Technical Assessment: NutriGuard Platform

🎯 The Product: NutriGuard

NutriGuard is a comprehensive food safety and nutrition management platform that serves hospitals, clinics, and healthcare facilities. Think of it as the "digital nutritionist" that helps healthcare providers monitor patient diets, ensure food safety compliance, and generate personalized nutrition plans.

The Problem We're Solving

Healthcare facilities struggle with:

- **Manual nutrition tracking** for patients with dietary restrictions
- **Food safety compliance** across multiple departments (kitchen, patient care, dietary)
- **Personalized meal planning** for patients with diabetes, heart conditions, allergies
- **Regulatory reporting** for health inspections and nutritional standards
- **Staff training** on food handling and nutritional guidelines

The Solution: NutriGuard Platform

A single platform where:

1. **Dietitians** create and monitor patient meal plans
2. **Kitchen staff** receive safety-compliant recipes and portion guidelines
3. **Nurses** log patient food intake and dietary compliance
4. **Administrators** generate reports for health inspectors and insurance
5. **AI assistant** provides real-time recommendations and alerts

---

🏥 Core User Workflows

Workflow 1: Patient Admission & Diet Planning

Patient admitted → Dietitian reviews medical history → AI suggests meal plan

→ Kitchen receives recipes → Meals prepared → Nurse logs intake → Reports generated

Workflow 2: Food Safety Monitoring

Ingredient delivery → Safety inspection logged → Temperature monitoring

→ AI flags expiration risks → Kitchen alerts → Compliance reports

Workflow 3: Dietary Compliance Tracking

Patient has diabetes → Restricted meal plan created → Meals served

→ Blood sugar tracked → AI correlates food/glucose → Plan adjusted

---

🛠️ Technical Stack Requirements

Mandatory Technologies

- **Backend**: FastAPI (Python 3.11+)
- **AI/LLM**: Google Gemini 2.5 Pro
- **Queue System**: Bull MQ (Redis-based)
- **Database**: PostgreSQL 15+
- **File Storage**: MinIO (S3-compatible)
- **Image Generation**: Nano Banana API
- **Containerization**: Docker & Docker Compose

📱 Core Features to Build

1. Patient Management System

Purpose: Track patients and their dietary requirements

Features:

- Patient registration with medical conditions (diabetes, hypertension, allergies)
- Dietary restriction management (gluten-free, low-sodium, diabetic, etc.)

- Meal plan assignment and tracking
- Progress monitoring and reporting

Key Endpoints:

- `POST /patients` - Register new patient
- `GET /patients/{id}/meal-plan` - Get current meal plan
- `POST /patients/{id}/dietary-restrictions` - Update restrictions
- `GET /patients/{id}/nutrition-history` - View nutrition timeline

2. Meal Planning & Recipe Management

Purpose: Create compliant, personalized meal plans

Features:

- Recipe database with nutritional information
- AI-powered meal plan generation based on medical conditions
- Portion size calculations for different dietary needs
- Recipe modification suggestions (reduce sodium, increase fiber)
- Visual meal cards generated using Nano Banana

Key Endpoints:

- `POST /meal-plans/generate` - AI generates meal plan for patient
- `GET /recipes/search` - Search recipes by dietary filters
- `POST /recipes/{id}/modify` - AI modifies recipe for restrictions
- `POST /meal-plans/{id}/images` - Generate visual meal cards

3. Food Safety & Compliance Tracking

Purpose: Ensure regulatory compliance and food safety

Features:

- Ingredient tracking with expiration dates
- Temperature monitoring logs
- Food safety inspection records
- Compliance report generation
- AI-powered safety risk assessment

Key Endpoints:

- `POST /inventory/ingredients` - Log ingredient deliveries
- `POST /safety/temperature-logs` - Record temperature readings

- `POST /safety/inspections` - Log safety inspections
- `GET /compliance/reports` - Generate regulatory reports

4. Nutrition Monitoring & Analytics

Purpose: Track patient nutrition intake and outcomes

Features:

- Daily meal intake logging
- Nutritional analysis and reporting
- Health outcome correlation (blood sugar vs meals)
- Alert system for dietary violations
- Progress dashboards

Key Endpoints:

- `POST /nutrition/intake` - Log patient meal consumption
- `GET /nutrition/analytics/{patient_id}` - Get nutrition insights
- `POST /nutrition/analyze-correlation` - AI analyzes health outcomes
- `GET /alerts/dietary-violations` - Get compliance alerts

5. AI-Powered Recommendations

Purpose: Provide intelligent insights and recommendations

Features:

- Personalized meal recommendations
- Food safety risk predictions
- Nutritional deficiency alerts
- Recipe optimization suggestions
- Health outcome predictions

Key Endpoints:

- `POST /ai/recommend-meals` - Get meal recommendations
- `POST /ai/assess-risks` - Assess food safety risks
- `POST /ai/predict-outcomes` - Predict health outcomes
- `POST /ai/optimize-recipe` - Optimize recipe for health goals

🗄️ Database Schema Requirements

Core Entities

Patients → Medical conditions, dietary restrictions, meal plans

Recipes → Ingredients, nutritional data, preparation instructions

Meal Plans → Patient assignments, scheduled meals, portions

Ingredients → Inventory tracking, nutritional values, safety data

Safety Logs → Temperature records, inspections, compliance data

Nutrition Entries → Patient intake records, meal consumption

Staff → Role-based access, certifications, activity logs

Key Relationships

- Patients have many Meal Plans
- Meal Plans contain many Recipes
- Recipes use many Ingredients
- Patients have many Nutrition Entries
- Safety Logs track Ingredients and Facilities

🎯 Required API Functionality

Authentication & Authorization

- **Healthcare role-based access**: Admin, Dietitian, Kitchen Staff, Nurse
- **Patient data protection**: HIPAA-compliant access controls
- **Session management**: Secure JWT with role permissions

Background Processing

- **Meal plan generation**: AI-powered planning via background jobs
- **Report generation**: Compliance reports for inspectors
- **Alert processing**: Real-time notifications for safety violations
- **Data import**: Bulk patient/recipe imports from CSV

File Management

- **Recipe images**: Generated meal visuals using Nano Banana
- **Compliance documents**: PDF reports stored in MinIO
- **Patient photos**: Secure image storage with access controls

---

🧪 Testing Requirements

Critical Test Scenarios

1. **Patient meal plan generation** with multiple dietary restrictions
2. **Food safety violation detection** and alert system
3. **Nutritional compliance tracking** for diabetic patients
4. **Role-based access control** across different user types
5. **Background job processing** for report generation
6. **AI service integration** with proper error handling

📊 Success Metrics

Minimum Viable Product

- [ ] Patient can be registered with dietary restrictions
- [ ] AI can generate basic meal plan for patient conditions
- [ ] Kitchen staff can view assigned recipes and portions
- [ ] Nutrition intake can be logged and tracked
- [ ] Basic compliance reports can be generated
- [ ] All services run in Docker environment

Excellent Implementation

- [ ] Complex dietary restrictions properly handled
- [ ] Real-time alerts for safety violations
- [ ] Advanced analytics correlating food intake with health outcomes
- [ ] Comprehensive role-based access control
- [ ] High-performance API with proper caching
- [ ] Production-ready error handling and logging

🎯 Evaluation Focus

Backend Architecture (40%)

- **Domain modeling**: How well you structure the healthcare/nutrition domain
- **API design**: RESTful endpoints that make sense for the workflows
- **Data integrity**: Proper handling of sensitive patient data

AI Integration (25%)

- **Gemini usage**: Intelligent meal planning and safety risk assessment
- **Context management**: Proper prompting for healthcare-specific recommendations
- **Error handling**: Graceful degradation when AI services fail

DevOps & Scalability (20%)

- **Container orchestration**: Properly networked services
- **Background processing**: Reliable job queuing for reports and alerts
- **Data persistence**: Proper volume management and backups

Healthcare Compliance (15%)

- **Security**: Patient data protection and access controls
- **Documentation**: Clear API docs for healthcare professionals
- **Testing**: Comprehensive coverage for critical healthcare workflows

---

💡 Key Design Decisions to Make

1. **How do you model complex dietary restrictions?** (diabetes + gluten-free + low-sodium)
2. **How do you ensure meal plans meet nutritional requirements?** (AI validation vs rules engine)
3. **How do you handle food safety alerts in real-time?** (WebSockets, polling, notifications)
4. **How do you structure role-based permissions?** (hierarchical, resource-based, attribute-based)
5. **How do you correlate nutrition data with health outcomes?** (data modeling, analytics approach)

📋 Submission Requirements

Deliverables

1. **Working NutriGuard platform** with all core features
2. **Docker environment** that starts with `docker-compose up`
3. **Sample data** showing realistic healthcare scenarios
4. **API documentation** with healthcare workflow examples
5. **Test suite** covering critical healthcare use cases

Documentation Must Include

- **Product overview**: What NutriGuard does and why
- **User workflows**: How different roles use the system
- **Architecture decisions**: Why you chose specific approaches
- **Security considerations**: How you protect patient data
- **Deployment guide**: How to set up in production

🚀 The Big Picture

You're building a platform that could genuinely improve patient outcomes in healthcare facilities. The code you write should reflect the critical nature of healthcare applications - reliable, secure, and user-friendly for medical professionals who are focused on patient care, not technology.

Think like a healthcare provider: What would make their daily workflows easier? How can technology reduce medical errors and improve patient nutrition?

Good luck building something that matters! 🏥