Hochschule Worms

Fachbereich Informatik

Studiengang Angewandte Informatik B.Sc.

**PetPal Dokumentation**

*Modul:*

*502 Entwicklung Mobiler Anwendungen*

vorgelegt von:

**Mikulas Willaschek**

Matrikelnummer: 677 636

[Inf3825@hs-worms.de](mailto:Inf3825@hs-worms.de)

bei: Prof. Dr. Kurpjuweit

Bearbeitungszeitraum: Sommersemester 2023

Abgabedatum: 8. August 2023

**Inhaltsverzeichnis**

[1 Kontext 4](#_Toc140660682)

[1.1 Aufgabenstellung 4](#_Toc140660683)

[1.2 PetPal 4](#_Toc140660684)

[1.3 Test – Account 5](#_Toc140660685)

[2 Anforderungen 5](#_Toc140660686)

[2.1 Allgemeine Anforderungen 5](#_Toc140660687)

[2.2 Spezifische Anforderungen 5](#_Toc140660688)

[3 Code 7](#_Toc140660689)

[3.1 XML-Dateien 7](#_Toc140660690)

[3.2 Kotlin Activities 8](#_Toc140660691)

[3.2.1 MainActivity 8](#_Toc140660692)

[3.2.2 ChangePasswordActivity 9](#_Toc140660693)

[3.2.3 ImprintActivity 9](#_Toc140660694)

[3.2.4 MessageDetailsActivity 9](#_Toc140660695)

[3.2.5 MessagesListActivity 10](#_Toc140660696)

[3.2.6 MessageWriteActivity 10](#_Toc140660697)

[3.2.7 PetAddActivity 10](#_Toc140660698)

[3.2.8 PetProfileEditActivity 11](#_Toc140660699)

[3.2.9 PetProfileShowActivity 11](#_Toc140660700)

[3.2.10 PetsittterDetailsActivity 11](#_Toc140660701)

[3.2.11 PetsitterRegisterActivity 11](#_Toc140660702)

[3.2.12 PetsListActivity 12](#_Toc140660703)

[3.2.13 SearchCreateActivity 12](#_Toc140660704)

[3.2.14 SearchDetailsActivity 12](#_Toc140660705)

[3.2.15 SearchesListActivity 13](#_Toc140660706)

[3.2.16 UserLoginActivity 13](#_Toc140660707)

[3.2.17 UserProfileEditActivity 13](#_Toc140660708)

[3.2.18 UserProfileShowActivity 14](#_Toc140660709)

[3.2.19 UserRegisterActivity 14](#_Toc140660710)

[3.2.20 NavigationDrawer 14](#_Toc140660711)

[3.3 Daten Modelle 15](#_Toc140660712)

[3.3.1 Message 15](#_Toc140660713)

[3.3.2 Pet 15](#_Toc140660714)

[3.3.3 Petsitter 15](#_Toc140660715)

[3.3.4 Search 15](#_Toc140660716)

[3.4 Datenquellen 15](#_Toc140660717)

[3.5 Adapter 16](#_Toc140660718)

[3.5.1 MessageAdapter 16](#_Toc140660719)

[3.5.2 PetsitterAdapter 16](#_Toc140660720)

[3.5.3 SearchesAdapter 16](#_Toc140660721)

[3.5.4 SelectPetsAdapter 16](#_Toc140660722)

[3.5.5 ShowPetsAdapter 16](#_Toc140660723)

[3.6 Test 16](#_Toc140660724)

# Kontext

## 1.1 Aufgabenstellung

Im Modul 502 Entwicklung mobiler Anwendungen, ist es unsere Aufgabe eine natives (Kotlin/Swift) App Projekt mit bestimmten Anforderungen (vgl. Kap. 2.1) zu programmieren. Zusätzliche Nice-to-have Anforderungen wurden ebenfalls implementiert sowie eigene Anforderungen (vgl. Kap. 2.2.) und werden im Laufe dieses Kapitels näher erläutert. Ich arbeitete allein und meine Wahl fiel auf die Programmierung mit Kotlin für Android Systeme.

## 1.2 PetPal

Meine Idee war eine App, die es Haustierbesitzern ermöglicht, in der App Anzeigen (Searches) zu schalten, um andere Personen zu suchen, die Ihre Tiere für eine Abwesenheit, wie z.B. Urlaub, betreuen können.

Im Laufe der Entwicklung kam die weitere Idee, tierlieben Nutzern, mit passenden Rahmenbedingungen sowie Freude an Tieren, die Option zu geben Ihr Profil hochzustufen, auf die Rolle eines „Petsitters“/Haustiersitters. Damit sind Sie in der Lage Ihr Profil in einer separaten Liste anzeigen zulassen, die Haustierbesitzern die Möglichkeit gibt spezifisch nach sogenannten „Petsittern“ zu suchen.

Eine nice-to-have Anforderung war, Firebase für die Entwicklung zu nutzen, in meinem Projekt wurden folgende Firebase Services verwendet:

**Firebase Auth:**

Verwaltet die gesamte Authentifizierung, also die Registrierung neuer Nutzer mittels E-Mail und Passwort, den Login bereits registrierte Nutzer sowie weitere Funktionen wie z.B. eine „Passwort vergessen“ Funktion.

**Firestore Database:**

Dient als Speicher für die systemrelevanten Daten und enthält folgende collections:

* Users
* Pets
* Searches
* Petsitters
* Messages

**Firebase Storage:**

Dient als Speicher für Profilbilder der Nutzer sowie der im Profil eines Nutzers gespeicherten Bilder von dessen Haustieren.

## 1.3 Test – Accounts

**E-Mail**: user@test.de  
**Passwort**: 123456

**E-Mail**: petsitter@test.de  
**Passwort**: 123456

# Anforderungen

## 2.1 Allgemeine Anforderungen

2.1.1 Die App muss mehrere Screens haben.

2.1.2 Der Nutzer muss Daten eingeben können.

2.1.3 Daten, die eingegeben wurden, müssen gespeichert werden.

2.1.4 Die App muss mindestens einen RecyclerView beinhalten.

2.1.5 Die App kann sofern sinnvoll, offline und lokal arbeiten.

2.1.6 Es muss mindestens ein Unit Test implementiert sein.

## 2.2 Spezifische Anforderungen

2.2.1 Die App muss dem Nutzer die Möglichkeit geben sich mit seiner E-Mail und einem Passwort, für einen Account zu registrieren.

2.2.2 Die App muss dem Nutzer die Möglichkeit geben sich mit seiner E-Mail-Adresse und Passwort anzumelden.

2.2.3 Die App muss dem Nutzer die Möglichkeit geben sein Passwort zurücksetzen zu können.

2.2.4 Die App muss es Nutzern ermöglichen, neue Haustiere speichern zu können.

2.2.5 Die App muss es Nutzern ermöglichen, Haustiere mit Bild, Name und Beschreibung hinterlegen zu können.

2.2.6 Die App muss es Nutzern ermöglichen, eine Search zu erstellen.

2.2.7 Die Search muss mit einem Anfangs- und Enddatum, einem Titel, einer Beschreibung und mit den dazugehörigen Haustieren näher beschrieben werden, bevor sie erstellt werden kann.

2.2.8 Die App muss es Nutzern ermöglichen seine aktuellen Searches und Searches aus der Vergangenheit anzeigen zu lassen.

2.2.8 Die App muss es Nutzern ermöglichen, bereits angelegte Haustiere bearbeiten zu können.

2.2.9 Die App muss es Nutzern ermöglichen, Ihr Profil auf einen Petsitter hochzustufen.

2.2.10 Die App muss Nutzern ermöglichen, alle aktuellen Searches auch von anderen Nutzern sehen zu können.

2.2.11 Die App muss Nutzern ermöglichen, Petsitter sehen zu können.

2.2.12 Die App muss Nutzern ermöglichen, Searches mit Ihren Details ansehen zu können.

2.2.13 Die App muss Petsittern ermöglichen, Ihr Profil detaillierter beschreiben zu können.

2.2.14 Die App muss Nutzern ermöglichen, Profile von Petsittern mit Details zu sehen

2.2.15 Die App muss die Funktion zur Verfügung stellen Nachrichten zwischen Nutzern und Petsittern verschicken sowie wieder abrufen zu können.

2.2.16 Die App soll ein Impressum beinhalten.

2.2.17 Die App soll allen Nutzern ermöglichen Ihr Profil zu bearbeiten.

# Code

## 3.1 XML-Dateien

*~/app/res/layout/*

Die XML-Dateien im o.g. Ordner enthalten die Designs der jeweiligen Screens und sind i.d.R. nach den Namenskonvention activity\_[ACTIVITY\_NAME].xml (z.B. activity\_main.xml) benannt. Sie enthalten XML-Code, der das Design und Layout für einen Screen enthält.

Im Falle von Recyclerviews, gibt es eine XML-Datei, die das Layout beschreibt, so wie eine weitere Datei, die das list item selbst beschreibt und folgt der Namenskonvention list\_item\_[ACTIVITY\_NAME] (z.B. list\_item\_message).

Zusätzlich gibt es für den NavigationDrawer (Burger Menu) drei weitere Dateien.

~/app/res/menu/nav\_menu.xml  
beschreibt den Inhalt des NavigationDrawers, wenn er ausgeklappt ist.

nav\_header.xml  
beschreibt den Header des ausgeklappten NavigationDrawers mit dem Profilnamen und der E-Mail-Adresse.

nav\_layout.xml

## 3.2 Kotlin Activities

*~/app/java/com.mikuw.coupler/*

Viele Dateien haben Überschneidungen, die in jeder Klasse vorkommen. Diese werden hier erklärt:

Die Funktion **handleNotLoggedInUser()** überprüft, ob der Benutzer eingeloggt ist. Wenn der Benutzer nicht eingeloggt ist, wird der Hintergrund des Layouts geändert und ein Text und ein Button werden sichtbar gemacht. Der Button führt den Benutzer zur UserLoginActivity, wenn er daraufklickt.

In der **onCreate()** Methode, die aufgerufen wird, wenn die Aktivität erstellt wird, wird zuerst die super.onCreate(savedInstanceState) Methode aufgerufen, um die Erstellung der Aktivität von der darüberliegenden Klasse zu initialisieren. Danach wird das Layout für die jeweilige Aktivität mit setContentView(R.layout.activity\_[X]) festgelegt.

Für jede Aktivität wird der Titel der ActionBar gesetzt.

Für den Navigation Drawer wird ein DrawerLayout Objekt abgerufen. Ein ActionBarDrawerToggle Objekt namens "toggle" wird erstellt, um den Zustand des Navigation Drawers zu steuern. Dieser "toggle" wird dann als Listener zum DrawerLayout hinzugefügt, so dass er auf Zustandsänderungen des Drawers reagieren kann. Mit toggle.syncState() wird der Zustand des "toggle" mit dem tatsächlichen Zustand des DrawerLayout synchronisiert.

Schließlich wird die Methode setupNavigationDrawer(this) in jeder Aktivität aufgerufen, um den Navigation Drawer zu konfigurieren.

Die Methode **onOptionsItemSelected()** wird überschrieben, um auf Menüaktionen zu reagieren. Wird auf den Burger Menu Button gedrückt, wird die onOptionsItemSelected Methode des "toggle" aufgerufen. Wenn diese Methode true zurückgibt, was bedeutet, dass das Ereignis behandelt wurde, wird true zurückgegeben. Andernfalls wird die onOptionsItemSelected Methode der Oberklasse aufgerufen.

### 3.2.1 MainActivity

In der **onCreate** Methode wird ein SwitchMaterial-Element abgerufen, das als Umschalter zwischen der Anzeige von Haustiersuchanfragen und Petsittern dient. Abhängig vom Zustand des Schalters werden entweder die loadSearches oder loadPetsitters Methoden aufgerufen.

Die Methode **loadPetsitters()** ist dafür verantwortlich, die Daten der Haustiersitter aus der Firebase-Datenquelle zu laden und sie in einem RecyclerView darzustellen. Hier sind die Schritte, die in dieser Methode durchgeführt werden:

1. Es wird eine Instanz von Datasource\_Firebase\_Petsitter erstellt, die als Schnittstelle zur Firebase-Datenquelle dient.

2. Ein RecyclerView und ein PetsitterAdapter werden initialisiert. Der Adapter wird mit einer leeren Liste initialisiert.

3. Der Adapter wird dem RecyclerView zugewiesen.

4. Die Methode loadPetsitter der Datenquelle wird aufgerufen, um die Haustiersitter-Daten zu laden. Diese Methode nimmt eine Callback-Funktion entgegen, die aufgerufen wird, wenn die Daten geladen sind.

5. In der Callback-Funktion wird der Adapter des RecyclerViews mit den geladenen Daten aktualisiert und ein ItemClickListener gesetzt. Wenn ein Item angeklickt wird, wird die PetsitterDetailsActivitygestartet und die Daten des angeklickten Haustiersitters werden als Extra an die Aktivität übergeben.

6. Schließlich wird setHasFixedSize auf dem RecyclerView aufgerufen, um die Leistung zu verbessern, wenn sich die Größe des RecyclerViews nicht ändert.

Die Methode **loadSearches()** funktioniert ähnlich wie loadPetsitters(), lädt aber Suchanfragen anstelle von Haustiersitter-Daten. Sie erstellt eine Instanz von Datasource\_Firebase\_Searches, initialisiert einen RecyclerView und einen SearchesAdapter, lädt die Suchanfragen mit der loadSearches Methode der Datenquelle und aktualisiert den Adapter mit den geladenen und sortierten Daten. Ein ItemClickListener wird gesetzt, um die SearchDetailsActivity zu starten und die Daten der angeklickten Suchanfrage als Extra zu übergeben, wenn ein Item angeklickt wird.

### 3.2.2 ChangePasswordActivity

Die Funktion **changePassword()** ist dafür zuständig, das Passwort des aktuellen Benutzers zu ändern. Sie überprüft zunächst, ob das neue Passwort und die Bestätigung des neuen Passworts übereinstimmen und ob das neue Passwort mindestens 6 Zeichen lang ist. Wenn diese Bedingungen erfüllt sind, wird der Benutzer aufgefordert, sich mit seinem aktuellen Passwort zu reauthentifizieren. Nach erfolgreicher Reauthentifizierung wird das Passwort aktualisiert. Bei Erfolg wird der Benutzer zur Hauptaktivität weitergeleitet und eine Toast-Nachricht wird angezeigt, um den Benutzer über den erfolgreichen Passwortwechsel zu informieren.

### 3.2.3 ImprintActivity

Die ImprintActivity zeigt ein beispielhaftes Impressum der App an.

### 3.2.4 MessageDetailsActivity

Die Funktion **handleAnswerButton()** wird verwendet, um auf das Klicken des Antwort-Buttons zu reagieren. Wenn der Benutzer auf den Antwort-Button klickt, wird eine neue MessageWriteActivity gestartet, wobei die Benutzer-ID des Nachrichtensenders und der Titel der Nachricht als Extras an die Intent übergeben werden.

Die Funktion **displayMessage()** wird verwendet, um eine Nachricht anzuzeigen. Sie holt zunächst die Referenzen zu den TextViews, die den Absender, die Zeit und den Inhalt der Nachricht anzeigen. Dann holt sie die Daten des Absenders der Nachricht aus der Firestore-Datenbank und setzt den Text des Absender-TextViews auf den vollständigen Namen des Absenders. Sie formatiert das Zeitstempel der Nachricht in ein menschenlesbares Format und setzt den Text des Zeit-TextViews auf dieses formatierte Datum und die Uhrzeit. Schließlich setzt sie den Text des Inhalt-TextViews auf den Inhalt der Nachricht.

### 3.2.5 MessagesListActivity

Die Funktion **loadMessages()** lädt Nachrichten aus der Firebase-Datenquelle und zeigt sie in einem RecyclerView an. Die Nachrichten werden nach ihrem Zeitstempel sortiert, so dass die neuesten Nachrichten zuerst angezeigt werden. Wenn der Benutzer auf eine Nachricht klickt, wird die Nachricht als gelesen markiert und die MessageDetailsActivity wird gestartet, um die Details der Nachricht anzuzeigen.

Die Funktion **markMessageAsRead()** markiert eine Nachricht als gelesen in der Firestore-Datenbank. Sie sucht die Nachricht, die den gleichen Inhalt hat und an den aktuellen Benutzer gesendet wurde, und aktualisiert das Feld isRead dieser Nachricht auf true. Bei Erfolg wird eine Log-Nachricht ausgegeben, die angibt, dass die Nachricht erfolgreich als gelesen markiert wurde. Bei einem Fehler wird eine Log-Nachricht ausgegeben, die den Fehler beschreibt.

### 3.2.6 MessageWriteActivity

Die Funktion **sendMessage()** erstellt eine Nachricht und fügt sie zur Firestore-Datenbank hinzu. Die Nachricht enthält den Titel, den Sender, den Empfänger, den Inhalt, den Zeitstempel und den Lesezustand. Bei Erfolg wird eine Toast-Nachricht angezeigt, die besagt, dass die Nachricht erfolgreich gesendet wurde, und der Benutzer wird zur MainActivity weitergeleitet.

### 3.2.7 PetAddActivity

Die Funktion **openImagePicker()** öffnet einen Dialog, der dem Benutzer die Möglichkeit gibt, ein Foto aufzunehmen oder ein Foto aus der Galerie auszuwählen. Abhängig von der Auswahl des Benutzers wird entweder die Kamera-App oder die Galerie-App geöffnet.

Die Funktion **setImageFromPicker()** setzt das ausgewählte Bild aus der Galerie als Bild für iv\_pet\_add\_image.

Die Funktion **onActivityResult()** wird aufgerufen, wenn die Kamera-App oder die Galerie-App geschlossen wird. Sie ruft entweder setImageFromCamera oder setImageFromPicker auf, abhängig davon, welche App geschlossen wurde.

Die Funktion **setImageFromCamera()** setzt das aufgenommene Bild als Bild für iv\_pet\_add\_image.

Die Funktion **uploadImageToFirebaseStorage()** lädt das ausgewählte oder aufgenommene Bild in den Firebase Storage hoch.

Die Funktion **createPetInFirestore()** erstellt ein Haustier in der Firestore-Datenbank. Sie überprüft zuerst, ob ein Haustier mit demselben Namen bereits existiert. Wenn nicht, wird das Bild hochgeladen und das Haustier wird erstellt.

Die Funktion **checkIfPetAlreadyExists()** überprüft, ob ein Haustier mit demselben Namen bereits in der Firestore-Datenbank existiert. Sie gibt true zurück, wenn das Haustier existiert, und false, wenn es nicht existiert.

### 3.2.8 PetProfileEditActivity

Die Funktion **uploadImageToFirebaseStorage()** lädt ein Bild in den Firebase Storage hoch. Sie erstellt einen eindeutigen Namen für das Bild, indem sie den Namen des Haustiers und den aktuellen Zeitstempel verwendet. Wenn das Bild erfolgreich hochgeladen wurde, wird die Funktion updateUriInFirestore aufgerufen, um die URI des Bildes in der Firestore-Datenbank zu aktualisieren.

Die Funktion **updatePetInFirestore()** aktualisiert die Beschreibung eines Haustiers in der Firestore-Datenbank. Sie sucht zuerst nach dem Haustier in der Datenbank und aktualisiert dann die Beschreibung.

Die Funktion **updateUriInFirestore()** aktualisiert die URI eines Bildes in der Firestore-Datenbank. Sie sucht zuerst nach dem Haustier in der Datenbank und aktualisiert dann die URI des Bildes.

Die Funktion **displayImage()** lädt ein Bild von einer gegebenen URI und zeigt es in einem ImageView an. Sie verwendet die Picasso-Bibliothek, um das Bild zu laden und zu bearbeiten.

### 3.2.9 PetProfileShowActivity

Diese Klasse ruft das Bild und die Daten aus Firestore und Firebase Storage am und zeigt sie im ImageView und Textview an.

### 3.2.10 PetsittterDetailsActivity

Die Funktion **handleButtonClick()** konfiguriert einen Button, der, wenn geklickt, eine neue Nachrichtenaktivität startet, um eine Kontaktanfrage an einen Haustiersitter zu senden. Die Funktion getPetsitterDocIdByEmail wird verwendet, um die Dokument-ID des Haustiersitters zu erhalten.

Die Funktion **getPetsitterDocIdByEmail()** sucht in der Firestore-Datenbank nach einem Haustiersitter-Dokument, das eine bestimmte E-Mail-Adresse hat. Wenn ein solches Dokument gefunden wird, wird die Dokument-ID an den Callback übergeben.

### 3.2.11 PetsitterRegisterActivity

Die Funktion **createPetsitterInFirestore()** erstellt ein neues Haustiersitter-Dokument in der Firestore-Datenbank. Sie holt zuerst das Benutzerdokument des aktuellen Benutzers und extrahiert dann die benötigten Informationen, um das Haustiersitter-Dokument zu erstellen. Wenn das Haustiersitter-Dokument erfolgreich erstellt wurde, wird die Funktion setAsPetsitter aufgerufen, um das Benutzerdokument zu aktualisieren und anzugeben, dass der Benutzer nun ein Haustiersitter ist.

Die Funktion **setAsPetsitter()** aktualisiert das Benutzerdokument in der Firestore-Datenbank und setzt das Feld "isPetsitter" auf true. Dies zeigt an, dass der Benutzer ein Haustiersitter ist.

### 3.2.12 PetsListActivity

Die **PetsListsActivity()** nutzt die datasource FirebasePets um alle Tiere des Nutzers in den RecyclerView zu laden, und startet die PetProfileShowActivity wenn drauf geklickt wird.

### 3.2.13 SearchCreateActivity

Die Funktion **createSearchInFirestore()** erstellt eine neue Suche in der Firestore-Datenbank. Sie holt zuerst das Benutzerdokument des aktuellen Benutzers und extrahiert dann die benötigten Informationen, um das Suchdokument zu erstellen.

Die Funktion **showDatePicker()** zeigt einen Datumsauswahl-Dialog an. Der Benutzer kann ein Datum auswählen, das dann in einem Button angezeigt wird. Die Funktion calculateDaysBetweenDates wird verwendet, um die Anzahl der Tage zwischen zwei Daten zu berechnen.

### 3.2.14 SearchDetailsActivity

Die Funktion **handleSearchCreator()** holt den Namen und den Standort des Erstellers einer Suche und zeigt diese Informationen in der Benutzeroberfläche an.

Die Funktion **getDocId()** holt die Dokument-ID einer Suche in der Firestore-Datenbank. Sie führt eine Abfrage durch, um ein Suchdokument zu finden, das einen bestimmten Ersteller und Titel hat. Wenn ein solches Dokument gefunden wird, wird die Dokument-ID an den Callback übergeben.

Die Funktion **handleButtonClick()** konfiguriert einen Button, der, wenn geklickt, eine neue Nachrichtenaktivität startet, um eine Kontaktanfrage an den Ersteller der Suche zu senden, oder die Suche als erledigt markiert, wenn der aktuelle Benutzer der Ersteller der Suche ist.

Die Funktion **markSearchAsDone()** markiert eine Suche als erledigt in der Firestore-Datenbank. Sie führt eine Abfrage durch, um ein Suchdokument zu finden, das einen bestimmten Ersteller und Titel hat. Wenn ein solches Dokument gefunden wird, wird das Feld "isDone" des Dokuments auf true gesetzt.

Die Funktion **getCreatorImage()** holt das Profilbild des Erstellers einer Suche und zeigt es in der Benutzeroberfläche an.

Die Funktion **formattedDate()** formatiert ein Datum in einen String im Format "dd.MM.yy".

Die Funktion **calculateDaysBetweenDates()** berechnet die Anzahl der Tage zwischen zwei Daten.

### 3.2.15 SearchesListActivity

Die Funktion **setView()**konfiguriert die Sichtbarkeit eines RecyclerView und das Bild eines ImageView, basierend auf dem Klick auf ein LinearLayout. Wenn das LinearLayout angeklickt wird, wechselt die Sichtbarkeit des RecyclerView zwischen Sichtbar und Unsichtbar, und das Bild des ImageView wird entsprechend geändert.

Die Funktionen **loadMySearches()** und **loadMyPastSearches()** laden die aktuellen und vergangenen Suchen des Benutzers aus der Firestore-Datenbank und zeigen sie in einem RecyclerView an. Sie verwenden einen Adapter, um die Daten in der RecyclerView anzuzeigen, und setzen einen ItemClickListener, um eine neue Aktivität zu starten, wenn ein Element in der RecyclerView angeklickt wird.

### 3.2.16 UserLoginActivity

Dient der Authentifizierung des Benutzers. Zuerst werden die E-Mail- und Passwort-Felder und der Login-Button aus der Benutzeroberfläche geholt. Wenn der Login-Button angeklickt wird, wird die E-Mail und das Passwort aus den Textfeldern geholt. Wenn beide Felder ausgefüllt sind, wird versucht, den Benutzer mit der Firebase Authentifizierung anzumelden. Wenn die Anmeldung erfolgreich ist, wird eine Toast-Nachricht angezeigt und die Hauptaktivität gestartet. Wenn die Anmeldung fehlschlägt, wird eine Fehlermeldung angezeigt. Wenn eines der Felder leer ist, wird eine Meldung ausgegeben, dass alle Felder ausgefüllt werden müssen.

### 3.2.17 UserProfileEditActivity

Die Funktion **updateUserOnSubmit()** holt die eingegebenen Daten aus den Textfeldern der Benutzeroberfläche und aktualisiert die Benutzerdaten in der Firestore-Datenbank. Wenn der Benutzer ein Petsitter ist, wird auch das Petsitter-Dokument aktualisiert.

Die Funktion **updatePetsitter()** überprüft, ob der Benutzer ein Petsitter ist, und aktualisiert das Petsitter-Dokument in der Firestore-Datenbank, wenn dies der Fall ist.

Die Funktion **checkIfPetsitter()** überprüft, ob der Benutzer ein Petsitter ist, indem sie das isPetsitter-Feld des Benutzerdokuments in der Firestore-Datenbank abruft.

Die Funktion **updateImageUriInFirestore()** aktualisiert das imageUri-Feld des Benutzer- oder Petsitter-Dokuments in der Firestore-Datenbank.

Die Funktion **uploadToStorageAndUpdateFirestore()** lädt das Benutzerbild in den Firebase-Speicher hoch und aktualisiert das imageUri-Feld des Benutzer- und Petsitter-Dokuments in der Firestore-Datenbank.

Die Funktion **uploadImageToFirebaseStorage()** lädt das Benutzerbild in den Firebase-Speicher hoch und gibt die URL des hochgeladenen Bildes zurück.

### 3.2.18 UserProfileShowActivity

Diese Klasse ruft das Bild und die Daten aus Firestore und Firebase Storage am und zeigt sie im ImageView und Textview an.

### 3.2.19 UserRegisterActivity

Die Funktion **registerUser()** registriert einen neuen Benutzer mit den eingegebenen Daten. Sie überprüft, ob alle Felder ausgefüllt sind und ob die Passwörter übereinstimmen. Sie überprüft auch, ob die Nutzungsbedingungen akzeptiert wurden. Wenn alle Bedingungen erfüllt sind, wird ein neuer Benutzer mit der Firebase Authentication erstellt und die Benutzerdaten werden in der Firestore-Datenbank gespeichert.

Die Funktion **createUserInFirestore()** erstellt ein neues Benutzerdokument in der Firestore-Datenbank mit den eingegebenen Daten und der Benutzer-ID als Dokument-ID. Sie lädt auch das Benutzerbild in den Firebase-Speicher hoch und speichert die URL des Bildes im Benutzerdokument.

Die Funktion **uploadImageToFirebaseStorage****()** lädt das Benutzerbild in den Firebase-Speicher hoch und gibt die URL des hochgeladenen Bildes zurück.

### 3.2.20 NavigationDrawer

Die Funktion `setupNavigationDrawer` richtet den NavigationDrawer für die Aktivitäten ein. Sie fügt einen ActionBarDrawerToggle hinzu, um das Öffnen und Schließen des Menüs zu ermöglichen, und setzt OnClickListener für die Button - Schaltflächen "Home", "Register" und "Settings". Sie legt auch fest, welche Aktionen ausgeführt werden, wenn die Benutzer auf die verschiedenen Elemente im Navigationsmenü klicken.

Die Funktion `handleNotLoggedInUser` überprüft, ob ein Benutzer eingeloggt ist. Wenn kein Benutzer eingeloggt ist, wird der Text des TextViews auf "PetPal" gesetzt und die Sichtbarkeit einiger Menüelemente auf unsichtbar gesetzt. Wenn ein Benutzer eingeloggt ist, wird das Symbol der Registrierungsschaltfläche auf ein Logout-Symbol geändert.

Die Funktion `getUserNameAndEmail` holt den Namen und die E-Mail-Adresse des eingeloggten Benutzers aus der Firestore-Datenbank und setzt den Text der entsprechenden TextViews im NavigationDrawer.

## 3.3 Daten Modelle

*~/app/java/com.mikuw.coupler/model*

### 3.3.1 Message

Die Datenklasse Message besteht aus:

* title: Titel der Nachricht
* sender: Sender ID
* receiver: Empfänger ID
* content: Text der Nachricht
* timestamp: Zeitpunkt des Absendens
* isRead: Bool Wert, ob die Nachricht schon gelesen wurde

### 3.3.2 Pet

* name: Name des Haustiers
* desc: Beschreibung
* ownerId: Firebase ID des Besitzers
* imageUrl: Image Url aus Firebase Storage

### 3.3.3 Petsitter

Ein Petsitter hat folgende Attribute:

* firstname
* lastname
* email
* imageUri
* postalcode
* street
* streetNr
* city
* desc

### Search

* title
* location: Ort
* from: Startdatum
* to: Enddatum
* creator: Ersteller ID der Search
* desc: Beschreibung der Search
* pets: Liste von Haustieren in der Suche

## 3.4 Datenquellen

*~/app/java/com.mikuw.coupler/data*

Insgesamt gibt es 4 Datasource\_Firebase\_[X] – Dateien. Sie mappen die Daten die aus Firestore kommen in ein Kotlin Objekt, um diese weiterzuverarbeiten.

## 3.5 Adapter

*~/app/java/com.mikuw.coupler/adapter*

### 3.5.1 MessageAdapter

Der `MessageAdapter` ist ein RecyclerView-Adapter, der eine Liste von Nachrichtenobjekten darstellt. Jedes Nachrichtenobjekt wird in einem eigenen Listenelement dargestellt, das den Absender, den Zeitstempel und den Titel der Nachricht anzeigt. Wenn eine Nachricht noch nicht gelesen wurde, wird der Titel fett dargestellt. Der Adapter ermöglicht auch das Klicken auf ein Listenelement, um eine mehr Information über die Message zu bekommen.

### 3.5.2 PetsitterAdapter

Der `PetsitterAdapter` ist ebenfalls ein RecyclerView-Adapter, der eine Liste von Petsitter-Objekten darstellt. Jedes Petsitter-Objekt wird in einem eigenen Listenelement dargestellt, das den Namen, die Stadt und ein Bild des Petsitters anzeigt. Das Bild wird mit der Picasso-Bibliothek geladen und angepasst. Ähnlich wie der MessageAdapter ermöglicht auch der PetsitterAdapter das Klicken auf ein Listenelement.

### 3.5.3 SearchesAdapter

Der `SearchesAdapter` ist ein RecyclerView-Adapter, der eine Liste von Suchobjekten darstellt. Jedes Suchobjekt wird in einem eigenen Listenelement dargestellt, das den Titel, das Datum, die Anzahl der Tage und den Ort der Suche anzeigt. Der Adapter ermöglicht auch das Klicken auf ein Listenelement, um eine bestimmte Aktion auszuführen, die durch einen OnItemClickListener definiert wird.

### 3.5.4 SelectPetsAdapter

Der `SelectPetsAdapter` ist ebenfalls ein RecyclerView-Adapter, der eine Liste von Haustier-Objekten darstellt. Jedes Haustier-Objekt wird in einem eigenen Listenelement dargestellt, das den Namen und ein Bild des Haustiers sowie eine Checkbox zur Auswahl des Haustiers anzeigt. Der Adapter ermöglicht das Klicken auf ein Listenelement und das Auswählen oder Abwählen der Checkbox, um eine bestimmte Aktion auszuführen, die durch einen OnItemClickListener definiert wird.

### 3.5.5 ShowPetsAdapter

Der `ShowPetsAdapter` ist ein weiterer RecyclerView-Adapter, der eine Liste von Haustier-Objekten darstellt. Jedes Haustier-Objekt wird in einem eigenen Listenelement dargestellt, das den Namen und ein Bild des Haustiers anzeigt. Der Adapter ermöglicht das Klicken auf ein Listenelement, um mehr Details zu bekommen.

## 3.6 Testing

Getestet wird hier beispielhaft an der RegisterActivity, und zwar genauer die Registrierungsfunktion.

### 3.6.1 isRegistrationValid\_emptyMail

Dieser Test überprüft, ob die Methode „false“ zurückgibt, wenn das E-Mail-Feld leer ist.

### 3.6.2 isRegistrationValid\_emptyPassword

Dieser Test überprüft, ob die Methode „false“ zurückgibt, wenn Passwort Feld leer ist.

### 3.6.3 isRegistrationValid\_passwordsDoNotMatch

Dieser Test überprüft, ob die Methode „false“ zurückgibt, wenn die Passwörter nicht übereinstimmen.

### 3.6.4 isRegistrationValid\_agbNotAccepted

Dieser Test überprüft, ob die Methode „false“ zurückgibt, wenn der „AGB’s akzeptieren“ nicht gesetzt ist.

### 3.6.5 isRegistrationValid\_validData

Dieser Test überprüft, ob die Methode „true“ zurückgibt, wenn alle Felder korrekt ausgefüllt sind.