

Test Project Session 3

IT Software Solution for Business

Independent Test Project Designer: Ramin Mohammaddoust

Independent Test Project Validator: Afshin Dehghani

Introduction

In this session, you will take on the role of a full-stack developer, responsible for building the front-end user interface (UI) and the back-end API for the bakery's new software system.

The focus of this session is to translate the design and requirements from Session 2 into a functional desktop application for staff and a robust API to manage product and order data. You will be evaluated on your ability to create intuitive and visually appealing user interfaces, as well as your proficiency in developing secure and efficient APIs.

This session is designed to assess your skills in:

- **Front-End Development:** Designing and implementing user interfaces that are user-friendly, visually appealing, and aligned with Belle Croissant Lyonnais' branding.
- **Back-End Development:** Building a RESTful API that adheres to industry best practices, handles data interactions securely, and provides a seamless experience for the front-end application.
- **Integration:** Ensuring smooth communication and data exchange between the front-end and back-end components.
- **Problem-Solving:** Identifying and resolving technical challenges that arise during development.

Contents

This session package includes the following materials:

1. **Session Instructions (PDF):** Detailed instructions outlining the tasks to be completed and deliverables expected for this session.
2. **Common Folder:** This folder contains additional resources such as the Belle Croissant Lyonnais logo, icons, style guide, and other design assets that can be used throughout the development of the application.
3. **Database Schema (SQL):** A SQL script containing the structure for the tables Promotions and LoyaltyProgram, which you will use in this session.

Description of Project and Tasks

In this session, you will create the foundation for the Belle Croissant Lyonnais desktop application.

Guidelines:

1. **Easy to Use:** Make the interface simple and easy for staff to understand.
2. **Looks Good:** Follow the Belle Croissant Lyonnais Style Guide for the design.
3. **Works Well:** Check that all parts of the application work correctly and without errors.
4. **Secure:** Protect customer data and ensure the application is safe to use.
5. **On Time:** Finish all tasks within the time limit.

Technical Considerations:

1. **Database Setup:** Create and populate the database according to the provided schema.
2. **API Development:** Implement a RESTful API that follows best practices and includes authentication.
3. **User Interface:** Develop intuitive screens for product and order management.
4. **Data Validation:** Ensure user input is correct and complete for all operations.
5. **Error Handling:** Display clear messages to the user if there are any problems.

Additional Considerations:

- The application should work smoothly and quickly on the provided development environment.
- Use clear labels and instructions for all UI elements.
- Organize information in a way that is easy for staff to understand.
- Consider edge cases and potential errors in user input and data handling.
- Implement comprehensive unit tests to ensure API functionality.

Instructions to the Competitor

3.1 Database Setup and Data Import

Objective:

Create and populate the Belle Croissant Lyonnais database according to the provided schema, ensuring data accuracy and integrity.

Tasks:

1. **Database Creation:**
 - Create a database named BelleCroissantLyonnais (or a similar, relevant name).
2. **Schema Execution:**
 - Execute the provided SQL script (Database_Schema.sql) to create the following tables and their relationships:
 - Products
 - Customers
 - Orders
 - OrderItems
3. **Data Import:**
 - Import the data from the cleaned CSV files (products_cleaned.csv, customers_cleaned.csv, and sales_transactions_cleaned.csv) into the corresponding database tables. Ensure the data types match the schema definition.

Deliverables:

- **Database Credentials:** Provide the connection string or credentials required to access your database (server name, database name, username, password). You can store this information in a simple text file named Session3_DatabaseCredentials.txt.

Additional Notes:

- Pay attention to the data types in the CSV files and ensure they match the data types defined in the schema.
- Use appropriate import tools or techniques to efficiently transfer the data into the database.
- Double-check the data after import to ensure its accuracy and completeness.
- The evaluation will focus on the correctness and completeness of the database setup and data import process.

3.2 Backend API Development

Objective:

Develop a secure and efficient RESTful API using .NET Web API that enables the Belle Croissant Lyonnais desktop application to interact with the database.

Tasks:

1. **API Endpoints:** Make the following ways for the app to talk to the database:
 - **Products:**
 - GET /api/products: Get all products
 - GET /api/products/{id}: Get one product by ID
 - POST /api/products: Add new product
 - PUT /api/products/{id}: Change product by ID
 - DELETE /api/products/{id}: Remove product by ID
 - **Customers:**
 - GET /api/customers: Get all customers
 - GET /api/customers/{id}: Get one customer by ID
 - POST /api/customers: Add new customer
 - PUT /api/customers/{id}: Change customer by ID
 - **Orders:**
 - GET /api/orders: Get all orders
 - GET /api/orders/{id}: Get one order by ID
 - POST /api/orders: Add new order
 - PUT /api/orders/{id}/complete: Finish order by ID
 - PUT /api/orders/{id}/cancel: Cancel order by ID

2. Data Check:

- Make sure the app sends the right kind of data to the database (numbers, letters, dates, etc.).
- Clean up data to make sure it's safe to use.

3. Error Handling:

- Send the right messages back to the app if there's a problem.

4. Authentication:

- Make sure only the right people can use the API. Use a password system (Basic Authentication) where the username is "staff" and password is "BCLyon2024".

Deliverables:

- **Running API:** Make the API work in the given place.
- **Session3_API_Endpoints.txt:** A file listing what the API can do, how to ask for things, and what to expect back (in JSON).

3.3 UI Development - Product Management

Objective:

Develop a user-friendly desktop interface for managing products in the Belle Croissant Lyonnais application, utilizing the API endpoints created in Task 3.2.

Tasks:

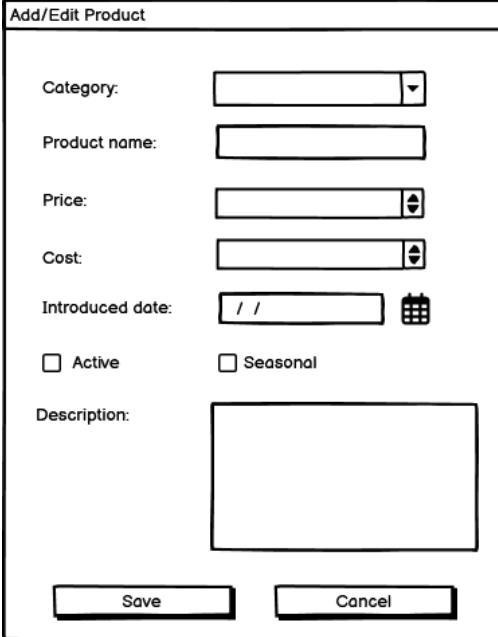
1. Product Listing:

Product management					
<input type="text" value="search on products/categories"/>					<input type="button" value="Add new product"/>
Active	ProductName	Category	Price	Cost	Action
<input type="checkbox"/>	Baguette Tradition	Bread	6.17	5.2	edit delete
<input checked="" type="checkbox"/>	Croissant	Tarte	4.32	3.3	edit delete
<input checked="" type="checkbox"/>	Eclair au Chocolat	Pastries	2.83	2.0	edit delete
<input checked="" type="checkbox"/>	Pain au Chocolat	Pastries	5.53	4.6	edit delete

- Display all products fetched from the API endpoint GET /api/products in a table.

- Table columns: ProductName, Category, Price, Cost, Active (Yes/No).
- Add sorting (ascending/descending) for each column.
- Add a search bar to filter products by name or category.

2. Add/Edit Product Form:



The form is titled "Add/Edit Product". It contains the following fields and controls:

- Category:** A dropdown menu.
- Product name:** A text input field.
- Price:** A numeric input field with a spinner.
- Cost:** A numeric input field with a spinner.
- Introduced date:** A date picker with a calendar icon.
- Active:** A checkbox.
- Seasonal:** A checkbox.
- Description:** A large text area.
- Buttons:** "Save" and "Cancel" buttons at the bottom.

- Create a form for adding new products or editing existing ones.
- Form fields:
 - ProductName (text input, required, max 100 characters)
 - Category (dropdown list, required, values from products_cleaned.csv)
 - Price (numeric input, required, must be a positive number)
 - Cost (numeric input, required, must be a positive number and less than Price)
 - Description (text area, optional)
 - Seasonal (checkbox)
 - Active (checkbox)
 - IntroducedDate (date picker, required)
- Form should have "Save" and "Cancel" buttons.
- Validate input data.
- Use the API to:
 - POST /api/products to save new products.
 - PUT /api/products/{id} to update existing products.
 - Handle API responses and display appropriate messages (success/error) to the user.

3. Delete Product Functionality:

- Add a "Delete" button next to each product in the listing.
- Show a confirmation dialog before deleting.
- Use the API (DELETE /api/products/{id}) to delete the product.
- Handle API response and update the product list accordingly.

Deliverables:

- Incorporate the UI into the existing Session3_DesktopApp.exe file.

Additional Notes:

- Follow best practices for UI design and development.
- The UI should be responsive and adapt to different screen sizes.
- Consider error handling and user feedback in the UI.

3.4 UI Development - Order Management

Objective:

Develop a user-friendly desktop interface for managing customer orders within the Belle Croissant Lyonnais application, utilizing the API endpoints created in Task 3.2.

Tasks:

1. Order Listing:

Order listing					
<input type="text" value="search on ID/Customer name/Date"/>				<input type="button" value="Add new product"/>	
Order ID	Customer Name	Date	Total Amount	Status	Order Detail
802	Manon Dupont	4/11/2024	6.17	pending	Detail View
34	Jean Roux	10/06/2023	8.52	Completed	Detail View
33	Manon Martin	10/06/2023	2.83	Completed	Detail View
14	Jean Petit	Pastries	9/06/2023	Cancelled	Detail View

- Display all orders from the database in a table.

- Table columns: Order ID, Customer Name, Date, Total Amount, Status (Pending, Processing, Completed, Cancelled).
- Allow sorting (ascending/descending) by any column.
- Add a search bar to filter orders by ID, customer name, or date.

2. Order Details View:

Order details view

Order ID: 802
Customer Name: Manon Dupont
Order Date: 4/11/2024
Total Amount: 6.17
Order Status: Pending
List of items:

Items	Quantity	Price

Save
Cancel

- Create a separate window or section to display the details of a selected order.
- Display the following information:
 - Order ID
 - Customer Name
 - Order Date and Time
 - Total Amount
 - Order Status
 - List of ordered items with quantity and price

3. Update Order Status:

- Add buttons or a dropdown menu to update the order status.
- Available statuses: "Processing," "Completed," "Cancelled."
- Use the API to update the order status in the database.

Deliverables:

- Incorporate the UI into the existing Session3_DesktopApp.exe file.

Additional Notes:

- Use your preferred .NET UI framework.
- The UI should be responsive and adapt to different screen sizes.
- Error handling and user feedback are essential (e.g., display messages when an order is successfully updated or if an error occurs).

3.5 Acceptance and Black Box Testing

Objective

Develop acceptance tests to verify the API's functionality and ensure it meets the specified requirements. Conduct black box testing to validate the API's behavior without knowledge of the internal workings.

Tasks

1. **Setup Testing Environment:**
 - Ensure that the Flask application is running locally.
 - Use the provided test_backend.py script to run tests.
2. **Acceptance Tests:**
 - **Product Endpoints:**
 - Verify that GET /api/products returns a list of all products.
 - Verify that GET /api/products/{id} returns the correct product details.
 - Verify that POST /api/products creates a new product with valid data.
 - Verify that PUT /api/products/{id} updates an existing product.
 - Verify that DELETE /api/products/{id} deletes the specified product.
 - **Order Endpoints:**
 - Verify that GET /api/orders returns a list of all orders.
 - Verify that GET /api/orders/{id} returns the correct order details.
 - Verify that POST /api/orders creates a new order with valid data.
 - Verify that PUT /api/orders/{id}/complete marks an order as completed.
 - Verify that PUT /api/orders/{id}/cancel cancels an order and restores stock.
3. **Black Box Testing:**
 - Test the API without knowledge of the internal code structure.
 - Focus on input-output validation and error handling.
 - Use various data inputs to test edge cases and invalid scenarios.

Deliverables

- **File Name:** Session3_AcceptanceTests.zip
- **File Type:** Compressed archive (.zip)

- **Contents:**
 - All the test files (.cs files for .NET) for acceptance and black box test solution.
 - Any necessary configuration files.

Additional Notes

- Use mocking or stubbing techniques to simulate external dependencies.
- Ensure tests run in isolation and do not depend on external resources.
- Refer to the "Web API Design Best Practices" document for guidance on writing effective tests.

3.6 Unit and White Box Testing

Objective:

The objective of this module is to develop and test a bakery inventory management system using unit and white box testing techniques. Competitors will implement classes to manage bakery items and inventory, design comprehensive test cases, and ensure code quality and functionality through rigorous testing.

Tasks

1. **Implement the BakeryItem Class:**
 - Create a class named BakeryItem with properties for Name, Price, Quantity, and ExpirationDate.
 - Implement a method IsExpired() to determine if the item is expired.
2. **Implement the BakeryInventory Class:**
 - Create a class named BakeryInventory to manage a collection of BakeryItem objects.
 - Implement methods to:
 - AddItem(BakeryItem item): Add a new item to the inventory.
 - RemoveItems(string name): Remove an item or items from the inventory by name.
 - GetTotalValue(): Calculate the total value (Price * Quantity) of all items in the inventory.
3. **Write Unit Tests:**
 - Use NUnit to write unit tests for both classes. Verify the following:
 - The BakeryItem properties are correctly set and retrieved.
 - The IsExpired method accurately determines if an item is expired.
 - The BakeryInventory methods correctly add, remove, and calculate the total value of items.
 - Handle edge cases, such as attempting to remove an item that does not exist.
4. **Conduct White Box Testing:**
 - Analyze the code to identify critical paths and potential areas of risk.

- Ensure that all possible execution paths are covered by the test cases.

Deliverables

- **File Name:** BakeryInventoryTesting.zip
- **File Type:** Compressed archive (.zip)
- **Contents:**
 - C# class files for BakeryItem and BakeryInventory.
 - Testing project using NUnit for testing both classes.

Additional Notes

- Ensure your code is well-documented with comments explaining the logic.
- Use best practices for naming conventions and code organization.
- Ensure all unit tests pass before submission.