Phase 3

Web Application Development

Scenario

Introduction and Deliverables

In this phase, you will build a cloud native, AI-enabled web application. You will deploy the models you developed in Phase 2 using simple Flask applications for online predictions, build a Django application to display your results, and instrument all of these using open source tools. This pattern of separate, containerized applications gives you and your team development flexibility and allows you to update and scale components individually.

Week	**Task**	**Deliverable**
9	Create a Flask application (one per team member) to respond to HTTP requests with prediction results	3 Flask apps
10	Create a Django web application	Django App
11	Containerize your applications	5 Container Images
12	Instrument your applications using Prometheus and visualize with Grafana	Grafana Dashboard
13	Deploy your web application in LTAC	Deployed Application
14	Web user interface development	Landing Page
15	Flex week: Catch up, or add a new feature to Django application	Django Application
16	Flex week: Catch up, or add a React.js component to your web page	React Component

Table 3.1: Deliverables in Phase III. Progress review weeks are highlighted.

Setting Up Your Development Environment

Python, Git, and Directories

Before you start creating your application, there a few things that will make it easier and faster for you to develop as an individual and member of a team.

1. Directories: Computers and spaces don't go well together. Many of the steps that we will guide you through in the upcoming weeks use the terminal (Mac), shell (Linux), or PowerShell (Windows). If you have used spaces in the directory path it will be much harder to navigate your file system in the terminal and some development tools will be more challenging to configure. With that in mind, a few recommendations to make it easier for you to work in the terminal. If you prefer Windows but struggle with PowerShell you can alternatively consider Git Bash to enjoy many of the features of working in a Mac or Linux environment on a Windows OS.

   i. lowercase everything. usernames, filenames, directory names are all easier to type if you avoid capital letters. (There are exceptions to this, and even some languages where it isn't the standard practice, but you should use those by exception only)

   ii. single word. Name your directories and files with a single word. Generally, try to pick ones that differ in the first few characters. It will make it faster for you to type and you can use tab completion to avoid mistakes and speed up your navigation.

   iii. snake case. If you must use multiple words in a file or directory name, separate them with underscores, these interrupt your ability to read less than hyphens. Never use spaces.

2. Editors: We recommend you use Visual Studio Code (VSCode) as your code editor. There are many editors available and the choice of which you want to use is highly preference based. Until you are experienced in the language you are working in, we **do not** recommend you use a fully featured Integrated Development Environment (IDE) like PyCharm. A more simple text editor is better for learning the fundamentals. VSCode can grow with you as you can add extensions that speed up your development but at its core, it is just a text editor.

3. Python: There are many versions of Python available and currently supported. For most of what you do, the version will not matter so you should pick the one that is available in your **deployment** environment. If it isn't specified, generally use the latest stable release. If you are using Mac or Linux, use a package manager like Homebrew, apt, or yum.

4. Virtual Environments: Virtual environments are a critical tool for Python developers. The language comes with a builtin virtual environment creation tool – venv – we recommend you use for all your Python work. In week 9 we will provide you with the basic commands to get started in a virtual environment, but for now a few first principles.

   i. environment name. Name your virtual environment the same thing in every project, '.venv' is the standard naming convention.

   ii. create once, initialize always. Only create your virtual environment when you start a new project, initialize it everytime you work on it. Some editors (like VSCode) will do this for you automatically if properly configured.

5. Source Control: To be an effective member of a team that builds software (and realize all machine learning resides in software) you must be familiar with Git. There are books written on the topic and we will not replicate that here. We highly recommend you read, at minimum, Chapters 1 & 2 in Pro Git. We will provide you with commands to guide you in working with this tool over the coming weeks, but you will need to expand your knowledge of this tool to be an effective contributor to your team.

Getting Started

In Phase 3, your team will work together on a shared Git repository to complete each week's assignments. Only one of you needs to follow steps 1-9 below to create your team's git repository. The remaining team members should complete steps 10-13. (Note: All team members will need to execute step 5 upon cloning the repository, and before they begin working on this phase of the project)

1. Create a new directory on your computer named **capstone**. We recommend putting this in an easy to access location on your computer (i.e. `/Users/<username>/Documents/computing/capstone`)

   `mkdir capstone && cd capstone`

2. In your newly created directory, initialize a git repository. If you have not already, you will need to install git.

   `git init --initial-branch=dev`

   Here we specify dev as the initial or default branch. We use dev to align the terminology between deployments/environments and the branch names in our source code repositories.

3. Open this directory in Visual Studio Code.

4. Create a file in this directory named `README.md`. In that file, add the following pieces of information and save the file.

   i. A Team Name
   ii. Your name

5. Create a Python Virtual Environment. In the integrated VSCode terminal window run:

   `python -m venv .venv`

   Once created, you can **activate** this virtual environment as follows:

   (Linux/Mac) `source .venv/bin/activate`

   (Windows) `.venv\Scripts\activate`

6. Create a new file in your directory named `.gitignore`. In this file, add the following line:

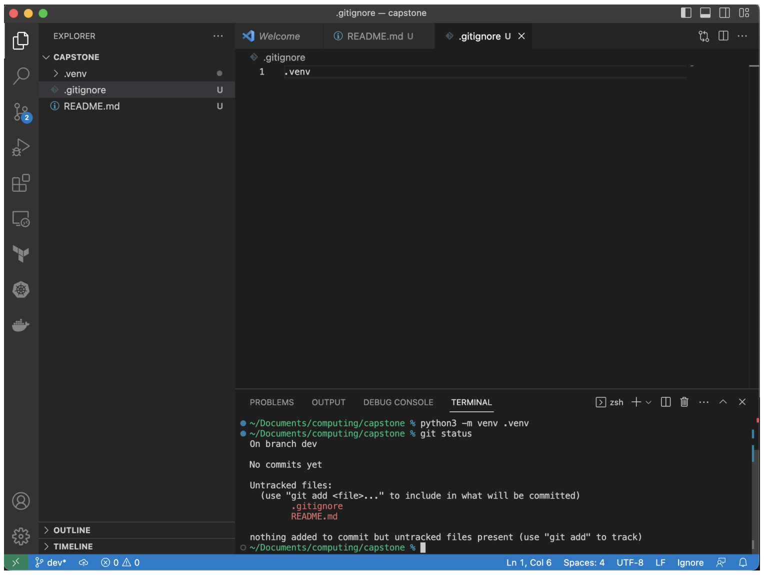
   ```
   .venv
   ```

   This tells git to recursively ignore any files in the .venv directory.

7. Commit these files to your git repository. Before doing so, check the status of changes in your working directory.

   `git status`

   You should see something like the following in your terminal. If not, revisit the previous steps.



```
~/Documents/computing/capstone % python3 -m venv .venv
~/Documents/computing/capstone % git status
On branch dev

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        README.md

nothing added to commit but untracked files present (use "git add" to track)
~/Documents/computing/capstone %
```

Figure 3.1: git status output after creating new files

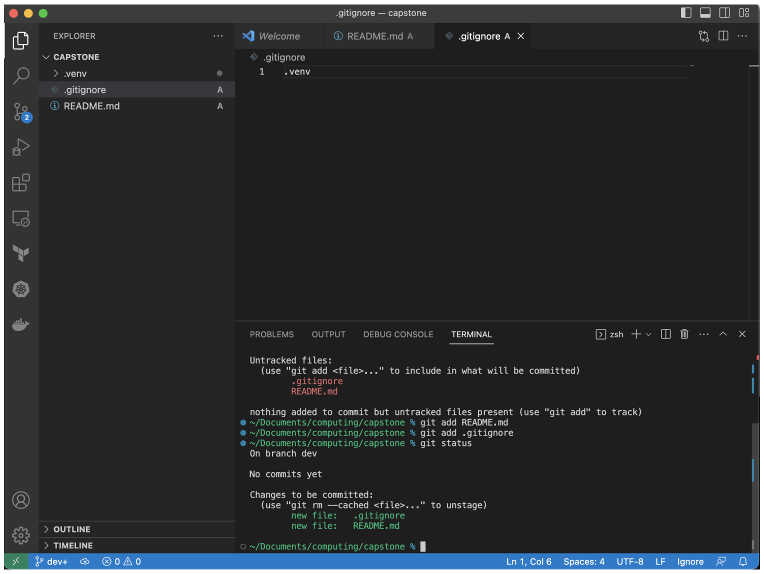
   If your output looked like the screenshot above, run the following to "stage" your changes.

`git add README.md`

`git add .gitignore`

Now that you have added the file to your 'staged' changes, when you run `git status` again you will see a different outcome, verify that now:

`git status`



```
.gitignore
.venv
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        README.md

nothing added to commit but untracked files present (use "git add" to track)
~/Documents/computing/capstone % git add README.md
~/Documents/computing/capstone % git add .gitignore
~/Documents/computing/capstone % git status
On branch dev

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   README.md

~/Documents/computing/capstone % 
```

Figure 3.2: git status output after staging your work

If your output looked like the image above, commit your changes.

`git commit`

Note: if you have not used git before, this command will fail and you will need to configure your settings as follows before you re-run the above command.

`git config --global user.name <Your GitHub username here>`

`git config --global user.email <Your GitHub email here>`

Note: A commit message or "gist" should follow the following format. First, there should be a short, single line summary which will serve as the commit title. Then, after an empty line, a sequence of short descriptions should describe every change (there should only be a small number included in each commit) included in the commit. Example below.

```
Initialize Project

Created our team's initial README.md
```

As you can see, this commit message is succinct, but clearly describes what was changed in the repository in this commit. Beyond serving as an esoteric reference of past work, this makes reviewing different versions of work and tracking down future bugs faster and more reliable on your team.

8. Create a remote repository. While you would typically do this yourself (and select the repository location best suited for your team's access and development needs) we have created a remote repository for you in the AFC-AI2C GitHub. Insert the link below where specified in the below command. This command specifies a remote repository that your newly created git repository will 'track' at the specified URL.

   `git remote add origin <remote repo link here>`

   i. `https://github.com/AFC-AI2C/capstone-team-1.git` (CW2 Gonzalez, SPC Lacovara, SPC Fortuna)
   ii. `https://github.com/AFC-AI2C/capstone-team-2.git` (CW2 Vickers, SFC Schulze, SPC Morales)
   iii. `https://github.com/AFC-AI2C/capstone-team-3.git` (SFC Thames, SSG Malagon, SPC Tarila)
   iv. `https://github.com/AFC-AI2C/capstone-team-4.git` (1LT Cheng, SSG Nguyen, SSG Mangual)
   v. `https://github.com/AFC-AI2C/capstone-team-5.git` (1LT Owens, SFC Thompson, SSG Kuewa)
   vi. `https://github.com/AFC-AI2C/capstone-team-6.git` (SFC Belyayev, SSG Shields, SSG Prickett)

   vii. `https://github.com/AFC-AI2C/capstone-team-7.git` (1LT Robinson, SSG Ballew)

9. Push to the remote repository. This command pushes your changes to the specified remote repository, setting the local branch called `dev` to "track" the remote branch with the same name in the remote repository named `origin`.

   ```
   git push --set-upstream origin dev
   ```

10. Now that you have a repository setup, each additional team member needs to setup the directory by "cloning" this repository as follows.

    In a shell, command prompt, or terminal, navigate to the directory you want to store Phase 3 work in (suggested: `/Users/<username>/Documents/computing`) and run the command below. You can find the repo link in your team's link above under the green "Clone" button.

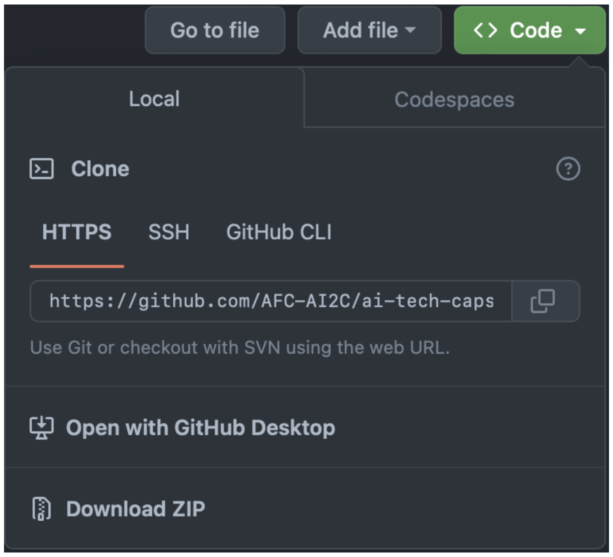


Figure 3.3: GitHub repository clone button

    ```
    git clone <remote repo link here> capstone
    ```

    This will create a directory called "capstone" that will store the contents of your team's remote repository. Open this directory in VSCode.

11. Create a new branch. Open the integrated terminal in VSCode and run:

    ```
    git checkout -b add-<your name>
    ```

    You have now created a place you can make changes to your team's source code safely and in parallel with your other teammates.

12. Open the README.md file and add your name to the list of names. Save the file.

13. Run the following sequence of commands to stage, commit, and push your work.

    i. `git status`
    ii. `git add README.md`
    iii. `git commit`
    iv. `git push -u origin add-<your name>`

14. Open your team's repository in GitHub and create a pull request from your branch to dev.

15. After another team member reviews your work, merge your work into dev. Once you, or anyone else on your team starts to work on the project again, start first by running `git pull` to fetch the latest changes to your base branch, dev, before you create and checkout a new branch.

Best Practices for working with Git

1. Commit and push your changes whenever you either a: finish a major change/feature or b: finish working for the day.
2. Have a team mate to review your work; another set of eyes and a different perspective are always helpful.
3. Be explicit in actions. Only add files you are sure you want to add. Use `git status` to check yourself and your work before you commit.

Git Command Sequence Reference

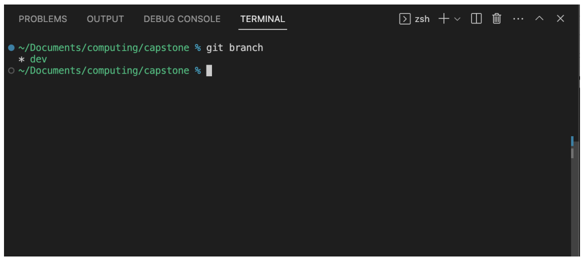
The following references all assume you are working in the terminal and have navigated to the base directory for this project `capstone`.

Create and Checkout a New Branch

Over the next two months of capstone, we will start each task telling you to ”Create and Checkout a New Branch”. We summarize that process below for your reference. This assumes you have cloned your team’s git repository and that you have navigated to your ’capstone’ directory before you run any of the commands.

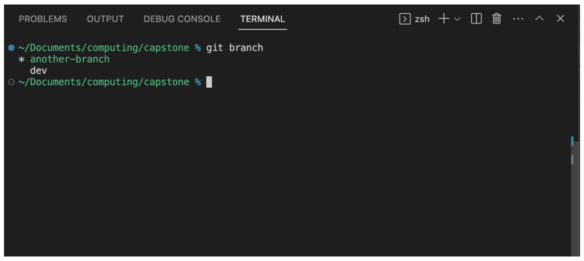
i. Verify you are on the branch, ’dev’.

   `git branch`



```
~/Documents/computing/capstone % git branch
* dev
~/Documents/computing/capstone %
```

Figure 3.4: good



```
~/Documents/computing/capstone % git branch
* another-branch
  dev
~/Documents/computing/capstone %
```

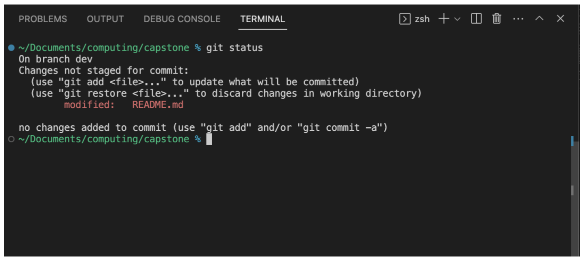
Figure 3.5: bad

   If you were not on the branch dev, checkout dev now.

   `git checkout dev`

ii. Verify you do not have any unsaved work.

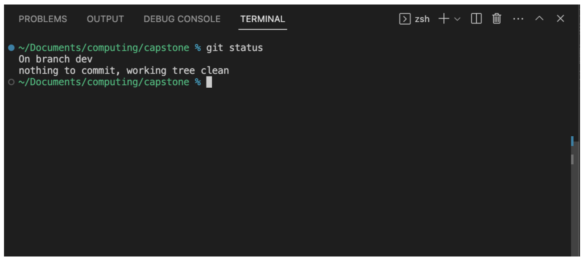
   `git status`



```
~/Documents/computing/capstone % git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
~/Documents/computing/capstone %
```

Figure 3.6: good



```
~/Documents/computing/capstone % git status
On branch dev
nothing to commit, working tree clean
~/Documents/computing/capstone %
```

Figure 3.7: bad

   If you do not have any changes present, then proceed to pull the latest changes. If you do, resolve this by saving your work.

iii. Pull the latest changes from your remote repository.

   `git pull origin dev`

iv. Create and Checkout a New Branch.

   `git checkout -b <new_branch_name>`

v. Begin working on your new changes

Stage, Commit, and Push your Changes

Below is the sequence of commands we refer to when we say to ”stage, commit, and push your changes”.

i. Verify which files you have changed.

ii. Optionally inspect which changes are present with git diff

iii. Add the files you wish to check in to your repository with git add ¡filename¿

iv. Run git commit and add a descriptive commit message

v. Run git push (if it is the first time you have pushed from this branch, add the –set-upstream flag)

vi. Continue working on your branch.

Open a Pull Request and Merge when Ready

When we say to ”open a pull request and merge when ready”, we are referring to the following steps.

i. Verify you have pushed all the lastest changes from your branch to your remote repository.

ii. Open your team’s repository in GitHub

iii. Open the pull requests page

iv. Select ”New Pull Request”

v. Set the base branch to dev and specify your newly created branch to merge into it

vi. Add a teammate as a reviewer and let them know you have opened a PR

vii. Once a teammate reviews your code, merge using the GitHub gui or manually in the terminal

viii. Resolve any merge conflicts, should they arrise