CMU 2019/20	Mobile and Ubiquitous Computing	Class Project	
Mobile Application Development for Android		**Issued:**	06/Mar/2020
FoodIST: Finding Food on Campus		**Checkpoint:**	22/Apr/2019
Prof. Luis D. Pedrosa		**Due:**	15/May/2019
Prof. João C. Garcia		**Revision:**	1.0

1 Overview

In this project you will learn how to develop a non-trivial mobile application for the Android Operating System. This application will explore several key aspects to mobile development, including location awareness, judicious use of limited resources, and social behavior. Towards this end you will be developing a simple App to help your fellow students decide where to eat on campus: *FoodIST* .

The goal of the *FoodIST* application is to improve the experience of using the IST campi dining resources (cafeterias, bars, etc.). The app aims to provide real-time information about where to eat at each campus, what is on the menu, and what is the expected wait time in queue. Much of this information will be crowd-sourced, i.e. supplied by the users themselves, including the menu items and their respective meta-data (e.g. price) and photos.

The project will be built as the sum of a set of mandatory (Section 2) and additional (Section 3) components and features. The idea is that everyone implements the mandatory features (worth up to 75% of the grade), and then selects a combination of additional features that add up to complete the grade at 100%. Grades from the optional features are allowed to accumulate beyond 25% (and compensate for any limitations in the mandatory part of the submission), though the final grade is capped at 100%.

2 Mandatory Features

As the user opens the application, they should be given a list of dining options at their current campus. You may use the user's location to infer the campus but, if unsure, prompt the user to select one explicitly. The user should also have the option of changing the campus (via said prompt) and overriding any such inferences.

For each currently available food service, the app should show its name (e.g. Civil Engineering Cafeteria), the estimated walk time to get to it (using the Google Directions API, or equivalent), and the expected queue wait time (Section 2.1). Users should be able to setup a personal profile (accessible via a context menu) where they can indicate their status within the University (Student, Professor, Researcher, Staff, General Public), and the list of food services should be filtered accordingly to only show options that are available to the user at that point in time (considering the varying opening hours for each group).

Selecting a food service should bring up additional information about it, including:

- The name of the food service.
- Information about when it is open to the specific user.
- A map showing where it is located (clicking on the map should show the user how to walk there based on their current location).

- The currently expected wait time in queue.
- Today's menu (including the option to add new entries).

The menu should identify each individual dish and its cost. Clicking on a menu item should bring up additional specific information, including:

- The name of the dish.
- Its cost.
- Photos taken by users (and the option to submit new ones). Photos should be cached to avoid excessive use of data plans (Section 2.2). Tapping a photo displays it in full screen mode.

The list of food services, their locations, opening hours, and access restrictions should be statically defined but other information like menu items and photos are collected from the users themselves. Information pertaining to individual dishes should be associated with the dish and food service (the pork chops at the Math Cafeteria are very different from the pork chops at the Student Association Cafeteria and their photos and price shouldn't be mixed up). The information about the length of the queue is inferred indirectly via Bluetooth beacons (Section 2.1). The sharing of user-supplied data, including menu items and photos, is coordinated by a central server that is not the focus of this class but is nonetheless necessary for the project to work. Feel free to implement your own server as you see fit (e.g. using standard TCP sockets and Java Object Serialization, or gRPC), or otherwise use an existing server software or database. The server can be kept simple for debug and development purposes and can hold data in memory alone (no need for persistence). In any case, be prepared to justify your choices.

2.1 Queue Wait Time Estimate

Estimating the amount of time a user may expect to wait in queue can be a challenge. One way to address this is to monitor users as they arrive and leave from the queuing area to infer the wait time for each individual. This wait time, on its own, isn't immediately useful, as it can only be computed once the user has left the queue, but it can be used for a basic learning algorithm. Another useful metric is the number of users that are currently waiting in the queuing area. Assuming a constant queuing service rate (each person takes about the same amount of time to get their food), the estimated queue length can be used to infer the wait time, while also being measurable in real-time.

This is the approach that we will be exploring in this project. We will assume the cafeteria has installed in the queuing area a Bluetooth Low Energy (BLE) Beacon. In the real world, this can pose challenges, as these approaches need to assume that the beacon is carefully placed so that everyone waiting in queue can detect it and that once they have been served and seated, they are no longer in range. In practice, we will be emulating these beacons in software and therefore such concerns go beyond the scope of the project.

BLE Beacons are small simple devices that periodically broadcast a packet identifying themselves. The beacons themselves do not detect passersby but a user with an application that is aware of the beacons can use its proximity to trigger an event. In our case, this means that we cannot accurately infer how many people are waiting in queue, just how many of them have *FoodIST* installed. This is not a problem if we assume an even distribution of *FoodIST* users within the crowd. As such we will be inferring the queue time, based on the number of *FoodIST* users in queue.

Towards this end, the *FoodIST* application tracks when the phone comes in proximity to a known beacon and when it has left (it has not heard the beacon in a while). This information is reported to the central server which calculates how long each user waited in queue, and how many people where in the vicinity of the beacon at each point in time. After enough data is collected, a simple linear regression can be used to infer how long the wait is, given how many users are currently close to the beacon. This learning process should be done independently for each food service, as the queue service time, as well as the distribution of *FoodIST* users may vary between them.

2.2 Caching

Most of the data shared between the *FoodIST* application and its central server is small (short text strings) and time sensitive (I want the queue wait time now, not an hour ago). That said, user supplied photos do not fit this pattern. Photos are much larger, which can be taxing on a metered data connection, and also not of a timely nature. This makes this kind of data particularly well suited to caching and pre-loading.

Towards this end, the *FoodIST* application should build a local cache for photos. Whenever a photo becomes visible in the user interface, the application should first check the cache and only if the image is not present should it then download the data from the server. Once downloaded, the photo should remain in cache, in case it is needed again. The cache size should be capped at 100 MB and as new downloads exceed that quota, older items should be evicted (following a least recently used or LRU policy).

Furthermore, a cache aware WiFi pre-loading mechanism should be put in place that downloads the first few photos that show up for each food item, whenever the user connects to a WiFi network. With all of this in place, a user that just clicks on food items and doesn't scroll down far may never need to download a photo over their data connection.

CMU 2019/20	**Mobile and Ubiquitous Computing**	**Class Project**	
Mobile Application Development for Android		**Issued:**	06/Mar/2020
FoodIST: Finding Food on Campus		**Checkpoint:**	22/Apr/2019
Prof. Luis D. Pedrosa		**Due:**	15/May/2019
Prof. João C. Garcia		**Revision:**	1.0

3 Additional Components

In this section we list a series of features that can be combined to add value to *FoodIST* . Each feature lists the grading percentage it is worth, reflecting the relative difficulty in implementing it.

The project core (Section 2) is worth 75%, which leaves at least 25% for these additional components. Select a combination of these features that adds up to or exceeds that threshold.

Many of these features can be naturally integrated with each other and gain from being implemented in such a manner. Implementing your selection of additional features in such a cohesive manner is encouraged.

3.1 Securing Communication [5%]

Sending data across a network in the clear poses a significant risk to users. A malicious party can eavesdrop the data being transmitted and use it to infer all sorts of private information, including e.g. a user's location. Unprotected transmissions can also be modified in transit, feeding users fake data that can influence their decisions (e.g. make them eat at a bad place) or even put them in danger (give them directions to the middle of a construction site).

To avoid such scenarios, upgrade the *FoodIST* infrastructure to secure all communications between the mobile application and the back-end server to use SSL. Also, do be careful with how you manage your certificates to be sure the client only communicates with your authorized *FoodIST* server, preventing an untrusted party from conducting a man-in-the-middle attack.

3.2 Meta Moderation [10%]

Another way to attack the integrity of *FoodIST* data is for the attacker to simulate one or more valid users and to then simply feed fake data into the system. There is no easy way to completely solve this problem but a meta-moderation system raises the bar.

Create a mechanism whereby users not only submit data (menu items, photos, etc.), but can also flag them as inauthentic. Whenever a user-submitted data item is brought to main focus (the menu item information screen, or a photo shown in full screen), add the option to flag it. As items get flagged, keep track of the trustworthiness of individual items and also the users who submitted them. Use the trustworthiness of a user's recent submissions to set the initial trustworthiness of any new data they submit.

When showing user-supplied data, sort it by its trustworthiness and filter out any data below a threshold. This way, useful data percolates to the top making it more likely to be seen.

3.3 User Ratings [10%]

Not all food is created equal and price and queue wait times may not be enough information for a user to decide where to eat. Food quality also matters and can vary wildly from one food service to another.

Add the option for users to rate their meals on a 5-star system in the menu item information screen. As with photos and price, keep track of rating per menu item at each food service (again, the pork chops at the Math Cafeteria are very different from the pork chops at the Student Association Cafeteria). Use the average food rating for a given food service to also gauge the average rating for the service itself.

With this data in hand, include average ratings for food services and menu items whenever they are listed. Also include a more detailed breakdown of user ratings (e.g. histogram), in their respective information panels.

3.4 User Accounts [5%]

The project core (Section 2) does not include a login/logout procedure so users can be tracked via simple GUIDs. This simplifies the user experience, as users need not setup an account before using *FoodIST* but it also prevents users from using the application on multiple device while keeping their profiles and submission data consistent.

Allow users to create accounts and have their profile data synced on the server. Include Login/Logout buttons in the context menu. User accounts should be optional, so users can read data from the app without creating one, but submitting data should now require an account. If a user is not logged in when they try to do so, prompt them to log in and remember their credentials for the future.

3.5 Social Sharing [5%]

Another important aspect in mobile development is social sharing. It's often convenient to share information with friends to coordinate getting together to eat.

Add the option to share individual menu items or entire food services from within their respective information panels. This should allow users to use common social media (e.g. Facebook, Twitter) and communication applications (E-Mail) to tell a friend where/what they are thinking about eating.

3.6 Friend Tracking [10%]

Often social gatherings are not carefully planned out and a user might want to get an idea where their friends are without explicitly asking or sharing. Add the option for users to submit a selfie in their profile and add a switch to opt into sharing location when dining.

Use BLE beacons (Section 2.1) to place a user at a food service (while they are waiting in queue and for 30 minutes thereafter). Use this data to show a sequence of small photos of people currently eating at each food service. Show a small number of such photos in the food service list view and show the full list in the food services specific information view.

Keep track of who eats at the same places at the same times to predict how likely they are to be friends. Order the photos so the people most likely to be a user's friend show at the top.

3.7 User Prompts [5%]

Crowd-sourcing is a great way to collect large amounts of data and to tap into the wisdom of the crowds. Unfortunately bootstrapping the process can be a challenge as users often contribute more when more data is already available.

To nudge users towards submitting more data on *FoodIST* , implement a notification that prompts the user to submit data when it is most timely. Particularly, prompt the user to submit menu items once they join the queue (Section 2.1) and to take photos of (and rate) food once they leave it.

3.8 Localization [5%]

The IST community is diverse and multi-lingual and *FoodIST* should reflect that. Localization, also called L10n, is a collection of tools and techniques that allow different users that speak different languages to use the application and share data without barriers.

Translate all static strings presented to the user into at least two languages (e.g. English and your own native language) and store these strings in such a way that adding new translations is easy and does not require refactoring the application. When the user first uses the application, prompt them to select their language and remember this setting for later. Allow the user to change the setting in their profile, as well.

A database of static strings works well for menus and food service names, but cannot handle user submitted content. Use the Google Translate API, or equivalent, to translate all user submitted text to their chosen language. Clearly indicate that the new text is a translation and add the option to show/hide the original.

3.9 Dietary Constraints [10%]

Many users have dietary constraints that limits which menu items they can eat. For these users, having to wade through the full list of food service and menu items can be a tedious process.

Augment the *FoodIST* application to track which constraints each menu item adheres to and to allow users to specify their constraints in their profile. To keep things simple, include only the following static categories: Meat, Fish, Vegetarian, Vegan. As users submit menu items, they should select *one* of these categories. In their profile, they should indicate whether they can eat *each* of them.

With this information in hand, filter the food services and the menu items based on the user's dietary constraints. Whenever the filter has ruled out something, let the user know that some items have been hidden based on their diet and offer the option to temporarily show them.

4 Alternative Projects

Groups have the option to develop a different project if they so wish and with faculty approval. Consider this option if you already had a project in mind, whether for some collaboration with industry, to explore an idea for a start-up, or as a passion project. We might need to make adjustments to manage complexity, given the time-frame and effort allocated to the course. There are also some features that we consider essential to mobile development and we may need to add or adjust functionality to your idea for the sake of the class project. Start a discussion with the faculty and seek approval as early as possible if you wish to try this option.

5 Grading Process and Milestones

Projects will be evaluated on a variety of dimensions. The most important factors are: the degree to which baseline and additional features are implemented, how well integrated the features are, the technical quality of algorithms and protocols, and resource efficiency decisions. We will also assess the responsiveness and intuitiveness of the application interface. This is not a course in graphic design so, beyond basic utility and effectiveness, GUI aesthetics will not be graded. Do not invest too much time in creating pretty icons or other superficial assets and focus on making information accessible to the user. Consider using Unicode characters in place of hand drawn icons, when appropriate.

There are two important project milestones:

1. April 22nd: Project Checkpoint – Students should submit their current prototype so that they can receive feedback on the architecture and status of the prototype. Try to have most of the mandatory functionality (Section 2) ready, as well as a plan for the set of additional features you intend to implement. Consider including an Activity Wireframe and a prepared list of questions to focus the feedback.

2. May 15th: Project Submission – Students should submit a fully functional prototype and a final report. All source code and the report must be submitted through the course Fénix website. A template of the report will be published on the website. The report must describe what was and was not implemented, and it is limited

to 5 pages (not including the cover). The cover must indicate the group number, and the name and student ID number for each of the group's elements.

6 Collaboration, Fraud and Group Grading

Student groups are allowed and encouraged to discuss their project's technical solutions without showing, sharing, or copying code with other groups or any other person, whether attending the course or otherwise. Fraud detection tools will be used to detect code similarities. Instances of fraud will disqualify the groups involved and will be reported to the relevant authorities. Furthermore, within each group all students are expected to have a working knowledge of the entire project's code.