# Extended Flask API Exercise: Cake Price Prediction System with Database Integration

## Background

You've successfully created a basic Flask API for cake price prediction. Now, your bakery business is growing, and you need to enhance your system by adding data persistence, improving API documentation, and implementing more sophisticated features.

## Requirements

### 1. Database Integration with PostgreSQL in Docker

- Set up a PostgreSQL database using Docker
- Create a `predictions` table with the following schema:
  - `id` (UUID, primary key)
  - `timestamp` (datetime, when the prediction was made)
  - `radius` (float)
  - `layers` (integer)
  - `topping` (string)
  - `predicted_price` (float)
  - `customer_id` (string, nullable - for future use)

### 2. Enhanced API Endpoints

#### 2.1 Modified Prediction Endpoint

- Update the `/predict` endpoint to:
  - Accept the input parameters (radius, layers, topping)
  - Generate a unique ID for the prediction
  - Save the prediction results to the database
  - Return the prediction ID along with the predicted price

#### 2.2 New Prediction Retrieval Endpoint

- Create a new endpoint `/predictions/<prediction_id>` that:
  - Accepts a prediction ID as a path parameter

    - If the ID exists in the database, retrieves and returns the prediction details
    - If the ID doesn't exist, runs the model to make a new prediction, saves it with the provided ID, and returns the results
    - Includes appropriate error handling for invalid IDs

### 2.3 Prediction History Endpoint

- Create a new endpoint `/history` that:
    - Retrieves the last 10 predictions from the database
    - Optional query parameter `limit` to specify how many records to retrieve
    - Optional query parameter `customer_id` to filter by customer

## 3 Environment Configuration

- Use environment variables for configuration
- Create different configuration profiles (development, testing, production)
- Include a sample `.env` file and documentation on required environment variables