**Setting up Cloud9**

Welcome to Cloud9! We’re relying on this system as our test and development environment, letting us create and use individual or group workspaces without setting up individual tools, dependencies, and the rest.

Think of it as creating a virtual machine (a ‘workspace’, in their parlance) that one or more people can use at the same time.

What’s involved in setting this up? The first step is to create an account at c9.io. You can use your GitHub account info to create the account, but it will also ask for a credit card. The service does not cost anything, so the card is just for verification. If you’re not comfortable using yours, I’m happy to provide mine.

The best way to connect to GitHub is to use an SSH key to authenticate whatever terminal you’re connecting from. To prepare for that, go to your new Cloud9 account, click on the gear icon in the top right, and click on the SSH Keys option on the left hand side. Copy the text under “connect to your private git repository” to your clipboard.

Log in to your GitHub account, click on your user portrait at the top right, and click on Settings. Select SSH and GPG Keys from the left-hand side. On the next page, click on New SSH Key at the top right, give the new key a name, and then enter the text you copied from Cloud9.

Return to and log in to Cloud9. Create a new workspace. Give it a name, a description, and choose to make it private.

In the Clone from Git field, add the following: [git@github.com:panzerama/NSC-AD410-TIP-GROUP1.git](mailto:git@github.com:panzerama/NSC-AD410-TIP-GROUP1.git)

Finally, select the Php, Apache & … template.

You’ll be using git from the command line within Cloud9, so here are a few basic commands you’ll need.

git clone git@github.com:panzerama/NSC-AD410-TIP-GROUP1.git

If you forgot to fill in the Clone from Git field earlier, this will pull the existing codebase from GitHub to your workspace.

git pull [origin] [master]

Pull will retrieve the current codebase from the shared git repository. You can specify a source to pull from (origin is the most common and probably the only one you’ll use for this project) and a branch to pull (in this case, master)

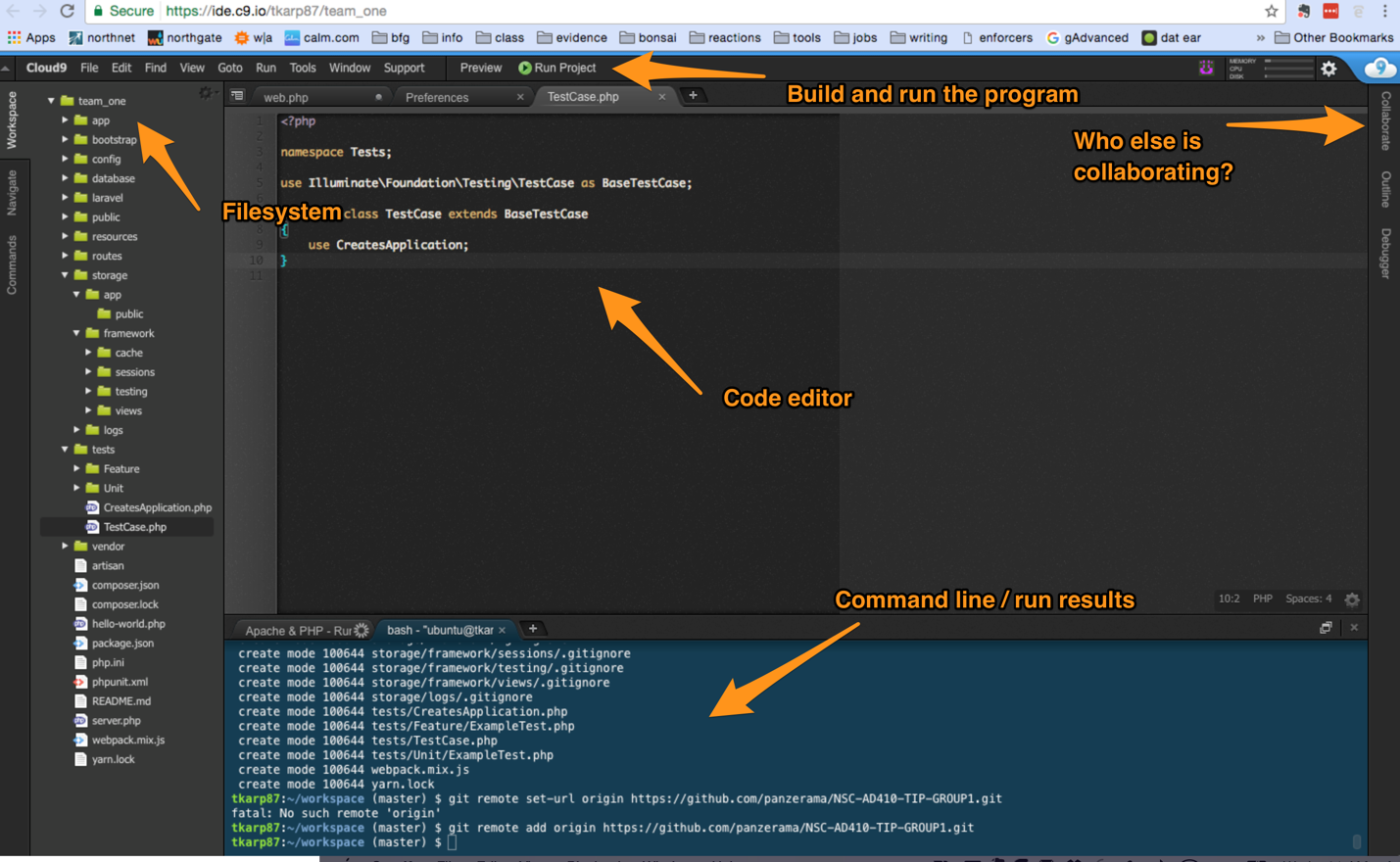
git checkout –b new/branch

create a new branch to work on. You should **always create a new branch when starting in on a new code change. Period.**

git push origin new/branch

When you’ve made the desired code changes, confirmed that they work, and returned to a working test state, you’re ready to push your changes up to the repo. Use this command to do so, and change new/branch to the name of whatever branch you’ve been working on.

Cloud9 provides its own welcome tour, so I’ll keep this brief.



Laravel

Laravel is an MVC framework written in php, compatible with a wide variety of other tools and databases.

For those of you who aren’t familiar with the Model-View-Controller design pattern, it’s a common pattern not just in web development but in all software engineering. A model in this case is a representation of a database record, a view is an interface, and a controller is the instructions that point actions and behaviors to the right views. As you can tell, these are very general terms.

In the context of Laravel, the default file structure helps manage those different functions. There is a single entry point to a Laravel site, the index.php file. We wont be writing full php pages, just views that can be served by Laravel to a browser session. Routes (controllers) direct an incoming user to the appropriate view. The code behind a view (which is where we write our actual php) may end up in one of serveral places, or several of several places, but when we need to fetch and manipulate data, we rely on a Model (the Eloquent system in Laravel) .

Sound daunting yet? Don’t worry, there’s plenty of instruction on this, and for the purposes of this app, we won’t be doing anything like rocket-surgery.

Where do we start?

The front-end team is going to start by mapping out routes to templates.

The back-end team is going to start by working on models and example queries so that we’re prepared for reporting later on.

Watch this before doing anything else: https://laracasts.com/series/laravel-5-from-scratch/episodes/2

Artisan utility:

Eloquent, Artisan, Blade… why with the pretentious names?

Every framework comes with it’s own terminology, and for Laravel, each of these terms refers to a different kind of function. Blade is the templating language for views. Eloquent is the library/syntax/toolset for models. Artisan is the command-line utility that will let you test changes, automate certain development tasks, and otherwise manage your development.

Request-Response Cycle

At the core of all of this is the request-response cycle. If you’re familiar with it, just understand that Laravel routes are designed to interpret a request (college.tips/form-in) and return a response by calling a view, some function, or event just ‘return “Hello World!”;’

For the uninitiated:

When I load a page like Facebook, my browser is asking the Facebook servers for information. That’s the request. The response is what my computer interprets and displays.

Requests may take the form of Get, Post, Put, Delete, Head, and a few other less common ones (<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods)>. Laravel interprets requests by parsing the URL and mapping it to a view or some logic. That’s the purpose of Routes. If I say ‘jdsblog.com/post/this-one-time’, Laravel will look for a route that matches that url. If, in my routes, I have the following:

Route::get(‘/post/{name}’, function () {

$name = ‘name’;

return view(‘something’)->with(‘name’);

});

Then Laravel will take the string ‘this-one-time’ and pass it into the view named ‘something’. This is a nonsense example, but the core concept is that I send a request to Laravel, it looks for a matching route, and returns what we designate based on that route. There’s a lot we can do under the hood, but it all spins around that concept.

What to do next: <https://laracasts.com/series/laravel-5-from-scratch/episodes/3>

Finish this series. It’s not long, and it will get you up to speed.

After that?

Front-end: <https://laracasts.com/series/laravel-5-fundamentals/episodes/5>

Back-end: <https://laracasts.com/series/laravel-5-fundamentals/episodes/8>

Questions? Hit me up in Slack. I promise you that I’m learning along with you, and we’ll figure it out as a team.