Assignment 2: Service-oriented Applications, Testing, and Containerization

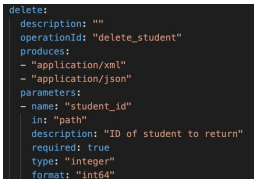
Craig Threadgold, 12547875

Introduction

Plan: The plan of what to develop has been created in the week2lab.pdf and is followed
Code**:** Altering the swagger, controller, and travis files in order to achieve the plan
Build**:** Committing the code to git and initiating a build in TravisCI to create the docker images and run the containers
Test**:** Executing the unit tests on the container that has been built by TravisCI
Release**:** Tagging the images built and tested by TravisCI then pushing them to Docker Hub

Tasks

2.1a. The issue was that the APIs used a standardised schema for the response to the client. This schema is defined in the swagger yaml file and reused throughout it in order to stop duplications of definitions. The schema needs a data type of what the response is going to be and this was incomplete. To fix it, the datatypes of the response were added to match those that were kept in the database. This allows the result from the database to be returned to the client via the API.



2.1b. The function that the API is linked to requires an input parameter of the student_id in order to be able to pass to the `delete_student` function the id for the database query to take place. This was fixed by adding an input parameter to the API and defining its type.

2.5a. This can be done by building, tagging, and pushing the `swagger-spring-example:1.0.0` image on the command line. In order to automate this with TravisCI, the .travis.yaml file was updated to match the Docker Hub repository name. TravisCI was then linked to GitHub in order to trigger a build of the .travis.yaml file when a commit to the remote master branch is made. TravisCI is linked to the Docker Hub repository in order to automatically push the image.

Service Granularity

The functions contained within student_service.py all take the input parameters from the API and return the API's response. This is a granular service as it performs one quite specific function. However, the script also contains the creation of a database. This means that there is lock-in to that database and will create a database (or db connection) whenever the script is utilised. In order to make the database configurable and increase the granularity of the script, the database could be created in a different package, and then service_student.py interacts with the new database script in a database-agnostic way. The disadvantage is that there is overhead in creating new services and making the link to other services in a way that is stateless. This is likely to be a small-scale operation so a local database run on a docker container would suffice as regular scaling is not required (the number of students is not going to be a Big Data challenge). I would also use MySQL as the data is relational in nature: a subject is shared by many students. This would allow local deployment and development with the ability to scale the database if deployed to the cloud.

GitHub Repository: https://github.com/threadgoldc/devops_assignment_02

Docker Hub Repository: `threadgoldc/assignment 2:submission`

Advanced Questions

2. Agile works well when there are changing requirements and timescales. It allows quicker software deployment as each increment must produce a functioning product. Instant feedback is acquired by real-life deployment so that bugs are found during real use (rather than testing) are found often and early, rather than all at once during final deployment. However, agile projects are prone to documentation being deprioritised. They also are easy to slip back into non-Agile methods, which can lead to none of the benefits of Agile being realised.