# Dokumentation:

## Welche Softwarevoraussetzungen werden benötigt (mit Versionen)

## Systemvoraussetzungen

**Betriebssystem**
Windows 10 oder höher
Getestet unter: Windows 10.0.26100.4351 (Windows 11 24H2 Build)

## Flutter und Dart

**Flutter SDK**
Version: 3.29.3
Channel: `stable`
Git-Revision: `ea121f8859`
Datum: 11. April 2025

**Dart SDK**
Version: 3.7.2

**Flutter DevTools**
Version: 2.42.3

## Abhängigkeiten aus pubspec.yaml

Diese Pakete werden im Projekt verwendet und müssen durch `flutter pub get` verfügbar sein:

- `flutter` (SDK)
- `logger: ^2.5.0`
- `supabase_flutter: ^2.0.0`
- `shared_preferences: ^2.5.3`
- `uuid: ^4.0.0`
- `http: ^0.13.6`
- `cupertino_icons: ^1.0.8`
- `flutter_lints: ^5.0.0` (nur für Entwicklung)

## Methoden

### `Future<void> upsertUser(String uid, String name)`

Sendet einen PUT-Request, um einen Benutzer zu erstellen oder zu aktualisieren. Falls der Benutzer ( `uid` ) schon existiert, wird sein Name überschrieben. Andernfalls wird ein neuer Benutzer mit der `uid` erstellt. Erfolgreiche Antworten haben Statuscode 200 oder 201.

### `Future<String> createGame()`

Erzeugt ein neues Spiel durch einen POST-Request. Der Server gibt eine neue Spiel-ID ( `gid` ) zurück. Diese ID wird vom Client verwendet, um weitere Spielaktionen dem richtigen Spiel zuzuordnen.

### `Future<bool> joinGame(String gid)`

Versucht, einem bestehenden Spiel mit der gegebenen Spiel-ID ( `gid` ) beizutreten. Der Server gibt 200 zurück, wenn der Beitritt erfolgreich war, 404 falls das Spiel nicht existiert. Bei anderen Fehlern wird eine Exception geworfen.

### `Future<List<Spieler>> loadPlayers(String gid)`

Lädt alle Spieler, die bereits einem Spiel beigetreten sind. Gibt eine Liste von `Spieler` -Objekten zurück, welche jeweils `uid` , `name` und `playernumber` enthalten. Die Daten kommen vom Endpoint `/games/{gid}/players` .

### `Future<List<Jasskarte>> getAllCards()`

Lädt alle verfügbaren Spielkarten aus der Datenbank. Die Antwort enthält eine Liste von Kartenobjekten, die zu `Jasskarte` -Instanzen umgewandelt werden. Jede Karte enthält Symbol, Typ, ID und Pfad zur Bilddatei.

### `Future<Jasskarte> getCardByCid(String cid)`

Lädt eine einzelne Karte anhand ihrer eindeutigen Karten-ID ( `cid` ). Die Rückgabe ist eine `Jasskarte` , basierend auf den vom Server gelieferten Daten.

### `Future<void> shuffleCards(String gid)`

Fordert den Server auf, die Karten für ein bestimmtes Spiel zu mischen. Der Endpunkt `/games/{gid}/cards/shuffle` führt die Logik im Backend aus. Diese Funktion hat keinen Rückgabewert, wirft aber bei Fehler eine Exception.

### `Future<String> getCurrentRoundId(String gid)`

Lädt die aktuelle Runden-ID ( `rid` ) eines Spiels. Diese ID wird für alle rundenbezogenen API-Aufrufe (z. B. Spielzüge, Gewinner) benötigt. Gibt einen leeren String zurück, wenn keine Runde aktiv ist.

### `Future<void> startNewRound(String gid, int whichround)`

Startet eine neue Spielrunde für das angegebene Spiel. Die Rundennummer ( `whichround` ) muss angegeben werden. Die Methode ruft den Endpunkt `/games/{gid}/rounds` mit JSON-Daten auf.

### `Future<void> addPlayInRound(String rid, String uid, String cid)`

Speichert einen Spielzug: welcher Spieler ( `uid` ) welche Karte ( `cid` ) in einer bestimmten Runde ( `rid` ) gespielt hat. Die Daten werden per POST an den Server geschickt.

### `Future<List<Jasskarte>> getPlayedCards(String rid)`

Gibt alle Karten zurück, die in einer Runde bereits gespielt wurden. Die Karten werden als Liste zurückgegeben und enthalten dieselben Informationen wie bei `getAllCards()` .

### `Future<String?> getFirstCardCid(String rid)`

Fragt ab, welche Karte in einer Runde zuerst gespielt wurde. Wird oft für Spielregeln oder zum Erkennen der angespielten Farbe benötigt.

### `Future<String> getWhosTurn(String rid)`

Gibt die `uid` des Spielers zurück, der laut Server gerade am Zug ist. Diese Information wird laufend benötigt, um die Spielreihenfolge korrekt einzuhalten.

### `Future<void> updateWhosTurn(String rid, String uid)`

Aktualisiert auf dem Server, welcher Spieler als Nächstes an der Reihe ist. Dies wird nach jedem gültigen Spielzug aufgerufen.

### `Future<void> updateTrumpf(String gid, String symbol)`

Setzt das Trumpf-Symbol eines Spiels (z. B. "Herz", "Schelle", "Eichel"). Dieses Symbol wird für die Spiellogik beim Vergleichen von Karten verwendet.

**`Future<int> getUrPlayernumber(String uid, String gid)`**

Liefert die Spielerposition ( `playernumber` ) eines Benutzers innerhalb eines bestimmten Spiels. Wird z. B. gebraucht, um die Zugreihenfolge zu bestimmen.

**`Future<String> getNextPlayerUid(String gid, int playernumber)`**

Fragt den Server, wer nach dem Spieler mit der angegebenen Nummer ( `playernumber` ) an der Reihe ist. Die Rückgabe ist die `uid` des nächsten Spielers.

**`Future<List<Jasskarte>> getUrCards(String gid, String uid)`**

Lädt die Karten, die einem bestimmten Spieler in einem Spiel zugewiesen sind. Wird typischerweise einmal zu Beginn der Runde verwendet.

**`Future<int> savePointsForUsers(String gid, Map<String, dynamic> body)`**

Speichert die erzielten Punkte nach einer Runde. Die Methode akzeptiert ein JSON-Objekt, das UID und Punktestand enthält, und gibt die neue Gesamtpunktzahl zurück.

**`Future<String> determineWinningCard(String gid, List<Jasskarte> cards)`**

Fragt den Server, welche der gespielten Karten die Runde gewinnt. Der Server berücksichtigt Trumpf und Spielregeln. Die Rückgabe ist die `uid` des Spielers mit der besten Karte.

**`Future<int> getWhichRound(String gid)`**

Gibt zurück, die wievielte Runde aktuell läuft. Nützlich für Anzeigen oder Rundenwechsel.

**`Future<void> updateWinner(String rid, String uid)`**

Speichert den Runden-Gewinner auf dem Server. Die Methode wird aufgerufen, nachdem die Gewinnkarte bestimmt wurde.

# Mögliche Probleme und ihre Lösung:

Wenn ein Spieler keine Internetverbindung mehr hat könnte das Spiel pausieren und warten bis er wieder da ist.

Die Rooms werden nicht gelöscht solang sie nicht ganz gefüllt sind und bleiben unendlich. -> Man könnte schauen ob überhaupt noch jemand im Room ist der wartet und wenn nicht ihn löschen.

# Wie wurde die Software getestet?

Unit-Tests

Meistens 4 Terminals gestartet, Room erstellt und alle dem selben Spiel gejoined.