**TESTING**

TABLE OF CONTENTS

[**1**](#_1fob9te) **INTRODUCTION 3**

[**2**](#_49x2ik5) **TESTING MODEL 3**

[**3**](#_tyjcwt) **TESTING TYPES BEING USED 3**

[3.1](#_3dy6vkm) List all the Different Types of tests WHICH you plan to run 3

[3.2](#_1t3h5sf) List the Phases of the LifeCycle and the V&V done for each phase in the project 3

[3.3](#_4d34og8) Set out the Test adequacy criteria for your project 3

[**4**](#_2s8eyo1) **TEST CASES 3**

[4.1](#_17dp8vu) Test case 1 3

[4.2](#_3rdcrjn) Test case 2 3

[4.3](#_26in1rg) Test case 3 3

[4.4](#_lnxbz9) Test case 4 4

[4.5](#_35nkun2) Test case 5 4

[**5**](#_1ksv4uv) **TEST RESULTS 4**

[**6**](#_1pxezwc) **REQUIREMENT TRACEABILITY MATRIX 5**

# INTRODUCTION

The test plan document documents specifies the strategy being followed for testing of the project and sample of the test cases being written for the ConfHub.

# TESTING MODEL

The evaluation model of testing was used during the development phase of the project as each module was analysed to detect faults using review techniques to each module in the early phases of the project. The progress of the project was enhanced by validation (detect faults in requirements and design documents) of these test cases and hence the evaluation model seemed most appropriate for testing.

# TESTING TYPES BEING USED

## List all the Different Types of tests WHICH you plan to run

**1) Unit Testing:**

→ Each individually executable code unit was tested atomically for its syntactic and semantic correctness along with test cases which includes boundary conditions, and the response of these test cases were noted and verified.

→ Each API call in the backend was tested using ***Postman*** and output verified, and the frontend was designed to make the API calls.

→ Database querying was replanned to improve response time of the website along with many other optimizations and decisions involving which technologies to use.

**2) Integration testing:**

→ The components and individual modules from each scrum team had to be integrated and tested to confirm the working of the system as a whole.

→ The backend APIs were integrated with the frontend web application and the APIs were called using frontend web application for testing, and the backend communication with the database was also integrated and tested along with the frontend.

**3) Grey Box testing:**

→ A combination of testing the components and testing the system led to the usage of grey box testing

→ To test the individual APIs and to test the functionality from the user perspective, grey box testing was used.

**4) Static testing:**

→ Static testing was used and the algorithms and semantic correctness of the system was tested using various tools such as Postman and flask

→ Preconditions using system representations along with inspection checklist was used to implement static testing

## List the Phases of the LifeCycle and the V&V done for each phase in the project

**1) Requirements elicitation phase:**

a) During this phase, validation was done by checking if the right feature is being built using techniques such as DFDs and UML class diagrams. Checked whether a particular requirement is meeting its criteria.

b) Verification was done to check if the product will be built right using pseudo code.

**2) Design phase:**

a) Followed the principle of high cohesion and low coupling as strictly as possible by keeping together components which work together.

b) Encapsulation was used to hide the working of a component and show only its interface (Information hiding).

c) System design review was done to validate if the system meets the requirements.

**3) Implementation phase:**

a) Validation on this phase was mostly related to classes and flow of control between them using object flow diagram, which tells whether the object flow diagram was designed to meet the requirements or not.

b) Verification was done by using many techniques such as encapsulation where the working of a component is hidden and only its interfaces is shown (Implemented using classes).

c) Web based HTTP protocol was used along with REST methodology for communication between the server and browser.

d) Validation was also done based on login of the user and sessions using tokens for security.

e) RSS Feeds were implemented to provide the correct data required (Validation)

Implementation was improvised to use periodic refresh and caching to improve  
latency (Verification).

**4) Testing phase:**

a) The overall integrity of the system was tested to complete validation.

b) Non functional requirements such as db access time were improved based on test results for huge datasets.

## Set out the Test adequacy criteria for your project

The adequacy criteria was based on the following:

→ The functionalities were identified and if the functionalities were found to work as expected, testing could be stopped for that phase, in our project, it was the consistent working of the backend server, correct response for each API and error handling for each API call.

→ Important and crucial components (such as retrieving the conference list API) were tested in each phase, and if they satisfied the goal of the component in that phase, testing could be stopped

→ Database access was tried and tested and the response time recorded, testing could be stopped when the response time seemed convincing for usage

# TEST CASES

## TEST CASE 1

* Input validation for user account duplicate
* If a user should not be able to register twice using the same credentials
* Query the DB and check existence of user before updating the db
* SQLALCHEMY wrapper along with python flask
* Alert message telling user is already registered

## TEST CASE 2

* Email notification for a user when he subscribes to a conference
* User should be registered on the database
* Email id should be valid
* A conference must be selected
* A cronjob runs a script that scans the database for details of registered users and sends an email to all the users whose registered conference is in two days from now or less
* CronJob and Python3 with SQLite3 library support for python
* Email notification received.

## TEST CASE 3

* Duplicate entries for email notification(User tries to subscribe more than once)
* User should be registered on the database  
  Email address should be valid  
  A conference must be selected
* Query the database table(used to send emails) to check if the combination of user email id and the conference name exists, if exists the API returns false response using which the frontend takes appropriate action
* Tested using POSTMAN and test data stored in SQLite3 Database
* Alert on the webpage which tells the user that he has already subscribed

## TEST CASE 4

* Duplicate conferences/university programs
* Conference must exist in the database
* Send a POST request to add a conference which already exists
* SQLALCHEMY wrapper for SQLite3 with flask
* Alert message saying (“Conference already exists”)

## TEST CASE 5

* Periodicity and caching of RSS Feeds
* Working RSS Feed
* Reload the webpage and compare latency, add RSS content and wait for it to refresh
* HTML Web Browser
* Improved latency of RSS feeds and No need to reload page every time as it is done implicitly

## 4.6 TEST CASE 6

* Integration testing of website and backend
* Working frontend and backend
* Call the backend APIs using frontend UI
* Backend and Frontend API for Confhub
* Expected output as mentioned in the requirements

# TEST RESULTS

## 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TEST  CASE # | OBJECTIVE/functionality | Test Kind | TEST STATUS | WH BY | WHEN |
| 1 | Disallow duplicate user registration | functionality testing/ white box testing | successful | param p, nihali, | 17/11/2019 |
| 2 2 | email notification | system wide testing/ functionality test /whitebox testing | successful | milan m l, megha k | 17/11/2019 |
| . 3 | multiple subscription | unit testing/white box testing/ input validation | successful | param p, aftab | 17/11/2019 |
| . 4 | duplicate conferences | unit testing/whitebox testing | successful | param p, maanvi | 17/11/2019 |
| . 5 | rss feeds | component testing/unit testing /white box testing | successful | omkar, vivek | 17/11/2019 |
| . 6 | working website | integration testing/blackbox testing/ input validation/ dynamic testing | successful | nihali, nihal, neha m | 17/11/2019 |

# REQUIREMENT TRACEABILITY MATRIX

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sl. No | Req Id | Brief Desc | Arch  Ref  Section | Design  Ref | Code File Ref | Unit Test Cases/ Function and System test cases |
| 1 | 1 | User registration | FR-1 | 2.2.3 | Signup page and DB update file | Postman check |
| 2 | 2 | User login | FR-2 | 2.2.3 | Login and DB update file | Postman check |
| 3 | 3 | User logout | FR-2 | 2.2.3 | Login and DB update file | Postman check |
| 4 | 4 | Presentation of list of conferences | FR-3 | 2.2.7 | View conferences file and REST API | Check with the frontend and unit testing on REST API |
| 5 | 5 | Managing user reminders | FR-3 | 2.2.2 | Subscription file and REST API | Check with the frontend and unit testing on REST API |
| 6 | 6 | Search and filter over conference data | FR-3 | 2.2.5 | View conferences file and REST API | Check with the frontend and unit testing on REST API |
| 7 | 7 | Presentation of conference analytics data | FR-3 | 2.2.8 | TopConf file and REST API | Check with the frontend and unit testing on REST API |
| 8 | 8 | Periodic scraping and cleaning of conference data | FR-3 | 2.2.1 | Scraping folder and cronjob file | Manual Check |