Technical Document

Online Restaurant Ordering System API

Team: Group 15
Brad, Josias, Kenny, Maddy, Robert

Date: November 30, 2025

Online Restaurant Ordering System API

Table of Contents

1. Architecture Overview

1.1 System Architecture

The SLIME Restaurant Ordering System follows a layered architecture pattern built with FastAPI and SQLAlchemy ORM.

```
+-------------------------------------------------------------+
|                      Client Layer                           |
|              (Web Apps, Mobile Apps, API Clients)           |
+-------------------------+-----------------------------------+
                          | HTTP/REST
+-------------------------v-----------------------------------+
|                    API Layer (FastAPI)                      |
|  +-----------------------------------------------------+  |
|  |                   Routers                            |  |
|  |  customers.py | orders.py | menu_items.py | etc.     |  |
|  +-----------------------------------------------------+  |
|  +-----------------------------------------------------+  |
|  |                   Schemas (Pydantic)                 |  |
|  |  Request/Response validation and serialization       |  |
|  +-----------------------------------------------------+  |
+-------------------------+-----------------------------------+
                          |
+-------------------------v-----------------------------------+
|                  Business Logic Layer                       |
|  +-----------------------------------------------------+  |
|  |                   Controllers                        |  |
|  |  customers.py | orders.py | payments.py | etc.       |  |
|  +-----------------------------------------------------+  |
+-------------------------+-----------------------------------+
                          |
+-------------------------v-----------------------------------+
|                   Data Access Layer                         |
|  +-----------------------------------------------------+  |
|  |              Models (SQLAlchemy ORM)                 |  |
|  |  Customer | Order | MenuItem | Payment | etc.        |  |
|  +-----------------------------------------------------+  |
+-------------------------+-----------------------------------+
                          |
+-------------------------v-----------------------------------+
|                    Database Layer                           |
|                      MySQL/MariaDB                          |
|                 (sandwich_maker_api)                        |
+-------------------------------------------------------------+
```

1.2 Directory Structure

```
project_root/
+-- api/
|   +-- __init__.py
|   +-- main.py                 # Application entry point
|   +-- controllers/            # Business logic layer
|   |   +-- customers.py
|   |   +-- menu_items.py
|   |   +-- orders.py
|   |   +-- order_details.py
|   |   +-- payments.py
|   |   +-- promotions.py
|   |   +-- recipes.py
|   |   +-- resources.py
|   |   +-- reviews.py
|   +-- dependencies/           # Configuration and DB setup
|   |   +-- config.py
|   |   +-- database.py
|   +-- models/                 # SQLAlchemy ORM models
|   |   +-- model_loader.py
|   |   +-- customers.py
|   |   +-- menu_items.py
|   |   +-- orders.py
|   |   +-- order_details.py
|   |   +-- payments.py
|   |   +-- promotions.py
|   |   +-- recipes.py
|   |   +-- resources.py
|   |   +-- reviews.py
|   +-- routers/                # API endpoint definitions
|   |   +-- index.py
|   |   +-- customers.py
|   |   +-- menu_items.py
|   |   +-- orders.py
|   |   +-- order_details.py
|   |   +-- payments.py
|   |   +-- promotions.py
|   |   +-- recipes.py
|   |   +-- resources.py
|   |   +-- reviews.py
|   +-- schemas/                # Pydantic validation schemas
|   |   +-- customers.py
|   |   +-- menu_items.py
|   |   +-- orders.py
|   |   +-- order_details.py
|   |   +-- payments.py
|   |   +-- promotions.py
|   |   +-- recipes.py
|   |   +-- resources.py
|   |   +-- reviews.py
|   +-- sql/
|   |   +-- seed_data.sql       # Sample data
|   +-- tests/
|       +-- test_orders.py
|       +-- test_payments.py
+-- requirements.txt
+-- readme.md
```

1.3 Technology Stack

Component	Technology	Version
Web Framework	FastAPI	Latest
ASGI Server	Uvicorn	Latest

ORM	SQLAlchemy	2.0.x
Database Driver	PyMySQL	Latest
Data Validation	Pydantic	2.x
Testing	Pytest	Latest
Encryption	Cryptography	Latest

1.4 Design Patterns Used

- Repository Pattern - Controllers abstract database operations

- Dependenc

2. Development Environment Setup

2.1 Prerequisites

- Python 3.12+

- MySQL 8.0

Step 1: Clone the Repository

```
git clone <repository-url>
cd group15done
```

Step 2: Create Virtual Environment

```
# Create virtual environment
python -m venv env

# Activate (Linux/Mac)
source env/bin/activate

# Activate (Windows)
.\env\Scripts\activate
```

Step 3: Install Dependencies

```
pip install -r requirements.txt
```

requirements.txt contents:

```
fastapi
uvicorn
sqlalchemy
pymysql
pytest
pytest-mock
httpx
cryptography
```

Step 4: Configure Database

Edit api/dependencies/config.py:

```
class conf:
```

```
    db_host = "localhost"
    db_name = "sandwich_maker_api"
    db_port = 3306
    db_user = "root"
    db_password = "your_password"  # Change this
    app_host = "localhost"
    app_port = 8000
```

Step 5: Create Database

```
CREATE DATABASE sandwich_maker_api CHARACTER SET utf8mb4 COLLATE utf8mb4_unic...
```

Step 6: Run the Application

```
# From project root
python -m api.main

# Or using uvicorn directly
uvicorn api.main:app --reload --host localhost --port 8000
```

Step 7: Seed Sample Data (Optional)

```
mysql -u root -p sandwich_maker_api < api/sql/seed_data.sql
```

2.3 Verify Installation

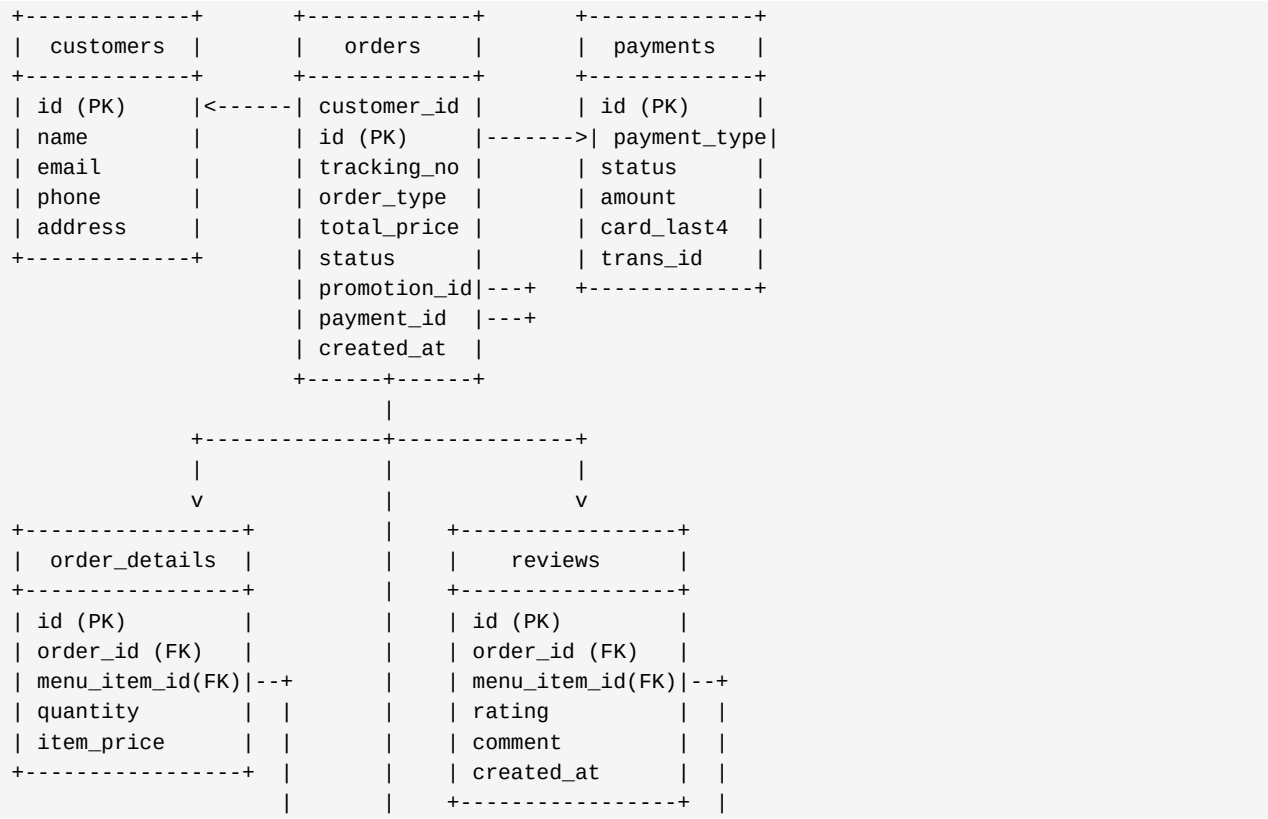
Visit http://localhost:8000 - you should see:

```
{"message": "Welcome to the SLIME API"}
```

Visit http://localhost:8000/docs for interactive Swagger documentation.

3. Database Schema

3.1 Entity Relationship Diagram



3.2 Table Definitions

customers

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY, AUTO_I..	Unique identifier
name	VARCHAR(100)	NOT NULL	Customer full name
email	VARCHAR(120)	NULLABLE	Email address
phone	VARCHAR(20)	NOT NULL	Phone number
address	VARCHAR(255)	NOT NULL	Delivery address

menu_items

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY, AUTO_I..	Unique identifier
name	VARCHAR(100)	NOT NULL	Item name
description	TEXT	NULLABLE	Item description
price	FLOAT	NOT NULL	Item price
calories	INTEGER	NULLABLE	Calorie count
category	VARCHAR(50)	NULLABLE	Category (comma-sep..
is_available	BOOLEAN	DEFAULT TRUE	Availability status

orders

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY, AUTO_I..	Unique identifier

tracking_number	VARCHAR(50)	UNIQUE, NOT NULL	Order tracking code
customer_id	INTEGER	FOREIGN KEY	Reference to custom..
promotion_id	INTEGER	FOREIGN KEY, NULLABLE	Reference to promot..
payment_id	INTEGER	FOREIGN KEY, NULLABLE	Reference to payments
order_type	VARCHAR(20)	NOT NULL	"takeout" or "deliv..
total_price	FLOAT	NOT NULL	Calculated total
status	ENUM	DEFAULT 'RECEIVED'	Order status
created_at	DATETIME	NOT NULL	Timestamp

Order Status Enum Values:

- RECEIVED

- PENDING

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY	Unique identifier
order_id	INTEGER	FOREIGN KEY	Reference to orders
menu_item_id	INTEGER	FOREIGN KEY	Reference to menu_i..
quantity	INTEGER	NOT NULL	Number of items
item_price	FLOAT	NOT NULL	Price at time of or..

payments

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY	Unique identifier
payment_type	VARCHAR(30)	NOT NULL	Card type
status	VARCHAR(30)	NOT NULL	Payment status
masked_card_last4	VARCHAR(4)	NULLABLE	Last 4 digits
transaction_id	VARCHAR(100)	NULLABLE	Transaction reference
amount	FLOAT	NOT NULL	Payment amount

promotions

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY	Unique identifier
code	VARCHAR(50)	UNIQUE, NOT NULL	Promo code
discount_percent	FLOAT	NOT NULL	Discount percentage
expiration_date	DATE	NOT NULL	Expiry date
is_active	BOOLEAN	DEFAULT TRUE	Active status

resources

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY	Unique identifier
name	VARCHAR(100)	NOT NULL	Ingredient name
quantity_available	FLOAT	NOT NULL	Stock quantity
unit	VARCHAR(20)	NOT NULL	Unit of measurement

recipes

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY	Unique identifier
menu_item_id	INTEGER	FOREIGN KEY	Reference to menu_i..
resource_id	INTEGER	FOREIGN KEY	Reference to resour..
required_quantity	FLOAT	NOT NULL	Amount needed per i..

Unique Constraint: (menu_item_id, resource_id)

reviews

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY	Unique identifier
order_id	INTEGER	FOREIGN KEY, UNIQUE	Reference to orders
menu_item_id	INTEGER	FOREIGN KEY	Reference to menu_i..
rating	INTEGER	NOT NULL	1-5 rating
comment	TEXT	NULLABLE	Review text
created_at	DATETIME	NOT NULL	Timestamp

4. API Endpoint Documentation

4.1 Customers Endpoints

Method	Endpoint	Description
POST	`/customers/`	Create a new customer
GET	`/customers/`	Get all customers
GET	`/customers/{id}`	Get customer by ID
PUT	`/customers/{id}`	Update customer
DELETE	`/customers/{id}`	Delete customer

POST /customers/

Request Body:

```
{
  "name": "string (required)",
  "email": "string (optional)",
  "phone": "string (required)",
  "address": "string (required)"
}
```

Response (201 Created):

```
{
  "id": 1,
  "name": "John Smith",
  "email": "john@email.com",
  "phone": "704-555-0101",
  "address": "123 Main Street, Charlotte, NC 28202"
}
```

4.2 Menu Items Endpoints

Method	Endpoint	Description
POST	`/menuitems/`	Create menu item
GET	`/menuitems/`	Get all menu items
GET	`/menuitems/search?categor..	Search by category
GET	`/menuitems/popularity`	Get popularity stats

GET	`/menuitems/{id}`	Get menu item by ID
PUT	`/menuitems/{id}`	Update menu item
DELETE	`/menuitems/{id}`	Delete menu item

GET /menuitems/search?category=vegetarian

Response (200 OK):

```
[
  {
    "id": 3,
    "name": "Vegetarian Pizza",
    "description": "Bell peppers, mushrooms, onions, and olives",
    "price": 13.99,
    "calories": 780,
    "category": "Pizza,Vegetarian",
    "is_available": true
  }
]
```

GET /menuitems/popularity

Response (200 OK):

```
[
  {
    "id": 1,
    "name": "Margherita Pizza",
    "category": "Pizza",
    "total_ordered": 25,
    "avg_rating": 4.8,
    "review_count": 5
  }
]
```

4.3 Orders Endpoints

Method	Endpoint	Description
POST	`/orders/`	Create order (registered u..
POST	`/orders/guest`	Create guest order
GET	`/orders/`	Get all orders
GET	`/orders/date-range?start_..	Orders by date range
GET	`/orders/revenue?start_dat..	Revenue report
GET	`/orders/tracking/{trackin..	Get by tracking number
GET	`/orders/{id}`	Get order by ID
PUT	`/orders/{id}`	Update order
DELETE	`/orders/{id}`	Delete order

POST /orders/

Request Body:

```
{
  "customer_id": 1,
  "order_type": "delivery | takeout",
  "order_items": [
    {"menu_item_id": 1, "quantity": 2},
    {"menu_item_id": 5, "quantity": 1}
  ],
  "promotion_code": "WELCOME10"
}
```

Response (201 Created):

```
{
  "id": 12,
  "tracking_number": "ORD-A1B2C3D4",
  "customer_id": 1,
  "order_type": "delivery",
  "total_price": 35.97,
  "status": "RECEIVED",
  "promotion_id": 1,
  "payment_id": null,
  "created_at": "2025-11-30T14:30:00.000000",
  "order_details": [
    {
      "id": 15,
      "menu_item_id": 1,
      "quantity": 2,
      "item_price": 12.99
    }
  ]
}
```

POST /orders/guest

Request Body:

```
{
  "guest": {
    "name": "Jane Doe",
    "phone": "704-555-1234",
    "email": "jane@email.com",
    "address": "123 Main Street"
  },
  "order_type": "delivery",
  "order_items": [
    {"menu_item_id": 2, "quantity": 1}
  ],
  "promotion_code": null
}
```

GET /orders/revenue

Query Parameters:

- start_date (optional): YYYY-MM-DD format

- end_date (

```
{
  "start_date": "2025-11-28",
  "end_date": "2025-11-30",
  "total_orders": 7,
  "total_revenue": 183.86,
  "daily_breakdown": [
    {
      "date": "2025-11-28",
      "total_orders": 2,
      "total_revenue": 43.97
    },
    {
      "date": "2025-11-29",
      "total_orders": 2,
      "total_revenue": 70.95
    }
  ]
```

```
}
```

PUT /orders/{id}

Request Body:

```
{
  "order_type": "takeout",
  "status": "PREPARING"
}
```

4.4 Order Details Endpoints

Method	Endpoint	Description
POST	`/order_details/`	Create order detail
GET	`/order_details/`	Get all order details
GET	`/order_details/{id}`	Get by ID
PUT	`/order_details/{id}`	Update order detail
DELETE	`/order_details/{id}`	Delete order detail

4.5 Payments Endpoints

Method	Endpoint	Description
GET	`/payments/`	Get all payments
POST	`/payments/orders/{order_i..	Create payment for order
GET	`/payments/orders/{order_i..	Get payment for order
GET	`/payments/{id}`	Get payment by ID
PUT	`/payments/{id}`	Update payment
DELETE	`/payments/{id}`	Delete payment

POST /payments/orders/{order_id}

Request Body:

```
{
  "payment_type": "Credit Card",
  "status": "Processing",
  "amount": 35.97,
  "masked_card_last4": "4242",
  "transaction_id": null
}
```

Response (201 Created):

```
{
  "id": 8,
  "payment_type": "Credit Card",
  "status": "Completed",
  "amount": 35.97,
  "masked_card_last4": "4242",
  "transaction_id": "a1b2c3d4e5"
}
```

Important: The payment amount MUST match the order's total_price exactly.

4.6 Promotions Endpoints

Method	Endpoint	Description
POST	`/promotions/`	Create promotion
GET	`/promotions/`	Get all promotions

GET	`/promotions/{id}`	Get promotion by ID
PUT	`/promotions/{id}`	Update promotion
DELETE	`/promotions/{id}`	Delete promotion

POST /promotions/

Request Body:

```
{
  "code": "NEWYEAR30",
  "discount_percent": 30.0,
  "expiration_date": "2026-01-31",
  "is_active": true
}
```

4.7 Recipes Endpoints

Method	Endpoint	Description
POST	`/recipes/`	Create recipe
GET	`/recipes/`	Get all recipes
GET	`/recipes/{id}`	Get recipe by ID
PUT	`/recipes/{id}`	Update recipe
DELETE	`/recipes/{id}`	Delete recipe

POST /recipes/

Request Body:

```
{
  "menu_item_id": 1,
  "resource_id": 3,
  "required_quantity": 0.25
}
```

4.8 Resources (Ingredients) Endpoints

Method	Endpoint	Description
POST	`/resources/`	Create resource
GET	`/resources/`	Get all resources
GET	`/resources/{id}`	Get resource by ID
PUT	`/resources/{id}`	Update resource
DELETE	`/resources/{id}`	Delete resource

POST /resources/

Request Body:

```
{
  "name": "Olive Oil",
  "quantity_available": 10.0,
  "unit": "liters"
}
```

4.9 Reviews Endpoints

Method	Endpoint	Description
POST	`/reviews/`	Create review
GET	`/reviews/`	Get all reviews

GET	`/reviews/negative`	Get negative reviews (rati..
GET	`/reviews/menu/{menu_item_..	Get reviews for menu item
GET	`/reviews/{id}`	Get review by ID
PUT	`/reviews/{id}`	Update review
DELETE	`/reviews/{id}`	Delete review

POST /reviews/

Request Body:

```
{
  "order_id": 12,
  "menu_item_id": 1,
  "rating": 5,
  "comment": "Excellent pizza!"
}
```

5. Code Examples and Explanations

5.1 Application Entry Point (main.py)

```
import uvicorn
from fastapi import Depends, FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from .routers import index as indexRoute
from .models import model_loader
from .dependencies.config import conf
from .dependencies.database import Base, engine

app = FastAPI(title="SoftDash Linear Input Maintenance Exporter (SLIME)")

# Enable CORS for all origins (development)
origins = ["*"]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Load all models for table creation
model_loader.index()

# Create database tables
Base.metadata.create_all(bind=engine)

# Register all routers
indexRoute.load_routes(app)

@app.get("/")
def root():
    return {"message": "Welcome to the SLIME API"}

if __name__ == "__main__":
    uvicorn.run("api.main:app", host=conf.app_host, port=conf.app_port, reloa...
```

Key Points:

- CORS middleware enables cross-origin requests

- model_loa

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base
from .config import conf
from urllib.parse import quote_plus

# URL-encode password to handle special characters
SQLALCHEMY_DATABASE_URL = f"mysql+pymysql://{conf.db_user}:{quote_plus(conf.d...

engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

def get_db():
    """Dependency that provides database session."""
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Key Points:

- Uses PyMySQL driver for MySQL/MariaDB

- UTF-8 cha

```
from sqlalchemy import Column, Integer, String, Float, ForeignKey, DateTime, ...
from sqlalchemy.orm import relationship
from sqlalchemy.sql import func
import enum
from api.dependencies.database import Base

class OrderStatus(enum.Enum):
    RECEIVED = "RECEIVED"
    PENDING = "PENDING"
    PREPARING = "PREPARING"
    OUT_FOR_DELIVERY = "OUT_FOR_DELIVERY"
    COMPLETED = "COMPLETED"
    CANCELLED = "CANCELLED"

class Order(Base):
    __tablename__ = "orders"

    id = Column(Integer, primary_key=True, index=True)
    tracking_number = Column(String(50), unique=True, nullable=False)
    customer_id = Column(Integer, ForeignKey("customers.id"), nullable=False)
    promotion_id = Column(Integer, ForeignKey("promotions.id"), nullable=True)
    payment_id = Column(Integer, ForeignKey("payments.id"), nullable=True)
    order_type = Column(String(20), nullable=False)
    total_price = Column(Float, nullable=False)
    status = Column(Enum(OrderStatus), default=OrderStatus.RECEIVED, nullable...
    created_at = Column(DateTime(timezone=True), server_default=func.now(), n...

    # Relationships
    customer = relationship("Customer", back_populates="orders")
    promotion = relationship("Promotion", back_populates="orders")
    payment = relationship("Payment")
    order_details = relationship("OrderDetail", back_populates="order", casca...
    review = relationship("Review", back_populates="order", uselist=False)
```

Key Points:

- Python Enum maps to MySQL ENUM type

- cascade="a

```
def create(db: Session, request: schema.OrderCreate):
    """
    Create an order with one or more items.
    - Validates customer exists
    - Validates all menu items exist and are available
    - Automatically computes total_price
    - Auto-generates tracking_number
    - Checks ingredient (resource) inventory and deducts usage
    """
    try:
        # Ensure customer exists
        customer = db.query(customer_model.Customer).filter(
            customer_model.Customer.id == request.customer_id
        ).first()
        if not customer:
            raise HTTPException(status_code=400, detail="Customer does not ex...

        # Compute total price AND calculate ingredient usage
        total_price = 0.0
        ingredient_usage = {}  # resource_id -> quantity needed

        for item in request.order_items:
            # Check menu item exists and is available
            menu_item = db.query(menu_model.MenuItem).filter(
                menu_model.MenuItem.id == item.menu_item_id,
                menu_model.MenuItem.is_available == True,
            ).first()
            if not menu_item:
                raise HTTPException(
                    status_code=400,
                    detail=f"Menu item {item.menu_item_id} not found or unava...
                )

            total_price += menu_item.price * item.quantity

            # Look up recipe rows (ingredients needed)
            recipes = db.query(recipe_model.Recipe).filter(
                recipe_model.Recipe.menu_item_id == item.menu_item_id
            ).all()

            for recipe in recipes:
                needed = recipe.required_quantity * item.quantity
                ingredient_usage[recipe.resource_id] = (
                    ingredient_usage.get(recipe.resource_id, 0.0) + needed
                )

        # Apply promotion if provided
        promotion_id = None
        if request.promotion_code:
            promotion = db.query(promo_model.Promotion).filter(
                promo_model.Promotion.code == request.promotion_code
            ).first()
            # Validate promotion...
            promotion_id = promotion.id
            discount_amount = total_price * (promotion.discount_percent / 100)
            total_price -= discount_amount
```

```
        # Check and deduct inventory
        for resource_id, needed in ingredient_usage.items():
            ingredient = db.query(resource_model.Resource).filter(
                resource_model.Resource.id == resource_id
            ).with_for_update().first()

            if ingredient.quantity_available < needed:
                raise HTTPException(
                    status_code=400,
                    detail=f"Not enough '{ingredient.name}' in stock."
                )
            ingredient.quantity_available -= needed

        # Create the order
        tracking_number = _generate_tracking_number()
        order = order_model.Order(
            tracking_number=tracking_number,
            customer_id=request.customer_id,
            order_type=request.order_type.lower(),
            total_price=total_price,
            status=order_model.OrderStatus.RECEIVED,
            promotion_id=promotion_id
        )
        db.add(order)
        db.flush()

        # Create OrderDetail rows
        for item in request.order_items:
            menu_item = db.query(menu_model.MenuItem).filter(
                menu_model.MenuItem.id == item.menu_item_id
            ).first()
            od = od_model.OrderDetail(
                order_id=order.id,
                menu_item_id=item.menu_item_id,
                quantity=item.quantity,
                item_price=menu_item.price,
            )
            db.add(od)

        db.commit()
        db.refresh(order)
        return order
    except SQLAlchemyError as e:
        db.rollback()
        raise HTTPException(status_code=400, detail=f"Database error: {str(e)}")
```

Key Points:

- .with_for_update() locks rows for concurrent access protection

- db.flush() g

```
from pydantic import BaseModel, Field, field_validator
from typing import List, Optional

class OrderItemCreate(BaseModel):
    menu_item_id: int
    quantity: int = Field(gt=0, description="Quantity must be positive")

class OrderCreate(BaseModel):
    customer_id: int
    order_type: str
```

```
    order_items: List[OrderItemCreate]
    promotion_code: Optional[str] = None

    @field_validator("order_type")
    @classmethod
    def validate_order_type(cls, v: str) -> str:
        allowed = {"takeout", "delivery"}
        if v.lower() not in allowed:
            raise ValueError(f"order_type must be one of {allowed}")
        return v.lower()

    @field_validator("order_items")
    @classmethod
    def validate_items_not_empty(cls, v: List[OrderItemCreate]) -> List[Order...
        if not v:
            raise ValueError("order_items must contain at least one item")
        return v
```

Key Points:

- Field(gt=0) enforces quantity > 0

- Custom va

```
from fastapi import APIRouter, Depends, status, Query
from sqlalchemy.orm import Session
from datetime import date
from typing import Optional
from ..controllers import orders as controller
from ..schemas import orders as schema
from ..dependencies.database import get_db

router = APIRouter(
    tags=["Orders"],
    prefix="/orders"
)

@router.post("/", response_model=schema.OrderRead, status_code=status.HTTP_20...
def create_order(request: schema.OrderCreate, db: Session = Depends(get_db)):
    return controller.create(db=db, request=request)

@router.get("/revenue", response_model=schema.RevenueReport)
def get_revenue_report(
    start_date: Optional[date] = Query(None, description="Start date (YYYY-MM...
    end_date: Optional[date] = Query(None, description="End date (YYYY-MM-DD)"),
    db: Session = Depends(get_db)
):
    return controller.get_revenue_report(db, start_date=start_date, end_date=...
```

Key Points:

- Depends(get_db) injects database session

- response_

6. Testing

6.1 Running Tests

```
# Run all tests
pytest
```

```
# Run with verbose output
pytest -v

# Run specific test file
pytest api/tests/test_orders.py
```

6.2 Test Examples

```
# api/tests/test_orders.py
import pytest
from pydantic import ValidationError
from ..schemas import orders as schema

def test_order_create_valid():
    """Basic happy-path validation for OrderCreate."""
    payload = schema.OrderCreate(
        customer_id=1,
        order_type="takeout",
        order_items=[
            schema.OrderItemCreate(menu_item_id=1, quantity=2),
            schema.OrderItemCreate(menu_item_id=2, quantity=1),
        ],
    )
    assert payload.order_type == "takeout"
    assert len(payload.order_items) == 2
    assert payload.order_items[0].quantity == 2

def test_order_create_invalid_order_type():
    """order_type other than takeout/delivery should fail validation."""
    with pytest.raises(ValidationError):
        schema.OrderCreate(
            customer_id=1,
            order_type="pickup",  # invalid
            order_items=[schema.OrderItemCreate(menu_item_id=1, quantity=1)],
        )

def test_order_create_requires_items():
    """order_items must not be empty."""
    with pytest.raises(ValidationError):
        schema.OrderCreate(
            customer_id=1,
            order_type="delivery",
            order_items=[],  # invalid
        )
```

7. Error Handling

7.1 HTTP Status Codes

Code	Meaning	When Used
200	OK	Successful GET/PUT
201	Created	Successful POST
204	No Content	Successful DELETE
400	Bad Request	Validation error, business..
404	Not Found	Resource doesn't exist
500	Internal Server Error	Unexpected server error

7.2 Error Response Format

```
{
  "detail": "Customer does not exist"
```

```
}
```

7.3 Common Error Scenarios

Error	Cause	Solution
"Customer does not exist"	Invalid customer_id	Create customer first or u..
"Menu item X not found or ..	Item doesn't exist or is_a..	Choose available items
"Not enough 'X' in stock"	Insufficient ingredient in..	Order fewer items or wait ..
"Invalid promotion code"	Code doesn't exist	Check spelling (case-sensi..
"Promotion code has expired"	Expiration date passed	Use a valid code
"Incorrect payment amount"	Amount doesn't match order..	Use exact order total

8. Future Improvements

8.1 Suggested Enhancements

- Authentication & Authorization

- JWT-base

- Database Indexing

- Add index

Appendix A: API Quick Reference

Resource	Create	Read All	Read One	Update	Delete
customers	POST /custom..	GET /custome..	GET /custome..	PUT /custome..	DELETE /cust..
menuitems	POST /menuit..	GET /menuite..	GET /menuite..	PUT /menuite..	DELETE /menu..
orders	POST /orders/	GET /orders/	GET /orders/..	PUT /orders/..	DELETE /orde..
order_details	POST /order_..	GET /order_d..	GET /order_d..	PUT /order_d..	DELETE /orde..
payments	POST /paymen..	GET /payments/	GET /payment..	PUT /payment..	DELETE /paym..
promotions	POST /promot..	GET /promoti..	GET /promoti..	PUT /promoti..	DELETE /prom..
recipes	POST /recipes/	GET /recipes/	GET /recipes..	PUT /recipes..	DELETE /reci..
resources	POST /resour..	GET /resourc..	GET /resourc..	PUT /resourc..	DELETE /reso..
reviews	POST /reviews/	GET /reviews/	GET /reviews..	PUT /reviews..	DELETE /revi..

Appendix B: Environment Variables

For production deployment, consider using environment variables:

```
export DB_HOST=localhost
export DB_NAME=sandwich_maker_api
export DB_PORT=3306
export DB_USER=root
export DB_PASSWORD=secure_password
export APP_HOST=0.0.0.0
export APP_PORT=8000
```

End of Technical Document