Enterprise Distributed System

Book Store and Warehouse System

Assignment #2

Master in Informatics and Computing Engineering

Distribution and Integration

Technologies

Mariana Duarte Guimarães, up201307777@fe.up.pt

Tiago Duarte Carvalho, up201504461@fe.up.pt

Faculty of Engineering of the University of Porto

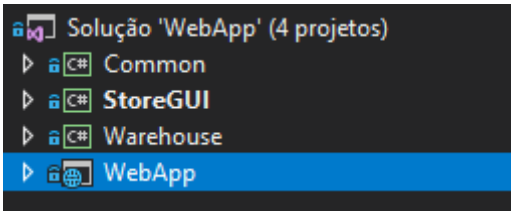
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

May 26, 2019

Architecture Specification

The implemented solution is comprised of a Web Application with the API (web application using .NetCore and React), two GUIs, Warehouse (console application using .Net Framework) and Store (console application and winForms using .Net Framework), a library Common (class library using .Net Standard) , and the information used is kept in a SQL Server Express database.

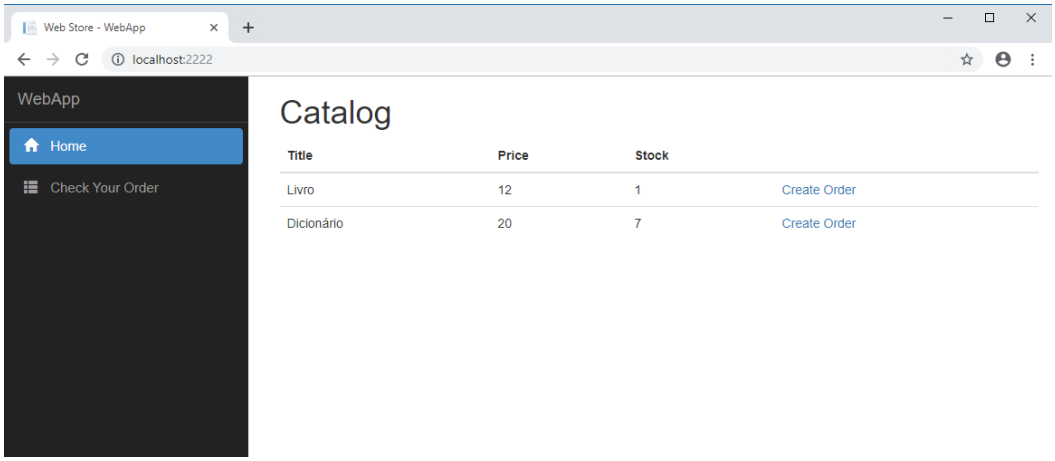
All components communicate through a REST API, implemented using AspNet Web API and messages queues, created using RabbitMQ.



Composing modules

Web application

The web application has three main pages, a list of books, a page to check the orders of a client, given their email, and a page to make an order of a book selected from the list.



Store GUI

The store is composed of:

- a console where receives and acknowledge the messages with the orders to be updated, sent by the warehouse,
- a list of books, shown in a winform, and a way to search for a specific book and make an order,
- a window to create a order for the selected book, also in winforms,
- and a receipt printer called when a sale is made.

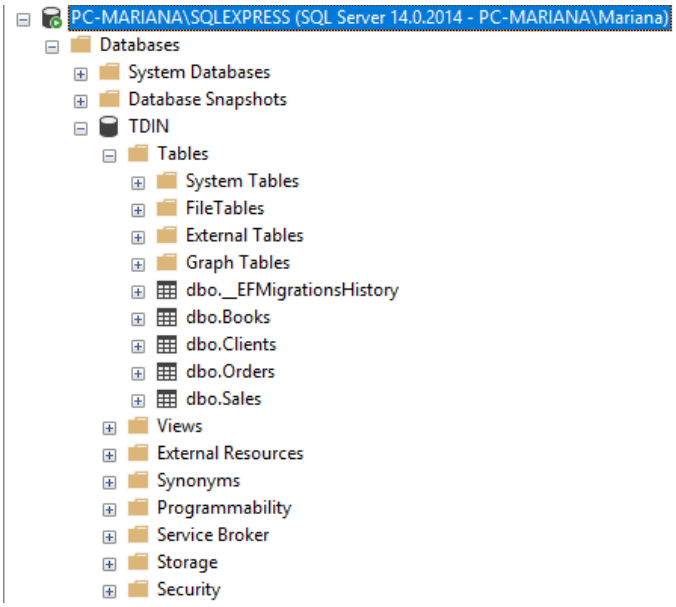
Warehouse

The warehouse is, when an order is made with the status “awaiting expedition”, the store sends a message through a message queue, where that message is received and acknowledged, sending another message to the store and updating the orders status.

Database

The database has the four models created in the library Common, Book, Client, Order, Sale. And is created through the Entity Framework by Microsoft, the Models, and the following code:

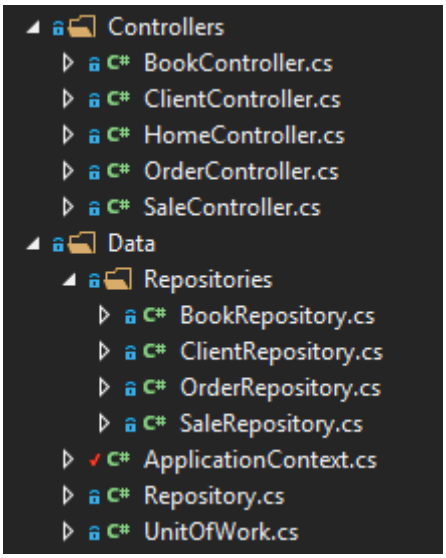
```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using Common.Models;

namespace WebApp.Data
{
    21 referências | niceStorm, há 7 dias
    public class ApplicationContext : DbContext
    {
        0 referências | niceStorm, há 17 dias | 0 exceções
        public ApplicationContext(DbContextOptions<ApplicationContext> options) : base(options) { }

        4 referências | niceStorm, há 16 dias | 0 exceções
        public DbSet<Order> Orders { get; set; }

        3 referências | niceStorm, há 16 dias | 0 exceções
        public DbSet<Client> Clients { get; set; }

        3 referências | niceStorm, há 16 dias | 0 exceções
        public DbSet<Book> Books { get; set; }

        3 referências | niceStorm, há 16 dias | 0 exceções
        public DbSet<Sale> Sales { get; set; }


        0 referências | niceStorm, há 7 dias | 0 exceções
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Book>().HasData(
                new Book { Id = 1, Title = "Livro", Price = 12, Amount = 12 },
                new Book { Id = 2, Title = "Dicionário", Price = 16, Amount = 27 },
                new Book { Id = 3, Title = "Romance", Price = 20, Amount = 6}
                );
            modelBuilder.Entity<Client>().HasData(
                new Client { ID = 1, Name = "Maria", Address = "Rua 123", Email = "maria@mail.com", OrdersClient = new List<Order>() }
                );
        }
    }
}
```



Composing services

API

The API is created in the Controllers folder, we created a controller for each data model, and then by unitOfWork and the repositories were able to create each request needed.

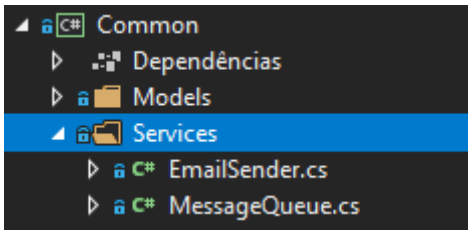


Message Queue and Email Sender

In the Common library we found two services:

EmailSender class, called each time there's an order is created and updated, that sends an email with the order information to the client's email.

MessageQueue, this class is called to create and send a message through the message queue created using RabbitMQ.

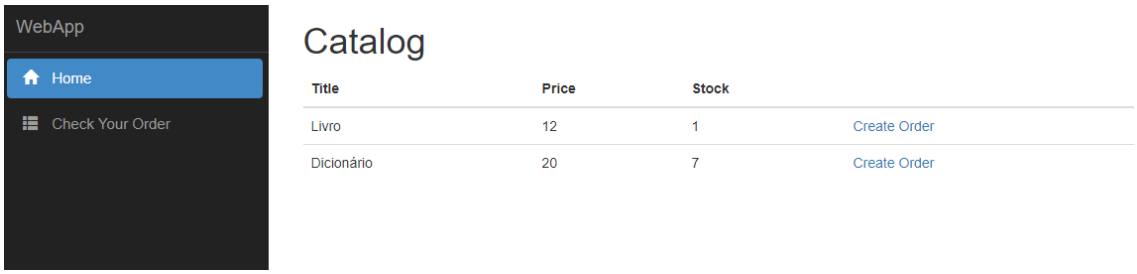


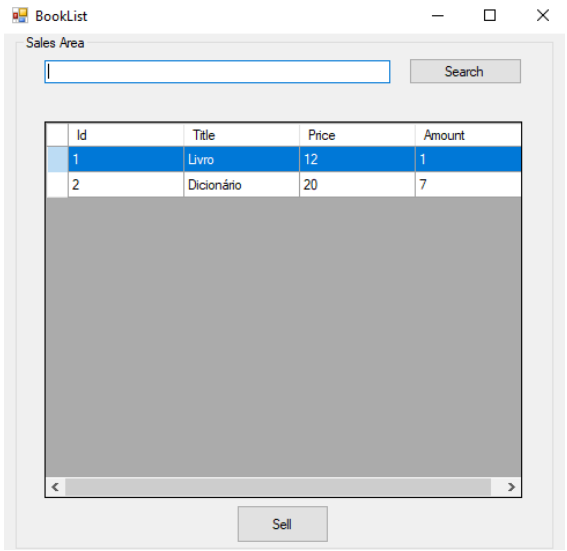
Functionalities

1. consult the store titles, stock and prices of each book
2. make a sale (updates the store stock and prints a receipt)
3. creates an order
4. when the stock is less then the amount requested sends a message to the warehouse requesting 10 units more of the amount requested in order to have enough stock to fulfil the demand.
5. changes the state of an order
6. sends an email with the orders information
7. the client can check their orders and the status of each order in the webapp
8. in the warehouse, a request is received, the orders are updated, and a message to the store, in the case of a store order, is sent.

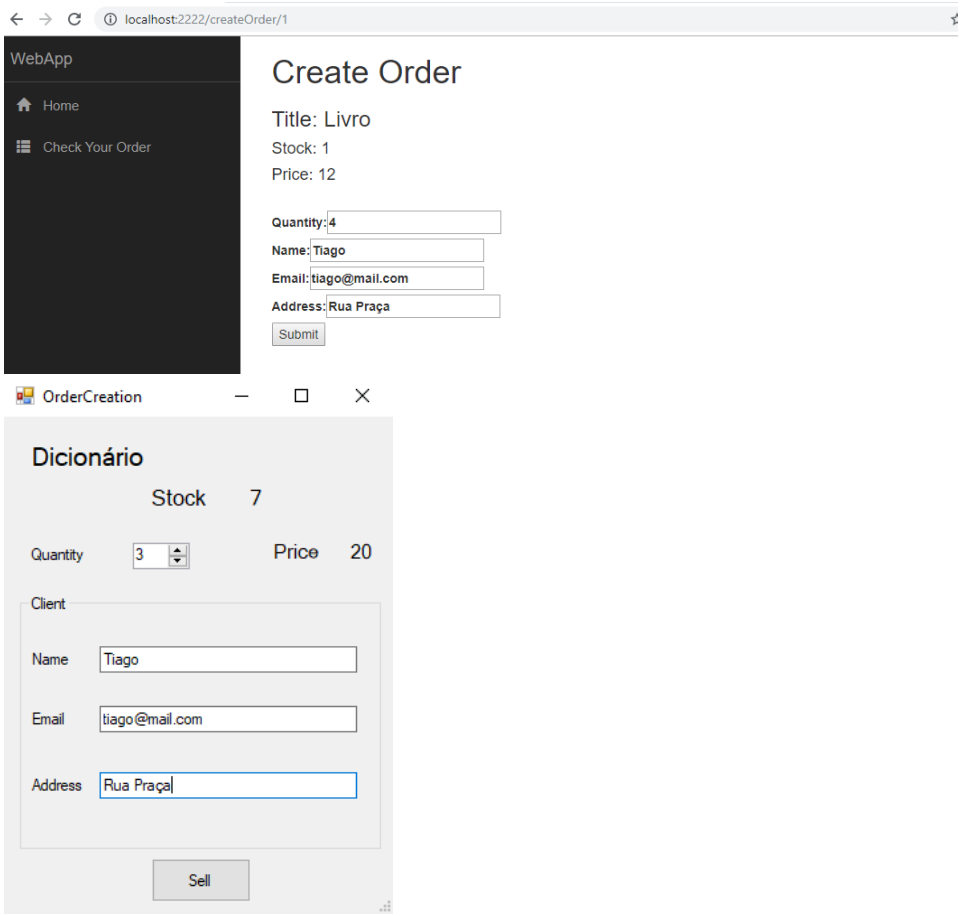
Main flows of use

Catalogues:

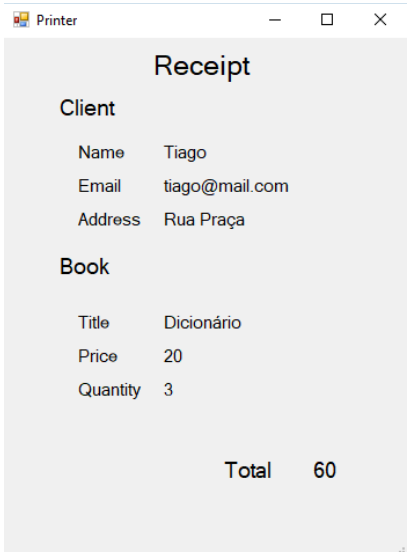




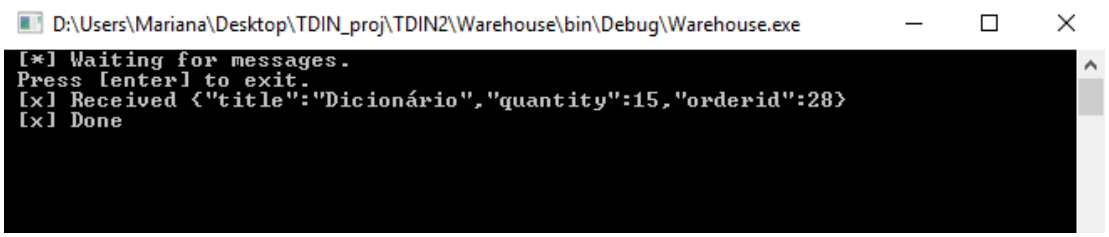
Order/Sale Creation:



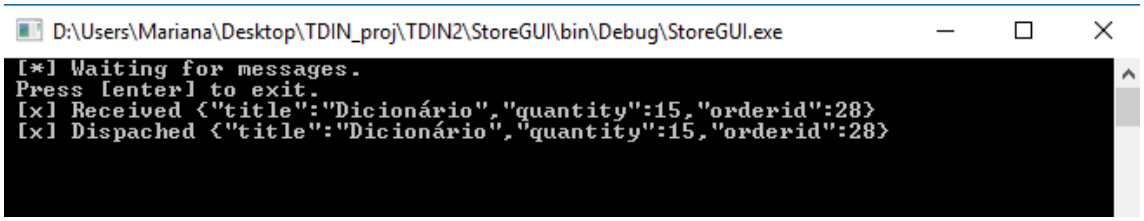
Receipt Printer:



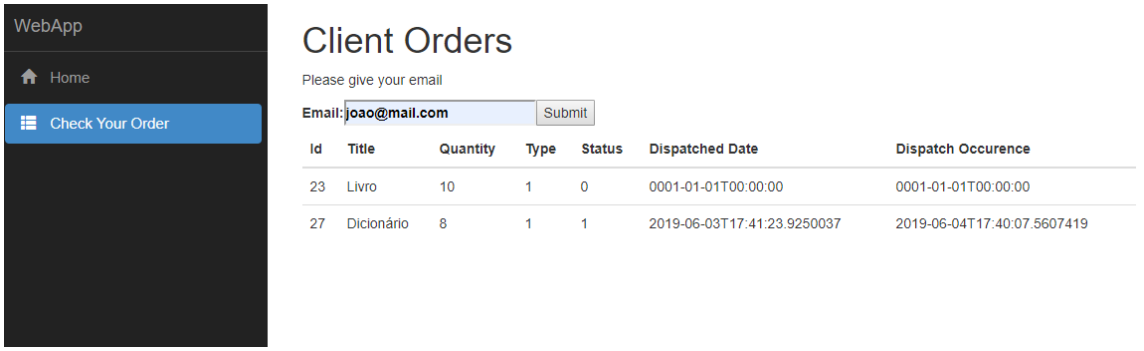
Warehouse:



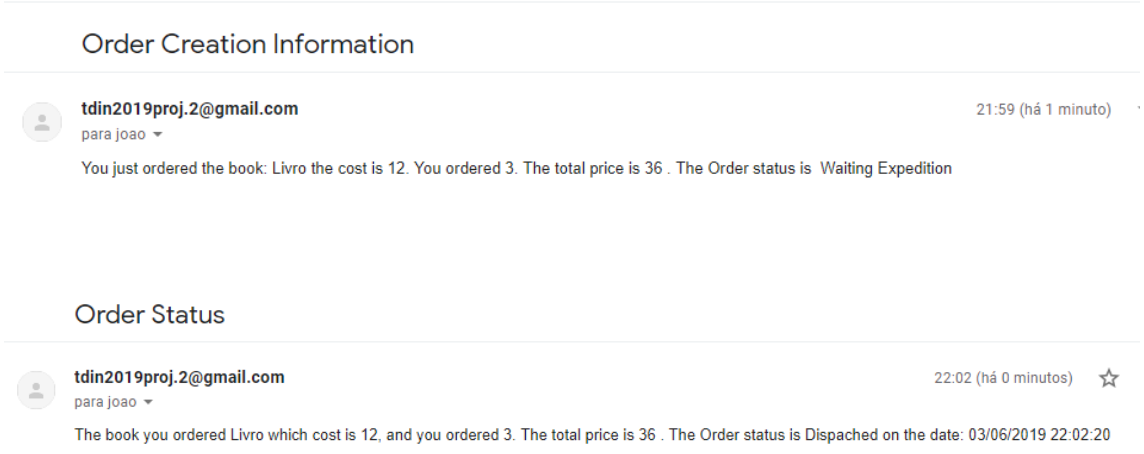
Store Orders Management



Client's Orders



Email Sent



Build and Run Conditions

In order to run this application, you need to install sql server express, and in the visual studio console, run Add-Migration (Migration name) and Update-Database.
Then first run the WebApp and then everything else.