ClubNet
Unit Test Plan
Version 1.0.0

Project team
J.G.C. Brouns | 0856180
S. Chen | 0842556
K. van Eenige | 0862649
S.S. Iyer | 0866094
T.L. Komar | 0870470
D. van der Laan | 0868405
T. Sostak | 0842556
K.W. Verhaegh | 0860736
J. Verhagen | 0816613

Project managers
C.N.I.W. Schappin
N.W. Wielinga

Project supervisor
N. Zannone

Customer
G. Budziak
Abstract

This document contains the unit tests for the ClubNet software system, which is developed by The Brofessionals development team. This document complies with the Software Engineering Standard, as specified by the European Space Agency[3].
# Contents

1 Introduction 6
   1.1 Purpose .............................................. 6
   1.2 Overview ............................................ 6
   1.3 Definitions and abbreviations ......................... 7
      1.3.1 Definitions ..................................... 7
      1.3.2 Abbreviations .................................... 7
   1.4 List of references .................................... 7

2 Test Plan 8
   2.1 Test Items ........................................... 8
   2.2 Features To Be Tested .................................. 8
   2.3 Test Deliverables ..................................... 8
   2.4 Testing Tasks ......................................... 8
   2.5 Environmental Needs .................................. 9
   2.6 Test Case Pass/Fail Criteria .......................... 9

3 Test Case Specifications 10
   3.1 Structure and Example ................................ 10
      3.1.1 Structure ......................................... 10
      3.1.2 Example Test Group ............................... 10
   3.2 Server Side Tests ..................................... 11
      3.2.1 Feed Methods .................................... 11
      3.2.2 Item Types Methods ............................... 22
      3.2.3 User Accounts Methods ........................... 23
      3.2.4 Access Control Methods .......................... 30
      3.2.5 Club Methods .................................... 47

4 Test Procedure 49
   4.1 Structure and Example ................................. 49
      4.1.1 Structure ......................................... 49
      4.1.2 Procedure ........................................ 49

5 Test Coverage 50

6 Test Reports 52
# DOCUMENT STATUS SHEET

## GENERAL

- **Document title:** Unit Test Plan v1.0.0  
- **Identification:** UTP/1.0.0  
- **Authors:** K. van Eenige, K. Verhaegh, J. Verhagen, S.S. Iyer  
- **Document status:** Draft version

## DOCUMENT HISTORY

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.1</td>
<td>11-05-2016</td>
<td>K. van Eenige, K. Verhaegh</td>
<td>Initial document structure</td>
</tr>
<tr>
<td>0.0.2</td>
<td>6-06-2016</td>
<td>K. van Eenige, J. Verhagen</td>
<td>Restructure</td>
</tr>
<tr>
<td>0.1.0</td>
<td>7-06-2016</td>
<td>K. van Eenige, J. Verhagen</td>
<td>Adding more unit tests</td>
</tr>
<tr>
<td>0.1.1</td>
<td>8-06-2016</td>
<td>K. van Eenige, J. Verhagen</td>
<td>Adding more unit tests</td>
</tr>
<tr>
<td>0.1.2</td>
<td>9-06-2016</td>
<td>K. van Eenige, J. Verhagen</td>
<td>Restructure</td>
</tr>
<tr>
<td>0.1.2</td>
<td>20-06-2016</td>
<td>K. van Eenige, S.S. Iyer</td>
<td>Restructure and adding more tests</td>
</tr>
<tr>
<td>1.0.0</td>
<td>30-06-2016</td>
<td>S.S. Iyer</td>
<td>Added much more tests, Modified all test cases, Reduced number of test procedures, Added coverage and report</td>
</tr>
</tbody>
</table>
## DOCUMENT CHANGE RECORDS

<table>
<thead>
<tr>
<th>Section</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Revised the test procedure</td>
</tr>
<tr>
<td>5</td>
<td>Added test coverage</td>
</tr>
<tr>
<td>6</td>
<td>Added test reports</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 PURPOSE

This is the Unit Test Plan (UTP) of the ClubNet application. This document will describe the test plan for doing the unit testing of developed units. Each unit will be tested and the results will be verified with the DDD[2]. The results should be as described in this document and in accordance with the DDD.

1.2 OVERVIEW

The rest of this document comprises of the test procedure and actual unit tests, as well as the test reports together with any encountered problems. Chapter 2 gives an overview of all items to be tested and the criteria for each UT. Chapter 3 specifies how the tests are defined, including their input and output specifications. Chapter 4 lists the test procedure(s), and chapter 5 describes the actual results and how test results are reported.
1.3 DEFINITIONS AND ABBREVIATIONS

1.3.1 DEFINITIONS

Brofessionals | The development team of ClubNet
CoachAssist | An independent software system developed by Intuitive Technologies B.V. [4]
Intuitive Technologies B.V. | A software engineering company situated in the Netherlands serving the role of client.

1.3.2 ABBREVIATIONS

ADD | Architectural Design Document
ATP | Acceptance Test Plan
URD | User Requirements Document
UTP | Unit Test Plan
ITP | Integration Test Plan
DDD | Detailed Design Document

1.4 LIST OF REFERENCES

References

2 TEST PLAN

2.1 TEST ITEMS
The software to be tested is the ClubNet application, the ClubNet web interface, and the ClubNet back end. Information about the detailed design of ClubNet can be found in the DDD[2].

2.2 FEATURES TO BE TESTED
ClubNet must meet the design as stated in the DDD[2]. Each component should adhere to the interfaces given in the DDD[2].

2.3 TEST DELIVERABLES
Before testing starts, the following documents must be delivered:

- DDD[2]
- UTP (this document)
- UT input data

After completing the tests, the following documents must be delivered:

- UT report (Chapter 5 of this document)
- UT output data
- Problem reports (if any)

2.4 TESTING TASKS
Before any testing in the UT phase can take place, the following tasks need to be done:

- Identifying components from DDD[2] that needs to be tested.
- Designing the unit tests
- Creation of the UT input data, eg.: setting up of database with fake data necessary for testing.
- Ensuring that all environmental needs for the UT have been satisfied, eg.: stubbing methods needed for testing for isolating test cases.

When these tasks have been done, a UT can be performed according to the procedures described in chapter 4.
2.5 **ENVIRONMENTAL NEEDS**

The following resources are needed in order to perform the UT, after setting up the environment:

- Testing framework for Meteor: practicalmocha
- Test runner, Test logger: practicalmocha
- A PC capable of running the (Mocha) testing framework and ClubNet.

2.6 **TEST CASE PASS/FAIL CRITERIA**

Every test should describe the criteria that should be met to pass a specific test. The overall UT pass can only be achieved when all tests described in chapter 3 have been performed and passed.
3 TEST CASE SPECIFICATIONS

3.1 STRUCTURE AND EXAMPLE

The tests that are implemented will be described in the following section. Within this section there will be subsections that further divide the tests into logical groups. These tests are described with the following structure

3.1.1 STRUCTURE

Test case identifier
UT<number> of unit that is being tested

Test items
The function/service/controller/code to be tested

Input specifications
What to provide to the function in the test

Output specifications
Expected/needed result of each test

Environmental needs
Special environmental needs for this test

3.1.2 EXAMPLE TEST GROUP

UTO

Test Items exampleFunction()

Input specifications

1. exampleValue: 35
2. exampleValue:"asdf"

Output specifications

1. true
2. Error

Environmental needs
Example function in examplefile.js
3.2 SERVER SIDE TESTS

3.2.1 FEED METHODS

UTS1

Test Items addFeedItem()
Adding feed item should fail adding a feed item without item type (invalid parameter).

Input specifications An item to insert with missing item type,

Output specifications Error: Schema error, missing type

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- A user is logged in.

UTS2

Test Items addFeedItem()
Adding feed item should succeed.


Output specifications Object ID of the added item

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- A user is logged in.

UTS3

Test Items getFeedItem()
Getting a feed item with wrong parameter.

Input specifications itemId: false

Output specifications Error: Match Failed

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- A user must be logged in.
UTS4

**Test Items** getFeedItem()

Getting a feed item successfully.

**Input specifications** itemId: The Object ID of the added item.

**Output specifications** The item object

**Environmental needs**

- The test case is stored in the following file: Feed.tests.js
- An item with _id = itemId inserted into the database that can be retrieved.
- A user must be logged in.

UTS5

**Test Items** updateFeedItem()

Updating a feed item with non-Object input.

**Input specifications** update: false

**Output specifications** Error: Match Failed

**Environmental needs**

- The test case is stored in the following file: Feed.tests.js
- A user must be logged in.

UTS6

**Test Items** updateFeedItem()

Updating a feed item with incorrect creatorID.

**Input specifications** update: { type: 'Voting', sticky: false }

**Output specifications** Error: Not Authorized

**Environmental needs**

- The test case is stored in the following file: Feed.tests.js
- Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
- A user must be logged in with userId: 'someoneElse'.

UTS7

**Test Items** updateFeedItem()

Updating a feed item clubID/creatorID/type/createdAt.
Input specifications update: { type: 'Voting', clubID: '2' }

Output specifications Error: Bad Request

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
- A user must be logged in with userId: 'me'.

UTS8

Test Items updateFeedItem()
Updating a feed item successfully.

Input specifications update: { type: 'Voting', sticky: true }

Output specifications updatedItem with sticky: true

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
- A user must be logged in with userId: 'me'.

UTS9

Test Items putResponse()
Putting a response to a feed item with non-String item ID.

Input specifications response: { itemId: false, itemType: 'Voting', value: '1' }

Output specifications Error: Match Failed

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
- A user must be logged in with userId: 'me'.

UTS10

Test Items putResponse()
Putting a response to a feed item with non-String item type.

Input specifications response: { itemId: 'itemId', itemType: false, value: '1' }

Output specifications Error: Match Failed

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- Insert item: { _id: 'itemId', creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
- A user must be logged in with userId: 'me'.

UTS11

Test Items putResponse()
Putting a response to a feed item with non-String itemId

Input specifications response: { itemId: 'itemId', itemType: 'Voting', value: false }

Output specifications Error: Match Failed

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- Insert item: { _id: 'itemId', creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
- A user must be logged in with userId: 'me'.

UTS12

Test Items putResponse()
Putting a response to a feed item successfully.

Input specifications response: { itemId: 'itemId', itemType: 'Voting', value: '1' }

Output specifications Object ID of inserted Response.

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
• A user must be logged in with userId: ‘me’.

**UTS13**

**Test Items** `getResponse()`  
Getting a response with non-String item ID.  
**Input specifications** Item ID of item of which to get responses, `itemId`: false  
**Output specifications** Error: Match Failed.  

**Environmental needs**

- The test case is stored in the following file: `Feed.tests.js`  
- Insert item: `{ creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' }` into the database.  
- Insert response: `{ itemId: 'itemId', itemType: 'Voting', value: '1' }` into the database.  
- A user must be logged in with userId: ‘me’.

**UTS14**

**Test Items** `getResponse()`  
Getting a response successfully.  
**Input specifications** Item ID of item of which to get responses, `itemId`: 'itemId'  
**Output specifications** Response Object in the database, `response`.  

**Environmental needs**

- The test case is stored in the following file: `Feed.tests.js`  
- Insert item: `{ creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' }` into the database.  
- Insert response: `{ itemId: 'itemId', itemType: 'Voting', value: '1' }` into the database.  
- A user must be logged in with userId: ‘me’.

**UTS15**

**Test Items** `getResponsesOfOneItem()`  
Getting all response of one item with non-String item ID.  
**Input specifications** Item ID of item of which to get responses, `itemId`: false  
**Output specifications** Error: Match Failed  

**Environmental needs**
July 1, 2016

- The test case is stored in the following file: Feed.tests.js

- Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.

- Insert response: { itemId: 'itemId', itemType: 'Voting', value: '1' } into the database.

- A user must be logged in with userId: 'me'.

**UTS16**

**Test Items** `getResponsesOfOneItem()`  
Getting all responses of non-existing item.  
**Input specifications** Item ID of item of which to get responses, itemId: 'someOtherItem'  
**Output specifications** Empty Responses array: []  
**Environmental needs**

- The test case is stored in the following file: Feed.tests.js

- Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.

- Insert response: { itemId: 'itemId', itemType: 'Voting', value: '1' } into the database.

- A user must be logged in with userId: 'me'.

**UTS17**

**Test Items** `getResponsesOfOneItem()`  
Getting all response of one item successfully.  
**Input specifications** Item ID of item of which to get responses, itemId: 'itemId'  
**Output specifications** Responses array: [{ itemId: 'itemId', itemType: 'Voting', value: '1' }]  
**Environmental needs**

- The test case is stored in the following file: Feed.tests.js

- Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.

- Insert response: { itemId: 'itemId', itemType: 'Voting', value: '1' } into the database.

- A user must be logged in with userId: 'me'.

UTS18

Test Items getResponsesOfOneItem()
Getting count of all response of an item with non-String item ID.

Input specifications Item ID of item of which to get responses, itemId: false

Output specifications Error: Match Failed

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
- Insert response: { itemId: 'itemId', itemType: 'Voting', value: '1' } into the database.
- A user must be logged in with userId: 'me'.

UTS19

Test Items getResponsesOfOneItem()
Getting count of all responses of non-existing item.

Input specifications Item ID of item of which to get responses, itemId: 'someOtherItem'

Output specifications Responses count: 0

Environmental needs

- The test case is stored in the following file: Feed.tests.js
- Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
- Insert response: { itemId: 'itemId', itemType: 'Voting', value: '1' } into the database.
- A user must be logged in with userId: 'me'.

UTS20

Test Items getNumberResponsesOfOneItem()
Get count of all responses of one item successfully.

Input specifications Item ID of item of which to get responses, itemId: 'itemId'

Output specifications Responses count: 1

Environmental needs

- The test case is stored in the following file: Feed.tests.js
• Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.

• Insert response: { itemId: 'itemId', itemType: 'Voting', value: '1' } into the database.

• A user must be logged in with userId: 'me'.

UTS21

Test Items getResponsesOfItemType()
Getting all responses of items of given item type with wrong parameters.
Input specifications itemType: false
Output specifications Error: Match Failed
Environmental needs

• The test case is stored in the following file: Feed.tests.js

• Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.

• Insert response: { itemId: 'itemId', itemType: 'Voting', value: '1' } into the database.

• A user must be logged in with userId: 'me'.

UTS22

Test Items getResponsesOfItemType()
Getting all responses of items of non-existing item type.
Input specifications itemType: 'someOtherType'
Output specifications Responses array: []
Environmental needs

• The test case is stored in the following file: Feed.tests.js

• Insert item: { creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.

• Insert response: { itemId: 'itemId', itemType: 'Voting', value: '1' } into the database.

• A user must be logged in with userId: 'me'.

TU/e ClubNet | Unit Test Plan 18
UTS23

**Test Items** getResponsesOfItemType()
Getting all responses of items of given item type successfully.

**Input specifications** itemType: 'Voting'

**Output specifications** Responses array: ```[[ itemId: 'itemId', itemType: 'Voting', value: '1' ]]```  

**Environmental needs**

- The test case is stored in the following file: Feed.tests.js
- Insert item: `{ _id: 'itemId', creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', created: new Date, modified: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' }` into the database.
- Insert response: `{ itemId: 'itemId', itemType: 'Voting', value: '1' }` into the database.
- A user must be logged in with userId: 'me'.

UTS24

**Test Items** getVotingResults()
Getting a voting results of an item with wrong parameters.

**Input specifications** Voting itemId: false

**Output specifications** Error: Match Failed

**Environmental needs**

- The test case is stored in the following file: Feed.tests.js
- Insert item: `{ _id: 'itemId', creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', created: new Date, modified: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' }` into the database.
- Insert response: `{ itemId: 'itemId', itemType: 'Voting', value: '1' }` into the database.
- A user must be logged in with userId: 'me'.

UTS25

**Test Items** getVotingResults()
Getting voting results of an item with one response (vote).

**Input specifications** String ID of Voting item of which to get results, itemId: '1'.

**Output specifications** Voting result: ```[[1, 0, 0]]```  

**Environmental needs**

- The test case is stored in the following file: Feed.tests.js
• Insert item: { _id: '1', creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.

• Insert response: { itemId: '1', itemType: 'Voting', value: '1' } into the database.

• A user must be logged in with userId: 'me'.

**UTS26**

**Test Items getVotingResults()**  
Getting voting results of an item with multiple responses (votes).  
**Input specifications** String ID of Voting item of which to get results, itemId: '1'.  
**Output specifications** Voting result: [[2, 1, 0]]  
**Environmental needs**

• The test case is stored in the following file: Feed.tests.js

• Insert item: { _id: '1', creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.

• Insert response: { itemId: '1', itemType: 'Voting', value: '1' } into the database.

• Insert another response: { itemId: '1', itemType: 'Voting', value: '1' } into the database.

• Insert another response: { itemId: '1', itemType: 'Voting', value: '2' } into the database.

• A user must be logged in with userId: 'me'.

**UTS27**

**Test Items deleteResponse()**  
Deleting only response of currently logged-in user from item of invalid item Id.  
**Input specifications** String ID of item, itemId: false  
**Output specifications** Error: Match Failed  
**Environmental needs**

• The test case is stored in the following file: Feed.tests.js

• Insert item: { _id: '1', creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.

• Insert response: { itemId: '1', itemType: 'Voting', value: '1' } into the database.

• A user must be logged in with userId: 'me'.
UTS28

**Test Items** deleteResponse()
Deleting only response of currently logged-in user from item of given item Id.

**Input specifications** String ID of item, itemId: '1'

**Output specifications** None but response deleted

**Environmental needs**

- The test case is stored in the following file: Feed.tests.js
- Insert item: { _id: '1', creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
- Insert response: { itemId: '1', itemType: 'Voting', value: '1' } into the database.
- A user must be logged in with userId: 'me'.

UTS29

**Test Items** deleteFeedItem()
Deleting a feed item with non-String item Id.

**Input specifications** itemId: false

**Output specifications** Error: Match Failed.

**Environmental needs**

- The test case is stored in the following file: Feed.tests.js
- Insert item: { _id: '1', creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' } into the database.
- A user must be logged in with userId: 'me'.

UTS30

**Test Items** deleteFeedItem()
Deleting a feed item successfully.

**Input specifications** String ID of existing item, itemId: '1'

**Output specifications** Deleted item 'item'. Count of items on database with itemId: '1' must be 0.

**Environmental needs**

- The test case is stored in the following file: Feed.tests.js
• Insert item: `{ _id: '1', creatorID: 'me', sticky: false, clubID: '1', type: 'Voting', createdAt: new Date, modifiedAt: new Date, title: '1', status: 'published', deadline: new Date, training_id: '1', teamID: '1' }` into the database.

• A user must be logged in with userId: 'me'.

**UTS31**

**Test Items** `getTrainings()`
Getting the trainings.

**Input specifications** None

**Output specifications** An array of trainings. May be empty if there are no trainings.

**Environmental needs**

• The test case is stored in the following file: Feed.tests.js

**UTS32**

**Test Items** `getExercises()`
Getting exercises of trainings with invalid training ID.

**Input specifications** trainingID: false

**Output specifications** Error: Match Failed

**Environmental needs**

• The test case is stored in the following file: Feed.tests.js

**UTS33**

**Test Items** `getExercises()`
Getting exercises of trainings with valid training ID.

**Input specifications** trainingID: false

**Output specifications** None, if there are no trainings with given training ID.
Array of exercises, if training found.

**Environmental needs**

• The test case is stored in the following file: Feed.tests.js

### 3.2.2 ITEM TYPES METHODS

**UTS34**

**Test Items** `utils.getAllItemTypes()`
Getting all the item types.

**Input specifications**

1. None
Output specifications

1. [itemType1, itemType2 ]

Environmental needs

- The test case is stored in the following file: ItemTypes.tests.js
- Insert itemType1: { _id: '1', name: 'testType', icon: 'testType.ClubNet' };
- Insert itemType2: { _id: '2', name: 'testType2', icon: 'testType2.ClubNet' };

UTS35

Test Items getItemType()
Getting a single item type with non-String type ID.

Input specifications typeID: false

Output specifications Error: Match Failed

Environmental needs

- The test case is stored in the following file: ItemTypes.tests.js
- Insert itemType1: { _id: '1', name: 'testType', icon: 'testType.ClubNet' };
- Insert itemType2: { _id: '2', name: 'testType2', icon: 'testType2.ClubNet' };

UTS36

Test Items getItemType()
Getting a single item type successfully.

Input specifications typeID: '1'

Output specifications itemType1

Environmental needs

- The test case is stored in the following file: ItemTypes.tests.js
- Insert itemType1: { _id: '1', name: 'testType', icon: 'testType.ClubNet' };
- Insert itemType2: { _id: '2', name: 'testType2', icon: 'testType2.ClubNet' };

3.2.3 USER ACCOUNTS METHODS

UTS37

Test Items addUser()
Adding a user without all attributes eg.: firstName

Input specifications user: { email: 'test@test.test', password: 'test', profile: { lastName: 'Test', type: 'player', clubID: 'test', teamID: 'test', notifications: {} } }
Output specifications Error: Missing firstName

Environmental needs

- The test case is stored in the following file: UserAccount.tests.js
- Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: [] } } into the database.
- Login with testPr.

UTS38

Test Items addUser()
Adding a user successfully.

Input specifications user: { email: 'test@test.test', password: 'test', profile: { firstName: 'Test', lastName: 'Test', type: 'player', clubID: 'test', teamID: 'test', notifications: [] } }

Output specifications String ID of inserted user, userId.

Environmental needs

- The test case is stored in the following file: UserAccount.tests.js
- Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: [] } } into the database.
- Login with testPr.

UTS39

Test Items updateUserProfile()
Updating a user profile with incorrect information.

Input specifications

1. userId: String ID of testUser
2. User profile information with incorrect lastName attribute, testProfile: { firstName: 'Test', lastName: 14, type: 'player', clubID: 'test', notifications: } }

Output specifications Error: lastName must be String

Environmental needs

- The test case is stored in the following file: UserAccount.tests.js
- Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: [] } } into the database.
• Login with testPr.

**UTS40**

**Test Items** `updateUserProfile()`  
Updating a user profile with unauthorized non-PR logged-in user.  

**Input specifications**

1. `userId`: String ID of `testUser`  
2. User profile information, `testProfile`: `{ firstName: 'Test', lastName: 'newTest', type: 'player', clubID: 'test', notifications: {} }`

**Output specifications** Error: Not Authorized

**Environmental needs**

- The test case is stored in the following file: `UserAccount.tests.js`
- Insert PR `testPr`: `{ email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: [] } }` into the database.
- Login with any user other than `testUser` and any PR user.

**UTS41**

**Test Items** `updateUserProfile()`  
Updating a user profile with unauthorized PR user of different club.  

**Input specifications**

1. `userId`: String ID of `testUser`  
2. User profile information, `testProfile`: `{ firstName: 'Test', lastName: 'newTest', type: 'player', clubID: 'test', notifications: [] }`

**Output specifications** Error: Not Authorized

**Environmental needs**

- The test case is stored in the following file: `UserAccount.tests.js`
- Insert PR `testPr`: `{ email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'someOtherClub', notifications: [] } }` into the database.
• Insert a player user, testUser: { email: 'test@test.test', password: 'test', profile: { firstName: 'Test', lastName: 'Test', type: 'player', clubID: 'test', teamID: 'test', notifications: {} } } into the database.

• Login with testPr user.

UTS42

Test Items update UserProfile()
Updating a user profile with correct information.

Input specifications

1. userId: String ID of testUser

2. User profile information, testProfile: { firstName: 'Test', lastName: 'newTest', type: 'player', clubID: 'test', notifications: {} }

Output specifications None but implies user profile was updated correctly.

Environmental needs

• The test case is stored in the following file: UserAccount.tests.js

• Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: {} } } into the database.

• Insert a player user, testUser: { email: 'test@test.test', password: 'test', profile: { firstName: 'Test', lastName: 'Test', type: 'player', clubID: 'test', teamID: 'test', notifications: {} } } into the database.

• Login with testPr.

UTS43

Test Items getUserInfo()
Getting user information with non-String user ID.

Input specifications userId: 1234

Output specifications Error: Match Failed

Environmental needs

• The test case is stored in the following file: UserAccount.tests.js

• Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: {} } } into the database.

• Insert a player user, testUser: { email: 'test@test.test', password: 'test', profile: { firstName: 'Test', lastName: 'Test', type: 'player', clubID: 'test', teamID: 'test', notifications: {} } } into the database.
• Login with testPr.

**UTS44**

**Test Items** getUserInfo()
Getting user information with non-existing user ID.

**Input specifications** userId: 'someoneElse'

**Output specifications** Resulting user is empty, result: undefined

**Environmental needs**

• The test case is stored in the following file: UserAccount.tests.js

• Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: [ ] } } into the database.

• Insert a player user, testUser: { _id: 'me', email: 'test@test.test', password: 'test', profile: { firstName: 'Test', lastName: 'Test', type: 'player', clubID: 'test', teamID: 'test', notifications: [ ] } } into the database.

• Login with testPr.

**UTS45**

**Test Items** getUserInfo()
Getting user information with unauthorized PR user from different club.

**Input specifications** userId: 'me'

**Output specifications** Error: Not Authorized

**Environmental needs**

• The test case is stored in the following file: UserAccount.tests.js

• Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'someOtherClub', notifications: [ ] } } into the database.

• Insert a player user, testUser: { _id: 'me', email: 'test@test.test', password: 'test', profile: { firstName: 'Test', lastName: 'Test', type: 'player', clubID: 'test', teamID: 'test', notifications: [ ] } } into the database.

• Login with testPr user.

**UTS46**

**Test Items** getUserInfo()
Getting user information successfully.

**Input specifications** userId: 'me'

**Output specifications** User Object

**Environmental needs**
• The test case is stored in the following file: UserAccount.tests.js

• Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: [] } } into the database.

• Insert a player user, testUser: { _id: 'me', email: 'test@test.test', password: 'test', profile: { firstName: 'Test', lastName: 'Test', type: 'player', clubID: 'test', teamID: 'test', notifications: [] } } into the database.

• Login with testPr user.

UTS47

Test Items getUserType(
Getting user type of currently logged-in user with invalid parameters.

Input specifications userId: 1234

Output specifications Error: Match Failed

Environmental needs

• The test case is stored in the following file: UserAccount.tests.js

• Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: [] } } into the database.

• Login with testPr user.

UTS48

Test Items getUserType()
Getting user type of currently logged-in user.

Input specifications None

Output specifications userType: 'pr'

Environmental needs

• The test case is stored in the following file: UserAccount.tests.js

• Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: [] } } into the database.

• Login with testPr user.

UTS49

Test Items getTeamSize()
Getting team size of currently logged-in PR user’s team.

Input specifications None

Output specifications Team size: 0 because PR users are not part of teams.

Environmental needs
The test case is stored in the following file: UserAccount.tests.js

Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: [] } } into the database.

Login with testPr user.

UTS50

Test Items getTeamSize()
Getting team size of currently logged-in player user’s team.
Input specifications None
Output specifications Team size: 1 because currently logged-in user is the only user with teamID: ‘test’.
Environmental needs

The test case is stored in the following file: UserAccount.tests.js


Login with testUser user.

UTS51

Test Items Meteor.users.remove()
Removing a user
Input specifications userId: ‘me’
Output specifications getUserInfo() with userId: ‘me’ returns undefined.
Environmental needs

The test case is stored in the following file: UserAccount.tests.js


Insert PR testPr: { email: 'pr@pr.pr', password: 'pr', profile: { firstName: 'Pr', lastName: 'Pr', type: 'pr', clubID: 'test', notifications: [] } } into the database.

Login with testPr user.
3.2.4 ACCESS CONTROL METHODS

UTS52

Test Items setPermissions()

PR user can set permissions

Input specifications

\[
\text{testPermissions} = \{ \text{id: 'pr', items: [[\text{id: 'testType', permissions: \{create: true, edit: true, view: true, delete: true\}}]]}\}
\]

Output specifications
None

Environmental needs:

- The test case is stored in the following file: AccessControlInsert.tests.js
- A PR user is logged in
- Insert a test item type, testType: \{_id: 'testType', name: 'testType', icon: 'testType.ClubNet'

UTS53

Test Items setPermissions()

Player user cannot set permissions

Input specifications

\[
\text{testPermissions} = \{ \text{id: 'pr', items: [[\text{id: 'testType', permissions: \{create: true, edit: true, view: true, delete: true\}}]]}\}
\]

Output specifications
Error: Match Failed

Environmental needs:

- The test case is stored in the following file: AccessControlInsert.tests.js
- A Player user is logged in
- Insert a test item type, testType: \{_id: 'testType', name: 'testType', icon: 'testType.ClubNet'

UTS54

Test Items setPermissions()

Coach user cannot set permissions

Input specifications

\[
\text{testPermissions} = \{ \text{id: 'pr', items: [[\text{id: 'testType', permissions: \{create: true, edit: true, view: true, delete: true\}}]]}\}
\]

Output specifications
Error: Match Failed

Environmental needs:

- The test case is stored in the following file: AccessControlInsert.tests.js
- A Coach user is logged in
- Insert a test item type, testType: { _id: ‘testType’, name: ‘testType’, icon: ‘testType.ClubNet’ } in the database.

**UTS55**

**Test Items** setPermissions()

General user cannot set permissions

**Input specifications** testPermissions = { id: ‘pr’, items: [{id: ‘testType’, permissions: {create: true, edit: true, view: true, delete: true}}]}

**Output specifications** Error: Match Failed

**Environmental needs:**

- The test case is stored in the following file: AccessControlInsert.tests.js
- A General user is logged in
- Insert a test item type, testType: { _id: ‘testType’, name: ‘testType’, icon: ‘testType.ClubNet’ } in the database.

**UTS56**

**Test Items** checkRights()

PR user should not be able to create a voting item

**Input specifications**

1. itemType: ’Voting’

2. requestedPermission: ’create’

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlVoting.tests.js
- A PR user is logged in.
- Voting permissions for PR user have been set, testControlPr = { _id: ‘pr’, items: [{id: ‘Voting’, permissions: {create: false, edit: false, view: false, delete: false}}]}

**UTS57**

**Test Items** checkRights()

PR user should not be able to edit a voting item

**Input specifications**
1. itemType: 'Voting'
2. requestedPermission: 'edit'

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlVoting.tests.js
- A PR user is logged in.
- Voting permissions for PR user have been set, testControlPr = { _id: 'pr', items: [[id: 'Voting', permissions: {create: false, edit: false, view: false, delete: false}]] }

**UTS58**

**Test Items** checkRights()
PR user should not be able to see a voting item

**Input specifications**

1. itemType: 'Voting'
2. requestedPermission: 'view'

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlVoting.tests.js
- A PR user is logged in.
- Voting permissions for PR user have been set, testControlPr = { _id: 'pr', items: [[id: 'Voting', permissions: {create: false, edit: false, view: false, delete: false}]] }

**UTS59**

**Test Items** checkRights()
PR user should not be able to delete a voting item

**Input specifications**

1. itemType: 'Voting'
2. requestedPermission: ‘delete’

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlVoting.tests.js
- A PR user is logged in.
- Voting permissions for PR user have been set, testControlPr = `{ _id: 'pr', items: [{id: 'Voting', permissions: {create: false, edit: false, view: false, delete: false}}] }`

---

**UTS60**

**Test Items** checkRights()

Player user should not be able to create a voting item

**Input specifications**

1. itemType: ‘Voting’
2. requestedPermission: ‘create’

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlVoting.tests.js
- A player user is logged in.
- Voting permissions for player user have been set, testControlPlayer = `{ _id: 'player', items: [{id: 'Voting', permissions: {create: false, edit: false, view: true, delete: false}}] }`

---

**UTS61**

**Test Items** checkRights()

Player user should not be able to edit a voting item

**Input specifications**

1. itemType: ‘Voting’
2. requestedPermission: ‘edit’
Output specifications  permission: false

Environmental needs:

• The test case is stored in the following file: AccessControlVoting.tests.js
• A player user is logged in.
• Voting permissions for player user have been set, testControlPlayer = { _id: 'player', items: [{id: 'Voting', permissions: {create: false, edit: false, view: true, delete: false}}]}

UTS62

Test Items checkRights()
Player user should be able to see a voting item

Input specifications

1. itemType: 'Voting'
2. requestedPermission: 'view'

Output specifications  permission: true

Environmental needs:

• The test case is stored in the following file: AccessControlVoting.tests.js
• A player user is logged in.
• Voting permissions for player user have been set, testControlPlayer = { _id: 'player', items: [{id: 'Voting', permissions: {create: false, edit: false, view: true, delete: false}}]}

UTS63

Test Items checkRights()
Player user should not be able to delete a voting item

Input specifications

1. itemType: 'Voting'
2. requestedPermission: 'delete'

Output specifications  permission: false

Environmental needs:
• The test case is stored in the following file: AccessControlVoting.tests.js

• A player user is logged in.


• Voting permissions for player user have been set, testControlPlayer = `{ _id: 'player', items: [{id: 'Voting', permissions: {create: false, edit: false, view: true, delete: false}}]}`

UTS64

Test Items checkRights()
Coach user should be able to create a voting item

Input specifications

1. itemType: 'Voting'

2. requestedPermission: 'create'

Output specifications permission: true

Environmental needs:

• The test case is stored in the following file: AccessControlVoting.tests.js

• A coach user is logged in.


• Voting permissions for coach user have been set, testControlCoach = `{ _id: 'coach', items: [{id: 'Voting', permissions: {create: true, edit: true, view: true, delete: true}}]}`

UTS65

Test Items checkRights()
Coach user should be able to edit a voting item

Input specifications

1. itemType: 'Voting'

2. requestedPermission: 'edit'

Output specifications permission: true

Environmental needs:

• The test case is stored in the following file: AccessControlVoting.tests.js
• A coach user is logged in.


• Voting permissions for coach user have been set, testControlCoach = { _id: 'coach', items: [[id: 'Voting', permissions: {create: true, edit: true, view: true, delete: true}]]}

UTS66

Test Items checkRights()
Coach user should be able to see a voting item
Input specifications

1. itemType: 'Voting'

2. requestedPermission: 'view'

Output specifications permission: true

Environmental needs:

• The test case is stored in the following file: AccessControlVoting.tests.js

• A coach user is logged in.


• Voting permissions for coach user have been set, testControlCoach = { _id: 'coach', items: [[id: 'Voting', permissions: {create: true, edit: true, view: true, delete: true}]]}

UTS67

Test Items checkRights()
Coach user should be able to delete a voting item
Input specifications

1. itemType: 'Voting'

2. requestedPermission: 'delete'

Output specifications permission: true

Environmental needs:

• The test case is stored in the following file: AccessControlVoting.tests.js

• A coach user is logged in.

• Voting permissions for coach user have been set, `testControlCoach = { _id: 'coach', items: [[id: 'Voting', permissions: {create: true, edit: true, view: true, delete: true}]]}`

**UTS68**

**Test Items** `checkRights()`

General user should not be able to create a voting item

**Input specifications**

1. `itemType: 'Voting'`

2. `requestedPermission: 'create'`

**Output specifications** `permission: false`

**Environmental needs:**

• The test case is stored in the following file: `AccessControlVoting.tests.js`

• A general user is logged in.


• Voting permissions for general user have been set, `testControlGeneral = { _id: 'general', items: [[id: 'Voting', permissions: {create: false, edit: false, view: false, delete: false}]]} `

**UTS69**

**Test Items** `checkRights()`

General user should not be able to edit a voting item

**Input specifications**

1. `itemType: 'Voting'`

2. `requestedPermission: 'edit'`

**Output specifications** `permission: false`

**Environmental needs:**

• The test case is stored in the following file: `AccessControlVoting.tests.js`

• A general user is logged in.

Voting permissions for general user have been set, testControlGeneral = { _id: 'general', items: [[id: 'Voting', permissions: {create: false, edit: false, view: false, delete: false}]]}

**UTS70**

**Test Items** checkRights()
General user should not be able to see a voting item

**Input specifications**

1. itemType: 'Voting'
2. requestedPermission: 'view'

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlVoting.tests.js
- A general user is logged in.
- Voting permissions for general user have been set, testControlGeneral = { _id: 'general', items: [[id: 'Voting', permissions: {create: false, edit: false, view: false, delete: false}]]}

**UTS71**

**Test Items** checkRights()
General user should not be able to delete a voting item

**Input specifications**

1. itemType: 'Voting'
2. requestedPermission: 'delete'

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlVoting.tests.js
- A general user is logged in.
- Voting permissions for general user have been set, testControlGeneral = { _id: 'general', items: [[id: 'Voting', permissions: {create: false, edit: false, view: false, delete: false}]]}
UTS72

**Test Items** checkRights()
PR user should not be able to create a form item

**Input specifications**

1. `itemType: 'Form'`
2. `requestedPermission: 'create'`

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlForm.tests.js
- A PR user is logged in.
- Insert a form type, `itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }` in the database.
- Form permissions for PR user have been set, `testControlPr = { _id: 'pr', items: [{id: 'Form', permissions: {create: false, edit: false, view: false, delete: false}}]}`

UTS73

**Test Items** checkRights()
PR user should not be able to edit a form item

**Input specifications**

1. `itemType: 'Form'`
2. `requestedPermission: 'edit'`

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlForm.tests.js
- A PR user is logged in.
- Insert a form type, `itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }` in the database.
- Form permissions for PR user have been set, `testControlPr = { _id: 'pr', items: [{id: 'Form', permissions: {create: false, edit: false, view: false, delete: false}}]}`
UTS74

**Test Items** `checkRights()`

PR user should not be able to see a form item

**Input specifications**

1. `itemType: 'Form'`
2. `requestedPermission: 'view'`

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: `AccessControlForm.tests.js`
- A PR user is logged in.
- Insert a form type, `itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }` in the database.
- Form permissions for PR user have been set, `testControlPr = { _id: 'pr', items: [{id: 'Form', permissions: {create: false, edit: false, view: false, delete: false}}]}`

UTS75

**Test Items** `checkRights()`

PR user should not be able to delete a form item

**Input specifications**

1. `itemType: 'Form'`
2. `requestedPermission: 'delete'`

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: `AccessControlForm.tests.js`
- A PR user is logged in.
- Insert a form type, `itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }` in the database.
- Form permissions for PR user have been set, `testControlPr = { _id: 'pr', items: [{id: 'Form', permissions: {create: false, edit: false, view: false, delete: false}}]}`
**UTS76**

**Test Items** `checkRights()`
Player user should not be able to create a form item

**Input specifications**

1. `itemType: 'Form'`

2. `requestedPermission: 'create'`

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: `AccessControlForm.tests.js`
- A player user is logged in.
- Insert a form type, `itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }` in the database.
- Form permissions for player user have been set, `testControlPlayer = { _id: 'player', items: [{id: 'Form', permissions: {create: false, edit: false, view: true, delete: false}}]}`

**UTS77**

**Test Items** `checkRights()`
Player user should not be able to edit a form item

**Input specifications**

1. `itemType: 'Form'`

2. `requestedPermission: 'edit'`

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: `AccessControlForm.tests.js`
- A player user is logged in.
- Insert a form type, `itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }` in the database.
- Form permissions for player user have been set, `testControlPlayer = { _id: 'player', items: [{id: 'Form', permissions: {create: false, edit: false, view: true, delete: false}}]}`
UTS78

Test Items checkRights()
Player user should be able to see a form item

Input specifications

1. itemType: 'Form'
2. requestedPermission: 'view'

Output specifications permission: true

Environmental needs:

• The test case is stored in the following file: AccessControlForm.tests.js
• A player user is logged in.
• Insert a form type, itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }
  in the database.
• Form permissions for player user have been set, testControlPlayer = { _id: 'player', items: [{id: 'Form', permissions: {create: false, edit: false, view: true, delete: false}}] }

UTS79

Test Items checkRights()
Player user should not be able to delete a form item

Input specifications

1. itemType: 'Form'
2. requestedPermission: 'delete'

Output specifications permission: false

Environmental needs:

• The test case is stored in the following file: AccessControlForm.tests.js
• A player user is logged in.
• Insert a form type, itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }
  in the database.
• Form permissions for player user have been set, testControlPlayer = { _id: 'player', items: [{id: 'Form', permissions: {create: false, edit: false, view: true, delete: false}}] }
UTS80

**Test Items** checkRights()
Coach user should be able to create a form item

**Input specifications**

1. itemType: 'Form'
2. requestedPermission: 'create'

**Output specifications** permission: true

**Environmental needs:**

- The test case is stored in the following file: AccessControlForm.tests.js
- A coach user is logged in.
- Insert a form type, itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' } in the database.
- Form permissions for coach user have been set, testControlCoach = { _id: 'coach', items: [{id: 'Form', permissions: {create: true, edit: true, view: true, delete: true}}]}

UTS81

**Test Items** checkRights()
Coach user should be able to edit a form item

**Input specifications**

1. itemType: 'Form'
2. requestedPermission: 'edit'

**Output specifications** permission: true

**Environmental needs:**

- The test case is stored in the following file: AccessControlForm.tests.js
- A coach user is logged in.
- Insert a form type, itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' } in the database.
- Form permissions for coach user have been set, testControlCoach = { _id: 'coach', items: [{id: 'Form', permissions: {create: true, edit: true, view: true, delete: true}}]}

ClubNet | Unit Test Plan
UTS82

Test Items checkRights()
Coach user should be able to see a form item

Input specifications

1. itemType: 'Form'
2. requestedPermission: 'view'

Output specifications permission: true

Environmental needs:

- The test case is stored in the following file: AccessControlForm.tests.js
- A coach user is logged in.
- Insert a form type, itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' } in the database.
- Form permissions for coach user have been set, testControlCoach = { _id: 'coach', items: [ {id: 'Form', permissions: {create: true, edit: true, view: true, delete: true}} ]}

UTS83

Test Items checkRights()
Coach user should be able to delete a form item

Input specifications

1. itemType: 'Form'
2. requestedPermission: 'delete'

Output specifications permission: true

Environmental needs:

- The test case is stored in the following file: AccessControlForm.tests.js
- A coach user is logged in.
- Insert a form type, itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' } in the database.
- Form permissions for coach user have been set, testControlCoach = { _id: 'coach', items: [ {id: 'Form', permissions: {create: true, edit: true, view: true, delete: true}} ]}
UTS84

**Test Items** `checkRights()`
General user should not be able to create a form item

**Input specifications**

1. `itemType: 'Form'`

2. `requestedPermission: 'create'`

**Output specifications** `permission: false`

**Environmental needs:**

- The test case is stored in the following file: `AccessControlForm.tests.js`
- A general user is logged in.
- Insert a form type, `itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }` in the database.
- Form permissions for general user have been set, `testControlGeneral = { _id: 'general', items: [{id: 'Form', permissions: {create: false, edit: false, view: false, delete: false}}]}`

---

UTS85

**Test Items** `checkRights()`
General user should not be able to edit a form item

**Input specifications**

1. `itemType: 'Form'`

2. `requestedPermission: 'edit'`

**Output specifications** `permission: false`

**Environmental needs:**

- The test case is stored in the following file: `AccessControlForm.tests.js`
- A general user is logged in.
- Insert a form type, `itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }` in the database.
- Form permissions for general user have been set, `testControlGeneral = { _id: 'general', items: [{id: 'Form', permissions: {create: false, edit: false, view: false, delete: false}}]}`
UTS86

**Test Items** checkRights()
General user should not be able to see a form item

**Input specifications**

1. `itemType: 'Form'`
2. `requestedPermission: 'view'`

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlForm.tests.js
- A general user is logged in.
- Insert a form type, `itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }` in the database.
- Form permissions for general user have been set, `testControlGeneral = { _id: 'general', items: [{id: 'Form', permissions: {create: false, edit: false, view: false, delete: false}}]}`

UTS87

**Test Items** checkRights()
General user should not be able to delete a form item

**Input specifications**

1. `itemType: 'Form'`
2. `requestedPermission: 'delete'`

**Output specifications** permission: false

**Environmental needs:**

- The test case is stored in the following file: AccessControlForm.tests.js
- A general user is logged in.
- Insert a form type, `itemType: { _id: 'Form', name: 'Practicality form', icon: 'Form.ClubNet' }` in the database.
- Form permissions for general user have been set, `testControlGeneral = { _id: 'general', items: [{id: 'Form', permissions: {create: false, edit: false, view: false, delete: false}}]}`
3.2.5 CLUB METHODS

UTS88

Test Items getClub()
Retrieving non-existing club information.

Input specifications None

Output specifications result: undefined

Environmental needs

- The test case is stored in the following file: Club.tests.js
- Login with any user with clubID: 'someOtherClub'

UTS89

Test Items getClub()
Retrieving club information successfully.

Input specifications None

Output specifications result: clubObject

Environmental needs

- The test case is stored in the following file: Club.tests.js
- Login with any user with clubID: 'testClub'

UTS90

Test Items updateClub()
Updating a club with invalid parameters

Input specifications updatedClub: false

Output specifications Error: Match failed

Environmental needs

- The test case is stored in the following file: Club.tests.js
- Login with any user with clubID: 'testClub'
Test Items `updateClub()`
Updating club name successfully.

Input specifications `updatedClub = { name: 'Name2' }`

Output specifications `result: {_id: 'testClub', name: 'Name2', logo: 'logo', colorPrimary: '#FFFFFF',
colorSecondary: '#FFFFFF', colorAccent: '#FFFFFF', heroesMax: 0 }`

Environmental needs

- The test case is stored in the following file: Club.tests.js

- Insert clubObject: `{ _id: 'testClub', name: 'Name1', logo: 'logo', colorPrimary: '#FFFFFF',
colorSecondary: '#FFFFFF', colorAccent: '#FFFFFF', heroesMax: 0 }` into database.

- Login with any user with clubID: 'testClub'
4 TEST PROCEDURE

4.1 STRUCTURE AND EXAMPLE

The procedure to execute the tests mentioned in the previous section is as follows. The procedure described is the same for all test cases because they are executed by the same testing framework: practicalmocha (Mocha.js).

4.1.1 STRUCTURE

Test case identifier
UT<number> of test that is to be executed

Procedure steps
A collection of steps that need to be followed to see the result of the test.

4.1.2 PROCEDURE

UTS<number>

Test case identifier
UTS<number>

Procedure steps

1. Opening your terminal.

2. Change directory to your application root directory.

3. Execute the following command: meteor test –driver-package practicalmeteor:mocha –port 3100

4. See the results in your browser at: http://localhost:3100/
5 TEST COVERAGE

Screenshots of HTML based test coverage report generated by Meteor-Coverage package is provided as follows. The coverage report includes test coverage of used packages as well which need not be tested because they are already tested packages. As such, the test coverage does not clearly represent the actual test coverage of our app. Regarding parts of code that is our own, not all code appears to be tested which is why we can see a lot of red cases indicating lack of test coverage in those regions. But most functionality is actually tested and it is simply difficult to find a proper test coverage tool that complements Meteor perfectly. Nevertheless, it has been added here to serve as a basis for further testing in the future.

![Test Coverage Header](image1.png)

**FIGURE 1: TEST COVERAGE HEADER**

![App Code Test Coverage](image2.png)

**FIGURE 2: APP CODE TEST COVERAGE**

![Packages Test Coverage](image3.png)

**FIGURE 3: PACKAGES TEST COVERAGE**
Steps to generate test coverage:

1. Install Meteor-Coverage package: `meteor add lmieulet:meteor-coverage`

2. On Windows, open up a Command Prompt.

3. Enter `set COVERAGE=1` to enable coverage.

4. Enter `set COVERAGE_APP_FOLDER=/path/to/your/meteor/app/` to set meteor app location.

5. Enter `set COVERAGE_VERBOSE=1` to see logs (optional).

6. `cd /path/to/your/meteor/app/` to change command directory to App directory.

7. Initialize meteor, enter `meteor` in the terminal.
Unit testing consists of both client-side (front-end) tests as well as server-side (backend) tests. Screen shots of HTML based test reports generated by Mocha is provided as follows. The red highlighted numbers across some of the test cases indicate the amount of time it took to complete those test cases. All test cases pass but some take longer than average time. The maximum timeout for each test case is set to default 2000ms.

### Client tests

**Chat**

- **Insert**
  - ✓ should deny insert access to non-logged in users
  - ✓ should deny remove access for a non-existing user
  - ✓ should deny insert access to non-participating user
  - ✓ should deny insert access to non-same club users
  - ✓ should deny insert access to non-same team, but same club, users
  - ✓ should deny insert access to same team and same club but no create rights
  - ✓ should allow insert access to same team and same club with the user having create rights

- **Update**
  - ✓ should deny update access for a non-logged in users
  - ✓ should deny remove access for a non-existing user
  - ✓ should deny update access for a non-participating user
  - ✓ should deny update access for a player editing non-last message
  - ✓ should deny update access for a player for his last message with a closed chat
  - ✓ should deny update access for a coach for his all messages with a closed chat
  - ✓ should allow update access for a player for his last message with an open chat
  - ✓ should allow update access for a coach for his all messages with an open chat

- **Remove**
  - ✓ should deny remove access for a non-logged in users
  - ✓ should deny remove access for a non-existing user
  - ✓ should deny remove access for a non-participating user
  - ✓ should deny remove access for a user without delete rights
  - ✓ should allow remove access for a user with delete rights

**FIGURE 4: CLIENT-SIDE TEST CASES**
Server tests

FIGURE 5: SERVER-SIDE TEST CASES

Club
✓ should throw error while getting non-existing club info
✓ should get club info
✓ should throw error when updating club with invalid parameters
✓ should update club name

FeedItems
addFeedItem()
✓ should fail adding a feed item without item type (invalid parameter)
✓ should succeed adding a feed item

getFeedItem()
✓ should fail getting feed item with non-String input
✓ should get feed item successfully

updateFeedItem()
✓ should fail updating a feed item with non-Object input
✓ should fail if logged in user is not item creator
✓ should succeed updating a feed item

putResponse()
✓ should throw error with non-String id
✓ should throw error with non-String type
✓ should throw error with non-String response value
✓ should add response successfully with valid input

getResponse()
✓ should throw error with non-String item ID
✓ should get the response successfully

getResponsesOfOneItem()
✓ should throw error with non-String item ID
✓ should get an empty array of responses for item without responses
✓ should get the responses on a feed item

getNumberResponsesOfOneItem()
✓ should throw error with wrong parameters
✓ should get 0 responses for item without responses
✓ should get 1 response for an item with 1 response

getResponseOfItemType()
✓ should throw error with invalid parameters
✓ should get an empty array for an item type that does not have responses
✓ should get all responses of given item type

getVotingResults()
✓ should throw error with invalid parameters
✓ should get the voting results
✓ should get the voting results with a new vote added

deleteResponse()
✓ should throw error with invalid parameter
✓ should delete response

deleteFeedItem()
✓ should throw error with invalid parameter
✓ should delete the feed item

trainings()
✓ should get trainings
✓ should throw error with invalid parameters
✓ should retrieve exercise successfully

FIGURE 6: SERVER-SIDE TEST CASES
Access Control Form

PR user
✓ should not be able to create a Form item
✓ should not be able to edit a Form item
✓ should not be able to view a Form item
✓ should not be able to delete a Form item

Player user
✓ should not be able to create a Form item
✓ should not be able to edit a Form item
✓ should be able to view a Form item
✓ should not be able to delete a Form item

Coach user
✓ should be able to create a Form item
✓ should be able to edit a Form item
✓ should be able to view a Form item
✓ should be able to delete a Form item

General user
✓ should not be able to create a Form item
✓ should not be able to edit a Form item
✓ should not be able to view a Form item
✓ should not be able to delete a Form item

Access Control Set Permissions
✓ should allow a PR user to set permissions
✓ should not allow a Player user to set permissions
✓ should not allow a Coach user to set permissions
✓ should not allow a General user to set permissions

Access Control Form

PR user
✓ should not be able to create a Voting item
✓ should not be able to edit a Voting item
✓ should not be able to view a Voting item
✓ should not be able to delete a Voting item

Player user
✓ should not be able to create a Voting item
✓ should not be able to edit a Voting item
✓ should be able to view a Voting item
✓ should not be able to delete a Voting item

Coach user
✓ should be able to create a Voting item
✓ should be able to edit a Voting item
✓ should be able to view a Voting item
✓ should be able to delete a Voting item

General user
✓ should not be able to create a Voting item
✓ should not be able to edit a Voting item
✓ should not be able to view a Voting item
✓ should not be able to delete a Voting item

FIGURE 7: SERVER-SIDE TEST CASES
ITEM_TYPES
- Get ItemTypes
- Get ItemType with wrong parameter
- Get ItemType with correct parameter

USER_ACCOUNT
  addUser()
  - should throw error with incomplete profile information
  - should succeed with complete information
  updateUserProfile()
  - should throw error with incomplete data
  - should throw error with unauthorized logged-in user
  - should throw error with unauthorized FR user from different club
  - should succeed with complete data
  getUserInfo()
  - should throw error with non-String user ID
  - should throw error with non-existing string ID
  - should throw error with unauthorized FR user of different club
  - should succeed with existing string ID
  getUserType()
  - should fail getting user type with incorrect ID
  - should get user type
  getTeamSize()
  - should fail for non-existing team
  - should get team size
  Meteor.users.remove()
  - should delete user with existing ID

FIGURE 8: SERVER-SIDE TEST CASES