Architecture Design

CodexHub

Version table

Author:	Florin Deleanu	Date
	Version 1	March 2022
	Version 2	May 2022

Contents

Back end design

My backend uses a microservices approach to architecture. The functionality is split between the different components that communicate to one another.

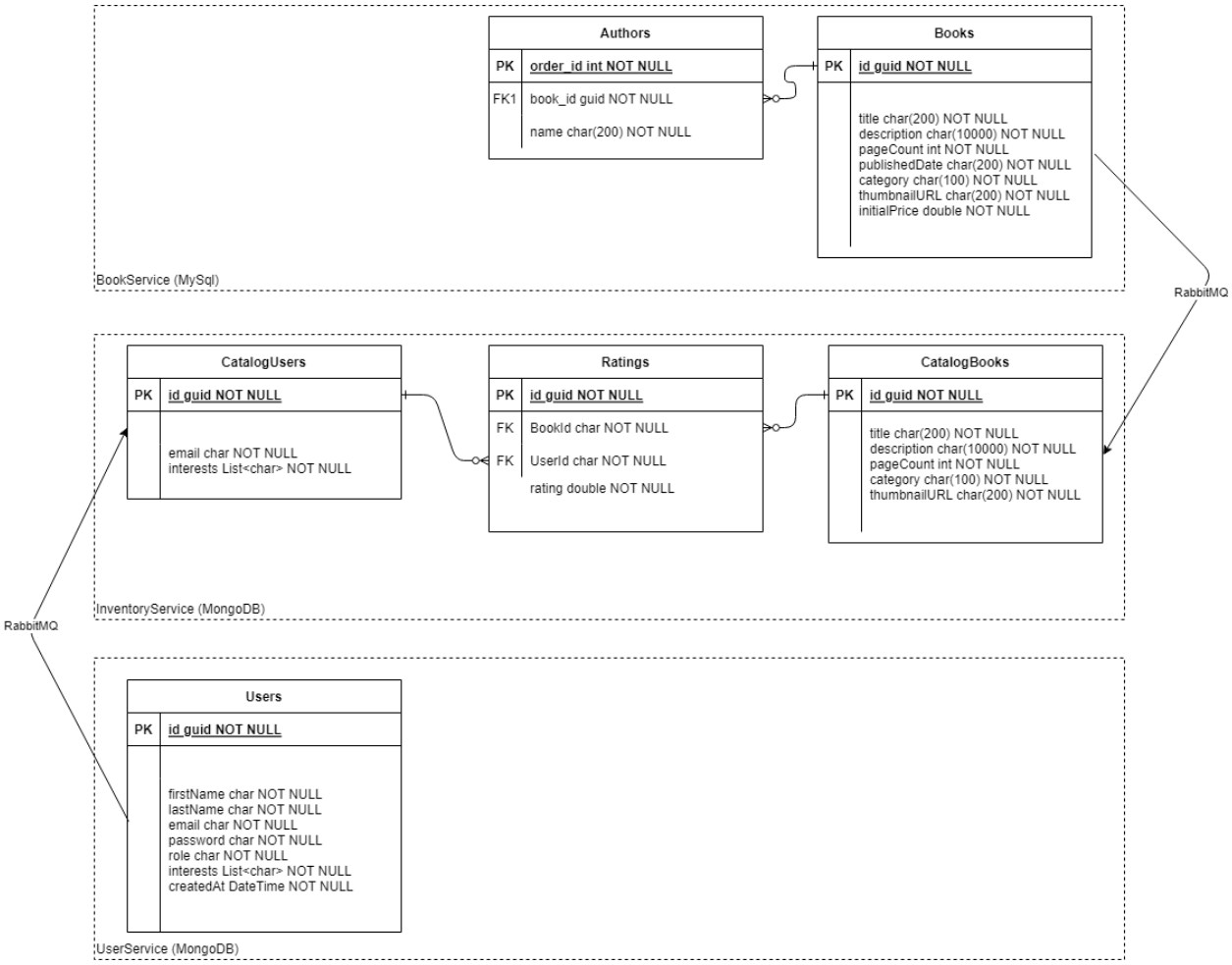
Each microservice uses a 3 layer model to split functionality. They use a layer for user access, one for business logic and one for data manipulation. The inventory and user services use a NuGet package created by me to improve code reusability. This package also provides other functionality such as all the settings for RabbitMQ, so the other services only need to import it and call the corresponding function

GDPR compliance

As the application is supposed to mimic a real world use case as close as possible I needed to keep in mind GDPR compliance. The 7 principles of protection are as follows: Lawfulness, fairness and transparency, Purpose limitation, Data minimization, Accuracy, Storage limitation, Integrity and confidentiality, Accountability.

My application only stores basic data about users such as their first/last name, email, interests and role. Their password is stored as a hash in database to protect against breaches. The user is allowed to delete their account at any point of using their application. This delete will be propagated to all the relevant microservices and all of the data related to the user in each of them will be deleted.

Entity Relationship Diagram

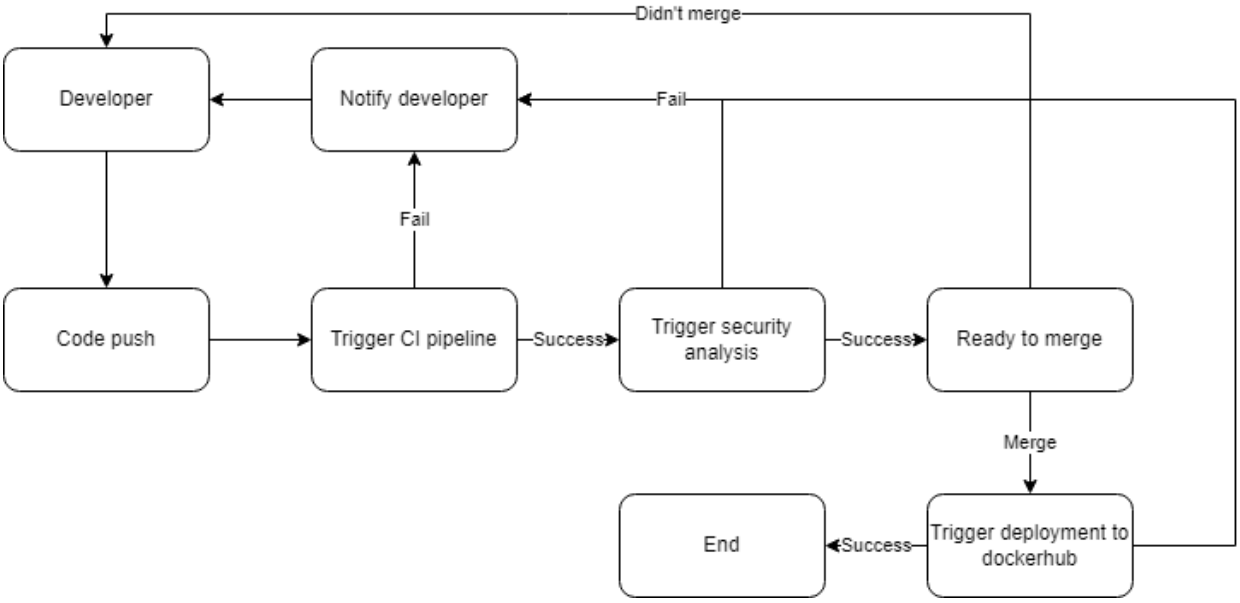


This diagram shows the different data layers and how they communicate with each other. There are 3 services that make use of databases: BookService, UserService and InventoryService.

Although MongoDB is used for the inventory microservice, the data format and link via foreign keys is guaranteed by code constraints in the back-end.

The data for the models from the InventoryService is taken via RabbitMQ and populated whenever its corresponding resource is created in any of the other microservices. The same goes for when that resource is updated or deleted. So, eventual consistency across data is also guaranteed by the use of this message broker.

Pipeline diagram

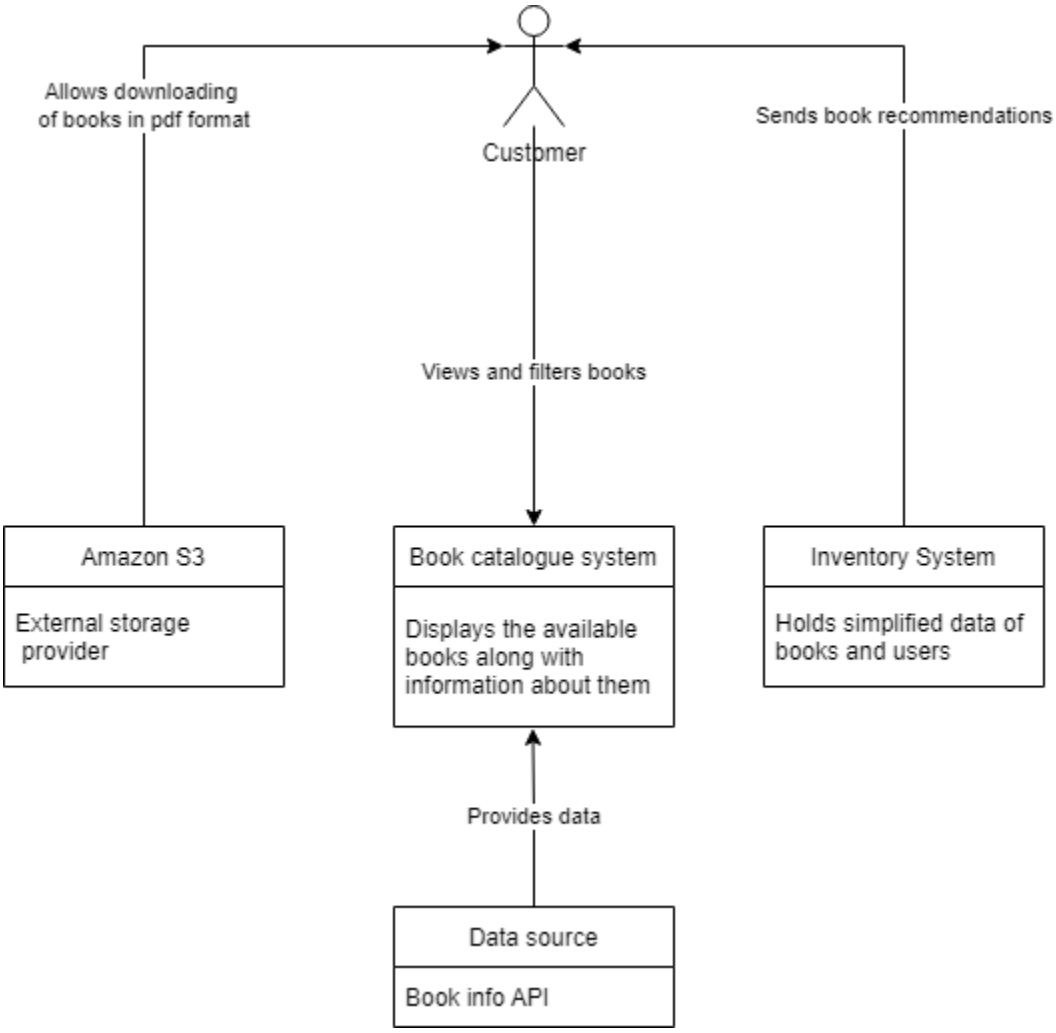


The automated pipeline follows a simple continuous integration/continuous deployment sequence. On each push by the developer the code is built and the tests are run. After that, a security analysis on the code is run using Snyk. Once everything performs as expected and the developer merges into master, the docker image is built and pushed to Dockerhub.

C4 design Model

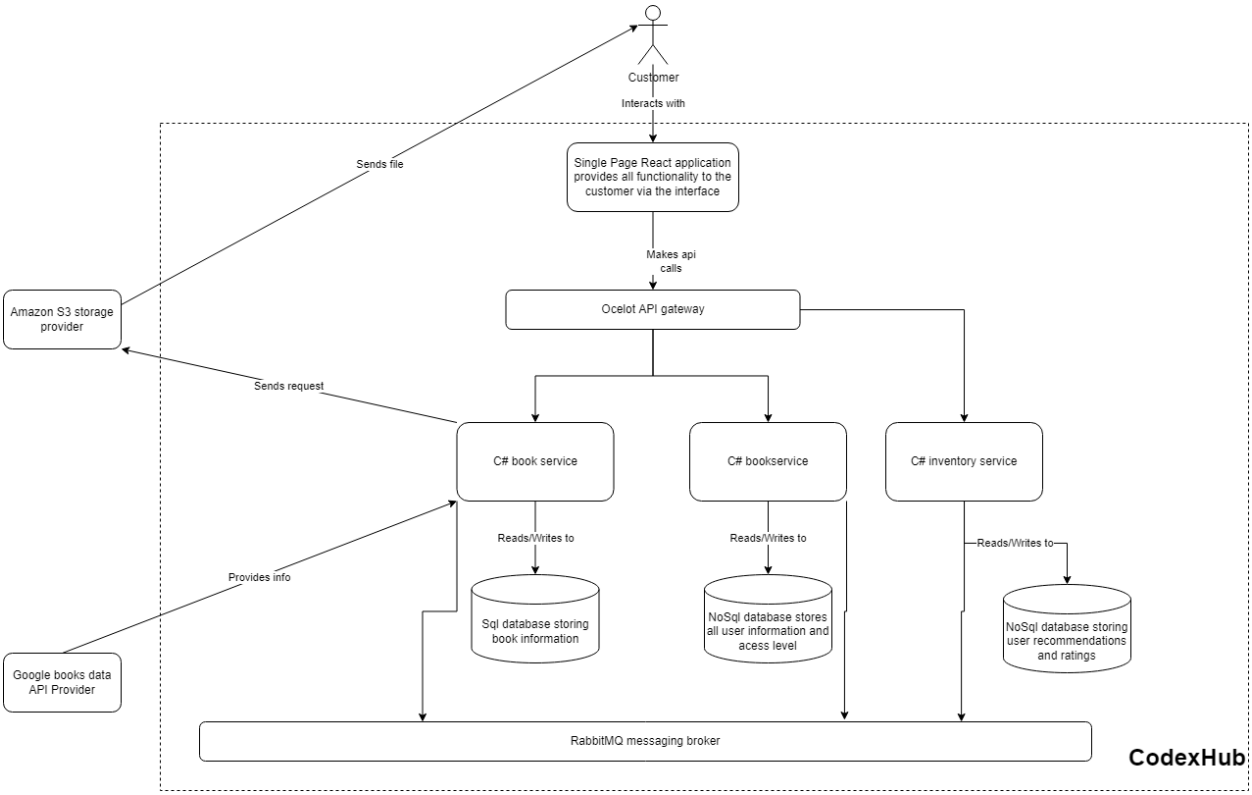
C1 Context Diagram

This diagram shows the big picture of the entire system. It shows how the user would interact with different existing parts of the system.



C2 Container Diagram

The diagram separates the entire system into individual components such as the microservices implemented and external systems. It also shows how the user would interact with these components and how these components interact with one another.



C3 Component Diagram

The component diagram looks further into the container and gives an overview of the individual microservices and their interactions. The simple overview is given in the diagram below.