GitHub Username: ereinecke

EatSafe

Description

EatSafe is an app aimed at travelers with dietary restrictions that need help understanding ingredients lists in foreign languages. Based on the Open Food Facts API (openfoodfacts.org), this app will allow a user to scan a barcode and get all available nutritional information. The user will be able to set preferences for specific allergies or sensitivities that they are managing, and those will be highlighted if found in the database.

The user should also be able to upload new products to the Open Food Facts database, along with ingredient data and any warnings they may have about it. A key feature to be implemented in a future release would be the ability to translate ingredient lists from labels.

Intended User

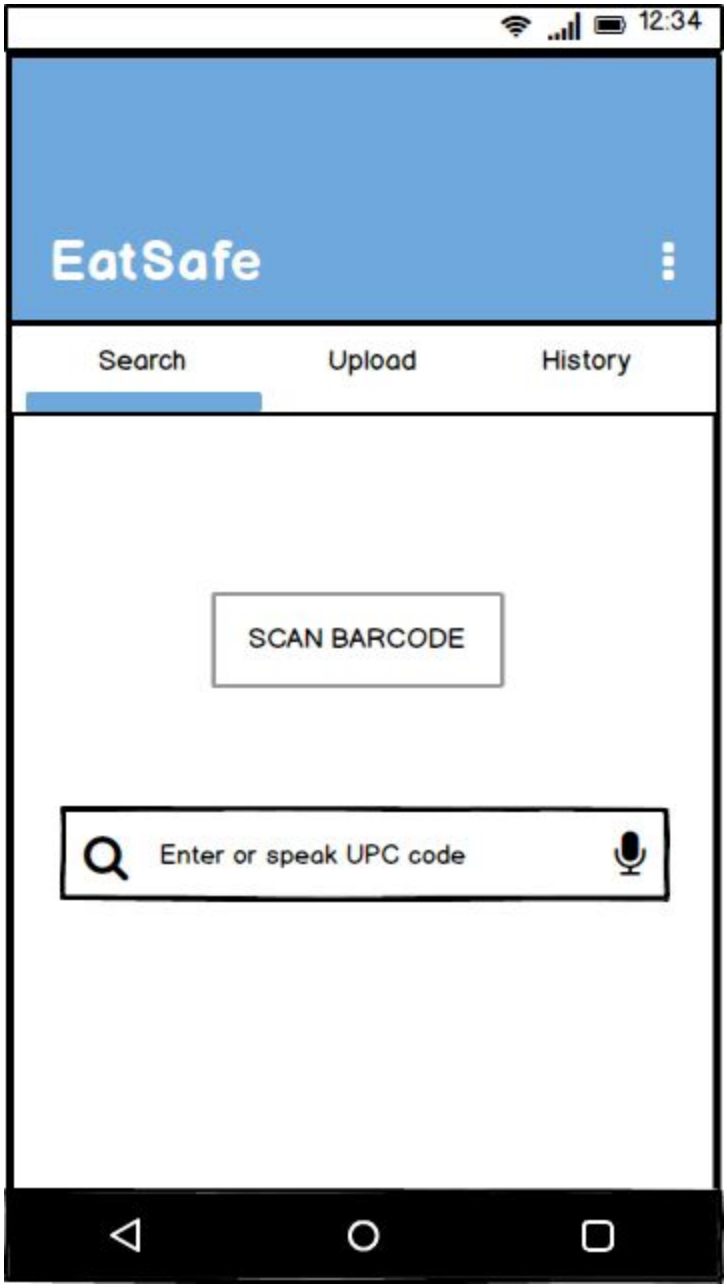
This app is aimed particularly at travelers and expats who have dietary issues and need to be able to understand labels in other languages. This can also be used by contributors to the Open Food Facts database to capture and upload product information.

Features

- Scans product barcode and downloads nutritional information from OpenFoodFacts.org database based on UPC code.
- Displays product ingredients and highlights allergens and additives.
- Allows users to submit product information, including UPC and photographs, if not found in the database.
- Allows users to review previously searched items when offline.
- Allows users to log in to openfoodfacts.org to upload data.
- Registration is handled by passing user to openfoodfacts.org/cgi/user.pl.
- A widget will launch directly into the scan activity.

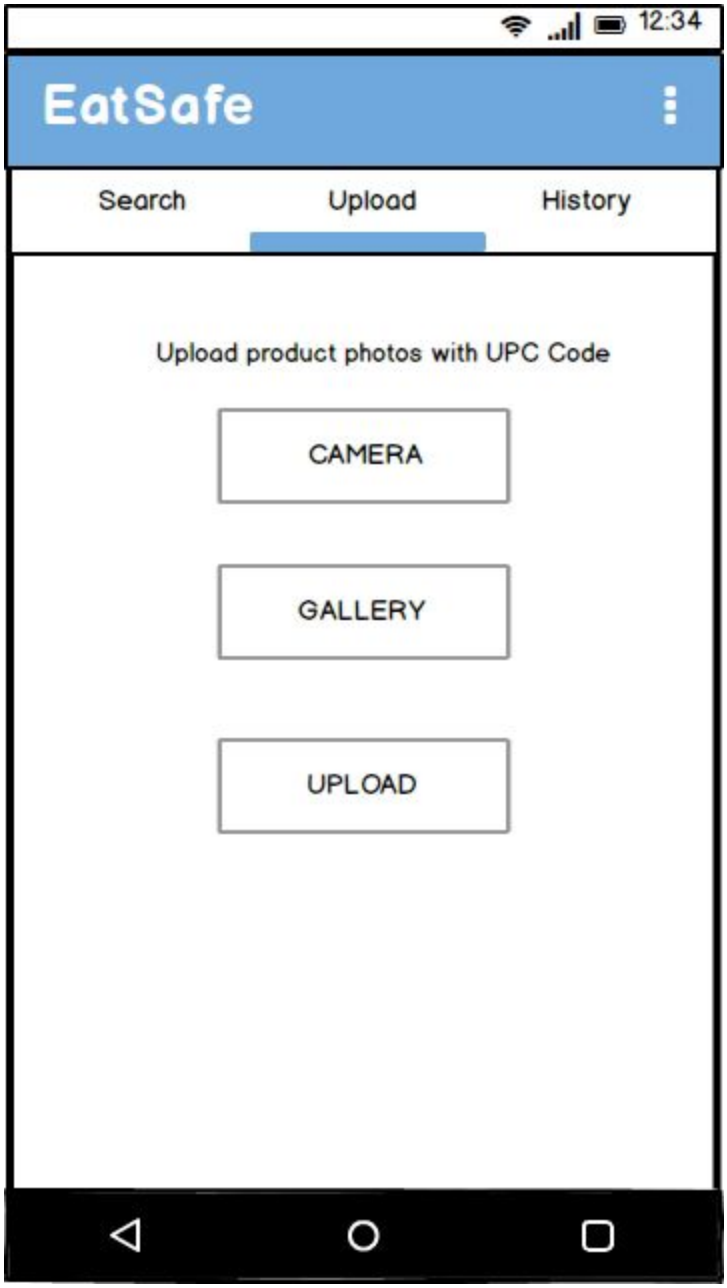
User Interface Mocks

Search Fragment



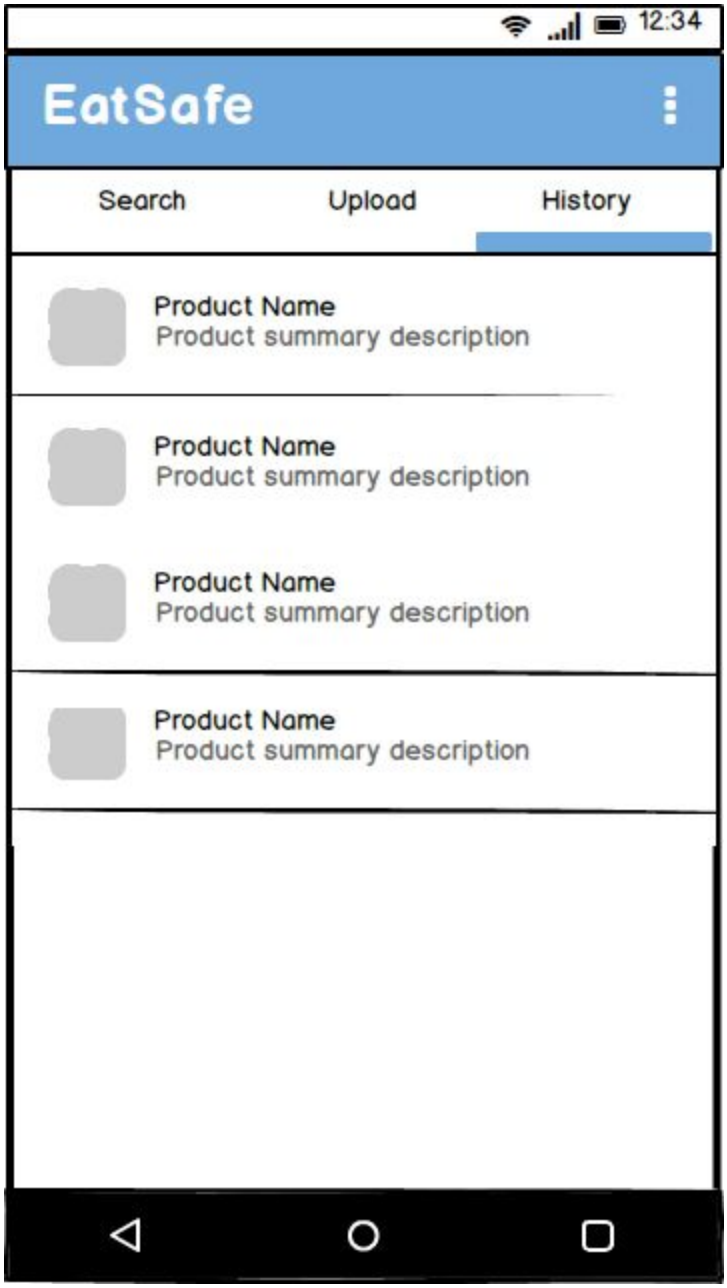
This fragment is the entry point, providing a button to launch a scanner as well as an entry field for a manually entered UPC code. This can also be entered verbally by pressing on the mic button.

Upload Fragment



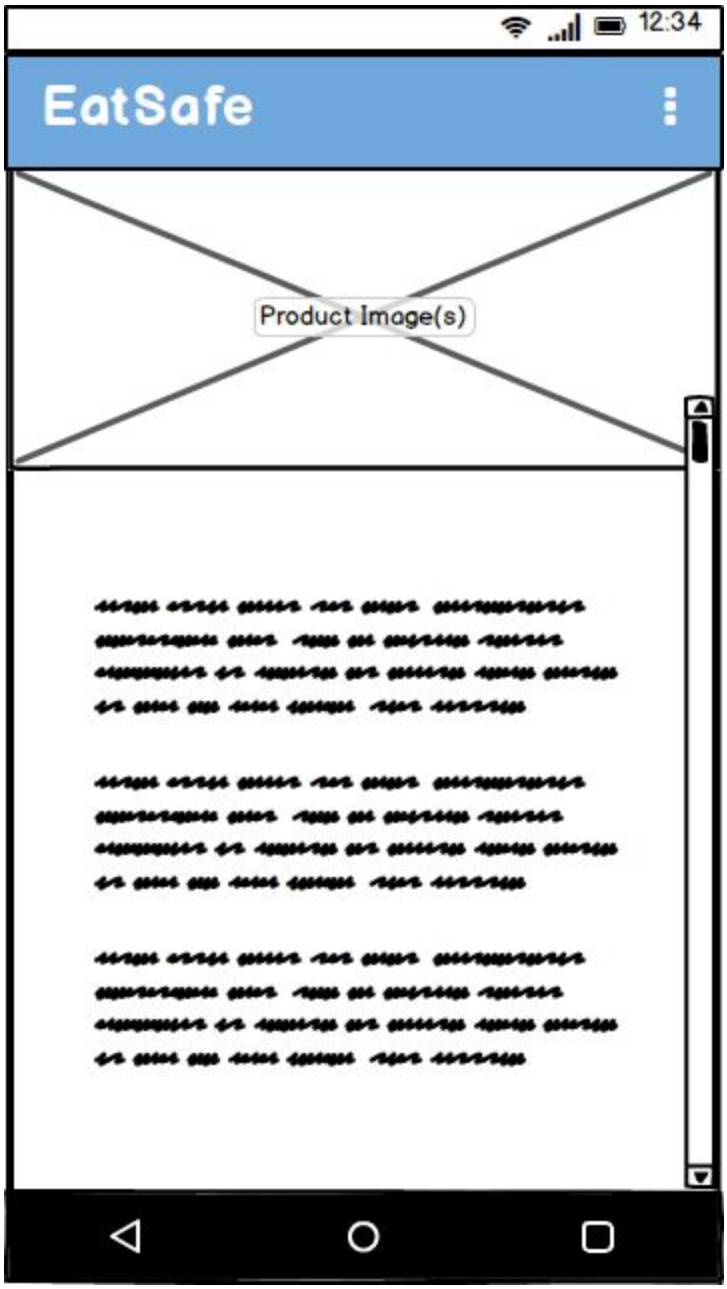
The upload fragment provides three buttons: one launches the camera, another lets a user select from their photo gallery, and the upload button will send the user to http://world.openfoodfacts.org/cgi/product.pl. If the user is not logged in, a message will display informing the user that they must register and log in to upload product info.

Results Fragment



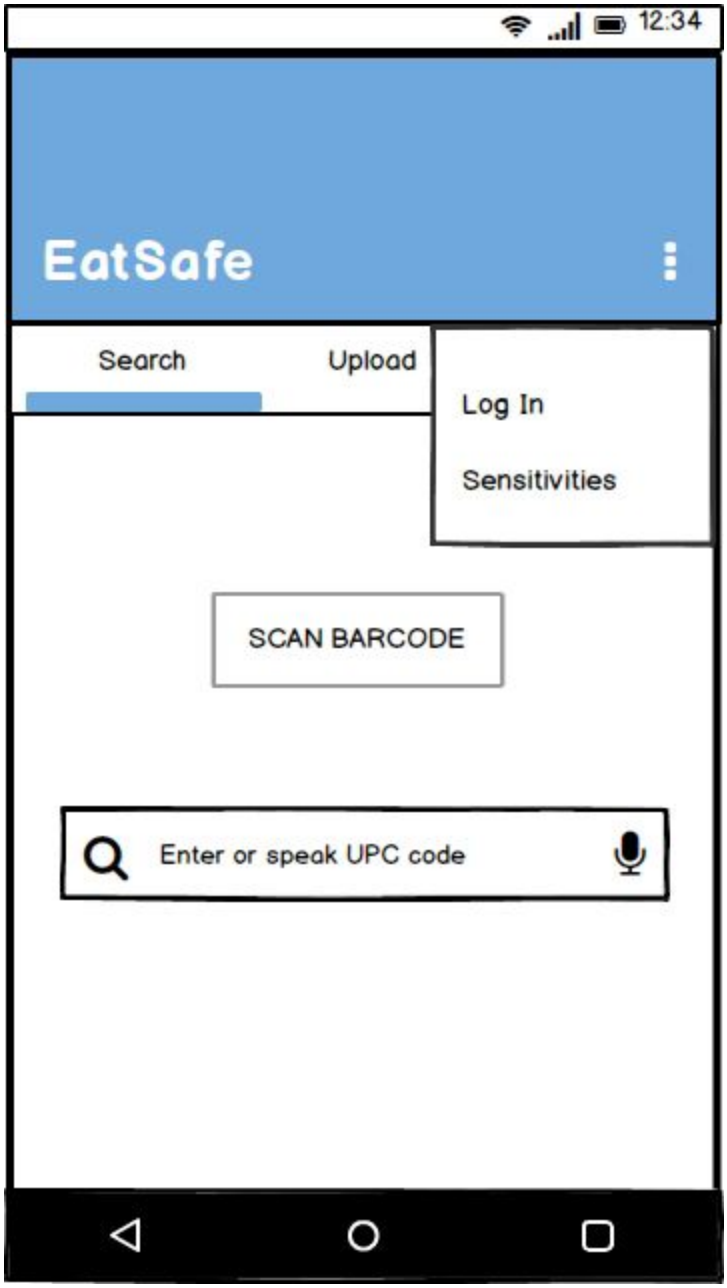
Items previously searched for will be stored locally and displayed in a listview in this fragment. Pressing on the list will launch the product info fragment.

Product Info Fragment



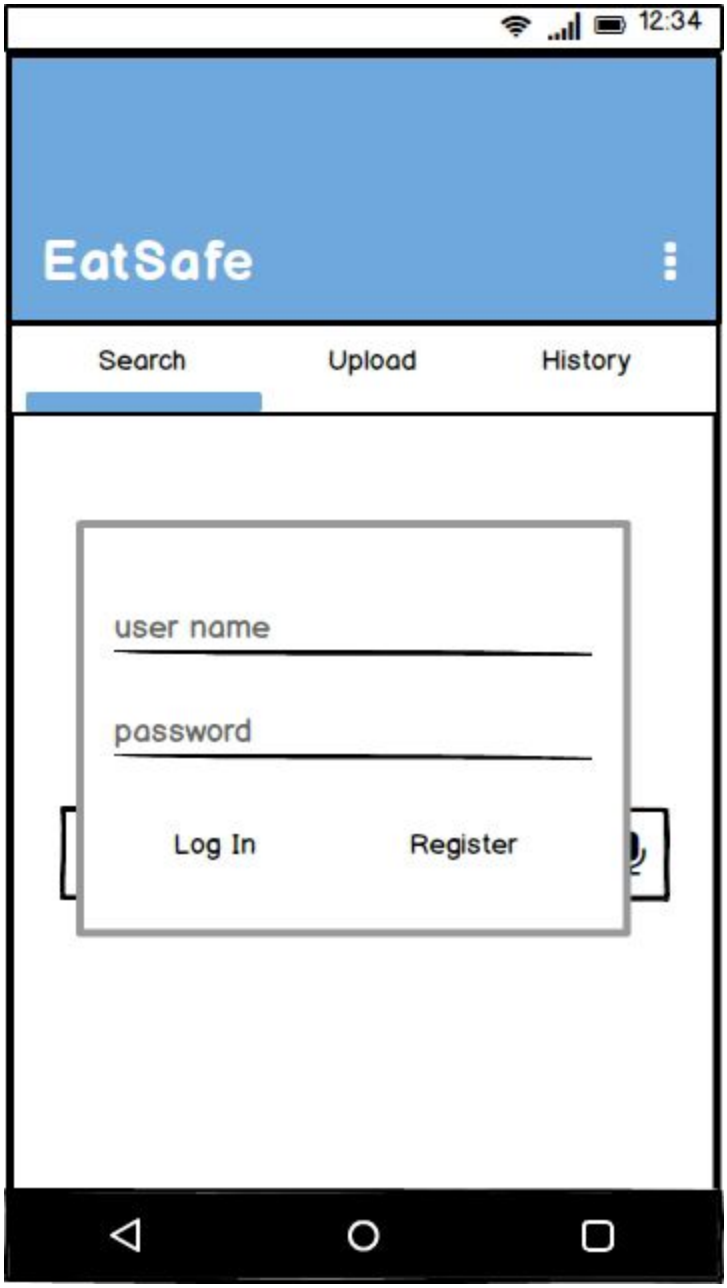
There are typically multiple product images - swiping left or right will reveal more photos, which will also scroll automatically. Product info will be in a ScrollView, consisting of summary, allergens, ingredients and nutritional summary.

Settings Menu



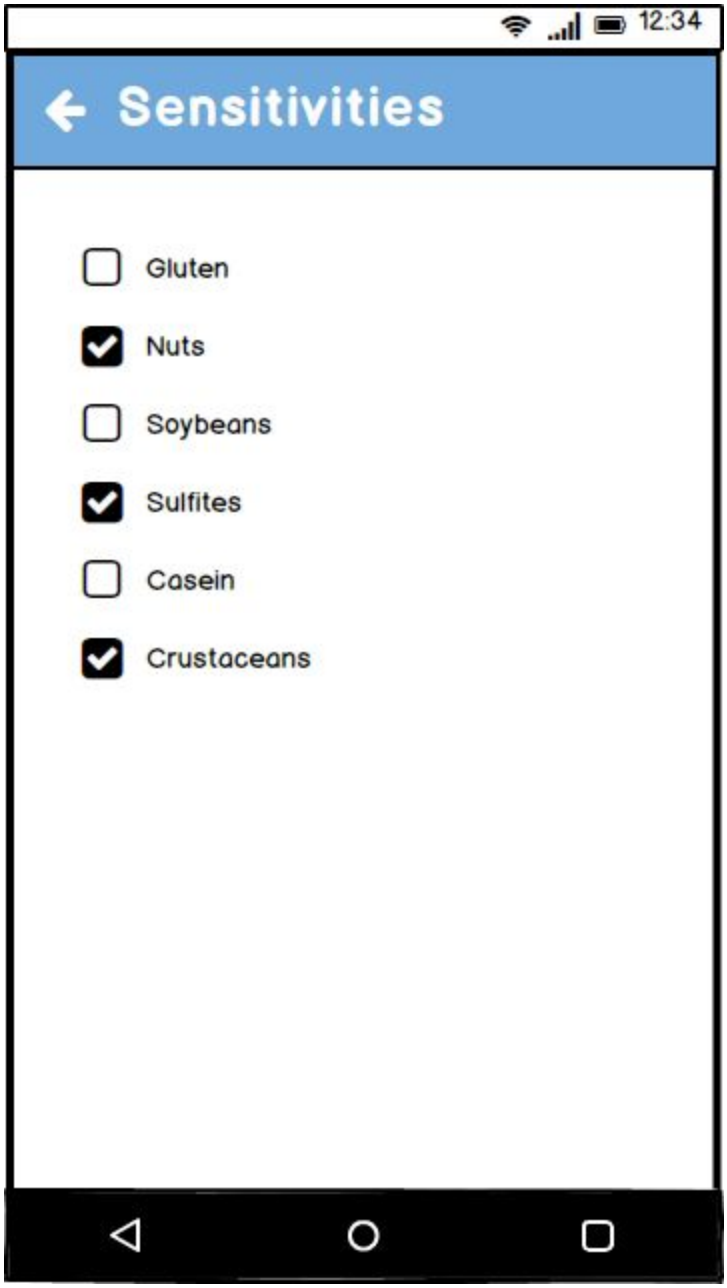
The Settings menu provides the option to log in to openfoodfacts.org and to tell the system what sensitivities you want to monitor.

Login DialogFragment



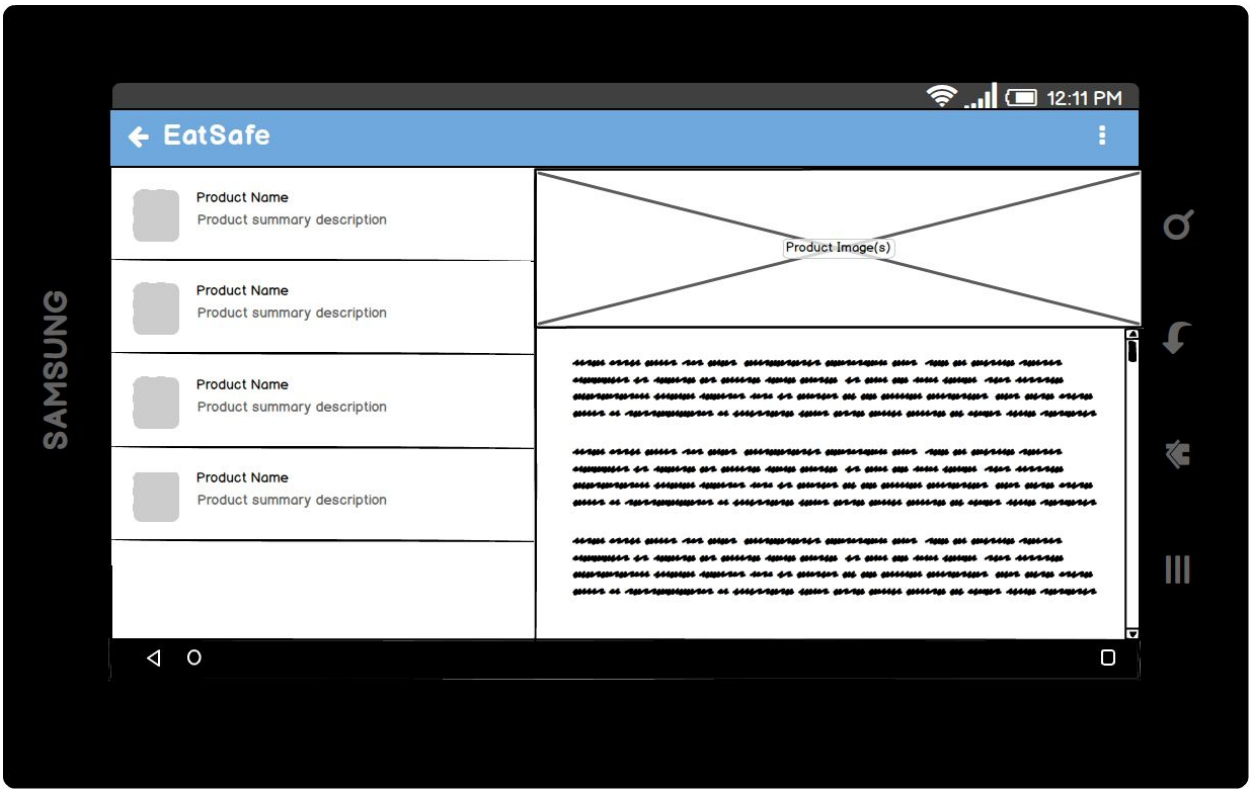
Dialog Fragment to log in into openfoodfacts.org. Clicking on Register takes you to http://world.openfoodfacts.org/cgi/user.pl

Sensitivities Fragment



The user can specify which ingredients they want to be alerted to; if present, the product description will highlight that with an icon.

Results Fragment and Product Info Fragment in Tablet mode



On tablets, the Results Fragment and Product Information Fragment are displayed side by side. Pressing on any result will cause the Product Information Fragment to display that information.

Key Considerations

How will your app handle data persistence?

Any product queried or uploaded will be stored in locally using a Content Provider to enable offline use. In a future revision, product information can be added offline and then uploaded.

Describe any corner cases in the UX.

- Searching for a product when offline should store the request (UPC code) and retry when back online.
- Uploading product information when offline should store information locally and upload when back online.
- If a product is not found from a successful API search, ask user if they want to upload product information.

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

For barcode scanning: ZXing (https://github.com/dm77/barcodescanner)
For JSON parsing: Jackson (https://github.com/FasterXML/jackson)
For asynchronous networking and image loading: Ion (https://github.com/koush/ion)
For image slider: LoyalNativeSlider (https://github.com/jjhesk/LoyalNativeSlider)

Google Play Services implementation.

Firebase analytics to collect events and user properties.
Firebase AdMob for advertising.

Next Steps: Required Tasks

Task 1: Project Setup

Write out the steps you will take to setup and/or configure this project. See previous implementation guides for an example.

- Configure libraries: Firebase, ZXing, Jackson, Ion, LoyalNativeSlider

Task 2: Implement UI for Each Activity and Fragment

- Build UI for Search Fragment
- Build UI for Upload Fragment
- Build UI for Results Fragment
- Build UI for Product Info Fragment
- Build UI for Sensitivities Fragment
- Build UI for Login/Register

Task 3: Build ContentProvider corresponding to Open Food Facts API

- Determine and configure OFF API endpoints
- Use an IntentService to pull data from API.
- Build a Product Model and corresponding Content Provider
- Retrieve data from local database when offline

Task 4: Configure intents for product upload and registration

User registration and product information upload will be handled by passing the user to their web browser to perform these operations on openfoodfacts.org.

Task 5: Implement barcode scanning

Implement barcode scanning. A successful scan will run a search on the API; if the search fails, the user will be asked if they want to upload product information. If so, launch the camera to capture images of the product.

Task 6: Implement image generation

The user has the option to upload product information, especially photographs, if the product is not found in OFF. The user can either launch the camera to take a photo or a gallery to select multiple photos.

Task 7: Implement SharedPreferences for user configuration

User configuration can include login information and dietary sensitivities.

Task 8: Implement analytics and AdMob

Configure analytics and AdMob using Firebase.

Task 9: Configure signing configuration

Configure signing configuration, with keystore and passwords in the repository.

Task 10: Implement widget

A widget will launch directly into the scan activity.