Restaurant Recommendation Service

A RESTful API that returns restaurant recommendations based on free-text natural language queries.

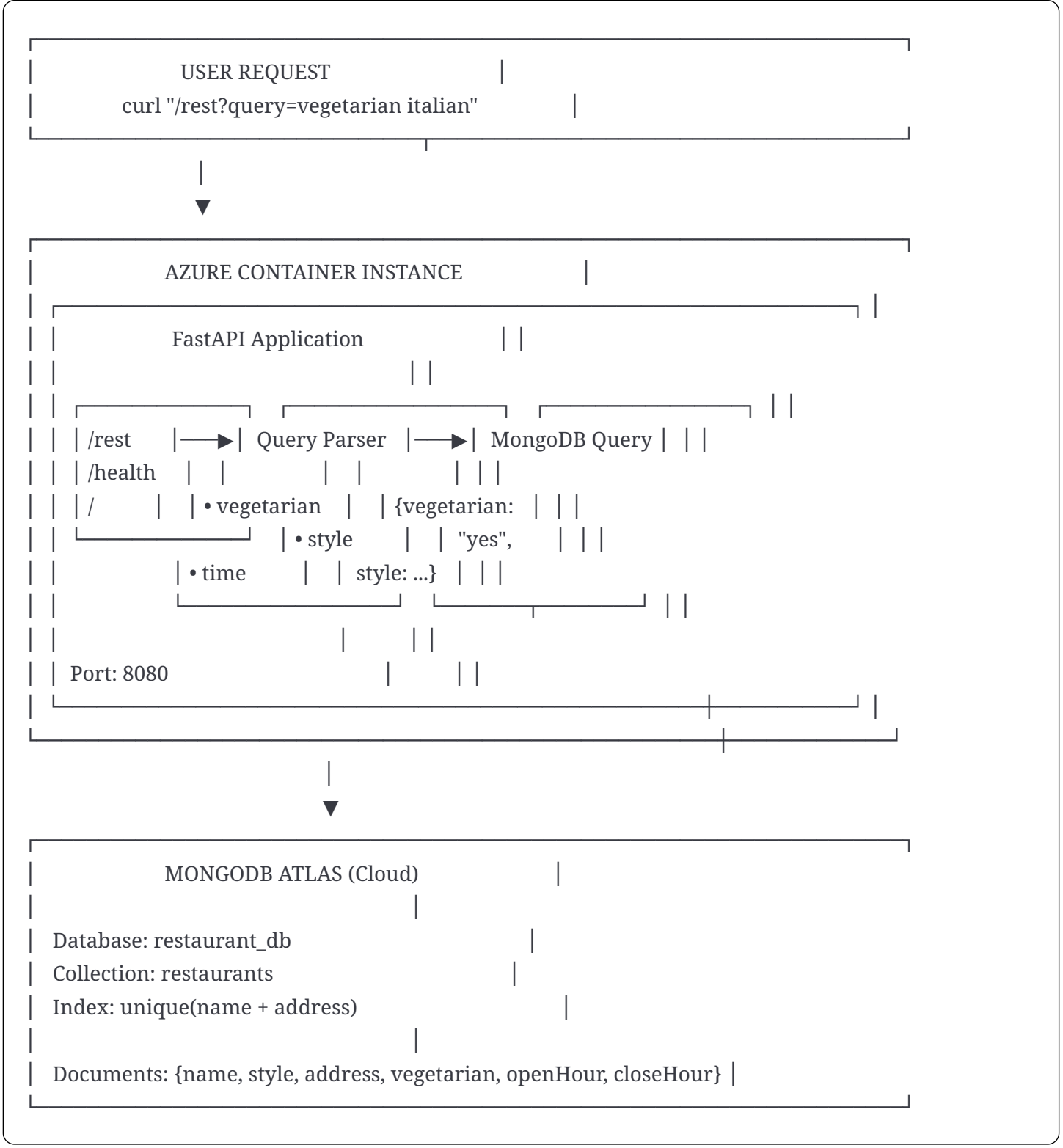
Show Image

Show Image

Table of Contents

---

1. High-Level Architecture



Data Flow

1. **Request** → User sends query to `/rest?query=...`
2. **Parse** → Query parser extracts: vegetarian (yes/no), style, time constraints
3. **Query** → MongoDB filter built and executed
4. **Response** → Results returned as JSON (or message if no results)

---

2. Components Overview

Component	Technology	Description
API Server	FastAPI + Uvicorn	Async REST API on port 8080
Database	MongoDB Atlas	Cloud-hosted document database
Validation	Pydantic v2	Strict schema validation (exactly 6 fields)
Container	Docker	Application packaging
Orchestration	Docker Compose	Local development environment
Infrastructure	Terraform	Azure Container Instances provisioning
CI/CD	GitHub Actions	Automated testing and deployment
Registry	GitHub Container Registry	Docker image storage

Project Structure

```
restaurant-recommendation/
├── app/
│   ├── config.py           # Environment configuration
│   ├── database.py         # MongoDB async operations
│   ├── main.py             # FastAPI application
│   ├── models.py           # Pydantic data models
│   └── query_parser.py     # Free-text query parsing
├── scripts/
│   └── load_restaurants.py  # Data loader script
├── restaurants/
│   └── restaurants.json     # Restaurant data
├── tests/
│   ├── test_api.py         # Model tests
│   └── test_query_parser.py  # Parser tests
├── terraform/
│   ├── main.tf             # Azure resources
│   ├── variables.tf        # Input variables
│   ├── outputs.tf          # Output values
│   └── providers.tf        # Provider config
├── .github/workflows/
│   ├── ci.yml              # Test on PR
│   └── cd.yml              # Deploy on push
├── .env.example
├── .dockerignore
├── docker-compose.yml
├── Dockerfile
├── requirements.txt
└── README.md
```

---

3. Running Locally

Prerequisites

- **Python 3.11**+ OR **Docker**
- **MongoDB Atlas account** (free tier available)

Option A: With Docker (Recommended)

```bash
# 1. Clone repository
git clone https://github.com/USERNAME/restaurant-recommendation.git
cd restaurant-recommendation

# 2. Create .env file
cp .env.example .env
# Edit .env and set MONGODB_URI to your Atlas connection string

# 3. Build and start API
docker-compose up -d api

# 4. Load restaurant data
docker-compose run --rm loader

# 5. Test the API
curl "http://localhost:8080/rest?query=vegetarian%20italian"

# 6. View logs
docker-compose logs -f api

# 7. Stop all services
docker-compose down
```

Option B: Without Docker

```bash
# 1. Clone repository
git clone https://github.com/USERNAME/restaurant-recommendation.git
cd restaurant-recommendation

# 2. Create and activate virtual environment
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate

# 3. Install dependencies
pip install -r requirements.txt

# 4. Set environment variable
export MONGODB_URI="mongodb+srv://user:pass@cluster.mongodb.net/restaurant_db"

# 5. Load restaurant data
python scripts/load_restaurants.py

# 6. Start the API server
uvicorn app.main:app --host 0.0.0.0 --port 8080 --reload

# 7. Test (in another terminal)
curl "http://localhost:8080/rest?query=vegetarian%20italian"
```

Verifying Setup

```bash
# Health check
curl http://localhost:8080/health
# Expected: {"status":"healthy","database":"connected","restaurant_count":10}

# Query test
curl "http://localhost:8080/rest?query=italian"
# Expected: {"restaurantRecommendation":[...]}
```

---

4. Seeding Data

Loading restaurants.json

The `restaurants.json` file contains restaurant data to be loaded into MongoDB.

With Docker:

```bash
docker-compose run --rm loader
```

Without Docker:

```bash
python scripts/load_restaurants.py
```

Loader Behavior

- **Validates** each entry against schema (exactly 6 required fields)
- **Skips** invalid entries (logs error, continues processing)
- **Skips** duplicates (same name + address combination)
- **Reports** summary at end (inserted, skipped, invalid counts)

Exit Codes

Code	Meaning
0	All valid entries loaded successfully
1	Some entries were invalid (but valid ones loaded)
2	Fatal error (file not found, DB connection failed)

Adding New Restaurants

1. Edit `restaurants/restaurants.json`:

```json
{
  "name": "New Restaurant",
  "style": "Italian",
  "address": "123 Main St, City",
  "vegetarian": "yes",
  "openHour": "10:00",
  "closeHour": "22:00"
}
```

2. Run the loader:

```bash
docker-compose run --rm loader
# OR
python scripts/load_restaurants.py
```

Validation Rules

- **Exactly 6 fields**: name, style, address, vegetarian, openHour, closeHour
- **No extra fields allowed** (will be rejected)
- **vegetarian**: Must be "yes" or "no"
- **Time format**: HH:MM or HHMM (normalized to HH:MM)

---

5. Running Tests

Unit Tests

```bash
# Run all tests
pytest tests/ -v

# Run with coverage
pytest tests/ -v --cov=app --cov-report=term-missing

# Run specific test file
pytest tests/test_query_parser.py -v
```

Test Coverage

Module	Tests	Description
`test_query_parser.py`	29	Time parsing, vegetarian detection, style extraction, midnight crossing
`test_api.py`	8	Pydantic model validation, required fields detection
Total	**37**	All passing ✅

Quick Test (No MongoDB)

The tests use mocking and don't require a real MongoDB connection:

```bash
pip install -r requirements.txt
pytest tests/ -v
```

---

6. Infrastructure with Terraform

Directory Structure

```
terraform/
├── providers.tf        # Azure provider configuration
├── variables.tf        # Input variables
├── main.tf             # Resource definitions
├── outputs.tf          # Output values
└── terraform.tfvars.example
```

Prerequisites

- Terraform CLI >= 1.5.0
- Azure subscription
- Azure CLI logged in: `az login`

Provisioning Infrastructure

```bash
cd terraform

# Initialize Terraform
terraform init

# Preview changes
terraform plan \
  -var="docker_image=ghcr.io/USERNAME/restaurant-recommendation:latest" \
  -var="mongodb_uri=$MONGODB_URI"

# Apply changes
terraform apply \
  -var="docker_image=ghcr.io/USERNAME/restaurant-recommendation:latest" \
  -var="mongodb_uri=$MONGODB_URI"

# Get outputs
terraform output api_url
terraform output rest_endpoint
```

Resources Created

Resource	Type	Description
Resource Group	`azurerm_resource_group`	Container for all resources
Container Instance	`azurerm_container_group`	Runs Docker container

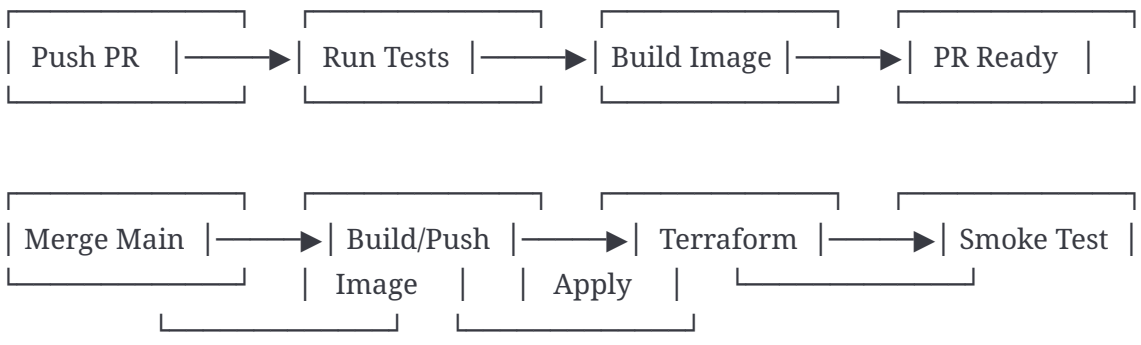
Destroying Infrastructure

```bash
terraform destroy
```

---

7. CI/CD Pipeline

Overview



Workflow Files

ci.yml - Runs on Pull Requests

Triggers: Push to any branch, PR to main

Steps:

1. Checkout code
2. Setup Python 3.11
3. Install dependencies
4. Run pytest
5. Validate restaurants.json
6. Build Docker image (no push)

cd.yml - Runs on Push to Main

Triggers: Push to main, manual dispatch

Steps:

1. Build and push Docker image to GHCR
2. Login to Azure
3. Terraform init
4. Terraform plan
5. Terraform apply
6. Smoke test health endpoint

Manual Trigger

```bash
# Trigger deployment manually via GitHub CLI
gh workflow run cd.yml

# Or via GitHub UI: Actions → CD - Deploy → Run workflow
```

---

8. Environment Variables & Secrets

Local Development (.env)

```bash
# Database (REQUIRED)
MONGODB_URI=mongodb+srv://user:pass@cluster.mongodb.net/restaurant_db

# Optional
DATABASE_NAME=restaurant_db
PORT=8080
LOG_LEVEL=INFO
LOG_FORMAT=json
```

GitHub Secrets

Configure in: Repository → Settings → Secrets and variables → Actions

Secret	Required	Description
`MONGODB_URI`	✅	MongoDB Atlas connection string
`AZURE_CREDENTIALS`	✅	Azure service principal JSON

Azure Service Principal

Create service principal:

```bash
az ad sp create-for-rbac \
  --name "github-restaurant-app" \
  --role Contributor \
  --scopes /subscriptions/<SUBSCRIPTION_ID> \
  --sdk-auth
```

Copy the entire JSON output to `AZURE_CREDENTIALS` secret.

Environment Variables Reference

Variable	Default	Description
`MONGODB_URI`	(required)	MongoDB Atlas connection string
`DATABASE_NAME`	`restaurant_db`	Database name
`COLLECTION_NAME`	`restaurants`	Collection name
`HOST`	`0.0.0.0`	Server bind address
`PORT`	`8080`	Server port
`LOG_LEVEL`	`INFO`	Logging level
`LOG_FORMAT`	`json`	Log format (json/text)
`RESTAURANTS_FILE`	`/app/restaurants/restaurants.json`	Data file path

---

9. API Documentation

Endpoints

Method	Endpoint	Description
GET	`/rest?query=...`	Get restaurant recommendations
GET	`/health`	Health check
GET	`/`	API information

Query Examples

```bash
# Vegetarian Italian (open now)
curl "http://localhost:8080/rest?query=vegetarian%20italian"

# Asian between 10:00 and 18:30
curl "http://localhost:8080/rest?query=asian%20between%2010:00%20and%2018:30"

# Steakhouse closing at 23:00
curl "http://localhost:8080/rest?query=steakhouse%20closes%20at%2023:00"

# Mediterranean vegetarian opening at 11:30
curl "http://localhost:8080/rest?query=mediterranean%20vegetarian%20opening%20at%2011:30"
```

Query Parsing Rules

Feature	Pattern	Result
Vegetarian	"vegetarian" in query	Filter: `{vegetarian: "yes"}`
Non-vegetarian	No "vegetarian"	Filter: `{vegetarian: "no"}`
Style	italian, asian, steakhouse, mediterranean	First match wins
Time range	`between HH:MM and HH:MM`	Must be open for entire range
Opens at	`opens at HH:MM`	Must open at or before time
Closes at	`closes at HH:MM`	Must close at or after time
Default	No time specified	Uses current server time

Response Examples

Success:

```json
{
  "restaurantRecommendation": [
    {
      "name": "Pasta Delight",
      "style": "Italian",
      "address": "Maskit St 35, Herzliya",
      "vegetarian": "yes",
      "openHour": "10:00",
      "closeHour": "22:00"
    }
  ]
}
```

No Results:

```json
{"restaurantRecommendation": "There are no results."}
```

Empty Query:

```json
{"restaurantRecommendation": "query is empty"}
```

Invalid Time:

```json
{"restaurantRecommendation": "Invalid time format: 25:99"}
```

Midnight Crossing:

```json
{"restaurantRecommendation": "Time ranges crossing midnight are not supported: 22:00 to 02:00"}
```

---

10. Assumptions & Limitations

Assumptions

Assumption	Rationale
MongoDB Atlas (cloud)	Company requirement
No "vegetarian" → non-vegetarian only	Company confirmed
First style match wins	Company confirmed
Server local time for "open now"	Company confirmed
Skip invalid JSON entries	Default behavior (not contradicted)
Azure for infrastructure	Spec says "may use Azure"
Port 8080	Matches specification example
Local Terraform state	Company said acceptable

Simplifications

Simplification	Impact
No pagination	All matching restaurants returned
Single container	No horizontal scaling
No authentication	API is public
No rate limiting	Potential for abuse
No caching	Every request hits database

Known Limitations

Limitation	Description	Workaround
Midnight crossing	Hours like 22:00-02:00 not supported	Returns clear error message
Timezone	Uses server local time	Document behavior
Multiple styles	Only first match used	Document behavior
No HTTPS	HTTP only by default	Add Azure Application Gateway for TLS

Security Considerations

Consideration	Status
Non-root container user	✅ Implemented
Sensitive vars hidden	✅ `secure_environment_variables` in Terraform
No secrets in code	✅ All via environment variables
Input validation	✅ Pydantic validation
SQL injection	N/A (NoSQL)

Troubleshooting

MongoDB Connection Failed

```bash
# Check connection string format
echo $MONGODB_URI | grep "mongodb+srv://"

# Test connection
python -c "
from pymongo import MongoClient
client = MongoClient('$MONGODB_URI')
print(client.server_info())
"

# Common issues:
# - IP not whitelisted in Atlas
# - Incorrect username/password
# - Special characters in password need URL encoding
```

No Results Returned

```bash
# Check if data is loaded
curl http://localhost:8080/health
# Look for "restaurant_count"

# Remember: no "vegetarian" in query = only non-vegetarian results
curl "http://localhost:8080/rest?query=vegetarian%20italian"  # With vegetarian
curl "http://localhost:8080/rest?query=italian"               # Without = non-veg only
```

Docker Issues

```bash
# Rebuild image
docker-compose build --no-cache

# View container logs
docker-compose logs -f api

# Enter container
docker-compose exec api bash
```

Terraform Issues

```bash
# Reset state
rm -rf terraform/.terraform terraform/.terraform.lock.hcl

# Re-initialize
cd terraform && terraform init

# Check Azure login
az account show
```

License

MIT License - see LICENSE file for details.

Contributing

1. Fork the repository
2. Create feature branch (`git checkout -b feature/xyz`)
3. Run tests (`pytest tests/ -v`)
4. Commit changes (`git commit -m 'Add xyz'`)
5. Push branch (`git push origin feature/xyz`)
6. Open Pull Request