- Description
- Intended User
- Features
- User Interface Mocks
  - Main/Receipts Screen
  - Main Screen with Left-Hand Drawer
  - Statistics Screen
  - The “Did I Win?” Screen
  - The Settings Screen
  - The Map Screen (large screen)
- Key Considerations
    - How will your app handle data persistence?
    - Describe any corner cases in the UX.
    - Describe any libraries you’ll be using and share your reasoning for including them.
- Next Steps: Required Tasks
  - Task 0: Proof of Concept [DONE]
  - Task 1: Project Setup
  - Task 2: Implement UI for Each Activity and Fragment
  - Task 3: Create Database and Content Resolver
  - Task 4: Train the TessTwo Library for Receipt Font
  - Task 5: Implement action for FAB button in the Main/Receipts screen
  - Task 6: Data for Statistics
  - Task 7: Maps Integration
  - Task 8: AdMob Integration

GitHub Username: abunghez

Flotto

Description

Each month, the Romanian government organizes a fiscal lottery. Every fiscal receipt is a valid lottery ticket. If you hold a valid fiscal receipt from an extracted day of an extracted sum, you can win up to 1 million RON.
Flotto is an application designed to keep track of your fiscal receipts. It will scan the total amount spent, as well as the date from your receipt so that you can easily find out if you are a winner in this lottery or not. This saves you a lot of time of searching for a potentially winning ticket, especially if you don’t have one :).
FLotto also provides statistics with the amount of money you spend each month.

Intended User

The typical Flotto user is any person who collects fiscal receipts in order to take part in the government organized lottery.

Features

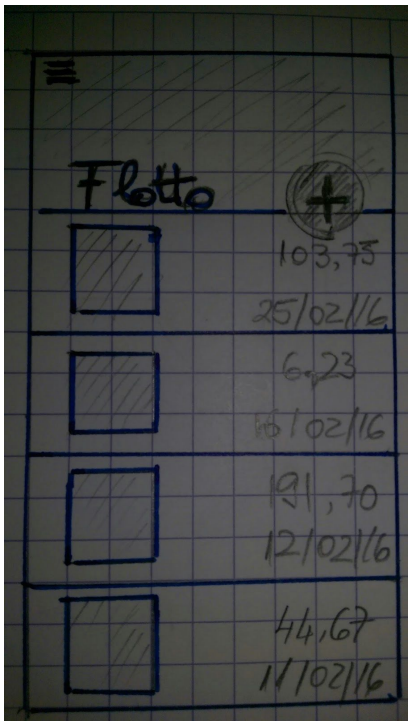
List the main features of your app. For example:

- Reads the total sum and date from your receipts using OCR.
- Based on the user's input of the winning day and sum, the app will let you know if the user has won or not.
- Tracks your receipts by total sum and date and creates statistics of your spending

Note: I will provide as many scanned receipts as needed in order for this app to be tested.

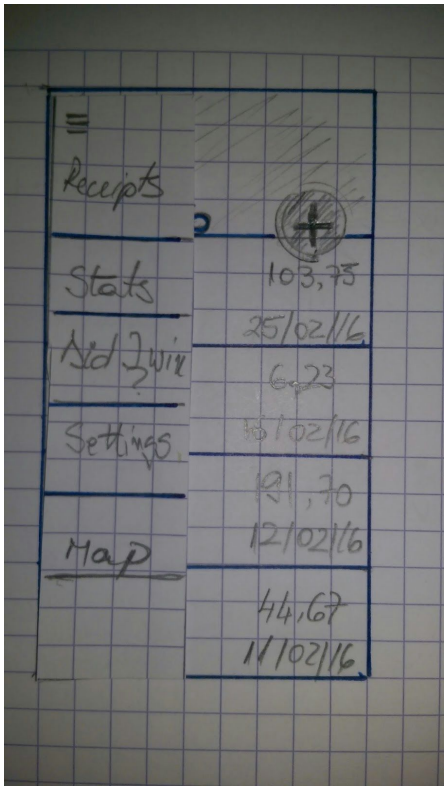
User Interface Mocks

Main/Receipts Screen

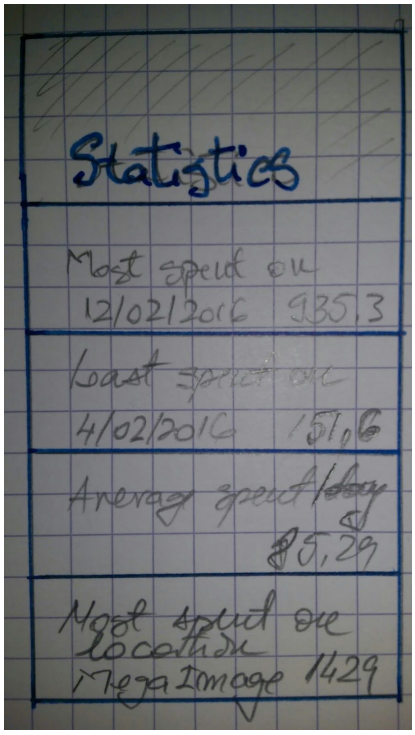


Main screen, displays list of receipts. Each list element include a thumbnail, the sum and the date of the receipt. Clicking on an element will display the full size image. The FAB button will add a new receipt, by starting the camera and taking a photo.

Main Screen with Left-Hand Drawer

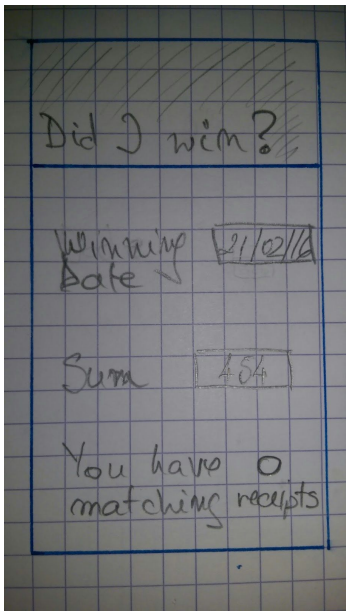


Statistics Screen



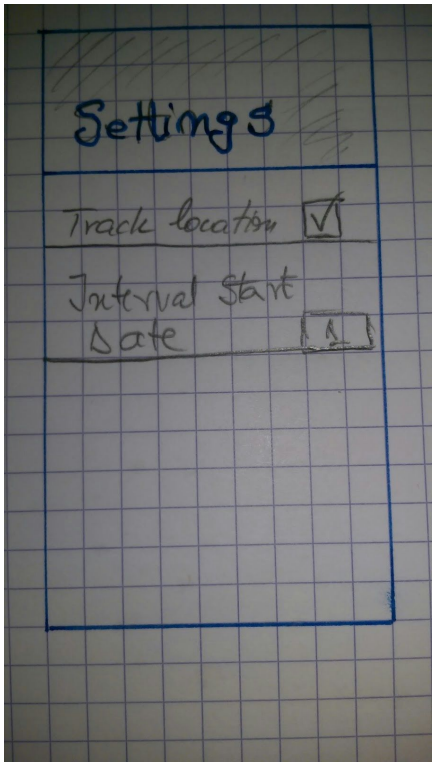
The statistics screen will display the days with most/least amount spent, average spending / day, the location where the user spends the most (in a month)

The “Did I Win?” Screen



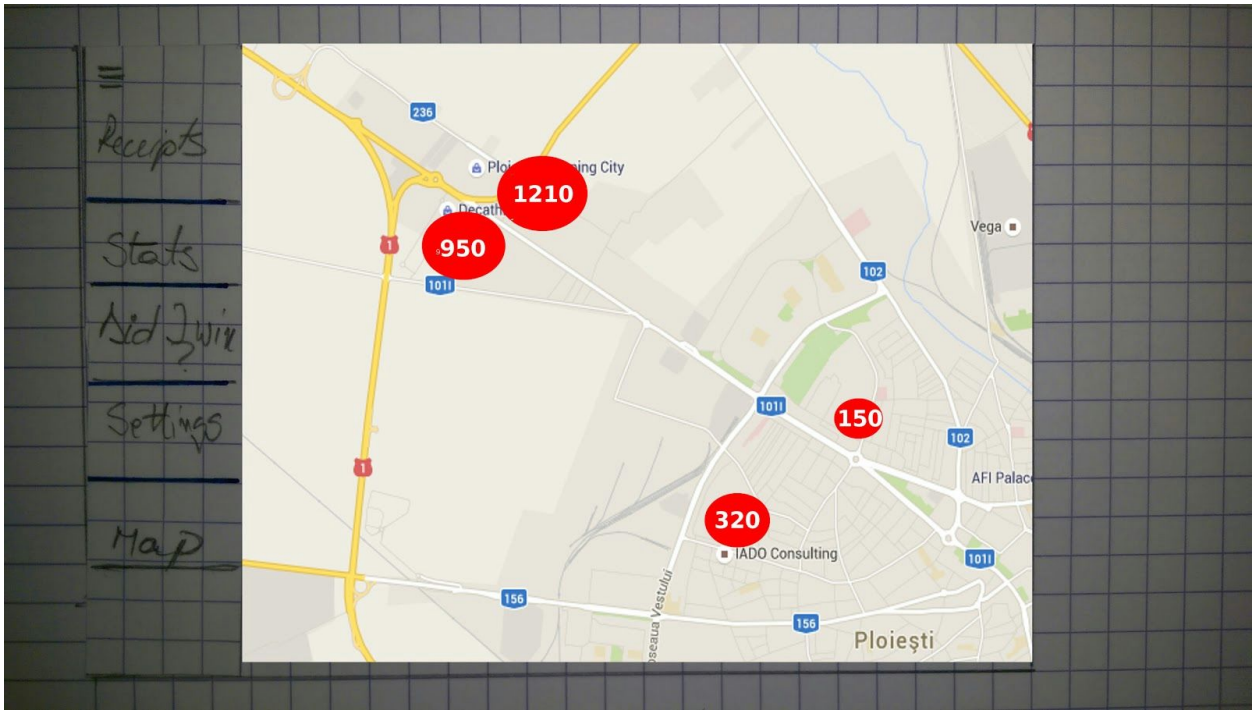
In the “Did I win” screen, the user will enter the winning date and sum. The app will return the number of matching receipts (usually 0, 1 if really lucky). In case there is one or several matching receipts, they will be displayed using a list with list items similar to the ones in the Receipts screen.

The Settings Screen



Currently plan settings are a checkbox for tracking location and the interval start date, probably the first of the month. If more settings are needed, they will be added here.

The Map Screen (large screen)



If the location is recorded for each receipt, the Map Screen will display the sums spent in a particular location (the sum of the amount spent in each receipt recorded from that location). For this to work accurately, the user would have to scan the receipt as soon as he/she receives it.

Key Considerations

How will your app handle data persistence?

For information about the receipts, the app will use its own ContentProvider. For settings, the app will use the SharedPreferences API.

Describe any corner cases in the UX.

- Incoming call while taking a picture of a receipt - The application will resume from the screen before taking a picture.
- No more space left on device - The app will notify the user there is no more space left for the picture.

- Receipt image does not yield a valid result through OCR - The application will partly rely on OCR data. Anything scanned will be confirmed by the user. The user has the possibility to alter the data received through OCR.

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

Picasso - for loading images.
TessTwo - com.rmtheis:tess-two - for OCR.

Next Steps: Required Tasks

Task 0: Proof of Concept [DONE]

- Check that OCR returns usable results when using the Tess Two library.

Task 1: Project Setup

- Configure libraries

Task 2: Implement UI for Each Activity and Fragment

- Build UI for MainActivity
- Build UI for ReceiptsFragment
- Build UI for StatsFragment
- Build UI for MapFragment
- Build UI for SettingsFragment

Task 3: Create Database and Content Resolver

- Build Database Contract
- Build ContentResolver
- Link Data to UI

Task 4: Train the TessTwo Library for the Receipt Fonts

- Take photos of as many receipts as possible
- The rest of the processes is documented here: https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract

Task 5: Implement action for FAB button in the Main/Receipts screen

- Initialize Tess API
- Use intent to take picture
- OCR through the tess-two library
- Scan result for TOTAL and DATA (date in Romanian) fields on the receipt
- Acquire location (if requested in the settings form)
- Ask user to confirm that the data is ok.
- Insert data into database

Task 6: Data for Statistics

- Compute minimum/maximum amount spent for each day.
- Compute average spending per day
- Compute maximum sum spent in one location.

Task 7: Maps Integration

Task 8: AdMob Integration

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"