TÉCNICO LISBOA

Mobile and Ubiquitous Computing — 2021/22		**Assignment:**	Project
Mobile Application Development for Android		**Issued:**	2022-05-09
ConversationalIST: Conversations on the Go		**Checkpoint:**	2022-06-03
Authors:	Prof. Luis D. Pedrosa Prof. João C. Garcia	**Due:**	2022-06-24
		Version:	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 of mobile development, including location awareness, judicious use of limited resources, and social behavior. Towards this end, you will be developing a simple but powerful messaging App: ConversationalIST.

ConversationalIST should work like many other text-based chat applications, with features akin to IRC, Slack, and Discord. User communication is centered around chatrooms, which users can create, join, participate in, or leave. Once a user has access to a chatroom, they can participate in the shared conversation and review older chat history, even retroactively prior to joining. Conversations can include a variety of media to aid in user communication, including text, photos, files, polls, and geographic locations.

The project will be built as the sum of a set of mandatory and additional 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

The mandatory features include the core messaging functionality. Though the exact UI design is up to you, the app should support the following features for conversations:

- Allow users to pick a unique username or handle to identify themselves in a conversation. This user ID should then be used to clearly mark their contributions to the shared conversation, along with additional meta-data like timestamps.
- Users can create new chatrooms or join existing ones. The user should have easy access to a list of chatrooms they have joined, allowing them to easily check which have unread messages and to quickly switch between them.
- After selecting a chatroom, users should be able to see the content published there. Chatrooms can be seen as living append-only documents in that once a user has access to the chatroom, they should be able to see all of its past content, including content submitted before they joined.
- Users should be able to submit a variety of simple media, including:
  - Simple text messages.
  - Photos taken from the phone's camera.
  - Geographic locations, to be shown as an embedded map with a button to request directions to the given location (e.g. by opening up Google Maps or other similar application). The user sharing a location should be able to

specify the location by selecting it on a map, searching an address, or using the phone's current location.

- If a message is sent to a chatroom the user has access to but the user is not actively watching, notify them via a Notification. Tapping the notification should take them directly to the specific chatroom to see the new messages.

2.1 Back-end

ConversationalIST supports a number of features that build on explicit data sharing and crowd-sourcing among multiple devices. To enable such functionality you will need some sort of back-end service that holds and processes shared data (e.g. chatrooms) and that each device communicates with to synchronize its state.

How you implement this back-end service 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 Java RMI, gRPC, or a RESTful service), 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.

2.2 Resource Frugality

When using real-time data in applications – such as instant messaging – developers often have to contend with a trade-off between timeliness and efficiency. It's nice if the user gets notified the moment a message is sent to them, but constantly polling the server can drain the battery and use up the user's metered data.

In ConversationalIST, conversations should sync across users and devices in a timely manner while also using the network efficiently. When the user is actively viewing a chatroom, ensure that any new content shows up quickly. If the user disengages from the application, use more efficient messaging to save network resources, even if at the expense of increased latency.

It's also important to avoid using resources unnecessarily. As conversations in a chatroom grow overtime, users are less likely to scroll up to older messages. Avoid wasting resources by only downloading data related to UI elements as they become visible to the user.

Particularly large content can be further optimized to avoid costly metered data. In ConversationalIST photos can represent a hefty data cost so, to optimize network usage, show a placeholder for them when the user is on a metered connection, retrieving the image only when the user taps it. If, on the other hand, the user is on WiFi, automatically retrieve photos when visible.

2.3 Context Awareness and Privacy

Access to chatrooms can be restricted to keep conversations private or in context. Three modes should be supported:

- *Public* - Any user can search for the chatroom by name and join.
- *Private* - Users must first use an Android App Link to join the chatroom. Anyone already in the chatroom can share these links using the Android simple data sharing API (share via SMS, social networks, E-Mail, etc.)
- *Geo-fenced* - Users can only participate (join/view) in the chatroom when located within a specific region defined by the chatroom creator. The creator picks a point (on a map, by address, or using their current location) and a radius. Other users within that radius can then search and join the chatroom and participate. Leaving the area blocks access, even for those who have already joined. Re-entering allows access once again, without needing to rejoin.

2.4 Caching

Often users have only spotty data-connections with metered data. As such, communication between the ConversationalIST application and its back-end server should be optimized to use the network judiciously and to compensate for short term outages. On one hand you should avoid downloading the same content multiple times when it could reasonably be avoided, on the other you want to minimize disruption during a momentary outage.

To address this challenge, use a cache to store content as you retrieve it from the server. With this in place, repeated downloads of images are minimized and any content recently viewed will be available offline if needed.

Further optimize your cache through careful pre-loading when the user connects to WiFi. As WiFi data is virtually free, use the opportunity to load the most recent content (the content immediately visible when opening the chatroom without scrolling) for each chatroom the user has access to. This way, later when the user no longer has WiFi, they can still browse all of their chatrooms with minimal data usage.

3 Additional Components

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

The project core (described above) 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. putting words in people's mouths) or even put them in danger (give them directions to the middle of a construction site).

To avoid such scenarios, upgrade the ConversationalIST 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 ConversationalIST 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 ConversationalIST data is for the attacker to simulate one or more valid users and to then simply feed bad 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 can flag inappropriate messages in chatrooms. As users flag messages, keep track of how many times each message has been flagged by distinct users. When a message is flagged, immediately hide it from the flagger's view, providing immediate feedback that their concern has been registered. As more users flag the message, hide it from everyone once a given threshold has been surpassed.

Also keep track of how many messages have been blocked in this manner for each poster in each chatroom. If a particularly obnoxious user has had enough of their messages blocked in a given chatroom, prevent them from submitting further content to the chatroom.

3.3 User Accounts [10%]

The project core 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 ConversationalIST but it also makes it more difficult to keep multiple devices consistent.

Allow users to create accounts so that they can login from multiple devices and seamlessly keep their list of chatrooms in sync. Include Login/Logout buttons in the context menu. Multiple devices logged in for the same user should automatically sync the same chatrooms, providing a seamless cross-device experience. Also allow users to share private chatrooms by adding and removing other users via a simple access control list. User accounts should be optional, so users can start using the app without an explicit account and later upgrade to using an account if they want.

3.4 Additional Media: Files [10%]

Though a picture may be worth a thousand words, sometimes that is not enough. In addition to text and photos, let users add files selected from local storage to the conversation in ConversationalIST. These files should show up in the middle of the conversation with a place-holder and should only be downloaded on request. Once downloaded, use the appropriate system API to open the file with a compatible application (e.g. a PDF viewer).

3.5 Additional Media: Polls [10%]

Often users want to reach consensus on a shared decision and text may not be the easiest way to tally opinions. In addition to text and photos, let users add polls to the conversation in ConversationalIST. When publishing the poll, the user indicates the question and a series of possible answers for everyone in the chatroom to vote on. The poll then shows up in the conversation at the point when it was published, but should update overtime and show running tallies as a simple bar plot[1]. Keep track of who voted for what on the back-end and if the same user votes again, it replaces their previous vote.

3.6 Social Sharing To Other Apps [5%]

Another important aspect in mobile development is social sharing. It's often convenient to share information with friends in context and directly from within an application, without needing to open a separate communication app to do so.

Add the option to share content from ConversationalIST with other apps, e.g. common social media (e.g. Facebook, Twitter) and communication applications (E-Mail). This includes not just messages in chatrooms (text and photos), but also access links to share access with entire chatrooms as well.

[1] Feel free to use publicly available libraries to render the plots.

3.7 Social Sharing From Other Apps [5%]

Its also nice if its easy to share content from other applications into ConversationalIST without needing to copy and paste or to coordinate file locations by hand. Register ConversationalIST to show up as a target for sharing in other applications. This is particularly useful for sharing files and photos from e.g. the camera or file explorer. When the user shares a resource with ConversationalIST, bring up a menu to select which chatroom to submit it to and then leave the application open in that chatroom with the content already submitted.

3.8 Localization [5%]

Users are diverse and multi-lingual and ConversationalIST 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. A database of static strings works well for menus and buttons, but cannot handle user submitted content. Use the Google Translate API, or equivalent, to translate user submitted text messages to their chosen language. When showing text like this, clearly indicate that the new text is a translation and add the option to show/hide the original.

3.9 UI Adaptability: Rotation [5%]

Users don't always use their phones with the same orientation and can rotate to view content either vertically or horizontally, as they see fit. Allow ConversationalIST to adapt to these circumstances and make sure all screens show correctly given the current orientation. Make sure the user can experience the application equally well with the phone oriented vertically or horizontally.

3.10 UI Adaptability: Light/Dark Theme [5%]

Though many applications default to a light theme with dark text on a light background, many users prefer a dark theme which is less harsh on the eyes, especially at night, and mitigates burn-in on OLED screens. The Android Operating System has native support for dark themes and lets the user pick between a light and dark mode within the phone settings. Applications can then adapt accordingly, using an appropriate theme to ensure a consistent look-and-feel with the rest of the phone UI.

Implement both a Light and a Dark theme in ConversationalIST. Make sure the user can experience the application equally well in either mode.

3.11 Recommendations [10%]

Finding relevant chatrooms that match users' interests is often a challenge but users typically share common interest profiles which can be crowd-sourced and used to suggest new chatrooms that might interest them. The family of algorithms that power these suggestions is known as *recommender systems* and they power all kinds of recommendations, from e-commerce ("people who bought X also bought Y."), to dating apps, and much more.

Build a simple recommender system for chatrooms and add a new dialog to ConversationalIST that suggests new chatrooms to the user that may interest them. This isn't an ML class so we can use a simple crowd-sourcing algorithm on the back-end: for every pair of chatrooms, track how many users have joined both of them. Whenever a user joins a new chatroom, iterate over each of the others they have already joined and increment the count on the back-end. When the user opens the recommended chatrooms dialog, prepare a list of recommendations on the back-end by going over all of the channels they have not yet joined and count how many users they have in common with the chatrooms the user has already joined. Sort the channels by this metric and present to the user the top few (enough to fit the screen), which are most likely to be the ones they will be interested in.

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 in your idea for the sake of the class project. You can also propose additional features to this project, if you so wish. 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 the specification is implemented, 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:

June 3rd: 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 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.

June 24th: 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 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.