Test Plan

Giuliano Rasper **2568405**, Jessica Werner **2563327**,
Stephan Ariesanu **2565385**, Muhammad Kamran **2572847**,
Alexander Dincher **2563446**, Simon Fedick **2563209**

January 27, 2020

Frontend

User Management

1. **Test ID:** Create_Inval_SameMailAddress
   Description: An Admin will create two attedees with completely different data except for the email address being the same. We will test this once using the "Add Attendee" button and once using a ".csv" file.
   Test Data: Two users with different data, only the mail address is the same for both of them.
   Expected Output: Only the first User should get created.

2. **Test ID:** Create_Val_DiffMailAddress
   Description: An Admin will create two different attendees with identical data except for the mail address. We will test this once using the "Add Attendee" button and once using a ".csv" file.
   Test Data: Two users with exactly the same data except for the mail address being the only field that differs, once being added manually and once using a ".csv" file.
   Expected Output: Both attendees should be created without any trouble. Also, different user names should be generated for them.

3. **Test ID:** Create_Inval_EmptyField{1-5}
   Description: An Admin creates a new Attendee using an empty field for their name, email address, group, function or residence in each test. We will test this once using the "Add Attendee" button and once using a ".csv" file each.
   Test Data: An Attendee file with empty fields as well as manual attendee creation using the dialog field of the page.
   Expected Output: The Attendee shouldn't be created and some error message should be displayed.

4. **Test ID:** Logout_Val_AttendeeLogout
   Description: An Admin will log an attendee out and that attendee will try to send different requests afterwards, e.g. try to reload the page, try to vote or try to submit a Request of Change.
   Test Data: A test Attendee who is currently logged in with exactly one device, a test voting which is currently active.
   Expected Output: The test Attendee gets logged out automatically (redirected to the index page) and they can't send any more requests before logging in again. The Admin should see a "Successful Logout" popup coming up as soon as the Logout was successful.

5. **Test ID:** Logout_Inval_AdminLogout
   Description: An Admin will try tolog another admin out. The other Admin will try to still use the page (especially Admin functionality) after the logout.
   Test Data: Two Admins who are currently logged in with exactly one device.
   Expected Output: The "Successful Logout" message shouldn't show and the other Admin should still be able to use all functionality that was available before.

6. **Test ID:** Login_OldPassword
   Description: An Attendee logs in with a valid password, gets logged out by an Admin and tries to log in again using their old password.
   Test Data: A test Attendee and a valid password.
   Expected Output: The test Attendee should get an "Invalid Password" message as the old password got destroyed.

7. **Test ID:** Login_Deleted
   Description: An Admin deletes an attendee who tries to send another request after the deletion, then, in case they get redirected to the index page, tries to log back in using a password that hasn't been used yet.
   Test Data: A test Attendee, a valid password and an admin deleting the attendee.
   Expected Output: The test Attendee's requests (e.g. trying to reload the page) should get rejected, he should be redirected and after entering their username and password should get an error message stating their login data wasn't valid.

8. **Test ID:** Delete_Inval_AdminTarget
   Description: An Admin tries to delete an attendee which also happens to be an Admin.
   Test Data: Two Admin accounts.
   Expected Output: The Admin should be told that deleteing other Admins is not allowed and therefore the request should be rejected.

9. **Test ID:** Delete_Val_AttendeeTarget
   Description: An Admin tries to delete an attendee which has no further admin rights.
   Test Data: A test Attendee without administrative power and an Admin.
   Expected Output: The Attendee should successfully be deleted from the Attendee list.

10. **Test ID:** Login_Inval_NewPassword
    Description: An Admin generates a new password for an Attendee and the Attendee tries to log in with their old password which they haven't logged in with before.
    Test Data: A test Attendee with a valid password and an Admin generating a new valid password (invalidating the old one).
    Expected Output: Generating a new password invalidates the old one. Therefore, the Attendee shouldn't be able to login using their former password even though they haven't been using it yet. The index page should tell them that their data is invalid.

11. **Test ID:** Login_Inval_NewQRCode
    Description: An Admin generates a new password for an Attendee by generating a new QR Code and the Attendee tries to log in with their old password which they haven't logged in with before.
    Test Data: A test Attendee with a valid password and an Admin generating a new QR Code (invalidating the old password).
    Expected Output: Generating a new QR Code and thus generating a new password invalidates the old password. Therefore, the Attendee shouldn't be able to login using their former password even though they haven't been using it yet. The index page should tell them that their data is invalid.

12. **Test ID:** LoginQR_Inval_NewQRCode
    Description: An Admin generates a new QR Code for an Attendee and the Attendee tries to log in with their old QR code which they haven't logged in with before.
    Test Data: A test Attendee with a valid QR Code and an Admin generating a new QR Code (invalidating the password in the old QR Code).
    Expected Output: Generating a new QR Code and thus generating a new password invalidates the password in the old QR Code. Therefore, the Attendee shouldn't be able to login using their former QR code even though they haven't been using it yet. They should get redirected to the index page, though they shouldn't be logged in automatically as their password is now invalid.

13. **Test ID:** LoginQR_Inval_NewQRCode
Description: An Admin generates a new QR Code for an Attendee and the Attendee tries to log in with their old QR code which they haven't logged in with before.
Test Data: A test Attendee with a valid QR Code and an Admin generating a new QR Code (invalidating the password in the old QR Code).
Expected Output: Generating a new QR Code and thus generating a new password invalidates the password in the old QR Code. Therefore, the Attendee shouldn't be able to login using their former QR code even though they haven't been using it yet. They should get redirected to the index page, though they shouldn't be logged in automatically as their password is now invalid.

14. **Test ID:** Sort_MultipleAttendees
Description: An Admin creates four Attendees and after adding them, they sort the Attendees by Name, Group and Function, testing whether the new list is sorted in the right alphabetical order.
Test Data: An Admin and four test Attendees. Two of the Attendees shall have the same group and two shall have the same function, but all of them should have distinct data apart from that.
Expected Output: When sorting by Group, the two Attendees with the same group shall be sorted by function in alphabetical order and when sorting by Function, the two Attendees with the same function shall be sorted by group in alphabetical order. Otherwise, they should just get sorted by their name in alphabetical order.

Voting Panel

Vote Section

1. **Test ID:** Display Vote
Pre conditon: Packet(vote question)
Expected Output: Vote question has been displayed properly
Description: Packet will be received from server and will display vote question with options properly

2. **Test ID:** Submit Vote:
Pre conditon: Vote will be displayed with options and will not be expired.
Expected Output: Vote has been submitted successfully.
Description: An option will be selected and then will click on submit button to submit a vote

3. **Test ID:**Display Previous Votes:
Pre Condition: Previous votes data available
Expected Output: Previous votes will be displayed properly
Description: packet(vote question) will be received from server and vote questions with attendees details will be displayed

4. **Test ID:** Create Vote:
Pre condition: create vote button and dialog box with some inputs fields.
Expected Output vote will be created with 4 buttons i.e. Add, Delete, save changes, start vote
Description: A dialog box will appear when click on create button. After, all inputs will be filled and then will click on confirm button to confirm inputs. Lastly, the result(vote question) will be displayed with 4 buttons.

5. **Test ID:** Input text field as option with remove button attached for a vote question:
Pre condition: Vote question and button(Add) will already be existed.
Expected Output: Input field with button(remove) has been created.

Description: Add button will be clicked to generate a new text field with button(remove) attached.

6. **Test ID:** Start Vote
   Pre condition: Vote question and button for start vote will be existed and options must be more than 2 available beforehand.
   Expected Output Vote will be started.
   Description: When click on the start button, vote will be started.

7. **Test ID:** Delete Vote
   Pre condition Vote question and delete button will be existed
   Expected Output Vote has been deleted successfully
   Description: When click on the delete button vote question will be deleted

8. **Test ID:** Save changes for options:
   Pre condition: Vote question and save button will be existed
   Expected Output All changes will be saved with a success message.
   Description: When click on the save button all changes and new options will be saved.

9. **Test ID:** Remove Option:
   Pre condition: Vote question, input field and remove button will be available
   Expected Output Option will be deleted.
   Description: When click on the remove button, option will be deleted

Communication

General Expectations

It is expected none of the following tests will crash the server.

Security and Performance Tests

These tests target to verify that handling certain security and performance features work properly to guarantee a certain degree of stability of our program.

1. **Test ID:** clientMaxConnectionLimitation
   Input: An authenticated client connects to the server multiple times
   Expected output: As soon as the connection limitation is reached, the newest connections are closed by the server

2. **Test ID:** stressTest
   Input: 1000 clients connect to the server in a tight timeframe and send requests
   Expected output: The clients connection to the server is not interrupted and they revieve a response

3. **Test ID:** clientTimeout
   Input: Non authenticated client connects to the server
   Expected output: The client is disconnected at most 2 seconds after the timeout exeeded

4. **Test ID:** maliciousRequest
   Input: Clients send malicious request e.g. non-existent packets
   Expected output: The client is disconnected.

Packet Tests

These tests target to verify that certain packets are working as documented. Packets tested here are not trivial i.e. they do sigificantly more than redirecting tasks to the conference object. If not explicitly mentioned it expected it is also expected that a ValidResponsePacket is sent if the test id does not contain the word "invalid". If the test contains the word "invalid" and nothing else is mentioned a FailureResponsePacket is expected.

1. **Test ID:** testInvalidToken
   Input: Request for which the users token does not authorize are sent.
   Expected output: The request are rejected and result in an InvalidTokenResponse

2. **Test ID:** testAddMultipleAttendeesRequestPacket
   Input: A AddMultipleAttendeesRequestPacket containing a significant amount of attendees is handled
   Expected output: The attendees are added to the conference whereas doublicate entries (same email) are only added once

3. **Test ID:** testAddTopicRequestPacket
   Input: Multiple AddTopicRequestPacket are handled.
   Expected output: The resulting agenda equals the expected output after the operations were executed

4. **Test ID:** testAddTopicRequestPacketInvalidID
   Input: An AddTopicRequestPacket is handled whereas the position of the topic is invalid
   Expected output: The agenda does not change

5. **Test ID:** testEditUserRequestPacket_invalidID
   Input: Different EditUserRequestPacket are handled containing invalid users id's are handled
   Expected output: The requests are rejected and result in an FaulureResponsePacket

6. **Test ID:** testLogoutAttendeeRequestPacket
   Input: A LogoutAttendeeRequestPacket containing a non admin id is handled
   Expected output: The password and token of the user are invalidated

7. **Test ID:** testLogoutAttendeeRequestPacketInvalid
   Input: A LogoutAttendeeRequestPacket containing an admin id is handled
   Expected output: Neither password nor token of the user are invalidated

8. **Test ID:** testSetRequestStatusRequestPacket
   Input: SetRequestStatusRequestPacket's for multiple requests are handled, setting different stati of the requests
   Expected output: The stati of the requests are updated

9. **Test ID:** testStartVotingRequestPacket
   Input: StartVotingRequestPacket's are handled for a voting with the CREATED status
   Expected output: The voting is started

10. **Test ID:** testStartVotingRequestPacketInvalid
    Input: A StartVotingRequestPacket's are handled for votings with non CREATED status, less than two options amd for start requests while there is already a running voting
    Expected output: The votings are not started

11. **Test ID:** testUpdateFileRequestPacketInvalid
    Input: A UpdateFileRequestPacket are handled whereas the user wasnever eligeble to upload
    Expected output: The file is not updated

12. **Test ID:** testRequestOfSpeechAndChange
    Input: Valid RequestOfSpeechRequestPacket and RequestOfChangeRequestPacket are handled
    Expected output: The Speech request is present / the result contains the correct requested data

Database

These tests are designed to check the individual functions of the database to ensure a proper code foundation on which the advanced functionality can rely on. This means that every function for storing and reading is tested thoroughly because the conversion of Java objects to a persistent database, and vice versa, needs to work perfectly.

Agenda

As the agenda itself contains most of the functionality, the database does not need to check as much and things can be shortened to just one larger test case.

1. **ID**: database.AgendaManagementTests.updateValidAgendaWithoutPrematureReconstruction
   Description: This tests creates a valid agenda, inserts it into the database, tries changing the agenda through the old reference and checks whether the database was in fact not changed while it should not. Afterwards, the database properly gets updated.
   Expected behaviour: The returned Agenda should be identical in the first two cases and properly changed after the update method.

Documents

Because documents should be unique and there is more interaction (upload, change revision, download), the tests are split to cover this functionality modularly.

1. **ID**: database.DocumentsManagmentTests.addAndGetDocuments
   Description: The database stores the name and path of the document. This test add a simple document to the database and compare the real document with the database entries. Furthermore the test checks the read functionality with a wrong documentname.
   Expected behaviour: Both data (DB document and real document) are the same. The database returns a null value if the admin tries a wrong documentname.

2. **ID**: database.DocumentsManagmentTests.deleteDocuments
   Description: First an admin add a new document and delete it later on. After that the document should be removed.
   Expected behaviour: the document entry was succesfully removed from the database.

3. **ID**: database.DocumentsManagmentTests.deleteWrongDocuments
   Description: An admin try to delete a wrong document in the database.
   Expected behaviour: the database entries do not change.

4. **ID**: database.DocumentsManagmentTests.updateDocuments
   Description: First an admin adds new documents and after that he tries to update the document three times. First he updates the right document. Second he tries to update a not existing document and third he updates the updated document.
   Expected behaviour:During a succesful update the revisionnumber of the document should increase by 1. Furthermore a wrong update should not influence the data.

5. **ID**: database.DocumentsManagmentTests.documentnameIsAlreadyUsed
   Description: the database needs to know if a documentname is already in use. The system can not store documents with the same name, because documentnames are unique.
   Expected behaviour: the database should return true if a documentname is already in the database and should return false otherwise.

6. **ID**: database.DocumentsManagmentTests.addSameDocumenttwice
   Description: According to the previous test admins ca not upload a document twice.
   Expected behaviour: the database stores the fist document in the database, but the second try gets ignored.

Users

Obviously, users do have a bit more functionality, because a lot of informations needs to be stored and passwords and tokens need to be validated in a persistent way, in order to easily restart the conference in case something crashed without relogging all users.

1. **ID**: database.UserManagementTests.validAttendeeCredentials
   Description: The system stores a new valid attendee to the database.
   Expected behaviour: The system can store the attendee with the right values and can read the same values from the database.

2. **ID**: database.UserManagementTests.validAdminCredentials
   Description: The system stores a new valid admin to the database.
   Expected behaviour: The system can store the admin with the right values and can read the same values from the database.

3. **ID**: database.UserManagementTests.removeUser
   Description: The database can remove existing admins and attendees. Furthermore the database should ignore removing wrong users.
   Expected behaviour: The database do not store the removed admins and attendees. Furthermore the database ignores removing a not existing user.

4. **ID**: database.UserManagementTests.logoutUser
   Description: The system can logout users. During this operation the database overwrites the password and the token of the existing user.
   Expected behaviour: The database stores the new password and token correctly and the database can identify the user with the new token.

5. **ID**: database.UserManagementTests.getCorrectPasswords
   Description: The system needs all passwords with the right users to garantie a successfull login.
   Expected behaviour: The system gets all passwords with the corresponding user in the right order.

6. **ID**: database.UserManagementTests.newTokenToUser
   Description: After a user logout the database should store a new token to the correct user. Furthermore the old token can not be found in the database.
   Expected behaviour: The users in the database get new token and the database can not convert a not existing token to an user id. in that case the database throws an IllegalArgumentException.

7. **ID**: database.UserManagementTests.newPasswordToUser
   Description: After a user logout the database should store a new password to the correct user.
   Expected behaviour: The users in the database get new password and the old password are overwritten.

8. **ID**: database.UserManagementTests.existUserName
   Description: The system should check if an username is already existing, because the database can only have unique usernames.
   Expected behaviour: The functionality returns true for an existing username and false for a non existing username.

9. **ID**: database.UserManagementTests.severalAttendeesAndAdmins
   Description: The database should store many attendees and admins with individual data.
   Expected behaviour: The database stores the users with the correct data.

10. **ID**: database.UserManagementTests.editUser
    Description: The system can edit the user in the database. Therefore the database should store the new values and overwrites the old ones.
    Expected behaviour: The database has the new entries.

11. **ID**: database.UserManagementTests.differentCheckLoginAndTokenCases
    Description: The system tries to login users and check their tokens. In order to this the database compare the given values with the database entries and creates different TokenResponse or LoginResponse.
    Expected behaviour: The database creates the right Token- or LoginResponse types.

12. **ID**: database.UserManagementTests.getallGroupsFromDb
    Description: The system needs all user groups without duplication.
    Expected behaviour: The database loads all groups without a duplication.

13. **ID**: database.UserManagementTests.getRightPresentValue
    Description: During a conference the admins can see if a user is present. Therefor the database should know if a user is present or not.
    Expected behaviour: the database contains the right present value.

Requests

Requests are the point where it starts to become a bit more complex (for the database), because a request contains information about the requester (user) and the agenda point or document the request refers to. This means that the database must be designed properly in order to ensure this.

1. **ID**: database.RequestManagerTests.addRequestTest
   Description: The system adds two new valid requests (Speech and Change) to the database.
   Expected behaviour: The system can add both requests with the right values.

2. **ID**: database.RequestManagerTests.getvalidRequestTest
   Description: After the system adds both request types to the database, the database stores the right values.
   Expected behaviour: The database contains the valid requests.

3. **ID**: database.RequestManagerTests.updateRequestTest
   Description: The system can edit the requests in the database. Therefore the database should store the new values and overwrites the old ones.
   Expected behaviour: After the update the database contains the new values.

4. **ID**: database.RequestManagerTests.deleteRequestTest
   Description: The system can remove existing requests. Furthermore the database should ignore removing wrong requests.
   Expected behaviour: The database do not store the removed requests. Furthermore the database ignores removing a not existing request.

Voting

Votings are only complex in a way that there are two different types of votings (designed as two subclasses) and the database needs to store both types without too much overhead and without loosing the abstraction to just one abstract Voting super-class. However, as votings are only stored after a voting was finished (in case somethings breaks and people were not allowed to vote), it suffices to just create and reconstruct each voting type and also check whether an open voting is rejected properly.

1. **ID**: database.VotingManagementTests.checkSimpleVotings
   Description: The system performes both voting types and after that the results get stored into the database.
   Expected behaviour: The database contains the right values of the previous votings.

2. **ID**: database.VotingManagementTests.addOpenVoting
   Description: The database can only store closed votings.
   Expected behaviour: During the vote, the system can not store the voting in to the database.

Conference

Documents Tests on the conference level

1. **ID**: documents.documentsTests.singleDocumentUpload
   Description: Uploads a single document.

Expected behaviour: The document should have the correct content and revision number 1

2. **ID**: documents.documentsTests.updateInexistent
   Description: Tries to update a document which does not exist.
   Expected behaviour: The conference should throw a IllegalArgumentException

3. **ID**: documents.documentsTests.updateInexistent2
   Description: Tries to update a document after it was deleted.
   Expected behaviour: The conference should throw a IllegalArgumentException

4. **ID**: documents.documentsTests.documentMultiUpdate
   Description: A documents gets updated multiple times.
   Expected behaviour: The content should be equal to the value of the last update and the revision number should be correct

5. **ID**: documents.documentsTests.deletedDocumentRecreate
   Description: A documents gets deleted and then uploaded again.
   Expected behaviour: The content should be available again with revision number 1

6. **ID**: documents.documentsTests.uploadLarge
   Description: A document over 500MB gets uploaded.
   Expected behaviour: The upload gets rejected

Request Tests on the conference level

1. **ID**: requests.requestTests.multipleRequests
   Description: Sends multiple requests from the same user to the same Request item. Furthermore one changerequest get copied and get approved and disapproved
   Expected behaviour: All requests should be logged and the one request contains the right values.

2. **ID**: requests.requestTests.deleteUser
   Description: Deletes a user that send a request. Furthermore one speechrequest get copied and get closed and reopened
   Expected behaviour: The request should also be deleted and the speechrequest contains the right values

3. **ID**: requests.requestTests.deleteDocument
   Description: Deletes a document to which a request refers
   Expected behaviour: The request should still identify the document by the name

4. **ID**: requests.requestTests.deleteTop
   Description: Deletes a top to which a request refers
   Expected behaviour: The request should still identify the top by the name

User Tests on the conference level

1. **ID**: user.userTests.addUserConcurrentlySameEmail
   Description: multiple threads try to add the same user (i.e. a user with the same email address) to the conference concurrently.
   Expected behaviour: A thread succeeds, while all other receive IllegalArgument Exceptions.

2. **ID**: user.userTests.addUserConcurrentlyDifferentEmail
   Description: multiple threads try to add different to the conference concurrently.
   Expected behaviour: All threads should succeed

3. **ID**: user.userTests.editUsersConcurrently
   Description: multiple threads try to edit different users concurrently.
   Expected behaviour:All threads should succeed.

4. **ID**: user.userTests.logAttendeesInAndOut
   Description: multiple threads try to log users in, some with valid and some with invalid credentials.
   Expected behaviour: All thread succeed, i.e. thread with valid login data should receive a positive response and threads with invalid credentials should not receive tokens

5. **ID**: user.userTests.invalidLogin
   Description: A attendee tries to log in after being logged out.
   Expected behaviour: The login should be rejected, as the password gets invalidated by a logout (even if the user never logged in).

6. **ID**: user.userTests.invalidLogin2
   Description: A attendee tries to log in after all users get logged out.
   Expected behaviour: The login should be rejected, as the password gets invalidated by a logout (even if the user never logged in).

7. **ID**: user.userTests.invalidLogin3
   Description: An attendee tries to log in after all attendees get logged out.
   Expected behaviour: The login should be rejected, as the password gets invalidated by a logout (even if the user never logged in).

8. **ID**: user.userTests.invalidLogin4
   Description: An attendee tries to log in after the user gets removed.
   Expected behaviour: The login should be rejected, as the user is no longer poart of the conference.

9. **ID**: user.userTests.invalidLogin5
   Description: An admin tries to log in after the user gets removed.
   Expected behaviour: The login should be rejected, as the user is no longer poart of the conference.

10. **ID**: user.userTests.invalidLogin7
    Description: A admin tries to log in with an old password (i.e. after a new one got generated) out.
    Expected behaviour: The login should be rejected, as the password gets invalidated by creating a new password (even if the user never logged in).

11. **ID**: user.userTests.validLogin
    Description: A admin tries to log in with valid credentials.
    Expected behaviour: The login should be succeed.

12. **ID**: user.userTests.loginTwice
    Description: A attendee tries to log in twice using the same password.
    Expected behaviour: The login should be rejected, as the password are one time use only

13. **ID**: user.userTests.loginTwice2
    Description: A admin tries to log in twice using the same password.
    Expected behaviour: The login should not be rejected, as the password are multi time use

14. **ID**: user.userTests.loginTwice3
    Description: A Admin tries to log in twice. The first time the password is incorrect, but the second time the password is correct
    Expected behaviour: Only the second login should succeed

15. **ID**: user.userTests.testTokens
    Description: A Admin and two attendees log in. One of the attendees gets removed.
    Expected behaviour: The token of the admin and of the first attendee should be valid admin / attendee tokens. The token of the third user should be invalid

16. **ID**: user.userTests.tokenReset
    Description: A Attendee logs in. later the token of the attendee gets reset
    Expected behaviour: The token should be invalid after the reset

Voting Tests on the conference level

1. **ID**: votings.votingTests.addMultipleVotings
   Description: Adds a anonymous and a named voting. Starts one of them and waits until its duration expires. Checks the voting status of both votes in the meantime
   Expected behaviour:The voting status should be consistent during the test run

2. **ID**: votings.votingTests.multipleVotesRegular
   Description: Multiple users submit votes in a anonymous and a names voting.
   Expected behaviour:The voting resuolt should only be available after the voting closes. Also it should be correct

3. **ID**: votings.votingTests.multipleVotes
   Description: A user tries to vote multiple times in a anonymous and a regular vote
   Expected behaviour:The second vote should get rejected

4. **ID**: votings.votingTests.multipleRunningVotes
   Description:Multiple votes get started at the same time, before the first one ends
   Expected behaviour:The operation fails

5. **ID**: votings.votingTests.editRunningVoting
   Description: Tries to edit a voting while it is running and after it ended
   Expected behaviour:The operation fails

6. **ID**: votings.votingTests.voteNonRunning
   Description: Tries to vote for a vote that does not have the status 'running'
   Expected behaviour:The operation fails

Tests for the configuration file parser

1. **ID**: config.parser.defaultValue
   Description: Uploads a configuiration file containing only mandatory fields
   Expected behaviour: The conference should start with the specified default values

2. **ID**: config.parser.multiKey
   Description: Uploads a configuiration file containing the 'name' field twice
   Expected behaviour: The parser should throw a IllegalArgumentEWxception

3. **ID**: config.parser.noAdmins
   Description: Uploads a configuiration file containing the no admin fields
   Expected behaviour: The parser should throw a IllegalArgumentEWxception

4. **ID**: config.parser.missingMandatory
   Description: Uploads a configuiration file that has no 'url' field
   Expected behaviour: The parser should throw a IllegalArgumentEWxception

5. **ID**: config.parser.missingValue
   Description: Uploads a configuiration file containing the line 'endTime:\'\".
   Expected behaviour: The parser should throw a IllegalArgumentEWxception

6. **ID**: config.parser.pastTime
   Description: Uploads a configuiration file that has the 'endsAt' field set to a past time
   Expected behaviour: The parser should throw a IllegalArgumentEWxception

7. **ID**: config.parser.persistency
   Description: Starts a conference and adds some documents and a voting. Start the conference again using the same data
   Expected behaviour: The data added to the conference (outside of the config file) should still exist and be the same as before closing the conference

8. **ID**: config.parser.persistency2
   Description: Starts a conference and adds some documents and a voting. Start the conference again using a different name and end time in the config file
   Expected behaviour: The data added to the conference (outside of the config file) should still exist. The fields specified in the config file should have the new values

9. **ID**: config.parser.adminChange
   Description: Uploads a configuiration file that has three admins. Start the conference again after replacing two admins in the file.
   Expected behaviour: Only the admins mentioned in the second file should exist

10. **ID**: config.parser.escape
    Description: Writes the name "e\\s\\\′ cap\#e" in the config file
    Expected behaviour: The coinference should be named "e\s\′cap#e"