

# Hochschule Darmstadt

## -Fachbereich Informatik-

### FAHRRARZT

Human Computer Interaction

vorgelegt von

**Patrick Chiu**

Matrikellnummer: 1118578

**Bryan Austin Suharta**

Matrikellnummer: 1123137

Lehrer: Prof Dr.Ing. Hans-Peter Wiedling

Tutorin: Eva Wacker

# ERKLÄRUNG

---

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Ich versichere, dass ich ansonsten keine Stellen der Arbeit durch einen Chatbot, eine KI oder ein anderes vergleichbares elektronisches Hilfsmittel oder Programm erstellt haben bzw. erstellen lassen.

Darmstadt, 4 Juli 2024

---

Patrick Chiu, Bryan Austin  
Suharta

# **ABSTRAKT**

---

Der Name FAHRRARZT ist eine Abkürzung von Fahrrad Arzt, denn man nutzt unsere App, um einen Reparaturtermin für sein Fahrrad zu buchen. Wir haben die Farbe blau als unsere Grundfarbe. Dies entspricht Verantwortung und Vertrauen.

In dieser Ausarbeitung dokumentieren wir Schritt für Schritt vom Problem zum Problemlösungsansatz, wie wir unseres Ziel schaffen können, und Faktoren, die unseren Entscheidungen beeinflussen können, berachte. Am Ende wird der Usability Test durchgeführt, dadurch kann die App aus Sicht der Rollen bewertet werden.

# INHALTVERZEICHNIS

---

<b>1 ANWENDUNGSFALL</b>	<b>6</b>
1.1 Rollen/Persona und deren Anforderungen . . . . .	6
1.2 Begeisterungsmerkmale . . . . .	6
1.3 Workflow Diagram . . . . .	7
Figure 1.2.1 . . . . .	7
1.4 Abstrakte Screens . . . . .	8
Figure 1.3.1 . . . . .	8
Figure 1.3.2 . . . . .	9
Figure 1.3.3 . . . . .	10
Figure 1.3.4 . . . . .	11
1.5 Konkrete Screens . . . . .	12
Figure 1 . . . . .	12
Figure 1.4.2 . . . . .	13
Figure 1.4.3 . . . . .	14
Figure 1.4.4 . . . . .	15
Figure 1.4.5 . . . . .	16
Figure 1.4.6 . . . . .	17
Figure 1.4.7 . . . . .	18
Figure 1.4.8 . . . . .	19
Figure 1.4.9 . . . . .	20
Figure 1.4.10 . . . . .	21
Figure 1.4.11 . . . . .	22
<b>2 ENTSCHEIDUNGEN</b>	<b>23</b>
2.1 Arbeitsmethodik . . . . .	23
2.2 Entscheidungen zum Backend . . . . .	23
2.2.1 Authentifizierung und Datenbanken . . . . .	23
2.2.2 Package . . . . .	23
2.3 Entscheidungen zum Frontend-Design . . . . .	24
2.3.1 Thema . . . . .	24
2.3.2 Homepage . . . . .	24
2.4 Entscheidungen zur Feature-Implementation . . . . .	24
2.4.1 Profile Page . . . . .	24
2.4.2 Login Page . . . . .	24
2.4.3 Aufträge Ansehen Page bei Techniker . . . . .	24
2.4.4 Ersatzteile Ansehen Page bei Techniker . . . . .	25
2.4.5 Einnahme Verfolgen Page bei Betreiber . . . . .	25
2.5 Änderbarkeit und Erweiterbarkeit . . . . .	25
2.5.1 User-Interface . . . . .	25
2.5.2 Klasse . . . . .	25

<b>3 Seitendokumentation</b>	<b>26</b>
3.1 welcome_screen.dart . . . . .	26
3.2 sign_in_page.dart . . . . .	26
3.3 sign_up_page.dart . . . . .	27
3.4 betreiber_home.dart . . . . .	28
3.5 kunde_home.dart . . . . .	28
3.6 techniker_home.dart . . . . .	28
3.7 betreiber_auslastung_verfolgen.dart . . . . .	28
3.8 betreiber_einnahme_verfolgen.dart . . . . .	28
3.9 kunde_auftrage_verfolgen.dart . . . . .	29
3.10 kunde_reparatur_buchen.dart . . . . .	32
3.11 techniker_auftrage_ansehen.dart . . . . .	33
<b>4 Klassedokumentation</b>	<b>35</b>
4.1 Klassen Diagram . . . . .	35
4.2 person.dart . . . . .	35
4.3 kunde.dart . . . . .	35
4.4 techniker.dart . . . . .	35
4.5 betreiber.dart . . . . .	35
4.6 booking.dart . . . . .	36
4.7 fahrrarzt.dart . . . . .	36
4.8 sparepart.dart . . . . .	38
<b>5 Usability Test</b>	<b>39</b>
5.1 Ausführung der Tests . . . . .	39
5.2 Bewertungen . . . . .	39
5.3 Reflexion . . . . .	40
<b>6 Literatur</b>	<b>41</b>

# ANWENDUNGSFALL

---

## 1.1 Rollen/Persona und deren Anforderungen

### 1. Kunde

Als Kunde möchte ich, dass

- ich eine Fahrradreparatur buchen und die Ersatzteile auswählen kann, die ich möchte.
- ich den Fortschritt der Reparatur verfolgen kann.
- ich bestätigen kann, wenn zusätzliche Ersatzteile benötigt werden, damit der Techniker die Reparatur fortsetzen kann.

### 2. Techniker

Als Techniker möchte ich, dass

- ich eine Reparaturbuchung auswählen kann, die ich durchführen möchte.
- ich die Reparatur durchführen kann.
- ich zusätzliche Ersatzteile vom Kunden anfordern kann, die ich benötige, um die Reparatur fortzusetzen.

### 3. Betreiber

Als Betreiber möchte ich, dass

- ich die Ausgaben und Einnahmen des Unternehmens verfolgen kann.
- ich den Lagerbestand an Ersatzteilen bestellen kann.

## 1.2 Begeisterungsmerkmale

- Die App verfügt über eine Mehrsprachigkeit-Funktion, die es dem Benutzer ermöglicht, die Sprache der App zwischen Deutsch und Englisch zu wählen.
- Die App verfügt über eine Anmeldeseite, die es mehreren Benutzern ermöglicht, auf die App zu benutzen, jeder mit einer unterschiedlichen Persona und Rolle.
- Die App nutzt eine Datenbank, um persistente Daten zu speichern, anstatt diese lokal zu speichern.

## WORKFLOW DIAGRAM

---

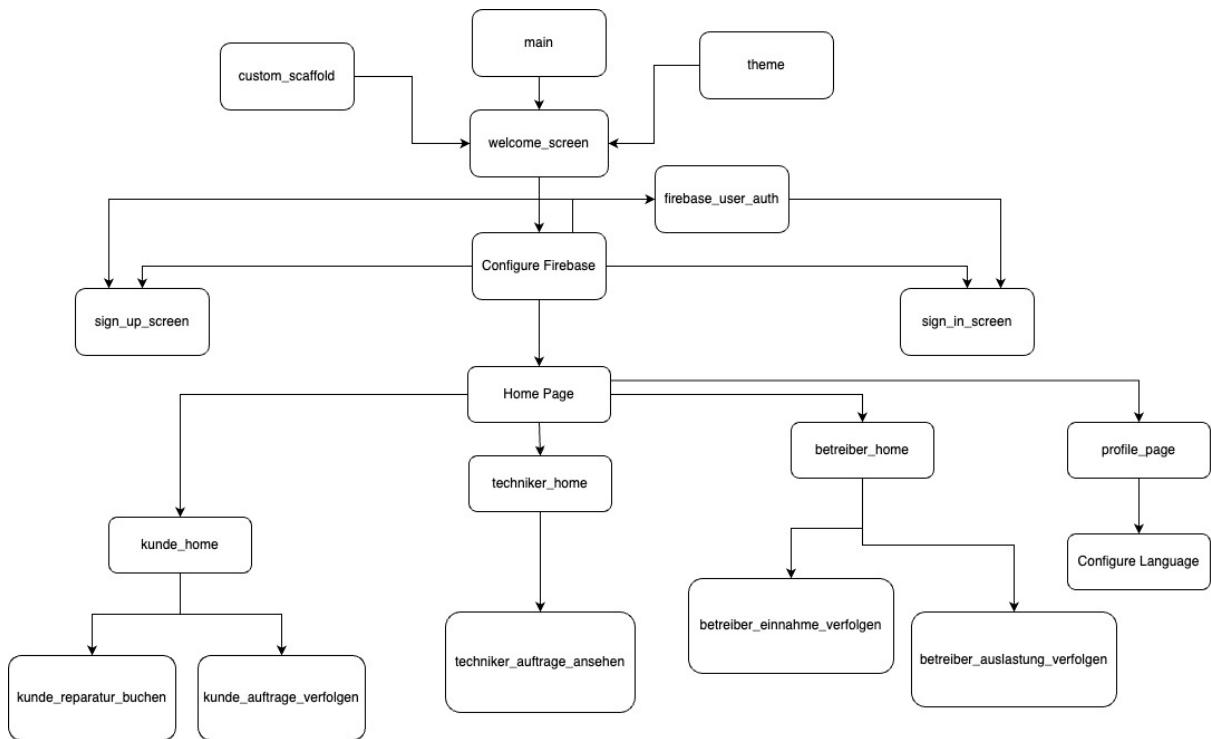


Figure 1.3.1: Workflow Diagram

## ABSTRAKTE SCREENS

---

① Login page

A hand-drawn sketch of a login page. At the top center, the word "FAHRRARZT" is written in a bold, sans-serif font. Below it is a large rectangular input field. To its left, the text "Email:" is followed by a smaller rectangular input field. Below that, the text "Password:" is followed by another small rectangular input field. At the bottom of the page, there is a horizontal button bar divided into two equal-sized rectangular buttons. The left button contains the text "Login" and the right button contains the text "Signup".

Figure 1.4.1: Login Seite

## ABSTRAKTE SCREENS

---

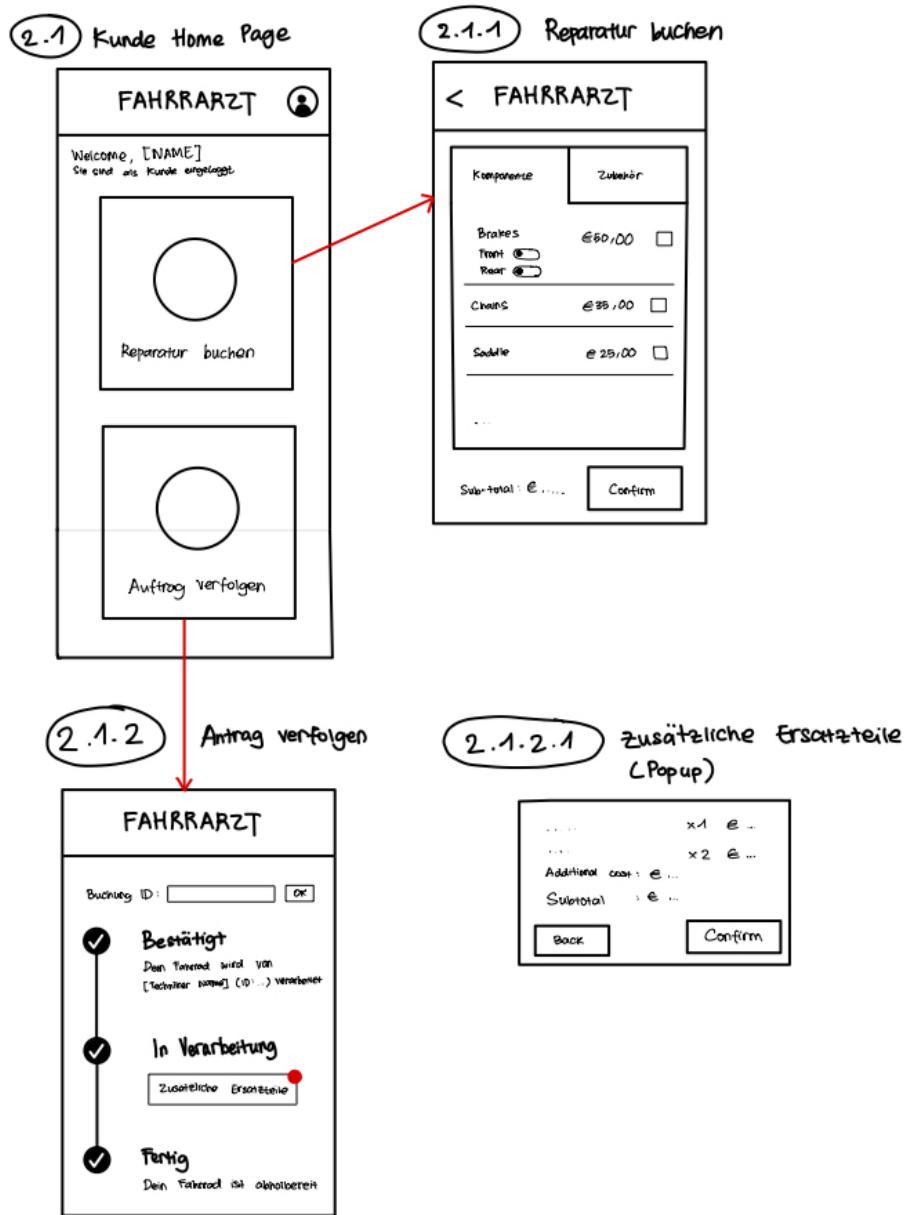


Figure 1.4.2: Kunde Seite

## ABSTRAKTE SCREENS

---

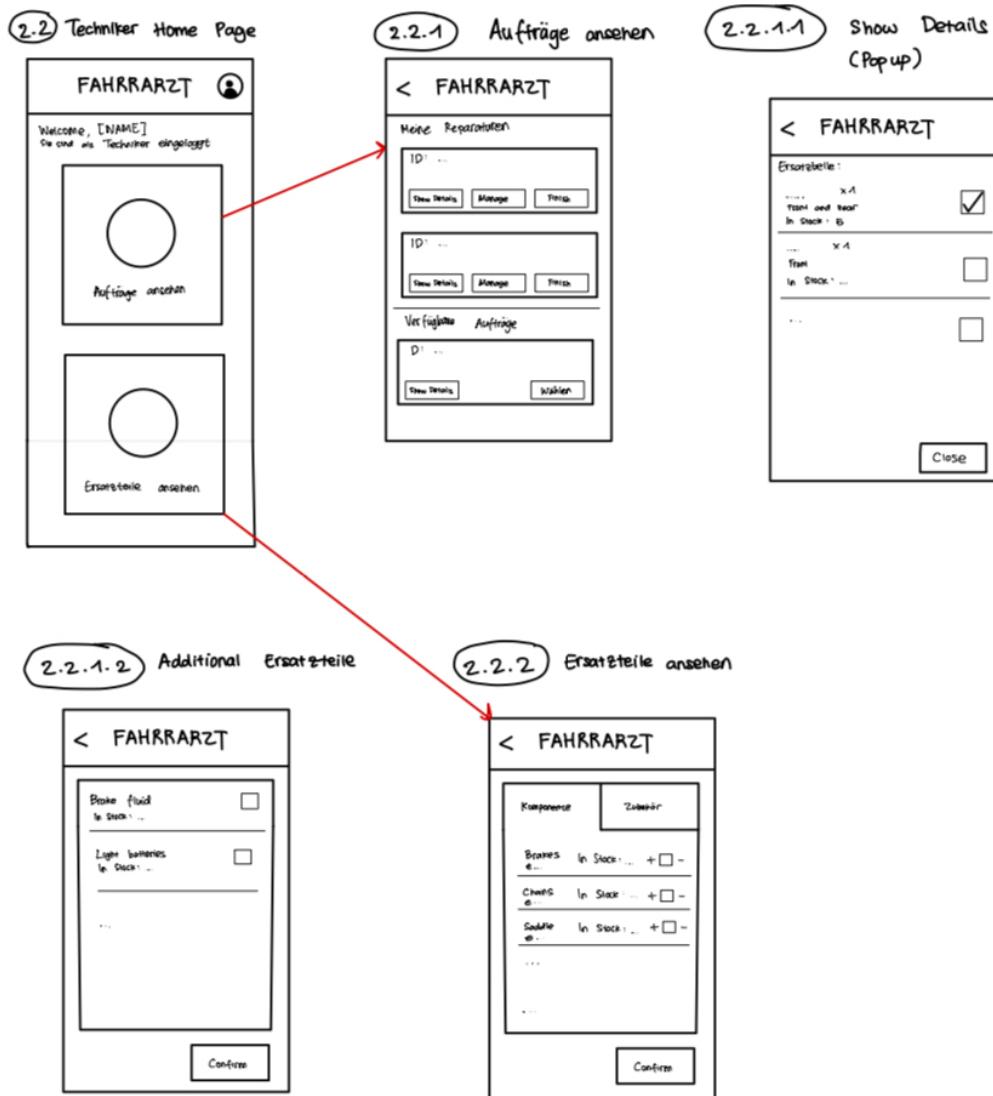


Figure 1.4.3: Techniker Seite

## ABSTRAKTE SCREENS

---

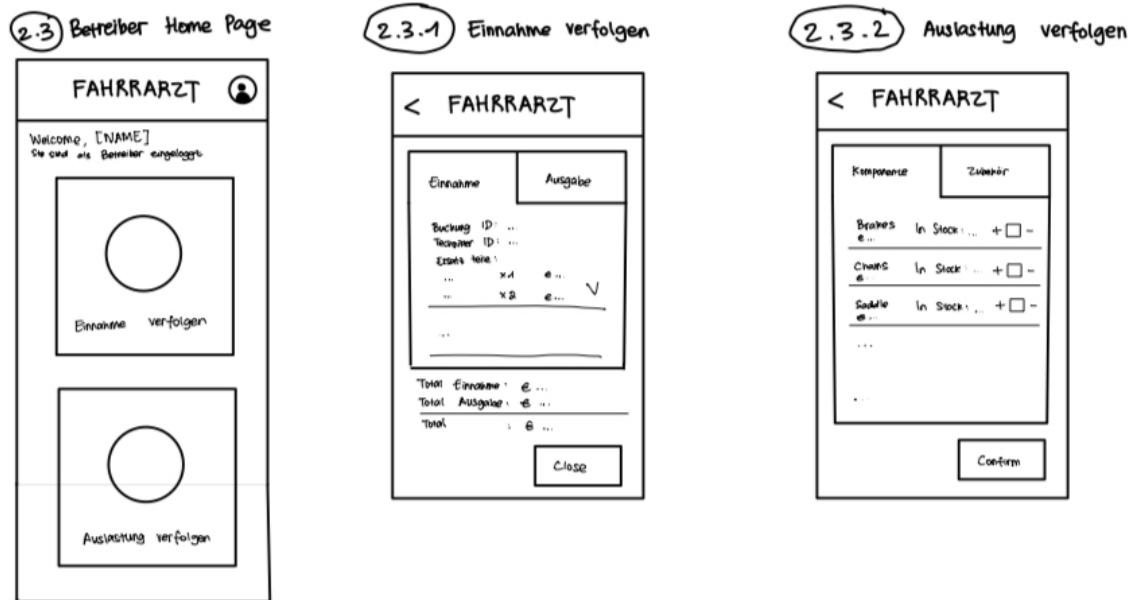


Figure 1.4.4: Betreiber Seite

## KONKRETE SCREENS

---

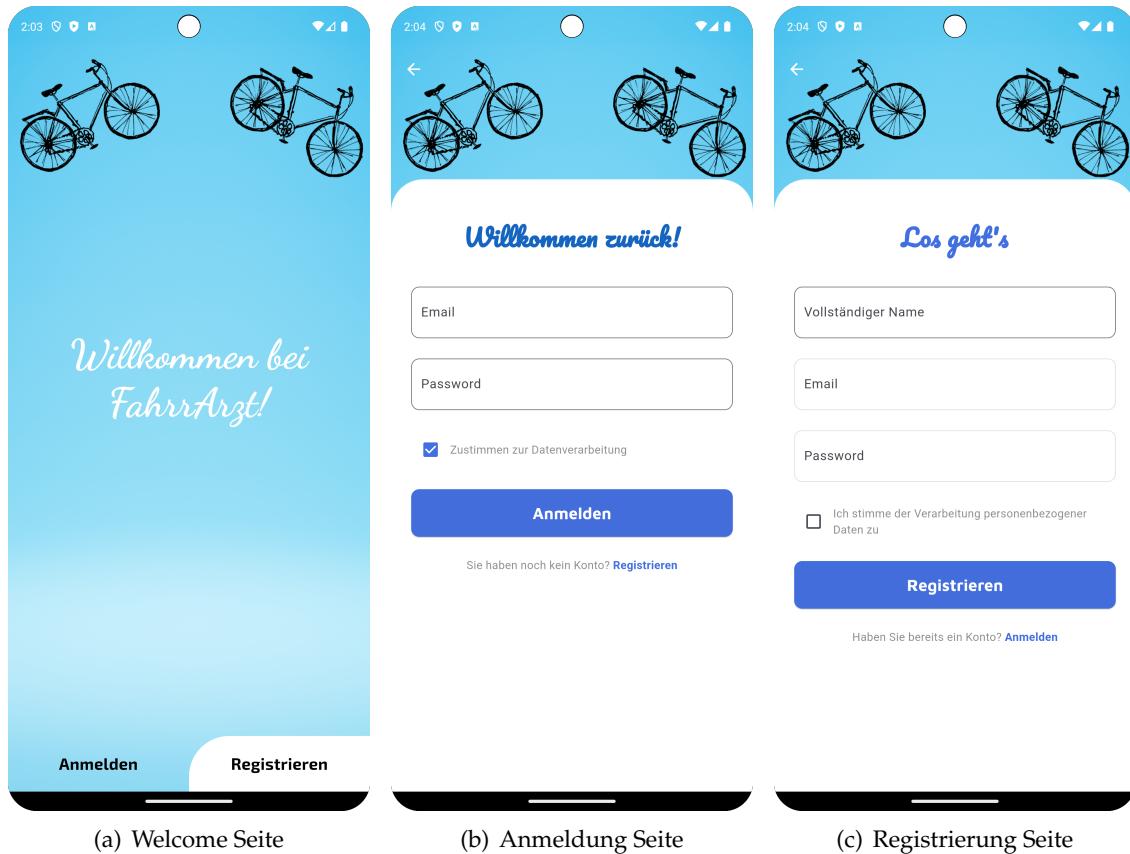


Figure 1.5.1: Login Seite

## KONKRETE SCREENS

---



Figure 1.5.2: Kunde Seite

## KONKRETE SCREENS

---

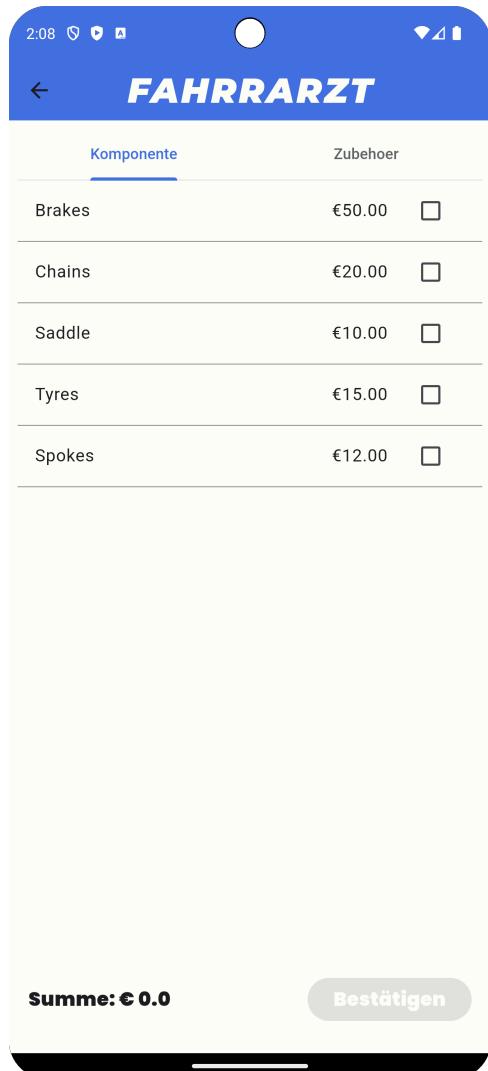


Figure 1.5.3: Reparaturbuchung

## KONKRETE SCREENS

---



Figure 1.5.4: Auftragverfolgen

## KONKRETE SCREENS

---

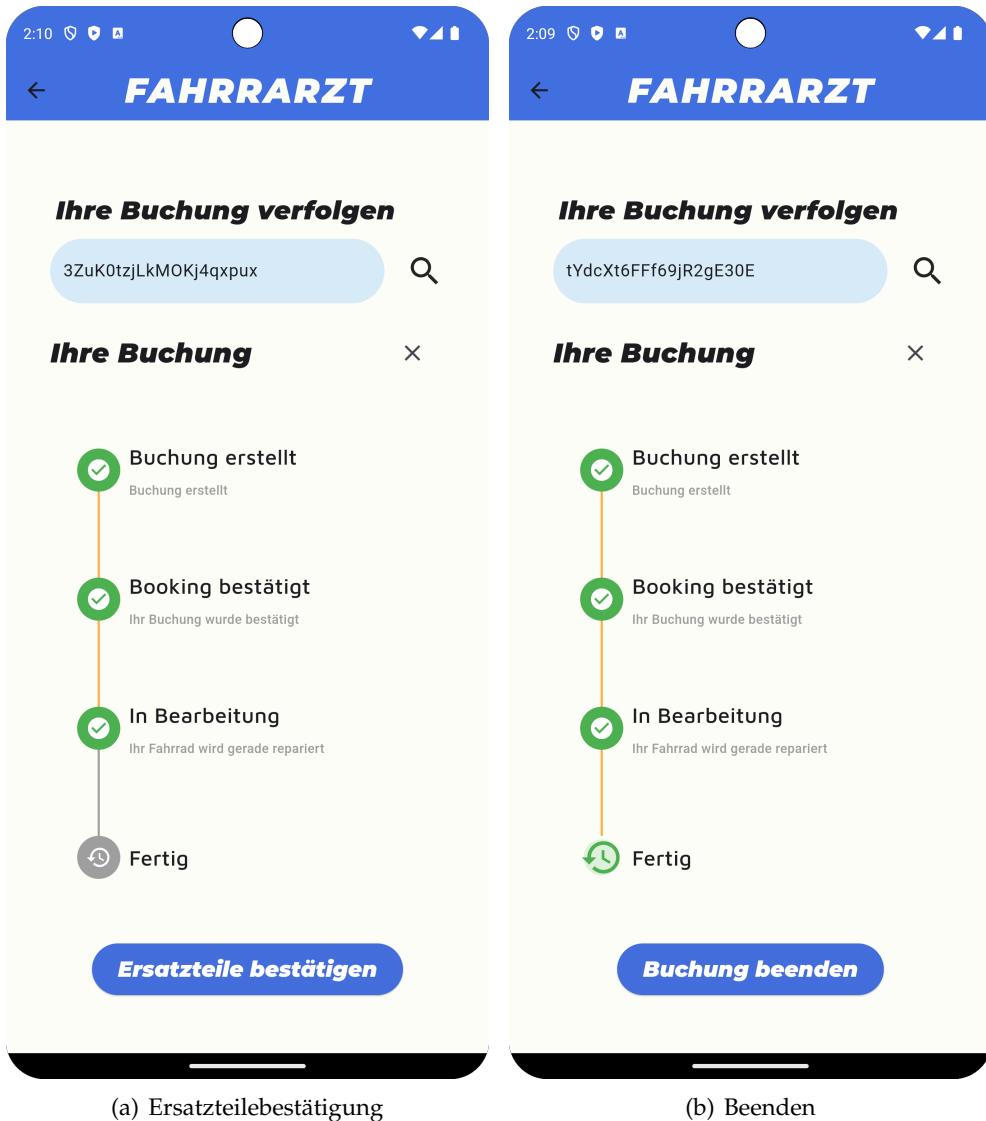


Figure 1.5.5: Bestätigen

## KONKRETE SCREENS

---



Figure 1.5.6: Technikerseite

## KONKRETE SCREENS

---

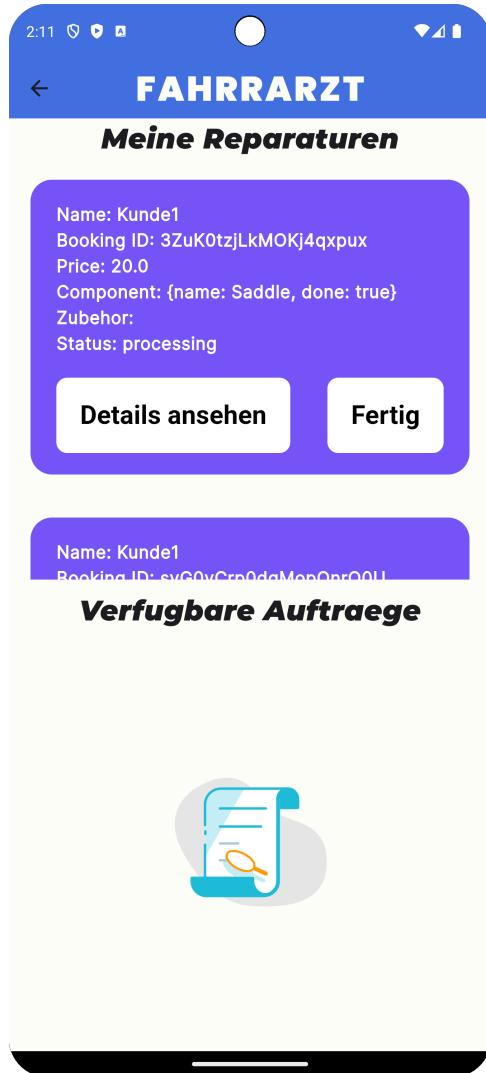


Figure 1.5.7: Auftrageansehen

## KONKRETE SCREENS

---

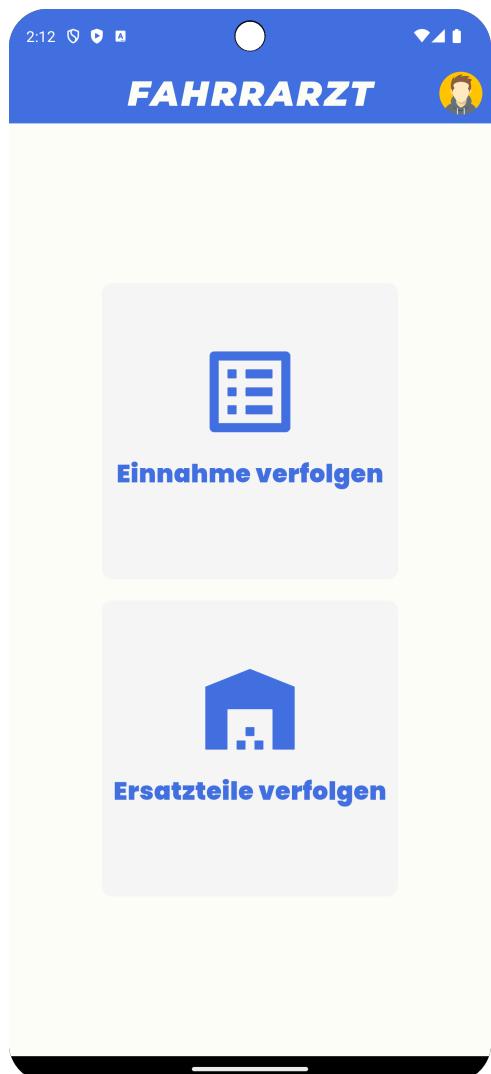


Figure 1.5.8: Betreiberstartseite

## KONKRETE SCREENS

The image displays two side-by-side screenshots of a mobile application interface. Both screens have a blue header bar with the title 'FAHRRARZT' and a back arrow icon.

**(a) Einnahme:** This screen shows three transactions under the 'Einnahme' tab. The transactions are:

Buchung ID	Preis
3ZuK0tzjLkMOKj4qpxux	€20.00
Y4SshI3kcOhZqGkSRci4	€25.00
svGOvCrp0dgMopOnrQ0U	€50.00

**(b) Ausgabe:** This screen shows various items under the 'Ausgabe' tab. The items are:

Brakes	Summe:
Anzahl: 23	€460.00
Chains	Summe:
Anzahl: 6	€60.00
Saddle	Summe:
Anzahl: 7	€35.00
Tyres	Summe:
Anzahl: 15	€105.00
Spokes	Summe:
Anzahl: 13	€65.00
Bike locks	Summe:
Anzahl: 8	€24.00
Lights	Summe:
Anzahl: 8	€16.00
Luggage racks	Summe:
Anzahl: 4	€48.00

**Summary:**

Einnahme:	€140.0
Ausgabe:	€843.0
Betrag:	€-703.0

**Buttons:** Each screen has a blue 'Close' button at the bottom right.

Figure 1.5.9: Einnahmeverfolgen

## KONKRETE SCREENS

---

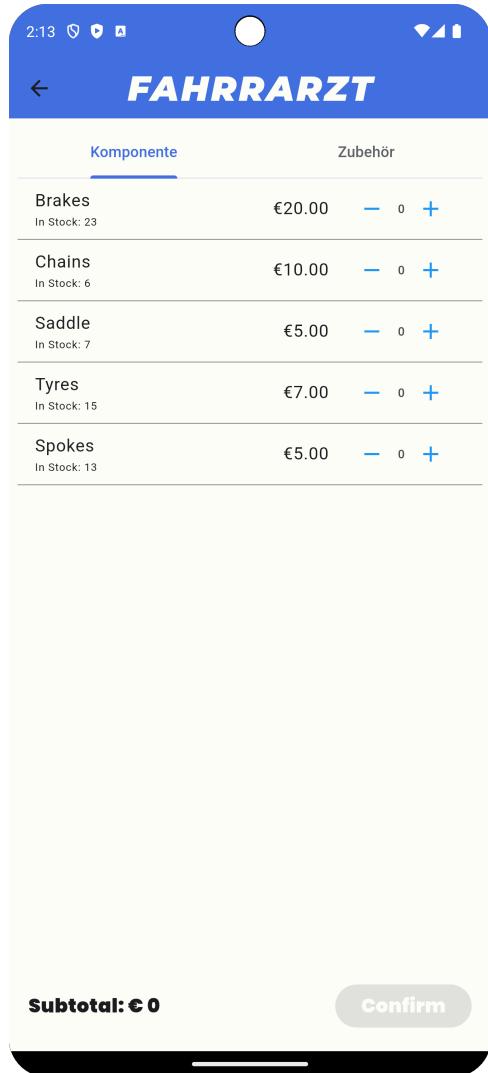


Figure 1.5.10: Ersatzteil

## KONKRETE SCREENS

---

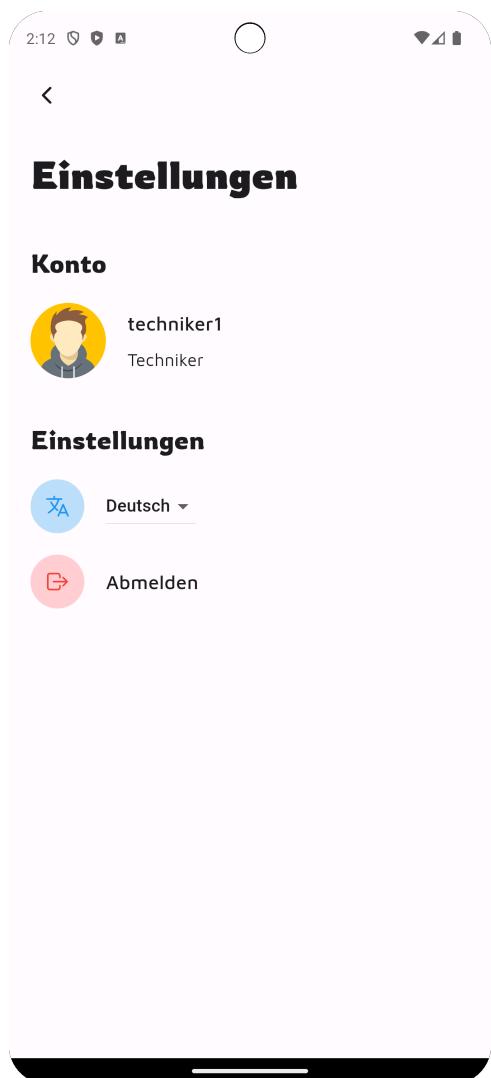


Figure 1.5.11: Einstellungen

# ENTSCHEIDUNGEN

---

## 2.1 Arbeitsmethodik

Um dieses Projekt zu beginnen, starten wir natürlich mit dem Design, das viele Diskussionen zwischen den Teammitgliedern und auch während des Praktikumstermins beinhaltet. Wir besprechen, wie die Benutzeroberfläche aussehen wird, wie wir das Backend der App implementieren, welche Farben wir verwenden sollten, welche Funktionen wir implementieren sollen und viele andere Dinge.

Wenn wir bereits eine Vorstellung davon haben, wie wir die App implementieren werden, haben wir in der ersten Woche des Projekts die Startseite der Benutzeroberfläche und die Anmeldeseite erstellt, die die Grundlagen der App darstellen. Damit die Anmeldeseite funktioniert, haben wir auch das Backend der Anmeldeseite eingerichtet. Dafür nutzen wir Firebase Authentication, das den Authentifizierungsprozess der Anmeldeseite verwaltet, sowie die Firestore-Datenbank zur Speicherung der Benutzerinformationen, einschließlich Rolle, Benutzername und anderer Informationen.

Nachdem wir die Grundlage der App abgeschlossen haben, konzentrieren wir uns auf die Implementierung der Funktionen. Wir begannen damit, das Frontend der Funktionen für jede Rolle zu erstellen, und als wir etwa halb mit der Benutzeroberfläche der Funktionen fertig waren, implementierten wir den Backend-Prozess. Dieser Teil des Projekts dauert etwa 2 Wochen, um abzuschließen.

Nachdem die Hauptteile der App fertiggestellt sind, fingen wir an, weitere zusätzliche Funktionen hinzuzufügen, wie z.B. eine Profilseite, auf der der Benutzer seine Identitätsinformationen sehen kann, sowie eine Option zur Sprachänderung. Abschließend haben wir das Projekt durch Refactoring fertiggestellt, die Funktionalität durch Benutzertests sichergestellt und bei Bedarf die Benutzeroberfläche aufgeräumt.

## 2.2 Entscheidungen zum Backend

### 2.2.1 Authentifizierung und Datenbanken

Wir haben uns entschieden, Firebase als Authentifizierung für die Anmelde- und Registrierungsseite zu verwenden, weil wir glauben, dass es realistischer ist und wegen drei Rollen wird es einfacher unterscheidet. Wir verwenden es auch, um alle Buchungen und den Gesamtbestand jedes Teils zu speichern.

### 2.2.2 Package

Wir benutzen 13 *Dependencies*, nämlich `firebase_core`, `firebase_auth`, `firebase_storage`, `cloud_firestore`, `fluttertoast`, `lineAwesome_flutter`, `ionicons`, `logger`, `flutter_localization`, `lottie`, `cupertino_icons`, `another_stepper`, and `flutter_spinkit`.

## 2.3 Entscheidungen zum Frontend-Design

### 2.3.1 Thema

Wir haben beschlossen, dass wir nur ein helles Thema verwenden. Als Primärfarbe verwenden wir Blau (0xFF416FDF), da es Verantwortung und Vertrauen repräsentiert. Da wir uns für ein helles Thema entschieden haben, verwenden wir Weiß (0xFFFFCFDF6) als Hintergrundfarbe und ein grauweißes (0xF5F5F5F5) als sekundäre Farbe, die wir zum Beispiel für den Button verwenden. Alle Farben sammeln wir in der Datei theme.dart.

### 2.3.2 Homepage

Wegen drei Rollen haben wir auch drei *Homepage* nämlich Kundeseite, Technikerseite, und Betreiberseite. Jede Seite haben unterschiedliche Aktivitäten.

In Kundeseite kann man ihre Reparatur buchen und ihres Auftrag verfolgen. In Technikerseite kann man die Buchungen von Kunden auswählen und man kann auch Zubehör hinzufügen und das Zustand von der Buchung einstellen. In Betreiberseite kann man sein Einnahme und die Ersatzteile verfolgen.

Wir benutzt kein BottomNavigationBar, weil wir glauben, dass wir viele Seite nicht benutzen und alle Funktionalität finden in einer Seite statt.

Als Alternative verwenden wir schwebende Buttons in der Mitte der Seite, und wir haben sie groß genug gemacht, damit sie der Mittelpunkt der App sind. Oben verwenden wir eine AppBar, um den Namen des Unternehmens sowie einen Profil-Button zu platzieren.

## 2.4 Entscheidungen zur Feature-Implementation

### 2.4.1 Profile Page

Die Kontoseite findet in rechte obere Ecke statt. Drin kann man ihre Rolle und ihre Name sehen. Drin gibt es auch Einstellung, die als Sprachenwechsel (English oder Deutsch) und Abmeldung benutzt wird.

### 2.4.2 Login Page

Bei der Planung des Designs haben wir uns entschieden, eine Login-Seite zu implementieren, anstatt drei Buttons, die jede Rolle repräsentieren. Obwohl dies aufwändiger zu implementieren ist als die Verwendung von drei Buttons, dachten wir, dass dies viel realistischer ist und es uns auch ermöglicht, mehr als einen Benutzer pro Rolle zu haben.

### 2.4.3 Aufträge Ansehen Page bei Techniker

Für die "Aufträge ansehen" Funktion bei Technikern haben wir zwei verschiedene Implementierungen entworfen:

1. Es gibt Listen der verfügbaren Buchungen und es gibt eine Option, eine Reparatur auszuwählen, die ein Techniker bearbeiten möchte. Darüber hinaus gibt es auch eine Liste, die die vom Techniker ausgewählten Reparaturen enthält.
2. Jede Reparatur wird automatisch dem Techniker mit der geringsten Anzahl an laufenden Reparaturen zugewiesen. Diese Seite zeigt nur die laufenden Reparaturen, nicht alle.

Wir haben uns entschieden, das erste Design zu wählen, da es den Technikern mehr Flexibilität bei der Auswahl der zu erledigenden Arbeiten bietet.

#### **2.4.4 Ersatzteile Ansehen Page bei Techniker**

Zunächst hatten wir geplant, die Funktion "Ersatzteile ansehen" sowohl für Techniker als auch für Betreiber zu implementieren. Nach Beratungen haben wir jedoch beschlossen, diese Funktion für Techniker zu entfernen d.h. nur Betreiber haben die Möglichkeit, Ersatzteile zu kaufen.

Der Grund dafür ist, dass die einzige Aufgabe eines Technikers die Reparatur der Fahrräder ist, und der Kauf der Ersatzteile nur die Aufgabe des Betreibers sein sollte.

#### **2.4.5 Einnahme Verfolgen Page bei Betreiber**

Für die Anzeige der Einnahme hatten wir zunächst geplant, alle Details der Reparatur auf einmal aufzulisten. Nach gründlicher Überlegung dachten wir jedoch, dass dies die Benutzeroberfläche überladen und mit zu vielen Informationen füllen würde. Daher haben wir uns entschieden, einen Dropdown-Button zu implementieren, der die Buchungs-ID und das Einkommen der entsprechenden Buchung anzeigt. Wenn Sie den Dropdown-Button drücken, werden Details wie Kunden-ID, Techniker-ID, verwendete Ersatzteile und der Status der Bestellung angezeigt.

Nach Beratungen haben wir uns auch entschieden, alle Buchungen unabhängig vom Status anzuzeigen, sodass auch Reparaturen, die noch im Gange sind, angezeigt werden.

### **2.5 Änderbarkeit und Erweiterbarkeit**

#### **2.5.1 User-Interface**

Um die Erweiterbarkeit der App zu erreichen, haben wir explizit eine Datei für jedes Widget oder jede "Seite" erstellt. Auf diese Weise können wir die Seite oder das Widget einfach in eine andere Datei kopieren und anpassen, wenn wir sie wiederverwenden möchten. Zum Beispiel haben wir die Startseite so gestaltet, dass sie einheitlich aussieht, da sie das gleiche Basis-UI-Format verwendet. Das ist viel einfacher, als drei verschiedene Startseiten zu erstellen. Dies haben wir auch für die Stepper-Widgets—ein grafisches Steuerelement, das Benutzer schrittweise durch einen mehrstufigen Prozess führt—implementiert, wie zum Beispiel für "Kunde Reparatur Buchen" mit "Betreiber Ersatzteile Verfolgen".

#### **2.5.2 Klasse**

Um die Benutzerklassen (Person-Klasse) zu erstellen, haben wir Polymorphie und Vererbung verwendet, um die drei verschiedenen Rollen Betreiber, Techniker und Kunde zu erstellen. Sie alle haben ähnliche Attribute wie Name und Benutzer-ID. Durch die Verwendung von Vererbung können wir in zukünftigen Updates des Projekts leicht neue Rollen erstellen. Durch die gleichen Attribute wird auch der Zugriff darauf vereinfacht, wenn wir mit vielen Personenobjekten mit unterschiedlichen Rollen arbeiten.

# Seitendokumentation

---

## 3.1 welcome\_screen.dart

Eine Seite, auf der Sie sich anmelden oder registrieren können

## 3.2 sign\_in\_page.dart

Eine Seite für Anmeldung

```
Future<void> _signIn() async {
    String email = _emailController.text;
    String password = _passwordController.text;

    try {
        User? user = await _auth.signInWithEmailAndPassword(email,
            password);

        if (user != null) {
            DocumentSnapshot userDoc =
                await _firestore.collection('user').doc(user.uid).get();
            if (userDoc.exists) {
                Map<String, dynamic>? userData =
                    userDoc.data() as Map<String, dynamic>?;
                String? role = userData?['role'];

                Fluttertoast.showToast(msg: "User is successfully signed in")
                    ;

                //Navigate to different homepages based on the role
                if (role == 'Betreiber') {
                    Fluttertoast.showToast(msg: "Navigate to betreiber homepage
                        ");
                    logger.t('User is successfully signed in with ' + user.uid)
                        ;
                    Navigator.pushAndRemoveUntil(
                        context,
                        MaterialPageRoute(builder: (context) => BetreiberHome(
                            userId: user.uid)),
                        (Route<dynamic> route) => false,
                    );
                } else if (role == 'Kunde') {
                    Fluttertoast.showToast(msg: "Navigate to kunde homepage");
                    logger.t('User is successfully signed in with ' + user.uid)
                        ;
                    Navigator.pushAndRemoveUntil(
                        context,
```

```

        MaterialPageRoute(builder: (context) => KundeHome(userId:
            user.uid)),
            (Route<dynamic> route) => false,
        );
    } else if (role == 'Techniker') {
        Fluttertoast.showToast(msg: "Navigate to techniker homepage
            ");
        logger.t('User is successfully signed in with ' + user.uid)
        ;
        Navigator.pushAndRemoveUntil(
            context,
            MaterialPageRoute(builder: (context) => TechnikerHome(
                userId: user.uid)),
            (Route<dynamic> route) => false,
        );
    } else {
        Fluttertoast.showToast(msg: 'Unknown role');
    }
} else {
    Fluttertoast.showToast(msg: "No Doc exists");
}
} else {
    Fluttertoast.showToast(msg: "Sign in failed");
}
} catch (e) {
    Fluttertoast.showToast(msg: "Error: ${e.toString()}");
}

User? user = await _auth.signInWithEmailAndPassword(email, password
);

if (user != null) {
    Fluttertoast.showToast(msg: "User is successfully signed in");
} else {
    Fluttertoast.showToast(msg: "Wrong email or password");
}
}
}

```

### 3.3 sign\_up\_page.dart

Eine Seite für User Registrierung

```

Future<void> _signUp() async {
String name = _nameController.text;
String email = _emailController.text;
String password = _passwordController.text;

try {
    User? user = await _auth.signupWithEmailAndPassword(email,
        password);

    if (user != null) {
        await _firestore.collection('user').doc(user.uid).set({
            'name': name,
            'email': email,
            'role': 'kunde',
        });
    }
}

```

```

        logger.t("User is successfully created");
        // Navigate to the desired screen or show success message
    } else {
        logger.e("Some error occurred");
        // Show error message
    }
} catch (e) {
    logger.e("Error: $e");
    // Show error message
}
}
}

```

### **3.4 betreiber\_home.dart**

In dieser Seite gibt es Button zu andere Seite, die Betreiber Funktionalität ist

### **3.5 kunde\_home.dart**

In dieser Seite gibt es Button zu andere Seite, die Kunde Funktionalität ist

### **3.6 techniker\_home.dart**

In dieser Seite gibt es Button zu andere Seite, die Techniker Funktionalität ist

### **3.7 betreiber\_auslastung\_verfolgen.dart**

In dieser Seite kann Betreiber den Bestand von aller Zubehöre verfolgen. Betreiber kann auch den Bestand addieren oder löschen. Wenn der Bestand addiert wird, muss Betreiber das Kaufpreis bezahlen. Das wird in unserem Datenbanken aktualisiert.

```

Future<void> updateFirestoreStock(int stock, String sparepartName)
{
    final firestore = FirebaseFirestore.instance;

    await firestore.collection('stock').doc('B1QHxe7XnhytZnMzDxNW').
        update({
            '$sparepartName' : stock
        });
}

```

### **3.8 betreiber\_einnahme\_verfolgen.dart**

In dieser Seite kann ein Betreiber die Einnahmen aus den Reparaturen sowie die Ausgaben für den Kauf der Ersatzteile verfolgen. Unterhalb der Seite werden die Gesamteinnahmen und Ausgaben sowie der Nettogewinn/-verlust angezeigt

### 3.9 kunde\_auftrage\_verfolgen.dart

In dieser Seite kann die Kunde ihre Aufträge durch das BookingId verfolgen. Die Kunde kann ihres BookingId eingeben. Dann wird das Zustand ihrer Buchung verfolgt.

Wenn die Buchung in Bearbeitung ist, gibt es einen Button, der anzeigt, ob es noch ein weiteres Ersatzteil zu kaufen oder zu bestellen gibt.

Wenn die Buchung fertig ist, dann kann die Kunde die Buchung löschen, wenn die Kunde schon die Fahrrad abgeholt wird.

```
Future<void> _searchBooking(String bookingId) async {
try {
    if (bookingId.isEmpty) {
        logger.e("Booking ID is empty");
        return;
    }
    logger.i("Searching booking with ID: $bookingId");

    final docSnapshot = await FirebaseFirestore.instance
        .collection('booking')
        .doc(bookingId)
        .get();
    if (docSnapshot.exists) {
        bookingSnapshot = docSnapshot;
        final data = bookingSnapshot!.data() as Map<String, dynamic>;
        final status = data['status'];
        logger.i("Booking found: ${data}");
        logger.i("Status from Firestore: $status");

        // update stepper data
        _updateStepperData(status);
    } else {
        logger.w("Booking not found");
        bookingSnapshot = null;
        stepperData = _getinitialStepperData();
        activeStep = 0;
    }
} catch (e) {
    logger.e(e);
    bookingSnapshot = null;
    stepperData = _getinitialStepperData();
    activeStep = 0;
}
}

Future<void> _confirmSpareParts(
    List<bool> selectedParts, List<dynamic> additionalSpareParts)
async {
if (bookingSnapshot != null) {
    final List<Map<String, dynamic>> updatedParts = [];
    double totalPrice = 0;

    for (int i = 0; i < selectedParts.length; i++) {
        if (selectedParts[i]) {
            updatedParts.add({
                'name': additionalSpareParts[i]['name'],
                'price': additionalSpareParts[i]['price'],
            });
        }
    }
    totalPrice = updatedParts.fold(0.0, (sum, part) => sum + part['price']);
    logger.i("Total price: $totalPrice");
}
}
```

```

        'confirmed': true,
    });
    totalPrice += additionalSpareParts[i]['price'];
} else {
    updatedParts.add(additionalSpareParts[i]);
}
}

await FirebaseFirestore.instance
    .collection('booking')
    .doc(bookingSnapshot!.id)
    .update({
    'additionalSpareParts': updatedParts,
    'price': FieldValue.increment(totalPrice),
});

Future<void> _deleteBooking() async {
if (bookingSnapshot != null) {
    await FirebaseFirestore.instance
        .collection('booking')
        .doc(bookingSnapshot!.id)
        .delete();
    setState(() {
        result = false;
        bookingSnapshot = null;
        stepperData = _getinitialStepperData();
        activeStep = 0;
    });
    logger.i("Booking deleted successfully");
} else {
    logger.w("No booking to delete");
}
}

void _updateStepperData(String status) {
stepperData = _getinitialStepperData(); // reset to initial stepper
    data

logger.i("Updating stepper data with status: $status");
// Update stepper data based on status
switch (status) {
    case 'pending':
        activeStep = 0;
        stepperData[0] = StepperData(
            title: stepperData[0].title,
            subtitle: stepperData[0].subtitle,
            iconWidget: _completedIcon(),
        );
        break;
    case 'confirmed':
        activeStep = 1;
        stepperData[0] = StepperData(
            title: stepperData[0].title,
            subtitle: stepperData[0].subtitle,
            iconWidget: _completedIcon(),
        );
        stepperData[1] = StepperData(

```

```

        title: stepperData[1].title,
        subtitle: stepperData[1].subtitle,
        iconWidget: _completedIcon(),
    );
    break;
case 'processing':
    activeStep = 2;
    stepperData[0] = StepperData(
        title: stepperData[0].title,
        subtitle: stepperData[0].subtitle,
        iconWidget: _completedIcon(),
    );
    stepperData[1] = StepperData(
        title: stepperData[1].title,
        subtitle: stepperData[1].subtitle,
        iconWidget: _completedIcon(),
    );
    stepperData[2] = StepperData(
        title: stepperData[2].title,
        subtitle: stepperData[2].subtitle,
        iconWidget: _completedIcon(),
    );
    break;
case 'finished':
    activeStep = 3;
    stepperData[0] = StepperData(
        title: stepperData[0].title,
        subtitle: stepperData[0].subtitle,
        iconWidget: _completedIcon(),
    );
    stepperData[1] = StepperData(
        title: stepperData[1].title,
        subtitle: stepperData[1].subtitle,
        iconWidget: _completedIcon(),
    );
    stepperData[2] = StepperData(
        title: stepperData[2].title,
        subtitle: stepperData[2].subtitle,
        iconWidget: _completedIcon(),
    );
    stepperData[3] = StepperData(
        title: stepperData[3].title,
        subtitle: stepperData[3].subtitle,
        iconWidget: _completedFinishIcon(),
    );
    break;
default:
    activeStep = 0;
    logger.w("Unknown status: $status");
    break;
}
setState(() {});
}

```

### 3.10 kunde\_reparatur\_buchen.dart

In dieser Seite kann die Kunde ihre Fahrradreparatur buchen. Es gibt Komponente und Zubehör. Die Kunde kann auch ihre gesamte Preis sehen.

```
Future<String> _confirmBooking() async {
    Fahrrarzt fahrrarzt =
        Provider
            .of<FahrrarztProvider>(context, listen: false)
            .fahrrarzt;
    final warehouse = fahrrarzt.warehouse;

    List<Map<String, dynamic>> komponenteList = [];
    List<Map<String, dynamic>> zubehoerList = [];

    for (int i = 0; i < _komponenteChecked.length; i++) {
        if (warehouse == null) {
            break;
        }
        if (_komponenteChecked[i] == true) {
            var sparepart = warehouse.keys.elementAt(i);
            if (i == 0 || i == 3 || i == 4) {
                if (i == 0) {
                    if (frontBrake) {
                        komponenteList
                            .add({"name": "front ${sparepart.name}", "done": false});
                }
                if (rearBrake) {
                    komponenteList
                        .add({"name": "rear ${sparepart.name}", "done": false});
                }
            } else if (i == 3) {
                if (frontTyre) {
                    komponenteList
                        .add({"name": "front ${sparepart.name}", "done": false});
                }
                if (rearTyre) {
                    komponenteList
                        .add({"name": "rear ${sparepart.name}", "done": false});
                }
            } else if (i == 4) {
                if (frontSpoke) {
                    komponenteList
                        .add({"name": "front ${sparepart.name}", "done": false});
                }
                if (rearSpoke) {
                    komponenteList
                        .add({"name": "rear ${sparepart.name}", "done": false});
                }
            }
        }
        komponenteList.add({"name": sparepart.name!, "done": false});
    }
}
```

```

        }
    }
}

for (int i = 0; i < _zubehoerChecked.length; i++) {
    if (warehouse == null) {
        break;
    }
    if (_zubehoerChecked[i] == true) {
        int realIndex = i + 5;
        var sparepart = warehouse.keys.elementAt(realIndex);
        zubehoerList.add({"name": sparepart.name!, "done": false});
    }
}

Booking booking = Booking(userId: widget.userId);

String? userName = await booking.fetchUserName();

Map<String, dynamic> userBookingMap = {
    'userId': widget.userId,
    'name': userName,
    'status': 'pending',
    'komponente': komponenteList,
    'zubehoer': zubehoerList,
    'price': _totalPrice,
};

String bookingId = await booking.addUserBooking(userBookingMap);
ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
        content: Text(LocaleData.confirm_text2.getString(context)),
    ),
);
return bookingId;
}

```

### 3.11 techniker\_auftrage\_ansehen.dart

In dieser Seite kann der Techniker die verfügbare Buchungen auswählen. Dann die Buchung wird in meiner Buchungen eingestellt. Der Techniker kann die Zustand von der Buchung in meiner Buchungen einstellen. Der Techniker kann auch weitere gebrauchte Komponente oder Zubehöre hinzufügen.

```

void addSpareParts(
    String bookingId,
    List<Sparepart> parts,
    List<bool> selectedParts,
    Map<Sparepart, TextEditingController> partPriceControllers) {
final List<Map<String, dynamic>> spareParts = [];
for (int i = 0; i < selectedParts.length; i++) {
    if (selectedParts[i]) {
        spareParts.add({
            'name': parts[i].name,
            'price': int.parse(partPriceControllers[parts[i]]!.text),
            'confirmed': false,
        });
    }
}

```

```
        }
    }
    FirebaseFirestore.instance.collection('booking').doc(bookingId).
        update({
            'additionalSpareParts': FieldValue.arrayUnion(spareParts),
        });
}

void updateBookingDetails(
    String bookingId,
    List<Map<String, dynamic>> components,
    List<Map<String, dynamic>> accessories) {
    FirebaseFirestore.instance.collection('booking').doc(bookingId).
        update({
            'komponente': components,
            'zubehoer': accessories,
            'status': 'processing',
        });
}
```

# Klassendokumentation

---

## 4.1 Klassen Diagram

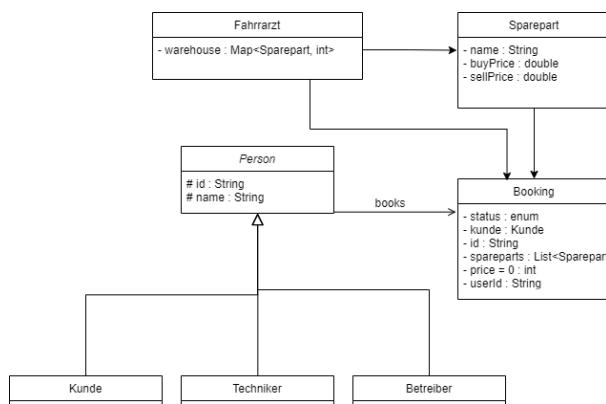


Figure 4.1.1: Klassendiagramm

## 4.2 person.dart

Diese Klasse ist das Parentclass von Kunde, Techniker, und Betreiber

```
class Person{
    String? id;
    String? name;

    Person (this.id, this.name);
}
```

## 4.3 kunde.dart

Diese Klasse ist für die Rolle von Kunde und das Childclass von Person

## 4.4 techniker.dart

Diese Klasse ist für die Rolle von Techniker und das Childclass von Person

## 4.5 betreiber.dart

Diseese Klasse ist für die Rolle von Betreiber und das Childclass von Person

## 4.6 booking.dart

Diese Klasse dient dazu, Buchungsdetails von Firestore zu erhalten

```
enum Status{
    PENDING,
    BESTATIGT,
    VERARBEITET,
    FERTIG
}

class Booking{
    Techniker? techniker;
    Kunde? kunde;
    String? id;
    List<Sparepart>? spareparts;
    int price = 0;
    Status status = Status.PENDING;
    String userId;

    Booking({
        required this.userId,
    });

    factory Booking.fromMap(Map<String, dynamic> map) {
        return Booking(
            userId: map['userId'],
        );
    }

    Future addUserBooking(Map<String, dynamic> userInfoMap) async {
        DocumentReference docRef = await FirebaseFirestore.instance.
            collection('booking').add(userInfoMap);
        return docRef.id;
    }

    Future<String?> fetchUserName() async {
        final userDoc = await FirebaseFirestore.instance.collection('user')
            .doc(userId).get();
        return userDoc.data()?['name'];
    }
}
```

## 4.7 fahrrarzt.dart

Diese Klasse initialisiert alle Bestand von der Komponente und Zubehöre zu Firestore.

```
class Fahrrarzt {
    static final Sparepart brakes = Sparepart(name: 'Brakes', buyPrice:
        20, sellPrice: 50);
    static final Sparepart chains = Sparepart(name: 'Chains', buyPrice:
        10, sellPrice: 20);
    static final Sparepart saddle = Sparepart(name: 'Saddle', buyPrice:
        5, sellPrice: 10);
    static final Sparepart tyres = Sparepart(name: 'Tyres', buyPrice: 7,
        sellPrice: 15);
```

```

        static final Sparepart spokes = Sparepart(name: 'Spokes', buyPrice:
            5, sellPrice: 12);
        static final Sparepart bikeLocks = Sparepart(name: 'Bike locks',
            buyPrice: 3, sellPrice: 10);
        static final Sparepart lights = Sparepart(name: 'Lights', buyPrice:
            2, sellPrice: 7);
        static final Sparepart luggageRacks = Sparepart(name: 'Luggage racks',
            buyPrice: 12, sellPrice: 20);
        static final Sparepart bikePumps = Sparepart(name: 'Bike pumps',
            buyPrice: 3, sellPrice: 10);

        List<Techniker>? alleTechniker;
        List<Booking>? alleBuchungen;
        List<Kunde>? alleKunden;

        Map<Sparepart, int>? warehouse = {
            brakes: 10,
            chains: 10,
            saddle: 10,
            tyres: 10,
            spokes: 10,
            bikeLocks: 10,
            lights: 10,
            luggageRacks: 10,
            bikePumps: 10,
        };
    }

    class FahrrarztProvider with ChangeNotifier {
        Fahrrarzt _fahrrarzt = Fahrrarzt();

        Fahrrarzt get fahrrarzt => _fahrrarzt;

        FahrrarztProvider() {
            _initFirestoreListener();
        }

        void _initFirestoreListener() {
            FirebaseFirestore.instance
                .collection('stock')
                .doc('BlQHxe7XnhytZnMzDxNW')
                .snapshots()
                .listen((snapshot) {
                    if (snapshot.exists) {
                        final data = snapshot.data();
                        _fahrrarzt.warehouse?.update(Fahrrarzt.brakes, (value) => data
                            ?['Brakes']);
                        _fahrrarzt.warehouse?.update(Fahrrarzt.chains, (value) => data
                            ?['Chains']);
                        _fahrrarzt.warehouse?.update(Fahrrarzt.saddle, (value) => data
                            ?['Saddle']);
                        _fahrrarzt.warehouse?.update(Fahrrarzt.tyres, (value) => data
                            ?['Tyres']);
                        _fahrrarzt.warehouse?.update(Fahrrarzt.spokes, (value) => data
                            ?['Spokes']);
                        _fahrrarzt.warehouse?.update(Fahrrarzt.bikeLocks, (value) =>
                            data?['Bike locks']);
                    }
                });
            }
        }
    }
}

```

```

        _fahrrarzt.warehouse?.update(Fahrrarzt.lights, (value) => data
            ?['Lights']);
        _fahrrarzt.warehouse?.update(Fahrrarzt.luggageRacks, (value) =>
            data?['Luggage racks']);
        _fahrrarzt.warehouse?.update(Fahrrarzt.bikePumps, (value) =>
            data?['Bike pumps']);

        notifyListeners();
    }
);
}
}

```

## 4.8 sparepart.dart

Diese Klasse ist die Details von jeden Komponenten und Zubehören

```

class Sparepart {
    String? name;
    int? buyPrice; //Price to restock item
    int? sellPrice; //Price to sell to customer

    Sparepart({
        this.name,
        this.buyPrice,
        this.sellPrice,
    });
}

```

# Usability Test

---

## 5.1 Ausführung der Tests

Wir haben Usability-Tests mit einigen Teilnehmern durchgeführt, um die Benutzerfreundlichkeit unserer App zu bewerten. Auf diese Weise können wir verstehen, was wir an unserer App falsch gemacht haben, was wir anders machen könnten und was wir im Hinblick auf die Benutzerfreundlichkeit richtig gemacht haben. Mit diesem Verständnis können wir uns reflektieren und uns für zukünftige Projekte verbessern.

Um diesen Test durchzuführen, haben wir einen Computer mit einem Handy-Emulator vorbereitet, der unser App-Prototyp-Programm ausführt. Da wir eine Anmeldeseite implementiert haben, gaben wir den Testnutzern die E-Mail und das Passwort, um sich in die App einzuloggen. Nach dem Einloggen ließen wir sie selbstständig mit unserer App interagieren, wobei wir versuchten, so wenig wie möglich zu erklären, wie die App funktioniert, um zu testen, ob unsere App von allen leicht zu bedienen ist.

## 5.2 Bewertungen

Die Ergebnisse unserer Tests waren eher neutral. Wir können die positiven Rückmeldungen wie folgt zusammenfassen: Die Funktionen erfüllen nicht nur die grundlegenden Anforderungen der App, sondern es gibt auch zusätzliche Funktionen, die das Benutzererlebnis verbessern, wie die Mehrsprachigkeit. Außerdem sind die Daten der App persistent. Das bedeutet, dass die Daten auch dann verfügbar bleiben, wenn die App geschlossen wird, da wir eine Datenbank zur Speicherung dieser Daten verwendet haben.

Es gab jedoch einige Funktionen, die wir anders gestalten könnten, um die Benutzerfreundlichkeit zu verbessern. Ein Punkt sind die Schaltflächen im Hauptmenü. Es wurde empfohlen, eine BottomNavigation zu verwenden, da dies die Anzahl der Klicks zur Nutzung der Funktionen reduziert. Besonders im Hauptmenü für Techniker haben wir nur eine Funktion, und wir hätten den Button einfach entfernen können.

Eine weitere Verbesserung, die wir vornehmen könnten, betrifft das Anzeigen von Aufträgen für den Kunden. Um die Buchungen des Kunden zu verfolgen, haben wir es so implementiert, dass der Kunde die Buchungsnummer in eine Suchleiste eingibt. Laut den Rückmeldungen unserer Tester fanden sie dies jedoch etwas kompliziert, da es viele Klicks und Eingaben erfordert. Stattdessen hätten wir einfach alle Buchungen anzeigen können.

Eine Funktion, die wir der App hinzufügen könnten, um das Benutzererlebnis zu verbessern, ist eine Rechnungsfunktion für den Kunden. Es wäre großartig, wenn ein Kunde die Rechnungen für seine laufenden oder vergangenen Reparaturaufträge einsehen könnte.

### **5.3 Reflexion**

Zusammenfassend lässt sich sagen, dass wir mit den durchgeführten Usability-Tests nicht nur die grundlegenden Anforderungen der App erfüllt haben, sondern auch Funktionen hinzugefügt haben, die zwar nicht notwendig waren, aber das Benutzererlebnis verbessern. Um diese Funktionen jedoch nutzen zu können, müssen wir sie zugänglicher machen, was bedeutet, dass weniger Klicks erforderlich sein sollten.

# Literatur

---

- [1] CodingxTalk. *Firebase Authentication in Flutter 2023 | Sign up & Login | Tutorials.* 2023. URL: <https://www.youtube.com/watch?v=YVYIPIow418&t=527s>.
- [2] ez-connect.net. *ionicons 0.2.2.* <https://pub.dev/packages/ionicons>. 2024.
- [3] Everyday flutter. *Building Flutter Screens | Login, Signup, Validation.* 2023. URL: [https://www.youtube.com/watch?v=lk6o7ukDYhs&list=PLniAq8IfyASJjnP4vX5e\\_vrQc0hHQG-Kq&index=27](https://www.youtube.com/watch?v=lk6o7ukDYhs&list=PLniAq8IfyASJjnP4vX5e_vrQc0hHQG-Kq&index=27).
- [4] flutter.dev. *cupertino\_icons 1.0.8.* [https://pub.dev/packages/cupertino\\_icons](https://pub.dev/packages/cupertino_icons). 2024.
- [5] Firebase Google. *cloud\_firestore 5.1.0.* [https://pub.dev/packages/cloud\\_firestore](https://pub.dev/packages/cloud_firestore). 2024.
- [6] Firebase Google. *firebase\_auth 5.1.2.* [https://pub.dev/packages/firebase\\_auth](https://pub.dev/packages/firebase_auth). 2024.
- [7] Firebase Google. *firebase\_core 3.2.0.* [https://pub.dev/packages/firebase\\_core](https://pub.dev/packages/firebase_core). 2024.
- [8] Firebase Google. *firebase\_storage 12.1.1.* [https://pub.dev/packages/firebase\\_storage](https://pub.dev/packages/firebase_storage). 2024.
- [9] Line Awesome Icon. *line\_awesome\_flutter 3.0.1.* [https://pub.dev/packages/line\\_awesome\\_flutter](https://pub.dev/packages/line_awesome_flutter). 2024.
- [10] khartikponnam.dev. *fluttershare 8.2.6.* <https://pub.dev/packages/fluttershare>. 2024.
- [11] Mitch Koko. *TIMELINE Widget • Flutter Tutorial.* 2023. URL: <https://www.youtube.com/watch?v=WP0h7utvaUc>.
- [12] Flutter Localization. *flutter\_localization 0.2.0.* [https://pub.dev/packages/flutter\\_localization](https://pub.dev/packages/flutter_localization). 2023.
- [13] Logger. *logger 2.3.0.* <https://pub.dev/packages/logger>. 2024.
- [14] material.io. *google\_fonts 6.2.1.* [https://pub.dev/packages/google\\_fonts](https://pub.dev/packages/google_fonts). 2024.
- [15] nixtomalon. *progress-tracker.* <https://github.com/nixtomalon/progress-tracker?tab=readme-ov-file>. 2024.
- [16] ogbomo.com. *flutter\_spinkit 5.2.1.* [https://pub.dev/packages/flutter\\_spinkit](https://pub.dev/packages/flutter_spinkit). 2024.
- [17] Ranga Reddy. *Custom Stepper in Flutter.* <https://dev.to/irangareddy/custom-stepper-in-flutter-40o8>. 2020.
- [18] App sense. *How to Connect Flutter App with Firebase [2024] Easy Setup!* 2024. URL: [https://www.youtube.com/watch?v=VCjG\\_v6oYyA&list=PLniAq8IfyASJjnP4vX5e\\_vrQc0hHQG-Kq&index=28](https://www.youtube.com/watch?v=VCjG_v6oYyA&list=PLniAq8IfyASJjnP4vX5e_vrQc0hHQG-Kq&index=28).
- [19] Bibek Timsina. *Custom Tab Bars.* <https://www.fluttertemplates.dev/templates/01-custom-tabbars>. 2024.
- [20] unverified uploader. *another stepper 1.2.2.* <https://pub.dev/packages/another stepper/changelog>. 2024.
- [21] xaha.dev. *lottie 3.1.2.* <https://pub.dev/packages/lottie>. 2024.