Geolocated Recommendations

Josep Bachs Barrio

Projecte Fi de Carrera
Director/a: Xavier Amatriain Rubio
September 2010
Enginyeria Tècnica en Informàtica de Sistemes
Universitat Pompeu Fabra

Abstract

In this is the development of an iPhone application for geolocated recommendations of movies. The application not only recommends movies, but the user can also consult movie showtimes in Barcelona, the cinema information and its location in a map. User can also consult the information about movies and rate the movies.

Using the user's ratings together with ratings of a set of experts the systems makes the recommendation to the user. And all information about movies, showtimes and ratings is collected periodically from web pages.

In fact the new feature that we present in this application compared with the wide range of cinema and movies applications for iPhone or mobile platforms is the Recommender System based on Expert Collaborative Filtering that provides recommendations tailored to the user's preferences.

Index

1. Introduction

In this chapter we describe the main characteristics of the project and the application that we designed. There are explained the features that we want to have in the application and there are also explained our goals.

1.1 Objective and scope of Project

During this project we want to design and develop an iPhone application related with movie recommendation but we don't want to limit the application only to the recommendation, we want that the users could check information about the cinemas, movies and showtimes. There are several iPhone applications related with movies and cinemas that use geolocation, but we want to in the recommendation based on expert opinions.

Our goal is to make a complete application that uses all the technologies available in a mobile device like the iPhone. Then we decided to use the geolocation together with the recommendation. We also want to maintain the application updated periodically. To make these updates possible we want to design an easy client-server framework.

The data needed by the application such is the information about the cinemas, the movies and the showtimes. We decided to collect that data from the web with different crawlers installed in a server where we also store the data to make the updates possible.

As we said before the recommendation is based on opinions of experts, and in our case the recommendation is based on opinions of critics of cinema who rate movies in the web.

We also said before that the application should have information about he cinemas like are the address, the number of screens and the showtimes of the movies that are being played at them. We also want to set all the cinemas on a map to locate them easily.

In short we want to have a project that includes some technologies and that they work well together in the device without problems.

1.2 Context

The context of this project is to work with a relatively new technology such is the iPhone and its applications. One of the things we want to do is to improve similar applications to our one that are now available in the AppStore. We want to do this with a new feature in the application, which is the recommendation system.

We also think that the applications for mobile devices, ad concretely the iPhone applications, are a growing market and in the next years it will grow up very quickly, event quicker than these last years.

2. Requirements Analysis

In this chapter I describe the requirements of the project, the functional and the non-functional. These requirements are the description and specification of the software. Include both the services required for the system as the working or developing constraints.

2.1 Functional requirements

The functional requirements of this project are related to the information that the application should show, the different interactions that the user should have and the jobs that the device or server should do.

The functional requirements of this project are:

- Application should allow the user to view information about movies, cinemas and shows.
- Application should allow the user to rate movies.
- Application must recommend movies to the user. This recommendation has to be based on the user ratings.
- Application should allow the user to restrict the criteria for recommendation.
- The system should collect the info about movies, shows and ratings from the web.
- Application has to be able to update the database automatically from the server.
- Application should show the similarity between the user and all critics in the database.

These functional requirements are related to the mock-ups design, explained in the chapter 7, because the design of the interface drawings has to reflect some of the requirements.

2.2 Non-functional requirements

The non-functional requirements of the project are related to constraints such are developing, working or time ones.

The functional requirements are:

- The project should be developed in Mac.
- Application should have an easy to use interface.
- The Recommender system in the device should not take too long to make the recommendation.
- The application should not take much to update or query the database.
- The application code has to be written in Objective-C.
- The crawler has to be written in Python.
- Should use Xcode Framework to develop the application.
- The application should work without internet connection.
- The database in the device should be designed in SQLite.
- The project must be finished before September 2010

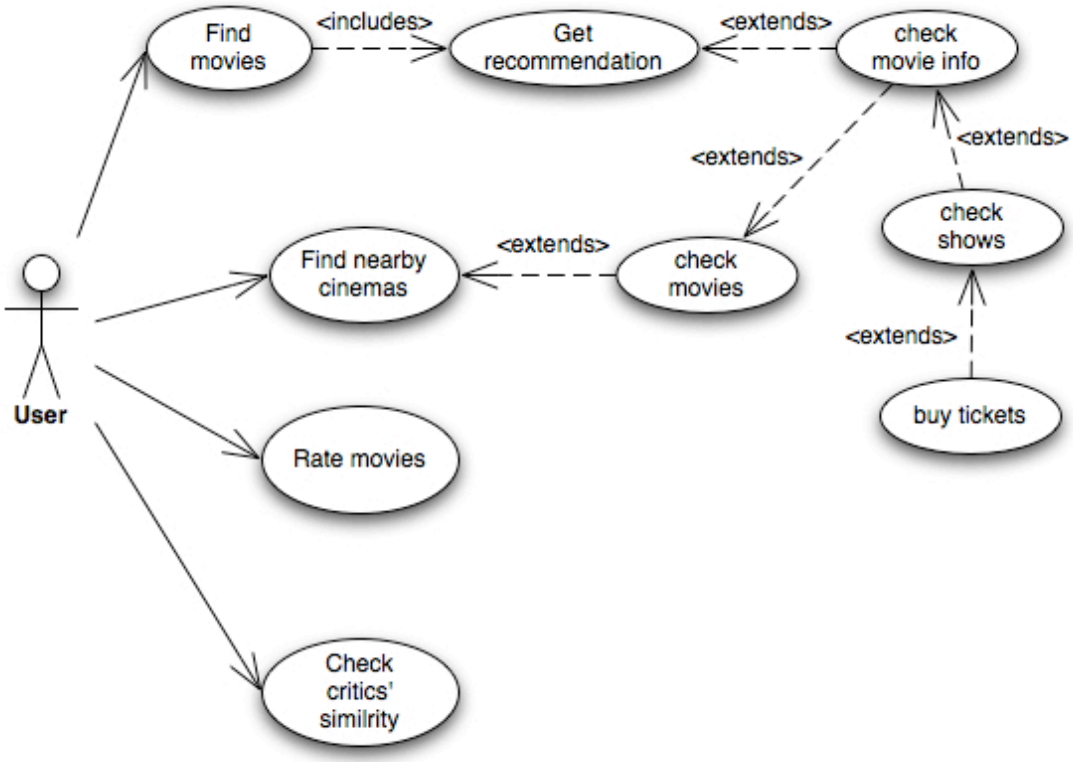

Figure 1. Use case diagram

3. Tools, Concepts and Methodologies

In this chapter I will explain the tools and concepts that are used in the project. I will describe tools such as the programming languages and concepts such as the particular paradigms or algorithms implemented in the project.

3.1 Tools

3.1.1 Xcode Tools

The Xcode toolset [1] is Apple's integrated suite of software development tool that includes compilers and applications, together with an extensive set of programming libraries and interfaces [2]. The centerpiece of these tools is the Xcode application, which provides a user interface for creating and managing software development projects [3].

To develop iPhone applications developers use the Xcode application that provides all the tools needed to design the application's user interface and write the code. Using Xcode, the user can organize and edit the source files, view documentation, build the application [4], debug the code and optimize the application's performance.

The Xcode suite includes a modified version of free software GCC, and supports C, C++, Fortran, Objective-C [5], Objective-C++, Java AppleScript, Python and Ruby source code with a variety of programming models, including but not limited to Cocoa, Carbon and Java. Third parties have added support for GNU Pascal, Free Pascal, Ada, C#, Perl, Haskell and D. The Xcode suite uses the GNU Debugger as the back-end for its debugger.

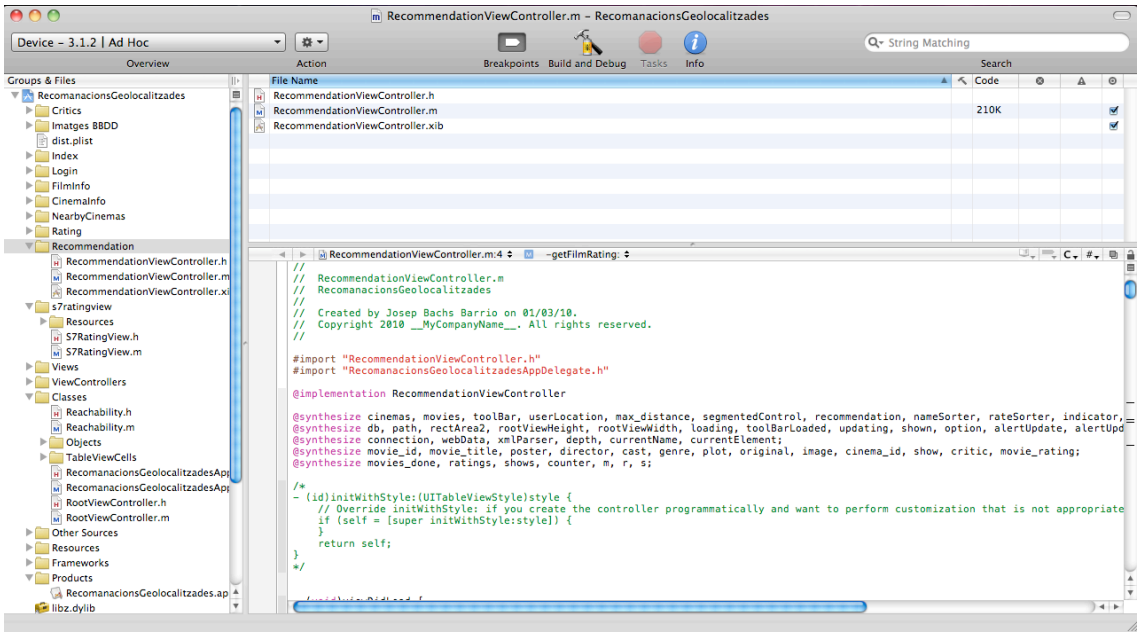

Figure 2. Screenshot of the interface of Xcode

3.1.2 iPhone SDK

The iPhone SDK is a software development kit developed by Apple Inc. [6], targeted at third-party developers to develop native applications for iPhone OS [7], released in February 2008. The SDK is available for free public download and an Intel Mac running Mac OS X Leopard or later is required by the iPhone SDK. Other operating systems like Microsoft Windows or Linux and older versions of Mac OS X, are not supported.

As the iPhone OS uses a variant of the same XNU kernel that is found in Mac OS X, the toolchain used for developing on the iPhone OS is also based on Xcode.

The SDK is broken down into the following sets:

The Cocoa Touch which includes multi-touch events [8] and controls, accelerometer, which is 3D and, can be used in applications, view hierarchy, localization, alerts, Web view, people picker, image picker and camera support.

The Media functionality that includes OpenAL, audio mixing and recording, video playback, image file formats like JPG, PNG or TIFF PDF, Quartz, Core Animation, and OpenGL ES.

The Core Services that is more lower-level oriented and includes Collections, Address book, Networking, file access [9], SQLite, Core Location, Net services, threading [10], preferences and URL utilities [11].

The Core OS that is comprised of OS X Kernel, Lib System, BSD TCP/IP, Sockets, Security, Power Management, KeyChain, Certificates, File System and Bonjour.

Along with the Xcode toolchain, the SDK contains the iPhone Simulator, a program used to emulate the look and feel of the iPhone on the developer's desktop, and the Interface Builder, a program to design the interface of the applications.

The SDK includes a comprehensive suite of tools for performance that allows developers to run applications on the iPhone and look at CPU, graphics, memory, etc. to test performance [12]. Live recording is available, enabling developers to compare interactions between different aspects of the program to assist in optimizing code.

The SDK enables the user to develop software that can be deployed on specified versions of iPhone OS. This technology enables the applications to be compatible with previous versions of the operating system.

3.1.3 Interface builder

The Interface Builder is an application used to design every aspect of the iPhone or Mac application's graphical user interface [13]. It features a simple drag and drop user interface with a library of controls, where the developer assembles windows and menus by dragging preconfigured objects into an Interface Builder document. Those objects can be repositioned and their attributes can be changed as needed to achieve the desired look for the interface. For some application types, the developer can create explicit relationships between those objects, which ultimately result in the creation of links between objects at runtime.

Interface Builder supports Model-view-controller to connect the view layer to the control layer.

Interface Builder provides several windows to allow the developer to display and modify the objects in the user interface. And the user interface objects contain items like text fields, data tables, sliders and pop-up menus. Interface Builder's palettes are completely extensible, meaning any developer can develop new objects and add palettes to Interface Builder.

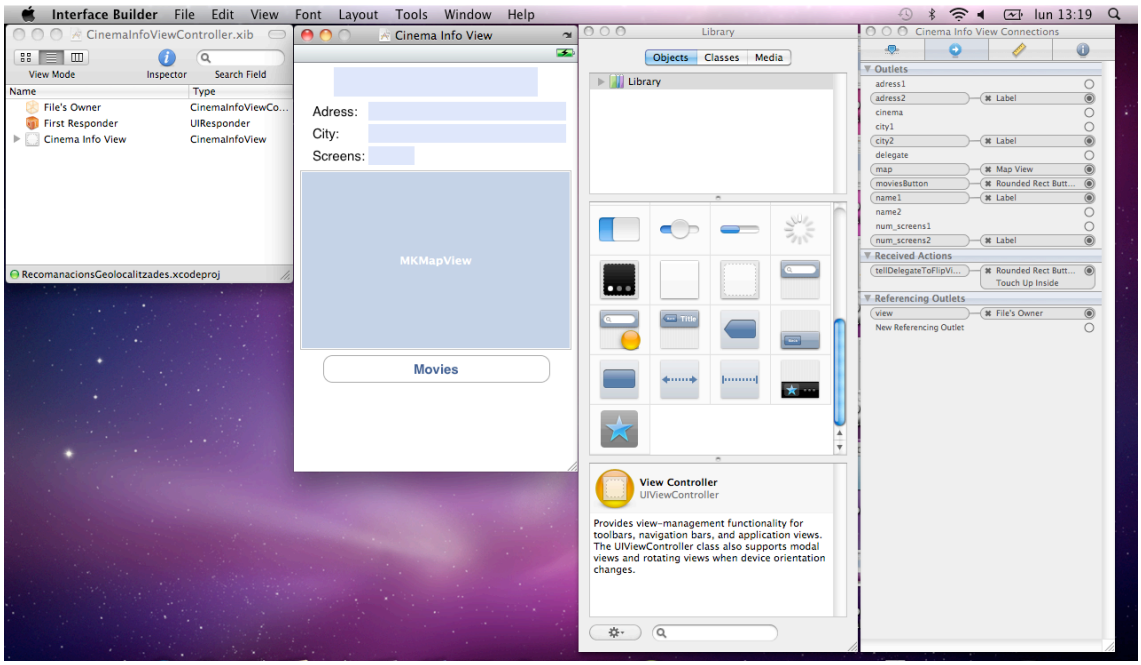

Figure 3. Screenshot of the interface of Interface Builder

3.1.4 iPhone Simulator

With the Xcode tool, the iPhone SDK contains an emulator, the iPhone Simulator.

The iPhone simulation environment allows developers to build and run iPhone applications on computers. The simulator environment is used to find and fix problems in the applications during the design an early testing, learn about Xcode development experience and the iPhone development environment before becoming a member of the iPhone Developer Program. Also I used it to lay out and test the application's user interface and measure the application's memory usage before carrying out detailed performance analysis on iPhone OS-based devices.

The iPhone simulator application presents the iPhone user interface in a window on the computer. The application provides several ways of interacting with it using the keyboard and mouse to simulate taps and device rotation, among other gestures.

The iPhone simulator can simulate two device families, iPhone and, recently, iPad, and can simulate more than one iPhone OS release.

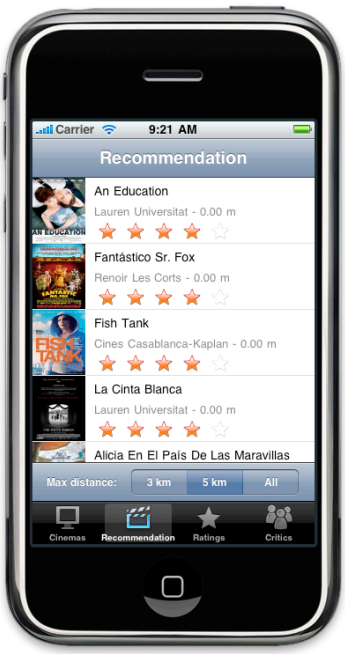

Figure 4. Screenshot of iPhone Simulator

3.1.5 Objective-C

Objective-C is a language focused on simplicity and the elegance of Object-Oriented design. This language brings many object oriented principles, but with a minimal amount of syntax.

The Objective-C language is a simple computer language designed to enable sophisticated object-oriented programming. Objective-C is defined as a small set of extensions to the standard ANSI C language. Its additions to C are mostly based on Smalltalk, one of the first object-oriented programming languages. Objective-C is designed to give C full object-oriented programming capabilities, and to do so in a simple and straightforward way.

The Objective-C language was chosen for the Cocoa frameworks for a variety of reasons. First and foremost, it's an object-oriented language. The kind of functionality that's packaged in the Cocoa frameworks can only be delivered trough object-oriented techniques. Second, because Objective-C is an extension of standard ANSI C, existing C programs can be adapted to use the software frameworks without losing any of the work that went into their original development. Since Objective-C incorporates C, all the benefits of C are got when working within Objective-C. There exists the option to choose when to do something in an object-oriented way and when to stick to procedural programming techniques.

Moreover, Objective-C is a simple language. Its syntax is small, unambiguous and easy to learn. Object-oriented programming, with its self-conscious terminology and emphasis on abstract design, often presents a steep learning curve to new recruits. A language like Objective-C can make becoming a proficient object-oriented programmer much less difficult.

Compared to other object oriented languages based on C, Objective-C is very dynamic. The compiler preserves a great deal of information about the objects themselves for use at runtime. Decisions that otherwise might be made at compile time can be postponed until the program is running.

3.1.6 SQLite

SQLite is a public domain database compact library that offers file and memory based relational database functionality from a single C library. SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A

complete SQL database with multiple tables, indices, triggers and views, is contained in a single disk file.

I chose SQLite to work with the database because the iPhone hat the SQLite library included and ready to roll and Apple expects the SQLite library to form the backbone of most applications' data storage requirements. It also has been chosen because it is extremely handy for quick and easy databases. And because an SQLite database requires little or no administration, SQLite is a good choice for devices or services that must work unattended and without human support. SQLite is a good fit for use in cell phones.

3.1.7 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics whose design philosophy emphasizes code readability. Python aims to "combine remarkable power with very clear syntax", and its standard library are large and comprehensive. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available without charge for all major platforms, and can be freely distributed.

Python has been used in the project to program different crawlers, that get the info needed from the web and insert it into the database. In that crawlers there are two python packages that has been used to get the info from the web, which are:

The first one is Beautiful Soup python package. Beautiful Soup is an HTML/XML parser for Python that can turn even invalid markup into a parse tree. It provides simple, idiomatic ways of navigating, searching, and modifying the parse tree. And Beautiful Soup automatically converts incoming documents to Unicode and outgoing documents to UTF-8, so thinking about encodings is not needed, unless the document doesn't specify an encoding and Beautiful Soup can't autodetect one.

For more information about Beautiful Soup visit:

http://www.crummy.com/software/BeautifulSoup/

The other one is IMDbPY package, which is a package to retrieve and manage the data of the IMDb movie database about movies, people, characters and companies. IMDbPY is platform-independent and is written in pre Python and few C lines. It is free software (open source) and it is released under the terms of the GPL license.

For more information about IMDbPY visit: http://imdbpy.sourceforge.net/

3.2 Concepts

3.2.1 Model-View-Controller (MVC)

Model-View-Controller (MVC) is software architecture, currently considered as an architectural pattern used in software engineering. The MVC design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other. Each of the three types of object is separated from other by abstract boundaries and communicates with objects of the other types across those boundaries. The collection of objects of a certain MVC type in an application is sometimes referred to as a layer.

The benefits of adopting this pattern are numerous. Many objects in the applications that use MVC tend to be more reusable, and their interfaces tend to be better defined. Applications having an MVC design are more easily extensible than other applications. MVC is central to a good design for a Cocoa application and many Cocoa technologies and architectures are based on MVC and require that objects play one of the MVC roles.

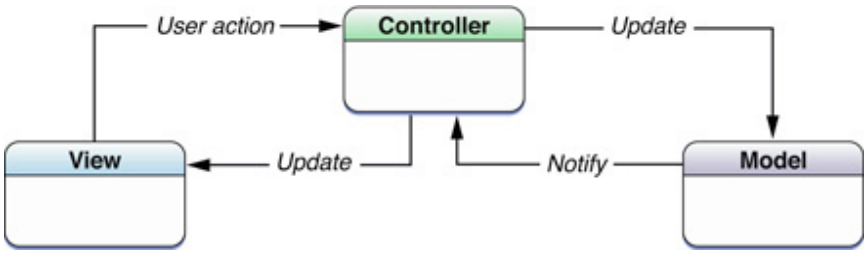

Figure 5. MVC process

3.2.2 Bubble sort

Bubble sort is a simple sorting algorithm. It works repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. Because it only uses comparisons to operate on elements, it is a comparison sort.

Although Bubble sort has one of the worst-case and average complexity, both $O(n^2)$, where n is the number of items being sorted, I decided to use this algorithm because I reused the code of a function that I programmed before.

3.2.3 Expert Collaborative Filtering

Expert Collaborative Filtering is an approach to recommender systems in which recommendations for users are derived from ratings coming from domain experts rather than other users. The Expert Collaborative Filtering replaces the peer user ratings with expert ratings and implements a collaborative filtering solution using only the experts' opinions.

3.2.4 REST

Representational State Transfer (REST) is style software architecture for distributed hypermedia systems such as the Wold Wide Web, emphasizing scalability of component interaction, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems. Systems following the REST architecture are referred to RESTful, and they usually have the following aspects: First we have the base URI for the web service; The next aspect is the MIME type of the data supported by the Web service; The last aspect is the set of operations supported by the web service using HTTP methods (GET, POST, PUT, or DELETE).

3.3 Methodologies

In this section I will explain the agile method followed during the project development. I specifically used Scrum, which is an iterative, incremental framework/methodology for project management and agile software development. Scrum is a methodology aimed primarily at project management. This management is not based on tracking a pre-set plan but continues to adapt to circumstances such as the project evolves. In my case I used an adapted version of scrum that I will explain in the next sub-section.

3.3.1 Scrum

During the project we had to use an adapted version of Scrum because we were only two people.

First of all I will talk about the roles of Scrum, which has three basic roles: the Product Owner, the Team and the Scrum Master. In this project one person represented the Product Owner and the Team Manager roles, this person was the tutor of this project. Regarding to the team role, it was represented by the tutor and me.

In Scrum we build the final product of iterative and incremental manner. The iterations in this framework are known as “Sprints”, which are development cycles with duration of one week, in our particular case, in which the team develops the selected customer’s requirements.

The goal of each sprint and activities are planned in the beginning of the “Sprint planning” meeting. After each iteration the team shows what has built in the “Sprint review” meeting, and analyzes the performance of the own team and decide how to improve it in the “Sprint Retrospective”. And the sprint’s progress is monitored using a “Sprint backlog”. In our case a single weekly meeting includes the aforementioned three meetings.

Now I will talk about the main documents of Scrum that we had during the project. The first document is the Product Backlog, which is a high-level document for the entire project. The Product Backlog is the inventory of features, improvements, technology, and error correction to be added into the project throughout the various iterations. This document represents everything that customers, users, and in general the stakeholders expect about the product. All that would be work to do by the team must be reflected in this backlog. The Product Backlog is never a finished document but that continually grows and evolves. It usually starts from a brainstorming meeting in sprint 0. The different tasks in the Product Backlog are sorted by priority and this document is property of the Product Owner.

The next document is the Sprint backlog, which contains information about how the team is going to implement the features for the upcoming sprint. This document is set at the beginning of the sprint and it is not modified during the sprint. This document is property of the Team.

The last document we had is the Impediments Backlog that is a document that contains impediments. Anything about the project or team that prevents getting the productivity, speed or quality required is considered an impediment. The Scrum Master is responsible to remove these impediments.

4. Similar Applications

In this chapter, I will talk about some similar applications available on the iTunes App Store. It must be said that these applications are all free.

4.1 OneTap Movies

OneTap Movies is a mobile application powered by Avantar LLC. It recognizes where the user is and shows him results of several cinemas beginning with the one nearest to him.

The features of this application are that mobile device locates where the user is or the user can set up a location and the application shows a list of all the cinemas near to him. There is the option to set a favourite cinema and its results would be showed first. In that list are shown titles, times, critic ratings (stars) and movie ratings (PG, R, etc.). There are two more features such are ready-to-play movie trailers, and an interactive map with driving directions to each cinema.

The application has different options to show the info such as the mentioned cinema list, and different ways to sort the movies like by popularity, by rating or by newest. And the last option is the upcoming movies that show the movies that will come and its info.

The info about the movies includes the title, the poster, the length, the rating, the cast, the plot, the genre and the release date. And you have a link to the movie info in IMDb.

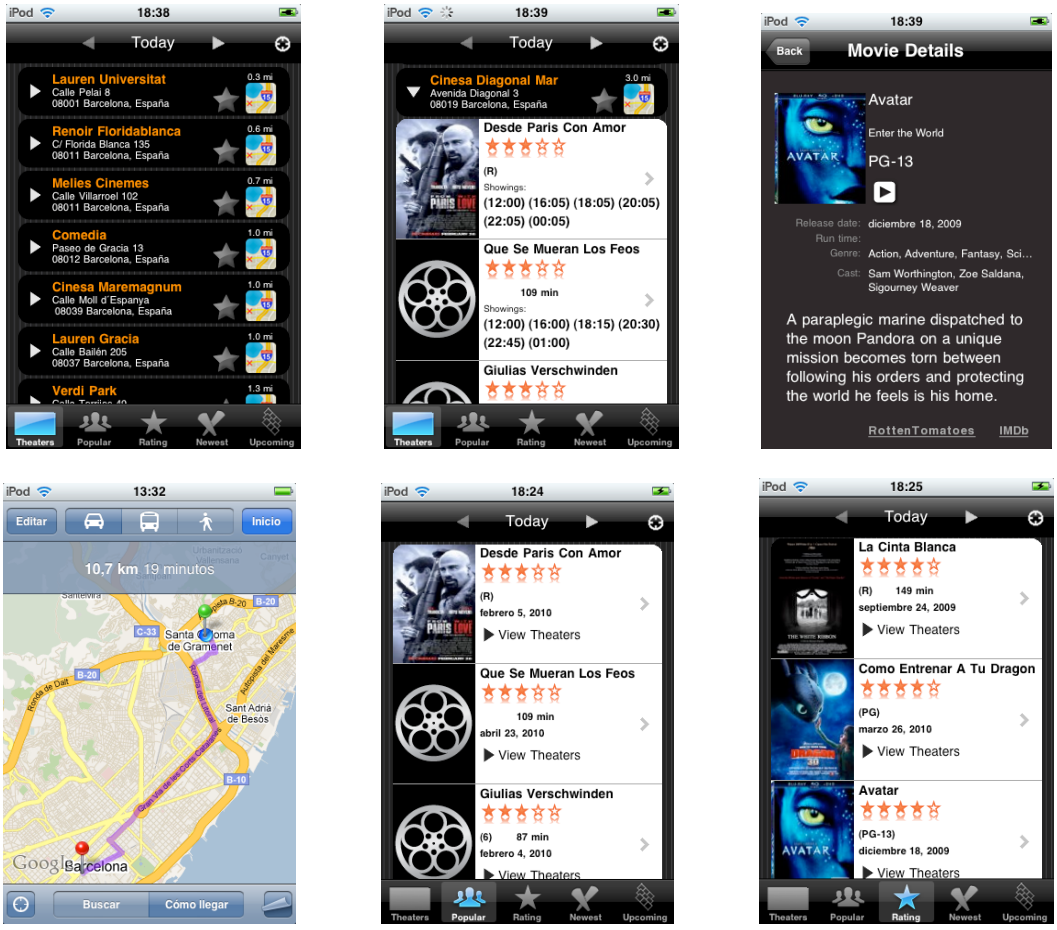

Figure 6. OneTap Movies screenshots

4.2 Showtimes

Showtime is another application powered by Avantar LLC. This application is very similar to OneTap Movies application, the only difference is the background colour.

4.3 Now Playing

Now Playing is an application done by Cyrus Najmabadi. The application uses the device position and the user settings to show the list of the movies that are in the cinemas sorted by rating. For each movie the user can see its info like poster, length, genre, cast and plot. The user can also check the cinemas where the movie is shown and its show times, and can check reviews from critics or play the movie trailer if it is available.

The user can search by cinemas and they are sorted by distance. The cinema info that the user can check is the address, a map with the cinema's position, the telephone and a list of the movies that are playing with their show times.

Another feature is the coming soon movies information, but this info is only updated for the United States.

The user can set up some options like are his position, the maximum distance to search cinemas, the search date, the site to get ratings and reviews, where the user can choose between RottenTomatoes, Metacritic and Google.

The last feature to comment is that the application keeps movie and cinema information in the device, even without a network connection.

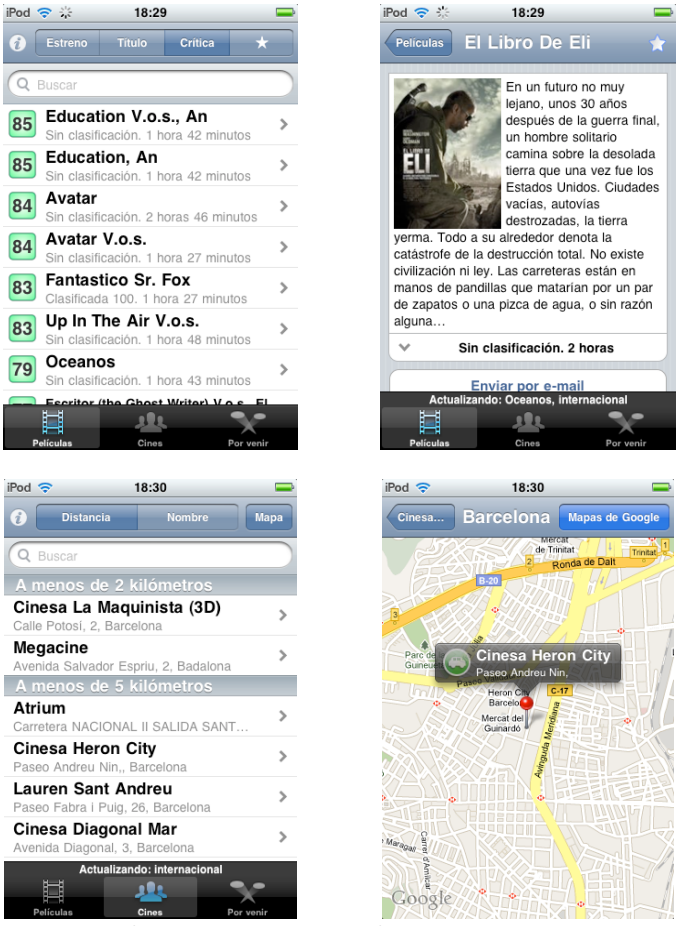

Figure 7. Now Playing screenshots

4.4 En tu Cine

"En tu Cine" enables all iPhone users in Spain to know which movies are shown and their show times in the movie theatres. It has a database that is constantly updated and integrates GPS functionality of the iPhone, through which the application will tell the user the nearest movie

theatres, even when you are out of your place of residence. You can also do searches by title, people, filtering films according to your criteria and enjoy the original film posters.

Figure 8. "En tu Cine" screenshots

4.5 Movie Finder

Movie Finder integrates geo-location data, shows and film ratings from Filmaffinity to help you decide what movie to see. Movie Finder offers films ordered by average score of the Filmaffinity users votes, and secondly by geographical proximity. The user can set another position if he wants. The user has to indicate a zip code or city cinema and Movie Finder calculate the best options for that position.

Movie Finder is integrated with Maps and Mail. The user can see the route chosen to get to the cinema or e-mail the selected movie information. The user can also explore the details of each movie by clicking its name.

Movie Finder is designed specially for Spain and the United States. In other countries it may not show all the movies.

Figure 9. Movie Finder screenshots

4.6 Other

There are other similar applications in the App Store but that are payment applications. Some of them are: "Cine ESPANA", "iLocate – Movie theatres", "Hoy Cinema", "Cartelera" and "NearCinema". Such are payment applications I could not check them and compare with my application. After some checks I could see that some of these applications have a free "lite" version, which has fewer features than the payment versions.

There is another group of applications containing applications like Flixster, “Movies Now” and Fandango. This group contains applications that only work in the United Stated and maybe in other countries, but they do not work in Spain. In this way I couldn’t cheek them too.

As we will see later on, our application has some similarities with the previous applications but also includes some novel functionality. In particular, and as some of the existing applications, it includes geolocation information, up-to-date cinema show times and critics’ information. It is also available online since it is not browser-based.

But our application has an important difference since it offers personalized recommendations for the user based on her taste. It does so by maintaining a local user profile extracted from the user rating the movies and then applying the Expert Collaborative Filtering algorithm explained in chapter 8.

5. Architecture

In this chapter, I describe the architecture of my project. The goal of the project is to implement a client-side expert-based collaborative filtering iPhone application. Figure 10 shows the components of the system. The system can be divided into different parts. First we have the data component, which is explained in the chapter 6 of this project and refers to the data sources in the web, the crawlers that fetch it and the databases where it is stored. Another part is the device's interface, explained in chapter 7 together with its design process. And the last part is the Expert Collaborative Filtering component that is explained in chapter 8.

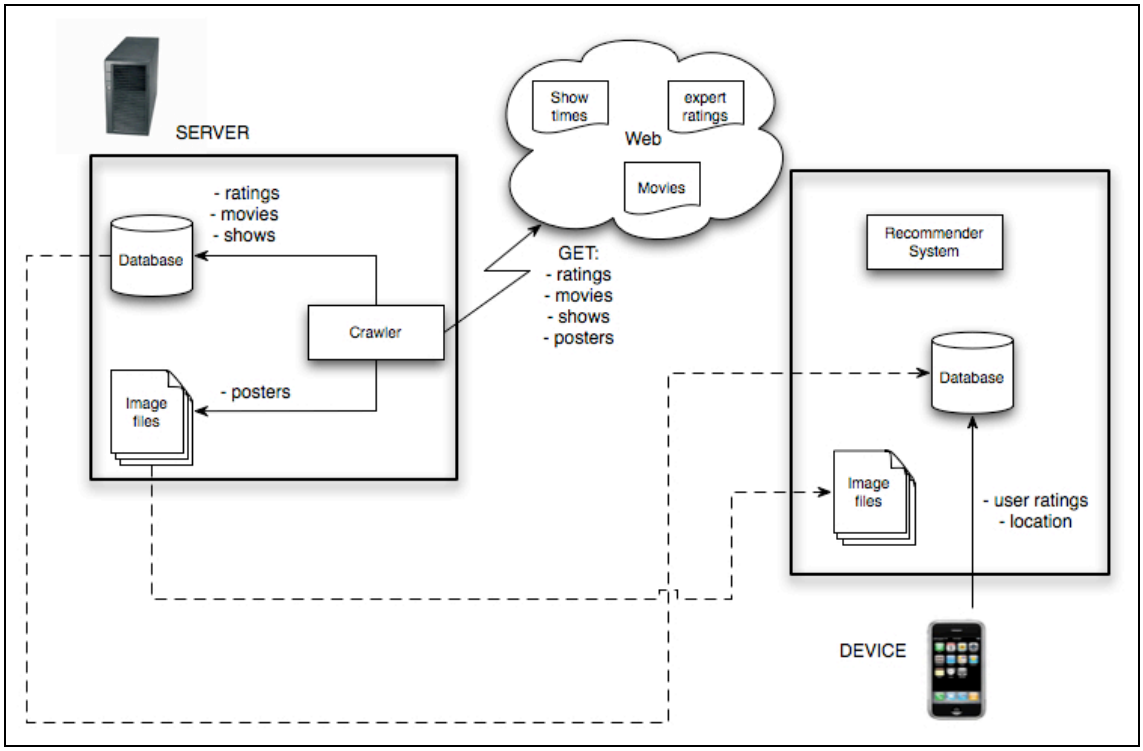

Figure 10. Architecture diagram

The server collects the expert ratings, the new movies and the show times from the web and stores them into the database. It also saves the posters of movies. Then the device can download the data from the server. The device updates the database downloading the new data from the database in the server when it detects that the database is out of date. When the application detects that the database needs to be updated, it asks the user to do it. Moreover, the image files of the posters are downloaded and stored into the device. The recommendation module resides in the device as well as the user ratings, which are stored only in the device and not in the server.

Figure 11 shows the different technologies used in the project, and in particular the technologies used in the server and in the device.

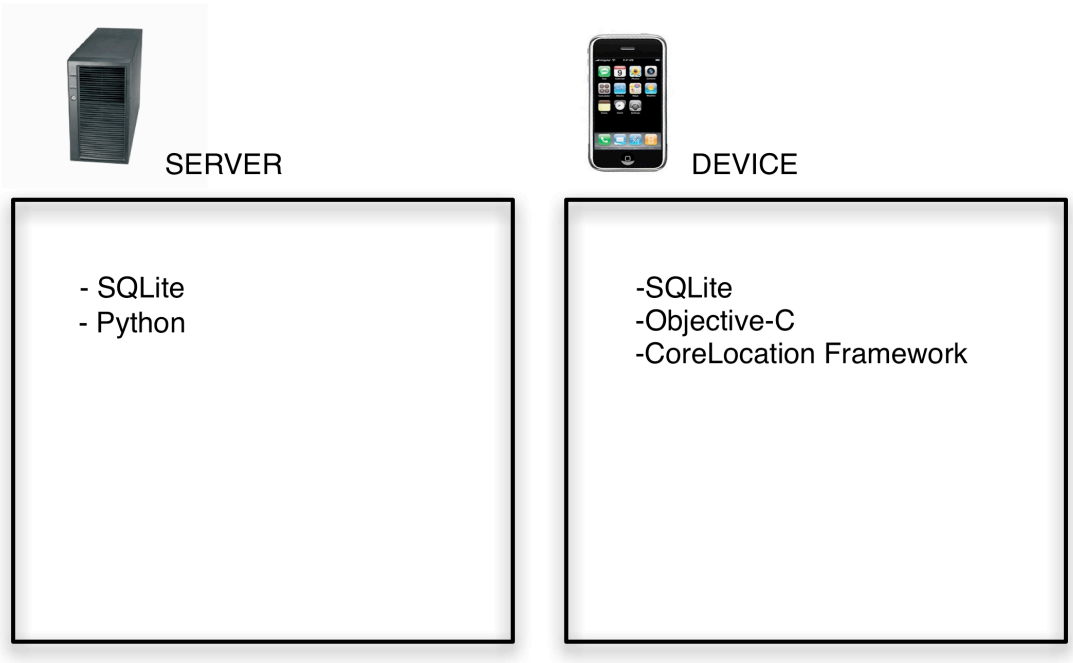

Figure 11. Technologies diagram

6. Data source

In this chapter I will explain the structure of the database used and the sources that are used to collect the data. Also, I will explain which data is needed by the application and how this data is collected and how it is fetched from the web and managed to insert it into the database.

6.1 Data sources

In this section I will explain the different data that the application needs.

The system needs static data, such as the list of cinemas and the list of critics. The list of cinemas is static because in this project I just considered the cinemas in Barcelona because new cinemas are not normally open. The list of critics is static because I need a fixed set of critics to make the recommendations with criteria.

Other data that we need are the movies, not just the current ones in cinemas, but older movies too to get enough ratings to allow the application to make recommendations. We also need the critic's ratings for the recommender system.

The last data that we need are the shows times of the movies in each cinema to show them to the user in the device.

All the data needed is fetched through the different crawlers explained in next section.

In Figure 12 we can see the data collection workflow diagram. This diagram shows how the data is collected and the process that follow from the web pages until the different data are saved in the database.

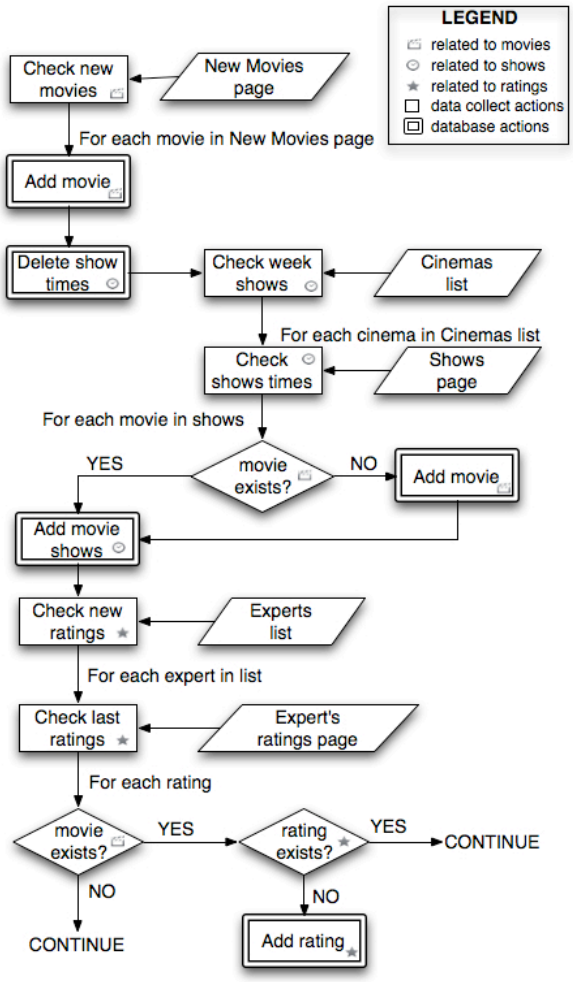

Figure 12. Data collection workflow

6.2 The crawlers

In this section I will explain the different crawlers that I implemented to collect the data from the web. There are crawlers that are running in the server once a week and other crawlers only once, and all the crawlers store the new data in the database and download the image files.

All the crawlers are written in Python and they use the Beautiful Soup package, explained in the chapter 3, to collect the data from the web. Another thing to comment about the crawlers is the encoding of the generated files by the crawlers. To encode all these text files we used the utf-8 encoding, which is the encoding used in the web pages where we collect the info.

Now in the next sub-sections are explained the different crawlers.

6.2.1 Cinemas crawler

The cinemas crawler is one of the crawlers that runs only once and it gets the list of the cinema in Barcelona together with a link to the webpage of the cinema from Yahoo Movies[1]. When the crawler collects the information it then saves it into a text file.

6.2.2 Experts crawler

The experts crawler is the other one that only runs once. This crawler basically gets all critics of Rotten Tomatoes[2] and stores the name of the critic, the link to the critic's page in Rotten Tomatoes, and the number of ratings made by the critic. Once the crawler has the info for all the critics in Rotten Tomatoes, then it sorts critics by number of ratings and it gets the first hundred. When the crawler has these hundred critics then it saves the name and link to the critic's page in Rotten Tomatoes in a text file.

6.2.3 New movies crawler

The "New movies" crawler gets the premiere movies of the week from the webpage Estrenos de Cine[3], which has a list containing the title of these movies. This crawler begins fetching the list of new movies and stores them in a text file. Once the crawler has the movies list, it begins to collect the movie info from IMDb[4] using the IMDb-py python package, that searches into the IMDb database. When the crawler has all the needed information stores it into the server database.

This crawler is installed as a cron job in the server that runs once a week every Wednesday.

[1] Yahoo Movies: http://es.movies.yahoo.com

[2] Rotten Tomatoes: www.rottentomatoes.com

[3] Estrenos de Cine: www.estrenosdecine.net

[4] The Internet Movie Database: www.imdb.com

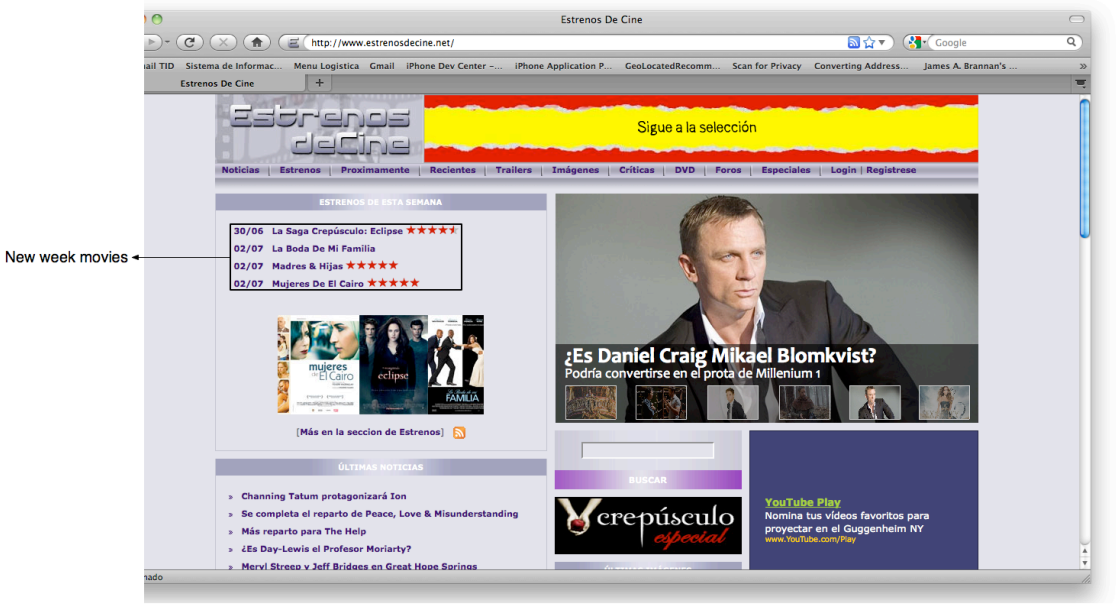

Figure 13. Screenshot of Estrenos de Cine

6.2.4 Week shows crawler

The "Week shows" crawler gets the show times of every cinema in the database. The crawler collects this info from Yahoo Movies. This crawler begins reading the text file made by the cinemas crawler and for each cinema the crawler gets the name and the Yahoo Movies link. Once it has the cinema and the link, it modifies that link to get the shows for each weekday because Yahoo Movies has a page for each cinema and each weekday containing the movies and the time of the shows. Then the crawler collects the all movies available for each cinema and then it checks if the movies are in the database, if not it gets the movie info with the IMDb-py module and stores into the database. If the movie is in the database or once it is inserted, the crawler collects the show times and stores them into a text file. When the crawler finishes collecting the show times then delete the content of tables in the database containing the outdated show times and insert the new show times.

This crawler is installed as a cron job in the server that runs once a week too. It runs every Friday because on Fridays it is when the show times are usually changed in Spain.

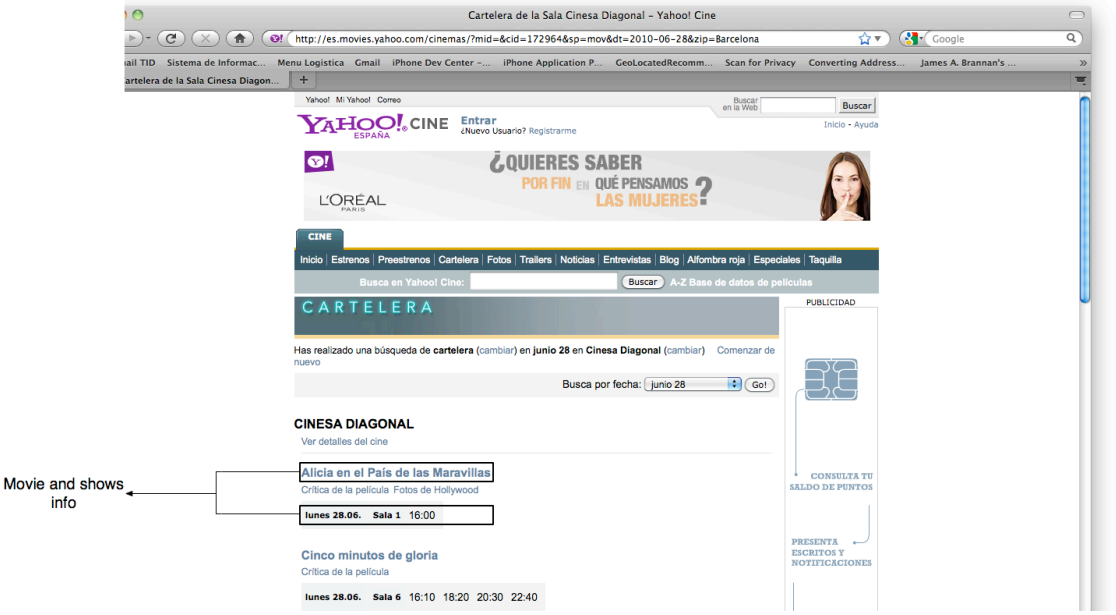

Figure 14. Screenshot of Yahoo Movies

6.2.5 Expert's ratings crawler

The "Expert's ratings" crawlers basically gets all movie ratings for each critic in the critics' list created by the Experts crawler. For each critic in the list it gets its link to the Rotten Tomatoes page. Once in that page it gets the last fifty ratings and stores the info in a text file. When the crawler has the ratings of the hundred critics then it begins to check all of them. It first checks if the movie is in the database, if not it continues with the next rating. If the movie exists then it checks if it is a new or valid rating and then inserts or updates it in the database.

This is another crawler that is installed as cron job in the server and runs once a week. This crawler runs every Thursday and it takes several hours to complete.

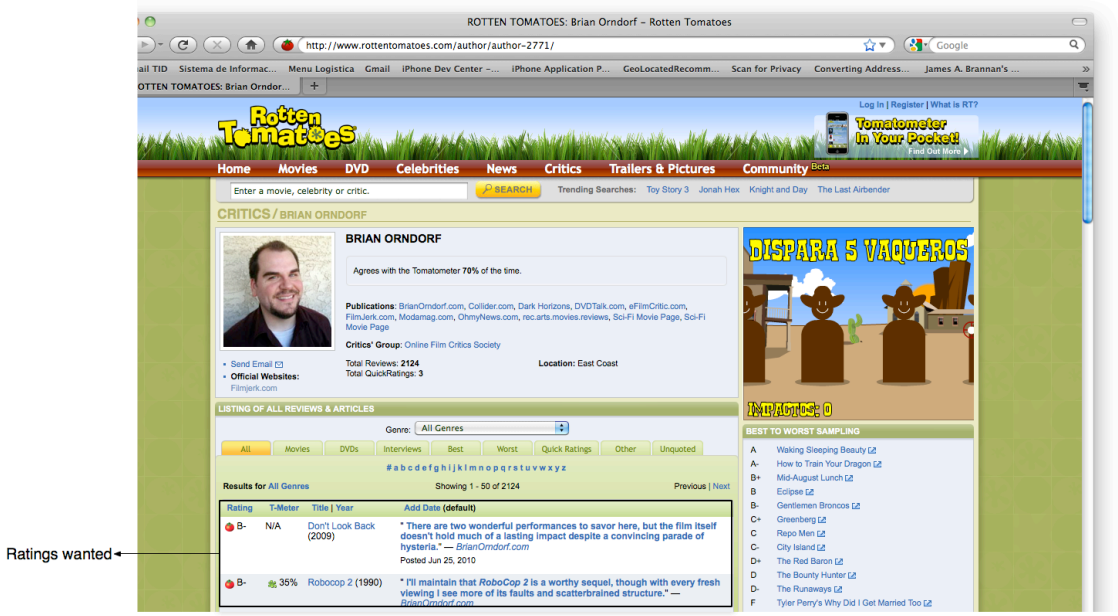

Figure 15. Screenshot of Rotten Tomatoes critic's ratings page

6.3 The database

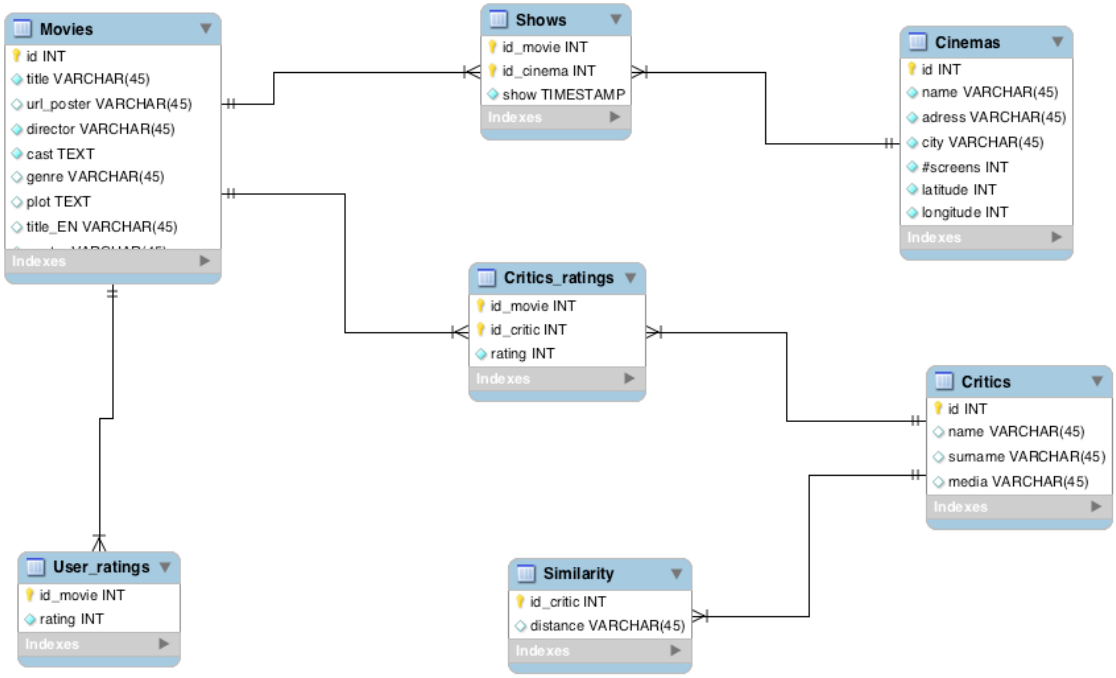

Figure 16. Database diagram

The database on the server has five tables and the database on the device has seven tables. The database on the device has two extra tables containing data that is only related to the user. The other five tables in the device database are the same as the tables in the server. The device has extra tables because doing it this way we can make the recommendation in the client side and the user feedback does not need to leave the device, so user privacy is fully preserved.

The first one of the common tables is the Movies table which contains the information about the movies, including a movie id, the title, the poster URL to download it, the director's name, the cast info limited to seven people. It also contains the genres of the movie, the plot or summary, the English title and the saved filename of the poster.

Anther table is the Cinemas table that contains the cinemas information. Each cinema has an id, the name, the address and the city where it is situated and the number of screens that the cinema has. It also contains the latitude and longitude coordinates to locate it in the map.

The next table is the Critics one, which has the info about to critics. This table contains an id for each critic, the critic name and surname, and the media she works for.

Another table is Shows table that contains all the info about the movie shows in all the cinemas. The table contains the movie id, which is a foreign key to the Movies table, it also has the cinema id, which is a foreign key to the Cinemas table, and it contains the show that is the timestamp of the show date.

The last common table is the Critics' ratings table that contains all the critics' ratings of the movies. The table has the movie id that is a foreign key to the Movies table, and it contains the critic id, which is a foreign key to the Critics table, and it also contains the rating value.

The first table that is only available in the device is the User ratings, which contains the ratings of the user. The table has the movie id that is a foreign key to the Movies table, and the value of the rating. One of the reasons because this table is only available in the device is to fully preserve the user privacy. This is because the user ratings do not need to leave the device to compute the recommendations.

The last table is the Similarity table that contains the similarity of the user with each one of the critics. The table has the critic id, which is a foreign key to the Critics table, and the similarity value.

6.4 Data updates

In this section is explained how the data is updated in he device and how the device communicates with the server.

We define an API for the client-server communication using the REST style. Using our RESTful API, which is and adapted version of the one explained in *"Towards Fully Distributed and Privacy-preserving Recommendations via Expert Collaborative Filtering and RESTful Linked Data"*[14], the device can easily get resources from the server with unique URIs. The resources are the information about the new movies for each week, about the expert ratings for each week too, and about the showtimes for each cinema. The URI for the new movies is <http://SERVER-ADDR/geolocatedRecommendations/movies/week/[WEEK]> and that URI returns a list of the new movies in the week number that is referred by [WEEK] in XML format. The URI for the ratings is <http://SERVER-ADDR/geolocatedRecommendations/ratings/week/[WEEK]> and it returns the new ratings made by the experts in the week number referred by [WEEK] in XML format. And the one for the showtimes is <http://SERVER-ADDR/geolocatedRecommendations/showtimes> and it returns the shows times for the current week in XML format. The server also contains a text file name *week.txt* that contains the current week number.

The XML files are generated from the DB dump of each week and table, once we have two week dumps then we can make de difference between these two weeks and we get the new info to be updated in the device.

At the client side every time that the applications starts then it check the week number stored in the device in a property list [15] and the week number in the server. If the week numbers are not the same then the application requests the update to the server. When the device get the XML file then it parses the document [16] and it updates the database. Once the update is finished the application continues working.

There are some disadvantages working with the RESTful architecture and the XML files instead of working directly with text files. The most important one is the time that the application lasts to parse the document to update the database.

Table comparing times

	TXT	XML/PARSER
MOVIES	1 – 3 s	1,5 – 3 s
SHOWS	20 – 40 s	52 -60 s
RATINGS	2,5 – 7 s	11 – 14 s

As we can see in the table above the solution with the text files is more optimal in terms of time than the solution with the XML files. But we prefer to keep the RESTful architecture because this solution is more flexible.

6.5 Other

In the application there is a special object that it could be in different states. This object is the movie and Figure 17 shows these states.

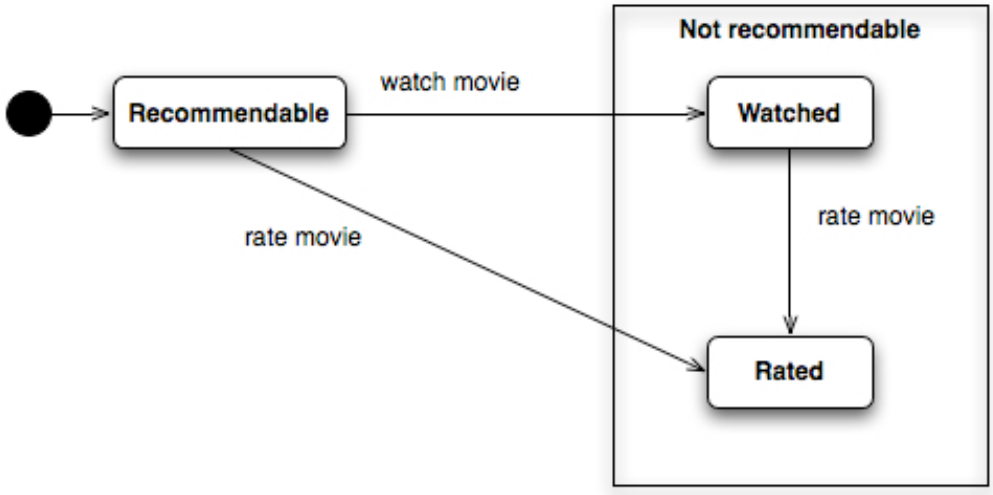

Figure 17. State diagram of object movie

As we can see in the state diagram when a movie is inserted in the application, it goes to the recommendable state, and when the user watches it or rates it, then the movie goes to the not recommendable state that is a super state, which includes the watched state and the rated state. If the movie is watched then goes to the watched state and if the movie is rated then the movie goes to the rated state. When a movie goes to the watched state then it can be also rated and then it goes to the rated state.

7. Interface

In this chapter I will explain the application's interface. Beginning with the state diagram and its explanation, then I continue explaining the design process followed from the first drawings to the final interface. And finally there is a detailed description of all different screens of the application's interface.

7.1 Design process

In this section is detailed the design process that has been followed to create the application's interface.

7.1.1 Mock-ups

The first step was to decide how many views or screens would have the application, and also decide the behaviour of the application. Once done, the next was to draw some pictures of the different views of the application, and then begin to implement them.

The first contemplated design had an interface with nine different screens, and these are the first drawings:

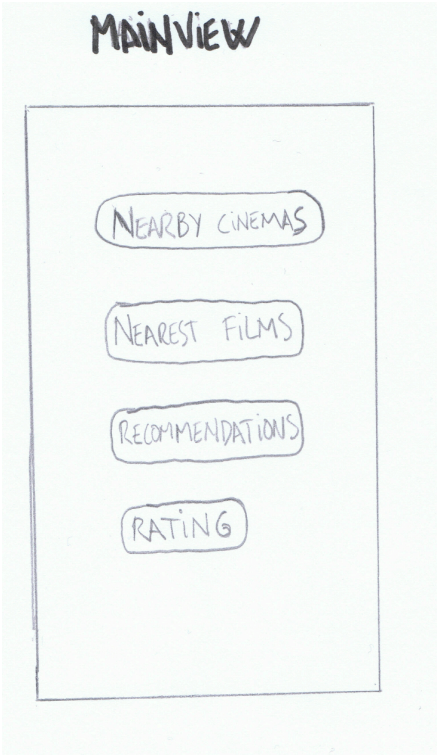

Figure 18. Mock-up of the Main View

The "Main view" screen (Figure 18) was designed as the first screen of the application where the user would choose what he would do. This screen would have three or four options to choose and it would be set as the home screen.

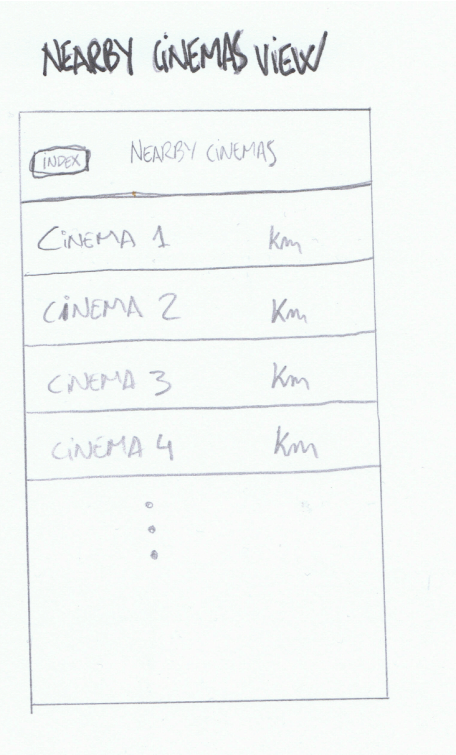

Figure 19. Mock-up of the Nearby cinemas view

The "Nearby cinemas view" (Figure 19) was designed as a simple table view with the cinemas sorted by distance.

Figure 20. Mock-up of the Cinema info view

The "Cinema info view" (Figure 20) would be the screen containing all the info about the cinema and it also would contain a little map showing the locations of the user and the cinema. It would contain a button to check the films that would be available at the cinema.

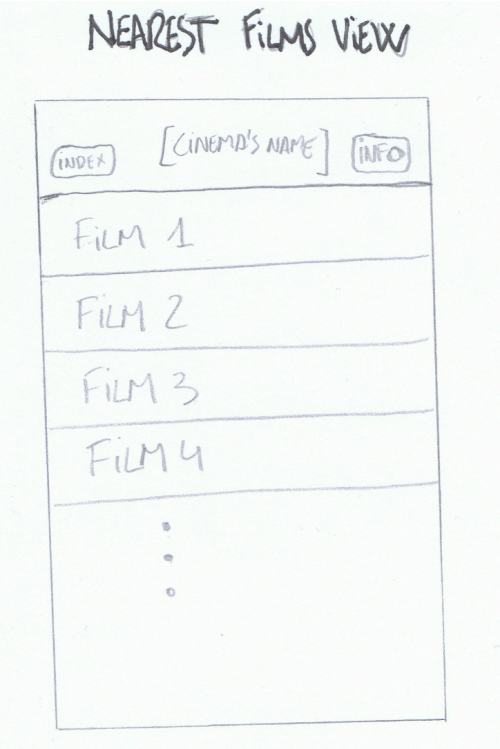

Figure 21. Mock-up of the Nearest films view

The "Nearest films view" (Figure 21) would be table view that would contain all the movies sorted by distance.

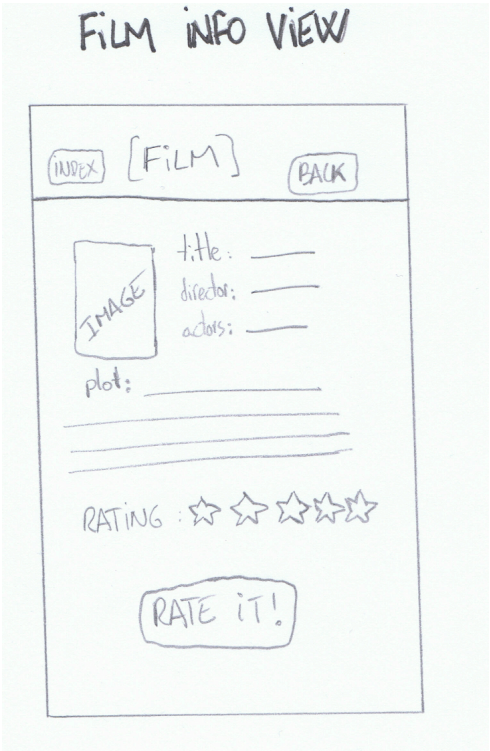

Figure 22. Mock-up of the Film info view

The "Film info view" (Figure 22) would be the screen containing the all the info about the movie together with the critics rating and a button to go to the "Rating view".

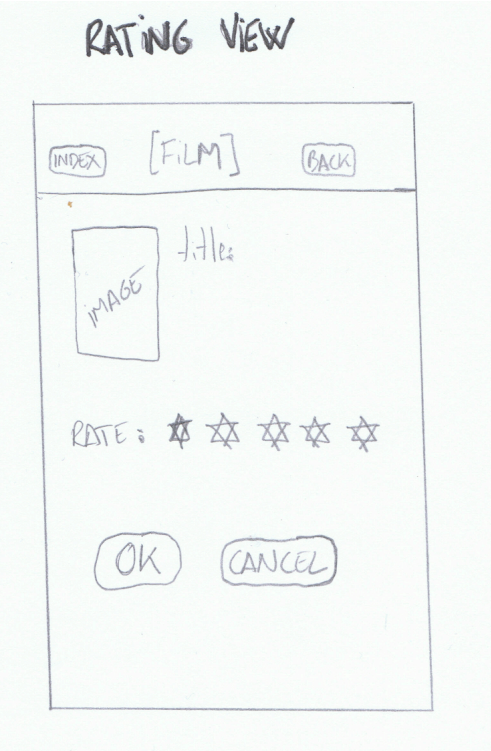

Figure 23. Mock-up of the Rating view

The “Rating view” (Figure 23) would be a very simple screen containing the movie title, the movie poster, the rating interface and two buttons to proceed or to cancel.

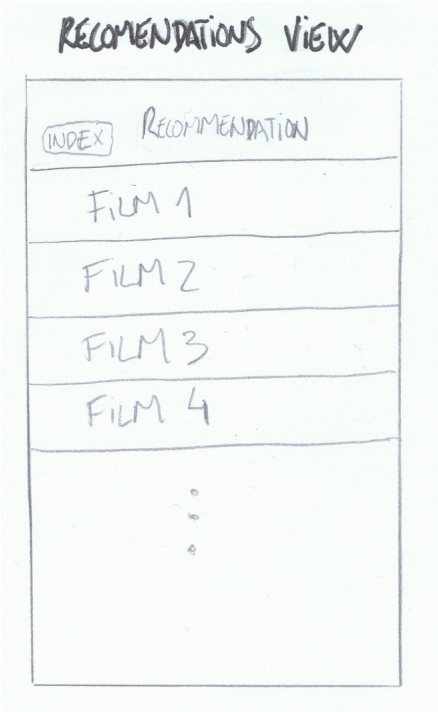

Figure 24. Mock-up of the Recommendation view

The “Recommendation view” (Figure 24) would be another simple table view with all the movies sorted by critics rating.

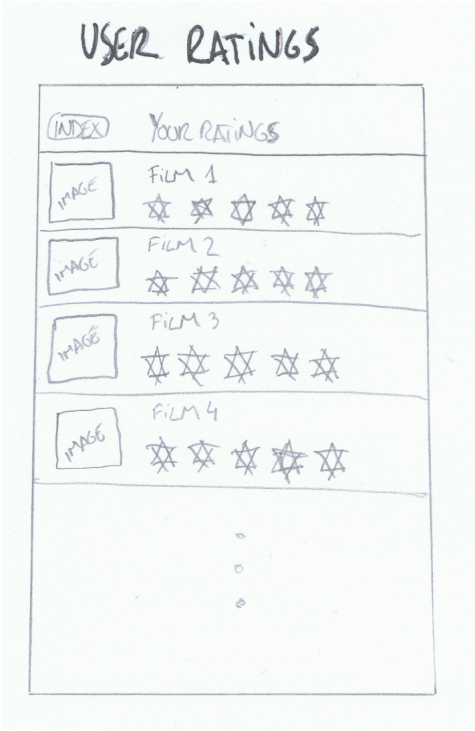

Figure 25. Mock-up of the User Ratings

The "User ratings" (Figure 25) screen would be a table view with all the movies that he had rated and where he could check his ratings and modify them.

There were some screens that I had not designed because they were too simple or because they were very similar to others that I had already designed.

With this first design of the application interface, I made a first version of the application to begin to perform user test and obtain feedback to improve the application.

7.2 Interviews / User tests

After some interviews with users and experts, the application interface underwent some major changes and even some of the screens were removed. And some new screens were designed according with the meetings results.

The first screen that was removed was the "Main view", which was the first screen of the application, because after the interviews my point of view had changed and I concluded that the user should not be forced to choose one option, but the application must submit information directly. And then the new first screen was the "Recommendation view" with the movie recommendation.

The other screen removed was the "Rating view". This screen was designed as a screen where the user only can rate the movie, with no other functionality. Then I decided to integrate the rating interface in other screens where the user can rate the movies without having a dedicated screen for rating each movie.

One of the most significant changes that I made was the info displayed in the screens that have table views such as the "Recommendation view", where the table cells now would have more information as the distance to the nearest cinema where the movie is showed and the critics' rating.

Another change was to include different sorting options in the movie lists where now the movies could be sorted by title alphabetically or sorted by the rating of the critics. In the "Recommendations view" now the user could choose the maximum distance for searching movies between three preset options.

In the “Film info view” screen I made little changes. The rating interface was added to this screen and the next shows info was added too. And I decide to reorganize the info in the screen. The most important change in this screen was to disable the scrolling in the text views corresponding to the cast and the movie plot, and build them dynamically depending on their content. It must be said that this screen has been duplicated and in this new screen I added a button for changing the cinema where the movie is shown when the user come from the “Recommendation view” and the nearest cinema is set as default.

Another screen that emerged from the meetings was the “Cinemas view” as a map with all the cinemas and the user position, and in this map view and in the original view with the list of cinemas I included a button to switch between these two screens. The view with the map was included to let the user to check all the cinemas in the map. With this new screen, the interface would become more interactive and more visually appealing.

Another important change was to enable scrolling in screens and not having a fixed size for each of them. Thus a screen may contain more information for the user. This change was also made because the iPhone allows large screens with scrolling and the iPhone users are used to deal with scrolls and large screens.

The last change I made was the removal of the “Home” or “Index” buttons because iPhone applications usually not have this kind of buttons. Instead of this kind of buttons the iPhone applications use the navigation bar, which have the back button to navigate through the viewed screens.

Also with the interviews I could detect some code errors and then fix them. Another thing that I fixed was the speed of the application, performing a code review and checking the function calls.

7.3 Final design

After making all the changes mentioned above the resulting application interface is as is shown in this section.

There are some common parts in all the screens like the navigation bar [17] or the tab bar [18]. The navigation bar (Figure 26) is set at the top of the screen containing the screen title and depending on the screen it could can contain different kind of buttons like back buttons, refresh buttons or switch buttons.

Figure 26. Set of navigation bars

The tab bar (Figure 27) is set at the bottom of the screen and it is like a switch button with four options. These four options are Cinemas, Recommendation, Ratings and Critics, which are the main options of the application. This bar don't change between screens, is always equal.

Figure 27. Tab bar

The Rating stars interface (Figure 28) is another common part in the application, which is like a slide bar, made of five stars, that allows the user to insert a rate for a movie because each interface is related to a movie. The interface works like a slide bar or it works touching one of the five stars. When the user finishes to use this interface the rating for the related movie is automatically inserted in the database.

Figure 28. Rating interface

There is another common set of stars, the Rating stars (Figure 29). The Rating stars are used to show the critics' rating obtained by the Recommender System explained in the next chapter.

Figure 29. Rating stars

The stars in Rating stars and in the Rating interface have different colour to differentiate between the user's rating and the critics' rating.

Other common parts are the movie cells, there are three different types depending on the screen. These cells are only in the table views and not in all screens of the application. The first type is the Movie cell (Figure 30), which has a label with the title of the movie, the movie poster and the rating stars with the critics' rating.

Figure 30. Movie cell

The second cell type is the Recommendation cell (Figure 31), which is too similar to the Movie cell. The Recommendation cell has a label with the movie title and it also contains another smaller label with the closest cinema and the distance between the user and the cinema. As in the Movie cell, the Recommendation cell has the movie poster and the ratings stars with the critics' rating.

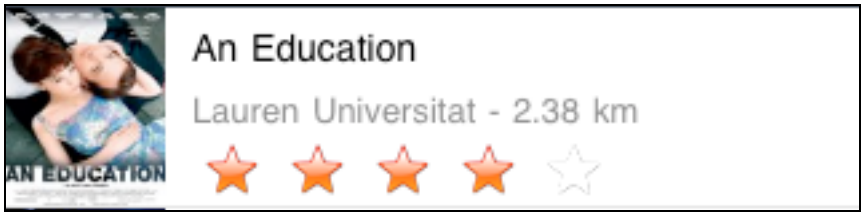

Figure 31. Recommendation cell

The last type is the Rating cell (Figure 32), which contains the movie title, the movie poster and the rating interface explained before where the user can rate the movie.

Figure 32. Rating cell

Now I will describe each of the screens in the application. The first screen in the application is the recommendation screen (Figure 33), which contains the recommendations done by the Recommender system. A table [19] containing the recommended movies composes this screen. The cells of this table are recommendation type cells described before. This screen also contains a tool bar with the distance selector, which is a switch button with three options that allows the user to select the maximum distance to search cinemas. And in the navigation bar there is the title.

The cells of the table are sorted by the critics rating and alphabetically and the table is refreshed every time that the user rates a movie in the application. While the screen is loading or the table is refreshing, a loading wheel appears in the title view.

When the switch button is pressed the table is refreshed with the new searching criteria.

If a cell is selected the application shows the movie info screen (Figures 36 and 37).

Figure 33. Recommendation screen

The cinemas screen is another screen is the cinemas screen, which has two versions. One that is a map (Figure 34) and other that is a list (Figure 35). These screens show all the cinemas that the application has in the database.

The cinemas map screen has the title and a switch button in the navigation bar. The switch button, with two options, allows the user to switch between the map and the list screens. The screen contains a map view with all the cinemas and the user position [20]. The cinemas have a red pin as a landmark and the user position has a blue round mark.

When a landmark of a cinema is touched a callout appears with the cinema's name and a button, which can be pressed to go to the cinema info screen (Figure 45). If the landmark pressed is the user position the message "*User location*" appears.

The cinemas list screen is a table that contains all the cinemas sorted by distance. The table cells contain the cinema name and the distance between the user and the cinema. If a cell is selected the application shows the cinema info screen.

This screen also contains the same switch button that the map screen in the navigation bar to switch between these two screens.

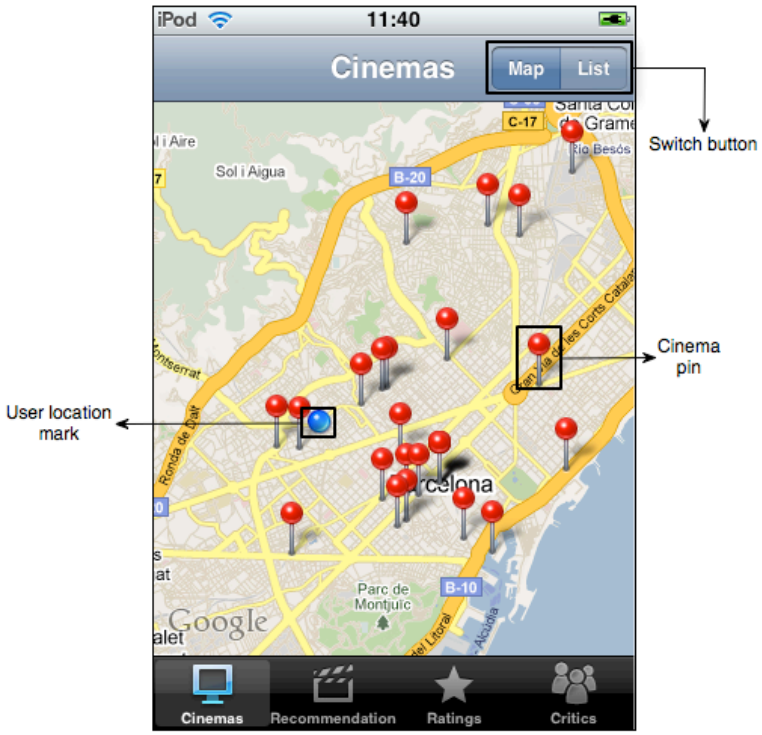

Figure 34. Cinemas' map screen

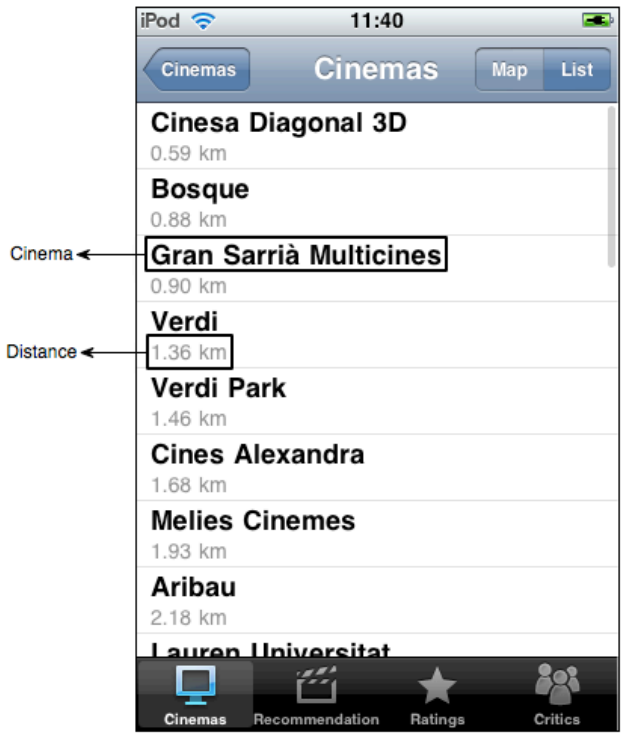

Figure 35. Cinema's list screen

The Movie info screen contains info about a movie. This screen has two versions, the first one (Figure 36 and figure 37) is showed when the application comes from the recommendation screen (Figure 33) and the second (Figure 38 and figure 39) is showed when the application comes from the movies screen (Figure 46 or figure 47). The difference between the screens is that the first version has a button that allows the user to change between the cinemas that are playing the movie.

The two screens have a switch button with two options in the navigation bar that allows the user to switch between the movie info screen and the "show times" screen (Figure 41). The navigation bar contains a back button too.

Figure 36. Movie info screen version 1 part 1

Figure 37. Movie info screen version 1 part 2

Other common parts in the two versions of the movie info screen are the movie related info objects like the title, the poster image, the director, the genres, the cast and the plot. The cinema and the next shows info are common objects in the two screens too. And the last common objects are the critics rating stars and the user rating stars interface.

The only difference between the versions is the "*other cinemas*" button that appears in the first version of the screen, as I said before. When this button is pressed the application shows the "playing at" screen (Figure 40).

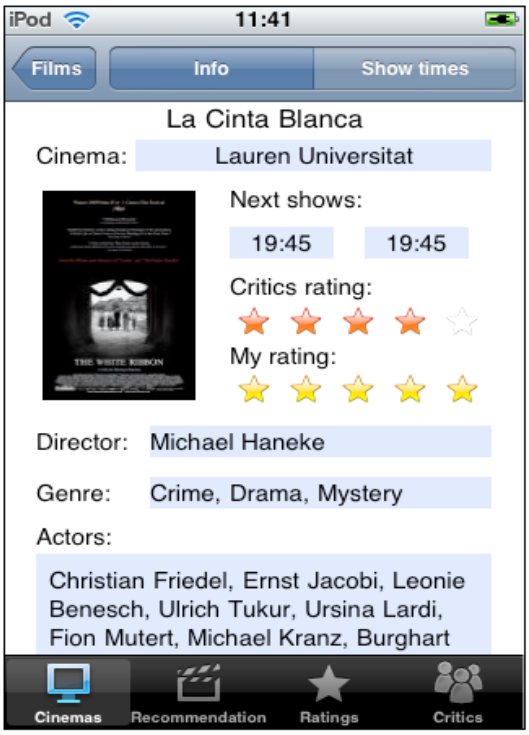

Figure 38. Movie info screen version 2 part 1

Figure 39. Movie info screen version 2 part 2

The “Playing at” screen (Figure 40) contains the list of cinemas that play a particular movie. It contains the title and a back button in the navigation bar. In the main view it contains a table with the cinemas that play a movie sorted by distance. Each cell of the table contains the name of the cinema, the distance between the user and the cinema and a round button.

If a cell is selected the application returns to the movie info screen (Figure 36 and figure 37) updating the cinema and shows info. But if the round button is pressed the application goes to the cinema info screen (Figure 45).

Figure 40. "Playing at" screen

The "show times" screen contains info about times of a movie in particular cinema. The screen has the same switch button that the movie info screen in the navigation bar and that bar also contains a back button. The main view of the "show times" screen contains the movie title and the cinema name info about the show times checked. The main view also contains three tables, the first one contains the today shows, the second one contains tomorrow shows and the last one contains other shows.

Each cell of the three tables contains the day of the show, with format DD-MM-YYYY, the hour of the show, with format HH:MM, and it also contains the "*Buy Tickets*" button.

If the "*Buy Tickets*" button is pressed appears an alert in the screen with a message (Figure 42 and Figure 43). The message content depends on whether the user has already seen the movie or not.

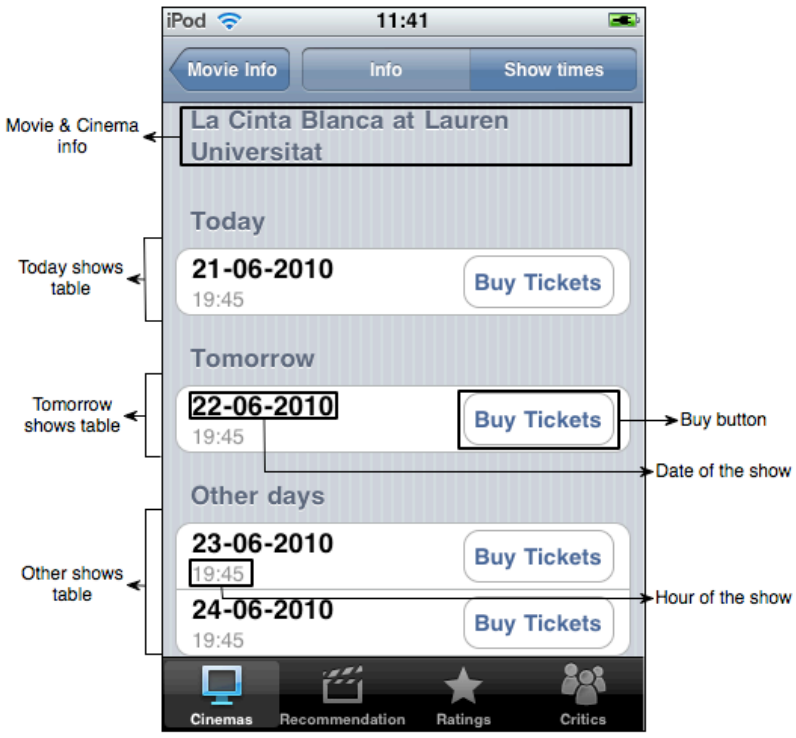

Figure 41. "Show times" screen

As I said before when the user presses the "*Buy tickets*" button an alert is shown. There are two different messages depending if the user has already seen the movie or not. If the user has not seen the movie before the application shows the alert in Figure 42. The message that contains the alert has information about the movie and cinema, for which the user buys the ticket. However if the user has already seen the movie the application shows the alert in Figure 43. The message in this alert warns the user that he has already seen the movie. In both cases if the user press the OK button, the application shows another alert (Figure 44) showing a thanks message.

If the user pressed the OK button the application goes to the Movie info screen (Figures 36 and 37 or Figures 38 and 39), but if the pressed button is the CANCEL one the applications stays in the "Show times" screen.

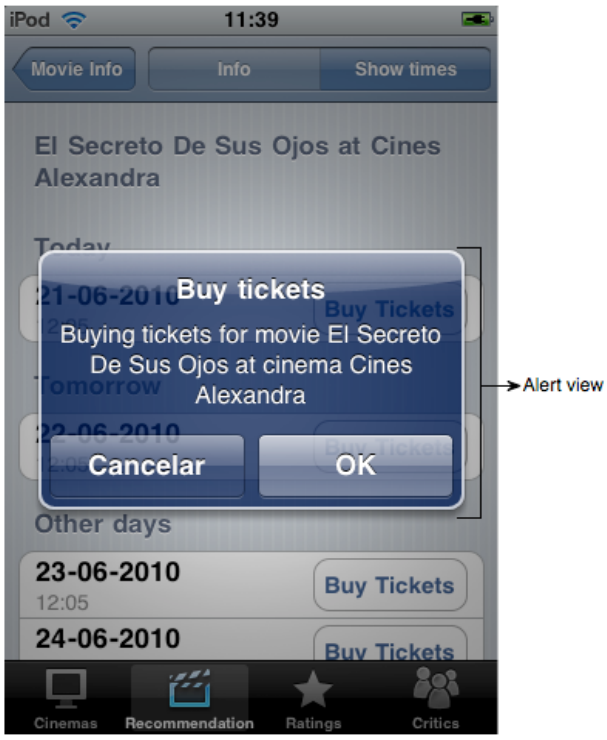

Figure 42. Buying new tickets screen

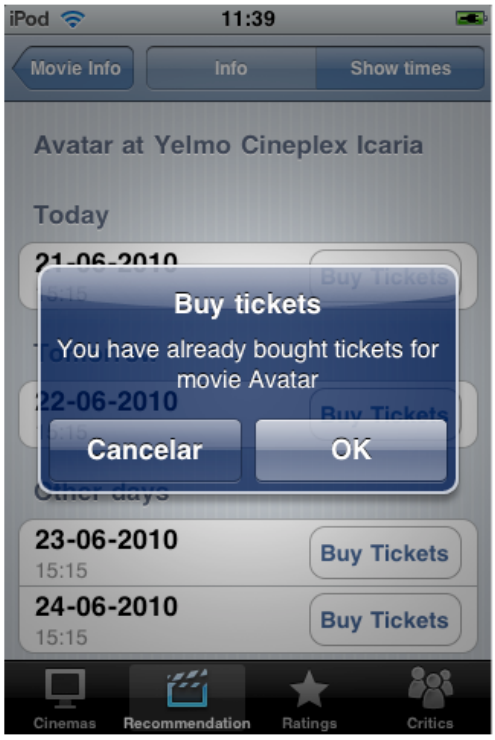

Figure 43. Buying already bought tickets screen

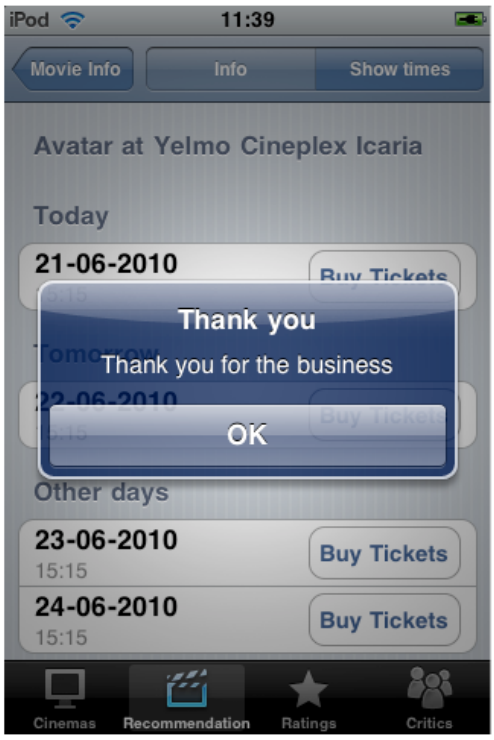

Figure 44. Thanks message screen

The Cinema info screen (Figure 45) contains info about a concrete cinema. The screen contains its title and a back button in the navigation bar. In the main screen there are some labels with the cinema name, the address and city of the cinema and the number of screens that the cinema has. The main screen also has a map view with the cinema and user locations. At the bottom of the main view there is the "Movies" button.

The "Movies" button is not showed if the application comes from the "Playing at" screen (Figure 40). If the button is pressed the applications goes to Movies screen (Figure 46).

Figure 45. Cinema info screen

The Movies screen (Figures 46 and 47) shows the list of movies that are in a concrete cinema. It contains a table with the movies of a cinema. The cells of the table are of type “Movie cell” (Figure 31) explained above. In this screen the navigation bar contains the title, a back button and a switch button with two options. The switch button allows the user to sort the table cells by title (Figure 46) or by recommendation value (Figure 47). This screen also contains a label with the name of the cinema between the navigation bar and the table.

If a cell is selected the application goes to the “Movie info” screen version two (Figures 38 and 39).

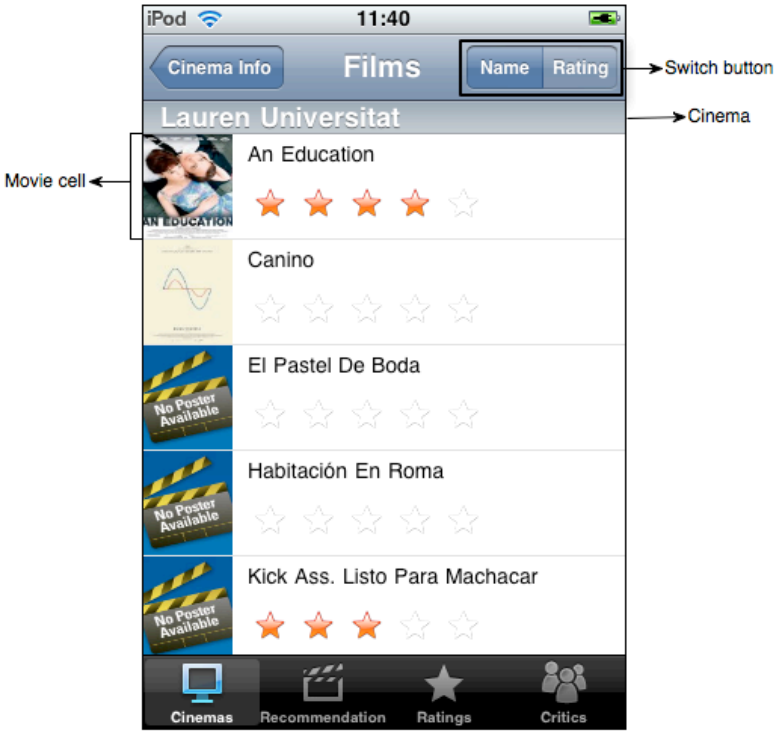

Figure 46. Movies screen sorted by title

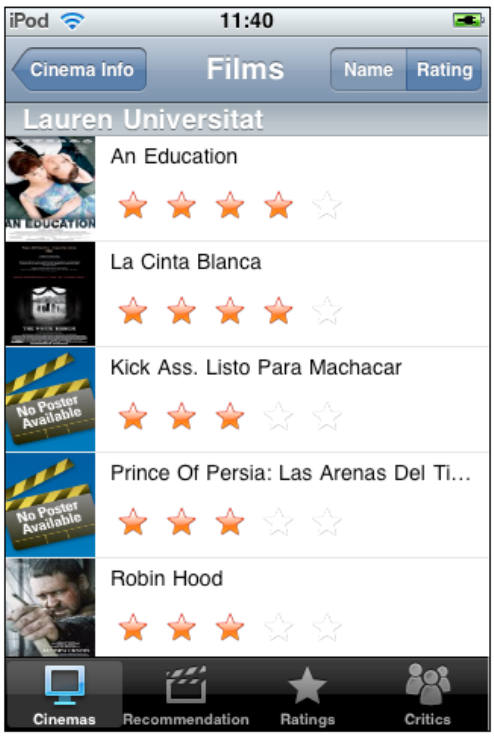

Figure 47. Films screen sorted by Rating

The Rating screen is the screen where the user can rate all the movies in the database. This screen can be divided in three views: watched movies (Figure 49), rated movies (Figure 48), and all movies (Figure 50). In the watched movies screen the user can rate the movies that he has purchased through the application. In the rated movies screen there are the movies that the user has already rated, that is the screen where the user can modify the ratings. And the last screen is the all movies screen that contains all the movies in the database, this way the user can rate all the movies without having bought tickets through the application.

These three different screens have the same structure. In the navigation bar there is the title and a switch button with three options to switch between the three screens. And in the main view there is a table with the corresponding films sorted alphabetically. The cells of this table are the type Rating cells.

When a movie in “Watched movies” screen is rated, automatically is deleted from this screen and added in the “Rated movies” screen. And when a movie in “All movies” screen is rated, is automatically added to the “Rated movies” screen.

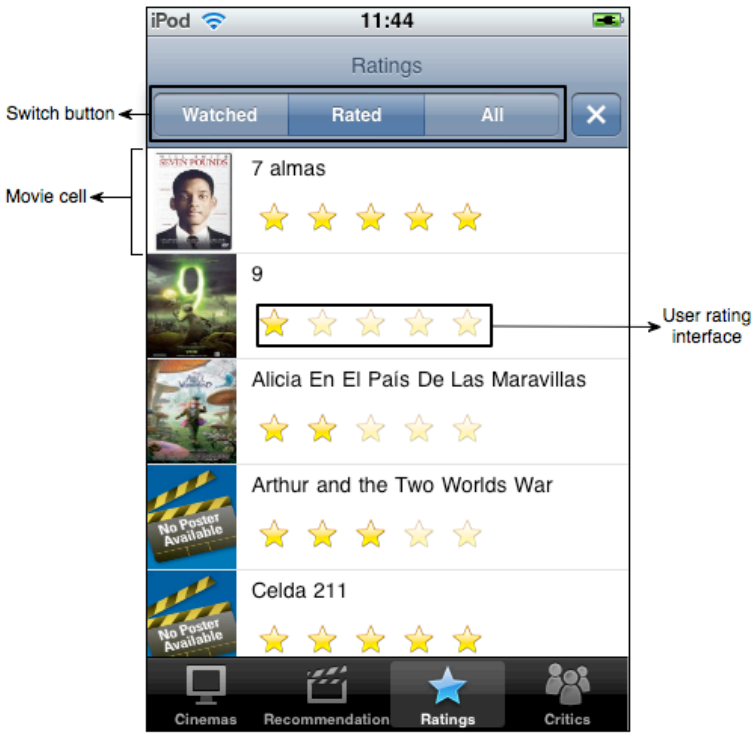

Figure 48. User ratings screen

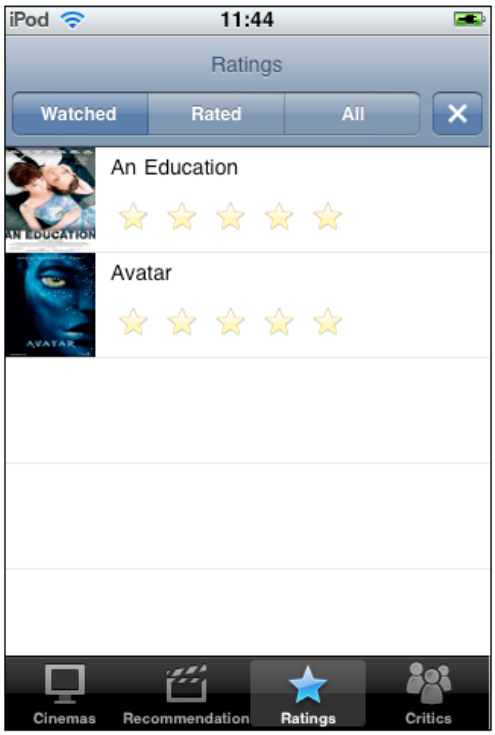

Figure 49. Pending user reviews screen

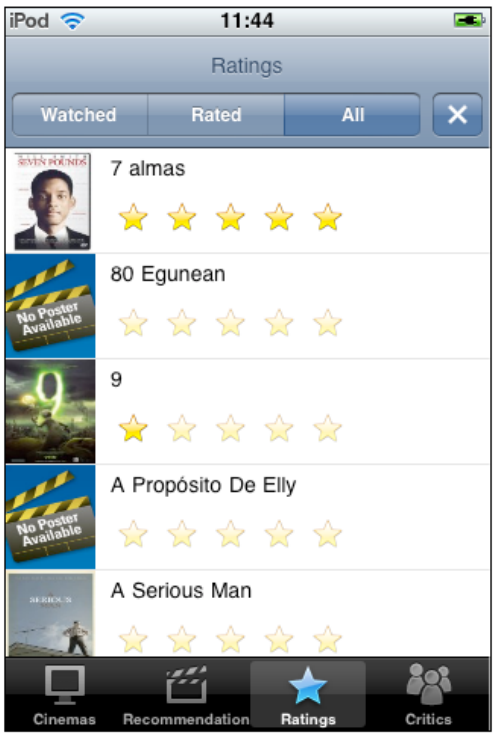

Figure 50. All movies rating screen

The Critics screen (Figure 51) contains information about the similarity between the user and the different critics that the application has.

In the navigation bar there is only the screen's title. In the main view this screen has a table containing the critics sorted by similarity. Each cell of the table contains a label with the name of the critic, another label with the different media, which the critic works for. And the cell also contains the percentage of similarity between the user and the critic.

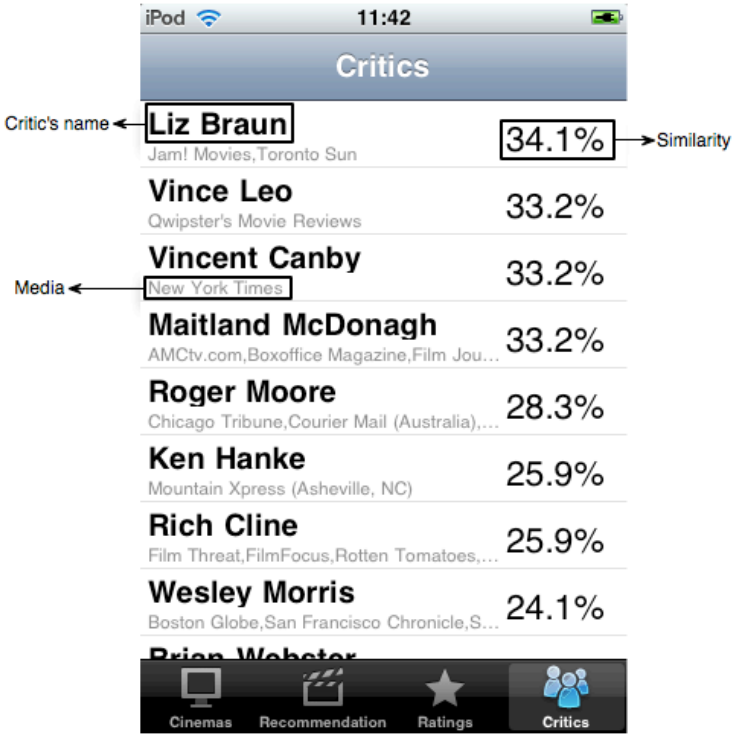

Figure 51. Critics' similarity screen

The figure 52 shows the alert view with message that appears when the application detects that the database is out of date.

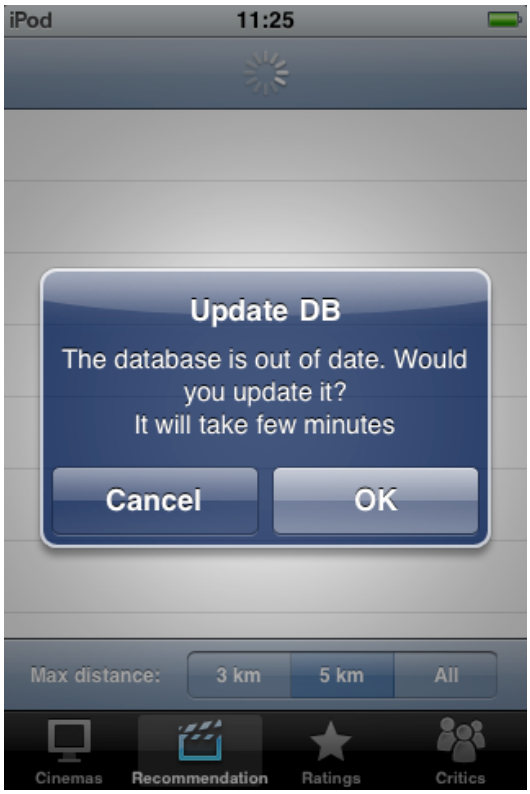

Figure 52. Update needed screen

The figures 53, 54 and 55 show the messages shown while the application is updating the database.

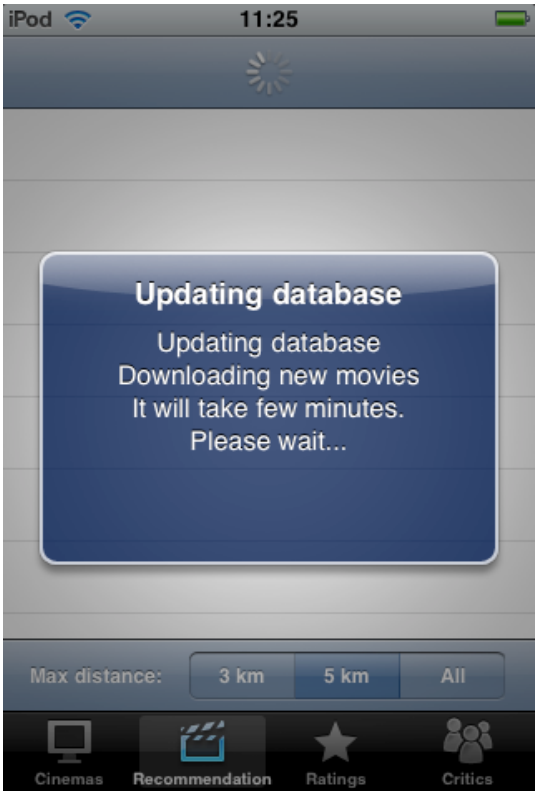

Figure 53. Updating movies screen

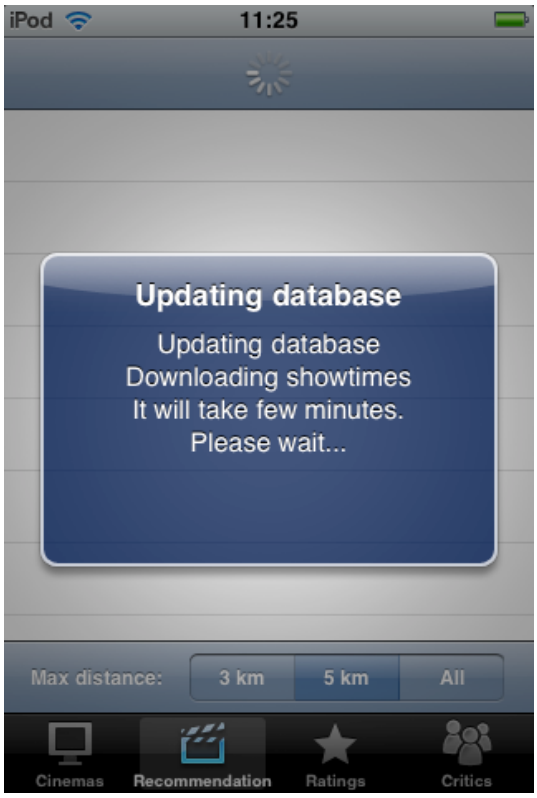

Figure 54. Updating showtimes screen

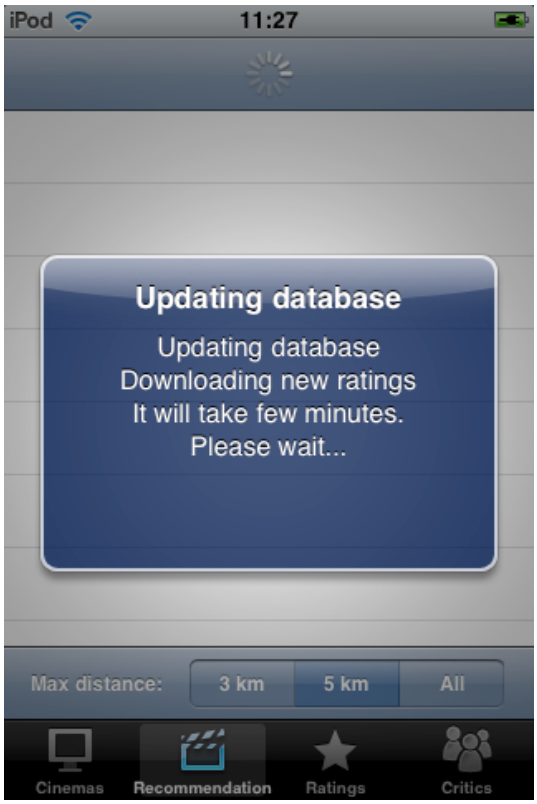

Figure 55. Updating ratings screen

The figure 56 shows the message shown if the update has finished correctly and the figure 57 shows the message shown if the user has cancelled the update.

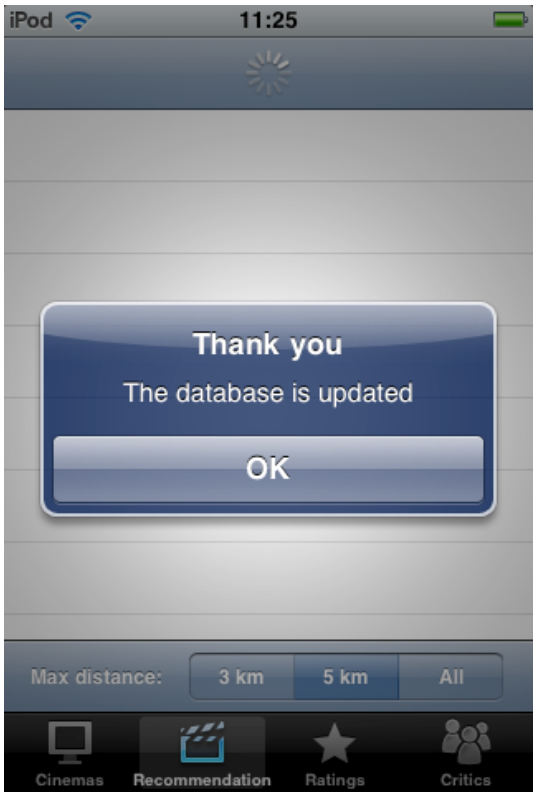

Figure 56. "Update finished" screen

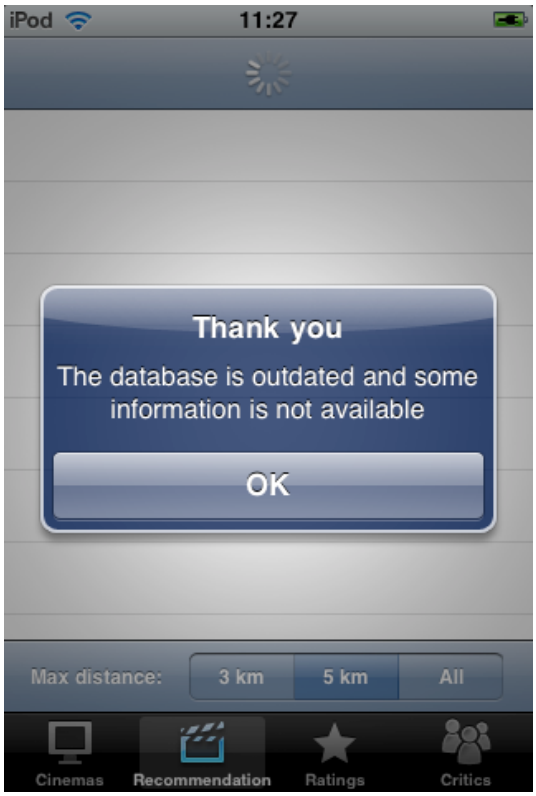

Figure 57. "Update cancelled" screen

7.4 Application logic – State diagram

This section shows the state diagram of the interface, which describes the behaviour of the application interface.

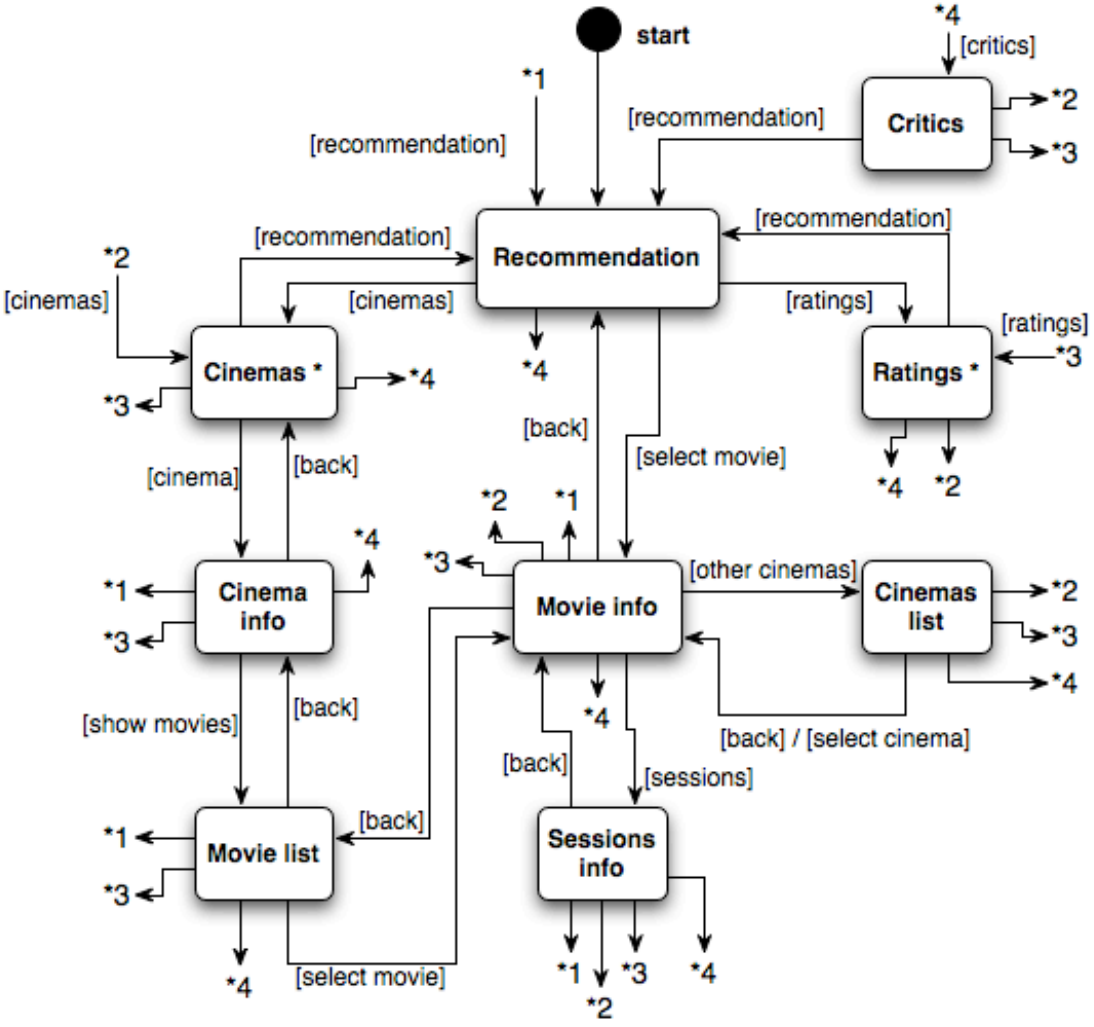

Figure 58. State diagram of the interface

The application starts at the Recommendation screen, where the user gets the recommended movies if she has rated some of them before. From the Recommendation screen the user can go to other screens like are the Cinemas, the Ratings or the Critics, or if the user wants, she can countinue with the Recommendation and she can go to check the information of a recommended movie.

If the user decides to check the information, she goes to the Movie info screen, where the information about the movie and ciema is shown and she can also check the showtimes for the movie going to the Sessions info, where the showtimes of the selected movie at the selected cinema are listed. Instead of checking the shows the user can modify the selected cinema going to the Cinemas list and selecting another one.

If the user decides to go to the Cinemas screen instead of continueing with the Recommendation, at this point the user can see the information in a map or listed in a table as we see in the Figure 59 part a, which represents the super state Cinema. In both screens the user can seect a cinema and the she goes to the Cinema info, where she cn check the information of the cinema, and from this screen the user can go to the Movie list, where are shown all the movies played at the selected cinema. If the user selects one movie then the application goes to the Movie info screen explained before.

But if the user decides to go to the Ratings instead of Cinemas, then she can check and rate the movies. As we see in Figure 59 part b, Ratings is a super state. There the user can move between the watched movies, which are the movies that the user has seen but she has not rate yet, the rated movies, which are the movies that the user has rated, and all the movies. The user

can rate the movies or modify some rating in every one of the screens of the Ratings super state.

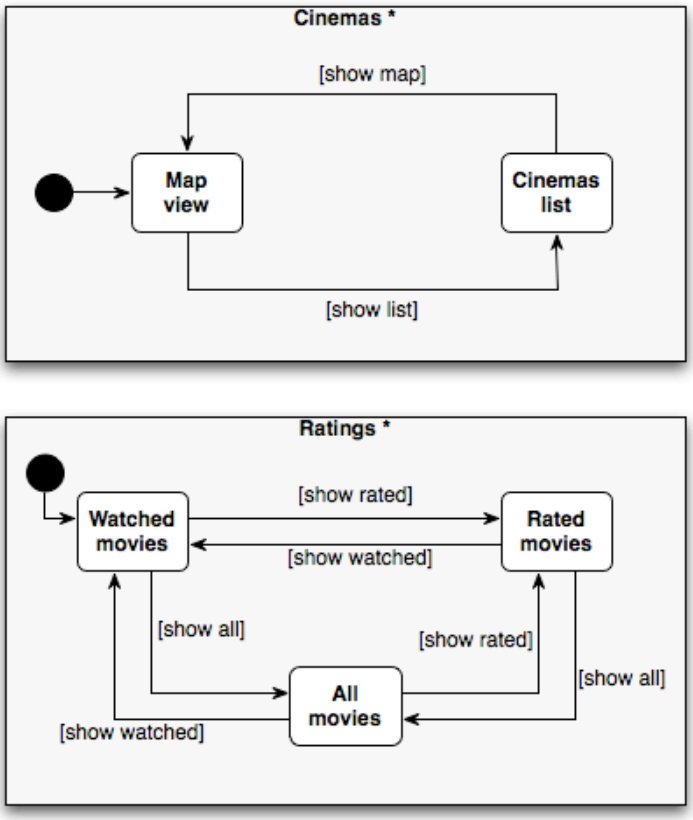

Figure 59. Super states of the state diagram (Figure XX)

The user can prefer going to Critics instead of go to Ratings, where she can check the similarity between her and all the critics.

We have to say that the user can navigate between the Recommendation, the Cinemas, the Ratings and the Critics at every time she wants in the application.

8. Recommendation

In this chapter I will explain the recommendation algorithm used in this project, which is a method for recommending items to users based on experts opinions. The Collaborative Filtering is a technique to filter or evaluate items through the opinions of other people. It makes use of peer user ratings in order to provide recommendations on the items that are unknown but my interest the target user. We use the Expert Collaborative Filtering, which is based on the *nearest-neighbour* (*k*NN) algorithm and predicts the estimated ratings of the movies. We use an adapted version of he algorithm explained in *The Wisdom of the Few* [21], a previous work of Xavier Amatriain. In this kind of algorithms similarity plays a central role. In our case the *k*NN algorithm always has one hundred neighbours because we have a fixed group of experts.

First of all we have a set of critics and their ratings crawled from the web, as I explain in chapter 6. Once we have this data we can calculate the similarity between the user and experts, to do it, we use a variation of the cosine similarity:

$$sim(u,c) = \frac{\sum^{i}(r_{ui}r_{ci})}{\sqrt{\sum^{i} r_{ui}^2}\sqrt{\sum^{i} r_{ci}^2}}$$

Where r_{ai} is the user rating of the movie i and r_{ei} is the expert rating of the movie i. When we have the similarity matrix between the user and experts we have to update it every time that a new rating is inserted or an old one is updated either by user or by experts.

Once we have obtained the first ten best neighbours $E = e_1 ... e_k$ for the user, we can predict the estimated rating value of a given movie i by computing a similarity-weighted average of the ratings from each neighbour, expert in our case, e in E:

$$r_i = \frac{\sum_{e \subseteq E} r_{ei} sim(u,e)}{\sum_{e \subseteq E} sim(u,e)}$$

Where r_i is the predicted rating of item i for the user, r_{ei} is the known rating for the expert e to movie i.

Using our approach of the Expert Collaborative Filtering addresses some of the traditional shortcomings of the Collaborative Filtering Systems. Ratings coming from authoritative experts in a given domain are much less likely to suffer from natural noise, which becomes when users introduce noise when giving their feedback to a recommender system in the form of careless ratings. Then using ratings from experts we expect ratings with less natural noise to produce more accurate recommendations. Also, because we are using ratings only coming from a small and supervised pool of experts, we will avoid any possibility that malicious ratings are injected into our datasets.

We expect experts to have a professional incentive to rate movies as soon as they appear. On the one hand, this means that, on average, they will rate many more movies than a regular user therefore minimizing sparsity of the ratings matrix. On the other hand, this will also help minimize the problem of the item cold-start, which is the lack of ratings in the new items or movies and then they can not be recommended. We expect to minimize the cold-start since we expect to have ratings even before any user has access to the content.

The scalability problem stems from the great size of the user pool that needs to be considered in usual Collaborative Filtering approaches. The number of users can grow up to millions or even

more depending on the size of the service subscribers. However, we can achieve quality ratings and recommendations from a relatively reduced number of experts.

Our approach also guarantees complete preservation of privacy for users reporting their ratings, while still offering the advantages of Collaborative Filtering. We can accomplish this because we draw a clear distinction between the target user and its related private ratings, and the public profiles and ratings from experts that can be transmitted to all clients without worrying about privacy issues. One of the reasons we can do this in our particular implementation is because we use a reduced number of experts. Traditional approaches to Collaborative Filtering do need to somehow share user information. Therefore, approaches to preserve privacy necessarily need to use more complex models.

Once we have the ratings for the movies to recommend, then we have to restrict the recommendation in two ways. The first one is to restrict the movies that are being played in the cinemas that are inside the maximum distance range selected by the user. When we have this restriction then we have to make another restriction, where we not consider the movies with a rating value lower than three. We do this restriction because we think that we should only recommend movies with a rating value bigger or equal to three. We do this way because we think movies with a less rating value are not good enough, we have to remember that values for rating go from 0 to 5, and movie with a value lower than three is not a movie for recommend. Finally we show the recommended movies sorted by rating using sort descriptors [22].

9. Conclusions

We think that this project is a complete one because we have used a wide range of technologies and features like are the iPhone, the web crawlers, the GPS, the recommendation, the database, and the client-server connection. During this project we also used a lot of programming languages like Objective-C, SQLite, XML, Python, and Shell script.

First of all we can say that although the application was designed for movies, it is not limited to them. We can easily adapt the application to any domain with the same structure.

In this project we have developed an application of recommendation of movies for the iPhone with a new feature that no other recommendation now available has. This new feature is the recommendation based on the user opinion and based on in the ratings of a set of experts. The recommender systems have some problems like the problem of ratings quality, and user privacy, which can affect to the system. With our Expert Collaborative Filtering can address these problems. Using expert ratings instead of plain user ratings we can minimize the effect of natural noise in ratings and the use of a reduced set of experts allows more efficient computation than the traditional Collaborative Filtering approaches and it avoids cold-start problems.

With the client based Expert Collaborative Filtering architecture, which maintains the user ratings in the user's device and calculates locally the recommendation with the expert ratings downloaded from the server. This approach can solve privacy issues because there is no chance that any private information is transmitted to remote machines.

We also use the GPS signal of the device to get the user position and the nearest cinemas and the nearest movies. We also use the user's position to rank the recommendation.

We try to design an easy to use interface for the application without a complicated navigation between screens and features, which can be easy to follow for all kind of users.

At the server side we designed the RESTful architecture completely independent from the device, which can be exploited for other kind of platform like could be another mobile devices, web pages and others.

There are some things in the application that can be improved such is the recommendation speed, which we could try to get the recommendation faster because now the recommendation takes a long time and we think that we could have better times. This improvement could be hard to achieve because we are limited by the device hardware but we could be few seconds better.

Another improvement would be the time it takes to upgrade the device. We could improve this time trying other XML parsers because we choose the default parser in he iPhone libraries and maybe with other parsers the updating times improve.

We also need to improve the ratings of Spanish movies. Now we are collecting rating from Rotten Tomatoes where few Spanish movies are rated. This problem is hard to solve because there is no similar site to Rotten Tomatoes in Spanish language or with more Spanish movies.

The last thing we could improve is the Python module that be use to get the movies info, which is the IMDbPY package. In the last weeks we have detected that the module does not return all the information correctly and this causes some problems. This could be improved updating this Python module to his last version available. Another option could be to get the information needed from the IMDb web page, or any other containing the information needed, with a new crawler.

Future work in the application developed could include the use of the iPhone's accelerometer to have some new features. We also could adapt the application to move together with the iPhone's rotator, and then the screen would change horizontally and vertically with the movement of the device.

More future work could be to link the application with some other application or web page to make possible to buy tickets. And we also could extend the application to have more cities, and doing it the application would not only have the information about Barcelona.

10. References and Bibliography

10.1 References

[1] Apple Inc. *A Tour of Xcode*. USA: Apple Inc. 2009

[2] Apple Inc. *Xcode Workspace Guide*. USA: Apple Inc. 2010

[3] Apple Inc. *Xcode Project Management Guide*. USA: Apple Inc. 2010

[4] Apple Inc. *Xcode Build System Guide*. USA: Apple Inc. 2010

[5] Apple Inc. *The Objective-C 2.0 Programming Language*. USA: Apple Inc. 2009

[6] Apple Inc. *SDK Compatibility Guide*. USA: Apple Inc. 2010

[7] Apple Inc. *iOS Technology Overview*. USA: Apple Inc. 2010

[8] Apple Inc. *Event Handling Guide for iOS*. USA: Apple Inc. 2010

[9] Apple Inc. *Low-Level File Management Programming Topics*. USA: Apple Inc. 2009

[10] Apple Inc. *Threading Programming Guide*. USA: Apple Inc. 2010

[11] Apple Inc. *URL Loading System Programming Guide*. USA: Apple Inc. 2010

[12] Apple Inc. *Instruments User Guide*. USA: Apple Inc. 2010

[13] Apple Inc. *Interface Builder User Guide*. USA: Apple Inc. 2010

[14] Amatriain, Xavier; Ahn, Jae-wook. *Towards Fully Distributed and Privacy-preserving Recommendations via Expert Collaborative Filtering and RESTful Linked Data*. 2010

[15] Apple Inc. *Property List Programming Guide*. USA: Apple Inc. 2010

[16] Apple Inc. *Event Driven XML Programming Guide*. USA: Apple Inc. 2010

[17] Apple Inc. *iPhone Human Interface Guidelines*, pages 76-79. USA: Apple Inc. 2009

[18] Apple Inc. *iPhone Human Interface Guidelines*, pages 81-84. USA: Apple Inc. 2009

[19] Apple Inc. *Table View Programming Guide for iPhone*. USA: Apple Inc. 2009

[20] Apple Inc. *Location Awareness Programming Guide*. USA: Apple Inc. 2010

[21] Amatriain, Xavier; Lathia Neal; Kwak, Haewoon. *The Wisdom of the Few*. 2009

[22] Apple Inc. *Sort Descriptor Programming Topics*. USA: Apple Inc. 2007

10.2 Bibliography

Apple Inc. *Coding Guidelines for Cocoa*. USA: Apple Inc. 2010

Apple Inc. *Cocoa Application Tutorial*. USA: Apple Inc. 2007

Apple Inc. *iPhone Development Guide*. USA: Apple Inc. 2009

Apple Inc. *iPhone Application Programming Guide*. USA: Apple Inc. 2009

Apple Inc. *Cocoa Fundamentals Guide*. USA: Apple Inc. 2010

Apple Inc. *iOS Application Programming Guide*. USA: Apple Inc. 2010

Apple Inc. *Your First iOS Application*. USA: Apple Inc. 2010

Apple Inc. *Resource Programming Guide*. USA: Apple Inc. 2010

Apple Inc. *View Controller Programming Guide for iOS*. USA: Apple Inc. 2010

Apple Inc. *Bundle Programming Guide*. USA: Apple Inc. 2010

Apple Inc. *Internationalization Programming Topics*. USA: Apple Inc. 2009

Apple Inc. *Locales Programming Guide*. USA: Apple Inc. 2008

Apple Inc. *Data Formatting Guide*. USA: Apple Inc. 2009

Apple Inc. *View Programming Guide for iOS*. USA: Apple Inc. 2010

Apple Inc. *Memory Management Programming Guide*. USA: Apple Inc. 2010

Apple Inc. *Device Features Programming Guide*. USA: Apple Inc. 2010

Apple Inc. *Date and Time Programming Guide*. USA: Apple Inc. 2010

iOS Overview:

http://developer.apple.com/iphone/library/referencelibrary/GettingStarted/URL_iPhone_OS_Overview/index.html#//apple_ref/doc/uid/TP40007592

Data Management Coding How-To's:

http://developer.apple.com/iphone/library/codinghowtos/DataManagement/index.html

User Experience Coding How-To's:

http://developer.apple.com/iphone/library/codinghowtos/UserExperience/index.html

Networking & Internet Coding How-To's:

http://developer.apple.com/iphone/library/codinghowtos/NetworkingAndInternet/index.html

Tools Coding How-To's:

http://developer.apple.com/iphone/library/codinghowtos/Tools/index.html

Tools for iOs Development:

http://developer.apple.com/iphone/library/referencelibrary/GettingStarted/URL_Tools_for_iPhone_OS_Development/index.html#//apple_ref/doc/uid/TP40007593

Creating an iPhone Application:

http://developer.apple.com/iphone/library/referencelibrary/GettingStarted/Creating_an_iPhone_App/index.html#//apple_ref/doc/uid/TP40007595

Xcode and the iPhone SDK:

http://idevkit.com/iphonedev/2009/09/xcode-and-the-iphone-sdk/

iPhone SDK: How To Test For Network Reachability:

http://www.iphonedevx.com/?p=657

iPhone SDK: A Beginner's Guide:

http://www.jamesabrannan.com/iphonesdk_tutorials.html

Best Practice iPhone SDK App Design:

http://www.slideshare.net/bess.ho/best-practicei-phone-svcc2009key

Improving iPhone App User Experiences with Activity Indicators:

http://avaimobile.com/blog/bid/22284/Improving-iPhone-App-User-Experiences-with-Activity-Indicators

SQLite Documentation: http://www.sqlite.org/docs.html

REST Web Services: http://www.xfront.com/REST-Web-Services.html