Working With Controllers

lab #lab04

James L. Parry
B.C. Institute of Technology

Lab Goals

The purpose of this lab is to help you explore and practice some of the controller and routing techniques for CodeIgniter.

The lab will result in fixing eight broken links in a starter webapp, using the routing techniques discussed in class.

The lab will be completed in teams of two or three, using proper collaborative workflow. The tutorial describes how to use the framework, while this lab writeup concentrates on the workflow to use.

Suggestion: you may want to skim the slideshow first, before working your way through it.

Lab Submission

Your lab will be completed in a github repository that has been forked from the starter-quotes repo.

Submit a readme to the lab dropbox. This readme should have a link to your github repository.

The link is the URL for your github repo page, **not** the cloning one (which has a `.git` extension). Further, it should not mention any branch ... see the last slide :-/

Due: three days after your lab, eg. Thursday lab period will have theirs due Sunday, Jan 31, 17:30 PST

Lab Marking Guideline

This lab will be marked out of 10, as follows:

- Eight marks for the link fixes, (one mark each)
- Two marks for proper collaboration.
- You may incur a marks penalty if your workflow is really poorly managed (wrong branch, not merged, inappropriate commits, etc)

Your Team

Form a team of two or three with classmates. Teams of one are not allowed, as this lab is about collaboration as well as routing.

Choose one of the team members to be the owner of your team repository.

You will be dividing the routing fixes between you. If a team of two, you are responsible for four fixes each; if a team of three, then the repository owner will be responsible for two fixes, and the other two team members for three each.

Workflow Overview

This lab is to be completed using "proper" collaborative workflow. This means forking and branching while working with your github repository!

Your team will have a single shared repository, with master (submittable) and develop (in progress) branches. This will be created in a new github "organization".

Each team member (including the owner) will fork the shared repo, into their own account, and clone their fork locally.

Link Fixing Workflow

A team member will create a feature branch locally, to hold the changes needed to fix a single broken link.

Changes are made in your local branch. Once happy with it, push it to your repository, and create a pull request to the team repo.

The team repo owner will review pull requests, merging those that work and which conform to your team coding conventions.

Once your branch has been accepted, you should delete it from your repo and resynch with the team repo.

Preparation - Team

Agree on a division of link fixes between the team members.

Ensure as much variety as possible with routing strategies. If a team of two, one of you should tackle all of the odd link fixes (#1, 3, etc) and the other the even link fixes (#2, 4, etc).

If working as a team of three, then the repo owner should tackle the first and fourth link fixes, a second team member the second, fifth and seventh ones; and the third team member the remaining ones, #3, 6 and 8.

Choose a coding style that the team will stick to, concerning comments, braces, and whitespace.

Agree on a changelog format and convention (txt, md or rst).

Preparation - Repo Owner

Create a github organization for your team.

Fork the starter repo there. This *is* the team repo

In the team repo, and on the github website, create or update the readme file to reflect the intended work allocation. Use initials or aliases for the team members if you want to preserve privacy.

Create a changelog file, in the repo root, with just an initial entry in it.

Make sure that the .gitignore file excludes any metadata folders or files for the IDEs your team members plan on using.

Commmit & push these changes to your `master` branch.

Create a new branch, `develop`, based on master. Make this the default branch.

Preparation - Team Members

Fork the team repo into your own account.

Yes, the team owner will have two repos, one in the team organization and one in his/her account.

Clone your forked repo to your development system.

Cloning will have given you an "origin" alias for your forked repo. Add an "upstream remote alias for the team repo. For instance,
`git remote add upstream team-repo-cloning-url-goes-here`

Make sure you have configured "git global" settings for your github username and email. Configure your IDE or set suitable defaults so that your commits will be signed.

Warmup Exercise

A great warmup exercise might help to make sure all the team members are clear on how to use the confusion of repositories!

Suggestion: have each team member "checkin", by adding a line to the readme file, agreeing to the conventions that the team plans to adopt. The real purpose of this is to make sure they have the aliases defined correctly, and that they can synchronize repositories :)

The slide following this one adresses synchronization.

Fixing a broken link

Resynch your repo with the team one. Hint: `git pull upstream develop` followed by `git push origin develop`

Create a new feature branch for the link fix you are about to undertake.

Fix away, adding or modifying routing configuration settings and controller code, to fix this one link.

Make sure your code conforms to the conventions the team agreed on as part of the preparaion.

Update the repo changelog with a line for the broken link fix you have just undertaken..

Commit your changes.

Publishing Your Fix

Resynch your repo with the team repo.

Push your feature branch to your fork.

On github.com, you should see a green button to create a pull request for the team repo. Do it.

You may now start on the next broken link.

Remember to create and switch to a new feature branch for the next broken link!

Process - Team Members

You will receive email confirmation of comments or acceptance of your pull request.

If your pull request has deficiencies reported, switch to the feature branch corresponding to it, fix those deficiencies, and commit/push to your branch again.

If your PR has comments or questions only, join the conversation on github.com.

If your PR is accepted, resynch repos, and you should then delete your feature branch, both on github in your fork and locally.

If your PR is rejected, continue working in your feature branch to correct defects, and then repush it and submit a new PR. If your feature branch is not easily fixed, delete that branch and start over.

Process - Repo Owner

You will get an email when a team members submits a pull request.

Reject any PRs that are not against your team repo's develop branch, or that are not properly signed off.

On github, check the PR, and add comments on any deficiencies you find. If major work is needed, feel free to reject the PR. They can resubmit one once they have fixed things. If you keep a badly broken PR open, then you will be party to the painful fixing process, as the submitter pushes changes to their feature branch.

There may be be corrections, but at some point you will be ready to either accept or reject the PR. If you reject a PR, make sure you explain why in the closing comment.

Are We Done Yet?

At the end of the lab, you will have merged eight feature branches, if all went well. On github, working with the team repo, create a PR to merge the develop branch into the master branch. Process it.

Check your repo for correctness. Seriously - the master branch is the only thing that your boss is going to consider for "pay" (marks).

Congratulations!

You have completed lab #lab04: Working With Controllers

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: ci-normal01 Controllers and Routing

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.