Testing Document

Organisation: `https://github.com/Hyperperform`

Developers:
Rohan Chhipa *14188377*
Avinash Singh *14043778*
Jason Gordon *14405025*
Claudio Da Silva *14205892*

Updated October 14, 2016

Client

MagnaBC
`http://www.magnabc.co.za/`

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Technologies . . . 3
- 1.3 Testing environment . . . 4
- 1.4 Assumptions . . . 4

2 Execution of unit tests **4**

3 Functional Features to be tested **5**
- 3.1 Event Listeners . . . 5
- 3.2 Event POJO's . . . 5
- 3.3 User management . . . 5

4 Features that will not be tested **6**

5 Components to be mocked **6**

6 Sample tests **7**

7 Unit Test Report **8**
- 7.1 Overview . . . 8
- 7.2 Test Cases . . . 8
  - 7.2.1 Test Class 1 - Rest Tests . . . 8
  - 7.2.2 Test Class 2 - User Management Tests . . . 10
  - 7.2.3 Test Class 3 - Entry/Exit Tests . . . 11
  - 7.2.4 Test Class 4 - Persistence Tests . . . 13

8 Conclusion **17**

9 Appendix **18**
- 9.1 Mock Events . . . 18

1 Introduction

This document explains implementation of unit tests which were used to test each component in an isolated environment. It also includes an overview of the technologies used as well as instructions on how to execute these tests.

1.1 Purpose

HyperPerform is an automated performance tracking tool, which sources information from many different integrations in order to provide information about various employee's performances levels and how they affect the current project's success level. One of the main goals of the system is for the system to be able to be used commercially or in an academic setting. To ensure that each component of the system is decoupled from every other; all aspects of the system are implemented such that they conform to the given contracts.

Thorough unit testing allows us to ensure that the each component of the HyperPerform system is working correctly. Unit testing gives us the ability to test the response of the system under different conditions.

All unit tests were written in such a way that the system is fully tested before being deployed, this includes testing all REST endpoints which are exposed, ensuring that the database is set up correctly and accepting objects to be persisted and ensuring that all subsystem contracts are met.

1.2 Technologies

Since the system is primarily coded in Java we decided to use the *JUnit testing framework* to carry out the unit tests for each of the components. We also used Spring to allow the use of dependency injection within each test, this allows us to easily inject mocks and test components in an isolated environment.

Maven was used as the build tool for this system. When building the system from its source code, all unit tests are automatically executed by Maven.

Another technology which proved useful was a Mock Dispatcher framework which ships with RESTEasy. The mock dispatcher allows one to easily and efficiently test REST API's without having to deploy the component to an application server.

1.3 Testing environment

- **Programming Languages:** Java with JavaEE for the back-end server, front-end relies on AngularJS with Bootstrap, JQuery and Sass.
- **Testing frameworks:** Unit testing is done using JUnit. This is combined with Spring, which is used to achieve dependency injection within the unit tests.
- **Operating System:** The unit tests are not platform specific. The only requirement in terms of operating systems is that Maven must be supported. If Maven is supported then all unit test dependencies will be downloaded and all the tests will be executed automatically.
- **Internet Browsers:** The front-end system supports the latest versions of *Google Chrome* and *Firefox*. There is limited support for other browsers.

1.4 Assumptions

- A database system is installed.
- The persistence xml corresponds to the current database installed on the platform.
- All necessary tables have already been created.
- Maven is installed.

2 Execution of unit tests

The project has been developed using Maven as a build tool. Thus each unit test can be easily found within the *src/test* directory of the project. To run the unit tests simply run the following command in terminal (in the same directory as the pom file):

```
mvn clean test
```

All the necessary dependencies for the project will be automatically downloaded.
Note: This process of downloading all the project dependencies might consume large amounts of data and time.

3 Functional Features to be tested

3.1 Event Listeners

The event listener are a crucial component to the HyperPerform system. Without them gathering information becomes very difficult. Listeners to be tested are:

- **GitListener** - Receives Push and Issue events from GitHub.
- **TravisListener** - Receives build events from Travis CI.
- **AccessListener** - Receives access events from Entry/Exit.

Mock JSON data has been passed to the GitListener, TravisListener and AccessListener classes (into the listen functions). These mock events can be found in the Appendix. Each of these components have dependencies on features such as the messaging queue. All of these features are mocked out to ensure each component is tested in isolation. These features were mocked through dependency injection which was provided by Spring.

3.2 Event POJO's

The event POJO's contain all the important employee data that needs to be processed at a later stage. The persistability of these POJO's is crucial for the HyperPerform system. The following POJO's will be tested:

- **GitPushEvent** - A POJO that contains information regarding a GitHub push event.
- **TravisEvent** - A POJO that contains information regarding a build triggered by a push event from GitHub.
- **GitIssueEvent** - A POJO that contains information regarding a GitHub issue that has been created.
- **AccessEvent** - A POJO that contains information regarding access in/out of a building containing a card reader.
- **User** - A POJO that contains information regarding a user who has been registered to the system.

3.3 User management

This section entails testing the user management functionallity of the system. The following items will be tested:

- **User creation** - Ability to successfully create and persist a new user to the database.

- **Invalid names** - Checks the validity of the username provided.
- **Invalid E-mail** - Checks the validity of the E-mail provided. Also checks if E-mail already exists within the system.
- **Invalid role** - Checks the validity of the employee role. Roles are provided by the system but integrity of the role must still be tested.
- **Invalid Position** - Checks the validity of the employee position. Positions are provided by the system but integrity of the position must still be tested.
- **Invalid Password** - Checks validity of password.

4 Features that will not be tested

Features that have been provided by frameworks will not be tested. Operations such as adding an object to a messaging queue will not be tested. Mapping of JSON data to request objects as this is also done by frameworks.

The ability to persist POJO's will not be tested within the listeners. These persistence tests are kept separate.

5 Components to be mocked

- The JPA entity manager was the first component that was mocked. When testing the possibility for persistence of the POJO's it would be best if the transactions do not affect the database. Thus every transaction that occurs with the mocked entity manager will leave the database system intact and unaffected. An alternate approach could have been to use an in-memory database - such as the one provided by H2 - however this was not deemed necessary.

- The second component to be mocked was the messaging queue provided by ActiveMQ. Once again we do not wish to have unnecessary objects in our queue that might affect actual program execution thus the default queue is replaced with a queue that does not retain objects.

- When testing the event listeners we can't wait for the event emitting systems to send out an event. So we send mock events to the listeners while testing, these mock events are structured in the same manner as their real-world counterparts.

6 Sample tests

The following figure is the GitListener dependency injected queue which allows multiple events to come through and not be lost and the creation of the entity manager.

```
@Path("/gitEvent")
public class GitListener implements IListener
{
    /**
     * Connection to the messaging queue. The object is provided through dependency injection.
     */
    @Inject
    QueueConnection queueConnection;

    /**
     * Persistence context which allows for persisting the events received.
     */
    EntityManagerFactory entityManagerFactory;
    EntityManager entityManager;

    @PostConstruct
    private void initConnection()
    {
        entityManagerFactory = Persistence.createEntityManagerFactory("PostgreJPA");
        entityManager = entityManagerFactory.createEntityManager();
    }

    @PreDestroy
    private void disconnect()
    {
        entityManager.close();
        entityManagerFactory.close();
    }
```

Figure 1: GitListener Dependency injection

The following figure is one of the GitListener Event tests.

```
POJOResourceFactory noDef = new POJOResourceFactory(GitListener.class);
Dispatcher dispatcher = MockDispatcherFactory.createDispatcher();

dispatcher.getRegistry().addResourceFactory(noDef);

MockHttpRequest request = MockHttpRequest.post("/gitEvent");

request.header("X-GitHub-Event", "push");
request.contentType(MediaType.APPLICATION_JSON_TYPE);

request.content(MockEvent.gitPushEvent.getBytes());

MockHttpResponse response = new MockHttpResponse();
dispatcher.invoke(request, response);

Assert.assertEquals(response.getStatus(), 200);
```

Figure 2: GitListener Push event test

In the above figure the *POJOResourceFactory* and *Dispatcher* are used to start up an embedded server which will allow for calls to be made to a particular URL, in this case \ gitEvent.

A post request is created and has the mock event as its content. This post request mirrors the post requests made by GitHub when sending events. Once the mock event data is loaded into the request, the request is sent. At the end, the response objects' HTTP status code is checked. This is checked in an assert statement, the value of the response should be 200 to indicate a successful retrieval.

```
POJOResourceFactory noDef = new POJOResourceFactory(GitListener.class);
Dispatcher dispatcher = MockDispatcherFactory.createDispatcher();

dispatcher.getRegistry().addResourceFactory(noDef);

MockHttpRequest request = MockHttpRequest.get("/gitEvent/random");
MockHttpResponse response = new MockHttpResponse();
dispatcher.invoke(request, response);

Assert.assertEquals(response.getStatus(), 404);
```

Figure 3: Exception handler for invalid URLs

7 Unit Test Report

7.1 Overview

All the tests in the system build correctly and pass the test. The only reason that a test may fail is if the database tables have not been created correctly. This may occur due to the fact that these tables are not created using JPA, since JPA destroys and creates the tables upon execution of the command: *mvn clean test*. Travis CI is used on our repository to ensure that the build do pass.

7.2 Test Cases

7.2.1 Test Class 1 - Rest Tests

Test case 1: Git Push Event Test
Objective: This test will ensure that the git listener REST endpoint is working and successfully receives events
Input: A mock git push event is send through to the link

Outcome: Once the event is sent through, a HTTP 200 response code is expected.

Test case 2: Git Issues Event Test
Objective: This test will ensure that the git listener REST endpoint is working and successfully accepts git issue events
Input: A mock git issue event is send through to the link
Outcome: Once the event is sent through, a HTTP 200 response code is expected.

Test case 3: Invalid link Test
Objective: This test is used to see if an appropriate response is sent for non-existing pages
Input: An intentional invalid link is accessed
Outcome: An HTTP 404 response code is expected.

Test case 4: Timezone Test
Objective: This test checks the adaptability of the Git Listeners in terms of timezones. Its aims to see if a correct timezone can be parsed from the received data.
Input: An alternative git push event is sent through
Outcome: An HTTP 200 response code is expected.

Test case 5: Travis Test
Objective: This test checks whether or not the travis listener is working. This is tested by sending through a mock event
Input: A travis event is sent through
Outcome: An HTTP 200 response code is expected.

Test case 6: Login Test
Objective: This test checks whether or not the login subsystem under the user management system is working.
Input: Mock user details are sent through to the REST endpoint
Outcome: An HTTP 200 response code is expected.

Test case 7: Access Test
Objective: This test checks whether or not the access listener is working. This is tested by sending through a mock event
Input: A mock access event is sent through
Outcome: An HTTP 200 response code is expected.

All these test cases above can be found on the following link: `https://github.com/HyperPerform/hyper-perform-server/blob/develop/src/test/java/me/hyperperform/rest/RestTest.java`

7.2.2 Test Class 2 - User Management Tests

All mock data can be found in the Appendix.
Test case 1: Create User Test
Objective: This test will create 2 different users with different roles and persist them into the database.
Input: There is no direct input into this test.
Outcome: The newly created users are successfully persisted to the database.

Test case 2: User Test
Objective: Check the integrity of the information of the above users and delete them from the database, to ensure no primary key violation.
Input: There is no direct input into this test.
Outcome: The newly created users data is as expected and then are successfully deleted from the database.

Test case 3: Registration Test
Objective: Check to see if a user can be added to the system with all valid information.
Input: A mock normalUser event is sent through
Outcome: An HTTP 200 response code is expected. There after the test will delete the user to ensure no primary key violation.

Test case 4: Invalid Name Test
Objective: Check to see if a user can be added to the system with no username.
Input: A mock noUsername event is sent through.
Outcome: An HTTP 200 response code is expected with a response message "Error: name"

Test case 5: Invalid Surname Test
Objective: Check to see if a user can be added to the system with no surname.
Input: A mock noSurname event is sent through.
Outcome: An HTTP 200 response code is expected with a response message "Error: surname"

Test case 6: Invalid Email Test
Objective: Check to see if a user can be added to the system with no email.
Input: A mock noEmail event is sent through.
Outcome: An HTTP 200 response code is expected with a response message "Error: email"

Test case 7: Invalid Role Test
Objective: Check to see if a user can be added to the system with an invalid role.
Input: A mock invalidRole event is sent through.
Outcome: An HTTP 200 response code is expected with a response message "Error: Role does not exist"

Test case 8: Invalid Position Test
Objective: Check to see if a user can be added to the system with an invalid position.
Input: A mock invalidPosition event is sent through.
Outcome: An HTTP 200 response code is expected with a response message "Error: Position does not exist"

Test case 9: Invalid GitUsername Test
Objective: Check to see if a user can be added to the system with no GitUserName.
Input: A mock noGitUsername event is sent through.
Outcome: An HTTP 200 response code is expected with a response message "Error: gitUserName"

Test case 10: Invalid Password
Objective: Check to see if a user can be added to the system with no password.
Input: A mock noPassword event is sent through.
Outcome: An HTTP 200 response code is expected with a response message "Error: password"

All these test cases above can be found on the following link: `https://github.com/HyperPerform/hyper-perform-server/blob/develop/src/test/java/me/hyperperform/user/UserTest.java`

7.2.3 Test Class 3 - Entry/Exit Tests

Test case 1: Create Access Event Test
Objective: The purpose of this test is to be a precursor to see if an access event can be created and then persisted to the database.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. The access event should be persisted in the database. This is tested in test case 3

Test case 2: POJO Test
Objective: The purpose of this test is to check if the access event from test case 1 mapped correctly to the *AccessEvent* POJO.

Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. Should assert that the expected name is the same as the name mapped on to the POJO.
2. Should assert that the expected email address is the same as the email address mapped on to the POJO.
3. Should assert that the expected surname is equal to the surname mapped on to the POJO.
4. Should assert that the expected userID is equal to the userID mapped on to the POJO.
5. Should assert that the expected deviceID is equal to the deviceID mapped on to the POJO.
6. Should assert that the expected day is equal to the day mapped on to the POJO.
7. Should assert that the expected timestamp is equal to the timestamp mapped on to the POJO.

Test case 3: Querying the database test
Objective: The purpose of this test is to query the AccessEvent table in the database and check if the access event from test case 1 persisted correctly to the database.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. Should assert that the expected name is equal to that of the one persisted.
2. Should assert that the expected email address is equal to that of the one persisted.
3. Should assert that the expected surname is equal to that of the one persisted.
4. Should assert that the expected userID is equal to that of the one persisted.
5. Should assert that the expected deviceID is equal to that of the one persisted.
6. Should assert that the expected day is equal to that of the one persisted.
7. Should assert that the expected timestamp is equal to that of the one persisted.

All these test cases can be found on the following link: `https://github.com/HyperPerform/hyper-perform-server/blob/develop/src/test/java/me/hyperperform/event/EntryExitTest.java`

7.2.4 Test Class 4 - Persistence Tests

Test case 1: Create Travis Event Test
Objective: The purpose of this test is to see if a Travis Event can be created, mapped to the POJO and then persisted to the database.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. The created Travis event should be persisted to the database. This is tested in test case 9.

Test case 2: Create Git Issue Test
Objective: The purpose of this test is to see if a Git Issue Event can be created, mapped to the POJO and then persisted to the database.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. The Git Issue that was created should be persisted to the database. This is queried and tested in test case 8.

Test case 3: Create Git Push Test
Objective: The purpose of this test is to see if a Git Push Event can be created, mapped to the POJO and then persisted to the database.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. The Git Push (along with their commits) that was created should be persisted to the database. This is queried and tested in test case 7.

Test case 4: Travis Event POJO Test
Objective: The purpose of this test is to see if the Travis Event in test case 1 mapped correctly to the POJO.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. Should assert that the expected committer is equal to that of the one mapped onto the POJO.
2. Should assert that the expected branch is equal to that of the one mapped onto the POJO.
3. Should assert that the expected status of the issue is equal to that of the one mapped onto the POJO.

4. Should assert that the expected timestamp is equal to that of the one mapped onto the POJO.
5. Should assert that the expected repository name is equal to that of the one mapped onto the POJO.

Test case 5: Git Issue POJO Test
Objective: The purpose of this test is to see if the Git Issue in test case 2 mapped correctly to the POJO.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. Should assert that the expected issueId is equal to that of the one mapped onto the POJO.
2. Should assert that the expected action is equal to that of the one mapped onto the POJO.
3. Should assert that the expected repository name is equal to that of the one mapped onto the POJO.
4. Should assert that the timestamp that was mapped onto the POJO is not null.
5. Should assert that the expected timestamp is equal to that of the one mapped onto the POJO.
6. Should assert that the expected assignee of the issue is equal to that of the one mapped onto the POJO.
7. Should assert that the expected creator of the issue is equal to that of the one mapped onto the POJO.

Test case 6: Git Push POJO Test
Objective: The purpose of this test is to see if the Git Push Event in test case 3 mapped correctly to the POJO.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. Should assert that the expected repository name/path is equal to that of the one mapped onto the POJO.
2. Should assert that the expected timestamp is equal to that of the one mapped onto the POJO.
3. Should assert that the expected username is equal to that of the one mapped onto the POJO.

4. Should assert that the commit size that was mapped onto the POJO cannot be equal to zero.
5. Should assert that the expected commit size is equal to that of the one mapped onto the POJO.

Test case 7: Git Push Query Test
Objective: The purpose of this test is to query the GitPush table in the database and check if the GitPush event from test case 3 persisted correctly to the database.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. Should assert that the query from the database does return results thus the number of results is not equal to 0 (zero).
2. Should assert that the expected repository name/path is equal to that of the one persisted.
3. Should assert that the expected Date (extracted from the timestamp) is equal to that of the one persisted.
4. Should assert that the expected username is equal to that of the one persisted.
5. Should assert that the commit size that was persisted to the database cannot be equal to zero.
6. Should assert that the expected commit size is equal to that of the one persisted.

Test case 8: Git Issue Query Test
Objective: The purpose of this test is to query the GitIssue table in the database and check if the GitIssue event from test case 2 persisted correctly to the database.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. Should assert that the query from the database does return results (i.e. the number of results is not equal to 0 (zero)).
2. Should assert that the expected action is equal to that of the one persisted.
3. Should assert that the expected repository name is equal to that of the one persisted.
4. Should assert that the expected timestamp is equal to that of the one persisted.
5. Should assert that the expected assignee of the issue is equal to that of the one persisted.
6. Should assert that the expected creator of the issue is equal to that of the one persisted.

Test case 9: Travis Event Query Test
Objective: The purpose of this test is to query the TravisEvent table in the database and check if the Travis event from test case 1 persisted correctly to the database.
Input: There is no direct input to the test.
Outcome: The following are expected outcomes for the functionality:

1. Should assert that the query from the database does return results thus the number of results is not equal to 0 (zero).
2. Should assert that the expected committer is equal to that of the one persisted.
3. Should assert that the expected branch, on which the builds was done, is equal to that of the one persisted.
4. Should assert that the expected status of the build is equal to that of the one persisted.
5. Should assert that the expected timestamp is equal to that of the one persisted.
6. Should assert that the expected repository name is equal to that of the one persisted.

All these test cases can be found on the following link: `https://github.com/HyperPerform/hyper-perform-server/blob/develop/src/test/java/me/hyperperform/event/PersistenceTest.java`

```
Results :

Tests run: 36, Failures: 0, Errors: 0, Skipped: 2

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 19.418 s
[INFO] Finished at: 2016-10-12T14:16:46+02:00
[INFO] Final Memory: 24M/58M
[INFO] ------------------------------------------------------------------------
```

Figure 4: Final Build Status

8 Conclusion

All 36 tests that have been run have passed (as shown by Figure 4 above). Additionally this was confirmed by Travis CI which we have used on our repository to make sure that there are no compilation failures or test failures. The limitations to these tests are the fact that they require the relevant tables to already exist in the Postgres database. JPA has been configured not to create these tables automatically. If this was done it would delete the tables with all the data during testing. A work around for this is to write a script or a unit test that will create these tables if they don't exist before the rest of the unit tests are executed.

9 Appendix

9.1 Mock Events

Please note that some of these mock events are very large. Due to this redundant data has been removed from the events so that they can be presented in this document.

The fully detailed mock events can be found through the following link: `https://github.com/HyperPerform/hyper-perform-server/blob/develop/src/test/java/me/hyperperform/event/MockEvent.java`

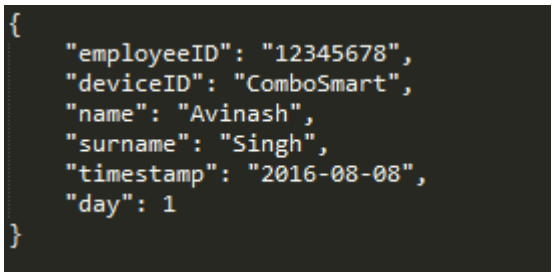

Figure 5: Access event payload

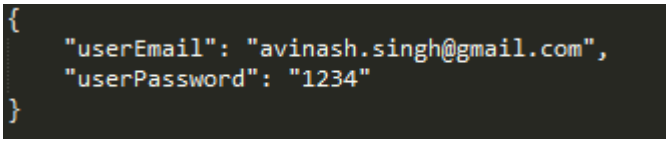

Figure 6: Login payload

```
{
    "ref": "refs/heads/changes",
    "before": "9049f1265b7d61be4a8904a9a27120d2064dab3b",
    "after": "0d1a26e67d8f5eaf1f6ba5c57fc3c7d91ac0fd1c",
    "created": false,
    "deleted": false,
    "forced": false,
    "base_ref": null,
    "compare": "https://github.com/baxterthehacker/public-repo/compare/9049f1265b7d...0d1a26e67d8f",
    "commits": [
        {
            "id": "0d1a26e67d8f5eaf1f6ba5c57fc3c7d91ac0fd1c",
            "tree_id": "f9d2a07e9488b91af2641b26b9407fe22a451433",
            "distinct": true,
            "message": "Update README.md",
            "timestamp": "2015-05-05T19:40:15-04:00",
            "url": "https://github.com/baxterthehacker/public-repo/commit/0d1a26e67d8f5eaf1f6ba5c57fc3c7d91ac0fd1c",
            "committer": {
                "name": "baxterthehacker",
                "email": "baxterthehacker@users.noreply.github.com",
                "username": "baxterthehacker"
            }
        }
    ],
    "head_commit": {
        "id": "0d1a26e67d8f5eaf1f6ba5c57fc3c7d91ac0fd1c",
        "tree_id": "f9d2a07e9488b91af2641b26b9407fe22a451433",
        "distinct": true,
        "message": "Update README.md",
        "timestamp": "2015-05-05T19:40:15-04:00",
        "url": "https://github.com/baxterthehacker/public-repo/commit/0d1a26e67d8f5eaf1f6ba5c57fc3c7d91ac0fd1c",
        "author": {
            "name": "baxterthehacker",
            "email": "baxterthehacker@users.noreply.github.com",
            "username": "baxterthehacker"
        },
        "committer": {
            "name": "baxterthehacker",
            "email": "baxterthehacker@users.noreply.github.com",
            "username": "baxterthehacker"
        },
        "added": [],
        "removed": [],
        "modified": [
            "README.md"
        ]
    },
    "repository": {
        "id": 35129377,
        "name": "public-repo",
        "full_name": "baxterthehacker/public-repo",
        "owner": {
            "name": "baxterthehacker",
            "email": "baxterthehacker@users.noreply.github.com"
        }
    },
    "pusher": {
        "name": "baxterthehacker",
        "email": "baxterthehacker@users.noreply.github.com"
    }
}
```


Figure 7: Git push event payload

```
{
    "ref": "refs/heads/master",
    "before": "afc7afa4d0703978a7941d6135a141aa06fe40d9",
    "after": "054d091c30d6744723e25534f5c9b5564908d730",
    "created": false,
    "deleted": false,
    "forced": false,
    "base_ref": null,
    "compare": "https://github.com/RohanChhipa/COS332/compare/afc7afa4d070...054d091c30d6",
    "head_commit": {
        "id": "054d091c30d6744723e25534f5c9b5564908d730",
        "tree_id": "48c110763039a2894181829a4dea730e10dd3cf2",
        "distinct": true,
        "message": "deletedfile",
        "timestamp": "2016-07-28T22:42:4402:00",
        "url": "https://github.com/RohanChhipa/COS332/commit/054d091c30d6744723e25534f5c9b5564908d730",
        "author": {
            "name": "rohanchhipa",
            "email": "rohan.chhipa@live.com",
            "username": "RohanChhipa"
        },
        "committer": {
            "name": "rohanchhipa",
            "email": "rohan.chhipa@live.com",
            "username": "RohanChhipa"
        },
        "added": [],
        "removed": [
            "testFile"
        ],
        "modified": []
    },
    "repository": {
        "id": 50978789,
        "name": "COS332",
        "full_name": "RohanChhipa/COS332",
        "owner": {
            "name": "RohanChhipa",
            "email": "u14188377@tuks.co.za"
        },
        "private": true,
        "html_url": "https://github.com/RohanChhipa/COS332",
        "description": "For COS332 practicals",
        "fork": false,
        "created_at": 1454480387,
        "updated_at": "2016-02-03T08:49:53Z",
        "pushed_at": 1469738587,
        "homepage": null,
        "size": 46,
        "default_branch": "master",
        "stargazers": 2,
        "master_branch": "master"
    },
    "pusher": {
        "name": "RohanChhipa",
        "email": "u14188377@tuks.co.za"
    }
}
```


Figure 8: Alternative Git push event payload

```
{
    "action": "opened",
    "issue": {
        "id": 73464126,
        "number": 2,
        "title": "Spelling error in the README file",
        "user": {
            "login": "baxterthehacker",
            "id": 6752317,
            "gravatar_id": "",
            "type": "User",
            "site_admin": false
        },
        "state": "open",
        "locked": false,
        "assignee": null,
        "milestone": null,
        "comments": 0,
        "created_at": "2016-07-28T22:42:4402:00",
        "updated_at": "2016-07-28T22:42:4402:00",
        "closed_at": null,
        "body": "It looks like you accidently spelled 'commit' with two 't's."
    },
    "repository": {
        "id": 35129377,
        "name": "public-repo",
        "full_name": "baxterthehacker/public-repo",
        "owner": {
            "login": "baxterthehacker",
            "id": 6752317,
            "gravatar_id": "",
            "type": "User",
            "site_admin": false
        },
        "private": false,
        "description": "",
        "fork": false,
        "created_at": "2015-05-05T23:40:12Z",
        "updated_at": "2015-05-05T23:40:12Z",
        "pushed_at": "2015-05-05T23:40:27Z",
        "homepage": null,
        "size": 0,
        "stargazers_count": 0,
        "watchers_count": 0,
        "language": null,
        "has_issues": true,
        "has_downloads": true,
        "has_wiki": true,
        "has_pages": true,
        "forks_count": 0,
        "mirror_url": null,
        "open_issues_count": 2,
        "forks": 0,
        "open_issues": 2,
        "watchers": 0,
        "default_branch": "master"
    }
}
```


Figure 9: Git issues event payload

```
{
    "id": 1,
    "number": "1",
    "status": null,
    "started_at": null,
    "finished_at": null,
    "status_message": "Passed",
    "commit": "62aae5f70ceee39123ef",
    "branch": "master",
    "message": "the commit message",
    "compare_url": "https://github.com/svenfuchs/minimal/compare/master...develop",
    "committed_at": "2011-11-11T11: 11: 11Z",
    "committer_name": "Sven Fuchs",
    "committer_email": "svenfuchs@artweb-design.de",
    "author_name": "Sven Fuchs",
    "author_email": "svenfuchs@artweb-design.de",
    "type": "push",
    "build_url": "https://travis-ci.org/svenfuchs/minimal/builds/1",
    "repository": {
        "id": 1,
        "name": "minimal",
        "owner_name": "svenfuchs",
        "url": "http://github.com/svenfuchs/minimal"
    },
    "config": {
        }
    },
    "matrix": [
        {
            "id": 2,
            "repository_id": 1,
            "number": "1.1",
            "state": "created",
            "started_at": null,
            "finished_at": null,
            "config": {
                "notifications": {
                    "webhooks": [
                        "http://evome.fr/notifications",
                        "http://example.com/"
                    ]
                }
            },
            "status": null,
            "log": "",
            "result": null,
            "parent_id": 1,
            "commit": "62aae5f70ceee39123ef",
            "branch": "master",
            "message": "the commit message",
            "committed_at": "2011-11-11T11: 11: 11Z",
            "committer_name": "Sven Fuchs",
            "committer_email": "svenfuchs@artweb-design.de",
            "author_name": "Sven Fuchs",
            "author_email": "svenfuchs@artweb-design.de",
            "compare_url": "https://github.com/svenfuchs/minimal/compare/master...develop"
        }
    ]
}
```


Figure 10: Travis build event payload

```
{
  "managerEmail": "admin@hyperperform.me",
  "userName":"rohan",
  "userSurname":"chhipa",
  "userEmail":"rohanchhipa@live.com",
  "userPassword":"1234",
  "role":"Employee",
  "position":"SoftwareDeveloper",
  "gitUserName": "rohan"
}
```


Figure 11: Normal User


```
{
  "managerEmail": "admin@hyperperform.me",
  "userName": "",
  "userSurname": "chhipa",
  "userEmail": "rohanchhipa@live.com",
  "userPassword": "1234",
  "role": "Employee",
  "position": "SoftwareDeveloper",
  "gitUserName": "rohan"
}
```


Figure 12: No Username


```
{
  "managerEmail": "admin@hyperperform.me",
  "userName": "rohan",
  "userSurname": "",
  "userEmail": "rohanchhipa@live.com",
  "userPassword": "1234",
  "role": "Employee",
  "position": "SoftwareDeveloper",
  "gitUserName": "rohan"
}
```


Figure 13: No Surname


```
{
  "managerEmail": "admin@hyperperform.me",
  "userName": "rohan",
  "userSurname": "chhipa",
  "userEmail": "",
  "userPassword": "1234",
  "role": "Employee",
  "position": "SoftwareDeveloper",
  "gitUserName": "rohan"
}
```


Figure 14: No Email

```
{
  "managerEmail": "admin@hyperperform.me",
  "userName": "rohan",
  "userSurname": "chhipa",
  "userEmail": "rohanchhipa@live.com",
  "userPassword": "",
  "role": "Employee",
  "position": "SoftwareDeveloper",
  "gitUserName": "rohan"
}
```


Figure 15: No Password


```
{
  "managerEmail": "admin@hyperperform.me",
  "userName": "rohan",
  "userSurname": "chhipa",
  "userEmail": "rohanchhipa@live.com",
  "userPassword": "1234",
  "role": "Employee",
  "position": "SoftwareDeveloper",
  "gitUserName": ""
}
```


Figure 16: No GitUsername


```
{
  "managerEmail": "admin@hyperperform.me",
  "userName": "rohan",
  "userSurname": "chhipa",
  "userEmail": "rohanchhipa@live.com",
  "userPassword": "1234",
  "role": "Jester",
  "position": "SoftwareDeveloper",
  "gitUserName": "rohan"
}
```


Figure 17: Invalid Role


```
{
  "managerEmail": "admin@hyperperform.me",
  "userName": "rohan",
  "userSurname": "chhipa",
  "userEmail": "rohanchhipa@live.com",
  "userPassword": "1234",
  "role": "Employee",
  "position": "Outside",
  "gitUserName": "rohan"
}
```


Figure 18: Invalid Position