LAM Project 2022

Federico Montori, Luca Sciullo

April 2022

Rules

In this document we describe one possible project for the exam of "Laboratorio di applicazioni mobili" course. The project can be implemented by **groups of maximum 2 people** (individual projects are obviously allowed as well). Each student/group can choose to develop the project proposed here (valid until February 2023) or suggest something else based on his/her personal interests. In the latter case, project proposals **must be individual only** and should be submitted via e-mail to Dr. Luca Sciullo (luca.sciullo@unibo.it), with a brief description of the application goals, contents and a list of **all** the features that you propose to implement. In extremely exceptional cases we can allow group projects outside the one proposed here, but they will need to be discussed with us in advance. The use of Git (or any other versioning system) is strongly encouraged. The following project description contains a minimum set of requirements for the application. Students are strongly encouraged to expand the track, by adding new features to the application, and/or further customizing the contents. Any question regarding the projects is to be asked to Dr. Luca Sciullo via e-mail (`luca.sciullo@unibo.it`). Regardless of the choice, every student/group is required to produce:

1. *One or more mobile applications*, there is no constraint on the language and the framework used: it can be native or hybrid, but it cannot be a Web Application (*i.e.* it must run natively on the device).

2. *A project report*, which is a document that describes the application produced, focusing on the workflow and the design choices. In particular, the Report should be named `SURNAME1_SURNAME2.pdf` and contain:

   - The name, surname, email and matriculation number of each component of the group.
   - Overview of the application with screenshots.
   - Implementation details on how you chose to implement the functionalities.

   A good report is probably between 10 and 15 pages. Less than 5 pages is probably bad, more than 15 is probably too much. The quality of the report WILL be part of the evaluation. **A good report also contains**

about 70% of implementation details and choices and only the remaining 30% of screenshots, overview...

3. *A presentation*, which consists in a set of slides that will help you in the project discussion. They should contain a brief recap of the report alongside with screenshots of the application. We suggest to produce around 10 - 15 slides since the discussion time is approximately 15 minutes (a group discussion may last longer). Furthermore, **each** component of the group must know the details of the entire implementation and will be possibly asked about it during the project discussion.

The **CODE** of the mobile application and the **REPORT** have to be uploaded exclusively on the Virtuale platform in the dedicated section[1] (there are 6 deadlines throughout the year). They must be enclosed in a single .zip file named `SURNAME1_SURNAME2.zip`. In case of a group, a single component is in charge of completing the upload. If the archive is too big for Virtuale, you can remove the directory "build" from your code (Android only). The SLIDES, instead, must be brought along the day of the oral examination.

Do not forget that the oral examination consists also in a theoretical part, which is **individual**. Therefore, the knowledge of the topics discussed in class (both iOS and Android) is required. The exam consists of the priject discussion (which is per group) and the oral examination (which is per student) and **must** be booked via Almaesami.

The Project: "Peer Gaming"

Overview

The project consists in the design and implementation of a distributed game where two entities -named for simplicity *client* and *server*- share information about the current state of a game and interact with each other in order for clients to play such game. We call *server* the entity in charge of setting up the game, organizing the interaction flow, and communicating the results to the participants. Clients instead are the competitors of the game, whose goal is to play the moves of the game. We note that *client* and *server* do not necessarily follow the typical architectural distinction from the architectural point of view: in this sense, a client could also act as a server accepting some requests from the server as well as the server could be in charge of gathering information from the clients. Client and server can be implemented as different applications or within the same one, with the same programming language or with different technologies. Students are free to design the interactions between the two entities, but the logic of the game must follow the flow detailed in the next sections. The game must be playable by at least two different clients, while there is no limitation on the number of clients that can join the game (as long as it works for two).

[1] `https://virtuale.unibo.it/course/view.php?id=28374`

Required Gaming Flow

1. Discovery Phase: during the *discovery phase*, the server advertises the beginning of a new match for the game, sharing also the rules and all the information required for joining the game. This means that clients can reach the server in order to get the information required and to register/subscribe to the current match. Server can close the registration phase depending on different criteria, for instance after a predefined timeout exceeded or because a maximum number of clients joined the match.

2. Playing Phase: in the *playing phase* the server communicates the current state of the game, indicating which player should make the move. If the game does not involve turns it just broadcasts a status. After that, it collects the answer of the players, updating the game state and notifying back the players. This loop is repeated as many times as needed by the game to be completed or until a winner is identified by the server.

3. Conclusion Phase: the *conclusion phase* starts as soon as the server identifies a possible winner of the game or the game finishes because a timeout exceeded. The server then communicates the results of the game to all the competitors. The results contain also a dashboard containing the previous results and it is updated with the last scores.

Examples

We here identify three different kind of games that can be implemented, each of them with a diverse levels/degree of difficulty. These are only suggestions, all the combinations of features and improvements are possible and will be evaluated consequently.

Basic: Bingo This game is well-known and, for whoever does not know the basics you can refer to `https://en.wikipedia.org/wiki/Bingo` or (for the Italian reference) `https://it.wikipedia.org/wiki/Tombola`. If we translate this game following the "Peer Gaming" dynamics, we can set up the interactions as follows: first the servers advertises the beginning of a new match, then clients subscribe to the match and, at some point, the server closes the subscriptions. Before starting the game the server distributes the "bingo cards" (one or more) to all clients. Then the server starts the game, which consists in multiple repetitions of the same phase. In each phase the server extracts randomly a number (from 1 to 90) and communicates it to all clients. Clients need to check their bingo cards and mark all the boxes containing the number (this can also be done automatically) and then they should notify back the server about any points that they have made (for instance, one of "terno", "cinquina", "tombola" or "nothing"). If one player has marked all the boxes in one of his/her cards, then the server declares that the game is over and that player is the winner. This is an example of a game that requires a basic interaction, where all players

play at the same time and the server only repeats the same phase over and over until a certain condition is met.

Logical Evolution: Uno A possible evolution that complicates the interaction pattern may be introduced by taking as an example the game "Uno" (`https://en.wikipedia.org/wiki/Uno_(card_game)`). In this game the server is the master of the deck and the clients are the players; they need to play in turns and each of them has the goal of playing all the cards in his/her hand. Basically, during his/her turn, a player can play one of the cards in his/her hand that matches one of the features of the card that was played previously (color or number) or draw a card if no card can be played. In this case the server, after gathering all the subscriptions from the clients, needs to shuffle the deck and distribute the initial set of cards to all clients. Then the server decides the order of the players and asks the first player to play a card, then broadcasts the card to all other players (so they know the status of the game) and then asks the second player to play, and so on so forth... This game is logically more complicated than bingo in that it introduces the concept of turns and some cards (as you know) may also change the turn dynamics.

Technological Evolution: FindSpot This game is very similar to bingo in terms of interactions but it requires the technological effort to make use of components that are specific of the mobile phone (e.g. the camera, the sensors, ...). The game starts in the usual way, the server advertises the beginning of a new game and clients subscribe to participate. In each game phase the server provides the users with a description of an object that clients need to photograph and gives a deadline for clients to provide the required picture. For instance, the server may ask clients to take a picture of "Fontana del Nettuno", then clients will need to rush to the place and take a picture of the subject. The picture is sent back to the server alongside with the GPS position as a further proof of authenticity. This game can be adapted to fit several cool use cases (e.g. a "Treasure Hunt" or a game of riddles) and presents advanced technical challenges compared to the simpler Bingo.

Best Practices

As previously explained, there is no clear distinction between clients and servers from the architectural point of view, since both these entities can be queried (server behaviour) and can ask for data (client behaviour) and for this reason, students should design the application taking into consideration this fundamental aspect, which can be implemented in several different ways. A first basic approach is based on the REST architectural pattern, and more in detail it consists in clients that continuously poll the state of the server in order to gather the information needed for the current state of the game. This holds for all the phases in which the client needs to read something from the server. On the contrary, in order to update the server, a simple POST can be performed to write on the server. A more advanced approach would instead include technologies

designed on purpose for bi-directional communications, for instance Web Sockets (`https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API`). Through them, both clients and servers are able to read and write information using the same channel. Another approach could be using a minimal backend like Firebase (and the real-time notification system) as a bi-directional communication channel: information to be shared is firstly written on to a minimal database, then notified back to all the observers and finally read in the database by the other entities. Firebase is a real-time database, really simple to interact with and making up the basis for e.g. instant messaging.