Projektbericht: Footprint Hero - CO2-Berechnungs-App

Sebastian Koch, Philipp Zimmer

August 31, 2023

1 Einleitung

Die **Footprint Hero** App ist eine mobile Anwendung, die Benutzern hilft, bewusstere Entscheidungen zu treffen, indem sie ihre CO2-Emissionen erfassen, berechnen und verfolgen können. Footprint Hero soll den Nutzern ein generelles Bewusstsein für ihren CO2-Fußabdruck geben, indem sie über eine grafische Oberfläche einen Überblick über ihre Emissionen mit verschiedenen Fahrzeugen über verschiedene Zeiträume erhalten.

2 Beschreibung der Funktionalität

- **Registrierung und Login**: Der Benutzer kann sich registrieren und ein Konto erstellen oder sich mit einem bestehenden Konto anmelden.
- **Fortbewegungsmittel**: Der Benutzer kann durch Buttons von Fortbewegungsmitteln auswählen, wie er sich fortbewegt hat, einschließlich Auto, Fahrrad, Flugzeug usw.
- **Eingabe der Fortbewegungsdauer**: Der Benutzer kann die Dauer der Fortbewegung eingeben, indem er einen Radknopf dreht, um die entsprechende Dauer in Minuten-Schritten nach oben oder unten einzustellen.
- **CO2-Berechnung**: Die App berechnet automatisch den CO2-Ausstoß basierend auf der gewählten Fortbewegungsmethode und der eingegebenen Dauer.
- **Wochentagsübersicht**: Zeigt den Co2 Ausstoß tagesweise für die aktuelle Woche an, um ihm zu zeigen, ob er seinen CO2-Ausstoÿ verringert hat oder nicht.
- **Wochenübersicht**: Die App zeigt dem Benutzer eine Wochenübersicht seines CO2-Ausstoßes an und vergleicht diese mit der Vorwoche, um ihm zu zeigen, ob er seinen CO2-Aussto verringert hat oder nicht.

- **Benachrichtigungen**: Die App sendet dem Benutzer Benachrichtigungen, wenn er sich 30 Minutenfortbewegt hat, um ihn daran zu erinnern, seine Fortbewegungen einzugebenund um ihm eine Wochenstatistik zu senden. Diese Funktionalität wird auch zu Verfügung gestellt, wenn die App im Hintergrund läuft.

3 Dokumentation des Systems

Die **Footprint Hero** App bietet eine Reihe von Funktionen, die den Benutzern dabei helfen, bewusstere Entscheidungen in Bezug auf ihre CO2-Emissionen im Verkehrssektor zu treffen. Nachfolgend werden die Funktionen im Detail erläutert:

- **Verkehrsmittel- und Dauerwahl**: Benutzer können aus einer Liste von Verkehrsmitteln auswählen, darunter Auto, Bahn, Bus, Fahrrad, Taxi und Flugzeug. Diese Auswahl erfolgt über den MainScreen, der durch das `MainViewModel` gesteuert wird. Hierbei wird die ausgewählte Option des Benutzers erfasst und an das `CO2CalculationViewModel` weitergegeben. Beispielhaft dazu der Codeauszug:

```
fun onVehicleSelected(vehicle: String) {
    selectedVehicle.value = vehicle
    co2CalculationViewModel.onVehicleSelected(vehicle)
}
```

- **CO2-Berechnung**: Die App verwendet vordefinierte Emissionswerte für verschiedene Verkehrsmittel, die im `CO2CalculationViewModel` definiert sind. Nach der Auswahl eines Verkehrsmittels und der Eingabe der Dauer berechnet die App die geschätzten CO2-Emissionen. Diese Berechnung erfolgt durch die Methode `calculateCO2` im `CO2CalculationViewModel`. Hier ist ein Beispiel, wie die CO2-Berechnung implementiert ist:

```
fun calculateCO2() {
    val co2Emission = model.transportationCO2[model.
        ↪ selectedTransportation] ?: 0f
    val calculatedCo2 = (co2Emission * model.duration) /
        ↪ 60f
    if (!calculatedCo2.isNaN()) {
        model.co2 = calculatedCo2
    }
    Log.d("CO2CalculationModel", "Calculated␣CO2:␣
        ↪ $calculatedCo2")
}
```

Quellen zu den Werten:

Vergleich der durchschnittlichen Emissionen einzelner Verkehrsmittel im Personenverkehr in Deutschland 2021

Quelle: Umweltbundesamt, TREMOD 6.42 (12/2022)

Verkehrsmittel		Treibhausgase [1]	Stickoxide	Partikel [4]	Auslastung
Pkw	g / Pkm	162	0,35	0,016	1,4 Pers./Pkw
Flugzeug, Inland		271 [2]	1,15	0,014	51 %
Eisenbahn, Fernverkehr		46 [3]	0,06	0,002	31 %
Linienbus, Fernverkehr [6]		37	0,04	0,003	42 %
sonstiger Busverkehr [5 6]		42	0,12	0,005	49 %
Eisenbahn, Nahverkehr		93	0,32	0,009	15 %
Linienbus, Nahverkehr [6]		108	0,33	0,012	14 %
Straßen-, Stadt- und U-Bahn		80	0,08	0,004	11 %

g/Pkm = Gramm pro Personenkilometer, inkl. der Emissionen aus der Bereitstellung und Umwandlung der Energieträger in Strom, Benzin, Diesel , Flüssig- und Erdgas sowie Kerosin

[1] CO_2, CH_4 und N_2O angegeben in CO_2-Äquivalenten

[2] inkl. Nicht-CO2-Effekte

[3] Die in der Tabelle ausgewiesenen Emissionsfaktoren für die Bahn basieren auf Angaben zum durchschnittlichen Strom-Mix in Deutschland. Emissionsfaktoren, die auf unternehmens- oder sektorbezogenen Strombezügen basieren (siehe z. B. „Umweltmobilcheck" der Deutschen Bahn AG), weichen daher von den in der Tabelle dargestellten Werten ab.

[4] ohne Abrieb von Reifen, Straßenbelag, Bremsen, Oberleitungen

[5] Reisebusse im Gelegenheitsverkehr wie Gruppen- und Tagesfahrten und sonstige (nicht gewerbliche) Busverkehre wie z. B. Werkverkehre bzw. Fahrservice

[6] vorläufige Werte

Für Informationen zu den Emissionen aus Infrastruktur- und Fahrzeugbereitstellung siehe UBA-Broschüre "Umweltfreundlich mobil!"

Da das Jahr 2021 noch stark pandemiegeprägt war, zeigt diese Grafik den Vergleich der Treibhausgasemissionen 2021 zum Vor-Corona-Jahr 2019:
https://www.umweltbundesamt.de/sites/default/files/medien/366/bilder/dateien/uba_emissionsgrafik_personenverkehr_2021.pdf

Figure 1: CO2 Ausstoß

- **Login**: Der LoginScreen ist der Startpunkt der App. Hier können Benutzer ihre Anmeldeinformationen eingeben, um Zugriff auf die Funktionalitäten der App zu erhalten. Der Login-Bildschirm stellt sicher, dass die persönlichen Daten und CO2-Berechnungen eines Benutzers privat und sicher bleiben.

  xxxx:

```
\textcolor{red}{ToDo}
```

- **Datenverfolgung**: Die berechneten CO2-Daten werden im `MainViewModel` gespeichert und verwaltet. Die Methode `merchList` aktualisiert die Liste der CO2-Daten und speichert sie in der Firebase-Datenbank. Diese Daten können dann auf dem MainScreen angezeigt werden, um den Benutzern einen Überblick über ihre CO2-Emissionen im Laufe der Zeit zu geben. Hier ist ein Beispiel für die Aktualisierung der CO2-Datenliste:

```
private fun merchList(co2: Float) {
    // ...
    val consumptionData = ConsumptionData(
        abbreviatedDayOfWeek, co2, getCurrentCalendarWeek
            ↪ ()
    )
    val updatedList = _co2DataList.value.co2Data.
        ↪ toMutableList()
    updatedList.add(consumptionData)
    _co2DataList.value = ConsumptionDataList(updatedList)
```

```
        writeCO2Data(_co2DataList.value)
        // ...
    }
```

- **Bewegungserkennung**: Die App erfasst Bewegungsdaten mithilfe des Beschleunigungssensors des Geräts. Dies erfolgt durch den `MotionDetectionService`. Wenn eine Bewegungsdauer von 30 Minuten erreicht wird, zeigt die App eine Benachrichtigung an, die den Benutzer dazu auffordert, die App zu verwenden. Hier ist die Methode zur Berechnung der Bewegungsdauer:

```
    private fun calculateMotionDuration(x: Float, y: Float, z
        ↪ : Float): Int {
        val acceleration = sqrt(x * x + y * y + z * z)
        val motionDurationMillis = System.currentTimeMillis()
            ↪  - motionStartTimeMillis
        return if (acceleration > 0.1f) {
            (motionDurationMillis / (1000 * 60)).toInt()
        } else {
            0
        }
    }
```

- **Datenbank**: Die `FirestoreDatabase`-Klasse behandelt die Kommunikation mit der Firebase Firestore-Datenbank. Sie bietet Methoden zum Schreiben, Lesen, Aktualisieren und Löschen von CO2-Daten.

- **Bewegungserfassung durch den Bewegungssensor**: Die erfassten Bewegungsdaten werden in Echtzeit abgefragt und auf eine Bewegungsdauer von 30 Minuten überprüft. Daraufhin wird eine entsprechende Notification einmalig ausgelöst.

- **Notification**: Die Benachrichtigung erfolgt durch die Abfrage des Bewegungssensors und wird erst nach 30 Minuten ausgelöst, wenn der Benutzer stehen bleibt. In diesem Fall startet die dreißigminütige Zählung von Neuem. Die `NotificationHelper`-Klasse vereinfacht das Erstellen und Anzeigen von Benachrichtigungen, um Benutzer zur Verwendung der App zu motivieren.
  Erzeugung:

```
    val notification = NotificationCompat.Builder(context,
        ↪ CHANNEL_ID)
            .setSmallIcon(android.R.drawable.ic_dialog_info)
            .setContentTitle("Footprint␣Hero␣calculation")
            .setContentText("You␣do␣a␣journy,␣use␣the␣Footrint
                ↪ ␣Hero␣calulation")
```

```
            .setPriority(NotificationCompat.PRIORITY_HIGH).
                ↪ setContentIntent(pendingIntent)
            .setAutoCancel(true).build()
```

Senden:

```
    val serviceIntent = Intent(context, MyForegroundService
        ↪ ::class.java)
            serviceIntent.putExtra("notification",
                ↪ notification)
            ContextCompat.startForegroundService(context,
                ↪ serviceIntent)
```

- **Observer**: Die Absorber befinden sich im MainViewModel. Sie erfassen sämtliche Veränderungen in der Liste der CO2-Werte und informieren entsprechenden Views. Hier die ensprechende Codestelle mit dem StateFlow:

```
  private val _co2DataList = MutableStateFlow(
      ↪ ConsumptionDataList(mutableListOf()))
    val co2DataList: StateFlow<ConsumptionDataList> get() =
        ↪ _co2DataList
```

- **Nebenläufigkeit**: Die Funktionalitäten für das auslesen des Bewegungssensors und das senden der Notification als push Nachricht sind als background Operation angelegt und werden nur beim beenden der App beendet. Die Nebenläufigkeit ist umgesetzt worden durch die Verwendung der Service Klasse:

```
class MyForegroundService : Service() {

    companion object {
        private const val SERVICE_NOTIFICATION_ID = 1
    }

    private var notification: Notification? = null

    override fun onStartCommand(intent: Intent?, flags: Int,
        ↪ startId: Int): Int {
        if (intent != null) {
            notification = intent.getParcelableExtra("
                ↪ notification")
            if (notification != null) {
                startForeground(SERVICE_NOTIFICATION_ID,
                    ↪ notification)
```

```
            }
        }
        return START_NOT_STICKY
    }

    override fun onBind(intent: Intent?): IBinder? {
        return null
    }

    override fun onDestroy() {
        super.onDestroy()
        stopForeground(true)
    }
}
```

3.1 Model-View-ViewModel (MVVM)

Die Footprint Hero App folgt dem MVVM-Architekturmuster, das eine klare Trennung von Daten, Benutzeroberfläche und Geschäftslogik ermöglicht.

- **(Model)**

  Im Modellbereich werden die Datenstrukturen definiert, die zur Speicherung von Informationen verwendet werden. Die `ConsumptionData`-Klasse speichert CO2-Daten wie den Wochentag, die CO2-Emissionen und die Kalenderwoche. Die `User`-Klasse enthält Informationen ToDo.

- **Sicht (View)**

  Die Benutzeroberflächenelemente und die Logik zur Anzeige von Daten werden im View-Bereich erstellt. Die Benutzerinteraktion erfolgt hier, indem sie Verkehrsmittel und Dauer auswählen. Die berechneten CO2-Daten werden angezeigt.

- **MainScreen**

  Hier können Benutzer ihre Auswahl des Verkehrsmittels und der Dauer treffen, um die CO2-Berechnungen durchzuführen. Zusätzlich bietet der MainScreen eine Übersicht über die bisherigen CO2-Emissionen für den ausgewählten Zeitraum. Die Benutzer können auch auf die Einstellungen zugreifen, um benutzerdefinierte CO2-Emissionswerte hinzuzufügen.

- **LoginScreen**

  Im LoginScreen können Benutzer ihre Anmeldeinformationen eingeben, um auf den MainScreen weitergeleitet zu werden. Der LoginScreen stellt sicher, dass die persönlichen Daten¡ eines Benutzers privat und sicher bleiben.

- **ViewModel** ToDo Die anderen ViewModels?
  Die ViewModel-Klassen `MainViewModel` und `CO2CalculationViewModel` sind für die Funktionalität der App verantwortlich. Sie verarbeiten die Auswahl von Verkehrsmitteln und Dauer, führen die CO2-Berechnung durch und aktualisieren die CO2-Datenliste.

4 Nicht umgesetze Ideen / Überlegungen

Nutzerfeedback und Verbesserungsvorschläge für zukünftige Versionen der App.

Anpassungsmöglichkeiten für den Benutzer, wie die Möglichkeit, benutzerdefinierte CO2-Emissionswerte hinzuzufügen.:

- **Anpassen von co2 Werten** Die vorhandenen CO2-Werte sollten über die GUI verändert oder gelöscht werden können. Dadurch würden die entsprechenden Daten sowohl lokal als auch in der Datenbank aktualisiert werden. Aufgrund von zeitlichen Einschränkungen konnte dies bisher jedoch noch nicht implementiert werden.
- **Gamification Elemente** Die Integration von Gamification-Elementen entspringt einer frühen Phase der Ideenfindung und soll entsprechende Anreize zur Nutzung des Systems bieten. Es ist jedoch möglich, dass aufgrund des Umfangs dieses Konzepts bisher noch nicht in das Projekt einbezogen wurde.
- **Community Elemente** Die Integration von Community-Elementen war zwar bereits in den frühen Planungsphasen der Ideenfindung vorhanden, wurde jedoch aufgrund des Umfangs bisher nicht in das Projekt aufgenommen. Diese Elemente sollten ebenfalls dazu dienen, die Nutzung des Systems durch die Benutzer zu fördern.

5 Diskussion besonderer Aspekte von Mobilität und mobiler Software

- +Verfügbarkeit des Devices, die App kann normalfall immer angewendet werden
- +Erkennung von Bewegung des Users und möglichkeit bewegungsart zu ermitteln
- +Notitikation in konstellation mit erkannter Bewegung
- -viele andere Ablenkungen, durch Device und Umwelt
- -Wenn umfangreichere Erweiterungen der App erfolgen könnte kleine Bildschirme zu herausgeforderungen werden.

6 Reflexion über den Entwicklungsprozess

- -Fokusierung der Appidee, entfall von community, gamification Funktionen
- -zeitliche Einteilung und arbeits Aufteilung war hindernis, [gründe?], [Lektion]
- **-Gruppen Probleme**
- -Anpassung des Umfangs für die Apppfertigstellung gegen Ende 3
- -Fokusierung der Entwicklung für entsprechende Meilensteine
- -Darstellung der App Komponenten in Bezug auf der Bildschirmgröße
- -Sensorik mit geschlossener App
- -Automatische Forbewegungsmittel erkennung

7 Teammatrix

Element	Version für Meilenstein	Finalisierung
Projektidee	Robin, (Sebastian)	Robin
MVVM	Sebastian	Philipp
Asynchronität	Philipp	Philipp
Background operations	Philipp	Philipp
Datenbank Anbindung	Philipp	Philipp
Sensor Anbindung	Philipp	Philipp
Notification werfen	Philipp	Philipp
Datenstruktur	Sebastian	Sebastian, Philipp
Main View	Robin	Philipp
Login View	Sebastian	Sebastian
Logik Co2 Berechnung	Robin	Philipp
Logik Cos Anzeige	Robin	Philipp

8 Zusammenfassung

Die Footprint Hero App stellt eine innovative Möglichkeit dar, Benutzern bewusstere Entscheidungen im Verkehrsbereich zu ermöglichen. Mit einer klaren Architektur, die das MVVM-Muster nutzt, bietet die App eine benutzerfreundliche Benutzeroberfläche, die einfache CO2-Berechnung und -Verfolgung ermöglicht. Die Integration von Sensordaten zur Bewegungserkennung und die Verwendung von Benachrichtigungen tragen zur Steigerung der Benutzereinbindung bei. Durch die Förderung nachhaltigerer Verkehrsgewohnheiten kann die App einen positiven Beitrag zur Umwelt leisten.

9 ToDos:

Maximum 12 pages, one extra page for team contribution matrix
Demo-Video 10 min MPEG-Video