CS-E4100 Fall 2018
Mobile Cloud Computing (5 cr)

Group Project – Real-time multimedia chat

Deadline **Sunday, 9 December 2018 at 23:59** (local time in Finland)

Submission Through the Aalto Version Control System (`https://version.aalto.fi`)

Revision `v1.1 (05.11.2018)`

Projects must be submitted according to the instructions in this document (see the "Submission" section for the details). No other form of submission will be accepted. Deadlines are hard. No extensions will be granted. Should you have any issues in accessing MyCourses, A+ or Aalto Version Control System, contact the course staff at `cs-e4100@aalto.fi` as soon as possible. Also refer to the MyCourses workspace for additional information (`https://mycourses.aalto.fi/course/view.php?id=20578`).

Overview

Summary The purpose of the group project is to realize a real-time chat application that supports both one-to-one and one-to-many (i.e., group) communication. The chat conversations must support both text messages and pictures. Additionally, the pictures shared in the chat messages have to be displayed in a gallery and categorized based on their features.

Components The application consists of two components: a frontend and a backend. The *frontend* is the user interface running on a mobile device and must be implemented as an Android application. The *backend* consists of the database and logic for processing and storage of messages. The backend will rely on Google Cloud's backend-as-a-service solution called Firebase (`https://firebase.google.com/`). It provides several products to easily create scalable backend services. Among the available products, you will primarily use Firebase Realtime Database, Cloud Storage, Cloud Functions, MLKit and Authentication.

Basic features

Frontend The frontend *must* be implemented as a *native Android application* and *must* support the following features.

- *User authentication.* The frontend should first show a login screen prompting for password-based authentication or signing up a new user. This can be achieved by using FirebaseUI Auth (`https://firebase.google.com/docs/auth/`) and any of its built-in features. The user information must contain a *display name* which uniquely identifies a user. Users may also provide an optional profile image during sign-up. Upon successful login, the frontend should show a list of the user's chats.
- *Profile settings.* Users must be able to edit their own profile through the application, i.e., they should be able to change their profile picture or edit their display name.
- *Search for users.* The frontend should allow a user to search for other registered users with their display names. Any sensible criteria can be used to implement the search functionality as long as all users are not displayed by default. For instance, an implementation could require that at least the first five characters of the display name are entered before displaying the list of names.

- *Initiate a chat.* Users can initiate one-to-one or group chats. The group chat functionality should allow users to be added at any time. A new user added to an existing group chat should have access only to the messages sent after the time he (she) joined the group chat. Users can belong to multiple chats at the same time.
- *Messaging.* Users should be able to send and receive messages containing either text or images. The application must provide the user with two options to send images: take a picture with the camera of a smartphone or pick a picture from the phone's gallery. The frontend should provide a live camera preview when taking a picture with the camera.
- *Notifications.* The application must notify users when a new chat message is received or when they are added to a new group chat. The notifications must be received even if the application is in the background. This can be achieved by using Cloud Functions on the backend.
- *Image resolution settings.* A settings screen should allow users to specify different resolutions for images. In particular, it allows to specify the image quality when uploading or downloading pictures. The image quality can be *low* (640×480 px), *high* (1280×960 px) and *full* (original size) resolution. The same resolution setting applies for both uploads and downloads. For instance, let's consider a scenario where user A sets the resolution to full and user B sets the resolution to low. When user A sends a picture, the full resolution version is uploaded. However, user B receives (or automatically downloads) only the low resolution image. When user B sends a picture, only a low resolution image is uploaded and user A can only receive the low resolution image. Note that the different resolution images are generated in the backend, Refer Section "Requirements" under "Backend".
- *Gallery of chat images.* Each chat (individual or group) should have a menu option to display all pictures from that chat as a gallery. The default view must be a gallery of pictures, where the pictures are sorted by recent date. Furthermore, there should be two options to view images – grouped by either *sender* or *image features* (refer to the sample user interface at the end of the document). When grouped by sender, the display name of the sender(s) followed by his/her images should be displayed. On the other hand, if the images are grouped by features, the gallery should display the pictures grouped by the following categories: *Food*, *Technology*, *Screenshots* and *Others* (i.e., those that do not belong to any of the previous categories). You may find that you need to use MLKit with *cloud-based image labeling* to categorize images based on these features. Additionally, the application must provide an option to open the picture (in portrait and landscape mode) and support zooming. In addition, the interface should allow to download the picture in full resolution (if available).
- *Leaving a chat group.* A user should be able to leave a group chat. No new messages or notifications from the group should be received once the user leaves it.
- *Responsive user interface.* The application must handle all operations without slowing down or freezing the user interface. Processing and downloading should be done in the background. Additionally, the app should properly manage loading and displaying large images without running out of memory.

Backend The backend must be realized according to the following requirements.

- *Data storage*. The Firebase Realtime Database (*the recommended option that is supported by the course staff*, `https://firebase.google.com/docs/database/`) and Cloud Storage (`https://firebase.google.com/docs/storage/`) should be used. The newer Cloud Firestore (`https://firebase.google.com/docs/firestore/`) could also be used, but it is discouraged at it has not been tested by the course staff. The realtime database is used

to save chat room information, messages and synchronize data across devices. Cloud Storage is used to store the pictures and any other user-generated content if required.

- *Access rules*. Only authorized users should be allowed to access the chat-related information on the database. Specifically, information about a chat room – including participant(s), messages and user-generated pictures – should be only accessible to members belonging to that particular chat. Moreover, the pictures in Cloud Storage should not be available publicly.
- *Serverless paradigm*. Cloud Functions should be used to send notifications to users when a chat message is received as well as to generate images of different resolutions. The function is triggered whenever an image is uploaded to Cloud Storage and generates the different versions of the same image with the desired resolution (Refer to Section "Image resolution settings" under "Features").

Advanced features

Description Implementing the following features *is not required* to complete the project but gives extra points. You can pick any combination of the features listed below. Even though the project itself will be given the extra points for correctly realized features, **the total number of extra points considered for grading is limited to a maximum of 20**.

- *Picture expiration* (**5 points**). Pictures expire after a certain time that can be configured by the system administrator. After expiring, pictures are removed from the Cloud Storage. Users trying to download or access these within the chat after their expiration should receive a "Picture no longer available" message. All pictures of a chat room also expire as soon as all members have left a chat room. Local copies at the mobile devices have no expiration.
- *Support for themes* (**5 points**). The frontend should support switching between different themes. A theme consists in a combination of colors, fonts, and a chat background (either a solid color or a background image). There should be a "default" theme and at least another theme. For instance, you can make a "light" and a "dark" theme. Themes can be selected from the settings screen.
- *Emoticon library* (**3 points**). Add an UI element that allows to insert custom emoticons from a pre-made library. Emoticons should be grouped by feature or theme. Animated emoticons should be supported.
- *Link preview* (**3 points**). Automatically add a preview to messages containing URLs. The preview should include the title and a screenshot (or some representative image) of the linked webpage.
- *Speech input* (**5 points**). Use the Google Speech-to-Text API (`https://cloud.google.com/speech-to-text/`) to allow sending text messages by talking while using the app.
- *Text processing* (**5 points**). Use the Google Cloud Translation API (`https://cloud.google.com/translate/`) to offer translated version of the text in the chat messages. Translation can be performed on-demand (i.e., by selecting a UI resource corresponding to a specific message) or automatically (i.e., to a target language for each message that contains text other than in that language). Automated translation and the target language should be configured from the settings screen.
- *Resource sharing* (**3 points**). Enable sharing of images or text messages to other chats or to social media.
- *Web frontend* (**7 points**). Write a web frontend that offers the same basic features of the Android application.

Submission

Instructions The project source code must be submitted through the Aalto Version Control System (`https://version.aalto.fi`) on the git repository associated with your group. Additional information about using the Aalto Version Control system and the group-specific repositories is available in MyCourses under the "Software project" section (`https://mycourses.aalto.fi/course/view.php?id=20578&section=3`). No explicit action is required to submit the code: **the last commit before the submission deadline will be used for evaluation**.

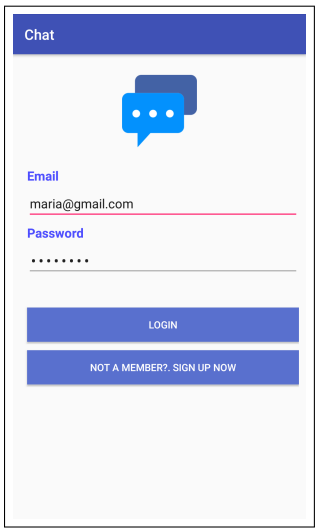
Evaluation Each group must show a demo of the software project to the teaching assistants. A time for the demo has to be booked according to the instructions available in MyCourses under the "Software project" section (`https://mycourses.aalto.fi/course/view.php?id=20578&section=3`). The evaluation criteria for the software project are available in MyCourses under the "Software project" section (`https://mycourses.aalto.fi/course/view.php?id=20578&section=3`). Note that these criteria apply for the basic features of the software project.

Revision history

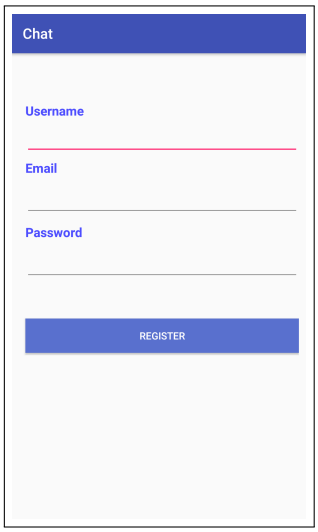
1.1 Clarified addition of users to group chats under "Initiate a chat".

1.0 Initial (non-preview) version of the project description.

Sample user interface

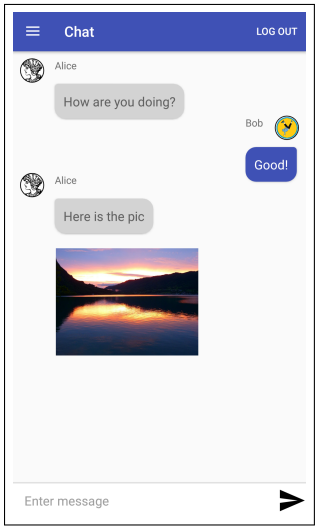


Login

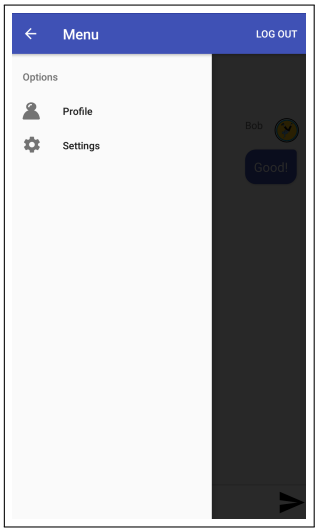


Sign up

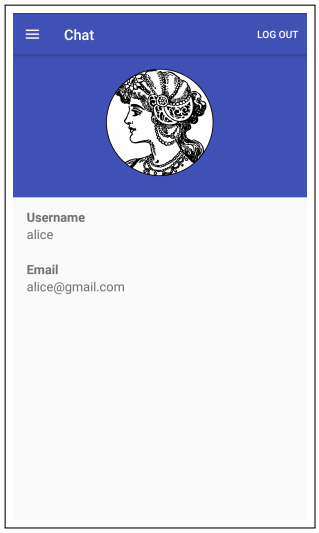
Chat list



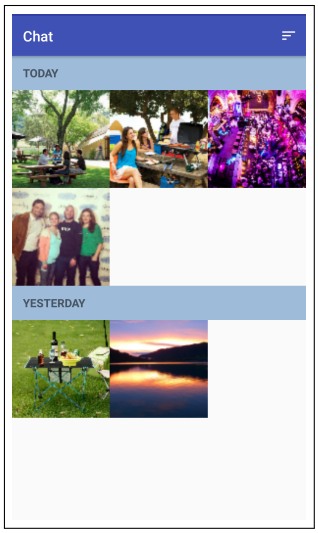
Group chat



Settings



User profile



Pictures sorted by date

Pictures grouped by features