Case Study 1: Promises and Async/Await

Scenario: Online Bookstore – Order Processing

A medium-sized **online bookstore** allows customers to place orders. The order process involves multiple asynchronous operations:

1. **Validate user login**
2. **Fetch book details** from an inventory service
3. **Check stock availability**
4. **Process payment**
5. **Generate invoice and send confirmation email**

Project Structure

```
/BookStore
  ├── auth/
  │     └── auth.js
  ├── bookstore/
  │     ├── cart.js
  │     ├── inventory.js
  │     └── payment.js
  ├── utils/
  │     └── logger.js
  ├── recommendations/
  │     └── recommend.js
  └── app.js
```

Workflow & Description

1. **Step 1 – User Login Validation**
    - The system receives login credentials and verifies them against a database asynchronously.
    - If login fails → the process stops immediately.
2. **Step 2 – Fetch Book Details**
    - After successful login, the system queries the book catalog service asynchronously to fetch book info based on the user's cart.
    - Uses a **Promise** to handle the async fetch.

3. **Step 3 – Stock Availability Check**
    - Each book in the cart is checked for stock availability asynchronously.
    - If any book is out of stock → an error is thrown and the user is notified.
4. **Step 4 – Payment Processing**
    - Payment is handled via an external API (async operation).
    - Any payment failure triggers proper error handling to rollback order.
5. **Step 5 – Invoice Generation & Confirmation**
    - On successful payment, the system generates an invoice and sends an email confirmation asynchronously.
6. **Error Handling**
    - Every async step has proper error handling:
        - `catch()` in promises
        - `try/catch` for async/await
    - Errors are logged and meaningful messages are returned to the user.

Key Concepts Demonstrated

- Chaining multiple Promises sequentially
- Handling errors at any stage of the chain
- Using `async/await` to write readable, sequential async code
- Ensuring process stops or rolls back on failure

Sample Output (Console / Logs)

```
✔ User login successful: User ID 101
✔ Book details fetched: ["JavaScript Basics", "Node.js
Guide"]
✔ Stock checked: All items available
✔ Payment successful: Transaction ID 56892
✔ Invoice generated: Invoice #INV1023
✔ Confirmation email sent to: user@example.com
```

Error Scenario Output (Out of Stock)

✔ User login successful: User ID 101
✔ Book details fetched: ["JavaScript Basics", "Node.js Guide"]
❌ Stock check failed: "Node.js Guide" is out of stock
Order processing stopped