Testbericht

Fangzhou Bian, Kathrin Blum, Matthias Bruns,
Leonhard Duda, Tan Grumser, Yuguang Lin

1. September 2019

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument verschafft einen Überblick über die Qualitätssicherung des Projekts. In dieser Phase wurde die Überdeckung der Unittests maximiert, die Testszenarien des Pflichtenhefts durchgegangen und gefundene Fehler behoben. Des Weiteren wurde ein Hallway Usability Test durchgeführt. Die dokumentierten Tests wurden wieder unterteilt in einen Server- und Clientteil.

2 Server

2.1 Bug Fixes

- Scheduler

  Symptom: Die Funktionen die einmal täglich aufgerufen werden sollten, wurden nicht ausgeführt. TimeController.deleteGroups() und TimeController.updateMensaData() wurden nicht aufgerufen.

  Ursache: Die Annotationen @Component hat in der Klasse TimeController hat gefehlt, damit hat SpringBoot die Klasse nicht gefunden und den Sceduler nicht Initialisiert.

  Behebung: Die Annotation wurde hinzugefügt.

- user.groupToken bei Mitternacht nicht gelöscht.

  Symptom: Nach löschen aller Gruppen durch die Funktion GroupController.removeAllGroups() wurde bei Usern nicht das groupToken auf null gesetzt.

  Ursache: Es wurde zwar über alle User iteriert und auf der Instanz die der Iterator zurück gegeben hat das Token auf null gesetzt, aber die User wurden nicht wieder im UserRepository und somit in der Datenbank aktualisiert.

  Behebung: Nach dem Löschen des Tokens werden die User wieder gespeichert.

2.2 Unittests

Für die Unit Tests wurde, wie schon zuvor, das Framework JUnit verwendet. Die Testabdeckung am Server wurde mit EclEmma überprüft. Einige Klassen enthalten keinerlei Logik und rufen nur Funktionen anderer Klassen auf, entsprechend wurden diese nicht getestet.
Zusätzlich zu den bisherigen Unit Tests, welche im Bericht der Implementierungsphase dokumentiert sind, wurden die folgenden hinzugefügt:

- GroupController - Test
  - getGroupByPreferecne Test: Zwei Testgruppen wurden dem Repository hinzugefügt jeweils mit der selben meetingTime, aber unterschiedlichen Essensli-

nien Dann wurde die Methode getGroupByPreference mit Start und Endzeit, sodass beide Gruppen sie erfüllen und einem Array aus Essenslinien, von denen es nur eine Übereinstimmung mit den Gruppen gibt, aufgerufen. Da GetGroupByPreference sowohl die angegebenen Zeiten, als auch die Essenslinien berücksichtigt, dürfte nur eine Gruppe zurückgegeben werden. zum überprüfen würde die Länge des zurückgegebenen Arrays überprüft.

Test2: Analog zu oben, nur dass keine Gruppe eine übereinstimmende Essenslinie mit den Preferenzen hatte. Ziel war es zu sehen, ob ein Array der Länge 0 zurückgegeben wird oder unerwartetes Verhalten auftritt. Wie erwartet hatte das zurückgegebene Array die Länge null.

Test3: Analog zu oben, gruppen haben passende Essenslinie aber nur eine Gruppe hat eine passende (Rand) Uhrzeit, und wie erwartet wird genau diese Gruppe zurückgegeben.

- UserController - Test
  - DeleteUser Test 1: Ein User (Üser1") wurde anhand seines Tokens ins Repository hinzugefügt. Nun wird versucht einen anderen User (Üser2"), der nicht im Repository liegt, aus dem Repository zu löschen. Wie erwartet, wird dabei kein anderer User gelöscht, sondern eine ResponseStatusException geworfen.
  - DeleteAllUser Test1: Zwei User werden ins Repository hinzugefügt. Danach wird DeleteAllUser aufgerufen, nun sollte kein User mehr im Repository liegen. Beim Versuch auf einen der User mit der Methode Üser getUser(String token)äus dem Repository zu holen, wird eine ResponseStatusException geworfen.
  - IntitalizeAdminUser Test 1: Die Methode intializeAdminUser wird aufgerufen, da es noch keinen AdminUser gibt, wird einer neu angelegt. Um das zu überprüfen, prüfen wir, dass das angelegte Userobjekt nicht leer ist.

    Test 2: Die Methode initalizeAdminUser wird zweimal aufgerufen. Beim ersten Aufruf wird ein Admin User angelegt, beim zweiten Aufruf sollte nichts passieren, da bereits ein Admin User existiert.

2.3 Integration-Test

Die nachfolgenden Testszenarien laufen ausschließlich auf dem Server und sind dazu da die klassenübergreifende Funktionalitäten zu testen.

- Szenario 1 (MK60, MK 80, MK100)
  Vorbedingung: Es ist bereits User Alice im UserRepository. Dieser User ist in keiner Gruppe.

Ablauf: Der User erstellt eine Gruppe und verlässt sie anschließend wieder.
Nachbedingung: Der User ist in keiner Gruppe und die erstellte Gruppe hat sich gelöscht, als Alice, als letzter User ausgetreten ist.

- Szenario 2 (MK70, MK 80)
  Vorbedingung: Es existieren die User A,B und C. Es existiert eine Gruppe X die mit User A und B voll ist.
  Ablauf: User B verlässt die Gruppe, sodass Platz für User C frei wird. Dieser tritt dann der Gruppe bei.
  Nachbedingung: User A und C sind in Gruppe X und User B ist in keiner Gruppe.

- Szenario 3 (MK 100)
  Vorbedingung: Es existiert ein User A der in der Gruppe X ist.
  Ablauf: Es werden (um Mitternacht) alle Gruppen gelöscht.
  Nachbedingung: User A ist in keiner Gruppe und die Gruppe X existiert nicht mehr.

- Szenario 4 (MK110)
  Vorbedingung: Es existiert ein User A der in der Gruppe X ist und ein Admin User.
  Ablauf: Der User A wird von dem Admin User gelöscht.
  Nachbedingung: Weder User A noch Gruppe X existieren noch.

3 Client

3.1 Allgemeines zum Testen und Verbessern

Die meisten Fehler wurden bereits beim „Ausprobieren“ der App bemerkt und mit dem Debugger lokalisiert, sodass passende Verbesserungen durchgeführt werden konnten. Durch Unit- und Integration-Tests wurden kaum noch Fehler gefunden, dafür aber Funktionalität der Verbesserungen abgesichert. Im Folgenden sind daher die Funktionalitäten der wichtigsten Klassen nach den Verbesserungen und dazu jeweils die diese absichernden Tests aufgelistet. Anschließend werden die Ergänzungen und Veränderungen seit der letzten Phase aufgezeigt und die Motivation dahinter, soweit nicht noch beschrieben, erklärt. Für die Unit-Tests reichte die in Android-Studio standardmäßig installierte Software nicht aus. Grund dafür ist, dass diese Testsoftware zu viele verwendete komplexere Klassen stubt und mockt, um die Tests schneller und unabhängiger zu machen. Eine Klasse wie Pair<String, State>, die essentiell für die Kommunikation zwischen ViewModel und Activity ist, wird bei den Tests durch einen leeren Stub ersetzt, der das Testen unmöglich macht. Daher mussten weitere Frameworks benutzt werden.

3.2 Tools zum Testen

3.2.1 Robolectric

Robolectric ist ein Framework für Tests in Android Studio, das in Sachen Stubbing und Mocking von verwendeten Klassen geschickter vorgeht als der Standard in Android Studio. Klassen wie Pair werden nicht gestubt, sodass deren Funktionalität gewährleistet ist und sinnvolles Testen möglich wird. Daher wurde dieses Framework installiert.

3.2.2 Mockito und Powermockito

Bei Unit-Tests ist die Überschaubarkeit des Testbereichs wichtig, daher müssen Klassen und Funktionen, die zu weit aus dem Testbereich hinausführen, gemockt werden, wozu sich Mockito eignet. Was Mockito jedoch nicht kann, ist unter anderem, statische Klassen und Methoden zu mocken. Diese werden in MensaMeet bei Serveranfragen verwendet. Abhilfe schafft hier die Erweiterung Powermockito.

3.2.3 Espresso

Mit Espresso ist es möglich Tests in der laufenden App durchzuführen. Man ersetzt damit einen menschlichen Tester, der sich durch die App klickt und testet ob alles so

funktioniert wie es soll. Der Hintergrund dabei ist, dass man die Tests automatisiert durchführen möchte und man auch in der Lage sein will, ein Testszenario zu reproduzieren, um die Fehlerursache ausfindig zu machen. Espresso macht genau diese Dinge möglich und nimmt dem Programmierer so jede Menge Arbeit beim Testen ab.

3.3 Berücksichtigung des Testens bei der Implementierung

Beim Testen mit den obengenannten Tools zeigte sich, dass es sinnvoll sein kann, schon bei der Implementation den Code leichter testbar zu machen. Ein Hauptproblem war die Injektion eines gespyten oder gemockten ViewModels in eine Activity. Das ViewModel wird nicht über einen Konstruktor übergeben, sondern normalerweise in der Activity durch die Methode ViewModelProviders.of(this).get(ViewModel.class) geholt. Die Lösung dieses Problems bestand darin, den Aufruf dieser Methode vom weiteren Programmablauf zu trennen. Hierzu wurde der Inhalt der Activity-Lebenszyklus-Methode onCreate nach diesem Aufruf in die Methode onStart verlegt, welche vom System direkt nach onCreate aufgerufen wird. Robolectric bietet mit der Klasse ActivityController ein Tool an, um onCreate und onStart separat aufzurufen. Zwischen diesen Aufrufen kann nun eine Methode der Activity aufgerufen werden, die ein gemocktes oder gespytes ViewModel-Objekt entgegennimmt und als Attribut viewModel setzt. Diese Methode wurde in Activities, die sie benötigen, als setViewModel implementiert und mit der Annotation @VisibleForTesting versehen, damit sie nur in Tests verwendet werden kann.

3.4 Models

3.4.1 IdEnum

Das Interface IdEnum wurde neu eingeführt, um die Umwandlung von Enums mit Konstante-String-Paaren wie Gender, Status und Subject in SpinnerItem-Objekte für die Darstellung in Spinnern (Auswahllisten) zu erleichtern. Zuvor wurde nicht mit Wertepaaren gearbeitet (siehe MensaMeetItem -> E & V -> Handhabung von Wertepaaren mit representedValues und SpinnerItem), was problematisch war. Die obengenannten Enums und MealLines implementieren dieses Interface.

3.4.2 MensaMeetSession

Dies ist das Singleton, das alle wichtigen activityübergreifenden Daten einer Sitzung wie den aktuellen Benutzer und seine Linienauswahl speichert.
Elemente (Auswahl)

- Attribut user
  Dies ist der aktuelle Benutzer. Es ist genau dann null, wenn kein Benutzer eingeloggt ist, und kann in diesem Sinne abgefragt werden. Die Gruppe, der der Benutzer beigetreten ist, wird in dessen Attribut groupToken spezifiziert.

- Methode userDataIncomplete
  Sie gibt an, ob die Profildaten des aktuellen Benutzers noch nicht ausreichend spezifiziert sind, also ob eine der obligatorischen Angaben Name und Status noch fehlt.

- Attribut createdGroup
  Hier wird der Gruppenentwurf des Benutzers nach dem Verlassen von CreateGroupActivity gespeichert, auch wenn er unvollständig ist, damit der Benutzer ihn zu einem späteren Zeitpunkt weiter bearbeiten kann.

- Methode createdGroupDataIncomplete
  Sie überprüft, ob die Angaben des Gruppenentwurfs vollständig sind, sodass sie zum Server geschickt werden kann.

- Methode initialize
  Mit ihr werden die Daten in MensaMeetSession zu Beginn einer Sitzung initialisiert. Die meisten Daten werden auf null gesetzt, der Parameter user als user gesetzt, der Speiseplan vom Server geladen.

- Methode invalidate
  Das Gegenstück zu initialize, dass am Ende einer Sitzung alle Daten auf null setzt.

Ergänzungen und Verbesserungen seit der letzten Phase

- Verzicht auf Attribut chosenGroup
  Die vom Benutzer ausgewählte Gruppe wird nur noch als groupToken in seiner Klasse spezifiziert. Es ist nicht sinnvoll, das ganze Group-Objekt zu speichern, da es sich ständig durch neue Mitglieder oder durch Löschung verändern kann.

- Verzicht auf Attribut receivedGroup
  Zuvor wurden die passenden Gruppen, die bei SelectGroupActivity vom Server geladen werden, in der Session-Datei gespeichert. Dies ist jedoch nicht nötig, da sie außer in SelectGroupActivity nirgends mehr gebraucht werden.

- createdGroupDataIncomplete hinzugefügt

- initialize und invalidate hinzugefügt
  Dadurch werden Ein- und Ausloggen standardisiert.

- Prinzip “Kein aktuelles User-Objekt, keine Sitzung“
  Scheitert das Anfordern des User-Objekts für den aktuellen Benutzer vom Server, so wird die Sitzung sofort beendet. Zum einen kann es immer sein, dass der aktuelle Benutzer zwischenzeitlich gelöscht wurde, zum anderen ist sind ohne aktuelles User-Objekt keine sinnvollen Aktionen in der App mehr möglich. Der Benutzer müsste versuchen, sich neu einzuloggen.

3.5 Util-Klassen

3.5.1 HTTPUtil

Die Klasse HTTPUtil dient zur Kommunikation mit dem Server. Ihre Methoden entsprechen größtenteils bekannten Standard-Beispielen.
Ergänzungen und Verbesserungen seit der letzten Phase

- Weiterleitung der Serverantwort auch bei Fehler
  Die bisherige Implementation der Methode fetch lieferte nur die Serverantwort als JSON-String, wenn kein Fehler vorlag. Ansonsten wurde eine Exception geworfen und die Methode ohne lesen der Serverantwort beendet. Es war jedoch wünschenswert, den genauen Fehlercode und die Fehlermeldung in der App auswerten und gegebenenfalls an den Benutzer weiterleiten zu können. Daher würde die Methode dahingehend verändert, dass sie im Fehlerfall keine Exception, dafür aber den JSON-String der Serverantwort, der Fehlercode und -meldung weitergibt.

3.5.2 RequestUtil

Die Methoden der Klasse RequestUtil entsprechend den Diensten, die der MensaMeet-Server anbietet. Sie akzeptieren Objekte der Klassen, die verarbeitet werden sollen, serialisieren sie zu JSON und führen mit Hilfe von HTTPUtil die konkreten Serveranfragen durch. Die JSON-Antwort wird dann wieder zu den jeweiligen Objekten deserialisiert und an den Aufrufer der Methode zurückgegeben.

Elemente

- Methode createUser
- *util.RequestUtilTest.userTest()*
- Methode getUser
- *util.RequestUtilTest.userTest()*
- Methode updateUser
- *util.RequestUtilTest.userTest()*
- Methode deleteUser
- *util.RequestUtilTest.tearDown()*
- Methode createGroup
- *util.RequestUtilTest.groupTest()*

- Methode getGroup
- *util.RequestUtilTest.groupTest()*
- Methode getGroupByPrefferences
- *util.RequestUtilTest.groupTest()*
- Methode deleteGroup
- *util.RequestUtilTest.groupTest()*
- Methode addUserToGroup
- *util.RequestUtilTest.groupTest()*
- Methode removeUserFromGroup
- *util.RequestUtilTest.groupTest()*
- Methode getMensaData
- *util.RequestUtilTest.mensaDataTest()*
- Klasse GroupForRequest
  Diese Klasse ist nötig, da sich die Group-Klasse, die vom Server verlangt wird, von der im Client unterscheidet. Im Client sind die Zeiten zur als Date-Objekte, beim Server als Strings hinterlegt. Der Konstruktor und die Methode parseToGroup dienen zur Umwandlung.
- Klasse GroupForRequestWithToken
  Diese Klasse ist nötig, da beim Erstellen einer neuen Klasse ein Group-Objekt ohne das Attribut token an den Server übertragen muss, um von diesem ein Group-Objekt mit servergeneriertem Token zu erhalten. Diese Klasse enthält token, die Klasse GroupForRequest nicht.
- Klasse GroupWithPrefferences
  Diese Klasse entspricht dem JSON-Objekt, das bei einer Anfrage in getGroupBy-Prefferences dem Server übermittelt wird.
- Klasse RequestException *Diese Exception dient dazu, den Aufrufer einer Methode von RequestUtil bei einem vom Server verursachten oder sonstigen Fehler zu benachrichtigen und ihm Informationen zum Fehler zu übergeben. Ein Konstruktor liest aus dem aus der Serverantwort erzeugten JsonNode Fehlercode- und Fehlernachricht und speichert sie. Liegt kein Serverfehler vor, werden die Daten von eventuell auftretenden anderen Exceptions weitergeleitet.*

Ergänzungen und Verbesserungen seit der letzten Phase

- GroupForRequestWithToken
  Zuvor wurden keine servergenerieten tokens vom Server empfangen, da immer ein Group-Objekt mit Attribut token gesendet und dieses vom Server übernommen wurde. Ein servergeneriertes Token ist jedoch notwendig für die eindeutige Identifizierbarkeit einer erstellten Gruppe.

- Weiterleitung von Server- und anderen Exceptions
  Zuvor konnte man einen Fehler nur erkennen, wenn das von einer Methode in RequestUtil zurückgegebene Objekt nicht den Erwartungen entsprach. Dies war jedoch nicht eindeutig und aussagekräftig.

3.6 Activity-ViewModel-, List-ListHandler und Item-ItemHandler Paare

Die Activities sowie ihre Unterelemente Lists und Items sind für die Darstellung der Benutzeroberfläche zuständig, während die zugehörigen ViewModels und Handlers die Logik dahinter enthalten.

Die Activity-Viewmodel-Paare sind nach ihrem frühestmöglichen Erscheinen in einem Benutzungsdurchlauf geordnet. Die Beschreibung des Verhaltens geschieht in chronologischer Reihenfolge.

3.6.1 MensaMeetActivity und MensaMeetViewModel

MensaMeetActivity ist die abstrakte Oberklasse aller Activities in MensaMeet, MensaMeetViewModel die abstrakte Oberklasse aller ViewModels. Beide Klassen enthalten Attribute, Methoden und Vorgänge, die in mehreren oder allen Activities und ViewModels benötigt werden.

3.6.1.1 Elemente in MensaMeetActivity

- Methode onCreate
  Sie ruft onCreate der Oberklasse AppCompatActivity auf und sollte immer am Anfang der onCreate-Methode einer Unterklasse aufgerufen werden, da die Activity sonst teilweise nicht richtig dargestellt wird. Zudem enthält sie die Anweisung, dass die App immer vertikal angezeigt wird und nicht gedreht wird, da dabei unerwünschtes Verhalten auftreten kann, dessen Behandlung zusätzlichen Aufwand erfordert.

- Attribut viewModel
  Das zur Activity gehörende ViewModel

- Methode initializeViewModel
  Jede Unterklasse der Activity kann diese Methode an beliebiger Stelle benutzen, um dem Attribut viewModel das konkrete ViewModel zuzuweisen. Die Initialisierung des Attributs ist Vorbedingung für den Aufruf weiterer Methoden, die darauf zugreifen.

- Methode checkAccess
  Sie überprüft, ob die Zugangsbedingungen zur Activity erfüllt sind. Da sie leer ist und auch nicht standardmäßig in einer anderen Methode aufgerufen wird, ist sie eher als Anregung für die Unterklassen zu verstehen. In diesen Enthält sie neben einer Auswertung den Befehl, die Activity im negativen Fall zu beenden, wobei der Benutzer in die vorhergehende Activity im Activity-Stack zurückfällt. Sie gibt zurück, ob im Programm weitergefahren werden kann, sodass die Activity die aktuelle Methode sofort mit return beenden kann. Da sie meistens über das ViewModel auf MensaMeetSession zugreift, muss sie nach der Methode initalizeViewModel aufgerufen werden.

- Methode observeLiveData
  Sie beobachtet das SingleLiveData-Objekt eventLiveData des ViewModels und muss daher nach initializeViewModel aufgerufen werden. Das Objekt dient zur Kommunikation des ViewModels mit der Activity, das nach dem MVVM-Modell nicht direkt auf die Activity zugreifen darf. Falls das ViewModel eventLiveData ändert, wird die Activity benachrichtigt und kann den veränderten Wert auslesen, der eine Statusänderung und eine zugehörige Nachricht enthält. Zur Auswertung dieser Veränderung wird die Schablonenmethode processStateChange aufgerufen.

- Methode processStateChange
  Sie wird bei Veränderung von eventLiveData im ViewModel aufgerufen und wertet die Änderung aus, die einen neuen Status und eine zugehörige Nachricht enthält. Jede Activity kann sie individuell überschreiben.

- Methode showMessage
  Sie blendet eine Kombination aus einer gegebenen Stringressource und der über eventLiveData vom ViewModel übermittelten Nachricht, die zum Beispiel vom Server stammt, in Form eines Toasts (temporär angezeigtes Nachrichtenfeld am unteren Bildschirmrand) ein. Damit lassen sich zum Beispiel Serverfehler anzeigen.

- Attribute buttonHome, buttonNext und buttonBack
  Diese Buttons werden in den meisten Activities angezeigt.

- Methode initializeButtons
  Sie findet mit Hilfe der View-Ids die Buttons in der View der Activity und weist sie den entsprechenden Attributen zu. Daher sollte die Methode erst dann aufgerufen werden, wenn die View der Activity initialisiert wurde. Zudem weist sie den Buttons ClickListener zu, die auf die Methoden onClickHome, onClickNext und onClickBack verweisen.

- Methoden onClickHome, onClickNext und onClickBack
  Sie werden beim Klicken auf die jeweiligen Buttons aufgerufen und können von den Unterklassen überschrieben werden, um das Verhalten anzupassen

- Methode onResume
  Sie wird bei einer Rückkehr zur Activity aufgerufen, falls diese zwischenzeitlich in den Hintergrund getreteten ist. Selbst ruft sie wiederum onResume der Oberklasse AppCompatActivity auf und danach die Schablonenmethode reloadData

- Methoden gotoActivity und gotoHome
  Starten eine andere Activity und leiten den Benutzer an sie weiter.

- Methode onBackPressed
  Sie wird vom System bei Betätigung des Android-Back-Buttons aufgerufen und leitet zur Activity weiter, die die aktuelle Activity gestartet hat. Dies ist manchmal jedoch nicht erwünscht, die Navigation in der App soll ausschließlich über die eigenen Buttons erfolgen. Daher wird die Methode leer überschrieben und der Button generell deaktiviert.

- Methode goBack
  Sie ruft onBackPressed in der Oberklasse auf und stellt somit deren Funktion zur Verfügung, falls sie doch einmal benötigt wird (siehe ShowUserActivity).

3.6.1.2 Elemente in MensaMeetViewModel

- Attribut singleLiveData
  Das SingleLiveData-Objekt, das von der Activity beobachtet wird (siehe oben).

- Weiteren Methoden der Klasse
  Sie rufen jeweils entsprechende Methoden in MensaMeetSession auf. Grund dafür ist, dass nach dem MVVM-Modell die Activity nicht direkt mit dem Model kommunizieren, sondern ihre Daten vom ViewModel beziehen sollte.

3.6.1.3 Ergänzungen und Verbesserungen seit der letzten Phase

- initializeViewModel, um die Zuweisung des ViewModel zu regulieren
  Zuvor wurde das ViewModel direkt super.viewModel zugewiesen.

- checkAccess, um mehr ungültige Zustände zu vermeiden

- SingleLiveEvent eventLiveData von Pair<MensaMeetViewModel, StateInterface> in Pair <String, StateInterface> geändert
  Die vorherige Lösung stammt aus der Android-Schulung, damit das sendende ViewModel immer als Quelle mitgeliefert wird. In den Anwendungsfällen von MensaMeet ist das ViewModel jedoch immer bekannt und braucht nicht mitgeliefert zu werden. Stattdessen ist es sinnvoller, eine eventuelle Fehlernachricht als String mitzuliefern.

- observeLiveData, um die Beobachtung von eventLiveData im ViewModel an beliebiger Stelle initialisieren zu können
  Zuvor war ihr Inhalt Teil der Methode onCreate, wodurch diese als super.onCreate erst nach Initialisierung des ViewModel aufgerufen werden konnte. Sie sollte jedoch immer als erstes in onCreate der Unterklassen aufgerufen werden, da es sonst zu Darstellungsproblemen wie dem Fehlen eines Hintergrundbildes kommen kann

- processStateChange nun in jeder Activity einheitlich genutzt
  Zuvor gab es bei manchen Activities individuelle Lösungen.

- initializeButtons
  Ihr Inhalt war zuvor ebenfalls Teil der Methode onCreate. Wenn diese jedoch als erstes in onCreate der Unterklassen aufgerufen werden soll, käme die Initialisierung der Buttons vor der Initialisierung der View der Activity, was unmöglich ist, daher die Auslagerung in eine eigene Methode.

- showMessage
  Sie bündelt die oft benutzte Anweisungsfolge.

- Deaktivierung des Android-Back-Buttons durch leere Methode onBackPressed

- MVVM-konformer Zugriff auf MensaMeetSession nur noch über ViewModel

- Attribut displayMode bei ViewModel entfernt
  Ursprünglich wurde angenommen, dass der displayMode auch für die Logik im ViewModel entscheidend sein könnte. Dies hat sich jedoch nicht bestätigt, weshalb das unnötige Attribut entfernt wurde.

- Methodenbeendigung nach jeder eventLiveData-Änderung
  Zuvor wurde fälschlicherweise angenommen, dass die Methode und alle weiteren Methoden sofort beendet wurde, sobald der Beobachter benachrichtigt wird. Dies ist jedoch nicht der Fall, daher müssen die Methoden explizit beendet werden.

- Methode reloadData entfernt
  Sie diente als Auslagerungsmethode für die Ladung der Daten und wurde sowohl einmal am Anfang in onCreate als auch in onResume, d.h. immer bei der Rückkehr zur Activity vom System standardmäßig aufgerufen. Da jedoch onResume auch am Anfang nach onCreate und onStart aufgerufen wird, entspricht die Funktion dieser Methode eigentlich der von reloadData und macht letztere überflüssig, der Inhalt von reloadData wurde in onResume transferiert.

3.6.2 BeginActivity und BeginViewModel

BeginActivity ist die erste Activity, die dem Benutzer nach dem Starten der App angezeigt wird. BeginViewModel enthält die Methoden, die beim Klicken der Buttons aufgerufen werden und wiederum bei der Activity den Seitenwechsel triggern (siehe MensaMeetActivity -> Elemente in MensaMeetActivity -> observeLiveData).

3.6.2.1 Sichtbare Elemente

- Login (Button)
- Registrieren (Button)

3.6.2.2 Verhalten

- Die Activity ist nicht zugreifbar, wenn der Benutzer eingeloggt ist.
  view.BeginActivityTest.onCreate_userNotNull_ActivityFinishes()
- Bei Klick auf Login kommt man zu LoginActivity
  viewmodel.BeginViewModelTest.toLoginSuccess()
- Bei Klick auf Registrieren kommt man zu RegisterActivity.
  viewmodel.BeginViewModelTest.toRegisterSuccess()

3.6.2.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)

3.6.3 LoginActivity und LoginViewModel

Auf LoginActivity kann sich der Benutzer mit seinen Firebase-Login-Daten bei MensaMeet einloggen. LoginViewModel kommuniziert dabei mit dem Firebase-Projekt von MensaMeet und dem Server.

3.6.3.1 Sichtbare Elemente

- E-Mail-Adresse (Textfeld)
- Passwort (Textfeld)
- Login (Button)
- Zurück (Button)

3.6.3.2 Verhalten

- Die Activity ist nicht zugreifbar, wenn der Benutzer eingeloggt ist.
  view.LoginActivityTest.onCreate_userNotNull_ActivityFinishes()
- Bei Klick auf Zurück gelangt der Benutzer auf BeginActivity.
  view.LoginActivityTest.clickBack_startBeginActivity()
- Bei Klick auf Login wird der Login-Prozess beim ViewModel gestartet.

- Falls ein Textfeld leer ist, Fehlermeldung.
  view.LoginActivityTest.clickLogin_oneEditTextEmpty_errorMessage()
- Mit den Daten wird bei Firebase das Einloggen versucht.
- Bei erfolgreichem Einloggen wird ein User-Token empfangen. Mit diesem wird beim Server das entsprechende User-Objekt angefordert.
- Bei gescheitertem Einloggen wird die Fehlermeldung von Firebase angezeigt.
- Wird ein User-Objekt vom Server empfangen, wird damit in MensaMeetSession versucht, die neue Sitzung initialisiert.
- Wird kein User-Objekt empfangen, wird der Benutzer bei Firebase wieder ausgeloggt und die Fehlermeldung des Servers angezeigt. Grundsatz: kein aktuelles User-Objekt, keine Sitzung. Der Benutzer verbleibt in der Activity.
- Ist die Initialisierung der neuen Sitzung erfolgreich, wird das aktuelle User-Objekt betrachtet.
- Ist die Initialisierung der neuen Sitzung nicht erfolgreich, was am Scheitern des Ladens der Mensalinien liegt, verleibt der Benutzer unangemeldet in der Activity. Er kann dann versuchen, sich erneut einzuloggen.
  view.LoginActivityTest.loginSuccess_initializationFails_notLoggedIn()
- Sind die Angaben im User-Objekt ausreichend (Name und Status dürfen nicht leer sein), wird der Benutzer zu HomeActivity weitergeleitet.
  view.LoginActivityTest.loginSuccess_userDataComplete_startHomeActivity()
- Sind die Angaben im User-Objekt nicht ausreichend, wird der Benutzer zu UserActivity weitergeleitet, damit er sie dort ergänzen kann.
  view.LoginActivityTest.loginSuccess_userDataIncomplete_startUserActivity()

3.6.3.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)
- Initialisierung der Sitzung bereits hier
- Verhalten beim Scheitern der Server-Anfragen
- Zurück-Button (siehe MensaMeetActivity -> E&V -> Deaktivierung des Android-Zurück-Buttons)
- Richtiger Umgang mit leeren Eingabefeldern
  Zuvor stürzte die App immer ab. Grund war, dass leere Eingabefelder nicht erkannt wurden, da sie immer standardmäßig einen Leerstring enthalten und nicht wie erwartet null. Der Leerstring wurde an Firebase übermittelt, wodurch es zu einer Exception und dem Absturz der App kam. Dies wurde behoben, indem auch der Leerstring erkannt wird.

- Methode matchPattern zur Datenüberprüfung im ViewModel entfernt
  In der danach aufgerufenen Methode login findet auch eine Überprüfung statt.
- Fehlermeldung von Firebase nun an Benutzer übermittelt

Anmerkungen zum Testen
Das automatische Testen gestaltete sich in diesem Fall schwierig, da im ViewModel mit verschachtelten Interface-Implementationen gearbeitet wird. Auch die Abhängigkeit von Firebase erschwert das Testen und die fehlende Möglichkeit, private Attribute zuzugreifen.

3.6.4 RegisterActivity und RegisterViewModel

Auf RegisterActivity kann sich der Benutzer mit Hilfe von Firebase bei MensaMeet als neuer Benutzer registrieren. RegisterViewModel kommuniziert dabei mit dem Firebase-Projekt von MensaMeet und dem Server.

3.6.4.1 Sichtbare Elemente

- E-Mail-Adresse (Textfeld)
- Passwort (Textfeld)
- Passwortwiederholung (Textfeld)
- Registrieren (Button)
- Zurück (Button)

3.6.4.2 Verhalten

- Die Activity ist nicht zugreifbar, wenn der Benutzer eingeloggt ist.
  view.RegisterActivityTest.onCreate_userNotNull_ActivityFinishes()
- Bei Klick auf Zurück gelangt der Benutzer auf BeginActivity.
  view.RegisterActivityTest.clickBack_startBeginActivity()
- Bei Klick auf Registrieren wird der Registrierungsprozess beim ViewModel gestartet.
- Falls ein Textfeld leer ist oder die Passwörter nicht übereinstimmen, Fehlermeldung.
  view.RegisterActivityTest.clickLogin_passwordsNotEqual_errorMessage()
- Mit den Daten wird bei Firebase die Registrierung versucht.
- Bei erfolgreicher Registrierung wird ein User-Token empfangen. Mit diesem wird beim Server ein entsprechendes User-Objekt erzeugt.

- Bei gescheiterter Registrierung wird die Fehlermeldung von Firebase angezeigt.
- Ist die Erzeugung des User-Objekt erfolgreich, wird es vom Server angefordert.
- st die Erzeugung des User-Objekts nicht erfolgreich, wird der Benutzer bei Firebase ausgeloggt und die Fehlermeldung des Servers angezeigt. Achtung: Wenn der Benutzer sich mit seinen Firebase-Daten nun einloggt, gelingt dies zwar bei Firebase, es wir jedoch kein User-Objekt vom Server zu bekommen sein. Es darf auch nicht versucht werden, ein neues User-Objekt anzulegen, da es auch sein kann, dass der Benutzer absichtlich beim Server gelöscht wurde. Momentan ist es nur möglich, den Benutzer manuell bei Firebase zu löschen. Die Benutzerverwaltung mit Firebase war jedoch kein Musskriterium und soll zu einem späteren Zeitpunkt vervollständigt werden.
- Wird das User-Objekt vom Server empfangen, wird damit in MensaMeetSession versucht, die neue Sitzung zu initialisieren.
- Wird kein User-Objekt empfangen, wird der Benutzer bei Firebase wieder ausgeloggt und die Fehlermeldung des Servers angezeigt. Grundsatz: kein aktuelles User-Objekt, keine Sitzung. Der Benutzer verbleibt in der Activity.
- Ist die Initialisierung der neuen Sitzung erfolgreich, wird der Benutzer zu UserActivity weitergeleitet, damit er seine MensaMeet-Benutzerdaten eingeben kann.
  view.RegisterActivityTest.loginSuccess_startUserActivity()
- Ist die Initialisierung der neuen Sitzung nicht erfolgreich, was am Scheitern des Ladens der Mensalinien liegt, wird der Benutzer uneingeloggt zu BeginActivity weitergeleitet. Er kann dann versuchen, sich einzuloggen.
  view.RegisterActivityTest.loginSuccess_initializationFails_startBeginActivity()

3.6.4.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)
- Initialisierung der Sitzung bereits hier
- Verhalten beim Scheitern der Server-Anfragen
- Zurück-Button (siehe MensaMeetActivity -> E&V -> Deaktivierung des Android-Zurück-Buttons)
- Richtiger Umgang mit leeren Eingabefeldern
  Zuvor stürzte die App immer ab. Grund war, dass leere Eingabefelder nicht erkannt wurden, da sie immer standardmäßig einen Leerstring enthalten und nicht wie erwartet null. Der Leerstring wurde an Firebase übermittelt, wodurch es zu einer Exception und dem Absturz der App kam. Dies wurde behoben, indem auch der Leerstring erkannt wird.

- Fehlermeldung an Benutzer, falls Passwörter nicht identisch
- Fehlermeldung von Firebase nun an Benutzer übermittelt

Anmerkungen zum Testen
Das automatische Testen gestaltete sich in diesem Fall schwierig, da im ViewModel mit verschachtelten Interface-Implementationen gearbeitet wird. Auch die Abhängigkeit von Firebase erschwert das Testen und die fehlende Möglichkeit, private Attribute zuzugreifen.

3.6.5 UserActivity und UserViewModel

Auf UserActivity kann der Benutzer die von ihm veränderbaren Daten seines MensaMeet-Benutzerprofils bearbeiten. UserViewModel lädt und speichert die Daten beim Server.

3.6.5.1 Sichtbare Elemente

- Auswahlbereich für Benutzerbild
- Name (Textfeld)
- Motto (Textfeld)
- Geburtsdatum (Textfeld mit Auswahldialog)
- Geschlecht (Auswahlliste)
- Status (Auswahlliste)
- Fachrichtung (Auswahlliste)
- Verwerfen (Button), nicht immer angezeigt (siehe Verhalten)
- Home (Button)
- Speichern (Button)

3.6.5.2 Verhalten

- Die Activity ist nur zugreifbar, wenn der Benutzer eingeloggt ist.
  view.UserActivityTest.onCreate_userNull_ActivityFinishes()
- Falls die Benutzerdaten nicht ausreichend sind (Name oder Status leer), wird der Verwerfen-Button nicht angezeigt. Der Benutzer kann die Änderungen zwar immer noch über den Home-Button verwerfen, soll aber dazu animiert werden, seine Daten zu vervollständigen und zu speichern.
  view.UserActivityTest.userDataIncomplete_noDiscardButton()

- Bei (Wieder-)Eintritt in die Activity wird vom ViewModel das User-Objekt des Benutzers jedes Mal neu vom Server geladen und in MensaMeetSession gespeichert.
  viewmodel.UserViewModelTest.reloadingUserFailed()

- Scheitert das Laden des User-Objekts vom Server, wird die Sitzung beendet und auf BeginActivity weitergeleitet. Grundsatz: Kein aktuelles User-Objekt, keine Sitzung.
  view.UserActivityTest. loadUserFailed_logout()

- Die Activity bindet ein UserItem im DisplayMode BIG_EDITABLE ein, in dem alle Elemente außer der Navigationsbuttons enthalten sind.

- Der Auswahlbereich für das Benutzerbild, ein UserPictureItem im DisplayMode BIG_EDITABLE, zeigt das aktuelle Benutzerbild an. Falls noch keins ausgewählt wurde, wird das Default-Bild angezeigt.

- Bei Klick auf den Auswahlbereich für das Benutzerbild öffnet sich eine Liste von Benutzerbildern. Wird eins angeklickt, wir dieses als neues Benutzerbild ausgewählt.

- Ist noch kein Geburtsdatum spezifiziert, wird ein Platzhaltertext angezeigt.

- Bei Klick auf das Feld des Geburtsdatums wird ein Datum-Auswahldialog geöffnet.

- Bei Klick auf Verwerfen gelangt der Benutzer auf HomeActivity, von wo er in diesem Fall gekommen sein muss. Es werden keine Daten gespeichert.
  view.UserActivityTest.clickDiscard_startHomeActivityAndNoDataSaved()

- Bei Klick auf Home gelangt der Benutzer auf HomeActivity, es werden keine Daten gespeichert.
  view.UserActivityTest.clickHome_startHomeActivityAndNoDataSaved()

- Bei Klick auf Speichern wird der Speicherprozess im ViewModel gestartet. Es wird versucht, das User-Objekt auf dem Server upzudaten.
  viewmodel.UserViewModelTest.saveUserFailed(), viewmodel.UserViewModelTest.saveUserSuccess(), viewmodel.UserViewModelTest.saveUserIsNull()

- Ist das Update des User-Objekts erfolgreich, wird auf HomeActivity weitergeleitet.

- Ist das Update des User-Objekts nicht erfolgreich, wird die Sitzung beendet und auf BeginActivity weitergeleitet. Grundsatz: kein aktuelles User-Objekt, keine Sitzung
  view.UserActivityTest.updateUserSuccess_startHomeActivity()

3.6.5.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)

- Unvollständigkeit der Benutzerdaten nun erlaubt
  Dass die Benutzerdaten ausreichend sind (Name und Status nicht leer), ist Voraussetzung, um in der App ab der Linienauswahl fortfahren zu können. Im Fall, das die Benutzerdaten nicht ausreichend sind, musste der Benutzer sie früher in UserActivity ausreichend ergänzen, um weiterfahren zu können, entsprechend gab es nur den Speichern-Button und es wurden die Daten überprüft. Nun wurde aber doch ein der Home-Button ergänzt und ausreichende Daten werden hier nicht mehr verlangt. Hierdurch soll der Benutzer die Freiheit bekommen, die Benutzerdaten zu einem späteren Zeitpunkt zu ergänzen. Von HomeActivity gelangt er nun mit nicht ausrechenden Daten auch nicht mehr weiter zu SelectLinesActivity, sondern wird wieder auf UserActivity umgeleitet.

- Verhalten beim Scheitern der Server-Anfrage

- Sofortige Beendigung nach checkAccess
  Gefunden durch view.UserActivityTest.onCreate_userNull_ActivityFinishes() Nach dem Befehl finish() lief die Activity weiter zu einem Befehl, der einen Mock erfordert und dadurch eine Exception produzierte. Grund: finish() beendet die aktuelle Methode nicht sofort. Lösung: checkAccess erhielt einen Boolean-Rückgabewert, mit dem die Activity entscheiden kann, ob die aktuelle Methode sofort mit return beendet wird.

- Weiterleitung nach Scheitern des Ladens des Users korrigiert
  Gefunden durch view.UserActivityTest.loadUserFailed_logout() Statt auf BeginActivity wurde auf HomeActivity weitergeleitet.

3.6.6 HomeActivity und HomeViewModel

HomeActivity ist die zentrale Seite des Benutzers im eingeloggten Zustand, von wo aus er zur Linienauswahl und der Bearbeitung seines Benutzerprofils gelangen und sich wieder abmelden, d. h. ausloggen kann. HomeViewModel überprüft unter anderem, ob Activity-Wechsel zulässig sind und führt den Logout durch.

3.6.6.1 Sichtbare Elemente

- Essen gehen (Button)

- Dein Profil (Button)

- Abmelden (Button

3.6.6.2 Verhalten

- Die Activity ist nur zugreifbar, wenn der Benutzer eingeloggt ist
  view.HomeActivityTest.start_userNull_ActivityFinishes()

- Bei Klick auf Essen gehen wird vom ViewModel überprüft, ob die Benutzerdaten ausreichen. Falls nein wird auf UserActivity weitergeleitet.
  viewmodel.HomeViewModelTest.goEatToUserPage()

- Ansonsten wird überprüft, ob der aktuelle Benutzer ein Grouptoken und damit eine Gruppe hat, in der er Mitglied ist. Falls ja, wird er auf GroupJoinedActivity weitergeleitet, falls nein auf SelectLinesActivity.
  viewmodel.HomeViewModelTest.goEatToUserPage(), viewmodel.HomeViewModelTest.goEatToSelectL
  viewmodel.HomeViewModelTest.goEatToJoinedGroupPage()

- Bei Klick auf Dein Profil wird auf UserActivity weitergeleitet.
  view.HomeActivityTest.clickYourProfile_startUserActivity()

- Bei Klick auf Abmelden wird die Sitzung invalidiert und auf BeginActivity umgeleitet.
  view.HomeActivityTest. clickLogout_logout()

3.6.6.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)

- Überprüfung der Benutzerdaten bei Klick auf Essen gehen (siehe UserActivity)

- Invalidierung der Sitzung beim Abmelden

- Weiterleitung korrigiert
  Gefunden durch viewmodel.HomeViewModelTest.goEatToUserPage() Bei nicht ausreichenden Benutzerdaten wurde nicht auf UserActivity weitergeleitet. Grund: Nach der Auswertung wurde die Methode nicht beendet, sondern lief zur Auswertung des groupTokens weiter. Deren Ergebnis war dann ausschlaggebend für die Weiterleitung. Lösung: return nach erster Auswertung ergänzt.

3.6.7 SelectLinesActivity und SelectLinesViewModel

Auf SelectLinesActivity kann der Benutzer eine oder mehrere Mensalinien auswählen, an denen er essen möchte. SelectLinesViewModel sorgt für die lokale Zwischenspeicherung der Auswahl beim Verlassen der Activity und ihr Laden bei einem erneuten Besuch.

3.6.7.1 Sichtbare Elemente

- Mensalinien mit Essen (Liste)

- Zurück (Button)

- Home (Button)

- Weiter (Button)

3.6.7.2 Verhalten

- Die Activity ist nur zugreifbar, wenn der Benutzer eingeloggt ist und ausreichende Benutzerdaten angegeben sind.
  view.SelectLinesActivityTest.start_userIncomplete_ActivityFinishes()
- Die Activity bindet eine LineList ein, die wiederum LineItems im DisplayMode SMALL enthält.
- Initialisiert wird die Liste mit dem Mensadaten aus MensaMeetSession.
- Beim Klicken auf eine Linie wird diese markiert und dabei dunkel hinterlegt, beim erneuten Klicken darauf wird die Markierung wieder entfernt.
- Bei (Wieder-)Eintritt in die Activity wird vom ViewModel überprüft, ob in MensaMeetSession bereits eine Linienauswahl gespeichert ist, die in die Mensalinien-Liste geladen wird.
- Die Activity bindet eine LineList ein, die wiederum LineItems im DisplayMode SMALL enthält.
- Beim Klicken auf eine Linie wird diese markiert und dabei dunkel hinterlegt, beim erneuten Klicken darauf wird die Markierung wieder entfernt.
  view.SelectLinesActivityTest.start_chosenLinesExist_selectInList()
- Es können mehrere Linien gleichzeitig markiert werden.
- Bei Klick auf Weiter wird überprüft, ob mindestens eine Linie ausgewählt wurde. Falls nicht, Fehlermeldung, der Benutzer verbleibt in der Activity. Ansonsten wird die Auswahl dem ViewModel übergeben, das sie in MensaMeetSession speichert. Anschließend wird SetTimeActivity aufgerufen.
  viewmodel.SelectLinesViewModelTest.noLinesSelected(), viewmodel.SelectLinesViewModelTest.linesSe
- Bei Klick auf Home oder Zurück wird die Auswahl, auch wenn sie leer ist, in MensaMeetSession gespeichert und HomeActivity aufgerufen.

3.6.7.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)
- Funktionalität des Zurück-Buttons ergänzt

3.6.8 SetTimeActivity und SetTimeViewModel

In SetTimeActivity kann der Benutzer die Zeit oder den Zeitraum auswählen, in der er sich zum Essen treffen möchte. SetTimeViewModel sorgt für die lokale Zwischenspeicherung der Angaben beim Verlassen der Activity und ihr erneutes Laden bei einer Rückkehr.

3.6.8.1 Sichtbare Elemente

- Endzeit (Textfeld mit Auswahldialog)
- Startzeit (Textfeld mit Auswahldialog)
- Zurück (Button)
- Home (Button)
- Weiter (Button)

3.6.8.2 Verhalten

- Die Activity ist nur zugreifbar, wenn der Benutzer eingeloggt ist und ausreichende Benutzerdaten angegeben sind.
  view.SetTimeActivityTest.start_userIncomplete_ActivityFinishes()
- Bei (Wieder-)Eintritt in die Activity wird vom ViewModel überprüft, ob in MensaMeetSession bereits eine Zeitintervall-Auswahl gespeichert ist, die in die Mensalinien-Liste geladen wird.
  view.SetTimeActivityTest.start_chosenTimeExist_load()
- Wurde noch keine Start- oder Endzeit ausgewählt, wird jeweils 12:00 angezeigt.
- Bei Klick auf Start- oder Endzeit wird ein Zeit-Auswahldialog geöffnet.
- Wurde eine Start- oder Endzeit ausgewählt im Auswahldialog und dieser geschlossen, wird überprüft, ob die Endzeit nach der Startzeit liegt. Ist dies nicht der Fall, ist die Auswahl wirkungslos.
  view.SetTimeActivityTest.chooseStartTime_afterEndTime_noEffect()
- Bei Klick auf Weiter wird das Zeitintervall dem ViewModel übergeben, das es in MensaMeetSession speichert. Anschließend wird SelectGroupActivity aufgerufen.
  viewmodel.SetTimeViewModelTest.saveTimeAndNextSuccess()
- Bei Klick auf Zurück wird das Zeitintervall dem ViewModel übergeben, das es in MensaMeetSession speichert. Anschließend wird SelectLinesActivity aufgerufen
  viewmodel.SetTimeViewModelTest.saveTimeAndBackSuccess()
- Bei Klick auf Home wird das Zeitintervall dem ViewModel übergeben, das es in MensaMeetSession speichert. Anschließend wird HomeActivity aufgerufen
  view.SetTimeActivityTest.clickHome_saveAndStartSelectHomeActivity()

3.6.8.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)
- Speicherung der Zeit auch bei Klick auf Home
  Gefunden durch view.SetTimeActivityTest.clickHome_saveAndStartSelectHomeActivity()
  Für diesen Fall war keine Speicherung implementiert.

3.6.9 SelectGroupActivity und SelectGroupViewModel

In SelectGroupActivity werden dem Benutzer die Gruppen angezeigt, die zu seiner Linien- und Zeitauswahl passen. Er kann einer Gruppe beitreten oder zu CreateGroupActivity wechseln, wo er selbst eine erstellen kann. SelectGroupViewModel ist für das Laden der Gruppen vom Server zuständig, jedoch nicht für den Gruppenbeitritt, der im GroupItemHandler der GroupItems realisiert wird.

3.6.9.1 Sichtbare Elemente

- Passende Gruppen (Liste), enthält pro Eintrag:
  - Name (Textanzeige
  - Motto (Textanzeige)
  - Zeit (Textanzeige)
  - Linie (Textanzeige)
  - Mitgliederzahl/Maximale Mitgliederzahl (Textanzeige)
  - Beitreten (Button), falls Gruppe nicht voll
  - Gruppe löschen (Button), falls aktueller Benutzer Administrator
  - Mitglieder (Liste), enthält pro Eintrag:
    - Benutzerbild
    - Name (Textanzeige)
    - Motto (Textanzeige)
    - Benutzer löschen (Button), falls aktueller Benutzer Administrator und nicht dem Benutzer identisch
- Neue Gruppe Erstellen (FloatingActionButton)
- Zurück (Button)
- Home (Button)

3.6.9.2 Verhalten

- Die Activity ist nur zugreifbar, wenn der Benutzer eingeloggt ist und ausreichende Benutzerdaten angegeben sind.
- Bei (Wieder-)Eintritt in die Activity wird vom ViewModel jedes Mal überprüft, ob ausgewählte Linien und eine ausgewählte Zeit in MensaMeetSession ungleich null sind, falls ja, eine neue Anfrage nach den passenden Gruppen an den Server gesendet und die erhaltenen Gruppen gespeichert.
  viewmodel.SelectGroupViewModelTest.noTimeChosen(), viewmodel.SelectGroupViewModelTest.noLin
  viewmodel.SelectGroupViewModelTest.loadingGroupFailed()

- Falls Linien oder Zeit gleich null sind, Fehlermeldung, Verbleib in Activity.
- Falls Serveranfrage nach den Gruppen fehlschlägt, Fehlermeldung, Verbleib in Activity.
- Die Activity bindet eine GroupList, die wiederum GroupItems im DisplayMode SMALL enthält. Jedes GroupItem enthält wiederum eine UserList mit UserItems im DisplayMode SMALL. Ein UserItem enthält zudem für die Darstellung des Benutzerbildes ein UserPictureItem im DisplayMode SMALL. Die Logik für die Items befindet sich jeweils in einem GroupItemHandler bzw. UserItemHandler.
- Zur Kommunikation mit dem Group- und UserItems beobachtet die Activity deren eventLiveData-Elemente (siehe MensaMeetActivity -> Elemente in MensaMeetActivity -> observeLiveData).
- Die Gruppen werden zunächst in einer Kurzdarstellung angezeigt. Klickt man auf sie, klappen sie auf und zeigen die Buttons sowie die Mitgliederliste.
- Es wird die Anzahl der Gruppenmitglieder ermittelt und neben der maximalen Mitgliederanzahl angezeigt.
- Bei Klick auf Beitreten in einer Gruppe sendet der GroupItemHandler eine Beitrittsanfrage mit dem aktuellen Benutzer an den Server.
- Falls die Beitrittsanfrage erfolgreich ist, wird das User-Objekt des aktuellen Benutzers, das nun mit der Gruppenzugehörigkeit aktualisiert ist, beim Server angefordert.
- Falls das User-Objekt aktualisiert werden konnte, wird die Activity über eventLiveData benachrichtigt, die dann zu GroupJoinedActivity wechselt.
- Falls die Beitrittsanfrage scheitert, Fehlermeldung, Benutzer verleibt in Activity.
- Falls das User-Objekt nicht aktualisiert werden konnte, wird die Sitzung beendet und auf BeginActivity weitergeleitet. Grundsatz: kein aktuelles User-Objekt, keine Sitzung.
- Bei Klick auf Gruppe löschen sendet der GroupItemHandler eine Löschanfrage an den Server.
- Falls die Löschanfrage erfolgreich ist, wird überprüft, ob der aktuelle Benutzer in der Gruppe war. Falls ja, wird sein User-Objekt neu vom Server angefordert. Ansonsten wird die Activity neu gestartet, damit die Gruppen neu geladen werden.
- Falls das User-Objekt aktualisiert werden konnte, wird die Activity ebenfalls neu gestartet. Ansonsten wird die Sitzung beendet und auf BeginActivity weitergeleitet. Grundsatz: kein aktuelles User-Objekt, keine Sitzung.
- Bei Klick auf Benutzer löschen wird eine Löschanfrage an den Server gesendet.

- Falls die Löschanfrage erfolgreich ist, wird die Activity neu gestartet. Ansonsten, Fehlermeldung.
- Bei Klick auf den Bereich eines Benutzers wird dieser detailliert angezeigt. Hierzu wird sein User-Objekt in MensaMeetSession als userToShow hinterlegt und ShowUserActivity aufgerufen.
- Bei Klick auf Neue Gruppe erstellen wird auf CreateGroupActivity weitergeleitet.
- Bei Klick auf Zurück wird auf SetTimeActivity weitergeleitet.
- Bei Klick auf Home wird auf HomeActivity weitergeleitet.

3.6.9.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)
- Überprüfung, ob Linien und Zeit in MensaMeetSession ungleich null sind
- Verhalten beim Scheitern der Server-Anfragen
- Erneutes laden der Daten vom Server bei jedem Eintritt in die Activity
  Zuvor wurden die Daten nur geladen, wenn die Activity neu gestartet wurde, nun werden sie auch in der Methode relaodData, die bei jedem Wiedereintritt in die Activity aufgerufen wird, neu geladen. Dies ist sinnvoll, da sich Gruppendaten durch neue Mitglieder häufig ändern können.
- Methode loadGroups im ViewModel gibt nun ein Boolean zurück
  Mit false wird der aufrufenden Methode in der Activity signalisiert, dass ein Fehler aufgetreten ist. Diese Methode kann sich dann sofort beenden, was nötig ist, dass erst dann in der Activity der Beobachter der veränderten eventLiveData mit Fehlerstatus und Fehlerstring benachrichtigt wird und den Fehler auswerten kann.

Hinweis: Administrator-Status
Um sich im Administrator-Status einzuloggen, kann das Konto admin@mensameet.de, Passwort adminadmin verwendet werden.

3.6.10 CreateGroupActivity und CreateGroupViewActivity

In CreateGroupActivity kann der Benutzer eine neue Gruppe erstellen. CreateGroupViewModel überträgt die Daten an MensaMeetSession zur Zwischenspeicherung oder an den Server.

3.6.10.1 Sichtbare Elemente

- Name (Textfeld)
- Motto (Textfeld)
- Zeit (Textfeld mit Auswahldialog)
- Linie (Textfeld mit Auswahldialog)
- Maximale Mitgliederanzahl (Auswahlliste)
- Zurück (Button)
- Home (Button)
- Weiter (Button)

3.6.10.2 Verhalten

- Die Activity ist nur zugreifbar, wenn der Benutzer eingeloggt ist und ausreichende Benutzerdaten angegeben sind.
- Bei (Wieder-)Eintritt in die Activity wird vom ViewModel überprüft, ob in MensaMeetSession bereits einen Gruppenentwurf gespeichert ist, der dann geladen wird. Falls nicht, wird als Linie die erste der in SelectLinesActivity auswählten Linien eingestellt und als Zeit die in SetTimeActivity ausgewählte Startzeit
- Die Activity bindet ein GroupItem im DisplayMode BIG_EDITABLE ein, das alle Elemente außer der Navigation enthält.
- Bei Klick auf Weiter speichert das ViewModel den Gruppenentwurf in MensaMeetSession und führt beim Server eine Anfrage zur Erstellung der Gruppe durch. *viewmodel.CreateGroupViewModelTest.noCreatedGroup(), viewmodel.CreateGroupViewModelTest.cre viewmodel.CreateGroupViewModelTest.addUserToGroupFails(), viewmodel.CreateGroupViewModelTe*
- Falls die Erstellung der Gruppe erfolgreich ist, wird eine Beitrittsanfrage in die Gruppe für den aktuellen Benutzer an den Server gesendet.
- Falls die Erstellung der Gruppe scheitert, Fehlermeldung, Benutzer verbleibt in Activity.
- Falls die Beitrittsanfrage in die Gruppe erfolgreich ist, wird das User-Objekt erneut beim Server angefordert.
- Falls die Beitrittsanfrage scheitert, Fehlermeldung, Benutzer wird auf SelectGroupActivity zurückgeleitet. Die Existenz einer Gruppe ohne Mitglieder ist nicht problematisch, der Beitritt kann auch zu einem späteren Zeitpunkt vom Benutzer versucht werden.

- Falls das User-Objekt aktualisiert werden konnte, war der gesamte Gruppenerstellungsprozess erfolgreich und es wird auf GroupJoinedActivity weitergeleitet. Ansonsten wird die Sitzung beendet und auf BeginActivity weitergeleitet. Grundsatz: kein aktuelles User-Objekt, keine Sitzung.
- Bei Klick auf Zurück speichert das ViewModel den Gruppenentwurf in MensaMeetSession und leitet auf SelectGroupActivity weiter.
- Bei Klick auf Home speichert das ViewModel den Gruppenentwurf in MensaMeetSession und leitet auf HomeActivity weiter.

3.6.10.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)
- Verhalten beim Scheitern der Server-Anfragen
- ViewModel-eigenes Attribut group entfernt
  Stattdessen wird direkt in das Attribut createdGroup von MensaMeetSession geschrieben, um unnötige Datenaktualisierungen zu vermeiden.
- Hinweis an den Benutzer, dass eine Gruppe gelöscht wird, wenn er sie als letztes Mitglied verlässt

3.6.11 ShowUserActivity und ShowUserViewModel

ShowUserActivity zeigt das Profil eines Benutzers an. ShowUserViewModel sorgt für die Kommunikation mit MensaMeetSession, ist aber leer, da alle benötigten Methoden von MensaMeetViewModel ererbt werden.

3.6.11.1 Sichtbare Elemente

- Benutzerbild
- Name (Textanzeige)
- Motto (Textanzeige)
- Geburtsdatum (Textanzeige)
- Geschlecht (Textanzeige)
- Status(Textanzeige)
- Fachrichtung(Textanzeige)
- Zurück (Button)
- Home (Button)

3.6.11.2 Verhalten

- Die Activity ist nur zugreifbar, wenn der Benutzer eingeloggt ist, ausreichende Benutzerdaten angegeben sind sowie userToShow in MensaMeetSession ungleich null ist.
- Bei (Wieder-)Eintritt in die Activity wird über das ViewModel aus MensaMeetSession der userToShow geladen.
- Die Activity bindet ein UserItem im DisplayMode BIG_NOTEDITABLE ein, das alle Elemente außer der Navigation enthält.
- Bei Klick auf Zurück wird auf die aufrufende Seite zurückgeleitet, die im Activity-Stack von Android gespeichert ist.
- Bei Klick auf Home wird auf HomeActivity weitergeleitet

3.6.11.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)

3.6.12 GroupJoinedActivity und GroupJoinedViewModel

Diese Activity zeigt die Gruppe an, in der der Benutzer Mitglied ist. Sie wird nach einem Gruppenbeitritt, nach einer Gruppenerstellung und dem darauf automatisch erfolgenden Beitritt sowie danach jedes Mal bei Klick auf Essen gehen in HomeActivity angezeigt. Der Benutzer kann hier auch aus der Gruppe austreten. GroupJoinedViewModel lädt die Gruppe vom Server und schickt an diesen die Austrittsanfrage.

3.6.12.1 Sichtbare Elemente

- Name (Textanzeige)
- Motto (Textanzeige)
- Zeit (Textanzeige)
- Linie (Textanzeige)
- Mitgliederzahl/Maximale Mitgliederzahl (Textanzeige)
- Gruppe löschen (Button), falls aktueller Benutzer Administrator
- Mitglieder (Liste), enthält pro Eintrag:
  - Benutzerbild
  - Name (Textanzeige)
  - Motto (Textanzeige)

  - Benutzer löschen (Button), falls aktueller Benutzer Administrator und nicht dem Benutzer identisch

- Gruppe verlassen (Button)
- Home (Button)

3.6.12.2 Verhalten

- Die Activity ist nur zugreifbar, wenn der Benutzer eingeloggt ist, ausreichende Benutzerdaten angegeben sind und das Grouptoken des Benutzers nicht null ist.
- Bei (Wieder-)Eintritt in die Activity wird vom ViewModel jedes Mal mit Hilfe des Grouptokens die Gruppe beim Server angefordert. Eine derart häufige Aktualisierung ist sinnvoll, da sich die Mitgliederanzahl einer Gruppe ständig ändern kann.
  viewmodel.GroupJoinedViewModelTest.groupNotFound(), viewmodel.GroupJoinedViewModelTest.loa
- Falls das Laden der Gruppe scheitert, wird der Fehlercode betrachtet. Zeigt er an, dass die Gruppe nicht (mehr) existiert (404), kann davon ausgegangen werden, dass die Gruppe gelöscht wurde. In diesem Fall wird das Grouptoken beim aktuellen Benutzer auf null gesetzt und versucht, sein User-Objekt am Server upzudaten.
  viewmodel.GroupJoinedViewModelTest.groupNotFound(), viewmodel.GroupJoinedViewModelTest.upd
- Zeigt der Fehlercode einen anderen Fehler an, wird der Fehler dem Benutzer gemeldet und der Benutzer auf HomeActivity weitergeleitet.
- Gelingt das Update des Users, wird dem Benutzer gemeldet, dass die Gruppe nicht mehr ladbar ist und er nicht mehr Mitglied darin ist. Er wird auf HomeActivity weitergeleitet und kann nun wieder nach neuen Gruppen suchen.
- Gelingt das Update des Users nicht, wird sein User-Objekt in MensaMeetSession auch nicht verändert. Er wird nach einer Fehlermeldung auf HomeActivity weitergeleitet und ist in der Lage vor dem Aufruf dieser Activity.
- Das Verhalten bei Klick auf Gruppe löschen und Benutzer löschen entspricht dem in SelectGroupActivity.
- Bei Klick auf Gruppe verlassen versucht das ViewModel zunächst, den Benutzer auf dem Server aus der Gruppe zu entfernen.
  viewmodel.GroupJoinedViewModelTest.leavingGroupFailed(), viewmodel.GroupJoinedViewModelTest.
- Gelingt das Entfernen des Benutzers, wird sein User-Objekt neu vom Server angefordert.
  viewmodel.GroupJoinedViewModelTest.reloadingUserFailed()
- Gelingt das Entfernen des Benutzers nicht, Fehlermeldung, Benutzer verleibt in der Activity.

- Kann das User-Objekt vom Server empfangen werden, wird es in MensaMeetSession aktualisiert und der Benutzer auf HomeActivity weitergeleitet. Ansonsten wird die Sitzung beendet und auf BeginActivity weitergeleitet. Grundsatz: kein aktuelles User-Objekt, keine Sitzung.

3.6.12.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Zugriffskontrolle (siehe MensaMeetActivity -> E&V -> checkAccess)
- Erneutes Laden der Gruppe bei jedem Eintritt in die Activity
  Zuvor wurde die Gruppe nach dem Beitritt nur noch lokal aus MensaMeetSession geladen. Aufgrund der Veränderlichkeit der Gruppenmitglieder sowie der Möglichkeit, dass die Gruppe zwischenzeitlich gelöscht wird, ist eine häufige Aktualisierung sinnvoll.
- Verhalten beim Scheitern der Server-Anfragen
- Methode setGroupByToken im ViewModel gibt nun ein Boolean zurück
  Mit false wird der aufrufenden Methode in der Activity signalisiert, dass ein Fehler aufgetreten ist. Diese Methode kann sich dann sofort beenden, was nötig ist, dass erst dann in der Activity der Beobachter der veränderten eventLiveData mit Fehlerstatus und Fehlerstring benachrichtigt wird und den Fehler auswerten kann.
- Beim Verlassen der Gruppe Bestätigungsdialog hinzugefügt

3.6.13 MensaMeetList, MensaMeetListHandler und MensaMeetAdapter

Diese Klassen und ihre Unterklassen für Line, Group und User sind für die Darstellung der Listen mittels RecyclerViews zuständig. Sie binden jeweils MensaMeetItem- bzw. deren Unterklassen-Objekte für die einzelnen Listeneinträge. Da die Funktionsweise sich am Standard für die Einbindung von RecyclerViews orientiert und es in diesem Bereich kaum Änderungen seit der letzten Phase gab, seien die Klassen hier nicht näher beschrieben.

3.6.13.1 Ergänzungen und Verbesserungen seit der letzten Phase

- Keine Auswahl bei LineList führte zum Absturz
  LineList ist in SelectOneLineDialog eingebunden, der in CreateGroupActivity bei der Auswahl einer Linie für die zu erstellende Gruppe aufgerufen wird. Wurde kein Objekt ausgewählt und der Dialog mit OK geschlossen, stürzte die App ab. Grund dafür war, dass in der Methode MensaMeetListAdapter beim hier vorliegenden displayMode SINGLE_SELECT der ausgewählte Listeneintrag mit seinem Listenindex in der Variable selectedId gespeichert wurde, deren Defaultwert -1 ist. In der Methode getSelectedObjects wurde mit selectedId direkt auf die Liste zugegriffen, ohne zu überprüfen, ob sie noch den Defaultwert enthält, was bei der Auswahl keiner Linie der Fall ist. Die Überprüfung wurde ergänzt.

3.6.14 MensaMeetItem und MensaMeetItemHandler

MensaMeetItem ist die abstrakte Oberklasse für die Item-Klassen für Line, Group und User. Items können sowohl Listenelemente sein als auch einzeln angezeigt werden, in groß und klein, bearbeitbar oder nicht bearbeitbar. Sie enthalten sichtbare Elemente, die über eine Id identifiziert werden, welche zugleich die Id der sie beschreibenden Stringressource ist. MensaMeetItemHandler und dessen Unterklassen enthalten analog zu den ViewModels die Logik. Die Kommunikation geschieht hier ebenso über ein SingleLiveData-Objekt, das beobachtet wird.

3.6.14.1 Elemente von MensaMeetItem

- Attribut handler
  Analog zum ViewModel bei Activity wird der Handler in einem Attribut gehalten. Auch die Initialisierung geschieht analog mit initializeHandler.

- Methode observeHandlerLiveData
  Analog zu Activity gibt es auch hier eine Methode zum Initalisierung der Beobachtung von eventLiveData im Handler. Sie wird in der App jedoch nicht gebraucht, da meistens die äußere Activity eventLiveData beobachtet, was auch möglich ist.

- Attribut view
  Es enthält die View, also die gesamte sichtbare Einheit des Items.

- Methode createView
  Sie erzeugt die View und speichert sie im Attribut ab. Sie muss von jeder Unterklasse implementiert werden, weshalb sie abstrakt ist. In diesen Implemetierung werden die anzuzeigenden Felder wie „Name“ oder „Motto“ nacheinander abgearbeitet, wobei jeweils falls nötig die verschiedenen DisplayModes unterschieden werden: Bei BIG_EDITABLE kann zum Beispiel ein Spinner angezeigt werden, während bei BIG_NOTEDITABLE ein einfaches Textfeld eingeblendet wird.

- Methode createTextField
  Um immer wieder gebrauchte sichtbare Elemente für die View in createView zu erzeugen, gibt es in MensaMeetItem viele Hilfsmethoden. Die am häufigsten benutzte ist createTextField, womit je nach displayMode (BIG_NOTEDITABLE oder BIG_EDITABLE) eine Textanzeige oder ein editierbares Textfeld erzeugt wird. Das Element, das wiederum eine View ist, erhält eine id, die zugleich die id der String-Ressource ist, welche sie beschreibt.

- Methode createLinkTextField
  Sie erzeugt eine Textanzeige, die dunkel hinterlegt ist und dadurch als Link gekennzeichnet ist. In createView einer Unterklasse wird ein ClickListener angehängt, der das Verhalten nach dem Link beschreibt.

- Methode createLabel
  Sie erzeugt eine beschreibende Textanzeige.

- Methode fillObjectData
  Sie sorgt dafür, dass die erzeugen Elemente der View mit Daten gefüllt werden, die aus dem im Attribut objectData des Handlers gespeichert werden. Sie wird bei allen Items gebraucht und ist daher abstrakt, damit sie implementiert werden muss.

- Methode fillTextField
  Sie soll in fillObjectData aufgerufen werden und befüllt ein Textfeld oder eine Textanzeige mit den zugehörigen Daten.

- Methode setSpinnerOrTextField
  Sie soll in fillObjectData aufgerufen werden und befüllt ein Element, das entweder ein Spinner, also eine Auswahlliste oder ein Textfeld bzw. eine Textanzeige ist. In der Auswahlliste wird das jeweilige Listenelement dann ausgewählt.

- Attribut representedValues
  Es gibt Daten, die eine Repräsentation besitzen, welche in einem sichtbaren Element angezeigt werden, und einen Wert besitzen, mit dem gearbeitet wird. In dieser HashMap werden, falls vorhanden, unter der Id eines sichtbaren Elements, das die Repräsentation enthält, der Wert des Datenelements gespeichert. Bei Spinnern werden diese Datenpaare mit Hilfe der Klasse SpinnerItem gespeichert, die diese wiederum aus IdEnums (siehe Model -> IdEnum) generieren können.

- Methode fillSublist
  Sie lädt eine verschachtelte Unterliste.

- Methode saveObjectData
  In dieser Methode soll im displayMode BIG_EDITABLE in der jeweiligen Unterklasse das Auslesen der sichtbaren Elemente implementiert werden, wozu es Hilfsmethoden wie extractTextField und extractSpinnerOrTextField gibt.

3.6.14.2 Elemente von MensaMeetItemHandler

- Attribut eventLiveData
  Analog zum ViewModel enthält auch der Handler ein beobachtbares SingleLiveEvent.

- Weiteren Methoden der Klasse
  Sie rufen jeweils entsprechende Methoden in MensaMeetSession auf. Grund dafür ist, dass nach dem MVVM-Modell die View nicht direkt mit dem Model kommunizieren, sondern ihre Daten vom ViewModel bzw. hier Handler beziehen sollte.

- Entfernung des Attributs objectData in MensaMeetItem
  Die Datenhaltung sollte ausschließlich im Handler geschehen, stattdessen wird daher nun immer handler.objectData benutzt.

3.6.14.3 Ergänzungen und Verbesserungen seit der letzten Phase

- Handhabung von Wertepaaren mit representedValues und SpinnerItem
  In der früheren Version gab es keine Handhabung von Wertepaaren. Bei Enums wie model.Status wurden die Werte für einen Spinner zum Beispiel immer „hin- und herübersetzt". Dies ist jedoch unsicher, da die Zuordnung Enum-Wert -> String (z. B. MALE -> „männlich") zwar eindeutig, der umgekehrte Fall es aber nicht sein muss. Nun wird nur noch mit den eindeutigen repräsentieren Werten gearbeitet.
- Einzelmethode observeLiveData (siehe MensaMeetActivity)
- MVVM-konformer Zugriff auf MensaMeetSession nur noch über Handler
- Attribut objectData statt in MensaMeetItem nun in MensaMeetHandler gespeichert, hierfür bei MensaMeetItemHandler Typ hinzugefügt
  Zuvor war das Datenobjekt inkonsequenterweise im Item gespeichert, nun wurde es MVVM-konform in den Handler verschoben.
- Attribut displayMode bei Handler entfernt *Ursprünglich wurde angenommen, dass der displayMode auch für die Logik im Handler entscheidend sein könnte. Dies hat sich jedoch nicht bestätigt, weshalb das unnötige Attribut entfernt wurde.*

3.6.15 GroupItem und GroupItemHandler

GroupItem enthält die Anzeige, GroupItemHander die Logik eines einzelnen Gruppenelements. Für Aufbau und Methoden siehe (siehe MensaMeetItem, SelectGroupActivity, CreateGroupActivity und GroupJoinedActivity).

3.6.15.1 Ergänzungen und Verbesserungen seit der letzten Phase

- Verarbeitung von Wertepaaren (siehe MensaMeetItem)
- Fehlerbehandlung der bei Scheitern der Serveranfragen im Handler (siehe Activities)
- Kein Beitreten-Button in GroupItem, falls Gruppe voll

3.6.16 UserItem und UserItemHandler

UserItem enthält die Anzeige, UserItemHander die Logik eines einzelnen Gruppenelements. Für Aufbau und Methoden siehe (siehe UserActivity, ShowUserActivity, SelectGroupActivity, und GroupJoinedActivity).

3.6.16.1 Ergänzungen und Verbesserungen seit der letzten Phase

- Verarbeitung von Wertepaaren (siehe MensaMeetItem)
- Fehlerbehandlung der bei Scheitern der Serveranfragen im Handler (siehe Activities)

3.7 Integration-Test

- BeginActivityTest: Testet auf der grafischen Oberfläche, ob man von der BeginActivity mit den entsprechenden Buttons auch das gewünschte Ziel erreicht
- HomeActivityTest: Testet auf der grafischen Oberfläche, ob man von der HomeActivity mit den entsprechenden Buttons auch das gewünschte Ziel erreicht
- LoginActivityTest: Testet auf der grafischen Oberfläche, ob der Login funktioniert, indem einmal getestet wird, ob es möglich ist, sich mit korrekten Daten anzumelden und ob man mit falschen Daten abgelehnt wird.
- RegisterActivityTest: Testet auf der grafischen Oberfläche, ob die Registrierung funktioniert, indem getestet wird, ob es möglich ist, sich mit gültigen Daten zu registrieren. Außerdem wird auch mit diversen ungültigen Daten getestet, wie einer falsch formatierten Email Adresse oder nicht übereinstimmender Passwörter, ob die Registrierung dann abgelehnt wird.
- EditProfileTest: Testet auf der grafischen Oberfläche, ob der Benutzer sein Profil bearbeiten kann.

3.8 Testszenarien

Die nachfolgenden Tests laufen auf dem Client und sind zur Überprüfung der korrekten Zusammenarbeit von mehreren Komponenten auf Server und Client.

- User Registrieren/Anmelden
- User Profil ändern und wieder lesen.
- Mensalinien auswählen, Zeit festlegen und Gruppen anzeigen lassen.
- Gruppe erstellen und anzeigen lassen. Gruppe verlassen.
- Nach Gruppen suchen und einer beitreten.
- Nach Gruppen suchen und User in der Gruppe anzeige lassen.
- Admin User: Gruppen suchen und eine gefundene Gruppe löschen.
- Admin User: Gruppen suchen und einen User in einer gefundenen Gruppe löschen.

4 Hallway Usability Testing

4.1 Einleitung

Hallway Usability Testing ist eine günstige und einfache Möglichkeit sein Produkt (in unserem Fall eine App) auf Benutzerfreundlichkeit zu testen. Dabei werden unbekannte Personen gebeten, das gegebene Produkt zu testen und zu bewerten.
Vorteil von diesem Vorgehen ist, dass man keine Speziallisten bezahlen muss, um Tests durchzuführen und dass man unbefangene Personen hat, die die App ohne Vorkenntnisse nutzen und daher neutral bewerten.

Nach Aussage von Jakob Nielsen, einem der führenden Persönlichkeiten auf dem Gebiet Benutzerfreundlichkeit und Gründer der Beratungsfirme Nielson Norman Group für Gebrauchstauglichkeit, reichen bereits 5 Probanden um 85 % der Usability Probleme zu finden [1].
Die Ergebnisse seiner Studie zur Notwendigen Anzahl an Testpersonen sind graphisch in der folgenden Abbildung (Abbildung 4.1) dargestellt.

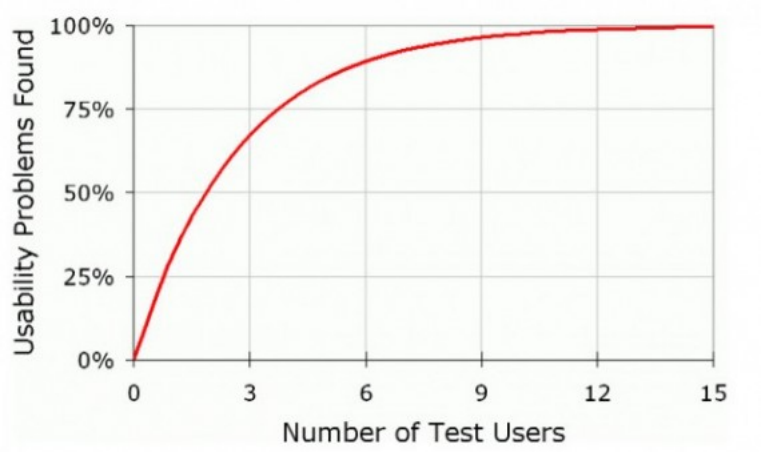


Abbildung 4.1: Stichprobengröße für Usability Tests

[1] Quelle: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

4.2 Vorbereitung

Für uns war es besonders wichtig zu sehen, ob Probanden die für sie unbekannte App, mühelos bedienen können und problemlos die vorgegebenen Ziele erreichen können. Generelles Feedback, z.b. der Wunsch nach weiteren Funktionalitäten, war im Hinblick auf eine eventuelle Veröffentlichung der Applikation ebenfalls interessant für uns.
Unter diesen Gesichtspunkten haben wir folgendne Fragebogen (Abbilding 5.2) zusammengestellt, den die Probanden, nach Durchführung von festgelegten Aufgaben(Listing 5.1.), ausfüllen durften.
Um sicherzustellen, dass unsere Aufgaben und Befragung nicht zu viel Zeit in Anspruch nehmen, haben wir es vorher selbst durchgespielt und uns dabei bewusst Zeit gelassen.

Listing 5.1. Aufgabenliste

- 1.) Bearbeite dein Profil: Ändere den Namen und das Profilbild
- 2.) Plane zwischen 12 und 14 Uhr an der Cafeteria Essen zu gehen
- 3.) Trete einer Gruppe bei
- 4.) Schau dir das Profil eines Gruppenmitgliedes an
- 5.) Verlasse die Gruppe
- 6.) Erstelle eine eigene Gruppe

Evaluation

Die App war intuitiv/leicht bedienbar.

☐ Trifft voll zu ☐ Trifft eher zu ☐ Neutral
☐ Trifft eher nicht zu ☐ Trifft gar nicht zu

Ich würde diese App selber benutzen/weiterempfehlen.

☐ Trifft voll zu ☐ Trifft eher zu ☐ Neutral
☐ Trifft eher nicht zu ☐ Trifft gar nicht zu

Das Design hat mir gefallen.

☐ Trifft voll zu ☐ Trifft eher zu ☐ Neutral
☐ Trifft eher nicht zu ☐ Trifft gar nicht zu

Wünscht du dir noch weitere Funktionen?

Was hat dir gefallen? Was hat dir nicht gefallen?

Abbildung 4.2: Evaluationsbogen

4.3 Durchführung

Um Probanden für unseren Test zu finden, haben wir Personen am KIT Campus, die im Forum saßen, folgendermaßen angesprochen:

Hallo, dürfen wir euch eine Frage stellen? Esst ihr regelmäßig in der Mensa?

Personen die diese Frage bejahten kamen als Probanden in Frage.

Würdet ihr an einer kleinen Umfrage teilnehmen, es dauert nur 5 Minuten und als Dankeschön bekommt ihr ein paar Schoko-Bons.

Bei Zusage wurde eine Person aus der Personengruppe ausgewählt.

Also, wir haben eine App entwickelt, mit der sich Leute anonym mit Fremden zum Essen gehen an der Mensa verabreden können und wir wollten uns etwas Feedback dazu einholen, deshalb darfst du die App jetzt kurz für uns testen. Du kriegst gleich ein Handy von uns, auf dem ist bereits ein User registriert und eingeloggt und dann musst du einfach diese sechs Aufgaben(zeigt Aufgabenliste) abarbeiten.

Während der Durchführung achteten wir auf die Zeit, schauten aufmerksam zu und notierten uns hinterher unsere Beobachtungen, die im Abschnitt Ergebnisse aufgeführt sind.
Nachdem die Aufgaben durchttps://www.google.com/maps/dir/Yorckstra
So, du hast jetzt während den Aufgaben alle Funktionalitäten der App kennen gelernt, jetzt musst du nur noch diesen Umfragebogen ausfüllen und dann bist du fertig.

4.4 Ergebnisse

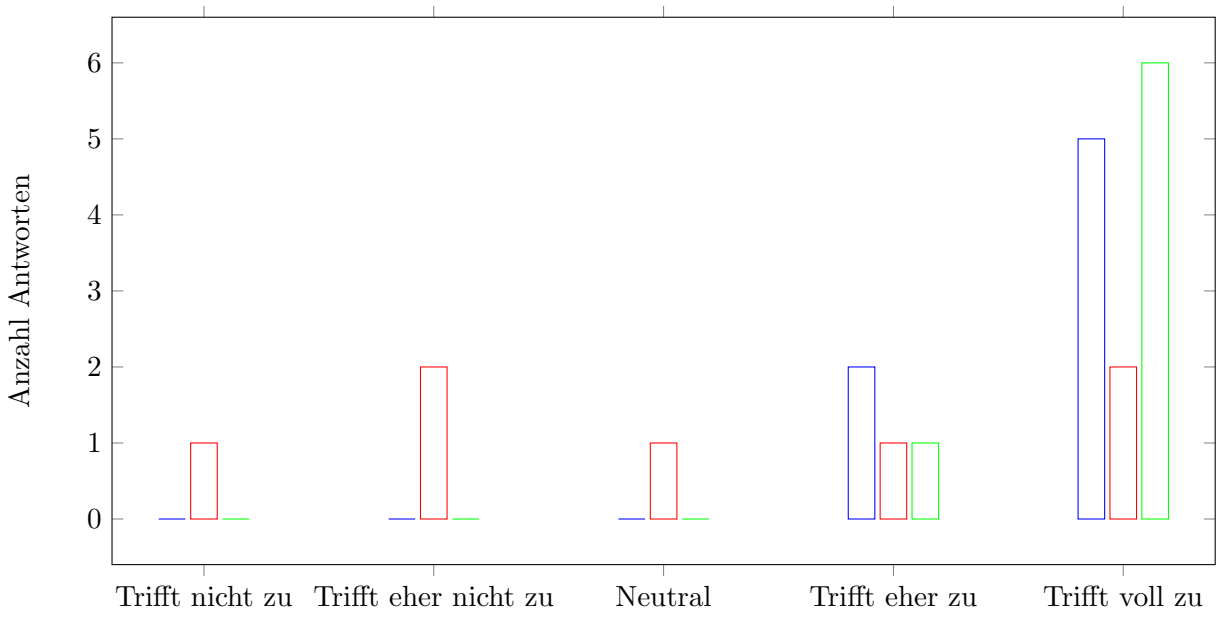
Unser Test umfasste sieben Probanden, die alle auf oben beschriebene Weise am Campus angesprochen wurden.

Beobachtungen

Die Probanden benötigten jeweils 5-7 Minuten für den gesamten Test, ab dem Zeitpunkt, ab dem sie das Handy und die Aufgabenliste bekamen. Hauptsächlich nahmen sie sich unterschiedlich viel Zeit für den Fragebogen und benötigten etwa die selbe Menge Zeit zum durchführen der Aufgaben.

Zwei Probanden speicherten ihre Profiländerung nicht über den Speichern-Button sondern drückten stattdessen auf Zurück, um wieder in das Home Menü zu gelangen.
Sechs von sieben Probanden bemerkten nicht, dass nicht nur drei, sondern sechs, Profilbilder zur Wahl standen.
Beim Erstellen einer Gruppe konnte die maximale Mitgliederzahl auf 1 eingestellt werden.

Auswertung der Evaluationsbögen



Legende:
Blau: Die App war intuitiv/leicht bedienbar.
Rot: Ich würde diese App selber benutzen/weiterempfehlen.
Grün: Das Design hat mir gefallen.

Antworten in den Freitextfeldern

Wünscht du dir noch weitere Funktionen?

Vier von sieben Probanden füllten dieses Feld aus:

- Hashtags für die Gruppen zum Suchen
- Bewertungssystem für das Essen
- Aktualisieren des Essenplans, wenn etwas ausgegangen ist
- Bewerten von Essen und Personen
- Chat innerhalb der Gruppe

Was hat dir gefallen, was hat dir nicht gefallen?

Zwei von sieben Probanden füllten dieses Feld aus:

- Einfache Bedienbarkeit, Gute Funktionalität und vorrausschauendes Texting
- Hübsches Design
- Maximale Mitgliederzahl kann 1 sein. Macht keinen Sinn

Fazit

Mit der Bedienbarkeit der App hatten die Probanden keine Schwierigkeiten und bewerteten diese im Allgemeinen sehr positiv. Auch das Design wurde von den meisten sehr ansprechend empfunden. Dass manche Probanden ihre Profiländerung nicht korrekt abspeicherten führen wir primär darauf zurück, dass sie sich beim Durchführen der Aufgabe nicht genug Zeit nahmen und daher den Speichern-Button übersahen. Dennoch ließe sich die Benutzerfreundlichkeit an der Stelle weiter verbessern, indem der Button stärker hervorgehoben würde, z.b. durch farbliche Kennzeichnung.

Zudem fehlt offenbar ein Hinweis darauf, dass man duch die Profilbild-Auswahl scrollen kann. Dies ließe sich beispielsweise durch kleine Pfeile an den Seiten oder durch unscharfe Ränder andeuten.

Die Möglichkeit die maximale Mitgliederzahl einer Gruppe aufs Eins zu setzen war von Anfang an nicht vorgesehen, da man zum alleine Essen gehen keine Gruppe braucht. Dieser Fehler wurde behoben, sodass die maximale Mitgliederzahl einer Gruppe mindestens Zwei beträgt.

Viele der gewünschten Funktionalitäten hatten wir bereits als Wunschkriterium im Auge, dennoch haben wir weitere Anregungen erhalten und nun eine bessere Vorstellung davon, welche Funktionen unseren potnenziellen zukünftigen App-Nutzern wichtig wären.