Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

HTW Berlin

Fachbereich 4 - Informatik, Kommunikation und Wirtschaft Angewandte Informatik

Wintersemester 2023/2024

Dokumentation zur Probearbeit für das Modul

“Mobile Betriebssysteme und Netzwerk”

Dozent: Prof. Dr. Thomas Schwotzer
Bearbeitet von: Ahmet Oguz
Matrikel Nummer: s0580985

Activity Tracking App

Meilenstein

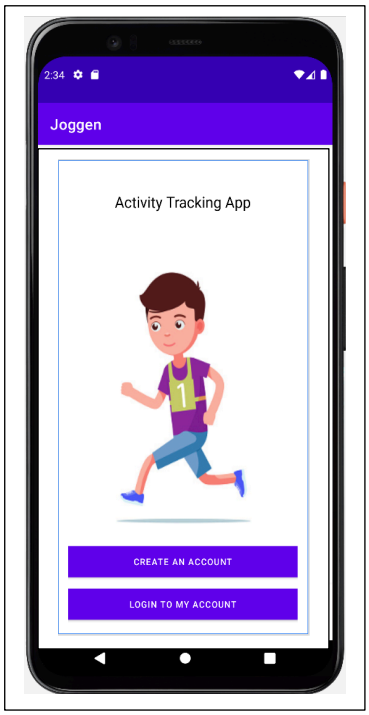
Die App-Idee ist eine Activity Tracking App, die Benutzern dabei hilft, ihre körperliche Aktivität zu verfolgen und ihre Gesundheit zu verbessern. Die App bietet eine Benutzeroberfläche für den Registrierungs- und Anmeldevorgang. Nach der Anmeldung gelangt der Benutzer zum Hauptmenü, welches eine Karte mit dem aktuellen Standort anzeigt.

Die Hauptfunktion der App besteht darin, dem Benutzer die Möglichkeit zu geben, seine Aktivitäten aufzuzeichnen, während er sich bewegt. Dies wird durch eine Play-Schaltfläche gestartet, und der Benutzer kann auf der Karte Markierungen setzen, um seinen Fortschritt zu visualisieren. Nach dem Drücken der Stopp-Taste werden Daten wie zurückgelegte Strecke, durchschnittliche Geschwindigkeit und verbrannte Kalorien berechnet und in einer Datenbank gespeichert.

Die App bietet auch eine Statistikseite, auf der der Benutzer seine Gesamtleistung in Form von wöchentlichen, monatlichen und jährlichen Gesamtstrecken in Balkendiagrammen sehen kann. Darüber hinaus werden detaillierte Informationen wie verstrichene Zeit, Distanz, verbrauchte Kalorien und durchschnittliche Geschwindigkeit in einer Liste angezeigt.

Die Integration von GPS ermöglicht eine genaue Verfolgung von Lauf-, Geh- oder Joggingrouten. Die App legt einen Schwerpunkt auf Benutzerfreundlichkeit, Datenschutz und Sicherheit. Durch eine ansprechende Benutzeroberfläche und klare Visualisierung der Aktivitätsdaten soll die App Benutzern dabei helfen, ihre Fitnessziele zu erreichen.

Im Projekt wird OSMdroid verwendet, um OpenStreetMap (OSM)-Karten in Anwendungen zu integrieren.

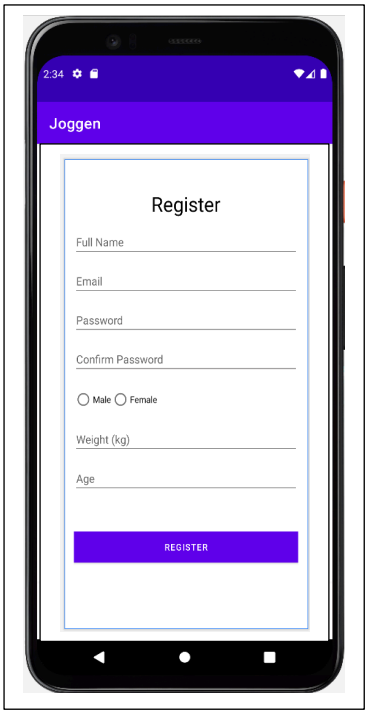


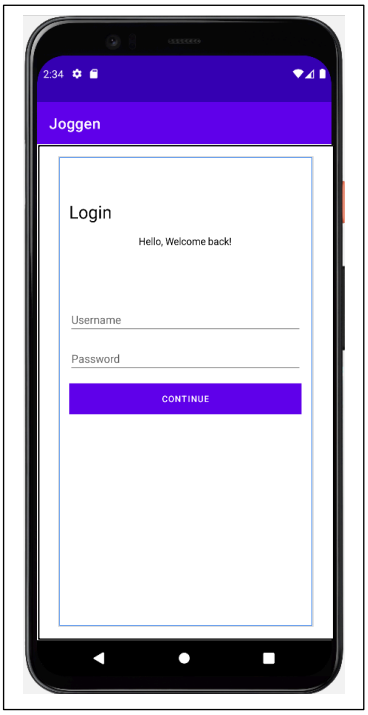
Lädt der Benutzer die „Activity Tracking App“ herunter ist diese Begrüßungsseite zu sehen. Hier kann der Benutzer zwischen zwei Schaltflächen aussuchen:

Um diese App benutzen zu können, muss sich der Benutzer vorerst registrieren.
Ist er noch nicht registriert, kann er die Schaltfläche „Create an Account“ auswählen so gelangt er auf die Seite „Registrierung“ Bei der Registrierung werden Daten wie Name, Vorname, E-mail etc. erfasst. Außerdem muss der Benutzer sein Gewicht und Geschlecht, Alter angeben, damit der Kalorienverbrauch ausgerechnet werden kann. Mit Klick auf die Schaltfläche „Register“ werden die Daten in die Datenbank gespeichert. (siehe 2. Skizze).

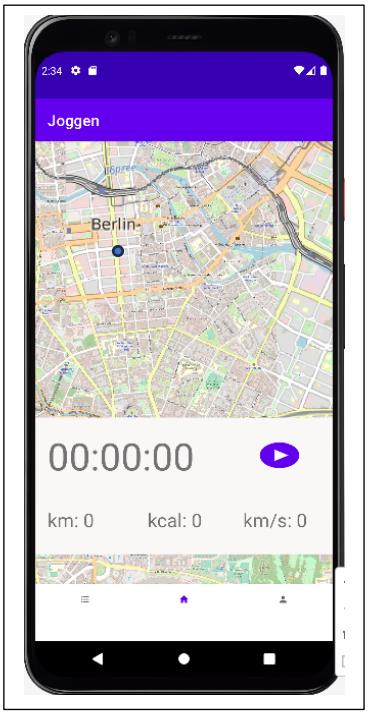
Nachdem die Registrierung abgeschlossen ist, gelangt er auf die Begrüßungsseite zurück.

Ist der Benutzer bereits registriert kann er die Schaltfläche „Login to my Account“ wählen.

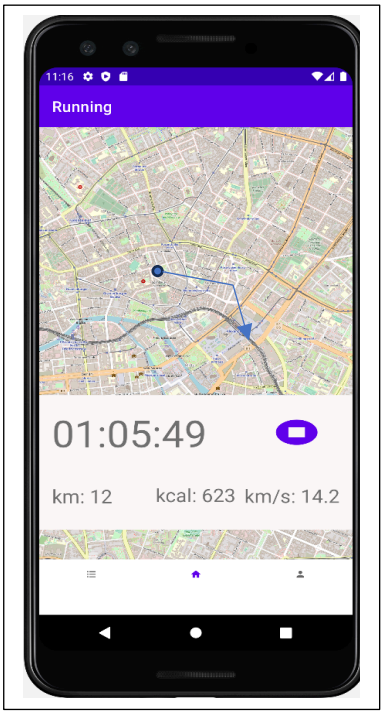




So gelangt der Benutzer auf die Seite Login und kann sich mit seiner E-Mail-Adresse und Passwort anmelden.



Nachdem sich der Benutzer eingeloggt hat, ist eine Karte zu sehen. Auf der Karte ist eine weitere Fläche, in der eine Stopuhr, ein Play-Button, die zurückgelegte Strecke in km, verbrauchte Kalorien und Geschwindigkeit pro Stunde angezeigt wird. Außerdem kann der Benutzer sehen, wo er sich gerade auf der Karte (blauer Kreis) befindet. Während einer Aktivität wird die Strecke aufgezeichnet (Start- & Endpunkt) und ist auf der Karte zu sehen (siehe nächste Skizze). In dieser Skizze ist alles auf 0, da der Benutzer noch nicht auf den Play-Button geklickt hat.
Unter der Karte gibt es ein Navigation-Bar, mit Hilfe dessen der Benutzer auf andere Seiten switchen kann.



Dieses Beispiel soll verdeutlichen: Nachdem der Benutzer auf den Play-Button klickt, beginnt die Erfassung der Zeit, Strecke, Kalorienverbrauch und Geschwindigkeit pro Stunde. Gleichzeitig wird der Startpunkt (markiert) und Endpunkt (markiert) der zurückgelegten Strecke angezeigt (blauer Linie).
Will der Benutzer die Erfassung beenden, dann kann er auf den Stop-Button klicken. Mit dem Klick werden die erfassten Werte bei der Datenbank gespeichert und der Anzeiger auf 0 zurückgestellt.

Klickt der Benutzer auf den List-Button, der sich auf dem Navigation-Bar befindet so die Seite (links) zu sehen.

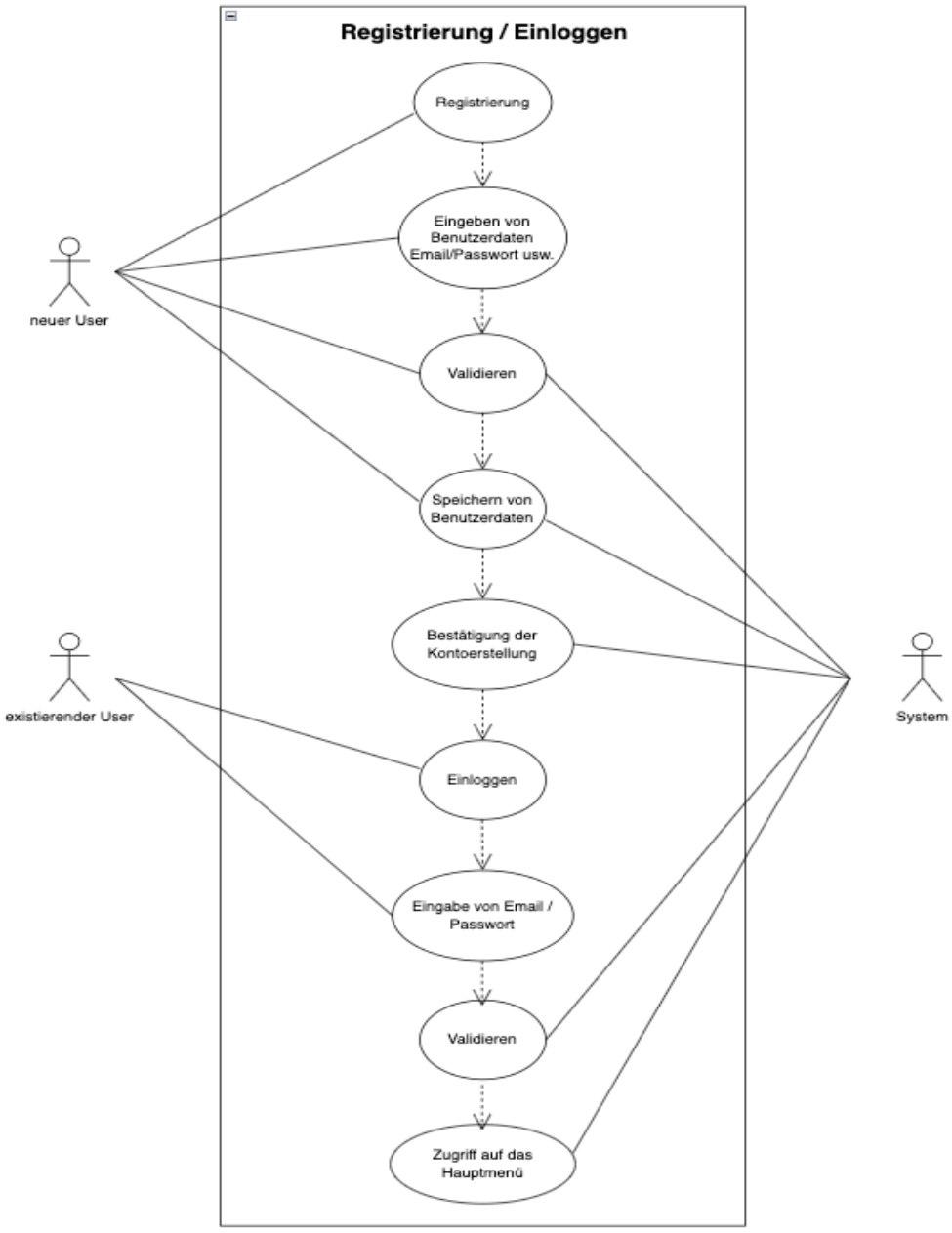
Im oberen Teil ist eine Bar-Graphik (x-Achse: Tag/Monat/Jahr; y-Achse: zurückgelegte Strecke in km) dargestellt. Hier kann der Benutzer sich mit Hilfe des Toggle-Buttons aussuchen, ob er eine wöchentliche, monatliche oder jährliche Ansicht wünscht.

Im unteren Teil befindet sich eine Liste. Hier können die, in der Datenbank gespeicherten Routen-Informationen (zurückgelegte Strecke in km, Datum, Geschwindigkeit, Kalorienverbrauch) abgerufen werden.

Klickt der Benutzer auf den Person-Button, der sich im Navigation-Bar befindet, so gelangt er auf eine Seite, auf der oberen Teil kann er den Namen der eingeloggten Person sehen.
Im Unteren Teil sind zwei Schaltflächen:

1. Personal Parameters: Hier kann er seine Daten umändern/ aktualisieren.
2. Auslogen: Hier kann sich der Benutzer ausloggen. Wenn er sich ausloggt, erscheint wieder die Begrüßungsseite.

Use Case Diagramme

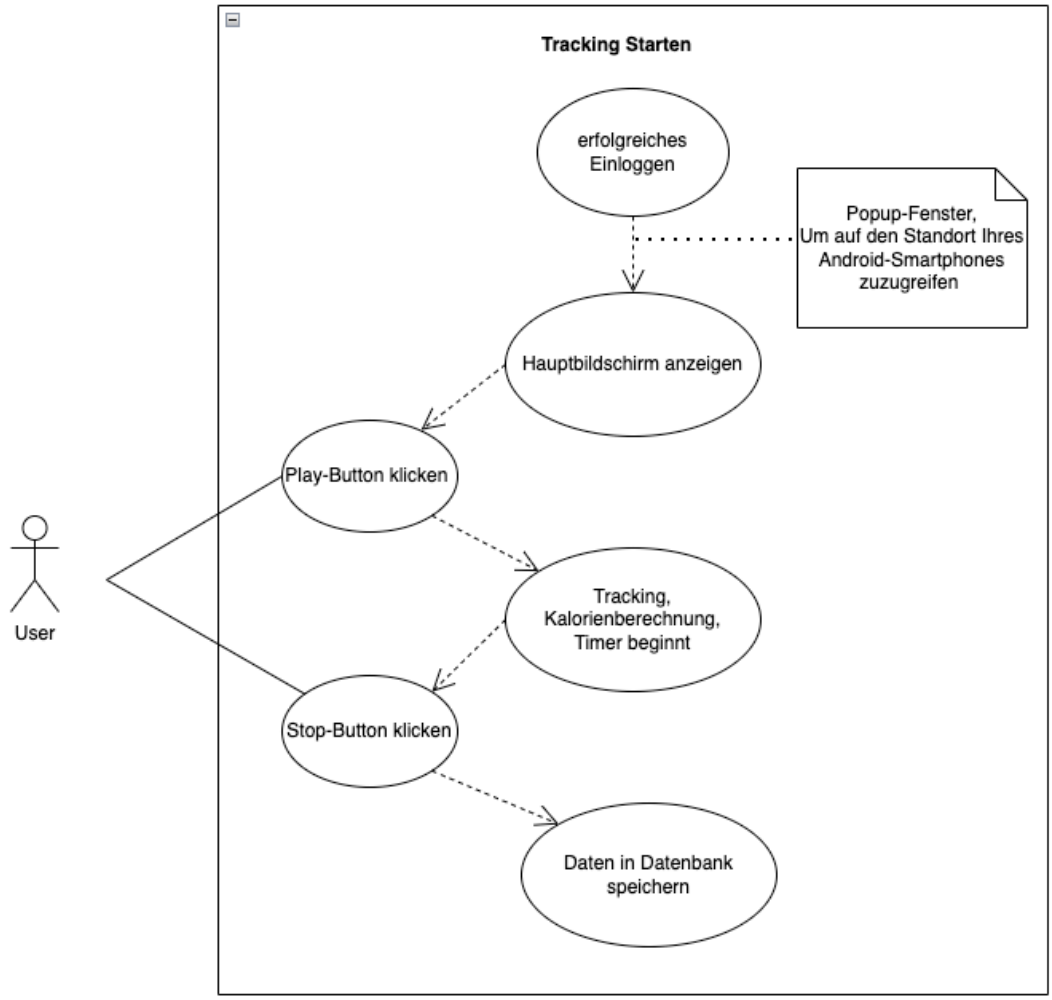


Registrierung:

Der Akteur möchte einen Account bei der Activity Tracking App anlegen:
- Hierfür müssen seine Benutzerdaten erfasst werden.
- Verifizieren der Daten
- Fehlmeldung, wenn alle Felder nicht ausgefüllt, wenn das wiederholte Passwort mit dem ersten Passwort nicht übereinstimmt, wenn der Akteur bereits existiert und wenn die Registrierung unerfolgreich ist
- Bestätigung bei erfolgreiche Accounterstellung, Weiterleitung zur Login-Seite

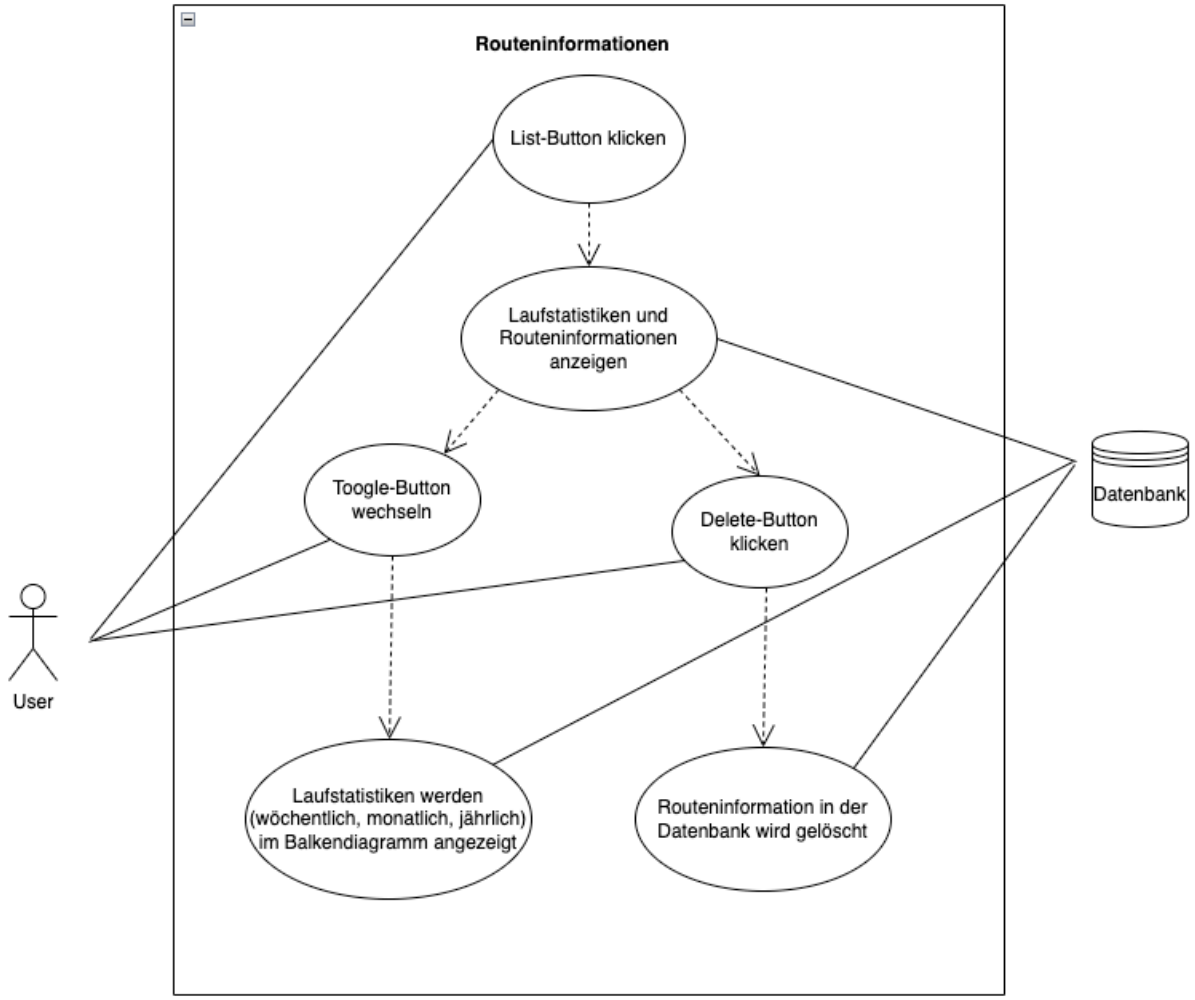
Einloggen:

Der Akteur kann sich mit seinen Benutzerdaten einloggen:
- Verifizieren der Logindaten
- Fehlmeldung, wenn die E-Mailadresse und das Passwort nicht übereinstimmen, wenn einer der Felder leer ist
- Mach erfolgreichem Login wird der Hauptbildschirm aufgerufen



Tracking Starten:

Nach erfolgreichem Login fordert die App den Akteur auf, die GPS-Koordinaten seines Standortes zu ermitteln. Wenn der Standort ermittelt wird, sieht der Akteur die Karte mit seinem Standort. Drückt er auf den Play-Button beginnt die Verfolgung, die Kalorienberechnung, der Timer und die Durchschnittsgeschwindigkeit wird berechnet. Der Akteur kann jederzeit auf den Stopp-Button drücken um die Verfolgung, Kalorienberechnung, den Timer und die Berechnung der Durchschnittsgeschwindigkeit zu beenden. Somit werden seine Daten in der Datenbank gespeichert.



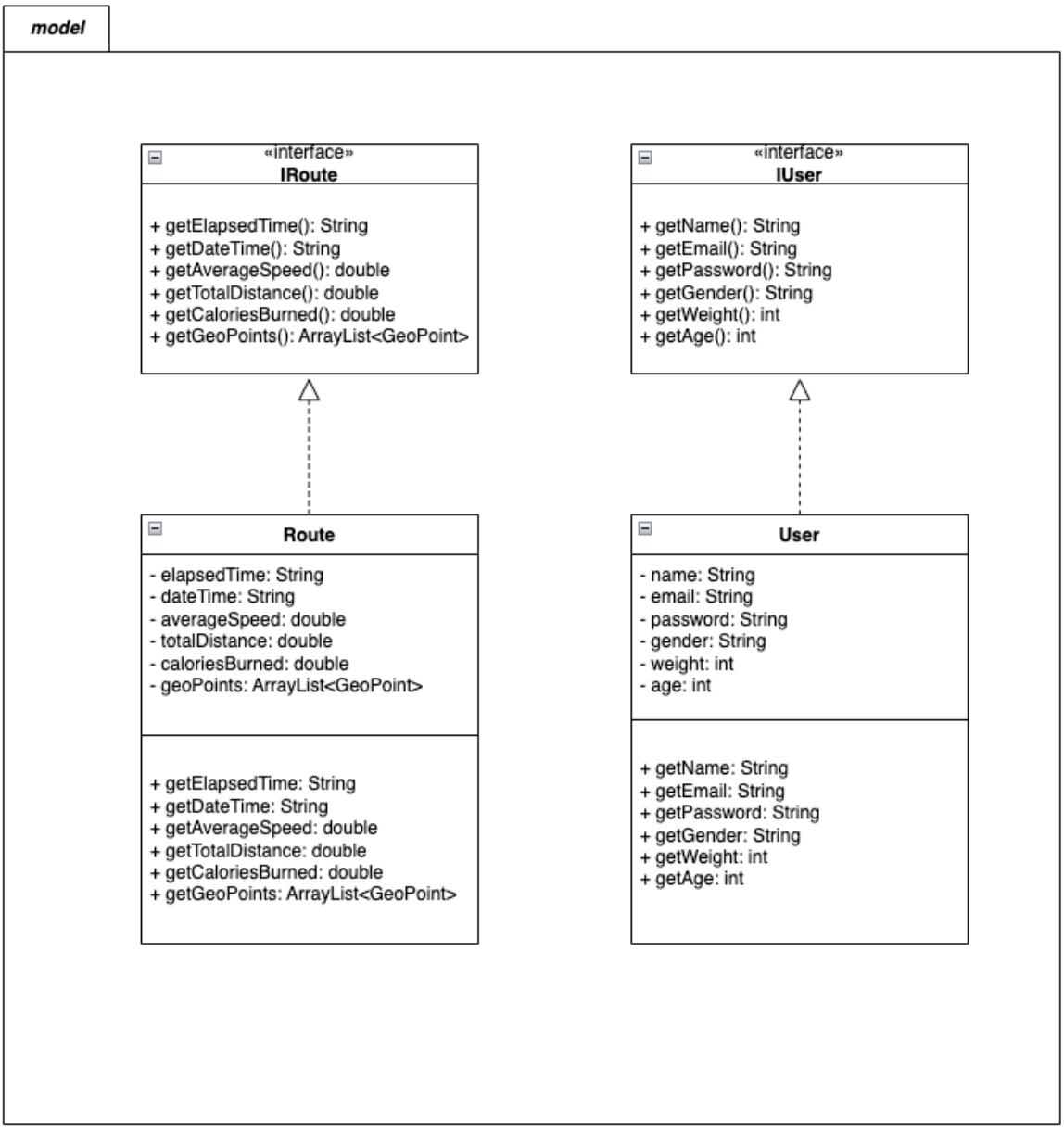
Routeninformation:

Hier hat der Akteur in zwei Formen Ansicht auf die gespeicherten Daten:
1. Als Balkendiagramm (Y-Achse: Zeit in Woche, Monat oder Jahr) X-Achse: die hinterlegte Strecke in km)
2. Als Liste (aufgelistet nach Datum der Speicherung; die hinterlegte Strecke, die Dauer und der Kalorienverbrauch sind zu sehen)

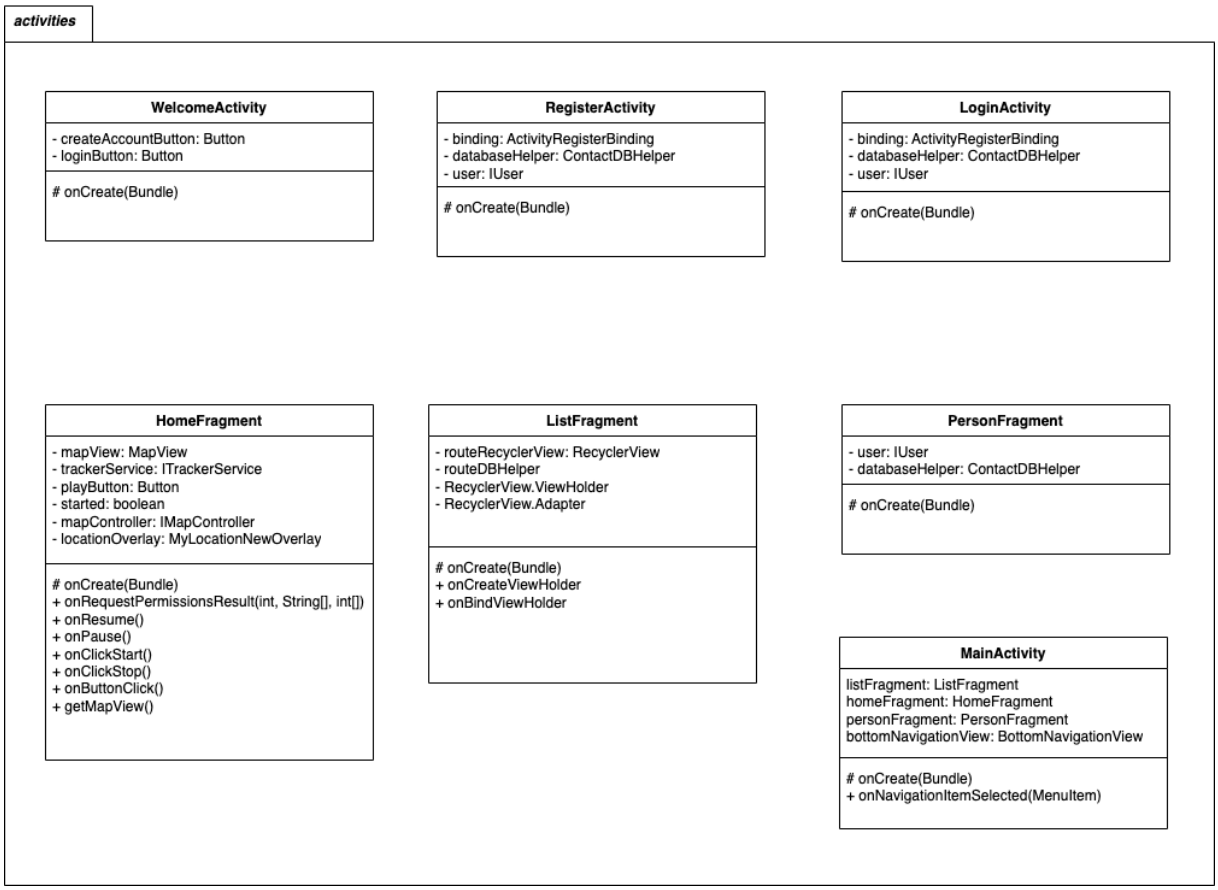
Wählt der Akteur den Toogle-Button wechselt er zwischen den Wochenansicht/Monatsansicht/ Jahresansicht und kann seine Entwicklung der Statistik entnehmen.

Klickt der Akteur auf den Delete-Button, der sich auf der Liste befindet, kann er einzelne Routeninformationen aus der Datenbank löschen.

Klassendiagramme mit einem Rahmen

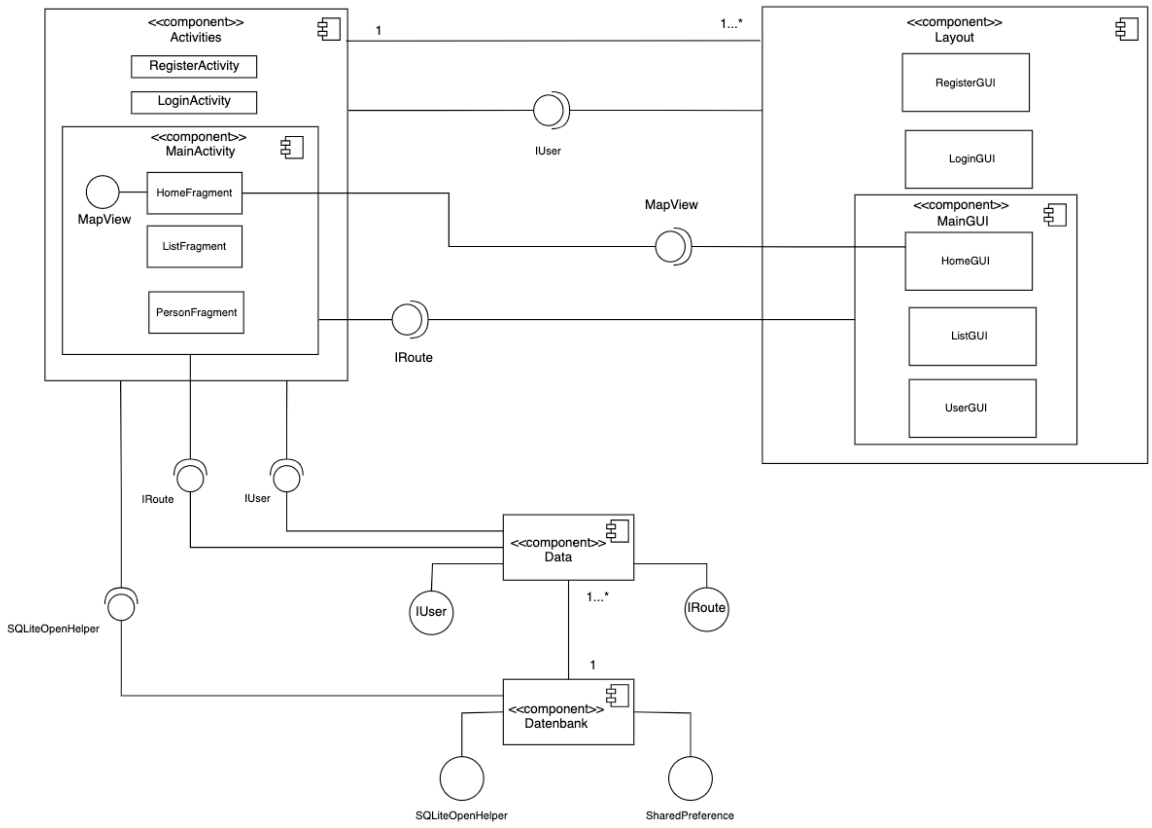


Das model-Paket enthält die beiden Hauptklassen für die Datenverwaltung in dieser Anwendung: User und Route. Beide Klassen implementieren spezifische Schnittstellen, die als externe Schnittstellen dienen.



Im "activities"-Paket werden sämtliche Aktivitäten der Anwendung abgebildet, wobei jede Aktivität einen eigenen Anzeigebildschirm repräsentiert.

Komponentenbeschreibung



Die Anwendung ist strukturiert in vier Hauptkomponenten: Data, Datenbank, Activities und Layout. Manche Hauptkomponenten enthält weitere Unterkomponenten, die im Detail untersucht werden können. Hierzu gehören die Data- und DB-Komponenten im Model, die Activities-Komponente im Controller und die Layouts in der View. Im Folgenden werden diese Komponenten detaillierter erläutert.

Model

1.1 Zusammenfassung

Das vorliegende Konzept beinhaltet die Definition von Benutzer- und Routen-Schnittstellen sowie die Implementierung konkreter Klassen (User und Route), die diese Schnittstellen implementieren.

1.2 Struktur/Technik

Die User-Klasse wurde erstellt, um Benutzerdaten darzustellen und zu verwalten.
Wenn sich ein Benutzer für die App anmeldet, muss man Informationen wie Name, E-Mail-Adresse, Passwort, Geschlecht, Gewicht und Alter speichern. Diese Daten sind wichtig, um personalisierte Funktionen anzubieten und den Kalorienverbrauch des Nutzers zu berechnen.
Die User-Klasse ermöglicht die eindeutige Identifizierung von Benutzern innerhalb der Anwendung. Es ist wichtig, Aktivitätsdaten und Statistiken einem bestimmten Benutzer zuzuordnen.

Die Route-Klasse wurde erstellt, um Informationen über die zurückgelegten Aktivitäten, insbesondere Lauf- oder Joggingrouten, zu repräsentieren und zu verwalten und ermöglicht die Erfassung von Daten während einer Aktivität, wie z. B. die zurückgelegte Strecke, die verstrichene Zeit, verbrannte Kalorien usw., die die Route repräsentieren.
Nachdem eine Aktivität beendet ist, wird die erfassten Daten in der Route-Klasse gespeichert und in der Datenbank abgelegt werden. Dies ermöglicht dem Benutzer, seine Fortschritte im Laufe der Zeit nachzuverfolgen und Statistiken über seine Aktivitäten zu generieren.
Jede Route wird eindeutig identifiziert, was es ermöglicht, Aktivitätsdaten einem bestimmten Benutzer zuzuordnen und mehrere Aktivitäten voneinander zu unterscheiden.

1.3 Schnittstelle

Diese Schnittstelle repräsentiert die Struktur einer Route in der Laufanwendung.
Sie definiert Methoden zum Abrufen verschiedener routenbezogener Informationen.

```
public interface IRoute {

  /**
   * Gibt die Gesamtdistanz der Route zurück.
   * @return Die Gesamtdistanz der Route.
   */
  double getTotalDistance();

  /**
   * Gibt die durchschnittliche Geschwindigkeit während der Aktivität zurück.
   * @return Die durchschnittliche Geschwindigkeit während der Aktivität.
   */
  double getAverageSpeed();
  /**
   * Gibt die verstrichene Zeit der Aktivität zurück.
   * @return Die verstrichene Zeit der Aktivität.
   */
  String getElapsedTime();
```

```
    /**
     * Gibt das Datum und die Uhrzeit der Aufzeichnung zurück.
     * @return Das Datum und die Uhrzeit der Aufzeichnung.
     */
    String getDateTime();

    /**
     * Gibt die verbrauchten Kalorien basierend auf der Aktivität zurück.
     * @return Die verbrauchten Kalorien basierend auf der Aktivität.
     */
    double getCaloriesBurned();

    /**
     * Gibt die Liste der GeoPoints zurück, die die Route repräsentieren.
     * @return Die Liste der GeoPoints, die die Route repräsentieren.
     */
    ArrayList<GeoPoint> getGeoPoints();
}
```

```
public interface IUser {

    /**
     * Gibt den Namen des Benutzers zurück.
     * @return Der Name des Benutzers.
     */
    String getName();

    /**
     * Gibt die E-Mail-Adresse des Benutzers zurück.
     * @return Die E-Mail-Adresse des Benutzers.
     */
    String getEmail();

    /**
     * Gibt das Passwort des Benutzers zurück.
     * @return Das Passwort des Benutzers.
     */
    String getPassword();

    /**
     * Gibt das Geschlecht des Benutzers zurück.
     * @return Das Geschlecht des Benutzers.
     */
    String getGender();

    /**
     * Gibt das Gewicht des Benutzers zurück.
     * @return Das Gewicht des Benutzers.
     */
    int getWeight();

    /**
     * Gibt das Alter des Benutzers zurück.
     * @return Das Alter des Benutzers.
     */
    int getAge();
```

1.4 Testkonzept

Die in IUser definierten Methoden werden grundsätzlich für fünf verschiedene Fälle getestet. Diese Fälle sollen die häufigsten Fehlerursachen abdecken. Das folgende Beispiel zeigt, welche Fälle für getName() getestet werden.

Testfall	**Testnamen**	**Erfolg**
Positive Test: Überprüfen, ob getName() den erwarteten Namen zurückgibt.	testGetNamePositive()	**ja**
Negative Test: Überprüfen, ob getName() case-sensitive ist.	testGetNameCaseSensitive()	**ja**
Negative Test: Überprüfen, ob getName() einen falschen Namen korrekt erkennt.	testGetNameWrongName()	**ja**
Negative Test: Überprüfen, ob getName() einen leeren Namen erkennt.	testGetNameEmptyName()	**ja**
Negative Test: Überprüfen, ob getName() einen Null-Namen erkennt.	testGetNameNullName()	**ja**

Die in IRoute definierten Methoden werden grundsätzlich für fünf verschiedene Fälle getestet. Diese Fälle sollen die häufigsten Fehlerursachen abdecken

Testfall	**Testnamen**	**Erfolg**
Überprüfen, ob alle Methoden den erwarteten Wert zurückgeben.	testGetTotalDistance() testGetAverageSpeed() …	**ja**
Überprüfen, ob getAverageSpeed() einen negativen Wert zurückgibt.	testGetAverageSpeedNegative(	**ja**
Überprüfen, ob getCaloriesBurned() einen negativen Wert zurückgibt.	testGetCaloriesBurnedNegative()	**ja**
Überprüfen, ob getTotalDistance() einen negativen Wert zurückgibt.	testGetTotalDistanceNegative()	**ja**
Überprüfen,ob alle Methoden null zurückgeben.	testGetElapsedTimeNull() testGetDateTimeNull() …	**ja**
Überprüfen, ob alle Methoden leer zurückgeben.	testGetDateTimeEmpty() testGetGeoPointsEmpty() …	**ja**
Überprüfen, ob getGeoPoints() einen Null-Wert in der GeoPoint-Liste enthält.	testGetGeoPointsNullWert()	**ja**

2. Datenbank Komponente

2.1 Zusammenfassung

Die Datenbankkomponente "ContactDbHelper" und RouteDbHelper" ermöglicht das persistente Speichern, Laden, Löschen von Benutzerdaten und Routen in der „Activitiy Tracking- App“.

2.2 Schnittstellen

ContactDbHelper und RouteDbHelper implementiert keine Schnittstellen.

2.3 Struktur und Technik

ContactDbHelper

Die Datenbank verwendet eine SQLite-Datenbank mit einer Tabelle "users", die die grundlegenden Benutzerdaten wie Name, E-Mail, Passwort, Geschlecht, Gewicht und Alter speichert. Die Struktur ist einfach und auf die Anforderungen der Benutzerverwaltung zugeschnitten.

RouteDbHelper

Die Datenbank verwendet SharedPreferences zur persistenten Speicherung von Routeninformationen. Die Routen werden als JSON-Strings gespeichert und durch Gson in Objekte umgewandelt, wenn sie benötigt werden. Jede Route wird unter einem eindeutigen Schlüssel in den SharedPreferences gespeichert, der auf das Datum und die Uhrzeit der Aufzeichnung basiert.

2.4 Testkonzept

ContactDbHelperTest

Testfall	Testnamen	Erfolg
um sicherzustellen, dass das Einfügen von Benutzerdaten und der Zugriff darauf ordnungsgemäß funktionieren.	testInsertUserData() testGetUserData()	**ja**
um sicherzustellen, dass die Komponente nicht vorhandene Benutzer angemessen behandelt.	testNonExistentUser()	**ja**
Überprüfung, ob das Abrufen von Benutzerdaten anhand der E-Mail-Adresse ordnungsgemäß erfolgt.	testGetUserByEmail()	**ja**
Tests, die sicherstellen, dass die Überprüfung von E-Mail-Adressen und Passwörtern korrekt funktioniert.	testValidEmailAndPassword() testInvalidEmailOrPassword()	**ja**
Anzeige aller in der Datenbank gespeicherten Benutzerdaten	testDisplayAllUserData()	**ja**

RouteDbHelperTest

Testfall	Testnamen	Erfolg
Überprüft, ob alle Routen aus der Datenbank auf einmal geladen werden können.	testLoadAllRoutes()	**ja**
Überprüft, ob mehrere Routen erfolgreich gespeichert und geladen werden können	testSaveAndLoadMultipleRoutes()	**ja**
Überprüft, ob eine Route erfolgreich gespeichert und anhand des Schlüssels geladen werden kann	testSaveAndLoadSingleRouteByKey()	**ja**
Überprüft, ob eine Route erfolgreich anhand des Schlüssels gelöscht werden kann	testRemoveRouteByKey()	**ja**
Überprüft, ob alle Routen erfolgreich aus der Datenbank gelöscht werden können.	testRemoveAllRoutes()	**ja**
Überprüft, ob alle Routen geladen werden können, wenn die Datenbank leer ist.	testLoadAllRoutesWhenEmpty()	**ja**
Überprüft, ob die korrekte Route gelöscht wird, wenn sich mehrere Routen in der Datenbank befinden	testRemoveCorrectRouteWithMultipleRoutes()	**ja**
Überprüft, ob man einen nicht existenten Schlüssel erfolgreich laden kann.	testLoadNonExistentKey()	**ja**

Controller

1. Activities Komponente

1.1 Welcome Activity

1.1.1 Zusammenfassung

Die Welcome Activity fungiert als Begrüßungsseite für neue Benutzer. Sie bietet Schaltflächen zum Erstellen eines Kontos und zum Einloggen in ein vorhandenes Konto.

1.1.2 Schnittstellen

Die Welcome Activity implementiert keine spezifischen Schnittstellen

1.1.3 Struktur und Technik

Die Activity verwendet das Android-Framework und erweitert die AppCompatActivity-Klasse. Sie enthält Schaltflächen für die Erstellung eines Kontos und das Einloggen. Der Code ist strukturiert, um auf Benutzerinteraktionen zu reagieren und Intents für den Übergang zu den entsprechenden Aktivitäten (RegisterActivity und LoginActivity) zu starten.

1.1.4 Testkonzept

WelcomeActivityTest enthält Espresso-Tests für die WelcomeActivity. Die Tests überprüfen die Benutzerinteraktionen und die Navigation zu anderen Aktivitäten.

Testfall	Testnamen	Erfolg
Überprüfen, ob das Klicken auf die Schaltfläche zur Kontoerstellung die RegisterActivity öffnet.	createAccountButtonClicked_opensRegisterActivity()	**ja**
Überprüfen, ob das Klicken auf die Schaltfläche zum Einloggen die LoginActivity öffnet.	loginButtonClicked_opensLoginActivity()	**ja**
Sicherstellen, dass die Activity ordnungsgemäß erstellt wird und die Benutzeroberfläche korrekt angezeigt wird	activityIsCreatedAndUIIsDisplayed()	**ja**

1.2 Register Activity

1.2.1 Zusammenfassung

Die RegisterActivity ist eine Android-Activity, die es Benutzern ermöglicht, sich in der App zu registrieren. Sie sammelt Benutzerinformationen wie Name, E-Mail, Passwort, Geschlecht, Gewicht und Alter und speichert diese Informationen in der Datenbank über die ContactDbHelper Klasse.

1.2.1 Schnittstellen

Diese Activity implementiert keine spezifischen Schnittstellen.

1.2.3 Struktur und Technik

Die Struktur der RegisterActivity ist darauf ausgelegt, die Benutzereingaben zu validieren, die eingegebenen Daten in einem User-Objekt zu speichern und diese Daten dann in die SQLite-Datenbank über die ContactDbHelper-Klasse einzufügen.

Die Activity enthält UI-Elemente wie Textfelder, Schaltflächen und Radio-Buttons, um Benutzereingaben zu sammeln.

Die Klasse ContactDbHelper wird verwendet, um Datenbankoperationen durchzuführen, wie das Überprüfen von vorhandenen E-Mails, das Einfügen neuer Benutzerdaten und das Anzeigen von Daten.

Es gibt Überprüfungen, um sicherzustellen, dass alle erforderlichen Felder ausgefüllt sind und dass das Passwort mit der Bestätigung übereinstimmt.
Toast-Nachrichten werden verwendet, um den Benutzer über den Erfolg oder das Fehlschlagen der Registrierung zu informieren.

1.2.3 Testkonzept

Toast-Nachrichten wird nicht mit Espresso getestet. Hierfür wurde eine eigene ToastMatcher.class erstellt. Die Tests funktionieren wie vorgesehen, Toast-Nachriten kann jedoch nicht korrekt getestet werden. wird versucht, zu lösen

Testfall	**Testnamen**	**Erfolg**
Überprüfen, wie die RegisterActivity, die die Anmeldung erfolgreich ist	testValidRegistration()	**ja**
Überprüfen, wie die Activity mit bereits vorhandenem User umgeht.	testExistingUserAlready()	**ja**
Überprüfen, wie die Activity mit der Passwortübereinstimmung umgeht.	testInvalidPassword()	**ja**
Überprüfen, wie die Activity, die Daten korrekt in die Datenbank einfügt.	testDatabaseIntegration()	**ja**
Überprüfen, ob Alert-Nachrichten korrekt angezeigt werden.		**ja**

1.3 Login Activity

1.3.1 Zusammenfassung

Die LoginActivity ermöglicht es Benutzern, sich in der App anzumelden. Die Activity sammelt die E-Mail und das Passwort des Benutzers, überprüft diese Daten anhand der in der SQLite-Datenbank gespeicherten Informationen und leitet den Benutzer bei erfolgreicher Anmeldung zur Hauptaktivität (MainActivity) weiter.

1.3.2 Schnittstellen

Diese Activity implementiert keine spezifischen Schnittstellen.

1.3.3 Struktur und Technik

Die Struktur der LoginActivity ist darauf ausgelegt, die eingegebenen Anmeldeinformationen zu überprüfen und den Benutzer entsprechend zu informieren. Die Activity enthält UI-Elemente wie Textfelder, Schaltflächen und Toast-Nachrichten, um Benutzereingaben zu sammeln und Feedback anzuzeigen. Die Klasse ContactDbHelper wird verwendet, um Datenbankoperationen durchzuführen, wie das Überprüfen von Anmeldeinformationen und das Abrufen von Benutzerdaten. Bei erfolgreicher Anmeldung werden Benutzerinformationen an die MainActivity übergeben, um personalisierte Inhalte anzuzeigen.

1.3.3 Testkonzept

Toast-Nachrichten wird nicht mit Espresso getestet. Hierfür wurde eine eigene ToastMatcher.class erstellt. Die Tests funktionieren wie vorgesehen, Toast-Nachriten kann jedoch nicht korrekt getestet werden. wird versucht, zu lösen

Testfall	**Testnamen**	**Erfolg**
Überprüfen, wie die LoginActivity, die die Anmeldung erfolgreich ist	testLoginWithValidCredentials()	**ja**
Überprüfen, wie die Activity auf ungültige Anmeldeinformationen reagiert.	testLoginWithInvalidCredentials()	**ja**
Überprüfen, wie die Activity auf leere E-Mail und Passwortfelder reagiert.	testLoginWithEmptyFields()	**ja**
Überprüfen, ob Alert-Nachrichten korrekt angezeigt werden.		**ja**

1.4 Main Activity

1.4.1 Zusammenfassung

Die MainActivity ist die Hauptaktivität der "Running" App und dient als Container für verschiedene Fragmente. Sie implementiert die Bottom Navigation Bar, die es dem Benutzer ermöglicht, zwischen den Hauptansichten der App zu wechseln: Home, Liste und Person. Die Aktivität empfängt auch Benutzerinformationen aus der Anmeldung (LoginActivity) und übergibt sie an die entsprechenden Fragmente**.**

1.4.2 Schnittstellen

Diese Activity implementiert keine spezifischen Schnittstellen.

1.4.3 Struktur und Technik

Die Struktur der MainActivity umfasst:
Bottom Navigation Bar: Die App verwendet eine Bottom Navigation Bar, um zwischen verschiedenen Fragmenten (Home, Liste, Person) zu navigieren.
Fragmente: Die Aktivität enthält Instanzen von HomeFragment, ListFragment und PersonFragment.
Datenübertragung zwischen Aktivitäten und Fragmenten: Benutzerinformationen, die während der Anmeldung empfangen wurden, werden an die entsprechenden Fragmente weitergegeben.

1.4.4 Testkonzept

Testfall	**Testnamen**	**Erfolg**
Testet die Bottom Navigation Bar, um sicherzustellen, dass sie korrekt zwischen den Fragmenten wechselt.	testBottomNavigation()	**ja**
Sicherstellen, dass die Benutzerinformationen erfolgreich an die HomeFragmente übergeben werden.	testDataTransmissionToHomeFragment()	**ausstehend**
Sicherstellen, dass die Benutzerinformationen erfolgreich an die PersonFragmente übergeben werden.	testDataTransmissionToPersonFragment()	**ausstehend**
Testet die Ansichten der Fragmente, um sicherzustellen, dass sie korrekt angezeigt werden.	testFragmentViews()	**ausstehend**

1.5 Home Fragment

1.5.1 Zusammenfassung

Das HomeFragment enthält eine Karte (MapView) zur Anzeige der Route, einen Timer, der die verstrichene Zeit misst, und Statistiken wie verbrannte Kalorien und zurückgelegte Entfernung und dient dazu die Position des Benutzers zu verfolgen. Zusätzlich steuert sie das Starten, Stoppen und Speichern einer Route.

1.5.2 Schnittstellen

Diese Fragment-Implementierung enthält keine spezifischen Schnittstellen.

1.5.3 Struktur und Technik

In dieser Activity wird eine Instanz einer OSMDroid MapView erstellt, die in der dazugehörigen View angezeigt wird und auf die derzeitige Position des Benutzers zentriert ist. Die Implementierung enthält Funktionen zum Starten und Stoppen des Timers, zur Aktualisierung der Kartenansicht basierend auf der aktuellen Position des Benutzers und zur Berechnung von Statistiken wie verbrannten Kalorien und zurückgelegter Entfernung.
Die App verwendet den Fused Location Provider von Google Play Services, um die Standortdaten des Benutzers abzurufen. Es gibt Funktionen zum Speichern der zurückgelegten Route in einer lokalen SQLite-Datenbank.
Die Benutzerinformationen werden aus dem Benutzerobjekt extrahiert, das von der Hauptaktivität übergeben wurde.

1.5.4 Testkonzept

Testfall	**Testnamen**	**Erfolg**
das Verhalten der App, wenn der Benutzer Standortberechtigungen verweigert.	testLocationPermissionDenied()	**ausstehend**
Sicherstellt, dass das Starten der Aktivität die UI-Elemente aktualisiert.	testPlayButtonBehavior()	**ja**
Sicherstellt, dass das Stoppen der Aktivität das Beenden der Aktivität sie zurücksetzt.	testStopButtonBehavior()	**ausstehend**
Überprüft, ob Standortaktualisierungen gestoppt werden, wenn die Aktivität beendet wird.	testStopLocationUpdates()	**ausstehend**
Sicherstellt, dass Standortaktualisierungen empfangen werden, wenn die Aktivität läuft.	testStartLocationUpdates()	**ausstehend**
Überprüft, ob er die verstrichene Zeit während der Aktivität korrekt zählt.	testTimerAccuracy()	**ausstehend**

Überprüft, ob die zurückgelegte Strecke basierend auf Standortaktualisierungen genau berechnet wird.	testDistanceCalculation()	**ausstehend**
Sicherstellt, dass das Gewicht des Benutzers, die Geschwindigkeit und die Aktivitätsdauer berücksichtigt werden.	testCaloriesBurnedCalculation()	**ja**
Überprüft, ob das Route-Objekt korrekt mit den erwarteten Informationen erstellt wird.	testCreateRouteObject()	**ja**
Testet das Speichern der Route in der Datenbank	testSaveRouteToDatabase()	**ausstehend**
Testet die Methoden onStart und onStop, um sicherzustellen, dass Standortaktualisierungen entsprechend gestartet und gestoppt werden	testFragmentLifecycle()	**ausstehend**
Überprüft, ob die Benutzerinformationen korrekt aus dem bereitgestellten JSON-String extrahiert werden.	testExtractUserInformation()	**ausstehend**
Führt End-to-End-Tests durch, um sicherzustellen, dass der gesamte Prozess, vom Start der Aktivität bis zum Speichern der Route, reibungslos funktioniert.	testEndToEndIntegration()	**austehend**

1.6 List Fragment

1.6.1 Zusammenfassung

Das ListFragment repräsentiert den Bildschirm der App, auf dem Benutzer ihre aufgezeichneten Routen sehen können. Es enthält eine Liste (ListView)), die bestimmte Elemente der Route anzeigt und ein Balkendiagramm (BarChart), das die wöchentliche, monatliche und jährliche Zusammenfassung der Aktivitäten anzeigt. Benutzer können lange auf ein Listenelement tippen, um es zu löschen.

1.6.2 Schnittstellen

Diese Fragment-Implementierung enthält keine spezifischen Schnittstellen.

1.6.3 Struktur und Technik

Zeigt eine Liste von aufgezeichneten Routen aus der lokalen SQLite-Datenbank an. Verwendet einen RouteDbHelper zur Interaktion mit der SQLite-Datenbank, um Routen zu laden und zu löschen. Benutzer können lange auf ein Listenelement tippen und es durch eine Bestätigungsdialogbox löschen.

1.6.4 Testkonzept

Testfall	Testnamen	Erfolg
Überprüfen, ob die Liste in Fragment Richtig anzeigt	testListViewIsDisplayed()	**ja**
Überprüfen, ob die Dialogbox zur Bestätigung des Löschvorgangs korrekt angezeigt wird.	testItemClickShowsAlertDialog()	**ausstehend**
Testen das Löschen von Listenelementen durch langes Tippen und Database	testRemoveItemFromListAndDatabase()	**ausstehend**
Überprüfen, ob das Balkendiagramm die korrekten Daten anzeigt.	testChartDisplay()	**ausstehend**

1.7 Person Fragment

1.6.1 Zusammenfassung

Um anzuzeigen, welcher Benutzer angemeldet ist, enthält PersonFragment ein Textfeld zur Anzeige des Benutzernamens sowie Schaltflächen, über die der Benutzer seine Daten aktualisieren und sich abmelden kann.

1.6.2 Schnittstellen

Diese Fragment-Implementierung enthält keine spezifischen Schnittstellen.

1.6.3 Struktur und Technik

SharedPreferences löscht Benutzerdaten beim Abmelden, um die Authentifizierungsinformationen zu entfernen. Intent navigiert den Benutzer zur Login-Seite nach erfolgreichem Abmelden.

1.6.4 Testkonzept

Testfall	Testnamen	Erfolg
Überprüfen, ob die personalisierte Willkommensnachricht korrekt den Benutzernamen anzeigt.	eingelogteUsernameIsDisplayed()	**ja**
Testen, das Abmelden und Sicherstellen, dass die Authentifizierungsinformationen gelöscht werden.	logoutClearsSharedPreferencesAndNavigatesToLogin()	**ausstehend**

Überprüfen, ob der Benutzer nach dem Abmelden zur Login-Seite navigiert wird.	logoutClearsSharedPreferencesAndNavigatesToLogin()	**ausstehend**