1. Mission Briefing
   Objective: To build a RESTful API (preferably in Ruby on Rails) that manages restaurants, their menus, and the items within them. The project must be developed iteratively across three levels of complexity.

Expected Outcome: A Git repository containing the application source code, a robust suite of automated tests, and a functional data import tool.

2. Key Implementation Principles
   The agent must prioritize the following concepts throughout development:

Extensibility: The code must be adaptable to future requirement changes.

Quality and Testing: A well-tested, low-bug application is preferred over one with more features but no tests.

Iterative Approach: Development must follow Levels 1, 2, and 3 in order. Git commits should reflect this progress.

Exception Handling: The application must handle errors and log information appropriately.

3. Provided Input Files

PopmenuInterviewProject.pdf: The requirements document (this project's manifest).

data.json: A JSON file containing sample restaurant, menu, and item data to be used for the Level 3 import.

4. Detailed Task Specification (By Level)
   Level 1: Basic Data Models

Objective: Create the initial Object Relational Model (ORM) and endpoints for Menu and MenuItem.

Models:

Menu

MenuItem

Attributes (Inferred):

Menu: name (string)

MenuItem: name (string), price (decimal/float), description (text, optional). (Note: price will be refactored in Level 2).

Association:

A Menu has many MenuItems (1:N relationship).

Endpoints:

Create appropriate RESTful/JSON endpoints to return the model data.

Tests:

Illustrate behavior via unit tests.

Level 2: Scalable Data Model (Multi-Restaurant)
Objective: Introduce the Restaurant concept and refactor the data model to allow MenuItems to be shared across menus, but with different prices.

New Model:

Restaurant.

Attribute: name (string).

New Associations:

A Restaurant has many Menus.

A Menu belongs to one Restaurant.

Critical Refactor: MenuItem and Price

Constraint 1: MenuItem names must not be duplicated in the database (must be unique).

Constraint 2: A MenuItem can be on multiple Menus of a Restaurant.

Insight (from data.json): The data.json file shows "Burger" exists in two menus for "Poppo's Cafe" with two different prices ($9.00 in "lunch" and $15.00 in "dinner").

Mandatory Solution: The Level 1 model is now obsolete. price cannot be an attribute of MenuItem.

The MenuItem model must only store global item data (e.g., name).

A new join model must be created (e.g., PricedItem, MenuEntry, or Offering).

This new model (e.g., PricedItem) will contain:

menu_id (reference to Menu)

menu_item_id (reference to MenuItem)

price (decimal/float)

Endpoints:

Create appropriate endpoints to return the new model data. (e.g., CRUD for Restaurant, refactored Menu endpoints nested under Restaurant).

Tests:

Illustrate the new behavior via unit tests.

Level 3: JSON Import Tool
Objective: Process the data.json file and persist its data into the Level 2 data model.

Import Service:

Create an HTTP endpoint that accepts JSON files.

Write a "conversion tool" (e.g., a Service Object, Rake task, or class called from the endpoint) to serialize and persist the file's data.

This tool must be available to run with instructions.

Processing Logic (Challenges from data.json):

Inconsistent Keys: The data.json file uses different keys for item lists: "menu_items" (in "Poppo's Cafe") and "dishes" (in "Casa del Poppo"). The importer must be able to process both keys.

Data Normalization: The importer must map the JSON data to the Level 2 model (Restaurant -> Menu -> PricedItem -> MenuItem).

Idempotency: The importer must use "find-or-create" logic for Restaurant and MenuItem (based on name).

Example: "Burger" should result in a single MenuItem record in the database but multiple PricedItem records (one for each menu/price).

Example: "Chicken Wings" appears twice under the "lunch" menu for "Casa del Poppo". The importer should be smart and create only one PricedItem entry for "Chicken Wings" on that specific menu.

Output and Logs:

The tool's output should return a list of logs for each menu item, indicating a success/fail result.

Robustness:

Apply adequate logging and exception handling.

Tests:

Illustrate the importer's behavior via unit tests