ChatGPT

TypeScript OpenAPI→Zod Codegen Tools

We identified the following tools that can generate Zod schemas (often with TS types/clients) from OpenAPI specs:

1. **typed-openapi (astahmer)** [1] – A CLI/library that generates a single-file TypeScript API client from an OpenAPI spec, with optional **Zod** (and other) runtime validators. **Features:** Headless client (bring-your-own fetch/axios), optional TanStack Query wrappers, discriminated union error handling, and choice of runtime (none, Zod, Yup, io-ts, etc.) [1]. It focuses on a *subset* of spec for IDE speed (unlike full spec generators) [2]. **Differences:** Unlike full-fledged generators, it outputs minimal code (single file) and emphasizes speed and IDE auto-complete. **Pros:** Instant type suggestions (when using `--runtime none`), easy setup for React/Next/TanStack Query (via `--tanstack` flag) [3] [4]. **Cons:** Limited support for advanced OpenAPI features (e.g. `regex`, file upload, advanced schemas) by design [5]. Integrates with React (TanStack Query), Next.js, or any JS/TS app via fetch/axios. Zod schemas (if `--runtime zod`) are generated via [TypeBox-Codegen] under the hood; one can also skip client generation (`--schemas-only`) to only emit Zod schemas [6].

2. **Orval** [7] – A mature CLI that generates TypeScript clients and Zod schemas (via axios or fetch) from OpenAPI. **Features:** Supports multiple output modes (single/multi-file), HTTP clients (Axios, Fetch), and can emit Zod validators directly (with `output.client: "zod"` in config). Steve Kinney notes: *"We can use a tool called orval to generate Zod schemas from an OpenAPI specification"* [7]. **Differences:** Orval is highly configurable (supports mocking, React Query plugin, Angular/Nuxt plugins, etc.) and is well-maintained. **Pros:** Very stable, supports React Query hook generation, MSW mocking, and multiple frameworks (Next.js plugin available) [8]. **Cons:** Large configuration surface and possibly heavy output. (Some users find the learning curve steep.) Integrates well with React (TanStack Query), Next.js (it has a Next.js/TS fetcher plugin), Vue, etc., thanks to its plugin ecosystem.

3. **openapi-zod-client (Astahmer/Zodios)** [9] – A CLI that reads an OpenAPI spec and generates a Zod-powered HTTP client using Zodios. **Features:** Produces a fully typed Zodios client (Axios-based by default) with Zod validation for each endpoint [9]. It also exports all schemas as Zod validators if desired. **Differences:** Focused on generating a ready-to-use client (rather than just types or schemas) with Zod validation. **Pros:** Automatically infers request/response schemas to build both TS types and runtime Zod checks, ensuring safety. It's actively tested against standard OpenAPI examples. **Cons:** Heavier output (client code can be large for big specs), and initial generation may be slower for large specs. Integrates in any TS project (e.g. Next.js/React) as a drop-in client (no framework-specific hooks by default). Because it targets Zodios, it's best for Node/React apps using Axios; no special support for tRPC (you'd still get plain client functions).

4. **openapi-to-zod-schema** [10] – A simple CLI/library that *only converts OpenAPI component schemas* (JSON Schemas) into Zod schemas. **Features:** Input is an OpenAPI or JSON Schema file, output is Zod schema code. It can emit multiple files (one per schema). **Differences:** It does *not* generate clients or TS types; it focuses exclusively on schema definitions. **Pros:** Lightweight and straightforward for cases where you only need Zod validators from an existing spec. **Cons:** No client code; limited

OpenAPI support (only schemas, no path parameters or operations). Integrates with any TS workflow – use it as a pre-build step to produce validators for Express, Fastify, Next.js API routes, etc.

5. **openapi-to-zod (@ibabkin)** [11] – A wrapper around json-schema-to-zod that **extracts JSON Schemas from an OpenAPI spec and generates Zod schemas** [11]. **Features:** CLI tool (`openapi-to-zod`) that generates one Zod file per OpenAPI schema (optionally TypeScript). It recognizes OpenAPI components and `$ref`. **Differences:** Similar goal to openapi-to-zod-schema but uses a different underlying engine and output style (often generating separate files per component) [11]. **Pros:** Can serve both Zod schema and TypeScript type needs (techsparx notes it generates Zod schemas plus infers types from them) [11]. **Cons:** Configuration and file naming are less flexible (each schema in own file, sometimes verbose). Integrates anywhere: you can import the generated Zod objects into Next.js API routes or Express/Fastify handlers.

6. **Hey API – openapi-ts with Zod plugin** [12] – Hey API's (`@hey-api/openapi-ts`) OpenAPI → TypeScript generator, now offering a first-party **Zod plugin**. **Features:** Fully featured OpenAPI-to-TS tool (clients, SDKs, validators) with an official plugin to generate Zod v4 schemas for requests, responses, and definitions [12]. The Zod plugin is *seamlessly integrated*, simply enabling `plugins: ['zod']` in the config [13]. **Differences:** Unlike others, this is part of a larger ecosystem (next-gen OpenAPI TS generator with many plugins). **Pros:** Generates Zod schemas that work smoothly with Hey API's TS clients or standalone; supports validators, transformers, and uses Zod's `.meta()` for OpenAPI docs if desired. **Cons:** Hey API is relatively new (though well-starred) and requires a config file; some advanced Zod customization requires digging into plugin options. Integrates out-of-box with React/Next/Express via the general TS output. (Plugins exist for Axios, Fetch API, TanStack Query, etc., so you can generate a client alongside your Zod schemas.)

7. **Kubb** [8] – A comprehensive **API toolkit** that can generate TypeScript types, React Query hooks, MSW mocks, and **Zod schemas** from OpenAPI. **Features:** Plugin-based; by using `@kubb/plugin-oas`, `@kubb/plugin-ts`, and `@kubb/plugin-zod`, one can generate TS models *and* Zod validators [14]. The landing page touts "OpenAPI to TypeScript, React-Query, Zod..." [8]. **Differences:** Instead of a single-purpose CLI, Kubb is a configuration-driven engine; it can also integrate with React Query, Faker, etc. **Pros:** Extremely flexible: can group schemas by tags, customize naming, and even attach Zod `.openapi()` metadata. Good for projects needing an all-in-one generator (models, schemas, clients, mocks). **Cons:** Quite complex config; overkill if you only need a quick schema conversion. Integrates well with React (it has plugins for react-query, swr, solid-query, etc.) and supports generating MSW handlers. Not tied to a specific framework, but its Angular plugin pipeline also exists via normal plugin usage.

8. **@ng-openapi/zod (Angular plugin)** [15] – A Zod schema plugin for the Angular-focused `ng-openapi` generator. **Features:** When using `ng-openapi` (an Angular HTTP client generator), adding the `@ng-openapi/zod` plugin will "generate Zod schemas from OpenAPI specifications" [15]. **Differences:** Specifically aimed at Angular projects: it outputs Zod validators alongside Angular services. **Pros:** Integrates OpenAPI validation into Angular+Zod apps (e.g. using Angular's HttpClient). **Cons:** Niche (Angular-only ecosystem). Must use `ng-openapi` CLI.

9. **@openapi-codegen/cli + ts-to-zod** [16] [17] – While not a single tool, this combo is worth noting. The `@openapi-codegen/cli` (Fabien0102's TS generator) produces clean TS types from OpenAPI, and

the `ts-to-zod` package can convert those types into Zod schemas. As one blog describes, *"use `openapi-to-zod` to generate Zod schemas, then use `@openapi-codegen/cli` to generate TypeScript types"* [16] (and `ts-to-zod` to turn those types into Zod) [17]. **Features:** `openapi-codegen` is very configurable (supports output folders, custom templates, etc.), and `ts-to-zod` generates Zod validators from TypeScript declarations. **Differences:** This is a two-step pipeline rather than a single command. **Pros:** Modern tooling with good TS codegen (JSDoc support) [16]. **Cons:** Indirect – you need to run two tools (`openapi-codegen` then `ts-to-zod`), and `ts-to-zod` has known issues with certain defaults [17]. Integrations depend on the generated code: generally works anywhere TS does (e.g. Next.js, Express) but no framework-specific helpers.

10. **ts-to-zod** [17] – A generic TypeScript → Zod generator (open-source). **Features:** Reads TypeScript type definitions (e.g. those produced by an OpenAPI generator) and emits equivalent Zod schema code. **Differences:** It's not OpenAPI-aware itself; it's used in workflows (see above). **Pros:** Useful for enforcing that generated TS types are backed by runtime schemas. **Cons:** Doesn't handle everything (e.g. default values in nested schemas are imperfect [17]). It's experimental but open-source.

Each tool has trade-offs: for example, **typed-openapi** is simple and fast but supports only a subset of OpenAPI; **Orval** and **Hey API** are very powerful but have steeper learning curves; **openapi-zod-client** provides a ready-to-use Zodios client; and **openapi-to-zod-schema** or **openapi-to-zod** are lightweight if you only need schemas. Integration varies: most tools output plain TS/Zod code you can use in Next.js, Express, or any Node app, while Orval and Kubb offer tight React/Next/Angular support via plugins (e.g. React Query hooks) [8] [12]. In contrast, lighter tools (openapi-to-zod-schema, ts-to-zod) require manual wiring.

Sources: Official docs and repos of each tool [1] [7] [9] [10] [11] [12] [8] [15] [16] [17]. These confirm features, Zod support, and use-cases for each.

---

[1] Enhancing Type Safety and Validation with Zod: Use Cases, OpenAPI Integration, and Comparison with Joi | by Rehmat Sayany | Medium
https://rehmat-sayany.medium.com/enhancing-type-safety-and-validation-with-zod-use-cases-openapi-integration-and-comparison-with-53ed009c85c6

[2] [3] [4] [5] [6] GitHub - astahmer/typed-openapi: Generate a headless Typescript API client from an OpenAPI spec - optionally with a @tanstack/react-query client using queryOptions
https://github.com/astahmer/typed-openapi

[7] Generating Zod Schemas from OpenAPI | Full Stack TypeScript | Steve Kinney
https://stevekinney.com/courses/full-stack-typescript/generating-zod-openapi

[8] Kubb | The ultimate toolkit for working with APIs
https://kubb.dev/

[9] GitHub - astahmer/openapi-zod-client: Generate a zodios (typescript http client with zod validation) from an OpenAPI spec (json/yaml)
https://github.com/astahmer/openapi-zod-client

[10] openapi-to-zod-schema CDN by jsDelivr - A CDN for npm and GitHub
https://www.jsdelivr.com/package/npm/openapi-to-zod-schema

11 16 17 Autogenerating TypeScript types and data validation for OpenAPI Schemas
https://techsparx.com/nodejs/openapi/schema-types.html

12 13 Zod v4 Plugin | Hey API
https://heyapi.dev/openapi-ts/plugins/zod

14 @kubb/plugin-zod | Kubb
https://kubb.dev/plugins/plugin-zod/

15 @ng-openapi/zod - npm Package Security Analysis - Socket
https://socket.dev/npm/package/@ng-openapi/zod