MYCURE

Junior Backend Developer Skills Assessment

Instructions

- Total time: 3 hours
- Choose your preferred backend language (Javascript)
- Use these stack: bun (js runtime), hono (api framework)
- You may use documentation and search engines
- Focus on clean, production-ready code
- Include comments explaining your reasoning

Section 1: API Design & Implementation (45 minutes)

Task 1.1: RESTful API Design

Design and implement a simple **Task Management API** with the following requirements:

Endpoints to implement:

- `GET /api/tasks` - List all tasks with pagination
- `GET /api/tasks/:id` - Get specific task
- `POST /api/tasks` - Create new task
- `PUT /api/tasks/:id` - Update task
- `DELETE /api/tasks/:id` - Delete task

Task Model:

```
{
  "id": "string",
  "title": "string",
  "description": "string",
  "status": "pending" | "in_progress" | "completed",
  "priority": "low" | "medium" | "high",
  "created_at": "datetime",
  "updated_at": "datetime",
  "due_date": "datetime (optional)"
}
```

Requirements:

- Implement proper HTTP status codes
- Add input validation
- Include error handling
- Add request/response logging middleware

Task 1.2: Query Parameters

Extend the `GET /api/tasks` endpoint to support:

- Filtering by status and priority
- Sorting by created_at, due_date, or priority
- Pagination (page, limit)
- Search by title/description

Example:
`/api/tasks?status=pending&priority=high&sort=due_date&page=1&limit=10&search=urgent`

Section 2: Database Operations (40 minutes)

Task 2.1: Database Schema Design

Design a database schema for a **Library Management System** with:

Entities:

- **Books:** id, title, author, isbn, publication_year, available_copies, total_copies
- **Users:** id, name, email, phone, membership_date, is_active
- **Borrowings:** id, user_id, book_id, borrowed_date, due_date, returned_date, fine_amount

Requirements:

- Write CREATE TABLE statements
- Define appropriate primary keys, foreign keys, and indexes
- Include necessary constraints

Task 2.2: Complex Queries

Write SQL queries for:

1. **Overdue Books:** List all overdue borrowings with user and book details
2. **Popular Books:** Top 5 most borrowed books in the last 6 months
3. **User Statistics:** For each active user, show total books borrowed and current outstanding books
4. **Revenue Report:** Calculate total fines collected per month for the current year

Task 2.3: Database Integration

Implement database operations in your chosen backend language:

- Connection management
- Prepared statements/parameterized queries
- Transaction handling for the borrowing process
- Connection pooling considerations

Section 3: Authentication & Authorization (30 minutes)

Task 3.1: Authentication System

Implement a JWT-based authentication system:

Required endpoints:

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

Requirements:

- Hash passwords securely
- Generate and validate JWT tokens
- Implement token refresh mechanism
- Add middleware to protect routes

Task 3.2: Authorization

Implement role-based access control:

- **Admin:** Can manage all tasks
- **User:** Can only manage their own tasks
- **Guest:** Read-only access

Show how you would modify the Task API from Section 1 to implement these permissions.

Section 4: Error Handling & Validation (25 minutes)

Task 4.1: Global Error Handler

Implement a centralized error handling system that:

- Catches and formats different types of errors
- Returns consistent error responses
- Logs errors appropriately
- Handles validation errors, database errors, and unexpected errors

Expected error response format:

```
{
  "success": false,
  "error": {
```

```
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data",
    "details": [
      {
        "field": "email",
        "message": "Invalid email format"
      }
    ]
  },
  "timestamp": "2024-01-15T10:30:00Z"
}
```

Task 4.2: Input Validation

Create a validation system for the Task API that validates:

- Required fields
- Data types
- String lengths
- Date formats
- Enum values (status, priority)
- Custom business rules

Section 5: Testing (30 minutes)

Task 5.1: Unit Tests

Write unit tests for:

- Task creation with valid/invalid data
- User authentication
- Password hashing utility
- Database query functions

Task 5.2: Integration Tests

Write integration tests for:

- Complete task CRUD operations
- User registration and login flow
- Protected route access

Include:

- Test setup and teardown
- Mock data creation
- Database test isolation
- Assert proper status codes and response formats

Section 6: Performance & Optimization (20 minutes)

Task 6.1: Caching Strategy

Design a caching strategy for the Task Management API:

- Which endpoints would benefit from caching?
- What caching mechanisms would you use? (Redis, in-memory, etc.)
- How would you handle cache invalidation?
- Implement caching for the `GET /api/tasks` endpoint

Task 6.2: Database Optimization

For the Library Management System:

- Identify potential performance bottlenecks
- Suggest database indexes
- Propose query optimizations
- Discuss N+1 query problem and solutions

Section 7: Security Considerations (20 minutes)

Task 7.1: Security Vulnerabilities

Identify and explain how to prevent:

- SQL Injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Rate limiting and DDoS protection
- Password security best practices

Task 7.2: API Security

Implement security measures for your Task API:

- Rate limiting middleware
- Input sanitization

- CORS configuration
- Security headers
- Request size limits

Section 8: System Design & Architecture (30 minutes)

Task 8.1: Microservices Architecture

Design a microservices architecture for an **E-commerce Platform** with:

- User Service
- Product Service
- Order Service
- Payment Service
- Inventory Service

Address:

- Service boundaries and responsibilities
- Inter-service communication
- Data consistency strategies
- Error handling between services

Task 8.2: Scalability Planning

For the Task Management API:

- How would you handle 10,000 concurrent users?
- Database scaling strategies
- Load balancing considerations
- Monitoring and logging requirements

Bonus Section: Real-World Scenario (Optional - 20 minutes)

You're working on a production API that suddenly starts receiving 500 errors for 20% of requests. The errors are related to database timeouts.

Questions:

1. What steps would you take to investigate?
2. What immediate actions would you implement?
3. How would you prevent this in the future?

4. What monitoring/alerting would you set up?

Submission Requirements

Deliverables:

1. **Code:** Complete implementation of core features
2. **Documentation:** API documentation (README with endpoints, setup instructions)
3. **Tests:** Test files with examples
4. **Architecture:** Diagrams or descriptions of your system design
5. **Deployment:** Basic deployment configuration (Docker, environment variables)

Evaluation Criteria:

- **Code Quality:** Clean, readable, maintainable code
- **Functionality:** Working implementation meeting requirements
- **Security:** Proper security practices implemented
- **Testing:** Comprehensive test coverage
- **Documentation:** Clear documentation and setup instructions
- **Problem Solving:** Thoughtful approach to complex problems
- **Best Practices:** Following industry standards and conventions