

# Table of Contents

- [Links](#)
- [Tech Stack](#)
- [Features](#)
- [Contributors](#)
- [Screenshots](#)

## Links

Here are the main links for accessing the ELibrary project and its documentation:

- **Web Application:** [ELibrary Application](#)  
Access the live version of the ELibrary application, deployed on Microsoft Azure.
- **Backend Documentation:** [ELibrary Backend Docs](#)  
Explore the backend's API documentation, generated with DocFX.
- **Frontend Documentation:** [ELibrary Frontend Docs](#)  
View the frontend documentation, generated with Compodoc.

## Tech Stack

- **Frontend:** Angular with NgRx for state management
- **Backend:** ASP.NET Core Web API
- **Database:** PostgreSQL with Entity Framework Core
- **Authentication:** JWT and OAuth 2.0
- **API Gateway:** Ocelot for routing across microservices
- **Resilience:** Polly for retry policies and fault tolerance
- **Containerization:** Docker and Kubernetes
- **CI/CD:** GitHub Actions for automated workflows
- **AI Integration:** OpenAI's GPT for recommendations

## Features

- **ASP.NET Web API Backend:** Built using ASP.NET Core, the backend provides a robust, RESTful API that supports a variety of CRUD operations and complex resource management. The application follows a “Code First” approach using Entity Framework Core with a PostgreSQL database, ensuring seamless database migrations and schema management.
- **Angular Frontend with NgRx:** The frontend is built with Angular, offering a dynamic and responsive user experience. It leverages NgRx (Redux pattern) for state

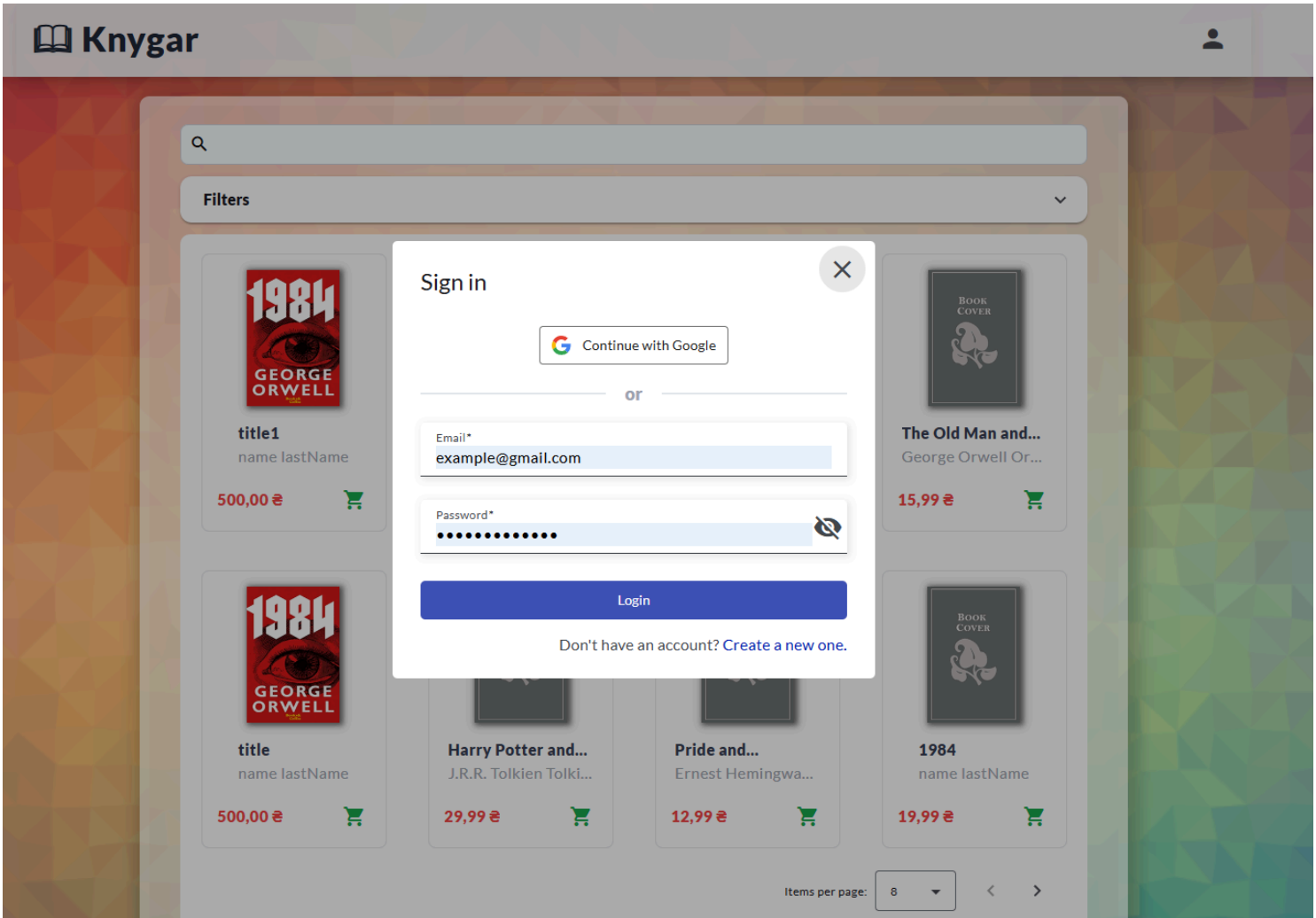
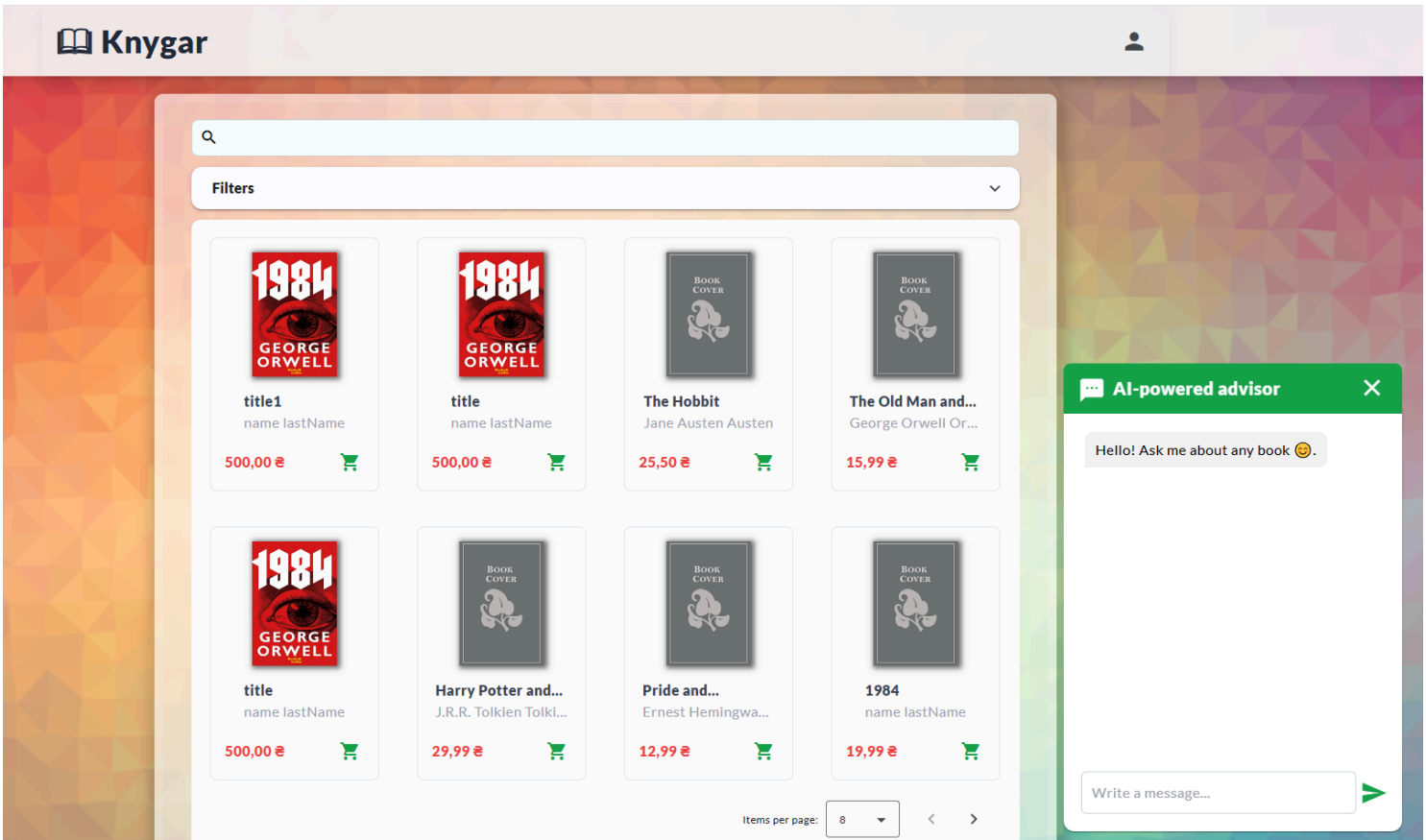
management, making the codebase scalable and maintainable, even as the application grows.

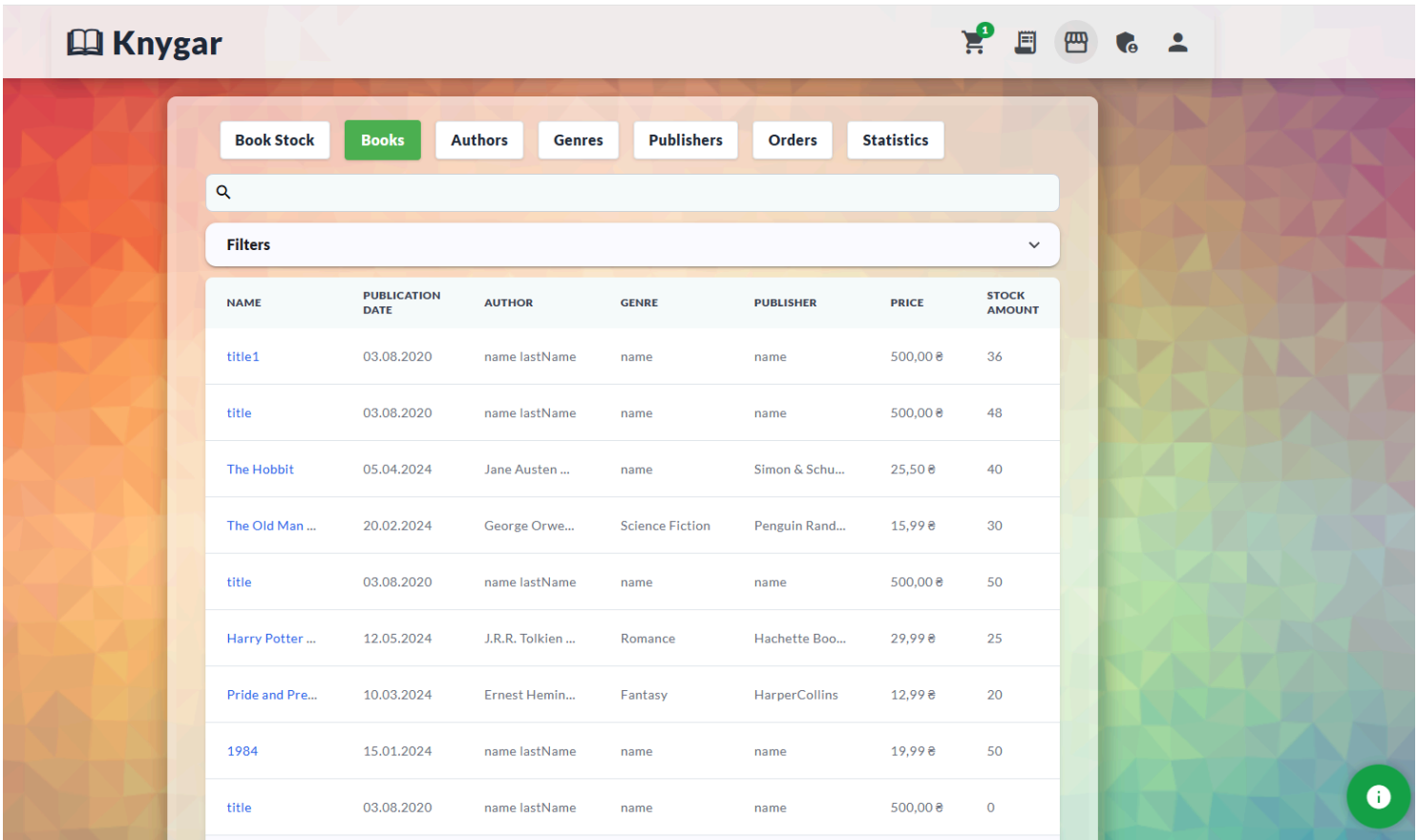
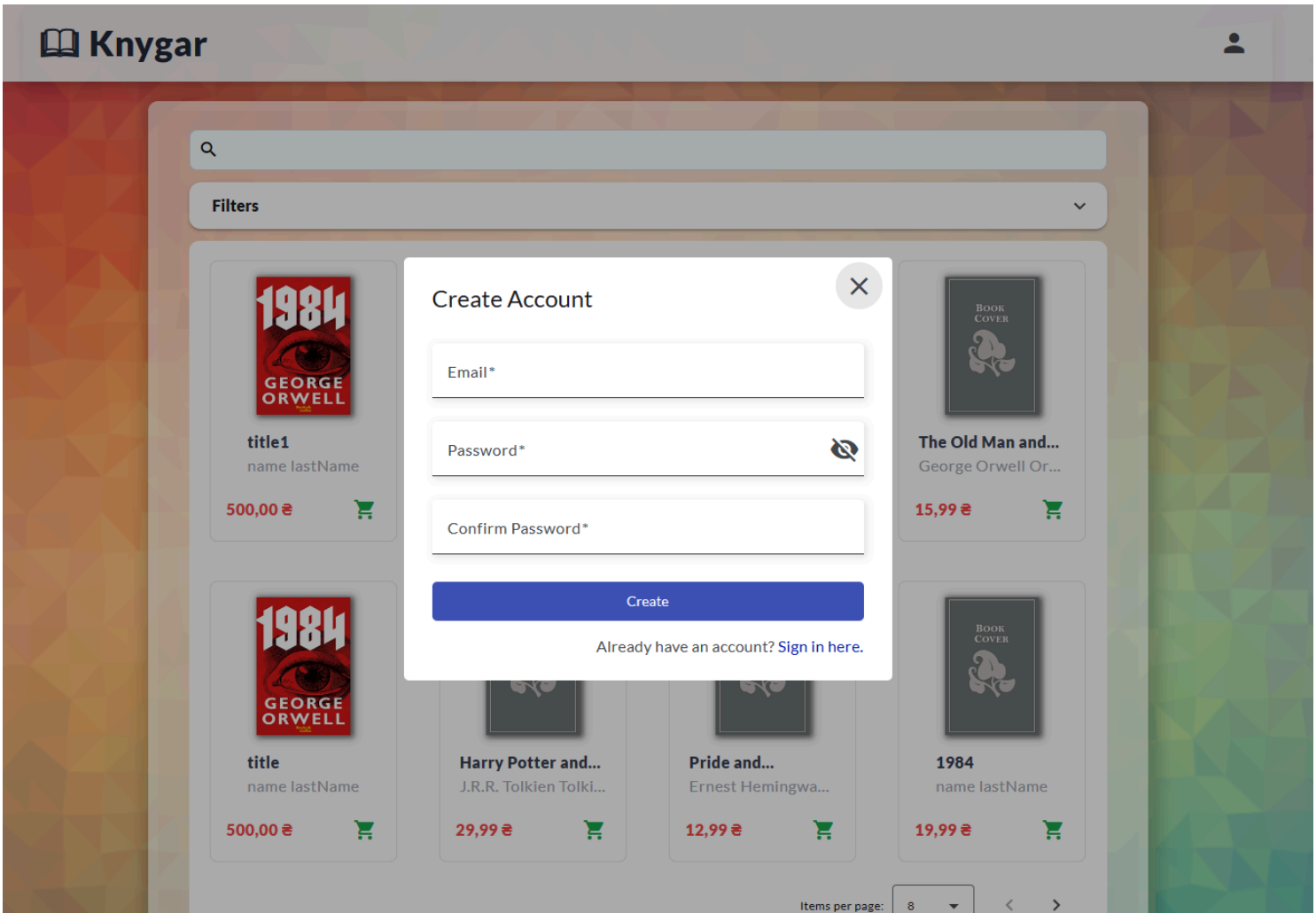
- **User Authentication:** Implements secure user authentication and authorization through JWT-based tokens for session management, and supports OAuth 2.0 for seamless third-party integrations.
- **API Gateway with Ocelot:** An Ocelot API Gateway is used to route and manage requests between microservices, improving scalability and simplifying service management. This setup helps in optimizing requests and load balancing.
- **Polly for Resilience:** The Polly library is integrated to handle transient faults with retry policies, circuit breakers, and timeouts, improving application resilience and reliability under different conditions.
- **OpenAI Integration for AI-powered Book Recommendations:** Leveraging OpenAI's GPT model, hosted on Azure, the application features an AI-based online consultant. The AI can provide personalized book recommendations by accessing real-time data from the database, enhancing user engagement.
- **Containerization with Docker Compose:** All services are containerized using Docker and managed via Docker Compose, ensuring consistent environments across development, testing, and production stages.
- **Continuous Integration and Continuous Deployment (CI/CD):** The project uses GitHub Actions for automated CI/CD pipelines, enabling seamless deployments to Azure. Testing is supported with NUnit and Test Containers to ensure reliability and code quality.
- **Design Patterns:** Incorporates various design patterns for better code organization and maintainability. This includes the Mediator pattern (via MediatR), which facilitates decoupled communication between services.
- **Testing and Quality Assurance:** Implements NUnit and Test Containers for unit and integration tests, ensuring that all components are thoroughly tested in isolated environments.

## Contributors



## Screenshots





Book Stock

Books

Authors

Genres

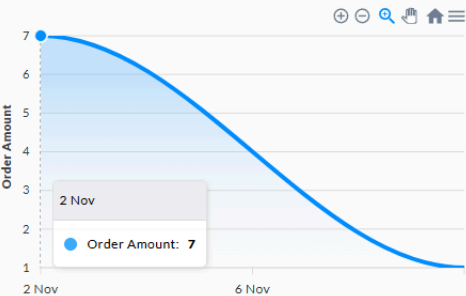
Publishers

Orders

Statistics

Statistics

In Cart Copies (At The Moment) ...	3
In Order Copies .....	17
Sold Copies .....	10
Canceled Copies .....	0
Order Amount .....	8
Canceled Order Amount .....	0
Average Order Price .....	1 062,50 ¢
Earned Money .....	5 000,00 ¢



Date From  
10.11.2024



Date To  
11.11.2024



Time From  
21:40



Time To  
21:40



Book

Contact Information

#7	Completed	03/11/2024, 06:30	2	5 000,00 ₺
#6	In Processing	03/11/2024, 06:30	2	500,00 ₺
#5	In Processing	03/11/2024, 06:30	2	500,00 ₺
#4	In Processing	03/11/2024, 06:30	2	500,00 ₺
#3	In Processing	03/11/2024, 06:30	2	500,00 ₺
#2	In Processing	03/11/2024, 06:30	2	500,00 ₺
#1	In Processing	03/11/2024, 06:30	2	500,00 ₺

Items per page: 10





**Title** ..... Harry Potter and the Sorcerer's Si  
**Author** ..... J.R.R. Tolkien Tolkien  
**Genre** ..... Romance  
**Publisher** ..... Hachette Book Group  
**Publication Date** ... 12/05/2024  
**Page Amount** ..... 400  
**Cover Type** ..... Hard

In stock

29,99 € 

Description

The story of a young boy who discovers he is a wizard and attends Hogwarts School of Witchcraft and Wizardry, uncovering his past and facing dark forces.



# Getting Started

## Prerequisites

Before you begin, ensure you have the following installed on your machine:

1. **Docker:** For building and running containers. [Install Docker](#).
  2. **Minikube:** A lightweight Kubernetes implementation for local testing. [Install Minikube](#).
  3. **kubectl:** Kubernetes command-line tool to manage clusters. [Install kubectl](#).
- 

## Kubernetes / Minikube Setup

1. **Clone the repository:**

```
git clone https://github.com/TEGT0/ELibrary.git
```

2. **Navigate into the Kubernetes folder:**

```
cd ELibrary/k8/dev
```

3. **Start Minikube:** Open a terminal in the folder and start Minikube:

```
minikube start
```

If Minikube is already running, ensure you're in the correct context:

```
kubectl config use-context minikube
```

4. **Optional: Enable Chat Service:** If you want to use the optional chat service:

- Open the `chatbot-conf.yml` file.
  - Set the `OPENAI_API_KEY` environment variable with your OpenAI API key.
- 

## Deployment Steps

Follow these steps in order:

1. Configure ConfigMaps and Secrets



```
kubectl apply -f db-conf.yml
kubectl apply -f backend-conf.yml
kubectl apply -f chatbot-conf.yml # Optional
```

## 2. Deploy the Database

Deploy the database and wait for it to be fully initialized:

```
kubectl apply -f db.yml
kubectl get pods # Verify that the database pod is running.
```

## 3. Deploy the Backend

Deploy the backend services:

```
kubectl apply -f backend.yml
```

## 4. Optional: Deploy the Chat Service

```
kubectl apply -f chatbot.yml
```

## 5. Deploy the Frontend

Deploy the frontend application:

```
kubectl apply -f frontend.yml
```

## 6. Access the Frontend

Expose and forward the frontend service using Minikube:

```
minikube service frontend
```

This command will open the frontend in your default web browser.

---