**CIS 4930 - Software Testing for Continuous Delivery - Fall 2019**

# Professional Practice Assignment 2 --- Continuous Integration, Dependencies - Part 2 of In-Class Assignment #1

*(teams of up to 2 allowed for this assignment - use same teams from Part 1)*

---

## Objective

Build your own continuous integration pipeline with build and test phases. Setup your PPA #1 code in Github using a CI server instance of your choosing. Use Docker to store entries in a database. Create test doubles and integration tests. Gain experience with setting up a CI server, using Docker, and writing test doubles. Assignment can be completed **individually *or* with a partner** (both students will receive the same grade - recommended to partner with same person for PPA#1, and In-Class Part#1).

## Assignment

### **Phase 1:** Setup your PPA#1 Code for continuous integration

Setup a continuous integration server and code repo of your choosing (see list of suggestions below) to ***build*** and ***test*** your original PPA#1 code. The initial build phase could be trivial (i.e., just echo a message to the command prompt / log) if there are no required build steps (e.g., compilation, 3rd-party libraries, minifying, etc). The test phase should run all unit tests. A failure in one of these phases should result in a failure for the build (due by the end of class Friday, Oct 11, 2019).

### **Phase 2:** Add data storage and retrieval by using a database that runs in a Docker container

Choose a database that can run in a docker container (e.g., MySQL, Redis, MongoDB, etc. - not SQLite). Re-write 2 of your original functions to store all entries (input and returned values) with a human readable timestamp (e.g., 2019-10-11 5:02:34pm) . These entries should persist past the life of the running application (i.e., stored in the database). Upon the next execution of your application or if the user switches from the one function to the next, the DB entries for that function are retrieved and shown. Your program should first show the prior entries (with the names of the columns) before accepting new input. As new entries are made by the user, those entries should be stored for retrieval any time the function is selected.

Create test doubles (fakes, mocks, spies, dummies) to test your database functionality. Examples: Create stubs to test data storage and retrieval for each function. Create mocks to test whether certain steps in your process have been made (e.g., database connection established)…you decide. You should create at least 4 test double tests for each function that connects to the DB with at least one stub and one mock object. Integrate this phases into you CI Pipeline.

### **Phase 3:** Create basic web-enabled interface for the 2 DB backed functions

Build a web service into the current application to serve each database enabled function (2) as either a Get/Post request with JSON or HTML as the response (with JSON, you are basically setting up a REST API). The user should be able to execute your application from the command line as before. When the application is started, the existing command line interface works as well as a web service that responds to either GET/POST requests at a specified port (use: 5000). As long as the application is running via the command line, it is also listening and responding to requests. You determine the type of request your program requires and the type of response. Web requests should also be stored in the DB.

The user should also be able to request the entire list of entries from the database for the specified function: (e.g., http://localhost:3000/retirement - should return html or JSON with all the data from the retirement function table).

Write at least 4 unit tests to test http functionality (see notes below). Integrate this phases into you CI Pipeline.

### **Phase 4:** Complete CI Pipeline with Build and Test stages

Your final CI pipeline should contain at least one build stage and at least two test stages (create logical test groupings).

## Notes

- You should choose either a commercial / hosted CI server (e.g., Travis-CI, CircleCI, CodeShip, others) with either a free tier or that's free for public repositories, an integrated CI solution (e.g., Gitlab, Bitbucket Pipelines) with available free / student accounts, or an open source solution that you can install locally (e.g., Jenkins, Drone). If you use Gitlab or Bitbucket, please add me to the repo.
- Use Postman to functionally test your web environment - https://www.getpostman.com/product/api-client - This is the tool that we will use to send queries to your code.
- Review:
  - Test Doubles — Fakes, Mocks and Stubs - https://blog.pragmatists.com/test-doubles-fakes-mocks-and-stubs-1a7491dfa3da
  - Mocks Aren't Stubs - https://martinfowler.com/articles/mocksArentStubs.html#TheDifferenceBetweenMocksAndStubs
- Examples HTTP Request Testing (use at your own risk / discretion):
  - https://auth0.com/blog/mocking-api-calls-in-python/
  - https://blog.questionable.services/article/testing-http-handlers-go/
  - https://googleapis.github.io/google-http-java-client/unit-testing.html
  - https://scotch.io/tutorials/nodejs-tests-mocking-http-requests
- You do not need an active database connection to run your tests locally or via CI (test doubles should be used).

## SUBMIT

**Link to all data / files in Git Repo.**

Create 4 separate screencasts to show work from each of phase. Screencasts are only required to show functionality and CI interface for that Phase. Therefore, you are required to record a screencast at the completion of each phase. For Phase 4, show the entire working application from execution to quit, along with a build of the CI server. Each screencast should last no more than 30 second with the exception of Phase 4.

**Due Tuesday, October 15 at 11:59pm**

Professional Practice Assignment: **120**

---

100%