

Semester project report for Testing

Bancho Petrov, Alexander Gyurov, Romyana Vaseva

Learning goals as defined at the start of the project:

One of our focuses will be to choose a proper CI tool to integrate with our application in order to achieve continuous integration when deploying to a cloud-service.

Apart from that we will use Test Driven Development where requirements will be turned into specific test cases using the combination of Mochajs and Chaijs and then the code will be implemented to pass +those tests. Then the cycle will be repeated with small steps(Extreme Programming) and continuous integration will help by providing revertible checkpoints (if a test fails unexpectedly).

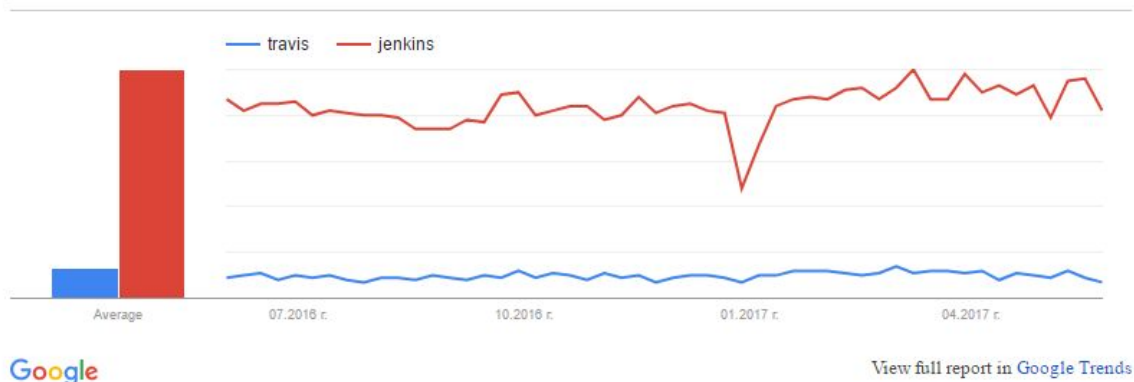
1. Continuous integration (Travis vs Jenkins)

In order to pick the proper tool we had to do some research on which would fit our needs better. Despite the many similarities between both tools and the fact that Jenkins seems to be much more popular (at least when it comes to Google searches, as visible in the figure below),

Interest over time of Travis and Jenkins

Note: It is possible that some search terms could be used in multiple areas and that could skew some graphs.

Interest over time. Web Search. Worldwide, Past 12 months, Programming.



we chose Travis CI because of the following reasons:

- Free Maintenance

Travis is a hosted service, so all that is needed to maintain is a configuration file, while Jenkins requires that one maintains a server for it.

- Easier and Quicker to Set-up

For the same reason stated above, Travis is also quite easier and faster to set-up. While Jenkins requires what could possibly be hours or even days of configuring host environments and dedicating a server for its needs, Travis is ready to use as soon as a github repository is linked to it and a travis.yml file is added to the root directory of the project.

- Free for Open Source Projects

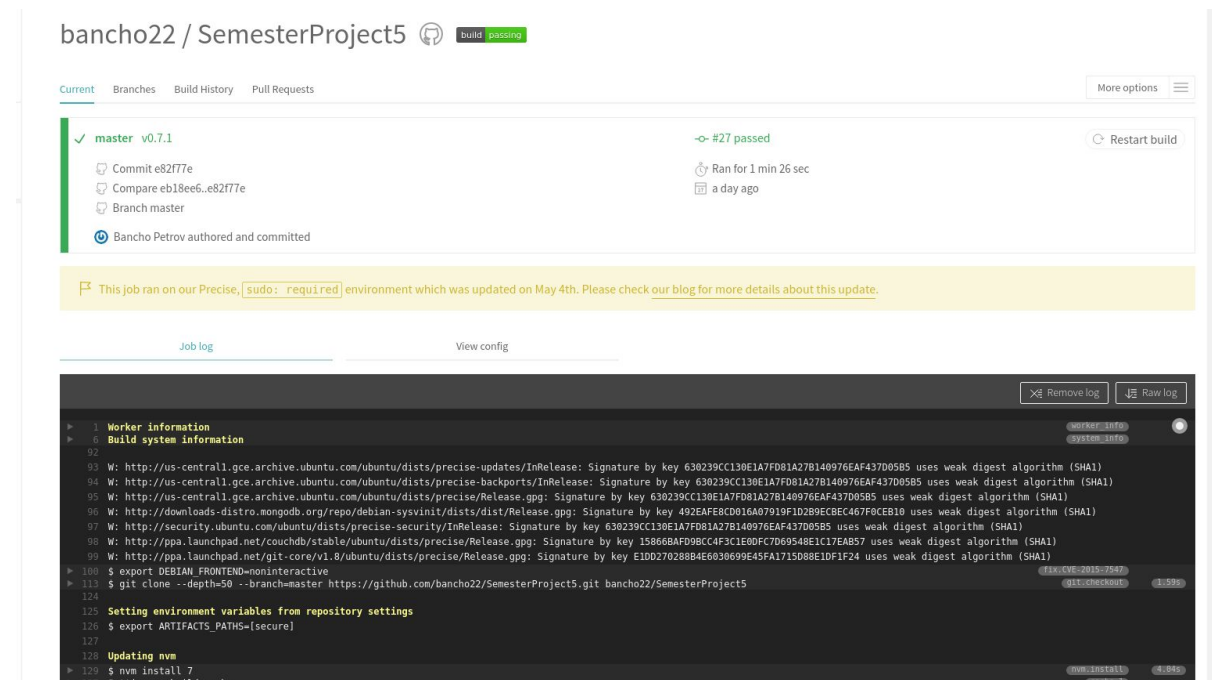
Travis can be quite expensive to use compared to Jenkins (which is open-source and therefore completely free) if using it for a private project, however, if integrating an open-source github project (as the one we are working on), Travis also comes for free.

Using Travis

We use Travis to deploy to Amazon Web Services. The way we've set-up our configuration file is the following:

- We've chosen master as the branch to be monitored and whenever there is a push to it, Travis will activate the sequence. This way we allow ourselves the freedom of using another branch (develop) to make pushes and share code that we do not necessarily want to be immediately tested and deployed.
- In the scripts member, we specify that each sequence should run the "npm test" command which we've specified in our package.json to execute our tests by invoking the mocha framework. Once the tests pass, we zip our project folder. This is needed for the next step which is deployment.
- The reason for zipping our project is that we are using AWS S3 in combination with AWS CodeDeploy and that requires that we upload a zip file which is then automatically unzipped and deployed to our EC2 instance. Unfortunately, due to what we believe to be some connectivity issues, once deployed, we are unable to access the running application on the remote machine, but we can confirm that the deploy is successful by ssh-ing the server and also by setting the 'wait-until-deployed' member to true in our travis.yml file.

In conclusion, we had no major issues setting up travis, while the integration with AWS proved to be much more challenging as Amazon documentation is lacking in terms of usability and readability.



The screenshot shows the Travis CI interface for the repository 'bancho22 / SemesterProject5'. The build status is 'passing' for the 'master' branch (v0.7.1). The job log is visible, showing the following steps:

```
1 Worker information
6 Build system information
92
93 W: http://us-central1.gce.archive.ubuntu.com/ubuntu/dists/precise-updates/InRelease: Signature by key 630239CC130E1A7FD81A27B140976EAF437D05B5 uses weak digest algorithm (SHA1)
94 W: http://us-central1.gce.archive.ubuntu.com/ubuntu/dists/precise-backports/InRelease: Signature by key 630239CC130E1A7FD81A27B140976EAF437D05B5 uses weak digest algorithm (SHA1)
95 W: http://us-central1.gce.archive.ubuntu.com/ubuntu/dists/precise/Release.gpg: Signature by key 630239CC130E1A7FD81A27B140976EAF437D05B5 uses weak digest algorithm (SHA1)
96 W: http://downloads-distroweb.org/repo/debian-sysvinit/dists/dists/Release.gpg: Signature by key 492EAF8C0816A07919F1D2B9ECBEC467F0CEB10 uses weak digest algorithm (SHA1)
97 W: http://security.ubuntu.com/ubuntu/dists/precise-security/InRelease: Signature by key 630239CC130E1A7FD81A27B140976EAF437D05B5 uses weak digest algorithm (SHA1)
98 W: http://ppa.launchpad.net/couchdb/stable/ubuntu/dists/precise/Release.gpg: Signature by key 15866BAF098CC4F3C1E00FC7D69548E1C17EAB57 uses weak digest algorithm (SHA1)
99 W: http://ppa.launchpad.net/git-core/v1.8/ubuntu/dists/precise/Release.gpg: Signature by key E10D27028884E6030699E45FA1715088E1DF1F24 uses weak digest algorithm (SHA1)
100 $ export DEBIAN_FRONTEND=noninteractive
113 $ git clone --depth=50 --branch=master https://github.com/bancho22/SemesterProject5.git bancho22/SemesterProject5
124
125 Setting environment variables from repository settings
126 $ export ARTIFACTS_PATHS=[secure]
127
128 Updating npm
129 $ npm install 7
131 Setting up build cache
```

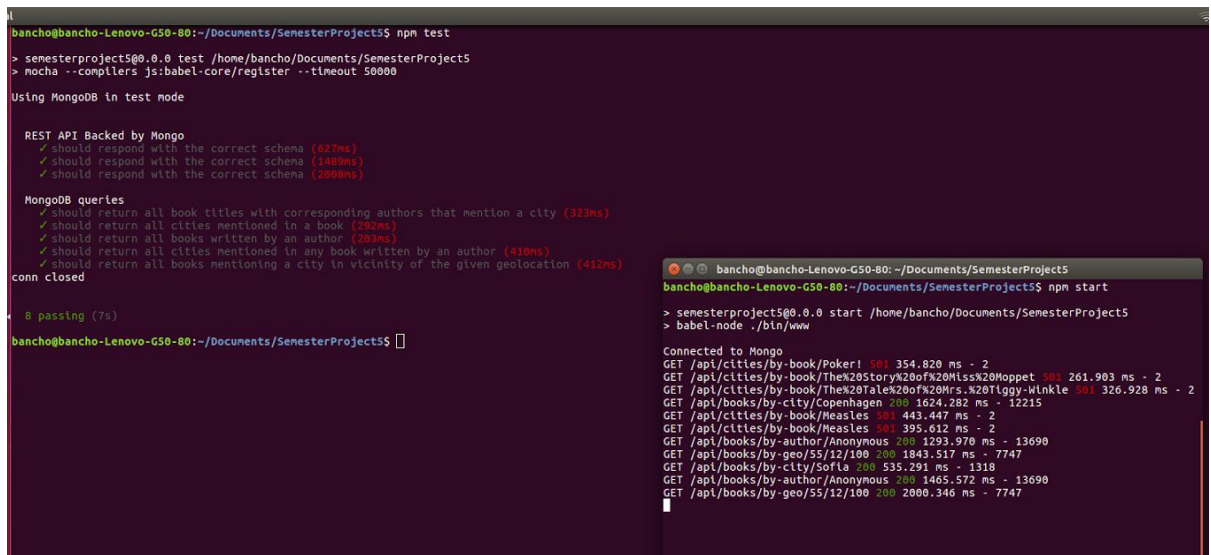
Link to our travis.yml: <https://github.com/bancho22/SemesterProject5/blob/master/.travis.yml>

NB: Even though, we are aware that this is extremely bad practise, we have neglected security and have some secret keys openly available in travis.yml file, due to the fact this is a school project not focusing on security issues.

2. TDD

2.1. Writing tests

We started by defining the json objects that would be contained in the responses of our REST API. The tests we implemented for that are designed to send a request to the server and validate the json schema. Due to the fact that we were unable to make the server accessible from AWS, we had to comment out this part of our test setup when deploying. Running the server on local environment, however, allows us to validate their correctness as can be seen in the figure below.



```
bancho@bancho-Lenovo-G50-80:~/Documents/SemesterProject5$ npm test
> semesterproject5@0.0.0 test /home/bancho/Documents/SemesterProject5
> mocha --compilers js:babel-core/register --timeout 50000

Using MongoDB in test mode

REST API Backed by Mongo
  ✓ should respond with the correct schema (627ms)
  ✓ should respond with the correct schema (1489ms)
  ✓ should respond with the correct schema (2068ms)

MongoDB queries
  ✓ should return all book titles with corresponding authors that mention a city (323ms)
  ✓ should return all cities mentioned in a book (292ms)
  ✓ should return all books written by an author (203ms)
  ✓ should return all cities mentioned in any book written by an author (410ms)
  ✓ should return all books mentioning a city in vicinity of the given geolocation (412ms)
conn closed

8 passing (7s)
bancho@bancho-Lenovo-G50-80:~/Documents/SemesterProject5$
```

```
bancho@bancho-Lenovo-G50-80:~/Documents/SemesterProject5$ npm start
> semesterproject5@0.0.0 start /home/bancho/Documents/SemesterProject5
> babel-node ./bin/www

Connected to Mongo
GET /api/cities/by-book/Poker! 501 354.820 ms - 2
GET /api/cities/by-book/The%20Story%20of%20Miss%20Moppet 501 261.903 ms - 2
GET /api/cities/by-book/The%20Tale%20of%20Mrs.%20Tiggy-Winkle 501 326.928 ms - 2
GET /api/books/by-city/Copenhagen 200 1624.282 ms - 12215
GET /api/cities/by-book/Measles 501 443.447 ms - 2
GET /api/cities/by-book/Measles 501 395.612 ms - 2
GET /api/books/by-author/Anonymous 200 1293.970 ms - 13690
GET /api/books/by-geo/55/12/100 200 1843.517 ms - 7747
GET /api/books/by-city/Sofia 200 535.291 ms - 1318
GET /api/books/by-author/Anonymous 200 1465.572 ms - 13690
GET /api/books/by-geo/55/12/100 200 2000.346 ms - 7747
```

Code can be seen here:

<https://github.com/bancho22/SemesterProject5/blob/master/test/endpointsTest.js>

2.2. Mocking the Database

The next step in our test-driven cycle was to create a test version of our database. We took a small part of the data we had and imported it to the mocked database.

Knowing exactly what we imported in the test version, how that data relates to itself and what the output of the queries would be, we wrote tests that would validate the correctness of our database layer. Once we made sure the tests can be executed and they fail (as no queries were implemented yet), we started implementing said queries.

Database layer test code can be seen here:

<https://github.com/bancho22/SemesterProject5/blob/master/test/mongoQueriesTest.js>

2.3. Implementing queries

An important feature of our database layer was the ability to specify which dataset to use. We achieved this by taking advantage of Javascript's functional features (as well as the latest ES7 syntax, making the code more compact and readable). We basically exported a function which takes in an optional parameter called `test` and returns an object containing functions which represent the queries themselves.

Code can be seen here:

<https://github.com/bancho22/SemesterProject5/blob/master/db/mongoQueries.js>