && brew install azure-cli

* **With pip** (pip3 since we are using Python3.x on MacOS)**:**

pip3 install azure-cli

Then we will need to edit our  shell's PATH environment variable .zshrc to ensure az command is in my system's PATH. Since we installed with pip and currently have Python 3.11 built-in, I will export this path to my .zshrc file.

A close-up of a computer

Description automatically generated

Figure 1. Edit system path file .zshrc

We can verify Azure CLI installation and version with az –version command.

A screenshot of a computer

Description automatically generated

Figure 2. Check Azure CLI version and installation

* **IDE**: As a Mac user, I decided to progress with this task onward using VS Code.

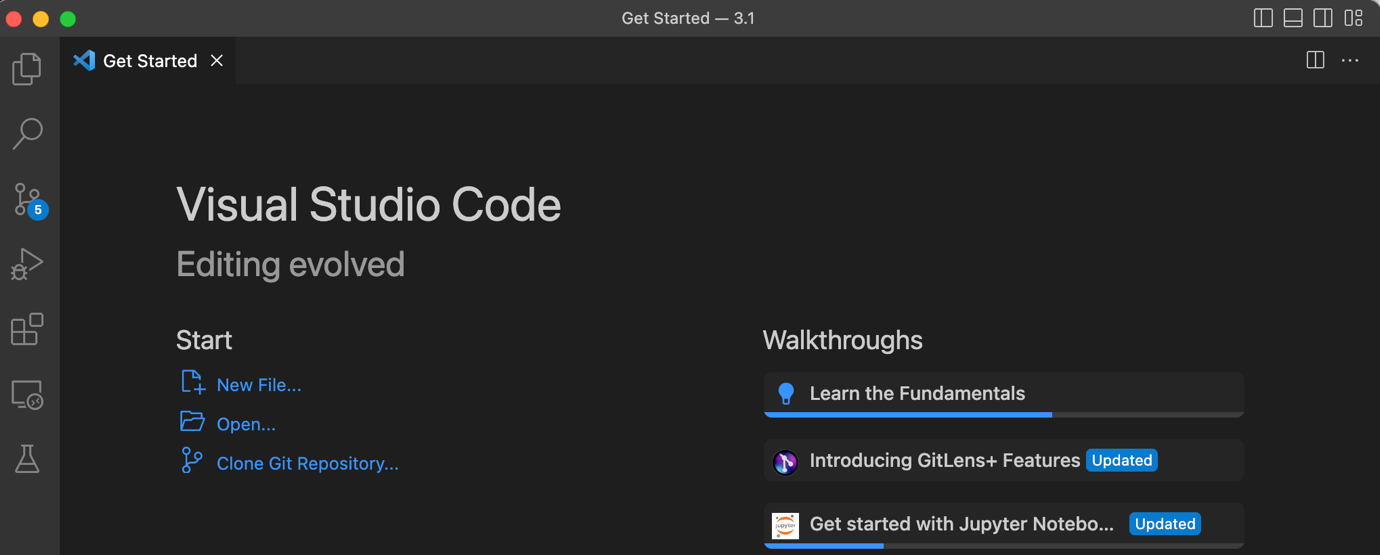


Figure 3. VS Code as IDE preference

**Azure Account**

I previously already created an Azure account from my student email, thus I only need to login with this account on Azure portal.

A screenshot of a computer

Description automatically generated

Figure 4. Microsoft Azure website dashboard

**Sign in and Create resource group**

I will login (verify account) and create resource group (named *rg-swe40006* with region/location at Australia East) from my device terminal with CLI commands following:

az login

az account show

az group create -n rg-swe40006 -l australiaeast

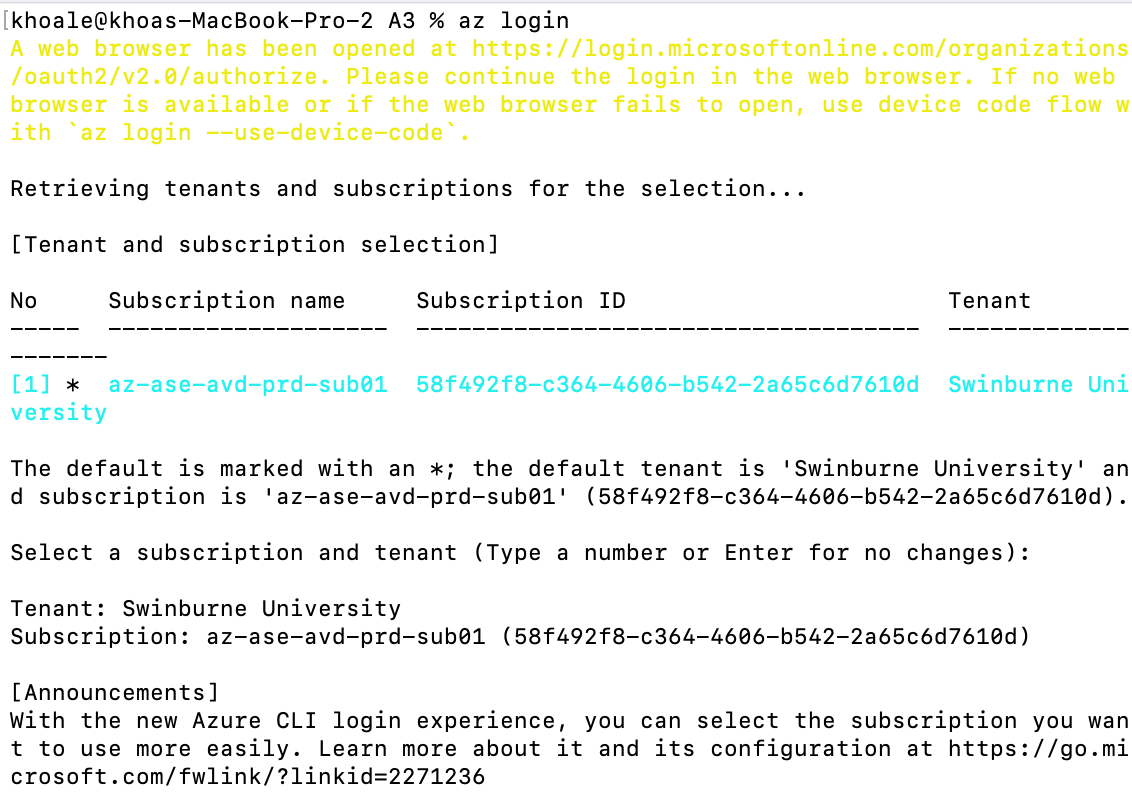


Figure 5. Azure CLI login

A screenshot of a computer code

Description automatically generated

Figure 6. Azure CLI verify account

I need to verify my current permitted/assigned roles (on subscription) for this student email account, which doesn’t exist any role:

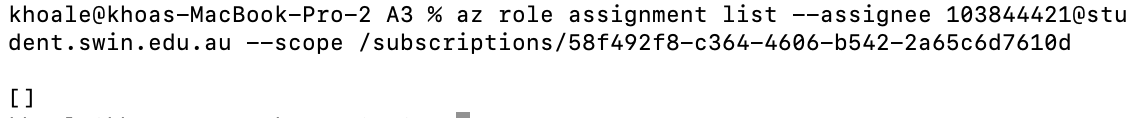
****

Figure 7. No available role on Azure CLI

I returned to Azure dashboard website and on Subscriptions, I choose Azure for Students option. Which then prompt me to sign up again for academic verification. Upon proceed successfully, I will have an Education account with $100 free credit.

A screenshot of a computer

Description automatically generated

Figure 8. Azure website Subscriptions page

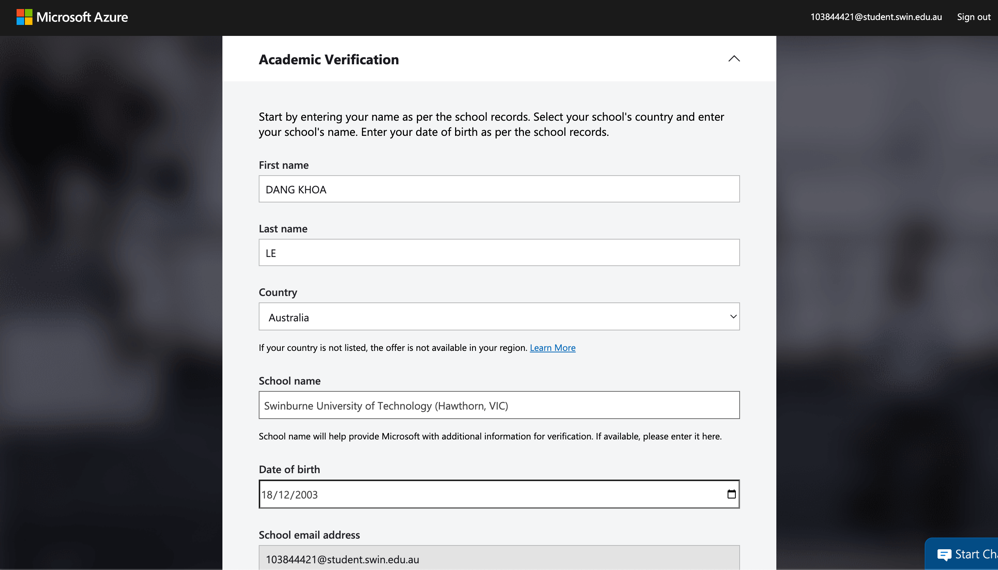


Figure 9. Azure website Academic Verification page

**A screenshot of a computer

Description automatically generated**

Figure 10. Azure website Education account page

Then I proceed to Subscription again, which show my ‘Azure for Students’ section displayed. I then check for its ID and set that subscription ID to my terminal CLI by running:

az account set –subscription id

A screenshot of a computer

Description automatically generated

Figure 11. Azure subscription ID

Once set the subscription, we will proceed with the resource group creation (named *rg-swe40006* with region/location at Australia East).

A screenshot of a computer

Description automatically generated

Figure 12. Azure resource group details

**3.1.1 Create Azure App Service**

**3.1.1.1 Create an App Service plan (free-tier)**

We need to create an App Service plan (named *plan-swe40006*) for our resource group. First, we will need to register Microsoft.Web resource provider, which is required to create App Services (Web Apps, App Service plans, etc):

az provider register –namespace Microsoft.Web

Then, we will configure a free Linux plan for our resource group with:

az appservice plan create -g rg-swe40006 -n plan-swe40006 --sku F1 --is-linux



Figure 13. App Service plan details

**3.1.1.2 Create the Python web app**

For this Azure Linux App Service’s Web app creation, the current supported Python runtime stack (locally) is Python 3.11, so we will explicitly use **3.11** in the --runtime flag. We create the webapp service named *swe40006-python-calculator* that is attached to our resource group and plan as previously configured. Note that the app name must be globally unique. This CLI command was executed:

az webapp create -g rg-swe40006 -p plan-swe40006 -n swe40006-python-calculator --runtime "PYTHON:3.11"

A screenshot of a computer program

Description automatically generated

Figure 14. swe40006-python-calculator details

**3.1.2 Implement FastAPI Calculator**

**3.1.2.1 Project layout**

The root project named “3.1” include main.py (Python script with main functionalities) and requirements.txt (for Python libraries and dependencies version control consistency when creating a project environment).

A screenshot of a computer

Description automatically generated

Figure 15. Calculator Python app setups

**3.1.2.2 Scripts**

**main.py:** A FastAPI app with routes including basic POST request for add, subtract, multiply and divide computations. It also support GET request to check for application health status.

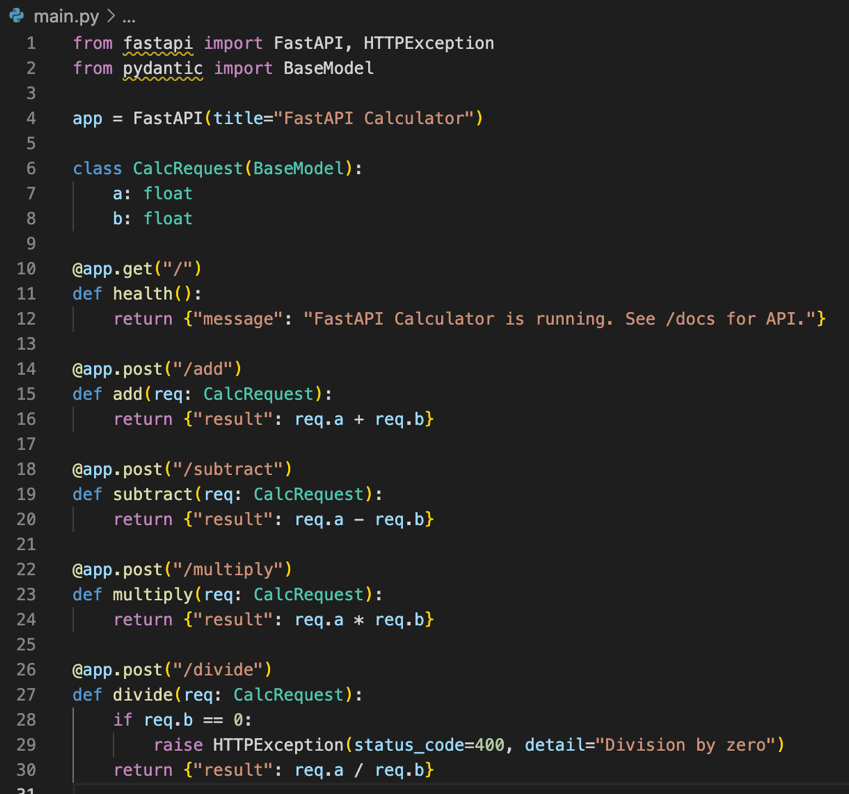
****

Figure 16. main.py

**requirements.txt:** This file storing all required dependencies and modules for our Python FastAPI, with version specifications that support our Python 3.11 runtime.

A screenshot of a computer

Description automatically generated

Figure 17. requirements.txt

**3.1.2.3 Local Testing**

At project directory *3.1*, we create a virtual environment (.venv) and building Python wheels with existing dependencies specified at *requirements.txt* (so this way our current Python environment setup locally will not be impacted from these version changes). Then we run the FastAPI app with unvicorn, following these CLI commands:

python -m venv .venv

pip install -r requirements.txt

uvicorn main:app –reload

Upon launching, the app will be available locally at port 8000 (default Python app startup port)

**A screenshot of a computer

Description automatically generated**

Figure 18. App local test

We will try to execute a few examples, editing value schema (JSON object), then trigger Execute button, which will simulate a CURL request to our backend which execute mathematical computation.

A screenshot of a computer

Description automatically generated

Figure 19. Example test with Add route

A screenshot of a computer

Description automatically generated

Figure 20. Accurate addition output

Other mathematical computation routes are tested also, which all show accurate outputs.

**3.1.3 Deploy from VS Code (Zip Deploy)**

**3.1.3.1 VS Code Azure setup**

* Install Azure App Service extension for deployment.

A screenshot of a computer

Description automatically generated

Figure 21. VS Code extension Azure App Service

A screenshot of a computer

Description automatically generated

Figure 22. Azure App Service extension details

* In Explorer, right-click *3.1* → Deploy to Web App → select *swe40006-python-calculator* app → Confirm.
* Or otherwise, we can use CLI directly:
  + Step 1: Prepare dependency environment (requirements.txt) and zip this project

A screenshot of a computer

Description automatically generated

Figure 23.Prepare environment and zip project

* + Step 2: Deploy using *az webapp deploy*. I sent the zip file over the pre-configured webapp by pre-configured resource group *rg-swe40006*.

az webapp deploy \

--resource-group rg-swe40006 \

--name swe40006-python-calculator \

--src-path swe40006-python-calculator.zip \

--type zip

A screenshot of a computer

Description automatically generated

Figure 24. Deploy zipped app

**3.1.3.2 Configure startup command (FastAPI on App Service)**

* Azure Portal → *swe40006-python-calculator* → Configuration → General Settings → Startup Command:

gunicorn -w 1 -k uvicorn.workers.UvicornWorker main:app

* Or using CLI. I configured main.py as my startup file that is rendered by *gunicorn* (with 1 uvicorn worker):

az webapp config set \

--resource-group rg-swe40006 \

--name swe40006-python-calculator \

--startup-file "gunicorn -w 1 -k uvicorn.workers.UvicornWorker main:app"

A screenshot of a computer program

Description automatically generated

Figure 25. Configured app startup

**3.1.4 Validate Live App**

Initially, the app isn’t accessible from the public URL (<https://swe40006-python-calculator.azurewebsites.net>), I rechecked the app building status, which it persisted at starting state, then I checked with *az webapp show* command, which state that our plan (registered for this webapp) has “QuotaExceeded” status. This could be a result of over-testing from myself while using a free-tier account.



Figure 26. Webapp QuotaExceeded status

So then, I recreate another alternative plan (or I will have to wait until the next day to try on this plan again), using the same resource group, with this CLI command:

az appservice plan create \

--resource-group rg-swe40006 \

--name plan-swe40006-alt \

--sku F1 --is-linux \

--location southeastasia

And recreate another app service container app with this new plan named *swe40006-python-calculator-2*:

az webapp create \

--resource-group rg-swe40006 \

--plan plan-swe40006-alt \

--name swe40006-python-calculator-2 \

--runtime "PYTHON:3.11"

Similarly, I re-setup the same startup command for this app:

az webapp config set \

--resource-group rg-swe40006 \

--name swe40006-python-calculator-2 \

--startup-file "gunicorn -w 1 -k uvicorn.workers.UvicornWorker main:app"

And redeploy the zip file to this app:

az webapp deploy \

--resource-group rg-swe40006 \

--name swe40006-python-calculator-2 \

--src-path swe40006-python-calculator.zip \

--type zip

The deployment is successful.

A screenshot of a computer program

Description automatically generated

Figure 27. Successful webapp deployment

We can also check the log with CLI command:

az webapp log tail -g rg-swe40006 -n swe40006-python-calculator-2

And to restart the web app with CLI command:

az webapp restart -g rg-swe40006 -n swe40006-python-calculator-2

**3.1.5 Inspect App from Public URL**

We can inspect the app from: <https://swe40006-python-calculator-2.azurewebsites.net/docs>

A screenshot of a computer

Description automatically generated

Figure 28. Web app demo on public URL

**3.2 - Credit: Deploy C# App to Azure**

**3.2.1 Create the ASP.NET Core minimal API**

In this task, we will create a simple Calculator C# app on VS Code and deploy to Azure cloud.

**3.2.1.1 Scaffold project**

We create a minimal ASP.NET Core 8.0 web API project named *3.2* with CLI command:

dotnet new web -o 3.2

**3.2.1.2 Code (Program.cs)**

We update the default Program.cs with a new setup that support POST request for add, subtract, multiply, and divide endpoints.

We import AspNetCore.Builder, AspNetCore.Http, and Extensions.Hosting from Microsoft as basic fundamentals modules to support our web app.

**Root Endpoint (Health Check + UI):**

* When user visit the **root URL** (/), it returns a basic HTML calculator form.
* Users can enter two numbers (a and b) and choose an operation (+, −, ×, ÷).
* Purpose: provide a simple **user interface** for manual calculation and app confirmation.

**Input Model (API side):**

record CalcRequest(double a, double b);

* Defines a simple data structure with two numbers (double): a and b.
* Used as the request body for **API endpoints** (JSON input).

**Calculator Endpoints (API):**

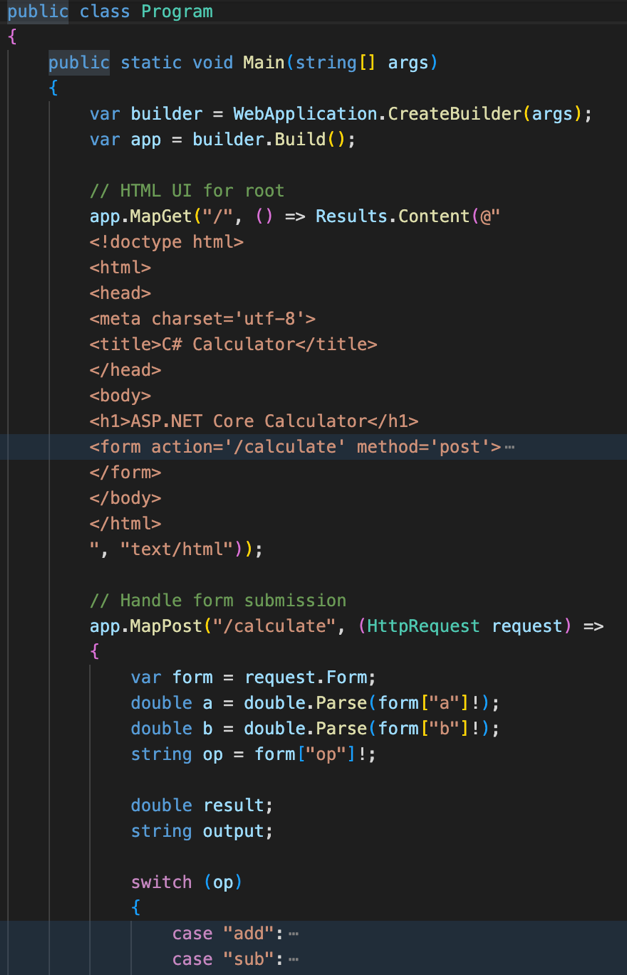
* Each operation is still exposed as a **POST endpoint** for programmatic access:
  + /add → returns sum of a and b
  + /subtract → returns difference
  + /multiply → returns product
  + /divide → returns quotient (returns HTTP 400 Bad Request with an error message if b = 0)

**Form Submission Endpoint (UI):**

* /calculate → handles HTML form submission from the root page and displays the result in a web page.

**App.Run():**

* Starts the web server and listens for incoming requests on both the HTML UI and the JSON API.

A screen shot of a computer program

Description automatically generated

Figure 29&30. C# calculator web app Program.cs

**3.2.1.3 Local Testing**

We run the app with command:

dotnet run

The app would be available at http://localhost:5244:

A screen shot of a computer program

Description automatically generated

Figure 31. C# calculator web app local run

Here is a demonstration on subtraction attempt 10 – 10. We can click to the = (equal) button, which show the result on /calculate endpoint.

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Figure 32&33. C# calculator web app test math operation

**3.2.2 Create Azure App Service (.NET 8.0)**

We first create the web app named *swe40006-csharp-calculator* sharing the same *rg-swe40006* resource group and *plan-swe40006-alt* resource plan with the app from *3.1*. Our runtime will be .NET 7.0 as configured at default when created the C# app locally.

az webapp create -g rg-swe40006 -p plan-swe40006-alt -n swe40006-csharp-calculator --runtime "DOTNETCORE:7.0"

However, upon running this CLI command, it shows that DOTNETCORE 7.0 runtime is not supported and indeed the compatible versions are listed to be 8.0, 9.0 and 10.0.

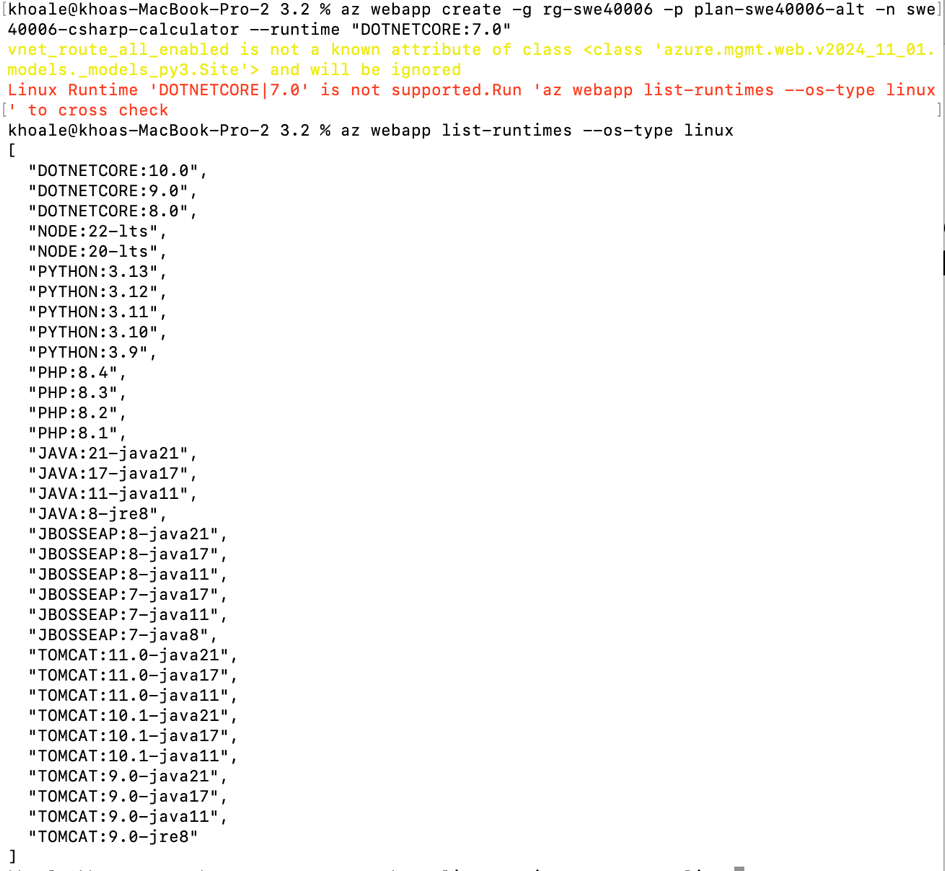


Figure 34. C# calculator web app creation failure with unsupported DOTNETCORE runtime

And since my local setup is configured with .NET7.0, I will have to update .NET version (to 8.0 preferably, from Microsoft website), then, I will only update 3.2.csproj file’s *TargetFramework* tag from net7.0 to net8.0, such:

<TargetFramework>net8.0</TargetFramework>

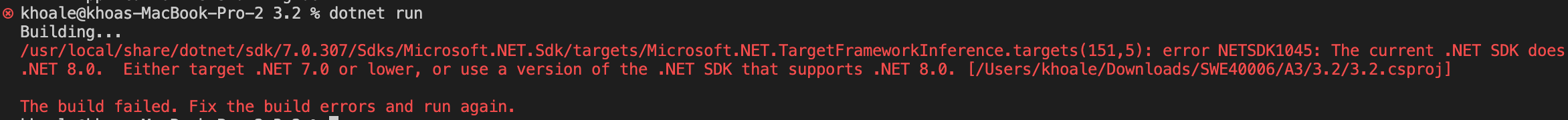


Figure 35. C# calculator web app local test failure on net8.0

And proceed to deployment with this net8.0 framework targeted, I rerun the App Service container creation CLI command:

az webapp create -g rg-swe40006 -p plan-swe40006-alt -n swe40006-csharp-calculator --runtime "DOTNETCORE:8.0

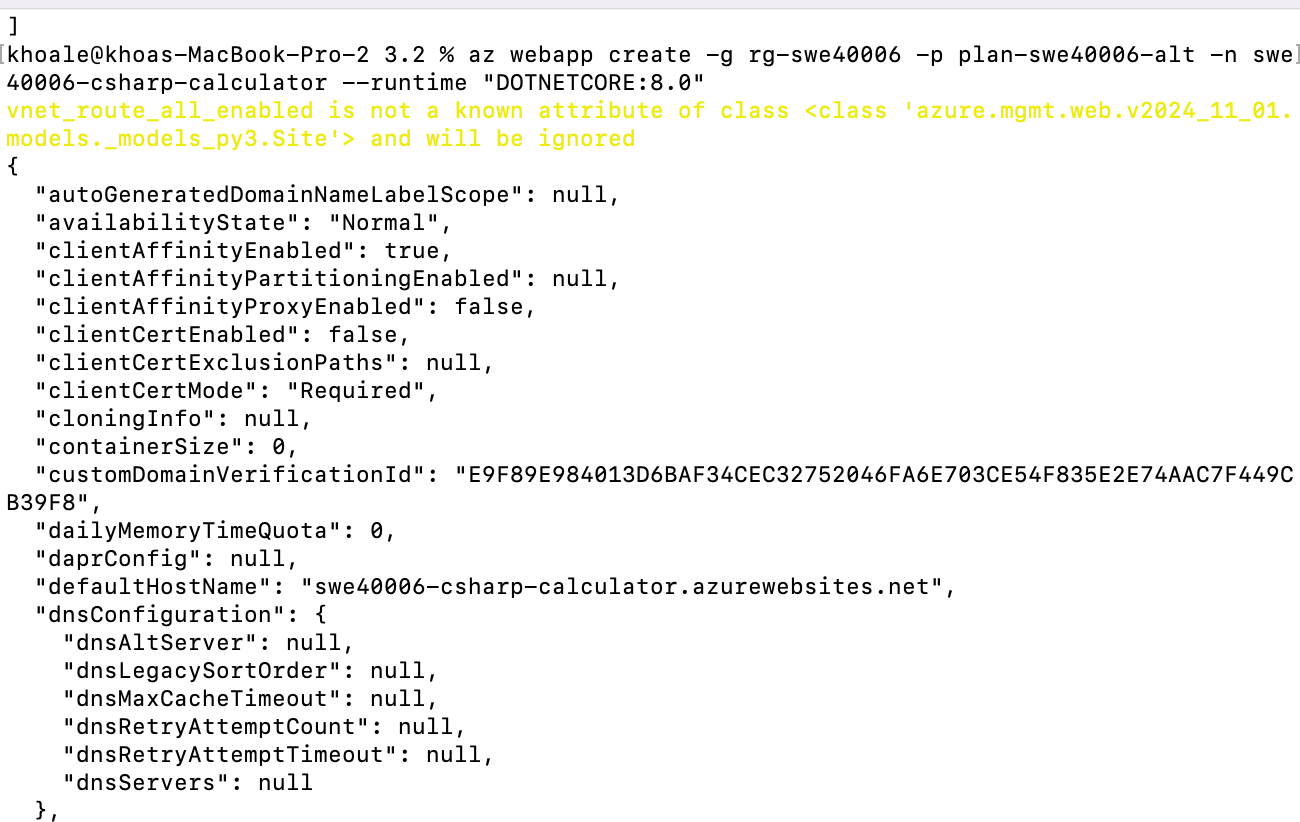


Figure 36. Successful C# calculator web app creation

Then, similar to task *3.1*, we have to zip deploy our app (or Visual Studio Publish via GUI).

* First, we will build the app:

dotnet publish -c Release -o ./publish



Figure 37. Successful C# app build

* Then zip it:

cd publish

zip -r ../swe40006-csharp-calculator.zip .

A screenshot of a computer program

Description automatically generated

Figure 38. Zip the app

* And deploy with Azure CLI (from 3.2 root directory):

az webapp deploy \

--resource-group rg-swe40006 \

--name swe40006-csharp-calculator \

--src-path swe40006-csharp-calculator.zip \

--type zip



Figure 37. Successful C# app deploy

* We can see the logs from:

az webapp log tail \

--resource-group rg-swe40006 \

--name swe40006-csharp-calculator

**3.2.3 Inspect App from Public URL**

We can inspect the app from: <https://swe40006-csharp-calculator.azurewebsites.net>

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedFigure 38&39. App live demo inspection via public URL

**3.2.4 Deactivate App**

* We can stop the app running the following CLI command:

az webapp stop -g rg-swe40006 -n swe40006-csharp-calculator

* And activate the app again with CLI command:

az webapp start -g rg-swe40006 -n swe40006-csharp-calculator

* Or restart the app with CLI command:

az webapp restart -g rg-swe40006 -n swe40006-csharp-calculator

We can also do these from Azure website UI directly. Access Azure account → App Services → *swe40006-csharp-calculator*. From this page, we can check the app states and configurations, make changes as well as directly take action on the app stopping/starting, and restart.

A screenshot of a computer

Description automatically generated

Figure 40. App Services configurations

**3.3 – High Distinction: Install & PHP web app Azure deployment**

**3.3.1 PHP Installations**

We can install PHP on macOS with either CLI options:

* The easiest way is to install via Homebrew: brew install php
* Or official PHP Source Build (so we can control version specific, for instance *8.3.11*).

First we will install the package source:

curl -O https://www.php.net/distributions/php-8.3.11.tar.gz

tar -xzf php-8.3.11.tar.gz

Then we need to prepare the build with install path, enable, and ompile PHP from source into binaries, which then copied into */usr/local/php* to be used system-wide:

cd php-8.3.11

./configure --prefix=/usr/local/php --enable-mbstring --with-openssl --with-zlib

make

sudo make install

A screenshot of a computer program

Description automatically generated

Figure 41. PHP installation

However, upon manual inspection, I found out that I already have PHP version 8.3.12, so we will proceed with this version instead.

A close-up of a list of events

Description automatically generated

Figure 42. PHP version checking validation

**3.3.2 PHP Web App Development**

We create a minimal **index.php** app file, with basic calculator app operational logics, HTML structuring and CSS stylings**:**

* **PHP Logic (Top of File):**
  + A *calc()* function handles the four operations: addition, subtraction, multiplication, and division.
  + Division by zero is checked to prevent runtime errors.
  + When the form is submitted (POST request), user inputs (a, b, and op) are read and passed into *calc()*.

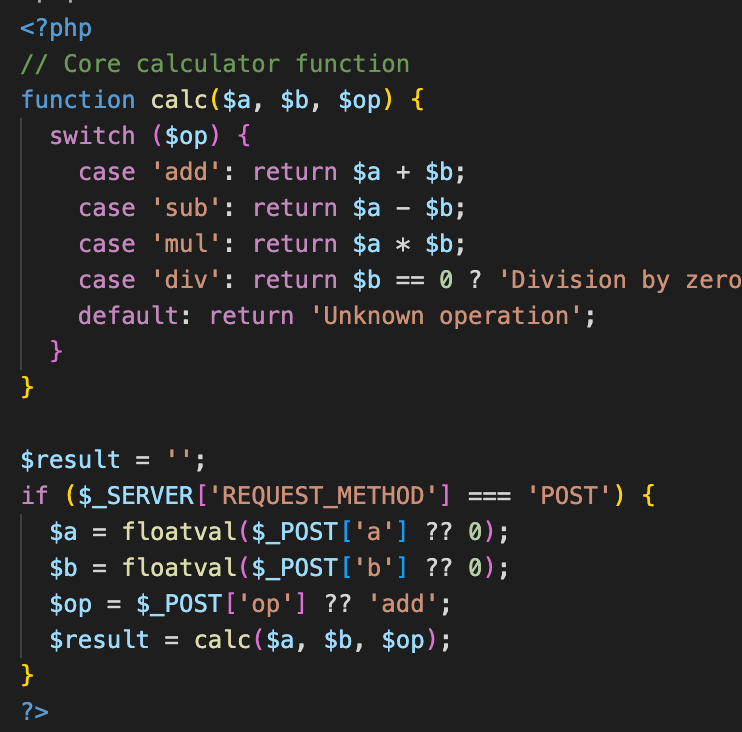


Figure 43. index.php logics

* **HTML Structure:**
  + The UI uses a centered *.calculator div* styled with CSS for neat card-like layout.
  + **Inputs:**
    - Two *<input type="number">* fields for numbers a and b.
    - A *<select>* dropdown for choosing the operation options (+, -, x, ÷).
  + A **submit button** triggers form submission.

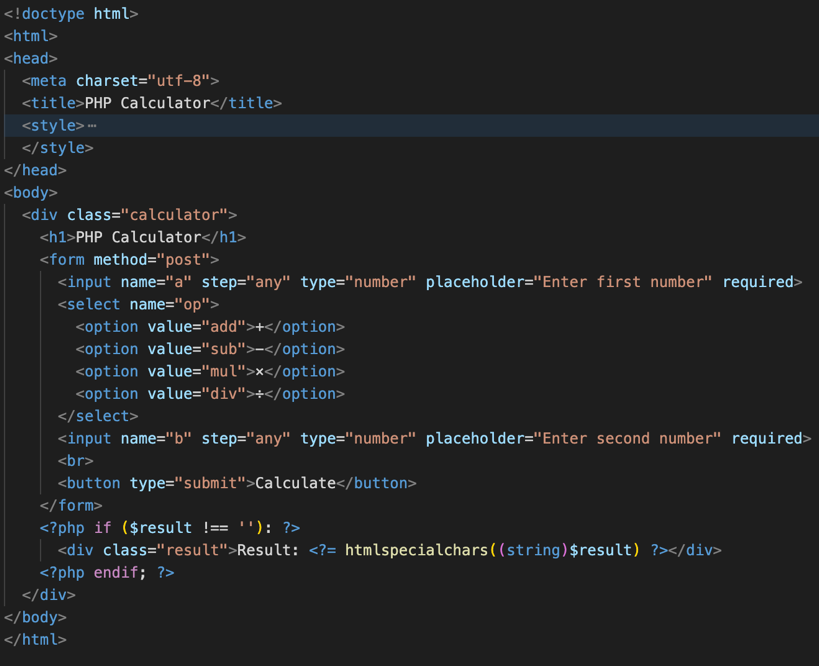


Figure 44. index.php HTML structures

* **Result Display:**
  + After the form is submitted, the result is computed and shown inside a styled *<div class="result">*.
* **Styling (CSS):**
  + The calculator is centered both vertically and horizontally.
  + Rounded corners, shadows, and hover effects on the button improve UX.
  + Result text is highlighted in green for visibility.

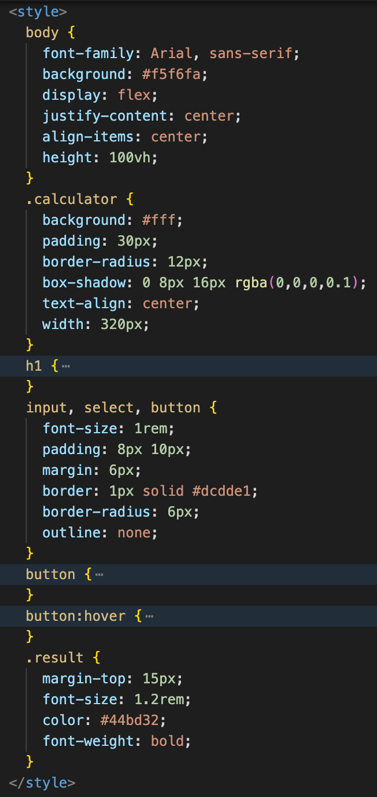


Figure 45. index.php CSS stylings

**Local Testing:**

We run the app following php app running command (specifying the port) locally.

php -S 127.0.0.1:8000 -t .

The app will then be accessible locally from <http://127.0.0.1:8000>, as shown in this demo:

A screenshot of a computer

Description automatically generated

Figure 46. PHP app demo

**3.3.3 Create Azure App Service (PHP 8.3) and Deployment**

We first create the web app container named *swe40006-php-calculator* sharing the same *rg-swe40006* resource group and *plan-swe40006-alt* resource plan with the app from *3.1*. Our runtime will be PHP8.3 as previously inspected in Figure 42.

az webapp create -g rg-swe40006 -p plan-swe40006-alt -n swe40006-php-calculator --runtime "PHP:8.3"

**A screenshot of a computer

Description automatically generated**

Figure 47. Successful web app container creation and details

Then we zip the app:

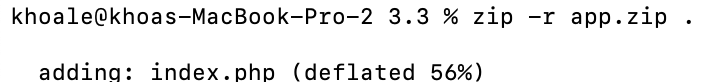


Figure 48. Zipping php webapp

And deploy with Zip Deploy:

az webapp deploy \

source config-zip \

--resource-group rg-swe40006 \

--name swe40006-php-calculator \

--src-path app.zip

A screenshot of a computer program

Description automatically generated

Figure 49. Successful app deployment and details

We can also check the log with CLI command:

az webapp log tail -g rg-swe40006 -n swe40006-php-calculator

And to restart the web app with CLI command:

az webapp restart -g rg-swe40006 -n swe40006-php-calculator

**3.3.4 Validate Live App**

The app can be accessible at public URL: <https://swe40006-php-calculator.azurewebsites.net>

A screenshot of a computer

Description automatically generated

Figure 50. App demo from live public URL

* We can stop/deactivate the app running the following CLI command:

az webapp stop -g rg-swe40006 -n swe40006-php-calculator

* And activate the app again with CLI command:

az webapp start -g rg-swe40006 -n swe40006-php-calculator

**Appendix**

All code resources are publicly shared via this Google Drive URL:

<https://drive.google.com/drive/folders/1zJqLMe6UWWzYt3agHPZJ5nsn-MdYfQpZ?usp=sharing>