

BACKEND DEVELOPMENT

Best Practices

Purpose

This document outlines the minimum standards and best practices for backend development. The goal is to improve code quality, security, maintainability, reliability, and consistency across all backend services and APIs.

1. Validate All Input

Never trust data received from clients, APIs, forms, or external systems.

Guidelines

- Validate required fields.
- Validate data types and formats.
- Sanitize user input where necessary.
- Reject unexpected parameters.
- Enforce business rules at the backend.

Why: Client-side validation can be bypassed. Backend validation protects data integrity and application security.

2. Use Parameterized Queries

Always use prepared statements or parameterized queries when interacting with databases.

Guidelines

- Never concatenate user input into SQL statements.
- Use query builders or ORM parameter binding where available.

Why: This prevents SQL injection vulnerabilities and improves query safety.

3. Centralize Error Handling

Application errors should be handled consistently across the entire system.

Guidelines

- Implement global exception handling.
- Return standardized error responses.
- Log all exceptions centrally.
- Avoid duplicate error handling logic in controllers.

Why: Centralized handling simplifies maintenance and improves troubleshooting.

4. Implement Proper Authentication and Authorization

Guidelines

- Authenticate every protected request.
- Verify permissions on the server.
- Never rely on frontend permission checks.

Why: Security controls must always be enforced on the backend. The backend must assume the UI is untrusted and can be modified, or bypassed.

5. Keep Business Logic Out of Controllers

Controllers should remain thin and focused.

Controller Responsibilities:

- Receive requests
- Validate input
- Call services
- Return responses

Avoid

- Complex calculations
- Database-heavy operations
- Business workflows

Why: Separating business logic improves testability, readability, and maintainability.

6. Log Important Events

Logging is essential for auditing and troubleshooting.

Log Events Such As:

- User logins
- Failed login attempts
- Data modifications
- API failures
- System exceptions

Why: Logs provide visibility into application behavior and simplify incident investigations.

7. Use Transactions for Critical Operations

Use database transactions whenever multiple operations must succeed or fail together. Example: Multi-table updates

Why: Transactions prevent partial updates and maintain data consistency.

8. Design Consistent API Responses

All APIs should follow a consistent response structure.

Why: Consistency simplifies frontend integration and debugging.

9. Protect Sensitive Data

Sensitive information must never be exposed unnecessarily.

Never Expose

- Password hashes
- API secrets
- Authentication tokens
- Database credentials

Guidelines

- Exclude secrets from what is returned to users.
- Follow the principle of least privilege.

Why: Protecting sensitive information reduces security risks and compliance issues.

10. Never Expose Internal Errors to the UI

Backend exceptions and technical details **should never be** returned directly to users.

Never Return

- Stack traces
- SQL errors
- File paths
- Database schema details
- Internal exception messages

Instead

- Log detailed errors internally.
- Return user-friendly messages.
- Include a request or correlation ID where appropriate.

Example:

Bad

```
{  
  "error": "SQLSTATE [23000]: Integrity constraint violation"  
}
```

Good

```
{  
  "status": "error",  
  "code": 500,  
  "message": "An unexpected error occurred. Please try again later."  
}
```

Why: Exposing internal errors creates security risks and provides little value to end users.

Principle: Log everything internally. Expose only what users need to know.

Development Priorities

When making technical decisions, prioritize the following:

1. Security
2. Data Integrity
3. Reliability
4. Maintainability
5. Observability
6. Performance Optimization

“A slow system can usually be optimized. A system that compromises security or corrupts business data can have far greater consequences.”

