Agile Processes

Group Project

**JDM\_BOYS**

Daniel - dodriscoll4mycit.ie - R00221504

Patryk - Patryk.Malinowski@mycit.ie - R00210173

Jamie - jamie.roche1@mycit.ie - R00151829

Mohammed - mohammed.al-hajri@mycit.ie - R00208964

A logo of a person running on a keyboard

Description automatically generated

Table of Contents

[Introduction 1](#_Toc152066728)

[Setup 1](#_Toc152066729)

[Configuration 3](#_Toc152066730)

[Experience 5](#_Toc152066731)

[Usefulness 5](#_Toc152066732)

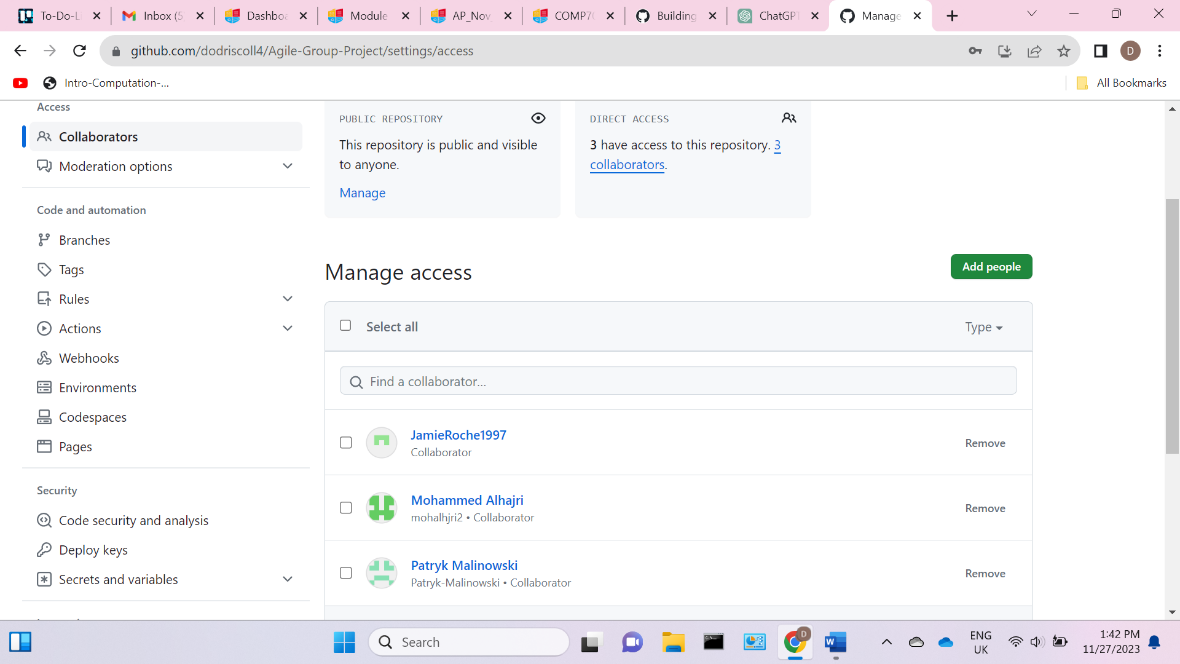
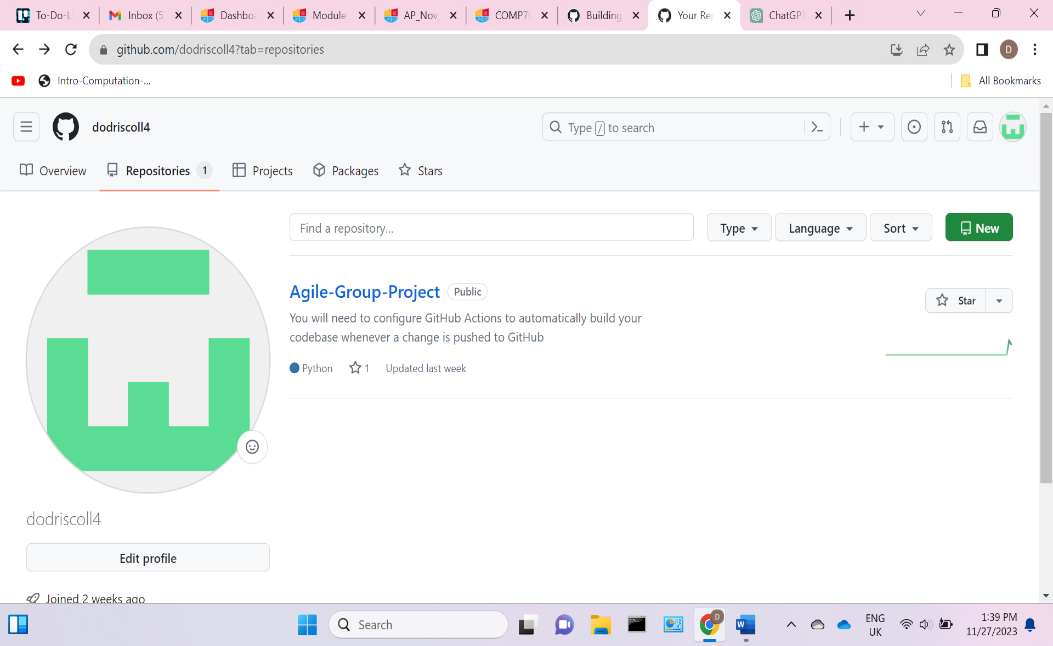
[Ease of Use 5](#_Toc152066733)

# Introduction

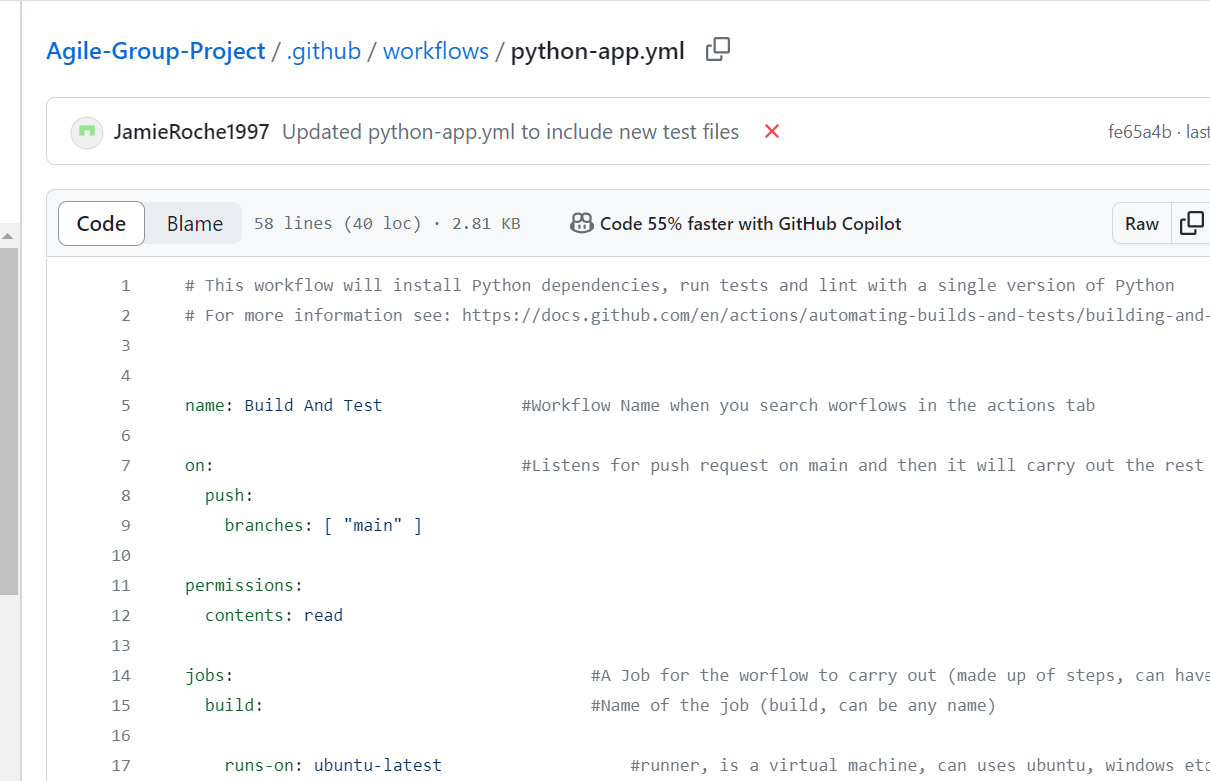
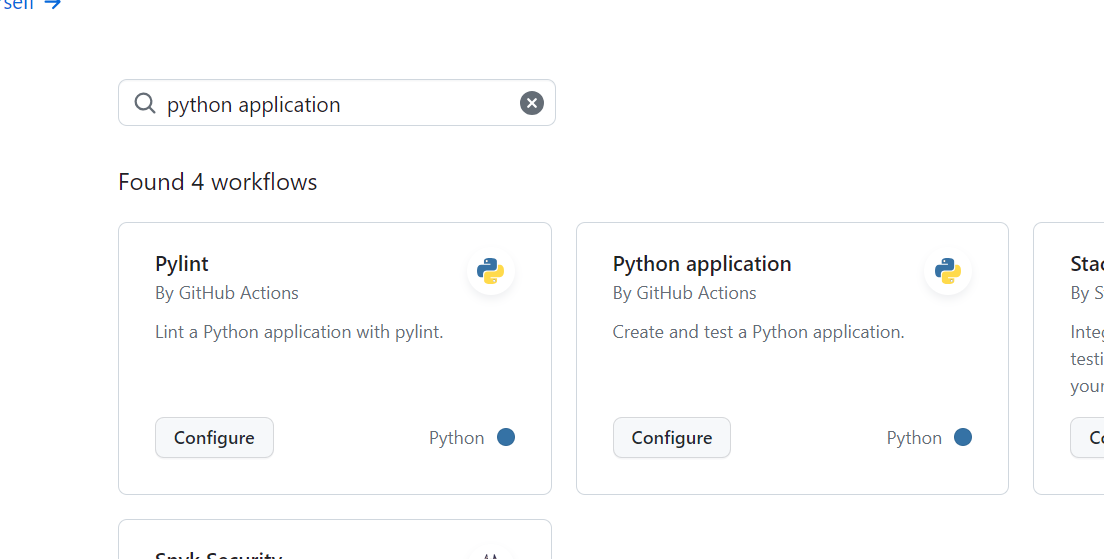
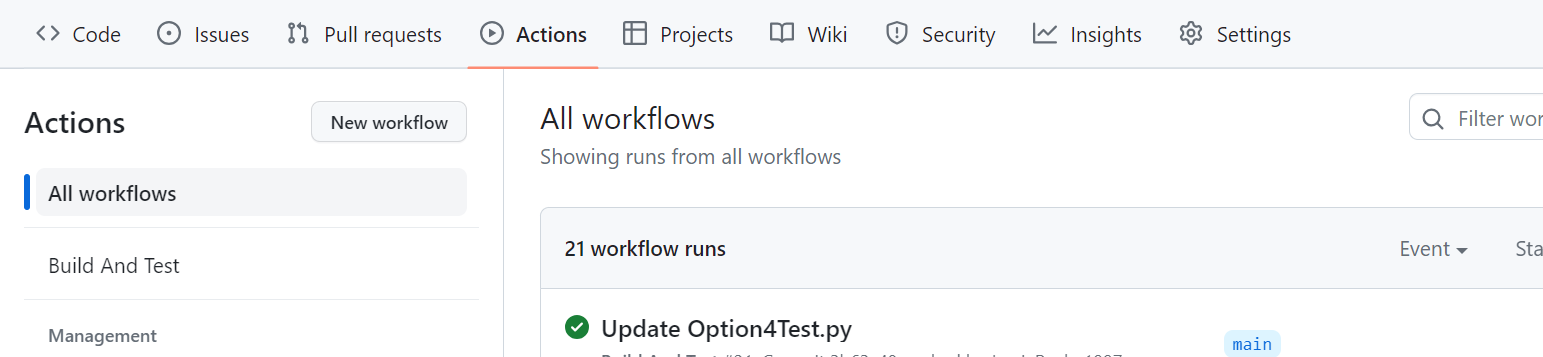
The goal of this group project was to investigate the usefulness of the GitHub Actions continuous integration facility to a group of software developers. For this project we created a continuous integration workflow specifically for a python project.

# Setup

Firstly, it was necessary to setup a shared repository on GitHub for our python project. This was simply done by creating a new repository, followed by settings, the collaborators tab and inviting other group members to the repository. Each member can then set up their own local repository on their local machine where they can make push and pull requests for the python project.



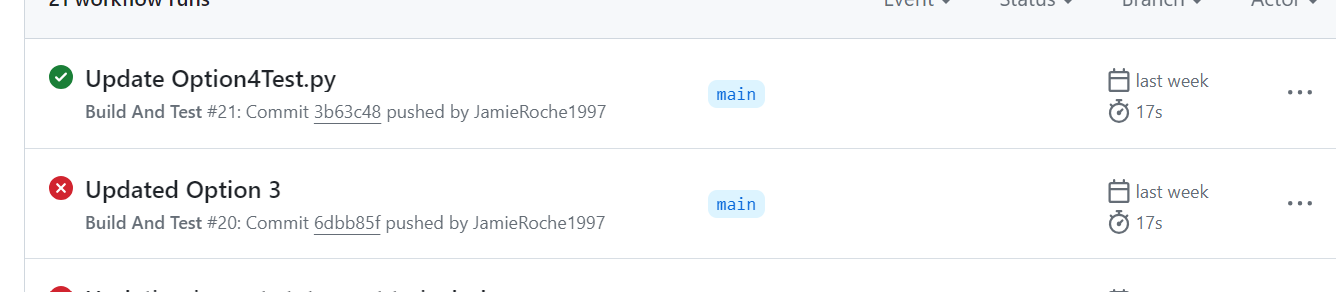
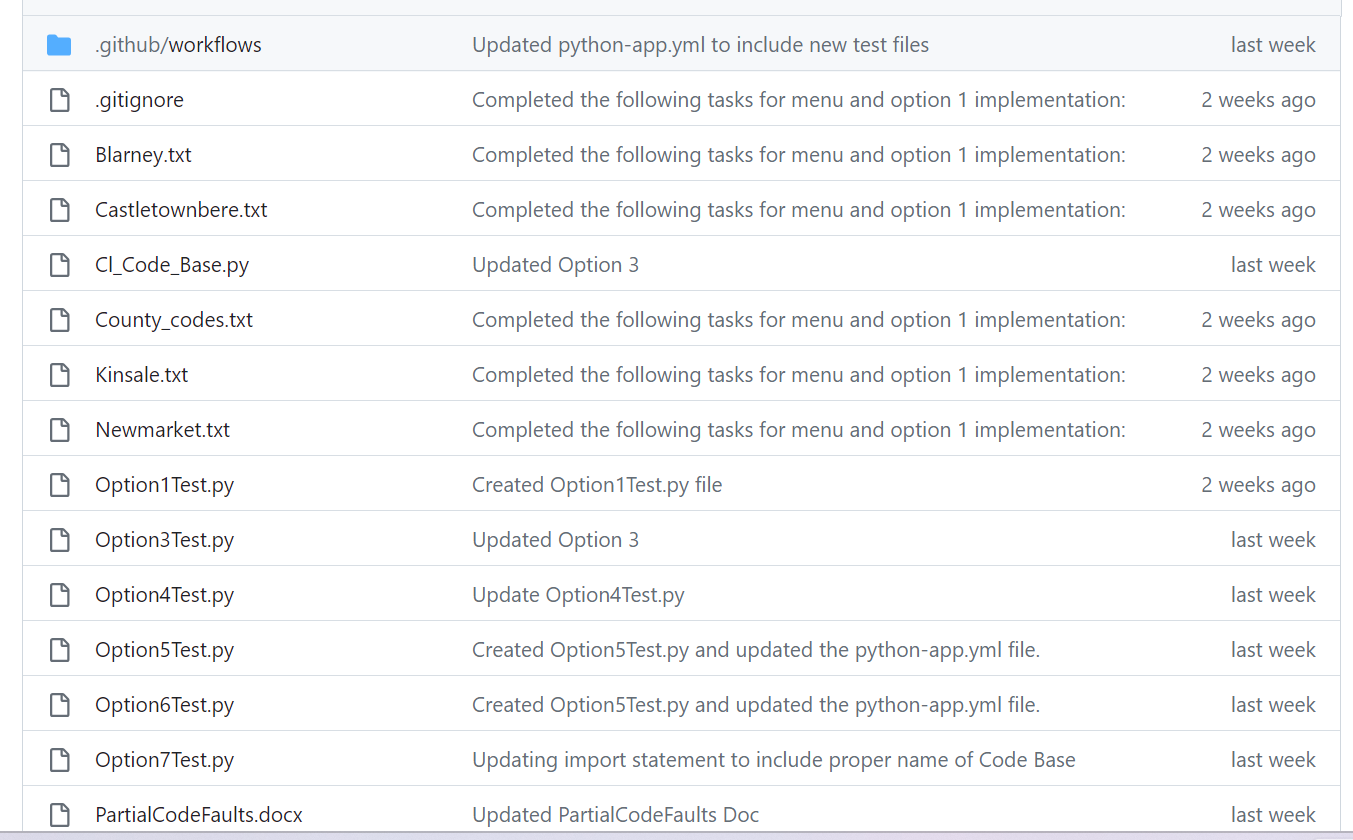
Next, we needed to create a GitHub Actions workflow for our repo. This was done by clicking actions, create new workflow and finding the python application workflow that is pre-built for GitHub Actions. This means that python dependencies are already included allowing us to build, run and test our python code through GitHub Actions. The workflow created will automatically run tests and linting operations with each push to the repo. We can then configure the workflow using the YAML language on the python-app.yml file.



Now with our shared repo setup on GitHub with our python workflow created, it was time to configure the process in which the workflow executes as well as uploading our python codebase, test files and text files.

# Configuration

* To begin our configuration, each group member had to download all necessary text files as well as the python codebase needed for the project. This allowed for each member to edit and change pieces of code related to their designated piece of work on the codebase file.
* Once a member was happy that their edited code works fine, it was pushed to GitHub where it was automatically built and tested.
* A test file for an edited function or code was created and tested locally before pushing. With both the code base file and test file working, the test file for that function was added to the GitHub workflow in the python-app.yml file.
* For each following push, the previous tests (eg. Option1Test.py) are ran on each python (codebase.py) build. GitHub Actions tells the user if a successful or unsuccessful push has been made as seen below.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Experience

# Usefulness

# Ease of Use

Following the initial configuration of GitHub Actions, it was necessary to establish the relevant Python tests and incorporate them into the .yml file. Whenever a test file was pushed to GitHub, any subsequent push triggered GitHub to automatically build and test the entire codebase, providing an indicator of whether any tests failed. This process essentially retained the familiar GitHub push workflow; the only deviation occurred when a test failed, requiring resolution of the issue before initiating another push.