BY LEARNSTACK

LearnStack Backend Development Series

GET /v1/resources -> 200 OK

POST /v1/auth/login -> JWT

2026 Edition

API Development Guide

(REST + JSON)

Design, Build & Secure Production-Ready REST APIs

GET /v1/handbook 200 OK

REST JSON HTTP Authentication JWT OpenAPI Microservices

Learn. Build. Grow.

linktr.ee/LearnStack

About This Handbook

API Development Guide (REST + JSON) is a practical, production-minded handbook for designing, building, securing, testing, documenting, and reasoning about REST APIs that return JSON.

It starts with the request-response model, then moves into real backend concerns: endpoint design, HTTP methods, status codes, authentication, CORS, pagination, idempotency, caching, rate limiting, OpenAPI documentation, and interview preparation.

It is written for backend-curious frontend developers, students, bootcamp graduates, self-taught programmers, API testers, and anyone preparing for full-stack or backend interviews.

Use this handbook as a course first and as a reference later. Read Chapters 1-7 sequentially, then keep Chapters 8 and 12 nearby when designing or reviewing real APIs.

Learning Path

- 01 API basics
- 02 REST + HTTP
- 03 JSON + errors
- 04 Auth + security
- 05 Docs + projects
- 06 Interviews

After finishing, you will be able to:

- Design clean REST endpoints with predictable resource naming.
- Choose correct HTTP methods, headers, status codes, and JSON shapes.
- Secure APIs with API keys, JWT, OAuth concepts, CORS rules, and rate limits.
- Document APIs with OpenAPI and explain designs clearly in interviews.

LINK: More resources

https://linktr.ee/LearnStack

Format	Value
Brand	LearnStack
Tagline	Learn. Build. Grow.
Series	Backend Development
Edition	2026

Designed for:

- Fast revision
- Portfolio projects
- Backend interviews
- Production habits

Professional mindset

Good APIs are not just endpoints that work. They are predictable contracts between teams, apps, partners, and future developers. This guide emphasizes naming, error shape, security, documentation, and long-term maintainability.

Table of Contents - Course Chapters

Navigation tip

Read Chapters 1-7 as a compact course. Use Chapter 8 as your methods, status codes, and headers reference. Use the Workbook pages when building portfolio projects or checking production readiness.

Table of Contents - Reference, Projects & Appendix

Navigation tip

Read Chapters 1-7 as a compact course. Use Chapter 8 as your methods, status codes, and headers reference. Use the Workbook pages when building portfolio projects or checking production readiness.

01 Introduction to APIs & REST

LearnStack Backend Development Series

1.1 What is an API?

An API is like a restaurant menu plus a waiter. The menu tells you what you can ask for, and the waiter carries your request to the kitchen and returns a result. In software, an API defines how one system can ask another system for data or actions.

INFO: Technical definition
An Application Programming Interface is a documented contract that describes operations, inputs, outputs, errors, and rules for interacting with a software system.

Term	Meaning	Example
Client	The app or service making a request	Mobile app, browser, backend service
Server	The system that receives the request and returns a response	API server
Endpoint	A URL that represents a resource or action	/v1/users/42
Contract	The expected format and behavior	Request JSON, response JSON, status codes

1.2 What is REST?

REST, short for Representational State Transfer, is an architectural style described by Roy Fielding in his 2000 dissertation. REST is not a library or protocol; it is a set of constraints for building scalable networked systems. HTTP is the most common protocol used to implement REST APIs.

Constraint	Practical meaning
Client-server	User interface and data storage evolve independently.
Stateless	Each request includes enough information to be understood on its own.
Cacheable	Responses can describe whether they may be reused.
Uniform interface	Resources are manipulated through consistent methods and representations.
Layered system	Clients do not need to know whether they hit a server, gateway, or proxy.
Code on demand	Optional ability to send executable code; rarely used in JSON APIs.

TIP: REST mindset
Model nouns as resources. Use HTTP methods to express actions. Use status codes to express outcomes. Use JSON consistently for request and response bodies.

1.3 REST vs SOAP vs GraphQL vs gRPC

Style	What it is	Best use	Trade-off
REST	Resource-oriented API style over HTTP	Public APIs, CRUD apps, mobile/web backends	Can overfetch or underfetch without careful design
SOAP	XML-based protocol with strict contracts	Legacy enterprise, banking, enterprise integrations	Verbose, heavier learning curve
GraphQL	Query language where clients ask for exact fields	Complex frontends, many related data types	Caching and authorization can be harder
gRPC	Binary RPC framework using Protocol Buffers	Internal microservices, low-latency service-to-service calls	Less browser-friendly than REST without extra tooling

1.4 The API ecosystem today

Modern software is a network of APIs. Public APIs let developers integrate payments, maps, weather, identity, and messaging. Internal APIs connect microservices. Mobile and web apps often share the same backend API. IoT devices stream telemetry through APIs. Even AI features commonly begin with JSON over HTTP.

REAL WORLD: How companies use APIs
A shopping app might call a product API, pricing API, inventory API, payment API, shipping API, notification API, and analytics API during one checkout flow.

1.5 Real-world public APIs

API category	Examples	What they expose
Payments	Stripe, Razorpay, PayPal	Checkout sessions, charges, refunds, webhooks
Maps	Google Maps, Mapbox	Geocoding, routes, places, map tiles
Social	X, LinkedIn, Instagram	Profiles, posts, metrics, publishing workflows
Weather	OpenWeather, WeatherAPI	Forecasts, alerts, current conditions
Developer tools	GitHub, GitLab	Repos, issues, pull requests, CI/CD metadata
AI services	OpenAI, Anthropic, Google	Text generation, embeddings, tool calls, file workflows

02 Why Learn API Development?

LearnStack Backend Development Series

2.1 Key advantages

- Decoupling: a frontend can change without rewriting the backend as long as the API contract stays stable.
- Reuse: mobile apps, web apps, and partner integrations can share one backend.
- Integration: APIs let your product connect to payment, email, analytics, AI, and identity providers.
- Team scalability: separate teams can own separate services with explicit boundaries.

REAL WORLD: Single backend, many clients
A school management system may have an admin dashboard, teacher mobile app, student portal, and parent notifications. A well-designed API powers all of them from one consistent backend.

2.2 Job market demand

Backend and full-stack roles commonly require HTTP, JSON, authentication, databases, and API design. In the United States, software developers are tracked as a high-pay technology occupation, and web developers remain a major entry path into API-connected applications. Salary and demand vary by country, city, experience, and company size; treat all numbers as planning signals, not promises.

Signal	What it suggests
Software developer median pay	U.S. BLS reported a May 2024 median annual wage of $133,080 for software developers.
Web developer median pay	U.S. BLS reported a May 2024 median annual wage of $90,930 for web developers.
Developer ecosystem size	The 2025 Stack Overflow Developer Survey received 49,000+ responses from 177 countries.
Practical hiring signal	Job descriptions often mention REST APIs, JSON, Postman, JWT/OAuth, SQL/NoSQL, CI/CD, and cloud deployment.

2.3 What you can build

SaaS backends
User accounts, billing, teams, subscriptions, roles, dashboards, and reporting APIs.

Mobile app backends
Auth, sync, notifications, user profiles, media upload, and offline reconciliation.

Integration layers
Connect CRM, payments, shipping, analytics, marketing, and ERP systems.

Internal tools
Admin dashboards, automation APIs, approval workflows, and operational consoles.

AI product APIs
Endpoints for prompts, documents, embeddings, chat sessions, usage limits, and audit logs.

IoT APIs
Device registration, telemetry ingestion, commands, alerts, and firmware metadata.

2.4 REST vs GraphQL vs gRPC comparison

Criteria	REST	GraphQL	gRPC
Learning curve	Low to medium	Medium	Medium to high
Browser friendliness	Excellent	Excellent	Requires web-specific tooling
Caching	Strong HTTP support	Requires custom strategy	Usually not HTTP cache based
Performance	Good for most apps	Good, but resolver design matters	Excellent binary performance
Public APIs	Very common	Common in developer platforms	Less common for browsers
Internal services	Common	Sometimes	Very strong

Criteria	REST	GraphQL	gRPC
Tooling	Postman, OpenAPI, curl	GraphiQL, Apollo tooling	Proto compilers, grpcurl

TIP: What to learn first
Learn REST and JSON first. It teaches HTTP fundamentals that transfer to GraphQL, gRPC gateways, webhooks, and cloud APIs.

03 Career Paths

LearnStack Backend Development Series

API knowledge is useful far beyond one job title. These roles differ in daily work, but all benefit from strong REST design, JSON, auth, testing, and documentation skills. Salary ranges below are intentionally broad because location, seniority, company type, and currency change the answer dramatically.

Backend Developer

What you do: implement endpoints, data models, business logic, auth, tests, and performance. Avg salary: entry to senior varies widely. Skills: Node/FastAPI/Spring, SQL, HTTP, auth.

API Engineer

What you do: design API contracts, SDKs, developer portals, gateway rules, and partner integrations. Skills: OpenAPI, versioning, OAuth, docs, observability.

Full-Stack Developer

What you do: connect frontend UI to backend APIs, handle forms, state, errors, and deployment. Skills: React/Next, REST, JSON, auth, databases.

Integration Engineer

What you do: connect systems such as CRM, payment, ERP, email, and analytics. Skills: webhooks, retries, idempotency, queues, OAuth.

Solutions Architect

What you do: design end-to-end systems for clients or internal platforms. Skills: API design, cloud, security, scaling, trade-off communication.

DevOps / Platform Engineer

What you do: run API infrastructure, gateways, monitoring, CI/CD, secrets, and rate limits. Skills: Docker, Kubernetes, cloud, logs, metrics.

API QA / Test Engineer

What you do: test contracts, edge cases, auth flows, load, and regressions. Skills: Postman, Newman, curl, test automation, status codes.

Technical Product Manager

What you do: define API requirements, prioritize use cases, document developer experience. Skills: API literacy, user stories, analytics, stakeholder alignment.

TIP: Portfolio strategy
Build one clean CRUD API, one auth-protected API, and one design-heavy API with pagination, filtering, error format, OpenAPI docs, and tests. That proves more than a dozen unfinished tutorials.

REAL WORLD: Interview advantage
Candidates who explain status codes, idempotency, validation, auth, and API documentation usually stand out because they reason like maintainers, not just coders.

04 Setting Up the Environment

LearnStack Backend Development Series

4.1 Primary stack: Node.js + Express

This handbook uses Node.js with Express as the primary stack because it is beginner-friendly, widely used, and maps cleanly to HTTP concepts. Python + FastAPI is mentioned as a strong alternative for readers who prefer Python.

Shell

```
# Install Node.js from https://nodejs.org, then verify:
node --version
npm --version

# Create a project
mkdir learnstack-api-guide
cd learnstack-api-guide
npm init -y
npm install express cors jsonwebtoken bcryptjs dotenv
npm install --save-dev nodemon
```

Alternative	Install	Best for
FastAPI	pip install fastapi uvicorn	Python developers, data apps, auto-generated docs
Flask	pip install flask	Small APIs, lightweight prototypes
Spring Boot	Use Spring Initializr	Java enterprise APIs
ASP.NET Core	dotnet new webapi	Microsoft/.NET ecosystems

4.2 Recommended tools

Tool	Best for	Platform	Free?
Postman	Manual API testing, collections, environments	Windows, macOS, Linux, Web	Free tier
Insomnia	Clean REST and GraphQL testing	Windows, macOS, Linux	Free tier
curl	Terminal requests and scripts	All platforms	Yes
httpie	Readable command-line HTTP client	All platforms	Yes
Thunder Client	VS Code embedded API testing	VS Code	Free tier
Swagger UI	Interactive API documentation	Browser	Yes

4.3 Hello World API

JavaScript

```
// server.js
const express = require("express");
const app = express();

app.use(express.json());

app.get("/hello", (req, res) => {
  res.status(200).json({
    message: "Hello from LearnStack API Guide",
    success: true
  });
});

app.listen(3000, () => {
  console.log("API running on http://localhost:3000");
});
```

GET `http://localhost:3000/hello`

```
Headers:
Content-Type: application/json
```

200 OK **Response**

```
{
  "message": "Hello from LearnStack API Guide",
  "success": true
}
```

4.4 Common beginner errors

CAUTION: Missing JSON parser
If you forget app.use(express.json()), req.body will be undefined for JSON requests.

CAUTION: Incorrect Content-Type
Clients should send Content-Type: application/json when sending JSON request bodies.

CAUTION: CORS confusion
CORS is enforced by browsers, not by curl or Postman. A request can work in Postman and still fail in a browser.

TIP: Developer workflow
Keep your server logs open, test one endpoint at a time, save working requests in Postman, and document every accepted request body.

05 Core Concepts: HTTP & REST Fundamentals

LearnStack Backend Development Series

5.1 HTTP Fundamentals

INFO: What it is
HTTP is the request-response protocol used by browsers, apps, and services to exchange messages. A client sends a request; a server returns a response with a status code, headers, and optional body.

HTTP/JSON

```
GET /health HTTP/1.1
Host: api.example.com
Accept: application/json
```

GET `/health`

```
Headers:
Content-Type: application/json
```

200 OK Response

```
{ "status": "ok", "uptime": 48210 }
```

TIP: Pro tip
Start with simple health and version endpoints. They help load balancers, monitoring systems, and developers verify the service is alive.

CAUTION: Common mistake
Do not put secrets in URLs. URLs may be logged by browsers, servers, proxies, and analytics tools.

5.2 HTTP Methods

INFO: What it is
HTTP methods describe intent. GET reads, POST creates or triggers processing, PUT replaces, PATCH partially updates, DELETE removes, HEAD reads headers, and OPTIONS describes capabilities.

HTTP/JSON

```
GET /users
POST /users
PUT /users/42
PATCH /users/42
DELETE /users/42
```

POST `/users`

```
Headers:
Content-Type: application/json

Body:
{
  "name": "Meera Rao",
  "email": "meera@example.com"
}
```

201 Created Response

LearnStack Free Preview

This was a free preview. Get the full book on LearnStack.

Visit: https://www.learnstack.co.in

Digital PDF delivery is handled through Gumroad email after purchase.