

# **Dokumentation der Payment App „Payero“**

**Mobile Computing (T3M40401)**

Studiengang Informatik

an der Dualen Hochschule Baden-Württemberg Center for Advanced Studies

von

**Calvin Reibenspieß und Florian Brändle**

Dezember 2023

**Bearbeitungszeitraum**  
**Kurs**  
**Gutachter**

4 Wochen  
Mobile Computing (T3M40401)  
Prof. Dr.-Ing. Dipl.-Inf. Kai Becher

## **Abstract**

Im Rahmen dieses Projekts soll eine mobile Applikation mit dem Framework Flutter entwickelt werden, die plattformunabhängig unter iOS und Android betrieben werden kann. Entsprechend der Aufgabenstellung soll die entwickelte App die Durchführung von Bezahlvorgängen mittels PayPal durch das Scannen eines QR-Codes ermöglichen. Die Zahlungsintegration wird mithilfe von Braintree realisiert.

# Inhaltsverzeichnis

<b>1 Softwaretechnik</b>	<b>1</b>
1.1 Softwarepakete . . . . .	1
1.2 Entwicklungsumgebung . . . . .	1
1.3 Emulation . . . . .	2
1.4 Hardware . . . . .	5
<b>2 Zielsetzung</b>	<b>6</b>
2.1 Anforderungen . . . . .	6
2.2 Umgesetzte Funktionen . . . . .	7
<b>3 Konzept und Design</b>	<b>8</b>
3.1 Markenentwicklung . . . . .	8
3.2 Wireframes . . . . .	9
<b>4 Implementierung</b>	<b>12</b>
4.1 App Tour . . . . .	12
4.2 Transaktionshistorie . . . . .	21
4.3 QR-Code Scanner . . . . .	22
4.4 Bezahlintegration mit Braintree . . . . .	26
4.5 Benutzer-Authentifizierung . . . . .	30
4.6 Backend . . . . .	30
<b>5 Klassendiagramm</b>	<b>34</b>
<b>6 Testing</b>	<b>36</b>
<b>7 Ausblick und Weiterentwicklung</b>	<b>38</b>
<b>8 Verzeichnisse</b>	<b>39</b>
<b>9 Copyright</b>	<b>40</b>
<b>Änderungshistorie</b>	<b>I</b>
<b>Abbildungsverzeichnis</b>	<b>II</b>
<b>Quellcodeverzeichnis</b>	<b>III</b>

# **1 Softwaretechnik**

Dieses Kapitel enthält alle benötigten Software- und Hardware-Tools sowie die verwendeten Versionen.

## **1.1 Softwarepakete**

Die Betriebssysteme auf denen gearbeitet wurde sind:

- Windows 10 Enterprise 22H2
- macOS Version 14.1 (23B74), Apple M1 Pro

## **1.2 Entwicklungsumgebung**

Die folgenden Entwicklungsumgebungen, Erweiterungen und Tools wurden verwendet:

- Visual Studio Code v.1.84.2
  - Erweiterung Dart v.3.78.0
  - Erweiterung Flutter v.3.78.0
- Android Studio Giraffe | 2022.3.1 Patch 4
- Android SDK v.34.0.0
- Flutter v.3.16.0
- Visual Studio Community 2022 v.17.8.1

## 1.3 Emulation

Die App wurde sowohl im Simulator, als auch auf zwei physischen Endgeräten getestet.  
Die verwendeten Geräte umfassen:

iOS:

- Simulator: iPhone SE (3rd Generation), iOS 17.0

Die iOS-Simulation wurde mit dem SimulatorKit von Apple in der Version 15.0.1 (1015.2) ausgeführt.

Android:

- Simulator: Pixel 3a, API-Version 27, Android 8.1 (Oreo)
- Simulator: Pixel 3a, API-Version 34, Android 14
- Simulator: Pixel 2, API-Version 34, Android 14

Die Emulatoren haben folgende Eigenschaften und Einstellungen:

```
Name: Pixel_3a_API_34_extension_level_7_arm64-v8a
CPU/ABI: arm64
Path: /Users/calvinreibenspiess/.android/avd/Pixel_3a_API_34_extension_level_7_arm64-v8a.avd
Target: google_apis [Google APIs] (API level 27)
Skin: pixel_3a
SD Card: 800 MB
Avdid: Pixel_3a_API_34_extension_level_7_arm64-v8a
PlayStore.enabled: false
avd.ini.displayname: Pixel_3a_API_34_extension_level_7_arm64-v8a
avd.ini.encoding: UTF-8
disk.dataPartition.size: 6G
fastboot.chosenSnapshotFile:
fastboot.forceChosenSnapshotBoot: no
fastboot.forceColdBoot: no
fastboot.forceFastBoot: yes
hw.accelerometer: yes
hw.arc: false
hw.audioinput: yes
hw.battery: yes
hw.camera.back: emulated
hw.camera.front: emulated
hw.cpu.ncore: 1
hw.dPad: no
hw.device.hash2: MD5:0e6953ebf01bdc6b33a2f54746629c50
hw.device.manufacturer: Google
hw.device.name: pixel_3a
hw.gps: yes
hw.gpu.enabled: yes
hw.gpu.mode: auto
hw.initialOrientation: Portrait
hw.keyboard: yes
hw.lcd.density: 440
hw.lcd.height: 2220
hw.lcd.width: 1080
hw.mainKeys: no
hw.ramSize: 1536
hw.sdCard: yes
hw.sensors.orientation: yes
hw.sensors.proximity: yes
hw.trackBall: no
image.sysdir.1: system-images/android-27/google_apis/arm64-v8a/
runtime.network.latency: none
runtime.network.speed: full
showDeviceFrame: yes
skin.dynamic: yes
tag.display: Google APIs
tag.id: google_apis
vm.heapSize: 256
```

Abbildung 1.1: Einstellungen des Android-Emulators von Calvin Reibenspieß.

```
Properties
avd.ini.displayname           Pixel 3a API 34
avd.ini.encoding              UTF-8
AvdId                         Pixel_3a_API_34
disk.dataPartition.size        6442450944
fastboot.chosenSnapshotFile   no
fastboot.forceChosenSnapshotBoot no
fastboot.forceColdBoot         no
fastboot.forceFastBoot         yes
hw.accelerometer              yes
hw.arc                          false
hw.audioInput                  yes
hw.battery                      yes
hw.camera.back                 virtualscene
hw.camera.front                emulated
hw.cpu.ncore                   4
hw.device.hash2                MD5:0e6953ebf01bdc6b33a2f54746629c50
hw.device.manufacturer          Google
hw.device.name                  pixel_3a
hw.dPad                         no
hw.gps                          yes
hw.gpu.enabled                  yes
hw.gpu.mode                     auto
hw.initialOrientation           Portrait
hw.keyboard                     yes
hw.lcd.density                 440
hw.lcd.height                  2220
hw.lcd.width                   1080
hw.mainKeys                     no
hw.ramSize                      1536
hw.sdCard                       yes
hw.sensors.orientation          yes
hw.sensors.proximity            yes
hw.trackBall                    no
image.androidVersion.api       34
image.sysdir.1                  system-images\android-34\google_apis_playstore\x86_64\
PlayStore.enabled               true
runtime.network.latency          none
runtime.network.speed            full
showDeviceFrame                 yes
skin.dynamic                     yes
tag.display                      Google Play
tag.id                           google_apis_playstore
vm.heapSize                      228
```

Abbildung 1.2: Einstellungen des Android-Emulators von Florian Brändle Pixel 3a.

```
Properties
avd.ini.displayname      Pixel 2 API 34
avd.ini.encoding         UTF-8
AvdId                   Pixel_2_API_34
disk.dataPartition.size  6442450944
fastboot.chosenSnapshotFile
fastboot.forceChosenSnapshotBoot no
fastboot.forceColdBoot    no
fastboot.forceFastBoot   yes
hw.accelerometer        yes
hw.arc                  false
hw.audioInput            yes
hw.battery               yes
hw.camera.back           virtualscene
hw.camera.front          emulated
hw.cpu.ncore              4
hw.device.hash2          MD5:55acbc835978f326788ed66a5cd4c9a7
hw.device.manufacturer   Google
hw.device.name            pixel_2
hw.dPad                 no
hw.gps                  yes
hw.gpu.enabled           yes
hw.gpu.mode              auto
hw.initialOrientation    Portrait
hw.keyboard              yes
hw.lcd.density           420
hw.lcd.height             1920
hw.lcd.width              1080
hw.mainKeys              no
hw.ramSize                1536
hw.sdCard                 yes
hw.sensors.orientation   yes
hw.sensors.proximity     yes
hw.trackBall              no
image.androidVersion.api 34
image.sysdir.1            system-images\android-34\google_apis_playstore\x86_64\
PlayStore.enabled         true
runtime.network.latency   none
runtime.network.speed     full
showDeviceFrame           yes
skin.dynamic              yes
tag.display               Google Play
tag.id                    google_apis_playstore
vm.heapSize                228
```

Abbildung 1.3: Einstellungen des Android-Emulators von Florian Brändle Pixel 2.

## 1.4 Hardware

Die zwei Endgeräte auf denen getestet wurde, sind folgende:

- iPhone 12 Pro, iOS 16.1.2 | IMEI: 35 861174 3743835
- Nokia 7 Plus, Android 10

# **2 Zielsetzung**

Ziel des Projektes ist die Realisierung einer plattformunabhängigen App. Dies geschieht mit dem Framework Flutter und der dazugehörigen Programmiersprache Dart. Die App soll dabei einen Bezahlvorgang durch das Scannen eines QR-Codes ermöglichen. Unsere App trägt den Namen „Payero“ und läuft auf iOS und Android.

## **2.1 Anforderungen**

Nachfolgend werden die technischen Anforderungen für die Umsetzung der Applikation formuliert. Anhand dieser Anforderungen sollen die Features der App umgesetzt und auch die Konzeption durchgeführt werden:

- Plattformunabhängigkeit
  - Erfassen von QR-Codes mit fest definiertem Empfänger und Zahlungsbetrag
  - Integration von PayPal
  - Integration von weiteren Bezahlmethoden (Zusatz-Feature)
  - Abwicklung von Zahlungsvorgängen mit individuellen Beträgen
- Backend-Anbindung (Zusatz-Feature)
  - Benutzer-Authentifizierung
  - Transaktions-Historie
  - QR-Code-Generierung
- Umsetzung der Screens
  - Unterstützung von Endgeräten mit unterschiedlichen Bildschirmgrößen
  - Anwenderfreundlichkeit
- Einstellungen (Zusatz-Feature)
  - Logout

Die als Zusatz-Feature angegebenen Punkte stellen über die Aufgabenstellung hinausgehende Anforderungen dar, die im Rahmen des Projekts umgesetzt werden könnten.

## 2.2 Umgesetzte Funktionen

Die App bietet folgende Funktionen:

- Scannen eines QR-Codes
  - Nur Payero eigene QR-Codes zulassen
  - BenutzerID im QR-Code hinterlegen
  - Betrag im QR-Code hinterlegen
- Anbieten mehrerer Bezahlverfahren
  - PayPal
  - Kreditkarten (VISA, MasterCard, AMEX über GooglePay und ApplePay)
  - GooglePay
  - ApplePay (nach Eingabe einer gültigen Merchant-ID)
- Datenbankanbindung
  - Speichert BenutzerID
  - Speichert Transaktionen eines Benutzers
  - Erzeugt dynamische nutzerspezifische QR-Codes
- Benutzerkonto
  - Enthält Transaktionshistorie
  - Enthält eigenen QR-Code
  - Enthält Option eigenen QR-Code zu speichern
- Einfache Bedienbarkeit durch Benutzer
- Designlanguage mit einer eigenen Corporate Identity mit MaterialDesign

Zur Erweiterung dieser grundlegenden Idee ist es geplant eine Peer-To-Peer Bezahl-Applikation zu entwickeln, mit der Anwender individuelle QR-Codes besitzen und über diese Bezahlvorgänge an die eigene Person abgewickelt werden können.

# 3 Konzept und Design

Zum Projektstart wurde sich zunächst mit der Konzeption des genauen Anwendungsfalles und der benötigten Funktionen beschäftigt. So konnte ein gemeinsames Verständnis für das Projekt entwickelt werden.

Angedacht ist die Entwicklung einer Peer-to-Peer Bezahl-Applikation, mit der Anwender untereinander durch das Scannen eines individuellen QR-Codes Transaktionen durchführen können. Der QR-Code sollte dafür entsprechend kodiert werden und den Zahlungsempfänger beinhalten, wohingegen der Betrag durch den Zahlungssender wählbar und durch ein numerisches Eingabefeld bestimmbar wäre.

Im weiteren Verlauf wurde daran gearbeitet, nicht nur die Applikation technisch zu implementieren, sondern auch ansprechend gestalterisch umzusetzen. Um der App ebenfalls einen Namen und ein Gesicht zu geben, konnte mit wenig Aufwand ein Name gefunden und ein Logo generiert werden.

Anschließend wurden noch Screendesigns angefertigt, an denen sich während der Umsetzung orientiert wurde.

## 3.1 Markenentwicklung

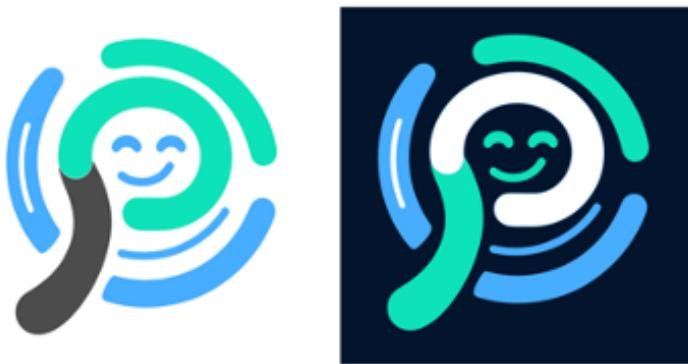
Für das Projekt sollte ein passender Name gefunden werden, der den Bezug zur Kernfunktion, dem Bezahlen von Geldbeträgen widerspiegelt. Um möglichst wenig Konfliktpotential zu bestehenden Unternehmen oder Institutionen zu bieten, wurde bei der Wahl des Namens im Oktober 2023 darauf geachtet, dass dieser möglichst keine Treffer bei der Suche mit Suchmaschinen wie Google bietet und auch der Name mit gängigen Top-Level-Domains noch verfügbar ist.

Mithilfe des KI-gestützten Tools Bing Chat, welches auf dem von OpenAI entwickeltem GPT-4 basiert, wurden dann Vorschläge entwickelt und anschließend durch die Autoren ausgewertet.

Es wurde sich für den Namen „Payero“ entschieden, da die ersten Silben in vielen Sprachen mit dem Verb „bezahlen“ konnotieren. So gibt es etwa im Französischen das Verb „payer“ und im Englisch das Nomen „payer“.

Für das Logo wurden ebenfalls Vorschläge mit Bing Chat gesammelt, die dann als Grundlage genutzt und mit Grafiktools aufgearbeitet wurden. Das Logo wurde um einen Schriftzug in der Schriftart Nunito ergänzt. In Abbildung 3.1 wird das Logo alleinstehend auf hellem und dunklem Untergrund dargestellt, sowie die Kombination mit Schriftzug.

Logo auf hellem und dunklem Untergrund



Logo mit Schriftzug



---

Abbildung 3.1: Darstellung des Logos auf hellen und dunklen Hintergründen, sowie in Kombination mit Schriftzug.

## 3.2 Wireframes

Das Konzept der Applikation sieht es vor, den Anwender durch mehrere Ansichten zu führen, bei denen unter anderem die Bezahlfunktionen umgesetzt, aber auch beispielsweise eine Transaktions-Historie ausgegeben wird. Konkret sind folgende Ansichten angedacht:

### SplashScreen: Bildschirm mit kurzer Anzeigedauer beim Start der App

Der SplashScreen sollte beim Öffnen der App für einige Sekunden angezeigt werden und das Logo in Kombination mit dem Namen der App darstellen. Aus technischer Sicht könnten im Hintergrund bereits Daten geladen werden.

### **MainScreen: Bildschirm mit Übersicht über letzte Transaktionen und Absprünge zu den Kernfunktionen**

Dieser Bildschirm stellt das zentrale Bedienfeld der App dar, da hier alle wichtigen Funktionen gebündelt werden. Es soll ein anklickbares Icon zum Absprung zum SettingsScreen dargestellt werden. Ebenso wird hier der persönliche und dynamisch generierte QR-Code präsentiert. Falls ein Benutzer seinen QR-Code an weitere Personen weitergeben möchte, kann er auf diesen bereits recht schnell nach dem Start der App zugreifen. Ein weiteres wichtiges Element dieses Bildschirms wird die Transaktions-Historie sein, da hier der Endanwender auf den ersten Blick die zuletzt durchgeföhrten Transaktionen einsehen kann. Um zielgerichtet auf die Kernfunktion der App, dem Einscannen von QR-Codes anderer Teilnehmer, zu verweisen, wird hier ein großer Button mit dem Text „Scannen“ angedacht. Ein Klick auf diesen wird auf den QRScanScreen umleiten.

### **SettingsScreen: Bildschirm mit Einstellungen und technischen Informationen zur App**

Der SettingsScreen wird die Versionsnummer der App darstellen, sowie Informationen zu den Herausgebern. Sinnvoll wäre es hier ebenfalls einen Changelog unterzubringen und beispielsweise Einstellmöglichkeiten für Sprachwahl, oder ähnliche Benutzereinstellungen anzubieten. Für das Projekt wäre es ebenfalls denkbar hier Debug-Informationen wie etwa Benutzer-ID darzustellen.

### **QRScanScreen: Bildschirm zum Erfassen von QR-Codes**

Der QRScanScreen wird die zentrale Funktion bieten, die individuellen QR-Codes von anderen App-Nutzern einzuscannen. Sofern ein valider QR-Code erfasst wird, sollte auf den FoundCodeScreen umgeleitet werden.

### **FoundCodeScreen: Bildschirm zur Tätigung der Transaktion**

Der FoundCodeScreen wird angezeigt, wenn ein valider QR-Code gescannt wurde und ein Zahlungsvorgang initiiert werden kann. Für die Implementierung der Zahlungsabwicklung wird der Bezahl Dienstleister Braintree vorgesehen, der unter anderem Zahlungen mit PayPal, Kreditkarte, Google Pay und Apple Pay ermöglicht. Ein entsprechendes Widget soll hierfür an dieser Stelle eingebunden werden.

#### **EndScreen: Bildschirm zur Darstellung einer Erfolgsmeldung bei getätigter Transaktion**

Nach erfolgreicher Zahlungsabwicklung sollte der Benutzer der App ein Feedback erhalten und wird daher zu einem EndScreen weitergeleitet. Hier werden noch einmal die wichtigsten Eckdaten zur kürzlich abgeschlossenen Transaktion dargestellt. Denkbar wäre es hier ebenfalls eine zum Design passende Illustration darzustellen, um den Endnutzer möglicherweise für die Tätigung weiterer Transaktionen zu animieren und den sonst langweiligen und tristen Bezahlprozess etwas aufzulockern.

In Abbildung 3.2 werden noch einmal gesammelt die angefertigten Entwürfe dargestellt.

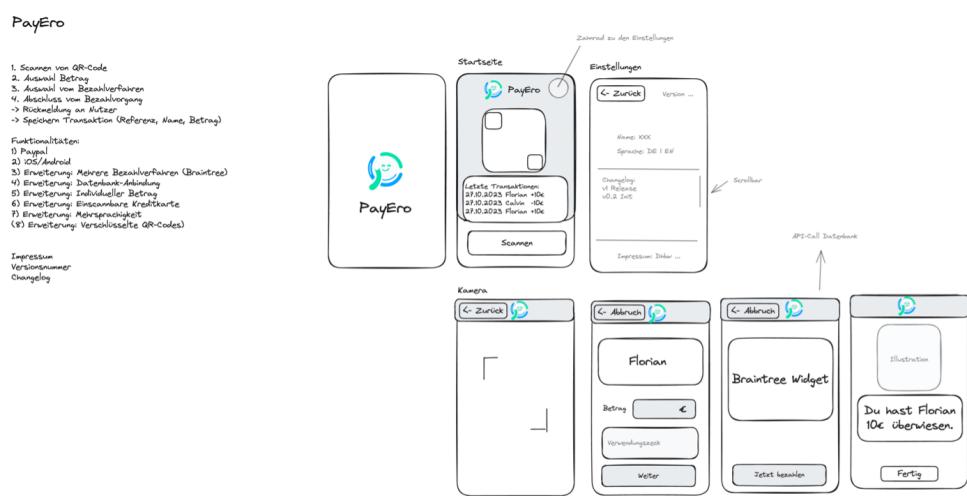


Abbildung 3.2: Darstellung der angefertigten Wireframes.

# 4 Implementierung

Dieses Kapitel beinhaltet sämtliche Implementierungen und Features in der App. Dabei werden benutzte Libraries und Code aus der App gezeigt.

## 4.1 App Tour

Im Folgenden wird der Ablauf eines Bezahlvorgangs vom Start der App bis zu ihrer Beendigung dargestellt.

Beim Start der App wird man mit einem SplashScreen begrüßt.

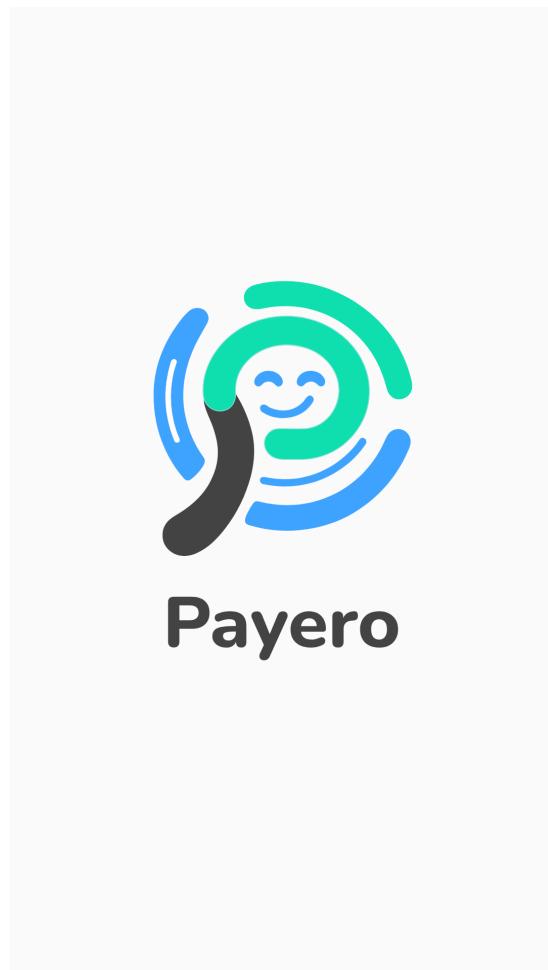


Abbildung 4.1: SplashScreen

Danach erfolgt beim ersten Start der Anwendung die Registrierung eines Benutzerkontos. Dabei wird geprüft, ob die Felder ausgefüllt sind und ob eine E-Mail-Adresse im richtigen Format angegeben ist. Dieser Schritt wird übersprungen, wenn bereits bei einer früheren Nutzung ein Benutzerkonto angelegt wurde.



Abbildung 4.2: Registrierungsvorgang

Danach gelangt man ins Hauptmenü. Falls bereits ein Benutzerkonto besteht, gelangt man direkt nach dem SplashScreen ins Hauptmenü.

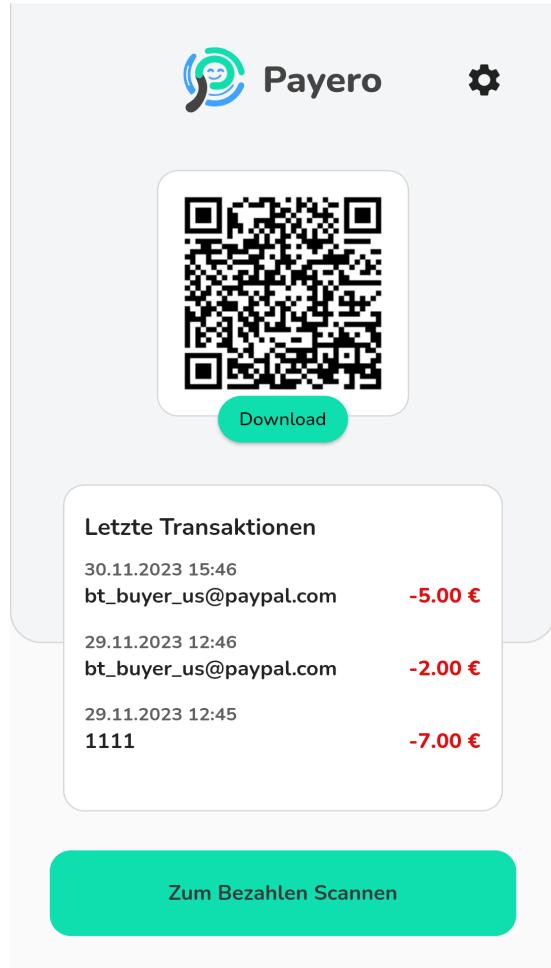


Abbildung 4.3: Hauptmenü

Im Hauptmenü finden Sie einen mit Ihrem Benutzerkonto verknüpften QR-Code, den Sie einscannen und herunterladen können. Außerdem kann eine Transaktionshistorie mit den letzten drei Zahlungen eingesehen werden. Ganz unten befindet sich ein Button. Mit diesem wird der Bezahlvorgang gestartet. In der rechten oberen Ecke befindet sich ein Icon für die Einstellungen. Ein Klick darauf öffnet die Einstellungsseite.

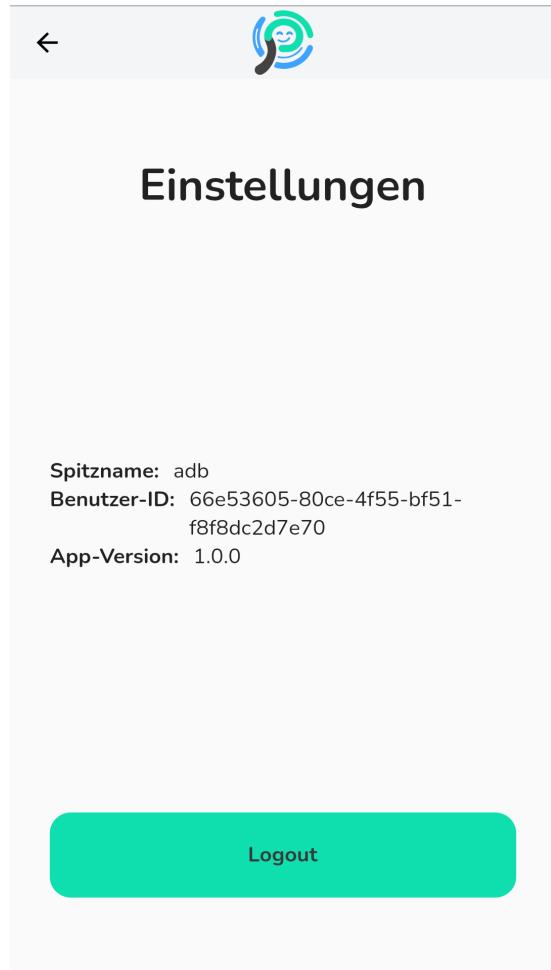


Abbildung 4.4: Einstellungen

Auf der Einstellungsseite sieht man seinen Benutzernamen und die dazugehörige ID. Auch die Version der Anwendung ist hier aufgelistet. Man kann sich auch von seinem Benutzerkonto abmelden. Geht man nun zurück ins Hauptmenü und startet den Bezahlvorgang mit einem Klick auf den Button, öffnet sich eine neue Seite. Hier wird die Kameraberechtigung zum Einscannen eines QR-Codes abgefragt. Dabei wird geprüft, ob es sich bei diesem QR-Code auch um einen Payero-QR-Code handelt. Ist dies nicht der Fall, wird dies so angezeigt:

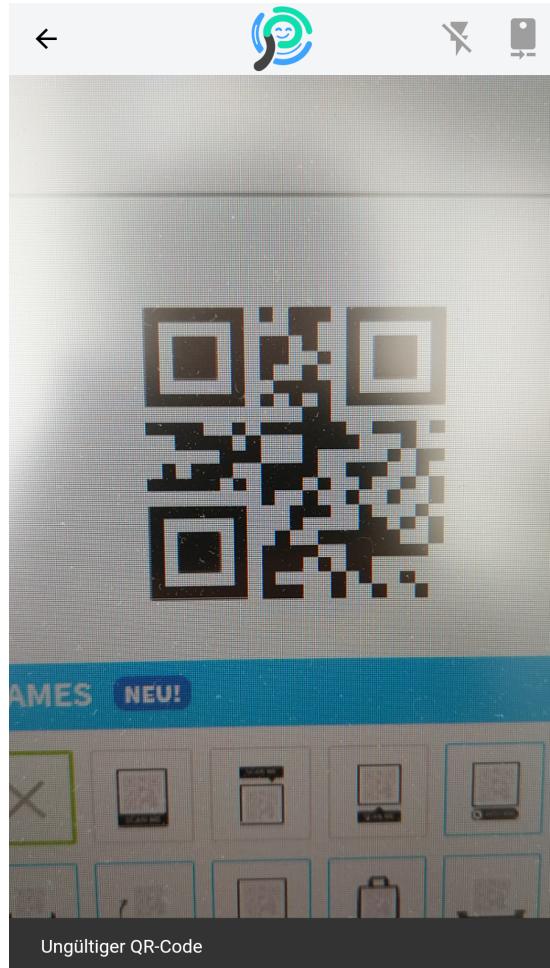


Abbildung 4.5: Ungültiger QR-Code

Eine Snackbar weist den Benutzer darauf hin, dass es sich nicht um einen gültigen QR-Code handelt. Zusätzlich kann der Benutzer oben rechts durch Klick auf das entsprechende Icon den Kamerablitz aktivieren, falls es zu dunkel zum Scannen ist. Es ist auch möglich, die Frontkamera zu aktivieren und zwischen dieser und der Rückkamera zu wechseln, indem man auf das entsprechende Symbol klickt. Das Scannen eines QR-Codes erfolgt automatisch. Der Benutzer muss dazu keine Schaltflächen anklicken. Wird ein gültiger QR-Code erkannt, wird der Benutzer automatisch auf eine Bestätigungsseite weitergeleitet.

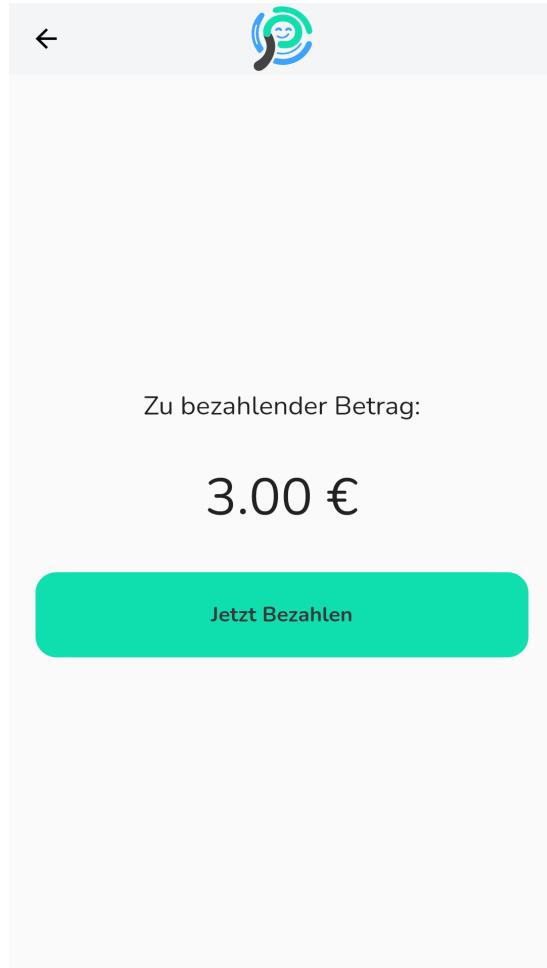


Abbildung 4.6: Bestätigungsseite

Auf dieser Seite wird der Benutzer über den zu zahlenden Betrag informiert. Durch Anklicken des Buttons kann der Benutzer den Bezahlvorgang starten oder über den Zurück-Button abbrechen. Beim Start des Bezahlvorgangs öffnet sich ein Dropin-Fenster.

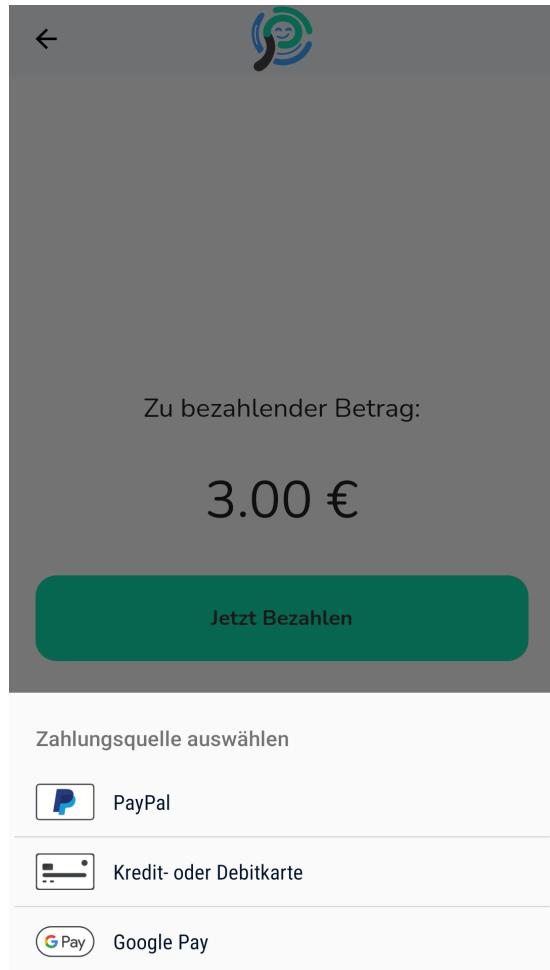


Abbildung 4.7: Bezahlmethoden

Hier sieht der Benutzer alle verfügbaren Zahlungsmethoden. Dazu gehören immer PayPal und Kreditkarte. Unter Android gehört auch GooglePay dazu, während unter iOS ApplePay zwar möglich wäre, aber aufgrund des fehlenden Entwicklerkontos nicht zur Verfügung steht. Je nach Bezahlmethode muss der Nutzer seine Kreditkartendaten oder andere Daten wie das PayPal-Konto eingeben. Dabei wird auch geprüft, ob alles passt, z.B. das Ablaufdatum der Kreditkarte. Im Fehlerfall wird der Benutzer darauf hingewiesen. Auch jetzt ist es noch möglich, den Bezahlvorgang abzubrechen. Der Benutzer wird darüber in einer Snackbar informiert. Führt der Benutzer die Zahlung jedoch aus, wird er in einer Snackbar über die Verarbeitung der Zahlung informiert. Diese Snackbar enthält immer die Zahlungsart und die Informationen dazu. Bei PayPal wäre dies PayPal und die dazugehörige E-Mail. Bei einer Kreditkarte ist es die Art der Kreditkarte und die Endziffern der Kreditkartennummer. In diesem Beispiel ist es eine VISA Kreditkarte mit den Endziffern 1111.



Abbildung 4.8: Verarbeitung des Bezahlvorganges

War der Vorgang erfolgreich, gelangt der Benutzer auf einen weiteren Bildschirm. Dieser zeigt dem Benutzer an, dass die Zahlung erfolgreich durchgeführt wurde. Außerdem wird der gezahlte Betrag angezeigt.

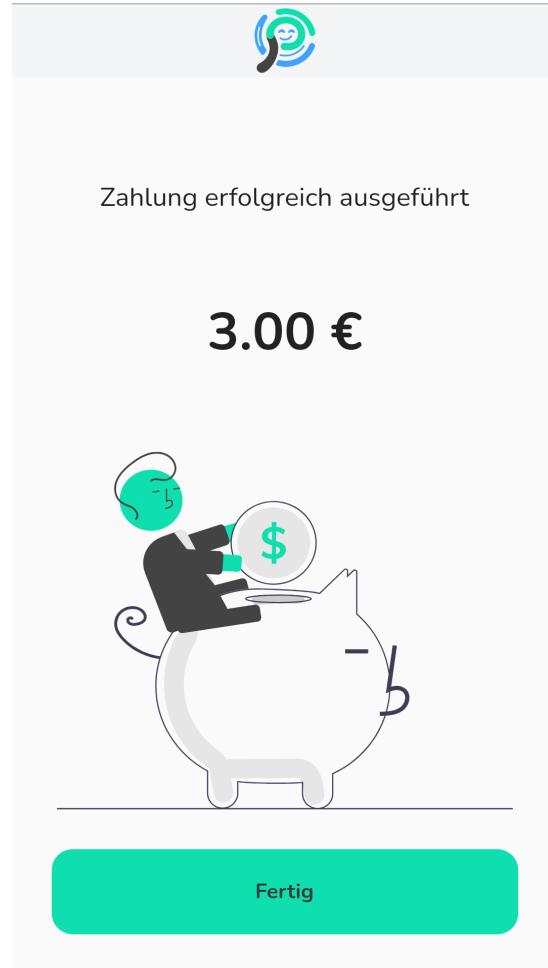


Abbildung 4.9: Erfolgreicher Bezahlvorgang

Klickt der Benutzer auf den Button Fertig, gelangt er zurück zum Hauptmenü. Dort ist in der Transaktionshistorie auch direkt ersichtlich, dass die Zahlung erfolgreich durchgeführt wurde.

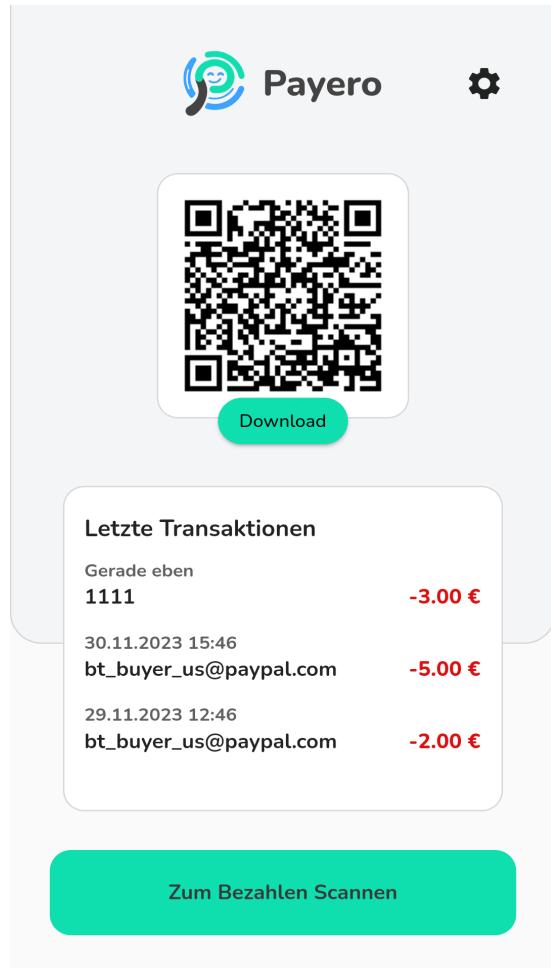


Abbildung 4.10: Zahlung in der Transaktionshistorie sichtbar

Der Benutzer kann nun einen weiteren Bezahlvorgang starten.

## 4.2 Transaktionshistorie

Die Transaktionshistorie wurde komplett mithilfe von den zwei selbst entwickelten Widgets TransactionHistory und TransactionEntry implementiert. Diese sind beide stateless, bekommen ihre Inhalte also vom MainScreen in Form eines Arrays übergeben.

Die TransactionHistory stellt mithilfe der take()-Funktion die ersten drei Einträge des übergebenen Arrays dar, wodurch nur die aktuellsten Transaktionen angezeigt werden. Sofern keine Transaktionen vorhanden sind, wird ein Text dargestellt.

Im TransactionEntry wird ein Eintrag mit dem Zeitstempel der Transaktion, der Beschreibung und dem Betrag gerendert. Ausgehende Transaktionen stellen negative Werte dar und werden in roter Farbe dargestellt. Andernfalls werden die Beträge grün dargestellt.

Bei der Überweisung per Kreditkarte kam es zu dem Problem, dass sehr lange Beschreibungstexte abgespeichert wurden. Um ein horizontales Überlaufen zu verhindern, wird der Text daher nach 20 Zeichen abgeschnitten.

Ebenfalls wurde eingebaut, dass die Zeitstempel formatiert je nach Alter als etwa „Gerade eben“, „Vor 24 Minuten“, „16:43 Uhr“ oder im Format „23.11.2023 16:00“ ausgegeben werden. Das Verhalten wird in Abbildung 4.11 dargestellt.

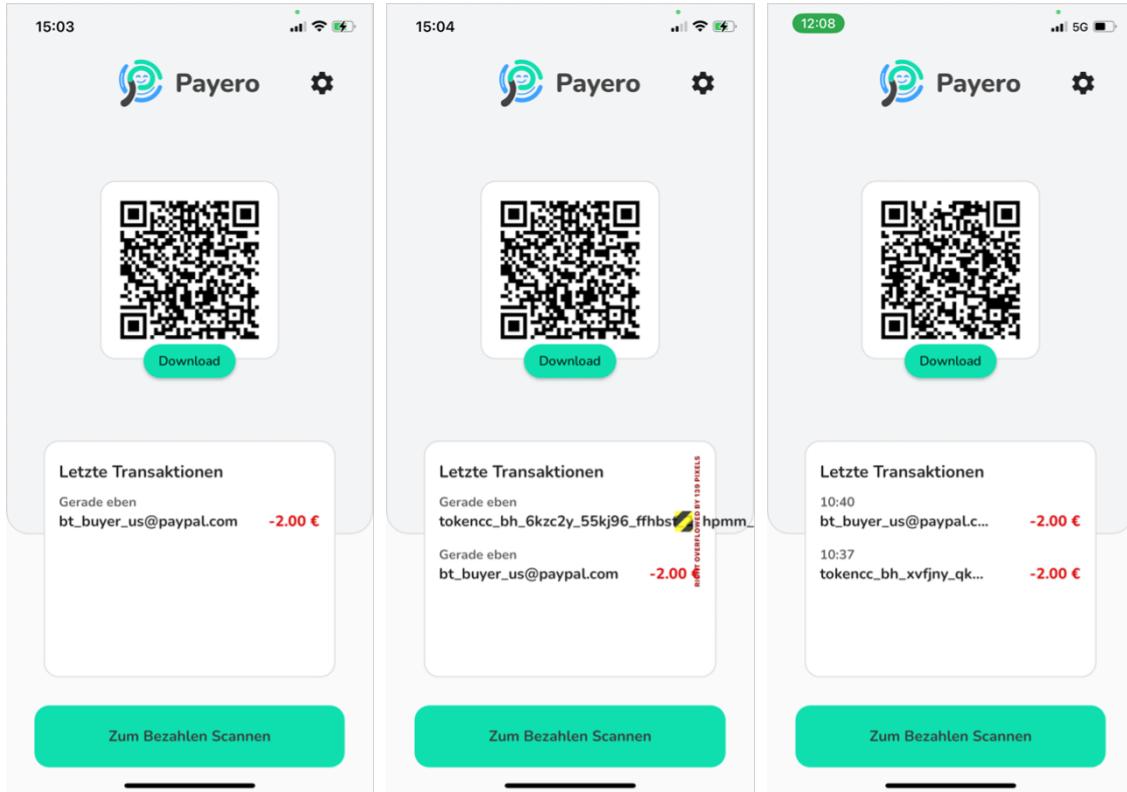


Abbildung 4.11: Darstellung der Transaktionshistorie in verschiedenen Fällen, mit einer Transaktion, ohne Textkürzung, und mit mehreren Transaktionen.

### 4.3 QR-Code Scanner

Zum Erkennen und Scannen der QR-Codes wird die Library „mobile\_scanner“ in der Version „3.5.2“ genutzt. Diese arbeitet sowohl unter iOS als auch Android zuverlässig. Die Integration findet in der „qr\_code\_scan\_screen.dart“ Datei statt. Es wird hierzu die Klasse „\_QRScanScreenState“ erstellt, in welcher die Funktionen der Library implementiert werden. Unter anderem ist hier die Steuerung der Kamera, das Umschalten zwischen Front- und Rückkamera und die Steuerung des Kamerablitzes möglich.

```

1  appBar: PayeroHeader(showBackButton: true, actions: [
2      IconButton(
3          color: Colors.white,

```

```

4     icon: ValueListenableBuilder<
5         valueListenable: cameraController.torchState,
6         builder: (context, state, child) {
7             switch (state as TorchState) {
8                 case TorchState.off:
9                     return const Icon(Icons.flash_off, color: Colors.grey)
10                    ;
11                 case TorchState.on:
12                     return const Icon(Icons.flash_on, color: Colors.yellow
13                        );
14             }
15         },
16         iconSize: 32.0,
17         onPressed: () => cameraController.toggleTorch(),
18     ),
19     IconButton(
20         color: Colors.white,
21         icon: ValueListenableBuilder<
22             valueListenable: cameraController.cameraFacingState,
23             builder: (context, state, child) {
24                 switch (state as CameraFacing) {
25                     case CameraFacing.front:
26                         return const Icon(Icons.camera_front, color: Colors.
27                             grey);
28                     case CameraFacing.back:
29                         return const Icon(Icons.camera_rear, color: Colors.
30                             grey);
31                 }
32             },
33         ),
34     )));

```

Quellcode 4.1: Kamerasteuerung

Der Scan wird mit einer kurzen Verzögerung gestartet, um sicherzustellen, dass alles vollständig geladen ist. Die erfolgreich gescannten Barcodes und QR-Codes werden als Liste gespeichert. Wir verwenden hier aber immer nur den ersten erfolgreichen Code in der Liste.

```

1 body: MobileScanner(
2     startDelay: true,
3     controller: cameraController,
4     onDetect: (capture) {

```

```

5     final List<Barcode> barcodes = capture.barcodes;
6     if (barcodes.isNotEmpty && !_screenOpened) {
7         final String code = barcodes.first.rawValue ?? "----";
8         debugPrint('QRCode found! $code');

```

Quellcode 4.2: QR-Code gefunden

Nun wird geprüft, ob der gescannte QR-Code ein gültiges Format hat, andernfalls wird er abgelehnt. Der Benutzer wird darüber in einer Snackbar informiert.

```

1 try {
2     final Map<String, dynamic> data = jsonDecode(code);
3
4     // Ueberpruefen, ob der Namespace 'payero' ist
5     if (data['namespace'] != 'payero') {
6         ScaffoldMessenger.of(context).showSnackBar(
7             const Snackbar(content: Text('Ungueltiger QR-Code')),
8         );
9         return; // Beenden der Funktion, wenn der Namespace nicht stimmt
10    }
11
12    // Ueberpruefen, ob der 'amount' zu einem double konvertiert werden
13    // kann
14    final double? amount = double.tryParse(data['amount'].toString());
15    if (amount == null) {
16        ScaffoldMessenger.of(context).showSnackBar(
17            const Snackbar(content: Text('Ungueltiger QR-Code')),
18        );
19        return; // Beenden der Funktion, wenn der amount ungueltig ist
20    }
21
22    final String formattedAmount = amount.toStringAsFixed(2);
23
24    ScaffoldMessenger.of(context).removeCurrentSnackBar();
25
26    // Weitermachen mit gueltigen Daten
27    _screenOpened = true;
28    Navigator.push(
29        context,
30        MaterialPageRoute(
31            builder: (context) => FoundCodeScreen(
32                screenClosed: _screenWasClosed,
33                userId: userId,
34                value: formattedAmount,
35            ),
36        ).then((_) => _screenWasClosed());
37    } catch (e) {

```

```
38     debugPrint("Error while scanning: ${e.toString()}");
39     // Generische Fehlermeldung anzeigen
40     ScaffoldMessenger.of(context).showSnackBar(
41         const SnackBar(content: Text('Ungültiger QR-Code')),
42     );
43 }
```

Quellcode 4.3: QR-Code Check

Das gültige Format eines QR-Codes ist JSON und sieht wie folgt aus:

```
1 {
2     "namespace": "payero",
3     "userID": "user1",
4     "amount": "3.00"
5 }
```

Quellcode 4.4: Gültiger QR-Code

Der entsprechende QR-Code sieht dann so aus:



Abbildung 4.12: Gültiger QR-Code

Bei der Anzeige des Betrags innerhalb der App wird der Betrag auf jeder Seite nur mit zwei Nachkommastellen angezeigt. Dies ist unabhängig davon, ob der „amount“ im QR-Code Nachkommastellen enthält oder nicht, und dient ausschließlich kosmetischen Zwecken, verbessert aber aufgrund der immer gleichen Formatierung das Nutzererlebnis. Dies gilt auch für die QR-Codes, die im Hauptmenü gescannt werden können. Um zu verhindern, dass immer weiter gescannt wird, wenn ein QR-Code gefunden wurde, wird über den Parameter „\_screenOpened“ ein Bool-Flag gesetzt. Danach wird bei einem gültigen QR-Code auf eine neue Seite gewechselt. Diese zeigt den zu zahlenden Betrag und einen Button, mit dem bezahlt werden kann. Ein Klick auf den Button öffnet das BrainTree Dropin-UI.

## 4.4 Bezahlintegration mit Braintree

Die Zahlungsmethode kann im BrainTree Dropin-UI ausgewählt werden. Derzeit werden PayPal, Kreditkarte (VISA, Mastercard) und GooglePay angeboten. ApplePay ist ebenfalls implementiert, jedoch muss eine gültige Merchant-ID angegeben werden, um die Funktion zu aktivieren. Diese Merchant-ID ist nur mit einer Mitgliedschaft im Apple Developer Programm erhältlich. Darüber hinaus wird ausschließlich der Sandbox-Account von Braintree genutzt, da es als Privatperson ohne gültige Umsatzsteuer-Identifikationsnummer und wahrheitsgemäße Angaben zu den Eigentumsverhältnissen oder der Stellung im Unternehmen gemäß §§10 bis 17 Geldwäschegegesetz nicht möglich ist, einen Produktiv-Account zu erhalten. Ist man im Besitz eines gültigen Braintree-Produktivaccounts, muss lediglich der „tokenizationKey“ ausgetauscht werden, um die Anwendung produktiv zu schalten.

```

1  class _FoundCodeScreenState extends State<FoundCodeScreen> {
2      String currency = "EUR";
3
4      void startBraintreeCheckout() async {
5          var request = BraintreeDropInRequest(
6              tokenizationKey:
7                  'sandbox_d53t3dpq_8fxhdjy2rnd33mtm', // Ersetzen mit echtem
8                  tokenizationKey aus BrainTree Account
9              collectDeviceData: true,
10             paypalRequest: BraintreePayPalRequest(
11                 amount: widget.value,
12                 displayName: 'Payero',
13                 currencyCode: currency,
14             ),
15             googlePaymentRequest: BraintreeGooglePaymentRequest(
16                 totalPrice: widget.value,
17                 currencyCode: currency,
18                 billingAddressRequired: false,
19             ),
20             applePayRequest: BraintreeApplePayRequest(
21                 paymentSummaryItems: [
22                     ApplePaySummaryItem(
23                         label: 'Payero',
24                         amount: double.parse(widget.value),
25                         type: ApplePaySummaryItemType.final_)
26                 ],
27                 displayName: 'Payero',
28                 currencyCode: currency,
29                 countryCode: 'DE',
30                 merchantIdentifier:
31                     'merchant.com.example.mobile_computing_payment_app', //
32                     Ersetzen mit echtem Apple Pay Merchant Identifier

```

```

31     supportedNetworks: [
32         ApplePaySupportedNetworks.visa,
33         ApplePaySupportedNetworks.masterCard,
34         ApplePaySupportedNetworks.amex
35     ],
36 ),
37 // Weitere Optionen und Konfigurationen
38 );

```

Quellcode 4.5: BrainTree Bezahlmethoden

Bei der Auswahl der Zahlungsmethode wird geprüft, ob eine Internetverbindung besteht. Erst dann erfolgt die vollständige Bezahlung und die Übermittlung der Zahlungsdaten an den Backend-Server. Auch hier wird der Benutzer in einer Snackbar über Fehler oder das weitere Vorgehen informiert. Besteht also keine Internetverbindung oder bricht der Benutzer den Bezahlvorgang ab, um z.B. noch die Bezahlmethode zu wechseln, wird er in einer Snackbar darüber informiert. Bezahlt der Benutzer nun, wird er in einer Snackbar über die Verarbeitung des Bezahlvorgangs mit seiner jeweiligen Bezahlmethode informiert. Wenn der Bezahlvorgang erfolgreich war, wird der Benutzer auf eine Abschlussseite weitergeleitet.

```

1 try {
2     BraintreeDropInResult? result = await BraintreeDropIn.start(request)
3         ;
4     if (result != null) {
5         // print(
6         //     'Zahlung erfolgreich: ${result.paymentMethodNonce.typeLabel
7         // } ${result.paymentMethodNonce.description}');
8         ScaffoldMessenger.of(context).showSnackBar(
9             SnackBar(
10                 content: Text(
11                     'Bearbeite Zahlungsvorgang mit ${result.
12                         paymentMethodNonce.typeLabel} ${result.
13                         paymentMethodNonce.description}'),
14                 duration: const Duration(seconds: 2)),
15         );
16
17         // ScaffoldMessenger.of(context).showSnackBar(SnackBar(
18         //     content: Text(
19         //         'Zahlung erfolgreich: ${result.paymentMethodNonce.
20             typeLabel} ${result.paymentMethodNonce.description}'),
21         //     duration: const Duration(seconds: 2),
22         // ));
23
24         await Future.delayed(const Duration(seconds: 3));
25
26         Navigator.push(context,

```

```

22     MaterialPageRoute(builder: (_) => EndScreen(value: widget.
23                               value)));
24
25
26     try {
27         if (widget.userId == "") {
28             throw Exception("userId is empty");
29         }
30
31         final response = await http.post(
32             Uri.parse("$SERVER_URL/${widget.userId}/transact"),
33             body: jsonEncode(<String, String>{
34                 "receiver": "mobile_computing_payment_app",
35                 "amount": '${double.parse(widget.value) * -1}',
36                 "currency": currency,
37                 "message": "${result.paymentMethodNonce.description}"
38             }));
39
40         debugPrint("response: " + response.body);
41
42         if (response.statusCode != 200) {
43             throw Exception('Failed to save transaction.' + response.body)
44             ;
45         }
46         } catch (e) {
47             debugPrint(e.toString());
48         }
49     } else {
50         ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
51             content: Text('Zahlungsvorgang abgebrochen'),
52             duration: Duration(seconds: 2)));
53     }
54 } catch (e) {
55     ScaffoldMessenger.of(context).showSnackBar(SnackBar(
56         content: Text(
57             'Zahlungsvorgang abgebrochen. Stellen Sie sicher, dass Sie
58             mit dem Internet verbunden sind. \n\n Fehler: $e')));
59 }
58 }
```

Quellcode 4.6: Durchführung des Bezahlvorganges

Auf dieser Abschlussseite „end\_screen.dart“ wird der Benutzer über den erfolgreich bezahlten Betrag informiert. Anschließend kann er den Vorgang mit dem Button „Fertig“ abschließen, um wieder ins Hauptmenü zu gelangen. Ein neuer Bezahlvorgang kann gestartet werden. Der „Zurück“ Button ist auf dieser Seite extra deaktiviert, damit ein Benutzer

nicht aus Versehen wieder auf der Bezahlseite landet und den Betrag erneut bezahlt. Dazu gehört auch der Hardware-Button unter Android, der auf dieser Seite deaktiviert ist. Ansonsten wäre auch bei ausschließlicher Deaktivierung in der Software ein Zurück über diesen Hardware-Button möglich. Dies geschieht bei Android mit „onWillPop“.

```
1 class EndScreen extends StatelessWidget {
2   final String value;
3
4   const EndScreen({Key? key, required this.value}) : super(key: key);
5
6   @override
7   Widget build(BuildContext context) {
8     return WillPopScope(
9       onWillPop: () async => false, // Verhindert das Zurueckgehen
10      child: Scaffold(
11        appBar: const PayeroHeader(
12          showBackButton: false,
13        ),
14        body: Stack(children: [
15          Column(children: [
16            Expanded(
17              child: Container(
18                padding: const EdgeInsets.only(
19                  bottom: 30, top: 80, left: 30, right: 30),
20                child: Column(
21                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
22                  children: [
23                    const Text('Zahlung erfolgreich ausgefuehrt',
24                      style: TextStyle(fontSize: 20)),
25                    Text('\$ value \euro',
26                      style: const TextStyle(
27                        fontSize: 40, fontWeight: FontWeight.bold)),
28                    SvgPicture.asset(
29                      'assets/images/payero-illustration.svg',
30                      fit: BoxFit.fitWidth,
31                      alignment: Alignment.center,
32                      // height: 50.0,
33                      )
34                    ],
35                  ),
36                ),
37              ),
38              Container(
39                margin:
40                  const EdgeInsets.only(left: 30, right: 30,
41                                bottom: 30),
42              )
43            ],
44          ),
45        ),
46      ),
47    );
48  }
49}
```

```

41         child: Column(children: [
42             PayeroButton(
43                 text: "Fertig",
44                 onClick: () => {
45                     Navigator.push(
46                         context,
47                         MaterialPageRoute(
48                             builder: (_) => const MainScreen()
49                             ,
50                         )
51                     ) ,
52                 ])),
53             ]));
54         }
55     }

```

Quellcode 4.7: EndScreen bei erfolgreichem Bezahlvorgang

## 4.5 Benutzer-Authentifizierung

Beim Starten der App wird im SplashScreen geprüft, ob eine user\_id in den SharedPreferences abgelegt wurde. Sofern der Wert existiert, wird anschließend zum MainScreen weitergeleitet. Ansonsten wird der RegistrationScreen ausgespielt. Im RegistrationScreen existieren zwei Eingabefelder für den Spitznamen und eine E-Mail-Adresse. Diese werden wie im folgenden Kapitel beschrieben an das Backend übergeben. Die Eingabefelder wurden mithilfe des TextFormField-Widgets gebaut, mit denen auch eine validator()-Funktion definiert werden kann. Über diese wird sichergestellt, dass beide Felder ausgefüllt sind und die E-Mail-Adresse ein gültiges Format hat. Nach dem Betätigen des Buttons wird der Post-Request /register ausgeführt und die vom Backend zurückgegebene UUID in den SharedPreferences gespeichert. Anschließend erfolgt eine Überleitung zum MainScreen.

## 4.6 Backend

Zur Realisierung der Transaktionshistorie wurde ein einfaches Backend mit Node.js entwickelt. Dieses kann später erweitert werden und die Grundlage für die Umsetzung einer Authentifizierung und weiterer Features darstellen. Es wird das Framework Next.js eingesetzt, da dies für die Autoren keine große Einarbeitungszeit erforderte. Mithilfe von

diesem können Applikation mit React.js umgesetzt werden, aber auch REST-API's implementiert werden. Als Datenbank wird eine einfache SQLite-Datenbank verwendet, da diese als Datei gesichert werden kann. Diese wird mithilfe des Frameworks Prisma angebunden, welches auf dem Konzept des Object-Relational-Mappings basiert. Das heißt, dass mithilfe von Prisma eine Datenbankstruktur in Form der Datei ./backend/prisma/schema.prisma definiert wird. Prisma erstellt daraufhin eine Tabellenstruktur und sichert diese in der SQLite-Datenbank ./backend/prisma/database.db. Der Prisma-Client wird dann innerhalb der Next.js-Anwendung aufgerufen und erlaubt die Abfrage der Datenbank in einer objekt-notierten Form, die ebenfalls bereits alle Relationen enthält. Zur Umsetzung des Datenbank-Schemas wurde das in Abbildung 4.13 dargestellte Entity-Relationship-Diagramm in Chen-Notation entworfen. Ein Benutzer kann mehrere Transaktionen empfangen haben, aber auch durchgeführt haben. Eine Transaktion hat immer einen Empfänger und einen Absender.

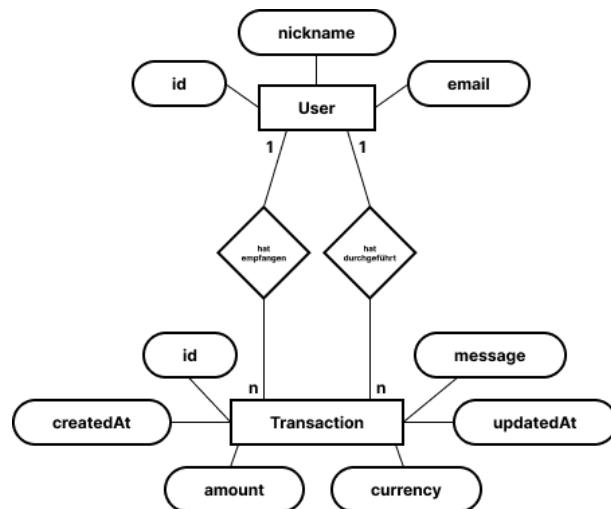


Abbildung 4.13: Darstellung des Entity-Relationship-Diagramms für die Datenbank

Als kleine Abweichung von diesem ER-Diagramm wird das eigentliche Schema so umgesetzt, dass nur die Relation Absender-Transaction abgebildet wird. Der Empfänger einer Transaktion wird als freies String-Feld umgesetzt, da dieser so frei definiert werden kann. Solange die Transaktionen im Rahmen des Projekts einen fixen Empfänger erreichen, ist man so flexibler in der Handhabung und kann einen Dummy-Empfänger wie z.B. die E-Mail-Adresse des Braintree-Accounts angeben.

Das Backend wird einem Docker-Container gehostet, welcher mithilfe von Docker-Compose deployed werden kann.

In Abbildung 4.14 wird dargestellt, welche API-Endpunkte umgesetzt werden sollten:

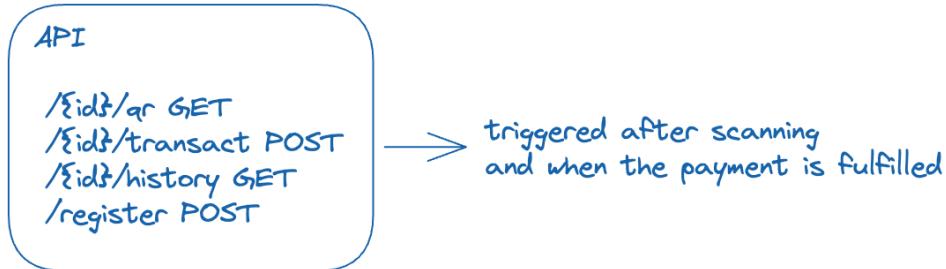


Abbildung 4.14: Darstellung der API-Endpunkte für das Backend

Zunächst ist ein Endpunkt zur Registrierung vorgesehen, welcher dazu dient den Benutzer mithilfe eines Nicknames und einer E-Mail-Adresse zu registrieren. Sofern ein Benutzer bereits existiert wird ein weiterer in der Datenbank angelegt, da hier keine Überprüfung stattfindet. Dieser bekommt eine individuelle UUID zugeteilt und wird anschließend in der Datenbank gesichert.

Es wurde an dieser Stelle bewusst auf eine reale Authentifizierung mittels bspw. Passwort und Session-ID oder JWT verzichtet, da so im Rahmen des Projekts auch auf das clientseitige Session-Handling verzichtet werden konnte. So konnte an anderen Bestandteilen der App gearbeitet werden.

Die UUID wird in den SharedPreferences der App gesichert und dient der Abfrage der weiteren Endpunkte. Über den Endpunkt /uuid/history kann die Transaktionshistorie abgefragt werden. Diese enthält alle Informationen über vergangene Transaktionen wie bspw. Betrag, Währung, Zeitpunkt und Empfänger. Eine Transaktion wird clientseitig durch das Aufrufen des POST-Endpunkts /uuid/transact gesichert, sofern eine Zahlung komplett abgeschlossen wurde. An diesen Endpunkt werden die o.g. Daten übergeben und diese so in der Datenbank gesichert.

Der weitere Endpunkt /uuid/qr dient der Generierung eines individuellen QR-Codes, welcher jeweils für einen Benutzer generiert wird. Der QR-Code enthält einen vordefinierten Betrag, dessen die Transaktionen gestartet werden können. Als Weiterentwicklung der Applikation könnte so später eine echte Peer-To-Peer-Überweisung ermöglicht werden, sodass die scannende Person an denjenigen überweist, der den QR-Code mit seiner kodierten ID präsentiert.

Der QR-Code kodiert folgende Informationen in Form eines JSON-Objekts:

```
1 { namespace: "payero", id:{UUID}, amount: "2.0" }
```

Quellcode 4.8: Vom Backend generierte Notation für den QR-Code

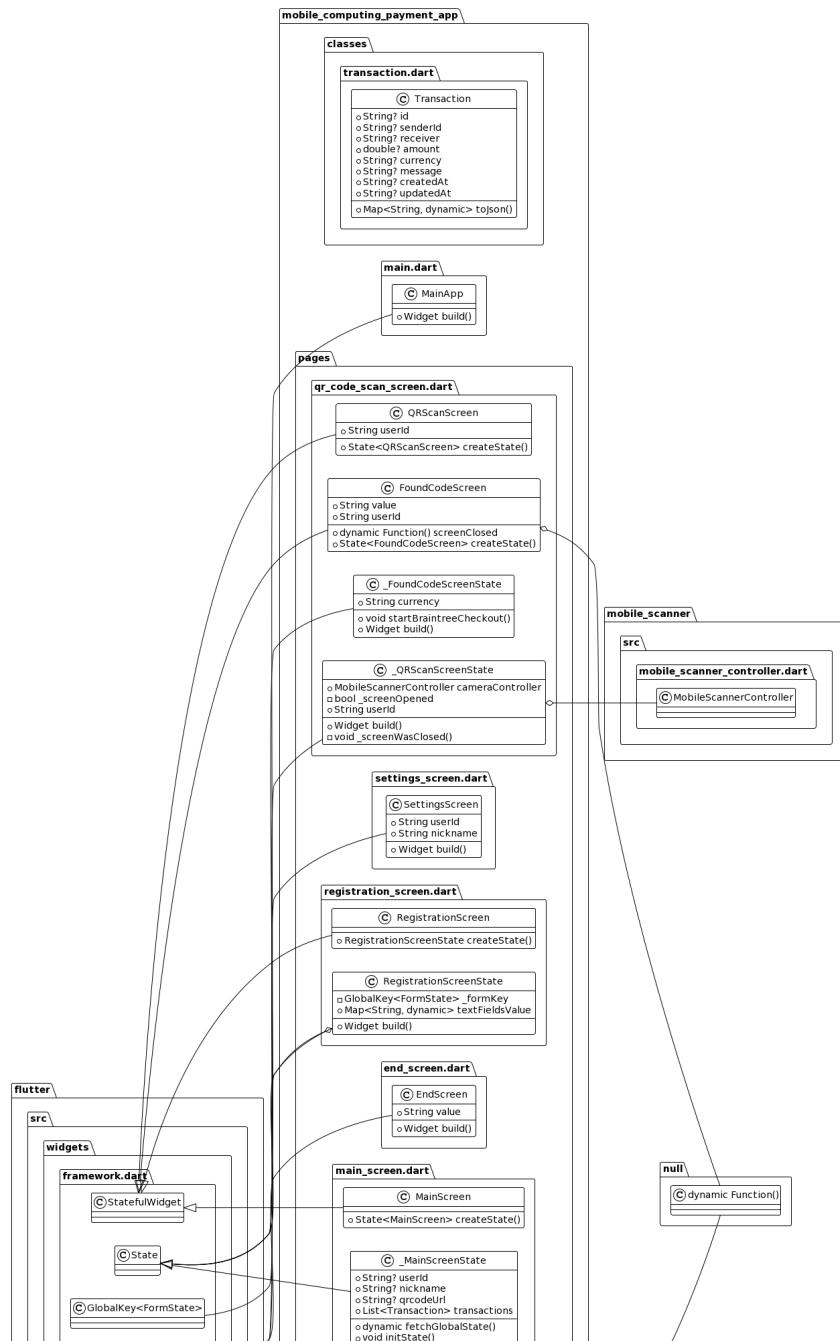
Mithilfe des Namespace-Attributes kann geprüft werden, ob der QR-Code für Payero vorgesehen ist. Ebenfalls enthält der QR-Code die Benutzer-UUID und den Betrag als Gleitkommazahl.

#### **4.6.1 Sicherheit**

Für eine produktive Nutzung der App sollte eine echte Authentifizierung über das Backend implementiert werden. Ebenso sollte sichergestellt werden, dass die QR-Codes nach der Erstellung nicht manipuliert werden können. Dies könnte etwa durch ein Public-Key-Verschlüsselungsverfahren sichergestellt werden. Als Implementierung könnten etwa JSON-Web-Tokens genutzt werden, die auf diesem Verfahren basieren. Ein weiterer Vorteil würde darin liegen, dass der QR-Code nicht beim Scannen mithilfe einer anderen App auslesbar wäre.

# 5 Klassendiagramm

Mithilfe des Tools PlantUML konnte folgendes Klassendiagramm generiert werden. Dieses zeigt in Abbildung 5.1 die Projektstruktur der entwickelten App.



## 5 Klassendiagramm

---

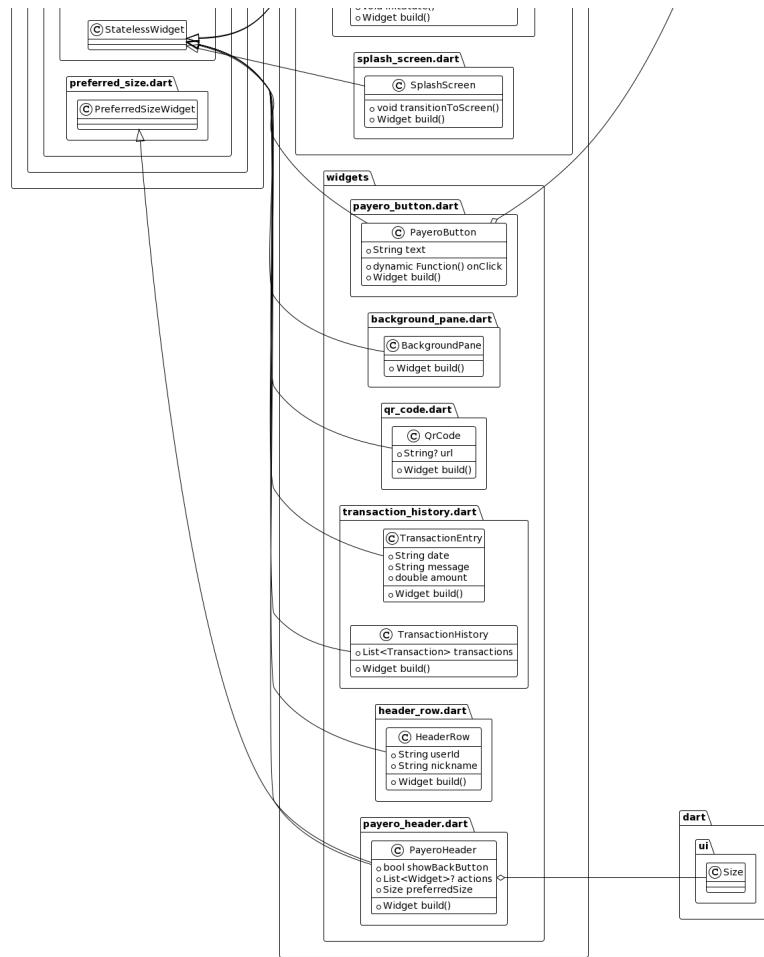


Abbildung 5.1: Mithilfe von PlantUML erstelltes Klassendiagramm

# 6 Testing

Umfangreiche Tests wurden durchgeführt, um sicherzustellen, dass die Anwendung auf den verschiedenen Plattformen wie erwartet funktioniert. Die Tests wurden auf verschiedenen Emulatoren und Endgeräten durchgeführt. Die folgenden Tests wurden durchgeführt:

- Regressionstests
- Funktionsprüfung
- Integrationstests
- Systemtests
- Leistungsprüfung
- GUI-Tests
- Abnahmeprüfung

Während der Entwicklung wurden die oben genannten Tests wiederholt durchgeführt. Durch Funktions- und Integrationstests wurde immer wieder sichergestellt, dass das Implementierte auch funktioniert. Dazu wurden immer wieder die Emulatoren und auch die realen Endgeräte eingesetzt. Durch die verschiedenen Softwareversionen wurde die Chance Fehler zu finden erhöht. Im Anschluss wurden immer wieder Systemtests durchgeführt, um feststellen zu können, ob weitere Funktionen nicht mehr wie geplant funktionieren. Dabei konnten Fehler gefunden werden, wie z.B. dass PayPal unter iOS problemlos funktionierte, unter Android aber nicht. Der Fehler konnte durch eine Änderung in der build.gradle behoben werden. Dazu wurde die applicationId auf eine com.example... URL geändert und alle Unterstriche entfernt. Damit funktionierte der PayPal Bezahlvorgang unter Android wieder. Zusätzlich wurden regelmäßig GUI Tests durchgeführt. Damit wurde sichergestellt, dass die Anzeigen und GUI Elemente auch auf unterschiedlich großen Geräten gut aussehen und bedienbar sind. Ein einfacher Performancetest wurde ebenfalls regelmäßig durchgeführt. Dieser diente lediglich dazu, festzustellen, ob es Probleme und lange Ladezeiten bei Funktionen oder Seiten gab. Dazu wurden die Zeiten gemessen, um lange Wartezeiten zu erkennen. Die oben beschriebenen Tests wurden während der Entwicklung mehrmals kontinuierlich durchgeführt. Ganz am Ende, kurz vor der Fertigstellung und Präsentation, gab es noch einen größeren Abnahmetest. Dabei wurden alle oben genannten Tests durchgeführt, um sicherzustellen, dass alles korrekt funktioniert und auch vom Design her

gut aussieht. Dabei wurde festgestellt, dass der Zugriff auf den Backend-Server im „eduroam“ WLAN blockiert wurde. Dies verhinderte den Registrierungsprozess zu Beginn und machte die App somit unbrauchbar. Durch den Wechsel zu einem anderen WLAN konnte das Problem behoben werden. Somit konnte sichergestellt werden, dass die Anwendung zum Zeitpunkt der Präsentation wie erwartet funktionierte.

# 7 Ausblick und Weiterentwicklung

Die App kann noch beliebig weiterentwickelt und unter bestimmten Voraussetzungen auch produktiv genutzt werden. Denkbar wäre etwa die Umsetzung der noch nicht umgesetzten Funktionen, insbesondere echtes Peer-To-Peer-Payment. Mittels Braintree kann nur eine Zahlung an den Händler erfolgen, aber es könnte eine Transaktion im Backend über andere Zahlungsdienstleister ausgelöst werden. Um das volle Potenzial der App nutzen zu können, bräuchte man aber definitiv einen BrainTree Product Account. Außerdem bräuchte man noch einen Apple Developer Account, um ApplePay anbieten zu können. Mit mehr Zeit wäre es auch möglich gewesen, weitere Features zu implementieren. Dazu gehört auf jeden Fall ein guter Darkmode, der auch dem Design der Corporate Identity von „Payero“ entspricht.

Wünschenswert wäre auch die Unterstützung weiterer Sprachen, wie etwa Englisch, Französisch und Spanisch. Die Braintree-Integration unterstützt bereits zahlreiche weitere Sprachen, so könnte der gesamte europäische Markt abgedeckt werden. Dabei ist allerdings zu beachten, in welchen Ländern BrainTree und PayPal jeweils operieren, da nicht alle Länder weltweit unterstützt werden.

Ein weiteres benutzerfreundliches Feature wäre die Implementierung des Scannens von Kreditkarten. Mit diesem Feature kann man seine Kreditkarte zum Bezahlen sicher einscannen, ohne die Nummern mühsam eintippen zu müssen. Dies erhöht den Benutzerkomfort. Dies kann innerhalb von BrainTree realisiert werden. Dies liegt daran, dass PayPal das ehemalige Start-up card.io übernommen hat, das die Technologie dafür besaß. Es ist auch möglich, den verwendeten Backend-Server so zu erweitern, dass er Client-Tokens generiert. Diese sind pro Transaktion einmalig und ermöglichen es, Transaktionen eines Nutzers sicher nachzuvollziehen. Dazu muss dann kein TransactionKey mehr in den Code eingebaut werden. Vor allem bei der Implementierung des 3D Secure 2.0 Verfahrens für Kreditkarten ist dies sehr hilfreich. Dieses Verfahren ist vor allem im produktiven Einsatz essenziell und erhöht die Akzeptanz von Kreditkarten. Um das Design der App noch schöner zu gestalten, kann man anstelle des BrainTree Dropin UI ein eigenes Design für die jeweiligen UI Elemente erstellen. Dies kann in BrainTree mit Hosted Fields realisiert werden, die individuell generiert und angepasst werden können. Dieser Aufwand ermöglicht eine noch bessere Umsetzung des eigenen Designs auf Basis einer Corporate Identity.

# 8 Verzeichnisse

Das Projekt ist auf [GitHub](#) verfügbar und besteht aus mehreren Verzeichnissen. Die Projektstruktur sieht dabei wie folgt aus:

- backend
- documentation
- screenshots
- sourcecode
- wireframes

Unter „backend“ befindet sich der Quellcode für die Installation eines Backend-Servers zur Verwaltung der Datenbank. In „documentation“ befindet sich die Dokumentation im LaTeX-Format. In „screenshots“ befinden sich Screenshots der Anwendung. In „sourcecode“ befindet sich der Quellcode, um das Projekt zu bauen. In „wireframes“ befinden sich die Wireframes und High Fidelity Bilder des Designs der Anwendung.

# **9 Copyright**

© 2023 Calvin Reibenspieß und Florian Brändle.

Hiermit gestatten wir Prof. Dr.-Ing. Dipl.-Inf. Kai Becher gemäß §31 Urheberrechtsgesetz die nicht kommerzielle Nutzung der Anwendung, des Quellcodes und aller beigefügten Dokumente.

# Änderungshistorie

## 1.0.1 (03.12.2023)

Es wurde ein Fehler im Backend beim Registrierungsprozess behoben, wodurch dieser innerhalb der App nicht abgeschlossen werden konnte.

## 1.0.0 (29.11.2023)

Die App enthält alle für die Abgabe vorgesehen Features und unterstützt das Generieren von QR-Codes, das Einscannen und die Überweisung eines fixen Betrags mittels PayPal, Kreditkarte und Google Pay.

# Abbildungsverzeichnis

1.1	Einstellungen des Android-Emulators von Calvin Reibenspieß . . . . .	3
1.2	Einstellungen des Android-Emulators von Florian Brändle Pixel 3a . . . . .	4
1.3	Einstellungen des Android-Emulators von Florian Brändle Pixel 2 . . . . .	5
3.1	Darstellung des Logos auf hellen und dunklen Hintergründen, sowie in Kombination mit Schriftzug . . . . .	9
3.2	Darstellung der angefertigten Wireframes . . . . .	11
4.1	SplashScreen . . . . .	12
4.2	Registrierungsvorgang . . . . .	13
4.3	Hauptmenü . . . . .	14
4.4	Einstellungen . . . . .	15
4.5	Ungültiger QR-Code . . . . .	16
4.6	Bestätigungsseite . . . . .	17
4.7	Bezahlmethoden . . . . .	18
4.8	Verarbeitung des Bezahlvorganges . . . . .	19
4.9	Erfolgreicher Bezahlvorgang . . . . .	20
4.10	Zahlung in der Transaktionshistorie sichtbar . . . . .	21
4.11	Darstellung der Transaktionshistorie in verschiedenen Fällen, mit einer Transaktion, ohne Textkürzung, und mit mehreren Transaktionen . . . . .	22
4.12	Gültiger QR-Code . . . . .	25
4.13	Darstellung des Entity-Relationship-Diagramms für die Datenbank . . . . .	31
4.14	Darstellung der API-Endpunkte für das Backend . . . . .	32
5.1	Mithilfe von PlantUML erstelltes Klassendiagramm . . . . .	35

# **Quellcodeverzeichnis**

4.1	Kamerasteuerung	22
4.2	QR-Code gefunden	23
4.3	QR-Code Check	24
4.4	Gültiger QR-Code	25
4.5	BrainTree Bezahlmethoden	26
4.6	Durchführung des Bezahlvorganges	27
4.7	EndScreen bei erfolgreichem Bezahlvorgang	29
4.8	Vom Backend generierte Notation für den QR-Code	32