

Testing: Route Tests & Working with a Database



BEW 1.2

- Learning Outcomes
- Warm-Up
- Writing Route Tests with unittest
- **BREAK**
- Lab Time

By the end of today, you should be able to...

1. **Use** the unittest library to write route tests for an existing project.
2. **Write** route tests for routes that interact with a database to check for certain operations.

Warm-Up

As we approach the end of the term, it can be helpful to reflect on what has worked well for you so far and what your challenges have been.

In a group of 3, discuss the following questions:

1. What is one thing you have struggled with the most in a remote learning environment?
2. What is one study habit that has helped you to succeed in a remote environment that you could share with others?

Writing Route Tests with unittest

Take a look at the existing route tests in [Homework 5](#).

The files `main/tests.py` and `auth/tests.py` contain several route tests that are partially written, as well as some example ones.


In these tests, before we can verify the contents of the page, we need to insert data into the database.

Essentially, we need to *set up the scenario* using a "fake" (in-memory) database to do so.

Before we run each test, we want to run a `setUp()` method that will do a lot of the required prep work.

```
def setUp(self):  
    """Executed prior to each test."""  
    app.config['TESTING'] = True  
    app.config['WTF_CSRF_ENABLED'] = False  
    app.config['DEBUG'] = False  
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///memory:'  
    self.app = app.test_client()  
    db.drop_all()  
    db.create_all()
```

We'll be using an in-memory database so that we can always start with a fresh copy and not "pollute" the existing data.



It can also be useful to write some helper functions to complete the database operations, since you'll likely be re-using that behavior in multiple tests.

For example, here's a helper function that creates a user:

```
def create_user():  
    # Creates a user with username 'me1' and password of 'password'  
    password_hash = bcrypt.generate_password_hash('password').decode('utf-8')  
    user = User(username='me1', password=password_hash)  
    db.session.add(user)  
    db.session.commit()
```

Watch as your instructor walks through how to create route tests for GET & POST routes.

Break - 10 min

Lab Time

With a partner, write each test required in [Homework 5](#), pair programming style. Switch driver & navigator after each test!

The person wearing the lighter colored shirt will be the driver first.

Warm-Up

Warm-Up (15 minutes)

Take 10 minutes to read [this article](#) on testing pipelines. Then with a group of 3, answer the following questions:

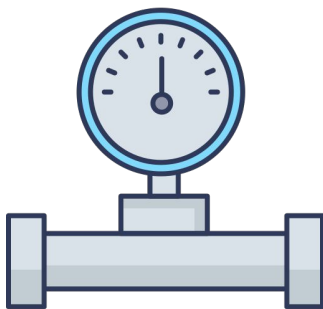
- Which part of the testing pipeline described do you think is **most important**? Why?
- Which part do you think is **most interesting**? Why?
- What is one thing in the article that is confusing, or that you'd like to learn more about?

Create a Test Pipeline with Git Hooks

What is a test pipeline?

A **test pipeline** is an automated process that finds and reports any errors in your code before it gets to the end users.

Test pipelines can be large or small, and make it safer to make changes to your code.



abort changes



commit changes

What is a commit hook?

A **commit hook** is a script that runs whenever you commit your code using **git commit**. It is usually used to:

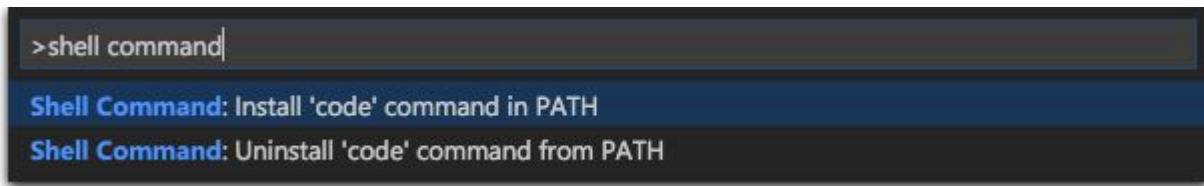
- Run a linter to check your code style
- Run unit tests
- Etc...

If the commit hook fails, you can't commit your code and need to fix any errors before trying again.

First, let's change your default editor to Visual Studio Code (unless you are already familiar with Vim):

```
$ git config --global core.editor "code --wait"
```

If you get a "command not found" error, go to VSCode, press **Cmd+Shift+P**, type "shell command", then select "Install code command in PATH".



Let's create a commit hook!

Watch your instructor complete the following steps before you give it a try.

First, choose a Git repository that you'd like to add a commit hook to. We'll be using the **Homework 5** repository as it already has unit tests.

All of the git hooks in a repository live in the **.git/hooks** folder. There are already some samples there, but we'll create our own.

Let's create a commit hook!

Create a pre-commit hook file in the project's root folder called **pre-commit.sh** with the following contents:

```
# Run unit tests
python3 -m unittest discover || exit 1
```

Then, make it an executable and copy it into the .git/hooks folder:

```
$ chmod +x pre-commit.sh
$ cp pre-commit.sh .git/hooks/pre-commit
```

Let's create a commit hook!

You can test your git hook by attempting to make a commit with no message:

```
$ git commit
```

Try making a test fail, then commit again! Your commit should abort.

If you ever want to make a commit **without** running the commit hook, you can use:

```
$ git commit --no-verify -m "your message"
```

Activity (10 minutes)

In a group of 3, make sure everyone is able to complete the steps. You should be able to make a commit and see your tests run!

Wrap-Up