GitHub Username: JWar

Capstone_Project

Description

The app will provide messaging services between users of the app, with the potential to include SMS fallback if contact is not a user.
Think WhatsApp but without the Facebook. Given the recent revelations from Cambridge Analytica it may be a good time to avoid Facebook and its related products.

Intended User

Anyone who needs to message someone. Introverts especially welcome!

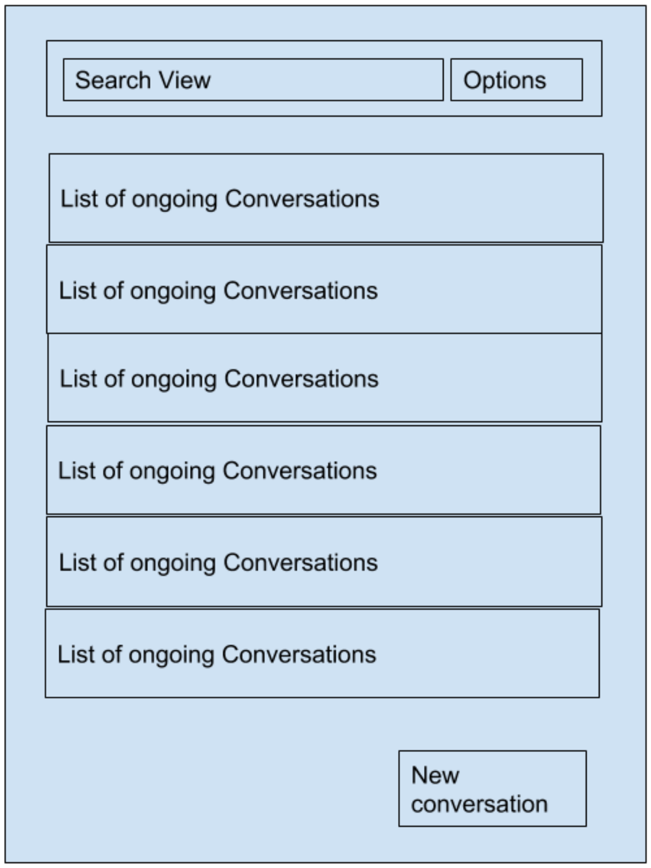
Features

Stores Contact list.
Allows messaging between Contacts. Resorting to SMS if needs be.
Displays all conversations involving Contacts.
Allows adding/deleting/updating Contact information.

User Interface Mocks

These can be created by hand (take a photo of your drawings and insert them in this flow), or using a program like Google Drawings, www.ninjamock.com, Paper by 53, Photoshop or Balsamiq.

Screen 1

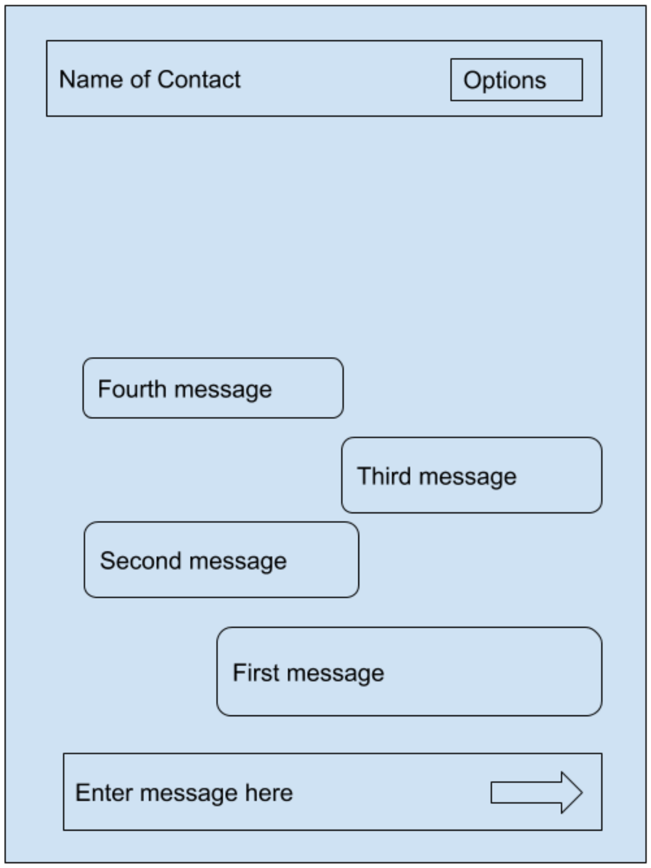

Search View
Options
List of ongoing Conversations
List of ongoing Conversations
List of ongoing Conversations
List of ongoing Conversations
List of ongoing Conversations
List of ongoing Conversations
New conversation

A simple list of the ongoing conversations. In the bottom New conversation is a Floating Action Button to start a new conversation...

Search View does what you'd expect, searches through the conversations.

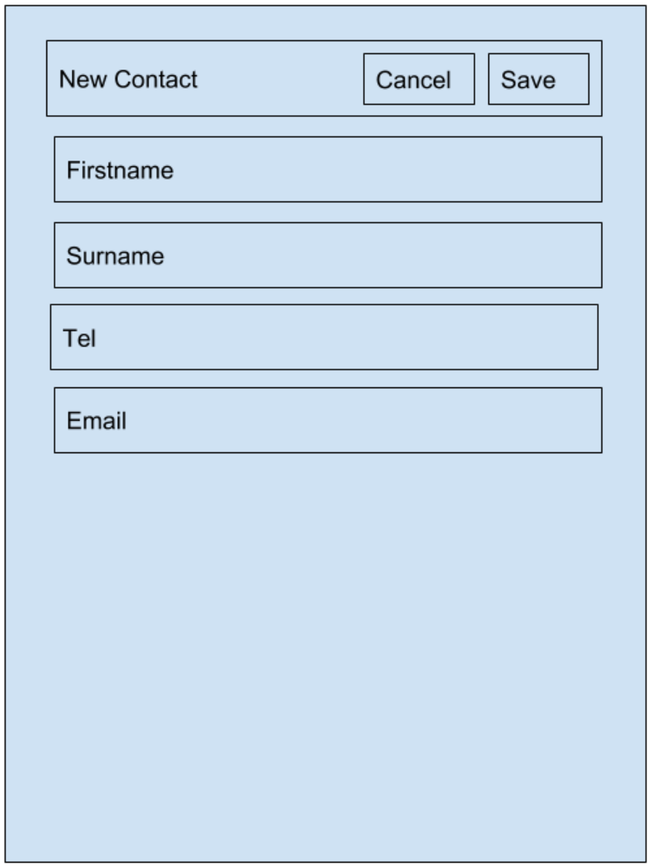
Options provides access to Settings (can add more options to this.) and includes New Contact which goes to the New Contact screen.

Screen 2



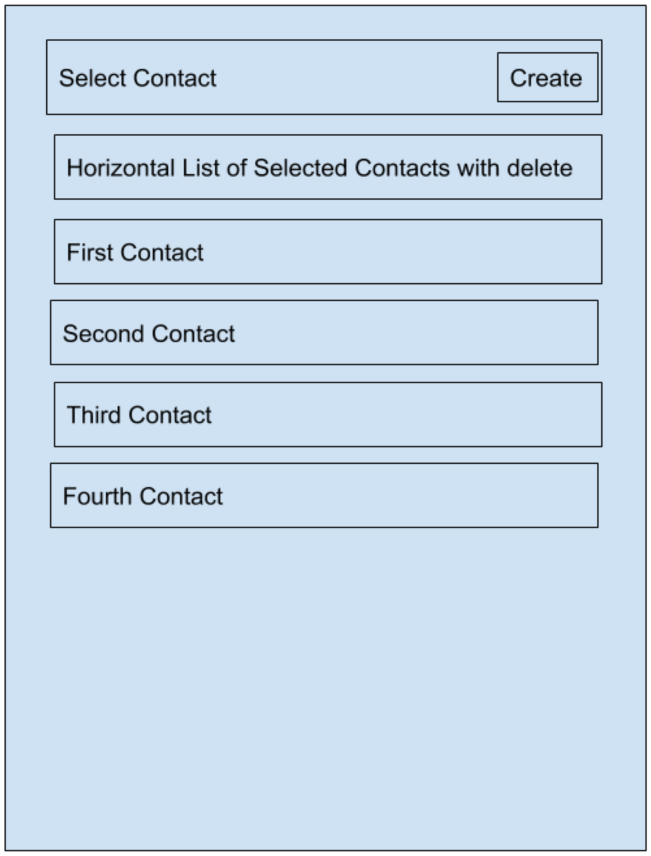
Standard messaging screen. Options will include delete (can add more!) which deletes the conversation and its messages. Messages are Textviews with background of rounded shape. Enter message is an Edittext with a ‘Send’ icon.

Screen 3



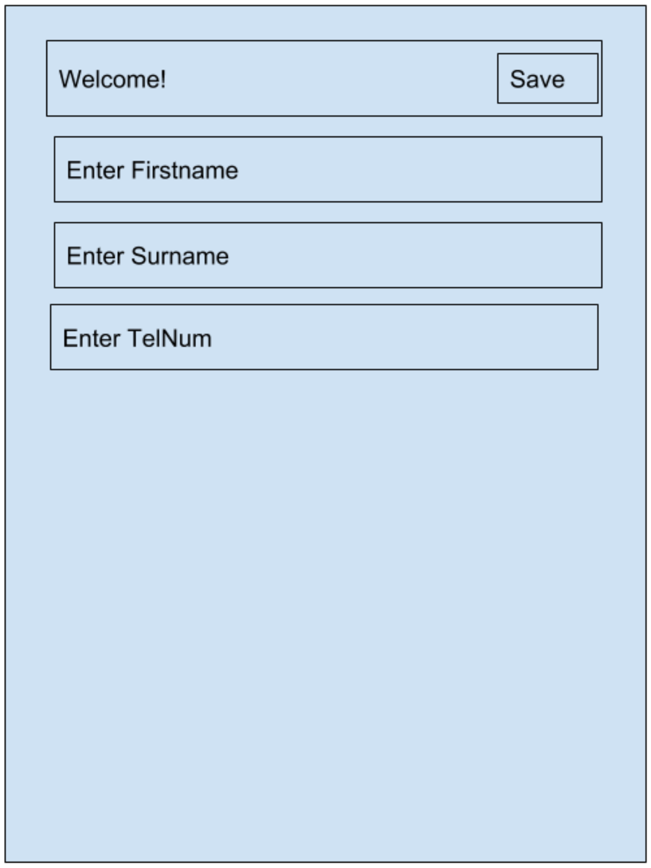
New Contact screen. Pretty straightforward, Edittexts with ‘Hint’ to explain what its for. Save and Cancel are Menu Action items.

Screen 4



Create new conversation screen. Select contact which goes into a list at the top. Press ‘Create’ to start. Quick way of handling both two person conversations and more than two.

Screen 5



Required Firstname, Surname and Tel. Pressing save will get list of Contacts and Firebase Device ID sent to server to store: linking users name, tel and their id. Server call will return, if successful will bring user to opening screen of app (Screen 1), if unsuccessful will notify user and do nothing else!

Key Considerations

How will your app handle data persistence?

Content Provider will be used to store the Contacts, Messages and Conversations in an SQLite database.

Describe any edge or corner cases in the UX.

Messages will be able to be accessed from notifications, pressing back from there will make sure it returns to the previous screen rather than returning 'UP' into the Conversation list.

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

Retrofit: for networking, because its efficient, and what I'm used to!
Espresso/Junit/Standard testing: for testing…

Describe how you will implement Google Play Services or other external services.

Firebase for Push notifications, letting the app know there are messages waiting to be downloaded.
Analytics, to monitor how the user uses the app.
Implemented via import/implementation and following the online tutorials/Android Studio wizards.

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and break them down into tangible technical tasks that you can complete one at a time until you have a finished app.

Task 1: Project Setup

Start new Project in Studio.
Add dependencies in build.gradle – testing, Retrofit, Play services.
Design app, set up packages to follow the MVP pattern (Remote/Local DataSources, Repository, UI, Model etc…).

Task 2: Build Variant prod/mock.

Add to build.gradle flavor dimensions/build types.
Create folders/packages.
Test!

Task 3: Implement UI for Each Activity and Fragment

Using Loaders to monitor database tables to ensure dynamic updates.
Create UI - using Fragments for View and POJO Presenter for P.
Will need Conversation View and Presenters with its Contract. This shows list of Conversations with a blurb and unread/read status.
Will need Messages View and Presenters with its Contract – this will be a list of messages in reverse order and have an edittext at the bottom for adding a new message.
Will need New Message View and Presenters with its Contract.

Will need Contact View and Presenters with its Contract.
Will need New Contact View and Presenters with its Contract (will this be done via Built In Contacts?).
Will need New Conversation View and Presenters with its Contract. At top has horizontal list displaying Contacts selected, with option of deleting from that list.
Widget to display most popular contact with option to access their msg list.
Test!

Task 4: Model

Create model/entity objects.
Create entity data sources (local and remote).
Create repositories.
Test!

Task 5: Installation screen

Create UI Installation, Activity with View/Fragment and Presenter/POJO. Simple screen welcoming people to app and letting them know what will happen. Ask user for firstname, surname and tel number!
This will require extracting Contacts list with Tel Nums updating backend (how else does the system know who is a user?). So parse Contacts list, get Name and Tel and JSONify then send to server with the Firebase Device ID, so the server can create a link between the users name, tel and id. This is how the server can forward push notifications to the correct phone.
This is where the necessary Firebase tokens are also sent to the back end (so Firebase knows which device to send to).

Task 6: Firebase/Push/Networking handling.

Firebase will receive push notifications triggering an Intent Service to Retrofit a call to the backend to download waiting messages.
Notifications of new messages once downloaded will appear. Will need classes (Singletons?) for that. All done in IntentService (so off main thread).

Task 7: basic tests

Basic MVP tests and androidTesting of UI.

Submission Instructions

- After you’ve completed all the sections, download this document as a PDF [File → Download as PDF]
    - Make sure the PDF is named “**Capstone_Stage1.pdf**”
- Submit the PDF as a zip or in a GitHub project repo using the project submission portal

If using GitHub:

- Create a new GitHub repo for the capstone. Name it “**Capstone Project**”
- Add this document to your repo. Make sure it’s named “**Capstone_Stage1.pdf**”