From Waste to Efficiency:

AI Solutions in Food Expiration Management

Business Challenge

MVP Prototype

MBAN2 - Group 7

Ania de Chavigny

Cátia Magalhães

Harsh Kammath

Jody Leaser

Mateus Gomes

Shenali Jayasekara

1. Model Structure

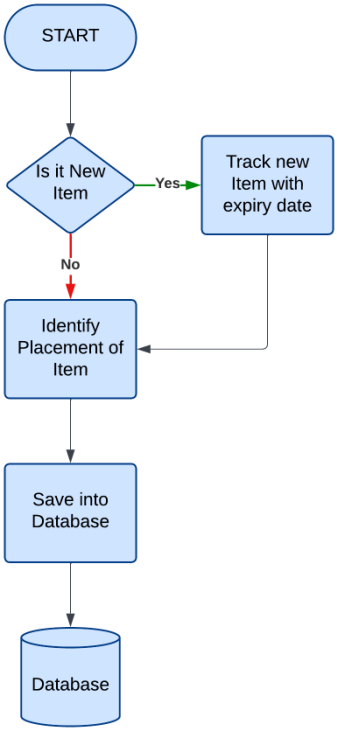


Figure 1 Flowchart Design for Implementing Camera Based Detection

The flowchart above describes the process that will be used to detect the placement of products and their exact position in the supermarket. The integrated AI camera will determine if a product is removed/kept on the shelves. For products being placed on the shelves, features such as YOLO (You Only Look Once) , DeepSORT (Deep Simple Online and Realtime Tracking) and OCR (Optical Character Recognition) will be used to identify the type of product, its active position and its expiry date respectively after which an API call (Application Programming Interface), will be used to update the underlying database for their respective positions in the supermarket. For products being removed, the same API call will be done to update the database.

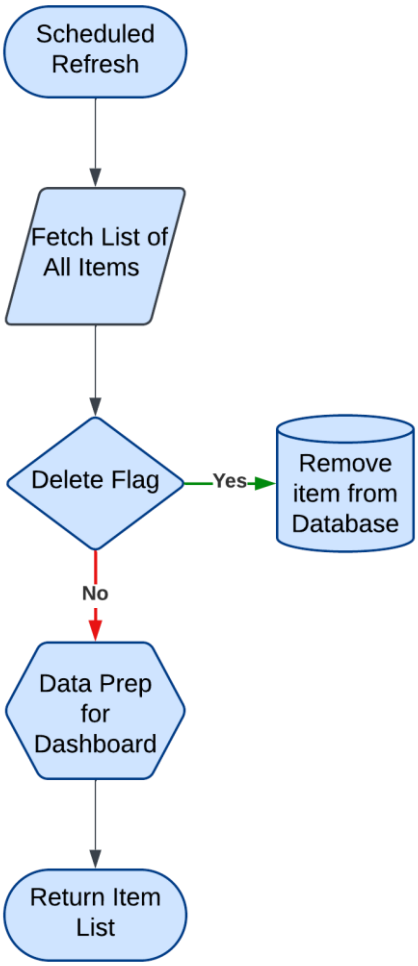


Figure 2 Flow chart Design for End of Day Product Removal

The flow chart describes how an end-of-day process would generate a report. Every day, at store closing time, a CRON-based scheduler will be used to call an API. The interface would then perform logical operations to determine what products need to be acknowledged. Accordingly, a dashboard update will provide the supermarket managers/employees with the following information: the number of products that need to be removed, product type, and the location of each product.

AI Structure

The AI system in the prototype consists of three primary models: object detection, object tracking and optical character recognition (OCR).

The object detection model utilizes a Convolutional Neural Network (CNN)-based architecture, such as YOLO, for real-time object detection (Redmon & Farhadi, 2018). Its purpose is to identify products within the camera's view. Each product is recognized based on pre-trained object detection models fine-tuned on labeled product datasets (YOLO: Real-Time Object Detection, n.d.). The output consists of bounding boxes around detected items, along with a class label (e.g., "orange juice carton", "milk carton").

The object tracking algorithm (DeepSORT) allows the distinction between similar products (e.g., two cartons of orange juice). It will then keep the real time position of the product. One additional feature of

this algorithm is that it keeps a unique ID associated with each of these products enabling the backend process also to identify the distinction made by the algorithm. (Wojke, Bewley, & Paulus, 2017)

The optical character recognition (OCR) model aims to read expiry dates from product packaging by leveraging open-source libraries like Tesseract or advanced OCR APIs (e.g., Google Cloud Vision, AWS Textract) to extract text from product labels (An overview of the Tesseract OCR engine, 2007). The output consists of text strings containing the expiry date.

All three of them are then combined to enable us to form the structure of a unique product, its position in the supermarket and its expiry date. With regards to the position of the product, the camera will have to be trained to send updates only when the tracking system (DeepSORT) picks up the location of the products to be in a shelf/rack (Singh & Verma, 2020).

Non-AI Structure

The MVP's architecture employs a Spring Boot application as the backend API. The architecture follows a RESTful (REpresentational State Transfer) design, enabling seamless interaction between hardware components (cameras/sensors) and the database (Spring boot, n.d.).

The system adheres to the RESTful framework (see Appendix 14).

- The controller module manages HTTP (Hypertext Transfer Protocol) requests from AI or hardware systems and provides endpoints for CRUD (Create, Read, Update, Delete) operations on product data, such as a POST /updatePosition endpoint to update the database with a product's location and expiry date based on camera inputs.
- The service layer processes business logic, including calculating shelf life from expiry dates and tracking product movements, while also triggering notifications for items nearing expiration.
- Data management is handled by the data access layer (DAO), which uses Spring Data JPA (Java Persistence API) to execute queries, such as retrieving products nearing expiry or updating product positions (Spring Data JPA, n.d.).
- A scheduler module automates tasks by running scheduled jobs, such as querying the database every evening for products approaching their expiration date using Spring's @Scheduled annotation and sending notifications with details like product ID and location to store managers (The @Scheduled Annotation in Spring, n.d.).
- The backend database, implemented as a relational database, includes tables for products, tracking attributes like product ID, name, location, expiry date, and status (e.g., sold or expired), as well as transactions, which log product movements, including removals or updates.

2. Data Sources

AI Data sources

To train and operate the AI models, we rely on a product image dataset, an expiry date text dataset, and an API/database integration.

The product image dataset consists of high-resolution images of products from various angles. The data is gathered from open datasets (e.g., Google Open Images), proprietary store databases, or manually curated images from the supermarket that we offer our services to (Kuznetsova, et al., 2020). It enables the object detection model to accurately recognize product types and packaging (Lin, et al., 2014).

The expiry date text dataset contains images with printed expiry dates from products. The data consists of manually captured label images or open datasets focusing on text recognition tasks. It trains the OCR model to recognize and interpret expiry dates.

Lastly, the API/database integration consists of product inventory data containing details like product IDs, stock levels, expiry dates, and location (e.g., fridge or shelf position). It's gathered from supermarkets' existing inventory management systems. It enables the integration of AI outputs into the inventory system to update product status.

Non-AI Data Sources

For processes not reliant on AI, our data consists of product metadata (includes product ID, name, location, expiry date, and category) and transaction logs (tracks product lifecycle events, e.g., added to inventory, moved, or sold). This initial data is sourced from the store's inventory management system and updates are provided by the camera or sensor system (via API).

These sources allow for accurate product tracking to ensure timely identification of items nearing expiry. Furthermore, location data helps store employees locate flagged products more efficiently.

3. Integration

AI Integration

The AI models are seamlessly integrated into the prototype to enhance operational efficiency. For input, the camera captures images whenever motion is detected, e.g., a product removed from the fridge; in processing, the object detection model identifies the product type, and the OCR model extracts the expiry date from the label. As an output, we get the product type (e.g., "Orange Juice"), expiry date (e.g., "20-12-2024"), and camera location for product position tracking.

As for the API integration, the extracted information (product type, expiry date, location) is sent to the Spring Boot API, updating the supermarket's inventory database. The API will update the database with the output (product type, expiry date and product location).

This system enhances effectiveness through proactive monitoring, real-time updates which ensure accurate stock tracking, waste reduction and improved inventory turnover, and scalability (the modular design allows for easy extension to other products or supermarket locations).

Non-AI Integration

The Spring Boot API acts as the middleware, connecting the hardware (camera/sensor) to the database. Upon detecting product movement or removal, the camera triggers the API with updated information. The scheduler processes data daily to flag products nearing expiry and notifies store employees.

This system enhances effectiveness through automation (it automates the tracking process, minimizing manual labor and human error), scalability (the modular design of the Spring Boot application allows for easy scaling to accommodate additional product categories or store locations), timely notifications, and data-driven decision-making (transaction logs provide insights into product demand and shelf life, enabling better inventory management).

4. Testing and Refinement Insights

Although no physical testing has been performed during the ongoing development phase, the proposed testing framework prioritizes iterative refinement and usability to tackle anticipated challenges. The following methodology underscores the importance of functionality, usability, and responsiveness.

The proposed tests could address critical gaps in functionality and practical limitations identified in the design stage. For example, potential issues in identifying products and tracking were acknowledged through the OCR integration in order to raise the precision level when reading labels and expiration dates of the products. The system will be able to differentiate between products even when items are taken out and then put back on the shelf. A flagging mechanism is also being worked on in order to track shelf position changes, so as to assure accurate monitoring of the inventory by automatically assigning binary statuses to products in real-time. These efforts are being made to enhance the accuracy of the system in managing food waste within supermarket shelves.

Preview of Interactive Dashboard

For a quick overview of the interactive dashboard prototype, use this **link** (access must be requested) for the desktop version or refer to the dashboard visualizations below, including the desktop and tablet displays. This development phase focused on enhancing the user experience by incorporating feedback and optimizing visualizations to ensure clarity and engagement across all devices.

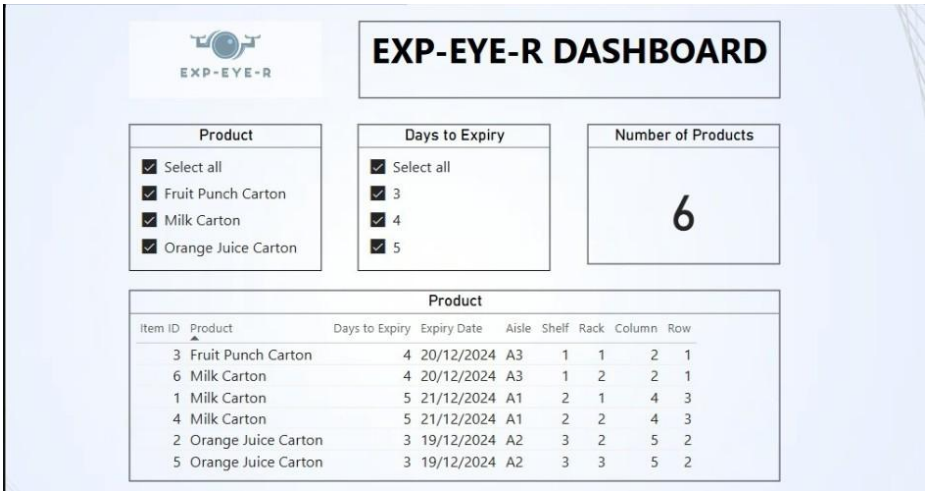


Figure 3 Desktop View

Figure 4 Tablet View

The interactive dashboard provides a clear, user-friendly interface for managing product expiration and inventory in real-time. It features dynamic visualizations that allow users to view product counts by type and track trends for items nearing expiry. Product filters enable users to customize their view based on product type and days to expiry, helping them focus on the most critical information. The dashboard also provides detailed location insights, including the product's positions, ensuring quick retrieval. Actionable summaries display total product counts and days-to-expiry breakdowns, streamlining decision-making for store staff.

5. Stakeholder Feedback Summary

As part of our feedback retrieval, we got testimonials from three supermarket workers.

Star Market Store Manager Feedback:

- Positive: "This system is great for managing inventory. It will save hours of manually checking expiration dates."
- Problem: "What if the cameras don't work effectively?"

Boston Convenience Staff Feedback

- Positive: "I love that it tells us the product exact location. It would make the job easier during end-of-day checks."

- Problem: “What happens if the product labels are hard to read? Will the system give false alerts?”

Practical Adjustments and Considerations

To address the store manager's concerns about camera malfunctions or AI misidentification that might cause disruptions in operations, we will create a redundancy plan including backup cameras and self-diagnostic features designed to automatically identify system errors and notify IT support. Further, manual overrides will be included in the system dashboard, thus allowing staff to correct misidentifications in real time. For the concern about hard-to-read labels, we will introduce confidence scoring in the OCR model to flag low-quality reads for manual review and recommend the incorporation of barcode-based identification as an alternative. Further improvement in OCR robustness would be through training the system with augmented datasets containing damaged or obscured labels.

The iterative process will also consider possible constraints in the hardware setup. In recognition that static cameras might not be able to capture everything in extended or complicated setups, a moving camera system is proposed as a scalable solution. This would ensure complete coverage in different environments. Further, concerns regarding hardware reliability and server downtime will be answered proactively by developing an overall technical support plan to strengthen the server infrastructure. We expect, through the integration of user feedback and analysis of test data, to continuously improve usability and functionality in the prototype.

References

An overview of the Tesseract OCR engine. (2007, September 1). https://ieeexplore.ieee.org/document/4376991

Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST). (n.d.). Retrieved December 10, 2024, from ics.uci.edu: https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Kuznetsova, A. R., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., . . . Ferrari, V. (2020). The Open Images Dataset V4. International Journal of Computer Vision, 128(7), 1956–1981. https://doi.org/10.1007/s11263-020-01316-z

Lin, T., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., . . . Zitnick, C. L. (2014). Microsoft COCO: Common Objects in context. In Lecture notes in computer science, 740-755. https://doi.org/10.1007/978-3-319-10602-1_48

Redmon, J., & Farhadi, A. (2018, April 8). YOLOV3: an incremental improvement. Retrieved December 10, 2024, from arXiv.org: https://arxiv.org/abs/1804.02767

Singh, R., & Verma, S. (2020). Computational of distribution of wind speed as preliminary information for fishers: case study in Lombok Sea. Computational of distribution of wind speed as preliminary information for fishers: case study in Lombok Sea, 9(3), 3584–3587: https://doi.org/10.30534/ijatcse/2020/165932020

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. Journal of Big Data, 6(1). https://doi.org/10.1186/s40537-019-0197-0

Spring boot. (n.d.). Retrieved December 10, 2024, from Spring Boot: https://spring.io/projects/spring-boot

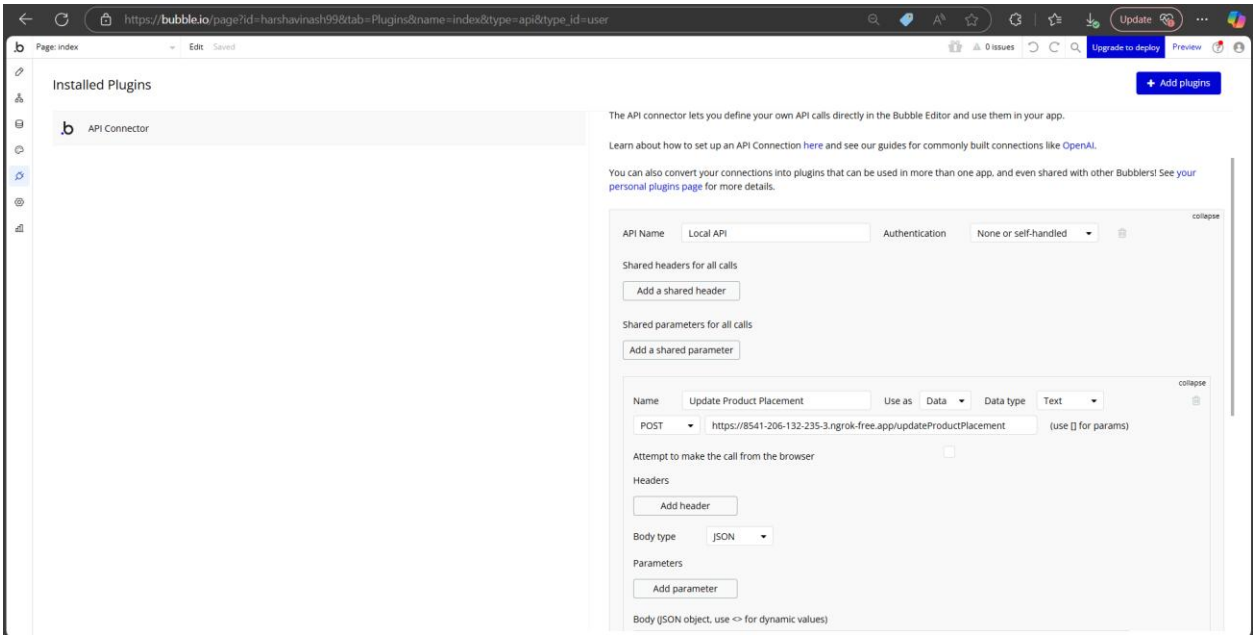
Spring Data JPA. (n.d.). Retrieved December 10, 2024, from Spring Data JPA: https://spring.io/projects/spring-data-jpa

The @Scheduled Annotation in Spring. (n.d.). Retrieved December 10, 2024, from Baeldung: https://www.baeldung.com/spring-scheduled-tasks

Wojke, N., Bewley, A., & Paulus, D. (2017, September 1). Simple online and realtime tracking with a deep association metric. doi: https://doi.org/10.1109/ICIP.2017.8296962

YOLO: Real-Time Object Detection. (n.d.). Retrieved December 10, 2024, from pjreddie.com.

Appendix



Appendix 1: BUBBLE.IO API INTEGRATION

AI CAMERA TO DETECT PRODUCT PLACEMENT & REMOVAL (PYTHON)

```python
def process_camera_feed():
    cap = cv2.VideoCapture(0)  # Start camera feed; use the appropriate camera
        index

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Failed to capture frame from camera.")
            break

        # Display the frame
        cv2.imshow("Camera Feed", frame)

        # Placeholder for product detection logic (use object detection model
            here)
        product_detected = True  # Simulated detection
        product_removed = False  # Simulated removal

        if product_detected:
            print("Product detected.")
```

```python
            ocr_result = perform_ocr(frame)
            print("OCR Result:", ocr_result)

            # Extract relevant details from OCR result (example parsing logic)
            product_id = 10001  # Example ID; update with actual parsing logic
            expiry_date = "2024-12-12"  # Example expiry date; update with
                actual parsing logic
            shelf_number = 1
            row_number = 1
            column_number = 1
            action = "Add"

            # Call the API
            call_api(product_id, shelf_number, row_number, column_number,
                action, expiry_date)
```

Appendix 2: Recognizing if Product is removed/placed

```python
# Function to perform OCR on a detected product
def perform_ocr(frame):
    # Convert frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Use Tesseract to extract text
    extracted_text = pytesseract.image_to_string(gray)
    return extracted_text
```

Appendix 3: OCR Recognition

```python
def call_api(product_id, shelf_number, row_number, column_number, action,
    expiry_date):
    data = {
        "productId": product_id,
        "shelfNumber": shelf_number,
        "rowNumber": row_number,
        "columnNumber": column_number,
        "action": action,
        "expiryDate": expiry_date
    }
    try:
        response = requests.post(API_URL, json=data)
        if response.status_code == 200:
            print("API call successful:", response.json())
        else:
            print(f"API call failed with status code {response.status_code}:
                {response.text}")
    except Exception as e:
        print("Error during API call:", e)
```

Appendix 4: Calling the API

REST API SPRING FRAMEWORK

```java
package com.dreamteam.product.controller;

import org.springframework.beans.factory.annotation.Autowired;

@RestController
public class UpdateProductPlacementController {

    @Autowired
    private UpdateProductPlacementService updateProductPlacementService;

    @PostMapping("/updateProductPlacement")
    public ResponseEntity<Boolean> updateProductPlacement(@RequestBody UpdateProductPlacmentInput input) {
        return ResponseEntity.ok(updateProductPlacementService.updateProductPlacement(input));
    }

}
```

Appendix 5: API Controller

```java
package com.dreamteam.product.service;

import org.springframework.beans.factory.annotation.Autowired;

@Service
public class UpdateProductPlacementService {

    private ProductPlacementRepository productPlacementRepository;

    @Autowired
    public UpdateProductPlacementService(ProductPlacementRepository productPlacementRepository) {
        this.productPlacementRepository = productPlacementRepository;
    }

    public Boolean updateProductPlacement(UpdateProductPlacmentInput input) {
        if (input.getAction() != null & !input.getAction().isEmpty()) {
            if (input.getAction().equals("Add")) {
                productPlacementRepository.updateProductDetailsForAddition(input.getProductId(), input.getShelfNumber(),
                        input.getRowNumber(), input.getColumnNumber(), input.getExpiryDate());
                return Boolean.TRUE;

            } else if (input.getAction().equals("Remove")) {
                productPlacementRepository.updateProductDetailsForRemoval(input.getProductId(), input.getShelfNumber(),
                        input.getRowNumber(), input.getColumnNumber(), input.getExpiryDate());
                return Boolean.TRUE;
            }
        }
        return Boolean.FALSE;
    }

}
```

Appendix 6: API Service

```java
package com.dreamteam.product.repository;

import java.util.Date;

@Repository
public interface ProductPlacementRepository extends JpaRepository<ProductPlacementEntity, Integer> {

    @Modifying
    @Transactional
    @Query("UPDATE ProductPlacementEntity p SET p.productPresentFlg = 2, p.expiryDate = :expiryDate "
            + "WHERE p.productId = :productId AND p.shelfNumber = :shelfNumber AND p.rowNumber = :rowNumber AND p.columnNumber = :columnNumber")
    int updateProductDetailsForRemoval(@Param("productId") Integer productId, @Param("shelfNumber") Integer shelfNumber,
            @Param("rowNumber") Integer rowNumber, @Param("columnNumber") Integer columnNumber,
            @Param("expiryDate") Date expiryDate);

    @Modifying
    @Transactional
    @Query("UPDATE ProductPlacementEntity p SET p.productPresentFlg = 1, p.expiryDate = :expiryDate "
            + "WHERE p.productId = :productId AND p.shelfNumber = :shelfNumber AND p.rowNumber = :rowNumber AND p.columnNumber = :columnNumber")
    int updateProductDetailsForAddition(@Param("productId") Integer productId, @Param("shelfNumber") Integer shelfNumber,
            @Param("rowNumber") Integer rowNumber, @Param("columnNumber") Integer columnNumber,
            @Param("expiryDate") Date expiryDate);

    List<ProductPlacementEntity> findByProductId(Integer productId);

}
```

Appendix 7: API Repository layer (Updating Database)

Process 1

AI Camera saves information into our PRODUCT_PLACEMENT table:

REST API (POST)		
INPUT	productId	From Barcode (trained by ML algorithm)
	expriryDate	From OCR Recognition
	shelfNumber	Placement of product (ML algorithm)
	rowNumber	
	columnNumber	
	action	‘Remove'/'Add'
OUPUT	successFlag	Fail/Success
EXCEPTION HANDLING	Follows existing norms	

Table 1: REST API call to API with endpoint /updateProductPlacementDetails

Process 2

CRON Scheduler used to return Products which have expired

REST API (FETCH)		
INPUT	none	
OUTPUT	List<MissedProducts>	productName
		shelfNumber
		rowNumber
		columnNumber

EXCEPTION HANDLING	Follows existing norms

Table 2: REST API call to API with endpoint /fetchExpiredProducts

Code Block:

```
@SpringBootApplication
@EnableScheduling
public class SchedulerApplication {
    public static void main(String[] args)
    {
        SpringApplication.run(SchedulerApplication.class,
                              args);
    }
}
```

Appendix 8: CRON code block:

```
@Component
public class Scheduler {
    // RUNNING A JOB EVERYDAY AT 6 PM
    @Scheduled(cron = "0 18 * * *")
    public List<MissedProducts> scheduleTask()
    {
        List<MissedProducts> missedProducts = restTemplate.exchange("URL",
            HttpMethod.FETCH, entity, MissedProducts.class).getBody();
        return missedProducts;
    }
}
```

Appendix 9: CRON Service Logic

```java
@Service
public class FetchExpiryProducts {

    private ProductPlacementRepository productPlacementRepository;

    @Autowired
    public FetchExpiryProducts (private ProductPlacementRepository
        productPlacementRepository) {
        this.productPlacementRepository = productPlacementRepository;
    }

    public List<MissedProducts> fetchExpiryProducts() {
        List<MissedProducts> expiryProducts = new ArrayList<>();
        List<ProductPlacementEntity> entityList = productPlacementRepository
            .fetchAllRecords();
        if (!entityList.isEmpty()) {
            for (ProductPlacementEntity entity : entityList) {
                if (entity.getProductPresentFlg()) {
                    if (entity.getExpiryDate() > LocalDate.now() - 1) {
                        expiryProducts.add(new MissedProducts());
                        // Initialize input to list with relevant values
                    }
                }
            }
        }
```

Appendix 10: Fetch API code block

DATABASE STRUCTURE

PRODUCT_PLACEMENT	DATATYPE
PRODUCT_ID	INT
SHELF_NUMBER	INT
ROW_NUMBER	INT
COLUMN_NUMBER	INT
EXPIRY_DATE	DATE
PRODUCT_PRESENT_FLG	INT

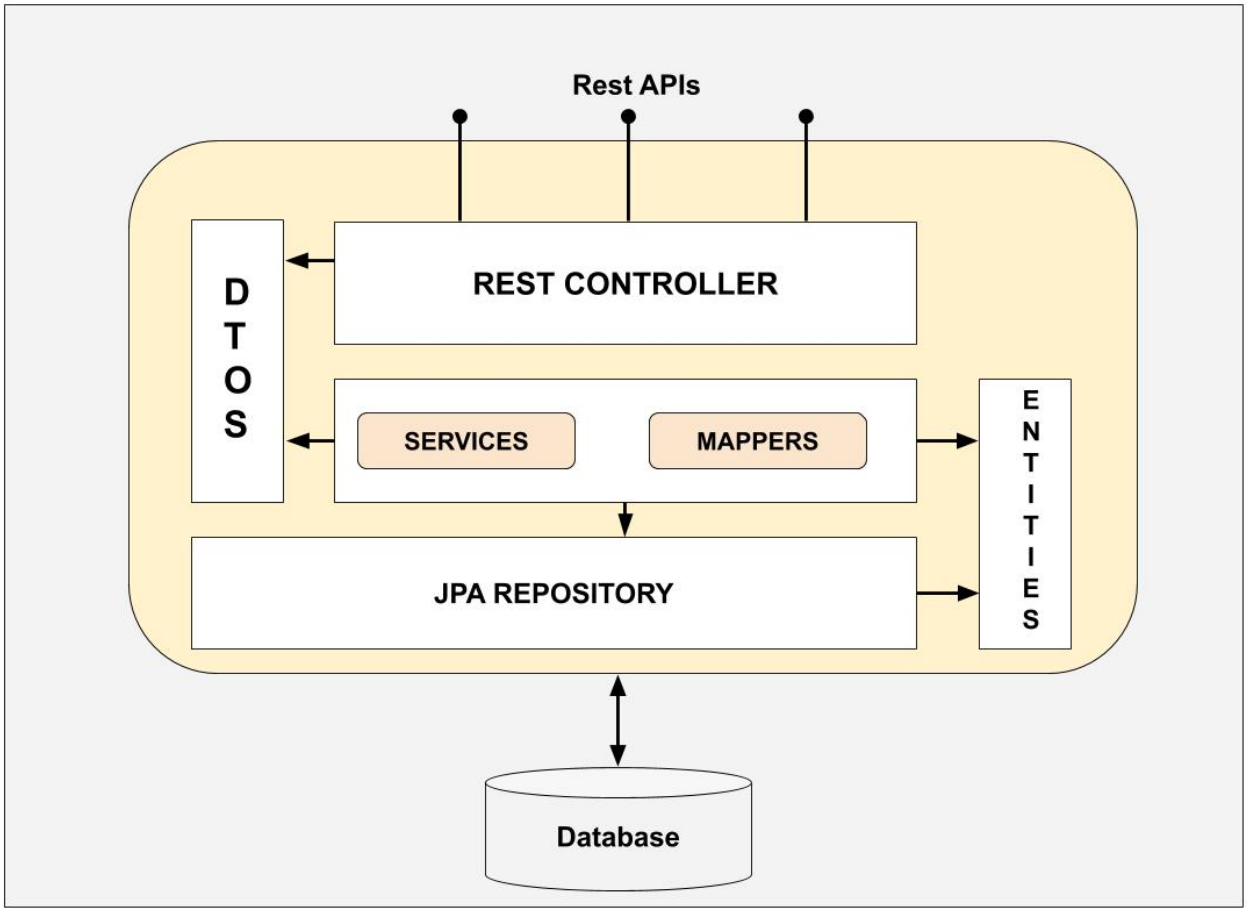
Appendix 11: Newly created table which will be used to host product placement information

PRODUCT_INVENTORY
BATCH_ID
PRODUCT_ID
TOTAL_UNITS
UNITS_SOLD
EXPIRY_DATE

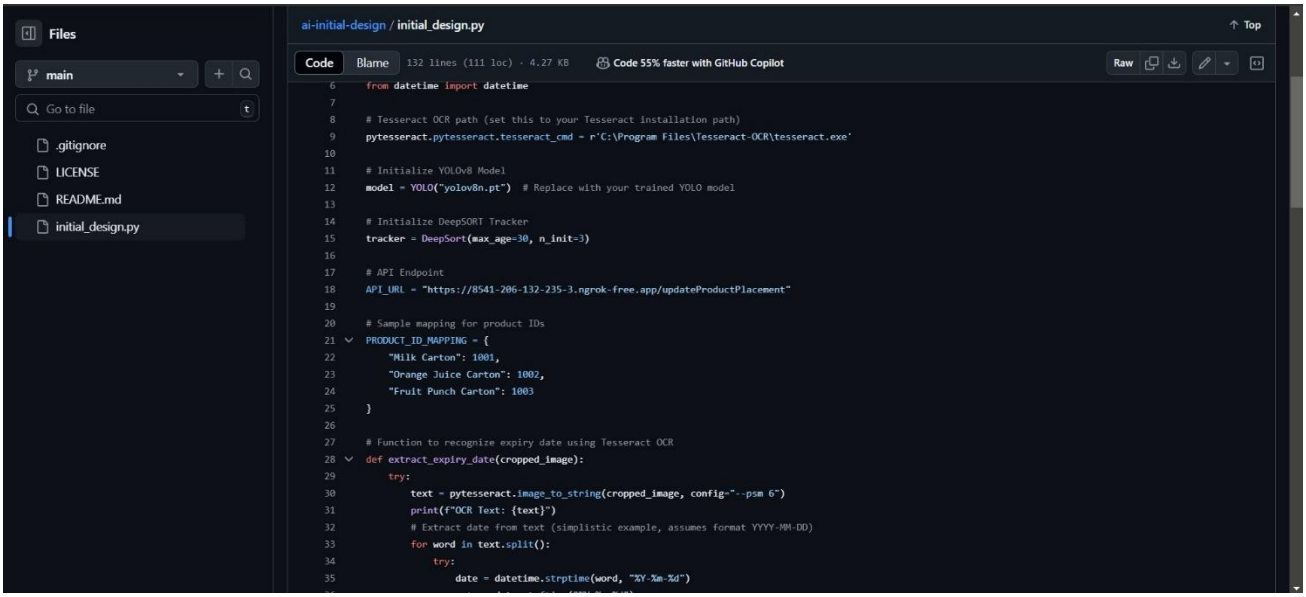
Appendix 12: Newly created table which will be used to host product placement information (Inventory)

PRODUCTS
PRODUCT_ID
PRODUCT_NAME

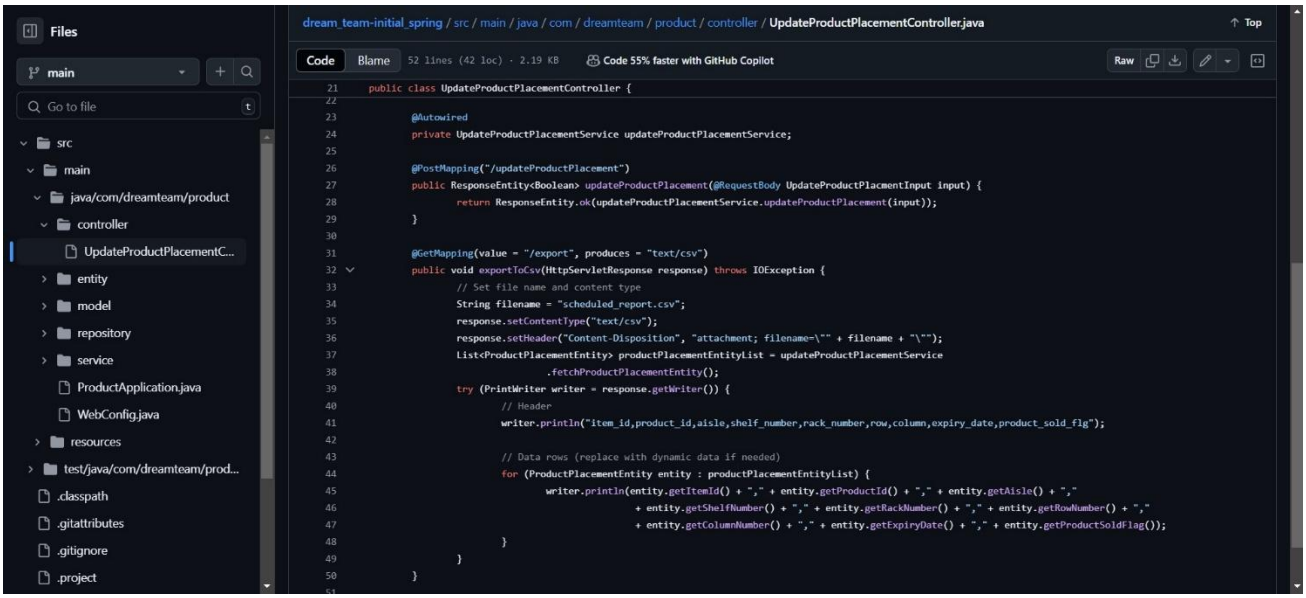
Appendix 13: Newly created table which will be used to host product placement information (Product Level Info)



Appendix 14: REST API Architecture



Appendix 15: Sample Code saved on GitHub, link: https://github.com/hkammath/ai-initial-design/blob/main/initial_design.py



Appendix 16: Spring Boot app with fully working API's, link: https://github.com/hkammath/dream_team-initial_spring/tree/main