Menu Catalog API

REST API untuk manajemen menu catalog dengan Express.js dan SQLite.

Menu Catalog API (Merged Documentation)

Comprehensive documentation merged from separate files: `CALORIES_API.md`, `CALORIES_FEATURE_SUMMARY.md`, `CALORIES_IMPLEMENTATION.md`, `ID_MANAGEMENT.md`, and `RECOMMENDATIONS_API.md` — consolidated here for easier reference.

Table of Contents

PROF

Project overview

REST API untuk manajemen menu catalog dengan Express.js dan SQLite.

Project layout (top-level):

```
gdgoc/
├── src/                # app source: config, controllers, models,
routes, services, middlewares, utils
├── .env                # Environment variables (not committed)
├── database.sqlite     # SQLite DB file
├── package.json
├── CALORIES_API.md
├── CALORIES_FEATURE_SUMMARY.md
├── CALORIES_IMPLEMENTATION.md
```

```
├── ID_MANAGEMENT.md
├── RECOMMENDATIONS_API.md
└── README.md            # This merged documentation
```

🚀 Setup

1. Install dependencies:

```
npm install
```

2. Copy environment file:

```
cp .env.example .env
```

3. Run the app:

```
# Development
npm run dev

# Production
npm start
```

📡 API Endpoints (summary)

Base URL: `http://localhost:3000/api`

Key endpoints:

- POST /menu — create menu
- GET /menu — list menus (filter, pagination)
- GET /menu/:id — get menu by id
- PUT /menu/:id — update menu
- DELETE /menu/:id — delete menu
- DELETE /menu/all — delete all menus (resets ID sequence)
- GET /menu/group-by-category — group by category
- GET /menu/search — search
- POST /menu/auto-generate — AI generate menu
- POST /menu/recommendations — AI-powered recommendations (see section)
- POST /menu/calculate-calories — Calculate calories & exercise recommendations (detailed below)

Refer to the dedicated sections for full request/response examples, validation rules, and error cases.

Calories Calculator & Exercise Recommendations (Full API)

This endpoint computes total calories from user-selected menu items and returns exercise recommendations to burn those calories using Google Gemini AI.

POST /api/menu/calculate-calories

Request body example:

```
{
  "menu_items": [
    { "id": 1, "quantity": 2 },
    { "name": "Nasi Goreng", "quantity": 1 }
  ]
}
```

Field descriptions:

- menu_items (array, required): list of selected menu items
  - id (integer, optional): menu id in DB (use id or name)
  - name (string, optional): menu name in DB (use id or name)
  - quantity (integer, optional): number of portions (default: 1)

Response (200 OK) example:

```
{
  "status": "success",
  "message": "Calorie calculation and exercise recommendations generated
successfully",
  "data": {
    "total_calories": 1450,
    "nutritional_breakdown": {
      "protein": "45g",
      "carbohydrates": "180g",
      "fats": "55g",
      "fiber": "12g"
    },
    "menu_details": [
      { "name": "Nasi Goreng Spesial", "calories": 650, "quantity": 1,
"subtotal_calories": 650 },
      { "name": "Es Teh Manis", "calories": 200, "quantity": 2,
"subtotal_calories": 400 },
      { "name": "Pisang Goreng", "calories": 400, "quantity": 1,
"subtotal_calories": 400 }
    ],
    "exercise_recommendations": [
      { "name": "Jogging", "duration_minutes": 120, "intensity":
"moderate", "calories_burned_per_hour": 450, "description": "Jogging at
a steady pace of 5-6 mph", "tips": "Maintain a consistent pace and stay
```

PROF

```
hydrated" },
      { "name": "Swimming", "duration_minutes": 90, "intensity":
"moderate", "calories_burned_per_hour": 500, "description": "Swimming
laps using various strokes", "tips": "Mix different strokes for a full-
body workout" }
    ],
    "health_notes": "This meal provides a good balance of
macronutrients. Consider incorporating more vegetables for additional
fiber and micronutrients.",
    "summary": "Your selected meal contains 1,450 calories. To burn
these calories, you could jog for 2 hours, swim for 1.5 hours, or choose
from the other exercise options provided. Remember to stay hydrated
during exercise!"
  }
}
```


Error examples:

- 400 Bad Request — empty menu_items


```
{
  "error": "Validation Error",
  "message": "Invalid calories calculation request",
  "details": ["menu_items array cannot be empty"]
}
```


- 400 Bad Request — invalid format


```
{
  "error": "Validation Error",
  "message": "Invalid calories calculation request",
  "details": [
    "Menu item at index 0 must have either 'id' or 'name' field",
    "Menu item at index 1: quantity must be a positive integer"
  ]
}
```


- 404 Not Found — menu not found


```
{
  "status": "error",
  "message": "Menu item \"Rendang Sapi\" not found in database"
}
```


- 500 Internal Server Error — Gemini API error

```
{
  "status": "error",
  "message": "Failed to calculate calories and generate exercise
recommendations"
}
```

Usage examples (curl):

```
curl -X POST http://localhost:3000/api/menu/calculate-calories \
  -H "Content-Type: application/json" \
  -d '{"menu_items": [{"id": 1, "quantity": 1}, {"id": 5, "quantity":
2}]}'
```

If quantity omitted, default is 1.

Calories Feature Summary

Key changes and files added/modified for the calories feature:

- `/src/services/geminiService.js` — adds `calculateCaloriesAndExercise(menuItems)` which calls Google Gemini (model: gemini-2.5-flash) to compute total calories, nutrition breakdown, and exercise recommendations.
- `/src/controllers/menuController.js` — adds `calculateCaloriesAndExercise(req, res, next)` to validate input, fetch menu items from DB (by id or name), and call geminiService.
- `/src/routes/menuRoutes.js` — adds POST `/menu/calculate-calories` with a validator middleware.
- `/src/middlewares/validator.js` — adds `validateCaloriesRequest` to ensure `menu_items` is an array & valid.
- New docs & test artifacts: `CALORIES_API.md`, `CALORIES_IMPLEMENTATION.md`, `CALORIES_FEATURE_SUMMARY.md`, `calories-api-postman.json`, `test-calories-api.sh`.

PROF

Highlights:

- Flexible input: lookup by `id` or `name`.
- Quantity support.
- Nutrition breakdown and 3–5 exercise recommendations returned.
- Gemini model and prompt engineered to return JSON in the specified schema.

Calories Implementation Details

Overview of how the feature is implemented and the components involved.

Architecture flow:

- Client -> `menuRoutes` -> `menuController.calculateCaloriesAndExercise`
- Controller fetches menu data via `menuService` (by id or name), prepares menu details with quantities, and calls `geminiService.calculateCaloriesAndExercise(menuDetails)`.
- `geminiService` sends a prompt to Google Gemini (gemini-2.5-flash) with configuration: temperature 0.7, topK/topP, max tokens, and requests JSON response in the agreed schema.

Output JSON schema (from Gemini):

```
{
  "total_calories": number,
  "nutritional_breakdown": { "protein": "string", "carbohydrates":
"string", "fats": "string", "fiber": "string" },
  "menu_details": [...],
  "exercise_recommendations": [{ "name": "string", "duration_minutes":
number, "intensity": "low|moderate|high", "calories_burned_per_hour":
number, "description": "string", "tips": "string" }],
  "health_notes": "string",
  "summary": "string"
}
```

Validation & errors handled at controller/middleware level: missing items -> 400, menu not found -> 404, Gemini errors -> 500.

Testing suggestions:

- Unit tests for validator, controller (happy path + menu-not-found), and geminiService (mock Gemini responses).
- Integration: run `test-calories-api.sh` or use the provided Postman collection.

Performance notes:

- Gemini latency typically 2–5s; consider caching frequent combinations.
- Keep prompt concise to reduce token usage.

PROF

Menu Recommendations API (Full)

Endpoint: POST `/api/menu/recommendations`

Purpose: Use Gemini AI to generate personalized menu recommendations using only menu items present in the database.

Request example:

```
{
  "budget": 100000,
  "dietary_restrictions": ["vegetarian","halal"],
  "dislikes": ["kopi","pedas","durian"],
  "preferences": ["manis","coklat","cheese"],
```

```
  "meal_type": "lunch",
  "cuisine": "Indonesian",
  "occasion": "casual dining",
  "additional_notes": "Saya ingin makanan yang mengenyangkan dan tidak
terlalu berminyak"
}
```

Response example (success):

```
{
  "success": true,
  "message": "Menu recommendations generated successfully",
  "data": {
    "recommendations": {
      "main_course": { "id": 1, "name": "Nasi Goreng Spesial Sayuran",
"category": "main-course", "description": "...", "price": 35000,
"calories": 450, "ingredients": [...], "reason": "..." },
      "beverage": { "id": 15, "name": "Es Teh Manis", "price": 8000,
"calories": 120, "reason": "..." },
      "dessert": { "id": 23, "name": "Cheese Cake Coklat", "price":
45000, "calories": 380, "reason": "..." }
    },
    "total_price": 88000,
    "total_calories": 950,
    "summary": "Kombinasi menu ini dipilih khusus untuk Anda..."
  }
}
```

Errors:

- 400 — missing preferences or insufficient menu items in DB for required categories.
- 500 — Gemini or processing error.

PROF

Notes & tips:

- System only recommends items that exist in DB; ensure DB has `main-course`, `beverage`, and `dessert` categories.
- Provide as much detail as possible in request to improve recommendations.

ID Management (Full)

This project contains an ID management system that ensures predictable IDs and auto-reset behavior when DB becomes empty.

Key behaviors:

- New items receive ID = MAX(id) + 1; if DB empty, ID starts at 1.

- When all rows are deleted (`DELETE /api/menu/all`) or when the last item is removed, the ID sequence is reset so the next inserted item is ID 1.
- Implementation uses SQLite sequences / `sqlite_sequence` and AUTOINCREMENT behavior; reset performed with `DELETE FROM sqlite_sequence WHERE name='menu'` when appropriate.

API examples:

- DELETE /api/menu/:id — deletes an item; if DB becomes empty, reset sequence.
- DELETE /api/menu/all — deletes all items and resets sequence (use with caution).

Warnings:

- `DELETE /api/menu/all` is destructive and permanent. Back up DB before use.

Database schema

```
CREATE TABLE menu (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  category TEXT NOT NULL,
  calories INTEGER NOT NULL,
  price REAL NOT NULL,
  ingredients TEXT NOT NULL,
  description TEXT,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  updated_at TEXT DEFAULT CURRENT_TIMESTAMP
)
```

🧪 Testing

PROF

Test scripts and Postman collection provided in the project root:

- `test-calories-api.sh` — bash script with test cases for the calories endpoint.
- `calories-api-postman.json` — Postman collection.

Run the test script:

```
chmod +x test-calories-api.sh
./test-calories-api.sh
```

Or import the Postman collection.

🛠 Tech stack

- Express.js
- better-sqlite3
- dotenv
- cors
- morgan
- nodemon (dev)
- (Optional) @google/generative-ai for Gemini integration (service files reference it)

📄 License

ISC