deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

TQS: Product specification report

Carolina Marques [85084], Diogo Silveira [85117], Edgar Morais [89323], Gabriela Santos [51531]
v2020-06-05

1 Introduction

1.1 Overview of the project

Our project consists of a web app designed to work as a marketplace for books. Our system will allow for consumer to business interactions as well as business to business ones. This means that consumers will be able to buy the books displayed in our application, while other businesses, namely book publishers, will be able to add their own books for sale in our platform. The promoter will make money from all the transactions that are being conducted on its platform, in the form of commissions from the book sales.

Since this project is being developed for the TQS course, our solution and its development are expected to be well documented and tested, in order to ensure a quality product. This means that, through the development of this project, we will be able to use and learn in a more practical environment the different practices related to software quality we have been acquiring throughout this course.

1.2 Limitations

Some of the features that we would implement if we had more time to develop the project would be:

- Book reviews;
- More elaborate search filtering (through dates that were missing from Orders, Revenues and Commissions);
- Estimated shipping date;
- More realistic payments;
- The UI could present information in more elaborate ways.

2 Product concept

2.1 Vision statement

Our web app will work as a MarketPlace where users will be able to buy books and companies/publishers will be able to put up their books for sale. This way users will have a centralized way to buy books from various publishers and publishers will be able to market their books to an online audience and expand their reach.

With this in mind we created the following use cases to aid us in the development of this project:

U.C.1:
Title: User sees list with available books
As a user/client
I want to see the available books in the web application
So that I can find books that interest me

U.C.2:
Title: User buys displayed book
As a normal user/client
I want to buy a book available on the web application
So that I can own a copy of said book

U.C.3:
Title: Publisher proposes book for sale
As a publisher representative
I want to propose a book to be on sale on the web application
So that the proposed book gets more exposure and sells more copies

U.C.4:
Title: User checks book details
As a normal user/client or a publisher representative
I want to check the details of a book
So that I can decide whether or not I want to buy it

U.C.5
Title: User searches book by title
As a normal user/client
I want to find a book by its title

So that I can check its details

U.C.6:
Title: User searches for book according to preferences/many parameters
As a normal user/client
I want to find a book that I like through a series of filters/parameters
So that I can find books that might interest me.

U.C.7:
Title: User adds books to shopping cart
As a normal user/client
I want to add books to my shopping cart
So that I can buy more than one book at a time

U.C.8:
Title: User buys many books at once
As a normal user/client
I want to buy many books at once
So that I don't have to buy them individually

U.C.9:
Title: Publisher checks books associated to it/checks all the books that were published by them and are available in the web application
As a publisher representative
I want to check the books published by my publisher
So that I can keep track of which books were proposed to the web application

U.C.10:
Title: User checks purchase history
As a normal user/client
I want to check my purchase history
So that i can know which books I already ordered

U.C.11:
Title: User checks order through order number
As a normal user/client
I want to check the details of an order
So that I can verify its details

U.C.12:
Title: System administrator checks the total value of commissions
As a system administrator of the service
I want to check the total values of the commission applied to the book sales
So that I can keep track of the total value accumulated by the applied commissions

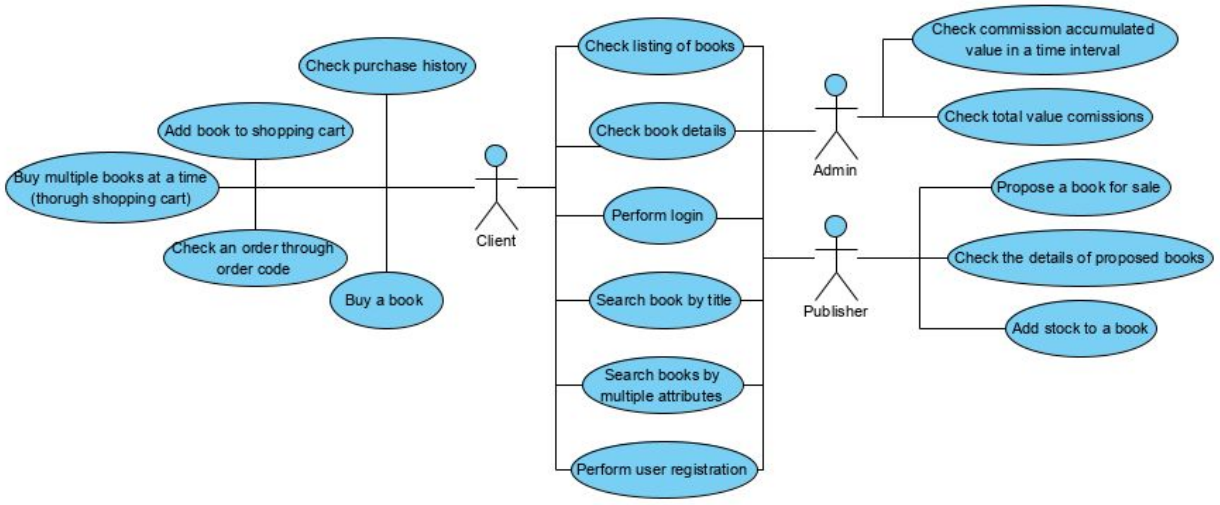
U.C.13:
Title: System administrator checks the value of a commission
As a system administrator
I want to check the values of a commission made by a sale
So that I can keep track of each sale's commission value

U.C.14:
Title: User performs login
As any user (client, publisher or administrator)
I want to login to the web application
So that I can access the features of the system specific to my user permissions

U.C.15:
Title: User registers in application
As an unregistered user in the application
I want to register a personal account in the web application
So that I can access the features of the application that require a logged in user

U.C.16:
Title: Publisher adds stock of an already available book
As a publisher
I want to add new stocks to a book I put up for sale
So that I can sell more copies of that book

Use case diagram



2.2 Personas

Maria is 21 years old and lives in Aveiro. She likes being at home and spends a lot of her time in various websites related with various stores. Even though she spends a lot of time on web stores, she ends up visiting anonymously a lot of them, just checking what is on sale. **Motivation**: Maria would like to check which books are on sale, regardless of their publisher, so that she doesn't have to search for book sales in many websites.

Tiago is 20 years old and lives in a village in the countryside. He works as a farmer on his family's farm. There he works, more specifically, in the management of the farm and, as such, he works mainly indoors. He really likes reading books, but he doesn't buy them very often because the nearest bookstore is far away from his area of residence. **Motivation:** Tiago would like a way to buy/order books from a wide selection of publishers remotely, so that he doesn't have to spend time to go to the nearest book store.

Editora Campos is a publisher that has been in existence for 50 years. They already have a relatively wide audience, but they haven't yet expanded to the online market. **Motivation:** Editora Campos would like to expand their business beyond physical book stores in order to increase their sales and to access a whole new audience.

Andreia is a 37 year old financial analyst that works at BookStore. In her free time she enjoys jogging, going to the gym and watching movies. In order to do her job she has to have access to the analytics related to the sales made through the web application. For this she has to ask the development team for the analytics, this meaning that her work is dependent on the availability of the development team. **Motivation:** Andreia would like have a way to access at least some values related to the commissions generated by the sales made by the web application.

2.3 Main scenarios

The first scenario depicts the scenario where Tiago, having already created customer account, finds a book and buys it:

Tiago finds and buys a book - Tiago opens up the web application and sees a welcome message, a list of available books and a login option. He first chooses to log in to the application through the login option and the application displays the fields necessary to login on the application. Tiago inputs his account information, submits it and is redirected to the buyer's main page that, besides displaying a list of books also informs him that is login was successful. Tiago explores the list of books that is displayed and chooses clicks on one of his liking. Then it is displayed a age with the book details and an option to buy. Tiago clicks the buy option and is taken to a page where he has to fill out a form with the informations required to complete the transaction. After the form is filled and submitted successfully the application displays a message informing Tiago that is purchase was executed successfully.

In the second scenario a representative of the publisher Editora Campos registers said publisher in the web application and proposes a book for sale:

The representative proposes a new book for sale in the application - The representative opens up the application and sees a welcome message, a list of available books and a login option. Then chooses to login and logs in with her business account. The representative will be redirected to the publisher's main page, where all their books for sale are being displayed, along with an option to add new stock. The representative chooses the option to propose a new book for sale. This leads to a form that has to be filled with the details of the book that is going to be added for sale. After the form is completed it is displayed a message informing that the book proposal was completed/successful.

In the third scenario Andreia checks the total value of the commissions generated by the sales:

Andreia checks commission total value - Andreia opens up the application and sees a welcome message, a list of available books and a login option. She chooses to login and logs in with her administrative account. She has now access to new functionality that is exclusively shown to administrators, such as querying the commission values generated by the sales made through the application. She is presented with the total amount of commissions that the platform has collected since its inception, along with an option to choose a start date and an end date to check the commissions in more detail (all the commissions' operations can be displayed for the selected time period).

In the fourth scenario Maria checks the various books on sale:

Maria checks the books on sale - Maria opens up the application and sees a screen with a list of books, a search bar and a login option. She scrolls down and is able to check what books are on sale.

2.4 Project epics and priorities

We organized the user stories mainly in 5 epics. These are:

- Search service;
- Order service;
- Stock service;
- Commission service;
- Login service;

The Search service epic consists in the implementation of the features/stories that are related to the display of the available books. This includes the user being able to see the list of books, searching for one by name and checking its details.

The Order service is related to the implementation of mechanisms that allow the user to buy books. These include the user being able to order one or more books at a time, checking a specific order and checking a history of purchases.

The Stock service is related to mechanisms in which the publishers/companies add books for sale in the web application and check the books they already added.

The Commission service is related to features in which a system administrator checks the values of commissions generated through transactions of the platform, weather they are total values or accumulated values through a time interval.

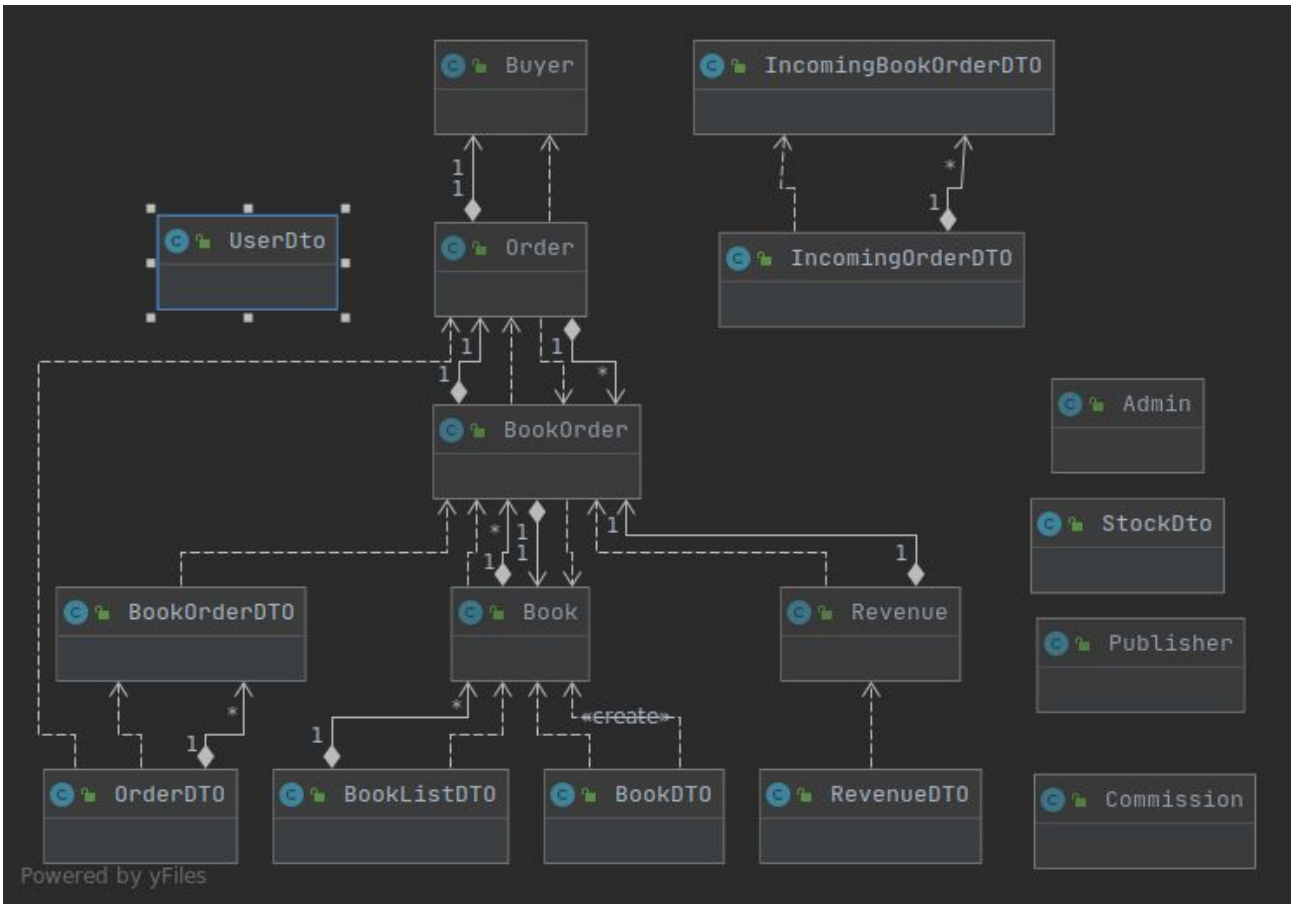
Finally the Login service is related to the mechanism of login and registration that lead to different options depending on the user type.

3 Domain model

Our project consists of a book store where the main entities are publishers, books, buyers and orders. The publishers' job is to make their stock of books available so that the buyers can place an order. Also the platform makes money by demanding a commission for each order and provides information about the publishers' revenues. The commissions and revenues themselves are entities of our model. Other two important entities are the admin which saves admin users of the platform and the book-order which relates books to orders. These data can be gathered by a set of REST controllers:

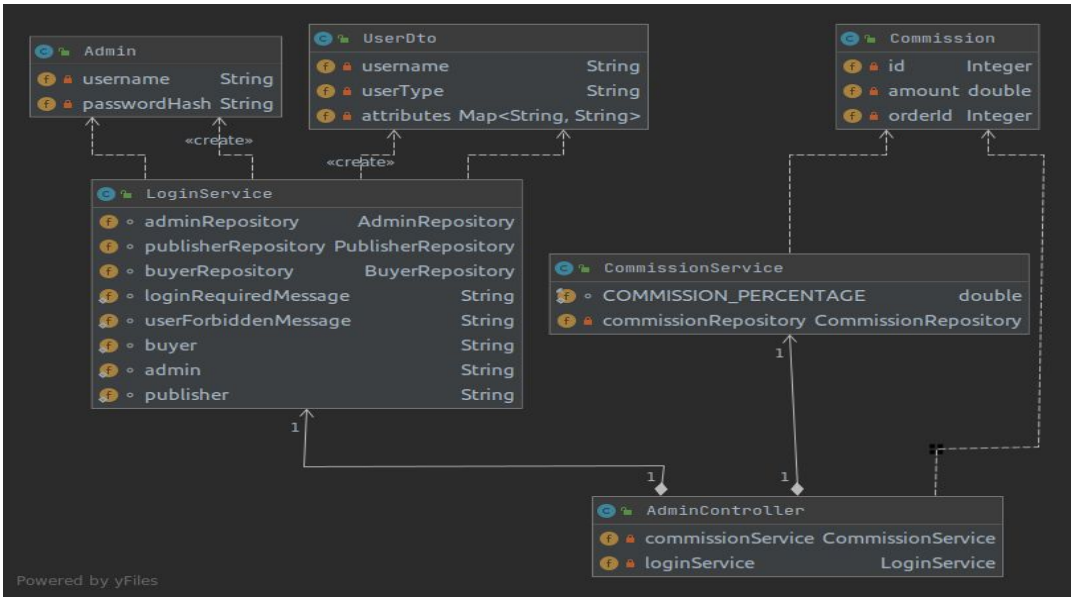
- The admin controller with access to the commissions and their total value;
- The books controller which provides a way to search books - individually, by title or isbn, or a set of the available books;
- The buyers controller which can find a history of the orders made by a buyer;
- The orders controller which can create and get orders and also compute their final price;
- The publishers controller which manages the publishers' stock as well as it retrieves their (total) revenues.
- The session controller is used for creating new accounts and logging in as well as getting information about the users.

The connection between the repositories that persist those entities and the controllers is made by a set of services built in the business logic layer.

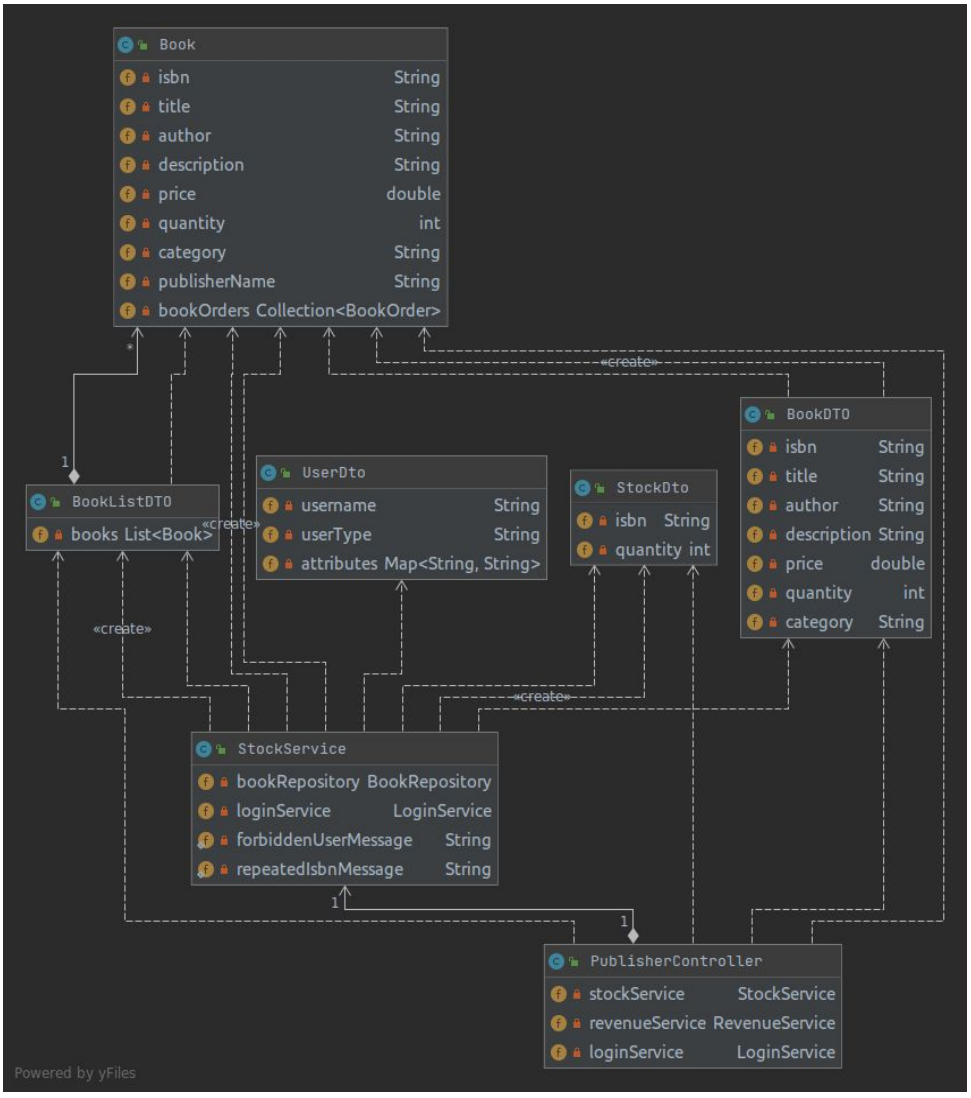


The diagram above shows all the entities from the domain of the project and the data transfer objects needed, as well as the direct relations between them. Some of the entities and DTO's have no direct relation between them but indirect ones are achieved in the upper layer.

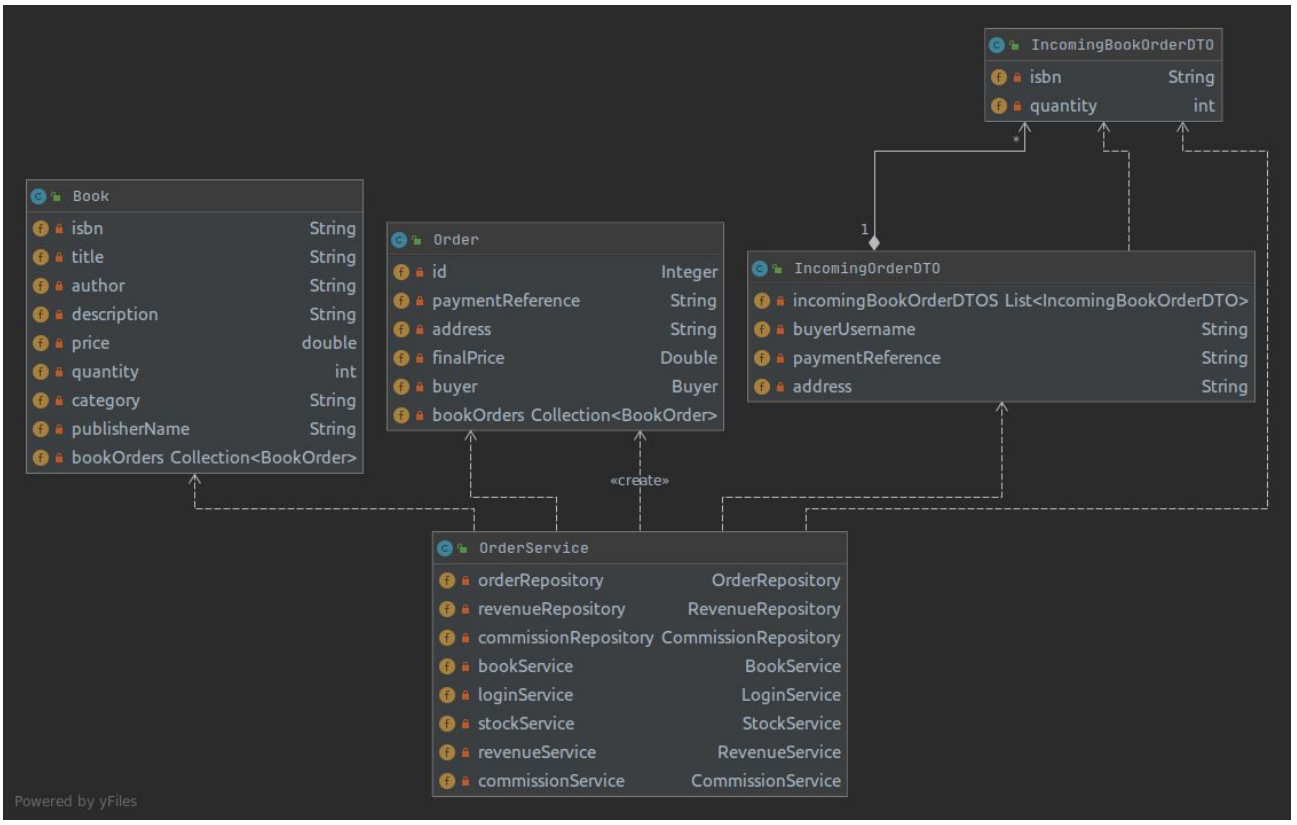
As we can see below, the Admin has access to the Commissions and the UserDTO is needed for the login:



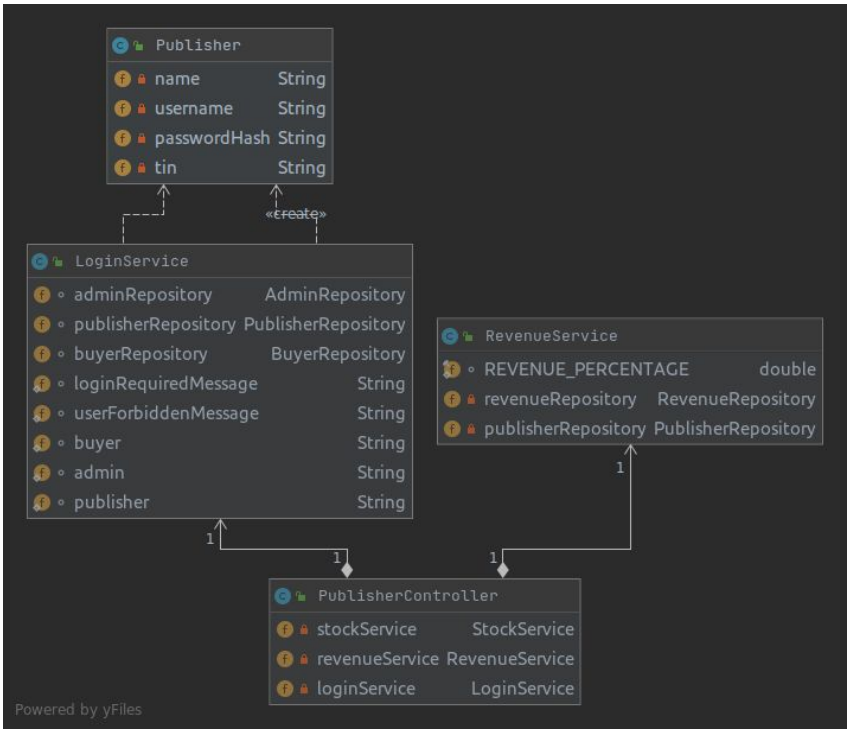
The StockDTO is used by the publishers for managing their stocks, as we can see in the following diagram which shows some of the functionality of the PublisherController:



The IncomingOrderDTO and IncomingBookOrderDTO are needed for the OrderService to manage the Orders:



Finally, he Publisher is one type of the users of the system, so the entity is needed by the LoginService class:



Other services and controllers are used to achieve the main objective of the system: searching for and buying products.

4 Architecture notebookKey requirements and constraints

- The system is expected to handle a large amount of users and a high volume of requests per user, particularly due to the search feature. Its performance should be stable nonetheless.
- The system should enforce three different user profiles: Buyer, Publisher and Admin. The access levels to the platform between all the user profiles will vary and privacy between sessions should always be enforced.
- New users will be able to register themselves in the platform and thus create a new user account, along with the ability to login whenever they want. The mechanism for authentication and mechanism will rely on generated API authorization tokens for each individual user, so that privacy can be guaranteed. Passwords should not be directly exposed by the API and will only be stored in the hash format.
- The system should provide a user-friendly interface, where all the operations that the system provides can be intuitive for new users. Due to time constraint for development, simplicity in design will be privileged. This interface will work independently as a web application that interacts with the platform's own API.
- A publishing company, which will be a fully autonomous application working as an external API client, will interact with the marketplace. Both applications should be deployed separately. The publishing company will be making periodic API requests to the platform (GET and POST requests regarding stock management for one of their books that is being sold in the marketplace) and process them with simple business logic.
- The system should offer a stable persistence mechanism, so that all data that is handled by the marketplace can be permanently recorded without any losses. Handling the concurrency between transactions (dealing with multiple buyers for the same product) wil be a key issue for the DBMS.
- The deployment solution should rely on a containers-based deployment strategy for each autonomous application. These applications should always be readily available for API requests and operating in a server environment without any disruptions.

4.1 Architectural view

The architecture can be broken down into three main modules: Persistence, Backend and Client. This solution was chosen for several different reasons:

- Easier division of tasks for the development team, due to the lack of coupling between the tiers;
- Maintainability. Changes to the business logic in the backend will not affect the performance of the user interface of the client, for example;
- The technological stack could be updated for one tier, if the need arises, without impacting the performance of the others.

The Persistence module will be responsible for storing all the data relevant to the main application. PostgreSQL was selected as the DBMS, which will be able to insert, retrieve and manage the data. PostgreSQL was chosen for the technological stack due to its availability as an open source tool, relational nature of the data that we will be working with, ease of use in large systems that need quick read and write operations and the fact that it provides a very intuitive GUI auxiliary tool (PgAdmin).

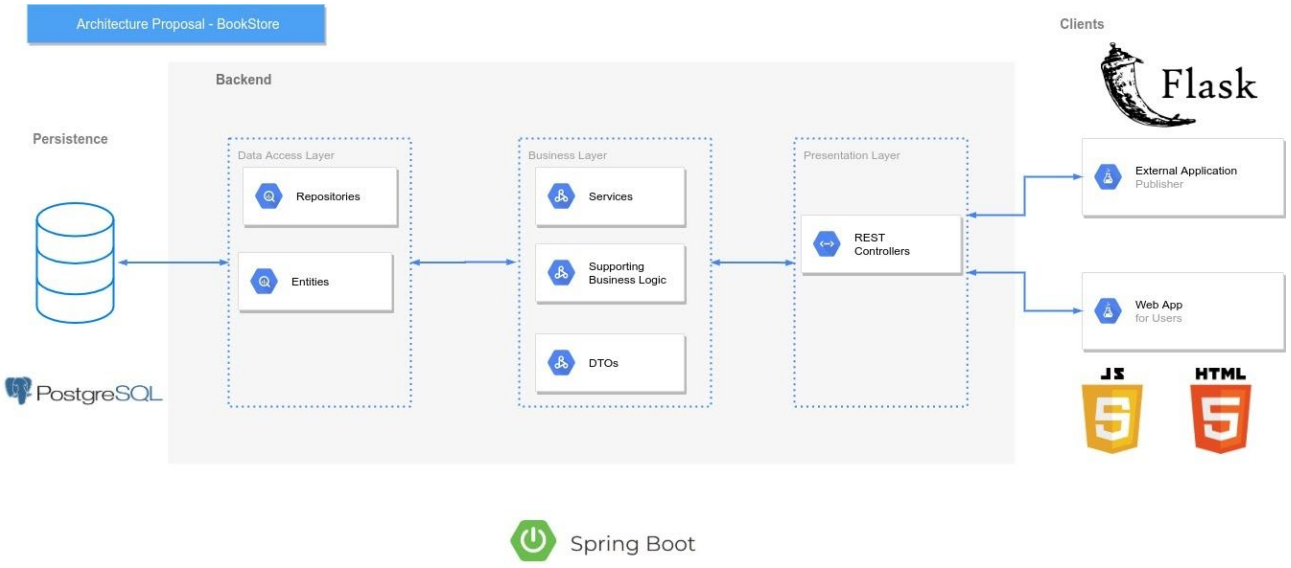
The Backend module will maintain a constant two-way type of communication with Persistence module, in order to exchange data. The Backend will make requests for insertion operations in the Persistence module, along with requests for retrieving and modifying data. The DBMS will handle these requests and send data back according to the Backend's needs. The Backend will deal with all the heavy business logic of the main

application and will behave as the central point of communication between the other two modules. The Backend module will be developed using the Spring Boot framework. As a project requirement, we had to select a Spring Boot/Java EE technology and Spring Boot ended up being a clear cut choice, since it is the most streamlined and intuitive tool. Additionally, we also had more experience with Spring Boot, which was previously used in the past semester in the IES course.

Furthermore, the backend module will follow a traditional multi-layered scheme. This 3-tier models consists of the following layers: Data Access, Business and Presentation. The Data Access layer will be concerned with the Object-Relational Mapping between the Spring Boot Entities and the tables contained in the database and performing data transactions. The Business layer will hold the core business logic for the services of the main application. Finally, the Presentation layer corresponds to the REST API, which will be divided into several different Rest Controllers offering endpoints for communication with external clients.

Lastly, the Client module consists of two separate clients. One of them will be a web application that is the main interface of the marketplace, with the goal of providing an intuitive tool for the regular users of the platform. This web application will be served by the same server as the main marketplace application (it is part of the same Maven project, to make it easier to deploy) and will communicate with the backend in the form of requests to its API and exchange of resources within a JSON payload. These calls to the API will be performed using jQuery and the results will be rendered as HTML to the user's browser.

The other client will represent a publishing company that will perform a typical Business2Business scenario with our main application: they will be making GET requests to login, find out the amount of remaining copies of the books that make up their stock in the marketplace and they will be making a PUT request with new copies. The amount of new copies that will be sent to the marketplace API will depend on the performance of the publisher's total revenues, so the client also makes a GET request to the revenue's endpoint to gather that data. Since this application is supposed to be very simplistic, we selected Flask for the technological stack, because this is a lightweight framework that will only need a smaller amount of configuration to work as intended for this scenario. Overall, the client consumes 4 endpoints from the marketplace's REST API and offers 2 endpoints itself.

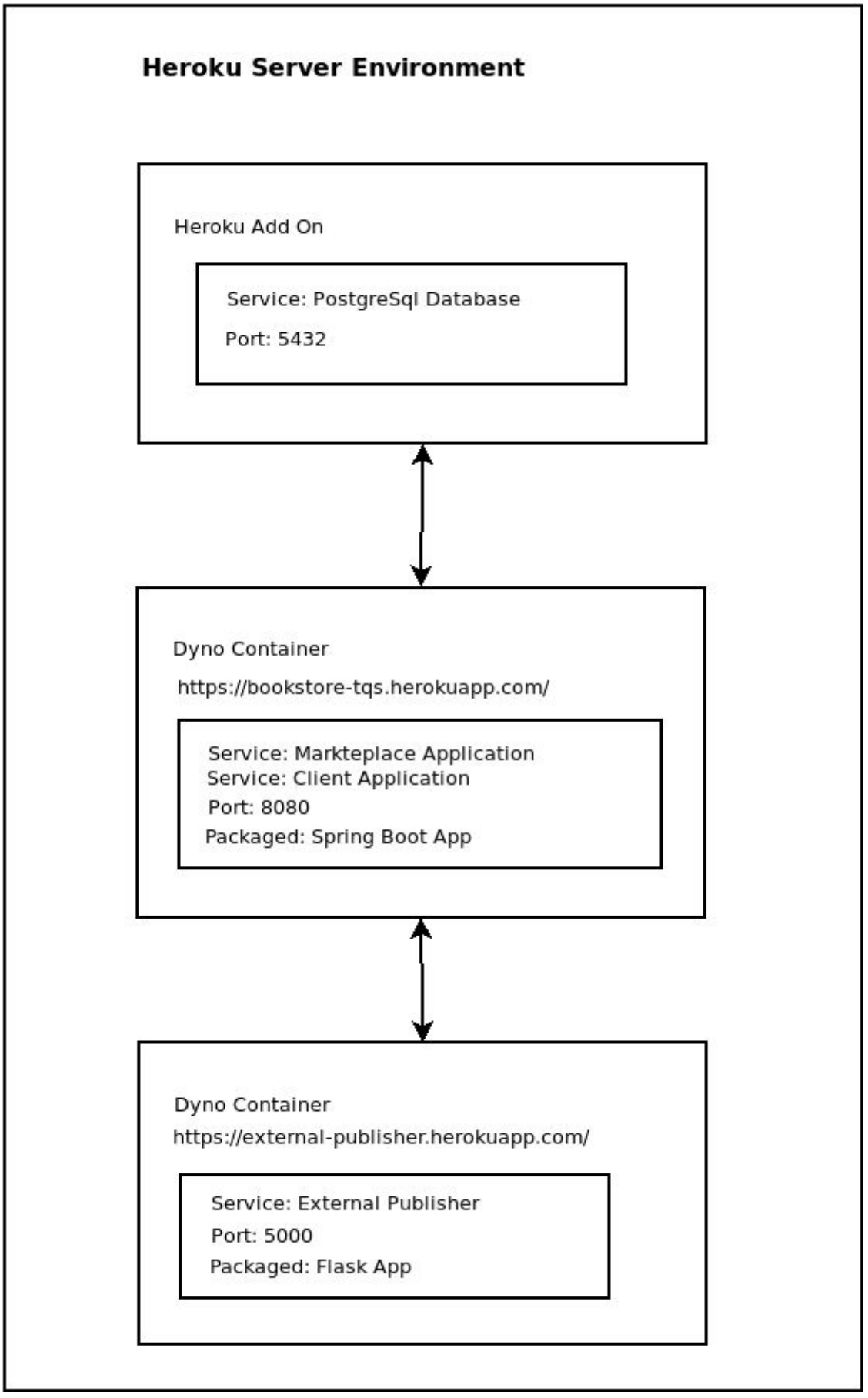


4.2 Deployment architecture

As mentioned in the key requirements, our product will follow a containers-based deployment strategy for each autonomous application. Furthermore, we will use a continuous deployment methodology, which means that every new increment on the product that passes all stages of the established CI pipeline and is added to the master branch of the project, should also be released and readily available for the customer (this approach is the basis of our CD pipeline).

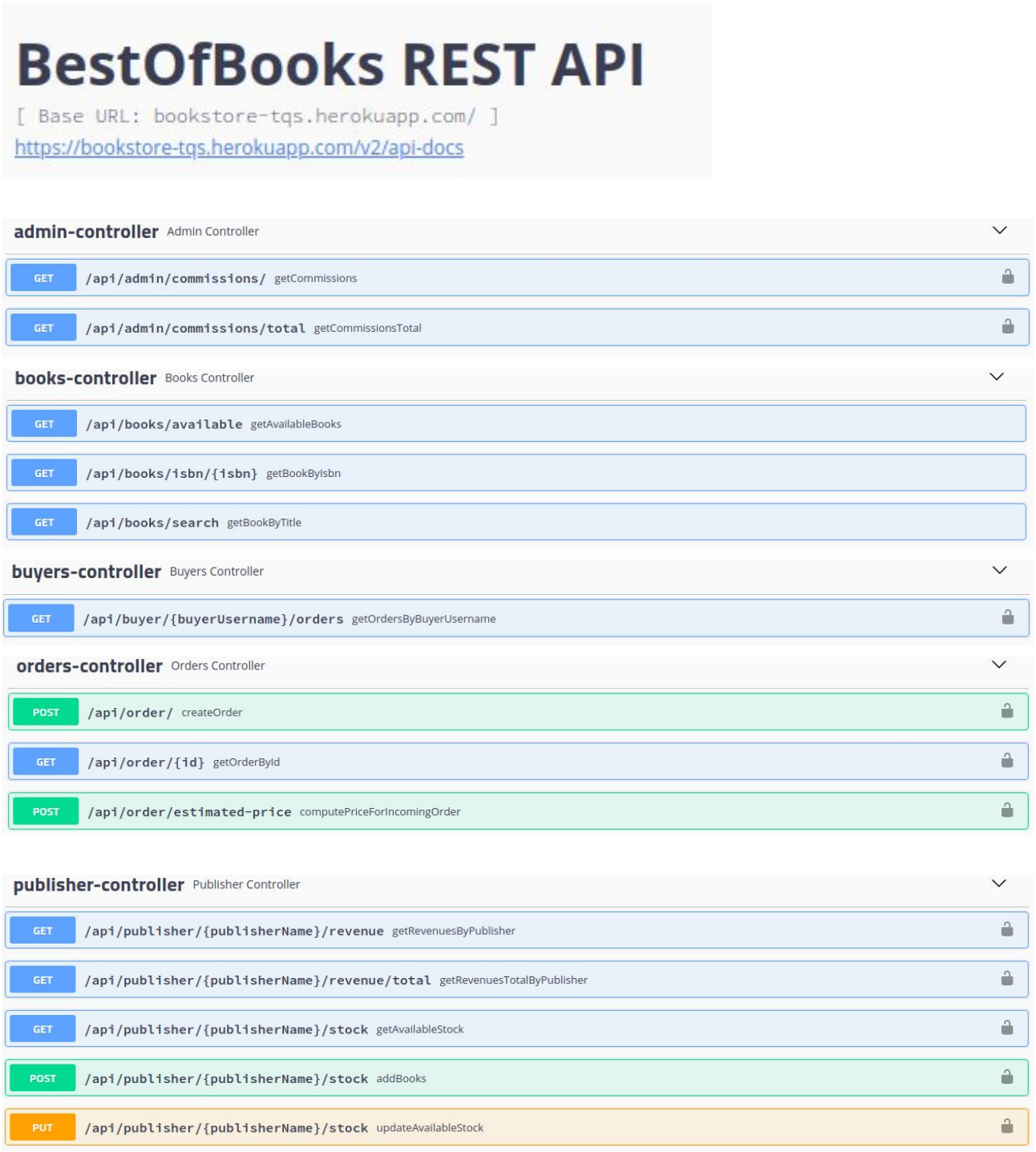
Therefore, the orchestration of the infrastructure of our project should be divided into several dynos that will be deployed to a cloud application platform, through Heroku CLI. Dynos are isolated, virtualized Linux containers, working as a packaging solution for its each app and its dependencies.The PostgreSQL database should also be deployed as a separate service: an Heroku add-on.

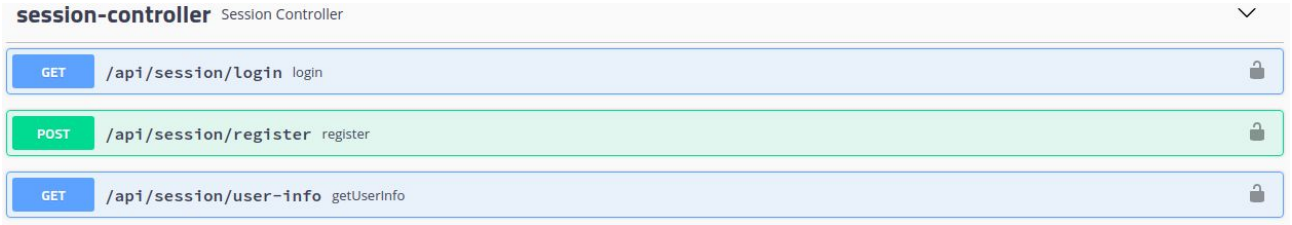
We will be using Heroku, since there is integration between GitHub and Heroku that will streamline the process of automatic deploys through GitHub Actions. Our project has two repositories (one for the marketplace application and the other for the external publisher application) and both were configured in order to be integrated with Heroku.


Heroku Server Environment
Heroku Add On
Service: PostgreSql Database
Port: 5432
Dyno Container
https://bookstore-tqs.herokuapp.com/
Service: Markteplace Application
Service: Client Application
Port: 8080
Packaged: Spring Boot App
Dyno Container
https://external-publisher.herokuapp.com/
Service: External Publisher
Port: 5000
Packaged: Flask App

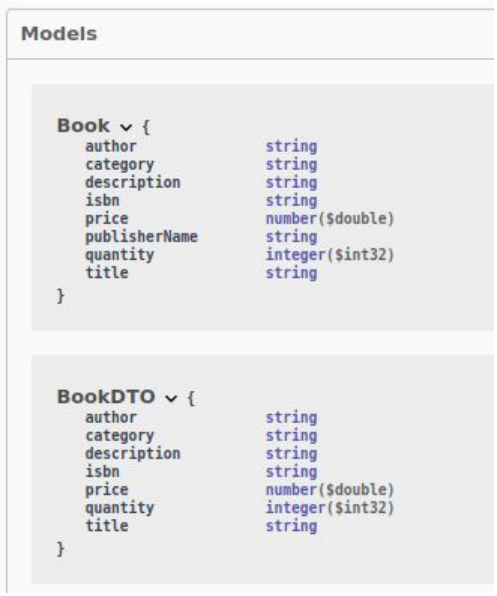
5 API for developers

Our API documentation was developed using Swagger, so that we can provide direct interaction with our endpoints. This documentation will be hosted online and served with our main marketplace application. Use the following link to try it.

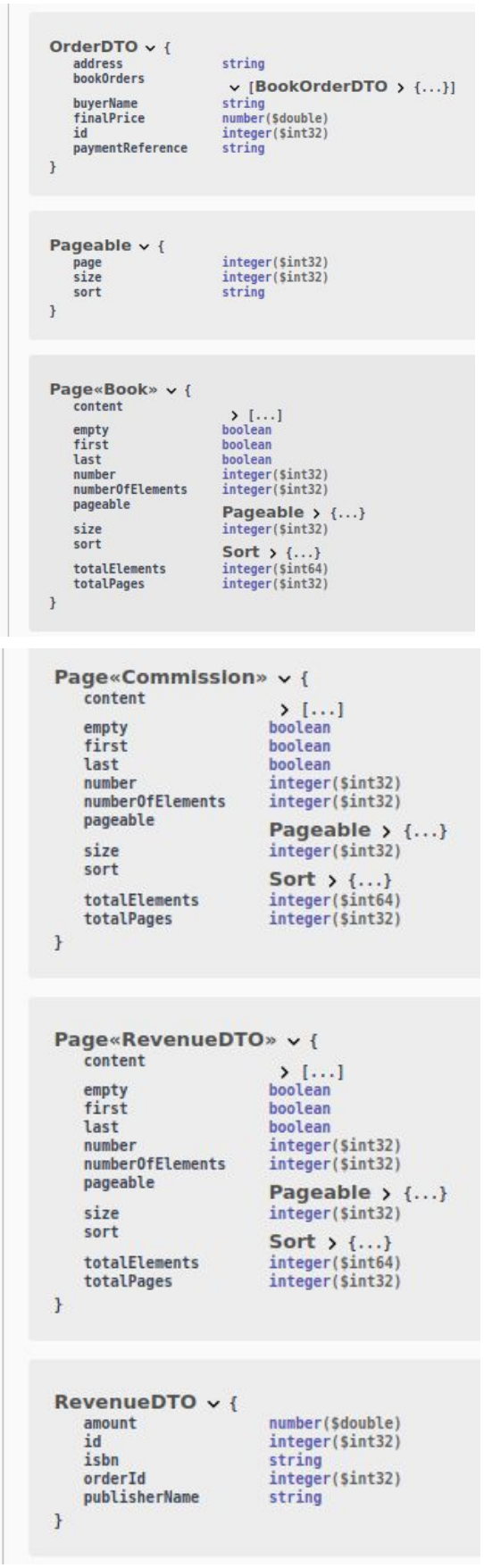



session-controller Session Controller
GET /api/session/login login
POST /api/session/register register
GET /api/session/user-info getUserInfo

Resources


Models
Book { author string category string description string isbn string price number($double) publisherName string quantity integer($int32) title string }
BookDTO { author string category string description string isbn string price number($double) quantity integer($int32) title string }


BookListDTO { books [Book > {...}] }
BookOrderDTO { author string isbn string quantity integer($int32) title string }
Commission { amount number($double) id integer($int32) orderId integer($int32) }
IncomingBookOrderDTO { isbn string quantity integer($int32) }
IncomingOrderDTO { address string bookOrders [IncomingBookOrderDTO > {...}] buyerUsername string paymentReference string }


OrderDTO {
address string
bookOrders [BookOrderDTO {...}]
buyerName string
finalPrice number($double)
id integer($int32)
paymentReference string
}
Pageable {
page integer($int32)
size integer($int32)
sort string
}
Page«Book» {
content [...]
empty boolean
first boolean
last boolean
number integer($int32)
numberOfElements integer($int32)
pageable Pageable {...}
size integer($int32)
sort Sort {...}
totalElements integer($int64)
totalPages integer($int32)
}
Page«Commission» {
content [...]
empty boolean
first boolean
last boolean
number integer($int32)
numberOfElements integer($int32)
pageable Pageable {...}
size integer($int32)
sort Sort {...}
totalElements integer($int64)
totalPages integer($int32)
}
Page«RevenueDTO» {
content [...]
empty boolean
first boolean
last boolean
number integer($int32)
numberOfElements integer($int32)
pageable Pageable {...}
size integer($int32)
sort Sort {...}
totalElements integer($int64)
totalPages integer($int32)
}
RevenueDTO {
amount number($double)
id integer($int32)
isbn string
orderId integer($int32)
publisherName string
}

```
Sort ⌄ {
   empty                boolean
   sorted               boolean
   unsorted             boolean
}

StockDto ⌄ {
   isbn                 string
   quantity             integer($int32)
}

UserDto ⌄ {
   attributes
                         › {...}
   userType             string
   username             string
}
```

Regarding the external publisher application, there are only two available endpoints and resources. The API documentation is also available on the base url.

External Publisher REST API

Overview

The External Publisher API allows client software to retrieve and operate on a particular publisher's stock by making HTTP requests to transfer JSON-encapsulated data.

Base url

The default base url is `https://external-publisher.herokuapp.com/api`.

Endpoints

`https://external-publisher.herokuapp.com/api/stock/renew`

Request type: GET

Goal: Updates the quantities of the previously sold-out books from External Publisher in the Marketplace. The quantity in the update is based on revenue metrics.

Successful responses to this request return one instance of the StockDTO resource for each book that is being updated.

`https://external-publisher.herokuapp.com/api/stock/sold-out`

Request type: GET

Goal: Retrieves all the books associated from the External Publisher that currently have zero copies available in the Marketplace platform.

Successful responses to this request return a Pageable with zero or several instances of the Book resource.

Resources

Book

Properties:

Name	Type	Description
author	String	Author of the book.
category	String	Genre that the book belongs to.
description	String	Optional description of the book.
isbn	String	Isbn-13 of the book.
price	double	Selling price of the book.
publisherName	String	Name of the publishing company.
quantity	int32	Amount of copies of the book that are available for sell in the marketplace.
title	String	Title of the book.

StockDTO

Properties:

Name	Type	Description
isbn	String	Isbn of the book that will be updated.
quantity	int32	Amount of new copies that will be added to the previous available stock (in this case, the previous stock is always 0).

6 References and resources

https://www.bookdepository.com/
https://www.treinaweb.com.br/blog/documentando-uma-api-spring-boot-com-o-swagger/
https://dev.to/heroku/deploying-to-heroku-from-github-actions-29ej
https://www.heroku.com/flow
https://www.baeldung.com/javax-validation
https://www.linkedin.com/pulse/running-selenium-web-tests-github-actions-moataz-nabil
https://www.sebastianhesse.de/2016/10/14/deploy-multi-module-maven-project-heroku/