

Dokumentation

CoLivingPilot (CoLiPi)

Programmierung Mobiler Endgeräte
WISE 23/24



Kevin Matz
Florian Renz
Dario Daßler
Hendrik Lendeckel

Inhaltsverzeichnis

1	Idee	1
1.1	Problemstellung	1
1.2	App Idee.....	1
1.3	Grundfunktionen.....	1
1.3.1	Registration/Login und Erstellung der WG	1
1.3.2	Aufgabenplanung	2
1.3.3	Einkaufsliste.....	2
1.3.4	WG-Bereich / Highscore	2
2	Backend.....	3
2.1	ER-Modell	3
2.2	Tech-Stack	3
2.3	REST-API	5
3	Android App	6
3.1	ER-Model	6
3.2	Repository.....	6
3.3	Datenbankzugriff.....	6
3.4	Refresh-Methode	6
3.5	Netzwerk.....	7
3.6	Aktualisierung und Synchronisation.....	7
3.7	Login, Registrierung und WG-Beitritt	7
3.8	Einkaufsliste	8
3.8.1	Fragmente	8
3.8.2	ViewModel	8
3.8.3	Benutzerinteraktion.....	8
3.9	Aufgabenliste	9
3.10	WG-Einstellungen.....	9
4	Testbare Funktionalitäten der App	9

4.1	Registrierung Benutzers, anlegen einer WG und Login	9
4.2	Einer bestehenden WG beitreten.....	10
4.3	Aufgaben erstellen, bearbeiten und abhaken.....	11
4.4	Einkaufsliste erstellen, bearbeiten und abhaken	11
4.5	Bearbeitung der WG	12
4.6	Einstellungen.....	13

1 Idee

1.1 Problemstellung

Das Zusammenleben in einer Wohngemeinschaft birgt zahlreiche Vor- und Nachteile. Zu den Herausforderungen zählt insbesondere die gerechte Verteilung von Aufgaben unter den Bewohnern. Die Planung dieser regelmäßig wiederkehrenden Tätigkeiten gestaltet sich oft als anspruchsvoll, wenn sie auf herkömmliche, analoge Weise durchgeführt wird. Zudem ist die Effektivität eines solchen Plans maßgeblich von der konsequenten Umsetzung abhängig. Dieses Problem beabsichtigen wir in unserem App-Prototypen zu adressieren.

1.2 App Idee

CoLivingPilot, kurz CoLiPi, ist eine App zur effizienten Organisation von Wohngemeinschaften (WG). Inspiriert von erfolgreichen Apps wie Flatastic und FlatUp!, erweitern wir das Konzept der Aufgabenverteilung unter den WG-Bewohnern um eine innovative Gamification-Komponente. Jede Aufgabe kann individuell einem Nutzer zugeordnet werden und ist mit einer bestimmten Anzahl von Punkten, im Folgenden als "Bierchen" bezeichnet, versehen. Nach erfolgreicher Erledigung einer Aufgabe werden diese Bierchen dem entsprechenden Nutzerkonto gutgeschrieben. Auf diese Weise entsteht ein WG-interner Highscore, der die aktive Beteiligung der Bewohner visualisiert.

Die Einlösung der gesammelten Bierchen gegen Realwelt-Belohnungen bleibt den Nutzern vorbehalten und ist nicht Bestandteil dieses Projektes. Mit CoLiPi streben wir an, die WG-Organisation durch eine spielerische Komponente zu erleichtern und die Motivation der Bewohner zur Übernahme von Aufgaben zu steigern.

1.3 Grundfunktionen

1.3.1 Registration/Login und Erstellung der WG

Die Anwendung ermöglicht Benutzern die Registrierung und die Erstellung einer Wohngemeinschaft (WG). Alternativ können sie sich mithilfe eines QR-Codes unkompliziert einer bestehenden WG anschließen. Nach erfolgreicher Registrierung haben die Benutzer die Möglichkeit, sich bequem in ihrer WG einzuloggen. Diese Funktion erleichtert den Zugang zu den gemeinsamen Wohnbereichen und fördert eine reibungslose Interaktion zwischen den WG-Mitgliedern.

1.3.2 Aufgabenplanung

Innerhalb der Wohngemeinschaft (WG) kann ein Benutzer im ersten Abschnitt Aufgaben erstellen. Zu den Details einer Aufgabe gehören der Titel, eine kurze Notiz, die Anzahl der Punkte bzw. "Bierchen" und die zuständige Person, die für die Ausführung verantwortlich ist. Die Benutzer haben die Möglichkeit, erstellte Aufgaben zu überprüfen, indem sie sie abhaken, und darüber hinaus besteht die Option, sie aus der Liste zu entfernen. Dieser Funktionsbereich ermöglicht eine strukturierte und effiziente Verwaltung der anstehenden Aufgaben in der WG.

1.3.3 Einkaufsliste

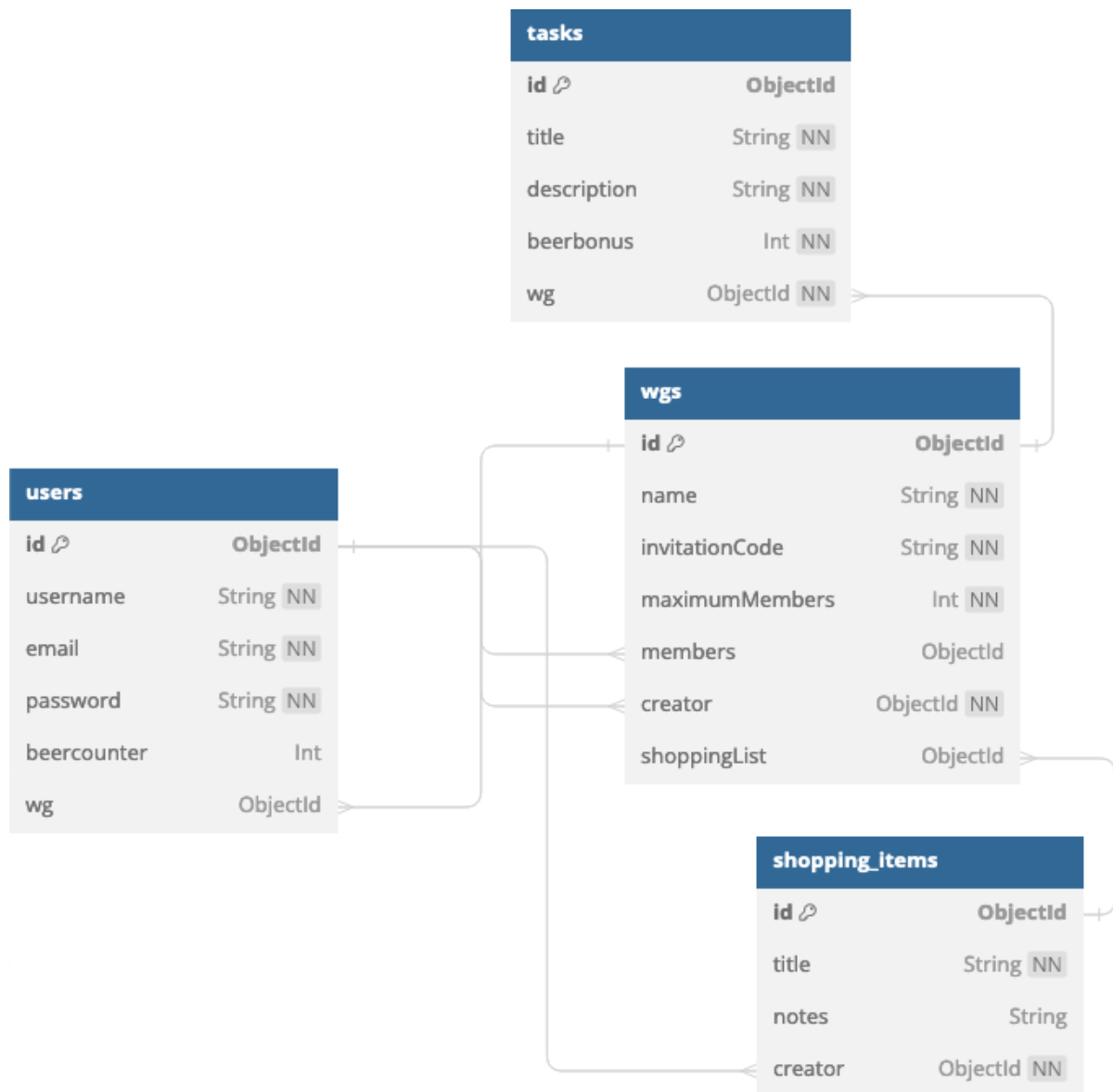
Ein zusätzlicher Abschnitt der App widmet sich dem Einkaufslistenbereich. Hier hat jeder Benutzer die Möglichkeit, Positionen, auch als "Items" bezeichnet, hinzuzufügen. Diese Items setzen sich, analog zu den Aufgaben, aus einem Titel und einer Notiz zusammen. Die Benutzer haben die Option, die Items abzuhaken, um deren Berücksichtigung im Einkaufsprozess zu markieren. Nach dem Einkauf können alle abgehakten Items bequem aus der Liste gelöscht werden, um eine stets aktuelle und übersichtliche Einkaufsliste zu gewährleisten.

1.3.4 WG-Bereich / Highscore

Im WG-Bereich werden die Highscore-Funktion ("Bierchen-Counter") sowie der Einstellungsbereich zusammengeführt. Jedes Mitglied der Wohngemeinschaft kann dort seinen individuellen Bierchen-Zähler einsehen. Sowohl die Benutzereinstellungen als auch die Einstellungen der gesamten WG können in diesem Bereich vorgenommen werden. Dies ermöglicht den WG-Mitgliedern eine zentrale Anlaufstelle zur Überwachung ihres Fortschritts und zur Anpassung der App-Einstellungen entsprechend ihren Bedürfnissen.

2 Backend

2.1 ER-Modell



2.2 Tech-Stack

In unserem Backend verwenden wir JavaScript und setzen auf den MERN-Stack, der aus einer Node.js/Express-Server-Applikation und einer MongoDB-Datenbank besteht. Zur Umsetzung nutzen wir Docker, wobei die Container mongodb, mongo_express und backend vorhanden sind. Das Backend ist in Model, Controller und Routes strukturiert. Hier ein Beispiel für eine Route zum Hinzufügen von Items zur Einkaufsliste:

Model:

```
const shoppingItemSchema = new Schema({
  title: {
    type: String,
    required: true
  },
  notes: String,
  creator: {
    type: mongoose.SchemaTypes.ObjectId,
    ref: "User",
    required: true
  },
  isChecked: Boolean
});
```

Controller:

```
async function addShoppingListItem(request, response) {
  try {
    const user = await User.findById(request.auth.userId);
    if (!user.isInWG()) {
      return response.forbidden(ResponseCodes.NotInWG);
    }

    const { title, notes } = request.body;
    const wg = await user.getWG();

    await wg.addShoppingListItem(user, title, notes);

    response.success({
      shoppingList: wg.shoppingList
    });
  } catch (error) {
    console.error(error);
    response.internalError();
  }
}
```

Route:

```
router.get('/shoppinglist',
  useJWT(),
  Controller.viewShoppingList
);

router.post('/shoppinglist',
  useJWT(),
  Controller.addShoppingListItem
);

router.put('/shoppinglist/check/:id',
  useJWT(),
  Controller.checkShoppingListItem
);
```

Den Login und die Berechtigungen haben wir via JWT Token und bcryptJS gelöst.

2.3 REST-API

GitLab Backend: <https://git.ai.fh-erfurt.de/ke5483ma/pme-colipi-backend.git>







URL API: <http://tool.colivingpilot.de:20013>

Folgende Routen sind über unsere API aufrufbar:








/api/wg

Methode	Pfad	Beschreibung
 GET	/api/wg/	WG-Infos abrufen
 POST	/api/wg/	WG erstellen
 PUT	/api/wg/	WG umbenennen
 DELETE	/api/wg/	WG entfernen
 GET	/api/wg/join?code=...	WG beitreten
 GET	/api/wg/leave	WG verlassen
 GET	/api/wg/kick/:id	Nutzer aus WG entfernen
 GET	/api/wg/shoppinglist	Einkaufsliste abrufen
 POST	/api/wg/shoppinglist	Einkaufsliste Eintrag hinzufügen
 PUT	/api/wg/shoppinglist/check/:id	Einkaufsliste Eintrag abhaken
 PUT	/api/wg/shoppinglist/:id	Einkaufsliste Eintrag verändern
 DELETE	/api/wg/shoppinglist/:id	Einkaufsliste Eintrag entfernen

/api/user

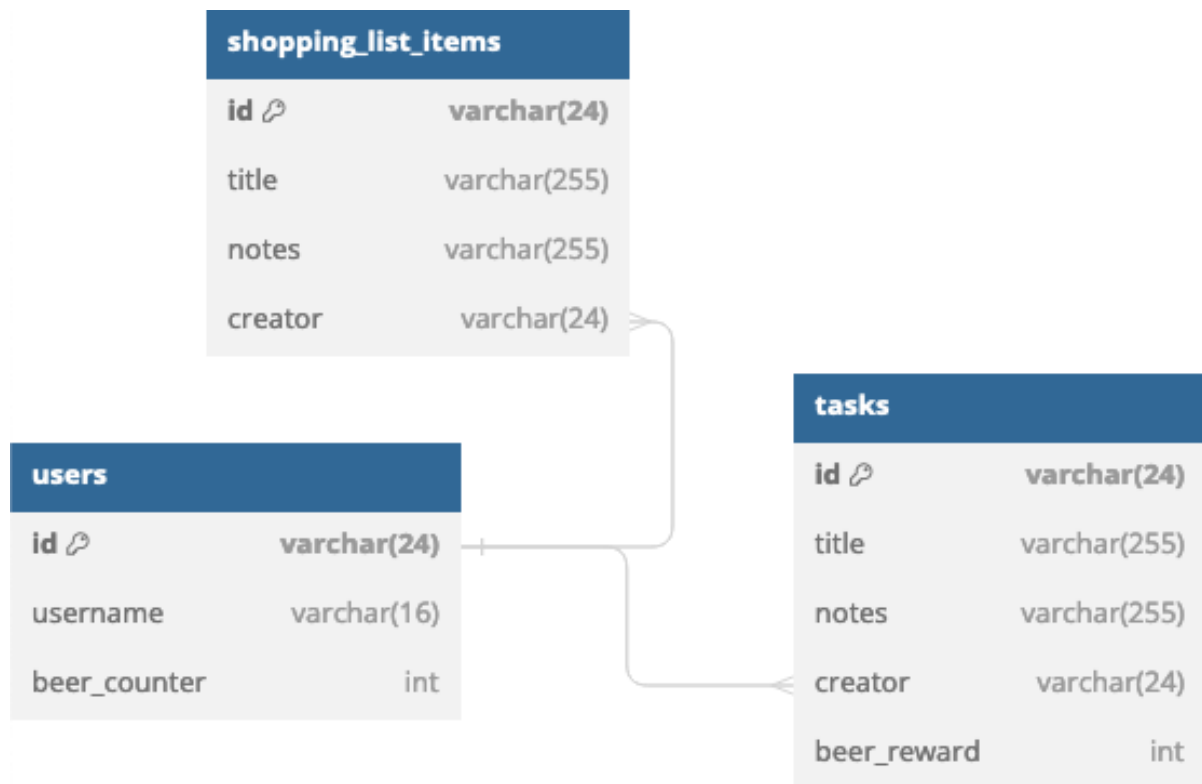
Methode	Pfad	Beschreibung
 GET	/api/user/	Gibt alle User mit allen Infos zurück
 GET	/api/user/:id	Gibt einen User anhand der ID zurück
 POST	/api/user/	Registrierung eines neuen Users
 POST	/api/user/login	Loggt einen User ein
 DELETE	/api/user/	Löscht den User der gerade angemeldet ist
 PUT	/api/user/	Updatet den User der gerade angemeldet ist

/api/task

Methode	Pfad	Beschreibung
 POST	/api/task/	Legt einen neuen Task an
 GET	/api/task/	Gibt alle Tasks einer WG aus
 GET	/api/task/filter/:id	Gibt einen bestimmten Task basierend auf der ID zurück
 GET	/api/task/filter	Gibt gefilterte Tasks zurück
 PUT	/api/task/:id	Updatet einen bestimmten Task basierend auf der ID
 DELETE	/api/task/:id	Löscht einen Task basierend auf der ID
 DELETE	/api/task/done/:id	Markiert einen Task als erledigt basierend auf der ID

3 Android App

3.1 ER-Model



3.2 Repository

Die `Repository`-Klasse ist verantwortlich für die Datenhaltung und die Interaktion mit der Backend-Datenbank über Retrofit. Sie enthält Methoden zum Aktualisieren der App-Daten, Abrufen von Benutzerinformationen, Verwalten von Aufgaben und Bearbeiten von Einkaufslistenartikeln.

3.3 Datenbankzugriff

Die Klasse verwendet die `WgDatabase`, um auf lokale Daten zuzugreifen. Dazu gehören DAOs für `User`, `Task` und `ShoppingListItem`. Die Datenbank wird initialisiert, und die DAOs werden für den Zugriff auf die jeweiligen Entitäten erstellt.

3.4 Refresh-Methode

Die `refresh`-Methode wird verwendet, um die App-Daten zu aktualisieren, indem sie die relevanten Daten vom Server abrufen und die lokale Datenbank aktualisiert. Diese Methode wird in einem Coroutine-Scope auf einem Hintergrundthread ausgeführt.

3.5 Netzwerk

Netzwerkanfragen, wie das Hinzufügen, Aktualisieren oder Löschen von Listenelementen, werden durch das `Repository` durchgeführt, das die Kommunikation mit der Backend-Datenbank steuert. Fehler oder Erfolgsrückmeldungen werden durch Callbacks wie `NetworkResultNoData` behandelt.

3.6 Aktualisierung und Synchronisation

Die Aktualisierung der Liste und die Synchronisation mit dem Backend erfolgen durch das refresh-Verfahren im `ShoppingListViewModel`. Hierbei wird die `refresh`-Funktion der `Repository` Klasse aufgerufen, um die neuesten Daten abzurufen, neue Datensätze in die lokale Datenbank einzutragen, veränderte Einträge zu aktualisieren und fehlende Einträge auch lokal zu löschen – dadurch wird folglich die Benutzeroberfläche aktualisiert.

3.7 Login, Registrierung und WG-Beitritt

Wir nutzen mehrere Activities in unserer App, um es dem Nutzer leicht zu machen, sich anzumelden/zu registrieren und einer WG beizutreten. Die `UnauthenticatedActivity` wird aufgerufen, wenn kein abgespeicherter JSON-Web-Token in den Shared Preferences gefunden werden konnte, oder dieser ungültig ist. In dieser Activity wird der Nutzer aufgefordert, sich anzumelden oder zu registrieren. Sollte er sich anmelden wollen, wird die `LoginActivity` aufgerufen, in der ein Formular für das Login bereitsteht. Falls er sich jedoch registrieren möchte, so wird die `RegisterActivity` aufgerufen, die ein ähnliches Formular anzeigt. Wenn die respektiven Formulare dann ausgefüllt worden sind, wird die jeweilige Anfrage an unser Backend gesendet und die Antwort dessen interpretiert. Ist diese positiv, so wird der zurückgegebene JSON-Web-Token in den Shared Preferences abgespeichert. Sollte der Nutzer sich angemeldet haben und bereits zu einer WG gehören, so wird er zur `MainActivity` weitergeleitet. Andernfalls wird er zur `NotInWgActivity` weitergeleitet, wo er aufgefordert wird, eine WG zu erstellen oder aber einer bereits bestehenden WG beizutreten. Wenn eine neue WG angelegt werden soll, wird die `CreateWgActivity` aufgerufen, in der ein Formular ausgefüllt werden muss. Sollte der Nutzer jedoch einer bestehenden WG beitreten wollen, wird die `JoinWgActivity` aufgerufen, in der er die Möglichkeit hat, einen Invitation-Code einzugeben oder einen Einladungs-QR-Code einzuscannen. Wenn die Erstellung der WG/der Beitritt zu einer WG erfolgreich war, wird die `JoinedWgActivity` aufgerufen, in

der der Nutzer mit etwas Konfetti willkommen geheißen wird – von hier aus kann er nun zur MainActivity navigieren.

3.8 Einkaufsliste

Der Einkaufslisten-Teil der App ermöglicht es den Benutzern, ihre Einkaufsliste zu verwalten, Artikel hinzuzufügen, zu bearbeiten und zu löschen. Die wichtigsten Komponenten dieser Funktion sind das Datenmodell (`ShoppingListItem`), der Adapter (`ShoppingListAdapter`), die ViewModel-Klasse (`ShoppingListViewModel`), und die zugehörigen Fragmente (`ShoppinglistFragment` und `ShoppingListItemAddDialogFragment`).

3.8.1 Fragmente

Das `ShoppinglistFragment` stellt die Hauptansicht für die Einkaufsliste dar. Es enthält eine RecyclerView für die Anzeige der Listenelemente, einen SwipeRefreshLayout für das Aktualisieren der Liste und Buttons zum Hinzufügen neuer Elemente oder Löschen abgeschlossener Einkäufe. Das `ShoppingListItemAddDialogFragment`, Ein BottomSheetDialogFragment, dass dem Benutzer die Möglichkeit gibt, ein neues Element zur Einkaufsliste hinzuzufügen. Es enthält Eingabefelder für den Titel und optionale Notizen. Das `ShoppingListItemEditDialogFragment` ist ein Bottom Sheet Dialog Fragment, das es dem Benutzer ermöglicht, einen existierenden Einkaufslistenartikel zu bearbeiten. Es enthält die gleichen Eingabefelder wie das vorherige `ShoppingListItemAddDialogFragment`.

3.8.2 ViewModel

Die `ShoppingListViewModel`-Klasse fungiert als Vermittler zwischen der Benutzeroberfläche und dem Datenrepository (`Repository`). Sie bietet Funktionen zum Hinzufügen, Aktualisieren, Löschen von Listenelementen und zum Aktualisieren der Ansicht. Außerdem bietet sie eine LiveData-Instanz für die Beobachtung von Änderungen in der Liste.

3.8.3 Benutzerinteraktion

Benutzerinteraktionen wie das Anhaken von Elementen, Klicken auf Elemente für detaillierte Ansichten oder Hinzufügen neuer Elemente werden durch `ShoppingListActionListener` und entsprechende Callbacks in `ShoppinglistFragment` und `ShoppingListAdapter` verarbeitet.

3.9 Aufgabenliste

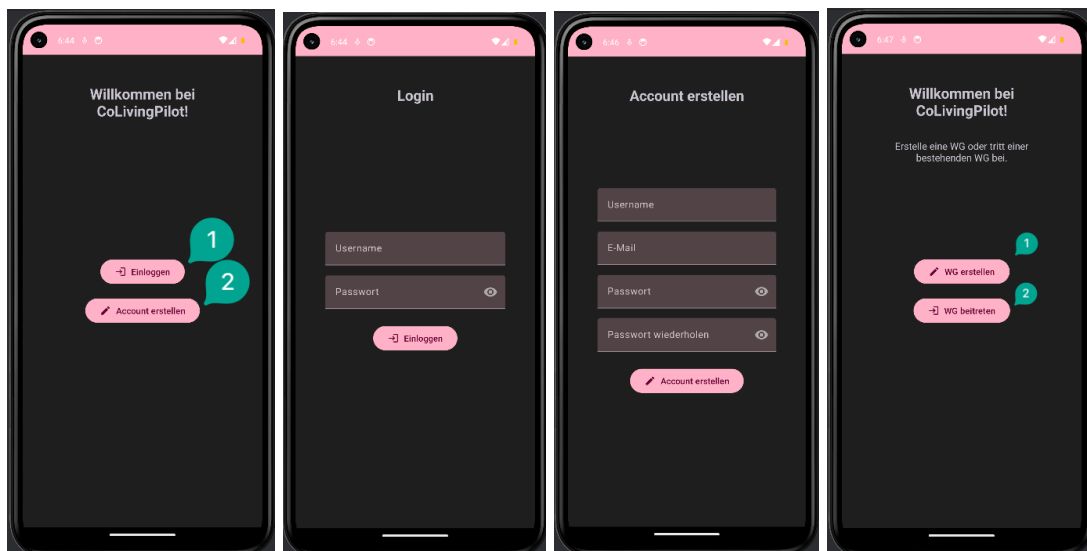
Der Strukturaufbau der Aufgabenliste (Tasks) weist eine deutliche Ähnlichkeit zur Implementierung des ShoppingList-Moduls auf. Beide basieren auf einem RecyclerView und den zugehörigen Komponenten.

3.10 WG-Einstellungen

Das WgFragment sendet WgEvents an das WgViewModel, um auf Benutzerinteraktionen zu reagieren, z.B. bei langem Klicken auf einen User wird das WgEvent "onLongClickUser" mit dem Username als Parameter an das Viewmodel übergeben. Das ViewModel verarbeitet diese Events und kann daraufhin Änderungen im Backend anstoßen oder UiEvents über einen Channel zurück an das Fragment senden, um wiederum UI-Änderungen im WgFragment anzustoßen. Dies ermöglicht eine bidirektionale Kommunikation zwischen UI (Fragment) und Logik (ViewModel) in einer sauberen, entkoppelten Architektur.

4 Testbare Funktionalitäten der App

4.1 Registrierung Benutzers, anlegen einer WG und Login

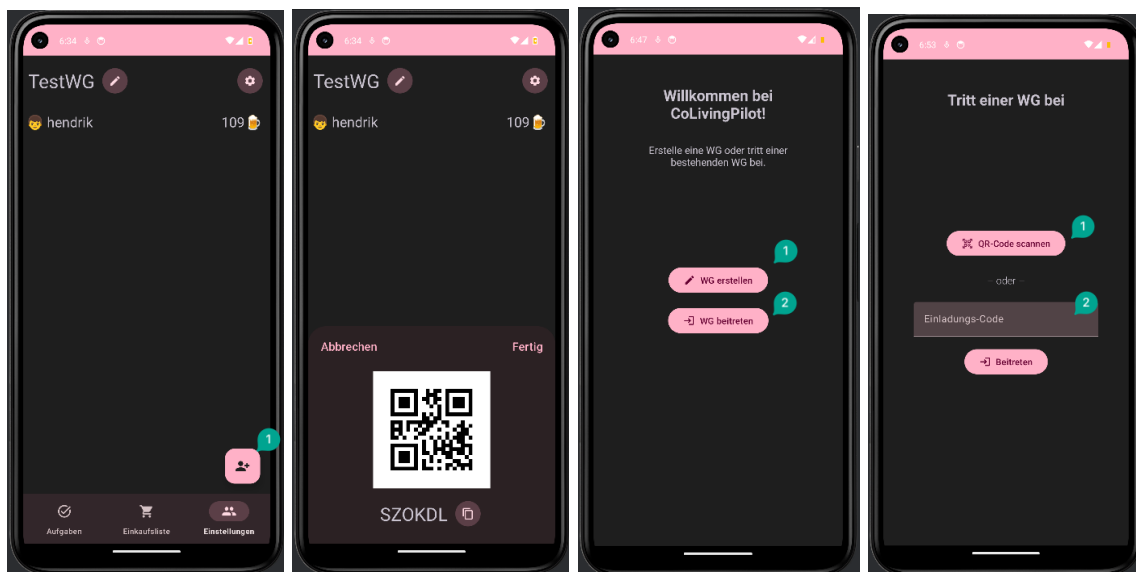


Wird die App zum ersten mal gestartet landet auf einer Welcome-Page und kann sich entweder einloggen (1) oder einen neuen Account erstellen (2). Nachdem man einen neuen Account erstellt hat wird man aufgefordert eine WG zu erstellen (1) oder einer bestehenden WG beizutreten (2) -> siehe 4.2.



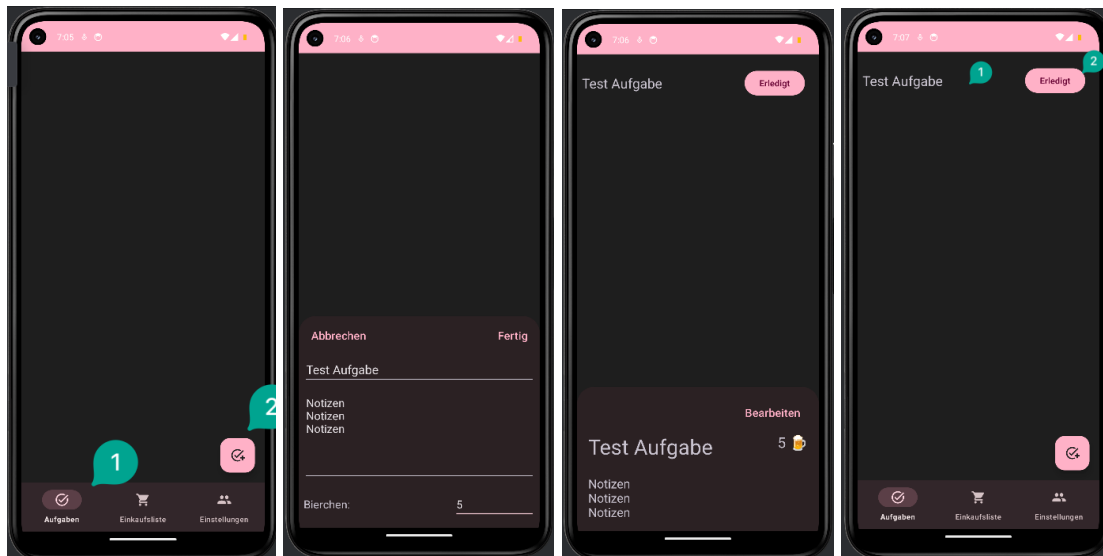
Eine WG besteht aus einem Namen und der Anzahl der WG-Mitglieder.

4.2 Einer bestehenden WG beitreten



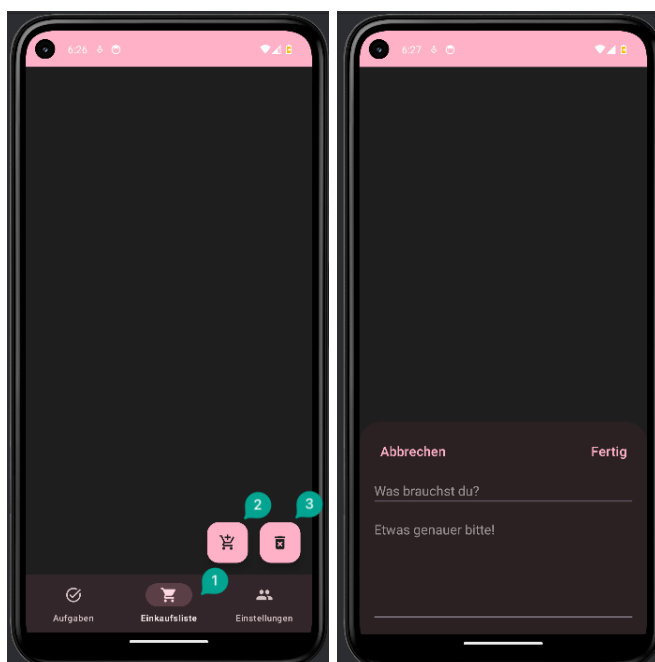
Möchte man einer bestehenden WG beitreten, benötigt man entweder den QR-Code oder den alphanumerischen Code. Dieser Kann im Bereich Einstellungen über den Mitglied-hinzufügen-Button (1) generiert werden. In der App des neuen Mitgliedes, im Bereich WG Beitreten kann man nun den QR-Code Scannen (1) oder den alphanummerischen Code eingeben (2).

4.3 Aufgaben erstellen, bearbeiten und abhaken

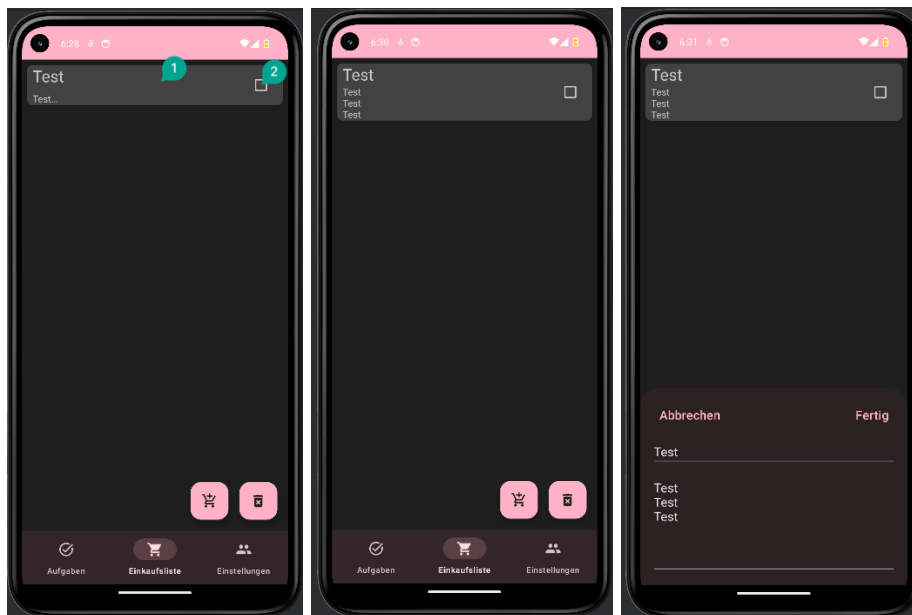


Im Bereich Aufgaben (1) kann eine neue Aufgabe durch klicken des Häckchensymbols (2) angelegt werden. Nachdem das ToDo angelegt wird erhält man ein Bestätigungsview. Dieser dient zu Kontrolle und bietet die Möglichkeit die Aufgabe direkt zu bearbeiten. Klickt man im Aufgaben bereich auf eine Aufgabe öffnet sich erneut der Detailview. Führt man einen langen Klick aus, so öffnet sich dierekt der Bearbeitungskontext. Über den Erledigt-Button (2) lässt sich die Aufgabe löschen.

4.4 Einkaufsliste erstellen, bearbeiten und abhaken

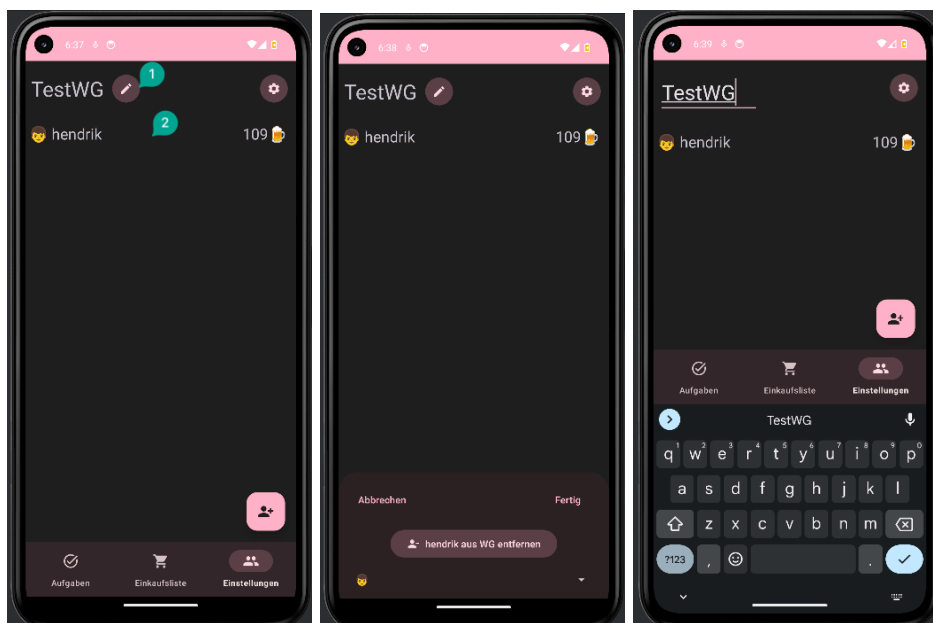


Im Einkaufslistenbereich (1) lässt sich durch einen Click auf das Einkaufswagensymbol (2) ein neues Item anlegen. Alle abgehakten Items lassen sich über das Mülltonnensymbol (3) löschen.



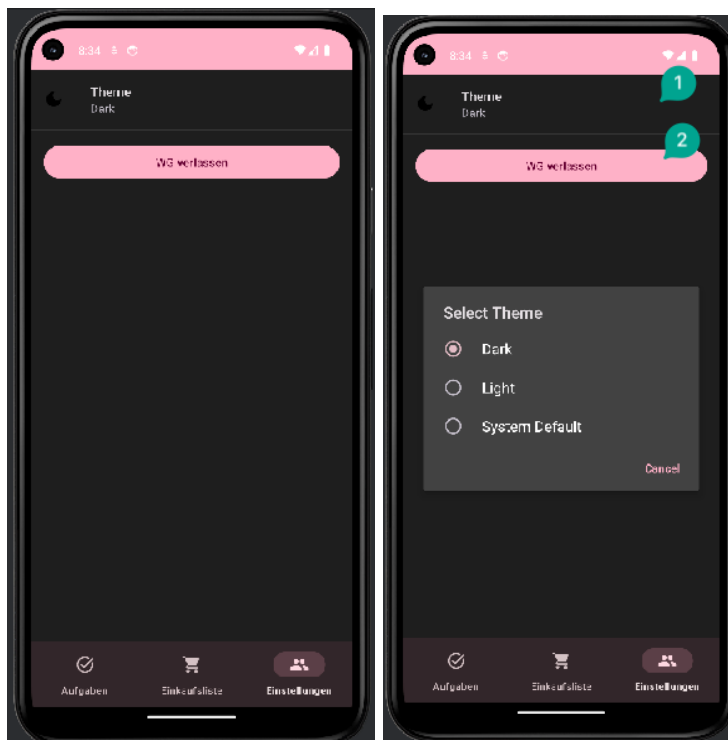
Über einen Klick auf das Item (1) lässt sich die Komplette Notiz anzeigen. Ein langer Klick auf das Item (1) öffnet den Bearbeitungskontext.

4.5 Bearbeitung der WG



Durch einen langen Klick auf ein WG-Mitglied (2), lässt sich dieses aus der WG entfernen oder das Emoji ändern. Ein Klick auf den Zeichenstift (1) aktiviert die Bearbeitung des WG-Titels.

4.6 Einstellungen



Innerhalb der WG-Einstellungen bietet die App den Benutzern die Möglichkeit, die Wohngemeinschaft zu verlassen oder das Farbschema nach ihren Präferenzen anzupassen.