Backend Development

Ritwik

Date: 2025.06.30

CONTENTS

01

What is Backend Development?

What is Backend Development?

1. The server-side of an application.
2. Handles the logic, data processing, and core functions you don't see.
3. The "engine" that powers websites, mobile apps, and digital systems.

02

Why is it Important?

Why is it Important?

Logic & Functionality: It's the brain of the application.

Data Management: Securely stores, manages, and retrieves all user and application data.

Performance & Scalability: Ensures the application is fast, reliable, and can handle growth.

Security: Protects user data and the system from attacks.

Backend vs. Frontend

Backend (Server-Side)	Frontend (Client-Side)
How it works	How it looks
Logic, Databases, APIs	UI/UX, HTML, CSS, JavaScript
Runs on a server	Runs in the user's browser
The "kitchen" of a restaurant	The "dining room" and menu

Roles and Responsibilities

Building and maintaining server-side application logic.

Designing and managing databases.

Creating and documenting robust APIs.

Implementing authentication, authorization, and security measures.

Optimizing for speed, scalability, and reliability.

05

Programming Languages for Backend

Programming Languages for Backend

1. The fundamental tool for building the backend.
2. Choice depends on the project, team, and performance needs.
3. No single "best" language, only the "right tool for the job."

Popular Language: Python

Key Traits

Clean syntax, versatile, great for rapid development.

Common Frameworks

Django, Flask.

Ideal For

Startups, AI/ML applications, data science, general-purpose backend.

Popular Language: Java

1 **Key Traits**

High performance, stable, robust, massive ecosystem.

2 **Common Frameworks**

Spring Boot, Hibernate.

3 **Ideal For**

Large enterprises, banking systems, Android apps, mission-critical applications.

Popular Language: JavaScript (Node.js)

Key Traits

Asynchronous, non-blocking, fast for I/O operations.

Common Frameworks

Express.js, NestJS.

Ideal For

Real-time applications (chat, games), microservices, data-intensive apps.

09

Other Key Languages

Other Key Languages

1 **PHP**

Powers a huge portion of the web (WordPress). Modern frameworks like Laravel.

2 **Ruby**

Famous for developer happiness and rapid development with the Ruby on Rails framework.

3 **Go**

Developed by Google. Excellent for high-concurrency and high-performance systems.

4 **C/#**

Strong in the Microsoft ecosystem with .NET. Used for enterprise and game development.

Language Use Cases at a Glance

Startups, AI/ML

Python

Large Enterprises

Java

Real-time Apps

Node.js

Content Management

PHP

Rapid Prototyping

Ruby on Rails

11

Web Servers and How They Work

Web Servers and How They Work

Software that listens for incoming requests from clients (browsers).

Processes requests by running backend code.

Sends back responses (data, web pages, error messages) over HTTP/HTTPS.

The HTTP/HTTPS Request Lifecycle

Client Sends Request: Uses methods like:

- GET: Retrieve data.
- POST: Create new data.
- PUT/PATCH: Update existing data.
- DELETE: Remove data.

Server Processes Request: Executes application logic.

Server Returns Response: Includes a status code (e.g., 200 OK, 404 Not Found) and data.

14

Databases: The System's Memory

Databases: The System's Memory

The core of any data-driven application.

Responsible for the persistent storage, organization, and retrieval of data.

The backend's most critical component for state management.

15

Types of Databases: SQL vs. NoSQL

Types of Databases: SQL vs. NoSQL

Relational (SQL)	Non-Relational (NoSQL)
Structure: Rigid, table-based	Structure: Flexible (document, key-value)
Schema: Predefined	Schema: Dynamic
Best For: Structured, related data	Best For: Big data, scalability, flexible data
Examples: MySQL, PostgreSQL	Examples: MongoDB, Redis, Cassandra

16

Database Design & ORM Tools

Database Design & ORM Tools

Database Design: Involves creating a schema and ensuring data integrity through concepts like Normalization.

ORM (Object-Relational Mapper): A tool that acts as a translator.

- Lets you write database queries in your preferred programming language (e.g., Python, Java).
- Improves developer productivity and code maintainability.
- Examples: SQLAlchemy (Python), Hibernate (Java), Sequelize (Node.js).

17

APIs and RESTful Services

APIs and RESTful Services

1. API (Application Programming Interface): A contract that allows different software systems to communicate.
2. The API is the "waiter" that takes requests from the frontend and communicates them to the backend (the "kitchen").
3. Enables decoupling of frontend and backend development.

18

REST: The Standard for Web APIs

REST: The Standard for Web APIs

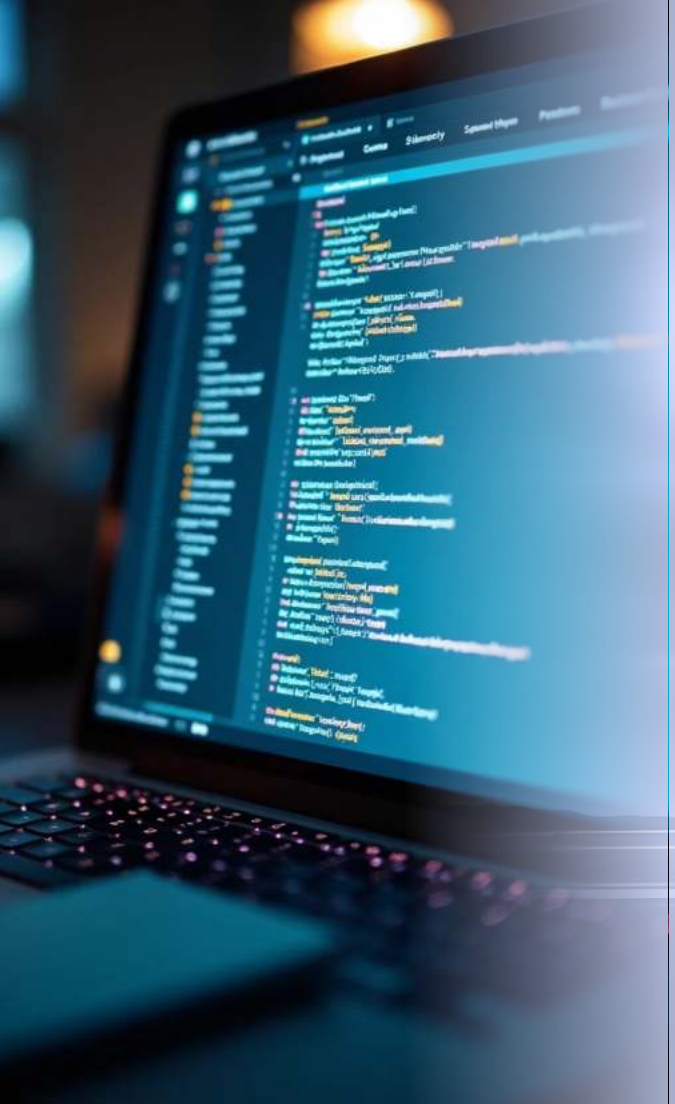
REST (REpresentational State Transfer): An architectural style for designing networked applications.

Key Principles:

- Client-Server Architecture
- Stateless (each request is independent)
- Cacheable
- Uses standard HTTP Methods (GET, POST, etc.)
- Resource-based URLs (e.g., /users/123)

19

Building and Testing APIs



Building and Testing APIs

1 **Postman**

A powerful tool for manually sending API requests to test endpoints and view responses. Essential for development and debugging.

2 **Swagger / OpenAPI**

A specification for documenting your API. Creates interactive, machine-readable documentation that makes your API easy to understand and use.

20

Authentication vs. Authorization

Authentication vs. Authorization

Authentication (AuthN): "Who are you?"

- The process of verifying a user's identity.
- Example: Logging in with a username and password.

Authorization (AuthZ): "What are you allowed to do?"

- The process of granting permissions to an authenticated user.
- Example: An admin can delete users, but a regular user cannot.

21

Common Authentication Methods

Common Authentication Methods

Basic Authentication: Sending username/password with every request (less secure).

JWT (JSON Web Tokens): A compact, URL-safe standard for creating access tokens. The server issues a signed token after login, which the client sends with future requests.

OAuth 2.0: A standard for delegated authorization. Used for "Login with Google/Facebook," allowing one app to access resources from another on behalf of the user.

22

Critical Security Practices

Critical Security Practices

1

Hash Passwords: NEVER store passwords in plain text. Use strong, salted hashing algorithms like bcrypt.

2

Prevent SQL Injection: Always sanitize and validate all user input before using it in database queries.

3

Use HTTPS: Encrypt all data in transit between the client and server to prevent eavesdropping.

4

Rate Limiting: Protect against brute-force attacks by limiting login attempts.

23

Web Frameworks: Building Faster

Web Frameworks: Building Faster

1. A set of tools, libraries, and conventions that provide a structure for building applications.
2. Saves you from "reinventing the wheel."
3. Handles common tasks like routing, database integration, and security.
4. Greatly increases development speed and code quality.

24

Popular Frameworks by Language

Popular Frameworks by Language

Node.js

Express.js, NestJS

Python

Django, Flask

PHP

Laravel, Symfony

Java

Spring Boot, Quarkus

Ruby

Ruby on Rails

Framework Philosophies

1. Minimalist / Unopinionated:
   - Provides only the bare essentials (e.g., routing).
   - Offers maximum flexibility.
   - Examples: Express.js (Node.js), Flask (Python).

2. Batteries-Included / Opinionated:
   - Comes with everything you need out of the box (ORM, admin panel, auth).
   - Steeper learning curve but extremely productive.
   - Examples: Django (Python), Laravel (PHP), Ruby on Rails.

The Testing Pyramid

1

Unit Tests (Base): Test individual functions or components in isolation. Fast and numerous.

2

Integration Tests (Middle): Verify that multiple components work together as expected (e.g., API endpoint and database).

3

End-to-End Tests (Top): Simulate a full user journey through the application. Slow but valuable.

Tools for Testing and Debugging

Testing Frameworks:

- Jest, Mocha: For JavaScript/Node.js
- PyTest, Unittest: For Python
- JUnit: For Java

Debugging Techniques:

- Logging: Writing application status and errors to a file or service.
- Monitoring Tools: Tracking performance and errors in real-time.
- Debuggers: Step-through code line by line to inspect its state.

30

Deployment and DevOps

Deployment and DevOps

Deployment

The process of taking your finished code and making it available to users on a live server.

DevOps

A culture and set of practices that combines software development (Dev) and IT operations (Ops) to shorten the development lifecycle and deliver high-quality software continuously.

31

CI/CD - The Automated Assembly Line

CI/CD - The Automated Assembly Line

1. CI (Continuous Integration): Automatically building and testing code every time a change is pushed.
2. CD (Continuous Deployment/Delivery): Automatically deploying the code to production if all tests pass.
3. Goal: Make deployments frequent, fast, and reliable.
4. Tools: Jenkins, GitHub Actions, GitLab CI.

34

Scalability and Performance

Scalability and Performance

Scalability

The ability of your system to handle an increasing amount of load. Can your app support 10 users? What about 10 million?

Performance

The speed and responsiveness of your system under load. How quickly does a page load or an API respond?

How to Scale: Vertical vs. Horizontal

1. Vertical Scaling ("Scaling Up"):
   - Making a single server more powerful (more CPU, RAM, disk space).
   - Simple, but has a hard physical limit and can be expensive.

2. Horizontal Scaling ("Scaling Out"):
   - Adding more servers to a pool to distribute the load.
   - The foundation for massive, web-scale applications.
   - More complex, but highly flexible and resilient.

36

Key Techniques for Performance

Key Techniques for Performance

Load Balancing: Distributes incoming traffic across multiple servers in a horizontal scaling setup.

Caching: Storing frequently accessed data in a fast, in-memory store (like Redis) to avoid slow database queries.

Database Optimization: Using techniques like indexing, query analysis, and read replicas to speed up data retrieval.

37

Monolithic vs. Microservices Architecture

Monolithic vs. Microservices Architecture

1 Monolith

- The entire application is a single, tightly-coupled unit.
- Simple to start, but becomes difficult to scale and maintain as it grows.

2 Microservices

- The application is broken down into small, independent services.
- Each service can be developed, deployed, and scaled independently.

Version Control and Collaboration

1 **Version Control**

A system that records changes to a file or set of files over time so that you can recall specific versions later.

2 **Git**

The industry-standard distributed version control system.

3 **GitHub/GitLab**

Web-based platforms that provide cloud hosting for Git repositories and powerful collaboration tools.

40

Git Basics: The Core Workflow

Git Basics: The Core Workflow

clone

Get a local copy of a remote repository.

branch

Create a separate line of development to work on a feature.

commit

Save your changes to your local branch with a descriptive message.

push

Upload your committed changes to the remote repository.

pull

Download changes from the remote repository to your local copy.

41

GitHub Workflows for Collaboration

GitHub Workflows for Collaboration

1 Pull Request (PR)

A formal request to merge your branch into the main codebase. This is the heart of collaboration.

2 Code Review

Team members review the code in a Pull Request, leaving comments and suggestions for improvement before it's approved.

3 Issues

Track bugs, feature requests, and other tasks.

43

Simple Backend App: A TODO List

Simple Backend App: A TODO List

1. **Frontend**
   React, Vue, or Angular

2. **API**
   REST API with endpoints like:
   - GET /tasks
   - POST /tasks
   - DELETE /tasks/{id}

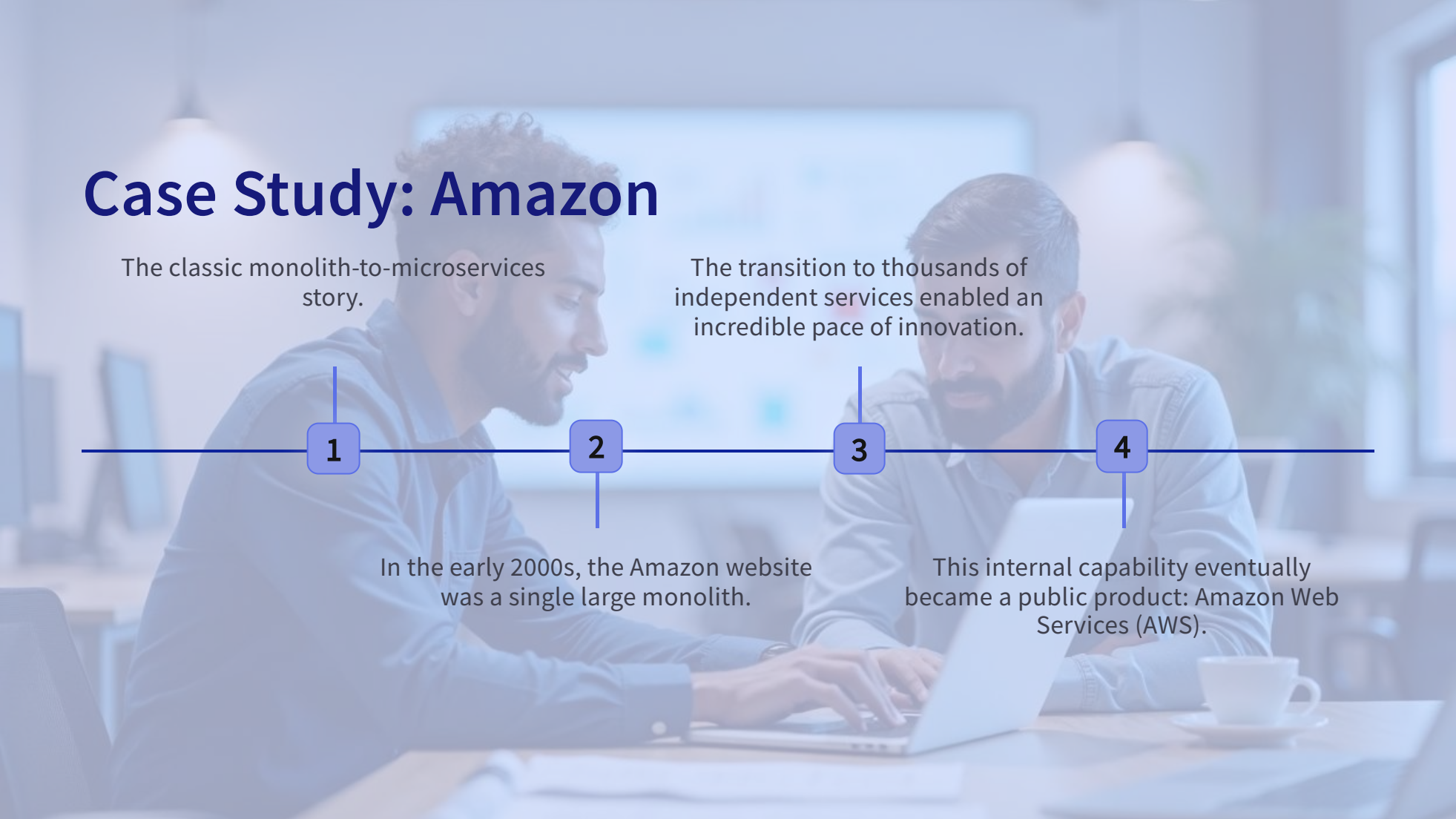
3. **Backend**
   Node.js + Express

4. **Database**
   MongoDB or PostgreSQL

Case Study: Netflix

1. Key Concepts: Cloud-native (AWS), Scalability, Performance.
2. Pioneers in microservices and resilience engineering.
3. Chaos Engineering: Intentionally breaking parts of their own system in production to find weaknesses before they cause a real outage.
4. Backend optimizes video streaming in real-time based on your location and network speed.

Case Study: Amazon



The classic monolith-to-microservices story.

1

In the early 2000s, the Amazon website was a single large monolith.

2

The transition to thousands of independent services enabled an incredible pace of innovation.

3

This internal capability eventually became a public product: Amazon Web Services (AWS).

4

Conclusion & Recap

1. The backend is the critical engine of the digital world.
2. Key Pillars: Languages, Databases, APIs, Frameworks, Deployment, and Scalability.
3. The field is dynamic, deep, and constantly evolving.
4. Continuous learning is the most important skill for a backend developer.

49

Q&A

Thank You