**SW Quality Assurance & Testing: Continuous Web Integration and Deployment for BMI Application**

# Project Overview

Author: Cassie Javorsky

NetID: clj500

GitHub Username: clj500

GitHub Repository: [clj500/BMI-Calculator (github.com)](https://github.com/clj500/BMI-Calculator)

This document contains all planning, development, and testing details for a Body Mass Index calculator application and web interface. The initial BMI application was developed through Python and prompts users to enter their height and weight in inches and pounds; and then calculates and returns the user’s Body Mass Index number based on those measurements. After completion of the initial application, a web interface was developed through Django. The project documents the continuous integration and unit testing of the application and its corresponding web interface.

Table of Contents

[**SW Quality Assurance & Testing: Continuous Web Integration and Deployment for BMI Application** 1](#_Toc163417301)

[Project Overview 2](#_Toc163417302)

[Setup and Execution Instructions 4](#_Toc163417303)

[Downloading and Running the Python Code and Pytests 4](#_Toc163417304)

[Download Project Files 4](#_Toc163417305)

[Download Python 3.12 4](#_Toc163417306)

[Download Pytest 4](#_Toc163417307)

[Run the BMI Test File 4](#_Toc163417308)

[Run the BMI Program File 4](#_Toc163417309)

[Downloading Django and Running the Web Interface 4](#_Toc163417310)

[Setup Virtual Environment 4](#_Toc163417311)

[Activate Virtual Environment 4](#_Toc163417312)

[Download Django 4](#_Toc163417313)

[Run The Web Interface: 4](#_Toc163417314)

[Part 1: Test-Driven Development for BMI Application 4](#_Toc163417315)

[Functions & Respective Unit Tests 4](#_Toc163417316)

[Boundary Testing Techniques & Boundary Shift 6](#_Toc163417317)

[Part 2: Continuous Web Integration and Deployment for BMI Application 10](#_Toc163417318)

[Tool Descriptions 10](#_Toc163417319)

[Django: Python-Based Web Framework 10](#_Toc163417320)

[CircleCI: Continuous Integration and Development Program 11](#_Toc163417321)

[Selenium: End-to-End Testing 11](#_Toc163417322)

[Pytest: Automated Unit Testing 11](#_Toc163417323)

[Google : Hosting 11](#_Toc163417324)

[Coveralls: Code Coverage Reporting 11](#_Toc163417325)

[Cloud Platform Usage 12](#_Toc163417326)

[Code Coverage Report 13](#_Toc163417327)

[References 13](#_Toc163417328)

# Setup and Execution Instructions

## Downloading and Running the Python Code and Pytests

### Download Project Files

All project documents and python files can be found in the GitHub repository located here:

[clj500/BMI-Calculator (github.com)](https://github.com/clj500/BMI-Calculator)

These files should all be saved to the same location on your computer.

### Download Python 3.12

To run the BMI program, the user should first download Python 3.12 for Windows 10 from the following page: [Download Python | Python.org](https://www.python.org/downloads/)

The following site provides more in-depth instruction for downloading Python 3.12 and offers multiple download options for Windows and Mac:

[Python 3 Installation & Setup Guide – Real Python](https://realpython.com/installing-python/)

### Download Pytest

To install Pytest for Windows, open the command prompt and type the following command:

pip install pytest

To check that Pytest was successfully downloaded type:

pytest --version

This blog contains more help if this process does not work for you:

[How to Install pytest in Python? – Be on the Right Side of Change (finxter.com)](https://blog.finxter.com/how-to-install-pytest-in-python/#:~:text=How%20to%20Install%20pytest%20on%20Windows%3F%201%20Type,Wait%20for%20the%20installation%20to%20terminate%20successfully.%20)

### Run the BMI Test File

To run the test file (titled “**test\_BMI.py**”) open your command prompt and change directories (cd) into the same location as where you saved the python files. Make sure there are no other python test files in this location and then type “pytest” into the command line. This should run the test file.

A screenshot of a computer program

Description automatically generated

### Run the BMI Program File

To run the program file (titled “**bmiProgram.py**”), right-click on the file and select “Edit with…” and choose IDLE 3.12. Once you have opened the file in IDLE click the run tab and run the program.

A screenshot of a computer

Description automatically generated

## Downloading Django and Running the Web Interface

### Setup Virtual Environment

Before installing Django, a virtual environment should be set up. Open the command prompt and cd into the directory where you would like to install Django (the directory which will contain the website). Then type the following command:

**python -m venv virtualenv**

This command will not show any output in the command line but should create a new folder titled “virtualenv” within the folder you are working in.

### Activate Virtual Environment

Make sure you are in the same folder as your virtual environment, and run the command:

**virtualenv\Scripts\activate**

This should activate the virtual environment and should show the name of the virtual environment in front of the directory as shown below:

A computer screen with white text

Description automatically generated

### Download Django

To download an official version of Django, cd into the directory where you would like to install Django and activate the virtual environment (this should already be done if you just completed the Activate Virtual Environment step). Once you are in the correct directory run the following command:

**python -m pip install Django**

### Run The Web Interface:

Save the folder titled “bmi\_calculator” from GitHub in the same folder you downloaded the virtual environment and Django. Open the command prompt and change directories into the folder. Activate the virtual environment and then cd into the “bmi\_calculator” folder. Then run the following command:

**python manage.py runserver**

This should result in the following output:

A screenshot of a computer program

Description automatically generated

The website can then be found at: **http://localhost:8000**

# Part 1: Test-Driven Development for BMI Application

## Functions & Respective Automated Unit Tests

Function 1: Input User Height

Function prints a statement: “Enter height (in inches):” which prompts the user to enter their height in inches. This value is stored as a float called height for later conversion and use in the BMI equation.

Function 2: Input User Weight

Function prints a statement: “Enter weight (in lbs):” which prompts the user to enter their weight in pounds. This value is stored as a float called weight for later conversion and use in the BMI equation.

Function 3: Convert Height

Function converts any height from inches to centimeters to plug into the BMI equation. The conversion happens by multiplying the height in inches by 0.025 and then squaring that answer. The test (ad-hoc) for this function checks that the conversion outputs the correct corresponding value:

A screenshot of a math problem

Description automatically generated

Function 4: Convert Weight

Function converts any weight from pounds to kilograms to plug into the BMI equation. The conversion happens by multiplying the weight in pounds by 0.45. The test (ad-hoc) for this function checks that the conversion outputs the correct corresponding value:

A white background with black text

Description automatically generated

Function 5: Calculate BMI

Function takes height and weight in the parameters and divides the weight by the height to calculate a corresponding BMI value. The test (ad-hoc) for this function checks that the calculation outputs the right result.

A white background with black numbers and red text

Description automatically generated

Function 6: Identify BMI Category

A screenshot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generatedFunction takes BMI in the parameter and uses an if-else condition statement to determine which of four categories the BMI value falls under. There are four separate tests for this function: one for each category. Each test verifies that the boundaries for each category are established correctly in the if-else condition statement through Weak N x 1 boundary testing strategies.

## Boundary Testing Techniques & Boundary Shift

This project utilized a Weak N x 1 boundary testing technique. A Weak N x 1 boundary test is ideal for identifying the maximum number of boundary issues, and can easily catch boundary shift problems.

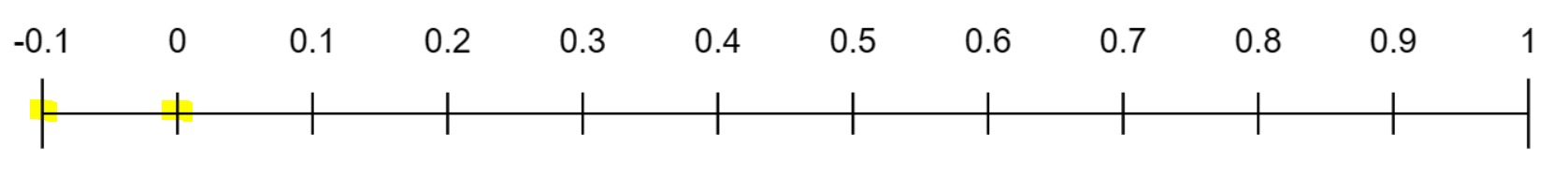
The categories for BMI were provided in the following format:

|  |  |
| --- | --- |
| **BMI** | **Category** |
| <18.5 | Underweight |
| 18.5-24.9 | Normal Weight |
| 25-29.9 | Overweight |
| >=30 | Obese |

This information was interpreted and translated to the following format:

|  |  |
| --- | --- |
| **BMI** | **Category** |
| [0, 18.5) | Underweight |
| [18.5, 25) | Normal Weight |
| [25, 30) | Overweight |
| [30, ∞] | Obese |

To test the Underweight boundary, testing took place at the following points:



A black line with yellow circle

Description automatically generated

-0.1 = OFF

0 = ON

18.4 = interior

18.5 = ON

18.6 = OFF

To test the Normal Weight boundary, testing took place at the following points:

A black line with a yellow circle

Description automatically generated

A black line with a yellow dot

Description automatically generated

18.4 = OFF

18.5 = ON

18.6 = interior

24.9 = OFF

25 = ON

To test the Overweight boundary, testing took place at the following points:

A number on a white background

Description automatically generated

A black and white image of a number

Description automatically generated

24.9 = OFF

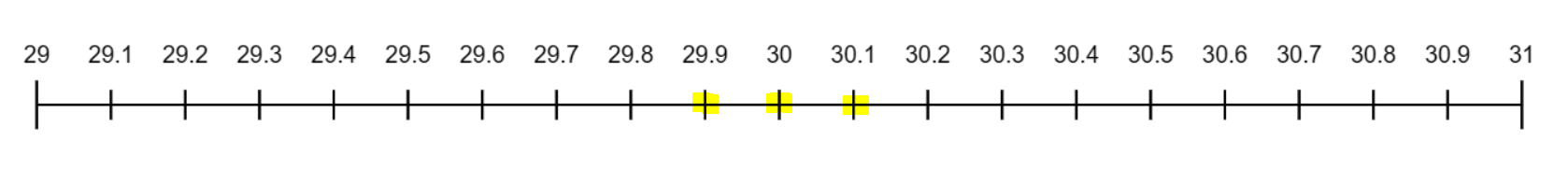
25 = ON

25.5 = interior

29.9 = OFF

30 = ON

To test the Obese boundary, testing took place at the following points:



29.9 = OFF

30 = ON

30.1 = OFF/interior

A screenshot of a computer program

Description automatically generated

After applying a boundary shift of 0.1 to the lower boundary of the Normal Weight category:

A screenshot of a computer code

Description automatically generated

A computer screen shot of a program

Description automatically generated

Two tests failed because their value was now completely excluded from any boundaries. They caught this because they tested exactly on the boundary.

# Part 2: Continuous Web Integration and Deployment for BMI Application

## Tool Descriptions

### Django: Python-Based Web Framework

The project utilizes Django, a python-based web framework, to implement the web interface for the BMI application. Since the original script for the BMI program was written in Python, Django was the ideal choice for web development as it allows seamless integration of Python code into a website. Without Django, this process would require more time and external tools and integrations.

The link to Django can be found here: <https://www.djangoproject.com/>

### CircleCI: Continuous Integration and Development Program

To allow for continuous integration throughout the project, CircleCI is utilized by developers to monitor, maintain, and communicate about updates to the project, including coding and deployment. CircleCI is free and includes integration with GitHub which makes the process easier.

The link to CircleCI can be found here: <https://circleci.com/>

### Pytest: Automated Unit Testing

Pytest is a free python-based framework for building automated functional unit tests. Since the main testing focus for this project is on unit testing, Pytest provides the necessary framework. Additionally, Pytest easily integrates into Django and GitHub. Automatic unit tests were written using Pytest for each function and integrated with Django by using shared includes and imports.

The link to Pytest can be found here: <https://docs.pytest.org/en/8.0.x/>

### Google Cloud: Hosting

For cloud hosting, Google Cloud offers affordable and intuitive services. This project plans to use Google Cloud to host its web server due to the cost-effectiveness, ease of use, and high level of control and security. A main feature of Google Cloud is its ability to handle big data; although this project is small now, this could be beneficial if the application is expanded. Google Cloud also integrates well with Django and its website includes informative tutorials for the process.

The link to Google Cloud can be found here: <https://cloud.google.com/>

### Coveralls: Code Coverage Reporting

To ensure that the project test cases cover every scenario, the developers utilize a code coverage reporting tool called Coveralls. Coveralls is free and integrates with GitHub making it an ideal choice for coverage reporting. Pytest does include a coverage plug-in but Coveralls provides a more clear UI and system for performing reporting.

The link to Coveralls can be found here: <https://coveralls.io/>

## Deployment Pipeline

### Step 1: Source Control

The first step for the deployment pipeline is to establish a form of source control so that all members of the team can track their changes to the project and view others’ changes, as well as previous versions of the project. For this project, source control will be managed with the use of GitHub and Visual Studio Code. All project changes will be shared through a GitHub repository: when a team member makes a change to the code or adds new files they will upload these files to the repository. GitHub Desktop makes this easy by allowing users to make a local clone of the repository, edit it on their own device, and merge the changes into the repository. As the development team and the site size grow, it may be important to implement separate branches for different team members so that changes can be reviewed before being merged. This will also help prevent overlap of changes. GitHub is an ideal tool for source control as it is free, widely used, and offers many different tool integrations.

### Step 2: Develop the Web Interface

The next step of the deployment pipeline is to begin extending the application for web access. The project will be developed using Django as its framework and Visual Studio Code to write and edit code. Members of the development and testing teams should download and familiarize themselves with both Django and Visual Studio Code. Since the BMI application is simple, the website will only need to consist of two pages for now: the home page, which will prompt the user to enter their measurements, and the results page which will display the user’s BMI and category. Most of the novel code is expected to be code required to set up the environmental requirements and connections for Django, the server, and the pre-existing Python code. Once the connections are complete, the Python code can be easily translated to Django (keeping most pre-existing lines of code intact). As the web server expands, more integrations may be required to host various forms of content.

### Step 3: Continuous Integration

In order to prevent from disrupting the site when changes are being made or bugs are being fixed, the team will need to integrate CircleCI to manage continuous integration. CircleCI can be integrated with GitHub easily and can sometimes provide a pre-made configuration file based on the repository contents. However, in the case that CircleCI is unable to provide a relevant configuration file, the team may have to dedicate some time to researching how to build an appropriate configuration file (config.yml) for the usage of Django’s framework and Python. Once complete, this will allow CircleCI to screen each contribution. The use of continuous integration also helps the team stay on track with the deployment pipeline.

### Step 4: Static Analysis

Currently there is no plan for static analysis in this project. Although this is feasible for now due to the project’s small scale, a plan for static analysis will need to be created in the case that the web server hopes to incorporate larger, more complex features.

### Step 5: Automated Unit Tests

Once the web framework has been created and a plan for continuous integration and source control has been implemented, the team will move to the next step of the deployment pipeline: unit testing. Unit testing checks individual functionalities of the project for any errors. All testing team members will utilize Pytest to write unit tests for each functionality. These functionalities and unit tests have already been defined in the previous section “Functions & Respective Automated Unit Tests” located under Part 1 of this document. Therefore, the tests will only need to be adapted to suit the Django framework.

## Code Coverage Report

# References

Website Development with Django: [Learn Django by Building a Calculator App (freecodecamp.org)](https://www.freecodecamp.org/news/how-to-create-a-calculator-app-in-django/)

Setting up End-to-End testing and Automated Unit Testing with Pytest and Selenium: [Pytest Django Tutorial: Testing Your Django Apps With Selenium Pytest | LambdaTest](https://www.lambdatest.com/blog/pytest-django-tutorial/)

Configuring a Shared Lamba Test Tunnel: [Localhost Testing With Shared Tunnel | LambdaTest](https://www.lambdatest.com/support/docs/sharing-lambda-tunnel/)