

Distribution and Integration Technologies

An enterprise distributed system

João Almeida | João Mendes



FEUP **FACULDADE DE ENGENHARIA**
UNIVERSIDADE DO PORTO

Rua Dr. Roberto Frias, s/n 4200-465 Porto PORTUGAL

| | |
|---------------------------------------|-----------|
| 1. Introduction | 3 |
| 2. Technologies Used | 3 |
| 3. Architecture Overview | 3 |
| 3.1 - Bookstore and Warehouse Servers | 4 |
| 3.2 - Bookstore and Warehouse GUI | 4 |
| 3.3 - Requests and Book Orders | 4 |
| 4. Networking Details | 5 |
| 4.1 Bookstore Networking | 5 |
| 4.1.1 Bookstore Server | 5 |
| 4.1.2 Bookstore Printer | 6 |
| 4.1.3 Bookstore GUI Client | 7 |
| 4.1.4 Bookstore Web Application | 7 |
| 4.2 Warehouse Networking | 8 |
| 4.2.1 Warehouse Server | 8 |
| 4.2.2 Warehouse GUI Client | 8 |
| 5. User Interface | 9 |
| 5.1 Bookstore User Interface | 9 |
| 5.1.1 Order Creation Window | 9 |
| 5.1.3 Bookstore Client Details Window | 10 |
| 5.1.4 Bookstore Arriving Books Window | 10 |
| 5.2 Warehouse User Interface | 11 |
| 5.2.1 Warehouse Clerk Interface | 11 |
| 5.3 Web Application Interface | 12 |
| 5.3.1 Client Details Form | 12 |
| 5.3.2 User Homepage | 13 |
| 6. Conclusion | 14 |

1. Introduction

A book editor prints and sells books in his own installations and book shop. It intends to develop a system for coordinating its sells, orders and stock management. The editor owns two facilities physically different, the bookshop store with a public exposition area, and a warehouse for storing larger quantities of books. The editor intends also to make available a web application for remote consulting and ordering.

In the bookshop store a web server is running, hosting the web application, and some persistent record of available titles, price and existences (stock) in the bookshop and warehouse. Some services can also be hosted in the same server. This server is always on and is connected to the internet. Between the store and warehouse there is also a network connection, but the warehouse computers usually run only in labor hours, including its own server.

2. Technologies Used

Since both of the group members do not have constant access to a Windows machine, and the Mono libraries for C# in Linux are still widely undeveloped we have decided that it would be best for us to develop our project in Java. To allow the easier handling of compiling and running the project we have used Maven to keep track of dependencies. We used RabbitMQ for the messaging queue service and Tomcat for the deployment of the web application. Similarly to the first project the GUI was made using Gtk, this time since the Java libraries are more mature than the C# Mono libraries we were able to use the most recent version of Gtk, which is version 3, as opposed to the first project where we were forced to use Gtk2.

3. Architecture Overview

Similarly to the last project we have applied a Model View Controller (MVC) architecture in each of the main modules of the project, the Bookstore and the Warehouse. The GUI, WebApp and Printer serve as the View, and the server has incorporated both the Controller and the Model through the use of a database.

Below is a simplified representation of the project's various components and how they communicate with one another.

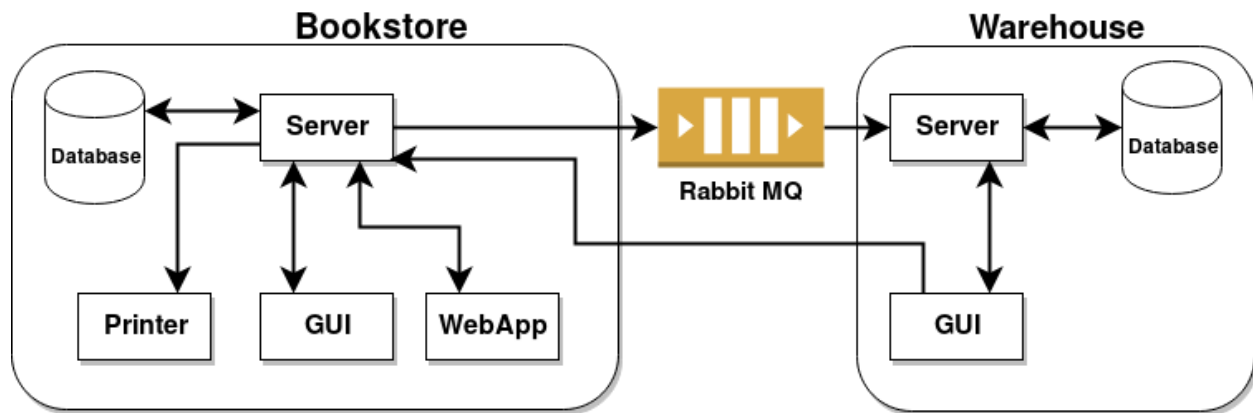


Fig. 1 - Project's Architecture Overview

In the overview, squared boxes represent different processes, so there are possible 5 different processes running.

3.1 - Bookstore and Warehouse Servers

There is a single server running in both the Bookstore and the Warehouse. It is responsible for handling the database of the project. The database is consisted of several data structures, however the server of the Bookstore keeps a local copy of the book stocks in a Comma Separated Values (CSV) file, so that that information is persistent across multiple runs of this server. On the other hand the Warehouse server need not store its stock values since we are to assume that it always can meet the book requests, so it only has information about which books are sold in the store. That information is also stored in a CSV file, and is read at the startup of both servers to fetch the books.

Bookstore server is able to send messages asynchronously to the Warehouse server through the use of an external Messaging Queue service, namely RabbitMQ.

3.2 - Bookstore and Warehouse GUI

The GUI are the one of the components of the project that will be handled by an employee and as such should be fairly intuitive and easy to use. They both have direct communication with the server.

3.3 - Requests and Book Orders

Although specified in the project specification that a request could only have a single book associated, we have decided that it would be more realistic and more challenging to have multiple books associated with a single Request. By choosing to follow this model, it means that a single Request will now have associated multiple Book Orders and those Orders is where

the book amount will be stored, as well as the dispatch state of the book. This translates into the possibility that a single Request has multiple states, since depending on the Book Order the state might be 'waiting' or 'dispatched'.

4. Networking Details

4.1 Bookstore Networking

Bookstore is the most central infrastructure of this project since it has a server that is constantly running.

4.1.1 Bookstore Server

Server that is assumed to be always running, so most of the operations are handled in this component. It has various remote methods, which can be executed by a process running on a different computer. This server is the one that produces messages that are in turn sent to the Warehouse Server through the RabbitMQ service.

Remote Method Invocation (RMI) is the main communication mechanism used in the project. In the case of the Bookstore Server it has the following remote methods:

```
LinkedList<Request> getUserRequests(String username);
LinkedList<Book> getAllBooks();
void putRequest(String name, String addr, String email, Map<String, Integer> books);
LinkedList<BookRequests> getArrivedBooks();
void booksStored(LinkedList<BookRequests> books);
void booksDispatched(Map<String, Integer> books_amount, LinkedList<Long> req_uuids);
```

- `getUserRequests`

Called only by the web application to fetch all of the user requests. A Request is consisted of a unique UUID, a client name, address and email, as well as all the Books that were Requested by the user, regardless of the state of the book request.

- `getAllBooks`

Called by both the web application and the GUI client to fetch all of the available books in the bookstore. This includes books that might not be physically present in the Bookstore but they exist in the Warehouse.

- putRequest

Called by both the web application and the GUI client to signal the server that a new Request needs to be created for the given user. Server will handle dealing with which books are available in the Bookstore and can be immediately dispatched and which books need to be requested from the Warehouse. Request UUID is only assigned here, due to this being the only component that is always running and has information about what was the last assigned Request UUID.

- getArrivedBooks

Called by the GUI client to fetch the books that were previously requested from the Warehouse and that the Warehouse has dispatched to the Bookstore. Returns a list of books with associated Request UUID's so that the Bookstore server knows which Requests to associate with the given arrived Books.

- booksStored

Called by the GUI client to signal the Bookstore server that the given books were stored in the Bookstore, so the Book Requests with the passed UUID's can now have its state changed from dispatching to dispatched.

- booksDispatched

Called by the Warehouse GUI client to signal that the given books will be dispatched from the Warehouse to the Bookstore. This method is common to both the Bookstore and the Warehouse Server and is called at the same time by the Warehouse GUI client. It has the Request UUID's so that the given books are associated with the correct Requests.

4.1.2 Bookstore Printer

Component that serves as the Bookstore receipt printer. It is a very simple component which prints directly into the command line stdout. Once again it uses Java RMI capabilities to allow communication from the Server to the Printer. It has a single remote method:

```
void printRequest(Request req);
```

- printRequest

Called by the Bookstore Server whenever a new Request is created. It prints the details of all of the books ordered, even the ones that will need to be requested from the Warehouse.

4.1.3 Bookstore GUI Client

Component that will be running in the Bookstore and operated by a store clerk. Allows the creation of new Requests and the acceptance of books that were dispatched from the Warehouse. The available remote methods are the following:

```
void booksArriving();
```

- booksArriving

Called from the Bookstore server to signal that the Client GUI should update its internal list of the books that have arrived from the Warehouse. Client GUI will in turn call the *getArrivedBooks* method of the Bookstore Server in order to update its own list and what is displayed to the user.

4.1.4 Bookstore Web Application

Apart from the Client GUI the Bookstore also allow the communication with the server through HTTP using REST services. It is a very simplistic web application, it only has a couple of routes defined and the security is likely very lacking.

The routes defined and handled by the project are the following:

- /bookstore

Main route of the web application that shows the initial page where the user is asked to insert its details, namely the name, email and address, only after these details have been filled out is the user able to make new requests.

- /bookstore/login

Route to send the details the user has filled out. It will then show the user homepage, with a list of its requests and available books in the store, allowing the user to create a new Request.

- /bookstore/request

Route to be sent using a GET request and that should contain both the user details and all of the books and amounts the user has selected to make the Request.

4.2 Warehouse Networking

Unlike the Bookstore infrastructure the warehouse components will only be running on labor hours, hence the need for an asynchronous messaging queue system. RabbitMQ is the service we have used, which allows us to send messages to the Warehouse server that are guaranteed to arrive even though the server might not be running.

4.2.1 Warehouse Server

Server that is only running on labor hours. Since it needs to receive messages from the Bookstore Server which is always up, it is the consumer endpoint in the RabbitMQ service. Apart from that it also implements several remote methods listed below:

```
LinkedList<Request> getWaitingRequests();  
void booksDispatched(Map<String, Integer> books_amount, LinkedList<Long> req_uuids);
```

- `getWaitingRequests`

Called by the Warehouse GUI in order to fetch the book requests that were sent by the Bookstore Server to the Warehouse Server. The server internally sorts which Requests have the 'waiting' status and which do not.

- `booksDispatched`

Called by the Warehouse GUI to dispatch the given books to the Bookstore. It has associated Request UUID so that the Servers know what Requests to associate with the books being dispatched, otherwise it might associate wrong requests.

4.2.2 Warehouse GUI Client

Regular GUI for the warehouse which is operated by a warehouse clerk. Responsible for dispatching books from the Warehouse to the Bookstore. Implements a single remote method:

```
void newBookRequest();
```

- `newBookRequest`

Called by the Warehouse Server whenever a new Book Request has arrived from the Bookstore Server to the Warehouse Server, it signals that the GUI Client should once again call the *getWaitingRequests* method of the Warehouse Server in order to update its internal lists.

5. User Interface

5.1 Bookstore User Interface

5.1.1 Order Creation Window

In this window the store clerk is able to create new orders to match what the customer wants. It should be a simple interface that quickly allows the adding and removing of books to the order. In this window we can see the stock amount of the books in the Bookstore, books that have no stock will immediately be sent as a request to the Warehouse to replenish stock.

Bookstore Window

See Arrived

Book List

| Name | Price | Amount | Add |
|---------------------------------|---------|--------|-----|
| Hamlet | 7.50 € | 2 | + |
| Don Quixote | 11.65 € | 5 | + |
| The Stranger | 12.11 € | 0 | + |
| Invisible Man | 5.15 € | 6 | + |
| Alices Adventures in Wonderland | 17.45 € | 0 | + |
| The Sun Also Rises | 2.55 € | 0 | + |
| Ulysses | 0.80 € | 0 | + |

Book Order

| Name | Price | Amount | Delete |
|---------------|---------|--------|--------|
| Invisible Man | 5.15 € | 3 | - |
| The Stranger | 12.11 € | 2 | - |

Submit

Reset

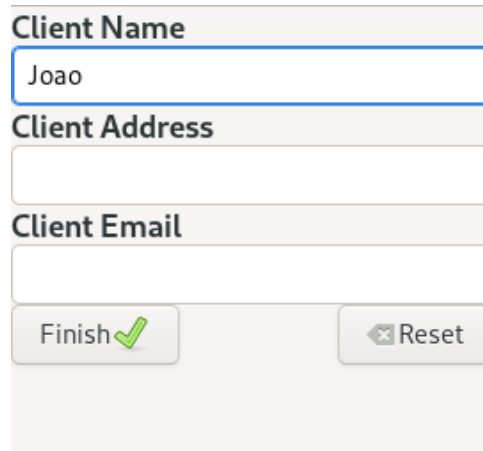
39.67 €

Total Price

Fig. 2 - Bookstore order creation window screenshot

5.1.3 Bookstore Client Details Window

After the store clerk has finished setting up the right amounts of each book in the previous window, the store clerk will need to fill up the necessary client details to associate with the Request.

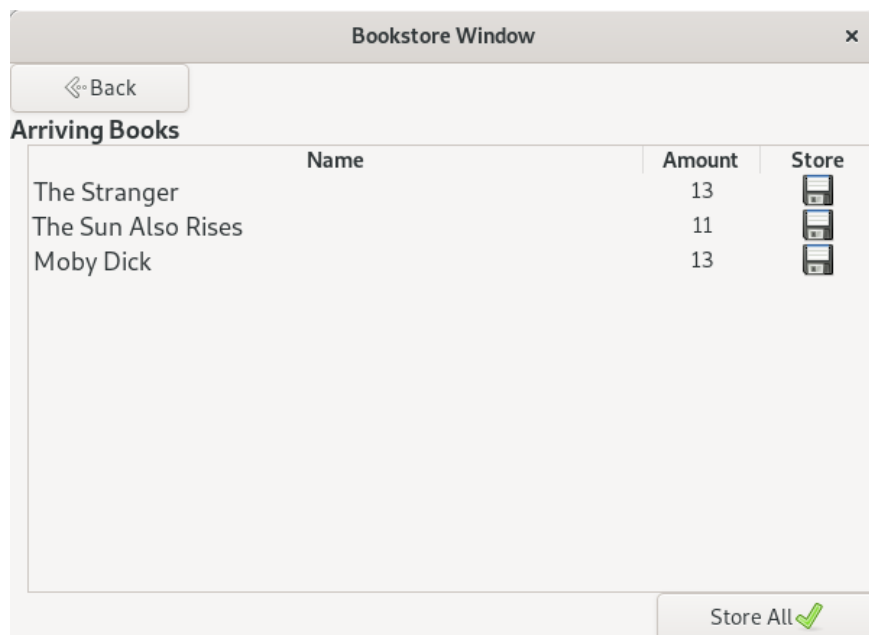


The screenshot shows a form titled "Client Details Window". It contains three input fields: "Client Name" with the text "Joao", "Client Address", and "Client Email". At the bottom, there are two buttons: "Finish" with a green checkmark icon and "Reset" with a grey arrow icon.

Fig. 3 - Client details window screenshot

5.1.4 Bookstore Arriving Books Window

Window that only displays the books that the Warehouse has dispatched to the Bookstore. In here the store clerk can allow the storage of those books. After the storage has been accepted the requests that were pending on a particular book arriving will be updated accordingly.



The screenshot shows a window titled "Bookstore Window" with a close button (X) in the top right corner. Below the title bar is a "Back" button. The main content area is titled "Arriving Books" and contains a table with three columns: "Name", "Amount", and "Store". The table lists three books: "The Stranger" with an amount of 13, "The Sun Also Rises" with an amount of 11, and "Moby Dick" with an amount of 13. Each row has a "Store" button with a book icon. At the bottom right of the window is a "Store All" button with a green checkmark icon.




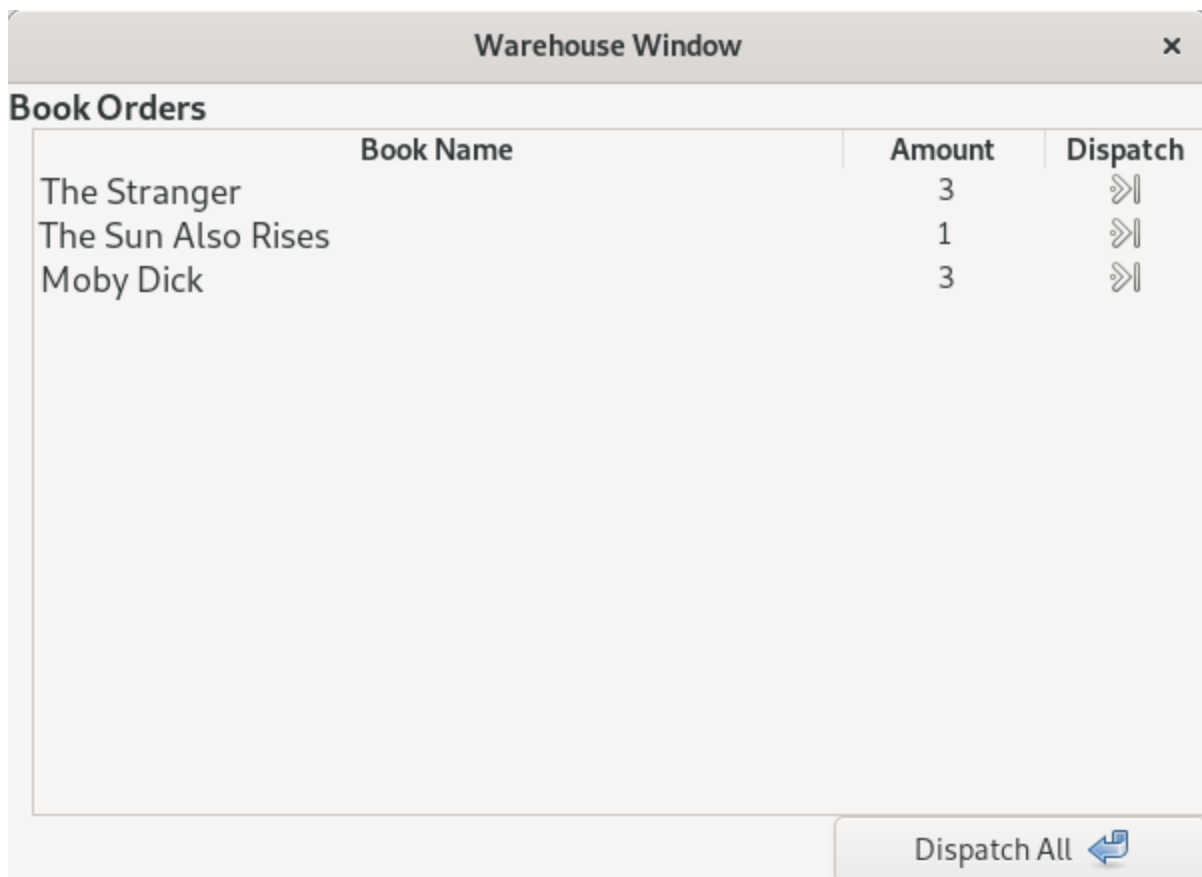
| Name | Amount | Store |
|--------------------|--------|---|
| The Stranger | 13 |  |
| The Sun Also Rises | 11 |  |
| Moby Dick | 13 |  |

Fig. 4 - Bookstore arriving books window screenshot

5.2 Warehouse User Interface

5.2.1 Warehouse Clerk Interface

Simple GUI that should allow the clerk to view the pending book requests. Internally the Warehouse Server merges all of the requests so that in this interface the requests are grouped by Book type, as opposed to by actual Requests.



The screenshot shows a window titled "Warehouse Window" with a close button (X) in the top right corner. Below the title bar is a section labeled "Book Orders". Inside this section is a table with three columns: "Book Name", "Amount", and "Dispatch". The table contains three rows of data:

| Book Name | Amount | Dispatch |
|--------------------|--------|----------|
| The Stranger | 3 | ⇒ |
| The Sun Also Rises | 1 | ⇒ |
| Moby Dick | 3 | ⇒ |

At the bottom right of the window is a button labeled "Dispatch All" with a blue arrow icon pointing to the right.

Fig. 5 - Warehouse clerk window screenshot

5.3 Web Application Interface

5.3.1 Client Details Form

Entry point to the web application where the user is asked to fill in its details, so that the web application can fetch it's requests, as well as ensuring all client details are correct and validated upon entering the homepage.

User Login

Client Name:

Address:

Email:

Sign Up

Reset

Fig. 6 - Web application client details form screenshot

5.3.2 User Homepage

In this page the user is able to see its currently opened requests and is able to create a new order. Similarly to the Bookstore Client GUI, the user can select multiple amounts of multiple book titles and the Bookstore Server will handle processing them.

Hello Joao!

Requested Books

Order #0

| Title | Amount | State | Dispatched |
|---------------------------------|--------|--------------------|------------|
| The Stranger | 3 | Waiting Expedition | No Date |
| Alices Adventures in Wonderland | 1 | Waiting Expedition | No Date |

Order #1

| Title | Amount | State | Dispatched |
|--------|--------|--|------------|
| Hamlet | 1 | Dispatch should occur at Mon May 27 18:55:55 WEST 2019 | |

Available Books

| Title | Price | Amount | | |
|---|-------|--------|---|--|
| Hamlet | 7.5 | 0 | <input data-bbox="946 1228 1047 1276" type="button" value="+"/> | <input data-bbox="1052 1228 1153 1276" type="button" value="-"/> |
| Don Quixote | 11.65 | 0 | <input data-bbox="946 1285 1047 1333" type="button" value="+"/> | <input data-bbox="1052 1285 1153 1333" type="button" value="-"/> |
| The Stranger | 12.11 | 0 | <input data-bbox="946 1341 1047 1390" type="button" value="+"/> | <input data-bbox="1052 1341 1153 1390" type="button" value="-"/> |
| Invisible Man | 5.15 | 0 | <input data-bbox="946 1398 1047 1446" type="button" value="+"/> | <input data-bbox="1052 1398 1153 1446" type="button" value="-"/> |
| Alices Adventures in Wonderland | 17.45 | 0 | <input data-bbox="946 1455 1047 1503" type="button" value="+"/> | <input data-bbox="1052 1455 1153 1503" type="button" value="-"/> |
| The Sun Also Rises | 2.55 | 0 | <input data-bbox="946 1512 1047 1560" type="button" value="+"/> | <input data-bbox="1052 1512 1153 1560" type="button" value="-"/> |
| Ulysses | 9.89 | 0 | <input data-bbox="946 1568 1047 1617" type="button" value="+"/> | <input data-bbox="1052 1568 1153 1617" type="button" value="-"/> |
| Moby Dick | 12.55 | 0 | <input data-bbox="946 1625 1047 1673" type="button" value="+"/> | <input data-bbox="1052 1625 1153 1673" type="button" value="-"/> |
| The Divine Comedy | 10.2 | 0 | <input data-bbox="946 1682 1047 1730" type="button" value="+"/> | <input data-bbox="1052 1682 1153 1730" type="button" value="-"/> |
| Oedipus the King | 21.25 | 0 | <input data-bbox="946 1738 1047 1787" type="button" value="+"/> | <input data-bbox="1052 1738 1153 1787" type="button" value="-"/> |
| <div><input data-bbox="259 1795 427 1843" type="button" value="Submit"/> <input data-bbox="435 1795 586 1843" type="button" value="Reset"/></div> | | | | |

Fig. 7 - Web application user homepage screenshot

6. Conclusion

Overall we believe we have developed a mature project, that with a few tweaks might be ready for enterprise deployment. The weakest features of the project lie in the web application, since it is an area which neither of us is really comfortable working with and therefore all we have made is the Minimum Viable Product (MVP), it fits the required functionalities but it is not very appealing and likely has many security flaws.

An aspect we would like to point is that the error we had in the GUI, that was explained in the previous project report, is no longer present even though the interface and how it is handled is basically the same. Therefore we believe the previous error we had might have been due to both the usage of a C# Mono library which as we have said is still very undeveloped, as well as the usage of the legacy Gtk2 version.

The instructions for compiling and running the program are present in the *README.md* file, as well as the project's needed dependencies. The work was divided between the group members as follows:

- João Almeida 50%
- João Mendes 50%