How to learn API?

with the list of all important resources you will ever need.

DR MILAN MILANOVIĆ
NOV 14, 2024

Our daily work as software engineers involves creating or utilizing these APIs. Creating **well-designed REST APIs is crucial**—not only should they be easy to work with and concise, but they should also be well-designed against misuse to prevent future issues.

In this issue, we explore the art of **API design through a curated selection of resources**, including insightful books, articles, and real-world examples from industry leaders like Stripe and Slack. We'll learn about the proper use of HTTP methods, RESTful URI structures, effective error handling, caching strategies, and the importance of thorough API documentation.

Additionally, we'll present **common API anti-patterns** that often lead to bigger problems later.

So, let's dive in.

WebStorm and Rider are now free for non-commercial use (Sponsored)

Great news for all hobbyists, students, content creators, and open-source contributors! Now web, game, and .NET developers can use WebStorm and Rider for free for learning and honing their skills, developing open-source projects, creating content, and developing as a hobby.

API Roadmap

APIs are one of the first things software engineers should learn, much before other, more complex things, such as distributed systems, cloud computing, and others. They are the basis of all development today, and they will help you understand how software systems work and how to use proper programming languages and frameworks.

Whether you're a beginner or an experienced developer looking to learn about APIs, this API learning roadmap will help you understand the key concepts and technologies.

1. Introduction to APIs

Before diving deep into API development, it's crucial to understand the fundamentals. This section covers the basic building blocks that form the foundation of API knowledge, from core protocols to different APIs used in modern software development.

- **Basics**: Learn about HTTP protocol, DNS, URLs, and more.
- **API Definition**: An API is a set of protocols, routines, and tools for building software applications. It specifies how software components should interact.

- **API Types**:
  - 🌐 **Public APIs**: Open for use by external developers (e.g., Twitter API)
  - 🔒 **Private APIs**: Used internally within an organization
  - 🔑 **Partner APIs**: Shared with specific business partners
  - 🧩 **Composite APIs**: Combine multiple data or service APIs

2. API Architectures

Understanding different API architectural styles is essential, as each serves specific use cases and has advantages and trade-offs. These architectures form the core of your API's structure and client interaction.

- **REST (Representational State Transfer):** A widely used architectural style for web APIs
- **GraphQL:** A query language for APIs that allows clients to request specific data
- **SOAP (Simple Object Access Protocol)**: A protocol for exchanging structured data
- **gRPC**: A high-performance, open-source framework developed by Google
- **WebSockets**: Enables full-duplex, real-time communication between client and server
- **Webhook**: Allows real-time notifications and event-driven architecture

Learn more about API Architecture types:

Tech World With Milan Newsletter

What are the main API Architecture Styles?

In today's issue, we will discuss the main API architectural styles, their usage, the pros and cons of each style, and recommendations for when to use each style...

Read more

3 months ago · 43 likes · Dr Milan Milanović

And what are the fundamental differences between them:

Tech World With Milan Newsletter

When to use GraphQL, gRPC, and REST?

Building APIs is one of the most important tasks for developers in modern engineering. These APIs allow different systems to communicate and exchange data. While REST has been the de facto standard for implementing APIs for many years, new emerging standards, such as...

Read more

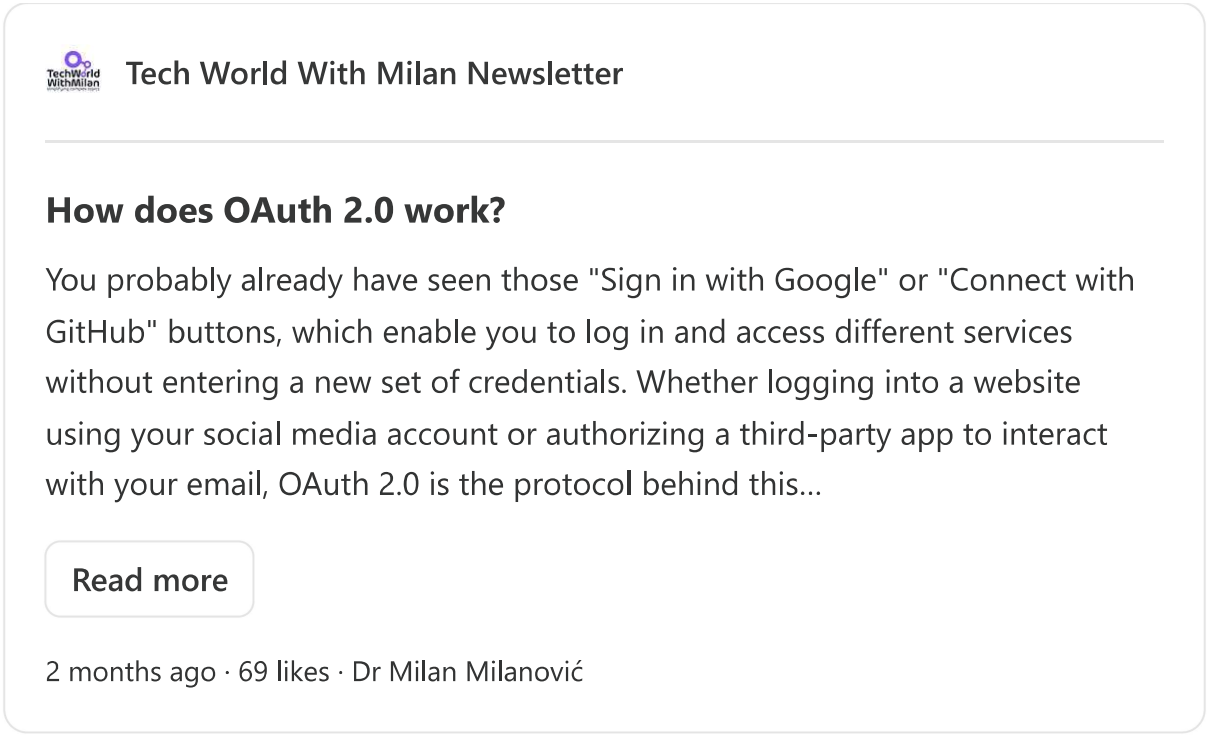
8 months ago · 60 likes · 2 comments · Dr Milan Milanović

3. API Security

Security is one of the most critical parts of API development. This section covers essential security concepts and best practices that protect your API from unauthorized access and potential threats while ensuring secure data transmission.

- **Authentication**: Basic, OAuth 2.0, JSON Web Tokens (JWT)
- **Authorization**: Controlling access rights to resources
- **Rate Limiting**: Preventing abuse by limiting the number of requests
- **Encryption**: Protecting data in transit using HTTPS
- **Best practices**: OWASP Top 10 Security Risks

Learn more about OAuth 2.0:



4. API Design Best Practices

Good API design differentiates between an API developers love to use and one they avoid. These best practices have evolved from years of industry experience and help create intuitive, efficient, and maintainable APIs.

- **RESTful conventions**: Using HTTP methods correctly, proper resource naming
- **Versioning**: URI versioning (e.g., /v1/users), Query parameter versioning (e.g., /users?version=1), Header versioning (e.g., Accept: application/vnd.company.v1+json).
- **Pagination**: Efficiently handling large datasets
- **Error Handling**: Proper use of HTTP status codes and informative error messages
    - RFC 7807 - Problem Details for HTTP APIs (outdated)
    - RFC 9457 - Problem Details for HTTP APIs (new version)

Learn more about API Design Best Practices:

Tech World With Milan Newsletter

REST API Design Best Practices

What is API...

Read more

2 years ago · 9 likes · Dr Milan Milanović

And what is the API-first approach:

Tech World With Milan Newsletter

What is API-First Development?

You probably heard about terms such as API-first, Code-first, and Design-first. Yet, although we as developers are primarily familiar with the Code-first approach to developing APIs, there are also some other approaches...

Read more

2 years ago · 9 likes · Dr Milan Milanović

5. API Documentation

Great APIs are only as good as their documentation. This section covers tools and practices for creating clear, comprehensive documentation that makes your API accessible to developers.

- **Swagger/OpenAPI Specification**: A standard for describing RESTful APIs
- **Postman**: A popular tool for API development and documentation

- **ReDoc**: A tool for generating beautiful API documentation
- **DapperDox**: is Open-Source and provides rich, out-of-the-box rendering OpenAPI specification.
- **Slate**: A popular tool with the project being forked more than 15,000 times.
- **ReadMe**: Transforms static API documentation into real-time, interactive developer hubs.

6. API Testing

Testing ensures your API works as intended and maintains its quality over time. This section explores various testing approaches and tools that help validate functionality, performance, and reliability.

- **Postman**: Allows creating and running API tests
- **SoapUI**: A tool for testing SOAP and REST APIs
- **JMeter**: Used for performance and load testing
- **API Mocking**: Tools like Mockoon or Postman's mock servers for simulating API responses
- **Pact**: A contract testing tool that ensures APIs and microservices adhere to defined agreements, facilitating reliable integration.
- **Insomnia**: An open-source, cross-platform API client for designing, debugging, and testing REST, GraphQL, and gRPC APIs.
- **Rest-Assured**: A Java-based library that simplifies the testing of REST services by providing a domain-specific language for writing tests.
- **Katalon Studio**: A comprehensive test automation tool supporting web, mobile, and API testing with a user-friendly interface and robust features
- **Newman**: A command-line collection runner for Postman, enabling automated and continuous integration of API tests

7. API Management

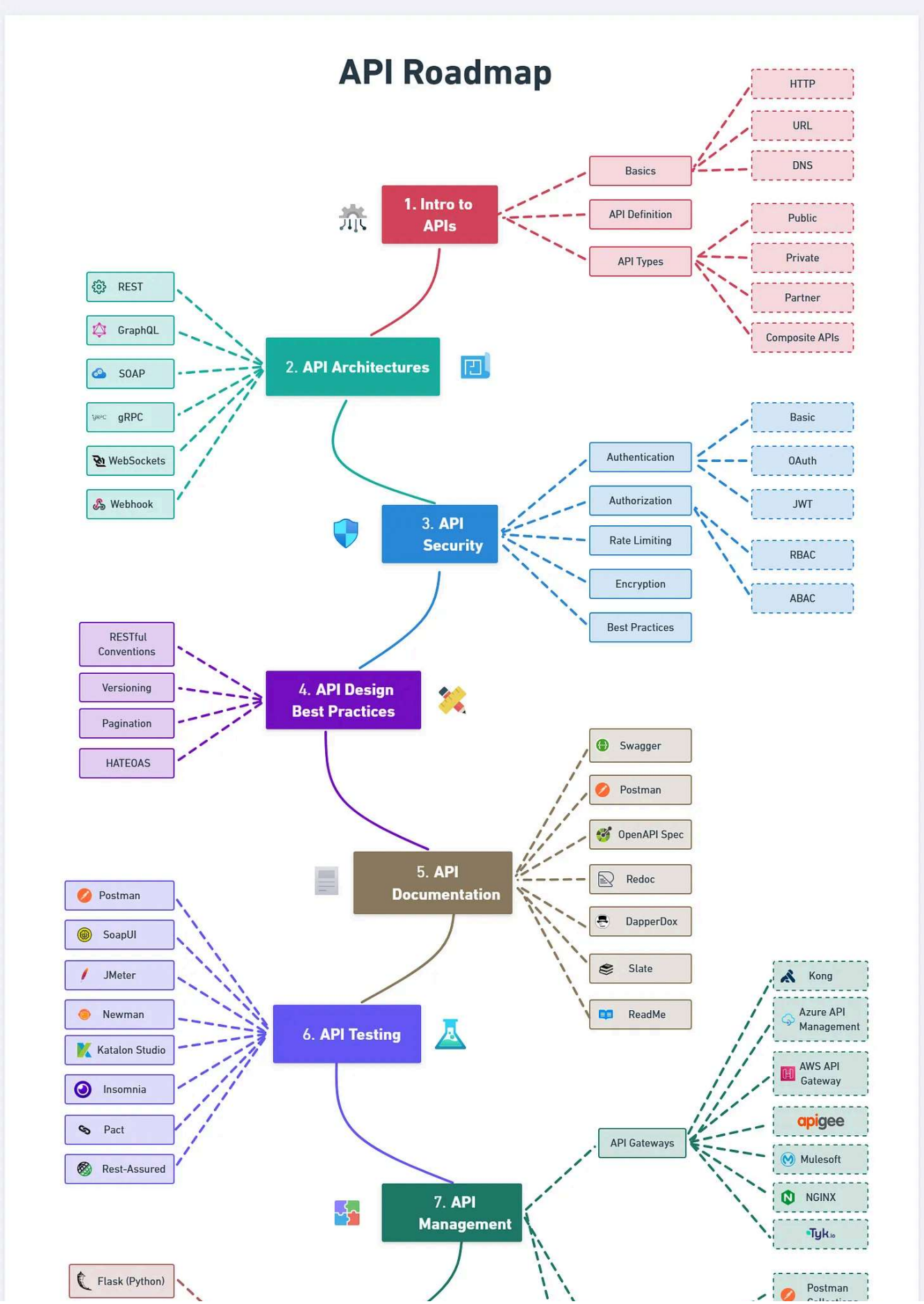
As APIs grow in complexity and usage, proper management becomes crucial. This section covers tools and practices for handling the full lifecycle of APIs, from deployment to monitoring.

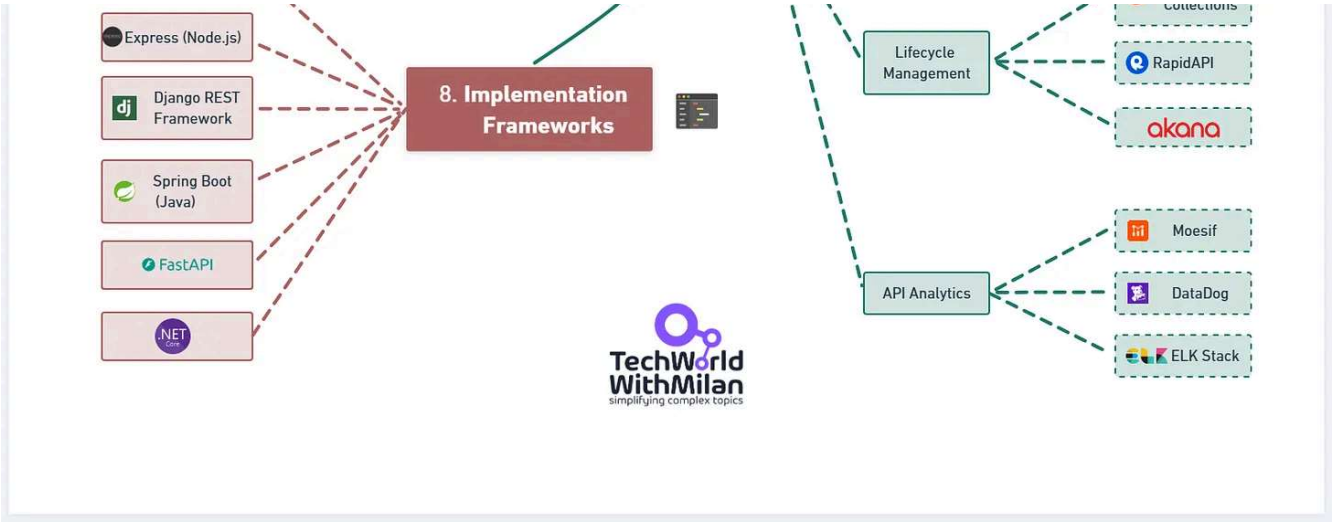
- **API Gateways**: Azure API Management, AWS API Gateway, Kongk, Apigee.
- **Lifecycle Management**: Postman Collections, RapidAPI, Akana.
- **API Analytics and Monitoring**: Moesif, Datadog, ELK Stack (Elasticsearch, Logstash, Kibana)

8. Implementation Frameworks

Choosing the proper framework can significantly impact your API development experience. This section covers popular frameworks across different programming languages, each offering unique features and capabilities.

- **Python**: Flask, Django REST framework, FastAPI
- **JavaScript**: Express.js
- **Java**: Spring Boot
- **.NET**: ASP.NET Core

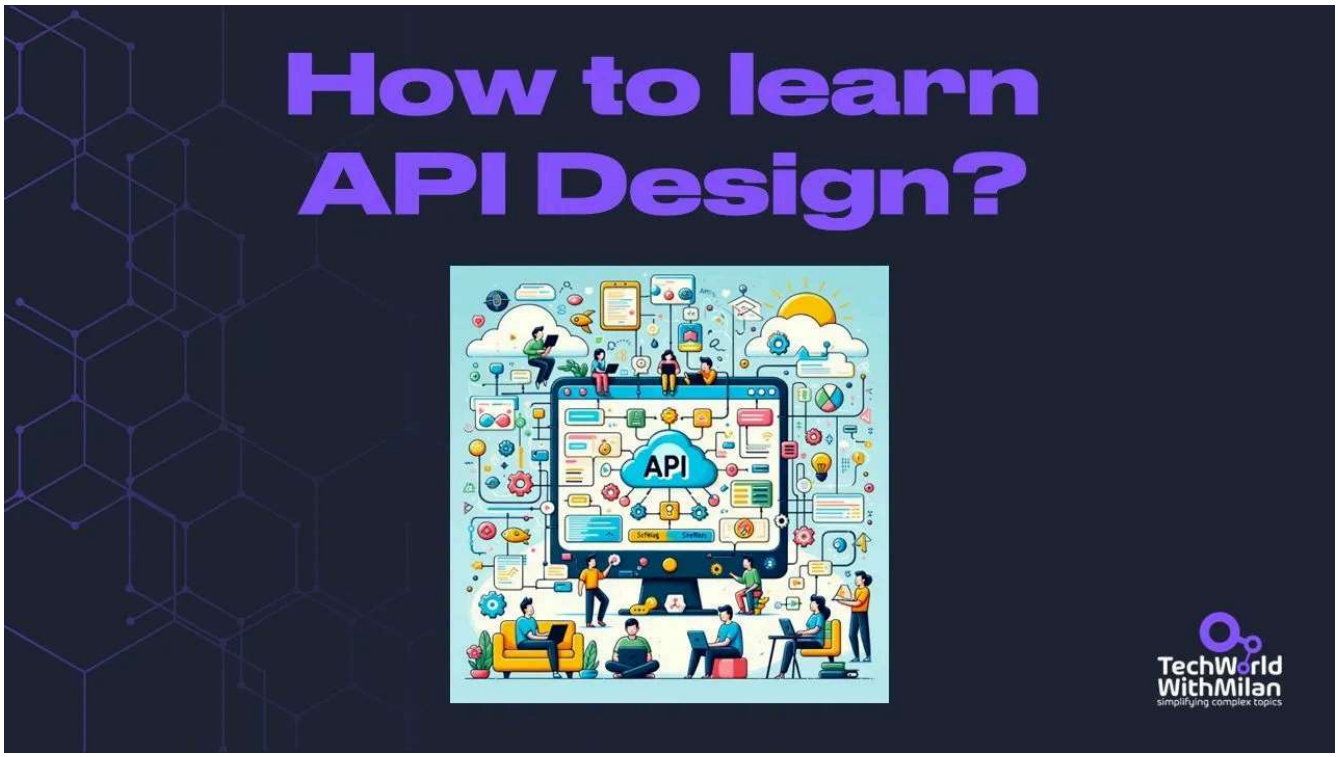

API Roadmap
1. Intro to APIs
Basics
HTTP
URL
DNS
API Definition
API Types
Public
Private
Partner
Composite APIs
2. API Architectures
REST
GraphQL
SOAP
gRPC
WebSockets
Webhook
3. API Security
Authentication
Basic
OAuth
JWT
Authorization
RBAC
ABAC
Rate Limiting
Encryption
Best Practices
4. API Design Best Practices
RESTful Conventions
Versioning
Pagination
HATEOAS
5. API Documentation
Swagger
Postman
OpenAPI Spec
Redoc
DapperDox
Slate
ReadMe
6. API Testing
Postman
SoapUI
JMeter
Newman
Katalon Studio
Insomnia
Pact
Rest-Assured
7. API Management
API Gateways
Kong
Azure API Management
AWS API Gateway
apigee
Mulesoft
NGINX
Tyk.io
Flask (Python)
Postman



API Roadmap

Recommended resources to learn API Design

Most of our daily work as software engineers utilizes or creates REST APIs. APIs are the standard method of communication between systems. Therefore, building REST APIs properly is crucial to avoid future issues. A well-defined API should be easy to work with, concise, and hard to misuse.



Here are some references that could help you to learn API design:

1. Books

- **Designing Web APIs: Building APIs That Developers Love**, Brenda Jin, Saurabh Sahni, Amir Shevat. A comprehensive guide that covers the entire API design lifecycle, from planning to production.
- **The Design of Web APIs**, Arnaud Lauret. Focuses on practical approaches to API design with real-world examples and best practices.
- **Principles of Web API Design**, James Higginbotham. Explores fundamental principles and patterns for creating sustainable web APIs.
- **REST In Practice**, Jim Webber, Savas Parastatidis, and Ian Robinson, O'Reilly.

2. Articles

- Roy Thomas Fielding **Ph.D. dissertation that introduced REST architectural style** in 2000:
- **API design guide** by Google. Official guidelines from Google explaining their approach to API design and best practices.
- **Microsoft REST API Guidelines**
- **Learn API Design** GitHub Repo—a curated collection of resources and tutorials for API design learning.
- **Zalando RESTful API and Event Guidelines**. Practical guidelines from Zalando's engineering team on building consistent RESTful APIs.
- **How to design better APIs**, by Ronald Blüthl. 15 language-agnostic, actionable tips on REST API design.
- **How to implement better APIs** by Ronald Blüthl. The Next.js reference implementation.
- **How to use undocumented web APIs** by Julia Evans
- **How to design a RESTful API architecture from a human-language spec** by Filipe Ximenes and Flávio Juvenal
- **HATEOAS Driven REST APIs**

- **How We Design Our APIs at Slack**, API Design Guidelines You Can Use Today

3. Testing APIs

- **API Testing with Postman** - A comprehensive guide to testing APIs using the popular Postman tool.

4. API Security

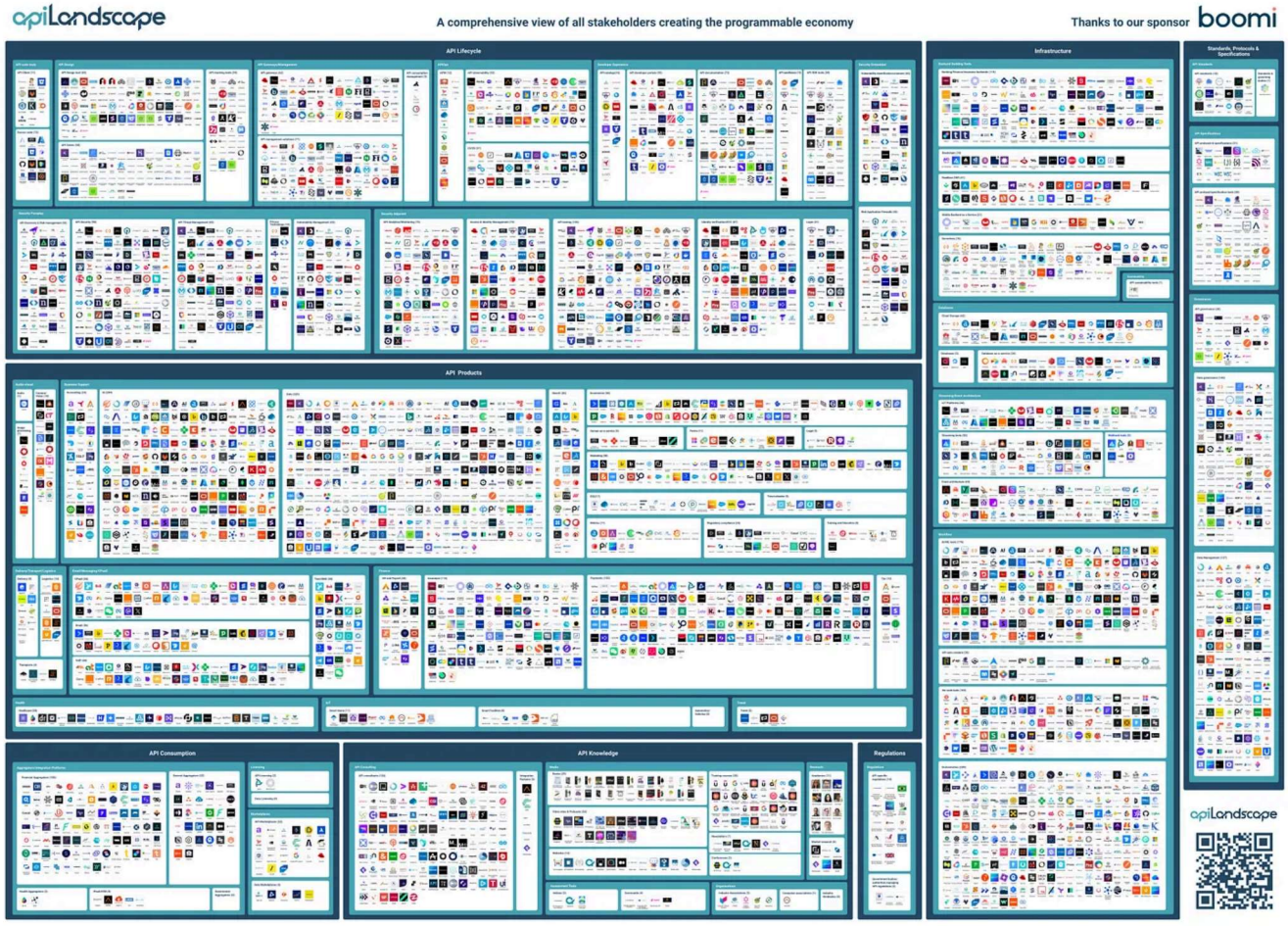
- **API Security Checklist** - Essential security considerations and best practices for API development.
- **OWASP Top 10 API Security Risks** – 2023. Current security vulnerabilities and threats specific to APIs.

5. API Examples

- **Stripe API** and **How Stripe Build APIs** - Known for their excellent documentation and developer experience.
- **Twitter API** - A widely-used example of REST API implementation at scale.
- **Twilio API** - Notable for its consistency and developer-friendly design.
- **How We Design Our APIs at Slack** - Insights into Slack's API design principles and practices.

6. Other

- **API Patterns** - Common design patterns and architectural approaches for APIs.
- **API Landscape** - Overview of the current API ecosystem and trends.
- **The state of public API** - Analysis of current trends and best practices in public API design.
- **2024 State of the API Report** by Postman. Annual report on API industry trends, challenges, and opportunities.



The API Landscape

BONUS: API Anti-patterns

There are design patterns in software design, which are good practices, but we also have anti-patterns. These anti-patterns often seem like good ideas initially but can lead to different problems over time.

This holds not only for general software design but also in the API world.

Here are some common API anti-patterns we can see in the wild:

1. Using the wrong HTTP Methods

If we don't align with RESTful design principles, this can lead us to a confusing and unpredictable API.

Here are some examples of such problems:

- Using POST for everything instead of appropriate GET, PUT, and DELETE methods
  - ❌ `POST /updateUser`
  - ✅ `PUT /users/{id}`
- Focusing on actions rather than resources
  - ❌ `GET /getLatestCheckout`
  - ✅ `GET /checkouts/latest`

2. URI is not RESTful

There could be multiple issues in this area:

- Inconsistent resource naming (mixing singular/plural)
  - ❌ `/user/{id}` vs `/companies`
  - ✅ `/users/{id}` and `/companies`
- Mixing verbs and nouns
  - ❌ `/createPost` and `/comments`
  - ✅ `/posts` and `/comments`

The best practice is to:

- Use consistent plural nouns for collections
- Keep URLs resource-focused, not action-focused
- Maintain consistent casing (preferably kebab-case)

3. Bad Error Handling

An example could be generic error messages, such as always returning "An error occurred" instead of "Invalid email format."

Another example is using incorrect or non-standard status codes, such as Returning 200 OK for all responses, even when an error occurs. We need to know some major response

status codes, but we can also create our own.

Some common mistakes are:

- Generic error messages
  - ❌ "An error occurred"
  - ✅ "Invalid email format: user@domain missing top-level domain"
- Incorrect HTTP status codes
  - ❌ Using 200 OK for errors
  - ✅ Using appropriate codes (400 for client errors, 500 for server errors)

Best practices are:

- Use specific, actionable error messages
- Include error codes and documentation references
- Follow HTTP status code conventions:
  - 2xx for success
  - 4xx for client errors
  - 5xx for server errors

4. Ignoring caching

We usually don't use any caching mechanism with our REST APIs, even though we have many options. This can degrade application performance.

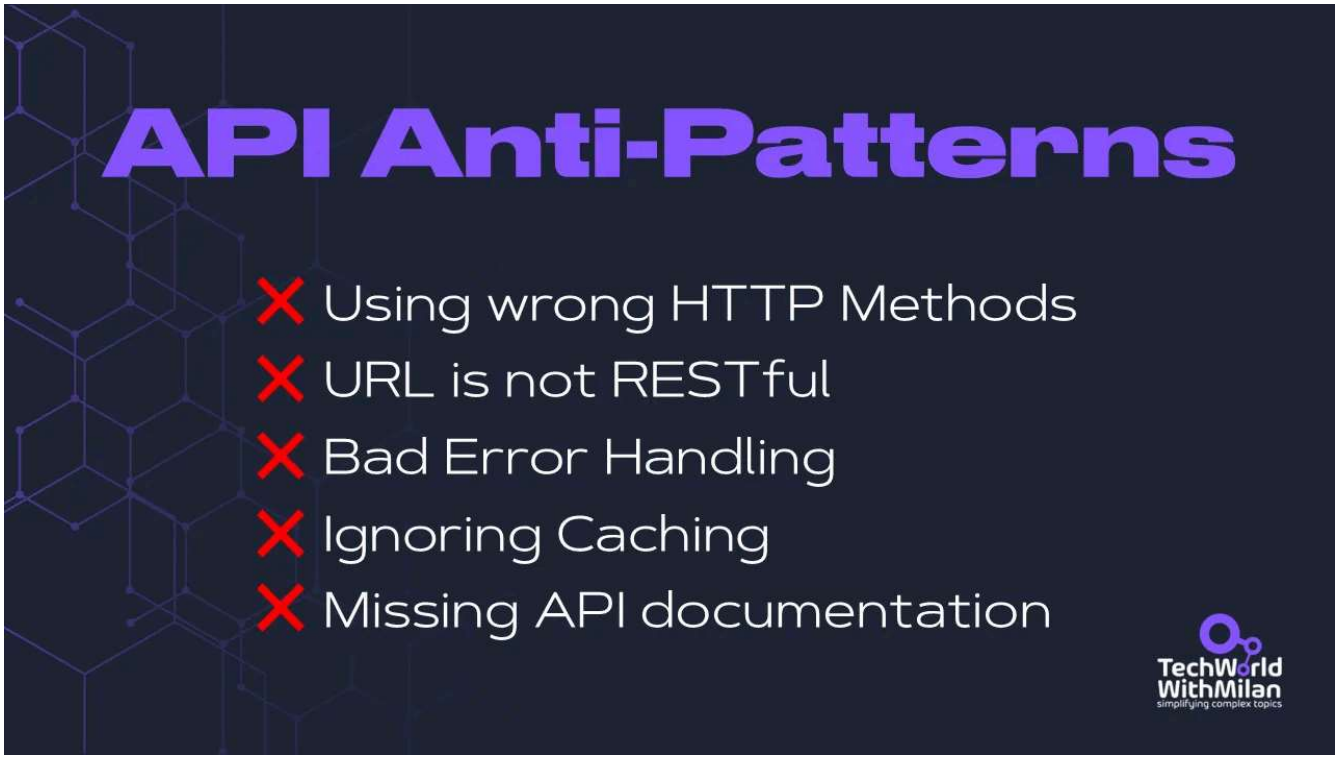
For example, we can use HTTP caching headers like ETag, Cache-Control, and Last-Modified to control how responses are cached by clients, which will increase application stability and performance.

```
HTTP/1.1 200 OK
Cache-Control: public, max-age=3600
ETag: "33a64df551425fcc55e4d42a148795d9f25f89d4"
Last-Modified: Wed, 21 Oct 2024 07:28:00 GMT
```

5. Missing API documentation

Proper documentation can help us understand how to interact with our APIs. We want to invest time in creating proper documentation covering all API aspects, including endpoints, parameters, data models, error codes, and examples of typical requests and responses. We can use tools like Swagger (OpenAPI Specification) to generate interactive documentation.

Also, today, we can use **AI tools to generate documentation from your API's codebase or annotations**.



API Anti-patterns

More ways I can help you

1. **[LinkedIn Content Creator Masterclass](). In this masterclass, I share my strategies for growing your influence on LinkedIn in the Tech space. You'll learn how to define your target audience, master the LinkedIn algorithm, create impactful content using my writing system, and create a content strategy that drives impressive results.

2. **Resume Reality Check"**. I can now offer you a new service where I'll review your CV and LinkedIn profile, providing instant, honest feedback from a CTO's perspective. You'll discover what stands out, what needs improvement, and how recruiters and engineering managers view your resume at first glance.
3. **Promote yourself to 37,000+ subscribers** by sponsoring this newsletter. This newsletter puts you in front of an audience with many engineering leaders and senior engineers who influence tech decisions and purchases.
4. **Join my Patreon community**: This is your way of supporting me, saying "**thanks**," and getting more benefits. You will get exclusive benefits, including all of my books and templates on Design Patterns, Setting priorities, and more, worth $100, early access to my content, insider news, helpful resources and tools, priority support, and the possibility to influence my work.
5. **1:1 Coaching:** Book a working session with me. 1:1 coaching is available for personal and organizational/team growth topics. I help you become a high-performing leader and engineer 🚀.

143 Likes · 12 Restacks

← Previous Next →

Discussion about this post

Comments Restacks

Write a comment...

Egor Voronianskii 16 Nov

Liked by Dr Milan Milanović

Thank you for your article, Dr. Milan.

I would like to highlight that you consider RFC 7807 outdated in favor of RFC 9547.

LIKE (1) REPLY SHARE

1 reply by Dr Milan Milanović

Marcos F. Lobo 15 Nov

Liked by Dr Milan Milanović

About versioning of APIs.

In my opinion, versioning whether via URI, query parameter, or header, makes things complicated for the maintenance of the API.

Instead, using same URL and leveraging contract testing strategy, will help:

- Owners of the API to understand if some change they is breaking some user.

- Users will know when they have to adapt their client/s to the changes of the API

Sounds a bit drastic and there are occasions, like public APIs, were my proposal is not the best

LIKE (1) REPLY SHARE

1 more comment...