**KATA - ANALYST -  Online Bookstore (JR)**

[1 Assignment 2](#_Toc208153486)

[2 Requirements 3](#_Toc208153487)

[3 Swagger API Documentation 4](#_Toc208153488)

[4 Complementary Word Document 4](#_Toc208153489)

[4.1 Field mapping 4](#_Toc208153490)

[4.2 Conversions 4](#_Toc208153491)

[4.3 Error handling 4](#_Toc208153492)

[5 SQL Queries 5](#_Toc208153493)

[6 ERD Diagram 7](#_Toc208153494)

[7 Flow Diagrams 8](#_Toc208153495)

[8 Sequence Diagram 9](#_Toc208153496)

[9 User Stories And Acceptance Criteria 10](#_Toc208153497)

[10 Use Case Diagram 11](#_Toc208153498)

[11 Test Case Scenarios 12](#_Toc208153499)

# Assignment

**KATA - ANALYST -  Online Bookstore (JR)**

You are a Functional Analyst at a company that specializes in creating APIs for various businesses. Your task is to design and document an API for a bookstore. The bookstore wants to create an online platform where customers can browse and purchase books. The API should allow customers to search for books, retrieve book details, and place orders.  
  
The system will consist of a front-end part for the user interface and a backend part using REST APIs for processing the requests of the user  
  
**Requirements:**  
Your task is to create a comprehensive API design document that includes the following:

* **Swagger API Documentation:**
* Create a Swagger API documentation that describes the API endpoints, request and response formats, and any other relevant  details. Ensure that the Swagger documentation follows the company's standards.
* **Complementary Word Document:** Create a Word document that provides additional information about the API.
* **Field mapping**: Describe how the API fields map to the database fields.
* **Conversions:** Explain any data conversions that occur between the API and the database.
* **Error handling:** Describe how the API handles errors and exceptions.
* **Acceptance Criteria:** Explain the acceptance criteria for the API
* What constitutes a successful API request?
* What are the expected response formats?
* How will the API handle edge cases and errors?
* **Test Case Scenarios:** Create test case scenarios that cover the following:
* Happy path scenarios (e.g., successful search, retrieve book details, place order)
* Error scenarios (e.g., invalid search query, book not found, invalid order data)
* Edge cases (e.g., searching for books with special characters, ordering multiple books)
* **SQL Queries**: Create SQL queries to create the database tables and relationships.
* **Use Case Diagrams and Flow Diagrams:** Create use case diagrams and flow diagrams that illustrate the API's functionality and how it interacts with the database.
* **User Stories:** create user stories that you would plan in a sprint to have this development in a team.

Keep in mind that you have to make this as you would do it in real life. There is no good or wrong solution, we just want to know how you think, what you find important, what you think of,...  
  
Good Luck.

# Requirements

Create an online platform where customers can browse and purchase books.

Business requirements:

* Browse books
  + Browse all books
  + Browse books with filtering
  + Search book by ISBN code
    - Showing a detailed view of a book
* Purchase books
  + Shopping basket
  + Payment system

Deliverables:

* Swagger Api Documentation
* Word Document
  + Field mapping
  + Conversions
  + Error handling
* Acceptance Criteria
  + Successful api
  + Expected response formats
  + Edge case handling
* Test scenarios
  + Happy path scenarios
  + Error scenarios
  + Edge cases
* SQL Queries
* Use Case Diagrams
* Flow Diagrams
* User Stories

# Swagger API Documentation

For the Swagger documentation, refer to ”GreatBooksSwagger.yml”.

# Complementary Word Document

## Field mapping

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter in Swagger** | **DB Column** | **Type** | **Notes** |
| OrderId | order\_ID | Integer |  |
| OrderDatetime | order\_date\_time | DateTime |  |
| TotalPrice | total\_price | Float |  |
| Quantity | quantity | Integer |  |
| OrderItemId | order\_item\_ID | Integer |  |
| UserId | user\_ID | Integer |  |
| UserName | name | String |  |
| Address | address | String |  |
| BookId | book\_ID | Integer |  |
| Summary | summary | String |  |
| Description | description | String |  |
| Rating | rating | Integer | Between 0 and 10 |
| IsbnCode | isbn\_code | String |  |
| Picture | picture | String | Url path to picture |
| Title | title | String |  |
| Price | price | Float |  |
| Author | author | String |  |
| Category | category | String | Enum with different categories:  - science fiction  - fantasy  - horror  - historical fiction  - mystery  - romance  - memoir  - graphic novel |

## Conversions

There are no conversions needed.

Except for the automatic ID creations in the db (these are UUID).

## Error handling

HTTP Status codes:

|  |  |  |
| --- | --- | --- |
| **Status** | **Meaning** | **Example** |
| 400 Bad Request | Invalid input | Filter on price but typing a name |
| 401 Unauthorized | Not enough permissions | Try to use admin functions without admin permissions |
| 403 Forbidden | Not enough permissions | Try to use admin functions without admin permissions |
| 404 Not Found | Resource is not found | Searching for book that is not known by the db |
| 500 Internal Server Error | Unexpected exception | An unexpected action that creates an error |

There are quite a few errors that could happen in the system.  
Some examples are:

|  |  |  |
| --- | --- | --- |
| **Name** | **Example** | **HTTP Status codes** |
| Title to long | Titles that are longer then 50 characters | 400 |
| Price as string | Typing a string instead of a float | 400 |
| Author to long | Authors that are longer then 50 characters | 400 |
| ISBN code to long | ISBN codes longer then 13 | 400 |
| ISBN code to short | ISBN codes shorter then 13 |  |
| Unknown category | Chosing a category not in the Enum | 404 |
| Category to long | Categories that are longer then 25 characters | 400 |
| No books found with filters | No books match the current filters | 404 |
| No books found with ISBN | No books match the current ISBN code | 404 |

# SQL Queries

To build the database I have chosen to code it according to the Microsoft SQL Server standards, since I am most comfortable with this technology.

**To generate the database:**

*-- DB Generation*

*DROP TABLE IF EXISTS Book;*

*DROP TABLE IF EXISTS OrderItem;*

*DROP TABLE IF EXISTS [Order];*

*DROP TABLE IF EXISTS [User];*

*CREATE TABLE Book(*

*book\_ID INT NOT NULL IDENTITY(1, 1),*

*title VARCHAR(50) NOT NULL,*

*picture VARCHAR(150),*

*price FLOAT NOT NULL,*

*summary VARCHAR(75),*

*description VARCHAR(500),*

*rating INT,*

*author VARCHAR(50),*

*isbn\_code VARCHAR(13),*

*category VARCHAR(25),*

*CONSTRAINT book\_pk PRIMARY KEY (book\_ID),*

*CONSTRAINT chk\_category CHECK (category in ('science fiction', 'fantasy', 'horror', 'historical fiction', 'mystery', 'romance', 'memoir', 'graphic novel')),*

*CONSTRAINT chk\_rating CHECK (rating > 0 AND rating < 11));*

*CREATE TABLE [User](*

*user\_ID INT NOT NULL IDENTITY(1, 1),*

*name VARCHAR(50) NOT NULL,*

*address VARCHAR(100),*

*CONSTRAINT user\_pk PRIMARY KEY (user\_ID));*

*CREATE TABLE [Order](*

*order\_ID INT NOT NULL IDENTITY(1, 1),*

*user\_ID INT NOT NULL,*

*order\_date\_time DATETIME,*

*total\_price FLOAT NOT NULL,*

*CONSTRAINT order\_pk PRIMARY KEY (order\_ID),*

*CONSTRAINT user\_fk FOREIGN KEY (user\_ID) REFERENCE User (user\_ID));*

*CREATE TABLE OrderItem(*

*order\_item\_ID INT NOT NULL IDENTITY(1, 1),*

*order\_ID INT NOT NULL,*

*book\_ID INT NOT NULL,*

*quantity INT NOT NULL,*

*price FLOAT NOT NULL,*

*CONSTRAINT orderItem\_pk PRIMARY KEY (order\_item\_ID),*

*CONSTRAINT book\_fk FOREIGN KEY (book\_ID) REFERENCES Book (book\_ID),*

*CONSTRAINT order\_fk FOREIGN KEY (order\_ID) REFERENCES Order (order\_ID),*

*CONSTRAINT chk\_quantity CHECK (quantity > 0));***To generate data for easy testing (Test DB):**

*-- Generating Data*

*INSERT INTO Book (title, picture, price, summary, description, rating, author, isbn\_code, category)*

*VALUES*

*('Dune', 'https://www.GreatBooks.com/images/dune.jpg', 19.99, 'Classic sci-fi', 'Desert world epic', 9, 'Frank Herbert', '9780441172719', 'science fiction'),*

*('IT', 'it.jpg', 14.50, 'Horror novel', 'Evil clown story', 8, 'Stephen King', '9781501142970', 'horror'),*

*('Pride and Prejudice', 'https://www.GreatBooks.com/images/', 9.99, 'Romantic classic', 'Love and society', 10, 'Jane Austen', '9780141199078', 'romance');*

*INSERT INTO [User] (name, address)*

*VALUES*

*('Sam Rotthier', 'Warandeberg 3, 1000 Brussel'),*

*('Bob Smith', 'Warandeberg 3, 1000 Brussel');*

*INSERT INTO [Order] (user\_ID, order\_date\_time, total\_price)*

*VALUES*

*(1, GETDATE(), 34.49),*

*(2, GETDATE(), 14.50);*

*INSERT INTO OrderItem (order\_ID, book\_ID, quantity, price)*

*VALUES*

*(1, 1, 1, 19.99),*

*(1, 3, 1, 9.99),*

*(2, 2, 1, 14.50);*

**Some useful queries that can be used to explore the database:**

*-- Useful Queries*

*SELECT TOP 1000 \* FROM [User];*

*SELECT TOP 1000 \* FROM Book;*

*SELECT TOP 1000 \* FROM [Order];*

*SELECT TOP 1000 \* FROM OrderItem;*

*SELECT TOP 1000 \* FROM [User]*

*WHERE name like '%Sam%';*

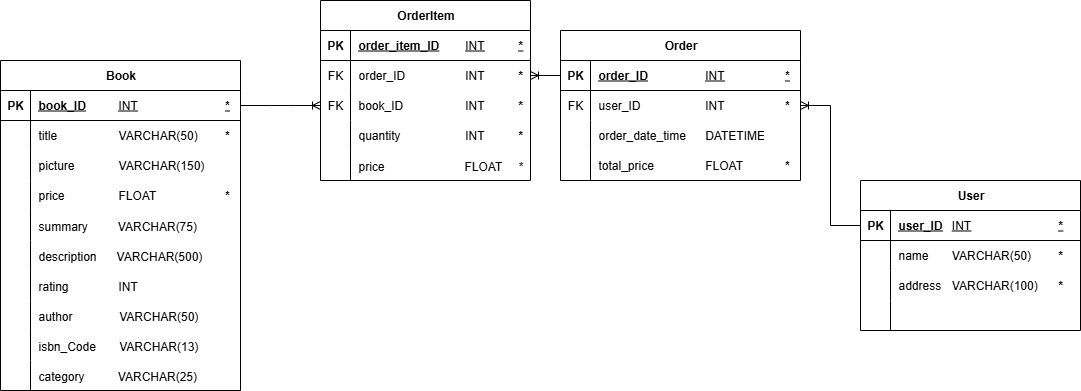
*SELECT TOP 1000 \* FROM Book*

*WHERE title LIKE '%Dune%';*

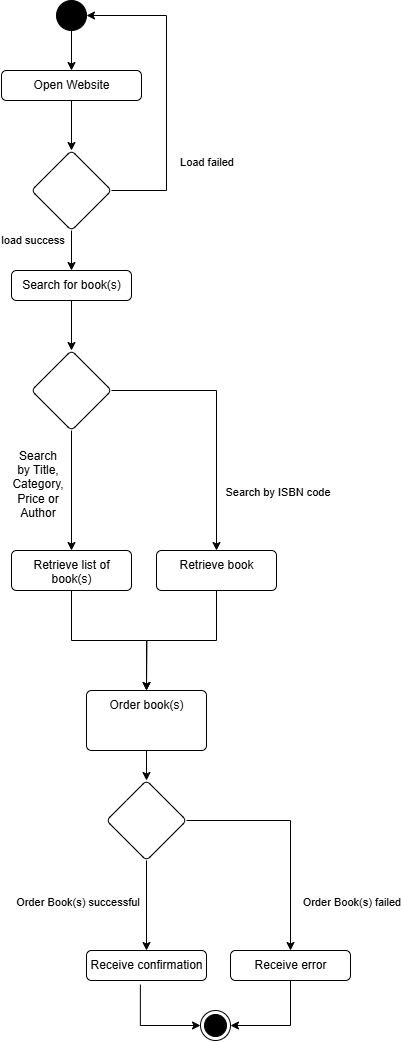
*SELECT top 1000 \* FROM [OrderItem]*

*Where order\_ID = 10;*

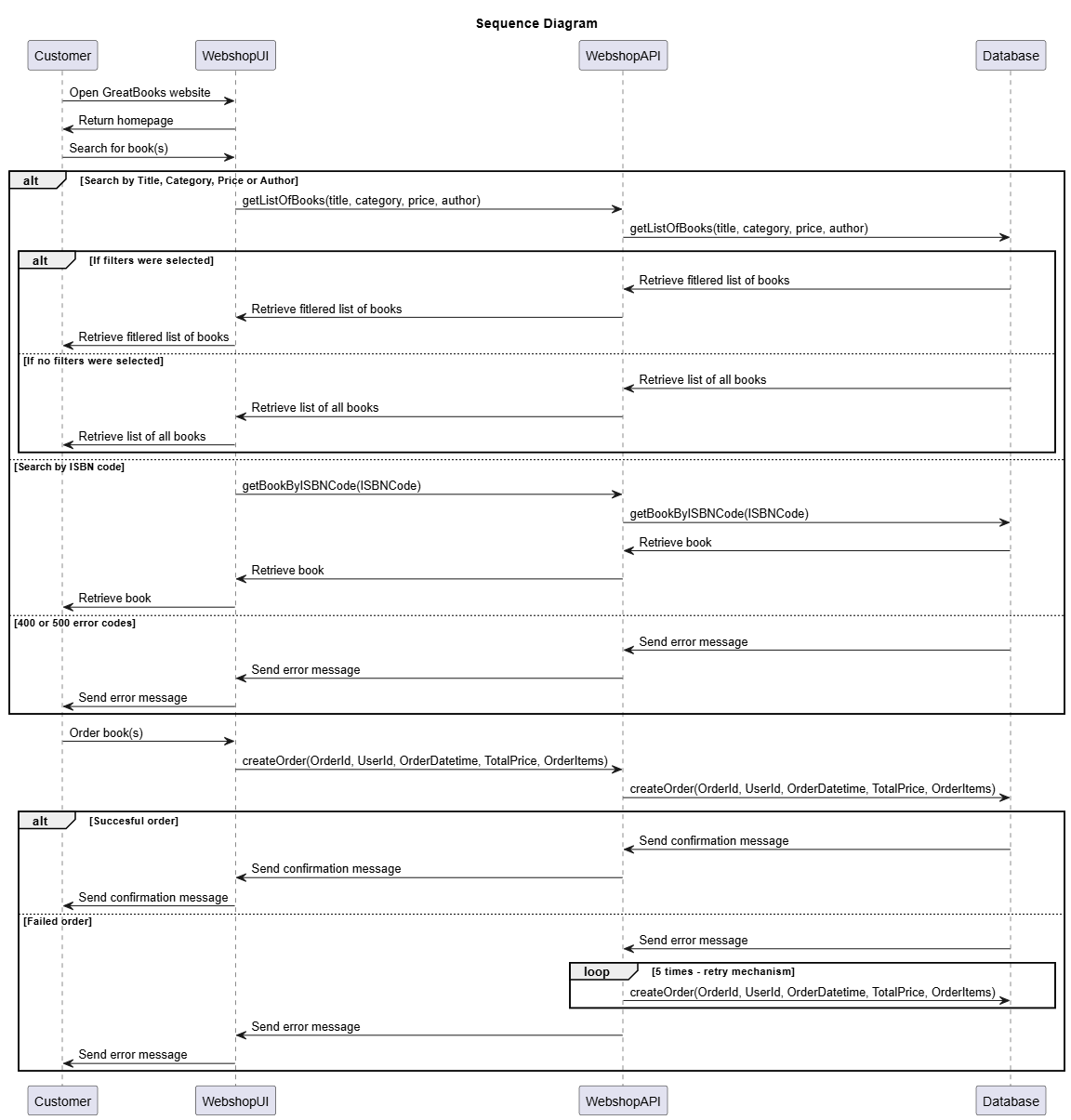
# ERD Diagram



# Flow Diagram



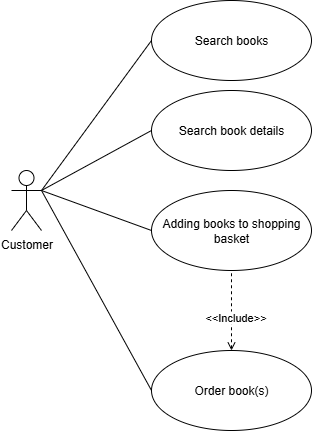
# Sequence Diagram



# User Stories And Acceptance Criteria

|  |  |  |
| --- | --- | --- |
| **User Story** | **Acceptance Criteria** | **Notes** |
| Browse books | As a Customer, I should be able to browse all the books whether or not I use filters to refine my browsing. |  |
| Get book details | As a Customer, I should be able to get more information on a certain book. |  |
| Shopping basket | As a Customer, I should be able to add/remove books from my shopping basket (to eventually place an order). | This will be handled in front-end |
| Place order | As a Customer, I should be able to place an order of book(s) |  |

# Use Case Diagram



# Test Case Scenarios

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Precondition | Steps | Expected Result | Type |
| Search books by filters (title) | Database exists and has data | - Call:  https://GreatBooks.com/api/v1/books?title=TestValue  - Verify the returned list of book(s) | - 200 OK  - JSON array with books containing “TestValue” in the title | Happy flow |
| Search books by filters (title with special characters) | Database exists and has data | - Call:  https://GreatBooks.com/api/v1/books?title=Test#Value  - Verify the returned list of book(s) | - 200 OK  - JSON array with books containing “Test#Value” in the title | Edge flow |
| Search books by filters (Category) | Database exists and has data | - Call: https://GreatBooks.com/api/v1/books?category=horror  - Verify the returned list of book(s) | - 200 OK  - JSON array with books from the category “Horror” | Happy flow |
| Search books by filters (Price) | Database exists and has data | - Call:  https://GreatBooks.com/api/v1/books?price=10  - Verify the returned list of book(s) | - 200 OK  - JSON array with books that cost €“10” | Happy flow |
| Search books by filters (Author) | Database exists and has data | - Call:  https://GreatBooks.com/api/v1/books?author=Sam Rotthier  - Verify the returned list of book(s) | - 200 OK  - JSON array with books from the author “Sam Rotthier” | Happy flow |
| Search books by filters (combination of the previous filters) | Database exists and has data | - Call:  https://GreatBooks.com/api/v1/books?author=Sam Rotthier&category=fantasy  - Verify the returned list of book(s) | - 200 OK  - JSON array with books from the author “Sam Rotthier” and from the category “fantasy” | Happy flow |
| More combinations of the previous scenario … | Database exists and has data | … | - 200 OK  - JSON array with books from the filters … | Happy flow |
| Search Books with no filters | Database exists and has data | - Call: https://GreatBooks.com/api/v1/books?  - Verify the returned list of book(s) | - 200 OK  - JSON array with all books in the system | Happy flow |
| Invalid filters while searching for books | Database exists and has data | - Call: https://GreatBooks.com/api/v1/books?price=TestString | - 400 Bad request | Error flow |
| Unauthorized action while searching for books | Database exists and has data | - User does an action that requires a higher level of elevation | - 401 Unauthorized | Error flow |
| Forbidden action while searching for books | Database exists and has data | - User does an action that requires a higher level of elevation | - 403 Forbidden | Error flow |
| No books found with filter | Database exists and requested data is not present | - Call:  https://GreatBooks.com/api/v1/books?title=TestValueNon existent | - 404 Not found | Error flow |
| Internal Server Error while searching for books | Database exists and has data | - User does an unexpected action that creates an error  - Example call:  https://GreatBooks.com/api/v1/books?FaultFilter=TestValue | - 500  Internal Server Error | Error flow |
| Search book by ISBN | Database exists and has data | - Call:  https://GreatBooks.com/api/v1/books/9780201310061  - Verify the returned book | - 200 OK  - JSON object of book in the system | Happy flow |
| Invalid ISBN length | Database exists and has data | - Call:  https://GreatBooks.com/api/v1/books/978020131006 | - 400 Bad request | Error flow |
| Unauthorized action while searching for book by ISBN | Database exists and has data | - User does an action that requires a higher level of elevation | - 401 Unauthorized | Error flow |
| Forbidden action while searching for book by ISBN | Database exists and has data | - User does an action that requires a higher level of elevation | - 403 Forbidden | Error flow |
| No books found with ISBN code | Database exists and requested data is not present | - Call:  https://GreatBooks.com/api/v1/books/{Unknown ISBN Code} | - 404 Not found | Error flow |
| Internal Server Error while searching for book by ISBN | Database exists and has data | - User does an unexpected action that creates an error  - Example call:  https://GreatBooks.com/api/v1/books/ExampleStringValue | - 500  Internal Server Error | Error flow |
| Place an order of book(s) | Database exists and has data | - Call:  https://GreatBooks.com/api/v1/orders with a valid request body | - 200 OK  - Message with order confirmation and details | Happy flow |
| Place a flawed order of book(s) | Database exists and has data | - Call:  https://GreatBooks.com/api/v1/orders with a invalid request body | - 400 Bad request | Error flow |
| Unauthorized action while placing an order | Database exists and has data | - User does an action that requires a higher level of elevation | - 401 Unauthorized | Error flow |
| Forbidden action while placing an order | Database exists and has data | - User does an action that requires a higher level of elevation | - 403 Forbidden | Error flow |
| One or more books in order is not found | Database exists and requested data is not present | - Call:  https://GreatBooks.com/api/v1/orders with a (partially) valid or invalid request body | - 404 Not found | Error flow |
| Internal Server Error while placing an order | Database exists and has data | - User does an unexpected action that creates an error  - Example call:  https://GreatBooks.com/api/v1/orders/ExampleStringValue | - 500  Internal Server Error | Error flow |