测试项目说明文档

* 如何部署和运行服务的说明：

一、本地运行（开发环境）

1、设置环境变量

在项目根目录配置 `.env` 文件：

```
PORT=4000
MONGODB_URI=mongodb://localhost:27017/mercury #此处本人配置的是mongodb官方免费测试的账号地址
JWT_SECRET=your_strong_secret_here
NODE_ENV=development             #改为development 开发环境    测试的时候此处改为test
```

2、安装依赖

```
npm install
```

3、启动项目

```
npm run dev
```

二、服务验证（开发环境）

1. 测试REST端点（身份验证）

```
# 注册用户
curl -L -X POST 'http://localhost:4000/api/auth/register' \
  -H 'Content-Type: application/json' \
  --data-raw '{
      "username": "testuser2",
      "email": "testuser2@hotel.com",
      "password": "testuser123",
      "role": "guest"
  }'

# 获取JWT令牌
curl -L -X POST 'http://localhost:4000/api/auth/login' \
  -H 'Content-Type: application/json' \
  --data-raw '{
    "email": "testuser2@hotel.com",
    "password": "testuser123"
  }'
```

2. 测试GraphQL端点（需使用返回的JWT）

```
# 查询预订
curl -L -X POST 'http://localhost:4000/graphql' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: ••••••' \
  -d '{"query":"{ \r\n  reservations { \r\n
    id \r\n
    guestName \r\n
    status\r\n
    arrivalTime\r\n
    tableSize\r\n
    phone\r\n
    email\r\n } \r\n}","variables":{}}'

# 创建预订
curl -L -X POST 'http://localhost:4000/graphql' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: ••••••' \
  --data-raw '{"query":"mutation { \r\n  createReservation(\r\n
      guestName: \"testuser4\", \r\n
      phone: \"12548564591\",\r\n
      email: \"testuser4@hotel.com\", \r\n
      arrivalTime: \"2023-07-21T18:00:00Z\", \r\n
      tableSize: 4\r\n    ) { \r\n
          id \r\n } \r\n}","variables":{}}'

# 更新预订状态
curl -L -X POST 'http://localhost:4000/graphql' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: ••••••' \
  -d '{"query":"mutation { \r\n  updateReservationStatus(\r\n
    id: \"67f75bf40870e36627998518\", \r\n
    status: approved) \r\n    { \r\n
      status \r\n  } \r\n}","variables":{}}'

# 取消预订
curl -L -X POST 'http://localhost:4000/graphql' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: ••••••' \
  -d '{"query":"mutation { \r\n  cancelReservation(\r\n
    id: \"67f75bf40870e36627998518\") { \r\n
      status\r\n    } \r\n}","variables":{}}'
```

* 本项目使用的技术栈及其选择背后的原因：

技术栈	技术特性	选择原因
Node.js	1. 基于V8引擎的JavaScript运行时 2. 非阻塞I/O模型 3. 事件驱动架构	1. 高并发处理：适合餐厅高峰期的密集预定请求 2. 开发效率：熟悉使用JavaScript，效率高 3. 生态丰富：npm拥有超过上百万个可用模块
Express.js	1. 最轻量的Node.js Web框架 2. 中间件架构 3. RESTful路由支持	1. 快速搭建API：能在短时间内构建出完整的REST API 2. 中间件扩展性：方便集成日志、鉴权等模块 3. 社区支持：中文文档完善，问题解决资源丰富
MongoDB	1. 文档型NoSQL数据库 2. 动态Schema设计 3. 水平扩展能力	1. 灵活的数据结构：适应餐厅预订字段的频繁变更需求 2. 高性能读写：优化高频次的小数据操作（如状态更新） 3. 地理空间支持：为未来扩展分店位置查询预留能力
RESTful API		1. 简单易懂：适合基础的身份验证需求； 2. HTTP语义化：利用标准状态码（401未授权等）
GraphQL		1. 灵活查询：员工后台需要组合多种筛选条件 2. 减少请求次数： 单次获取预订列表及关联数据 3. 强类型定义： 通过Schema规范接口格式
JWT认证		1. 无状态认证：适合分布式部署架构 2. 安全可控：可设置短时效令牌增强安全性 3. 跨域支持：方便未来移动端接入
Winston日志		1. 多通道输出： 同时记录到文件和控制台 2. 日志分级：区分DEBUG/INFO/ERROR等级别 3. 可扩展性：可集成Sentry等日志分析平台

---

* 项目结构及其设计理由：

一、项目整体结构概览

```
/hotel-reservation
├── config/      # 环境配置
├── controllers/ # 业务逻辑
├── models/      # 数据模型
├── routes/      # API路由
├── schema/      # GraphQL API路由及其业务逻辑
├── test/        # 测试用例
└── server.js    # 服务入口
```

二、后端结构详解

1. config/ – 配置管理

```
├── db.js          # 数据库连接配置
├── logger.js      # 日志配置
└── test.env       # 测试环境变量文件
```

设计理由：

- **环境隔离**：区分开发/生产环境配置
- **敏感信息保护**：数据库凭证等不进入代码仓库
- **集中管理**：所有配置项单一入口修改

2. models/ – 数据模型

```
├── User.js        # 用户模型
└── Reservation.js # 预订模型
```

设计理由：

  - **数据验证**：通过Schema定义强制字段约束
  - **业务逻辑集中**：如预订状态流转规则
  - **可扩展性**：方便新增字段（如未来添加菜品偏好）

3. routes/ – 路由定义

```
├── auth.js        # 认证相关路由
```

设计理由：

- **关注点分离**：路由只负责HTTP层逻辑
- **中间件链**：实现管道式处理（校验→鉴权→业务）
- **版本控制**：通过 `/api/v1/` 目录结构支持多版本API

4. controllers/ – 业务逻辑

```
├── authController.js       # 实现注册、登录业务逻辑
```

设计理由：

- **可维护性**：修改业务逻辑时无需改动路由配置
- **可测试性**：控制器可以独立进行单元测试
- **代码复用**：多个路由可共用同一个控制器方法

5. schema/ – GraphQL API路由及其业务逻辑

```
├── index.js       # graphQL api路由及graphQL操作
```

设计理由：

- **关注点分离**：Schema定义与实现逻辑分离
- **灵活性**：客户端按需请求数据
- **维护性**：类型系统保障API稳定性
- **扩展性**：轻松添加新类型和解析器

三、测试结构设计

1. test/ – 分层测试

```
├── unit/                # 单元测试
│   └── models.test.js
└── integration/         # 集成测试
    ├── auth.test.js
    └── reservations.test.js
```

测试策略：

- **金字塔模型**：70%单元测试 + 30%集成测试

- **隔离性**：使用内存数据库进行测试
- **覆盖率阈值**：关键业务模块要求100%行覆盖

* 测试报告：

以下为单元测试和集成测试的结果：具体请打开根文件：./coverage/lcov-report/index.html（用浏览器打开）

```
    POST /api/auth/register
      ✓ Should handle validation errors (400) (91ms)
      ✓ should register a new user (519ms)
      ✓ should deny duplicate email registration (492ms)
    POST /api/auth/login
      ✓ email and password do not exist
      ✓ should login with correct credentials (493ms)
      ✓ should reject invalid password (529ms)

  Reservation API
    GraphQL Operations
error: Apr-14-2025 13:41:34:    无效的 Token 格式，应为 Bearer <token>
      ✓ requests without tokens should be rejected
error: Apr-14-2025 13:41:35:    GraphQL operation Authentication failed: 用户不存在
      ✓ reject non-existent users
info: Apr-14-2025 13:41:36:     GraphQL operation success!
      ✓ reservation should be able to be queried by ID (104ms)
info: Apr-14-2025 13:41:37:     GraphQL operation success!
      ✓ reservation should be filtered by status and date (89ms)
info: Apr-14-2025 13:41:38:     GraphQL operation success!
      ✓ should create reservation with authenticated user
info: Apr-14-2025 13:41:39:     GraphQL operation success!
      ✓ should update status if staff (91ms)
info: Apr-14-2025 13:41:40:     GraphQL operation success!
      ✓ Reservation should be cancelled (98ms)

  Data Models
    User Model
      ✓ should hash password before saving (905ms)
      ✓ should require email field
    Reservation Model
      ✓ should default status to requested
      ✓ should validate tableSize


  17 passing (16s)
```

此处，各个部分测试覆盖率设定达到 **80%** 即为通过

```
=============================== Coverage summary ===============================
Statements   : 94.39% ( 101/107 )
Branches     : 83.33% ( 20/24 )
Functions    : 100% ( 12/12 )
Lines        : 95.23% ( 100/105 )
================================================================================
```

All files

94.39% Statements 101/107 **83.33%** Branches 20/24 **100%** Functions 12/12 **95.23%** Lines 100/105

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
config	84.21%	16/19	100%	0/0	100%	2/2	84.21%	16/19
controllers	93.54%	29/31	91.66%	11/12	100%	2/2	93.54%	29/31
models	92.85%	13/14	50%	1/2	100%	2/2	100%	13/13
routes	100%	6/6	100%	0/0	100%	0/0	100%	6/6
schema	100%	37/37	80%	8/10	100%	6/6	100%	36/36

* 其他说明：

整个项目未包含SPA（单页应用程序），原因：

1. 有很久没用过了，做测试的时候，有尝试过，但熟悉和学习在极短的时间没法交付良好的SPA，故放弃；