Exercise 9

In this exercise, you will deploy a web app in a more production-ready way. Use your app from Exercise8 (and the same Git repository). If for some reason, you'd prefer to work with your exercise 6/7 tutorial code, you're welcome to (as long as you meet the other requirements of this exercise).

Most of you have “deployed” your work for previous exercises using development tools: the devel server that came with your framework, SQLite, etc. That's a perfectly reasonable way to work on code, but not a reasonable way to deploy a production site.

Option 1: Vagrant VMs

There are three things that you should set up in a production way for this exercise: the web server, the database, and static file serving. You did a little work on this stuff in Exercise4.

Note: it's probably totally unrealistic to do all of these things on one server in a production environment, but if you can do it on one machine, you can do it on three separate ones easily. Separate VMs would just complicate this exercise.

See the DeployConfig page for hints on particular framework/server combinations.

Video: Production-mode deployment of a web app

Web Server

Get a web server like Nginx or Apache installed, and serve your application from it.

This could be with a server module (like mod_wsgi, mod_passenger, mod_php) or with a separate application server (nodejs, unicorn, gunicorn, uWSGI) proxied by the frontend server.

Database

Get a database server like PostgreSQL or MySQL running, and use it as the backend database for your application.

That will involve changing some configuration in your app, and setting up the database structure.

Static Files

Your project will certainly include some static files: CSS, JS, images, etc. If there was none in your original app, add one (even if it's just a one-line CSS).

Use the web server to serve these static files, not your application code: the frontend web server is the right place to do this. That will involve some configuration on the web server, and maybe getting the app's static files (or sometimes “assets”) somewhere to serve them from.

Initial Data (optional)

In the video, I added some initial data to my app's deployment. You aren't required to do that for this exercise, but this might be a good opportunity to try it.

You'll likely want some initial data in your project's deployment, so you can test and experiment right away without manually populating the database, which is boring.

Option 2: Docker Containers

For this exercise, you will need (at least) three containers: front-end web server, application server, database. You must set up your containers so the front-end web server is serving your static files. You did a little work on this stuff in Exercise4.

See the tech instructions page for hints on particular frameworks.

Web Server

Get a web server like Nginx or Apache installed, and serve your application from it.

The easiest is probably to create an image based on `nginx:latest` with a configuration file placed in `/etc/nginx/conf.d/default.conf` as you did for exercise 4.

Database

You can probably just use the `postgres:latest` image here, with a few environment variables set, so you know the username and password to configure in your app.

Static Files

Your project will certainly include some static files: CSS, JS, images, etc. If there was none in your original app, add one (even if it's just a one-line CSS).

Use the web server to serve these static files, not your application code: the frontend web server is the right place to do this. That will involve some configuration on the web server, and maybe getting the app's static files (or sometimes “assets”) somewhere to serve them from.

Initial Data (optional)

In the video, I added some initial data to my app's deployment. You aren't required to do that for this exercise, but this might be a good opportunity to try it.

You'll likely want some initial data in your project's deployment, so you can test and experiment right away without manually populating the database, which is boring.

Submission

We expect a working URL at http://localhost:8080/. In your `README`, include the link to the where the static files would be served (e.g /static).

If there are any other special instructions, include them in the `README`.

Commit everything and create a Git tag, as you have before. Submit your Git tag through the CourSys activity Exercise 9.

Updated Mon Aug. 30 2021, 07:36 by tienv.