GitHub Username: jkozh

Worderly

Description

Worderly is an online multiplayer game conquering the hearts and minds of children and adults. The goal is to construct a word by its definition using all given letters. Who'll guess the word faster - wins. It is a crackerjack way to surprise your opponent with your knowledge of a copious vocabulary, and your typing skills test.

Intended User

This app for everyone who wants to enrich their knowledge of the language, as well as show off their erudition and vocabulary.

Features

- Log in with your Google account, or if you don't want, then just be as a guest.
- Play the game with the real random person in real-time.
- Chat with opponent while playing the game.
- The app fetches random words from Words API using google's volley to Post and Get Requests.
- The app uses Firebase Real Time Database to store user's information, and information about the games, and chats.
- The app stores a fetched word data locally by implementing a ContentProvider, and uses a Loader to move the data to its views.
- The app developed mostly based on Model View Presenter pattern plus Interactors to fetch data from Firebase Database.

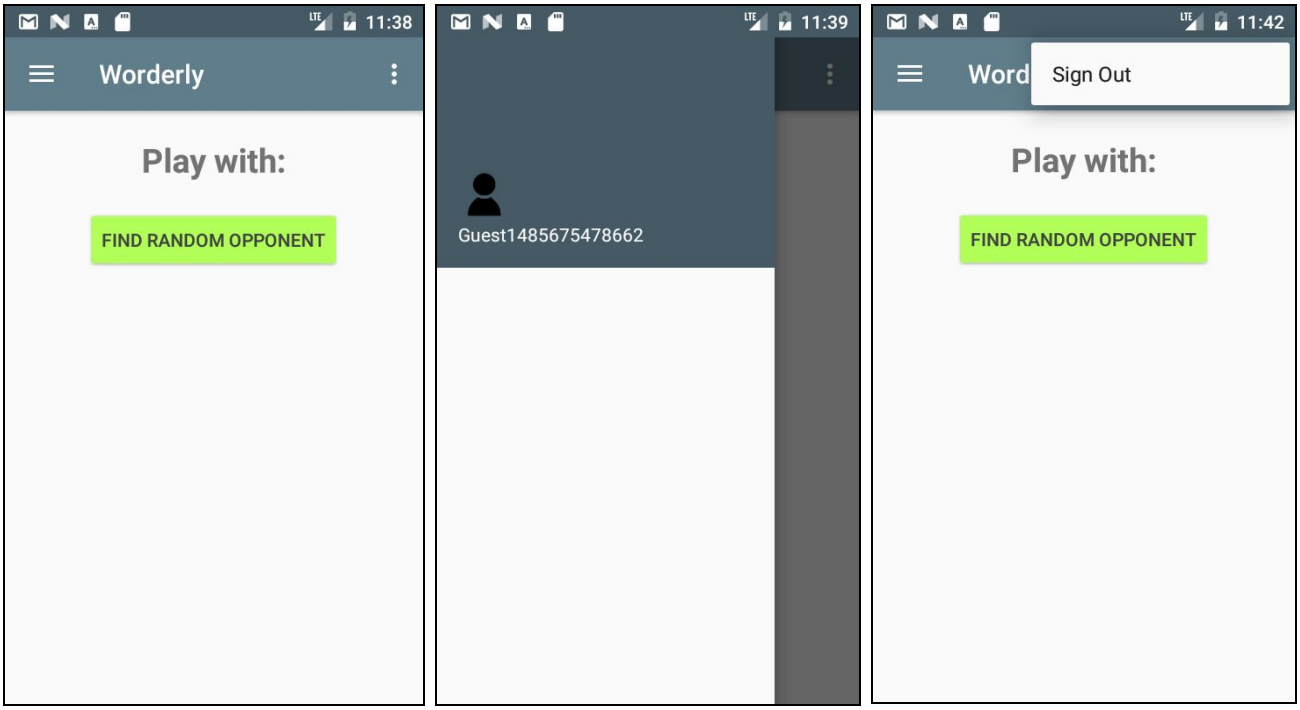
User Interface Mocks

Screen 1

SignIn Activity.
This screen appear when the user launches Worderly application for the first time. The user can choose how to sign in into application by Google account, or just as a Guest. Both operations handled by the Firebase. The logged user will be saved to Firebase Database under 'users' child into <User_ID> child. Similar as on the picture:


users
3fbEzwLxTsgmmmvupwKNtSbC6h53
email: "0"
id: "3fbEzwLxTsgmmmvupwKNtSbC6h5
photoUrl: "https://cdn4.iconfind
username: "Guest148565799686

Screen 2

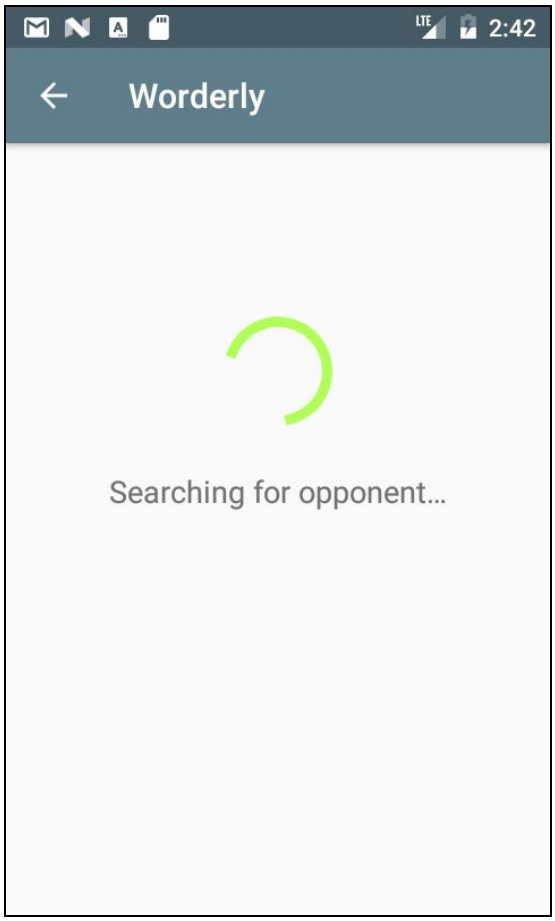


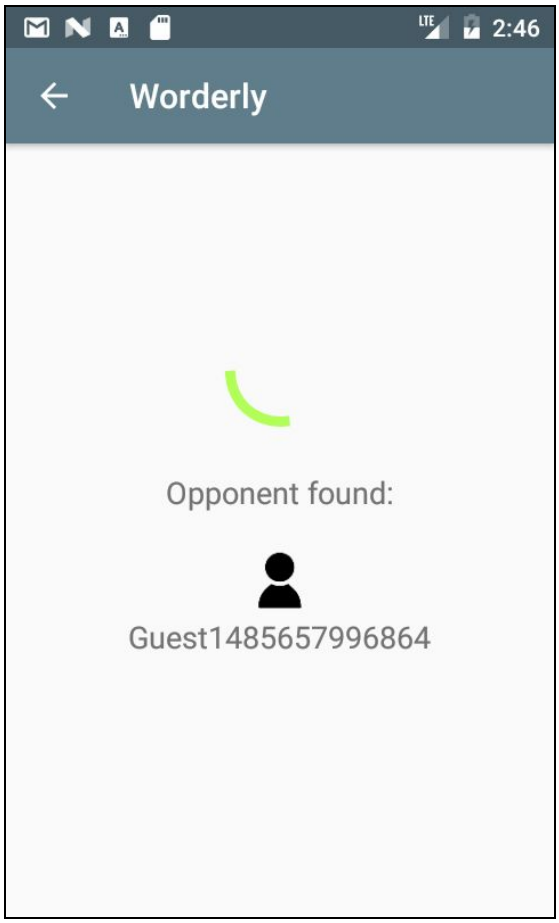
Main Activity.
It shows after the user successfully signed in. It has the navigation drawer on the left. Which holds the user's information such as the picture and the username.

Main Activity has a menu in the right top corner with Sign Out option.

The press on "FIND RANDOM OPPONENT" button will open the Search Opponent Activity, and will add the current user to Firebase under "usersOnline" child, so the user will be marked as available to play a game.

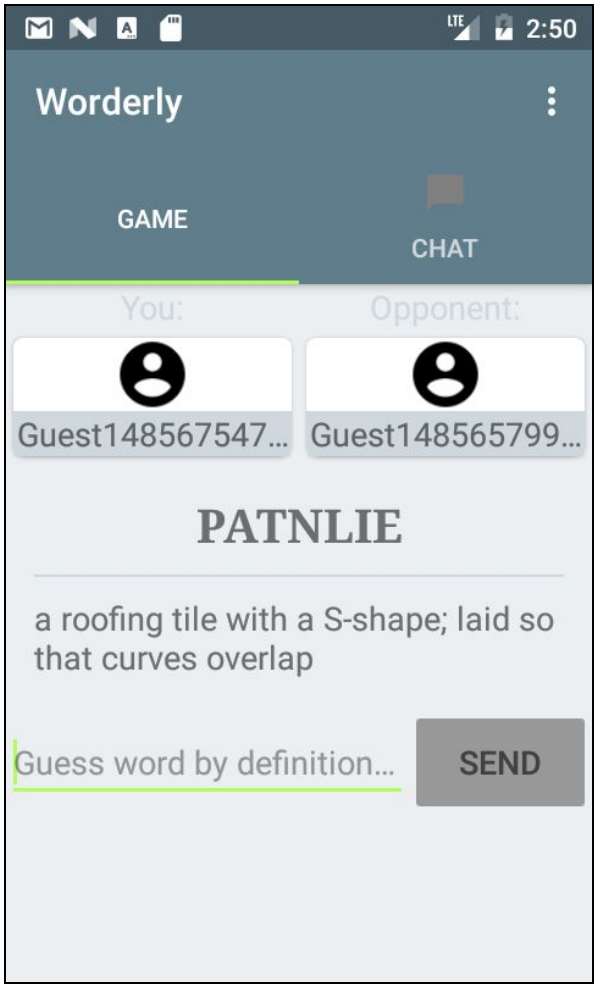
Screen 3

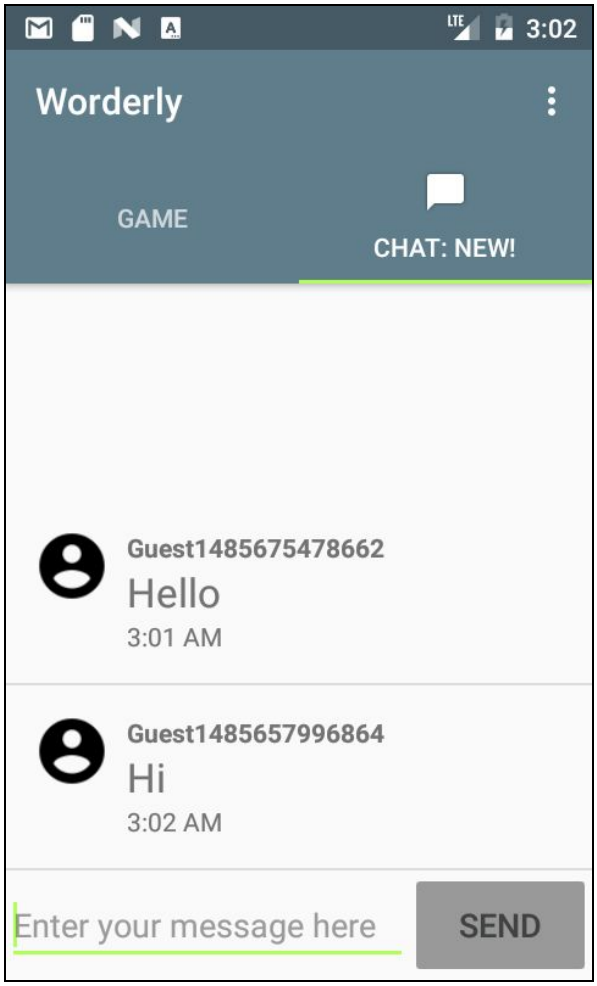




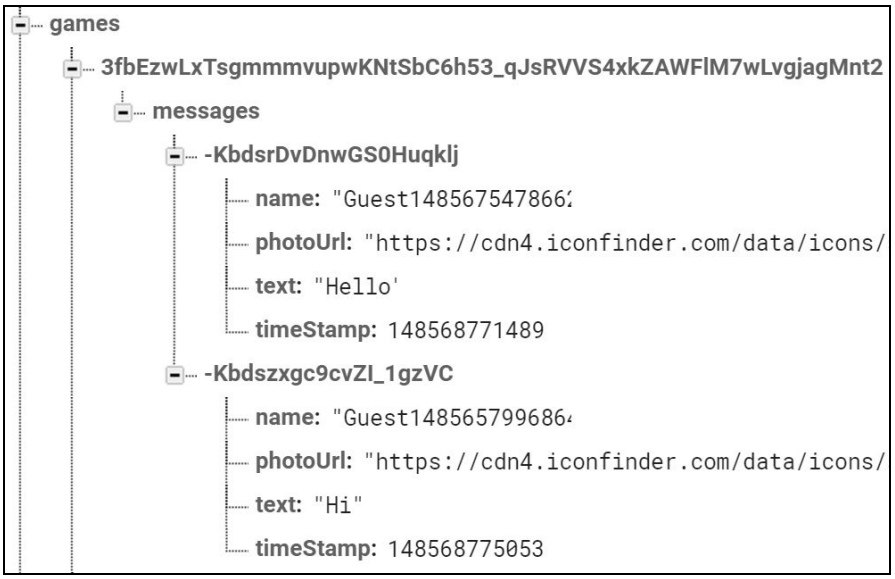
SearchOpponent Activity.
This screen will show the progress bar while searching for the available user. And when the user was found it will show the found user: his name and his picture.

Screen 4





Game Activity.
This activity consists of two fragments **Game Fragment** and **Chat Fragment**. You can swipe between them. Game fragment has CardViews with players information, text views with the word, that needs to be constructed, and definition of that word. In Edit text you should input the right needed word. If it's correct - you win, if it's not - you have to try input again. If the opponent was faster - you lose.
In the menu you has a Resign option.
If the length of the word is incorrect - the "SEND" button won't enables.
In the Firebase it will look something like this:


games
3fbEzwLxTsgmmmvupwKNtSbC6h53_qJsRVVS4xkZAWFlM7wLvgjagMnt2
messages
-KbdsrDvDnwGS0Huqklj
name: "Guest148567547866
photoUrl: "https://cdn4.iconfinder.com/data/icons/
text: "Hello"
timeStamp: 148568771489
-Kbdszxgc9cvZI_1gzVC
name: "Guest148565799686
photoUrl: "https://cdn4.iconfinder.com/data/icons/
text: "Hi"
timeStamp: 148568775053

Key Considerations

How will your app handle data persistence?

I will build a Content Provider to store a fetched word from the Words API, along with its definition. I also need to store a scrambled version of that word, to put it into view of my game. So the database will look like this:

_id	word	scrambled	definition
1	passion	sasonip	definition

Also, I will be using a Firebase Database to store user's accounts, current games and messages.

Describe any corner cases in the UX.

If the user hit the back button while searching an opponent he goes to Main Activity Screen. If the user hits the back button while playing the game he goes to Home Screen activity, and to continue the game he can click on recent tasks and choose the app, or on icon of the game.

Describe any libraries you'll be using and share your reasoning for including them.

I will be using Volley with GSON to Post and Get Requests because it is better and faster than AsyncTask, to fetch the word data from Words API. A Glide for loading images. Firebase as real-time database. De.hdodenhof:circleimageview to circle images. GSON for storing word data when fetched from API. ButterKnife for binding views. Play-services for Google sign in.

Describe how you will implement Google Play Services.

I will use a Firebase as real-time database, Google sign in.

Next Steps: Required Tasks

Task 1: Project Setup

- Configure libraries
- Make sure all dependencies are up to date
- Add your own Words API key

- Divide the classes into separate packages: ui, data, model, network, utils.

Task 2: Implement UI for Each Activity and Fragment

- Build UI for SignInActivity
- Build UI for MainActivity
- Build UI for SearchOpponentActivity
- Build UI for for GameActivity
- Build UI for GameFragment
- Build UI for ChatFragment

Task 2: Implement Sign In

- Implement sign in via Google
- As alternative implement sign in via Anonymous Firebase service, to be as a guest.

Task 3: Implement Firebase operations

- Implement Firebase storing the user data
- Implement Firebase storing the users who wants to find opponent
- Implement Firebase storing the current games information
- Implement Firebase storing the chat messages in the game
- Implement Firebase deleting online users
- Implement Firebase deleting finished games and messages

Task 4: Implement fetching words data from API

- Implement Volley requests to fetch the data from API
- Implement model classes to handle GSON format

Task 5: Create Content Providers

- Create a table to store words data fetched from API
- Implement Loaders to access the data and move that data into views
- Implement cleaning of the database after the game was finished

Task 6: Create Shared Preferences to store the user

- Create Shared Preferences for storing the current user
- Implement sending opponents data along with intent while launching the activities.