

# Program for Splitting Expenses Qualitätssicherungsbericht

Praxis der Softwareentwicklung  
Wintersemester 2024/25

PSE-Team

24. April 2025

Karlsruher Institut für Technologie  
Fakultät für Informatik  
Postfach 6980  
76128 Karlsruhe

# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>5</b>
<b>2. Bugs und Verbesserungen</b>	<b>6</b>
<b>3. Code Qualität</b>	<b>18</b>
3.1. Statische Analyse . . . . .	18
3.2. Verbesserungen . . . . .	19
3.2.1. Entfernen von Duplikaten im View Preview Code . . . . .	19
3.2.2. Korrekte Instanziierung von BigDec . . . . .	19
<b>4. Unit Testing und Abdeckungen</b>	<b>20</b>
4.1. Client . . . . .	20
4.2. Server . . . . .	20
<b>5. Integrationstests</b>	<b>22</b>
<b>6. Systemtests</b>	<b>23</b>
6.1. Server Verfügbarkeit . . . . .	23
<b>7. Usability Testing</b>	<b>24</b>
7.1. Auswertung . . . . .	24
7.2. Fazit . . . . .	28
7.3. Umgesetzte Verbesserungen . . . . .	28
<b>8. Abnahmetests</b>	<b>30</b>
<b>A. Anhang</b>	<b>31</b>
A.1. detekt-Konfiguration auf dem Server . . . . .	31
A.2. detekt-Konfiguration auf dem Client . . . . .	33
A.3. Hallway Usability Testing Fragebogen . . . . .	34
A.4. Client Coverage . . . . .	43



# 1. Einleitung

Anschließend an den Implementierungsbericht werden im Folgenden alle in der Qualitätssicherungsphase des *Program for Splitting Expenses*, kurz *PSE*, gefundenen Bugs, ihr Auftretensgrund sowie ihre Behebung beschrieben. Zudem wird die Usability der App untersucht, die statische Analyse ausgewertet und die Code Coverage analysiert.

## 2. Bugs und Verbesserungen

In diesem Kapitel werden alle gefundenen Bugs, der Grund ihres Auftretens (Reason) sowie die von uns gewählte Behebung (Fix) beschrieben. Außerdem werden alle Verbesserungen (Enhancement) ausgeführt, die uns durch eigenes Testing oder Hallway Usability Testing (siehe Abschnitt 7) aufgefallen sind und wir umgesetzt haben.

### **Server hält mit Exit Code ungleich Null bei `systemctl stop`**

**Bug** Siehe oben.

**Reason** Der Server hat die Signale SIGINT und SIGTERM, die bei `systemctl stop` gesendet werden, nicht explizit behandelt und hat dadurch implizit mit einem Exit Code ungleich Null beendet.

**Fix** Die Signale SIGINT und SIGTERM wurden explizit behandelt.

### **Client wird bei Serverneustart abgemeldet**

**Bug** Siehe oben.

**Reason** Da sich die Signierschlüssel der Access Tokens bei Serverneustart ändern, ist das Access Tokens eines jeden Clients dann ungültig. Die Refresh Tokens sind allerdings weiterhin gültig, sie wurden aber nur bei einem abgelaufenen Access Token als Fallback verwendet.

**Fix** Der Client verwendet zuerst den Access Token und anschließend den Refresh Token zur Authentifizierung und meldet sich erst ab, wenn beide Authentifizierungsversuche fehlgeschlagen sind.

---

## Keine Währungszeichen in der Gruppenübersicht

**Bug** Siehe oben.

**Reason** Es wird die falsche Komponente zum Anzeigen von Geldbeträgen verwendet.

**Fix** Es wird nun die richtige Komponente zum Anzeigen von Geldbeträgen verwendet.

## Exception bei Abgleich

**Bug** Nach dem Ausführen eines Abgleichs navigiert der Client ins Main Menu, aber die durch den Abgleich entstandenen Transaktionen erreichen den Server nicht und es steht CancellationException in den Logs.

**Reason** Da die Navigation ins Main Menu und das Senden der Anfrage an den Server zur Erstellung der Transaktionen gleichzeitig gestartet werden und der Server Request abgebrochen wird, sobald die Navigation beendet wurde, stellt dies eine Race Condition dar.

**Fix** Der Client navigiert erst, nachdem der Server Request durchgelaufen ist.

## Saldoänderungen in Zahlungen sind unintuitiv

**Bug** Wenn man eine neue Zahlung verbuchen möchte und dafür bei einer Person \$ 10 einträgt, so ist unklar, was die resultierende Buchung repräsentiert.

**Reason** Geplant war, stets Saldoänderungen einzutragen, aber diese sind für Außenstehende nicht sofort verständlich, wie auch das Usability Testing (siehe Abschnitt 7) ergeben hat.

**Fix** Es wird nun ein bezahlendes Mitglied ausgewählt und eingetragen, wie viel dieses Mitglied jedem anderen Mitglied zurückgezahlt bzw. vorgestreckt hat. Die eingetragenen absoluten Beträge wirken sich also jeweils negativ auf den Saldo der Mitglieder bezüglich der Gruppe aus.

## Gruppenreihenfolge in der Gruppenübersicht ist nicht klar definiert

**Bug** Siehe oben.

**Reason** Es gibt keine klar definierte Reihenfolge, in der der Server Gruppeninformationen auf Anfrage zurückgibt. Der Client hat eine feste Reihenfolge angenommen, dies ist aber nicht unbedingt gegeben.

**Fix** Der Server liefert für jede Gruppe nun den Zeitpunkt der letzten Transaktion. Der Client sortiert die Gruppen nach diesen Zeitpunkten.

## Ausgabe mit nur einem Nutzer mit von Null verschiedener Saldoänderung

**Bug** Das UI ermöglicht es, eine Ausgabe anzulegen, bei der nur ein Nutzer eine von Null verschiedene Saldoänderung hat. Das Modell erlaubte es, diese Ausgabe anzulegen, während der Server dies verweigerte. Dies führte zu einem Crash des Clients.

**Reason** Der Modell Code um die Validität einer Ausgabe zu überprüfen wies einen Fehler bei der Berechnung der Summe der Saldoänderungen auf, sodass sein Verhalten von dem des Servers abwich.

**Fix** Der Modell Code wurde an das Verhalten des Servers angepasst und es ist nun nicht mehr möglich, eine Ausgabe mit nur einem Nutzer mit von Null verschiedener Saldoänderung zu erstellen.

## Top Bar Padding in Main Menu scrollt nicht mit

**Bug** Beim Scrollen im Main Menu liegt zwischen der Top Bar und der Liste an Gruppen etwas Padding. Dieses Padding scrollt nicht mit, wodurch sich eine Lücke ergibt und die Liste an Gruppen teilweise abgeschnitten wird.

**Reason** Das Padding zwischen Top Bar und der Liste an Gruppen war nicht Teil der LazyColumn und scrollte somit nicht mit.

**Fix** Das Padding zwischen Top Bar und der Liste an Gruppen ist nun ein Teil der LazyColumn.

---

## Farben sind schlecht zu lesen im Nachtmodus

**Enhancement** Siehe oben. Besonders das dunkle Blau ist auf dunklem Hintergrund schwer zu erkennen gewesen.

**Fix** Es wurden geeigneter Farben gewählt.

## Fehlerhaftigkeit und Verbesserungswürdigkeit von angezeigten Texten

**Enhancement** Einige angezeigte Texte wie z.B. „You are not a member *in* any group“ weisen grammatischen Fehler auf oder sind verbessерungswürdig wie z.B. „Balance must be zero to kick“ oder „Really want to log out?“.

**Fix** Die betroffenen Texte wurden korrigiert oder durch von uns für geeigneter empfundene Texte ersetzt.

## Visuelles Feedback von Textfeldern bei invaliden Eingaben

**Bug** Manche Textfelder weisen kein visuelles Feedback bei invaliden Eingaben auf. Zahlenfelder weisen keinen unterstützenden Fehlertext auf.

**Reason** Es wurde vergessen, manchen Textfeldern auch visuelles Feedback zu geben sowie Zahlenfeldern unterstützenden Fehlertext zuzuweisen.

**Fix** Die betroffenen Textfelder wurden mit visuellem Feedback versehen und Zahlenfeldern wurde analog zu Textfeldern ein unterstützender Fehlertext zugewiesen.

## Zahlenfelder weisen leichte vertikale Verschiebung auf

**Bug** Siehe oben.

**Reason** Regression die durch den zu Zahlenfeldern hinzugefügten unterstützenden Fehlertext auftrat.

**Fix** Anstatt eines leeren Lambdas wurde null verwendet, wenn kein Fehlertext anzeigen ist.

## Transaktions- und Gruppenübersicht sind leer

**Enhancement** Anstatt einfach eine leere Transaktions- und Gruppenübersicht anzuzeigen, soll stattdessen Text angezeigt werden, der den Nutzer darüber informiert, dass es hier nichts zu sehen gibt.

**Fix** Transaktions- und Gruppenübersicht zeigen nun erklärenden Text, falls es nichts anzuzeigen gibt.

## ReceiveChannel gibt Implementationsdetails preis und sollte ein Flow sein

**Bug** ReceiveChannel kann von außen zum Schließen des Channels verwendet werden, indem z.B. consumeAsFlow anstelle von receiveAsFlow verwendet wird.

**Reason** ReceiveChannel garantiert keinen ausreichenden Schutz und sollte stattdessen ein Flow sein.

**Fix** Anstatt von ReceiveChannel wird nun ein Flow nach außen bereitgestellt. Code, der den ReceiveChannel verwendet hat, wurde an Flow angepasst.

## Auswahl an Nutzern, unter denen eine Ausgabe aufgeteilt werden soll, wird teils vergessen

**Bug** Bei der Erstellung einer Ausgabe wurde die Abwahl der Nutzer, die nicht an der Ausgabe teilnehmen sollen, in manchen Situationen vergessen, z.B. wenn sich die Konfiguration ändert.

**Reason** Es wurde gespeichert, welche Nutzer an der Ausgabe teilnehmen sollen. In onEntry wurden hierbei initial alle Mitglieder der Gruppe ausgewählt. Bei einem entsprechenden Checkbox click werden die teilnehmenden Nutzer geupdated. Wir haben aber keine Kontrolle darüber haben, wann onEntry ausgeführt wird. Beispielsweise wird es auch bei Konfigurationsänderungen ausgeführt, was zu obigem Bug führt.

**Fix** Es werden nun die Nutzer gespeichert, die *nicht* an der Ausgabe beteiligt sind anstelle der Nutzer, die beteiligt sind. Diese werden nur geupdated, wenn die entsprechende Checkbox geklickt wurde. Dies findet alles unabhängig von onEntry statt, wodurch die Auswahl nun nicht mehr vergessen wird.

---

## **Lange Nutzereingaben sorgen für kaputtes Layout**

**Bug** Nutzereingaben wie lange Anzeigennamen, Transaktionsnamen oder –kommentaren sowie Geldbeträge führen oft zu kaputtem Layout.

**Reason** Da Nutzereingaben nicht in ihrer Länge begrenzt sind und meistens in voller Länge angezeigt werden, kann dies zu kaputtem Layout führen.

**Fix** Anstatt lange Nutzereingaben vollständig anzuzeigen werden diese nun durch Ellipsenkürzung in ihrer Länge eingeschränkt. Die vollen Eingaben sind dann jeweils als Tooltips sichtbar.

## **Neu erstellte sowie gerade beigetretene Gruppen werden nicht sofort angezeigt**

**Bug** Wenn man eine Gruppe erstellt hat oder einer Gruppe beigetreten ist und anschließend ins Main Menu navigiert, so erscheint die neue Gruppe erst nach einem Refresh (also einer Anfrage an den Server), obwohl der Client bereits die Gruppe anzeigen können sollte.

**Reason** Ein interner Bug in LocalGroupRepo hat dafür gesorgt, dass die Liste an Gruppen in denen der aktive Nutzer Mitglied ist, nicht sofort geupdated wird.

**Fix** Es wurde setGroupData in LocalGroupRepo so angepasst, dass die Existenz der neuen Gruppe korrekt vermerkt wird.

## **App zeigt fälschlicherweise an, dass der Nutzer in keiner Gruppe ist**

**Bug** Nach Neustart der App wird vor Abschluss des Refreshs angezeigt, dass der Nutzer in keiner Gruppe ist, obwohl die App zu diesem Zeitpunkt keine Aussage darüber treffen sollte.

**Reason** Grund für diesen Bug ist, dass in der Schnittstelle des Models nicht zwischen einer Abwesenheit von gültiger Gruppeninformation, wie sie bei einem Neustart auftritt und einer Abwesenheit von Gruppenmitgliedschaften unterschieden wird.

**Fix** Statt `List<Group>` meldet das Model nun `List<Group>?`. Das Model sendet also nach Neustart nun `null` statt einer leeren Liste und es wird ein Spinner angezeigt, der das Laden der Gruppeninformationen verdeutlicht.

## Submit Buttons, die eine Anfrage an den Server schicken, zeigen kein visuelles Feedback und sind mehrmals betätigbar

**Bug** Die folgende nicht erschöpfende Liste an Submit Buttons zeigen kein visuelles Feedback und sind mehrmals betätigbar, wodurch mehrere Requests an den Server geschickt werden:

- Bestätigung von Anzeigenamenänderungen
- Gruppen- und Transaktionserstellung
- Gruppenbeitritt
- Abgleich

**Reason** Da die Buttons jeweils nicht disabled werden, nachdem sie betätigt wurden, sind sie mehrmals betätigbar und zeigen auch kein visuelles Feedback.

**Fix** Die Buttons werden nach dem Betätigen nun disabled und durch einen Spinner ersetzt, bis der Request vom Server bearbeitet und ein entsprechender Response vom Client erhalten wurde.

## Swipe für Refresh

**Enhancement** Bei einem Swipe von oben nach unten passiert aktuell nichts. Stattdessen könnte man aktuelle Daten vom Server anfordern und diese anzeigen.

**Fix** Bei einem Swipe von oben nach unten wird `onEntry` aufgerufen.

## User Cards in den Gruppeneinstellungen sind zu groß und haben kein Spacing

**Bug** Die hervorgehobenen Cards um einzelne User und ihr jeweiliges Saldo in den Gruppeneinstellungen einer Gruppe sind zu groß und weisen kein Spacing zwischen-einander auf.

---

**Fix** Die Größe der hervorgehobenen Cards wurde verringert und es wurde Spacing eingefügt.

## **Exception statt Throwable fangen**

**Bug** An mehreren Stellen im Client wird Throwable gefangen.

**Reason** Statt Throwable sollte Exception gefangen werden. In Kotlin ist dies zwar sogar umstritten, aber wir haben uns so entschieden.

**Fix** Es wird nun überall Exception gefangen, wo vorher Throwable gefangen wurde.

## **Durchsichtige Profilebilder verdecken nur teilweise die generierten Platzhalterbilder**

**Bug** Wenn ein Nutzer ein durchsichtiges Profilbild hat, dann überdeckt dieses nur teilweise das aus den Initialien generierte Platzhalterprofilbild.

**Reason** Da das generierte Platzhalterbild stets unter dem Profilbild angezeigt wird, entsteht obiger Bug bei durchsichtigen Profilbildern.

**Fix** Das generierte Platzhalterbild wird nun nur angezeigt, wenn kein Profilbild vorliegt.

## **Unwichtigen Teilen des UIs wird zu viel Platz zugeschrieben**

**Enhancement** Teilen des UIs wie Benutzernamen wird wesentlich mehr Platz zugeschrieben als wichtigeren Teilen wie Saldoänderungen. Dies sorgt dafür, dass letztere verkürzt dargestellt werden, während erste meist in voller Länge zu sehen sind.

**Fix** Das UI teilt nun wichtigeren Komponenten mehr Raum zu, beispielsweise Saldoänderungen. Namen werden hingegen in einem kleineren Font dargestellt.

## **MonetaryFormatter und Zahlenfelder verwenden andere Zahlenformate**

**Bug** Je nach Lokalität verwenden MonetaryFormatter und Zahlenfelder andere Zahlenformate.

**Reason** MonetaryFormatter verwendet Zahlen, die im durch Locale.getDefault() gegebenen Zahlenformat vorliegen, während Zahlenfelder einfach die Serialisierungsmethoden von BigDec verwenden.

**Fix** Die verwendete Währung wird nun vom Server an den Client als ISO 4217 geschickt. Der Client findet anhand dieser Währung heraus, wie viele Nachkommastellen benötigt werden. Sowohl zur Anzeige von Geldbeträgen als auch für die Zahlenfelder werden diese Nachkommastellen dann verwendet. Das Zahlenformat sowie die Position der Währung hängen hingegen beide von der Lokalität des Android-Geräts ab, auf dem die App läuft.

## **UI Komponenten sollten Feedback zeigen, wenn sie gedrückt gehalten oder betätigt werden**

**Enhancement** Die Hallway Usability Tests (siehe Abschnitt 7) haben gezeigt, dass es unintuitiv ist, dass das Gedrückthalten von Nutzern keine Interaktion auslöst oder visuelles Feedback zeigt. Dies ist beispielsweise für das Starten des Abgleichprozesses relevant.

**Fix** Das Gedrückthalten von gewissen UI Komponenten öffnet nun ein Menu mit Aktionen, die mit der durch die Komponente repräsentierte Entität ausgeführt werden können.

## **Navigationsleiste sollte Hinweis anzeigen, wo man sich aktuell befindet**

**Enhancement** Siehe oben.

**Fix** Die Navigationsleiste zeigt nun das Icon ausgegraut an, welches einen dorthin befördert, wo man sich bereits befindet.

---

## **Eigene Exception für Ktor ConnectTimeoutException**

**Enhancement** Die Fehlermeldung von Ktors ConnectTimeoutException ist ungeeignet.

**Fix** Es wird eine eigene Exception mit eigener Fehlermeldung anstelle von Ktors ConnectTimeoutException eingeführt. Dies scheint weitverbreitet zu sein.

## **Spinner zeigen kein visuelles Feedback bei Timeouts und bemerken keine Fehler**

**Bug** Siehe oben. Ein Beispiel hierfür ist folgendes: Wenn die App gestartet wird, ohne dass der Server verfügbar ist, so wird irgendwann ein Netzwerkfehler geworfen, aber der Spinner der Ladeanimation dreht sich weiter.

**Reason** Angezeigte Spinner reagieren nicht auf geworfene Fehler.

**Fix** Betroffene Spinner werden durch einen Ladebildschirm ersetzt, der im Falle eines Timeouts einen Retry Button anzeigt.

## **Kurze Transaktionsliste beginnt am unteren Ende des Bildschirms**

**Enhancement** Kurze Transaktionlisten innerhalb einer Gruppe starten aktuell am oberen Ende des Bildschirms und wachsen nach unten hin. Andere Apps (siehe verbreitete Messenger-Dienste) lassen kurze Listen an Nachrichten am unteren Ende des Bildschirms beginnen und nach oben wachsen, bis diese genug Einträge enthalten, um den gesamten Bildschirm zu füllen.

**Fix** Das Verhalten von kurzen Transaktionlisten wurde an das aus anderen Apps bekannte Verhalten angepasst.

## **Währungszeichen wird teilweise als „XXX“ dargestellt**

**Bug** Siehe oben.

**Reason** Die Währung wird als Währung mit dem ISO Code „XXX“ (keine Währung) initialisiert. Die BalanceText Komponente verwendet die Währung sowie Lokalität zur Zeit ihrer Komposition. Falls der Client die Währung also erst nach dem Kompositionszeitpunkt einer BalanceText Komponente vom Server erhält, so wird die Währung mit ISO Code „XXX“ verwendet. Da DecimalFormatter diese Währung tatsächlich als „XXX“ darstellt (wovon wir nicht ausgegangen sind), erhalten wir in einem solchen Fall den obigen Bug.

**Fix** Die Währung ist nun nullable und wird erst gesetzt, wenn der Client eine Währung vom Server erhält. Falls in der Zwischenzeit ein Geldbetrag angezeigt werden muss, die Währung aber noch null ist, so wird eine Währung aus der Lokalität des Android-Geräts abgeleitet.

## Kein explizites Vorzeichen bei Geldbetrag Null

**Bug** Geldbeträge mit Wert Null werden mit explizitem positivem Vorzeichen angezeigt.

**Reason** Der Zahlenformattierer wurde so verwendet, dass er stets ein explizites Vorzeichen verwendet.

**Fix** Der Zahlenformattierer wird nun so verwendet, dass er bis auf Null stets ein explizites Vorzeichen verwendet.

## Gruppen sind in der Gruppenübersicht nicht vollständig anclickbar

**Bug** Wenn in der Gruppenübersicht auf die rechte Hälfte einer Gruppen Card geklickt wird, so öffnet sich die zugehörige Gruppenansicht nicht.

**Reason** Dies ist eine Regression, die dadurch ausgelöst wurde, dass der Geldbetrag, der auf der rechten Seite der Card angezeigt wird, auch anclickbar ist und einen Tooltip zeigt, wenn er lange gehalten wird.

**Fix** Die Composable für den Geldbetrag TooltippedOverflowingTextBox wurde so geändert, dass sie nur noch Clickevents fängt, wenn der Geldbetrag das Feld überfließt und die Tooltips erwünscht sind.

---

## **Abgleich mit sich selbst ist von den Gruppeneinstellungen aus möglich**

**Bug** In den Gruppeneinstellungen ist es möglich, sich selbst für einen Abgleich auszuwählen. Dies ist allerdings niemals sinnvoll.

**Reason** Dies stellt eine Regression dar, die beim Einführen der Buttons für den Abgleich mit Mitgliedern einer Gruppe in den Gruppeneinstellungen entstanden ist. Der Abgleichsbutton für den aktiven Nutzer wurde nicht deaktiviert.

**Fix** Der Abgleichsbutton in den Gruppeneinstellungen für den aktiven Nutzer ist nun deaktiviert.

# 3. Code Qualität

## 3.1. Statische Analyse

Für die statische Analyse wurde *detekt*<sup>1</sup> verwendet. Dabei wurde mit einigen Ausnahmen die Standardkonfiguration verwendet. Die getroffenen Ausnahmen finden sich für den Server in der Datei `server/detekt.yml` und für den Client in der Datei `client/detekt.yml`, bzw. jeweils in den Anhängen A.1 und A.2. In den Dateien sind die Ausnahmen auch im Detail begründet.

Die häufigsten gefundenen Probleme waren:

- *Magic Numbers*, insbesondere für Zeitangaben von Timeouts oder maximalen Retry-Werten.
- *Exception* bezogene Fehler, insbesondere wenn zu generische Ausnahmen gefangen oder geworfen wurden. Davon waren allerdings auch viele *false-positive*, z.B. die `main`-Methode, die alle Ausnahmen fangen muss, um den Server bei einer Ausnahme korrekt zu beenden. Weitere falsch-positive Fehler gab es, da es wegen Designentscheidungen von Kotlin teilweise nötig ist alle Ausnahmen zu fangen und danach erneut zu werfen, was von *detekt* nicht erkannt wurde.
- *Zu komplexe Methoden*, vor allem Methoden, die komplizierte Datenbank-Transaktionen durchführen. Diese wurden auf mehrere Methoden aufgeteilt.
- *Zu lange Zeilen*.

Alle von *detekt* gefundenen Probleme wurden behoben, außerdem wird *detekt* zusammen mit den Unit-Tests ausgeführt und lässt den Build fehlschlagen, wenn Fehler auftauchen, sodass auch in Zukunft mögliche Fehler mit *detekt* schnell erkannt werden.

---

<sup>1</sup><https://github.com/detekt/detekt>

## 3.2. Verbesserungen

### 3.2.1. Entfernen von Duplikaten im View Preview Code

Um bei der Entwicklung der Composables direkt ihr Aussehen zu sehen, entwickelten wir in der Implementierungsphase zu jeder Composable eine entsprechende Preview Composable mit Mock Implementationen von `User` und `Group`. Diese Mock Implementationen wurden jedoch jedes mal in Form von

```
1  val mockedUser = object: User {  
2      ...  
3  }
```

umgesetzt. Durch Einführung von `PreviewUser` und `PreviewGroup`, welche jeweils die Schnittstellen erben und nur das mindeste umsetzen, konnten wir solche Duplikate entfernen:

```
1  val mockedUser = PreviewUser(name)
```

### 3.2.2. Korrekte Instanziierung von `BigDec`

An manchen Stellen im Code wurde `BigDec` als `BigDec(BigDecimal(<Zahl>))` instanziiert. Da das nicht die vorgesehene Verwendung von `BigDec` ist, wurden diese Aufrufe durch `BigDec(<Zahl>)` ersetzt.

# 4. Unit Testing und Abdeckungen

## 4.1. Client

Auf dem Client wurde mit *JUnit 4*<sup>1</sup> getestet und mit *kover*<sup>2</sup> die Coverage berechnet. Die Wahl von JUnit 4 ist für Android Standard, da *JUnit 5*<sup>3</sup> noch nicht in Instrumented Tests unterstützt wird. Zur Unterstützung benutzen wir zusätzlich die Testing Libraries aus dem Implementierungsbericht sowie zusätzliche Utilities wie *assertj*<sup>4</sup>.

Insgesamt erreichen wir eine line coverage von 91,8% für den Client. Eine vollständige Zusammenfassung der Coverage befindet sich in Anhang A.4.

Angemerkt sei, dass in der Testabdeckung beim Methodenanteil teils leere, ungenutzte Methoden mitgezählt werden, die als Artefakt durch Inlining entstehen. Zudem wurde die UI, also die View Pakete, in der Coverageberechnung ausgeschlossen. Es wurde eine Instruction Coverage von 90% – 95% ohne UI angepeilt.

## 4.2. Server

Die Testüberdeckung wurde auf dem Server auch mittels kover gemessen. Wir haben festgestellt, dass wir bereits mit den in der Implementationsphase implementierten Tests eine relativ hohe Testüberdeckung von 85,3% Instruction Coverage erreichten. Außerdem haben wir festgestellt, dass große Teile des nicht abgedeckten Codes nicht sinnvoll durch Unit-Tests testbar sind. Dazu zählt beispielsweise der Code, der das Docker-Image der Postgres-Datenbank herunterlädt. Dieser wird zwar während der Tests ausgeführt, allerdings nur, wenn das Image zu dem Zeitpunkt noch nicht heruntergeladen wurde. Auch der Code zur Verbindung mit der Production-Datenbank kann im Unit-Test nicht ausgeführt werden, da während der Tests eine Docker-Datenbank verwendet wird, damit die Tests unabhängig von der Systemumgebung lauffähig sind.

---

<sup>1</sup><https://junit.org/junit4/>

<sup>2</sup><https://github.com/kotlin/kotlinx-kover>

<sup>3</sup><https://junit.org/junit5/>

<sup>4</sup><https://github.com/assertj/assertj>

---

	<i>class</i>	<i>method</i>	<i>branch</i>	<i>line</i>	<i>inst</i>
server.data	100,0%	85,7%	72,7%	92,6%	89,5%
server.data.config	100,0%	96,0%	85,3%	97,0%	95,4%
server.database	90,5%	88,1%	60,5%	83,1%	83,3%
server.executor	100,0%	100,0%		100,0%	100,0%
server.group	100,0%	100,0%	90,0%	100,0%	99,3%
server.handler	75,6%	65,0%	47,9%	83,6%	82,1%
server.transaction	100,0%	100,0%	80,0%	96,7%	94,0%
server.user	100,0%	100,0%	70,8%	95,7%	96,9%

Tabelle 4.1.: Testabdeckung auf dem Server nach Paket und Typ

Anstatt also insgesamt eine Zahl als Zielwert für die Test-Coverage festzulegen, wurde stattdessen jede Quellcodedatei des Servers einzeln durchgegangen und dabei mittels der von kover bereitgestellten Übersicht über die getesteten Zeilen besonders relevante Code-Pfade festgestellt, die nicht getestet waren. Für diese wurden dann entsprechende Tests hinzugefügt.

Insgesamt ergibt sich die in Tabelle 4.1 gezeigte Testabdeckung für den Server.

# 5. Integrationstests

Als Zwischenstufe zu den Systemtests (siehe Kapitel 6) und den Unit Tests (siehe Kapitel 4) enthält unsere Codebase auch eine begrenzte Anzahl an Model-Server Integrationstests. Diese Tests verwenden Methoden der Model Fassade, welche wiederum mit einem tatsächlich laufenden Server plus Datenbank verbunden ist.

Auf diese Art werden die Refresh-Funktionen des Models, sowie das Ändern des Nutzernamens getestet. Dabei verwenden wir den gleichen Control Server Mechanismus, wie für die Systemtests. Da die Integrationstests im Gegensatz zu den Systemtests nicht von der UI abhängen, sind sie leichter umzusetzen und erlauben eine genauere Inspektion der Ergebnisse.

Die Integrationstests befinden sich in `ModelTests`.

# 6. Systemtests

Um automatisierte Systemtests umzusetzen haben wir instrumentierte Android Compose Tests verwendet. Wir haben hierzu einen Control Server geschrieben, der auf dem Host läuft und dafür verantwortlich ist das Backend korrekt zu starten. Für jeden Test der auf dem Android-Gerät oder im Emulator läuft wird so bei Bedarf ein neuer Applikationsserver gestartet und erreichbar gemacht. Jeder Applikationsserver verwendet eine nicht-persistente Datenbank im selben Docker Container, die jeweils mit gewissen Daten initialisiert wird.

Automatisch überprüft werden alle Produktfunktionen aus dem Pflichtenheft; einzig die Anmeldung war nicht möglich durch Compose Tests zu überprüfen. Diese muss manuell überprüft werden. Um eine Produktfunktion zu überprüfen werden jeweils die nötigen Eingaben auf dem Android-Gerät oder dem Emulator getätigter und anschließend überprüft, dass die richtigen Informationen angezeigt werden. Zu beachten ist hierbei, dass Animationen auf dem Android-Gerät oder dem Emulator ausgeschaltet sein müssen, da diese bekannterweise für ein Fehlschlagen mancher Compose Tests sorgen.

Die Systemtests befinden sich in `FunctionalityTests` und der Control Server im `integration` Projekt.

## 6.1. Server Verfügbarkeit

Wir haben mit [uptimerobot.com](https://uptimerobot.com)<sup>1</sup> über zwei Wochen die Verfügbarkeit unseres Servers getestet. Bei IPv4 Verbindungen beträgt die Verfügbarkeit 100%, während bei IPv6 Verbindungen die Verfügbarkeit, aufgrund von externen Ursachen, bei 98,191% liegt. Beide Werte liegen über der im Pflichtenheft geforderten 95% Verfügbarkeit der Serversoftware.

---

<sup>1</sup><https://stats.uptimerobot.com/SPaLHE4CVP>

# 7. Usability Testing

Wir haben Hallway Usability Testing mit einem eigens entwickelten Fragebogen (siehe Anhang A.3) durchgeführt. Dieser beinhaltet unter anderem die (meisten) Fragen aus dem System Usability Scale. Im Folgenden werden wir die Ergebnisse der Hallway Tests auswerten und umgesetzte Verbesserungen erläutern. Insgesamt haben wir 10 Personen interviewt.

## 7.1. Auswertung

**Installation** Die Installation war für Nutzer, die F-Droid nicht kannten schwer und die Interviewer mussten häufig helfen, was zu erwarten war. Ein Nutzer meinte, dass er es einfacher fände, wenn die apk-Datei direkt von der Webseite herunterladbar wäre.

**Anmeldung** Bei der Anmeldung musste keinem Teilnehmer geholfen werden. Jeder Teilnehmer vergab die Note 1. Wir konkludieren hieraus, dass die Anmeldung keinerlei Verbesserungen bedarf.

**Gruppe erstellen** Jeder Teilnehmer erstellte erfolgreich die Gruppe und lud den Interviewer ein. Die vergebenen Noten waren meist eine 1, zwei Teilnehmer vergaben eine 3. Der Grund für die schlechtere Benotung war beide Male die Verortung des Einladungslinks in den Gruppeneinstellungen, aufgrund derer die Teilnehmer leichte Schwierigkeiten hatten, den Einladungslink zu finden. Wir leiten ab, dass das Erstellen einer neuen Gruppe problemlos, das Einsehen des Einladungslinks hingegen verbesserungswürdig ist. Als Vorschläge wurde hier beigetragen, den Einladungslink bei einer neu erstellten Gruppe direkt in der Gruppenübersicht anzuzeigen, was wir umgesetzt haben. Bei diesem Schritt des Hallway Testings ist auch Bug 2 aufgefallen.

**Einkäufe eintragen** Jeder Teilnehmer trug erfolgreich den gegebenen Einkauf ein. Die Teilnehmer vergaben hierfür fast ausschließlich die Note 1, bis auf einen Teilnehmer, der die Note 5 vergab und einen Teilnehmer, der eine 2 vergab. Letzterer merkte an, dass das UI zwar gut gedacht sei, ein Info Button oder Erklärtext jedoch zur Bedeutung von Zahlung und Ausgabe jedoch hilfreich gewesen wäre. Der Teilnehmer, der die 5 vergab, verstand das UI anfangs falsch, was durch einen Erklärtext auch behoben wäre. Diese Erklärtexte setzten wir teils auch um. Wir konkludieren aus diesem Teil des Hallway Testings, dass das Eintragen von Ausgaben verständlich ist.

**Bezahlung eintragen** Jeder Teilnehmer schaffte es, die erste gegebene Bezahlung einzutragen. Hierbei wurden fast ausschließlich gute Noten vergeben, bis auf ein Teilnehmer, der eine 3 vergab. Dieser war vom einzutragenden Vorzeichen (positiv) verwirrt. Alle anderen Teilnehmer wählten intuitiv kein Vorzeichen. Allerdings waren die Teilnehmer teils von den angezeigten Vorzeichen der erstellten Zahlung verwirrt.

Es scheiterten mehrere Nutzer daran, die zweite gegebene Zahlung einzutragen, weshalb es dementsprechend zweimal die Note 6 gab. Die restlichen Nutzer vergaben gute Noten. Alle gescheiterten Nutzer versuchten, die Zahlung als Ausgabe zu verbuchen und kamen nicht auf die Idee, ein negatives Vorzeichen bei einer Zahlung einzutragen. Erst auf Hinweis der Interviewer konnten diese Teilnehmer die gegebene Zahlung verbuchen. Wir leiten ab, dass die von uns entworfene Art, Zahlungen einzutragen, also jeweils für jeden beteiligten Nutzer die Saldoänderung anzugeben, für Außenstehende unintuitiv sein kann. Um dies zu beheben sind wir auf das Feedback der gescheiterten Nutzer eingegangen: Es können nur noch absolute Geldbeträge eingetragen werden. Um das Vorzeichen einzutragen, wählt man nun die Richtung der Zahlung, also wer Geld an wen zahlt.

Anschließend den Betrag abzulesen, den der Interviewer dem Teilnehmer nun noch schuldet, stellt hingegen kein Problem dar. Jeder Teilnehmer vergab hier eine gute Note.

Insgesamt konkludieren wir, dass die von uns entworfene Art, Zahlungen einzutragen, für Außenstehende teils unintuitiv wirkt und entweder eine Einführung bzw. Erklärtexte oder aber eine Abänderung von Nöten ist. Die Darstellung von Saldi scheint allerdings sinnvoll gewählt zu sein.

**Abgleich eintragen** Den Abgleich einzutragen erledigte jeder Teilnehmer erfolgreich. Die meisten Teilnehmer vergaben eine gute Note. Die wenigen Teilnehmer, die eine schlechte Note vergaben, hatten Schwierigkeiten die Abgleichsfunktion zu finden. Sie waren die ersten paar Teilnehmer, an denen wir den Hallway Usability Test durchführten. Sie versuchten alle, auf einen Nutzer zu clicken oder diesen gedrückt zu halten, in der Erwartung, dass sich so eine Möglichkeit zum Abgleich mit diesem

Nutzer ergibt. Diese Funktion realisierten wir nach den ersten anfänglichen Usability Tests. Anschließend vergab jeder weitere Teilnehmer eine gute Note, wobei nicht alle die neu implementierte Funktionalität nutzten, um zum Abgleich zu gelangen. Einige verwendeten auch den von uns anfangs intendierten Weg der Navigation über die Navigationsleiste im Main Menu.

Wir stellen fest, dass die ursprünglich von uns vorgesehene Art der Navigation und Verortung der Abgleichsfunktionalität anscheinend nicht so intuitiv ist, wie vorgesehen. Nachdem wir allerdings eine weitere, aus anderen modernen Apps vertraute Möglichkeit der Navigation hinzugefügt haben, schienen die Teilnehmer soweit zufrieden.

**Ich kann mir vorstellen, die App regelmäßig zu nutzen** Die meisten Teilnehmer stimmten der Frage zu, bis auf einen Teilnehmer. Dieser merkte an, dass die App nicht ausgereift sei und eine schlechte UX habe. Von den anderen Nutzern merkten viele an, dass sie die Funktionalität der App praktisch für den Alltag fänden. Hierbei ist allerdings anzumerken, dass unsere Gruppe an Teilnehmer vor allem aus anderen Studenten bestand, weshalb ein gewisser Bias vorhanden ist.

Wir leiten ab, dass unsere App für Studenten durchaus sinnvoll sein kann.

**Ich empfinde die App als unnötig komplex** Alle Teilnehmer bewerteten die App als nicht unnötig komplex.

Wir konkludieren, dass die App keine unnötige Komplexität aufweist.

**Ich empfinde die App als einfach zu nutzen** Es gab starke Abweichungen in der Bewertung der Einfachheit der Nutzung der App. Teilnehmer, die die App als schwer zu benutzen bewerteten, merkten alle an, dass sie einfach zu benutzen sei, sobald man verstanden hat, wie Zahlungen und Ausgaben funktionieren und die verschiedenen Vorzeichen zu deuten sind.

Wir haben versucht, durch Erklärtexte sowie eine Abwandlung des UIs beim Eintragen von Zahlungen auf die Kritik einzugehen.

**Ich denke, dass ich technische Unterstützung brauchen würde, um die App zu nutzen** Alle bis auf einen Nutzer stimmten überhaupt nicht zu, dass sie technische Unterstützung bräuchten, um die App zu nutzen. Dieser eine Nutzer meinte, dass er ohne textuelle Anleitungen oder die Unterstützung des Interviewers die App nicht hätte nutzen können. In Zukunft meinte der Nutzer aber, dass er die App auch ohne Unterstützung nutzen könne.

Wir konkludieren, dass sich die App auch von technisch nicht aversierten Nutzern bedient werden kann.

**Ich finde, dass die App zu viele Inkonsistenzen aufweist** Die meisten Nutzer stimmten nicht zu, dass die App zu viele Inkonsistenzen aufweist. Der eine Nutzer, der zustimmte, meinte, dass die verschiedenen Funktionalitäten (primär Transaktionen anlegen und Abgleichen) schlecht verortet und zu weit voneinander entfernt seien. Auf dieses Feedback sind wir eingegangen, indem wir Aktionsmenüs eingefügt haben, wenn Nutzer oder auch Gruppen gedrückt gehalten werden. Außerdem wurde das Farbschema kritisiert, woraufhin wir dieses leicht angepasst haben.

Nachdem diese Inkonsistenzen behoben wurden, gehen wir davon aus, dass keine weiteren großen Inkonsistenzen bei der App vorliegen.

**Ich kann mir vorstellen, dass die meisten Leute die App schnell zu beherrschen lernen** Alle Nutzer stimmten zu, dass die meisten Leute die App schnell zu beherrschen lernen. Ein Nutzer merkte an, dass Personen, die nicht regulär moderne Apps verwenden, Schwierigkeiten bei der Bedienung haben würden. Da dies aber kein explizites Entwurfskriterium darstellt, werden wir dies nicht weiter behandeln.

**Ich empfinde die Bedienung der App als sehr umständlich** Alle bis auf einen Nutzer stimmten nicht zu, dass sie die Bedienung der App als sehr umständlich empfanden. Der eine Nutzer, der die Bedienung der App als umständlich empfand, kritisierte die Kleinschrittigkeit der Funktionalität. Dies stellt aber eine absichtlich getroffene Entwurfsentscheidung dar.

Wir folgern, dass die Bedienung der App nicht besonders umständlich ist, zumindest nicht umständlicher als inhärent notwendig.

**Ich musste eine Menge Dinge lernen, bevor ich die App nutzen konnte** Fast alle Nutzer stimmten nicht zu, dass sie eine Menge Dinge lernen mussten, bevor sie die App nutzen konnten. Dem einen Nutzer, der leicht zustimmte, musste die Bedeutung der Vorzeichen erklärt werden.

Wir leiten ab, dass nur eine geringe Anzahl an Dingen zu lernen sind, bevor Nutzer die App verwenden können. Diese Dinge versuchen wir durch die hinzugefügten Erklärtexte zu erklären.

**Was würdest du an der App verändern?** Hier wurde ein Fix des Bugs 2 angemerkt. Außerdem wurde angemerkt, dass neben den Buttons zum Gruppe Verlassen in den Gruppeneinstellungen weitere Buttons z.B. für Abgleichen eingefügt werden sollen. Dies haben wir umgesetzt. Außerdem wurde eine Anpassung des Farbschemas vorgeschlagen, was wir auch umgesetzt haben.

**Sonstige Anmerkungen** Ein Teilnehmer merkte an, dass ihm die Vergabe von Schulnoten schwer falle, weil er kein Vergleichsprodukt kenne. Da die Ergebnisse des Teilnehmers aber nicht stark von denen der anderen Teilnehmer abweichen, sollte dies kein Problem darstellen.

## 7.2. Fazit

Insgesamt sind wir nach anfänglichen Problemen mit den Ergebnissen der Hallway Usability Tests recht zufrieden. Das Nutzerfeedback konnte größtenteils umgesetzt werden und die App scheint nach wenigen Erklärungen verwendbar zu sein.

## 7.3. Umgesetzte Verbesserungen

Aufgelistet sind im Folgenden Verbesserungen, die wir ausgehend vom Feedback der Teilnehmer umgesetzt haben:

- Bei einer neu erstellten Gruppe bzw. einer Gruppe ohne Transaktionen wird nun der Einladungslink in der Gruppenansicht angezeigt, sodass man nach Erstellung einer neuen Gruppe nicht zuerst in die Einstellungen navigieren muss, um den Einladungslink zu teilen.
- Erklärtexte, die den Nutzer etwas leiten. Insbesonders wurden teils vorzeichenbehaftete Geldbeträge wie „+5€“ durch eine textuelle Beschreibung des Vorzeichens ersetzt: „You receive: 5€“.
- Bei Zahlungen trägt man nicht mehr die Saldoänderungen für jeden betroffenen Nutzer ein, sondern wählt den zahlenden Nutzer sowie die absoluten Geldbeträge, die dieser an die anderen Nutzer zahlt. Dies stellt zwar eine Einschränkung der Mächtigkeit von Zahlungen dar, sodass nun teils mehrere Zahlungen nötig sind, wo vorher das Eintragen einer Zahlung ausreichend war, ist aber deutlich intuitiver.
- Neben dem Button zum Entfernen eines Nutzers befindet sich in den Gruppeneinstellungen jetzt auch ein Button zum Abgleichen.
- Beim Gedrückthalten von Nutzern oder Gruppen wird ein Aktionsmenü mit Aktionen bezüglich diesen Entitäten angezeigt. Insbesondere können die Gruppeneinstellungen nun auch durch Gedrückthalten eines Gruppennamens geöffnet werden.

- Leichte Anpassung des Farbschemas.

## **8. Abnahmetests**

Wir sind die im Pflichtenheft beschriebenen Nutzungsszenarien durchgegangen und haben hierbei keinerlei Fehler festgestellt. Jedes Nutzungsszenario konnte mittels implementierter Funktionalität umgesetzt werden. Die einzige Ausnahme ist Szenario 6.6, welches das Wunschkriterium  $\langle RC8 \rangle$  zum Rückbuchen von Buchungen verlangt. Dieses Wunschkriterium wurde nicht implementiert.

# A. Anhang

## A.1. detekt-Konfiguration auf dem Server

Hier findet sich die Konfiguration des statischen Analysewerkzeugs *detekt* für den Server inklusive Begründungen für die getroffenen Ausnahmen.

```
1 style:
2   # The maximum line length is relaxed to 132 characters which is the width of a
3   # DEC VT-340 terminal.
4   MaxLineLength:
5     maxLength: 132
6     ignoreAnnotated: [ "org.junit.jupiter.api.Test" ]
7
8   # Wildcard imports make the code more readable because they shrink the size of
9   # the import section. This is especially beneficial to extension methods as to
10  # not import every extension method separately.
11  WildcardImport:
12    active: false
13
14  # This check can be suppressed using an annotation. This proved useful in some
15  # contexts, where giving the constants a name is not a helpful documentation.
16  MagicNumber:
17    ignoreAnnotated: [ "com.pse_app.server.data.Magic" ]
18
19
20 naming:
21   # Our domain name and therefore our packages contain an underscore. This isn't
22   # allowed in the default settings for some reason.
23   PackageNaming:
24     packagePattern: "[a-zA-Z_]+(\\.[a-zA-Z_][A-Za-z0-9_]*)*"
25
26
27 exceptions:
28   # These two checks are disabled on a per-method basis as they produce valuable
29   # results in most cases but in some scenarios it is a necessity to catch every
30   # exception. Per-method reasoning can be found below:
31   # - main
32   #   The entrypoint function catches every exception to give it proper logging.
33   #   Any exception that makes it here is a critical failure that will result in
34   #   the immediate termination of the server.
35   # - tryTo
36   #   This is a 'Result' method that is supposed to catch only a single concrete
37   #   type of exception. However, Kotlin does not allow catch clauses on reified
38   #   type variables. Therefore, every exception is caught. If the exception has
39   #   the wrong type it is rethrown which is opaque to the surrounding code.
40   # - startIdTokenAuth
41   #   This method starts a thread that periodically reloads the OpenID discovery
42   #   data. If the reload fails for, the thread must be prevented from dying, so
```

```
43  # further reload attempts are attempted. This requires a catch-all exception
44  # handler.
45  # - logOnError
46  # This method catches exceptions during route handling so they can be logged
47  # properly.
48  TooGenericExceptionCaught:
49    ignoreFunction:
50      - "main"
51      - "tryTo"
52      - "startIdTokenAuth"
53      - "com.pse_app.server.handler.RequestRouter.logOnError"
54  InstanceOfCheckForException:
55    ignoreFunction:
56      - "tryTo"
57
58  # This check virtually always produces false-positive results due to the error
59  # handling strategy used on the server: exceptions are converted to Results as
60  # early as possible which swallows the exception but creates an errored result
61  # that propagates up until it is properly handled.
62  SwallowedException:
63    active: false
64
65
66  performance:
67    # The spread operator is necessary to call java vararg methods. Therefor it is
68    # not possible to abstain from using it.
69  SpreadOperator:
70    active: false
71
72
73  complexity:
74    # These thresholds have been lowered especially because the interfaces between
75    # the server components exceed the default settings of this check. To keep the
76    # structure and interfaces from the design phase it was necessary to relax
77  TooManyFunctions:
78    thresholdInFiles: 30
79    thresholdInClasses: 30
80    thresholdInInterfaces: 30
81    thresholdInObjects: 30
82    thresholdInEnums: 30
83
84    # The new two checks are relaxed primarily due to test classes containing lots
85    # of test cases that belong together semantically and use common functionality
86    # for mocking. Because test cases are independent of each other and don't call
87    # reference other test cases this was deemed reasonable.
88  LongMethod:
89    threshold: 250
90  LargeClass:
91    threshold: 1000
92
93    # The entrypoint function should not be treated like a method but like a class
94    # especially as it has its own source file with just one method. And while the
95    # main method is complex in relation to other methods, its complexity is about
96    # average for a class containing program logic. The same reasoning applies for
97    # the bootstrap function.
98  CyclomaticComplexMethod:
99    ignoreFunction: [ "main", "bootstrap" ]
100
101   # Named default parameters don't necessarily contribute to the complexity of a
102   # method. Often times it is useful to have many named default parameters where
103   # only a few get set when the function is used.
104  LongParameterList:
```

```
105 ignoreDefaultParameters: true
```

## A.2. detekt-Konfiguration auf dem Client

Hier findet sich die Konfiguration des statischen Analysewerkzeugs *detekt* für den Client inklusive Begründungen für die getroffenen Ausnahmen.

```

1 style:
2   # The maximum line length is relaxed to 132 characters which is the width of a
3   # DEC VT-340 terminal.
4   MaxLineLength:
5     maxLength: 132
6     ignoreAnnotated: [ "org.junit.Test" ]
7
8   # Wildcard imports make the code more readable because they shrink the size of
9   # the import section. This is especially beneficial to extension methods as to
10  # not import every extension method separately.
11  WildcardImport:
12    active: false
13
14 MagicNumber:
15   # Composables may need a large amount of different arbitrary numbers with no
16   # clear meaning to ensure a good looking UI
17   ignoreAnnotated:
18     - Preview
19     - Composable
20
21 UnusedPrivateProperty:
22   ignoreAnnotated:
23     - SpecifiedInDesign
24
25 ReturnCount:
26   max: 4
27
28 ThrowsCount:
29   max: 10
30
31 LoopWithTooManyJumpStatements:
32   maxJumpCount: 2
33
34 config:
35   validation: true
36   warningsAsErrors: false
37
38 naming:
39   PackageNaming:
40     packagePattern: '[a-z]+(\.[a-z][A-Za-z0-9_]*)*'
41   FunctionNaming:
42     ignoreAnnotated:
43       - Composable
44
45 complexity:
46   # These thresholds have been lowered especially because the interfaces between
47   # the client components exceed the default settings of this check. To keep the
48   # structure and interfaces from the design phase it was necessary to relax
```

```
50  TooManyFunctions:
51    thresholdInFiles: 30
52    thresholdInClasses: 30
53    thresholdInInterfaces: 30
54    thresholdInObjects: 30
55    thresholdInEnums: 30
56
57    # The new two checks are relaxed primarily due to test classes containing lots
58    # of test cases that belong together semantically and use common functionality
59    # for mocking. Because test cases are independent of each other and don't call
60    # reference other test cases this was deemed reasonable.
61  LongMethod:
62    threshold: 250
63  LargeClass:
64    threshold: 1000
65
66  # MonetaryOutlinedTextField is a false positive report of a too complex method
67  # because detekt counts inline functions toward the complexity of an enclosing
68  # scope. However, inline functions often times reduce complexity while keeping
69  # all the functionality not only enclosed in the class but directly inside the
70  # method, thereby employing an even stronger secrecy principle than the common
71  # secrecy principle employed on classes alone.
72  CyclomaticComplexMethod:
73    ignoreFunction: [MonetaryOutlinedTextField]
74
75  LongParameterList:
76    # Named default parameters don't necessarily contribute to the complexity of
77    # a method. In a lot of cases it is useful to have quite a few named default
78    # parameters where only a few get set when the function is used.
79    ignoreDefaultParameters: true
80
81    # Composables may need many parameters to fully specify appearance
82    ignoreAnnotated:
83      - Composable
84
85  ComplexCondition:
86    threshold: 5
```

### A.3. Hallway Usability Testing Fragebogen



## PSE Usability Test

### Durchführungsanleitung

Der Interviewer füllt diesen Fragebogen aus. Am besten erst im Nachhinein ausfüllen und stattdessen das Gespräch aufnehmen. Der Teilnehmer wird zuerst mit ein paar Aufgaben durch die Nutzung der App geleitet und soll die Leichtigkeit der Umsetzung jeweils bewerten. Falls der Teilnehmer eine Aufgabe nicht absolvieren kann, wird diese vom Interviewer durchgeführt und zur nächsten Aufgabe übergegangen. Abschließend folgen weitere Fragen über die insgesamte Benutzbarkeit.

Es wird vorausgesetzt, dass der Teilnehmer ein Android-Gerät sowie einen Google Account besitzt. Der Interviewer sollte auf seinem Gerät die selbe Version der PSE App installiert haben und mit seinem Google Account eingeloggt sein.

Alternativ kann der Interviewer die App im Android-Emulator laufen lassen und dem Teilnehmer sein Android-Gerät zur Verfügung stellen. **In diesem Fall kann die Installation übersprungen werden.**

**Interviewer \***

Name des Interviewers

**Teilnehmer \***

Name des Teilnehmers

**App Version/Commit \***

v1.2

## Szenarien

### Installieren (nur ausfüllen, wenn der Teilnehmer die App selbst installiert hat)

"Installiere die PSE App über F-Droid, indem du den Repository Link auf [pse-app.com](http://pse-app.com) verwendest."

#### Interviewer musste bei der Installation

A Ja

B Nein

#### Intuitiv? (Schulnote)

1

2

3

4

5

6

Anmerkungen

## Anmeldung

"Melde dich über deinem Google Account in der App an."

#### Interviewer musste bei der Anmeldung helfen \*

A Ja

B Nein

#### Intuitiv? (Schulnote) \*

1	2	3	4	5	6
---	---	---	---	---	---

Anmerkungen

### Gruppe erstellen

"Erstelle eine Gruppe mit einem von dir gewählten Namen und lade den Interviewer in diese Gruppe ein."

**Erfolg? \***

**A** Ja

**B** Nein

**Intuitiv? (Schulnote) \***

1	2	3	4	5	6
---	---	---	---	---	---

Anmerkungen

### Einkäufe eintragen

"Wir (du und der Interviewer) haben zusammen einen Einkauf getätigt und du hast den gesamten Einkauf (24 Euro und 39 Cent) bezahlt. Wir wollen den Betrag aber später gleichmäßig aufteilen. Trage den Einkauf so in der App ein."

**Erfolg? \***

**A** Ja

**B** Nein

**Intuitiv? (Schulnote)** \*

1    2    3    4    5    6

Anmerkungen

**Bezahlung eintragen**

"Der Interviewer hatte nach dem Eintragen des Einkaufes noch etwas Kleingeld in der Tasche und gab dir dieses (5 Euro und 42 Cent). Trage dies in der App ein."

**Erfolg?** \*

A Ja

B Nein

**Intuitiv? (Schulnote)** \*

1    2    3    4    5    6

Anmerkungen

"Nachdem der Interviewer dir in der Vergangenheit ein Mensa-Essen für 3 Euro 20 Cent ausgegeben hat, entscheidest du dich nun, dies in der App einzutragen."

**Erfolg?** \*

A Ja

B Nein

**Intuitiv? (Schulnote)** \*

1    2    3    4    5    6

Anmerkungen

"Finde in der App heraus, wie viel der Interviewer dir jetzt noch schuldet."

**Erfolg?** \*

**A** Ja

**B** Nein

**Intuitiv? (Schulnote)** \*

1    2    3    4    5    6

Anmerkungen

**Abgleich eintragen**

"Eine Woche später haben wir uns getroffen und wollen nun untereinander Schulden begleichen. Finde heraus, wie man dies in der App einträgt, ohne selbst den zu begleichenden Betrag berechnen zu müssen und trage den Abgleich ein, sodass ihr am Ende keine Schulden untereinander habt."

**Erfolg?** \*

**A** Ja

**B** Nein

**Intuitiv? (Schulnote)** \*

1	2	3	4	5	6
---	---	---	---	---	---

Anmerkungen

**Feedback**

**Ich kann mir vorstellen, die App regelmäßig zu nutzen.** \*

1	2	3	4	5	6
---	---	---	---	---	---

stimme voll und  
ganz zu

stimme überhaupt  
nicht zu

Anmerkungen

**Ich empfinde die App als unnötig komplex.** \*

1	2	3	4	5	6
---	---	---	---	---	---

stimme voll und  
ganz zu

stimme überhaupt  
nicht zu

Anmerkungen

**Ich empfinde die App als einfach zu nutzen.** \*

1	2	3	4	5	6
---	---	---	---	---	---

stimme voll und  
ganz zu

stimme überhaupt  
nicht zu

Anmerkungen

**Ich denke, dass ich technische Unterstützung brauchen würde, um die App zu nutzen.** \*

1	2	3	4	5	6
---	---	---	---	---	---

stimme voll und ganz zu

stimme überhaupt nicht zu

Anmerkungen

**Ich finde, dass die App viele Inkonsistenzen aufweist.** \*

1	2	3	4	5	6
---	---	---	---	---	---

stimme voll und ganz zu

stimme überhaupt nicht zu

Anmerkungen

**Ich kann mir vorstellen, dass die meisten Leute die App schnell zu beherrschen lernen.** \*

1	2	3	4	5	6
---	---	---	---	---	---

stimme voll und ganz zu

stimme überhaupt nicht zu

Anmerkungen

**Ich empfinde die Bedienung der App als sehr umständlich.** \*

A. Anhang

---

1	2	3	4	5	6
---	---	---	---	---	---

stimme voll und  
ganz zu

stimme überhaupt  
nicht zu

Anmerkungen

**Ich musste eine Menge Dinge lernen, bevor ich die App nutzen konnte.**

\*

1	2	3	4	5	6
---	---	---	---	---	---

stimme voll und  
ganz zu

stimme überhaupt  
nicht zu

Anmerkungen

**Was würdest du an der App verändern?**

Anmerkungen

**Sonstige Anmerkungen**

Anmerkungen

**Submit**

## A.4. Client Coverage

client: Overall Coverage Summary					
Package	Class, %	Method, %	Branch, %	Line, %	Instruction, %
all classes	91% (111/122)	69% (31/451)	65.% (201/306)	91.8% (1148/1250)	89.8% (7348/8181)
<b>Coverage Breakdown</b>					
Package	Class, %	Method, %	Branch, %	Line, %	Instruction, %
com.pse_app.client	42.9% (3/7)	33.3% (3/9)	30% (3/10)	34% (33/97)	
com.pse_app.client.model.data_layer	80% (4/5)	82.4% (14/17)	54.2% (13/24)	90.2% (37/41)	92.9% (235/253)
com.pse_app.client.model.facade	90.5% (19/21)	52.6% (5/97)	64.5% (45/76)	85.9% (219/255)	81% (1824/2253)
com.pse_app.client.model.repositories	75% (3/4)	60% (3/5)	75% (6/8)	66.7% (8/12)	62.5% (95/152)
com.pse_app.client.model.repositories.data	100% (9/9)	100% (9/9)		100% (34/34)	100% (163/163)
com.pse_app.client.model.repositories.local	100% (6/6)	100% (22/22)	71.4% (10/14)	100% (53/53)	98.8% (498/504)
com.pse_app.client.model.repositories.remote	92.3% (12/13)	97.4% (38/39)	71.2% (37/52)	93.9% (185/197)	95% (1970/2074)
com.pse_app.client.ui	0% (0/2)	0% (0/21)	0%	0% (0/18)	0% (0/72)
com.pse_app.client.ui.view_model	100% (6/6)	87% (20/23)	54.5% (12/22)	86.4% (38/44)	90.5% (286/316)
com.pse_app.client.ui.view_model.create_group	100% (2/2)	83.3% (5/6)	100% (2/2)	100% (15/15)	100% (32/32)
com.pse_app.client.ui.view_model.expense	100% (4/4)	75% (18/24)	66.7% (12/18)	97.6% (83/85)	99.5% (374/376)
com.pse_app.client.ui.view_model.group	100% (4/4)	72.2% (13/18)	100% (2/2)	100% (27/27)	100% (152/152)
com.pse_app.client.ui.view_model.group_settings	100% (7/7)	74.2% (23/31)	63.6% (14/22)	98.6% (70/71)	97.8% (361/369)
com.pse_app.client.ui.view_model.join_group	100% (5/5)	80% (12/15)	83.3% (5/6)	97.3% (36/37)	100% (99/99)
com.pse_app.client.ui.view_model.login	100% (6/6)	76.5% (13/17)	77.8% (14/18)	90.5% (57/63)	94.6% (193/204)
com.pse_app.client.ui.view_model.main_menu	100% (6/6)	81% (17/21)	50% (5/10)	95.7% (44/46)	92.8% (269/290)
com.pse_app.client.ui.view_model.payment	100% (3/3)	58% (14/25)	50% (3/6)	100% (65/65)	100% (221/221)
com.pse_app.client.ui.view_model.profile	100% (3/3)	61.9% (13/21)	62.5% (5/8)	95.1% (39/41)	95.2% (180/189)
com.pse_app.client.ui.view_model.settle_up_group_selection	100% (5/5)	70% (14/20)	66.7% (12/18)	99% (104/105)	99.3% (279/281)
com.pse_app.client.ui.view_model.settle_up_user_selection	100% (4/4)	81.8% (9/11)	100% (3/3)	100% (84/84)	

Abbildung A.1.: Code Coverage auf dem Client