Comprehensive Exercise Report

Team X of Section XXX

Team members : Alexandre LOGUT 250AEB021, Alec MARCHAL 250ADB073, Clément CROUAN 250AEB001

Requirements/Analysis

Week 2

Journal

The following prompts are meant to aid your thought process as you complete the requirements/analysis portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- The project is to develop a web-based application called "Bakery Manager" to help bakeries in mid-sized French cities efficiently manage their inventory by tracking stock levels, streamlining the ordering process, and improving daily operations for bakery staff.
- **Known requirements from client description:**
    - The system must allow for real-time monitoring of raw materials and finished goods.
    - It should provide smart restocking alerts and supplier order generation.
    - The system aims to reduce waste through accurate stock tracking.
    - It should improve daily operations and organization for bakery staff.
    - The primary users are bakers in mid-sized French cities.
    - The application will be web-based.
    - The team had to rely on advice of friends and acquaintances in a profession where this project could be useful, as a customer couldn't be found.
    - A key identified need was the ability to access stock information that is present and missing from anywhere.
    - The initial consideration was between a smartphone application and a website.
    - The decision was made to develop a website linked to a server so that several people could access it without having data synchronization problems, the team also noted they have a lot more knowledge in website development.
    - Security for accessing data was identified as a necessary addition.

- **Question 1:** Are there any technology preferences or constraints regarding frameworks, databases, or hosting?
    - **Answer:** We prefer a solution using React for the front-end and Node.js for the back-end, with a PostgreSQL database. Hosting will be on a cloud server.
- **Question 2:** How many simultaneous users should the system support, and what are the different access permission levels?
    - **Answer:** We expect around 20 simultaneous users. There will be 3 access levels: bakers (production staff), managers, and suppliers, each with specific permissions.
- **Question 3:** Should the application be available only in French, or should it support multiple languages?
    - **Answer:** For now, only French is required, but a multilingual version might be considered in the future.

Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.

- This section requires input from the team regarding their familiarity with the chosen technologies (React, Node.js, MySQL) and concepts. This information has not been fully provided, though the presentation mentions challenges with database integration and request handling, suggesting learning occurred in these areas.

Describe the users of this software (small child, high school teacher who is taking attendance).

- The users of this software are:
    - Bakers (production staff) in mid-sized French cities.
    - Managers of bakeries.
    - Suppliers to the bakeries.

Describe how each user would interact with the software

- **Bakers (production staff) :**
    - Log in to the system.
    - View current stock levels of raw materials and finished goods.
    - Update stock quantities after production or usage.
    - Potentially receive notifications for low stock items relevant to their work.

    - Add new products they have produced to the inventory (via an "Add a Product" form including details like product name, category, price, ingredients, quantity, production date, expiration date).

- **Managers :**
    - Log in to the system with higher privileges.
    - Oversee all inventory data (view, add, update, delete products).
    - Manage user accounts for bakers and possibly suppliers.
    - Generate reports on stock levels, waste, and orders.
    - Configure smart restocking alerts.
    - Manage supplier information and generate orders.
    - Monitor overall bakery operations through the system.
    - Delete products from the inventory using product ID or name.
- **Suppliers :**
    - Log in to the system with restricted access.
    - View orders placed by the bakery.
    - Update the status of orders.
    - Potentially manage their product catalog available to the bakery.

What features must the software have? What should the users be able to do?

- **User Authentication and Authorization:**
    - Secure user login (username and password).
    - "Create an account" functionality.
    - "Forgot password" functionality.
    - Role-based access control for three distinct user types: bakers, managers, and suppliers, each with specific permissions.

- **Inventory Management :**
    - Real-time monitoring of raw materials and finished goods.
    - Ability to add new products with details: Product Name, Category, Price (€), Ingredients, Quantity, Production Date, Expiration Date.

    - Ability to update existing stock information (quantity, price).
    - Ability to view inventory in a structured format (table with ID, Product name, Category, Price (€), Ingredients, Quantity, Production Date, Expiration Date).
    - Ability to delete products from the inventory (using ID or product name).
- **Ordering System :**
    - Smart restocking alerts for low stock levels.
    - Generation of supplier orders.
- **Operational Efficiency:**
    - Mechanisms for waste reduction through accurate tracking.
    - Tools to improve daily operations and organization for bakery staff.
- **System Administration (primarily for Managers):**
    - User management.
    - System configuration (alert thresholds).
- **Language :**
    - The application interface must be in French.
- **Technology Stack :**
    - Front-end: React.
    - Back-end: Node.js.
    - Database : MySQL.
    - Hosting: Cloud server.
- **Accessibility :**
    - Accessible from anywhere (web-based).
- **Feedback Mechanism :**
    - Allow users to provide their name, a message, and rate features.

Other notes :

- The team initially considered a smartphone app but pivoted to a website due to better familiarity with web development and to simplify multi-user server-linked access.

- Security for data access is a key consideration.
- The team chose the Agile model for this project due to its flexibility for evolving ideas and unexpected changes.
  - Its short iterations mean constant feedback and collaboration, keeping everyone on the same page.
  - Agile allows for shared responsibility and focus on what's most important.

Software Requirements

Project Overview: Bakery Manager is a web-based application designed to provide mid-sized French bakeries with a comprehensive solution for efficient inventory management. The system will enable real-time tracking of raw materials and finished goods, facilitate smart restocking through automated alerts and supplier order generation, and ultimately help reduce waste and improve daily operational organization for bakery staff. The application will be developed using React for the front-end, Node.js for the back-end, and a MySQL database, hosted on a cloud server. It will feature role-based access for bakers, managers, and suppliers, ensuring secure data handling and tailored functionality for each user type. The primary language for the application will be French. The project prioritizes security, efficiency, and usability.

Functional Requirements:

- **FR1: User Authentication**
    - **FR1.1:** The system shall provide a secure login mechanism for registered users using a username and password.
    - **FR1.2:** The system shall allow new users to create an account.
    - **FR1.3:** The system shall provide a "forgot password" feature for users to reset their password.
- **FR2: Role-Based Access Control**
    - **FR2.1:** The system shall support three user roles: Baker, Manager, and Supplier.
    - **FR2.2:** Users assigned the Baker role shall be able to view and update stock levels and add new products they produced.
    - **FR2.3:** Users assigned the Manager role shall have full access to inventory management (add, view, update, delete products), user management, order generation, and system configuration.
    - **FR2.4:** Users assigned the Supplier role shall be able to view orders placed by the bakery and update order statuses.
- **FR3: Inventory Tracking**
    - **FR3.1:** The system shall allow users with appropriate permissions to add new products to the inventory. Required information includes Product Name, Category, Price, Ingredients, Quantity, Production Date, and Expiration Date.

- **FR3.2:** The system shall display the current inventory, including details such as ID, Product name, Category, Price (€), Ingredients, Quantity, Production Date, and Expiration Date.
- **FR3.3:** The system shall allow authorized users to update the details of existing products in the inventory, such as quantity.
- **FR3.4:** The system shall allow authorized users to delete products from the inventory, using either the product ID or name for selection.
- **FR3.5:** The system shall track stock levels of raw materials and finished goods in real-time.

- **FR4: Restocking and Ordering**
  - **FR4.1:** The system shall generate smart alerts for items whose stock levels fall below predefined thresholds.
  - **FR4.2:** The system shall allow Managers to generate purchase orders for suppliers.
- **FR5: Language and Localization**
  - **FR5.1:** The application interface shall be exclusively in French.
- **FR6: Feedback System**
  - **FR6.1:** The system shall allow users to submit feedback including their name, a message, and ratings for application features.

Non-Functional Requirements:

- **NFR1: Security:** The system must ensure secure access to data and smooth authentication.
- **NFR2: Usability:** The system should be intuitive, easy to use for all defined user roles, and have a user-friendly design for intuitive navigation. The interface should be visually appealing.
- **NFR3: Performance:** The system should support approximately 20 simultaneous users without significant degradation in performance. Real-time stock updates should be reflected promptly.
- **NFR4: Technology:**
  - **NFR4.1:** The front-end shall be developed using JavaScript.
  - **NFR4.2:** The back-end shall be developed using Node.js.
  - **NFR4.3:** The database shall be MySQL.

    - **NFR4.4:** The application shall be hosted on a cloud server.
- **NFR5: Accessibility:** The web application should be accessible from common web browsers. The team aimed to enhance accessibility.
- **NFR6: Data Management:** The system must ensure adequate database storage and allow for easy data access and modification.

User Stories (Examples) :

- **As a Baker,** I want to quickly log in to the system so I can update the inventory with the bread I just baked.
- **As a Baker,** I want to see the current quantity of flour so I know if we have enough for today's production.
- **As a Manager,** I want to add new products to our offerings, including their price and ingredients, so they are tracked in the system.
- **As a Manager,** I want to view a list of all products in the inventory with their current stock levels, production, and expiration dates, so I can monitor our stock.
- **As a Manager,** I want to be able to delete a discontinued product from the inventory so it doesn't clutter our records.
- **As a Manager,** I want to receive an alert when we are running low on a critical ingredient so I can reorder it in time.
- **As a Manager,** I want to generate a purchase order for a supplier so I can restock items.
- **As a Manager,** I want to define different access levels for my staff (bakers) and suppliers so that they only see and do what's relevant to their role.
- **As a Supplier,** I want to log in and see new orders from the bakery so I can process them.
- **As a User,** I want to provide feedback on the application features so the developers can improve them.

Design

Implementation

Instructions: Week 8

Journal

- **Front-End Development :**
  - Component-based architecture: To build reusable UI elements for features like login forms, product addition forms, inventory displays, and feedback forms.
  - State management: To handle data within components and across the application (form inputs, inventory lists).
  - Props: To pass data between components.
  - Hooks: (useState, useEffect) For managing component state and side effects like fetching data.
  - Routing: To navigate between different views of the application (login, inventory, add product).
- **Back-End Development (Node.js):**
  - Server-side logic: To handle incoming HTTP requests from the client.
  - API development (RESTful principles): To define endpoints for operations like user authentication, product management (add, update, delete, view), and order processing.
  - Middleware: For tasks like request parsing, authentication, and error handling.
  - Database interaction: To connect to the MySQL database and perform CRUD (Create, Read, Update, Delete) operations.
- **Database Management (MySQL) :**
  - Relational database design: To structure data in tables (Users, Products, Orders, Categories, Ingredients).
  - SQL queries: To insert, retrieve, modify, and delete data.
  - Data integrity: Ensuring data consistency and relationships.
- **HTTP Requests :**

    - Client-server communication: The front-end will use HTTP requests (GET, POST, PUT, DELETE) to interact with the back-end API for data exchange.
- **Version Control (Git/GitHub) :**
    - Collaboration: To manage codebase changes, allow multiple developers to work concurrently, and track project history.
    - Branching: To work on different features in isolation before merging.
- **Package Management (npm) :**
    - Dependency management: To manage project libraries and dependencies for both front-end and back-end development.
    - Script automation: To run development servers, build processes, and tests.
- **Styling (CSS) :**
    - User Interface Styling: To apply custom styles and layouts for a visually appealing and user-friendly interface.
- **Agile Methodology & Collaboration :**
    - Iterative Development: Building the project in short iterations with constant feedback.
    - Team Collaboration: Regular meetings and coding sessions at the university to ensure synchronization, brainstorming, and continuous improvement.

Other notes :

- The team focused on smooth data insertion, modification, and retrieval for efficient stock management.
- Collaboration was facilitated through GitHub for version control and feature development. Regular meetings and coding sessions at the university helped ensure synchronization, brainstorming, and continuous improvement.

Implementation Details

README.md

Markdown

~~~
# Bakery Manager Application

## Overview

Bakery Manager is a web-based application designed to help bakeries efficiently
manage their inventory by tracking stock levels and streamlining the ordering process. It
features user authentication, inventory management (add, update, view, delete
products), and a feedback mechanism.

## Technologies Used

* **Front-End:** JavaScript

* **Back-End:** Node.js

* **Database:** MySQL

* **Package Manager:** npm

* **Version Control:** Git & GitHub

## Prerequisites

* Node.js (v14.0.0 or higher) and npm (v6.0.0 or higher) installed.

* MySQL installed and running.

* Git installed on your system.

## Setup and Run Instructions

1.  **Clone the repository:**

   ```bash

   git clone [https://github.com/Giougt/DIP392-Data-Management-
shop](https://github.com/Giougt/DIP392-Data-Management-shop)
~~~

cd DIP392-Data-Management-shop

2. **Install Dependencies:**

* **Back-End:** Navigate to the back-end directory and run:

```bash
npm install
```

* **Front-End:** Navigate to the front-end directory and run:

```bash
npm install
```

3. **Database Setup:**

* Ensure your MySQL server is running.

* Create a database named appropriately (`bakery_db`).

* Configure the database connection details in the back-end (in a `.env` file or configuration file), including host, user, password, and database name.

* Run any necessary database migrations or seed scripts if provided.

4. **Running the Application:**

* **Back-End:** Navigate to the back-end directory and run:

```bash
npm start
```

* **Front-End:** Navigate to the front-end directory and run:

```bash
npm start
```

The application should now be accessible in your web browser (typically at `http://localhost:3000` for the front-end).

## How to Interact with the System

### 1. Login/Registration

* **Login:** Upon visiting the application, you will be presented with a "Connexion" page. Enter your existing **Username** and **Password**, then click **Login**.

* **Create an Account:** If you don't have an account, click on the "**Create an account**" link on the login page. Fill in the required details and submit to register.

* **Forgot Password:** If you've forgotten your password, click the "**Forgot password ?**" link on the login page and follow the instructions.

### 2. Main Dashboard / Navigation

* After logging in, you will be directed to the main interface.

* A navigation bar at the top allows you to access different sections:

  * **Add Product:** To add new items to the inventory.

  * **Update Stock:** To modify details of existing products.

  * **Inventory:** To view the list of all products.

  * **Delete product:** Functionality to remove products.

  * **Feedback:** To provide feedback about the application features.

  * **LOGOUT:** To securely log out of the system.

### 3. Managing Inventory

* **Add a Product:**

  * Navigate to the "Add Product" section.

  * Fill in the form with details such as Product Name, Category (select from dropdown), Price (€), Ingredients, Quantity, Production Date (jj/mm/aaaa), and Expiration Date (jj/mm/aaaa).

  * Click the submit button to add the product to the inventory.

* **View Inventory:**

  * Navigate to the "Inventory" section.

   * Products will be listed in a table format showing ID, Product name, Category, Price (€), Ingredients, Quantity, Production Date, and Expiration Date.

* **Update Stock:**

   * Navigate to the "Update Stock" section.

   * Select the product to update using its ID or Name.

   * Enter the "New Quantity" or other fields you wish to modify.

   * Click "Update Product" to save changes.

* **Delete Product:**

   * This functionality may be present in the "Inventory" view or via a dedicated "Delete product" interface where you can select a product by ID or name to remove it.

### 4. Providing Feedback

* Navigate to the "Feedback" section.

* Enter "Your Name" and "Your Message".

* Rate different features of the application (Add a Product, Update Stock, Inventory, Delete Product) using the dropdown menus.

* Click "Send" to submit your feedback.

Testing

Instructions: Week 10

Journal

Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.

- This question cannot be fully answered as the black-box test plan (Week 4) was skipped. However, based on the presentation, a "Feedback" functionality was implemented, which might have been an addition or refinement to the initial requirements. The project followed an Agile methodology, which allows for iterative development and evolving requirements.

Conceptual Components/Modules and Tests:

- **User Authentication Module (Front-end & Back-end)**
  - **Equivalence Classes :**
    - Valid username/password, invalid username/password, empty username/password.
    - User roles (Baker, Manager, Supplier) for authorization.
  - **Boundary Values :**
    - Password length (min/max if defined).
    - Username length/format if defined.
  - **Paths:** Successful login, failed login (wrong credentials, user not found), successful registration, registration with existing username/email, password reset flow.
- **Product Management Module (Front-end & Back-end)**
  - **Classes/Components (Conceptual) :** ProductForm (Add/Edit), ProductList/InventoryTable, ProductController/Service.
  - **Equivalence Classes (for "Add Product" form fields):**
    - Product Name: Non-empty string, empty string.
    - Category: Valid selection, no selection.
    - Price (€) : Positive number, zero, negative number, non-numeric input.

        - Ingredients: Text, empty text.
        - Quantity: Positive integer, zero, negative integer, non-integer.
        - Production/Expiration Date : Valid date format, invalid date format, expiration date before production date.
    - **Boundary Values :**
        - Price/Quantity : Min/max allowable values (quantity > 0).
        - Date : Realistic date ranges.
    - **Paths:** Adding a new product successfully, adding with missing required fields, adding with invalid data types, viewing inventory (empty, with items), updating product quantity successfully, updating with invalid quantity, deleting a product (existing, non-existing), searching/filtering products.
- **Feedback Module (Front-end & Back-end)**
    - **Equivalence Classes :**
        - Name : Non-empty, empty.
        - Message : Non-empty, empty.
        - Ratings: Valid selection for each feature, no selection.
    - **Boundary Values:** Length of message if constraints exist.
    - **Paths:** Submitting feedback successfully, submitting with missing fields.
- **HTTP Request Handling (Back-end Controllers/Routes)**
    - **Equivalence Classes:** Valid POST/GET/DELETE requests, malformed requests, requests with missing data, requests to invalid endpoints.
    - **Boundary Values:** Large data payloads (if applicable).
    - **Paths:** Successful data retrieval (GET), successful data creation (POST), successful data deletion (DELETE), handling of server-side errors (database errors), security checks (authentication tokens).

Other notes :

- The team implemented HTTP request testing to validate the functionality of POST, GET, and DELETE operations.
- Automated and manual tests ensured smooth data handling, security, and user experience consistency.

- Key scenarios tested included new user account creation, normal site usage, and bug detection.
- The tests simulated real interactions, verifying database updates, proper request handling, and potential system errors.
- The project was continuously tested in collaboration, ensuring that each feature met user needs efficiently.
- Regular evaluations, feedback loops, and team discussions allowed for quick improvements, refining the design and functionality throughout development.

Testing Details

Testing Strategy Overview: The testing strategy employed a combination of automated and manual tests to ensure the reliability, functionality, and usability of the Bakery Management system. The primary focus was on validating HTTP requests (POST, GET, DELETE), database operations, user authentication, inventory management workflows, and overall user experience. The Agile methodology facilitated continuous testing and feedback incorporation throughout the development cycle.

Types of Tests Conducted :

1. **HTTP Request Testing (Back-End API Testing):**
    - **Description:** Tests were designed to directly interact with the Node.js API endpoints to validate the core logic for data manipulation and retrieval. This involved testing POST, GET, and DELETE operations.
    - **Tools (Hypothetical as not specified):** Postman, Jest with Supertest, or similar API testing tools.
    - **Specific Tests (Conceptual based on functionalities):**
        - **POST /api/products:** Verify successful creation of a new product with valid data. Test with invalid data (missing fields, incorrect data types) to ensure proper error responses.
        - **GET /api/products:** Verify successful retrieval of all products. Test when no products exist.
        - **GET /api/products/:id:** Verify successful retrieval of a specific product. Test with a non-existent product ID.
        - **PUT /api/products/:id :** Verify successful update of an existing product's details (stock quantity). Test with invalid data or non-existent ID.

        - **DELETE /api/products/:id:** Verify successful deletion of a product. Test deletion of a non-existent product.
        - **POST /api/auth/login:** Verify successful user login with correct credentials and token generation. Test with incorrect credentials.
        - **POST /api/auth/register:** Verify successful new user registration. Test registration with an existing username/email.
        - **POST /api/feedback:** Verify successful submission of feedback. Test with incomplete feedback data.

2. **Manual Testing (User Interface and Experience):**
    - **Description:** Manual walkthroughs of the application by team members to simulate real user interactions and identify usability issues, visual bugs, and workflow inconsistencies.
    - **Key Scenarios Tested :**
        - New user account creation and login process.
        - Adding a new product to the inventory, including filling all fields on the "Add a Product" form.
        - Viewing the inventory list and verifying data accuracy.
        - Updating the stock quantity of an existing product.
        - Deleting a product from the inventory.
        - Submitting feedback via the "Leave Feedback" form.
        - Navigating between different sections of the application.
        - Normal site usage.
        - Bug detection.
3. **Database Interaction Testing :**
    - **Description:** Ensuring that data was correctly persisted, updated, and retrieved from the MySQL database without loss or corruption. This was often verified as part of HTTP request testing and manual testing. Testing confirmed smooth database operations.
    - **Checks :**
        - Data integrity after product creation, update, and deletion.
        - Correct storage of user credentials.

        - No data loss was detected.

Test Results Summary :

- The database performed well during testing, handling requests smoothly with no lost data.
- No error messages were detected from the script, and controllers functioned correctly, ensuring reliable operations.
- The tests confirmed that the software meets the objectives. The system efficiently manages inventory and orders, providing a stable and user-friendly experience that aligns with customer expectations.
- Testing confirmed smooth database operations, effective request handling, and a user-friendly interface.
- No data loss or system errors were detected, ensuring reliability and efficiency.
- Debugging scripts and optimizing controllers were crucial to prevent data loss and improve system reliability.

Presentation

Instructions: Week 12

Preparation

- The final project, "Data-Management-shop" (Bakery Manager), is a web application designed to help bakeries efficiently manage their inventory. It allows for tracking stock levels of raw materials and finished goods, and streamlining the ordering process. The system aims to reduce waste through accurate stock monitoring and ensure smooth operations. It features user authentication, product management (add, update, view, delete), and a feedback mechanism, built using Node.js, and MySQL. The users are bakers in mid-sized French cities who need an efficient tool to manage their bakery inventory and orders.

- **Assumptions :**
  - Users (bakers, managers, suppliers) would have basic computer literacy to navigate a web application.
  - The primary need was for a system accessible from anywhere, leading to the web-based solution over a standalone app.
  - Initial requirements were gathered from advice from friends and acquaintances due to the unavailability of a direct client.
- **Additions/Refinements during development :**
  - The "Feedback" functionality was implemented, allowing users to rate specific features and provide textual feedback.
  - Specific user roles (baker, manager, supplier) with distinct permissions were defined based on instructor feedback.
  - Security for data access was identified as a necessary addition by the team and prioritized.
  - The choice of Agile methodology allowed for evolving ideas and iterative refinement of requirements.
  - Functional requirements included a login system for account creation and access control.
  - Non-functional requirements focused on adequate database storage, easy data access and modification, and a user-friendly design. These shaped the project by prioritizing security, efficiency, and usability.

- **Initial Architectural Consideration:** The team initially considered two main ideas: a smartphone application and a website.
  - **Smartphone Application :**
    - Pros: Potentially better for on-the-go access.
    - Cons: Concerns about data synchronization for multiple users, requiring a server link which added complexity.
  - **Website Application :**
    - Pros: Easier to manage multi-user access when linked to a server, the team had more knowledge in website development. Inherently accessible from various devices with a browser.
    - Cons : Might require a constant internet connection.
- **Architectural Decision:** The team opted for a website application, as it better facilitated multi-user access via a server and aligned with their existing knowledge base.
- **UI/UX Design :**
  - Since the client had no specific design preferences, multiple design options were created and discussed within the team.
  - The final choice was refined through collaboration to ensure clarity and usability. Various designs were explored before settling on the best fit.
  - Colors were carefully selected with the UI designer (Alec Marchal) to enhance user experience and ensure a visually appealing interface.
  - The team ensured a well-structured database, smooth authentication, and a redesigned interface to enhance accessibility and user experience.

How did the extension affect your design?

- Information regarding a specific "extension" and its effect on the design was not explicitly provided in the documents. The project has plans for future enhancements such as adapted visual styles for accessibility, improved database control on the user side, and a mobile-responsive version, which would influence future design iterations.
- The team implemented a testing strategy that included HTTP request testing for validating the functionality of POST, GET, and DELETE operations.
- Automated and manual tests were conducted to ensure smooth data handling, security, and user experience consistency.

- **Key scenarios tested :**
    - New user account creation and login.
    - Adding, viewing, updating, and deleting products from the inventory.
    - Normal site usage.
    - Bug detection.
    - Submission of feedback.
- **Equivalence classes (conceptual examples) :**
    - For login: valid/invalid credentials, empty fields.
    - For product quantity : positive integers, zero, negative integers, non-numeric input.
    - For dates : valid format, invalid format, logical date sequences (expiration after production).
- Tests simulated real user interactions, verifying database updates, proper request handling, and potential system errors. The database performance was confirmed to be smooth with no data loss. No error messages were detected from the script, and controllers functioned correctly.

What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?

- **Technical Lessons :**
    - **Effective server usage:** The team realized the importance of a well-structured server to manage client requests and deliver real-time data updates efficiently.
    - **Handling HTTP requests properly:** Designing clear and consistent API endpoints improved communication between the front-end and back-end, reducing errors and simplifying debugging.
    - **Creating and maintaining test files:** Writing automated test cases early on helped catch bugs faster and maintain code quality throughout development.
    - The team faced difficulties in ensuring **seamless database integration and maintaining error-free request handling**. Debugging scripts and optimizing controllers were crucial to prevent data loss and improve system reliability.
    - Gained valuable experience in **database management, error detection**.

- **Process and Collaboration Lessons :**
  - **Team communication:** Frequent and clear communication within the team was crucial to align on design decisions, share progress, and resolve challenges quickly.
  - The importance of **rigorous testing, clear code organization, and teamwork**.
  - The benefits of the **Agile methodology**: The demonstration highlights the Agile methodology, emphasizing iterative development and teamwork. Regular evaluations, feedback loops, and team discussions allowed for quick improvements, refining the design and functionality throughout development.
  - Gained experience in **collaborative development**.

What functionalities are you going to demo?

- The demo showcases key functionalities, including:
  - User authentication.
  - Inventory tracking (adding a product, updating stock, viewing inventory, deleting products).
  - Seamless order management (mentioned as a key functionality).
  - The feedback submission feature.
- The demo highlights how the system efficiently handles stock updates and allows bakers to manage their inventory with ease.

Who is going to speak about each portion of your presentation?

- **Alec Marchal:** Design and Conception parts.
- **Clément CROUAN:** Development of JavaScript code.
- **Alexandre LOGUT:** Demo, code review, and server-side (MySQL) development.

Other notes :

- The project successfully delivered a functional inventory management system tailored to bakery needs.
- Testing confirmed smooth database operations, effective request handling, and a user-friendly interface.
- No data loss or system errors were detected, ensuring reliability and efficiency.
- **Future enhancements considered :**

- Adapted visual styles for accessibility, ensuring better usability for individuals with visual impairments.
- Improved database control on the user side to provide more flexibility and security in managing inventory.
- A mobile-responsive version to enhance accessibility across different devices, making bakery management more convenient.