Backend Architecture: APIs, Microservices, and Production Workflows

Subtitle: Understand how real backend systems are structured, how APIs work in production, and why modern applications are split into services.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Most beginners think backend development is:
• writing APIs
• connecting database
• returning responses

That is only a small part.

Real backend systems in production are:
• distributed
• modular
• monitored
• scalable
• fault-tolerant

This PDF explains:
• APIs in real systems
• microservices architecture
• production workflows used in companies

Chapter 1: What Backend Really Means

Backend is not just code.

It is:

the system that manages data, logic, and communication between services at scale

Core responsibilities:

• authentication
• data processing
• business logic
• database communication
• system coordination

Chapter 2: APIs (Application Programming Interface)

API is how systems communicate.

Simple definition:

API = a bridge between frontend and backend

Example:

Frontend sends request:
• “Get user profile”

Backend responds:
• user data

Types of APIs:

1. REST API

Most common type
Uses HTTP methods

2. GraphQL

Client asks only what it needs

3. WebSockets

Real-time communication

Chapter 3: How APIs Work in Real Systems

Request Flow:

1. user action (frontend)
2. API request sent
3. backend processes logic
4. database query (if needed)
5. response returned

Important Insight:

API is not just a function
It is a controlled entry point to system logic

Chapter 4: Microservices Architecture

In beginner projects:
• everything is in one backend

This is called:
• monolithic architecture

Problem with monolith:

• hard to scale
• hard to maintain
• risky updates

Solution: Microservices

System is split into small services:

• user service
• payment service
• notification service
• post service

Each service:

• works independently
• has its own database (sometimes)
• communicates via APIs

Chapter 5: Why Big Companies Use Microservices

Because it allows:

1. Independent scaling

Only heavy service scales

2. Faster development

Teams work separately

3. Better reliability

One failure doesn’t break everything

Chapter 6: Real Production Backend Workflow

Backend is not just request-response.

It includes:

1. Request handling

API receives request

2. Validation

Check if data is correct

3. Business logic

Apply rules

4. Database interaction

Store or fetch data

5. Response generation

Send output

Chapter 7: Middleware Concept

Middleware sits between request and response.

It handles:

• authentication
• logging
• error handling
• rate limiting

Example:

Request → Middleware → API → Response

Chapter 8: Authentication in Real Systems

Authentication is not just login.

It includes:

• session management
• token generation
• permission checks

Common method:

JWT (JSON Web Token)

Chapter 9: Error Handling in Production

In real systems:
• errors are expected

Types of errors:

• server crash
• invalid input
• database failure

Solution:

• structured error handling
• fallback systems
• logging

Chapter 10: Logging and Monitoring

Production systems must track everything.

Logs include:

• user actions
• system errors
• API requests
• performance data

Monitoring tools:

• system health tracking
• error alerts
• performance dashboards

Chapter 11: API Versioning

APIs evolve over time.

Problem:

New changes can break old apps

Solution:

• version APIs
Example:
• /api/v1/users
• /api/v2/users

Chapter 12: Real Backend Thinking

Backend developers think about:

• system stability
• scalability
• failure handling
• performance

Not just:

• writing endpoints

Chapter 13: Monolith vs Microservices

Monolith:

• simple
• easy to build
• hard to scale

Microservices:

- scalable
- complex
- production standard

Chapter 14: When to Use What

Use monolith when:

- small projects
- early stage apps

Use microservices when:

- large traffic systems
- multiple teams
- complex apps

Chapter 15: Key Engineering Principles

1. Everything fails eventually

Systems must handle failure

2. Separation of concerns

Each service has a purpose

3. Scalability is planned

Not added later

Key Takeaways

- APIs are system communication layers
- Backend is more than just endpoints
- Microservices split systems into independent parts
- Middleware handles cross-cutting logic
- Logging and monitoring are essential in production
- Authentication is a full system, not just login
- Real systems are designed for failure
- Architecture matters more than code

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app