UNIVERSITÀ DI PISA

Computer Engineering
Distributed Systems and Middleware Technologies

Project Specification

Academic Year: 2021/2022

Biondi Matteo

Cristofani Federico

Tumminelli Gianluca

1 Contents

1. Project Specification

LibrarInk is a distributed web-app that allows users to consult the catalogue of library and make online reservations of the available books.

1.1. System Requirements

System must:

- Store users' account information
- Store library's books
- Keep track of book reservations
- Keep track of book loans
- Keep track of transactions history
- Notify to the user the availability of a book in their wishlist

1.2. Use Cases

Unregister user can:

1. Sign-up to the library web-app

Unlogged user can:

1. Sign-in to the library web-app

Logged user can:

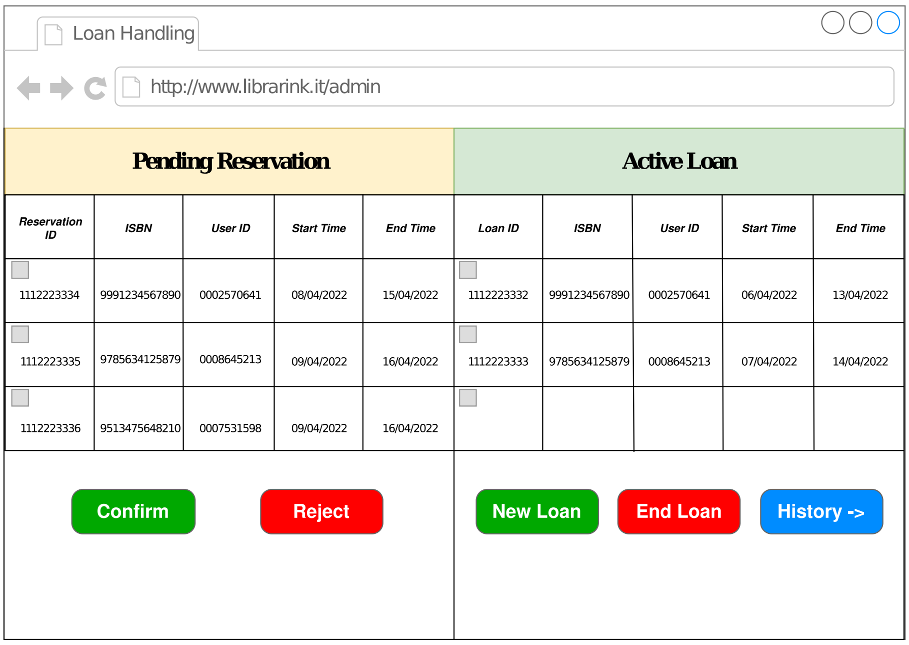
1. Consult the catalogue
2. Reserve a book
3. Insert a book on their private wishlist
4. Rate a book
5. Delete a book reservation
6. Sign-out from the library web-app

Librarian can:

1. Consult pending reservations and pending loans
2. Confirm or cancel a reservation
3. Register a book loan
4. Register a book return
5. Modify the number of a book's copies
6. Consult ended reservations and ended loans

2. Mockup

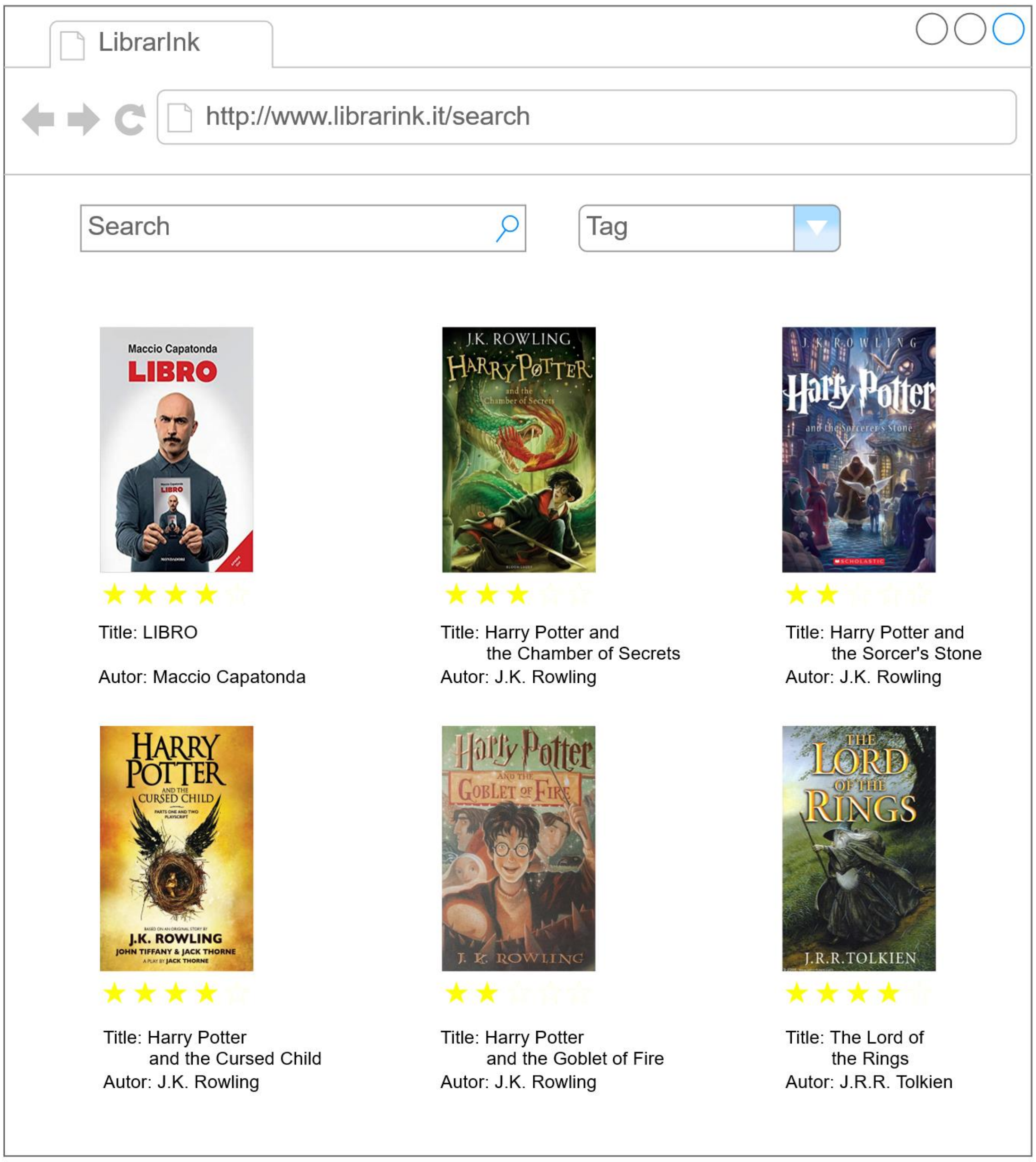
2.1. Admin page mock-up



Pending Reservation					Active Loan				
Reservation ID	***ISBN***	***User ID***	***Start Time***	***End Time***	***Loan ID***	***ISBN***	***User ID***	***Start Time***	***End Time***
☐ 1112223334	9991234567890	0002570641	08/04/2022	15/04/2022	☐ 1112223332	9991234567890	0002570641	06/04/2022	13/04/2022
☐ 1112223335	9785634125879	0008645213	09/04/2022	16/04/2022	☐ 1112223333	9785634125879	0008645213	07/04/2022	14/04/2022
☐ 1112223336	9513475648210	0007531598	09/04/2022	16/04/2022	☐				

Figure 1: Admin page mock-up

2.2. Search book page mock-up



Title: LIBRO
Autor: Maccio Capatonda

Title: Harry Potter and the Chamber of Secrets
Autor: J.K. Rowling

Title: Harry Potter and the Sorcer's Stone
Autor: J.K. Rowling

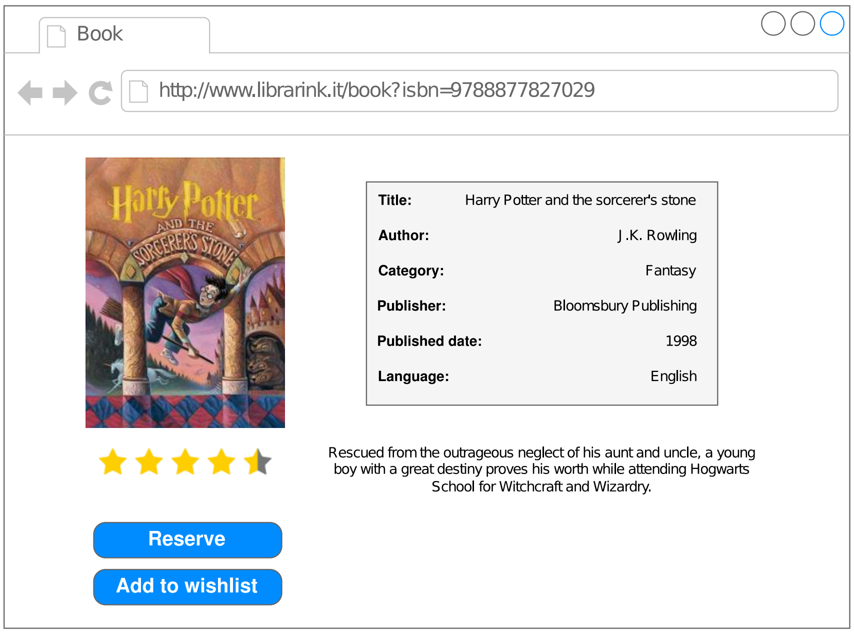
Title: Harry Potter and the Cursed Child
Autor: J.K. Rowling

Title: Harry Potter and the Goblet of Fire
Autor: J.K. Rowling

Title: The Lord of the Rings
Autor: J.R.R. Tolkien

Figure 2: User homepage mock-up

2.3. Book’s details page mock-up



Title:	Harry Potter and the sorcerer's stone
Author:	J.K. Rowling
Category:	Fantasy
Publisher:	Bloomsbury Publishing
Published date:	1998
Language:	English

Rescued from the outrageous neglect of his aunt and uncle, a young boy with a great destiny proves his worth while attending Hogwarts School for Witchcraft and Wizardry.

Figure 3: Book's details page mock-up

3. System Architecture

3.1. General introduction on the distribution of tasks and data

The application offers via a web interface a portal for reserve and loan books from a library. The user can access the service by connecting to the URL of the site. The server is written mainly in JAVA language for the WebApp part and Erlang for handling requests in sensitive and concurrent tasks. The databases used are different: we use MySQL for the storage of users, books and old loans and reservations, while a Mnesia database (which we access via Erlang) is used for the management of copies and the lending/reserving of books, to handle the concurrency adequately.

The web application is composed of JSP pages and each request is handled via servlets. We also use EJB for the definition of the business logic and the database access (via Entity according to the JPA specification for each database table). Some of the user requests need access to MySQL or Mnesia. In the former case, the requests are directed to the relevant DBMS and then answered. In the second case, the requests are sent to the proxy which forward them to Mnesia replica(s), receive the relative response and finally send back the response to the application server which will make appropriate use of it.

The requests are forwarded by the proxy according to the MD5 hash of the book's ISBN, thus forwarding the requests in such a way as to ensure an even distribution on the active nodes. Any changes are then reflected in the backup replicas for the active node affected by them.

3.2. Database organization: MySQL + Mnesia

As previously mentioned, the web application makes use of data from two different sources. Two different databases, one in MySQL and one in Mnesia, have been set up to support the storage of the site. Access to the data contained in the Mnesia database and any manipulation action on the data itself is regulated by means of several Erlang application layers, with the task of forwarding the request, identifying it, validating it and finally executing it.

3.2.1 MySQL Database

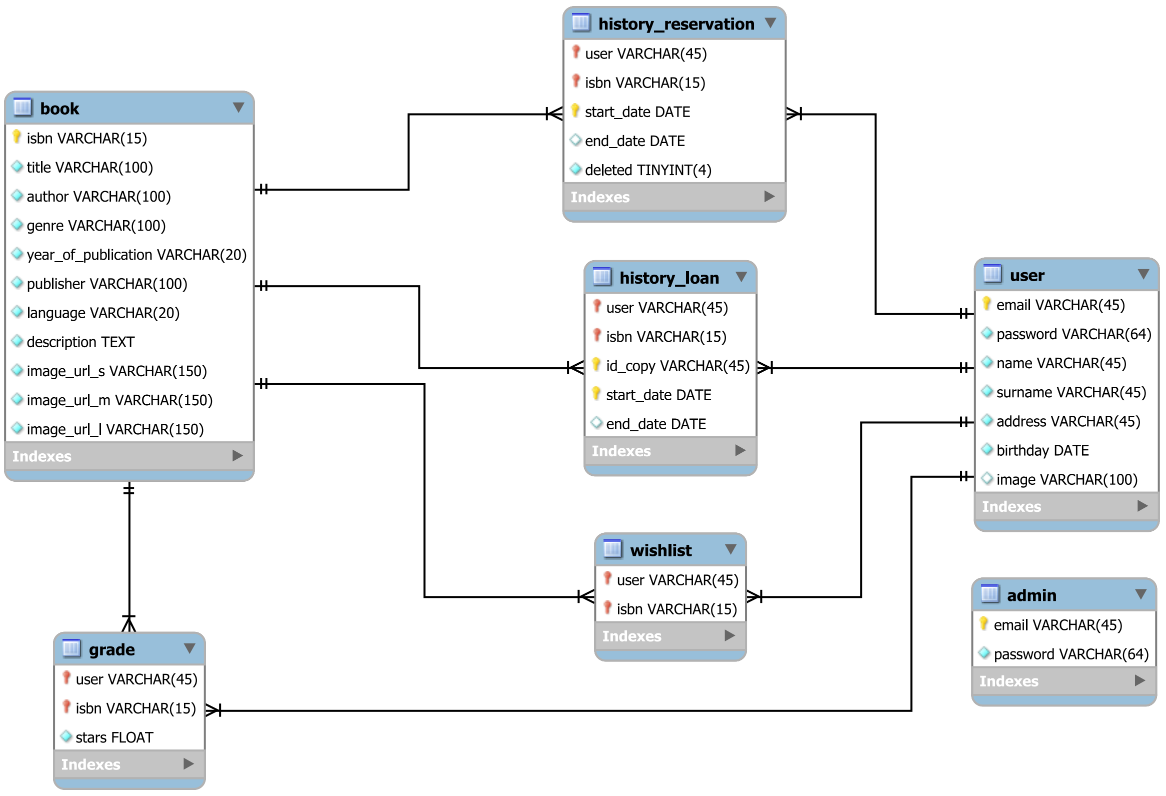


Figure 4: MySQL ER Schema

The MySQL database consists of the following tables:

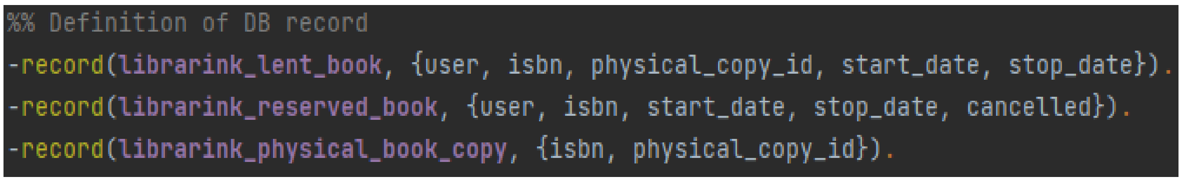
- **book:** Contains all information about a book in the library.
- **user:** Contains all information about a registered user.
- **admin:** Contains the login credentials of only administrator users.
- **history_reservation:** Contains the old reservations of users.
- **history_loan:** Contains users' old, now completed loans.
- **wishlist:** Contains the books that users would like to book.
- **grade:** Contains the ratings that users assign to books.

The data in these tables were identified as not leading to competitive access. Therefore, there is no justification for using a database such as Mnesia that handles competition in a more efficient way by guaranteeing data consistency. The MySQL database and the Mnesia database are in some way related to each other. The tables 'history_reservation' and 'history_loan' contain the terminated 'reservation' and 'loan' data, respectively, and are therefore extracted from the Mnesia database since they are not considered necessary for the service.

Transferring them to the MySQL database allows the Mnesia database to be kept as lean as possible and speeds up access to it. Saving this information only serves the purpose of maintaining a log for loans and reservations.

Only one replica exists of the MySQL database.

3.2.2 Mnesia Database



```
%% Definition of DB record
-record(librarink_lent_book, {user, isbn, physical_copy_id, start_date, stop_date}).
-record(librarink_reserved_book, {user, isbn, start_date, stop_date, cancelled}).
-record(librarink_physical_book_copy, {isbn, physical_copy_id}).
```

Figure 5: Record composition of Mnesia tables. The primary key is the first item in the record definition.

Mnesia is not meant to replace the standard SQL database or to handle terabytes of data across many nodes. Mnesia is rather made for smaller amounts of data, on a limited number of nodes.

The Mnesia Database consists of the following tables:

- **librarink_lent_book:** Contains any information relating to the loan of a book to a user
- **librarink_reserved_book:** Contains any information relating to the reservation of a library book
- **librarink_physical_book_copy:** Contains the physical copies of a book

Tables are maintained both in memory and on disk ("disc_copies") in active nodes and only on disk ("disc_only_copies") in backup nodes. The reason for our choice is that having copies in memory is usually useful when more complex queries and searches are to be made, since they can be done without having to access the disk, which is often the slowest part of any computer's memory access, especially if hard disks are involved. But we need to store data permanently to avoid loss in the event of a shutdown. The tables we use are of the 'bag' type. This type of table can have several entries with the same key, as long as the tuples are different.

Indexes have been defined for each table in the most important fields, in addition to the primary key. The data in these tables have been identified as data leading to competitive access. An example of this is the release of a (previously unavailable) copy of a book in the wishlist of several users and the subsequent reservation of the last available book copy by several users at the same time. In such a case, the use of a database such as Mnesia that manages competition and guarantees data consistency is therefore justified. The information on reservations and loans is subsequently removed from the respective tables and transferred to the MySQL database. This is only done for tuples describing terminated loans or reservations, thus not deemed necessary for the service. This transfer is triggered by reservation handling (confirm or delete) or loan handling (addition or termination) by admin user.

Transferring them to the MySQL database keeps the Mnesia database as lean as possible and speeds up access to it. The Mnesia database consists of n active nodes and n backup nodes. In our architecture, we have chosen to use n=3, but a different configuration (n=4, n=5, ...) would work without requiring any changes in the code. Each piece of information is distributed among these n replicas according to the MD5 hash of the ISBN of the book to which that information refers. The division into n replicas allows load balancing in an evenly distributed manner. The presence of backup nodes, maintained on machines other than the active nodes to which they refer, makes the database resistant to failures.

To know how to store tables on disk, how to load them, and what other nodes they should be synchronized with, Mnesia needs to have something called a "schema", holding all that information. By default, the schema will be created in the current working directory, wherever the Erlang node is running. We changed this by putting in the configuration file a "dir" variable used to set the Mnesia application "dir" variable. This allows us to pick where the schema will be stored. For the tables to be created on all nodes, Mnesia needs to run on all nodes. For the schema to be created, Mnesia needs to run on no nodes. You don't need to be connected to the other nodes involved in the same schema when creating it, but they need to be running. After creating the schema, we use a multicall to activate Mnesia in all the nodes. Note that there can be a noticeable delay between the time Mnesia starts and the time it finishes loading all tables from the disk, especially if they're large. We need to wait for any node. In our project, we chose to wait for at most 5 seconds or until the tables are available. Each node can identify itself as active or backup based on its passed configuration file.

3.3. Server Side

The application server-side components can be divided into two domains:

- ***Erlang servers***, takes care of synchronization issues derived from concurrency on book copies, leveraging the functionalities of Mnesia database
- ***Application server***, takes care of user interaction and handling of static information

The interaction between components is realized through message passing. This choice, mainly driven by the nature of Erlang functional programming language, allows to simplify the complex architecture obtained putting together all the necessary components that will be described in the following.

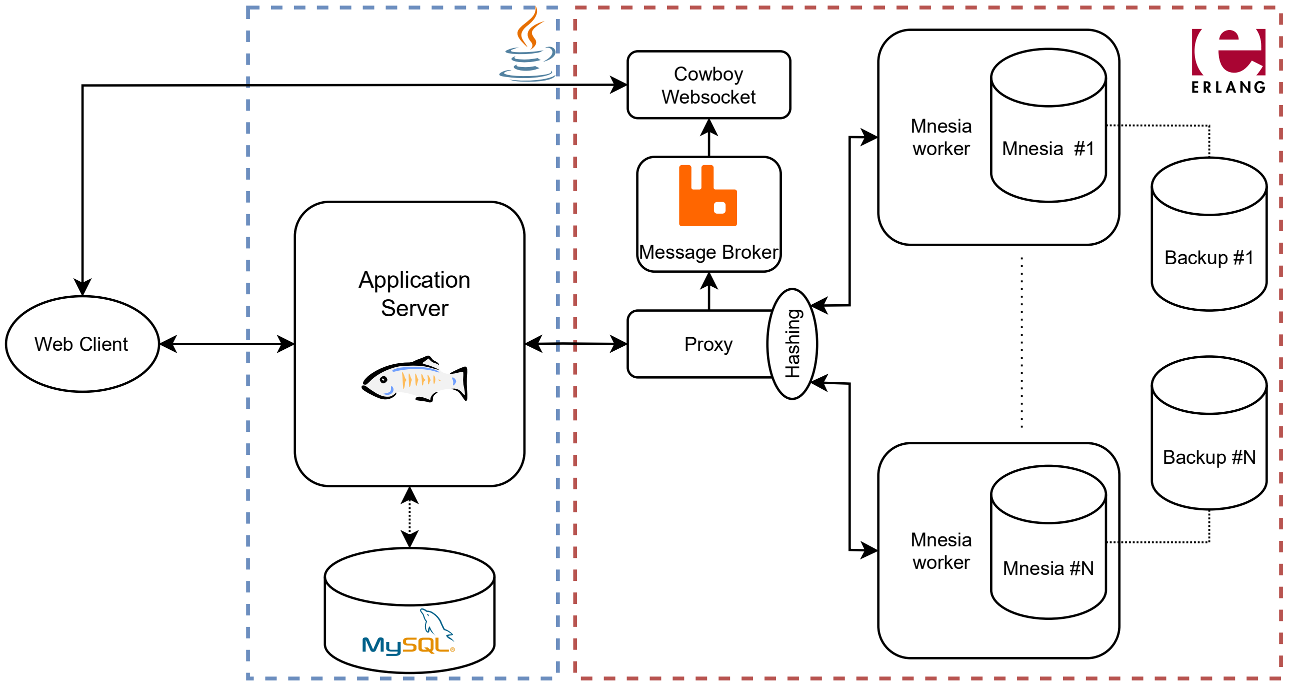


Figure 6: Application architecture

3.3.1 Erlang Server: Components

3.3.1.1 Mnesia and the structure of its modules

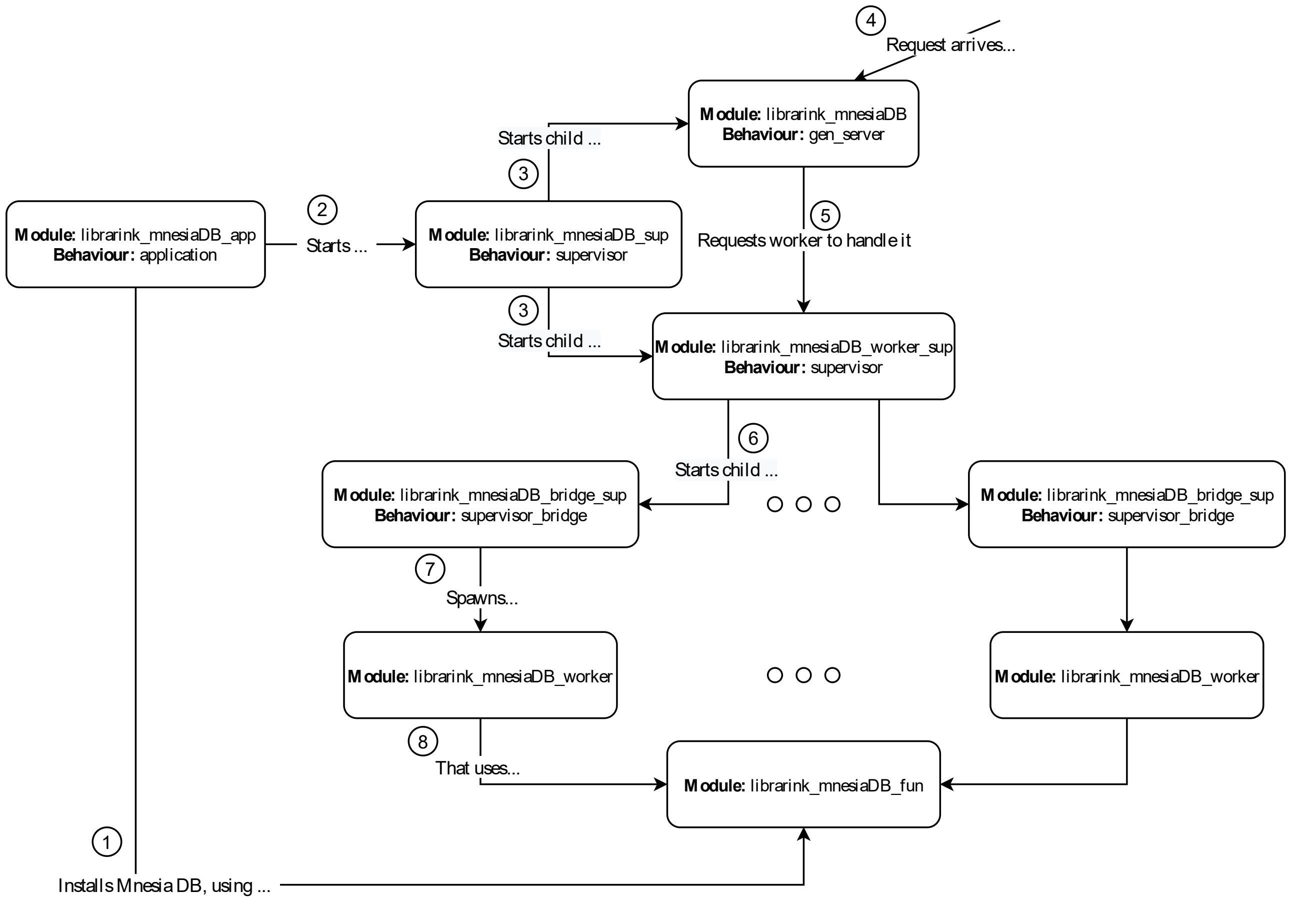

4
Request arrives...
Module: librarink_mnesiaDB
Behaviour: gen_server
Starts child ...
3
Module: librarink_mnesiaDB_app
Behaviour: application
2
Starts ...
Module: librarink_mnesiaDB_sup
Behaviour: supervisor
5
Requests worker to handle it
3
Starts child ...
Module: librarink_mnesiaDB_worker_sup
Behaviour: supervisor
6
Starts child ...
Module: librarink_mnesiaDB_bridge_sup
Behaviour: supervisor_bridge
Module: librarink_mnesiaDB_bridge_sup
Behaviour: supervisor_bridge
7
Spawns...
Module: librarink_mnesiaDB_worker
Module: librarink_mnesiaDB_worker
8
That uses...
Module: librarink_mnesiaDB_fun
1
Installs Mnesia DB, using ...

Figure 7: Mnesia modules and their ordered interactions

In the figure above, we can identify the main modules that manage the reception of access requests to the Mnesia database and interaction with the database itself. We distinguish the following modules:

1) **librarink_mnesiaDB_app:** this is the entry point. The application starts a supervisor that will manage gen_server and processes dynamically spawned to handle operations on MnesiaDB. Whenever the application is started using application:start/[1,2], it should start the processes of the application. Because the application is structured according to the OTP design principles, this means starting the top supervisor of the tree. The only operations needed are to create schema, activate Mnesia DB, create tables and then launch the MnesiaDB supervisor.

This is used also in case of a takeover of the active node. Taking over is the act of a dead node coming back from the dead, being known to be more important than the backup nodes (maybe it has better hardware) and deciding to run the application again. This is usually done by gracefully terminating the backup application and starting the main one instead.

2) **librarink_mnesiaDB_sup:** Module implementing librarink_mnesia top level supervisor. Whenever a supervisor is started using supervisor:start_link/[2,3], the init/1 function is called by the new process to find out about the restart strategy, maximum restart frequency, and child specifications. This is needed to initialize the supervisor process. The initial configuration includes two children spawned:

1) the mnesiaDB gen_server to handle incoming request

2) the supervisor that handles the worker processes.

In case of a crash of supervised processes, they will be replaced with others, following the one-for-one semantic.

3) **librarink_mnesiaDB_worker_sup:** Module implementing librarink_mnesia higher worker level supervisor. Launched by the top-level supervisor, it has no default supervised child. This module is in charge to start new supervised children, i.e. the Librarink supervisor bridge associated with a master process, for each newly received request. The supervisor bridge will be used to supervise a worker.

4) **librarink_mnesiaDB:** Module implementing librarink_mnesia gen_server that receives all the requests from the proxy and asks the worker supervisor to fetch a new process to handle them. This gen_server only admits requests made via gen_server:call/2,3 and therefore synchronous.

5) **librarink_mnesiaDB_bridge_sup:** Module implementing librarink_mnesia worker level supervisor. It has a default child supervised. In its init/1 function, it spawns a new worker process to handle a specific request. The worker processes don't implement any OTP behavior so the linking with OTP hierarchy must be done exploiting a dedicated "bridge"

6) **librarink_mnesiaDB_worker:** Module that handles requests received from the proxy, directly performing operations on the MnesiaDB instance by invoking the functions defined in librarink_mnesiaDB_fun. Once the elaboration is concluded the process is destroyed.

7) **librarink_mnesiaDB_fun:** Module that contains all the allowed operations on Mnesia DB. One function is the one in charge of creating Mnesia schema, activate Mnesia in the active and backup nodes and at the end create all the tables. Other functions are used for CRUD operations.

3.3.1.2 Proxy

All the requests handled by the Erlang server are received by a proxy that is in charge to forward them to the correct Mnesia server instances and later send back the response the original client, namely the application server that needs the information to process original client request.

This component is the key to enable to main features of our Erlang server:

- ***Load balancing***, the requests are not directly sent to a high performance "super-node" but are shared among the multiple available instances. In this way each instance has to handle a smaller amount of load, increasing overall performance exploiting horizontal scaling.
- ***Fault tolerance***, the selected instance of Mnesia server may be temporarily not available due to a local system crash or even to a hardware failure. If proxy detects that situation the request will be not forwarded to the failing instance, but to the associated backup replica.

The proxy allows to configure several parameters, obtaining a pretty general component. The configurable number of Mnesia instances and the number of its associated backup replicas allows to tune the system in order to meet the requirements in terms of performance and resiliency.

Architecture details

More in details the proxy implements an OTP application, composed by:

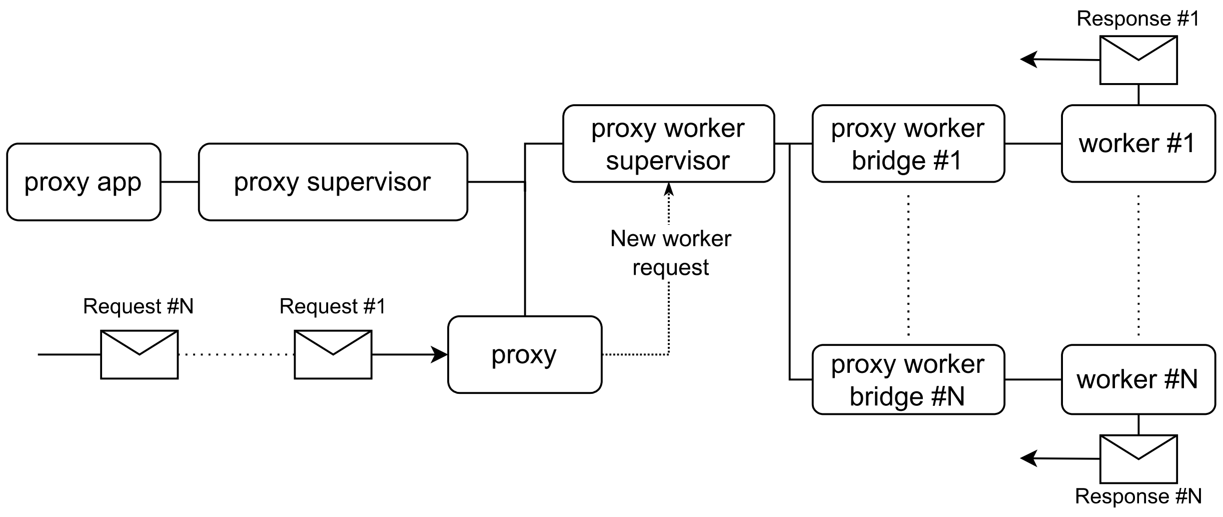


Figure 8: Proxy architecture

- ***Proxy app*** (*OTP/Application behavior*), represents the entry point of the application on which are defined the starting/stopping functionalities
- ***Proxy supervisor*** (*OTP/Supervisor behavior*), starts and supervises the core of the application, composed by two long-term processes. In case of failure the supervisor restarts only the failing process up to a maximum number of times on certain time interval
    - ***Proxy*** (*OTP/Genserver behavior*), represents the endpoint on which requests are received. All the requests are received by this process, therefore it cannot directly handle them, but outsources the handling to a dedicated process to reduce the load on this component
    - ***Proxy worker supervisor*** (*OTP/Supervisor behavior*), spawns dynamically new "worker" processes on proxy subcomponent demand. The nature of requests is such that failures can be tolerated, so the policy implemented by supervisor is to not restart worker process
        - ***Proxy worker bridge*** (*OTP/Supervisor Bridge behavior*), the worker processes don't implement any OTP behavior so the linking with OTP hierarchy must be done exploiting a dedicated "bridge"
        - ***Proxy worker***, it's only dedicated to the handling of request and associated response. Once the elaboration is concluded the process is destroyed.

The workflow implemented by the worker is the following:

1. Check request format
2. Determines if request can be handled by only one instance of Mnesia server or the request must be broadcasted to all instances, in case of request that requires to obtain information about more than one book
3. In case of single instance, it is identified by the so call "*groupID*", namely the MD5 hash of book ISBN, interpreted as unsigned integer, modulus number of Mnesia instances. The result is the entry index of the list of instances
4. Retrieve instance(s) location and checks connectivity, in case of issues it tries the connectivity with the backup replicas
5. Forward request to instance(s) and wait for response(s). In case of multiple responses, they are merged to form a single response. To simplify the interaction with external non-erlang nodes, the response is encoded in JSON format before sending it back to client
6. If the operations modify the number of available copies of a book, that information is published on message broker to live update all interested running clients

3.3.1.3 *Cowboy Websocket*

One of the requirements of our application is to keep the counter of available book copies updated for the users that are displaying such information. This requirement needs a communication from Erlang server toward the clients, therefore in reverse order from that of a classic web-application. That communication is implemented via a web-socket. Although application server is able to realize such communication, we have decided to implement web-socket directly in the Erlang server level. This choice is driven by the fact that all needed information, i.e. the number of book copies, are available on Mnesia instances, so we can save the application server level and reduce latency of live updates.

Architecture details

More in details we exploit the position of proxy, that knows all the operations performed on Mnesia instances, and a message broker to realize an indirect communication between the latter and the cowboy server.

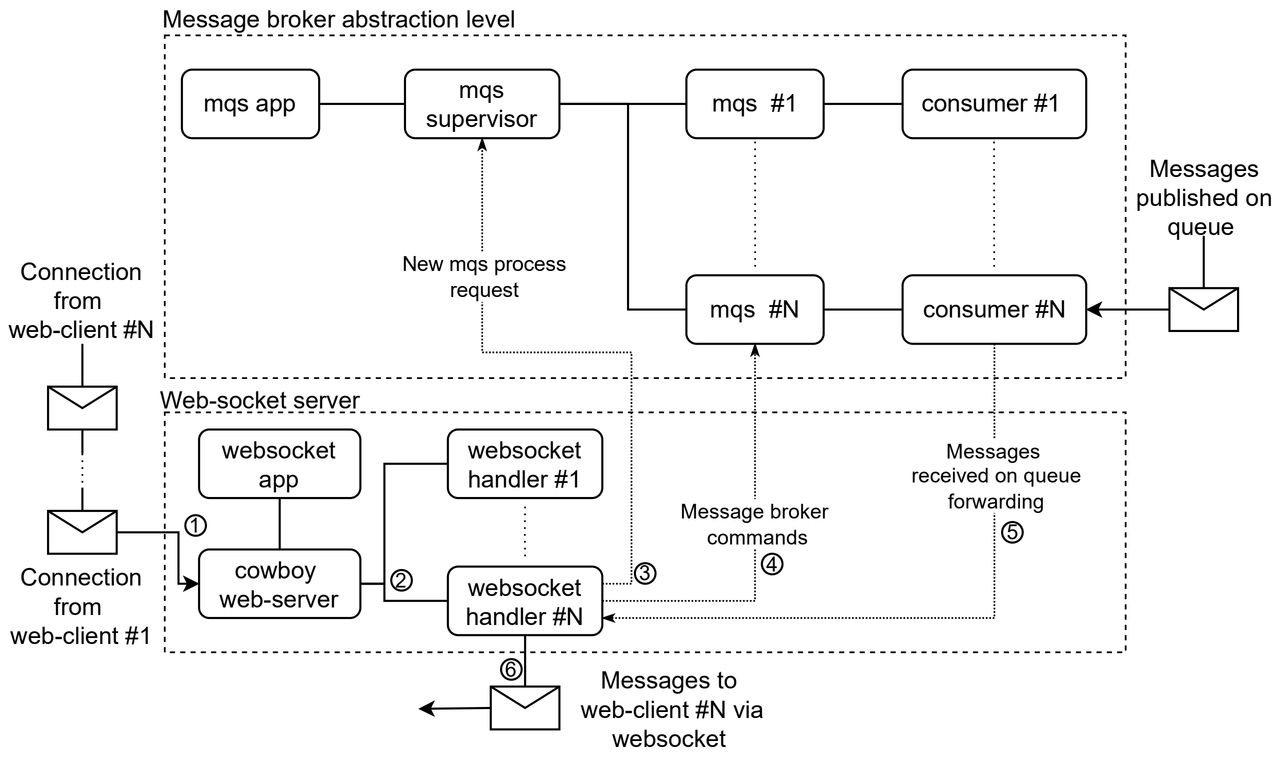


Figure 9: Websocket and MQS architecture (some arrows are missing to keep the image understandable)

The web-socket server is composed by:

- ***Websocket app (OTP/Application behavior)***, represents the entry point of the application in which the starting/stopping functionalities are defined. During the starting phase it configures the endpoints and relative handlers of the web server and runs such web server
    - ***Cowboy web server***, that component is an open-source web server that implements, among classic functionalities, the web-socket function.
        - ***Websocket handler***, the tasks realized are customized during the initial configuration. Each web-socket connection is handled by a dedicated handler process that will stay alive for the entire duration of the connection.

The complexity to be addressed necessary to receive messages by message broker is abstracted by a second application, the “Message Queueing System” application, MQS for short. That application is shipped together with the web-socket server and handles completely the interactions with message broker, translating request from web-socket application in AMQP[1] requests sent to message broker and forwarding messages received on queues to web-socket application.

MQS application implements an OTP application:

- ***MQS app (OTP/Application behavior)***, represents the entry point of the application in which the starting/stopping functionalities are defined.
    - ***MQS supervisor (OTP/Supervisor)***, spawns dynamically new MQS processes that will be linked to the websocket handler. In case of failure the process will be restarted and the handler, informed by the link, will provide the missing configuration. The configuration cannot be provided by the supervisor because it depends on the ISBN on which users are currently interested, so must be dynamic
        - ***MQS (OTP/Genserver)***, takes care of the interaction with message broker, configuring the bindings of the queue on demand by the websocket handler associated. It also starts and links to a new process that will hang on waiting for messages on queue
            - ***Consumer***, the only purpose of this process is to implement a loop composed by blocking reception of a message on queue and forwarding of the latter to the websocket handler.

The websocket handler implements the following workflow:

1. On connection establishment, requests to MQS supervisor to spawn a new MQS process that will take care of all interactions with message broker as described above. The handler and MQS process will be linked together to manage possible crashes
2. Once the connection is established and the MQS process is ready, the handler waits for events:
    - Reception from web client of a list of book ISBNs that represents the books on which user is interested to receive updates in case of modification of copy counter. The handler requests to its MQS process to bind the message broker queue on the exchanges identified by book ISBNs in the received list
    - Reception of an update message from MQS consumer that will be forwarded to the web client, encoded in JSON format to allow an easier interaction with non-Erlang nodes
    - Crash of MQS process that will be notified to the handler that will request to MQS supervisor the spawning of new process
3. When the client decides to leave the connection, the websocket will terminate together with the MQS process(es) linked.

More details about message broker queues can be found on *chapter 4*.

3.3.2 Application Server

The entry point of our application resides on Glassfish application server, on which we have defined two Enterprise Java Beans and a web application, that provides the possibility to the users to interact with internal components.

3.3.2.1 EJBs

a) Java-Erlang interaction

The web application is handled by application server, however some pages require information saved on Mnesia database. The information can be retrieved only if web application and Erlang server can interact

[1] Advanced Message Queueing Protocol, the one implemented by RabbitMQ, the message broker used in our application

each other, sending request to the proxy that will take care of forwarding to the correct Mnesia instance(s) and will send back the response.

That interaction requires some complex operation, out of scope of a web application. Therefore, exploiting the functionalities of an application server, we have implemented a stateless Enterprise Java Bean, name "*ErlangClient*". That EJB is able to translate requests from Java to Erlang, relying on JInterface library, able to create a full Erlang node inside Java environment. Via that node is possible to establish connection with other real Erlang node, also leveraging EPMD component, and exchange messages.

The interface of such EJB allows to pass parameters as common Java strings and the result is translated to a Java strings (represented JSON) or custom Java classes (list of DTO), hiding all details to the consumer.

b) Java Persistence API

The application is based on information stored into relational databases: MySQL and Mnesia.

The connection with the two databases is handled following different approaches:

- Mnesia connection is handled via dedicated Erlang servers accessed by application servers, as described above
- MySQL connection is handled leveraging application server functionalities, defining a dedicated resource on Glassfish domain

The interaction with MySQL database relies on Hibernate framework, that allows to map each entity on a dedicated Java object (*ORM*). The query language is slightly different from the standard, but the advantage is given by the fact that each operation returns a plain object (or list of objects) that can be used directly by Java without parsing operations.

All the operations and configurations are embedded inside a stateless Enterprise Java Bean, called "LibrarinkRemote". That EJB allows easy data retrieval in response to web client request, hiding all details of the MySQL interactions.

c) WebApp

The web app is the real entry point from the user point of view, in which are defined all the endpoints accessible via web. The implementation of such web-application is based on dynamic JSP and servlet executed inside the application server obtaining simple http pages or JSON encoded data to send in response of client requests.

I. Servlet filter

Some servlet filters are also implemented in the project. These are applied so that requests received can be logged and we can check that

- Requests to pages other than the login and signup pages come from registered and logged-in users. If this is not the case, these users will be redirected to the login page.
- Requests to the login and signup page come from users who are not logged in. If this is not the case, these users will be redirected to the homepage.
- Logged user is not trying to access to admin pages or viceversa; If yes, redirect the user to the relative homepage.

II. Asynchronous request servlet

The structure of web application is defined by JSP pages built dynamically by application server, however the client makes extensive use of asynchronous requests to load data into the page.

This choice is driven by the fact that page structure is the same for a major part of the time, while the content can change several times and each modification requires to request data to the application server, e.g. show details of a selected book.

In the application server we have defined an endpoint that is able to receive all asynchronous requests, execute them and send back a JSON encoded response for the requested operation. This servlet exposes a single endpoint, the operation to be performed is identified by parameters of the asynchronous request.

This solution is exploited both from common user pages and admin pages, so inside the servlet has been defined a check on current requestor to limit the privileges of non-admin users.

3.4. Client Side

The users can access the application through a web page previous registration. The graphical interface is realized mainly through Bootstrap and CSS, while the behaviour is defined via JavaScript linked to the page.

Some aspects worth mentioning:

- **Asynchronous requests and caching**: some user information is required during all the entire lifespan of the session. Each time user accesses a page that requires that information, the web-client should request it to the application server, increasing the overall response time. In order to reduce this cost, the web-client after the first request save data in the local session cache of the browser, saving successive requests. This workaround is enabled by the fact that the information changes only subsequently to a user action, so the modification can be mirrored on local copy after the positive feedback of remote operation. The information cached locally are:
    - Book wishlist
    - Reserved books
    - Lent books[2]
- **Live update and notifications:** When the user accesses a book page, he registers to receive updates on the number of copies via WebSocket (live update). In addition, if the user has a book with zero copies on the wishlist and one becomes available, he is notified via RabbitMQ using cowboy WebSocket (as explained above).
- **Book paging**: Given the large number of books in the database, it was decided to implement a paging technique to return the searched books to the user by organising them into different pages, which can be accessed via the menu at the bottom of the page. This improves the server's response time to the user, avoids the server handling an excessive amount of data per request and has a positive impact on the user's navigation. Each page will contain 50 books in our case.

[2] Actually the lent books are controlled by librarian, by the way we suppose that at that time the user is in the library to get the book

4. Communication and Synchronization issues resolution

4.1. Inter-component communication

The application, as described above, is composed by several components, possibly distributed on different physical machines. The communication between components is a crucial aspect that requires different approaches according to the needs of all components.

We can identify different kinds of communication:

- **Direct synchronous communication**, Erlang proxy workers forward the request to Mnesia instance and wait for the response. This type of communication is realized via the standard message passing implemented by OTP middleware, i.e. through bang operator (or higher level of abstraction functions, such as the ones offered by *gen_server*).
    - **Multicast synchronous communication**, the case of forwarding request to multiple Mnesia instances requires the multicast towards multiple destinations, but the requirements of communication are still the same, the sender will block until the reception of all responses[3].
- **Websocket**, standard communication between web client and server is unidirectional, i.e. the client performs a request and the servers send back the response. This type of communication is a limit when client wants to receive as soon as possible some information which produce time is unknown. To avoid polling the server with requests we have introduced the websocket protocol, as described above. The communication is now bidirectional, the reception of a message triggers some callback function that will handle the information received, e.g. showing notification in dedicated area or updating copy counter.
- **Indirect communication**, Erlang proxy server needs to send updates of book copies to the Cowboy websocket handler processes in order to let the information reach web-client through websocket. Websocket handlers are spawned dynamically and are not registered on EPMD daemon, so the Proxy can't send direct messages. Moreover, is necessary to adopt a mechanism that allows to identify only the processes, among all the handlers, that are interested on updates. All these requirements lead to the introduction of a message broker that can connect two parties that don't know each other.
  The message broker used in our application is RabbitMQ, because of the easy integration with Erlang components. Each websocket handler declares an exclusive queue and binds it to one or more fanout exchange, identified by name. To associate an exchange to an update of a certain book, the exchange name is given by the book ISBN.
  In order to better understand this passage, we provide a simple example:

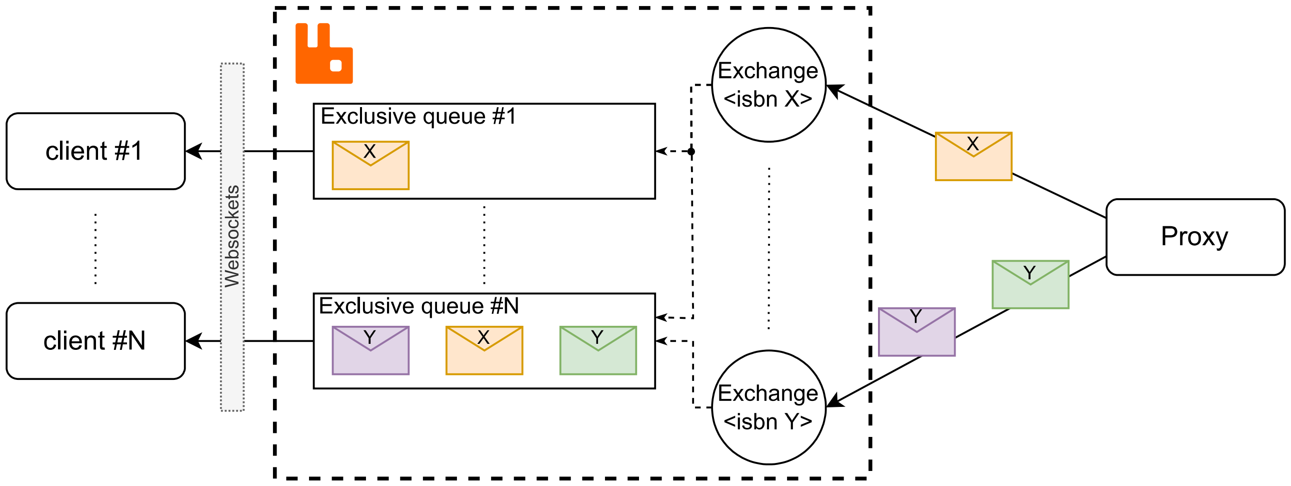


Figure 10: RabbitMQ queues details

[3] Or until the timeout expiration

Aside all these types of communication we have to mention the classic approach of HTTP protocol to pass web pages between web client and application server and the relative asynchronous requests.

4.2 Mnesia transactions

In the design and development phases of the application, we must handle concurrency about reservations and loan transactions on the same book. The number of available copies must be consistent after each operation. For this reason, Mnesia transactions were used where necessary, i.e. for operations that risked compromising the consistency of the data or could lead to inconsistent data readings. Transactions allow doing multiple operations on one or more tables as if the process doing them were the only one to have access to the tables. This is very important in case of mixed readings and writings as part of a single unit. A Mnesia transaction allows running a series of database operations as a single functional block. The whole block will run on all nodes or none of them; it succeeds entirely or fails entirely. When the transaction returns, we're guaranteed that the tables were left in a consistent state and that different transactions didn't interfere with each other, even if they tried to manipulate the same data. This type of activity context is partially asynchronous: it will be synchronous for operations on the local node, but it will only wait for the confirmation from other nodes that they will commit the transaction, not that they have done it. In case of problems, possibly due to failures in the network or hardware, the transaction will be reverted at a later point in time. Mnesia can run nested transactions avoiding deadlock.

5. Deployment configuration

The application components to be deployed are:

- Glassfish application server
- Erlang server
    - Proxy server
    - Mnesia database (3 instances + 3 backup replicas)
    - Cowboy server
- MySQL DB server
- RabbitMQ server

The components and subcomponents are distributed and replicated among the available machines.

The current configuration uses 6 Ubuntu machines[4], available at the following addresses:

- 172.18.0.28
- 172.18.0.29
- 172.18.0.30
- 172.18.0.31
- 172.18.0.32
- 172.18.0.33

Distribution

The components are distributed in order to improve performance, splitting the load on multiple machines. There are more components than available machines, therefore some components are place on same machine. However, the choices are not random, but take into account the needs of communication and the load that a component is supposed to handle.

Cowboy websocket server and RabbitMQ are on the same physical machine, but different Erlang node, reducing the latency of communication.

Glassfish AS and MySQl DB are place together in order to minimize client response time.

Replication

The only components that are replicated for data availability and fault tolerance are the Mnesia DB replicas. The distribution is made in such a way that replica and relative backup are on different physical machine, so in case of hardware failure of a physical node data will be still available.

Deployment configuration

Our final deployment is configured according to the following schema:

[4] The machines are actually provided through virtualization

Component	IP address	Erlang Node
Glassfish AS	172.18.0.28	-
MySQL DB Server	172.18.0.28	-
Proxy Server	172.18.0.29	proxy
Cowboy websocket server	172.18.0.30	websocket
RabbitMQ server	172.18.0.30	rabbitMQ
Mnesia instance #1	172.18.0.31	mnesia_active1
Mnesia instance #2	172.18.0.32	mensia_active2
Mnesia instance #3	172.18.0.33	mnesia_active3
Mnesia backup #1	172.18.0.32	mnesia_backup1
Mnesia backup #2	172.18.0.33	mensia_backup2
Mnesia backup #3	172.18.0.31	mnesia_backup3

6. User Manual

6.1 Login Page

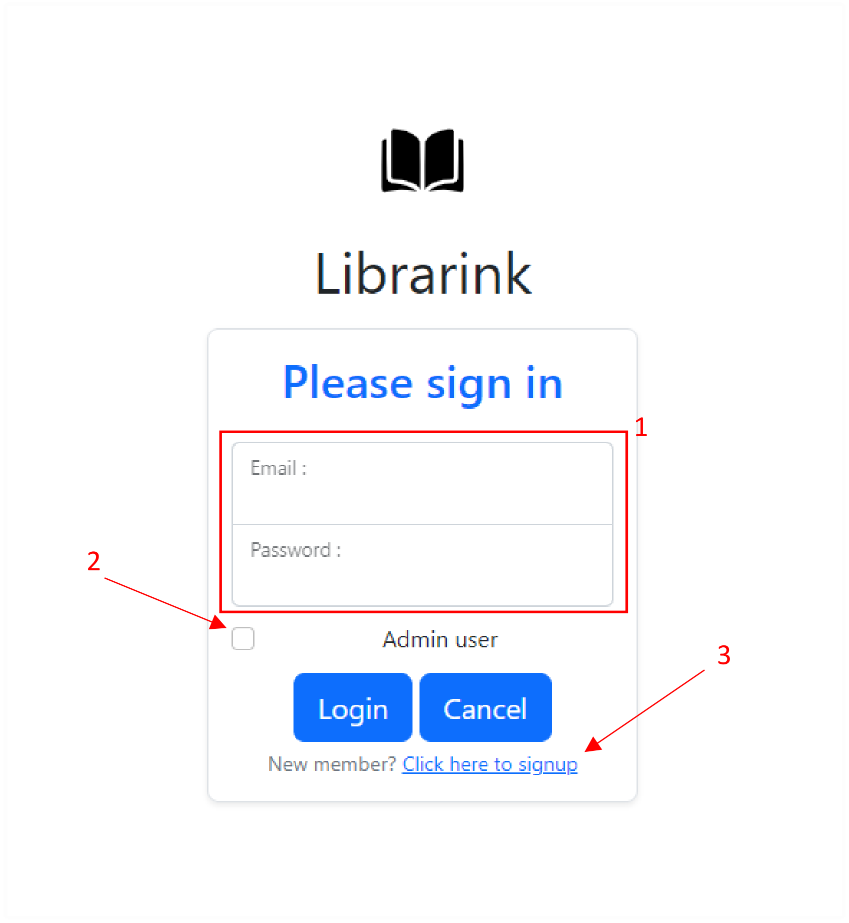
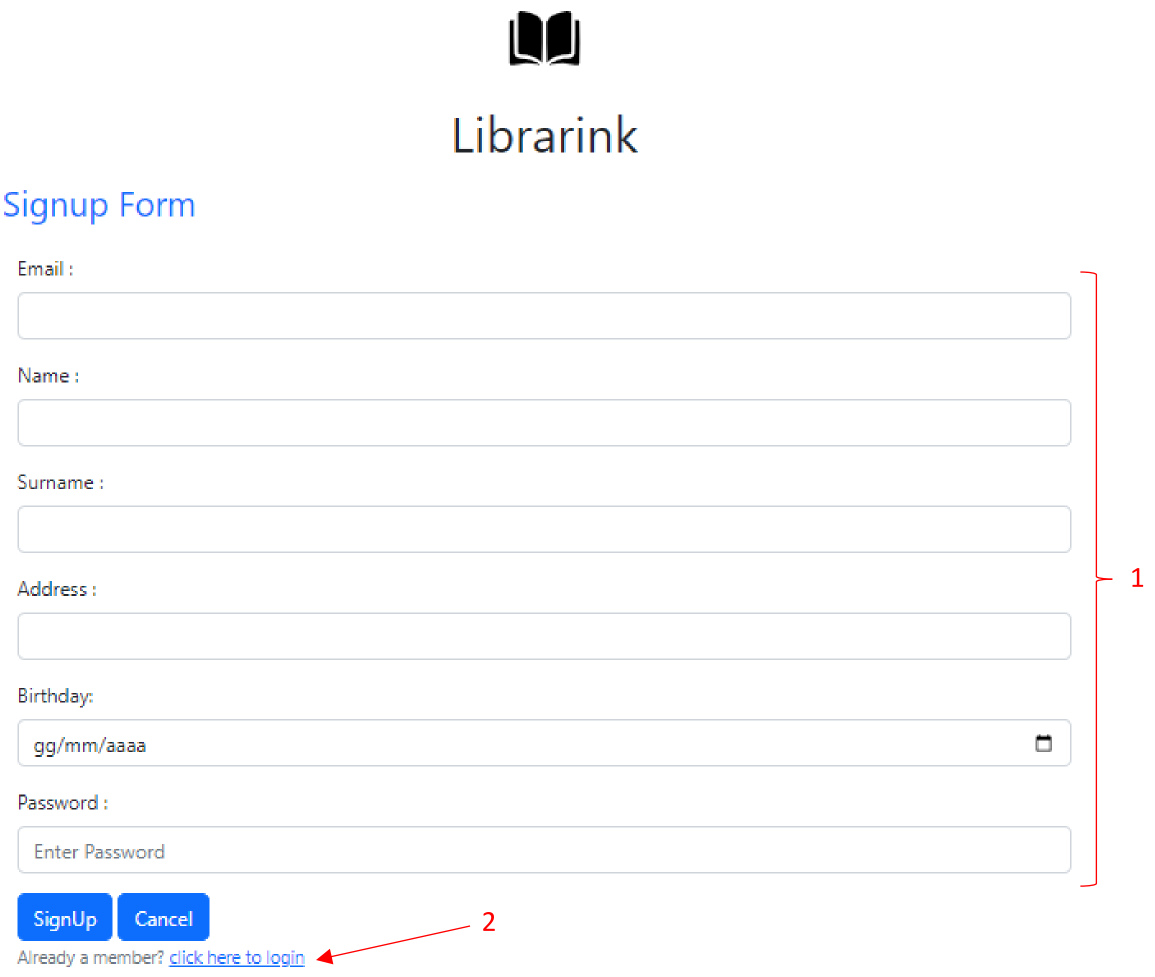


Figure 11: Login page

Legend:

1) Email and password fields required from the user for authentication
2) If the user is an administrator, tick this box before logging in
3) Link to the page by which an ordinary user can register on the site

6.2 Signup Page


Librarink

Signup Form

Email :

Name :

Surname :

Address :

Birthday:

gg/mm/aaaa

Password :

Enter Password

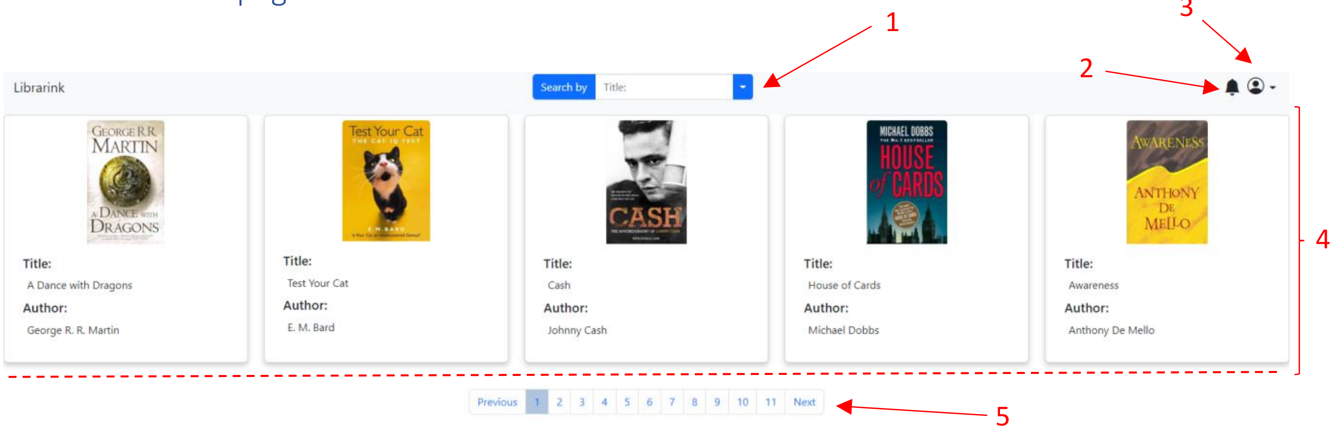
Already a member? click here to login

Figure 12: Signup page

Legend:

1) Form to be filled in for the user to register to the service
2) Link to the login page (see figure 11)

6.3 User Homepage



Title: A Dance with Dragons
Author: George R. R. Martin

Title: Test Your Cat
Author: E. M. Bard

Title: Cash
Author: Johnny Cash

Title: House of Cards
Author: Michael Dobbs

Title: Awareness
Author: Anthony De Mello

Figure 13: User homepage divided into two sections. At the top is the part containing the user bar and books. At the bottom the menu to navigate to previous or next pages.

Legend:

1) Field to select the search metric and enter all or part of the value (name, isbn, ..) of what you are searching for (see figure 14)
2) Section for user notifications (see figure 15)
3) Menu to log out or go to the user page.
4) Body of the page with the books resulting from the search
5) Menu to navigate between pages. Located at the bottom of the page

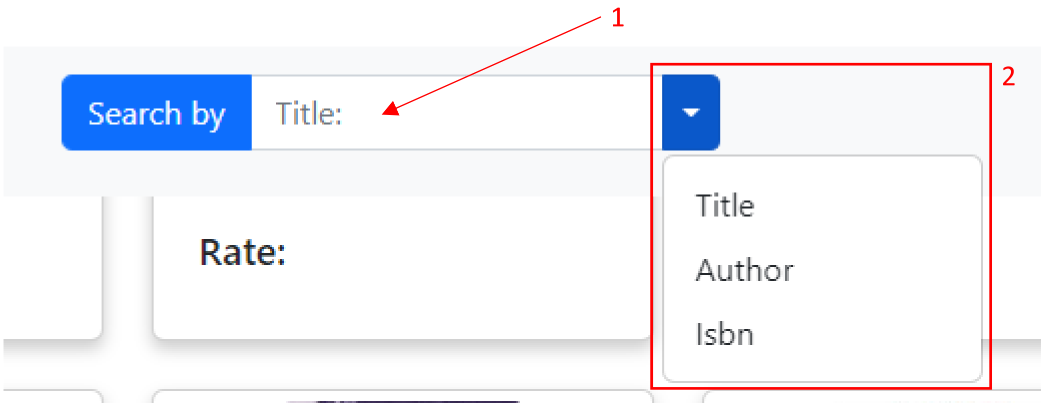
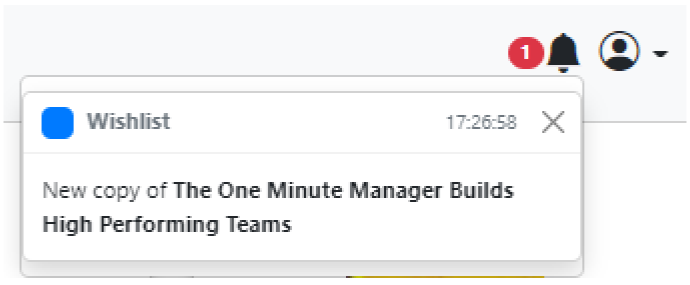


Figure 14: Field to select the search metric and enter all or part of the value (name, isbn, ..) of what you are searching for

Legend:

1) Field for entering the searched value (name, isbn, ...)
2) Field for selecting the search metric

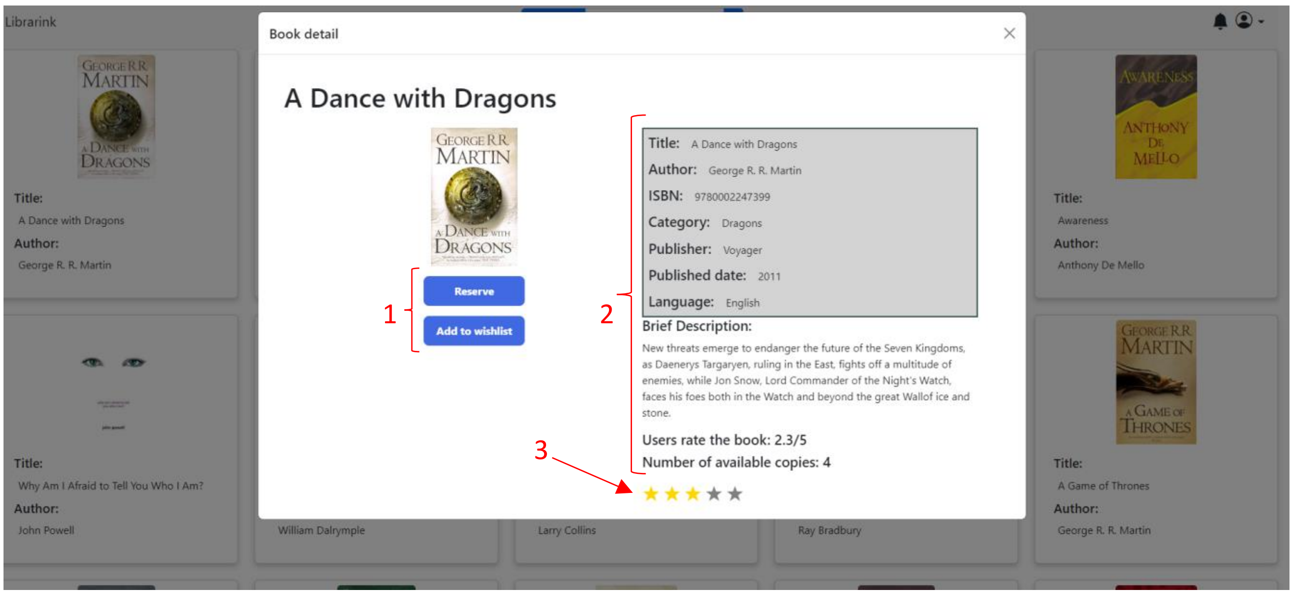


Wishlist 17:26:58

New copy of **The One Minute Manager Builds High Performing Teams**

Figure 15: Section for user notifications

6.4 Book's Details



A Dance with Dragons

Title: A Dance with Dragons
Author: George R. R. Martin
ISBN: 9780002247399
Category: Dragons
Publisher: Voyager
Published date: 2011
Language: English

Brief Description:
New threats emerge to endanger the future of the Seven Kingdoms, as Daenerys Targaryen, ruling in the East, fights off a multitude of enemies, while Jon Snow, Lord Commander of the Night's Watch, faces his foes both in the Watch and beyond the great Wallof ice and stone.

Users rate the book: 2.3/5
Number of available copies: 4

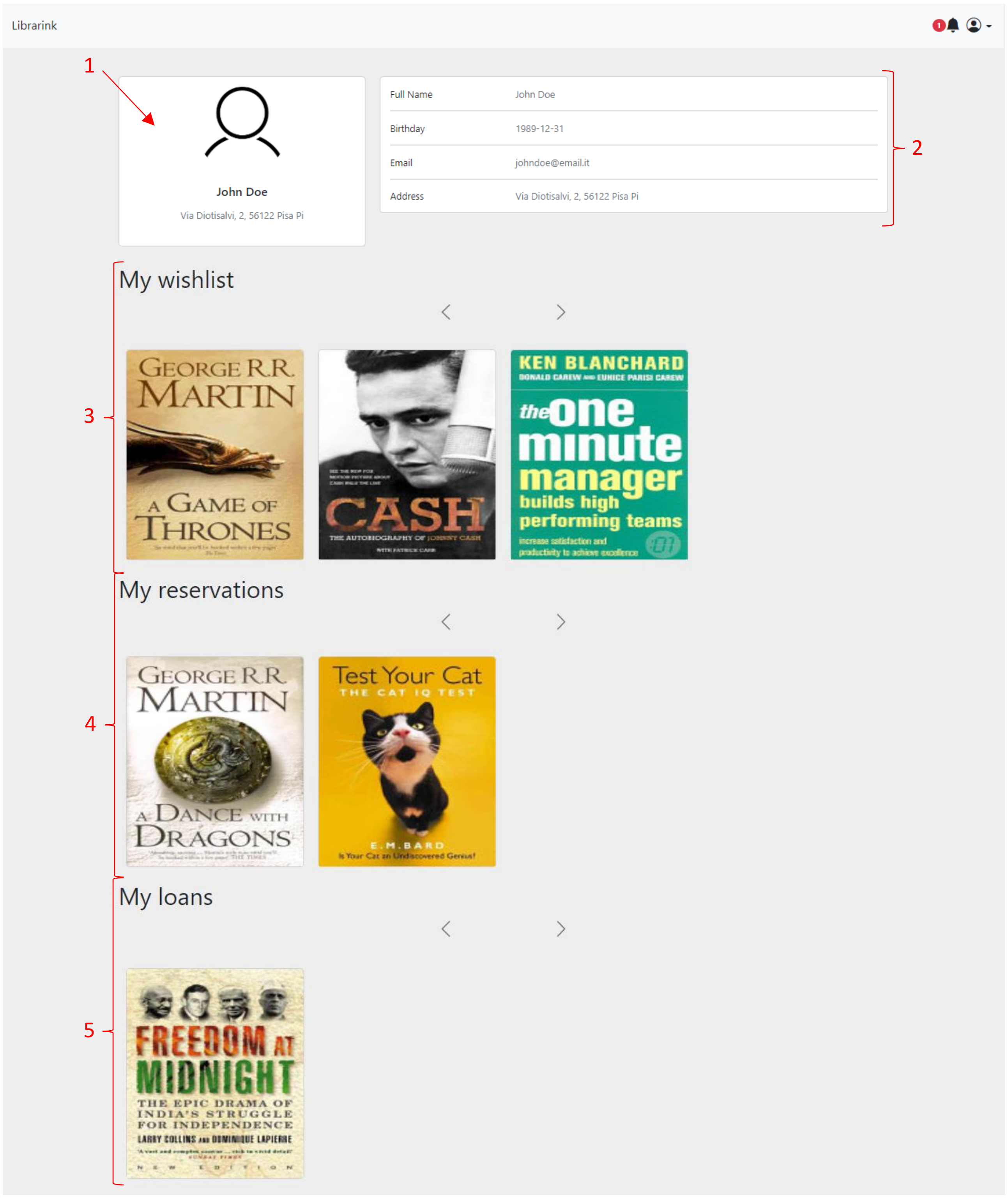
★★★☆☆

Figure 16: Page describing the details of a book. Appears after clicking on the book

Legend:

1) Button section with which you can put a book on a wishlist, reserve a book or cancel a reservation
2) Section containing any useful information about a book
3) Section where you can rate the book

6.5 User’s Page



Librarink

John Doe

Via Diotisalvi, 2, 56122 Pisa Pi

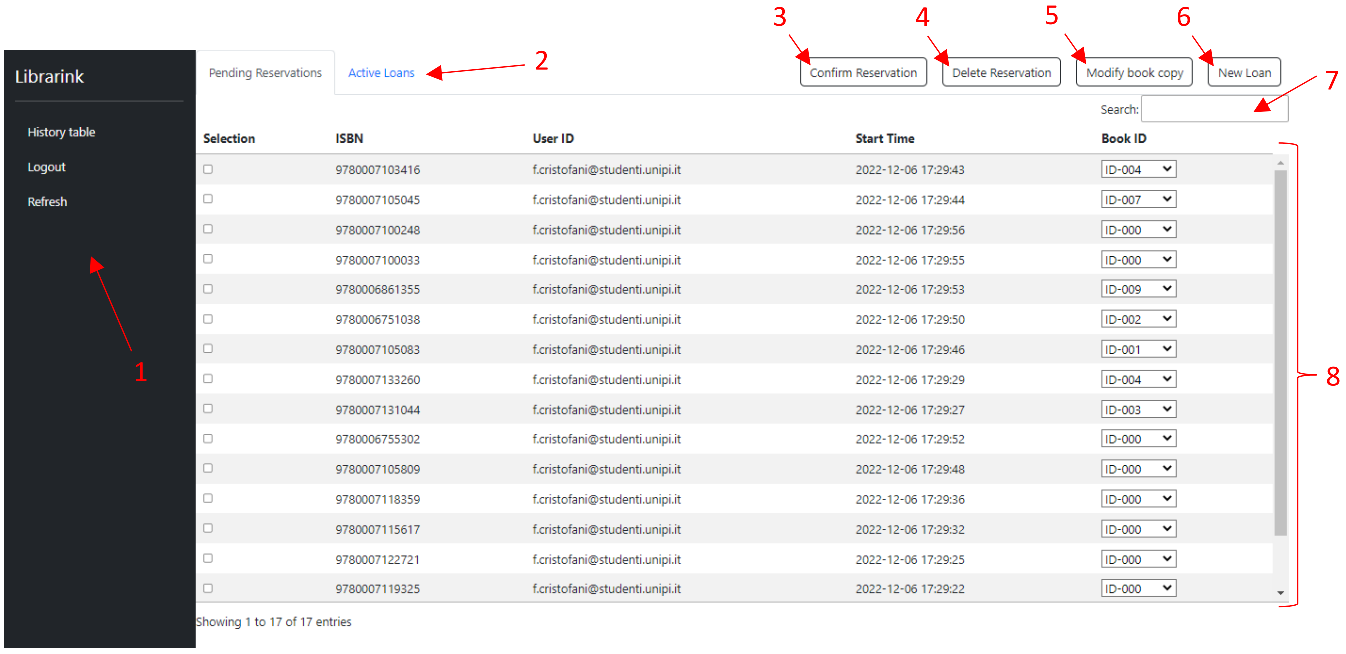
Full Name	John Doe
Birthday	1989-12-31
Email	johndoe@email.it
Address	Via Diotisalvi, 2, 56122 Pisa Pi

Figure 17: User's page with his personal information, his wishlist and a summary of his pending reservations and pending loans

Legend:

1) User summary card
2) User's personal information
3) Scrolling list of the user's wishlist
4) Scrolling list of user's pending reservations
5) Scrolling list of user's pending loans

6.6 Pending Reservations Admin Page



Selection	ISBN	User ID	Start Time	Book ID
☐	9780007103416	f.cristofani@studenti.unipi.it	2022-12-06 17:29:43	ID-004
☐	9780007105045	f.cristofani@studenti.unipi.it	2022-12-06 17:29:44	ID-007
☐	9780007100248	f.cristofani@studenti.unipi.it	2022-12-06 17:29:56	ID-000
☐	9780007100033	f.cristofani@studenti.unipi.it	2022-12-06 17:29:55	ID-000
☐	9780006861355	f.cristofani@studenti.unipi.it	2022-12-06 17:29:53	ID-009
☐	9780006751038	f.cristofani@studenti.unipi.it	2022-12-06 17:29:50	ID-002
☐	9780007105083	f.cristofani@studenti.unipi.it	2022-12-06 17:29:46	ID-001
☐	9780007133260	f.cristofani@studenti.unipi.it	2022-12-06 17:29:29	ID-004
☐	9780007131044	f.cristofani@studenti.unipi.it	2022-12-06 17:29:27	ID-003
☐	9780006755302	f.cristofani@studenti.unipi.it	2022-12-06 17:29:52	ID-000
☐	9780007105809	f.cristofani@studenti.unipi.it	2022-12-06 17:29:48	ID-000
☐	9780007118359	f.cristofani@studenti.unipi.it	2022-12-06 17:29:36	ID-000
☐	9780007115617	f.cristofani@studenti.unipi.it	2022-12-06 17:29:32	ID-000
☐	9780007122721	f.cristofani@studenti.unipi.it	2022-12-06 17:29:25	ID-000
☐	9780007119325	f.cristofani@studenti.unipi.it	2022-12-06 17:29:22	ID-000

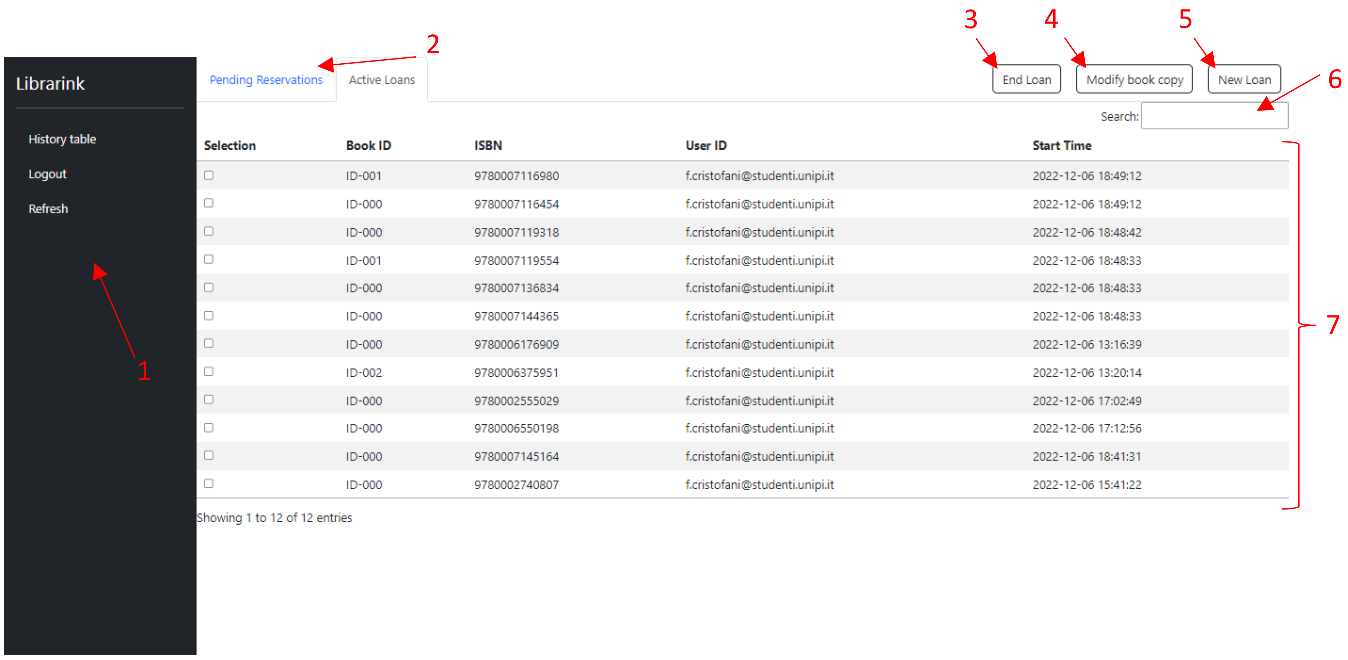
Showing 1 to 17 of 17 entries

Figure 18: Page where the administrator can check/confirm/delete pending reservations, modify book copies, or enter a new loan

Legend:

1) Menu through which the administrator can view the loan or reservation history table (see figures 20 and 21), logout or reload page content
2) Link to view table of active loans (see figure 19)
3) Button to confirm one or more previously selected reservations
4) Button to delete one or more previously selected reservations
5) Button to change the number of copies of a book (see figure 22)
6) Button to insert a new loan (see figure 23)
7) Field to search by ISBN for items in the displayed table
8) Table containing information for pending reservations. The leftmost column is used for selection

6.7 Active Loans Admin Page



Selection	Book ID	ISBN	User ID	Start Time
☐	ID-001	9780007116980	f.cristofani@studenti.unipi.it	2022-12-06 18:49:12
☐	ID-000	9780007116454	f.cristofani@studenti.unipi.it	2022-12-06 18:49:12
☐	ID-000	9780007119318	f.cristofani@studenti.unipi.it	2022-12-06 18:48:42
☐	ID-001	9780007119554	f.cristofani@studenti.unipi.it	2022-12-06 18:48:33
☐	ID-000	9780007136834	f.cristofani@studenti.unipi.it	2022-12-06 18:48:33
☐	ID-000	9780007144365	f.cristofani@studenti.unipi.it	2022-12-06 18:48:33
☐	ID-000	9780006176909	f.cristofani@studenti.unipi.it	2022-12-06 13:16:39
☐	ID-002	9780006375951	f.cristofani@studenti.unipi.it	2022-12-06 13:20:14
☐	ID-000	9780002555029	f.cristofani@studenti.unipi.it	2022-12-06 17:02:49
☐	ID-000	9780006550198	f.cristofani@studenti.unipi.it	2022-12-06 17:12:56
☐	ID-000	9780007145164	f.cristofani@studenti.unipi.it	2022-12-06 18:41:31
☐	ID-000	9780002740807	f.cristofani@studenti.unipi.it	2022-12-06 15:41:22

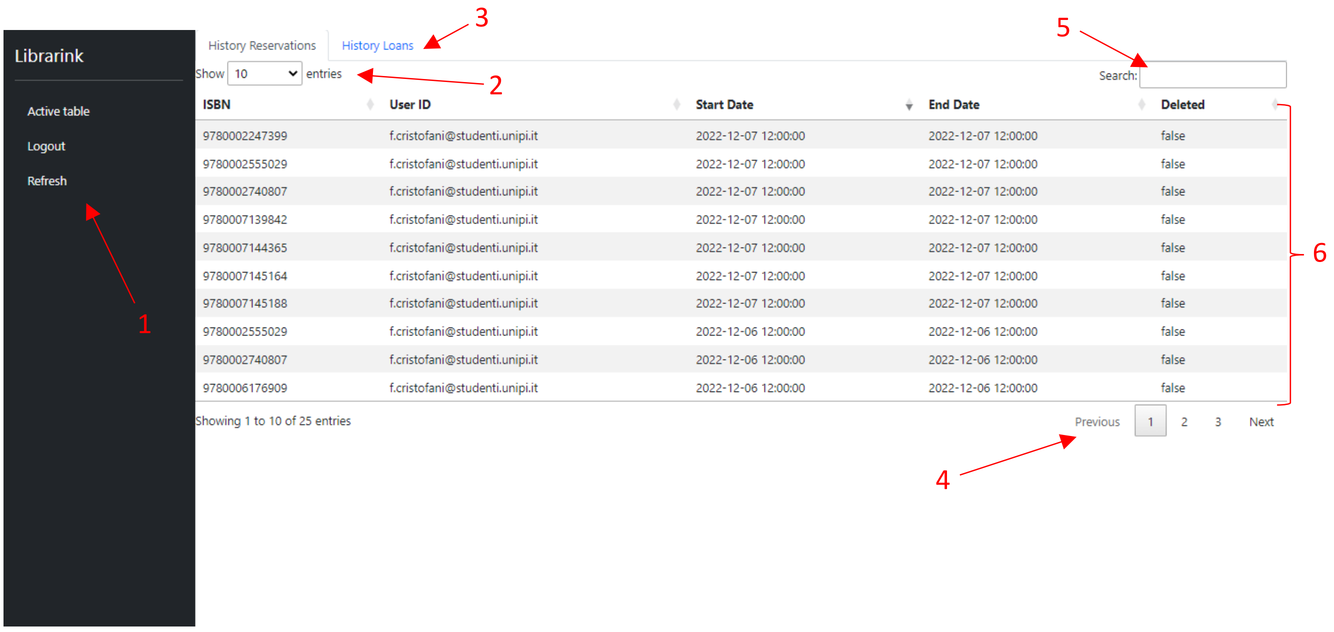
Showing 1 to 12 of 12 entries

Figure 19: Page where the administrator can check active loans, terminate a loan, modify book copies, or enter a new loan

Legend:

1) Menu through which the administrator can view the loan or reservation history table (see figures 20 and 21), logout or reload page content
2) Link to view the table of pending reservations (see figure 18)
3) Button to terminate a previously selected loan
4) Button to change the number of copies of a book (see figure 22)
5) Button to insert a new loan (see figure 23)
6) Field to search by ISBN for items in the displayed table
7) Table containing information for active loans. The leftmost column is used for selection

6.8 Reservations History Admin Page



Show 10 entries

Search:

ISBN	User ID	Start Date	End Date	Deleted
9780002247399	f.cristofani@studenti.unipi.it	2022-12-07 12:00:00	2022-12-07 12:00:00	false
9780002555029	f.cristofani@studenti.unipi.it	2022-12-07 12:00:00	2022-12-07 12:00:00	false
9780002740807	f.cristofani@studenti.unipi.it	2022-12-07 12:00:00	2022-12-07 12:00:00	false
9780007139842	f.cristofani@studenti.unipi.it	2022-12-07 12:00:00	2022-12-07 12:00:00	false
9780007144365	f.cristofani@studenti.unipi.it	2022-12-07 12:00:00	2022-12-07 12:00:00	false
9780007145164	f.cristofani@studenti.unipi.it	2022-12-07 12:00:00	2022-12-07 12:00:00	false
9780007145188	f.cristofani@studenti.unipi.it	2022-12-07 12:00:00	2022-12-07 12:00:00	false
9780002555029	f.cristofani@studenti.unipi.it	2022-12-06 12:00:00	2022-12-06 12:00:00	false
9780002740807	f.cristofani@studenti.unipi.it	2022-12-06 12:00:00	2022-12-06 12:00:00	false
9780006176909	f.cristofani@studenti.unipi.it	2022-12-06 12:00:00	2022-12-06 12:00:00	false

Showing 1 to 10 of 25 entries

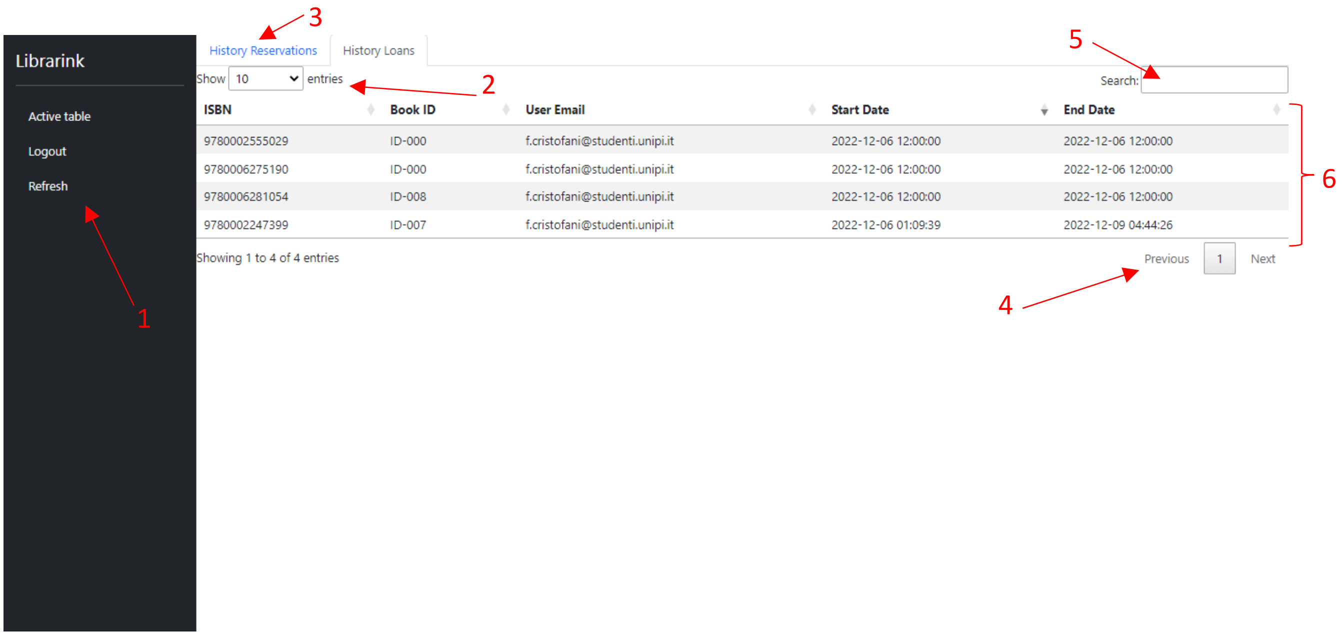
Previous 1 2 3 Next

Figure 20: Page where the administrator can consult the history of past reservations

Legend:

1) Menu through which the administrator can view the table of active loans or pending reservations see figures 18 and 19), logout or reload the page content
2) Field to define the number of items displayed on a single page
3) Link to view the loan history table (see figure 21)
4) Paging menu to scroll through the history by dividing the content into different pages
5) Field to search by ISBN for items in the displayed table
6) Table containing information for past reservations.

6.9 Loans History Admin Page



ISBN	Book ID	User Email	Start Date	End Date
9780002555029	ID-000	f.cristofani@studenti.unipi.it	2022-12-06 12:00:00	2022-12-06 12:00:00
9780006275190	ID-000	f.cristofani@studenti.unipi.it	2022-12-06 12:00:00	2022-12-06 12:00:00
9780006281054	ID-008	f.cristofani@studenti.unipi.it	2022-12-06 12:00:00	2022-12-06 12:00:00
9780002247399	ID-007	f.cristofani@studenti.unipi.it	2022-12-06 01:09:39	2022-12-09 04:44:26

Showing 1 to 4 of 4 entries

Figure 21: Page where the administrator can consult the history of past loans

Legend:

1) Menu through which the administrator can view the table of active loans or pending reservations (see figures 18 and 19), logout or reload the page content
2) Field to define the number of items displayed on a single page
3) Link to view the reservation history table (see figure 20)
4) Paging menu to scroll through the history by dividing the content into different pages
5) Field to search by ISBN for items in the displayed table
6) Table containing information for past loans

6.10 Admin’s Popup For Copies Modification

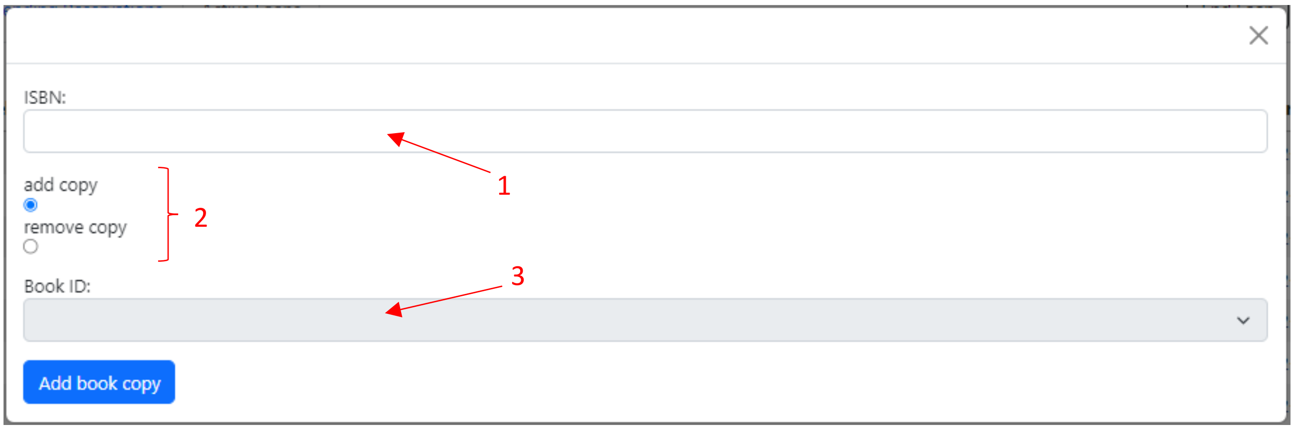


Figure 22: Popup to insert or remove a specific copy of a book

Legend:

1) Field in which to enter the ISBN of the book of which you wish to add/remove a copy
2) Selection of the type of operation to be performed
3) Field to select the ID of the copy to be deleted (Active if and only if "remove copy" is selected)

6.11 Admin’s Popup For Loan Insertion

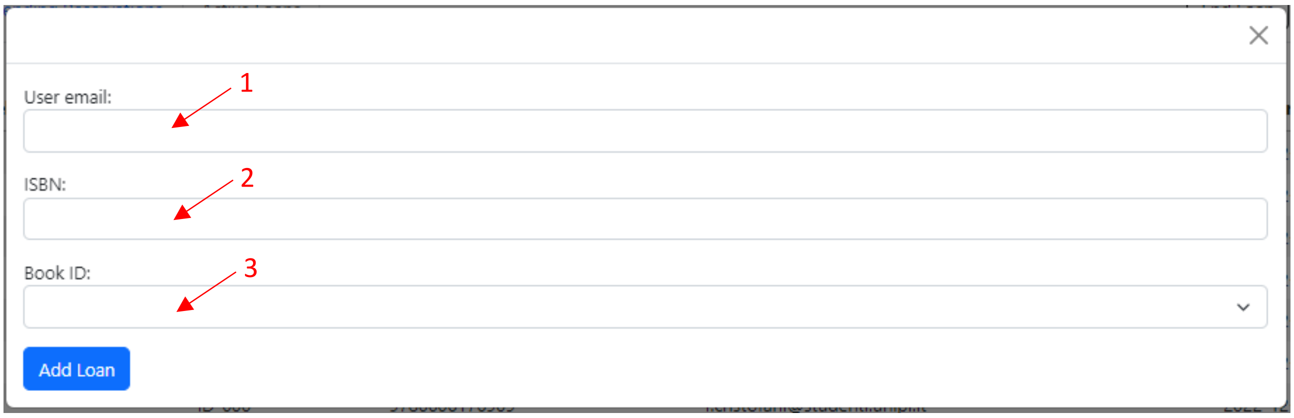


Figure 23: Popup to enter a loan to a user

Legend:

1) Field in which to enter the email of the user for whom the loan is being entered. Use the email provided by the user when registering on the site
2) Field to enter the ISBN of the lent book
3) Field to select the ID of the lent copy