Final degree project

Project Alexandria:

"Development of a social networking site prototype"

Student: José Luis Besante Alcayna
Jobeal1@etsinf.upv.es
Computer Science degree
Prague, June 28th 2011

Director: Ing. Božena Mannová, Ph.D.
Czech Technical University in Prague

Codirector: Ing. Juan Carlos Ruíz García, Ph.D.
Universidad Politécnica de Valencia

Index of contents

Index of images

Index of tables

1. Software concept

As a result of, both the important development of web technologies during these years and the generalization of the use of Internet web-services by almost all the population, social networking has become more and more well-known and used.

While the most famous social networks are intended to a wide and heterogeneous range of users, some of the smaller ones try to focus in a specific group of population with some aspects or habits in common.

The aim of this project is to develop a web platform prototype following the principles of social networking web-systems but intended to a specific range of users.

Project Alexandria is a social network prototype intended to users interested in literature. Besides usual functionality and communication mechanisms available in generic social network, the site tries to provide the user some tools to ease him the management of his own library or the possibility to find and borrow the book he is interested in reading.

2. Requisite specification

2.1 Introduction

2.1.1 Aim of this document

In the following pages of this document a list of the specified requirements linked to the developed application “Project Alexandria” will be found. The writing process has been done as precise as possible trying to avoid both redundancy and lack of information in order to make it suitable of its purpose, on the one hand guiding the development process, becoming a reference to check during the implementation phase and, afterwards, during the validation of the product, once the development has been completed. On the other hand, it can also be helpful for future users or developers who can improve the application in the future.

2.1.2 Application field

The developed application, which was called under agreement **“Project Alexandria”** can be classified, attending to its functionality, among the ones usually referred as social networks. Within these ones several classifications can be done, according to different approaches. One of them could be the classification regarding to the field they devote their functionality, once they have fulfilled the common requirements all the social networks have (f.i. association and communication mechanisms among users). Under this approach our application will be placed among the ones linked to literature. Some existing examples can be found searching through the network. This search was performed as a previous step to the development of the current application, a brief report of this process can be found at the end of this document (Appendix).

Since after years of development social networking applications have reach an important complexity level, the product developed within this project is just a prototype implementing the basic behavior all the social network PROPUESTAS have in common.

2.1.3 Definitions, acronyms and abbreviations

Avatar: Graphical representation of the user. Both avatar and nickname become users virtual *alter ego*.

Authorization: Process consisting of the definition of access control rules to decide whether access requests from users shall be approved or disapproved. Defined rules will control both resources and actions.

Authentication: Process whose aim is confirming the identity of a user. Information which identifies uniquely a user will be required in order to confirm his identity.

IEEE: Institute of Electrical and Electronics Engineers.

JavaScript: Scripting language used at client-side, implemented as part of a Web browser in order to provide enhanced user interfaces and dynamic websites.

MySQL: is a relational database management system that runs as a server providing multi-user access to a number of databases.

Nickname: A name chosen by the user as his identity in the system. It is not required this to be linked with user's real name.

Notification: Special sort of message whose aim is let users know about changes in the system being its administrators the ones who send it and its users its recipients.

Password: A secret word chosen carefully by the user that will have to be introduced each time he wants to access the site as a part of the authentication process.

PHP: General-purpose server-side scripting language originally designed for web development to produce dynamic web pages.

2.1.4 References

IEEE-STD-830-1998: ESPECIFICACIONES DE LOS REQUISITOS DEL SOFTWARE.

2.1.5 Global approach

Once the main context of the developed application has been detailed, our next step will be setting an accurate description of the product and the specific requirements linked to this one. This procedure will be performed following the rules set by IEEE-830 standard referred in the previous section.

2.2 General description

2.2.1 Product approach

The application whose specification is being set constitutes an independent software entity, it is not linked to any other application or a component of a bigger software product, therefore the only functional requirements and constraints we have to bear in mind are the ones directly related to the developed application and its context.

On the other hand, correct and expected behavior can be granted only within the technologies that have been chosen at the beginning of the process because they are considered as the best solutions to fulfill the application needs. A list of selected technologies that should be available either at the client side (user webbrowser) or at the server side (hosting service) besides some reasons to justify their election can be found in the following chapters of this document.

2.2.2 Product functionality

Main functionality :

In the following list can be checked the main features of the system that is to be developed.

- Allow user-profile creation where personal details, contact details and user preferences and interests can be checked.

- Allow user association by groups they can create. A user will be able to check easily which participants have joined the groups he belongs to. He will be able to share information with all of them by a group-wall.

- Allow private user communication by providing a private message mechanism.

- Provide a mechanism to ease users cataloguing their favourite books.

    - Searching and filtering features on system database are required for this purpose.
    - A book registration tool is required when a book has not been registered in the system.

- Allow users to define a set of genres they are interested in reading and let the system suggest them new books they might like.

- Allow users to write short book reviews and set a mark according to their satisfaction after reading the book.

Extra functionality :

Following features will be implemented as an extra depending on temporary constraints during the system development.

- Exchange book system among users living in the same area.
    - Define a set of books a user would be willing to share.
    - Define a set of books a user would like to borrow.
    - Define a sort of exchanging rules.

- Allow users to share their own literary works with the community.

- Establish an event system.
    - News, book signing, independent events, literary coffee shops.

2.2.3 User expected skills

The minimum skill level a possible user of the application should have in order to enjoy and take advantage of the developed functionalities are the ones expected to any user of other kind of websites.

The bunch of different technologies used during the development will be masked either by the server or by the user web-browser so site's appearance will not look different than any of the other sites visited by our users.
Within these rules could be stated that our application does not add an upper difficulty level to the ones usually found by internet or social networking users.

According to user's expected knowledge, closely linked not to the way the site is showed or the way a user interacts with it, but to the content itself, a simple and not ambiguous language will be used to ensure people from different environments and ages is able to use the site. Furthermore, as a further step in order to reach the maximum amount of users, the developed site will be available in several languages, offering Spanish and English at first but caring about the development of language selection mechanisms in such a way more languages can be added quickly and easily.

2.2.4 General restrictions

As it was stated previously, the only requirement to grant system's expected behavior is the availability of the required hardware or software components either at the client or the server side.

Client side:

- User has an Internet connection.
- User is using a web-browser supporting JavaScript to get access to the website.

Server side:

- Server supports PHP language.
- Server is able to get access, send commands and fetch information from the MySQL database hosting the site's information.

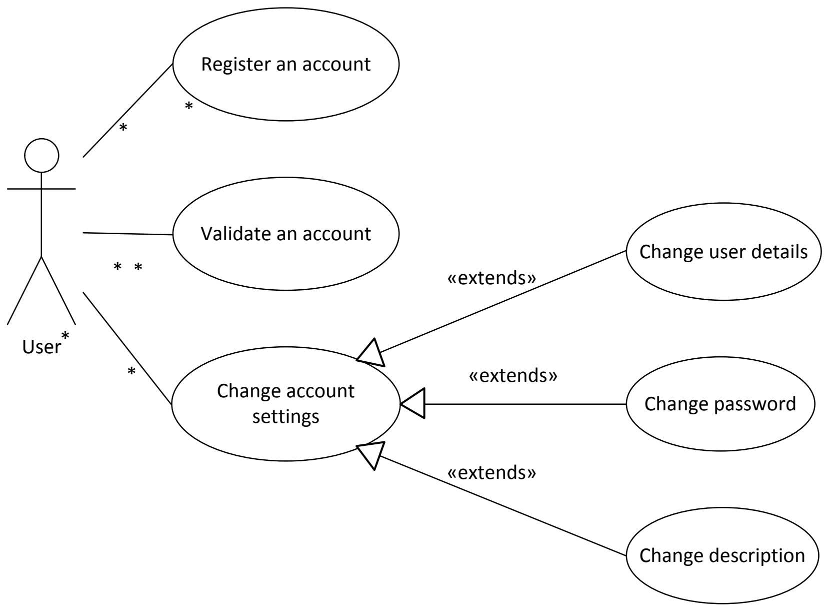
2.2.5 Assumptions and dependencies

There are no more assumptions or dependencies besides the ones stated in the sections 2.1, 2.3 and 2.4 of the current document.

2.3. Specifical requirements

2.3.1 Functional requirements

In the following pages, the complete list of agreed requirements to be fulfilled by the developed application is showed:



<u>**Register an account**</u>

General description:

Developed system will provide a registration system where interested internet users will be able to enclose their personal details in order to ask for a personal account and complete access to site's features.

The details to be filled in by the applicants during the registration process, can be divided in two groups depending on their compulsoriness:

- Mandatory details: Nickname, password, sex, current city, current country, e-mail address.
- Optional details: Birthday.

Inputs:

User must provide the system the following information:

- **Nickname**: A name chosen by the user as his identity in the system. It is not required this to be linked with user's real name.
- **Password**: A secret word chosen carefully by the user that will have to be introduced each time he wants to access the site.
- **Sex**: User's sex.
- **City**: User's current city.
- **Country**: User's current country.
- **E-mail address**: User's valid and working mail account.

Process description:

User will introduce the required information and, after that, the system will perform several tests in order to find out if they fulfill all the preconditions before creating the new account.
Defined preconditions to be fulfilled are:

- Nickname must be unique, not existing already in the system.
- Nickname must have a length of, at least, three characters.
- Both 'Admin' and 'admin' are forbidden as nicknames due to security reasons.
- Since password will have to be introduced twice, both typed words must be equal.
- Password's length must be at least six characters.
- Password must be chosen in such a way there is at least one letter and one number among the characters.
- User's introduced e-mail will have to fit the usual structure 'a@b.c'.
- User's introduced e-mail must work and be owned by him. At the end of the registration process a validation code will be sent by mail to the user. Check 2.2.2 – Validate an account for more information.

All of these preconditions must be satisfied in order to complete successfully the account registration process. If required information has been introduced correctly a user account will be created in the system if not the user will be asked again for correcting the wrong fields.

Outputs:

If all the information is right, a new account will be added to the system and a message will be showed to the user warning he has to activate his account in order to complete registration process. Otherwise the system will ask the user to introduce the wrong information again showing some messages including the reasons previously typed information is not correct.

<u>Validate an account</u>

General description:

After registration process, user's information will be stored in the system and his account will be created, however, he will not be able to log in the site since his account must be validated after its creation.

In order to validate user accounts some information must be sent to user's mail account to ensure he is the owner of that account. Once his mail is checked he will be able to type again that information in the site validating his account's ownership.

Inputs:

A generated string of characters sent to user's mail account.

Process description:

At the end of the registration process, the system will send an email to user's mail account. This mail will contain a multiple-seed generated string of characters and a link besides a message encouraging the user to end the registration process by typing sent string in the page showed following the link.

Once the system has the information the user has introduced through the link, a test will be performed in order to find out if this information fits with the one that was sent to his mail.
If the information was introduced correctly user's account will be activated and he will have access to the site.

Outputs:

A message will be showed informing the user about the situation of his account.

Change account settings – Registration details

General description:

A user must be able to change the information he introduced during the registration process in order to correct or adapt it to his new situation.

Nickname is excluded of this process since it must remain unchanged.

Inputs:

Modified information regarding password, sex, current city, current country or e-mail address.

Process description:

User will introduce the information he would like to update and, after that, the system will perform several tests in order to find out if changed information fulfills all the preconditions and keeps being valid.

Defined preconditions to be fulfilled are:

- Nickname cannot be changed.

If a password modification has been requested:

- He must be required to type current password in order to avoid another person to take advantage of a lonely computer or unclosed session.
- Since password will have to be introduced twice, both typed words must be equal.
- Password's length must be at least six characters.
- Password must be chosen in such a way there is at least one letter and one number among the characters.
-

If user's e-mail address has been modified:

- User's introduced e-mail will have to fit the usual structure 'a@b.c'.

If introduced information fulfills the established requirements, pertinent operations to update it will be performed. If not, no change will be performed.

Outputs:

Besides updating user’s account information when it is necessary, a message will be displayed to inform about the situation of the update operations he requested.

Change user details - User description

General description:

Each user will be able to write a short description in order to let other users know more about his life and interests. This description will be available in each user profile.
Developed system must provide a way users can read and modify this description.

Inputs:

A short description written by the user.

Process description:

Once a user has finished writing his description and the system receives it, proper operations will be performed in order to store this information. There are no special requirements the text must fulfill.

Outputs:

A confirmation message will be displayed for the user to know his description has been uploaded successfully.

Change user details – User avatar

General description:

Each user will be able to upload an image or avatar related to his nickname to enrich his virtual identity.
Developed system must provide a way users can choose an image owned by them and upload it to the system’s server. This procedure will be always available in order to allow several avatar changes during the time.

Inputs:

An image owned by the user.

Process description:

Once a user has selected an image in his computer to be uploaded and the system receives it, proper operations will be performed in order to store the file.

Some restrictions or extra operations, related to pictures' size or dimensions could be added during implementation or validations phases in order to ensure system's stability and correct behavior.

Outputs:

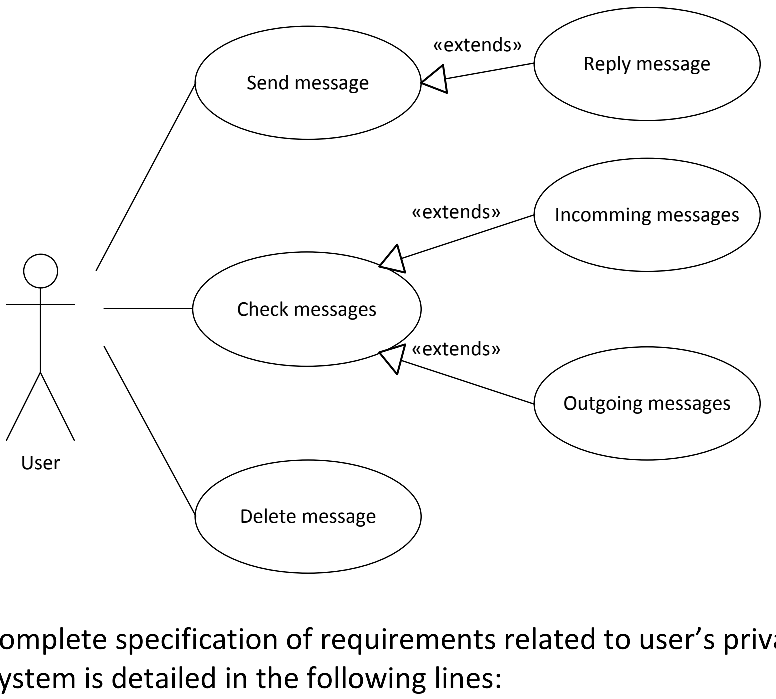
Selected avatar will appear in user's profile. System will display an informative message for the user to know his request was completed successfully if it is necessary.

User private message system

General description:

Since social networking main aim is to interconnect users and allow them to interact among them the development of any kind of private message system is required.

Each user will be able to send, receive and manage his messages by accessing his account. Messages involved in the communication process must remain private, they will be only readable by sender and recipient.



A complete specification of requirements related to user's private message system is detailed in the following lines:

Send a message

General description:

A user must be able to send a message to any other user of the system.

Inputs:

A message is composed of three fields to be filled in before it is sent:

- **Recipient**: Any other user of the system.
- **Subject**: As a title, sentence summarizing message's aim or content.
- **Body**: Whole message to be sent.

Process description:

When the system receives the information defined restrictions must be checked:

- Recipient field is mandatory, an existing user must be specified in this field.
- Either subject or body fields must be filled in. Both messages without subject and messages without body are allowed, besides the common ones.

If these conditions have been fulfilled proper operations will be performed in order to send the message to recipient's inbox.

Outputs:

A message will be showed informing the user about the situation of the action he requested.

<u>Reply a message</u>

General description:

A user must be able to reply a message he has been sent.

Inputs:

Even though a common message is composed of three fields in this case only one of them will have to be filled in by the user.

- **Recipient**: Sender of the message being replied.
- **Subject**: Subject of the message being replied.
- **Body**: Reply to be sent.

Process description:

When the system receives the information proper operations will be performed in order to send the reply to recipient’s inbox. No extra restrictions have been defined.

Outputs:

A message will be showed informing the user about the situation of the action he requested.

Check incoming messages

General description:

Users will be able to access their own inbox in order to check whether they have been sent any message or not.

Inputs:

This action has not got any input.

Process description:

Once the operation is authorized, storage system will be accessed and incoming messages will be displayed.
There is one restriction related to the way replied messages must be displayed in the site.

- A message chain must be managed as a single message. Once a message has been replied only the last one of its replies will be showed in incoming messages.

In order to speed up the access to the inbox, only a defined amount of the last received messages should be showed at once. There should be any navigation or filtering facilities in order to ease the reading and management of previously received messages.

Outputs:

A list of fixed size (f.i. 10 units, 15 units) messages.

Check outgoing messages

General description:

Users will be able to access their own outbox in order to check the messages they sent.

Inputs:

This action has not got any input.

Process description:

Once the operation is authorized, storage system will be accessed and outgoing messages will be displayed.
In order to speed up the access to the outbox, only a defined amount of the last sent messages should be showed at once. There should be any navigation or filtering facilities in order to ease the reading and management of previously sent messages.

Outputs:

A list of fixed size (f.i. 10 units, 15 units) messages.

<u>Delete a message</u>

General description:

Users will be able to delete messages they do not want to keep in their inbox.

Inputs:

Message or list of messages to be deleted.

Process description:

Once the operation is authorized, ensuring the requester of the action is either sender of recipient of current message, storage system will be accessed and selected message/s will be deleted.
System will be built in such a way sender/recipient can keep a message that was already removed by recipient/sender.

Outputs:

A message will be showed informing the user about the situation of the action he requested.

<u>Mark a message as read</u>

General description:

Users will be able to mark messages as read in order to keep them but avoid being noticed again about their reception in the future.

Inputs:

Message or list of messages to be marked.

Process description:

Once the operation is authorized, storage system will be accessed and selected message/s will be marked as read.

Outputs:

A message will be showed informing the user about the situation of the action he requested.

Manage several messages at the same time

General description:

In order to save time and avoid repetitive operations users will be able to apply available operations, as the ones described previously, to a selected group of messages. These operations will be performed at once.

Inputs:

Selected messages.

Process description:

Process description is related to requested operation.

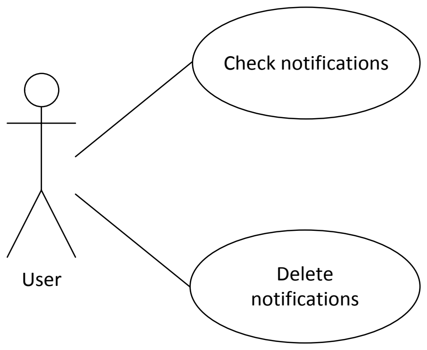
Outputs:

Output behavior depends on chosen operation.

Notification system

General description:

Sometimes a communication mean between system and users will be required. In these situations a notification system will be used.
While private messages are a bidirectional way of communication, notifications are unidirectional. System will be always the sender in a notification and common users will be always recipients.



This system could be used in many different cases but from our prototype’s approach only friend requests and friend confirmations will travel through this channel.

Check notifications

General description:

Users will have access to their own notification box where they will be informed about incoming friendship requests or friendship confirmations.

Inputs:

This action has not got any input.

Process description:

Once the operation is authorized, storage system will be accessed and user’s notifications will be displayed.
In order to speed up the access to the inbox, only a defined amount of the last received notifications should be showed at once. There should be any navigation or filtering facilities in order to ease the reading and management of previously received notifications.

Outputs:

A list of fixed size (f.i. 10 units, 15 units) notifications.

Delete notifications

General description:

Users will be able to delete remaining notifications they do not want to keep in their inbox.

Inputs:

A notification or a list of notifications.

Process description:

Once the action is authorized, selected notifications will be deleted.

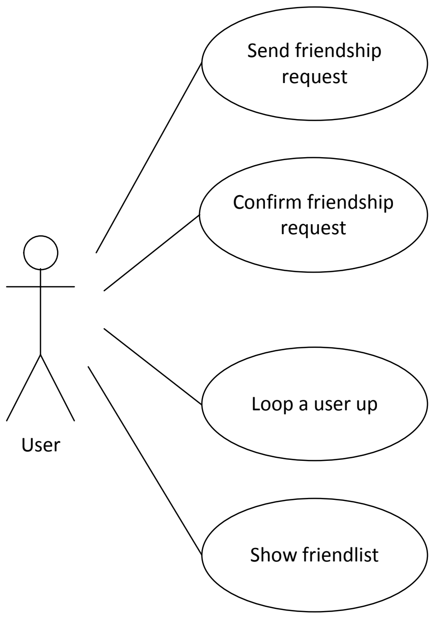
Outputs:

Selected notifications will disappear from user's notification box.

Friend networks

Creation and management of friend networks are the first aim of social networking besides providing communication mechanisms among them in order to reach a real interconnection.

Adding users as friends will keep them in a closer level, they will be easily accessible and their updates will be noticed soon.



On the other hand, as the amount of users increases keep users privacy becomes more and more important. If privacy levels are implemented in the future only user friends will be able to access his information and follow his actions in the site.

Send friendship request

General description:

Users will be able to establish their own friend network by adding other users as friends. The first step necessary to add another user as a friend will be sending him a friendship request.

Inputs:

Recipient of the friendship request.

Process description:

This system will work in the same way it does in current social networks. When a user is interested in having any other user as a friend, he will send a friend request to him. After that he will have to wait until the recipient accepts his request confirming their friendship relationship.

Outputs:

System will send a notification to the requested user once a friend request has been processed allowing him to accept requested friendship relationship.

<u>Confirm friendship request</u>

General description:

Users will be able to establish their own friend network by adding other users as friends. Once a friend request has been received, requested user must finish the linking process by confirming his received request.

Inputs:

Friendship confirmation command.

Process description:

-

Outputs:

System will send a notification to the user who request having the friendship relationship confirming that has been already established.

<u>Look up a user</u>

General description:

Users will be able to look other users up in the system in order to manage and expand their friend networks.
In order to ease this process several filtering options will be provided. Some user characteristics as name, sex, current city or current country will be available for this purpose.

Inputs:

Nickname, sex, current city and/or current country are used as filtering options.

Process description:

According to the filtering options selected by the user, searching operations on system storage platform will be commit.
There could be some restrictions regarding privacy at this point, some users could choose remaining hidden so some extra privacy rules could be discussed. Privacy settings have not been included in this prototype but this could be an interesting area to expand in future modifications.

Outputs:

List of suitable users according to defined filtering settings.

<u>**Show friend list**</u>

General description:

Users will be able to check their own friend list. They will be able to check any other user friend list as well.
If some privacy settings were defined in the future this statement should be revised.

Inputs:

Name of the user whose friend list will be showed.

Process description:

Once the operation is authorized, storage system will be accessed and user's friend list will be displayed.
In order to speed up the access, only a defined amount of user friends should be showed at once. There should be any navigation or filtering facilities in order to ease this process.

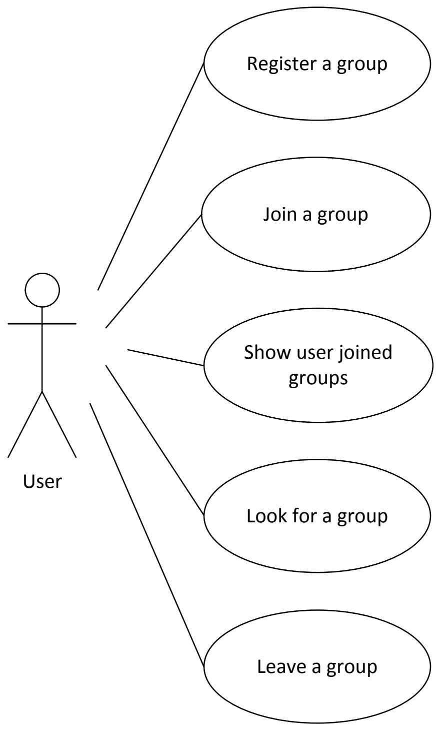
Outputs:

Selected user complete friend list.

Groups of users

Groups as gatherings of people sharing characteristics or interests are common in real life.

In order to encourage users to establish connections among them and expanding their friend networks with people they could be interesting in knowing a virtual group system will be developed.
To ease communication among the participants of each group a shared wall of posts will be implemented.



This is not a new approach and several examples can be found in the most important social networks as Facebook or twitter.

Register a group

General description:

Users will be able to create their own groups devoted to a topic or area they would like to find people interested in.

Inputs:

Group name, a short description and an optional picture to be showed.

Process description:

Once all the information is collected, if there is not an existing group with the same name, requested group will be created. Henceforth other users will be able to apply and become participants of this group.

Groups created within our developed prototype will remain public although some privacy settings could be added in the future.

Outputs:

A new group will be created. User who requested it will be displayed message informing about the situation of his request.

<u>Join a group</u>

General description:

Users will be able to join groups already created in the system. This way they will have an easier access to the information available in them.
On the other hand they will be also able to know and exchange information with the users who joined that group previously.

Inputs:

Name of the group users want to join.

Process description:

According to users request proper operations will be commit in order to register the user in the group.
Some notifications could be delivered to the participants of the group in order to let them know about the arrival, however, there could be different strategies to achieve the same goal.*

**User's "What's new" section has been implemented for this purpose.*

Outputs:

User who requests this action will become a member of selected group.

<u>Leave a group</u>

General description:

Users will be able to leave the groups they have joined. Notifications related to that group or group participants will not be available any more.

Inputs:

Name of the group users want to leave.

Process description:

According to users request proper operations will be commit in order to unregister the user in the group.
Selected group will be removed from user's favourite group list. Group notifications will no longer be showed in user's "What's new" page.

Outputs:

User who requests this action will stop being a member of selected group.

Look for a group

General description:

In order to let users know about the groups already registered in the system and allow them to look for the ones they would be interested in join a look up system will be developed.

To ease this process several filtering options will be provided. Some group characteristics as name or short description will be available for this purpose.

Inputs:

Name or short description of the group users want to look up.

Process description:

Once the operation is authorized, storage system will be accessed and a group list will be displayed.
In order to speed up the access, only a defined amount of groups should be showed at once. There should be any navigation facilities in order to ease the checking through all the results found.

Outputs:

A list of groups that match selected filtering settings will be displayed.

Show user joined groups

General description:

Users will be able to check their own joined group list. They will be able to check any other user's as well.
If some privacy settings were defined in the future this statement should be revised.

Inputs:

Name of the user whose group list will be showed.

Process description:

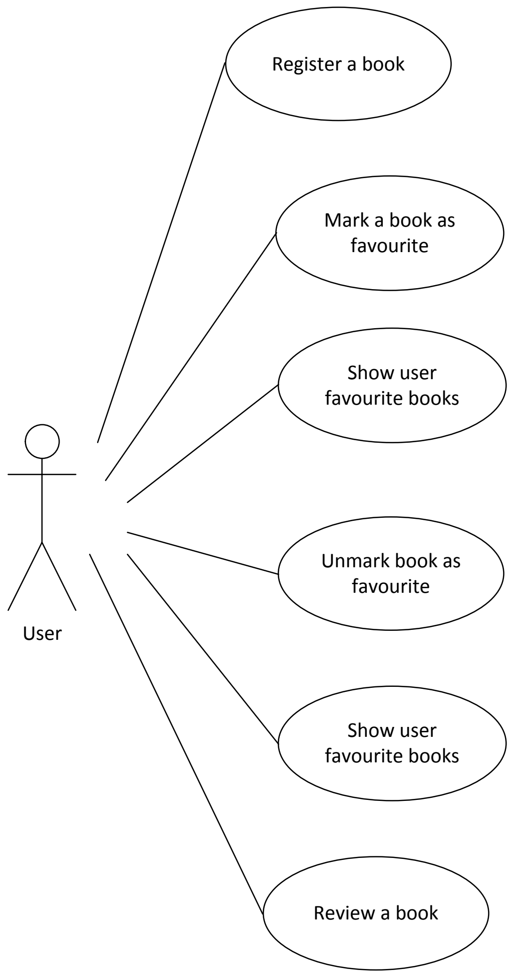
Once the operation is authorized, storage system will be accessed and selected user's group list will be displayed.
In order to speed up the access, only a defined amount of user groups should be showed at once. There should be any navigation or filtering facilities in order to ease this process.

Outputs:

Selected user's complete group list.

Books

As it was described before, Project Alexandria is a prototype of a social networking site devoted to literature. Therefore, besides providing a set of common communication or interconnection mechanisms, some extra functionality make this project different to major and generalist social networks.



Within the site users will be able to link their favourite books to their own profile pages. This way they will be able to find other users with a similar reading taste, gathering with them in groups or pick up advices about which books should we read according to their interests.

<u>Register a book</u>

General description:

Users will be able to register books in the system if they have not been already registered.

A book in developed system is defined according to its main attributes: title, author and genre it belongs to.

Inputs:

Book title, author, genre and an optional picture of its cover, owned by the user who is registering the book.

Process description:

Once all the information is collected, if there is not an existing book with the same title, requested book will be introduced in site's storage system. Henceforth other users will be able to access this book's profile, mark it as favorite or publish a review about their experiences when they read it.

Outputs:

A new book will be added. User who requested its addition will be displayed a message informing about the situation of his request.

Mark a book as favourite

General description:

Users will be able to add their favourite books to their user profile.

Inputs:

Title of the book user wants to mark as his favourite.

Process description:

According to users request proper operations will be commit in order to add requested book to user's list of favourite books.

Outputs:

A new book will be added to user's list of favourite books. User who requested its addition will be displayed a message informing about the situation of his request.

Unmark a book as favourite

General description:

Users will be able to unmark the books they have marked as their favourites.

Inputs:

Title of the book user wants to mark as his favourite.

Process description:

According to users request proper operations will be commit in order to remove requested book from user’s list of favourite books.

Outputs:

Selected book will be deleted from user’s list of favourite books.

Look a book up

General description:

In order to manage and expand their user’s list of favourite books users will be able to look the books up in the system as a previous step to access their book profiles.

Inputs:

A book title, short description or author.

Process description:

Once the operation is authorized, storage system will be accessed and a book list will be displayed according to defined filtering settings.
In order to speed up the access, only a defined amount of books should be showed at once. There should be any navigation facilities in order to ease the checking through all the results found.

Outputs:

A list of books that match selected filtering settings will be displayed.

Show user favourite books

General description:

Users will be able to check their own list of favourite books. They will be able to check any other user’s as well.
If some privacy settings were defined in the future this statement should be revised.

Inputs:

Name of the user whose list of favourite books will be showed.

Process description:

Once the operation is authorized, storage system will be accessed and selected user's list of favourite books will be displayed.
In order to speed up the access, only a defined amount of user books should be showed at once. There should be any navigation or filtering facilities in order to ease this process.

Outputs:

Selected user's list of favourite books.

Publish a book review

General description:

Users will be able to publish their own book reviews. This way, besides sharing their experiences when they read a book, they will be helping other users to choose a book to read according to their reading interests.

Inputs:

The following fields have to be filled in order to review a book successfully:

- Title of the book to be reviewed.
- Review to be published.
- Mark of reviewed book according the following table:

*	Not recommended
**	Not as a first option
***	Interesting
****	Recommended
*****	A must read

Process description:

Once the information has been picked up proper operations will be commit in order to publish the review. There are no extra requirements linked to these operations.
A notification will be showed in every user "What's new" to let them know there is another review available to read.

Outputs:

A new review will be published.

Security and privacy

Authentication

General description:

Project Alexandria is a private virtual community. All the information within the site is reserved to its members therefore a mechanism to authenticate users access into the site must be established.

Inputs:

Nickname and password of the user trying to access the site.

Process description:

Once both words have been collected, introduced password will be compared to the one stored in system's storage platform for current user. If they match access will be granted.

Additional details about authentication process can be checked in Implementation Details section.

Outputs:

If authentication ends up successfully user access to the site will be granted and user's profile will be showed.
If an error arises at the end of the authentication process an access denied notification will be showed.

Authorization

General description:

Besides authentication, an authorization layer must be applied once a user has gained access to the site to check whether requested operation can be done or not by him.

Inputs:

User's nickname

Process description:

System will have to check if current user has enough privileges to commit the operations he is requesting.

Any kind of operation within the site must be subject to some authorization rules which will either allow or deny user requests according to his privileges.

Additional details about authorization process can be checked in Implementation Details section.

Outputs:

If authorization ends up successfully, requested operation will be allowed.
If an error arises at the end of the authorization process an access denied notification will be showed.

Data validation

General description:

Since social networks are systems conceived to have a huge and heterogeneous range of users some precautions should be taken so as to ensure integrity and stability of the system against human errors or unfair behavior.

Users interaction with the site is performed mainly by introducing information, either text or files, to complete their profiles and communicate to each other. However among this introduced information there could be some expressions the system could interpret as commands leading it to its collapse or, at least, to perform some undetected and so unauthorized operations. All system entries will become, therefore, site's Achilles' heel.

In order to prevent problems to arise due to this reason incoming information must be checked before being used or stored.

Inputs:

Users' incoming data.

Process description:

After collecting the information, several operations should be performed so as to revise incoming data, focusing in expressions or suspicious strings which could be potentially dangerous, removing them before data are stored in the system.

Additional details about data validation process can be checked in Implementation Details section.

Outputs:

Revised and corrected or modified, if it is necessary, data to be stored safely.

<u>Accessibility</u>

<u>Language selection</u>

General description:

Achieving as many users as possible is one of the main aims of any social network. This way user network will be able to enlarge and user experience will improve since the amount of choices and possibilities to meet people with the same interests will increase.

In order to expand the range of users of the website, multi-language support will be implemented. Website will be fully translated both in English and Spanish as a first step. More languages may be added in the future.

Inputs:

User's language selection.

Process description:

Since this description would be deeply linked to implementation step and should include some low level details it can be checked in Implementation Details section.

Outputs:

Website content translated according to user language preferences.

2.3.2 Interface requirements

2.3.2.1 User interface requirements

About how a user interacts with the developed application:

- Visual interaction by means of a computer/laptop screen in which the application will be showed.
- User input information will require just a keyboard and a mouse.

There is no need of this kind of devices at the server's side, besides a way to manage and update the application's files.

2.3.2.2 Hardware interfaces

Both classical computers and laptops are able to use our platform, however, this one is not intended to mobile phones. Besides an uncomfortable visual experience some extra problems could arise because of the special configuration, security policies or lack of components usual in mobile web-browsers.

2.3.2.3 Software interfaces

There is no other software interface required rather than a web-browser installed in the computer, however, some visualization problems could arise depending on the web-browser used to get access to the website.

** A successful visualization has been checked under Mozilla Firefox and 1280x800 pixels resolution during the validation test at the end of the development process. Some visualization problems have arisen when Internet Explorer was used as a web-browser to get access to the site.*

2.3.3 Efficiency requirements

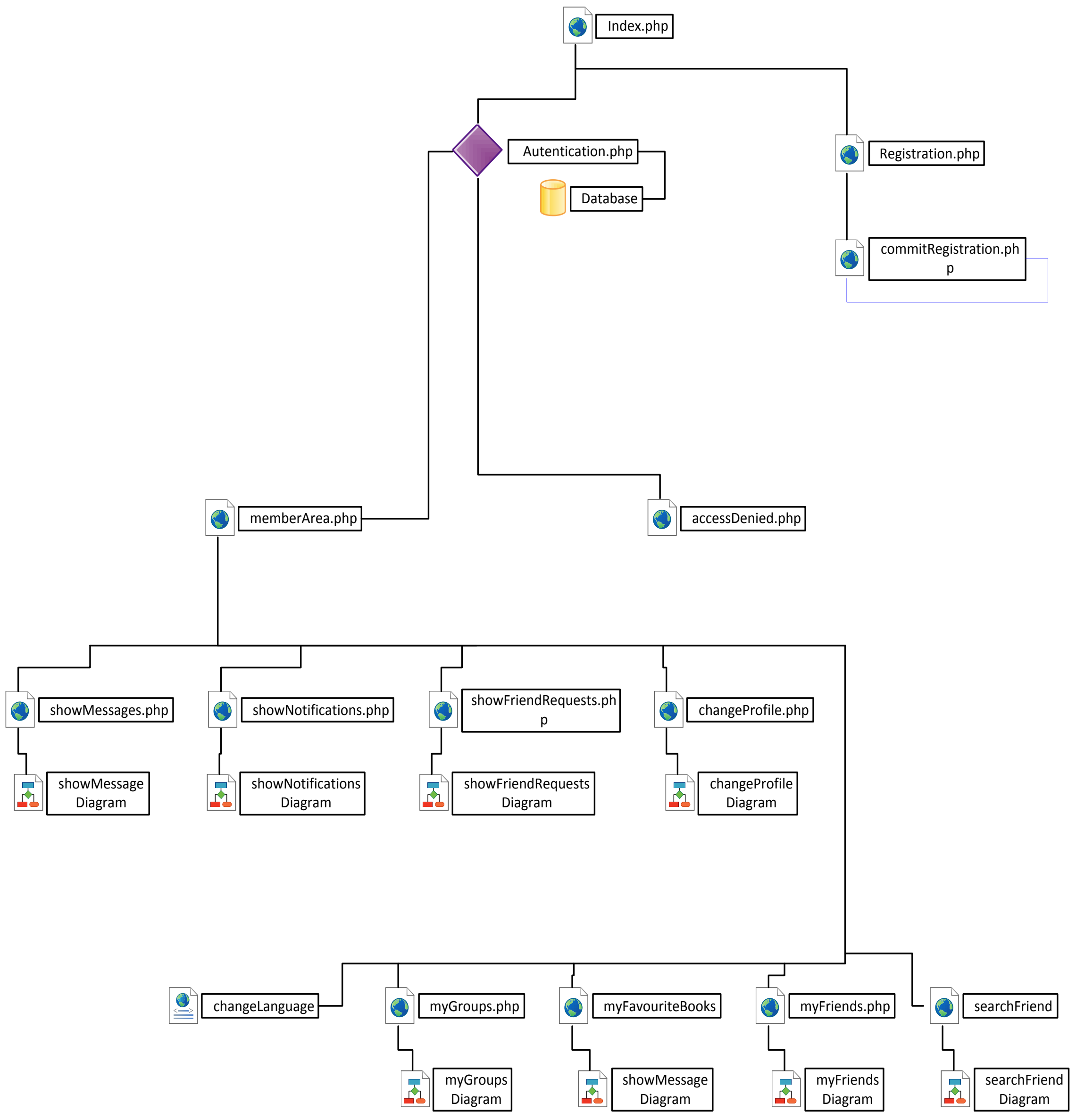
No directly related efficiency requirements have been set after the analysis phase besides expecting a correct behavior of the servers and components required to run the application successfully.

3. Analysis of the developed application

After setting the requirements that must be fulfilled by the prototype, the next step in the process would be the definition of a valid application structure able to reach that goals.

In the following pages the complete structure of the site will be explained by the use of flow diagrams. Since the application complexity level is too high, the complete diagram has been split in several sub-diagrams.

Site main structure



In this first diagram the main structure of the site is presented. While the action flow can be checked in the upper diagram, a brief explanation of the showed files is provided in the next lines:

- **Index.php**: Unique gateway to gain access to the site, must be accessible to all the users, either they are registered or unregistered ones. Depending on your situation you can get access to the site or register a new account.

- **Register.php**: Page whose aim is to collect all the required information about the user that wants to register a new account. As it was stated before, during the definition of requirements' phase a username, password, sex, date of birth and current city and country are required to complete the registration.

- **CommitRegistration.php**: This page receives the information coming from **register.php** and performs both the validation of it correctness and the registration of the user in the site's database if all the requested information was introduced correctly. If some errors have been found, this page will let the user know about what is wrong and will display again the proper forms to collect the invalid information again. This process will be repeated until all the information is collected properly.

- **Authentication.php:** Server side script in charge of checking whether a user has enough privileges to access the site or not. A registered account that has not been validated by its owner will not be allowed to get access to the site.

- **AccessDenied.php:** This page is reached each time an invalid operation has been requested. If the authentication process has not been completed successfully, the script in charge of performing that procedure will redirect the user to this page.

- **MemberArea.php:** Main page for incoming users. User information (profile) is displayed here.

  In the left side user's menu will be displayed. Within it user's profile picture will be shown besides information about new messages, friend requests and notifications together with its corresponding access links.

  In the main area user's personal information is displayed. Besides information introduced during the registration process, a user description, and his favourite books, literary genres and joined groups will be accessible through this page. If he has not added yet the latter information, some links to changeProfile page will be displayed to help him to complete his profile.

  Finally, in the right side a random selection of user friends are shown. Friend suggestions are displayed here as well.

  As it is shown in the diagram above, within this page are displayed the links to get access to different areas of the website, among them:

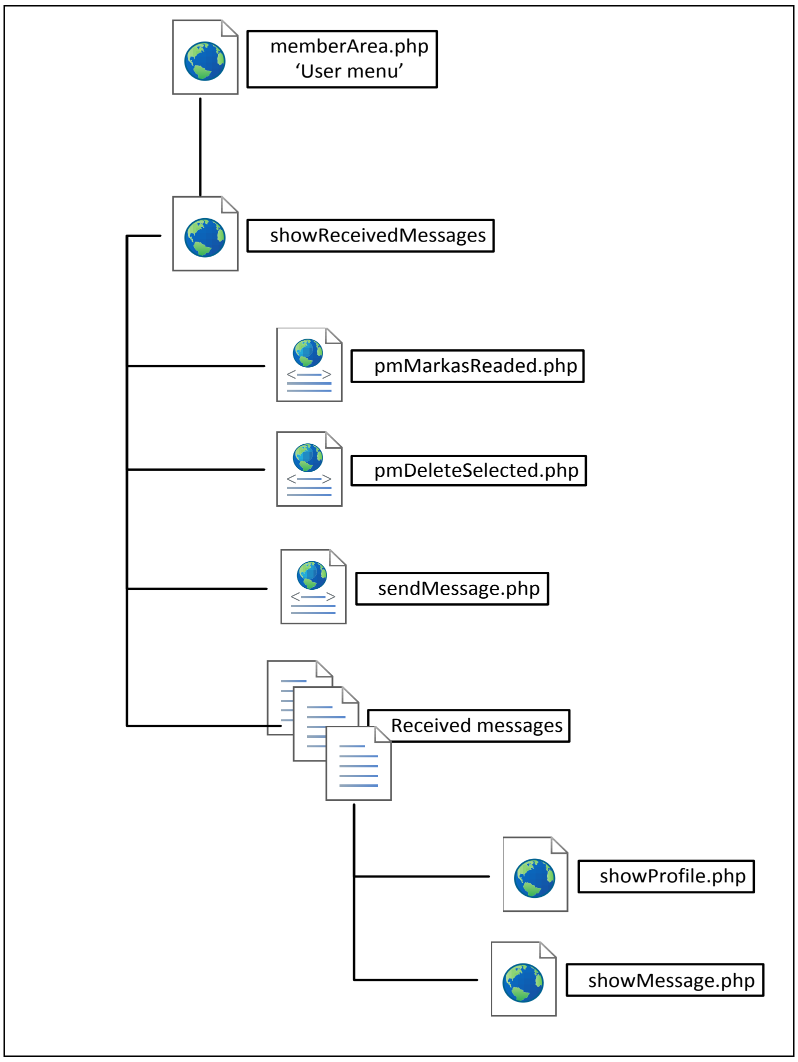
- The pages where private messages, friend requests and notifications are can be checked and managed from user menu.
- The pages where the complete list of user joined groups, favorite books or friends can be checked and filtered.
- Friend profile pages of the friends randomly selected and shown in the right side.
- Page where user information can be updated.

Each one of the pages that were not quoted here (leaves of memberArea.php node) will be explained in the explained together with their own diagrams.

User's menu

User's menu is displayed during almost all the time the user spend logged in the site. While the information displayed in the middle and right areas of the pages use to change depending on the requested content, user's menu, placed in the left side, remains visible because behaves as a bridge to the user to get access to his private messages, notifications and friend requests as soon as they are received.

Checking received private messages.



- **ShowReceivedMessages.php:** This page shows the incoming messages to the user.

  Some tools to manage them or send new ones are displayed as well.

  By clicking on a message the user will be redirected to showMessage.php where he will be able to read its full content.
  Each message contains a link to the sender's profile as well.

- **PmMarkasReaded.php:** Server script that changes selected message/s' status from unread to read.

- **PmDeleteSelected**: Server script that deletes selected message/s' from user's inbox folder. Deleted messages will not be shown again.

- **SendMessage**: Server script in charge of sending a message if all the required fields has been filled in correctly.

- **ShowProfile:** Shows sender's profile.

- **ShowMessage**: Displays the whole message in a different page. Message's information shown in **showReceivedMessages.php** can be cut to fit in the box that must contain it.

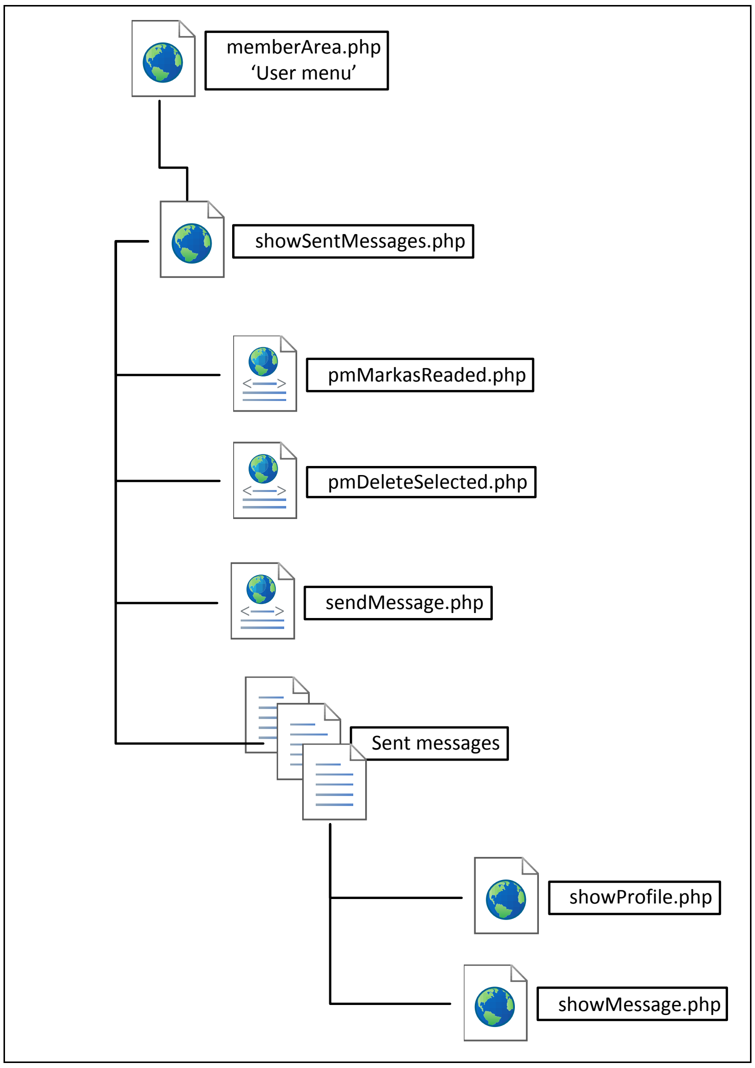
Checking sent private messages.

- **ShowSentMessages.php:** This page shows the outgoing messages to the user.

  Some tools to manage them or send new ones are displayed as well.

  By clicking on a message the user will be redirected to showMessage.php where he will be able to read its full content.
  Each message contains a link to the recipient's profile as well.

- **PmDeleteSelected**: Server script that deletes selected message/s' from user's inbox folder. Deleted messages will not be shown again.

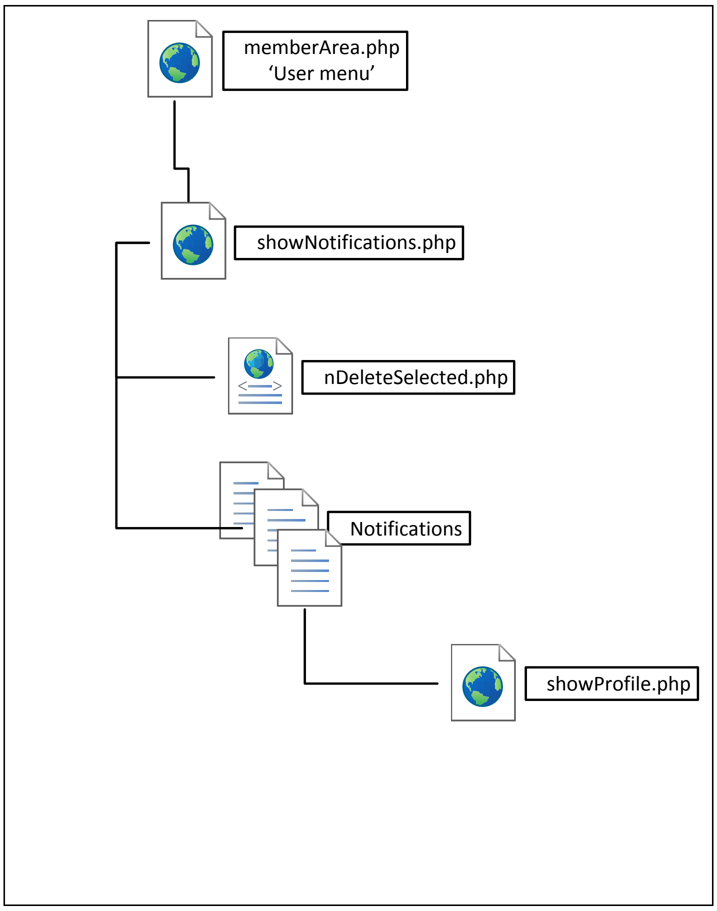


- **SendMessage**: Server script in charge of sending a message if all the required fields has been filled in correctly.

- **ShowProfile:** Shows recipient's profile.

- **ShowMessage**: Displays the whole message in a different page. Message's information shown in **showReceivedMessages.php** can be cut to fit in the box that must contain it.

Checking notifications



- **ShowNotifications:** This page shows the notifications received by the user.

  An option to delete the notifications that have been already read is included as well.

- **nDeleteSelected.php:** Server script that deletes selected notification/s'. Deleted messages will not be shown again.

- **showProfile:** If current notification implies another person (as it occurs when a friend confirmation is shown), it will show his/her profile.

Checking friend requests

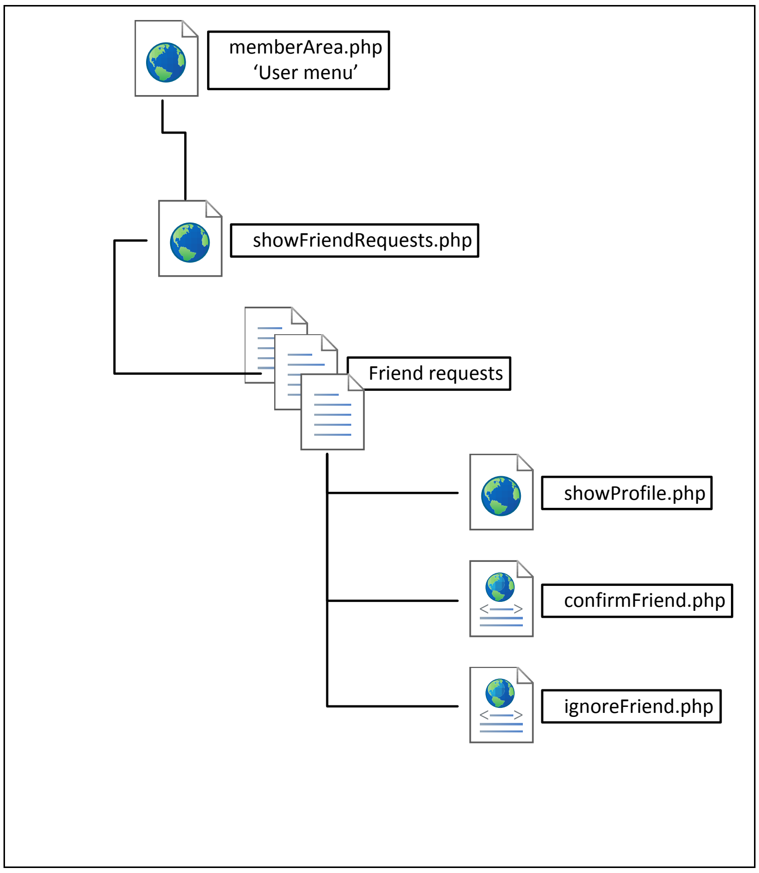
- **ShowFriendRequests.php:** This page shows pending friend requests to the user.

  User will have to choose whether ignoring the request or confirm their friendship.

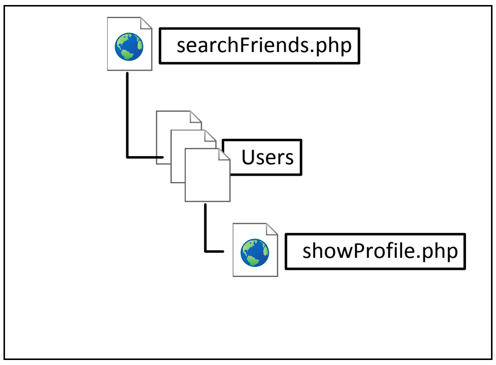
- **ConfirmFriend**: Server script that confirms the relationship between current user and the requester of the friendship.

- **IgnoreFriend**: Server script used when a friendship request must not be attended.

- **ShowProfile:** Shows requester's profile.



Looking for friends



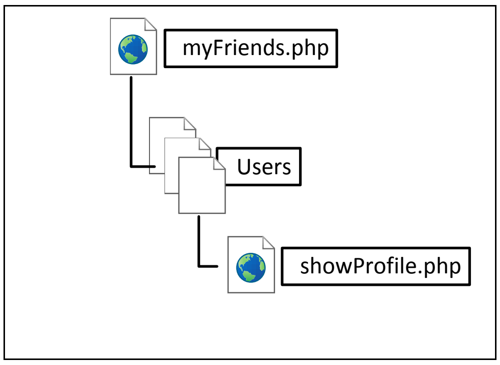
- **SearchFriends.php:** This page performs search operations into database according to some filtering parameters customized by the user and displays the results. Results are shown in groups of ten elements.

- **ShowProfile.php:** Shows selected user profile.

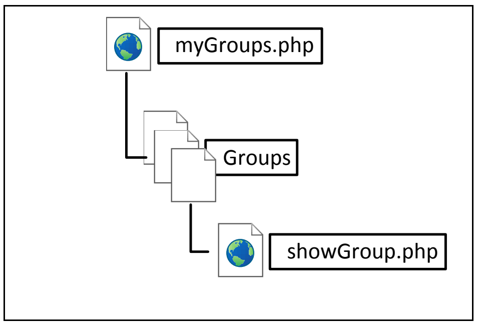
Checking user friends

- **MyFriends.php:** Displays current user friends.
  Shown results can be filtered according to several parameters. Results are shown in groups of ten elements.

- **ShowProfile.php:** Shows selected user profile.



Checking user favourite groups/interests



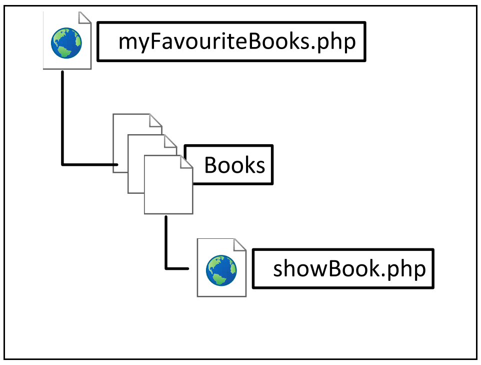
- **MyGroups.php:** Displays current user groups.
  Shown results can be filtered according to several parameters. Results are shown in groups of ten elements.

- **ShowGroup.php:** Shows selected group profile.

Checking user favourite books

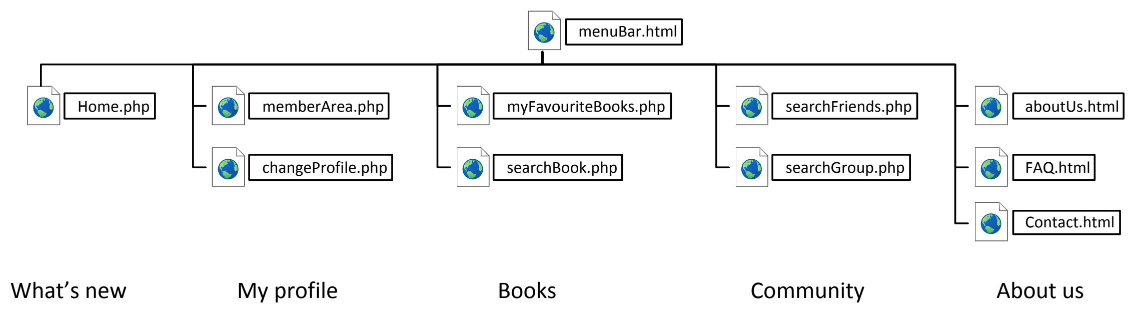
- **MyFavouriteBooks.php:** Displays current user favourite books.
  Shown results can be filtered according to several parameters. Results are shown in groups of ten elements.

- **ShowBook.php:** Shows selected book profile.



Site's menu bar

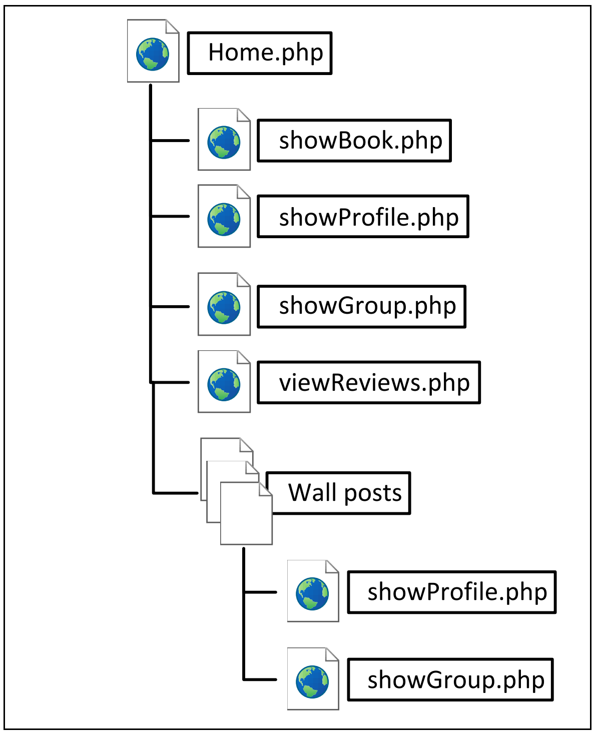
In order to ease user's experience without affecting the site's growth a menu-bar has been added above the content area. The content within this bar is organized and grouped regarding to several categories. The following diagram shows this structure:



What's new

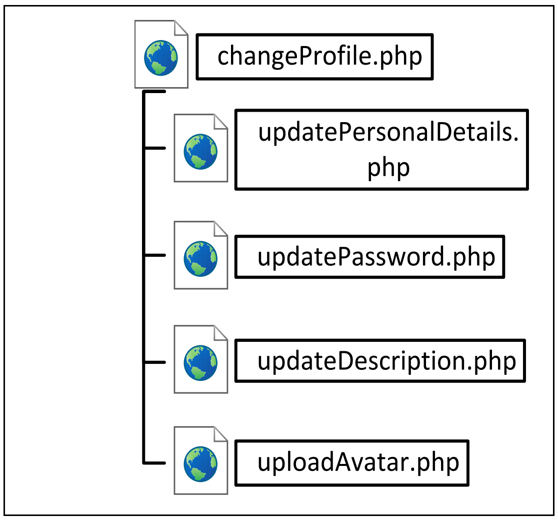
- **Home.php:** This page informs users about the last events happened in the site: last registered users, last groups created, last books registered and last reviews uploaded.
  There will be links to get access to all the shown elements (**showBook.php**, **showProfile.php**, **showGroup.php**, **viewReviews.php**)

  Last posted wall-posts from groups where current user is a member will be displayed as well.



My profile

Updating user personal information



- **changeProfile.php:** Page displaying some forms showing user's current data.
  From this page personal details and account's password can be changed.
  There is also a special tool to upload user's profile picture.

- **updatePersonalDetails.php**: This page receives the information coming from **changeProfile.php** and performs both the validation of it correctness and the update of user's personal details (sex, birthday, city, country) if all the requested information was introduced correctly.

- **updatePassword.php**: This page receives the introduced passwords (current and new ones) coming from **changeProfile.php** and performs both the validation of it correctness and the update of user's password.

- **updateDescription.php:** This page receives the introduced description coming from **changeProfile.php** and stores it in the database.

- **uploadAvatar.php:** Server script in charge of getting and storing in the server the picture selected by user in **changeProfile.php** proper form.

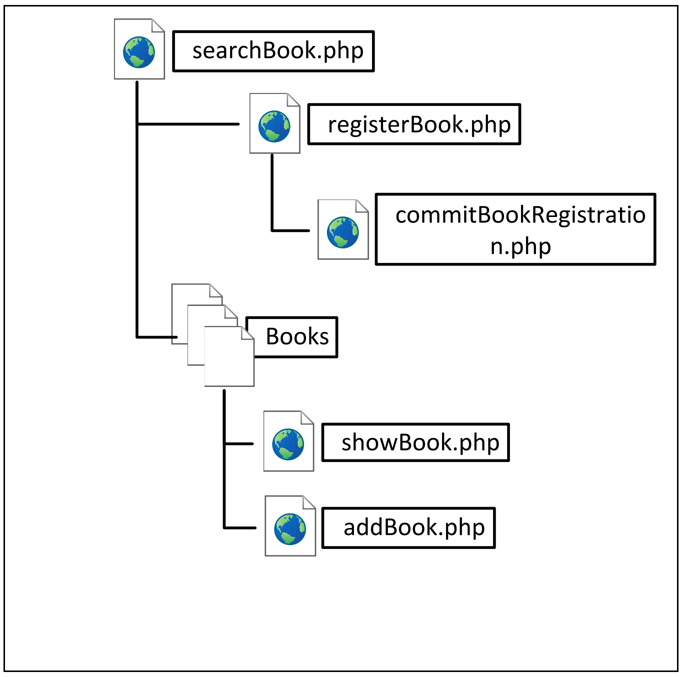
Books

Managing books

- **SearchBook.php:** Displays system's existing books in groups of ten elements. Each element includes a link to its main page (**showBook.php**) and an option to add it to user's favorite book list (**addBook.php**)

- **ShowBook.php:** Shows selected book's main page.

- **AddBook.php:** Add current book to user's favourite book list.



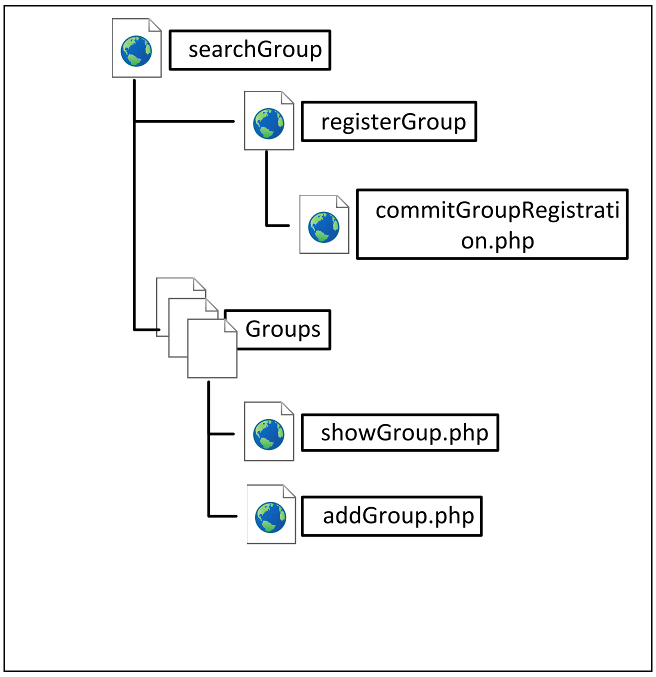
- **RegisterBook.php:** Allows users to register new books in the system by filling in the requested information.

- **CommitBookRegistration.php:** This page receives the information coming from **registerBook.php** and performs both the validation of it correctness and the registration of the book in the site's database if all requested information was introduced correctly. If some errors have been found, this page will let the user know about what is wrong and will display again the proper forms to collect the invalid information again. This process will be repeated until all the information is collected properly.

-

Community

<u>Managing groups</u>



- **SearchGroup.php:** Displays system's existing groups in groups of ten elements. Each element includes a link to its main page (**showGroup.php**) and an option to add it to user's favorite group list (**addGroup.php**)

- **ShowGroup.php:** Shows selected group's main page.

- **AddGroup.php:** Add current group to user's favourite group list.

- **RegisterGroup.php:** Allows users to register new groups in the system by filling in the requested information.

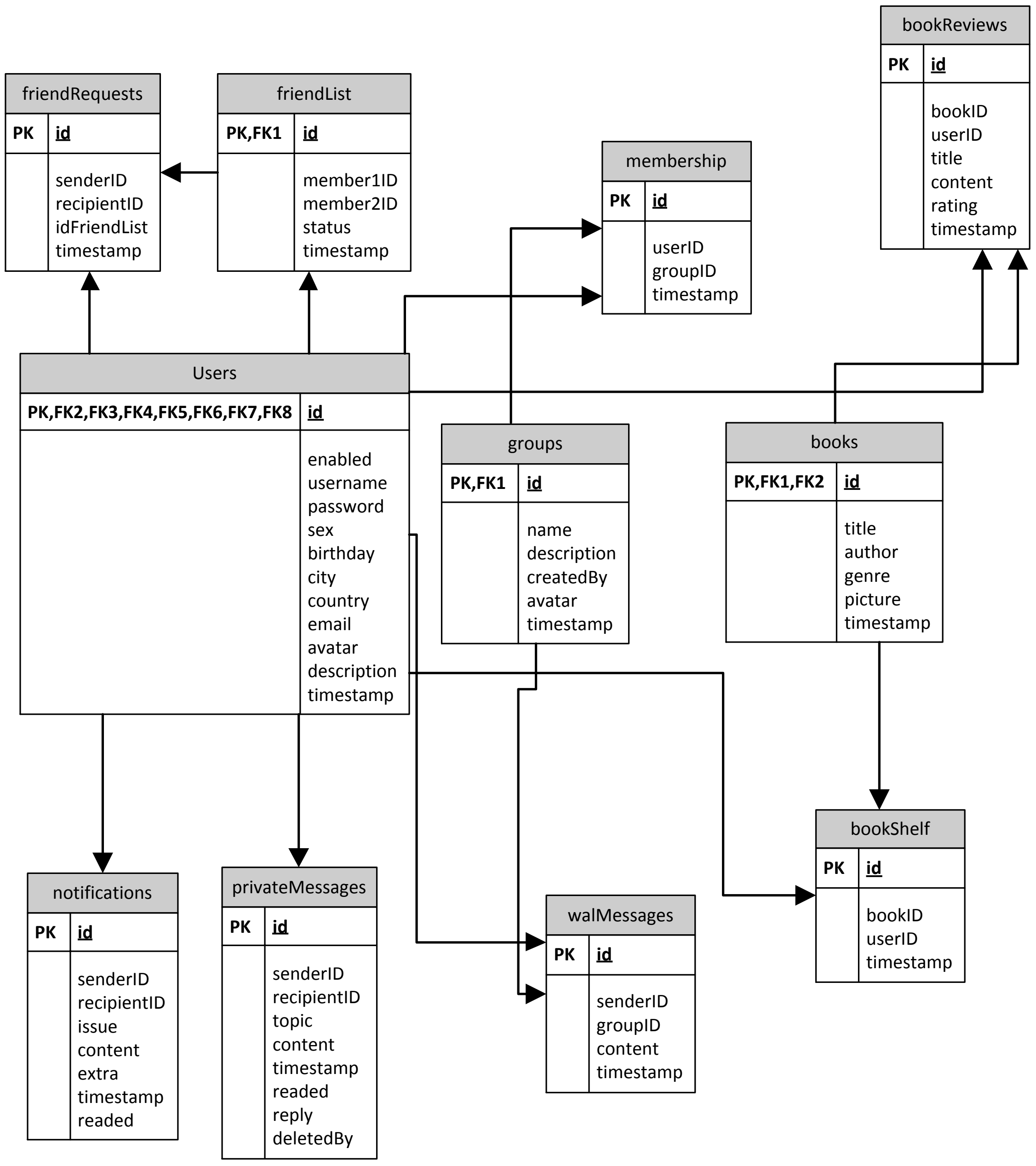
- **CommitGroupRegistration.php:** This page receives the information coming from **registerGroup.php** and performs both the validation of it correctness and the registration of the group in the site's database if all requested information was introduced correctly. If some errors have been found, this page will let the user know about what is wrong and will display again the proper forms to collect the invalid information again. This process will be repeated until all the information is collected properly.

About us

- **AboutUs.html:** Shows information related to the site.
- **FAQ.html:** Frequent asked questions document.
- **Contact.html:** Staff contact directions.

4. Data model

In the following diagram is shown the data model used in Project Alexandria prototype.



Description of database's tables:

Table: users

Stores personal information of the site's users.

Field identifier	Type	Default value	Description
Id	Integer	-	**Primary key**
Enabled	Integer	0	Flag. Shows whether an account has been activated (1) or not (0)
Username	Char	-	User's login name
Password	Char	-	User's password
Sex	Enum	-	Enum('male','female')
Birthday	Datetime	NULL	User's birthday
City	Char	-	User's city
Country	Char	-	User's country
Email	Char	-	User's email
Avatar	Char	Default_avatar	Location of user's profile picture
Description	Char	-	User's personal description
Timestamp	Datetime	-	User's registration date and time

Table: wallMessages

Stores group's wall-posts

Field identifier	Type	Default value	Description
Id	Integer	-	**Primary key**
SenderID	Integer	-	**Foreign key:** Users – Id of the sender
groupID	Integer	-	**Foreign key:** Groups – Id of the group in which current message was posted
Content	Char	-	Content of the message
Timestamp	Datetime	-	Posting date and time

Table: privateMessages

Stores user's private messages

Field identifier	Type	Default value	Description
Id	Integer	-	**Primary key**
senderID	Integer	-	**Foreign key:** Users – Sender's id
recipientID	Integer	-	**Foreign key:** Users – Recipient's id
Topic	Ingeter	-	Subject of the message
Content	Integer	-	Content of the message
Timestamp	Datetime	-	Sending date and time

Read	Integer	0	Flag – Shows whether a message has been read (1) or not (0)

Reply	Integer	-1	Flag – Shows message status:
			Value= -1 -> Non replied message
			Value= -2 -> Replied message
			Value> 0 -> Reply of a message

deletedBy	Integer	-1	**Foreign key**: Users **–** ID of the first user (sender/recipient) who deleted the message.

Table: notifications

Stores system notifications

Field identifier	Type	Default value	Description
Id	Integer	-	**Primary key**
recipientID	Integer	-	**Foreign key:** Users - Recipient's id
Issue	Enum	-	Enum('friend_request','friend_confirmation')
Content	Char	-	Notification's content
Extra	Char	-	Extra field if required
Timestamp	Datetime	-	Sending date and time
Readed	Integer	0	Flag – 0: non read – 1: read

Table: membership

Stores relations between users and groups.
Each record can be considered as a subscription of a user to a group.

Field identifier	Type	Default value	Description
Id	Integer	-	**Primary key**
userID	Integer	-	**Foreign key:** Users - Member id
groupID	Integer	-	**Foreign key:** Groups - Group id
Timestamp	Datetime	-	Sending date and time

Table: groups

Stores registered groups/interests.

Field identifier	Type	Default value	Description
Id	Integer	-	**Primary key**
Name	Char	-	Group name
Description	Char	-	Group description
createdBy	Integer	-	**Foreign key:** Users
Avatar	Char	Default_avatar	Location of group avatar

Timestamp	Datetime	-	Group registration date and time

Table: friendRequests

Stores generated friend-requests.
Temporary information, record is deleted when a friendship relationship is confirmed.

Field identifier	Type	Default value	Description
Id	Integer	-	**Primary key**
Sender	Integer	-	**Foreign key:** Users – Requester id
Recipient	Integer	-	**Foreign key:** Users – Recipient id
idFriendList	Integer	-	**Foreign key:** friendList
Timestamp	Datetime	-	Friendship requesting date and time

Table: friendList

Stores pairs users.id:users.id. Two records are needed for each single friendship relationship.

Field identifier	Type	Default value	Description
Id	Integer	-	**Primary key**
Member1ID	Integer	-	**Foreign key:** Users
Member2ID	Integer	-	**Foreign key:** Users
Status	ENUM	‘pending’	Enum(‘pending’,’confirmed’)
Timestamp	Datetime	-	Friendship confirmation date and time

Table: bookShelf

Stores generated pairs users.id:books.id. A record is created each time a user mark a book as favourite.

Field identifier	Type	Default value	Description
Id	Integer	-	**Primary key**
bookID	Integer	-	**Foreign key:** Books – Favourite book
userID	Integer	-	**Foreign key:** Users – User’s id
Timestamp	Datetime	-	Adding date and time

Table: groups

Stores registered books in the system.

Field identifier	Type	Default value	Description

Id	Integer	-	**Primary key**
Title	Char	-	Book title
Author	Char	-	Book author
Genre	Char	-	Book genre
Picture	Char	Default_cover	Location of book's cover picture
Timestamp	Datetime	-	Registration date and time

Table: bookReviews

Stores book reviews made by users.

Field identifier	Type	Default value	Description
Id	Integer	-	**Primary key**
bookID	Integer	-	**Foreign key:** Books – Reviewed book
userID	Integer	-	**Foreign key:** Users - Reviewer id
Title	Char	-	Review title
Content	Char	-	Review content
Rating	Float	-	User's rating for current book
Timestamp	Datetime	-	Review publishing date and time

5. Implementation

5.1 Technologies used during the coding phase

All technologies included in Project Alexandria were agreed at the beginning of its development.

Although several options were discussed at the beginning of the process only one seemed both feasible and suitable: building a HTML-PHP site.

A lot has been already written about the advantages of using this scripting language instead of any of its main competitors, however, this issue is not among this document's aims.

As for database management Mysql was the most suitable solution according to selected scripting language.

On the other hand, in order to save useless connections to the site, it was decided to perform some data validations at client-side. This way all forms filled in wrongly will not be submitted until its information is properly introduced. For this purpose some JavaScript scripts have been added to site's pages. To avoid unexpected behavior due to a JavaScript failure during data validation, all incoming data will be validated once again at server-side.

Finally, to improve a simple html site's design, cascading style sheets have been used. Site's base style sheet has been provided by **http://www.pixabella.com/**.

5.2 Implementation details

Language detection/selection

As it was stated in previous chapters of this document, Project Alexandria has included language settings among their functionalities, being available both in English and Spanish.

Each time a non logged user access the site, a language detection is done in order to adapt site's information to a language comfortable for him according to his preferences. For this purpose HTTP headers are checked and visualization language is set depending on them.

The following code shows how described action takes place:

```
if($_SERVER['HTTP_ACCEPT_LANGUAGE']!='')
{
            $languages =
explode(",",$_SERVER['HTTP_ACCEPT_LANGUAGE']); (1)
            $lang = 'en'; (2)
            for($i=0;$i<count($languages);$i++) (3)
            {
                  if(substr($languages[$i],0,2)=="es")
                  {
                        $lang = 'es';
                        break;
                  }
            }
            require('lang/'.$lang.'/registration.php');
}
```

Code from registration.php

1. Information is gotten from HTTP header "Accept Language". Content is stored in an array using "," as delimiter to split read string of characters.
2. English is set as default language
3. If a language appearing in HTTP Accept language is available system will switch to it becoming visualization language. Only Spanish is available as an alternative.

This system has been added to save users from configuring their language preferences, however, there could be some especial circumstances under which due to an inaccurate detection or user preferences the result of this detection does not fit user interests. In these cases a manual language selection can be used, being available in every page of the website.

As many links as available languages are in the site will be displayed at the top part of it, below site's banner.

```
<a class="postLink"
href="http://localhost/changeLanguage.php?lang=en"
style="display: inline-block;">ENG</a>

<a class="postLink"
href="http://localhost/changeLanguage.php?lang=es"
style="display: inline-block;">ESP</a>
```

Code from navigationBar.html

By clicking on the links proper operations will be performed to change current language to the chosen one.

```
If($_GET['lang']=='es')
$_SESSION['lang'] = 'es'; (1)
else $_SESSION['lang'] = 'en';
```

Code from changeLanguage.php

(1) Once a language has been set for a logged user this information will be stored in a variable within user's session. This way its selection will remain until the end of this one.

Once it has been explained how a user can change the variable which controls visualization language only the explanation about how the site manages this situation and shows the information in the proper language in each case is left. In order to achieve it without having to rewrite anything from source code each time a new language is added, php constants (1)(2) have been used in the places where a translated text should appear.

```
<?php echo $username.", ";
  if($sex=='male') echo (1)constant('profile-header-
sex-male');
  else echo (2)constant('profile-header-sex-female');
```

Code from memberArea.php

By replacing these constants with the expressions in the proper language all possible versions of the site can be obtained without having to adapt code for each case.

On the other hand a set of text files containing the translations of these constants has been created. This way, for each php or html file containing constants which should be substituted, there are as many text files containing these translated substitutions as languages are supported by website.

```
//Middle content
define('profile-header-sex-male','male');
define('profile-header-sex-female','female');
```

Code from /lang/en/memberArea.txt

```
//Middle content
define('profile-header-sex-male','usuario');
define('profile-header-sex-female','usuaria');
```

Code from /lang/es/memberArea.txt

Registering an account

Before being granted access to Project Alexandria users must register their personal account in the site.

As a first step a form must be filled in by applicants.

Once data are collected some tests are performed in order to validate all the restrictions established during analysis of requirements phase. Incoming information must fulfill all of them prior creating requested account.

According to "**Analysis of requirements**" document, these are the established restrictions and the code which ensures its fulfillment:

"Nickname must be unique, not existing already in the system"
"Both 'Admin' and 'admin' are forbidden as nicknames due to security reasons"

These two restrictions are checked at the same time in the following code. After querying site's database about user's chosen nickname ($username):

- Its uniqueness would not be ensured if site's database had found any match after querying USERS table (1).
- On the other hand, if no match was found but $username is equal to a forbidden word process must me stop as well (2).

```
$SQL = "SELECT * FROM users WHERE users.username=\"$username\"";
$result = mysql_query($SQL) or die('Error: '.mysql_error());
$row = mysql_fetch_row($result);
if(mysql_num_rows($result)!=0 (1) || $username==admin (2)||
$username==Admin (2))
{
      $e_login = constant('error-repeated-username');
      $username="";
      $n_errors++;
}
```

Code from commitRegistration.php

“Nickname must have a length of, at least, three characters”

```
if(strlen(trim($username))<2)
      {
            $e_login = constant('error-wrong-username');
            $username="";
            $n_errors++;
      }
```

Code from commitRegistration.php

“Since password will have to be introduced twice, both typed words must be equal”

```
if($passwd1!=$passwd2)
      {
            $e_passwd = constant('error-password-unequal');
            $passwd1="";
            $passwd2="";
            $n_errors++;
      }
```

Code from commitRegistration.php

“Password’s length must be at least six characters” (1)
“Password must be chosen in such a way there is at least one letter and one number among the characters” (2)

```
if(strlen($passwd1)<6 (1)|| preg_match('[^0-9]',$passwd1) (2))
      {
            $e_passwd = constant('error-password-length');
            $passwd1="";
            $passwd2="";
            $n_errors++;
      }
```

```
```

Code from commitRegistration.php

About this code just a comment is required, **preg_match()** is a php function whose aim is ensuring the syntax of a text (second parameter) fulfills a regular expression (first parameter)

“User’s introduced e-mail will have to fit the usual structure ‘a@b.c’”

Again, **preg_match()** function is used to ensure introduced e-mail direction fulfills expected structure.

```
if(!preg_match("/^([a-zA-Z0-9])+@([a-zA-Z0-9_-])+(\.[a-zA-Z0-
9_-]+)+/", $mail))
        {
                $e_mail = constant('error-wrong-mail');
                $mail = "";
                $n_errors++;
        }
```

Code from commitRegistration.php

If all the previous restrictions have been fulfilled system will create requested user’s account although this one will remain disabled until account validation is completed.

```
$passwd1 = md5($passwd1); (1)
$unixTime = time();
$sqlTime = gmdate("Y-m-d H:i:s", $unixTime);
//Connection to DB
include('dbAlexandria1.php');
include('connect.php');

//Starting a SQL transaction to ensure all operations will be
committed atomically
mysql_query('BEGIN');

//Registering user in DB
$SQL = "INSERT INTO users
(username,password,sex,birthday,city,country,email,timestamp)
VALUES
(\"$username\",\"$passwd1\",\"$sex\",\"$birthday\",\"$city\",\"$
country\",\"$mail\",\"$sqlTime\")";
mysql_query($SQL);
if(mysql_error())
        {
                mysql_query('ROLLBACK');
                die();
        }
```

Code from commitRegistration.php

(1) In order to ensure users privacy, users' password will not be directly stored in database but a checksum of it, This way nobody except its owner will be able to know it and it will remain protected even if database is accessed without authorization.

Validating an account

After completing registration process, account validation is left prior being allowed to access the site. Account validation procedure has been created to fulfill the last one of the registration restrictions defined in "Analysis of requirements" document.

"User's introduced e-mail must work and be owned by him"

In order to fulfill this restriction users should be "encouraged" to check their email accounts so as to get some information they are required to introduce before getting access to the site for the first time.

The following code shows how an email is sent to the users whose accounts have just been created. Among its content a unique string of characters is enclosed. For its generation some user details are used as seed (1), this way there is no need of storing this string in database since it can be easily reproduced knowing the way it can be generated.

```
$to = $mail;
$subject = constant('validation-mail-subject');
$body = constant('welcome')."
".$username."!".constant('validation-mail-
content').md5("alwayslookatthebrightsideoflife".$username.$mail.
$passwd1.$id) (1)." \n\n URL: ".$rootURL."validationForm.php (2))
";
mail($to, $subject, $body)
```

Code from commitRegistration.php

After generated string, a link to the validation page is attached as well (2). Users will be able to access it with no deadline to validate their accounts. Once there they will have to introduce their nickname and the character string they were sent. By reproducing the same generation procedure another character string will be generated and compared to the introduced one, if they match account will be enabled (3) and user will be able to access the site.

```
if($introducedString == $generatedString)
{
  $SQL = "UPDATE users SET users.enabled=1 (3) WHERE
users.username = \"$postLogin\"";
  mysql_query($SQL) or die('Error: ' . mysql_error());
}
```

Code from validateAccount.php

Authenticating a user

According to "**Analysis of requirements**":

"Project Alexandria is a private virtual community. All the information within the site is reserved to its members therefore a mechanism to authenticate users access into the site must be established"

To fulfill this requirement a simple password-based identification system is used each time a user requests an access to the site. This way before allow a user to access the site his password will be requested.

Once information, including nickname and password, is collected an authentication script (**authentication.php**) will be in charge of checking whether introduced password match the one stored in site's database or not.

The following lines show how this checking is performed:

```
$SQL = "SELECT * FROM users WHERE username =\"$postLogin\" (1)
AND password =\"$postPassword\" (2) AND enabled =\"1\" (3)";

$result=mysql_query($SQL) or die('Error: '.mysql_error());

if((mysql_num_rows($result))!=0) //User-pass match has been
found: user is succesfully authenticated
{
  //Successful access code
}
else
{
   header('Location: accessDenied.php') (4);
}
```

Code from authentication.php

Notice that $postLogin and $postPassword contain either nickname or password introduced by user to request access to the site.

As it can be read, besides a nickname (1)-password (2) matching it is required that user's account has been already validated (3). If any of both conditions is not fulfilled user attempt to log in will be denied and an access denied page will be showed (4).

On the other hand, if identification process ends up successfully, proper operations will be performed so as to create a new session under which user will be able to surf through the site without having to retype his password. For this purpose, PHP sessions are used.

```
$_SESSION['username'] = $postLogin;
$_SESSION['userID'] = $row[constant("users::id")];
(6)
session_name($postLogin); (5)

/* Getting user default language */
/* This code was already explained in "Language
detection/selection" section  */
/* … $_SESSION['lang'] = $lang; (7) … */

header('Location: memberArea.php') (8);
```

Code from authentication.php

A session identified by user's nickname will be created (5). Notice PHP sessions allow programmers to store as many variables as they need to keep information till session expiration. These variables are used, for instance, to store users' id number (6) (which is the index of their record in database) or user's language preferences (7). To use this information, **session_start()** command will have to be written at the very beginning of all the documents composing the site.

Once a user has been successfully authenticated he will be redirected to his own profile (8).

Authorizing a user

"Besides authentication, an authorization layer must be applied once a user has gained access to the site to check whether requested operation can be done or not by him"

Authorization procedure, once a session has been established during authentication phase, is reduced to a simple check.

```
session_start();

if(!isset($_SESSION['username']) || $_SESSION['username']=="")
{
      header('Location: accessDenied.php');
}
```

Code from authorisation.php

If no session with user's nickname is found, it means that current user did not identify himself properly and all kind of access must be denied to him. This check is performed each time a user requests access to any of the site's pages by the following expression:

```
require('authorisation.php');
```

Code from memberArea.php

On the other hand, notice that all authenticated users are able to access all the pages composing Project Alexandria, since there is not any kind of users hierarchy. This approach could be useful in bigger or more complicated social networks, with different types of users. Adapting this site to that situation would not be so complicate, an extra variable could be added to each user session containing the level or privilege ring that current user belongs to. Variable's content would be loaded during users authentication phase as it is done with other variables (6)(7).

Changing account details

Each user account will contain information related to its owner. Among customizable details the following ones can be found:

(1) User registration details as sex, birthday, city, country and email.
(2) User password.
(3) User avatar.
(4) User description.

(1) Changing registration details.

This process will be performed in a similar way it was done when data were collected for the first time during user's registration. In fact, defined restrictions are almost the same. The only difference lies on password definition which will be performed in a different section. Splitting the restrictions associated with password updates the following ones remain:

"Nickname cannot be changed"

Website will not allow nickname changes, there will not be any place in "**changeProfile.php**" to request this operation.

"User's introduced e-mail will have to fit the usual structure 'a@b.c'"

Preg_match() function will be used to ensure introduced e-mail direction fulfills expected structure.

```
if(!preg_match("/^([a-zA-Z0-9])+@([a-zA-Z0-9_-])+(\.[a-zA-Z0-
9_-]+)+/", $mail))
      {
            $e_mail = constant('error-wrong-mail');
            $mail = "";
            $n_errors++;
      }
```

Code from updatePersonalDetails.php

Notice that the rest of fields do not require a validation since they are introduced in such a way no errors can arise.

After this check, **updatePersonalDetails.php** will update changed details in database.

```
include('dbAlexandria1.php');
include('connect.php');
$SQL = "UPDATE users SET sex=\"$sex\", birthday=\"$birthday\",
city=\"$city\", country=\"$country\", email=\"$mail\" WHERE
username=\"$username\"";
mysql_query($SQL) or die('Error: '.mysql_error());
mysql_close();
```

Code from updatePersonalDetails.php

Notice that all the fields are included in the query and not only the altered ones as it should be expected. This is due to all variables have been initialized with their correct values so if user did not make any change their correct value will be put back in database.

(2) Changing password.

Users will be able to change their password at any time. Some fields to be filled in will be displayed in **changeProfile.php** for this purpose.

After collecting required information in **changeProfile.php**, **updatePassword.php** will start working.

Current password (1), requested password (2) and requested password retyped (3) will be collected and stored in three variables.

```
(1) $passwd0 = trim($_POST[tb_passwd0]);
(2) $passwd1 = trim($_POST[tb_passwd1]);
(3) $passwd2 = trim($_POST[tb_passwd2]);
```

Code from updatePassword.php

Once done, some conditions must be checked.

(4) **“User must be required to type his current password in order to avoid another person to take advantage of a lonely computer or unclosed session”**

(5) **“Since password will have to be introduced twice, both typed words must be equal”**

(6) **“Password’s length must be at least six characters”**

(7) **“Password must be chosen in such a way there is at least one letter and one number among the characters”**

```
(4) if($aPasswd==(8) md5($passwd0))
{
      (5) if($passwd1!=$passwd2)
      {
            $e_passwd = constant('error-password-
unequal')."<br>";
            $passwd1="";
            $passwd2="";
            $n_errors++;
      }
      else
      {
            if((6) strlen($passwd1)<6 || (7) preg_match('[^0-
9]',$passwd1))
            {
                  $e_passwd = constant('error-password-length');
                  $passwd1="";
                  $passwd2="";
                  $n_errors++;
            }
      }
}
```

Code from updatePassword.php

(8) In order to ensure users privacy, users’ password will not be directly stored in database but a checksum of it, This way nobody except its owner will be able to know it and it will remain protected even if database is accessed without authorization.

(3) Changing avatar.

ChangeProfile.php will display a form prepared to upload pictures to system’s database. The following box shows its code:

```
<form id=uploadAvatarForm action=uploadAvatar.php method="post"
enctype="multipart/form-data (1)" onSubmit="return
checkPath();(2)">
```

```
<input type="file" name="bx_file" style="display:none;"
onChange="refreshTb_fake();">

<input type="text" name="tb_fake" readonly><br>

<input type="button" name="b_file_fake" value="<?php echo
constant('button-browse');?>" onclick="displayFileWindow()(3);">

<input type="submit" name="b_submit" value="<?php echo
constant('button-upload');?>!">

 </form>
```

Code from changeProfile.php

(1) Enctype's value as "multipart/form data" illustrates the aim of this form, transferring a file.

(2) **checkPath()** function, will be the one in charge of authorizing form's submitting if a file has been correctly introduced. Its code is showed in the following box:

```
function checkPath()
{
  uploadForm = document.getElementById('uploadAvatarForm');

  if(uploadForm.bx_file.value=="") return false;
  else return true;
}
```

Code from changeProfile.php

(3) Since standard file's input does not fit design requirements some of his elements as its search button have been configured to not be displayed. A fake button has been added. Its behavior it's managed by displayFileWindow(), JavaScript function that will perform hidden button duties.

```
function displayFileWindow()
{
  uploadForm = document.getElementById('uploadAvatarForm');
   uploadForm.bx_file.click();
}
```

Code from changeProfile.php

After collecting file's information, **updateAvatar.php** will be accessed.

First of all code required to update the file which was selected by the user will be executed.

```
$name = $_FILES['bx_file']['name'];
$tmp_name = $_FILES['bx_file']['tmp_name'];
$location="images/avatars/$username";
move_uploaded_file($tmp_name,$location);
```

Code from updateAvatar.php

Finally user's record in database will be updated to the new location of his avatar.

```
include('dbAlexandria1.php');
include('connect.php');

$SQL="UPDATE users SET users.avatar=\"$location\" WHERE
users.username=\"$username\"";

mysql_query($SQL) or die(mysql_error());

mysql_close();
```

Code from updateAvatar.php

(4) Changing description.

Each user will be able to write a short description which will be showed in his own profile. A text box will be displayed in **ChangeProfile.php** for this purpose.

Once users finish composing their description, **updateDescription.php** will be accessed. There, introduced description will be stored in user's record in database.

```
$description = $_POST['tb_description'];

include('dbAlexandria1.php');
include('connect.php');

$SQL="UPDATE users SET users.description=\"$description\" WHERE
users.username=\"$username\"";

mysql_query($SQL) or die(mysql_error());

mysql_close();
```

Code from updateDescription.php

Sending a message

Developed private message system has no any special feature if we compare it to the one we can find in a common website providing this functionality.

Once user has filled in required information, his message is stored in site's database and it will be available henceforth when recipient checks his inbox.

As a first step to send a message, once user has selected the proper option, a hidden form will be showed to collect all the information.

As if was defined previously, there will be three fields to fill in: recipient, subject and body. Going back to the "Analysis of requirements" document we can find following restrictions related to this case:

"Recipient field is mandatory, an existing user must be specified in this field"
"Either subject or body fields must be filled in. Both messages without subject and messages without body are allowed, besides the common ones"

All these restrictions are checked by a JavaScript function implemented for this purpose. This function is executed each time a user clicks on send button and is the one in charge of authorizing the operation even though the same check is performed at server's side once the information has arrived, to prevent unexpected behavior or absence of JavaScript support in user's web-browser.

A fragment of getPData, JavaScript function just mentioned it is showed in the following box.

```
var topic = f.tb_topic.value;
var content = f.tb_content.value;
var recipient = f.tb_recipient.value; (1)

if(recipient!="" && (topic!="" || content!="")) (2)
{
      if(topic=="") f.tb_topic.value="(no subject)";
      f.destination.value=document.location.href;
      f.style.display = 'none'; (4)
      document.getElementById('sendMessage').submit();(3)
}
```

Code from showMessages.php

After collecting introduced values from a form with "f" as id (1). All restrictions defined in "Analysis of requirements" are checked (2). If all three conditions have been fulfilled, form is submitted (3) and hidden again (4).

According to form's definition, **sendMessage.php** will be accessed (5).

```
<form id=sendMessage action=sendMessage.php (5) method="post">

 <label><?php echo constant('label-recipient');?></label>
 <input type="text" name="tb_recipient" value=""> <br><br>
 <label><?php echo constant('label-subject');?></label><br>
 <textarea class=singleRowArea
name="tb_topic"></textarea><br><br>
 <textarea class=multipleRowArea
name="tb_content"></textarea><br><br>
 <input type="hidden" name="destination" value="">
 <input type="button" value="<?php echo constant('button-
send');?>" id="b_submit" name="b_submit" onClick="getPdata()" >
 <input type="button" value="<?php echo constant('button-
cancel');?>" name="b_cancel" onClick="getPdata(this.value)">

</form>
```

Code from showMessages.php

Once reached **sendMessage.php** the restriction check already performed in JavaScript will be repeated (6) and system will check that selected recipient exists in site's database (7).

```
If($_POST['tb_recipient']!="" && $_POST['tb_content']!="") (6)
{
     //Code to access to database
     $SQL = "SELECT * FROM users WHERE id = \"$tmpRecipient\"
OR username = \"$tmpRecipient\";" (7);

     $result = mysql_query($SQL) or die('Error:
'.mysql_error());

     if(mysql_num_rows($result)==0)  //non existing recipient
deny request
          {
               header('Location: accessDenied.php');
          }
```

Code from sendMessage.php

Finally if this process ends up successfully message will be stored in database. It will be available next time recipient checks his inbox.

```
$SQL = "INSERT INTO privateMessages
(`sender`,`recipient`,`topic`,`content`,`timestamp`) VALUES
('$userID','$recipient','$topic','$content','$sqlTime');";

mysql_query($SQL) or die(mysql_error());
```

Code from sendMessage.php

Replying a message

Since replying a message is an especial case of sending a message, a very similar procedure is followed to perform both operations.

Users will be allowed to reply the message they are reading. ShowMessage.php, the page which displays the content of a received message will provide an option to do so. First steps of this procedure are identical to the ones explained in the previous section; a form will be showed and message sender will be able to compose his message.

Once finished, data will be checked by getPData(), the same function used to send a common message. Both associated restrictions (message has a recipient, either subject or body field have been filled in, at least) will be checked, even though it would be difficult to find an error, since recipient and subject fields will be already filled when they are displayed to the user according to replied message.

After data validation, sendMessage.php will be required as it happened in the previous section. In this file there is a condition and a portion of code to be executed when message to be sent is a reply of a previous one.

```
if($_POST['reply'](1)!="") //then a reply of a previously sent
message is being sent
        {
                mysql_query('BEGIN') (2);

                $reply = $_POST['reply'];
                $SQL = "INSERT INTO privateMessages
(`sender`,`recipient`,`topic`,`content`,`timestamp`,`reply`)
VALUES
('$userID','$recipient','$topic','$content','$sqlTime','$reply')
;";
                mysql_query($SQL);

                //Checking if current request is the first reply to
the selected message (reply field must be changed from not
replied to replied)
                $SQL = "SELECT reply FROM privateMessages WHERE
id=\"$reply\";" (3);
                $tmpResult = mysql_query($SQL) or die(mysql_error());
                $tmpRow = mysql_fetch_assoc($tmpResult);
                if($tmpRow['reply']==-1)
                {
                        $SQL = "UPDATE privateMessages SET reply=\"-2\"
WHERE id=\"$reply\";" (4);
                        mysql_query($SQL);
                        if(mysql_error())
                        {
                                mysql_query('ROLLBACK');
                                die();
                        }
                }
                mysql_query('COMMIT'); (5)
            }
```

Code from sendMessage.php

(1) Notice that the id of the message being replied has been enclosed among the information which was sent after ShowMessage.php's form submitting.
(2) A MySQL transaction is started since replying a message requires more than one access to site's database and all of the operations should be performed atomically.
(3) First of all, message must be sent, so it is stored in database.
(4) Once reply has been sent, database must be corrected, since replied message's state could have changed if it current reply was the first one. Recall that a privateMessage record in database includes an especial field for this purpose called "reply". Depending on its value, system will know which procedure should be applied when a message or a chain of messages must be showed. This field's possible values are the following ones:

Reply = -1	Message has not been replied.
Reply = -2	Replied message, head of a message chain.
Reply > 0	Message reply. Reply field will contain the id of the first message in the message chain.

Since there is a message being replied, if current reply is the first one related to it its state will change from non replied to replied.

(5) If both operations have been performed without errors transaction can be finished.

Checking incoming/outgoing messages

The way incoming messages are displayed in Project Alexandria follows the rules or standards usually observed in major social networks. Two restrictions involving this function were defined during analysis of requirements phase:

"A message chain must be managed as a single message. Once a message has been replied only the last one of its replies will be showed in incoming messages"

On the other hand,

"In order to speed up the access to the inbox, only a defined amount of the last received messages should be showed at once. There should be any navigation or

filtering facilities in order to ease the reading and management of previously received messages”

First of all, a group of messages to be showed must be fetched from database. According to the second restriction, only the ten most recent messages ($selectedAmount = 10) (1) will be retrieved. Furthermore, considering the first restriction, only non-replied messages or the last replies of a message must be showed. As a first step to achieve it only non-replied messages (2) (reply=-1) and messages heading a message chain (3) (reply=-2) will be selected.

```
$SQL = "SELECT
privateMessages.id,privateMessages.sender,privateMessages.topic,
privateMessages.content,privateMessages.timestamp,privateMessage
s.readed,users.username,users.avatar FROM privateMessages INNER
JOIN users ON privateMessages.sender = users.id WHERE (recipient
= \"$userID\" AND reply<\"0\" AND deletedBy!=\"$userID\" (2)) OR
(sender = \"$userID\" AND reply=\"-2\" AND
deletedBy!=\"$userID\" (3)) ORDER BY timestamp DESC LIMIT (1)
$fromMessage,$selectedAmount; ";

$result = mysql_query($SQL) or die('Error: '.mysql_error());
```

Code from showMessages.php

Once ten results have been collected, an iterator in charge of showing each one of them will start.

First of all, according to the second restriction, if current message has been replied (4) the last one of its replies must be fetched and showed instead of the selected one (5). Remind that during the first selection just heads of message chains besides non replied messages were chosen.

```
$SQL = "SELECT
privateMessages.id,privateMessages.sender,privateMessages.topic,
privateMessages.content,privateMessages.timestamp,privateMessage
s.readed,users.username,users.avatar FROM privateMessages INNER
JOIN users ON privateMessages.sender = users.id WHERE
privateMessages.id = \"$msgID\" OR
privateMessages.reply=\"$msgID\" ORDER BY timestamp DESC LIMIT
$fromMessage,$selectedAmount;";

$replies = mysql_query($SQL) or die('Error: '.mysql_error());

if(mysql_num_rows($replies)>0) (4)
{
  //Fetching the last generated message (since they're ordered
by DESC timestamp)
  //This message will be showed instead of the one heading the
reply chain (oldest)
while($row=mysql_fetch_assoc($replies))
```

```
{
     if($row['sender']!=$userID)
     {
          //Getting reply information (5)
          $msgID=$row['id'];
          $readed = $row['readed'];
          $topic = $row['topic'];
          $content = $row['content'];
          $timestamp =$row['timestamp'];
          //Retrieving recipient's name
          $senderID=$row['sender'];
          $senderName = $row['username'];
          $senderAvatar = $row['avatar'];
          break;
     }
}
```

Code from showMessages.php

Once proper information has been corrected, if needed, message is displayed using html divs as containers. Non readed messages will be showed in a different color to ease its identification.

```
if($readed==1)
{
  echo   "<div class=\"messageMultipleContainerRow\">";
}
else
{
  echo   "<div class=\"messageMultipleContainerRowNonReaded\">";
}
```

Code from showMessages.php

Each message container will show some sender’s information as picture and nickname, and current message details, subject and body. Both of them will be split if needed in order to fit container dimensions. Complete subject and body will be fully displayed when a message is requested to be showed in **showMessage.php**.

```
echo  "<div class=\"messageSmallContainerInfoDiv\">\n";
//Checking if topic should be splitted to make it fit in the
container
if(strlen($content)<=75)
     {
          echo  "<p>".$content."</p>\n";
     }
else
     {
          echo  "<p>".substr($content,0,75)."...</p>\n";
     }
```

```
echo    "</div>\n";
```

Code from showMessages.php

On the other hand since only ten messages are displayed each time, a navigation system has been developed to have access to the older ones. A new variable called $fromMessage has been added for this purpose. Its value will establish from which message, selection must be done.

```
$SQL = "SELECT * FROM privateMessages INNER JOIN users ON
privateMessages.sender = users.id WHERE (recipient =
\"$userID\" AND reply<\"0\" AND deletedBy!=\"$userID\" (2)) OR
(sender = \"$userID\" AND reply=\"-2\" AND
deletedBy!=\"$userID\" (3)) ORDER BY timestamp DESC LIMIT (1)
$fromMessage,$selectedAmount; ";
```

Code from showMessages.php

This variable value will be initialized as zero each time incoming messages section is accessed if it had no previously assigned value.

```
if(isset($_GET['fromMessage']))
{
      $fromMessage =
$_GET['fromMessage'];
}
else
{
      $fromMessage = 0;
}
```

Code from showMessages.php

Its value will be updated and sent in displayed navigation links.

```
if(($fromMessage+10)<$totalMessages)
{
  $nextInitialPos = $fromMessage+10;
  echo  "<div class=\"bControlBar\"
onclick=\"location.href='".$rootURL."showMessages.php?fromMessag
e=$nextInitialPos'\">".constant('button-next')."</div>";
}

if($fromMessage>9)
{
   $nextInitialPos = $fromMessage-10;
   echo  "<div class=\"bControlBar\"
onclick=\"location.href='".$rootURL."showMessages.php?fromMessag
e=$nextInitialPos'\">".constant('button-previous')."</div>";
}
```

Code from showMessages.php

Deleting messages

Among the options available in showMessages.php single and multiple message deletion can be found.

```
echo "<div class=messageControlBar>";

echo '<div class="bControlBar"
onclick="selectAll();">'.constant('button-select-
all').'</div>';

echo '<div class="bControlBar"
onclick="deleteSelected();">'.constant('button-delete-
selected').'</div>';
 echo "</div>"; // of messageControlBar
```

Code from showMessages.php

As a first step messages to be deleted must be selected from current message list then, deletion button must be clicked.

After that proper form including selected message ids will be submitted by a JavaScript function, **pmDeleteSelected.php** will be accessed.

```
function deleteSelected()
{
      f = document.getElementById(
'selectMessages' );
      f.action='pmDeleteSelected.php';
      f.submit();
}
```

Code from showMessages.php

PmDeleteSelected.php will iterate through all selected messages deleting them. However a message only will be truly deleted from database once both sender and recipient have request its deletion. If this situation has not happened message will be marked, by using its “deletedBy” field. In this field the id of the user who has requested its deletion will be written.

```
while($msg = mysql_fetch_assoc($toBeRemoved))
{
      $msgID = $msg['id'];
      $deletedBy = $msg['deletedBy'];
      //Current is the first one (sender/recipient) who asks for
deleting the msg, msg must be preserved
```

```
	if($deletedBy==-1)
	{
		$SQL = "UPDATE privateMessages SET 
deletedBy=\"$userID\" WHERE id=\"$msgID\";";
		mysql_query($SQL);
		if(mysql_error())
		{
			mysql_query('ROLLBACK');
			mysql_close();
			die();
		}
	}
	//Both sender and recipient have already asked for 
deleting the msg, remove it from DB
	else
	{
		$SQL="DELETE FROM privateMessages WHERE 
privateMessages.id=\"$msgID\";";
		$result = mysql_query($SQL);
		if(mysql_error())
		{
			mysql_query('ROLLBACK');
			mysql_close();
			die();
		}
	}
```

Code from pmDeleteSelected.php

This way, next time he checks his messages system will notice that marked messages must not be displayed any more. At the same time, the other user involved in this message will be able to read it until he decides it must be deleted.

Notice that if a replied message is deleted, its replies will be deleted as well.

```
//Checking if an action committed on this message would require 
actions on other messages (reply chain)
if($selectedMessage['reply']==-2) //Replied message, head of 
the reply chain, its replies must be deleted too
{
	$SQL = "SELECT * FROM privateMessages WHERE 
privateMessages.id=\"$messageID\" OR 
privateMessages.reply=\"$messageID\" ;";
	$toBeRemoved = mysql_query($SQL);
}

if($selectedMessage['reply']>0) //Single reply of a message, 
its replies must be deleted too
{
	$reply = $selectedMessage['reply'];
	$SQL = "SELECT * FROM privateMessages WHERE 
privateMessages.id=\"$reply\" OR 
privateMessages.reply=\"$reply\" ;";
	$toBeRemoved = mysql_query($SQL);
}
```

Code from pmDeleteSelected.php

Marking several messages as not read

Sometimes marking a message that has been already read as not read can be useful as a reminder for the next time. This functionality is also available in **showMessages.php.**

```
echo "<div class=messageControlBar>";

echo '<div class="bControlBar"
onclick="markAsReaded();">'.constant('button-mark-as-
readed').'</div>';
```

Code from showMessages.php

To do so messages to be marked must be selected from current message list then, function button must be clicked.

After that proper form including selected message ids will be submitted by a JavaScript function, **pmMarkAsReaded.php** will be accessed.

```
function markAsReaded()
{
      f = document.getElementById(
'selectMessages' );
      f.action='pmMarkAsReaded.php';
      f.submit();
}
```

Code from showMessages.php

Finally database will be updated from **pmMarkAsReaded.php**. Readed field value in PrivateMessage record will be updated.

```
$SQL="UPDATE privateMessages SET privateMessages.readed=\"0\"
WHERE privateMessages.id=\"".$selectedMessages[$i]."\" AND
privateMessages.recipient=\"$userID\";";

$result = mysql_query($SQL);
```

Checking notifications

As it was stated in previous chapters of this document, notifications are an especial kind of messages whose main characteristic is that they always have the system as sender and cannot be replied. For this reason**, showNotifications.php** looks pretty similar to **ShowMessages.php** but it is simpler.

Following the same restriction, only the ten most recent notifications have to be showed at once while the rest of them will be accessible by using a navigation system.

After its selection which is performed in the same way message selection is done but with a simpler condition (1). Notifications will be showed according to their type.

```
//Getting 10 notifications to be showed
$SQL = "SELECT notifications.issue, notifications.content,
users.username, users.avatar, notifications.timestamp,
notifications.id FROM notifications INNER JOIN users ON
notifications.content = users.id WHERE recipient = \"$userID\"
ORDER BY timestamp DESC LIMIT (1)
$fromNotification,$notificationAmount";

$notificationList = mysql_query($SQL);
```

Code from showNotifications.php

Two types of notifications have been defined so far: friendship requests and friendship confirmations. Notifications type is stored in a field called issue which was created for this purpose in Notifications database.

According to their type notifications will be processed and showed.

- Friendship requests will display the picture and nickname of its sender besides two buttons where recipient will decide whether ignoring or accepting the request.

- Friendship confirmations are merely informative. They will be sent to the requester user once recipient has accepted him as a friend.

On the other hand since only ten notifications are displayed each time, a navigation system has been developed to have access to the older ones. Once again an identical procedure to the one used to manage private messages has been used. $fromNotification variable will have the same behavior as previously defined $fromMessage. Its value will establish from which notification, selection must be done.

```
$SQL = "SELECT notifications.issue, notifications.content,
users.username, users.avatar, notifications.timestamp,
notifications.id FROM notifications INNER JOIN users ON
notifications.content = users.id WHERE recipient = \"$userID\"
ORDER BY timestamp DESC LIMIT
```

```
$fromNotification,$notificationAmount";

$notificationList = mysql_query($SQL);
```

Code from showNotifications.php

This variable value will be initialized as zero each time incoming messages section is accessed if it had no previously assigned value.

```
if(isset($_GET['fromNotification']))
{
	$fromNotification = $_GET['fromNotification'];
}
else
{
	$fromNotification = 0;
}
```

Code from showNotifications.php

Its value will be updated and sent in displayed navigation links.

```
if(($fromNotification+10)<$notificationCount)
{
   $nextInitialPos = $fromNotification+10;
   echo  "<div class=\"bControlBar\" 
onclick=\"location.href='".$rootURL."showNotifications.php?fromN
otification=$nextInitialPos'\">".constant('button-
next')."</div>";
}

if($fromNotification>9)
{
    $nextInitialPos = $fromNotification-10;
    echo  "<div class=\"bControlBar\" 
onclick=\"location.href='".$rootURL."showNotifications.php?fromN
otification=$nextInitialPos'\">".constant('button-
previous')."</div>";
}
```

Code from showNotifications.php

Deleting notifications.

After reading a notification this one is no longer useful since its aim is merely informative. As time goes by the amount of stored notifications can increase a lot, so it would be useful to be able to select and remove the ones users do not want to keep any more.

In the same way it is done with messages, after selecting the notifications to be deleted by clicking the proper button **nDeletedSelected.php** will be accessed (1).

```
function deleteSelected()
{
	f = document.getElementById( 'selectMessages' );
	f.action='nDeleteSelected.php' (1).;
	f.submit();
}
```

Code from nDeleteSelected.php

Inside this file selected notification ids will be gathered in an array (2) and then they will be deleted (3).

```
$SQL = "DELETE FROM notifications WHERE notifications.id IN ($strNotifIDs (2)) AND
notifications.recipient=\"$userID\";" (3);

mysql_query($SQL) or die('Error: '.mysql_error());
```

Code from nDeleteSelected.php

Sending a friendship request.

Each time a user visits another user profile, if they do not have any kind of relationship yet, sending a friendship request option will be displayed.

```
<?php
   if($areFriends==false)
   {
        echo "<A
href=\"".$rootURL."addFriend.php?sUser=".$sUserID."\">".constant
('add-as-friend')."</A>";
   }
?>
```

Code from sUserMenu << showProfile.php

All required operations are performed in **addFriend.php**

Two database tables are involved in this operation.

- FriendList it is a table storing all friendship relationships established in the system. For each established relationship two records will be placed. The first of them is stored when friendship request is sent and the second when that request has been confirmed so both pending and confirmed relationships are stored inside this table.

- FriendRequests is the table where all pending friend requests in the system are stored. In contrast to the previous table information stored in this table is temporary and will be deleted once friendship relationship is confirmed or ignored by its recipient.

First of all the first record in FriendList will be stored.

```
$SQL = "INSERT INTO friendList
(`member1`,`member2`,`timestamp`) VALUES
('$userID','$sFriendID','$sqlTime');";

mysql_query($SQL);
```

Code from addFriend.php

Notice that $userID contains requester ID and $sFriendID recipient's.

After that friendship requests is sent.

```
$SQL = "INSERT INTO friendRequests
(`sender`,`recipient`,`idFriendList`,`timestamp`) VALUES
('$userID','$sFriendID','$idFriendList','$sqlTime');";

mysql_query($SQL);
```

Code from addFriend.php

If both operations have been completed without errors transaction will be finished and selected user's profile will be displayed again.

Confirming a friendship request.

When a user adds another one as a friend, a new notification will appear in the second's notification box. Recipient will have to choose whether accepting or denying this friendship request. If he accepts **confirmFriend.php** will be accessed.

Information about the users involved in this operation will be fetch from database as a first step.

```
$SQL = "SELECT * FROM friendList WHERE
friendList.id=\"$idFriendList\"";
$result = mysql_query($SQL) or die(mysql_error());
$row = mysql_fetch_row($result);
$member1 = $row[constant("friendList::member1")];
$member2 = $row[constant("friendList::member2")];
```

Code from confirmFriend.php

Recall that this record was added to friendList table when friendship request was sent. Another one switching member's order should be added at the end of this operation.

Afterwards friendList's record status will be updated from "pending" to "confirmed" (1). The second record with switched values will be added as well (2).

```
mysql_query('BEGIN');

//Confirming friendship on current DB score
$SQL = "UPDATE friendList SET friendList.status=\"confirmed\",
friendList.timestamp=\"$sqlTime\"  WHERE
friendList.id=\"$idFriendList\"(1)";
mysql_query($SQL);
if(mysql_error())
{
      mysql_query('ROLLBACK');
      die();
}

//Creating new score on friendship table
$SQL = "INSERT INTO friendList
(`member1`,`member2`,`timestamp`,`status`) VALUES
('$member2','$member1','$sqlTime','confirmed');" (2);
mysql_query($SQL);
if(mysql_error())
{
      mysql_query('ROLLBACK');
      die();
}
```

Code from confirmFriend.php

User who requested confirmed friendship request will be sent a notification to let him know about its confirmation.

```
//Notifying requester friendship request has been confirmed
$SQL = "INSERT INTO notifications (`recipient`,`issue`,`content`,`timestamp`) VALUES
('$member1','friend_confirmation','$content','$sqlTime');";
mysql_query($SQL);
if(mysql_error())
{
      mysql_query('ROLLBACK');
      die();
}
```

Code from confirmFriend.php

Finally confirmed friendship request record will be deleted from friendRequests table in database.

```
Deleting current friendship request from member2's box
$SQL = "DELETE FROM friendRequests WHERE friendRequests.id=\"$idFriendRequest\"";
mysql_query($SQL);
if(mysql_error())
{
      mysql_query('ROLLBACK');
      die();
}

//Committing changes
mysql_query('COMMIT');
```

```
```

Code from confirmFriend.php

If all mentioned operations have been performed without errors transaction will be finished and changes committed.

Looking a user up in the system.

In order to expand their friend network users will be able to look other users up in user directory. Since especial privacy rules, as hidden accounts, have been defined in this prototype users will be able to access the whole user directory.

According to **"Analysis of requirements"**' document, one statement was defined related to this issue:

"In order to ease this process (look up process) several filtering options will be provided. Some user characteristics as name, sex, current city or current country will be available for this purpose"

This process will take place in **searchFriend.php.** Once accessed the whole site's user directory will be showed. Only ten results will be displayed per page so there will be a navigation system to go along all the results.

In the box below ten results ($selectedAmount has been given this value (1)) are selected from site's user directory to be showed.

```
$SQL = "SELECT * FROM users $sqlWhere ORDER BY users.timestamp
DESC LIMIT $fromElement,$selectedAmount";
$result = mysql_query($SQL) or die('Error: '.mysql_error());
$cont=0;
while($friendList = mysql_fetch_assoc($result))
{
      $friendIDs[$cont]=$friendList["id"];
      $friendNames[$cont] = $friendList["username"];
      $friendAvatars[$cont] = $friendList["avatar"];
      $cont++;
}
```

Code from searchFriend.php

Notice that $fromElement variable is used to store which results must be showed each time as it happened in previous operations as showing messages or notifications. $fromElement value will contain an index from which results must be collected. Its value will be reset if previous page have not sent its previous value (2).

```
//Getting range of elements that are going to be showed
$selectedAmount=10 (1);
if(isset($_GET['fromElement']) (2))
```

```
{
      $fromElement = $_GET['fromElement'];
}
else
{
      $fromElement = 0;
}
```

Code from searchFriend.php

By using navigation buttons $fromElement's value will be updated.

```
if(($fromElement+$selectedAmount)<$friendCount)
{
   $nextInitialPos = $fromElement+$selectedAmount;
      echo  "<div class=\"bControlBar\"
onclick=\"location.href='".$rootURL."searchFriend.php?fromElemen
t=$nextInitialPos'\">".constant('button-next')."</div>";
}

if($fromElement>$selectedAmount-1)
{
   $nextInitialPos = $fromElement-$selectedAmount;
   echo  "<div class=\"bControlBar\"
onclick=\"location.href='".$rootURL."searchFriend.php?fromElemen
t=$nextInitialPos'\">".constant('button-previous')."</div>";
}
```

Code from searchFriend.php

In the right side of the page a form will be displayed, here users will be able to filter the information showed in the result window by introducing some details about people they are looking for. Each time page is loaded if any filtering setting was defined search will be performed according to it.

```
$sqlWhere="WHERE " (3);

if($_POST['tb_username']!="") $sqlWhere =
$sqlWhere."users.username LIKE '%".$_POST['tb_username']."%' AND
" (4);

if($_POST['tb_city']!="") $sqlWhere = $sqlWhere."users.city LIKE
'%".$_POST['tb_city']."%' AND " (4);

if($_POST['tb_country']!="") $sqlWhere =
$sqlWhere."users.country LIKE '%".$_POST['tb_country']."%' AND "
(4);

if($_POST['tb_mail']!="") $sqlWhere =
$sqlWhere.'users.email="'.$_POST['tb_mail'].'" AND ' (4);
```

```
if($sqlWhere=="WHERE ") $sqlWhere="" (5);

else $sqlWhere = substr($sqlWhere,0,strlen($sqlWhere)-5);
```

Code from searchFriend.php

Notice that $sqlWhere contains WHERE clause that must be used in database query (3) (see first code-box of this section). Defined settings will customize this field by adding information to $sqlWhere (4).

If none of them has been defined, $sqlWhere will remain unchanged so it will have to be reset in order to show the whole user directory (SELECT * FROM users) which is defined default behavior (5).

Both user picture (6) and nickname (7) will be showed in each one of the ten selected results. By clicking on them selected user profile will be accessed.

```
echo "<div class=\"searchResultRow\">";
echo "<A href=\"".$rootURL."showProfile.php?sUser=".$friendID."\"> <img
class=\"friendPicture\" src=$friendAvatar (6) alt=$friendName /> </A>";

echo "<div class=\"friendName\">";
echo   "<A class=userLink
href=\"".$rootURL."showProfile.php?sUser=".$friendID."\">".$friendName
(7)."</A><br>\n";
echo "</div>"; //of friendName div

echo "</div>";
```

Code from searchFriend.php

Showing a user's friend list

In order to show any user friend list, the same system described for looking users up is used. After all a this operation is a particular case of user look-up. The only difference lies on the searching condition, this one will be more complex this time to search only among the friends of the user whose friend list is being showed (1).

Query used in this case is showed in the following box:

```
$SQL = "SELECT users.id,users.username,users.avatar FROM users
INNER JOIN friendList ON users.id = friendList.member2 WHERE
$sqlWhere (2) friendList.member1=\"$sUserID\" AND
friendList.status=\"confirmed\ (1)" ORDER BY \"username\" LIMIT
$fromElement,$selectedAmount;";
```

```
$result = mysql_query($SQL) or die('Error: '.mysql_error());
```

Code from showSearchResults.php

Filtering settings among showed results are available as well. Its implementation and behavior are the same described in the previous section. Filtering conditions will be included in $sqlWhere variable (2).

```
if($_POST['tb_username']!="") $sqlWhere =
$sqlWhere.'users.username="'.$_POST['tb_username'].'" AND ';

if($_POST['tb_city']!="") $sqlWhere =
$sqlWhere.'users.city="'.$_POST['tb_city'].'" AND ';

if($_POST['tb_country']!="") $sqlWhere =
$sqlWhere.'users.country="'.$_POST['tb_country'].'" AND ';

if($_POST['tb_mail']!="") $sqlWhere =
$sqlWhere.'users.email="'.$_POST['tb_mail'].'" AND ';

$sqlWhere = substr($sqlWhere,0,strlen($sqlWhere)-1);
```

Code from showSearchResults.php

Registering a group.

If a user wants to find people sharing his interests and there is no group created for this purpose he will be able to create a new one. Group creation code is divided in two files, while **registerGroup.php** will be the one in charge of collecting proper information for its creation, **commitGroupRegistration.php** will perform required operations to create it.

As it was stated during the definition of the requisites linked to this issue, user will have to choose a name for the group and write a short description of its aim. In addition, he will be able to upload a picture related to group's aim that will be show in group's main page.

In the box below the code of the form displayed to collect all required information is showed.

```
<form id="groupRegistration"
action="commitGroupRegistration.php" method="post"
enctype="multipart/form-data" (1) onSubmit="return
checkPath()(2);">

        <label><?php echo constant('label-name');
?></label><input type="text" size=40 name="tb_name" value="" />
        <label><?php echo constant('label-description');
```

```
?></label><textarea name="tb_description"
class=multipleRowArea></textarea>
        <label><?php echo constant('label-group-picture');
?><font size="-2"> (<?php echo constant('label-optional'); ?>)
</font> </label>

         (3)<input id="bx_file" type="file" name="bx_file"
style="display:none;" onChange="refreshTb_fake();">
        <input type="text" name="tb_fake" readonly>
        <input type="button" name="b_file_fake" value="<?php
echo constant('button-browse'); ?>"
onclick="displayFileWindow();">
        <br>
        <center> <input type="submit" name="b_submit"
value="<?php echo constant('button-register'); ?>" /> </center>

</form>
```

Code from registerGroup.php

- Since a file could be attached, enctype attribute will be added. Its value shows this situation ("multipart/form-data") (1).
- Notice that form will not be submitted until CheckPath() a JavaScript function allows it (2).
- File selection pop up will work as the one described previously in "Changing account settings" section, whose aim was uploading an avatar (3).

CheckPath function will check both required fields (group name and description) have been introduced (4).

```
  function checkPath()
  {
    uploadForm = document.getElementById('groupRegistration');
    if(uploadForm.tb_name.value=="" ||
uploadForm.tb_description.value=="")(4)
        {
            alert('Group name and short description are
mandatory fields');
            return false;
        }
      else return true;
  }
```

Code from registerGroup.php

After its execution form will be submitted to **commitGroupRegistration.php.**

Once data are collected in **commitGroupRegistration.php,** group will be registered if there is not any group with the same name. Even though the introduction of required fields was checked previously by **checkPath()**, this check will be performed once again since javascript could not work properly in user's web-browser or could not be habilitated.

```
$SQL = "INSERT INTO groups
(`name`,`description`,`createdBy`,`timestamp`) VALUES
('$gName','$gDescription','$userID','$sqlTime');";


mysql_query($SQL) or die(mysql_error());
```

Code from commitGroupRegistration.php

If a picture was attached the following code will both upload (5) it and update proper field in database including its location in the server (6).

```
$name = $_FILES['bx_file']['name'];
$tmp_name = $_FILES['bx_file']['tmp_name'];
$location="images/groups/$idGroup";
$coverAdded = move_uploaded_file($tmp_name,$location) (5);

if($coverAdded==true)
{
    $SQL="UPDATE groups SET groups.avatar=\"$location\" WHERE
groups.id=\"$idGroup\"" (6);
    mysql_query($SQL) or die(mysql_error());
}
```

Code from commitGroupRegistration.php

Joining a group.

Once a user requests to join a group, selected group id will be sent to **addGroup.php.** This file will perform all required operations to commit his registration as a participant.

As a first step two checks will be performed to verify received group id has been received correctly and user is not already a participant of this group.

If check ends up successfully user will be registered as a participant of the group in Membership table.

```
$SQL = "INSERT INTO membership(`userID`,`groupID`,`timestamp`)
VALUES ('$userID','$groupID','$sqlTime');";
mysql_query($SQL);
if(mysql_error())
{
      die();
}
```

Code from addGroup.php

Leaving a group.

If a user decides to stop being a participant of a group, selected group id will be sent to **leaveGroup.php.** This file will perform all required operations to disassociate him from that group.

Similar checks to the ones performed when a user wants to join a group will be commit. On the one hand group's id reception will be verified. On the other hand user's membership to selected group will be corroborated.

If both checks end up successfully user will be unregistered as a participant of the group by removing proper record in Membership table.

```
$SQL = "DELETE FROM membership WHERE userID=\"$userID\" AND
groupID = \"$groupID\";";
mysql_query($SQL);
if(mysql_error())
{
      die();
}
```

Code from leaveGroup.php

Looking for a group.

In order to find people sharing their interests users will be provided with a system to look up and filter groups among all the ones registered in the site.

According to **"Analysis of requirements"**' document, one statement was defined related to this issue:

"To ease this process (look up process) several filtering options will be provided. Some group characteristics as name or short description will be available for this purpose"

This process will take place in **searchGroup.php.** Once accessed the whole site's group directory will be showed. Only ten results will be displayed per page so there will be a navigation system to go along all the results.

In the box below ten results ($selectedAmount has been given this value (1)) are selected from site's group directory to be showed.

```
$SQL = "SELECT * FROM groups $sqlWhere ORDER BY
groups.timestamp DESC LIMIT $fromElement,$selectedAmount (1)";

$groupList = mysql_query($SQL) or die('Error: '.mysql_error());
```

Code from searchGroup.php

Notice that $fromElement variable is used, once again, to store which results must be showed each time as it happened in previous operations as showing messages or notifications. $fromElement value will contain an index from which results must be collected. Its value will be reset if previous page have not sent its previous value (2).

```
//Getting range of elements that are going to be showed
$selectedAmount=10 (1);
if(isset($_GET['fromElement']) (2))
{
	$fromElement = $_GET['fromElement'];
}
else
{
	$fromElement = 0;
}
```

Code from searchGroup.php

By using navigation buttons $fromElement's value will be updated.

```
if(($fromElement+$selectedAmount)<$friendCount)
{
   $nextInitialPos = $fromElement+$selectedAmount;
      echo  "<div class=\"bControlBar\"
onclick=\"location.href='".$rootURL."searchGroup.php?fromElement
=$nextInitialPos'\">".constant('button-next')."</div>";
}

if($fromElement>$selectedAmount-1)
{
   $nextInitialPos = $fromElement-$selectedAmount;
   echo  "<div class=\"bControlBar\"
onclick=\"location.href='".$rootURL."searchGroup.php?fromElement
=$nextInitialPos'\">".constant('button-previous')."</div>";
}
```

Code from searchGroup.php

In the right side of the page a form will be displayed, here users will be able to filter the information showed in the result window by introducing some details about the group they are looking for (its name or description). Each time page is loaded if any filtering setting was defined search will be performed according to it.

```
$sqlWhere="WHERE " (3);

if($_POST['tb_name']!="") $sqlWhere = $sqlWhere."groups.name
LIKE '%".$_POST['tb_name']."%' AND " (4);
```

```
if($_POST['tb_description']!="") $sqlWhere =
$sqlWhere."groups.description LIKE
'%".$_POST['tb_description']."%' AND " (4);

if($sqlWhere=="WHERE ") $sqlWhere=""(4);

else $sqlWhere = substr($sqlWhere,0,strlen($sqlWhere)-5) (5);
```

Code from searchGroup.php

Notice that $sqlWhere contains WHERE clause that must be used in database query (3) (see first code-box of this section). Defined settings will customize this field by adding information to $sqlWhere (4).

If none of them has been defined, $sqlWhere will remain unchanged so it will have to be reset in order to show the whole group directory (SELECT * FROM groups) which is defined default behavior (5).

Group picture (6), name (7) and description (split if needed) (8) will be showed in each one of the ten selected results. By clicking on their container selected group main page will be accessed. In addition a link to become a participant of the group will be showed (9).

```
echo "<center> <A
href=\"".$rootURL."showGroup.php?sGroupID=".$groupID."\"> <img
class=\"picture150px\" src=$groupAvatar (6) alt=$groupName />
</A> </center>";
echo "</div>";

echo "<div style=\"width:340px;float:left;\">";
echo  "<A class=userLink
href=\"".$rootURL."showGroup.php?sGroupID=".$groupID."\">".$grou
pName (7)."</A><br>\n";

echo  "<p>".substr($groupDescription,0,75) (8)."</p>";

if(in_array($groupID,$myFavGroups)==false)
{
      (9) echo   "<br><A class=userLink style=\"float:right;\"
href=\"".$rootURL."addGroup.php?sGroupID=".$groupID."\">".consta
nt('join-group')."</A>\n";
}

echo "</div>"; //of groupInfo div
echo "</div>";
```

Code from searchGroup.php

Showing a user's favourite groups.

The same system described for looking for a group is used in order to show user's favourite groups. After all, this operation is a particular case of group search. The only difference lies on the searching condition, being this one more complex this time, to search only among user's favourite groups (1).

The box below, shows the query associated with this search:

```
$SQL = "SELECT * FROM membership INNER JOIN users ON
membership.userID = users.id INNER JOIN groups ON
membership.groupID = groups.id $sqlWhere ORDER BY
groups.timestamp DESC LIMIT $fromElement,$selectedAmount" (1);

$groupList = mysql_query($SQL) or die('Error: '.mysql_error());
```

Code from showFavouriteGroups.php

Filtering settings among showed results are available as well. Its implementation and behavior are the same described in the previous section. Filtering conditions will be included in $sqlWhere variable (2).

```
$sqlWhere="WHERE " (2);

if($_POST['tb_name']!="") $sqlWhere = $sqlWhere."groups.name
LIKE '%".$_POST['tb_name']."%' AND ";

if($_POST['tb_description']!="") $sqlWhere =
$sqlWhere."groups.description LIKE
'%".$_POST['tb_description']."%' AND ";

$sqlWhere = $sqlWhere.'membership.userID="'.$sUserID.'"';
```

Code from showFavouriteGroups.php

Registering a book.

Since developed site is devoted to literature it is important to provide the users with several tools to allow them to describe their literary interests. Once done so, they will be able to group according to their interests and share experiences. The code to perform operations related to the definition of a favourite book list is divided in two files. **RegisterBook.php** will be the one in charge of collecting proper information for its registration while **commitGroupRegistration.php** will perform required operations to register it.

According to the definition of the requisites linked to this issue, users will have to fill in some information about the book they want to register in the system. Besides its title, author and genre will be required. In addition, they will be able to upload a picture of the book's cover.

By the following code a form to collect all required details is showed.

```
<form id="bookRegistration" action="commitBookRegistration.php"
method="post" enctype="multipart/form-data (1)." onSubmit="return
checkPath()(2);">

<img id="pictureBox" class="profileIMG" style="float:right;
margin:25px 25px 0px 0px;" src="images/books/default_book.png"
alt="Book's cover picture" />

<label><?php echo constant('label-title');?></label><input
type="text" size=40 name="tb_title" value="" />

<label><?php echo constant('label-author');?></label><input
type="text" size=40 name="tb_author" value="" />

<label><?php echo constant('label-genre');?></label><input
type="text" size=40 name="tb_genre" value="" />

<label><?php echo constant('label-cover-picture');?><font
size="-2"> (<?php echo constant('label-optional');?>) </font>
</label>

(3) <input id="bx_file" type="file" name="bx_file"
style="display:none;" onChange="refreshTb_fake();">
<input type="text" name="tb_fake" readonly>
<input type="button" name="b_file_fake" value="<?php echo
constant('button-browse');?>" onclick="displayFileWindow();">
<br>
<center> <input type="submit" name="b_submit" value="<?php echo
constant('button-register');?>" /> </center>

</form>
```

Code from registerBook.php

- Since a book's cover picture could be attached, enctype attribute will be added. Its value shows this situation ("multipart/form-data") (1).
- Notice that form will not be submitted until CheckPath() a javascript function allows it (2).
- File selection pop up will work as the one described previously in "Changing account settings" section, whose aim was uploading an avatar (3).

CheckPath function will check all required fields (book's title, author and genre) have been introduced (4). If there are some of them missing a pop up warning will be showed and form will not be submitted.

```
function checkPath()
{
  uploadForm = document.getElementById('bookRegistration');
```

```
   if(uploadForm.tb_title.value=="" ||
uploadForm.tb_author.value=="" ||
uploadForm.tb_genre.value=="")(4)
      {
            alert('Book title, author and genre are mandatory
fields!');
            return false;
      }
   else return true;
  }
```

Code from registerBook.php

Once all conditions have been satisfied **commitBookRegistration.php** will be accessed**.**

After collecting incoming data, in **commitGroupRegistration.php,** the book will be registered if it does not exist yet in database.

Even though the introduction of required fields was checked previously by **checkPath()**, this check will be performed once again, since javascript could not work properly in user's web-browser or could not be habilitated.

```
$SQL = "INSERT INTO books
(`title`,`author`,`genre`,`timestamp`) VALUES
('$title','$author','$genre','$sqlTime');";


mysql_query($SQL) or die(mysql_error());
```

Code from commitBookRegistration.php

If a picture of the book's cover was attached, the following code will both upload (5) it and update proper field in database including its location in the server (6).

```
$name = $_FILES['bx_file']['name'];
$tmp_name = $_FILES['bx_file']['tmp_name'];
$location="images/books/$idBook";
$coverAdded = move_uploaded_file($tmp_name,$location)  (5);

if($coverAdded==true)
{
    $SQL="UPDATE books SET books.picture=\"$location\" WHERE
books.id=\"$idBook\""  (6);
    mysql_query($SQL) or die(mysql_error());
}
```

Code from commitBookRegistration.php

Marking a book as favourite.

When a user requests a book to be marked as one of its favourites, chosen book id will be sent to **addBook.php.** This file will perform all required operations to commit his addition to user's favourite book list.

After verifying that received book id is valid and that current book is not among user's favourites yet, database will be updated and selected book will be added to Bookshelf table.

```
$SQL = " INSERT INTO bookshelf (`bookID`,`userID`,`timestamp`)
VALUES ('$sBookID','$userID','$sqlTime');";
mysql_query($SQL);
if(mysql_error())
{
	die();
}
```

Code from addGroup.php

Unmarking a book as favourite.

If due to any reason a user decides that a book among its favourites should be unmarked, selected book id will be sent to **removeBookfromFavourites.php.** This file will perform all required operations to remove it from its favourite book list.

Similar checks to the ones performed to mark a book as favourite will be commit. On the one hand book's id reception will be verified. On the other hand book has to be marked. If both checks end up successfully book will be removed from user's favourites (Bookshelf table in database)

```
$SQL = "SELECT * FROM bookshelf WHERE bookID=$sBookID and
userID=$userID;"
mysql_query($SQL);
if(mysql_error())
{
	die();
}
```

Code from leaveGroup.php

Looking a book up.

Users will be provided with a system to look books up in site's database. This way they will be able to add new ones as favourites or to publish reviews about their experience when they read them. According to **"Analysis of requirements"**' document, one statement was defined related to this issue:

“In order to speed up the access, only a defined amount of books should be showed at once. There should be any navigation facilities in order to ease the checking through all the results found”

This process will take place in **searchBook.php.** Once accessed all registered books will be showed. Only ten results will be displayed per page so there will be a navigation system to go along all the results.

In the box below ten results ($selectedAmount has been given this value (1)) are selected among registered books to be showed.

```
$SQL = "SELECT * FROM books $sqlWhere ORDER BY books.timestamp
DESC LIMIT $fromElement,$selectedAmount(1)";
$result = mysql_query($SQL) or die('Error: '.mysql_error());
$cont=0;
while($bookList = mysql_fetch_assoc($result))
{
      $bookIDs[$cont]=$bookList["id"];
      $bookNames[$cont] = $bookList["title"];
      $bookAuthors[$cont] = $bookList["author"];
      $bookAvatars[$cont] = $bookList["picture"];
      $bookGenres[$cont] = $bookList["genre"];
      $cont++;
}
```

Code from searchBook.php

Notice that $fromElement variable is used to store which results must be showed each time as it happened in previous operations as showing messages or notifications. $fromElement value will contain an index from which results must be collected. Its value will be reset if previous page have not sent its previous value (2).

```
//Getting range of elements that are going to be showed
$selectedAmount=10 (1);
if(isset($_GET['fromElement']) (2))
{
      $fromElement = $_GET['fromElement'];
}
else
{
      $fromElement = 0;
}
```

Code from searchBook.php

By using navigation buttons $fromElement’s value will be updated.

```
if(($fromElement+$selectedAmount)<$friendCount)
```

```
{
   $nextInitialPos = $fromElement+$selectedAmount;
      echo  "<div class=\"bControlBar\"
onclick=\"location.href='".$rootURL."searchBook.php?fromElement=
$nextInitialPos'\">".constant('button-next')."</div>";
}

if($fromElement>$selectedAmount-1)
{
   $nextInitialPos = $fromElement-$selectedAmount;
   echo  "<div class=\"bControlBar\"
onclick=\"location.href='".$rootURL."searchBook.php?fromElement=
$nextInitialPos'\">".constant('button-previous')."</div>";
}
```

Code from searchBook.php

In the right side of the page a form will be displayed, here users will be able to filter the information showed in the result window by introducing some details about the book they are looking for. Each time page is loaded if any filtering setting was defined search will be performed according to it.

```
$sqlWhere="WHERE " (3);

if($_POST['tb_title']!="") $sqlWhere = $sqlWhere."books.title
LIKE '%".$_POST['tb_title']."%' AND " (4);

if($_POST['tb_author']!="") $sqlWhere = $sqlWhere."books.author
LIKE '%".$_POST['tb_author']."%' AND "(4);

if($_POST['tb_genre']!="") $sqlWhere = $sqlWhere."books.genre
LIKE '%".$_POST['tb_genre']."%' AND "(4);

if($sqlWhere=="WHERE ") $sqlWhere="" (5);

else $sqlWhere = substr($sqlWhere,0,strlen($sqlWhere)-5);
```

Code from searchBook.php

Notice that $sqlWhere contains WHERE clause that must be used in database query (3) (see first code-box of this section). Defined settings will customize this field by adding information to $sqlWhere (4).

If none of them has been defined, $sqlWhere will remain unchanged so it will have to be reset in order to show all registered books in the system (SELECT * FROM books) which is defined default behavior (5).

Book's cover picture (6), title (7),author (8), and genre(9) will be showed for each one of the ten selected results. By clicking on their container selected book profile will be accessed. In addition a link to mark that book as favourite will be displayed (10).

```
echo "<center> <A
href=\"".$rootURL."showBook.php?sBookID=".$bookID."\"> <img
class=\"picture150px\" src=$bookAvatar (6) alt=$bookName /> </A>
</center>";
echo "</div>";

echo "<div style=\"width:340px;float:left;\">";
echo  "<A class=userLink
href=\"".$rootURL."showBook.php?sBookID=".$bookID."\">".$bookNam
e (7)."</A><br>\n";

echo  "<A class=userLink
href=\"".$rootURL."searchBook.php?author=".$bookAuthor."\">".$bo
okAuthor (8)."</A><br>\n";

echo  "<A class=userLink
href=\"".$rootURL."searchBook.php?genre=".$bookGenre."\">".$book
Genre (9)."</A><br>\n";

if(in_array($bookID,$myFavBooks)==false)
{
      (10) echo  "<br><A class=userLink style=\"float:right;\"
href=\"".$rootURL."addBook.php?sBookID=".$bookID."\">".constant(
'add-to-favourites')."</A>\n";
}
```

Code from searchBook.php

Showing user favourite books.

A very similar system to the one just described for looking books up in database is used to show user complete list of favourite books. The only difference lies on the searching condition, which will be as follows:.

```
$SQL = "SELECT books.id, books.title, books.author,
books.picture, books.genre FROM books, bookshelf WHERE books.id
= bookshelf.bookID AND $sqlWhere bookshelf.userID=$userID
ORDER BY books.timestamp DESC LIMIT
$fromElement,$selectedAmount";

$bookList = mysql_query($SQL) or die('Error: '.mysql_error());
```

Code from myFavouriteBooks.php

Filtering settings among showed results are available as well. Its implementation and behavior are the same described in the previous section. Filtering conditions will be included in $sqlWhere variable (1).

```
if($_POST['tb_title']!="") $sqlWhere = $sqlWhere."books.title
LIKE '%".$_POST['tb_title']."%' AND "(1);

if($_POST['tb_author']!="") $sqlWhere = $sqlWhere."books.author
LIKE '%".$_POST['tb_author']."%' AND "(1);

if($_POST['tb_genre']!="") $sqlWhere = $sqlWhere."books.genre
LIKE '%".$_POST['tb_genre']."%' AND "(1);
```

Code from myFavouriteBooks.php

Reviewing a book.

From each book profile users will be able to publish their own reviews and rate them according to their satisfaction after finishing them. A form will be displayed for this purpose (**showBook.php**). Three fields will have to be filled in: review's title, content and book rating.

AddReview.php will be in charge of collecting introduced data and publishing it.

First of all, system will check all required data were introduced correctly (1).

```
if(isset($_POST['tb_title']) && isset($_POST['tb_content']) &&
isset($_POST['sBookID']) (1))
      {
            //Code to store a review in database
              }
else
      {
            header('Location: accessDenied.php');
      }
```

Code from addReview.php

After this, if no error arisen, review will be published by storing it in bookReviews table.

```
$SQL = "INSERT INTO bookReviews
(`bookID`,`userID`,`title`,`content`,`rating`,`timestamp`)
VALUES
('$sBookID','$userID','$title','$content','$rating','$sqlTime');
";

mysql_query($SQL);
```

Code from addReview.php

6. Bibliography

www.wikipedia.com

http://php.net/manual/es/index.php

http://www.desarrolloweb.com/javascript/

http://www.w3c.es/divulgacion/guiasbreves/hojasestilo

http://www.w3schools.com/html/default.asp

www.pixabella.com

Appendix I: Analysis phase: Similar website reviews

In order to get a wider approach about the requirements and services we want to provide by developing our own website, a pair of similar websites, related to literature field have been studied and reviewed as a part of the analysis phase.

The first one, called **"www.whatsonmybookshelf.com"**, is maybe one of the simplest examples, in terms of design and implementation, of exchanging book web-platform .
Although the exchange system applied in this case could not be the same one it's planned to use in our case (point-based system, user must buy or earn points by shipping his books to other users and then is able to get some books by "paying" their point price), some useful ideas could be extracted from it.

As a second example another website, called **www.bookcrossing.com,** has been analyzed. This one is closer to the idea we are interested in developing, because besides providing a mechanism to allow book sharing devotes a part of its structure and functionality to allow the interaction among users. More personal user profiles or forums have been implemented for this purpose.

WhatsOnMyBookshelf.com

Aim of the site:

Whatsonmybookshelf.com is a website providing a simple point-based system to allow their users purchasing books just by paying the shipping taxes of the books they are asked for. Since finding a match between two concrete users willing to share their books can be quite difficult, a different approach is applied to grant a user, who shared some of his books with another user, being able to get the books he is interested in for free from a different user.

Target user:

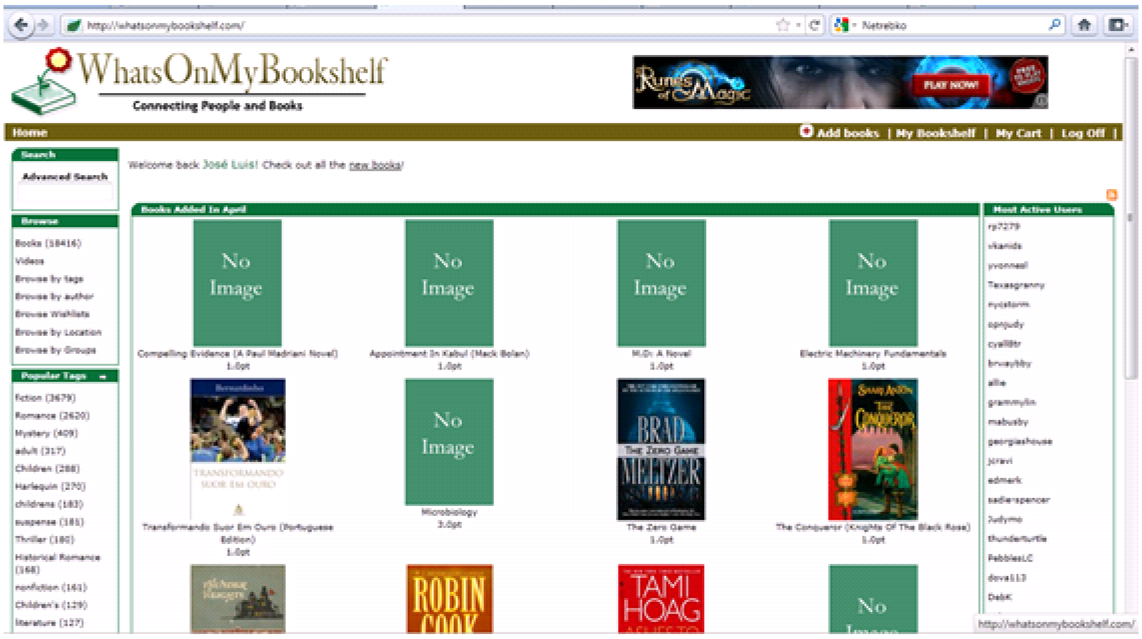
The site's book database includes all kind of books, even children's and teen's literature besides textbooks so this platform can be useful for almost all kinds of people. The only restriction is related to the exchange system that requires an adult able to do the shipping and picking up of the requested books.

Design approach and characteristics:

In order to reach the widest range of users the site's appearance looks quite simple and little redundant. Its main (and almost unique) aim is showing which books are available in each moment and providing filtering mechanisms to ease the look-up. Then you can purchase the books you are interested in by using a mechanism equal to a common web-store (adding books to your cart) but where points instead of money are used to commit the transactions.

According to your interests you can either look for a specific book typing its name or ISBN in the search bar or showing the available books classified by genre, topic or user tags as well.

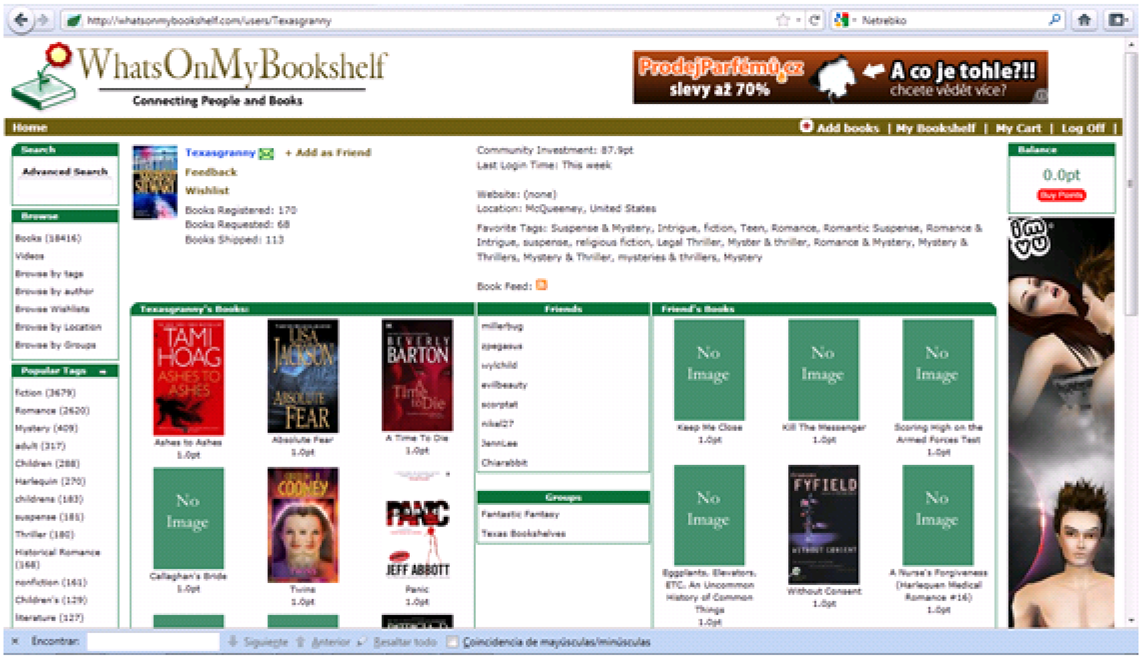
In the main page you are shown which books were added lately and which users are the most participative ones.



Following the same principles we reach the user's profile page where some details can be viewed:

- A few personal details: Username, locations, website and favorite tags.
- Statistics related to his activity in the community: books registered and available for sharing, books requested and books already shipped.
- List of friends *.
- List of books you could get from your friends.
- List of groups you have joined *.
- Number of points you own in order to purchase books from other users.

**It must be pointed out that having a friend or group list in this website doesn't allow you to interact with them in the way we are used to do it (for instance, in a social network).*
By adding friends you pretend to have quicker information about their updates (books they recently added and you could get) since they will be shown in the right side of your profile page.
In the same way groups haven't been created to provide a meeting point for users with similar interests. They are mere classifiers of users and books related/interested in one specific topic.



Texasgranny + Add as Friend
Feedback
Wishlist
Books Registered: 170
Books Requested: 68
Books Shipped: 113

Community Investment: 87.9pt
Last Login Time: This week

Website: (none)
Location: McQueeney, United States
Favorite Tags: Suspense & Mystery, Intrigue, fiction, Teen, Romance, Romantic Suspense, Romance & Intrigue, suspense, religious fiction, Legal Thriller, Myster & thriller, Romance & Mystery, Mystery & Thrillers, Mystery & Thriller, mysteries & thrillers, Mystery

Book Feed:

Books (18416)
Videos
Browse by tags
Browse by author
Browse Wishlists
Browse by Location
Browse by Groups

Popular Tags
fiction (3679)
Romance (2620)
Mystery (409)
adult (317)
Children (288)
Harlequin (270)
childrens (183)
suspense (181)
Thriller (180)
Historical Romance (168)
nonfiction (161)
Children's (129)
literature (127)

Ashes to Ashes 1.0pt
Absolute Fear 1.0pt
A Time To Die 1.0pt
Callaghan's Bride 1.0pt
Twins 1.0pt
Panic 1.0pt

millerbug
zpegasus
wylchild
evilbeauty
scorptail
nikal27
JennLee
Chiarabbit

Fantastic Fantasy
Texas Bookshelves

Keep Me Close 1.0pt
Kill The Messenger 1.0pt
Scoring High on the Armed Forces Test 1.0pt
Eggplants, Elevators, ETC. An Uncommon History of Common Things 1.0pt
Without Consent 1.0pt
A Nurse's Forgiveness (Harlequen Medical Romance #16) 1.0pt

Description of the exchange method

A user willing to ship his books register them into the website's database. He will be given 1 points for each 5 books registered. According to the current price of the books in stores, the system will establish its point-based price (1 point per 20$). This amount of points will be given to the owner after shipping the book as a response to a purchase.
Since the user has already received some of his points (due to the registration process), as soon as a book has been registered any other user is allowed to request its shipping.

A purchaser, by using the search engine of the website finds the book and adds it to his cart asking for its purchase.

The website removes the book from the available book list and sends a notification to the owner with the e-mail direction of the purchaser. If there is no special arrangement between them the seller has 5 days to do the shipping of the requested book. As soon as the book has been shipped the seller notifies it through the website.

Once the purchaser has received the book he must finish the transaction by notifying it to the website.

Although the aim of the website is promoting the free exchange of books between users from a non lucrative approach, you are allowed to buy a maximum of 5 points per month at a price of 2$ per point. Incoming money is invested in the improvement and maintenance of the website.

Implementation approach and feasibility

The website looks simple and schematic. It's organized in tables and the most important part of its elements are links to query the database in order to find the required book.

Since there is no direct interaction among users through the site, remaining the last one as a coordinator of the transaction, the logic associated to the main transaction of the site can be quite simple as well.

After studying its content organization, appearance and functionality needs it could be concluded that a development of a similar website would be feasible by using html, php and a sql database.

Conclusions

Interesting aspects: The point-based purchasing approach allows a fair exchange between users and is not really sensitive to fails due to unfair behavior of the seller/buyer.

Non-interesting aspects: Since the first aim of our project besides allowing book exchanging between users is providing some mechanisms to ease the interaction among them the analyzed website doesn't fulfill all the capabilities we would like to implement, looking rather aseptic.

Bookcrossing.com

Aim of the site:

Bookcrossing.com is website that proposes a new idea to allow book exchanging between users without shipping costs and even contact between them. The website acts as a news table where you can check which books have been left in your city by other users who wanted to "free" them. Books can be waiting for new readers in a park, a coffee shop or a museum so if you find the one you are interested in reading on the website you just have to go there and take it. The main aim is fulfilling your reading desires and then let another enjoy the book by setting it free again in another place and updating that information on the website.

Besides providing this main service the site allows the users to interact in a higher level than the previous one by including forums and a common wall where all the users can share their thoughts.

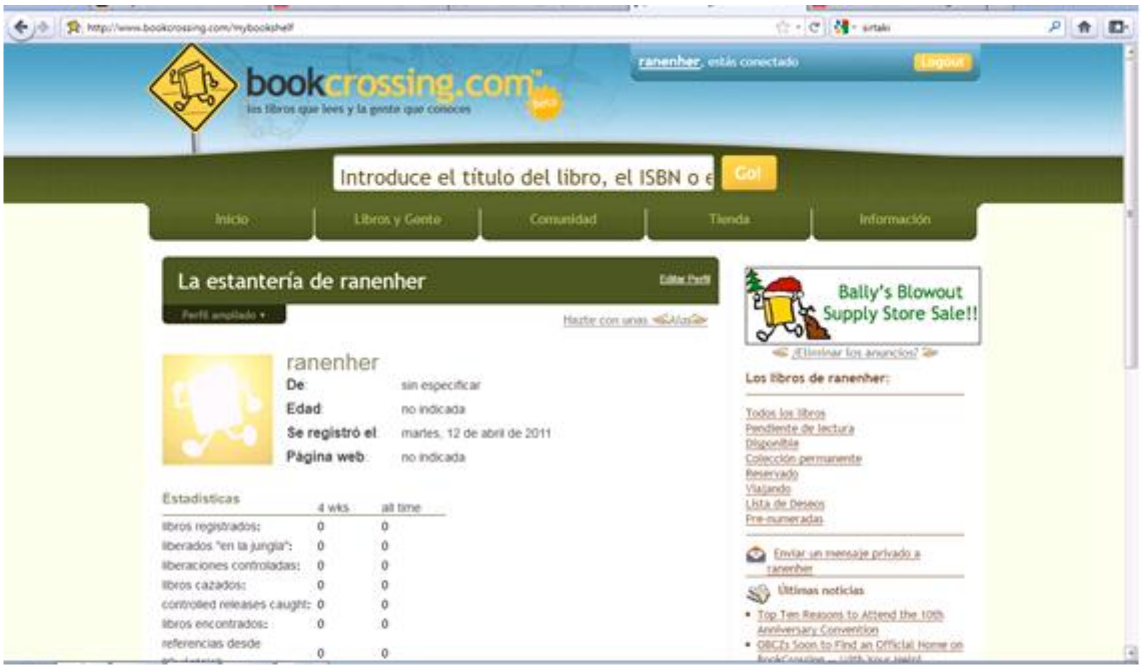
Target user:

As in the previous example there is no restriction according to the books that can be shared so there is no specific sector of people whom this site is focused on.

Since you want to reach as much people as possible a simple and intuitive design could be the best option to succeed.

Design approach and characteristics:

Site's design is simple but pleasant and attractive enough. It's organized in a hierarchical structure divided in five submenus.


http://www.bookcrossing.com/mybookshelf
bookcrossing.com
los libros que lees y la gente que conoces
ranenher, estás conectado
Logout
Introduce el título del libro, el ISBN o e
Go!
Inicio
Libros y Gente
Comunidad
Tienda
Información
Editar Perfil
Perfil ampliado
Hazte con unas Alas
Bally's Blowout Supply Store Sale!!
¿Eliminar los anuncios?

La estantería de ranenher

ranenher

De	sin especificar
Edad	no indicada
Se registró el	martes, 12 de abril de 2011
Página web	no indicada

Estadísticas

	4 wks	all time
libros registrados:	0	0
liberados "en la jungla":	0	0
liberaciones controladas:	0	0
libros cazados:	0	0
controlled releases caught:	0	0
libros encontrados:	0	0
referencias desde	0	0

Los libros de ranenher:

Todos los libros
Pendiente de lectura
Disponible
Colección permanente
Reservado
Viajando
Lista de Deseos
Pre-numeradas

Enviar un mensaje privado a ranenher

Últimas noticias

- Top Ten Reasons to Attend the 10th Anniversary Convention
- OBCZs Soon to Find an Official Home on BookCrossing ... With Your Help!

The first one, called "Home" gathers all the information related to the user who is using the platform. Besides a user's profile and a friend list you can find there the options to "free" your own books for the first time or release the ones you already found. Another interesting option is "My bookshelf" that allows the user to keep his bookcase organized by creating an inventory on the website.

In the second one a user will find search options for the books he is interested in finding. If there is a concrete book that must be found the search bar is the most useful election but there is another option available ("Go hunting") where a user can be shown a list of available books depending on the city he is living in. Besides these look-up facilities some statistics about the most exchanged books and the most active users can be found in this menu.

Inside "Community" menu all the interaction mechanisms provided by the website are grouped. As it was said before among these facilities there is a forum and a common wall to let the users express their thoughts, nevertheless, there is not any feedback or reply system available for the rest of the users besides writing another entry in the common wall. A convention list can be checked as well.

Fourth and fifth menus are not especially interesting since the first one leads to a virtual shop and the last one includes all the information related to the website itself (About us, FAQ, Privacy policy, etc).

Implementation approach and feasibility:

After comparing this site to the one analyzed previously a slight increase of the complexity is noticed.

This increase of the complexity has two main causes:

- On the one hand, the higher amount of pages developed as a result of expanding the range of services and functionalities provided by the website.

- On the other hand some of those services require a more complex implementation, especially those whose aim is to allow users to interact.

According to the technologies that could be used in order to program a similar solution, only CSS styles could be added to the one selected for the previous site (html-php-sql).

Conclusions

Interesting aspects: The whole site 's aim is going beyond than allowing users to exchange their books, is a little closer to a social network approach providing some interaction mechanisms.
Some functionalities, like the possibility of inventorying and organizing your book collection are interesting and could be used as an example for our own implementation.

Non-interesting aspects: Even though the book exchanging system is quite original, it doesn't require direct interaction among users. Since we would our website to allow the creation of little readers' communities perhaps we should think about a different one.