 Alexander Austin

Fred Barnes

CO600: Code Testing 9 March 2016

**Testing Form**

Basic Details

|  |  |  |  |
| --- | --- | --- | --- |
| Item Name: | API testing with Postman | |  |
| Author: | Alexander Austin | | |
| Item Description: | Test the Go lang API by using a Http request tool or similar. | | |
| Project Element: | Frontend: | Backend: X | App: |

Testing Cases

All the test cases use the Postman tool to create the requests. Simply enter the location of where the API is which should be <http://api-anexd.rhcloud.com/> followed by the name field for each test. The Body of the message will then be the payload as seen in the description and should be in a json format unless stated otherwise. You can test to see if the api is running by navigating to the API with /test which should return some json data.

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Description: | Expected Outcome | Actual Outcome |
| newUser (empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| newUser (invalid) | Send a request with the payload as {  "username": "test",  "password": "secret",  "email": [test@tester.test](mailto:test@tester.test)  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user) | The results should send a fail message. Execpt for the case with the extra fields these will be ingnored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash |
| newUser (valid) | Send A payload similar to {  "username": "test",  "password": "secret",  "email": [test@tester.test](mailto:test@tester.test)  }  Where it is in the correct format and also the values do not exist. | This should add the user to the database and return there id | Correctly added the user to the database and returned there id in the response. |
| getUser (empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| getUser (invalid) | Send a request with the payload as {  "email": “[test@tester.test](mailto:test@tester.test)”  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user) | The results should send a fail message. Except for the case with the extra fields these will be ignored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash. |
| getUser (valid) | Send A payload similar to {  "email": “[test@tester.test](mailto:test@tester.test)”  }  Where it is in the correct format you can use the user you created earlier. | This should fetch the data of that user without retrieving there password. | Correctly retrieved that user from the database in response. |
| Login (empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| Login (invalid) | Send a request with the payload as {  "email": “[test@tester.test](mailto:test@tester.test)”  “pass”: “secret”  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user) | The results should send a fail message. Except for the case with the extra fields these will be ignored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash. |
| Login (valid) | Send A payload similar to {  "email": “[test@tester.test](mailto:test@tester.test)”,  “pass”: “secret”  }  Where it is in the correct format you can use the user you created earlier. | This should send a success message if the user information entered is correct otherwise will fail. | Correctly returned success message for that user. |
| changeUserData(empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| changeUserData(invalid) | The URL for this is either changePassword or changeEmail. The payload then should be like follows {  "userID": 1g,  “pasds”: “secret”  “newpass”: “newp”,  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user) | The results should send a fail message. Except for the case with the extra fields these will be ignored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash. |
| changeUserData(valid) | The URL for this is either changePassword or changeEmail. Send A payload similar to {  "userID": 15,  “pass”: “secret”,  “newpass”: “newp”  }  Where it is in the correct format you can use the user you created earlier. | This will send a success message if the user information is correct and then will make the changes to the database. | Correctly changed the relevant information of the user. |
| delUser(empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| delUser(invalid) | The payload then should be like follows {  "userID": 15  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user) | The results should send a fail message. Except for the case with the extra fields these will be ignored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash.  NOTE: probably should also take the user password. |
| delUser(valid) | The payload then should be like follows {  "userID": 15  }  Where it is in the correct format you can use the user you created earlier. | This will send a success message if the user information is correct and then will make the changes to the database. | Correctly removed the user from the database.  NOTE: probably should also take the user password. |
| newLobby(empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| newLobby(invalid) | The payload then should be like follows {  "creator": "test"  "game": "1"  "size": "-25"  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user) and also use creator and games that do not exist. | The results should send a fail message. Except for the case with the extra fields these will be ignored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash. |
| newLobby(valid) | The payload then should be like follows {  "creator": "1",  "game": "1",  "size": "-25"  }  Where it is in the correct format and the Lobby doesn’t exist. | This will send a success message if the lobby information is correct and then will make the changes to the database. | Correctly added the lobby to the database and returned the relevant information in the response. Will also delete and existing lobby by same creator is new request comes in. |
| getLobby(empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| getLobby(invalid) | The payload then should be like follows {  "lobbyID": "test"  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user). | The results should send a fail message. Except for the case with the extra fields these will be ignored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash. |
| getLobby(valid) | The payload then should be like follows {  "lobbyID": "test"  }  Where it is in the correct format you can use the Lobby you created earlier. | This will send a success message if the lobby information is correct and then will get the lobby information. | Correctly fetched the relevant lobby and returned it in the response. |
| changeLobbyData(empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| changeLobbyData(invalid) | The URL for this is either changeLobbyPassword or changeLobbyEmail. Send A payload similar to {  "lobbyID": 15,  “newpass”: “newp”  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user). | The results should send a fail message. Except for the case with the extra fields these will be ignored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash. |
| changeLobbyData(valid) | The URL for this is either changePassword or changeEmail. Send A payload similar to {  "lobbyID": 15,  “newpass”: “newp”  }  Where it is in the correct format you can use the user you created earlier. | This will send a success message if the lobby information is correct and then will make the changes to the database. | Correctly changed the relevant information of the lobby. |
| delLobby(empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| delLobby(invalid) | The payload then should be like follows {  "LobbyID": 15  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user) | The results should send a fail message. Except for the case with the extra fields these will be ignored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash.  NOTE: probably should also take the user password. |
| delLobby(valid) | The payload then should be like follows {  "lobbyID": 15  }  Where it is in the correct format you can use the lobby you created earlier. | This will send a success message if the user information is correct and then will make the changes to the database. | Correctly removed the lobby from the database.  NOTE: probably should also take the user password. |
| newGame(empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| newGame(invalid) | The payload then should be like follows {  "creatorID": "test"  "name": "1",  "type": "4pp"  "des": "a app to play"  "image": "test.bat",  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user) and also use creator and games that do not exist. | The results should send a fail message. Except for the case with the extra fields these will be ignored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash. |
| newGame(valid) | The payload then should be like follows {  "creatorID": "1",  "name": "newgame",  "type": "game",  "des": "a game to play",  "image": "img.jpg"  }  Where it is in the correct format and the game doesn’t exist. | This will send a success message if the game information is correct and then will make the changes to the database. | Correctly added the game to the database and returned the relevant information in the response. |
| changeGameData(empty) | Send a request with the payload as {} | This should not do anything since there are no fields | Correctly displayed the failed request message. |
| changeGameData(invalid) | Send A payload similar to {  "gameID": 15,  “name”: “newname”  "notField": "data"  }  Also try payload with an extra invalid field, incorrect fieldname and values, incorrect JSON, existing data(user). | The results should send a fail message. Except for the case with the extra fields these will be ignored and the rest of the payload processed accordingly. | Correctly displayed Fail messages for all cases except incorrect JSON but did not crash. |
| changeGameData(valid) | Send A payload similar to {  "gameID": 15,  “name”: “newname”,  }  Where it is in the correct format you can use the user you created earlier. | This will send a success message if the game information is correct and then will make the changes to the database. | Correctly changed the relevant information of the game. |
| getAllGames | Send a request with the payload as {} | This should return all the games in the database in the response | Correctly returned all the games in the database. |

Sign Off

|  |  |
| --- | --- |
| Changes Made/ ToDo: | When Error from json parsing send a Fail request message. Delete should require a password of sorts. |
| Signed  Completion: |  |

1