

The ProxiConn Project

*A Localizing Proximate Users Project using GPS
for Mobile Devices*



Telecommunications Systems Institute
Supervisor: Papaefstathiou Yannis
Author: Kalodimas Panagiotis

CONTENTS

| | |
|--|----|
| CONTENTS | 2 |
| FIGURES | 4 |
| TABLES | 6 |
| INTRODUCTION | 7 |
| ABSTRACT | 8 |
| PROXICONN PROJECT ARCHITECTURE | 9 |
| 1.1. The Architecture | 9 |
| 1.2. Geographic Coordinates System | 11 |
| 1.3. Server – Client Communication Protocol | 12 |
| 1.3.1. Ping | 13 |
| 1.3.2. Login | 14 |
| 1.3.3. Logout | 15 |
| 1.3.4. Location Update | 15 |
| 1.3.5. Neighbors | 16 |
| 1.3.6. User Info | 17 |
| 1.3.7. Signup | 18 |
| 1.3.8. Sign out | 19 |
| 1.3.9. User Edit | 20 |
| 1.4. ProxiConn Communication Protocol Errors | 21 |
| 1.5. Protocol Data Format | 22 |
| DATABASE SERVER | 24 |
| 2.1. Users Data Table | 24 |
| 2.2. Online Users Data Table | 25 |
| 2.3. Procedures and Function | 26 |
| WEB SERVER APPLICATION | 28 |
| 3.1. Client Services | 28 |
| 3.2. Clients Requests | 30 |
| 3.2.1. Ping Request | 31 |
| 3.2.2. Login Request | 32 |
| 3.2.3. Logout Request | 33 |
| 3.2.4. Location Update Request | 34 |
| 3.2.5. Neighbors Request | 35 |

| | | |
|--|-----------------------------------|----|
| 3.2.6. | User Info Request | 36 |
| 3.2.7. | Sign up Request | 36 |
| 3.2.8. | Sign out Request..... | 38 |
| 3.2.9. | User Edit Request | 39 |
| USER-CLIENT APPLICATION | | 40 |
| 4.1. | iPhone 4 Mobile Device..... | 41 |
| 4.2. | iOS Operating System | 42 |
| 4.3. | User-Client iOS Application | 43 |
| 4.3.1. | App Manager Module..... | 44 |
| 4.3.2. | GPS Manager Module..... | 46 |
| 4.3.3. | Server Client Module | 46 |
| 4.3.4. | User GUI Module | 48 |
| IOS APPLICATION USER GRAPHICS INTEFACE | | 49 |
| 5.1. | Map View..... | 49 |
| 5.2. | Neighbors View | 52 |
| 5.3. | User Info View | 53 |
| 5.4. | Settings View | 54 |
| 5.4.1. | Application Status..... | 54 |
| 5.4.2. | User Settings..... | 55 |
| 5.4.3. | Server Settings..... | 56 |
| 5.4.4. | GPS Settings..... | 57 |
| 5.4.5. | Restore Settings..... | 58 |
| 5.5. | Station View..... | 59 |
| 5.6. | Account View | 59 |
| WEB CLIENT APPLICATION | | 61 |
| 6.1. | Web Client Application..... | 61 |
| 6.1.1. | Login View | 62 |
| 6.2. | Main View..... | 63 |
| 6.2.1. | Main View – Neighbors | 64 |
| 6.2.2. | Main View – Settings | 65 |
| 6.2.3. | Main View – User Profile | 65 |
| CONCLUSIONS..... | | 67 |

FIGURES

| | |
|---|----|
| ProxiConn Project Architecture | 10 |
| Latitude(Φ) and Longitude(λ) Angles..... | 11 |
| Latitude and Longitude | 12 |
| mySQL Logo..... | 24 |
| Ubuntu Server, Apache Server & PHP Logos | 28 |
| Web Application Services Architecture | 29 |
| Web Application – Clients Server | 30 |
| Web Application – General Flow Diagram..... | 31 |
| Web Application – Ping Request Flow Diagram | 32 |
| Web Application – Login Request Flow Diagram..... | 33 |
| Web Application – Logout Request Flow Diagram | 34 |
| Web Application – Location Update Request Flow Diagram..... | 35 |
| Web Application –Neighbors Request Flow Diagram..... | 35 |
| Web Application –User Info Request Flow Diagram | 36 |
| Web Application –Sign up Request Flow Diagram | 37 |
| Web Application –Sign out Request Flow Diagram | 38 |
| Web Application –User Edit Request Flow Diagram..... | 39 |
| proxiConn iOS user-client application on the Apple AppStore..... | 40 |
| iPhone 4 Specifications Summary | 42 |
| iOS Applications Layer..... | 42 |
| iOS Layers..... | 43 |
| User-Client Application Architecture | 44 |
| User-Client Application - App Manager Module | 45 |
| User-Client Application – Server Client Module Flow Diagram..... | 47 |
| iOS Application GUI – Map View (Satellite Type) | 49 |
| iOS Application GUI – Map View Connected & Disconnected..... | 50 |
| iOS Application GUI – Map Settings View..... | 51 |
| iOS Application GUI – Map View Landscape (Hybrid Type)..... | 52 |
| iOS Application GUI – Neighbors View – Orientation Definition | 52 |

| | |
|---|----|
| iOS Application GUI – Neighbors View | 53 |
| iOS Application GUI – User Info View | 53 |
| iOS Application GUI – Settings View – Application Status | 54 |
| iOS Application GUI – Settings View – User Settings (Logged & Unlogged) | 55 |
| iOS Application GUI – Settings View – Server Settings | 57 |
| iOS Application GUI – Settings View – GPS & Restore Settings | 58 |
| iOS Application GUI – Station View | 59 |
| iOS Application GUI – Account View (Edit & Signup) | 60 |
| Google Maps API Logo | 61 |
| Web-Client Application – Web-Client Application Flow Diagram | 62 |
| Web-Client Application – Signup View | 62 |
| Web-Client Application – Login View | 63 |
| Web-Client Application – Main View | 64 |
| Web-Client Application – Neighbors & Info Blocks | 65 |
| Web-Client Application – User Profile Block | 66 |

TABLES

| | |
|--|----|
| Technical University of Crete Geographical Coordinates Example | 12 |
| ProxiConn Project Protocol – Server-Client Requests | 13 |
| ProxiConn Communication Protocol – Ping Request..... | 13 |
| ProxiConn Communication Protocol – Ping Request Example | 14 |
| ProxiConn Communication Protocol – Login Request..... | 14 |
| ProxiConn Communication Protocol – Login Request Example | 15 |
| ProxiConn Communication Protocol – Logout Request | 15 |
| ProxiConn Communication Protocol – Logout Request Example..... | 15 |
| ProxiConn Communication Protocol – Location Request..... | 16 |
| ProxiConn Communication Protocol – Location Request Example | 16 |
| ProxiConn Communication Protocol – Neighbors Request..... | 17 |
| ProxiConn Communication Protocol – Neighbors Request Example | 17 |
| ProxiConn Communication Protocol – User Info Request..... | 18 |
| ProxiConn Communication Protocol – User Info Request Example | 18 |
| ProxiConn Communication Protocol – Signup Request..... | 19 |
| ProxiConn Communication Protocol – Signup Request Example | 19 |
| ProxiConn Communication Protocol – Sign out Request | 19 |
| ProxiConn Communication Protocol – Sign out Request Example..... | 20 |
| ProxiConn Communication Protocol – User Edit Request..... | 20 |
| ProxiConn Communication Protocol –User Edit Request Example | 21 |
| ProxiConn Communication Protocol Errors..... | 22 |
| JSON format Requests Data Example | 23 |
| Database Server – Users Data Table Structure..... | 25 |
| Database Server – Online Users Data Table Structure | 25 |

INTRODUCTION

This report is related to «proxiConn» project behalf of Telecommunication Systems Institute. The intention of this project was the implementation of a «proxi» server. A «proxi» server handles the geographical position of its clients and informs them which are theirs proximate ones. This is an outdoors project by the meaning of the technology means required. Mobile devices are the most familiar and widely used technology devices meeting the requirements of this project. The GPS, WWAN and WAN technologies are all supported by the synchronous «smart» mobile devices. The main constraint for this project is the instability and the low bandwidth capabilities of the corresponding technologies when used outdoors and on the move.

ABSTRACT

The «proxiConn» project is a project that aims on creating a positioning map of registered users and inform every one of them about his proximate ones. It demands the existence of a central positioning server that records the positioning changes of each user and computes the distance relations between them in order to send reliable information. On the user-client side the project demands from the users to have good knowledge of theirs geographical position and update it frequently in order to help the server's calculations accuracy.

On Chapter 1.1, the «proxiConn» project architecture is described in details. Some auxiliary information are also given as long as the communication protocol, the international geographical coordinates system and other application auxiliary means used intending a better understanding of the chapters coming afterwards where extendible details about the project implementation are apposed. In chapters 1 to 5 the implementation of the database server, the web application and the iOS application for the mobile devices are described particularly. At last an auxiliary web-client application designed for stationary and observator users is described in chapter 6.

CHAPTER 1

PROXICONN PROJECT ARCHITECTURE

As the «proxiConn» project is based on a central server, in the middle of the project architecture is the database. For this part of the project an Ubuntu linux web server is used with a mySQL server instance database installed. A web application was also developed in order to manage the incoming and outgoing data from the database to the clients and the opposite. It is obvious that at least for security reasons is not allowed clients to have direct access to the database.

On the other hand a «smart» mobile device is used by the client side. Synchronous mobile devices meet all the requirements for this purpose. The GPS, WAN and WWAN technologies are all supported by even the lowest cost mobile devices today. At the «proxiConn» project the representative of the «smart» devices family was an Apple Inc «iphone 4» model using the iOS 6.1.3 version firmware. As on the server's side, thus on the client's side, corresponding software was developed.

For extending the «proxiConn» project user-clients usability, an extra web-client application was also designed as an auxiliary application for stationary and observatory users. This application makes the «proxiConn» project usable for every user can access the internet using any of the free web browsers available in the market (Google Chrome, Mozilla Firefox, etc).

1.1. The Architecture

The «proxiConn» project architecture consists of a variety of modules as shown in Figure 1. In this chapter a short description is apposed for each, in addition to the following chapters where more details can be found.

- A **Web Server** is used in order to serve the user-clients requests for data. This Web Server is described on CHAPTER 3.
- A **Database Server** is used for managing the «proxiConn» project data, described on CHAPTER 2

- A **Mobile device** is the module that the users need for accessing the internet and the GPS service. It's the module running the «proxiConn» project software for mobile devices. This software is described in CHAPTER 4 and CHAPTER 5
- **GPS** (Global Positioning System) is also a part of the «proxiConn» project as it is the only accurate service that can specify with great accuracy the location of a user
- **WLAN** (Wireless Local Area Network) is one of the ways of getting internet access. Its drawback is that the user cannot carry it with him. Vehicular users, especially away of urban territories, face that problem.
- **WWAN** (Wireless Wide Area Network) is the other way of getting internet access. It is not the most preferable way, cause of its instability and low bandwidth, but its wide range of coverage, especially outdoors, sometimes makes it the only choice exists. Together with the WLAN services can handle the internet access needs of the users. It is worth to be mentioned that the mobile network services are improving themselves day by day and sooner or later they will be a very reliable choice.
- **TCP/IP** is used for creating connections between the users and the main server.

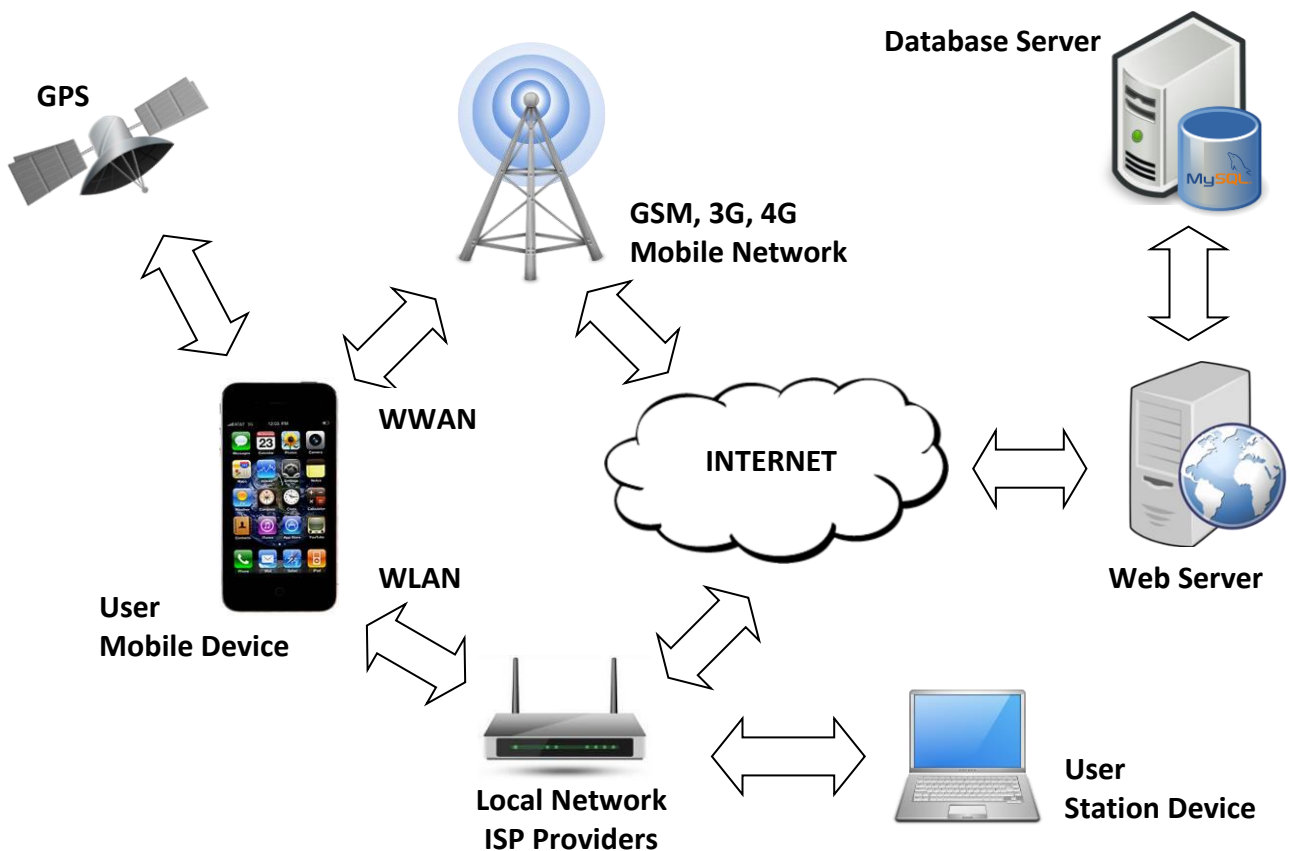


Figure 1: ProxiConn Project Architecture

1.2. Geographic Coordinates System

The Global positioning system is based on satellites and provides location information. On the «proxiConn» project a mobile device can be used as the hardware equipment connected to the GPS services for providing the user his location's information accurately. What is described on this subchapter is the geographical coordinate system that the GPS system uses on defining users location as this is mentioned lots of times in the lines of this reference.

The Earth has a spherical shape and it is not possible to specify a point on its surface using the Cartesians coordinates system. For doing that the geographical coordinates system uses the latitude and longitude numbers. If a line starts from a point on the surface of the Earth and passes through the center of the Earth, the angle is created within it and the equatorial plane of the Earth is called latitude and the one is created within the meridian plane is called longitude. This is a simplified explanation of the geographical coordinates system. Latitude and longitude is counted on either degrees or decimal numbers. For example the Technical University of Crete where the «proxiConn» project was developed has the coordinates shown in Table 1.

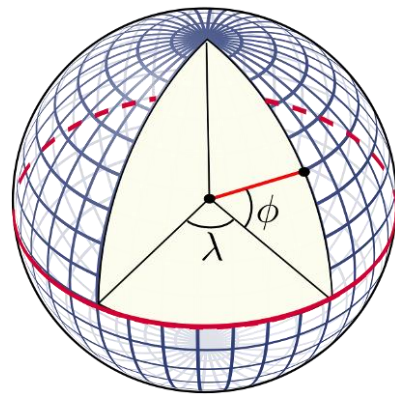


Figure 2: Latitude(Φ) and Longitude(λ) Angles

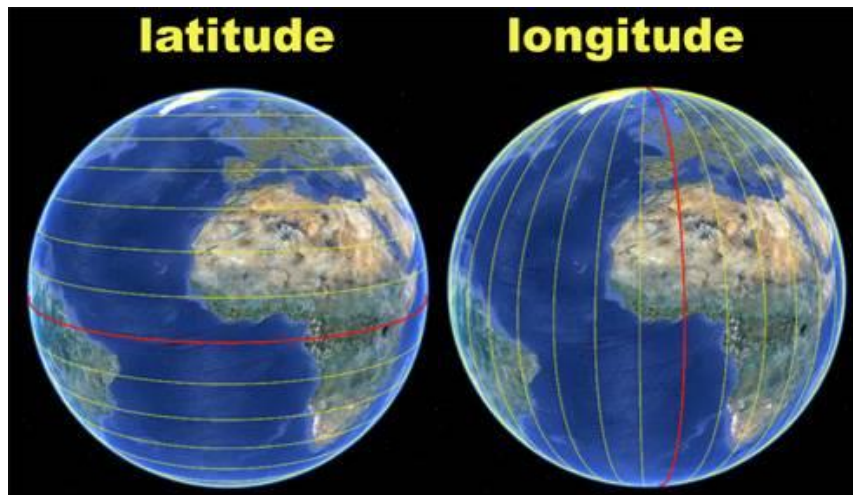


Figure 3: Latitude and Longitude

As it is understandable the Pythagorean mathematic type for calculating the distance between two points cannot be used for calculating the distance between two points on the Earth's surface. The mathematic type for measuring distances between two geographical points is,

$$D_{lon} = lon_1 - lon_2$$

$$D_{lat} = lat_1 - lat_2$$

$$a = \sin^2\left(\frac{D_{lat}}{2}\right) + \sin^2\left(\frac{D_{lon}}{2}\right) \cdot \cos(lat_2) \cdot \cos(lat_1)$$

$$c = 2 \cdot a \tan 2\left(\sqrt{a}, \sqrt{a-1}\right)$$

$$d = R \cdot c, \text{ where the } R \text{ parameter is the radius of the Earth}$$

| Technical University of Crete | | |
|-------------------------------|------------------------------------|-----------|
| Coordinates | Degrees | Decimal |
| Latitude | 35 Degrees, 31 Minutes, 53 Seconds | 35.531389 |
| Longitude | 24 Degrees, 4 Minutes, 5 Seconds | 24.068056 |

Table 1: Technical University of Crete Geographical Coordinates Example

1.3. Server – Client Communication Protocol

The connection between the client and the server is achieved using the TCP/IP protocol. The client side sends HTTP Post requests to the web application running on server and receives the responses. In this chapter the data transfer protocol used

between the server and the client is analyzed. The client passes all its data through the post request body to the server. The server data are passed to the client through the request response.

There are nine types of requests that the client can send to the server as shown in Table 2.

| Request Type | Code | Description |
|------------------------|------|---|
| Ping | 0 | The client sends this request to check the connectivity with the server |
| Login | 1 | The client requests to login to the server |
| Logout | 2 | The client requests to logout from the server |
| Update Location | 3 | The client requests to update his location coordinates |
| Neighbors | 4 | The client requests his proximate clients information |
| User Info | 5 | The client requests approximate user's public information |
| Sign up | 6 | The client requests the creation of a new account |
| Sign out | 7 | The client requests the server to delete his account |
| Edit User | 8 | The client requests to edit his user's profile |

Table 2: ProxiConn Project Protocol – Server-Client Requests

1.3.1. Ping

The «Ping» request is used when the user has not set his login information and the client application cannot send any other request to confirm the server is reachable. If the user has set his information for login the client application uses those data trying to login at the same time. At a «Ping» request the client sends a small amount of data required by the «Ping» request and receives also a very small amount of data as confirmation response.

| Ping | Data | Description |
|-----------------|------------------------------------|-----------------------------|
| Request | Command="Ping Request Code" | |
| Response | Error="error code" | The server is reachable |
| Response | Unknown Data or Connection Timeout | The server is not reachable |

Table 3: ProxiConn Communication Protocol – Ping Request

| Ping | Example Data | Description |
|-----------------|--------------|---------------------------------------|
| Request | Command=0 | |
| Response | Error=0 | The server is reachable and responded |

Table 4: ProxiConn Communication Protocol – Ping Request Example

1.3.2. Login

The «Login» request is used in order the user of the client application to login the «proxiConn» project and use its services. The login process needs a valid username, password and the location coordinates of the user. After a successful login request the server responds sending back to the clients its profile information and a unique session identity. This session identity is used in the rest of the session between the client and the server in order to identify the client him.

| Login | Data | Description |
|-----------------|---|-----------------------------|
| Request | Command="Login Request Code" Username="username" Password="Password" X="coordinates latitude" Y = "coordinates longitude" | |
| Response | Error="error code" Session="Session Identity" Email="user email" Name="name" Facebook="facebook ID" ... (More fields can be added) | The login request succeeded |
| Response | Error="error code" | The login request failed |

Table 5: ProxiConn Communication Protocol – Login Request

| Login | Example Data | Description |
|-----------------|--|-----------------------------|
| Request | Command=1 Username="panos" Password="1234" X=24.4789653995090 Y = 35.3636764876980 | |
| Response | Error=0 Session="52c56427b7e89" | The login request succeeded |

| | | |
|-----------------|---|---|
| | Email="panos@gmail.com" Name="Panos Kalodimas" Facebook="Panos K" | |
| Response | Error=11 | The login request failed with error message "Invalid username or password!" |

Table 6: ProxiConn Communication Protocol – Login Request Example

1.3.3. Logout

The «Logout» request is the inverse procedure of the «Login» one. The client requests to set himself offline of the «proxiConn» services and be invisible to his proximate users. The only information he needs to provide is its session identity. If a user is inactive for a period of time the server will automatically log him out.

| Logout | Data | Description |
|-----------------|---|------------------------------|
| Request | Command="Logout Request Code" Session="Session Identity" | |
| Response | Error="error code" | The logout request succeeded |
| Response | Error="error code" | The logout request failed |

Table 7: ProxiConn Communication Protocol – Logout Request

| Logout | Example Data | Description |
|-----------------|--------------------------------------|---|
| Request | Command=2 Session="52c56427b7e89" | |
| Response | Error=0 | The logout request succeeded |
| Response | Error=9 | The logout request failed due to invalid session identity |

Table 8: ProxiConn Communication Protocol – Logout Request Example

1.3.4. Location Update

The «Location Update» request updates the client's location coordinates and also the session with the server. The client uses its session identity to identify him and the values of his location coordinates

| Location | Data | Description |
|-----------------|--|--------------------------------|
| Request | Command="Location Request Code" Session="Session Identity" X="coordinates latitude" Y = "coordinates longitude" | |
| Response | Error="error code" | The location request succeeded |
| Response | Error="error code" | The location request failed |

Table 9: ProxiConn Communication Protocol – Location Request

| Location | Example Data | Description |
|-----------------|--|--|
| Request | Command=3 Session="52c56427b7e89" X=24.4789653995090 Y = 35.3636764876980 | |
| Response | Error=0 | The location update request succeeded |
| Response | Error=11 | The location update request failed due to invalid session identity |

Table 10: ProxiConn Communication Protocol – Location Request Example

1.3.5. Neighbors

The «Neighbors» request retrieves the proximate user-clients information. The client sends his session identity to validate himself, his location coordinates and a range value in order the server to return all the users inside the circle with the user's location as the center and the range value as its radius.

| Neighbors | Data | Description |
|----------------|--|-------------|
| Request | Command="neighbors Request Code" Session="Session Identity" X="coordinates latitude" Y = "coordinates longitude" Range="Neighbors Range" | |

| | | |
|-----------------|---|--|
| Response | Error="error code" Neighbors= [username, latitude, longitude] | The neighbors request succeeded returning back the proximate users array |
| Response | Error="error code" | The neighbors request failed |

Table 11: ProxiConn Communication Protocol – Neighbors Request

| Neighbors | Example Data | Description |
|-----------------|--|--|
| Request | Command=4 Session="52c56427b7e89" X = 24.4789653995090 Y = 35.3636764876980 Range = 1000 | |
| Response | Error=0 Neighbors= { "adonis",24.478965399509000, 35.36367648769800} { "kostas",24.473165398814000, 35.367076457188000} { "gogo",24.478965398610000, 35.357676457799000} { "john",24.477859835293000, 35.367998212177000} | The neighbors request succeeded returning back the proximate users array |
| Response | Error=11 | The neighbors request failed due to invalid session identity |

Table 12: ProxiConn Communication Protocol – Neighbors Request Example

1.3.6. User Info

When the user wants to have more information about a proximate user, the client application sends the «User Info» request asking the server for the public information of that user. Despite of the session identity the client sends the username of the user is interested in.

| User Info | Data | Description |
|----------------|--|-------------|
| Request | Command="User Info Request Code" Session="Session Identity" username="proximate user username" | |

| | | |
|-----------------|---|--|
| Response | Error="error code" Name="name" Facebook="facebook ID" ... (More fields can be added) | The user info request succeeded returning back the proximate user public information |
| Response | Error="error code" | The user info request failed |

Table 13: ProxiConn Communication Protocol – User Info Request

| User Info | Example Data | Description |
|-----------------|---|--|
| Request | Command=5 Session="52c56427b7e89" Username="Yannis" | |
| Response | Error=0 Name="Yannis Papas" Facebook="yannis p" | The user info request succeeded returning back the proximate user public information |
| Response | Error=51 | The user info request failed due to invalid username |

Table 14: ProxiConn Communication Protocol – User Info Request Example

1.3.7. Signup

Then «Signup» request is send by the client when the user tries to create a new account for joining the «proxiConn» project. The data required to be send is the username, the password and the user's e-mail address. There are some optional fields as user's public information like his name and his facebook identity. After this request the server send a validation e-mail to the user's e-mail address. The user has to validate that way the information he filled in the signup form.

| Signup | Data | Description |
|----------------|---|-------------|
| Request | Command="Signup Request Code" Username="username" Password="Password" Email="email address" (Optional) Name="name" Facebook="facebook ID" ... (More fields can be added) | |

| | | |
|-----------------|--------------------|------------------------------|
| Response | Error="error code" | The signup request succeeded |
| Response | Error="error code" | The signup request failed |

Table 15: ProxiConn Communication Protocol – Signup Request

| Signup | Example Data | Description |
|-----------------|---|--|
| Request | Command=6 Username="panos" Password="1234" Email="panos@gmail.com" Name="Panos Kalodimas" Facebook="Panos K" | |
| Response | Error=0 | The signup request succeeded. Server is waiting for the validation. |
| Response | Error=66 | The signup request failed with error message "E-mail address already exists" |

Table 16: ProxiConn Communication Protocol – Signup Request Example

1.3.8. Sign out

If the user wants to unregister from the «proxiConn» project he can choose to send a «Sign Out» request from the client application. The user must be logged first in order to send this request. The session identity is enough to validate its identity.

| Sign out | Data | Description |
|-----------------|---|--------------------------------|
| Request | Command="Sign out Request Code" Session="Session Identity" | |
| Response | Error="error code" | The sign out request succeeded |
| Response | Error="error code" | The sign out request failed |

Table 17: ProxiConn Communication Protocol – Sign out Request

| Sign out | Example Data | Description |
|-----------------|--------------------------------------|--------------------------------------|
| Request | Command=7 Session="52c56427b7e89" | |
| Response | Error=0 | The sign out request succeeded. User |

| | | |
|-----------------|---------|--|
| | | account is deleted. |
| Response | Error=9 | The neighbors request failed due to invalid session identity |

Table 18: ProxiConn Communication Protocol – Sign out Request Example

1.3.9. User Edit

The «User Edit» request is used for updating the user's information. The user can change the information fields of his profile any time he wants by sending this request from the client application. The only field of his account he cannot change is his username as it's his unique identity. This request includes his profile data with their new values.

| User Edit | Data | Description |
|-----------------|---|---------------------------------|
| Request | Command="User Edit Request Code" Password="Password" Email="email address" Name="name" Facebook="facebook ID" ... (More fields can be added) | |
| Response | Error="error code" | The User Edit request succeeded |
| Response | Error="error code" | The User Edit request failed |

Table 19: ProxiConn Communication Protocol – User Edit Request

| User Edit | Example Data | Description |
|-----------------|--|---|
| Request | Command=8 Session="52c56427b7e89" Password="1234" Email="panos@gmail.com" Name="Panos Kalodimas" Facebook="Panos K" | |
| Response | Error=0 | The User Edit request succeeded. User account is updated. |

| | | |
|-----------------|----------|--|
| Response | Error=64 | The User Edit request failed with error message "Password length less than 4 digits" |
|-----------------|----------|--|

Table 20: ProxiConn Communication Protocol –User Edit Request Example

1.4. ProxiConn Communication Protocol Errors

In this chapter some of the common error that the proxiConn project may produce to the user client side when using the client application is described. These error are listed in the Table 21,

| Error Code | Error Message | Description | Request Produce |
|------------|--|---|-----------------------------------|
| 0 | Operation successfully completed! | This is not actually an error but a successful request confirmation | ALL |
| 1 | Server connection error. Server not reachable! | The server is not reachable | ALL |
| 2 | User canceled request! | The user canceled the request himself | ALL |
| 5 | SQL Server connection Error! | The web application failed to connect with the MySQL server instance | ALL |
| 6 | SQL Server Error! | The MySQL server produced an error during the request process | ALL |
| 7 | Invalid request! | An invalid request code received to server | ALL |
| 8 | Invalid Data! | Invalid data received by the server and cannot be decoded | ALL |
| 9 | There is no user logged in! | The session identity is invalid | ALL except Login and Ping |
| 11 | Invalid username or password! | The username and password does not match to any active users | Login |
| 31 | Invalid coordinates value! | The location coordinates values are not the right type of data or empty | Login, Location Update, Neighbors |
| 41 | Invalid range value! | The range value is not the right type of data or empty | Neighbors |
| 51 | Username does not exists! | The username does not match any active user | User Info |

| | | | |
|----|-------------------------------------|---|------------------------|
| 61 | Invalid username! | The username value is not the right type of data or empty | Signup |
| 62 | Invalid password! | The password value is not the right type of data or empty | Signup, User Edit |
| 63 | Invalid email! | The email value is not the right type of data or empty | Signup, User Edit |
| 64 | Password length less than 4 digits! | The password length is less than 4 digits | Signup, User Edit |
| 65 | Username already exists! | The username value matches the username of already active user | Signup |
| 66 | Email already exists! | The email address value matches the email address of already active user | Signup, User Edit |
| 69 | Invalid username or hash code! | The username and the activation code does not match to any inactive user waiting for validation | New account activation |

Table 21: ProxiConn Communication Protocol Errors

1.5. Protocol Data Format

In the «proxiConn» project the amount of data is transmitted between the client and the server is attempted to be as lower as possible due to the low network bandwidth from the side of the clients as is described in chapter 1.1. The «proxiConn» project is an outdoors project and is mainly designed for clients using the WWAN network for accessing the internet. In this chapter the JSON (JavaScript Object Notation) data format that is used for exchanging data through the HTTP Post Data variable in the HTTP Post Requests will be referred. Both the clients and the server encode the data they intent to transmit in this data format type.

The JSON data format was chosen to be the data type format used at the «proxiConn» project because of the following benefits it offers,

- **Simplicity.** The JSON format is very simply to encode and decode and it is easy readable even when the data are encoded. The simplicity has collateral advantages as the next ones.
- **Low Processing Cost.** The JSON format has a low operational cost on encoding and decoding the data as a result of its simplicity.

- **Compatibility.** It is a widely used independent language data format and is supported by many software libraries.
- **Reliability.** It is supported by both the PHP5 and iOS providing official encode and decode software libraries.
- **Efficient Data Ratio.** The ratio of data encoded relative to data used (as far as the amounts of data used at the «proxiConn» project)

The JSON format is not analyzed in this reference because this is out of scope, although some examples of requests data encoded are presented in Table 22 in order to prove the efficiency of using this format as long as the ratio of the useful information in includes. The JSON format encodes the data in similar way that the associative arrays do.

| JSON Examples | |
|--------------------------------------|--|
| Login data send | <code>{"c"=1,"u"="panos","p"="1234", "x"=24.4779551755120,"y"=35.3679988438650}</code> |
| Login data received | <code>{"e"=0,"s"=" 52c56427b7e89", "m"="panos@gmail.com","n"="Panos","f"="PanoS"}</code> |
| Location update data send | <code>{"c"=3,"s"=" 52c56427b7e89", "x"=24.477955175512000,"y"=35.367998843865000}</code> |
| Location update data received | <code>{"e"=0}</code> |
| Neighbors data received | <code>{"n":[{"u":"adonis","x":"24.478965399509000","y":"35.3636764 87698000"}, {"u":"kostas","x":"24.473165398814000","y":"35.367076457188 000"}, {"u":"gogo","x":"24.478965398610000","y":"35.3576764577990 00"}, {"u":"john","x":"24.477859835293000","y":"35.3679982121770 00"},"e":0}</code> |

Table 22: JSON format Requests Data Example

CHAPTER 2

DATABASE SERVER

The database part of the «proxiConn» project architecture is implemented using the Oracle's MySQL server (version 5.1.72) running on an Ubuntu server (version 10.04) machine. The MySQL server service was chosen cause of its high reliability and its open source license.



Figure 4: MySQL Logo

For this project two data tables are used. The first one is maintains the project's users profiles and the second one keeps the online users data.

2.1. Users Data Table

The «Users» data table is used for maintaining the registered users' profiles. It's a simple information data table. Its structure is presented in Table 23. Its primary key is the username column as the username field of each user must be unique. Every user needs a username and a password field for joining the «proxiConn» project.

| Columns | Description |
|-----------------|---|
| Username | The username of each user. Is unique and used as primary key. |
| Password | The password of each user. |
| Email | The e-mail address of each user. It is also unique. |
| Status | This field defines if a user profile is active or not. Every time a new profile is created it must be activated afterwards. This process is accomplished through e-mail activation. |
| Hash | The hash code generalized during the creation of each profile in order to be identified in the activation process. |
| Name | Information field where a user can assign his name. It is optional and public information. |

| | |
|-----------------|---|
| Facebook | Information field where a user can assign his Facebook profile for sharing more information about himself. It is optional and public information. |
| ... | More information fields can be added in the future. |

Table 23: Database Server – Users Data Table Structure

2.2. Online Users Data Table

The «Online Users» data table is the online data structure as it maintains the online users' data. This data structure is the main destination and source of the majority of the data streams. After a user is logged in the «proxiConn» project all the information data he sends and receives comes and head to this data structure. Its structure is represented in Table 24. The primary key of this table is the session identity column. This identity is ascribed to each user by the web server application and is unique for each TCP/IP connection. Further details about session identities are described in chapter 3.1, where the web server application is described. The «Online Users» data table keeps information about the time and date each user last updated his location information. If that time overpasses a predefined period (defined by the Administrator), the user is considered offline.

| Fields | Description |
|------------------|--|
| Sid | The session identity. Is unique and used as primary key. |
| Username | The username of each user. |
| Latitude | The geographical latitude of the user. |
| Longitude | The geographical longitude of the user. |
| Time | The time and date the user last updated his location. If the user does not update his location for more than a standard time period he will be removed from the online users table |

Table 24: Database Server – Online Users Data Table Structure

2.3. Procedures and Function

For improving the compatibility, safety and the communication between the database and the web application client server, a functions library was designed. This library constitutes by the following functions and stored procedures.

- **Activate Account:** This function validates the username and the hash code of a user's activation request and activates its account if the validation succeeds.
- **Clear Logs:** This function checks the activity of each user and removes those considered offline from the «Online Users» data table.
- **New User:** This function validates the information attached in a user's «Signup» request (Chapter 1.3.7) and registers the new user in the «Users» data table if the validation succeeds.
- **Delete User:** When a user request to sign out (Chapter 1.3.8) from the «proxiConn» project, this function takes care for removing all data related to the specific user.
- **User Login:** When a user requests to log in (Chapter 1.3.2), «User Login» function validates the log in information and registers the user in the «Online Users» data table if the validation succeed.
- **User Logout:** This function removes a user from the «Online User» data table.
- **User Edit:** This function updates a user's profile information after the corresponding request (Chapter 1.3.9).
- **User Info:** This function returns a user's public information. These are information that a user can have about his proximate users. They are public and not private data that each user shares with the rest.
- **User Profile:** After a successful log in to the server, the client receives its profile information. This kind of information is private in addition to the public information that «User Info» function returns.

- **Location Update:** This function is used for updating the location information of each user every time he informs its location by a «Location Update» request (Chapter 1.3.4)
- **Calculate Distance:** This function calculates the distance in meters between two locations using the mathematic type mentioned in chapter 1.2. This function is called multiple times when a user requests his proximate users.
- **User Neighbors:** The main usage of the «proxiConn» project is enclosed in this function. It is the one that calculates the distance between the online users and returns all the proximate users within the range the requested user set. The «Calculate Distance» function is called multiple times in this store procedure.

This library works like a protocol used by the web application in order to communicate with the database server for importing or exporting data. In that way the minimum blend between the SQL and PHP scripting language was achieved, creating low dependency between the web application and the database server.

CHAPTER 3

WEB SERVER APPLICATION

Between the database server and the user-clients, mediates the web server application. The role of this application in the «proxiConn» project architecture is to operate as a security shield for the database server and extend the user-client services. The web application is developed using the PHP (version 5) scripting language and an Apache (version 2.2.14) HTTP server on an “Ubuntu 10.04” server machine.



Figure 5: Ubuntu Server, Apache Server & PHP Logos

3.1. Client Services

The web application offers some services in a layer upper the database server. Those services are,

- Encoding and decoding the data it receives and sends to the protocol data format (Chapter 1.5). The web application according to the «proxiConn» project architecture is the one that receives the requests from the clients and the one sending back the final responses. This makes it the layer that has to do the decoding and encoding process of the incoming and outgoing data. The PHP scripting language has already available libraries for doing that.
- Start and destroy session identities whenever a user is either logged in or out. One of the basic responsibilities of the web application is to start a session after every successful «Login» request (Chapter 1.3.2). When a new session starts, a unique session identity is created and is used as the online user’s identity. Every

online user owns a unique session identity and uses it for identifying himself on his requests.

- Sends verification emails to the new users. After a successful request for creating a new account, every user has to activate his account by verifying his email address. For that purpose the web application sends a verification email. For doing so, it has to connect to an SMTP server and use its services. In the «proxiConn» the university of Crete SMTP server is used.
- Validates the verification requests of the new users. For validating a verification request the web application uses hash streams created randomly using the MD5 algorithm.

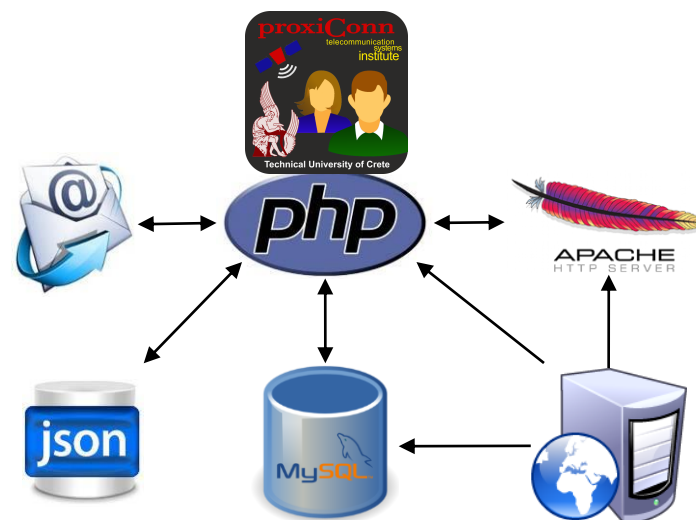


Figure 6: Web Application Services Architecture

- Handles database server errors. The web application connects to the database MySQL server in behalf of the users in order to extract or update data. The PHP scripting language is much more flexible than the SQL one, providing a more flexible environment for avoiding and handling errors. It is insecure if every user has a personal account for the database server. Direct communication between the users and the database could cause security issues, as there is no filter on what data streams can be directed to or from the database server. This could cause data leakages or bandwidth overflows. On that point the web server application creates a data streaming filter. The web application is the only

authorized user to access the database server and is the representative of the clients for updating or requesting data. Invalid data or requests by the side of the clients are rejected before reaching the database level, reducing the bandwidth and the amount of processing to the database server, reducing the possibility of run time errors.

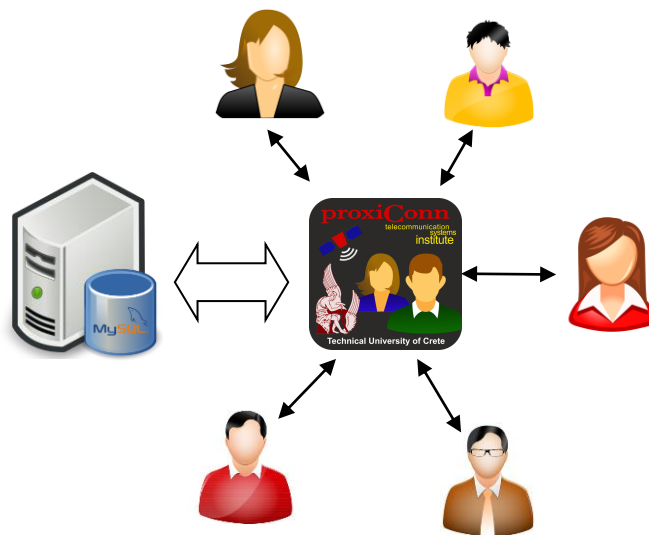


Figure 7: Web Application – Clients Server

3.2. Clients Requests

As it's already mentioned the web application is responsible for serving the clients' requests and interacting with the database server behalf them. In this chapter this procedure is described in detail.

The first thing the web application does is to check the data attached to the HTTP Post Body and try to decode them. Those data include all the information needed for the request's process. If those data fail to be decoded the request is rejected producing an error. After a successful decoding process the web application handles each request according to the type it represents (Chapter 1.3). The rest subchapters of this one are devoted on that issue. At the end, whatever the result of a request is, the web application has to create an encoded data package according to the predefined communication protocol and send it back to the client as the response to its request. In the Figure 5 below a general flow diagram of the web application is

exposed. Specific flow diagrams for each request type are shown in the related subchapters.

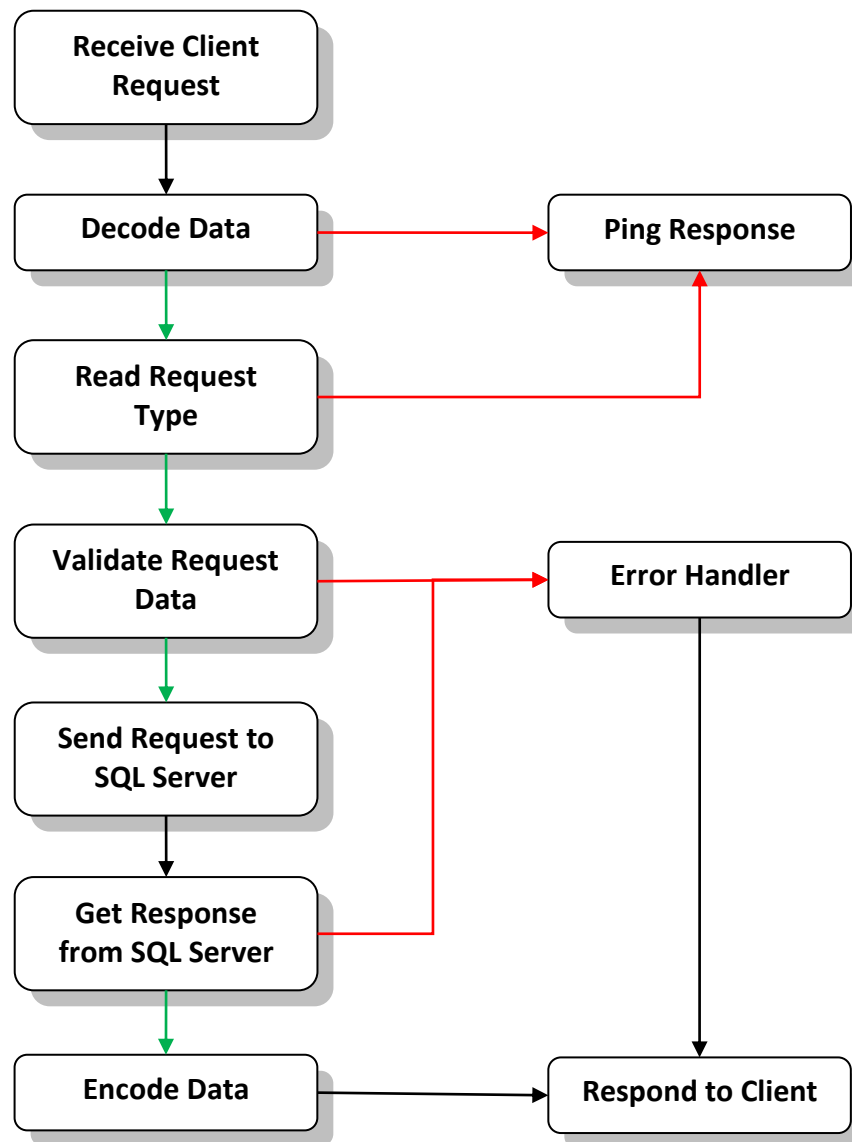


Figure 8: Web Application – General Flow Diagram

3.2.1. Ping Request

The web application treats as «Ping» requests not only the requests that does observer the «proxiConn» communication protocol (Chapter 1.3) but also all the requests that is not clear of what kind of requests are. If for example, the request's data fail to be decoded or the request type is not defined, the web application will treat it like it's a «Ping» request (Chapter 1.3.1) with the only difference on the error

code that will be returned. The database server does not participate to this request's procedure at all as the web application completely handles it.

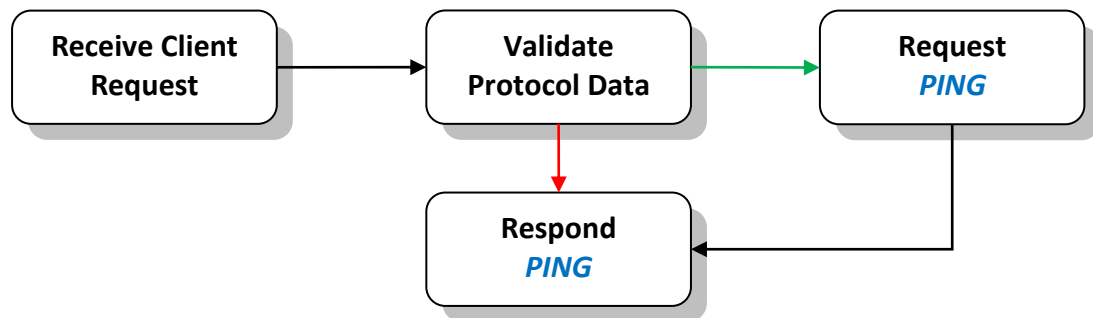


Figure 9: Web Application – Ping Request Flow Diagram

3.2.2. Login Request

At a «Login» request the web application makes a «Login» call (Chapter 2.3) to the database server. If it is completed successfully it makes a «User Profile» call (Chapter 2.3) to get the user's profile data. At last it starts a new session with the user as the communication protocol demands (Chapter 1.3.2).

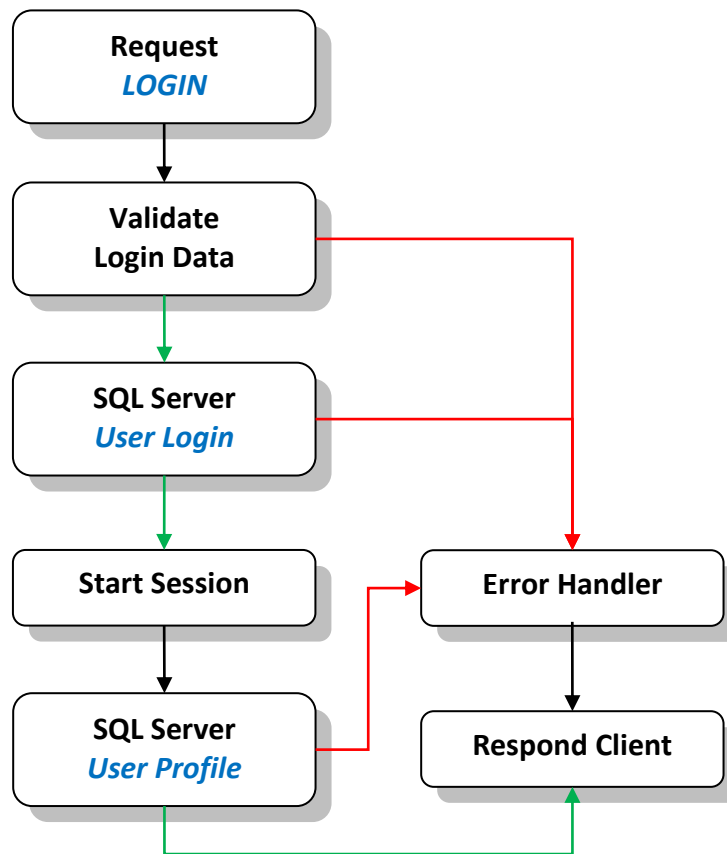


Figure 10: Web Application – Login Request Flow Diagram

3.2.3. Logout Request

At a «Logout» request (Chapter 1.3.3) the web application makes a «Logout» call (Chapter 2.3) to the database server and destroys the session with the user.

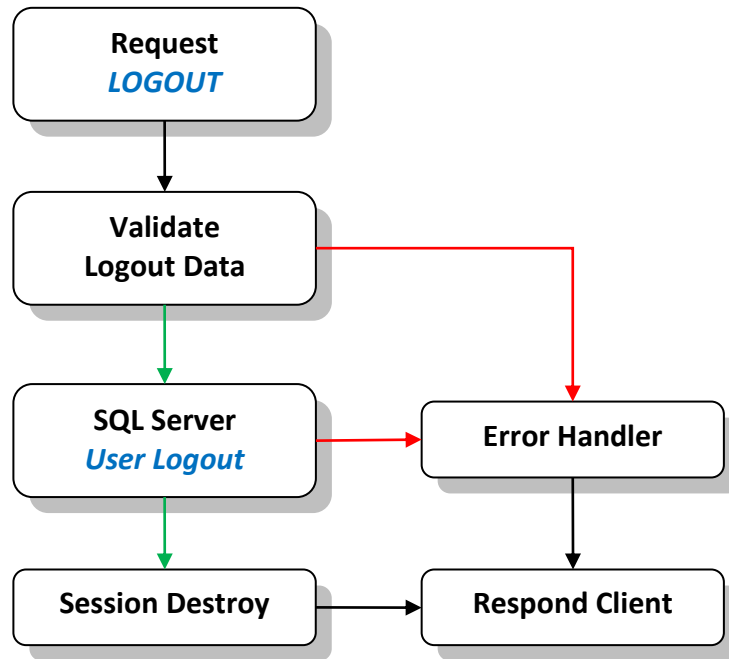


Figure 11: Web Application – Logout Request Flow Diagram

3.2.4. Location Update Request

The «Location Update» function (Chapter 2.3) is called on the «Location Update» requests (Chapter 1.3.4).

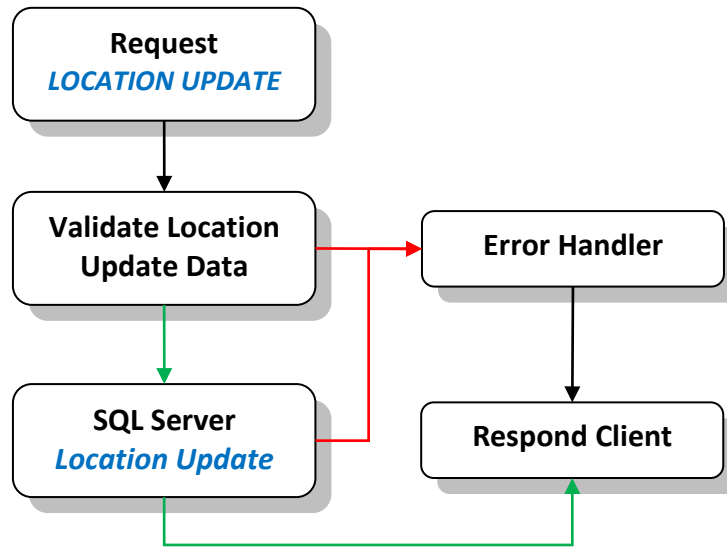


Figure 12: Web Application – Location Update Request Flow Diagram

3.2.5. Neighbors Request

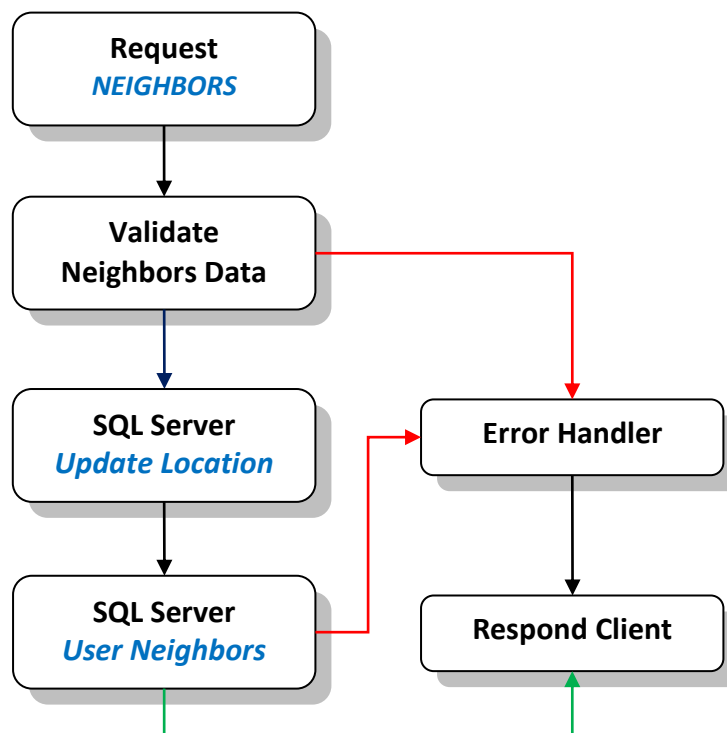


Figure 13: Web Application –Neighbors Request Flow Diagram

At a «Neighbors» request the web application first makes a «Location Update» call and then a «User Neighbors» one (Chapter 2.3). The last call returns a great size of

data (compared to the rest requests!) in success that the web application has to transform according to the communication protocol rules (Chapter 1.3.5). If the first function call fails, the web application does not returns an error as it does not influences the procedure flow. The «User Neighbors» call uses the last successfully updated location saved to the database. The reason why the system updates the user's location just before the «User Neighbors» call is because it tries to achieve the most accurate results possible.

3.2.6. User Info Request

On «User Info» requests, the web application forwards the username field to the «User Info» SQL stored procedure (Chapter 2.3). Then it encodes the data are returned to the HTTP Post request respond.

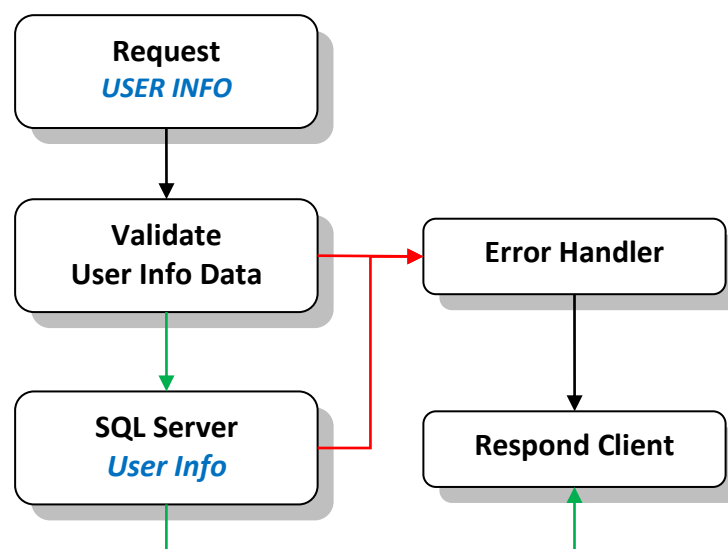


Figure 14: Web Application –User Info Request Flow Diagram

3.2.7. Sign up Request

In a «Sign up» request (Chapter 1.3.7) the web application has a more active role. It needs two parts to be completed. In the first part the web application validates the data fields it receives from the HTTP Post request and forwards them to the database server by calling the «New User» function (Chapter 2.3). It also creates a random hash code using the MD5 algorithm that will be used later in the account

activation (second part). When the «New User» call is completed, it returns an error code informing the operation result. If the «New User» call fails, the web application informs the client about the reasons of this failure. For example, an already in use e-mail address by another registered user produces an error. On the other hand, a successful «Sign up» request needs the activation process to be completed. The web application sends an email message to the user's email address asking him to activate his account using an attached URL address.

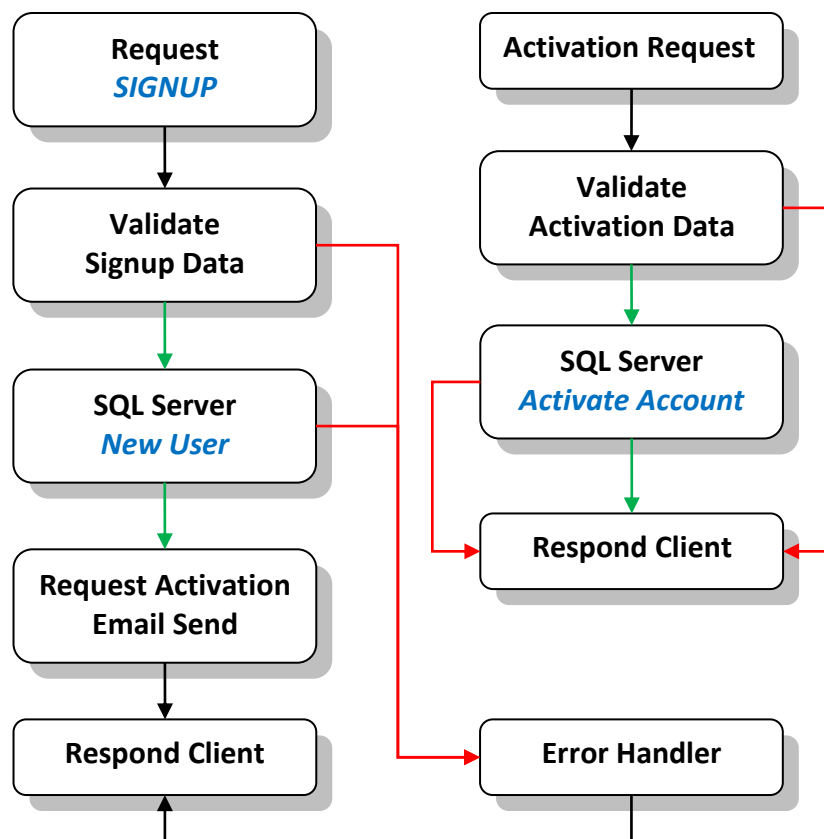


Figure 15: Web Application –Sign up Request Flow Diagram

The second part of the «Sign up» request begins when the user uses the activation URL attached in the activation email he received. This URL contains a hash code and his username. These parameters are passed to the database server using the «Account Activation» call (Chapter 2.3). If the process succeeds the new account is ready for use.

3.2.8. Sign out Request

The database server handles all the operations on a «Sign out» request (Chapter 1.3.8). The web application just calls the «Delete User» function (Chapter 2.3) and if it succeeds it destroys the user-client's session.

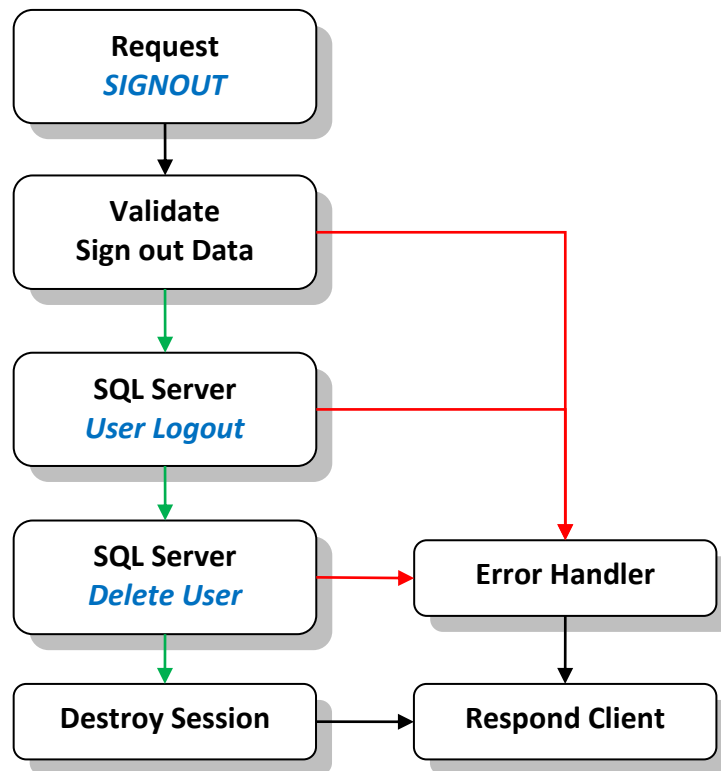


Figure 16: Web Application –Sign out Request Flow Diagram

3.2.9. User Edit Request

A «User Edit» request (Chapter 1.3.9) is a simple update request. The web application makes a «User Edit» call (Chapter 2.3), which makes all the operations and returns an error code defining its termination status.

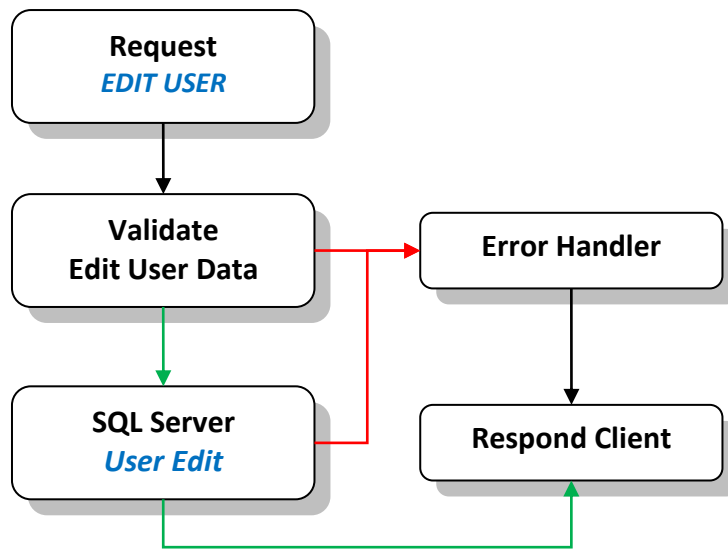


Figure 17: Web Application –User Edit Request Flow Diagram

CHAPTER 4

USER-CLIENT APPLICATION

For the implementation of the «proxiConn» project appropriate software for both server and user-client sides had to be developed. By the client's side, the Apple Inc «iPhone 4» mobile device was used, so the corresponding software was developed for the iOS operation system which is used by the whole Apple Inc's mobile devices family. In this chapter the architecture of this application will be described. In addition, a few words about the «iPhone 4» mobile device and the iOS operation system will be appose.

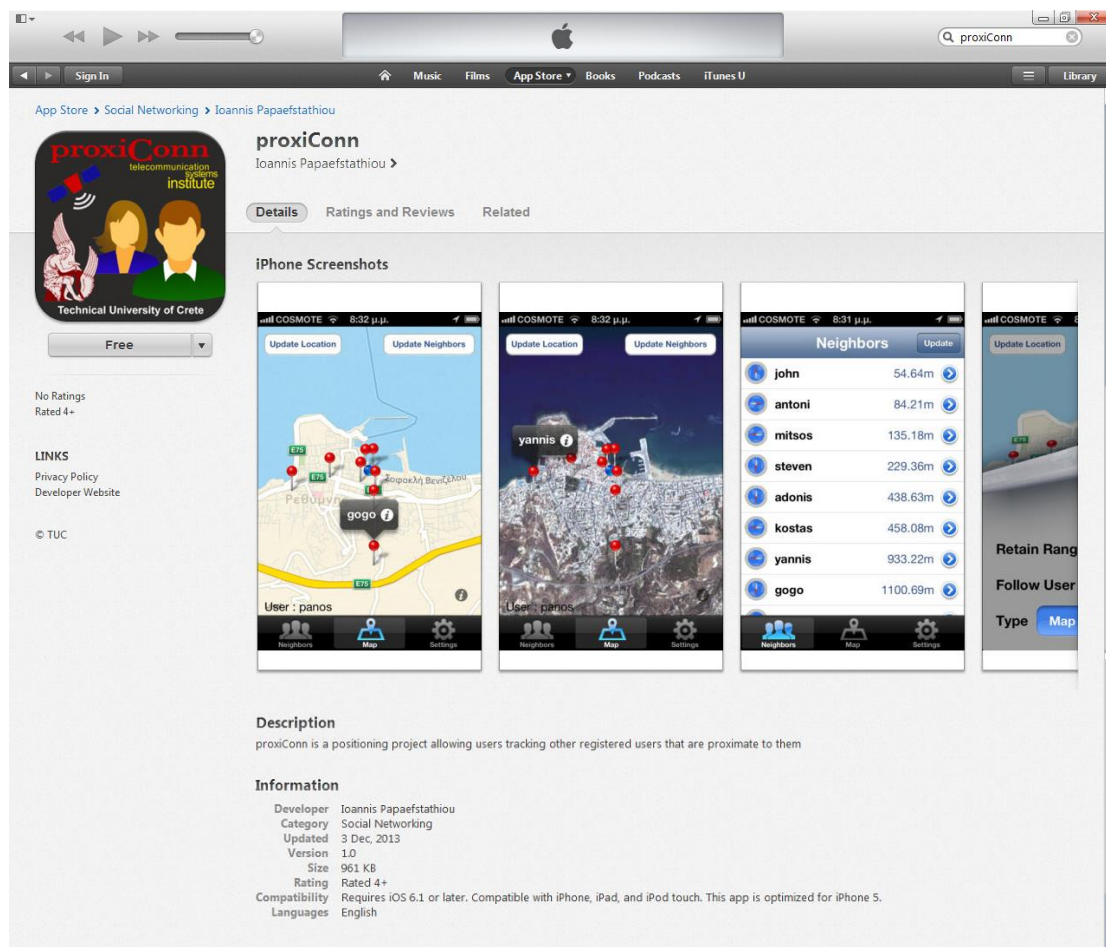


Figure 18: proxiconn iOS user-client application on the Apple AppStore

4.1. iPhone 4 Mobile Device

The «iPhone 4» mobile device is a device in the category of «smart» phones used by thousands of people every day for communicating through the GSM mobile network. Like most of the devices of this category, it is provided with a stack of accessory modules like a camera, an accelerometer, WLAN, WWAN, GPS and many other helpful modules. In the «proxiConn» project the connectivity modules and the GPS service module where are used. Fortunately these kinds of extensions are supported in the majority of «smart» devices.

The more important part of a candidate mobile device to be used in the «proxiConn» project is its connectivity modules and the support of the GPS service. The processing capabilities come in second hand as the user-client application is a low processor cost application. The «iPhone 4» meets these requirements. Despite its high processor's capabilities this device is not considered a state of art device as its production date started in March 2010 and is now bypassed by three newer versions published by Apple Inc (iPhone 4S, iPhone 5, iPhone 5S). In addition to its high price in market, the capabilities of «iPhone 4» are nowadays in the average of the «smart» phones family as newer devices are getting more and more ahead. Companies like Samsung, LG, Nokia, Sony and many others, produce lots of low price compatible to the «ProxiConn» project.

iPhone 4 Specification Summary

| | |
|-----------------|----------------------------|
| Produced | From March 2010 |
| OS | iOS 4 to 7 |
| Chipset | Apple A4 |
| CPU | ARM Cortex-A8, 1GHz |
| RAM | DDR SDRAM 512 Mb (2x256Mb) |
| GPU | PowerVR SGX535 |
| Storage | 8/16/32 Gb SSD |



Figure 19: iPhone 4 Specifications Summary

4.2. iOS Operating System

The «iPhone 4», like all iPhone family devices, uses the iOS mobile operating system. The iOS operating system is developed and distributed by Apple Inc. It manages the device hardware and provides the technologies required to implement native applications. iOS is derived from MAC OS X, with which it shares the Darwin foundation and is therefore a Unix operating system.

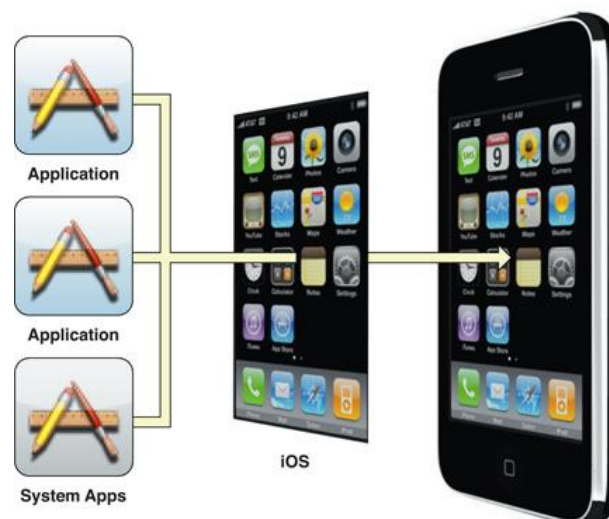


Figure 20: iOS Applications Layer

In the iOS architecture layer, at the highest level, the iOS acts as an intermediary between the underlying hardware and the applications that appear on the screen. The applications communicate with the hardware through a set of system interfaces that protect the application from hardware changes (Figure 20). This abstraction makes it easy to write applications that work consistently on devices with different hardware capabilities.

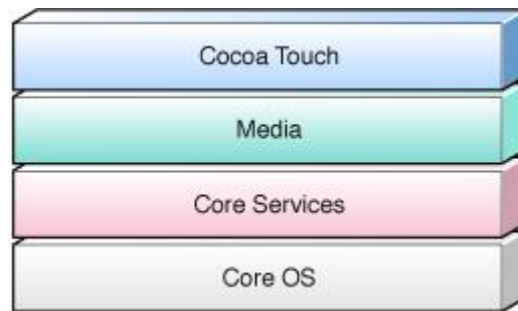


Figure 21: iOS Layers

In iOS, there are four abstraction layers (Figure 21), the Core Services layer, the Media layer, and the Cocoa Touch layer.

At the lower layers of the system are the fundamental services and technologies on which all applications rely. The Higher-level layers contain more sophisticated services and technologies.

The higher-level frameworks are there to provide object-oriented abstractions for lower-level constructs. These abstractions generally make it much easier to write code because they reduce the amount of code have to written and encapsulate potentially complex features, such as sockets and threads. Although they abstract out lower-level technologies, they do not mask those technologies. The lower-level frameworks are still available for developers who prefer to use them or who want to use aspects of those frameworks that are not exposed by the higher layers.

4.3. User-Client iOS Application

After the hardware and the operating system layer of the user-client application reference in this subchapter the user-client iOS application will be described.

The «ProxiConn» application architecture consists of the following modules,

- Application Manager
- Server Client
- GPS Manager
- User Graphics Interface (GUI)

As shown in Figure 22, in the center of the «proxiConn» user-client application is the «App Manager» module. This module takes the control unit role and handles the actions the user requests.

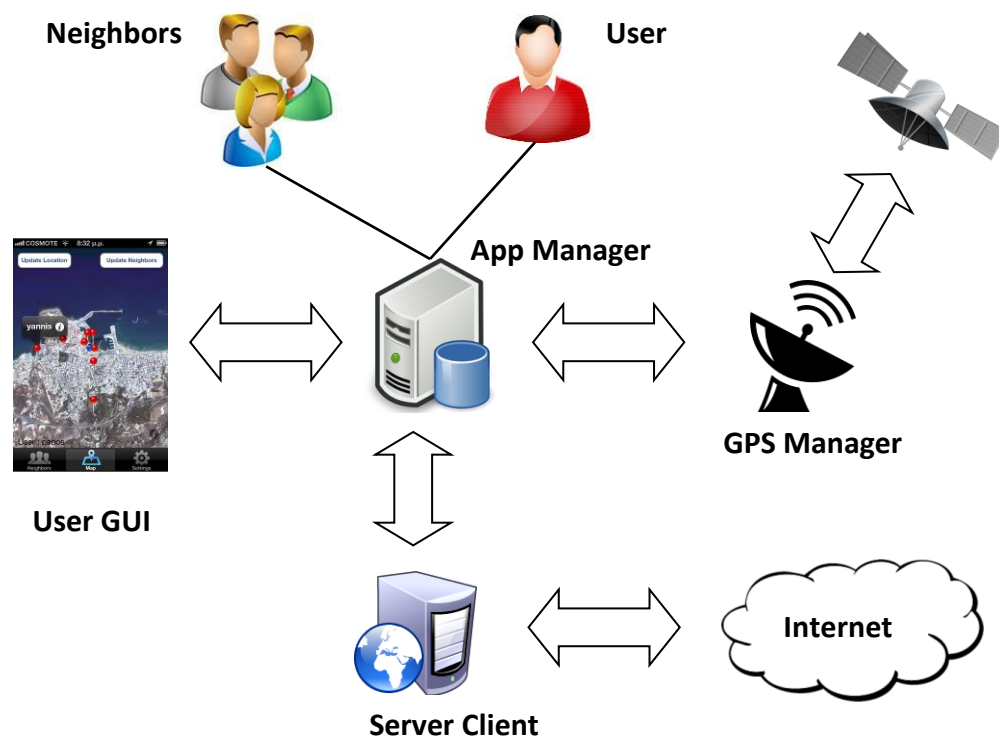


Figure 22: User-Client Application Architecture

4.3.1. App Manager Module

The «App Manager» module as mentioned above is in the center of the «proxiConn» iOS application architecture. All the streams pass through it and are filtered by this module. For example, if the user, using the graphics interface, requests a neighbors' update, the «User GUI» module passes this action to the «App Manager», where, if it satisfies all the requirements, it is forwarded to the «Server Client» module. The

«Server Client» module sends all the server requests' responses to «App Manager» where the last one distributes them to their recipients.

The «App Manager» module does also receives of all application's hardware, frameworks and modules changes like the network connectivity, GPS location changes, server responses, etc. It is its responsibility to synchronize the rest modules and apply all the corresponding actions for the appropriate operation of the «proxiConn» application. It is the module where all decisions are taken. That's why it's called manager!

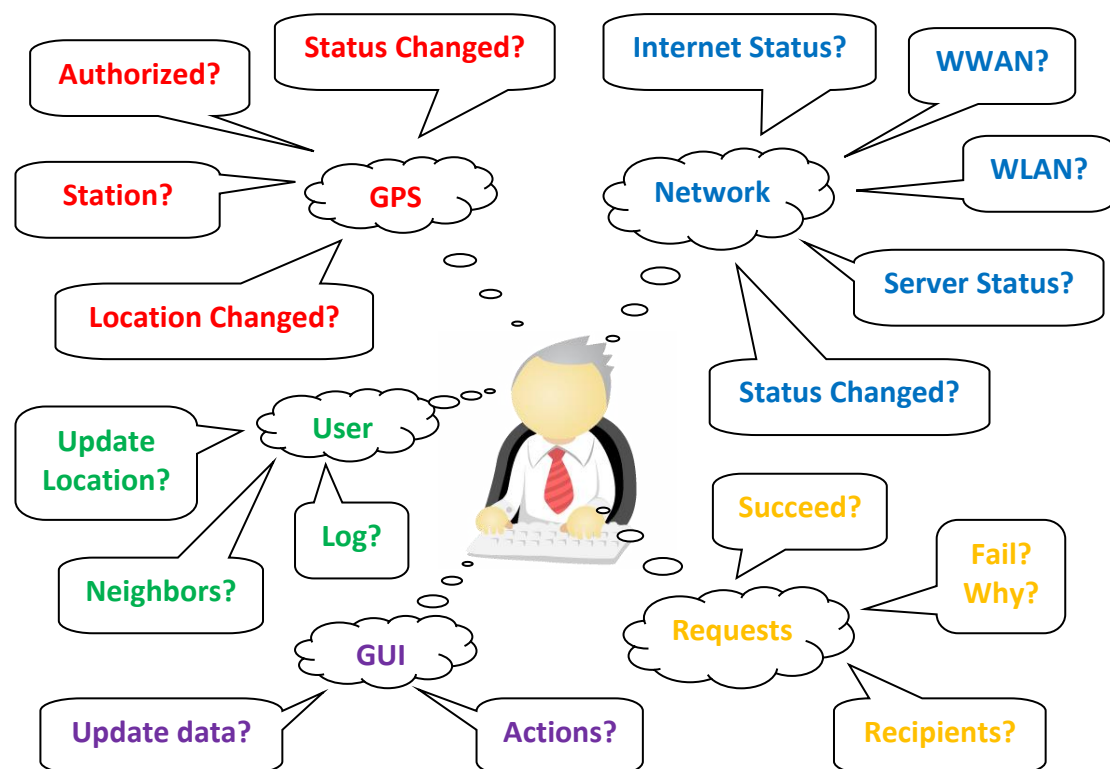


Figure 23: User-Client Application - App Manager Module

At last, the «App Manager» module keeps the data arrays of the application and delivers those data to the «User GUI» module to present them on the screen.

The «App Manager» is designed to work completely independently from the rest modules. Even if any other module stops working, this module is designed to stay stable and keep serving the rest «healthy» ones. If for example the «GPS Manager» or the «Server Client» module stops responding the «App manager» will continue

serving the «User GUI» one rejecting the streams directing to the faulty modules. All these security measures were taken cause of the instability of the «GPS Manager» and «Server Client» module as a result of the instable nature of the GPS, WLAN, WWAN services as explained in chapter 1.1.

4.3.2. GPS Manager Module

The «Gps Manager» module uses the location iOS framework for handling the GPS hardware module of the running iOS firmware device. The only responsibility of this module is to adjust the GPS module operations and inform the «App Manager» for its status and location changes detected. This module also implements the «pseudo-GPS» function of the «Station Mode» where the user defines himself his position. On the Station mode the «Gps Manager» module behaves like an ordinary GPS hardware module. That way none of the rest modules of the application are influenced, even the «App Manager» one. This gives the ability of the user-client application to be fully functional even with no GPS hardware support.

4.3.3. Server Client Module

The «Server Client» module is the one that handles the connections with the central «proxiConn» server. Its responsibilities are,

- Check network and server connectivity and inform the «App Manager» at changes detected.
- Create and handle HTTP Post requests and receive the server responses.
- Encode and decode the user's requests' data and attach them to the HTTP Post requests' body.

The «Server Client» module uses the iOS networking framework to check the existence of networking activity. If changes happen, informs the «App Manager» module to handle them. However, a active networking activity does not warranty connectivity with the «proxiConn» server. For that reason this module has to check on its own if the server is reachable. When the application user is logged in it is easy to check the connection with the server as HTTP Post requests are send frequently in order to update the user's location and its neighbors. On the other hand when the

user is not logged in, the «Server Client» module has to send «Ping» requests (Chapter 1.3.1) to the server to test its reach ability.

When the «App Manager» module decides to make a request to the server it has to call the «Server Client» module to do that. The last one receives the required data according to the request type (Chapter 1.3), transforms them according to the «proxiConn» protocol and attaches them to the HTTP Post request.

For every request, the «Server Client» module opens a TCP/IP connection with the server and sends the HTTP Post request through it. To avoid unwanted traffic to the server the «Server Client» module keeps an active request's array. Only one connection can be opened for each request type. If a module calls for a request while a similar request type connection is already open, the «Server Client» adds this module to the recipients list of the pending request. When the connection is completed all the modules in its recipients list gets a response callback. The request calls and the callback responses are always filtered by the «App Manager» module which is the control unit of the architecture.

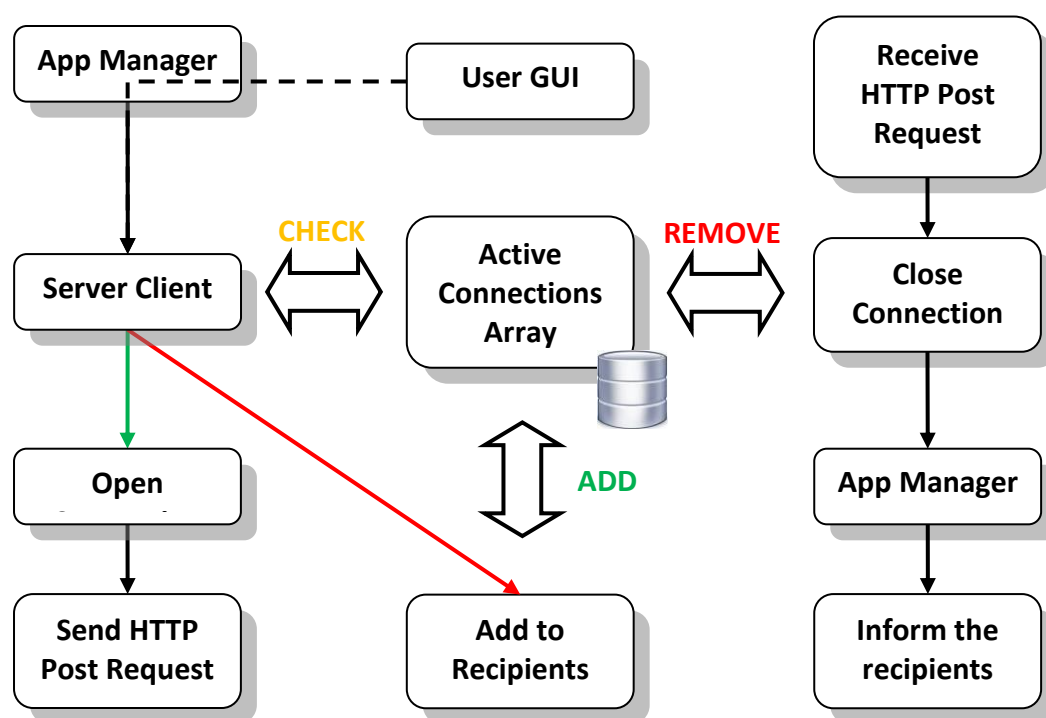


Figure 24: User-Client Application – Server Client Module Flow Diagram

At last the «Server Client» module is responsible for encoding and decoding the requests' data to the «proxiConn» protocol format (Chapter 1.3) and passes them to the «http_post_body» variable of the HTTP Post request. The module is also responsible for creating the error responses for the requests that finished cause of faulty connections and connection errors (never reached the server).

4.3.4. User GUI Module

The graphics user interface (GUI) module is the one that creates an easy and functional graphics interface to facilitate the user in his «proxiConn» project participation. It is responsible for the interaction between the user and the «App Manager» module. The «App Manager» uses this module to inform the user for changes and present him his proximate users. Using the «User GUI» module the user can adjust the application's settings and manually send requests to the «proxiConn» server. However, the «App Manager» decides if those requests will be forward to the server or not.

The description of the user graphics interface is apposed separately in the next chapter because it is also used as the iOS application user guide.

CHAPTER 5

IOS APPLICATION USER GRAPHICS INTEFACE

The «proxiConn» iOS application is a Tabbar based one using three tabs for presenting its contents. These tabs are,

- **Map Tab:** Is the first tab the user faces when starts using the application. In few words it presents a map view, from the iOS map framework, where the proximate users appear pinned. More details on chapter 5.1.
- **Settings Tab:** On that tab the user can see the application connectivity status and adjust many parameters on the way the application operates. He can also manage his account in the «proxiConn» project or create one if he has not. (See chapter 5.4)
- **Neighbors Tab:** This tab is a table view where the proximate users are listed. (See chapter 5.2)

5.1. Map View

The «Map» view (Figure 25) is used for presenting the proximate users of the application user as pins on a map. The user can update his location over the map and bring him on its center by pushing the «Update Location» button at the top left side of the view.

Under the «Update Location» button, as shown in Figure 26 (Right), details about applications services status appear whenever there is lack of at least one of them (network, GPS, server). The defective services are highlighted. That's because the user needs to be informed about the malfunction of the application cause of this loss. If all services work properly this view is



Figure 25: iOS Application GUI – Map View (Satellite Type)

hidden.

At the right top of the view there is the «Update Neighbors» button. By pushing that button the user can update its proximate users manually. The user location on the map is updated automatically every time a change is detected. The neighbors are updated periodically according to the user settings (Chapter 5.4.3).

At the left bottom corner of the view the log status of the application appears reporting the log status of the user as shown in Figure 26 below.

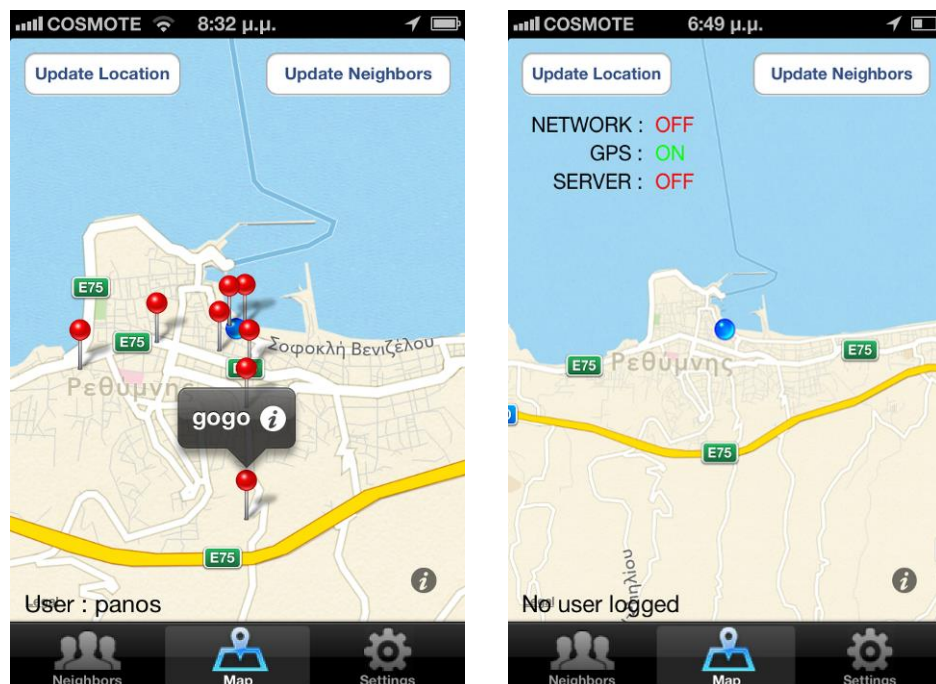


Figure 26: iOS Application GUI – Map View Connected & Disconnected

Whenever the user is interested in one of his proximate users, he can tap over the pin on the screen. By that time a small cloud view will appear presenting the user's «username» information and a small «Info» button as shown in Figure 26 (Left). By pushing that «Info» button the «User Info» view appears. This view will be described separately at chapter 5.3 as it is a common view use also by the «Neighbors» tab.

By pushing the small «Info» button at the bottom right corner of the view the «Map settings» view appears (Figure 27). In this view the user can enable some of the available options for the map view. These options are,

- **Type:** This option defines the type of map the user wants to use. There are three options for the map type parameter. The «Map» option shows a street map (Figure 26), the «Satellite» option shows a satellite map like Google earth does (Figure 25) and the «Hybrid» option is a combination of the two previous (Figure 28).
- **Follow User:** When this option is set to «Auto» segment locks the map centered to the user location. No sliding is allowed. Every time the user's location changes the map follows the user setting him in the center of the view. By setting this option to «Manual» segment the user can slide the map at any direction he prefers centered it only by pushing the «Location Update» button. This is a very useful option when the «proxiConn» application is used by a vehicular user and is set to «Auto» segment.
- **Retain Range:** This option, when set to «Auto» segment, locks the zoom action to the range of the neighbors, defined by the user at the «Settings» view (Chapter 5.4.3). If the user likes to change the map range he has to set this option to the «Manual» segment.

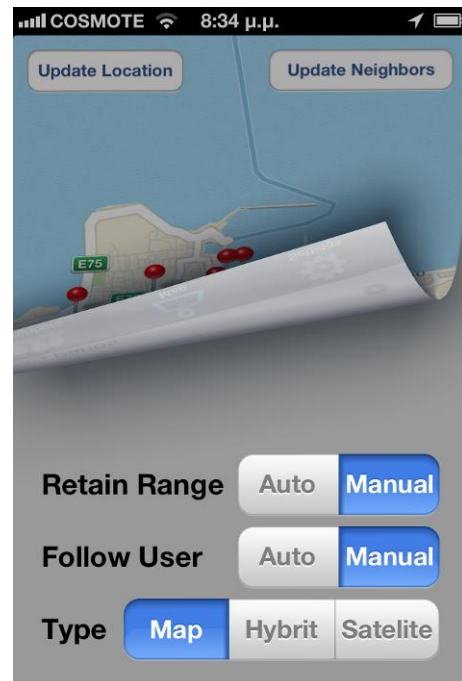


Figure 27: iOS Application GUI – Map Settings View

The client-user «proxiConn» iOS application was designed to be used also in the device's landscape orientation (Figure 28). That's because the landscape orientation offers a very nice appearance of the map view as well a better functionality sometimes.



Figure 28: iOS Application GUI – Map View Landscape (Hybrid Type)

5.2. Neighbors View

The «Neighbors» view, as shown in Figure 30, is a very simple table view presenting the list of the approximate users sorted by its distance from the application user. By tapping the «Details» button at the right side of every row the «User Info» view appears loaded with the selected user's public information as is described in subchapter 5.3 follows.

At the «Neighbors» view, at the left side of the «username» field of each user row, a compass denotes its orientation to the application user. This orientation definition is shown in Figure 29.

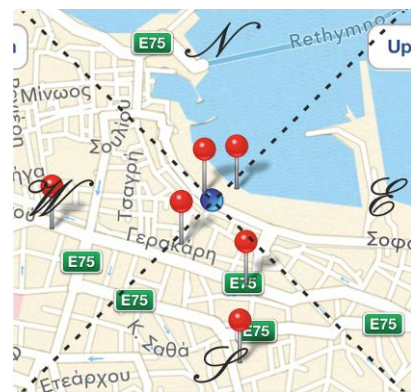


Figure 29: iOS Application GUI – Neighbors View – Orientation Definition



Figure 30: iOS Application GUI – Neighbors View

5.3. User Info View

The «User Info» view is a simple view presenting a selected user's public information and some location details. The location details are the distance between the selected user and the application user in meters, the latitude and the longitude coordinates and his orientation according to the application user (Figure 31).

The user can access the «User Info» view either from the «Map» view, as described in Chapter 5.1, or from the «Neighbors» view (Chapter 5.2).



Figure 31: iOS Application GUI – User Info View

5.4. Settings View

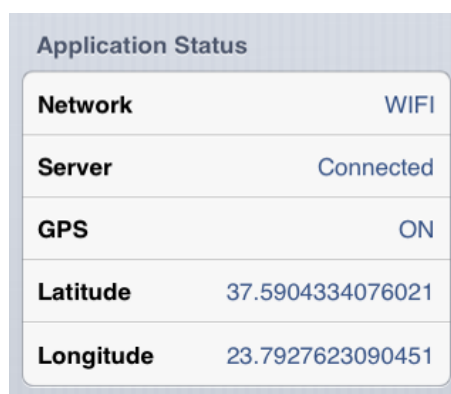
The «Settings» view is where the user can adjust the «proxiConn» application settings. This view has five sections of settings.

1. Application Status
2. User Settings
3. Server Settings
4. GPS Settings
5. Restore Settings

5.4.1. Application Status

In the «Application Status» section the application simply informs the user about the services status.

- The «**Network**» row shows the network connectivity status and which service is used for that (WLAN or WWAN).
- The «**GPS**» row shows if the GPS service is available or not.
- The «**Server**» row shows if the «proxiConn» server responds to the application requests or is not reachable.
- The «**Latitude**» and the «**Longitude**» rows inform the user about its geographical position



| Application Status | |
|--------------------|------------------|
| Network | WIFI |
| Server | Connected |
| GPS | ON |
| Latitude | 37.5904334076021 |
| Longitude | 23.7927623090451 |

Figure 32: iOS Application GUI – Settings View – Application Status

5.4.2. User Settings

The «User Settings» section adjusts some settings about the application behavior relative to the user (Figure 33). It offers the following option to the user,

- **Station Mode:** This option enables\disables the station mode of the «GPS Manager» module (Chapter 4.3.2). By tapping the blue «details» button the «Station» view appears in order to enable or disable the station mode and adjust the station position, as explained in chapter 5.5.
- **Remember Last User:** By enabling this option the user permits the application to keep «username» and «password» information about the last user successfully logged in.
- **Auto Login:** This option activates the application's functionality of automatically trying to log in the server every time all the required services are available. By enabling this option automatically the «Remember Last User» option is enabled for obvious reasons. This is a very useful option when the user has bad quality of services like unstable network connectivity. This option allows him to reconnect automatically whenever the server is reachable again.
- **Log Button:** By pushing the «Log» button the user can either log in or log out.

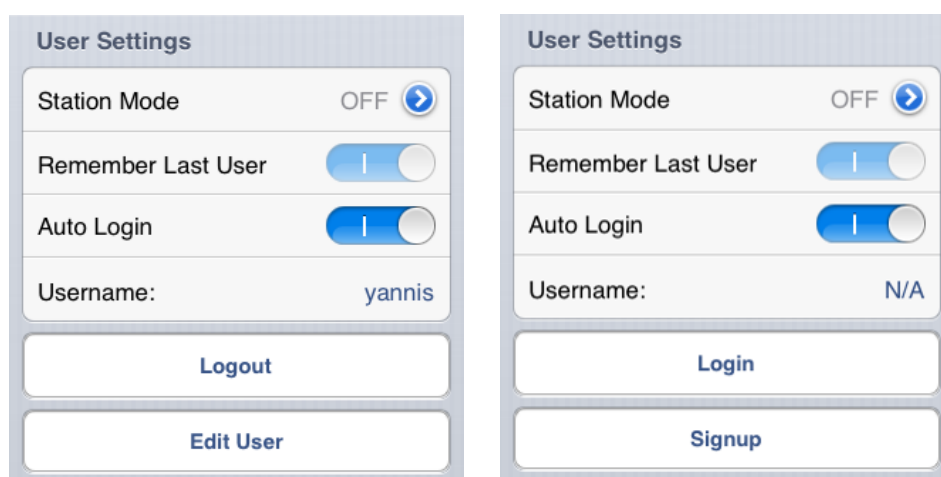


Figure 33: iOS Application GUI – Settings View – User Settings (Logged & Unlogged)

- **«Account» Button:** This button triggers the account view to appear. This view is described in chapter 5.5.

5.4.3. Server Settings

The «Server Settings» view (Figure 34) is for adjusting the application behavior in relation with the «proxiConn» server. Changing these settings will necessitate the «App Manager» module (Chapter 4.3.1) to restart the «Server Client» one in order to configure it to the new settings. To apply any changes to the application, the user has to push the «Save» button at the bottom of the section. The «Server Settings» section consists of the following options,

- **Neighbors Range:** This option defines the distance range around the user location inside of which proximate users are searched for. For a wide range of values there is a multiplication segment switch under the slider. The user can select the «ALL» segment if he likes to receive all the online users by the «proxiConn» server.
- **Neighbor Update Time:** With this option the user defines how often he would like his proximate users list to be updated. Very short time period is not recommended because it overloads both the «proxiConn» server and the user-client application. Vehicular user can use short periods for updating its proximate users, in addition to the pedestrian ones that can be satisfied with longer periods of updates.

- Session Update Time:** The session between the user-client and the «proxiConn» server is updated every time the user has a successfully «Location Update» request operation (Chapter 1.3.4). As the user updates his location coordinates also every time he requests for his proximate users, this option's value cannot be greater than the «Neighbor Update Time» option one. This would be pointless and only cause larger data traffic to the server. Although setting this option to be equal to the «Neighbors Update Time» option value causes reduction of the server traffic as only «Neighbor» requests (Chapter 1.3.5) are send and no «Location Update» ones. Whenever the server is not reachable, this option is used as the period of time the application tries to reach the server by sending «Ping» requests (Chapter 1.3.1).
- Server:** In this text field the «proxiConn» server domain is defined. If this option is wrong the application will consider the server unreachable and will not try to connect.
- Server URL:** In this text field the «proxiConn» server URL is defined. This is the URL where the application sends the HTTP Post requests. If this option is filed wrong the application requests will be all faulty.

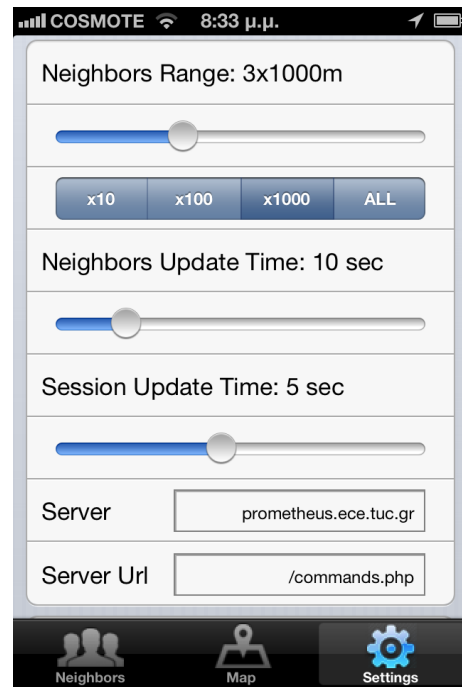


Figure 34: iOS Application GUI – Settings View – Server Settings

5.4.4. GPS Settings

The «GPS Settings» section (Figure 35) relates with the way that the device's GPS module will be adjusted. Changing these settings will cause the «App Manager» module (Chapter 4.3.1) to restart «GPS Manager» one (Chapter 4.3.2) in order to

adjust it to the new parameters. The user has to push the «Save» button to apply any changes he makes. The «GPS» adjustment options are the following,

- **Accuracy:** This option defines the accuracy the user would like the GPS module try to achieve. However the GPS services cannot warranty this accuracy. The lower this option is set the less battery is needed by the device's GPS module.
- **Distance Filter:** The «Distance Filter» option defines the minimum distance between two location changes that will cause a location change trigger by the «GPS Manager» module (Chapter 4.3.2). The higher this option value is the more stable the application is. The GPS service is not a stable service and detects a lot of fake location changes even if they do not happen. This filter removes that kind of unnecessary movement noise.
- **Activity:** This option informs the iOS operation system about the user's activity when using the GPS service. That way the iOS operating system uses extended techniques in order to reduce the energy consumed by the device's GPS module. For example the iOS uses the device's accelerometer to detect movements. If no movements are detected the GPS modules is set to sleep mode reducing energy consumption.

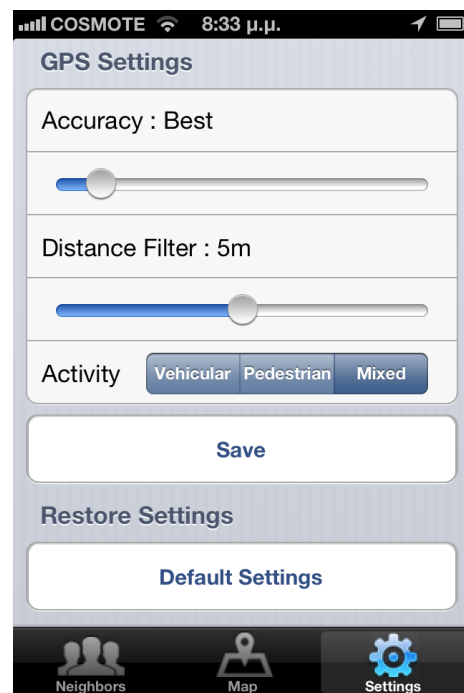


Figure 35: iOS Application GUI – Settings View – GPS & Restore Settings

5.4.5. Restore Settings

The «Restore Settings» section offers the capability of restoring the application to its factory default settings just by pushing the homonym button shown in the bottom of the Figure 35.

5.5. Station View

The Station view is where the user can enable or disable the GPS station mode. The station mode is used when the user wants to disable the GPS hardware module and define himself his location coordinates. This is very useful when the user is in a stable location and he wants to reduce the energy consumption of the hardware GPS module. Another reason for using this mode is when the GPS services are not available for the user-client application.

The user can define his location on the map (Figure 36) by tapping on it or by filling the latitude and longitude fields using the keyboard. If the GPS services are available the user can define his location by pushing the «current location» button.

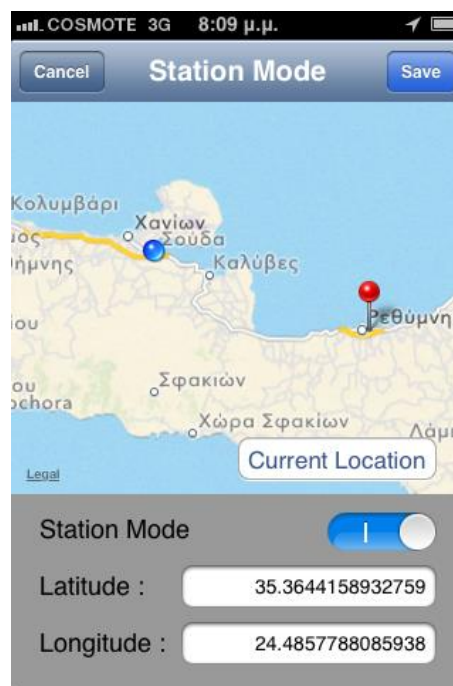


Figure 36: iOS Application GUI – Station View

5.6. Account View

The account view is where the user can either edit his account information or create a new account.

When the user is logged in the server the account view offers him the ability to change some of his private and public information like his password, e-mail, etc (Figure 37, Left). The user cannot edit his username. By pushing the «Save» button at the top right corner of the view the application send an «Edit Account» request (Chapter 1.3.9). If the server does not respond the user can push the “Back” button to cancel the request.

At the bottom of the view there is a «Sign out» button in case the user wants to unregister from the «proxiConn» project.

On the other hand, when the user’s log status is unlogged the «Account» view gives him the ability to create a new account and join the «proxiConn» project (Figure 37, Right). The only thing the user need is to use a unique username and e-mail address and validate its account using his email client application. After that the user is ready to use the application and join the «proxiConn» project.

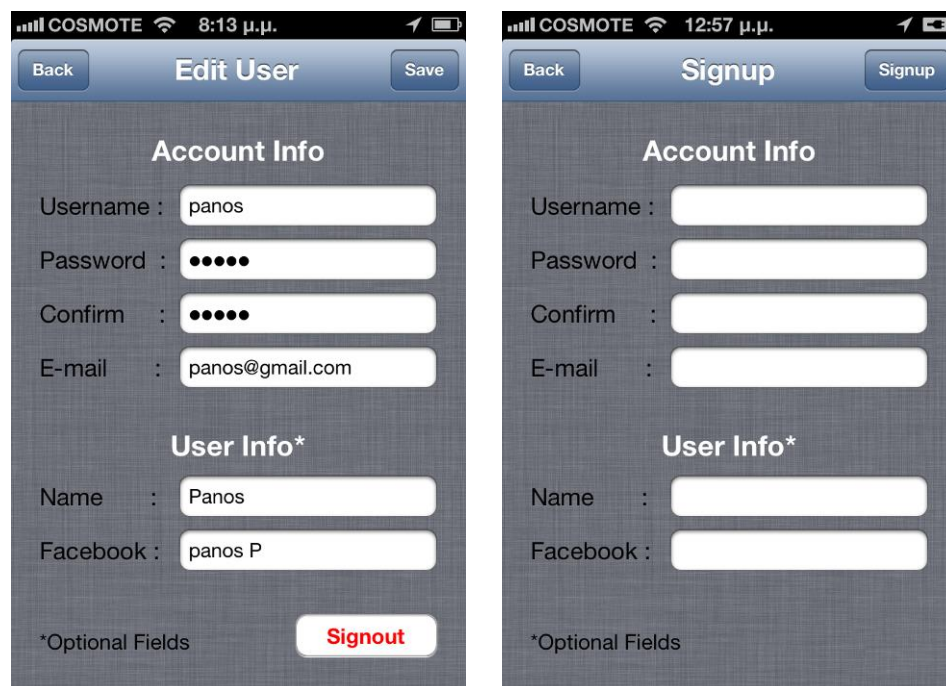


Figure 37: iOS Application GUI – Account View (Edit & Signup)

CHAPTER 6

WEB CLIENT APPLICATION

For the extension of the «proxiConn» project user's usability an auxiliary web-client application was implemented. This application was designed using the JavaScript and Ajax frameworks in order to move the processing weight at the client side instead of the server one. That way the whole application is running on the user's web browser sending requests to the server, just the same way the mobile user-client application does and described on chapter 5, using the «proxiConn» server-client communications protocol (Chapter 1.3).

6.1. Web Client Application

Any registered user of the «proxiConn» project can connect to the server using his web browser. The user can set his location coordinates just by clicking on a map and does not need to use a GPS module. The web-client application lets the user see his proximate users, update his location and update his profile information. It's a useful application for stationary users.

The web-client application is implemented using the JavaScript V3 scripting language and the Ajax frameworks as these technologies move the processing weight to the user's browser and alleviate the server. For the mapping functions of the web-client application, the Google Maps JavaScript API v3 was used.



Figure 38: Google Maps API Logo

When the user makes an HTTP connection to the server using his browser, the browser loads the «Login» view of the web-client application from the server. The user has to use his username, his password and his location coordinates to log in through this view. After a successful log in the browser loads the web-client application's «Main» view and starts the connection to the server. For every user request the applications send a clients

request to the server as the server-client communication protocol defines. All the processing of the data takes place on the user's web browser. The web-client application behaves exact as the user-client iOS application for mobile devices to the server. The Figure 39 below shows its flow diagram.

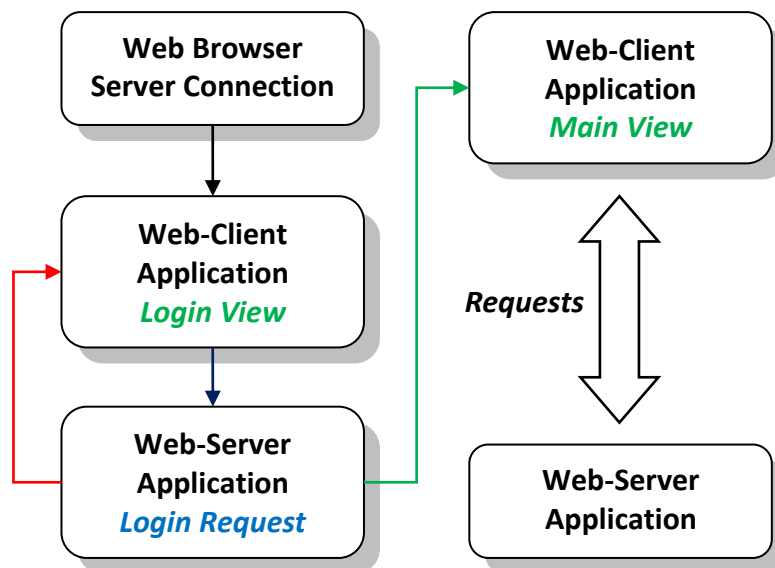


Figure 39: Web-Client Application – Web-Client Application Flow Diagram

6.1.1. Login View

When the user set his browser URL to the «proxiConn» server, the first view he comes up is the «Login» view. At this view the user has to set his username and his password and click on the map in order to auto fill his location coordinates. If the user log in successfully he is forwarded to the «Main» view of the web-client application (Chapter 6.2). By this view the user can also create a new account to the «proxiConn» project. Clicking on the «Signup» hyperlink the «Signup» view appears as shown in Figure 40 where the user has to fill his

The screenshot shows the 'Signup View' of the web-client application. It features a form with two sections: 'Required Fields' and 'Optional Fields'. The 'Required Fields' section includes input fields for 'Username', 'Password', 'Confirm', and 'E-mail'. The 'Optional Fields' section includes input fields for 'Name' and 'Facebook'. A red 'Signup' button is located at the bottom right of the form. The background of the form is a light gray, and the text is in a dark font.

Figure 40: Web-Client Application – Signup View

account information. After that he has to activate his account by answering the activation email he receives on his email box.



Figure 41: Web-Client Application – Login View

6.2. Main View

The «Main» view is actually the web-client application main graphics interface. By this view the user can handle all the applications functionalities. The «Main» view consists by three blocks. The upper left block contains the map where the user and its proximate users are shown. The bottom left block presents some of the user's information. The right block of the view is dynamic. It presents the user's neighbors list, or the user profile information or the web-client application settings, depending on the user choice on the menu bar. The menu bar stands over the blocks.



Figure 42: Web-Client Application – Main View

6.2.1. Main View – Neighbors

The neighbors list presents a list of the proximate users sorted by the closest first as shown in Figure 43 at the right block. Left to each proximate user username a compass presents the neighbor direction to the web-client application user. The user can also click on the username of each user or on any user on the map to see the user's public information. This information appear on a small block at right of the web page as also shown in Figure 43

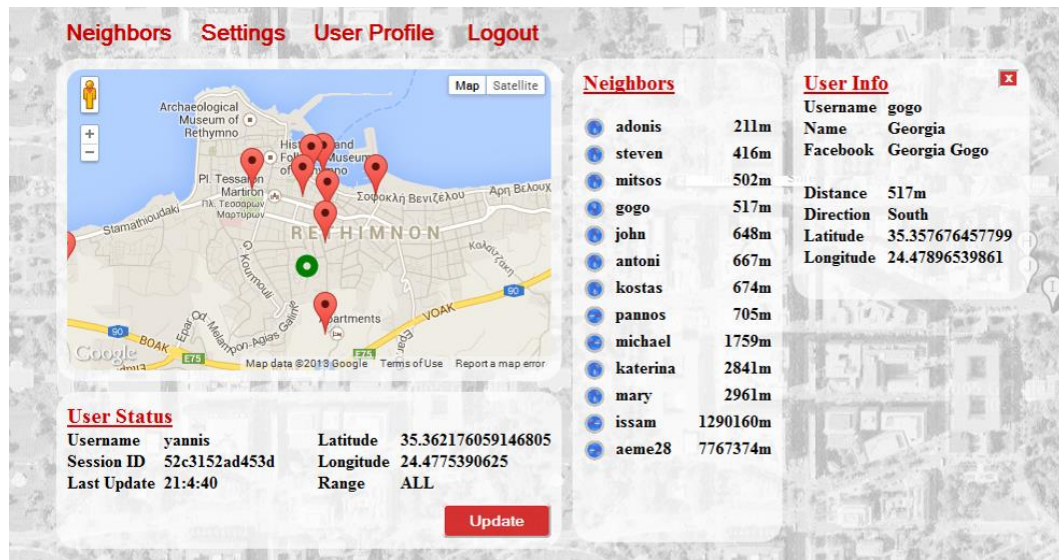


Figure 43: Web-Client Application – Neighbors & Info Blocks

6.2.2. Main View – Settings

By clicking at the «Settings» menu the user can adjust the web-client application behavior (Figure 42). As an auxiliary application there are just a few basic settings for it. The user can change his location coordinates by clicking on the map when the «Settings» menu is selected and define the radius around his position that he would like to look for proximate users. At last the user can decide if he wants to update himself his proximate users or the he would like the application to do that automatically at a certain time of period. To apply any change he has to push the «Apply» button.

6.2.3. Main View – User Profile

The profile block is where the user can just change his profile information

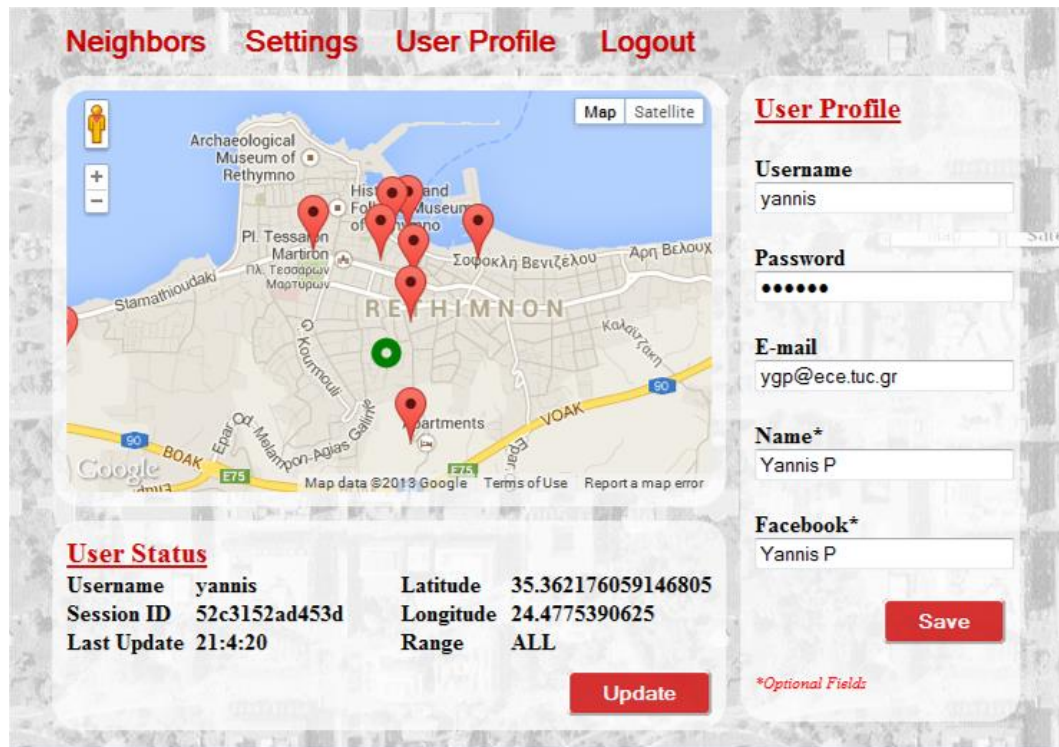


Figure 44: Web-Client Application – User Profile Block

CONCLUSIONS

It is enough years since central servers are used for several project and scopes. This is not something new either unprecedented. The «proxiConn» server is one of the thousand server used by multiple application around the world. The innovation exists in the aspect of using «smart» mobile devices that Apple Inc first design at June 2007 and today is one of the most popular personal gadgets with a variety of devices and manufactures to choose. It is less than five years since «smart» phones replaced the traditional mobile phones and even less years since they are cost feasible for the majority of the population in the western world. Low cost «smart» devices give the users the opportunity of using the internet and the GPS service almost everywhere but also the ability of designing custom applications to the developers.

«Smart» devices handle for certain a quota of the future technology. Years ago a computer was the size of a house room, later took some place in people's office, few years before a personal computers start fitting in a shoulder bag and today fits on a pocket. The «proxiConn» project focuses on that aspect of technology and computer science. It is an experimental project aiming to be the base of extended applications used for social networking and furthermore, like emergency, safety, hospitality, transport and many other kinds of applications.

The «proxiConn» project passed successfully the laboratory tests. What is needed now is getting out of the laboratory and be used in the real world.