Test Report

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

January 27, 2020

Total Coverage

Backend Test Coverage: Backend Tests were done with JUnit test, descriped below. The backend, especially the communication had a smaller coverage, which is explicit documented in the Communication section below. This component affects the total coverage of the project to a total coverage of 69%.
Frontend Test Coverage: Frontend Tests were most done manually to test whether things got displayed correctly and thus code coverage was hard to measure. As the Backend got tested on all functionality using JUnit testing, UI testing was most important here.

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

   Status: Success

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
   Status: Success

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
   Status: Success

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

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 **Status:** Success

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
   Status: Success

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

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

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
   Status: Success

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
   Status: Success

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

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
   Status: Success

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

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
    Status: Success

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

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
    Status: Success

Coverage

The total line coverage of the communication module is 46.87 percent. This has several reasons which I'll go into detail which also lead to additional manual testing.

Untestability

First off since we are talking about a module which ensures communication between the frontend and backend there is certain functionality which should not just be tested locally. Despite there were some tests (Security and Performance Tests) which actually establish a connection to the server, these kind of tests are vaguely representative since they test communication only locally. Therefore additional tests of this type are covered by manual tests for which we used an actual server running our software. Moreover tests involving certificates would simply not really work as unit tests since they would instantly fail if run on another machiene.

Code similarity

If one takes a closer look, especially at the "communication.packets" package, it can be seen that a lot of packets which are sent during communication are very similar. The functionality of most of these packets is guaranteed if

1. the general communication works (i.e. there are other packets for which communication works)

2. the backend provides the correct data

which is due to the fact that a lot of request packets redirect the actual task to the actual backend (mainly the conference class) whereas most of the response packets redirect data objects provided by the backend. Therefore it is important to note that the classes AuthenticatedRequestPacket, BasePacket, ResponsePacket and RequestPacket. all have a coverage of 100 percent since every other packet is a subclass of at least one of those. Packets which do sigificantly more than redirecting requests (or are of special importance) are the LoginRequestPacket (100%), RequestOfPacketWrapper (94%), RequestOfChangeRequestPacket (100%), GetAgendaRequestPacket (100%), StartVotingRequestPacket (95%), AddMultipleAttendeesRequestPacket (93%), EditUserRequestPacket (100%), AddTopicRequestPacket (100%), SetRequestStatusRequestPacket (100%), LogoutAttendeeRequestPacket (100%), GetAllRequestsRequestPacket (100%). Note that despite packets as UpdateFileRequestPacket and DownloadFileRequestPacket also privide sigificantly more functionality these are tested manually for ensuring that file transfer (for large files) works when actually sending data over the internet.

Manual Testing

For the following manual tests a client with a download speed of 200Mbps and a upload speed of 12Mbps was used. The server was running on a hetzner CX11 cloud server.

Encryption

The usage of secure communication (SSL / WSS) have been tested using certificates signed by LetsEncrypt and has been identified as working.

General communication

It has been tested that all kind of packets are sent and reveived when running the backend on an actual server.

File Upload and Download

We tested the upload and download functionalities of large files. This was tested with several small and large files (more than 100MB) which revealed that our software does not segnificantly hamper upload / download speeds. The incrementation of the revision number as well as upload limits (e.g. restrictions for not beeing able to replace .zip files with .pdf files) held true.

QR Code Download

Downloading the QR files generated in the backend has also been tested for conferences with many attendee's since depending on the download option all QR codes are included in one zip file.

Conferences with many Attendee's

For conferences with many attendees it has been tested that all tabs still work accodingly. It has especially been tested that many attendees (more than 1000) do not sigificantly decrease the performance if the user management tab since all attendees are displayed.

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.
   Status: Success

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 output: Both data (DB document and real document) are the same. The database returns a null value if the admin tries a wrong documentname.
   Status: Success

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 output: the document entry was succesfully removed from the database.
   Status: Success

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

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 output: During a succesful update the revisionnumber of the document should increase by 1. Furthermore a wrong update should not influence the data.
   Status: Success

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 output: the database should return true if a documentname is already in the database and should return false otherwise.
   Status: Success

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

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 output: The system can store the attendee with the right values and can read the same values from the database.
   Status: Success

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

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

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 output: The database stores the new password and token correctly and the database can identify the user with the new token.
   Status: Success

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

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 output: 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.
   Status: Success

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

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 output: The functionality returns true for an existing username and false for a non existing username.
   Status: Success

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

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 output: The database has the new entries.
    Status: Success

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 output: The database creates the right Token- or LoginResponse types.
Status: Success

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

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 output: the database contains the right present value.
Status: Success

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 output: The system can add both requests with the right values.
Status: Success

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

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 output: After the update the database contains the new values.
Status: Success

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

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 output: The database contains the right values of the previous votings.
Status: Success

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

Conference

Documents Tests on the conference level

1. **ID**: documents.documentsTests.singleDocumentUpload
Description: Uploads a single document.
Expected output: The document should have the correct content and revision number 1
Status: Success

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

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

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

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

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

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 output: All requests should be logged and the one request contains the right values.
Status: Success

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

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

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

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 output: A thread succeeds, while all other receive IllegalArgument Exceptions.
   Status: Success

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

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

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

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

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

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

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

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

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 output: The login should be rejected, as the password gets invalidated by creating a new password (even if the user never logged in).
    Status: Success

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

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

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

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 output: Only the second login should succeed
    Status: Success

15. **ID**: user.userTests.testTokens
    Description: A Admin and two attendees log in. One of the attendees gets removed.
    Expected output: 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
    Status: Success

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

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 output: The voting status should be consistent during the test run
   Status: Success

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

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

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

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

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

Tests for the configuration file parser

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

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

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

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

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

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

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 output: The data added to the conference (outside of the config file) should still exist and be the same as before closing the conference
   Status: Success

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 output: 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
   Status: Success

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 output: Only the admins mentioned in the second file should exist
Status:Success

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

Frontend

User Management Panel

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.
   Actual Output: Worked as expected (only tested manually).

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.
   Actual Output: Worked as expected (only tested manually).

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.
   Actual Output: Worked as expected (only tested manually).

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.
   Actual Output: Worked as expected (only tested manually).

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.
Actual Output: Worked as expected (only tested manually).

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.
   Actual Output: Worked as expected (only tested manually).

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.
   Actual Output: Worked as expected (only tested manually).

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.
   Actual Output: Worked as expected (only tested manually).

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.
   Actual Output: Worked as expected (only tested manually).

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.
    Actual Output: Worked as expected (only tested manually).

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.
    Actual Output: Worked as expected (only tested manually).

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.
Actual Output: Worked as expected (only tested manually).

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.
Actual Output: Worked as expected (only tested manually).

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.
Actual Output: Worked as expected (only tested manually).

Document Panel

1. **Test ID:** Upload_ValidPDF
Description: An Admin selects a pdf file of an ordinary size and content using the "browse" field and clicks on the "Submit" button while two files are already uploaded.
Test Data: An Admin account, a pdf file of around 20MB and two txt files already being uploaded to the server with ID 1 and ID 2.
Expected Output: The new file should get displayed in the file list on ID 3. Its revision number should furthermore be 1 by default.
Actual Output: Worked as expected (only tested manually).

2. **Test ID:** Edit_ValidExtension
Description: An Admin chooses to edit a file of file extension "X" replacing it with a file of extension "X". After that, the Admin downloads the file and tests which file he gets.
Test Data: Two files of the same file extension "X" (test for multiple extensions like .pdf, .mp3, .mov, .txt, .csv, ...).
Expected Output: The uploading should work flawlessly, the revision counter should be on 2 and downloading the file should get the Admin a file exactly identical to the second file uploaded.
Actual Output: Worked as expected (only tested manually).

3. **Test ID:** Edit_InvalidExtension
Description: An Admin chooses to edit a file of file extension "X" replacing it with a file of extension "Y". After that, the Admin downloads the file and tests which file he gets.
Test Data: One uploaded file of file extension "X" and one file of file extension "Y" while $X \neq Y$ (test

for multiple extensions like .pdf, .mp3, .mov, .txt, .csv, ...).
Expected Output: The uploading shouldn't work, the Admin should get an Error message, the revision counter should stay on 1 and downloading the file should get the Admin a file exactly identical to the first file uploaded.
Actual Output: Worked as expected (only tested manually).

4. **Test ID:** Delete_Success
Description: An Admin chooses to delete a file of any extension "X" that was uploaded before.
Test Data: An Admin account, an uploaded file of file extension "X".
Expected Output: After deleteing the uploaded file, it shouldn't be displayed in the list anymore and therefore, downloading and editing shall no longer be possible.
Actual Output: Worked as expected (only tested manually).

Agenda Panel

1. **Test ID:** CreateTopic_Indexing
Description: An Admin clicks the "plus" icon next to the first topic in an Agenda with two topics in total, without subtopics.
Test Data: An Agenda with two items, "one" on ID 1 and "two" on ID 2, an Admin account and the new topic "test".
Expected Output: The new topic "test" should get inserted between "one" and "two" now, resulting in "test" getting the index 2.
Actual Output: Worked as expected (only tested manually).

2. **Test ID:** CreateSubTopic_Indexing1
Description: An Admin clicks the "down arrow" icon next to the first topic in an Agenda with two topics in total, without subtopics at first.
Test Data: An Agenda with two items, "one" on ID 1 and "two" on ID 2, an Admin account and the new subtopic "test" to the topic "one".
Expected Output: The new topic "test" should get inserted as first subtopic of topic "one", giving it the ID 1.1.
Actual Output: Worked as expected (only tested manually).

3. **Test ID:** CreateSubTopic_Indexing2
Description: An Admin clicks the "down arrow" icon next to the first topic in an Agenda with two topics in total while the first topic already has two subtopics.
Test Data: An Agenda with two topics, "one" on ID 1 and "two" on ID 2 as well as the subtopics "sub1" on ID 1.1 and "sub2" on ID 1.2, an Admin account and the new subtopic "test" to the topic "one".
Expected Output: The new topic "test" should get inserted as second subtopic of topic "one" since it counts as clicking the "plus" icon on topic 1.1, giving the new topic "test" the ID 1.2.
Actual Output: Worked as expected (only tested manually).

4. **Test ID:** DeleteTopic_Subtopics1
Description: An Admin deletes the first topic on an agenda (clicking the respective icon) with two topics while the first topic has two subtopics and the second topic has one subtopic.
Test Data: An Agenda with two topics, "one" on ID 1 and "two" on ID 2 as well as the subtopics "sub1" on ID 1.1, "sub2" on ID 1.2 and "sub3" on ID 2.1, and an Admin account.
Expected Output: As "one" is about to get deleted, its subtopics should get deleted, too. After the execution, the only topics on the Agenda should be "two" on ID 1 and "sub3" on ID 1.1.
Actual Output: Worked as expected (only tested manually).

5. **Test ID:** EditTopic_Successful
Description: An Admin edits the second topic on an Agenda with three topics and a subtopic to the second topic.

Test Data: An Agenda with two topics, "one" on ID 1, "two" on ID 2 and "three" on ID 3, as well as the subtopic "sub1" on ID 1.1, an Admin account and the new name "test" for the topic "two".
Expected Output: When "two" is edited, its name should be changed to "test" while the name of all other subtopics shall stay the same.
Actual Output: Worked as expected (only tested manually).

Request Panel

1. **Test ID:** Submit_Change_Invalid
   Description: An Attendee tries to submit a Request of Change without selecting a topic or document to refer to.
   Test Data: An ordinary Attendee account.
   Expected Output: The Frontend should tell the Attendee to select a topic before submitting the Request and the Request should not get sent.
   Actual Output: Worked as expected (only tested manually).

2. **Test ID:** Submit_Change_Valid
   Description: An Attendee tries to submit a Request of Change with any Request text after selecting a topic or document to refer to. An Admin refreshes the "Manage Requests" page to see if the Request arrived.
   Test Data: An ordinary Attendee account, an Admin account and an Agenda Topic or an uploaded Document.
   Expected Output: The Frontend should send the Request to the backend, giving the Attendee a "Request sent successfully" message. The Admin should be able to see the Request with the correct data (Request Type, Requesting Attendee, Request Text, Request Topic, Time Stamp) being displayed. The Request Status should be open by default.
   Actual Output: Worked as expected (only tested manually).

3. **Test ID:** Submit_Speech_Valid
   Description: An Attendee tries to submit a Request of Speech after selecting a topic or document to refer to. An Admin refreshes the "Manage Requests" page to see if the Request arrived.
   Test Data: An ordinary Attendee account, an Admin account and an Agenda Topic or an uploaded Document.
   Expected Output: The Frontend should send the Request to the backend, giving the Attendee a "Request sent successfully" message. The Admin should be able to see the Request with the correct data (Request Type, Requesting Attendee, Request Topic, Time Stamp) being displayed. The Request Status should be open by default.
   Actual Output: Worked as expected (only tested manually).

4. **Test ID:** Approve_Change_Valid
   Description: After a Request of Change has been submitted, an Admin approves of it in the "Manage Requests" Page by clicking the tick icon.
   Test Data: An ordinary Attendee account, an Admin account, an Agenda Topic or an uploaded Document and a Request of Change (containing a Request text) referring to it.
   Expected Output: The Request Status displays "approved", the Request will then be considered closed and therefore moves to the bottom of the list as it has been the latest Request that has been closed.
   Actual Output: Worked as expected (only tested manually).

5. **Test ID:** Disapprove_Change_Valid
   Description: After a Request of Change has been submitted, an Admin approves of it in the "Manage Requests" Page by clicking the "X" icon.
   Test Data: An ordinary Attendee account, an Admin account, an Agenda Topic or an uploaded Document and a Request of Change (containing a Request text) referring to it.
   Expected Output: The Request Status displays "disapproved", the Request will then be considered

closed and therefore moves to the bottom of the list as it has been the latest Request that has been closed.
Actual Output: Worked as expected (only tested manually).

6. **Test ID:** Close_Speech_Valid
   Description: After a Request of Speech has been submitted, an Admin closes it in the "Manage Requests" Page by clicking the tick icon.
   Test Data: An ordinary Attendee account, an Admin account, an Agenda Topic or an uploaded Document and a Request of Speech referring to it.
   Expected Output: The Request Status displays "closed", the Request therefore will be considered closed and moves to the bottom of the list as it has been the latest Request that has been closed.
   Actual Output: Worked as expected (only tested manually).

Login Panel

See page below

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Invalid username or password	Enter username & password in input fields and then click on login button to see a message invalid user name or password.	1. Open Browser 2. Enter username 3. Enter Password 4. Click on Login button	• Username and password	• Invalid username or password	• Invalid username or password	Pass

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Valid username & password	Enter correct username & password in input fields and then click on login button to see logged in successfully message and navigate to home.html page.	1. Open Browser 2. Enter username 3. Enter Password 4. Click on Login button	• Username and password	• Logged in successfully	• Logged in successfully	Pass

Voting Panel

See page below

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Display Vote	• Receive Packet (Data) from backend and display it.	**1.** Open Browser **2.** Login with right credentials **3.** Navigate to vote page **4.** In Active vote panel, vote question with options are displayed.	**1.** Packet Data Received from backend **2.** Display a vote with options related.	• Vote question is displayed	• Vote question is displayed.	Pass

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Submit Vote	• Select an option and submit a vote by clicking on submit button.	**1.** Open Browser **2.** Login with right credentials **3.** Navigate to vote page **4.** Select option (if vote question is available) **5.** Submit a vote	• Select option • Click on submit button to see vote is submitted or not.	Vote has been submitted.	Vote has been submitted.	Pass

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Display Previous Votes	• Receive Packet (Data) from backend and display it.	**1.** Open Browser **2.** Login with right credentials **3.** Navigate to vote page **4.** Previous votes have been displayed (if available)	• Packet Data Received from backend • Display a vote with options and submission details of attendees.	• Previous votes have been displayed	• Previous votes have been displayed	Pass

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Create Vote	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.	**1.** Open Browser **2.** Login with right credentials **3.** Navigate to vote page **4.** Click on Create Vote button **5.** A dialogue box will appear fill all the input fields. **6.** Confirm it by clicking on confirm button.	• Create vote button • Input fields i.e. Vote question, duration of question, named vote. • Four buttons i.e. Start Vote, Add, Save Changes, Delete.	• Vote has been created with four buttons i.e. Start Vote, Add, Save Changes, Delete	• Vote has been created with four buttons i.e. Start Vote, Add, Save Changes, Delete	Pass

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Input text field as option with remove button attached for a vote question.	Add button will be clicked to generate a new text field with button attached.	**1.** Open Browser **2.** Login with right credentials **3.** Navigate to vote page **4.** Click on vote question and expand a list **5.** Click on Add button	• Input field with remove button attached	• Input field has been created with remove button attached	• Input field has been created with remove button attached	Pass

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Start Vote	When click on the start vote button under vote question if available vote will be started.	**1.** Open Browser **2.** Login with right credentials **3.** Navigate to vote page **4.** Click on vote question and expand a list **5.** Click on Add button **6.** Fill all the three input fields **7.** Click on start vote.	• Start a vote	• Vote has been started	• Vote has been started	Pass

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Delete Vote	When click on the delete button under vote question if available vote will be deleted.	**1.** Open Browser **2.** Login with right credentials **3.** Navigate to vote page **4.** Click on vote question and expand a list **5.** Click on Delete button	• Delete a vote completely.	• Vote has been deleted successfully	• Vote has been deleted successfully	Pass

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Save Changes for options	When click on the Save changes button under a vote question, changes will be saved.	**1.** Open Browser **2.** Login with right credentials **3.** Navigate to vote page **4.** Click on vote question and expand a list **5.** Click on Save Changes button	• Data in input fields.	• Options have been saved successfully	• Options have been saved successfully	Pass

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual results	Pass/Fail
Remove Option	When click on the Remove button under a vote question, option with button will be removed.	**1.** Open Browser **2.** Login with right credentials **3.** Navigate to vote page **4.** Click on vote question and expand a list **5.** Click on Remove button.	• Input field with button	• Option has been removed successfully	• Options has been removed successfully	Pass