

# MensaMeet Client Testing

# Tests

- **Unit-Tests**

- RequestUtil-Klasse
- Activities
- ViewModels

- **Integration-Tests**

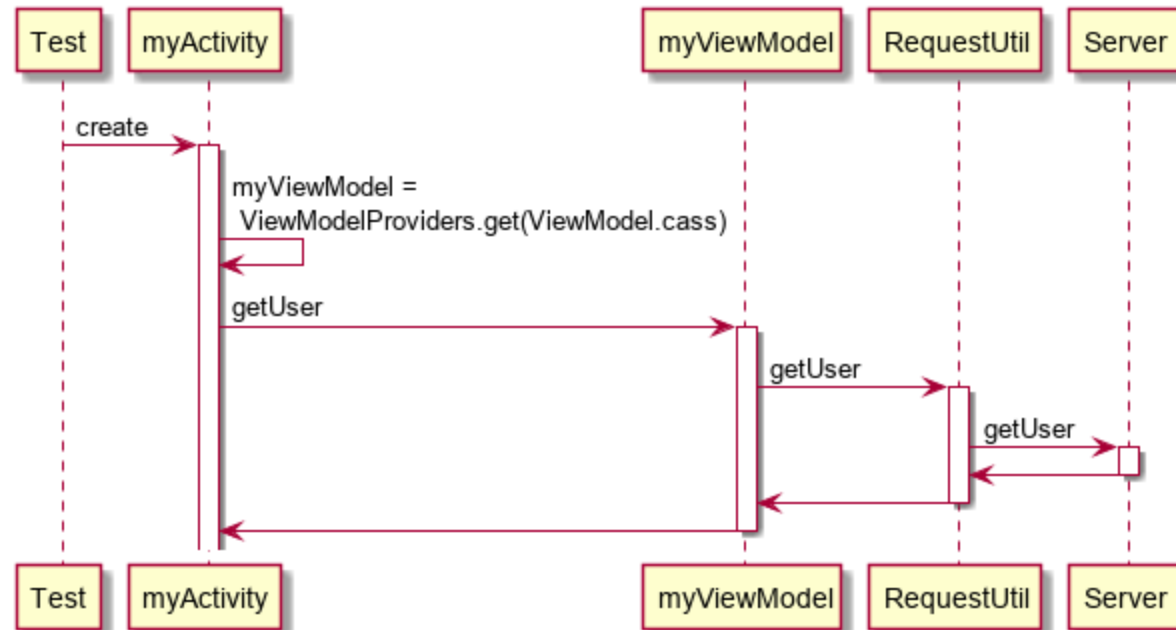
- Espresso

# MensaMeet Client Testing: Mocking

# Mockito: Mock vs. Spy

- Mock: *mock(ViewModel.class)*  
*Klasse nur ein Stub, alle Methoden müssten gemockt werden.*
- Spy: *spy(ViewModel.class)*  
*Klasse normal, nur ausgewählte Methoden werden gemockt.*

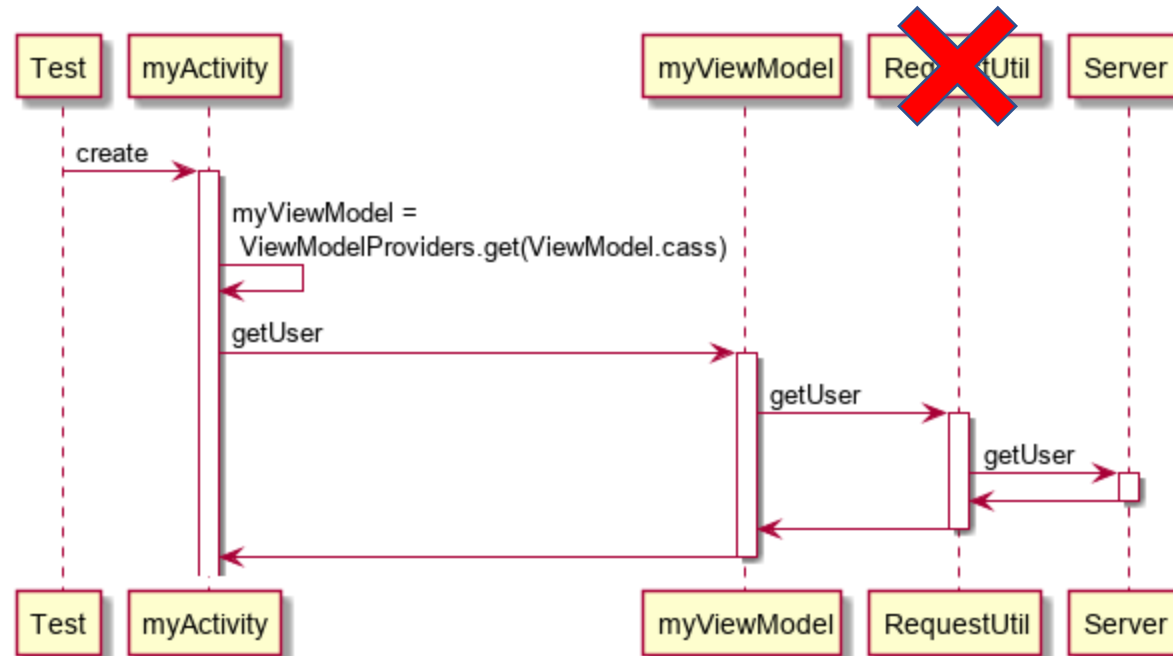
# Ausgangslage



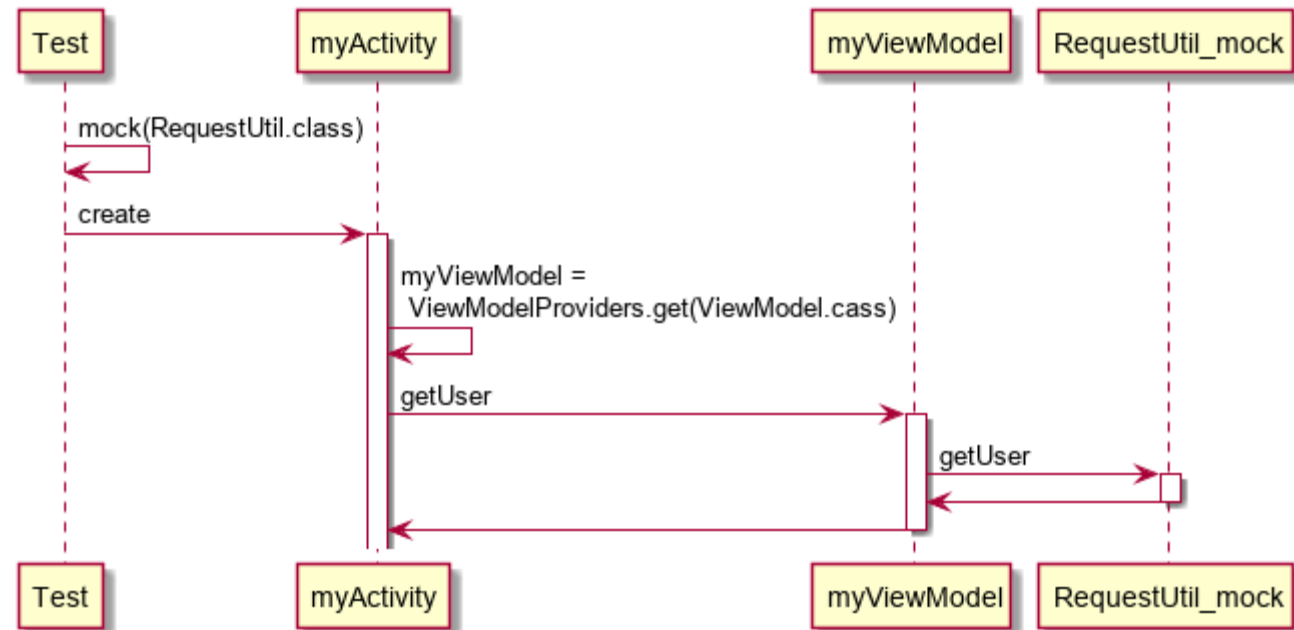
# Lösung 1: RequestUtil-Mock

- Statische Klasse, global mockbar, keine Injection nötig
- Powermockito nötig

# RequestUtil-Mock



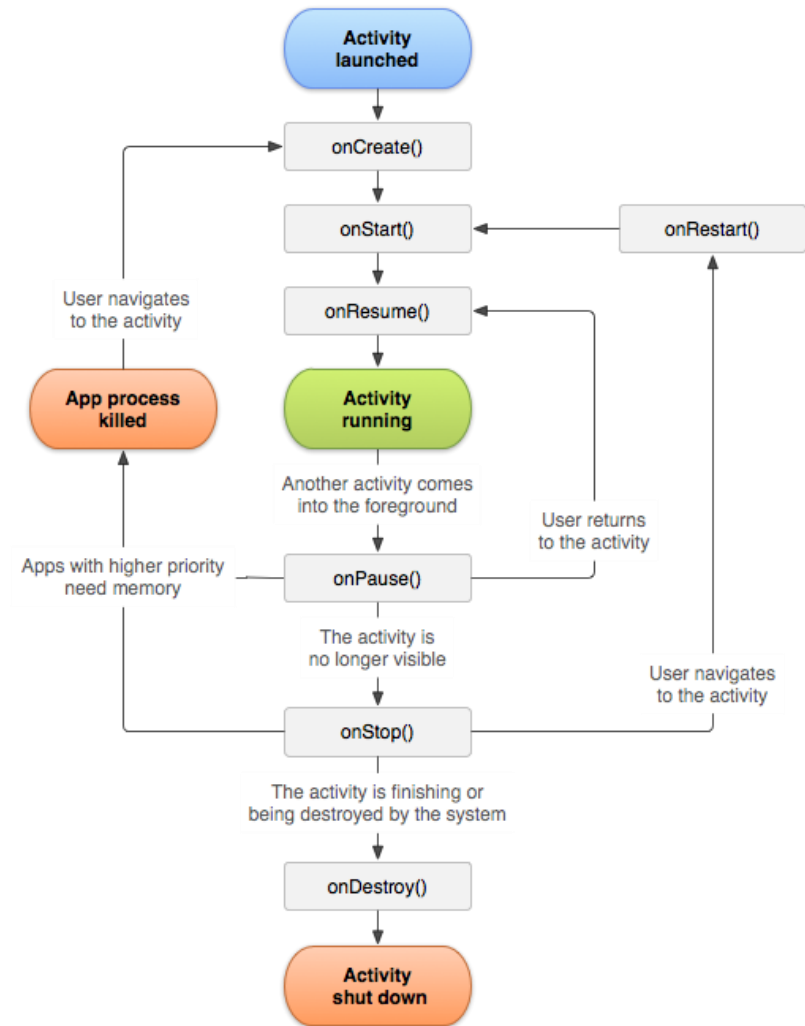
# RequestUtil-Mock



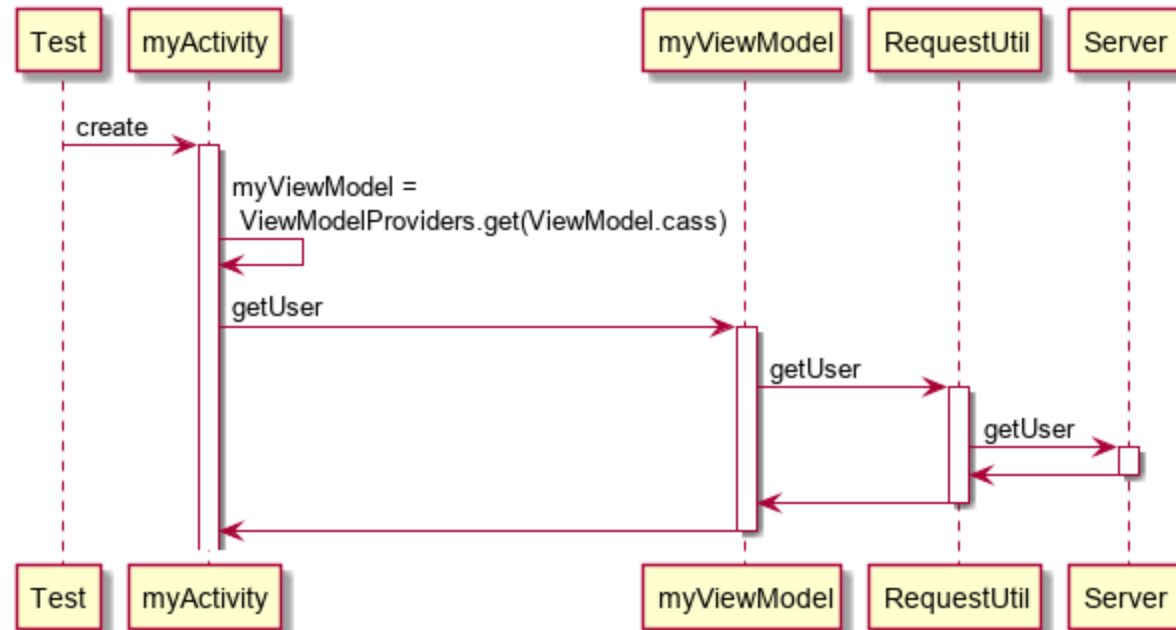


# Lösung 2: ViewModel-Injection

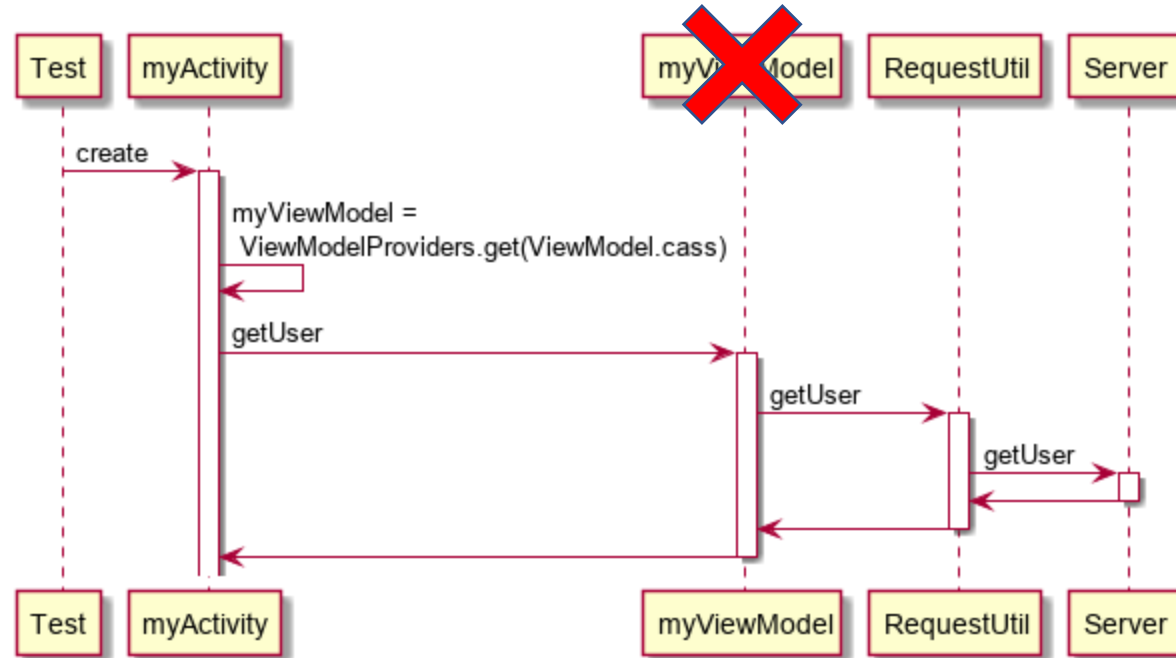
- Auftrennung des Inhalts von create() in create() und start()
- @VisibleForTesting setViewModel einbauen
- Zwischen create() und start() setViewModel(spyViewModel) aufrufen



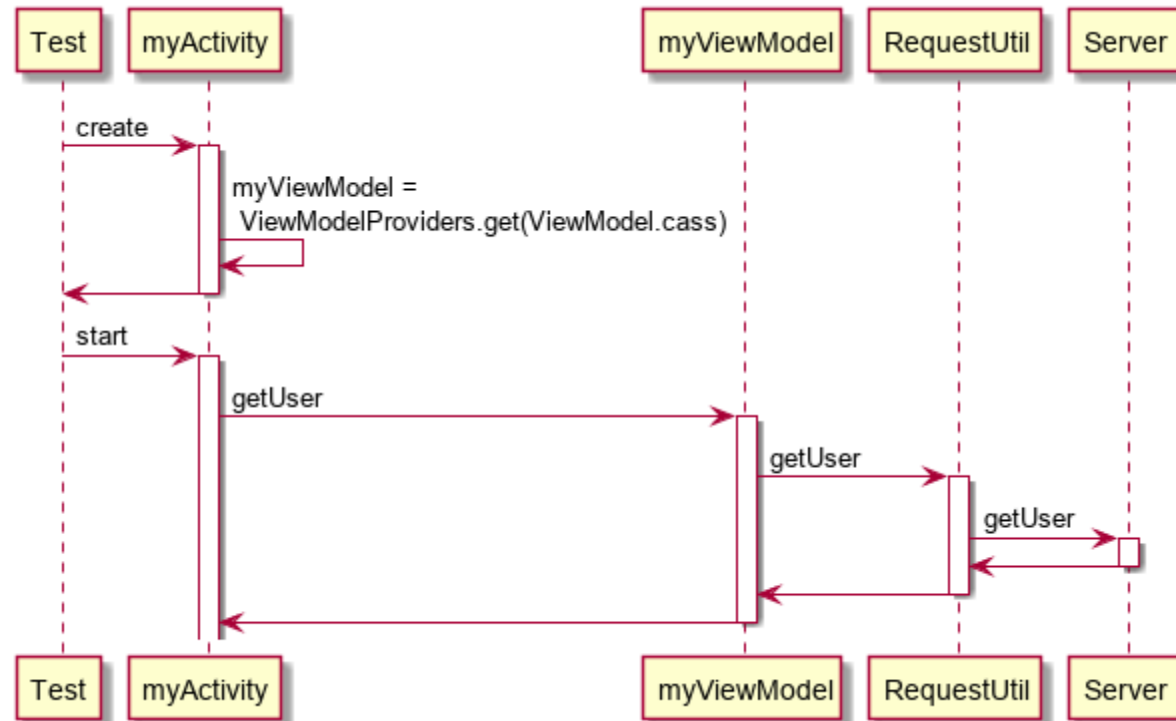
# ViewModel Injection



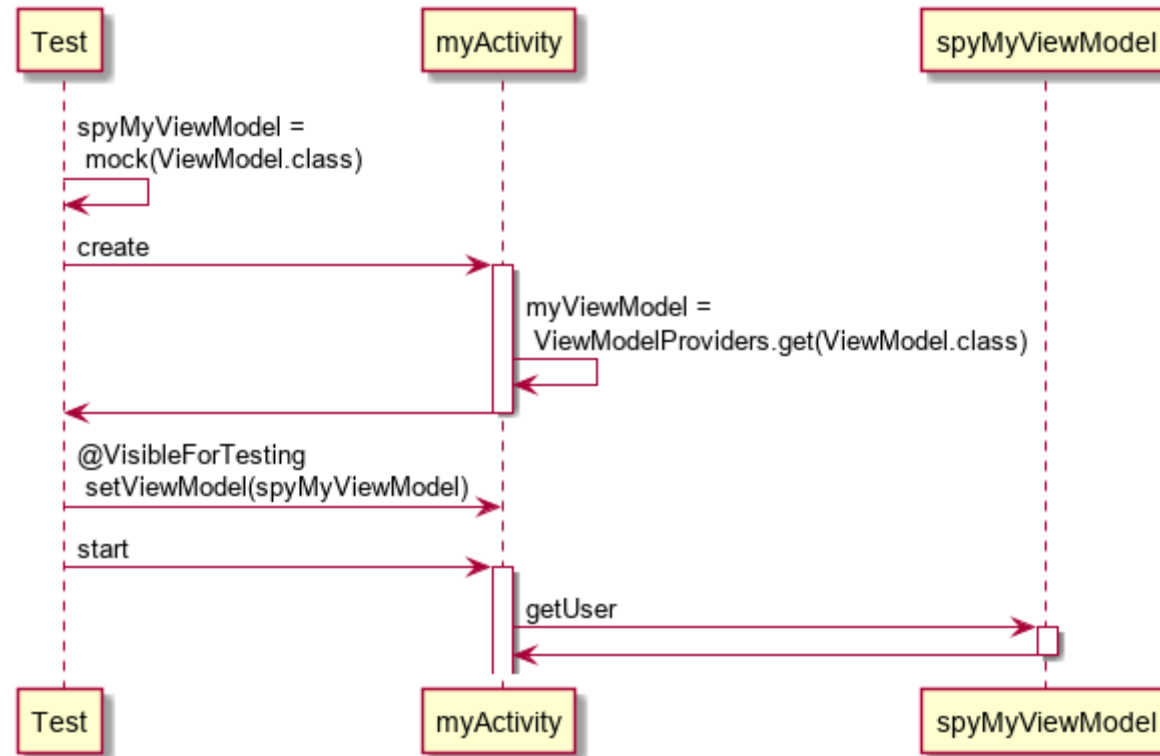
# ViewModel Injection



# ViewModel Injection



# ViewModel Injection



MensaMeet Client Testing:  
„Best-Of-“Verbesserungen

# „Best-Of“-Verbesserungen







































- Fehler-JSON bei HTTP-Fehlern auslesen und mit `RequestException` zum `ViewModel` weiterleiten
- Weiterleitung von Firebase-Fehlern
- `checkAccess` bei jeder Activity
- Methoden werden nicht automatisch nach `LiveData`-Aktualisierung beendet → Fix
- Umfassende Behandlung von Fehlerszenarien: z. B. kein aktuelles User-Objekt → keine Sitzung
- User muss nicht mehr sofort sein Profil vervollständigen
- Durch Unit-Tests gefunden: Verhalten beim Klick auf Navigationsbuttons war manchmal fehlerhaft
- Häufigkeit von Serveranfragen überdacht: Mensaplan nur am Anfang, Gruppen- und Userprofil bei jedem Activity-Aufruf
- Aufgabentrennung Activity-ViewModel konsequenter: Daten nur noch über VM
- Handhabung von Wert-Repräsentations-Paaren

# MensaMeet Client Testing: Integration-Tests mit Espresso

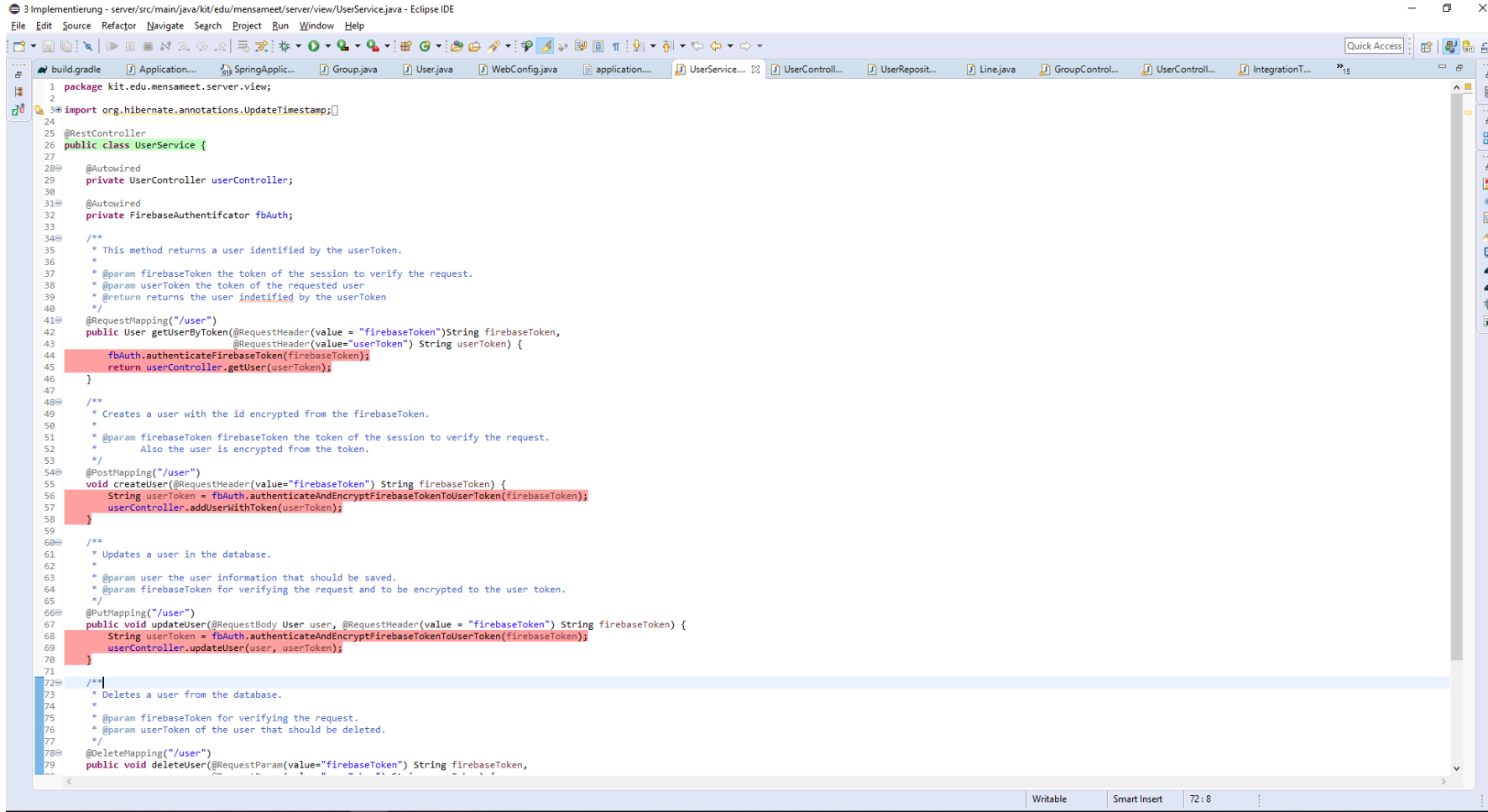


# MensaMeet Server Testing

# Code Coverage

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
▼  server	 57,9 %	2.172	1.581	3.753
▼  src/main/java	 46,8 %	1.345	1.527	2.872
>  kit.edu.mensameet.server.model	 50,9 %	941	909	1.850
▼  kit.edu.mensameet.server.controller	 48,0 %	372	403	775
>  MensaDataController.java	 2,0 %	6	300	306
>  FirebaseAuthificator.java	 7,3 %	3	38	41
>  GroupController.java	 86,8 %	151	23	174
>  UserController.java	 87,1 %	128	19	147
>  MembershipController.java	 84,4 %	81	15	96
>  TimeController.java	 27,3 %	3	8	11
▼  kit.edu.mensameet.server.view	 6,9 %	12	163	175
>  UserService.java	 4,8 %	3	60	63
>  GroupService.java	 5,0 %	3	57	60
>  MembershipService.java	 6,7 %	3	42	45
>  MensalineService.java	 42,9 %	3	4	7
▼  kit.edu.mensameet.server	 27,8 %	20	52	72
>  Application.java	 18,8 %	12	52	64
>  WebConfig.java	 100,0 %	8	0	8
>  src/test/java	 93,9 %	827	54	881

# Code Coverage



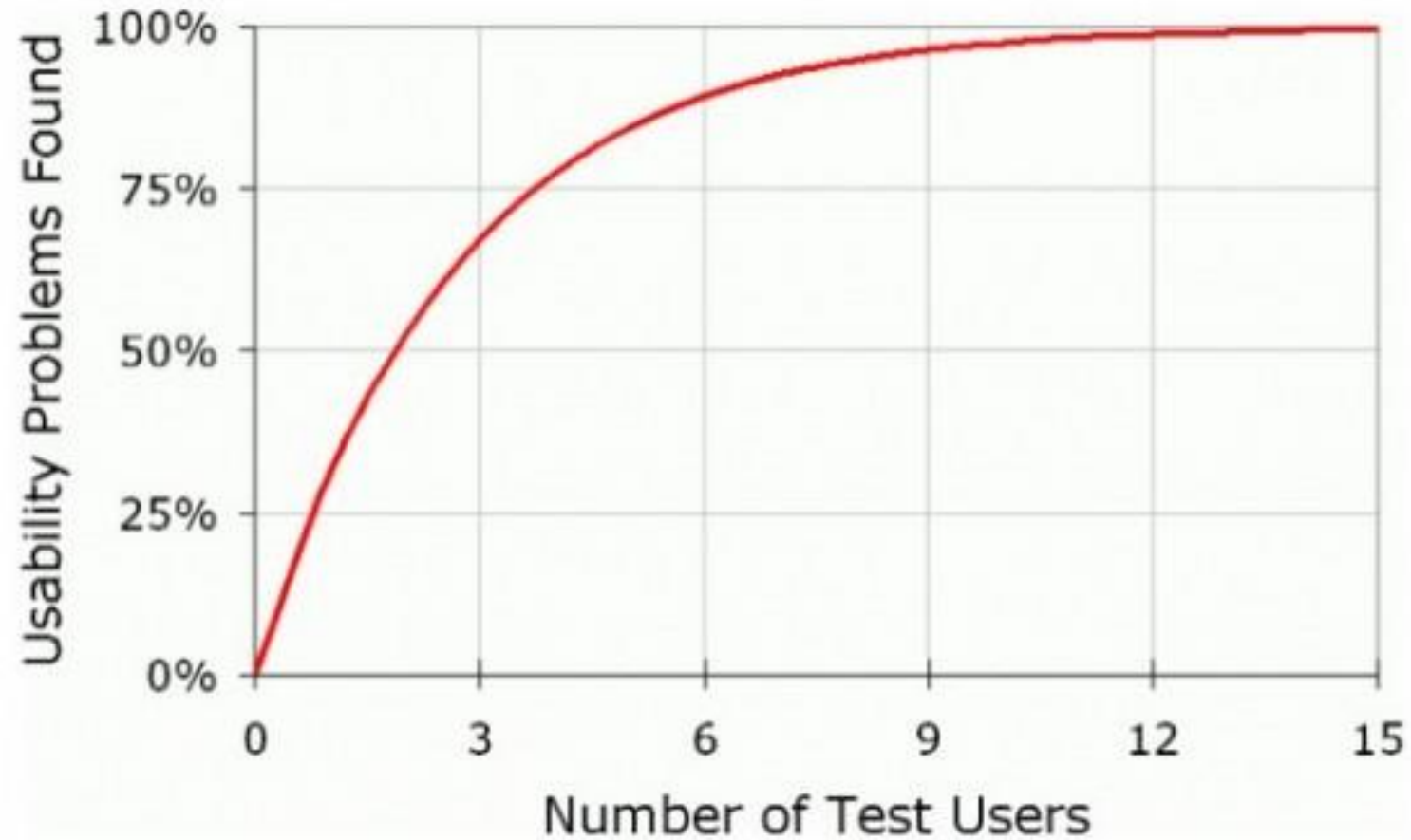
3 Implementierung - server/src/main/java/kit/edu/mensameet/server/view/UserService.java - Eclipse IDE

```
1 package kit.edu.mensameet.server.view;
2
3 import org.hibernate.annotations.UpdateTimestamp;
4
5 @RestController
6 public class UserService {
7
8     @Autowired
9     private UserController userController;
10
11     @Autowired
12     private FirebaseAuth fbAuth;
13
14     /**
15      * This method returns a user identified by the userToken.
16      *
17      * @param firebaseToken the token of the session to verify the request.
18      * @param userToken the token of the requested user
19      * @return returns the user identified by the userToken
20      */
21     @RequestMapping("/user")
22     public User getUserByToken(@RequestHeader(value = "firebaseToken") String firebaseToken,
23                               @RequestHeader(value = "userToken") String userToken) {
24         fbAuth.authenticateFirebaseToken(firebaseToken);
25         return userController.getUser(userToken);
26     }
27
28     /**
29      * Creates a user with the id encrypted from the firebaseToken.
30      *
31      * @param firebaseToken firebaseToken the token of the session to verify the request.
32      * Also the user is encrypted from the token.
33      */
34     @PostMapping("/user")
35     void createUser(@RequestHeader(value = "firebaseToken") String firebaseToken) {
36         String userToken = fbAuth.authenticateAndEncryptFirebaseTokenToUserToken(firebaseToken);
37         userController.addUserWithToken(userToken);
38     }
39
40     /**
41      * Updates a user in the database.
42      *
43      * @param user the user information that should be saved.
44      * @param firebaseToken for verifying the request and to be encrypted to the user token.
45      */
46     @PutMapping("/user")
47     public void updateUser(@RequestBody User user, @RequestHeader(value = "firebaseToken") String firebaseToken) {
48         String userToken = fbAuth.authenticateAndEncryptFirebaseTokenToUserToken(firebaseToken);
49         userController.updateUser(user, userToken);
50     }
51
52     /**
53      * Deletes a user from the database.
54      *
55      * @param firebaseToken for verifying the request.
56      * @param userToken of the user that should be deleted.
57      */
58     @DeleteMapping("/user")
59     public void deleteUser(@RequestParam(value = "firebaseToken") String firebaseToken,
```

# Bugfixes

- Scheduler
- Grouptoken

# Hallway-Tests



# Ausblick

## 2.2 Wunschkriterien

Diese Kriterien sind nach Prioritäten geordnet, beginnend mit der höchsten Priorität.

/WK10/ Bitmoji erstellen, welches als Profilbild angezeigt wird

/WK20/ Pushbenachrichtigungen (z.B. Erinnerung kurz vor dem Treffen)

/WK30/ Ranking System um die Zuverlässigkeit einer Person einzuschätzen.

Hierzu können Gruppenmitglieder angeben, welche Personen zur Verabredung erschienen bzw. nicht erschienen sind

/WK40/ Achievement-System: Erhaltene Achievements werden im Profil angezeigt

/WK50/ Filtermöglichkeiten der angezeigten Gruppen z.B. nach Treffzeitpunkt, aktueller oder maximaler Mitgliederanzahl

/WK60/ Die Möglichkeit, Benutzer oder Gruppen zu melden