Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Computer Science and Engineering

POLITECNICO
MILANO 1863

Tinfinity

Design and Implementation of Mobile Applications

Professor : Luciano Baresi

Fumagalli Alberto	Matr. 818097
Mariani Sebastiano	Matr. 817781
Mastellone Riccardo	Matr. 852341

Contents

List of Figures

1 Introduction

This report will describe the *Tinfinity* application, a proximity chat system for iOS platforms. The first part will introduce the application with a high level description and then the report will focus on the technical aspects.

2 High level application description

2.1 Description

Tinfnity is a mobile application that implements a proximity chat system aiming to connect people in the same area. In order to access this application, the user must connect his Facebook account and share some of his informations like photos and date of birth. This step, simplifies the login process and mitigates the fake users problem delegating the issue to Facebook. Once the user has been successfully logged in the application, he can discover all the people that are using Tinfinity near to him using the map as shown in Figure 1. If the user finds someone interesting on the map, he can tap on the marker and send a friendship request as shown in Figure 2.

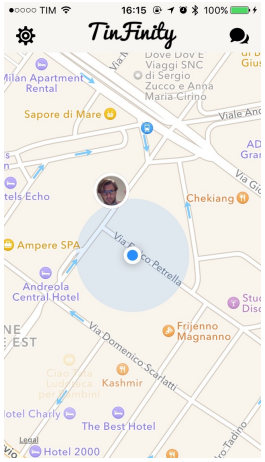

Figure 1: Users Map

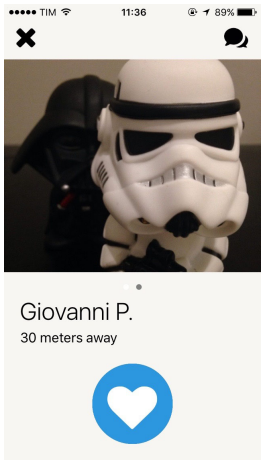

Figure 2: Friendship request

When and if recipient accepts the request, as shown in Figure 3, they can start chatting even when they are no longer close to each other, such a normal chat system as Whatsapp for example (shown in Figure 4), and maybe know each other directly in person. Additionally the full name and the age of the other user are disclosed.

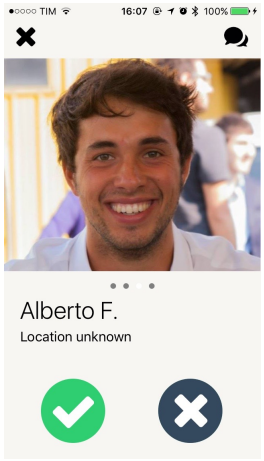

Figure 3: Friendship acceptance

Figure 4: Chat

The user can also see the informations relative to his profile (Figure 5) and personalise which photos have to be shared with other through the settings page as shown in Figure 6.

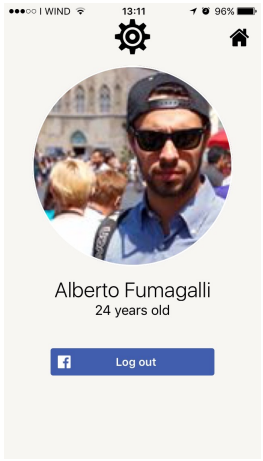

Figure 5: Profile

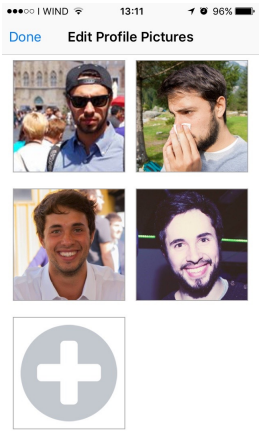

Figure 6: Profile

The application works in background and there is no need to open it in order to be localised by the others on the map.

3 Requirements

In this section the functional and non functional requirements which have been identified in the analysis phase are reported.

3.1 Functional Requirements

- Display to the user a map containing the position of other user within 800m from its position;
- Provide a chat system and allow the user to interact with it;
- Display the other users informations when request;
- Enable the user to personalise the photos that have to be shared;
- Enable the user to send a friendship requests and accept / decline the received one;
- Create a web application in order to manage the chat system and the location system;
- Create a push notification system in order to inform the user when a message is received;
- Develop an iOS mobile version of the application;

3.2 Non Functional Requirements

The non functional requirements are related to the devices which are target for the application:

- The application should run on devices with an iOS version $>$ 8.4;

4 Architecture

The architectures of the application is composed by two main parts:

- An client iOS application
- A server web application

The web application is necessary because it has to keep track of the users data, location and handle the chat system. A general overview of the communication between the mobile application and the web application is shown in Figure 7.

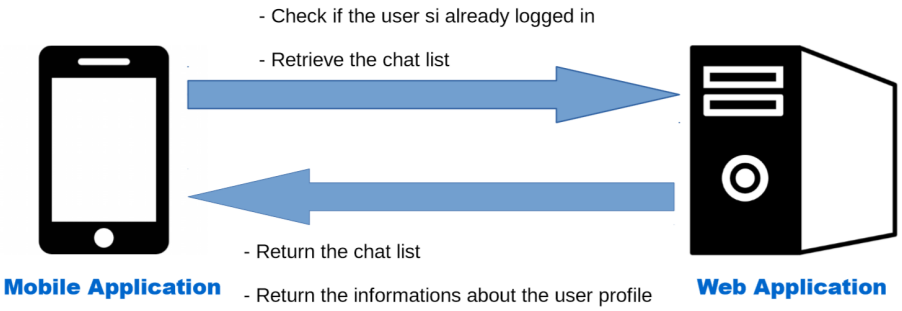

Figure 7: System architecture

5 Mobile application

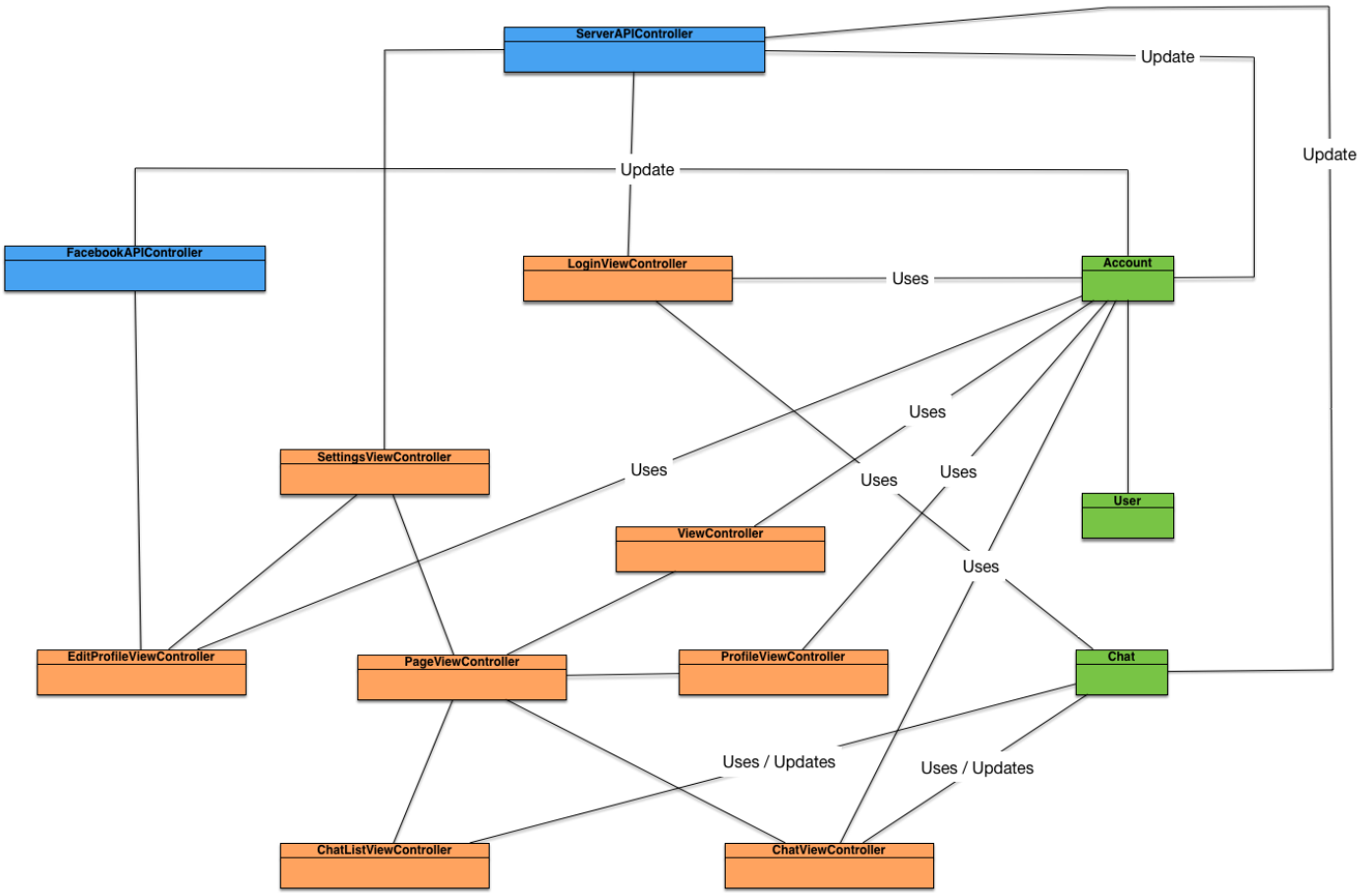

Figure 8: Simplified class diagram

The above class diagram shows a simplified version (only the most important classes are shown) of the application. The mobile application is based on eight main ViewController which handle the functionalities provided by Tinfinity:

- **LoginViewController** : Provides the login with Facebook functionalities and updates the account informations
- **ViewController** : Provides to the user a map that shows where the other user are located and their profiles.
- **SettingViewController** : Provides a recap view of the informations about the user and the ability to change them.
- **EditProfileViewController** : Provides the functionalities to decide which photos have to be displayed.

- **ProfileViewController** : Provides a view that shows the profile of other users which have sent a friendship request, and the abilities to accept / decline it.
- **ChatListViewController** : Provides a view with a list of the chat and the provides the functionalities to delete them.
- **ChatViewController** : Provides a view with the conversation with the specified user and the ability to send messages.
- **PageViewController** : Manages all the other views. It doesn't provide any functionality to the user, but it is used only for internal purposes.

5.1 LoginViewController

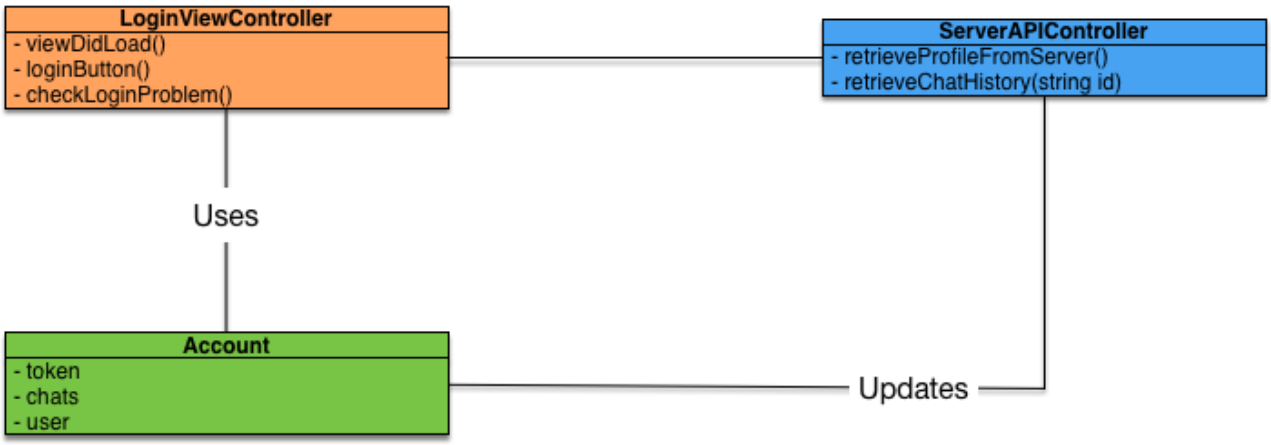

Figure 9: LoginViewController class diagram

The LoginViewController manages the view that implement the login functionalities with Facebook and set up the initial settings relative to the account.

- **viewDidLoad()** : When the view is loaded this callback tries to check if there is already a valid Facebook OAuth token binded to the account. If the token is present and valid then the application will display the main view of the application.
- **loginButton()** : This function is triggered when the login button is pressed. This function exploit the Facebook SDK in order to retrieve a valid OAuth token. If this operation succeed, then the other account settings, like the chat history, are retrieved and set and the main view of the application is displayed.

5.2 ViewController

Figure 10: ViewController class diagram

The ViewController manages the main view of the application and it show the map with the user location and the location of the other users around him. It is the delegate for the location manager and the mapView callback functions.

- **viewDidLoad()** : When the view is loaded this callback checks if the permissions needed in order to locate the user properly are given. if these are not given the map is hidden, otherwise the map is displayed and the user location is retrieved.
- **viewWillAppear()** : Every time the view is shown the location of the user is updated.
- **locationManager(didUpdateLocations error)** : Set the latitude and longitude of the user and add a marker on the map.
- **annotationClicked(UserAnnotation annotation)** : When a marker of another is clicked the information relative to him are retrieved and passed to the ProfileViewController

5.3 SettingsViewController

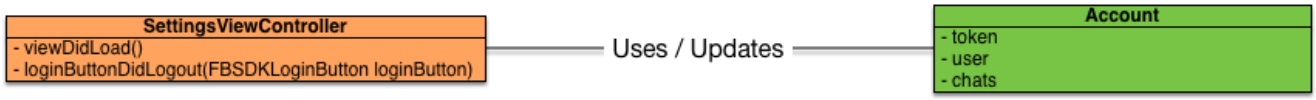

Figure 11: SettingsViewController class diagram

The SettingsViewController shows a view with the recap of the user informations and implement the logout functionality.

- **viewDidLoad()** : When the view is loaded all the informations about the user, like the profile picture, are retrieved and displayed properly.
- **loginButtonDidLogout()** : Exploit the Facebook SDK in order to invalidate the OAuth token and log out the user from the application.

5.4 EditProfileViewController

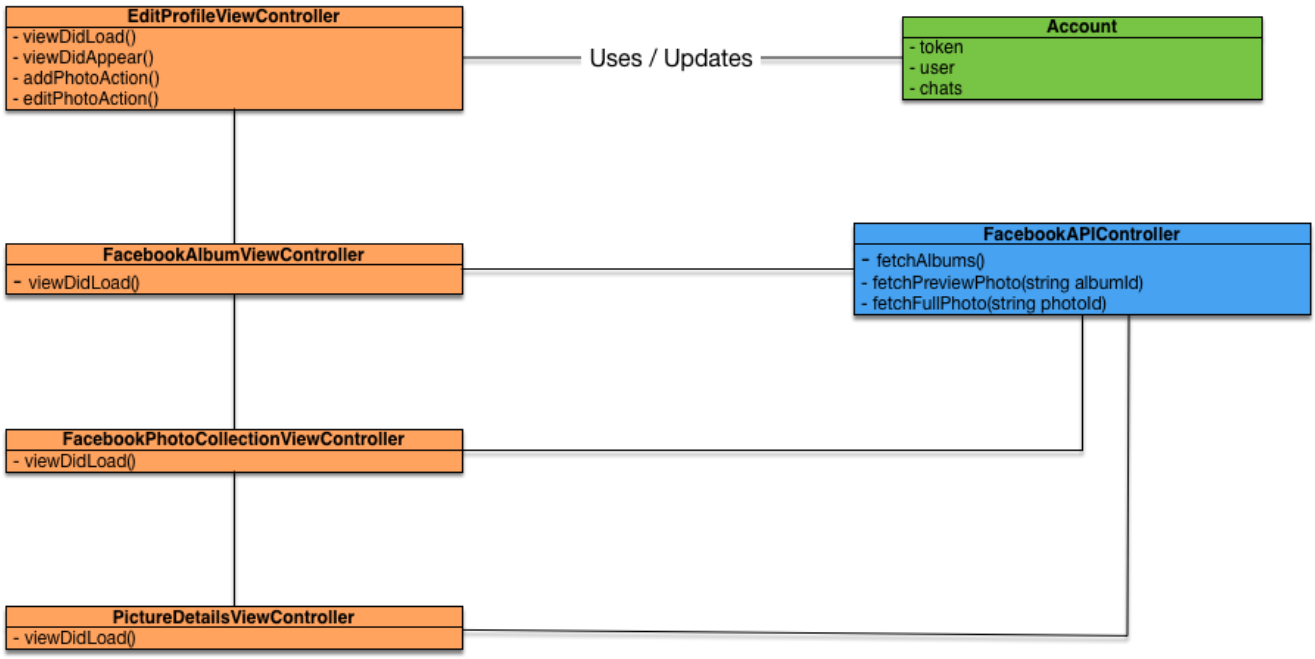

Figure 12: EditProfileViewController class diagram

The EditProfileViewController and the other auxiliary ViewControllers (FacebookAlbumViewController, FacebookPhotoCollectionViewController and PictureDetailsViewController) provides the functionalities of the photo editing. The user here can decide which photos of his Facebook profile have to be displayed in the application.

- **viewDidLoad() (FacebookAlbumViewController)** : Fetches the Facebook photos album relative to the user.
- **viewDidLoad() (FacebookPhotoCollectionViewController)** : Fetches the preview of the photos that are in the selected album.
- **viewDidLoad() (PictureDetailsViewController)** : Fetches the high resolution version of the selected photo.

5.5 ProfileViewController

Figure 13: ProfileViewController class diagram

The ProfileViewController manages the view displayed when the user click on a marker on the map or when he receive a friendship request. This view shows all the details relative to the other user like the distance, his photos and his name and age. It provides also the functionalities in order to accept or decline a friendship request, send a new request, or start a new chat with the selected user.

- **buttons()** : Based on the relationship with the other user, this function decides which buttons have to be displayed.
- **distance()** : This functions calculate the distance between the user and the other selected user.
- **createPageViewController()** : Creates the photos slideshows with the photos of the other user.
- **chatClick()** : Start a new chat with the other user.
- **sendRequestClick()** : Send a friendship request to the other user.
- **acceptClick()** / **declineClick()** : accept / decline a received friendship request.

5.6 ChatListViewController

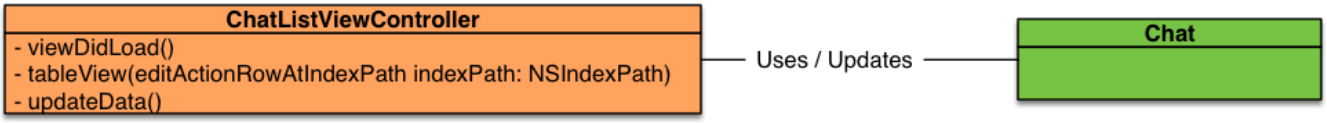

Figure 14: ChatListViewController class diagram

The ChatListViewController implements the application logic that takes care of these two thing:

1. Shows all the chat that the user has begun
2. Shows all the pending friendship request

It also provides different functionalities based on the relationship between the two users. If they are friend it is possible to revoke the friendship status or delete the chat, if they are not it is possible to accept or decline the request or delete the chat.

- **viewDidLoad()** : Configure the table view that will handle the chat list and the binded actions
- **tableView(editActionsForRowAtIndexPath indexPath: NSIndexPath)** : Set the proper action for each row:
  - **accept** : set the status between the two user as "Friend" and sycronize it with the server
  - **decline** : send a request to the server in order to delete the friendship request
  - **request** : send a friendship request to te selected user
  - **unfriend** : delete the relationship between the users and synchronise it with the server
- **updateData()** : refresh the information in the table view. this function is trigger when the user pull down the view

5.7 ChatViewController

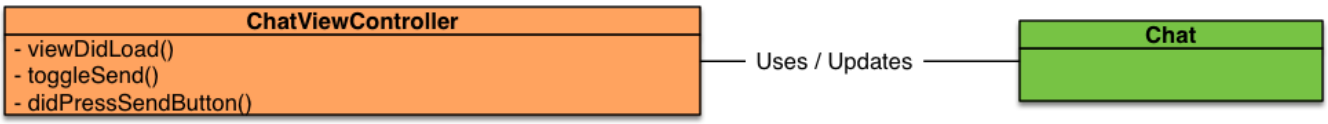

Figure 15: ChatViewController class diagram

The ChatViewController manages and display correctly the chat history between two users. Every message is displayed as a bubble, a blue one for the current user and a grey one for the other. It provides also the functionalities in order to send messages and updates the history when a new one is received.

- **viewDidLoad()** : Enables the chat only when the connection to the server is established
- **toggleSend()** : Disable the send button if the connection to the server is interrupted. When the connection has been re-established, it re-enable the send button again. It disable the chat also if two users are not friend and the distance of the other is out of range (the distance between the two people are greater than 800m)
- **didPressSendButton()** : Prepare the correct data structure for the message and send it. it also update the chat list appending the new message at the end of the list.

5.8 Persistent Data Design

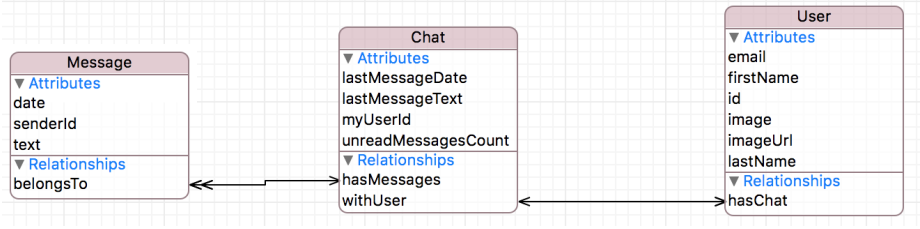

Figure 16: Database

The persistent Core Data is composed by 3 classes:

- **Message** : contains the informations of a single message(the date in which the message has been sent,the sender and the actual message text) and has a one-to-one relationship that associate it to a certain chat;
- **Chat** : contains the information of a chat(the date and text of the last message that has been received, the id of the logged user and the counter of unread messages) and has two relationships, one is a one-to-many relationship with the message class, that associate the chat with all of his messages, and the other a one-to-one relationship with the User class, that represent the other user.
- **User** : contains all the information of a user(email, name, surname, id, the user image and its url) and has a one-to-one relationship with the chat that is associated with it.

6 User Experience

6.1 Sender User Experience

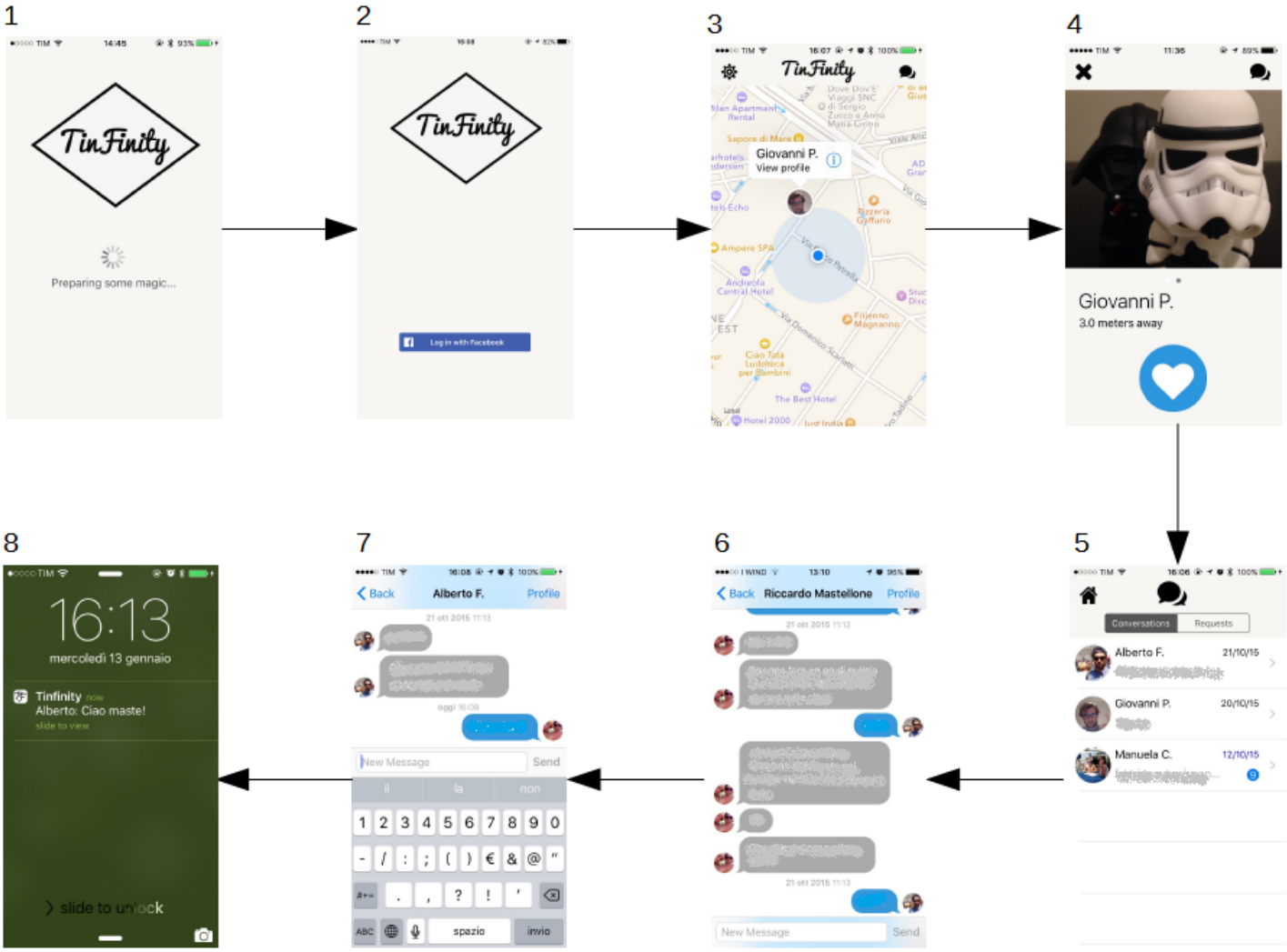

Figure 17: Sender User Experience

1. The application launcher contacts the server in order to retrieve to check if there isa already a valid login token present
2. From the login view the user can inserts his Facebook username and password and log in to the application
3. From the main view the user can see the people around him and choose to look at an interesting profile

4. The user can send a friendship request

5–8 When and if the other user accept the friendship request, they can start chatting as a normal chat application.

6.2 Receiver User Experience

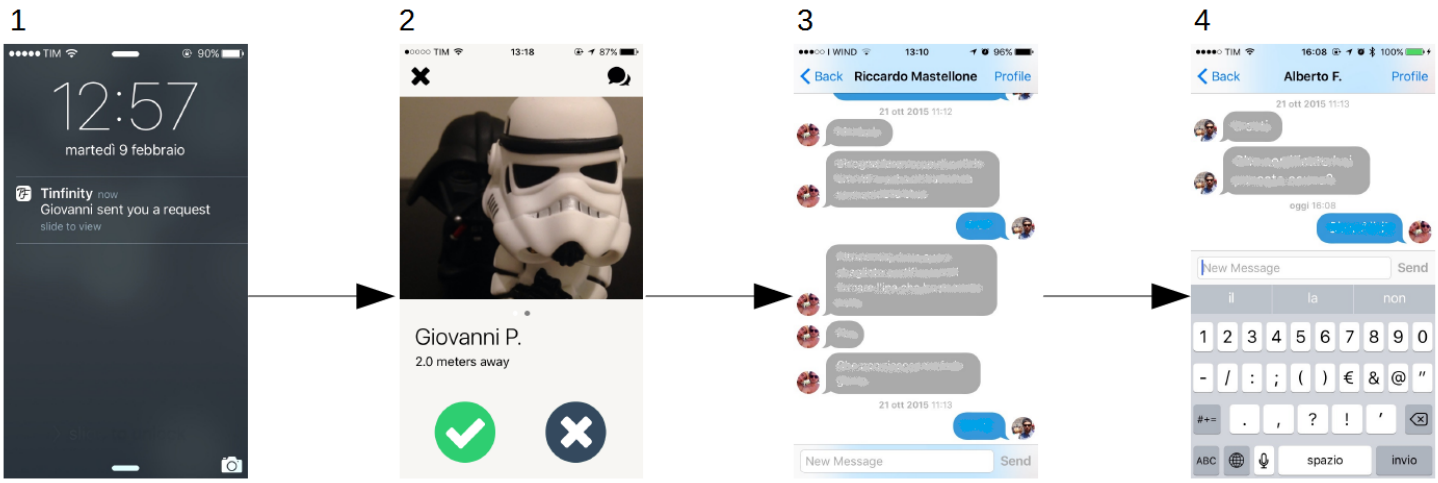

Figure 18: Receiver User Experience

1. The user receive a notification about a received friendship request

2. The user can choose if the request has to be accepted or declined

3–4 If the friendship is granted they can start chatting as a normal chat application.

6.3 Photo selection User Experience

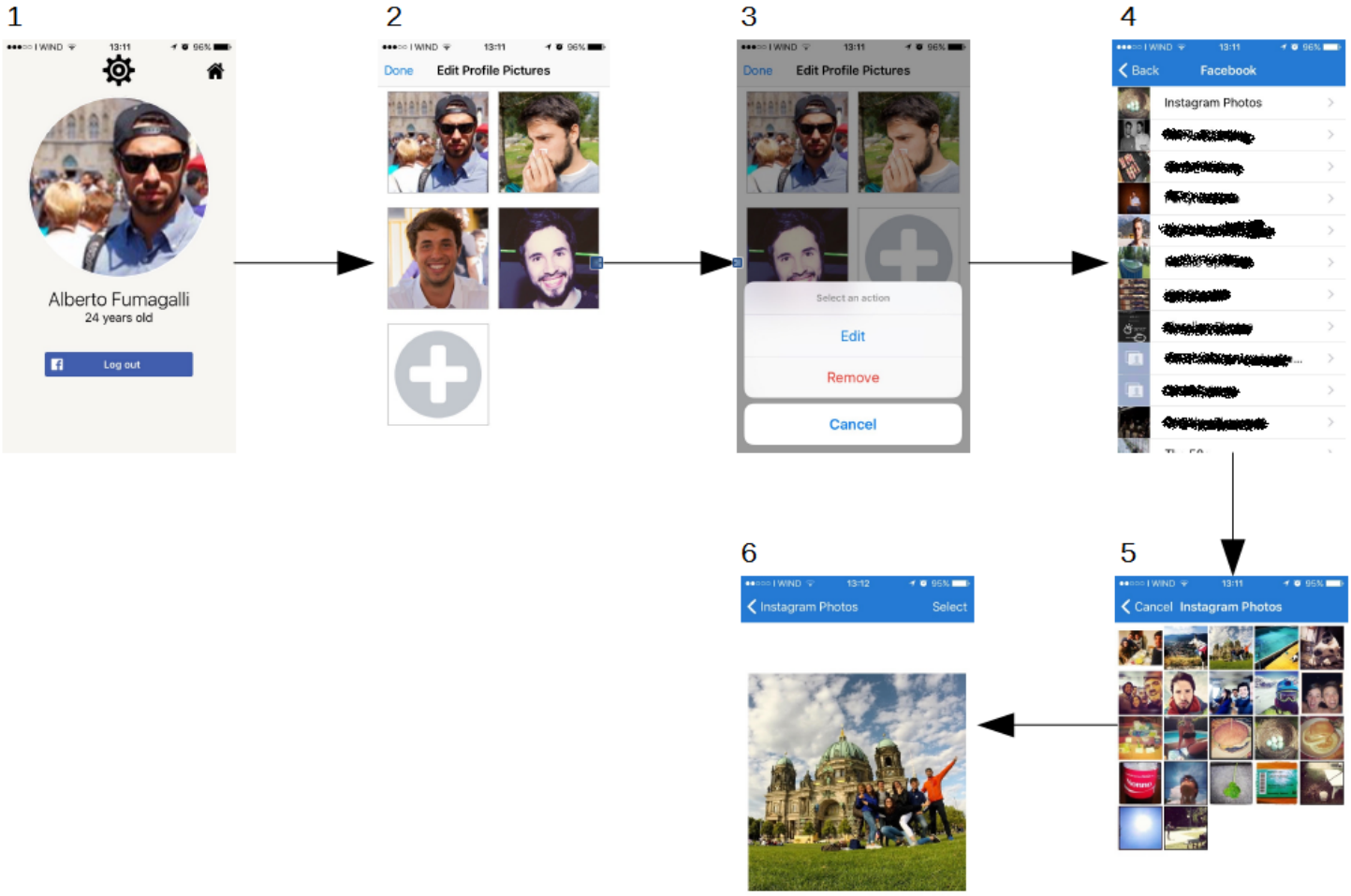

Figure 19: Photo selection User Experience

1. From the settings view the user can tap on his photo profile in order to modify it or add other photos which will be visible by other users.
2. The user can tap on the '+' button and chose whether to add or change a current photo
3. All the photo albums of the Facebook account are displayed
4. All the photos of the selected album are displayed
5. Finally tap on the select button the displayed photo is added to the user profile

7 Web app

This web server has been developed to support the iOS Tinfinity application.

It is written using Node.js with the Express framework and uses MongoDB as a database. These have been preferred over many alternatives for performance reasons, as they support a very high number of concurrent connections with very little resources. Additionally, websockets are natively supported using SocketIO, allowing a real-time, bidirectional and reliable chat system.

Figure 20: Web app overview

The structure of the web server has been developed so to not be tightly coupled with the client architecture (iOS), hence other clients (e.g., Web, Android) can be easily created starting from the APIs documentation.

All the security checks are done server-side, not disclosing any unwanted information unless explicitly requested by the user.

MongoDB offers natively Geospatial data type support, and near users are retrieved directly querying the database with the user current position.