



Search

Ctrl K

On this page

Ask Elysia (AI)

Open in



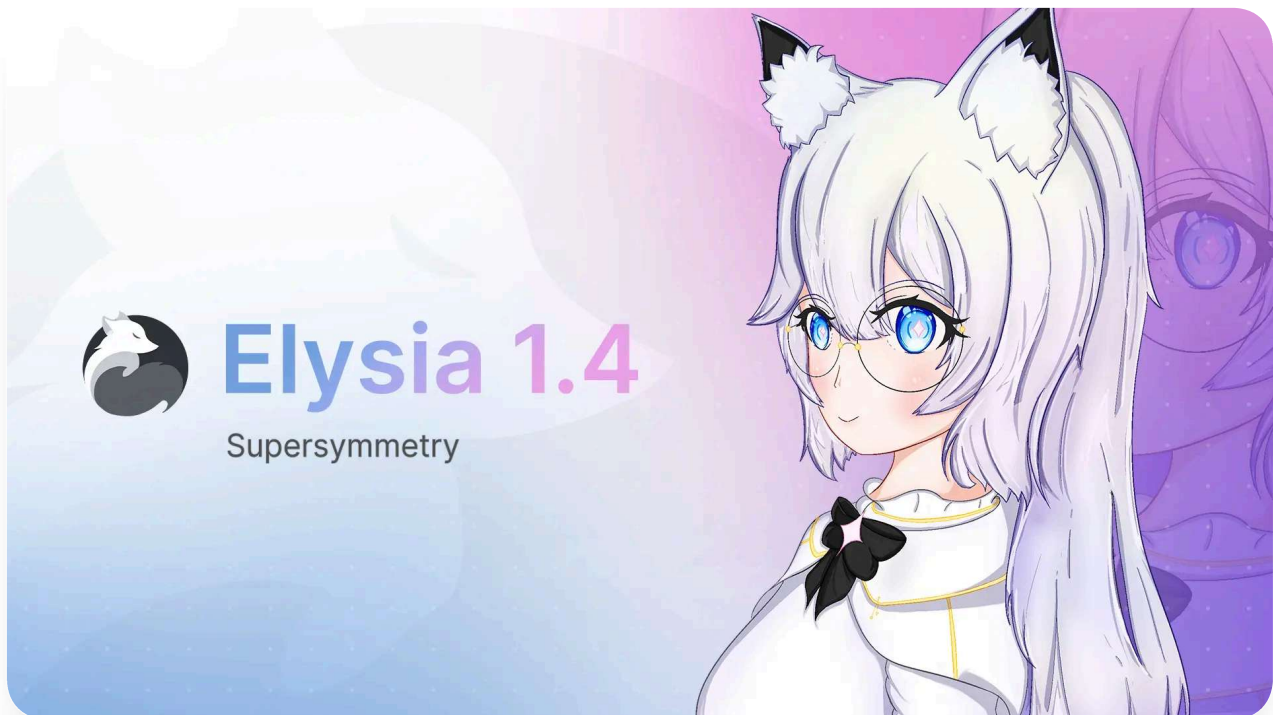
← Blog

Elysia 1.4 - Supersymmetry



saltyaom

13 Sep 2025 — @saltyaom



Named after the song [Supersymmetry](#) by Sta, a Tone Sphere ending theme.

The highlight of Elysia 1.4 is the introduction of Standard Schema and "Type Soundness".

- [Standard Schema](#)
- [Macro](#)
- [Lifecycle Type Soundness](#)
- [Group standalone schema](#)

Standard Schema

For 3 years, Elysia used TypeBox as the only validator. It became one of Elysia's most loved features due to its performance and type inference.

However, one of the most requested features from our community since the very beginning ([elysia#20](#)) was to support validators other than TypeBox.

Because Elysia was deeply tied to TypeBox, adding support for each validator individually required a lot of effort and ongoing maintenance to keep up with changes.

Fortunately, a new proposal called [Standard Schema](#) defines a standard way to use different schemas with the same API. This allowed us to support multiple validators without writing custom integrations for each one.

Elysia now supports Standard Schema, allowing you to use your favorite validators like:

- Zod
- Valibot
- Effect Schema
- ArkType
- Joi
- and more!

You can provide the schema in a way similar to TypeBox, and it will just work out of the box:

```
import { Elysia, t } from 'elysia'
import { z } from 'zod'
import * as v from 'valibot'

const app = new Elysia()
  .post(
    '/user/:id',
    ({ body, params }) => {
      body
```

ts

```
body: {
  name: "lilith";
}
```

```
      params

    },
    {
      params: z.object({
        id: z.coerce.number()
      }),
      body: v.object({
        name: v.literal('lilith')
      })
    })
  })
```

```
params: {
  id: number;
}
```

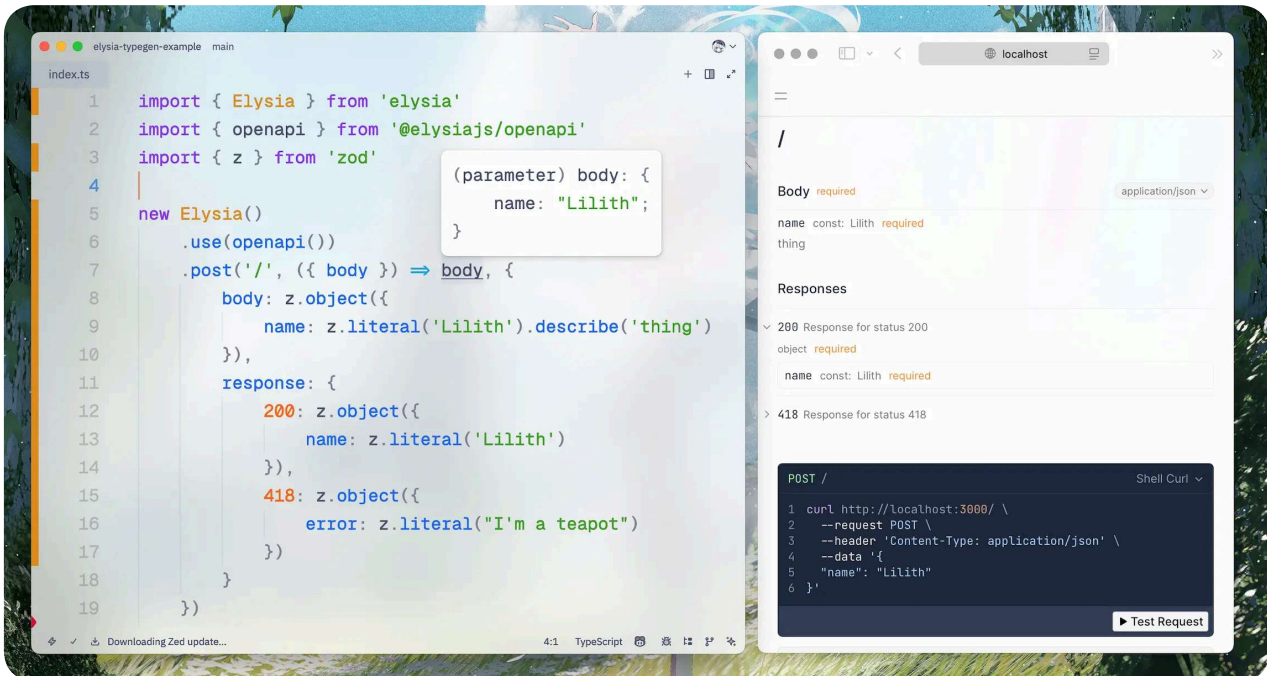
You can use multiple validators in a single route, and they will work together seamlessly with correct type inference.

OpenAPI

There is a request to support JSON Schema for OpenAPI generation with Standard Schema, but it is not implemented yet.

However, we provide a custom `mapJsonSchema` to `openapi` allow you to provide a custom function to map schema to Json Schema as a workaround.

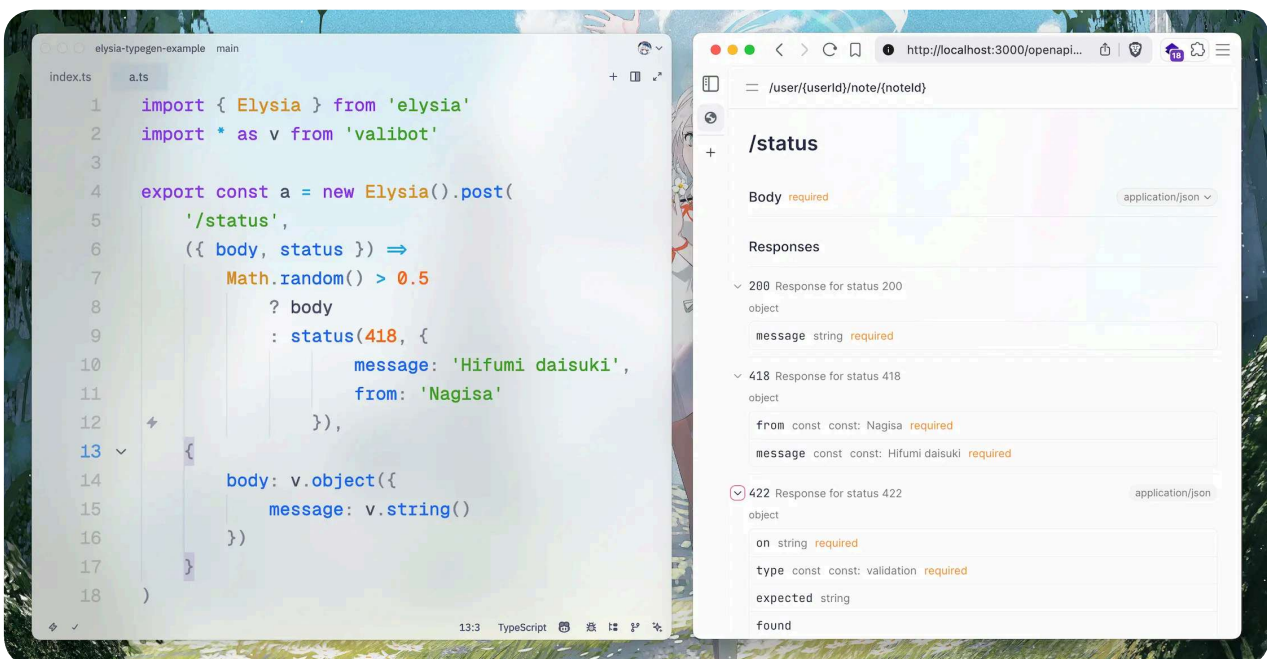
This allows you to have beautiful OpenAPI documentation with your favorite validator.



Using Zod's native OpenAPI schema support with **describe** to add description to the schema

But if your validator does not support JSON Schema, we provide **OpenAPI type gen** to generate OpenAPI schema directly from TypeScript type from your validator.

This means Elysia supports OpenAPI generation for all validators that support Standard Schema, even if they don't directly support JSON Schema.



Valibot doesn't directly support JSON Schema, but we use OpenAPI type gen to handle it

Not only will it generate correct input type, but OpenAPI type gen will also generate all possible output types, including error responses.

This is truly a unique feature for Elysia, and we are very proud to offer it.

Standalone Validator

You can also use multiple schemas to validate a single input using a standalone validator:

```
import { Elysia, t } from 'elysia'
import { z } from 'zod'
import * as v from 'valibot'
```

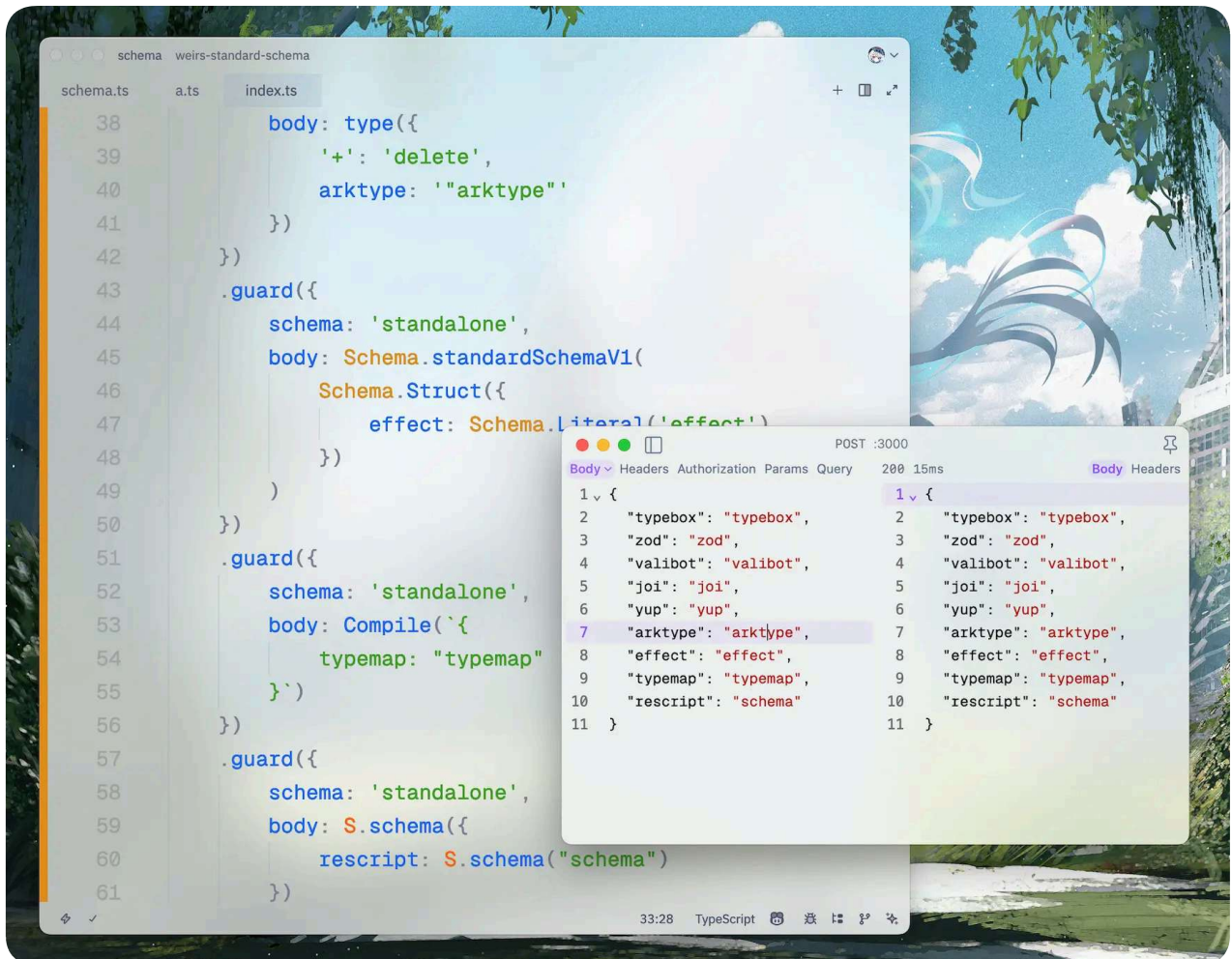
```
const app = new Elysia()
  .guard({
    schema: 'standalone',
    body: z.object({
      id: z.coerce.number()
    })
  })
  .post(
    '/user/:id',
    ({ body }) => body,

    {
      body: v.object({
        name: v.literal('lilith')
      })
    }
  )
```

```
body: {
  name: "lilith";
  id: number;
}
```

This example uses both Zod and Valibot to validate the body, allowing you to use existing schemas in your codebase from different validators together.

This works by using each validator to parse a part of the input, then storing each result as a snapshot that merges together to form a single output, ensuring type integrity.



Using TypeBox, Zod, Valibot, Joi, Yup, ArkType, Effect Schema, TypeMap, and ReScript Schema to validate different parts of the body

We tested 8 validators at the same time, validating each part of the input, and it just works flawlessly.

We are proud to support Standard Schema out of the box. This is a big step for Elysia to not be tied to a single validator, and we are excited to see what you will build with it.

Macro

Macro is one of the most powerful and flexible features of Elysia.

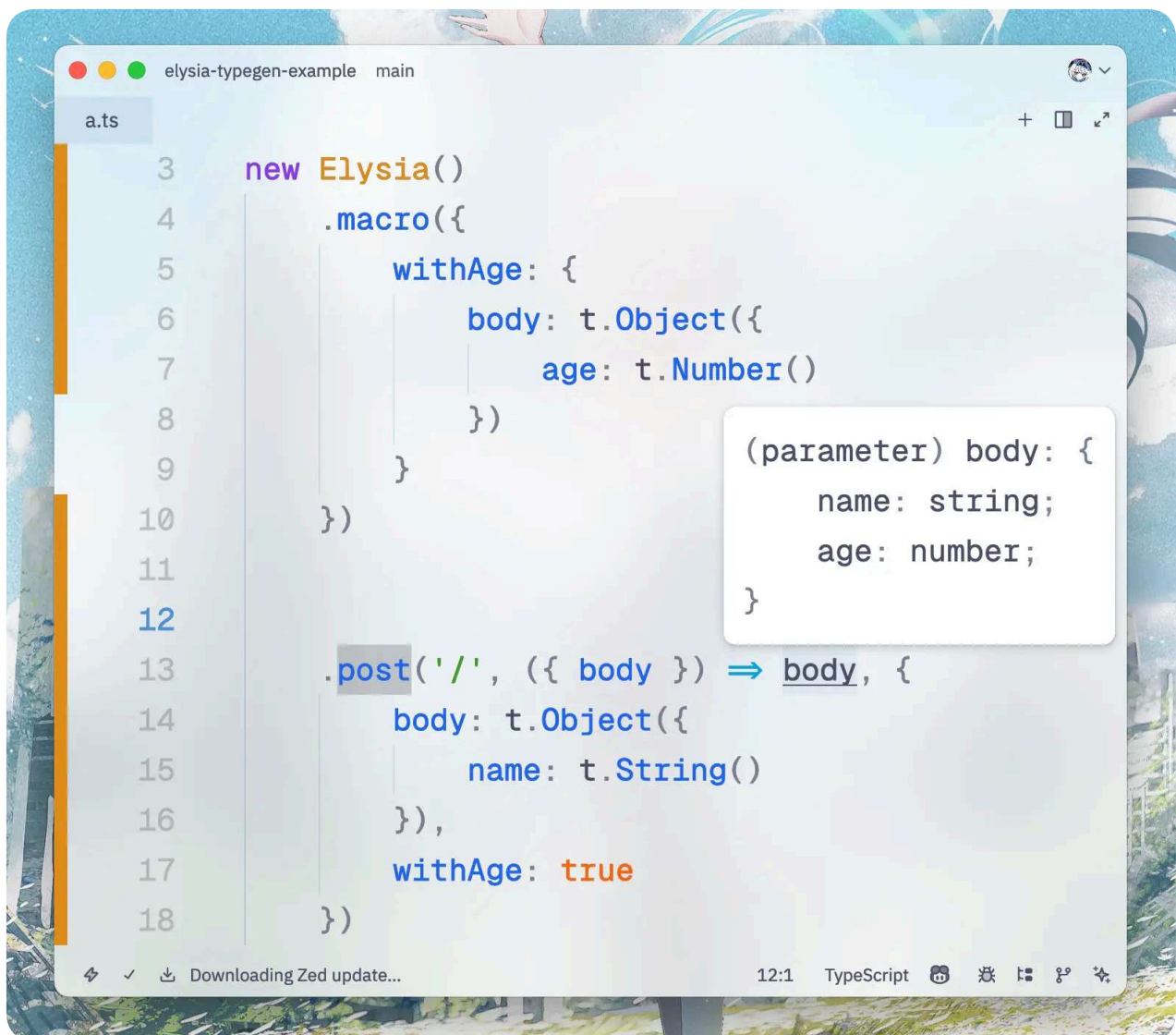
It allows you to define custom properties that can modify and extend the functionality of Elysia, enabling you to create your own "sub-framework" to your liking.

The versatility of Macro is truly amazing, letting you do things that are nearly impossible with other frameworks, effortlessly.

With Elysia 1.4, we bring several improvements to make macros even more versatile.

Macro Schema

You can now define a schema for your macro, allowing you to add custom validation directly from your macro.

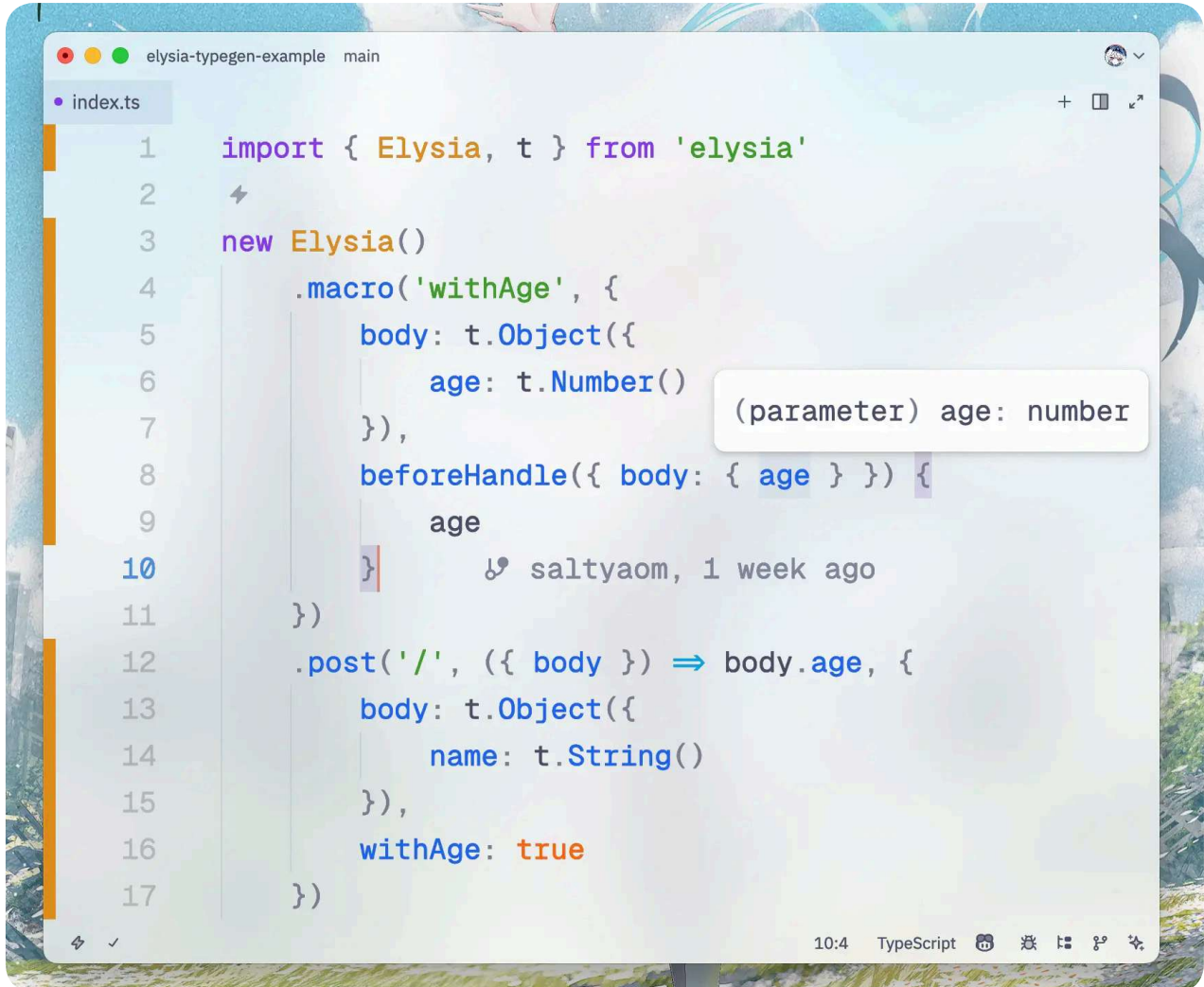


Macro with schema support

Macro with schema will automatically validate and infer types to ensure type safety, and it can coexist with existing schemas as well.

You can also stack multiple schemas from different macros, or even from Standard Schema, and they will work together seamlessly.

Macro schema also supports type inference for **lifecycle within the same macro** BUT only with a named single macro due to a TypeScript limitation.



```
1 import { Elysia, t } from 'elysia'
2 ⚡
3 new Elysia()
4   .macro('withAge', {
5     body: t.Object({
6       age: t.Number()
7     }),
8     beforeHandle({ body: { age } }) {
9       age
10    }
11  })
12  .post('/', ({ body }) => body.age, {
13    body: t.Object({
14      name: t.String()
15    }),
16    withAge: true
17  })
```

(parameter) age: number

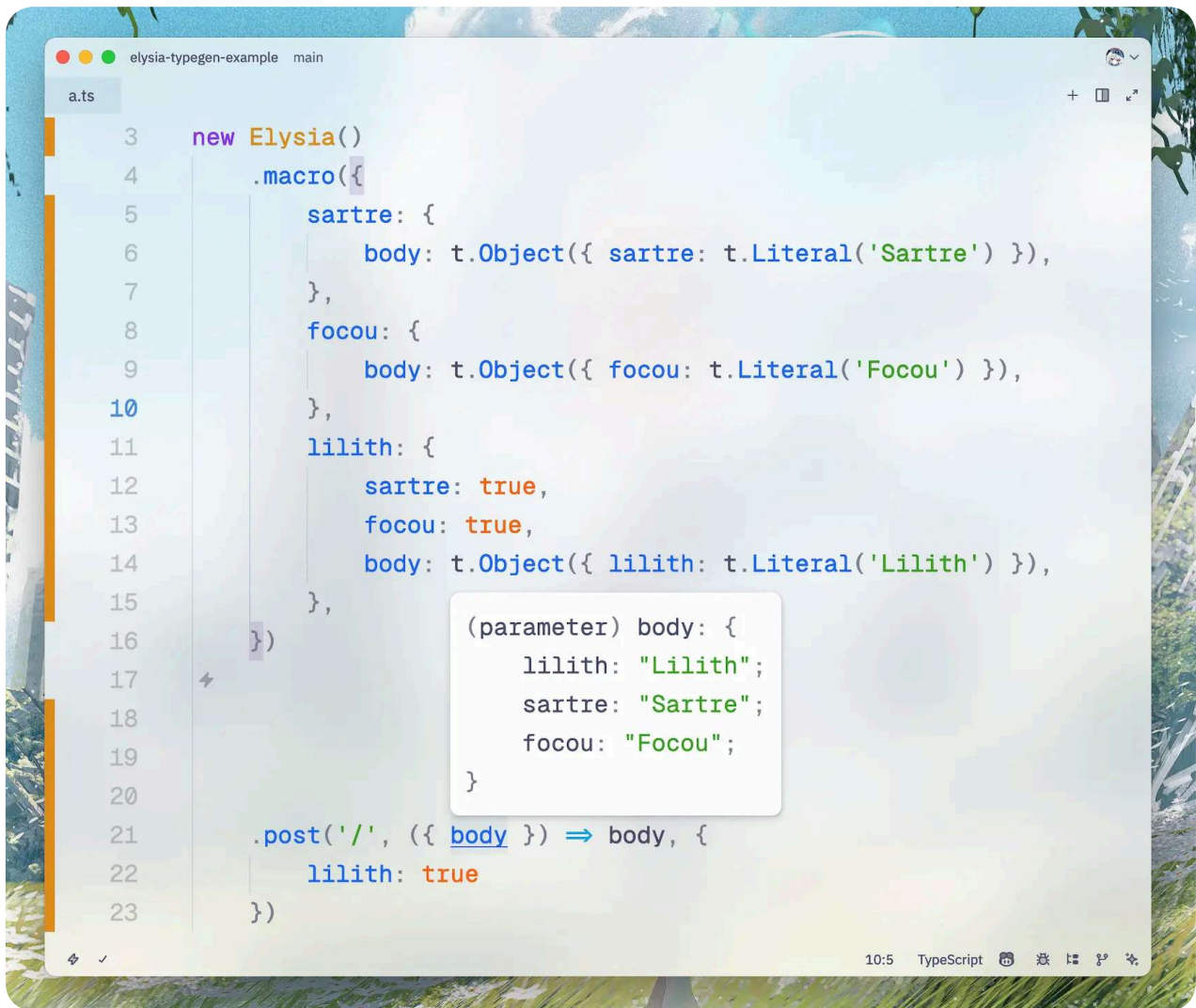
Using a named single macro to infer type into lifecycle within the same macro

If you want to use lifecycle type inference within the same macro, you should use a named single macro instead of multiple stacked macros.

Not to be confused with using macro schema to infer types into a route's lifecycle event. That works just fine—this limitation only applies to using lifecycle within the same macro.

Macro Extension

You can now extend existing macros, allowing you to build upon existing functionality.



Macro with extension support

This allows you to build upon existing macros and add more functionality to them.

It also works recursively with automatic deduplication, allowing you to extend existing macros that already extend other macros without any issues.

However, if you ever accidentally create a circular dependency, Elysia has a stack limit of 16 to prevent infinite loops in both runtime and type inference.

Macro Detail

You can now define OpenAPI detail for your macro, allowing you to add more detail to your OpenAPI documentation directly from your macro.

If the route already has OpenAPI detail, it will merge the detail together but prefers the route detail over macro detail.

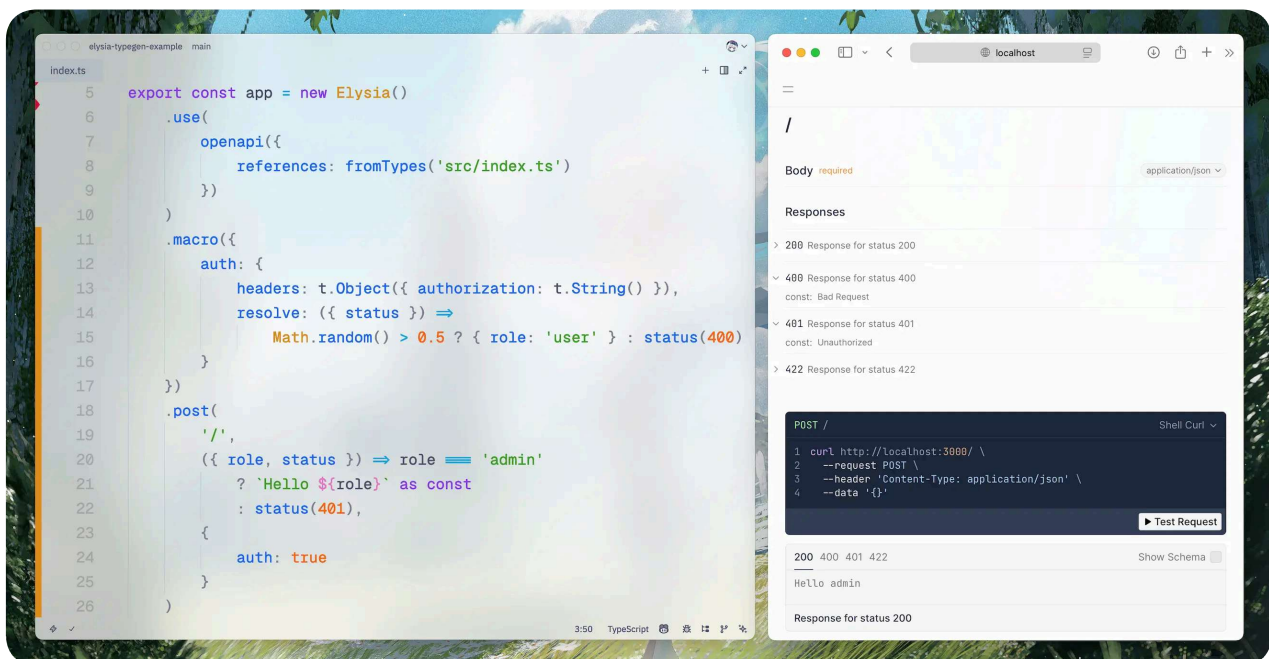
Lifecycle Type Soundness

Since the introduction of [OpenAPI Type Gen](#), which generates OpenAPI schema directly from types, we realized it would be great to have type soundness for every lifecycle event.

This way, we can accurately document the return type for each lifecycle event and macro, representing all the possibilities of what a single route can return.

By refactoring over 3,000+ lines of pure types, reconciling response status types, including unit tests at the type level for all lifecycle APIs to ensure type integrity, and optimizing type performance, we made sure that type inference doesn't get slower.

All of these complex achievements allow us to document all possibilities of what a single route can return.

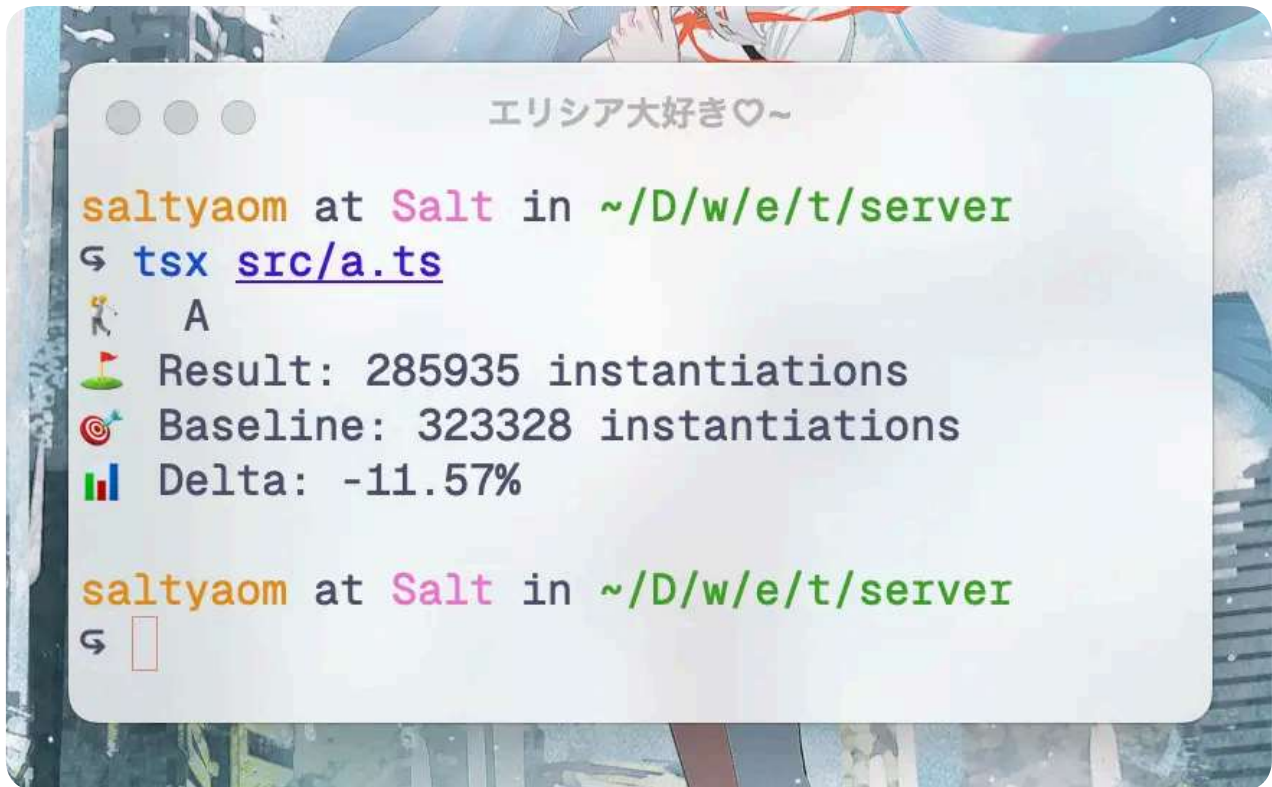


Documenting all the possibilities of what a single route can return

Not only does this improve the developer experience, but it also improves the reliability of your codebase by ensuring that all possibilities are accounted for in both the API documentation and the client with Eden Treaty.

Type Soundness covers all lifecycle events and macros, allowing you to have complete documentation of your API. The only exception is inline lifecycle events due to performance.

We also managed to improve type inference performance by ~9-11% and decrease type instantiation by 11.5%, despite the massive increase in type complexity.



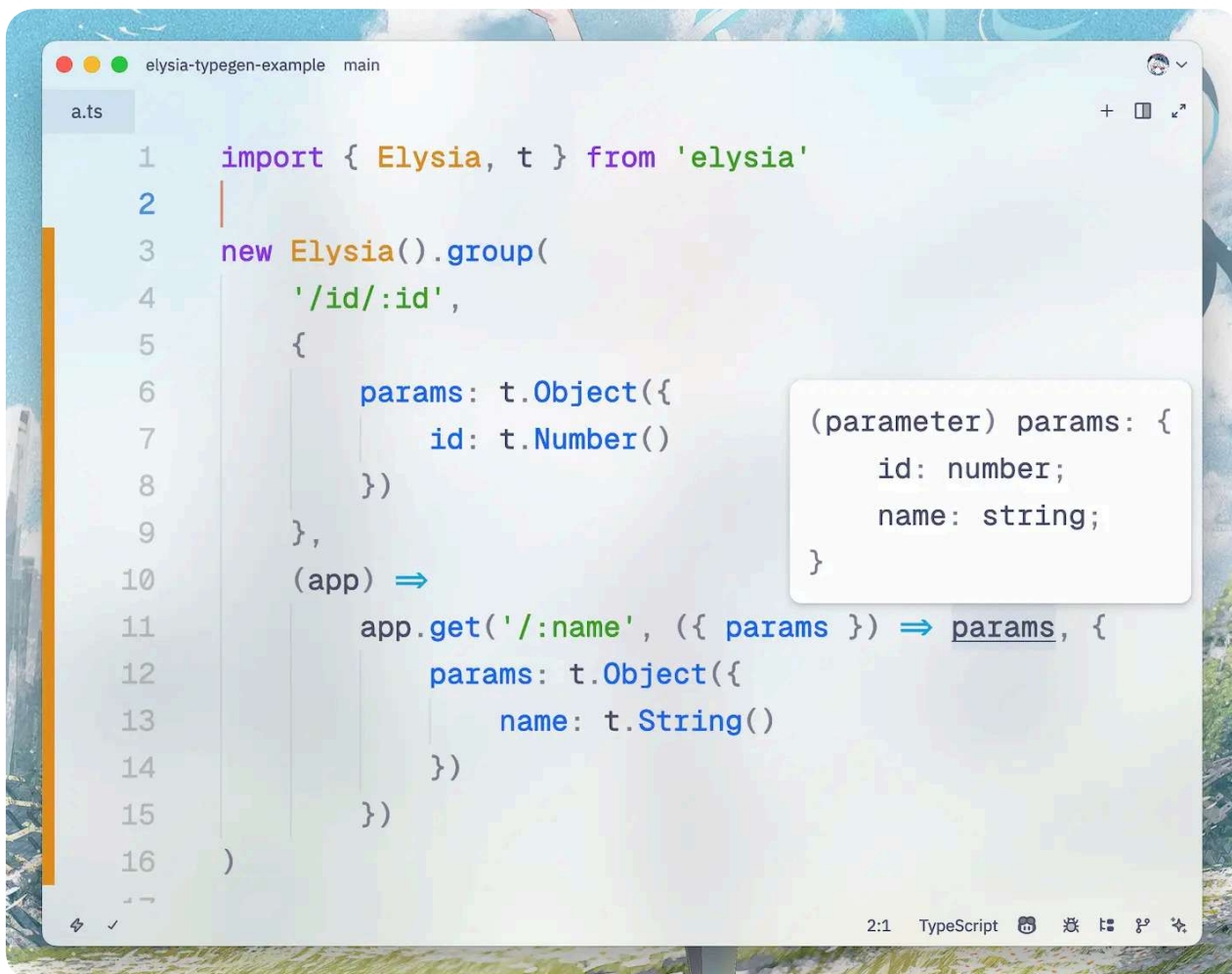
Type instantiation is reduced by 11.57% from our internal benchmark

Group standalone schema

Previously, `group` with schema used an overwrite strategy, meaning that if you defined a schema in `group`, it would overwrite the existing schema in the route.

If you wanted to define a new schema, you had to include the existing schema manually. This was not very ergonomic, and it could lead to errors if you forgot to include the existing schema.

Starting from 1.4, `group` with schema uses a standalone strategy, meaning that if you define a schema in `group`, it will not overwrite but will coexist with the route schema.



`group` with schema will coexist with route schema

This allows you to define a new schema in `group` without having to include the existing schema manually.

Notable changes

We closed around 300 issues in 1.3.9, so there aren't many bug fixes in 1.4—we've addressed most known issues.

Improvements

- [#861](#) automatically add HEAD method when defining GET route
- [#1389](#) NoValidate in reference model

Changes

- ObjectString/ArrayString no longer produce default values due to security reasons
- Cookie now dynamically parses when format is likely JSON
- export `fileType` for external file type validation for accurate response

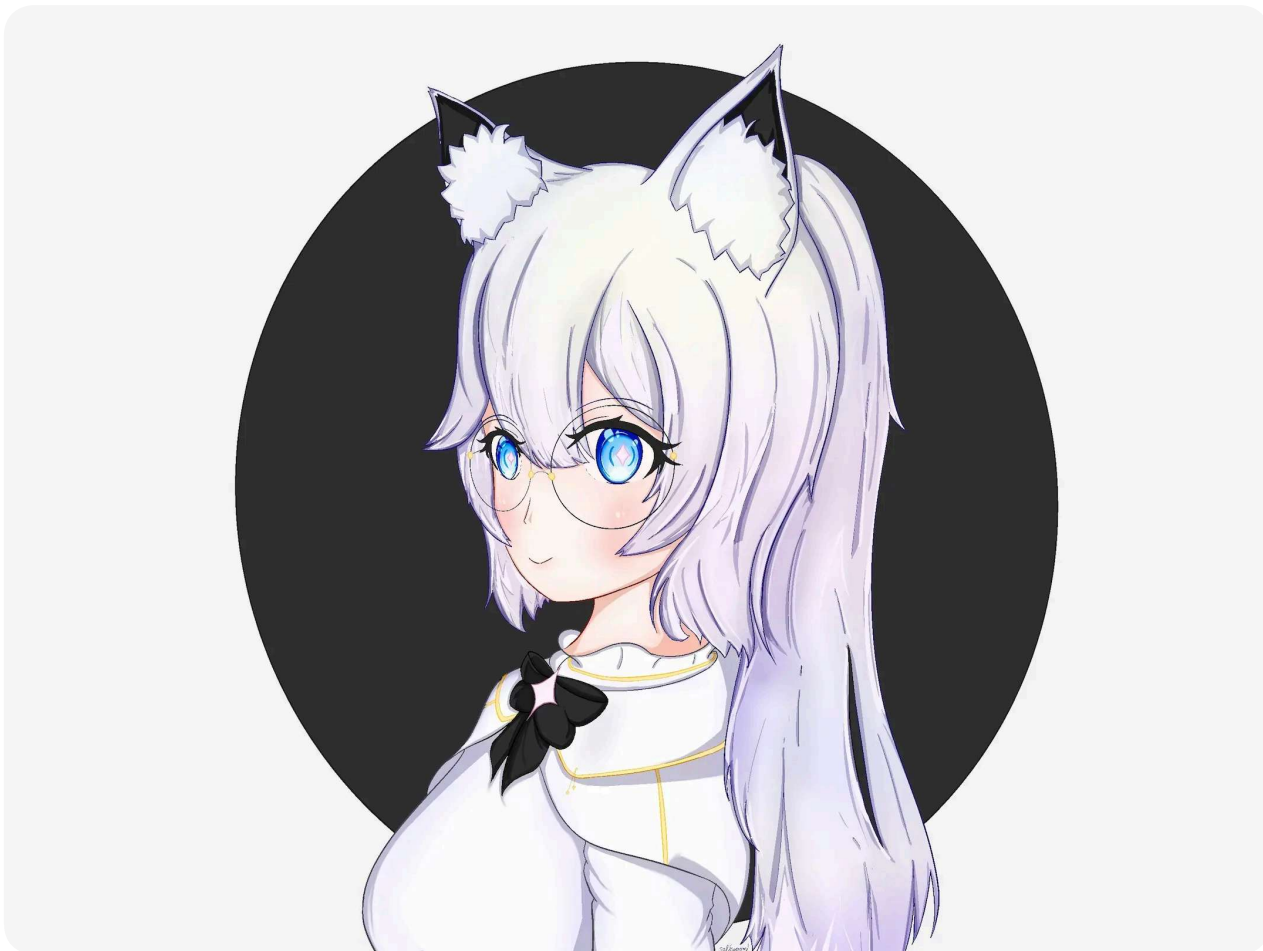
Breaking Changes

- remove macro `v1` due to lack of type soundness
 - remove `error` function; use `status` instead
 - deprecation notice for `response` in `mapResponse` , `afterResponse` ; use `responseValue` instead
-

Afterword

This is the first time we have featured our mascot, Elysia chan, as part of the release note cover image! This will become a tradition for future release notes as well!

Our cover art mirrors the theme of the Supersymmetry (music) cover art, where ElysiaJS chan is mirroring Weirs with a similar pose.



Elysia chan mirroring the same pose as Weirs from the Supersymmetry cover art ([pixiv](#)).

Isn't she so cute? I really love how she turned out! I personally worked hard to improve my art skills to be able to draw her. Hope you like it!

Anyway, I hope you like this release! We are excited to see what you will build with it!

Have a nice day!

I am all
In this tiny micro universe
Every morsel of your bittersweet heart
I loved all

You know
This game of life is our "riverrun,"
You were my lucky friend for this journey

There is not much time left
The sky is showing its fade

Stars parade

Time to change our fate

← Elysia: Ergonomic Framework for Humans



Elysia

Ergonomic Framework for Humans



Speed

Top Performance



Type Safety

Best in class



Developer Experience

Exceptional



OpenAPI Support

One of a kind

Get Started

Elysia in < 5 mins