Technical Assessment

Challenge Overview

Build a **Distributed Task Management API** demonstrating enterprise-level architecture, cloud-native patterns, and modern development practices.

Core Requirements

1. RESTful API Endpoints

Authentication (via Supabase)

- `POST /api/auth/register` - User registration
- `POST /api/auth/login` - User login
- `GET /api/auth/me` - Get current user profile

Projects

- `GET /api/projects` - List user's projects (paginated, filterable)
- `POST /api/projects` - Create new project
- `GET /api/projects/{id}` - Get project details with tasks
- `PUT /api/projects/{id}` - Update project
- `DELETE /api/projects/{id}` - Delete project
- `GET /api/projects/{id}/analytics` - Project statistics

Tasks

- `GET /api/projects/{projectId}/tasks` - List tasks (filterable by status, sortable)
- `POST /api/projects/{projectId}/tasks` - Create task
- `PUT /api/tasks/{id}` - Update task
- `PATCH /api/tasks/{id}/status` - Update task status only
- `DELETE /api/tasks/{id}` - Delete task

- `POST /api/tasks/{id}/assign` - Assign task to user

Health & Monitoring

- `GET /health` - Health check endpoint

2. Data Design

You are responsible for designing:

- Database schema and collections for MongoDB
- Data relationships and references
- Required fields and validation rules
- Indexing strategy

Consider:

- What fields are essential for projects and tasks?
- How will you handle user relationships?
- What metadata is needed for audit trails?
- How will you implement soft deletes?
- What status/priority enums make sense?

3. Technical Stack Requirements

Required

- ✅ **.NET 7+** Web API
- ✅ **MongoDB** with official C# driver (not Entity Framework)
- ✅ **Supabase** for authentication (JWT validation)
- ✅ **Docker** & **Docker Compose** for containerization
- ✅ **Redis** for caching (frequently accessed projects/tasks)
- ✅ **Serilog** for structured logging
- ✅ **Health Checks** for monitoring

Architecture Patterns

- Repository pattern for data access
- CQRS-lite (separate read/write models where beneficial)
- Middleware for cross-cutting concerns
- Request/Response DTOs (never expose domain models)
- Result pattern for error handling (no exceptions for business logic)

4. Docker Setup

Required Containers

- API container (.NET application)
- MongoDB container
- Redis container
- (Optional) Nginx reverse proxy

Docker Compose Features

- Environment variable configuration
- Volume mounts for data persistence
- Health checks for all services
- Network isolation
- Development and production profiles

5. Advanced Features (Choose at least 3)

Performance & Scalability

- ❑ Implement Redis caching with cache invalidation strategy
- ❑ Add response compression
- ❑ Implement database indexing strategy
- ❑ Add rate limiting per user/IP

Monitoring & Observability

- ❑ Structured logging with correlation IDs

- ❏ Request/response logging middleware
- ❏ Performance monitoring for slow queries

Resilience & Reliability

- ❏ Circuit breaker pattern for external dependencies
- ❏ Retry policies with exponential backoff
- ❏ Graceful shutdown handling
- ❏ Database transaction management

Security

- ❏ Input validation and sanitization
- ❏ Request size limits
- ❏ CORS configuration
- ❏ Security headers middleware

API Design

- ❏ API versioning strategy
- ❏ HATEOAS links in responses
- ❏ ETags for caching
- ❏ WebSocket endpoint for real-time task updates

Submission Requirements

1. GitHub Repository (Public)

Your repository must include:

- Complete source code with clear project structure
- Docker configuration files (Dockerfile, docker-compose.yml)
- Postman collection JSON file
- Environment configuration examples (.env.example)
- README.md

- ARCHITECTURE.md
- Any test files

2. Postman Collection

Must Include

- Complete API collection with all endpoints
- Pre-request scripts for authentication
- Environment variables for local and production
- Test scripts for key endpoints
- Example requests with sample data
- Documentation for each endpoint

Export as Postman Collection v2.1 format

3. Documentation

README.md must include:

- Project overview and objectives
- Tech stack justification
- Local setup instructions (step-by-step)
- Docker setup commands
- How to run tests
- Environment variables explanation
- Known limitations

ARCHITECTURE.md must include:

- System architecture diagram (can be ASCII or image)
- Database schema design with justification
- Design patterns used and why
- Caching strategy

- Authentication flow
- Error handling approach
- Scalability considerations

Setup Testing

We will test your submission by:

1. Cloning your repository
2. Running `docker-compose up` (should work out of the box)
3. Importing Postman collection
4. Running automated tests against local and production APIs
5. Reviewing code with AI analysis tools
6. Reading architecture documentation

Your API must be accessible at:

- Local: `http://localhost:5000`
- Production: Deploy to a free tier service (Railway, Render, or Azure free tier)

Bonus Points

- CI/CD pipeline (GitHub Actions)
- Database migrations strategy
- API versioning implemented
- Comprehensive integration tests
- Real-time features (SignalR/WebSockets)
- Message queue integration (RabbitMQ/Azure Service Bus)
- Distributed tracing (OpenTelemetry)
- Performance benchmarks

What We're Really Assessing

Beyond the code, we want to see:

- **Design Thinking**: How you approach data modeling and system architecture
- **Pragmatism**: Can you balance perfection with delivery?
- **Production Mindset**: Do you think about monitoring, errors, and edge cases?
- **Communication**: Can you explain complex decisions clearly?
- **Modern Practices**: Are you comfortable with cloud-native patterns?
- **Trade-off Analysis**: Do you understand when to use what?

Questions During Development?

Document your assumptions in ARCHITECTURE.md. We value well-reasoned decisions over perfect implementations.

Timeline:

- Receive challenge: Day 0
- Submit by: Day 3 (72 hours)
- We'll review within 5 business days
- Top candidates advance to Stage 3 technical deep-dive

Good luck! We're excited to see how you architect solutions.