GitHub Username: luanalbineli

Burger Delivery

Description

This app will bring you an easy way to order your favorite burger. You're just in a few taps to order the best burger in the city!

Intended User

This app is indeed to everyone hungry. We have burgers made with red and soya meat.

Features

List the main features of your app. For example:

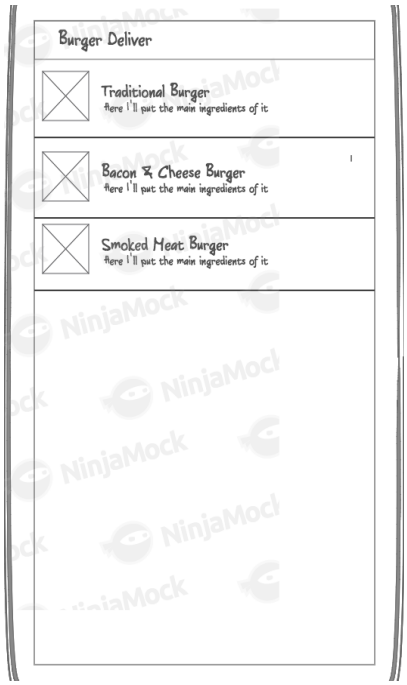
- Show the burger list;
- Possibility to increment your burger (additionals);
- Automatically saves your order (locally);

- Check the order status (Sent, Cooking, In Route, Delivered)
- Show a list with your made orders
- Show a widget with the status of your current order

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

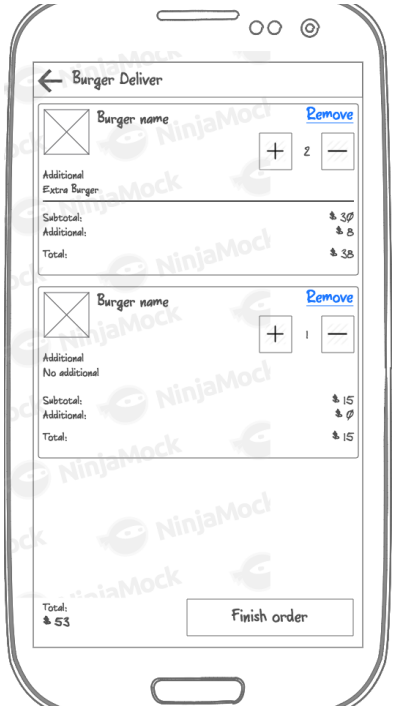


This is the burger list (if the user clicks on the row, he will go to the detail). The data will came from a Loader.

Screen 2

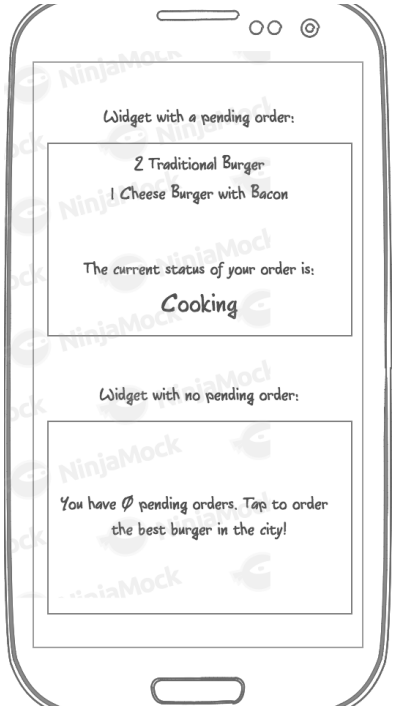
This is the burger detail screen. He can add it to the order (with some additionals).

Screen 3



This is the order screen. Here the user can finish it's order and change the quantity of hamburgers.

Screen 4



This is the widget screen with/without pending orders (it will show the informations about the last made order).

Key Considerations

How will your app handle data persistence?

- The hamburger list will come from the API (I'll use a Loader bring to the UI).
- The orders made by the user will be stored on a ContentProvider + SQLite database.

Describe any edge or corner cases in the UX.

I'll follow the Material Design Guidelines to implement the UX, implementing the default behaviours (back button, FAB button for the main action, etc).

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

- Support libraries: support older versions of Android (minimum version will be 17);
- RxJava + RxAndroid: avoid the callback hell, let the library handle async tasks threads execution (with a small set of configurations);
- Gson + Retrofit: handle the server communication by parsing the objects from/to JSON;
- Dagger2: handle dependency injection (make the dependencies mocked easily on tests);
- Timber: no need to use a tag to log, and you can set how it will log (level, usage of Crashlytics) depending on your build type (debug/release);
- EventBus: make your components more independent (making the communication easy between them, specially with Fragments and Activities);
- Fresco: handle the image loading/caching.

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

- Firebase Cloud Messaging: I'll use it to notify the user about the order status changes: https://firebase.google.com/docs/cloud-messaging/

- Firebase Crashlytics: track the application errors and usage: https://firebase.google.com/products/crashlytics/

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

Tasks:

- Create the Github repository;
- Create the Android Project;
- Configure the dependencies;
- Push the initial commit;
- Start to build the project.

Task 2: Implement UI for Each Activity and Fragment

- Create an Activity to list the hamburgers:
  * Add the RecyclerView;
  * Configure the Adapter;
  * Create the layout for the Adapter;
  * Consume the service that brings the list of hamburger from the API;

- Create an Activity that the user can add an hamburger to the order, with possibility to add some increments (double cheddar, bacon, onion crisps, etc)
  * Show the detail of the selected hamburger;
  * Show the ingredients of the selected hamburger;
  * Show a list with the increments for that hamburger;

- Create an Activity to show the orders made by the user (stored locally):
  * Add the RecyclerView;
  * Configure the Adapter;
  * Create the layout for the Adapter;
  * Consume the service that brings the list of orders from the local storage (ContentProvider + SQLite);

- Create an Activity to show the current order of the user:
  * Add the RecyclerView;
  * Configure the Adapter;
  * Create the layout for the Adapter;
  * Consume the service that brings the list of items of the current order from the local storage (ContentProvider + SQLite);

- Create an widget to show the status of the last pending order of the user. If the user don't have any pending orders, it'll show an empty message.

Task 3: Configure the Firebase Cloud Messaging on the Client

I'll need to configure the FCM on the client. The server (an simple Node.JS application) will updated the status randomly, between 1 ~ 5 minute(s).

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**”