



# Beyond Type Checking

Building Bulletproof TypeScript Applications



DevWorld Conference

# Today's Journey



## The Problem

Runtime type safety challenges



## The Solution

Runtime validation



## Implementation

Real-world examples



## Live Demo

End-to-end validation

# Joseph Anson

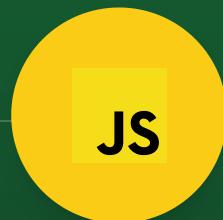
Senior Web Consultant at Passionate People  
TypeScript Expert & Developer Experience  
Advocate

 [josephanson.com](http://josephanson.com)

 [josephanson](https://github.com/josephanson)



# The Type Safety Journey



JavaScript's  
"Trust Me" Era



TypeScript  
Compile-Time Safety



Runtime  
Validation Gap

From implicit trust to bulletproof validation

# The Trust Boundary

## Trust Boundary



```
// Inside Frontend: TypeScript keeps us safe
const user: User = {
  id: '123',
  name: 'Alice'
} // ✅ Compile-time check

// Outside Frontend: We just hope these match
const response = await api.getUser()
const user: User = response.data // 🤯

const { id } = useRoute().query as { id: string } // 🤯

const apiKey = process.env.API_KEY as string // 🤯

const storage = localStorage.getItem('user') // 🤯
const user = JSON.parse(storage) as User // 🤯
```

# Real-World Impact

## Common Pain Points

- ✗ Missing required fields
- ✗ Incorrect data types
- ✗ Invalid enum values
- ✗ Malformed dates/timestamps
- ✗ Security vulnerabilities

## The Cost

- 🐝 Data-related bugs are costly
- ⌚ Significant debugging time
- 😡 User experience degradation
- 🔒 Potential security risks

```
// Real-world example
const userData = {
  id: '123', // ✓
  email: 'not-an-email', // ✗
  createdAt: 'Invalid Date', // ✗
  permissions: ['admin', 'invalid'], // ✗
  settings: {
    theme: 'dark',
    notifications: 'yes' // ✗ Should be boolean
  }
}

// TypeScript thinks this is fine
const user: User = userData

// Runtime consequences
user.email.toLowerCase() // ⚡ TypeError
new Date(user.createdAt) // ⚡ Invalid Date
```

# The Runtime Validation Gap

```
// What you expect
interface APIResponse {
  id: string
  loanStatus: {
    amount: number
    currency: 'USD' | 'EUR'
    status: 'pending' | 'approved' | 'rejected'
    interestRate: number
  }
  total: number
  createdAt: Date
}

// What you get
const response = {
  id: 12345, // 🤦 Number!
  loanStatus: null, // 🤦 Should be an object!
  total: '-10', // 🤦 Should be positive!
  createdAt: '2025' // 🤦 Should be a date!
}
```

# What's Out There?

## Validation Libraries

Joi

Yup

Valibot

Zod

Arktype



## Why Zod?

TypeScript-first design

Zero dependencies

Expressive API

Ecosystem integration

# Zod Fundamentals



```
// Schema Definition
const ProductSchema = z.object({
  id: z.string().uuid(),
  price: z.number().positive(),
  variants: z.array(
    z.object({
      size: z.enum(['S', 'M', 'L']),
    })
  )
})

// Type Inference
type Product = z.infer<typeof ProductSchema>
/*
{
  id: string
  price: number
  variants: {
    size: "S" | "M" | "L"
  }[]
}
```

```
// Runtime Validation - No thrown error
const result = ProductSchema.safeParse(data)
if (!result.success) {
  // Detailed error reporting
  console.log(result.error.format())
}
```

```
// Runtime Validation - Throws error
try {
  const result = ProductSchema.parse(data) // ✖ Error
}
catch (error) {
  console.error(error)
}
```

# Validation in Practice 🔧

```
// Environment Variables
const envSchema = z.object({
  DATABASE_URL: z.string().url(),
  PORT: z.number().min(1024).max(65535),
  NODE_ENV: z.enum(['development', 'production', 'test'])
})
envSchema.parse(process.env)
```

```
// Query Parameters
const querySchema = z.object({
  page: z.number().min(1).default(1),
  limit: z.number().min(1).max(100).default(10),
  search: z.string().optional()
})
querySchema.parse(req.query)
```

```
// Form Validation with @Shadcn-vue
const formSchema = z.object({
  username: z.string().min(3),
  email: z.string().email(),
  password: z.string().min(8)
})
const form = useForm({
  validationSchema: toTypedSchema(formSchema)
})
```

```
// API Response Validation
const apiSchema = z.object({
  data: z.array(z.object({
    id: z.string(),
    name: z.string(),
  })),
  meta: z.object({
    page: z.number(),
    total: z.number()
  })
})
apiSchema.parse(await fetch('/api/data'))
```

# Ecosystem Integration



## API Validation

Seamless integration with frameworks like Express, Fastify, Nitro to validate incoming requests.

## Frontend Safety

Type-safe forms with React Hook Form, FormKit, Veevalidate, Shadcn, etc.

## Single Source of Truth

Zod schemas can be used in the frontend, backend, and generated from your database schema.

## Type-safe API Clients

Auto-generate type-safe clients for your API with Zod.

## Generate Mocks from Schemas

Generate realistic mock data for testing and development.

## Type Inference

Type inference works with Zod schemas.

# Key Benefits



## Fewer Production Bugs

Runtime validation catches issues pre-deployment



## Faster Debugging

Detailed error paths & validation messages



## DevEx Improvement

Autocomplete & type safety across boundaries



1:1

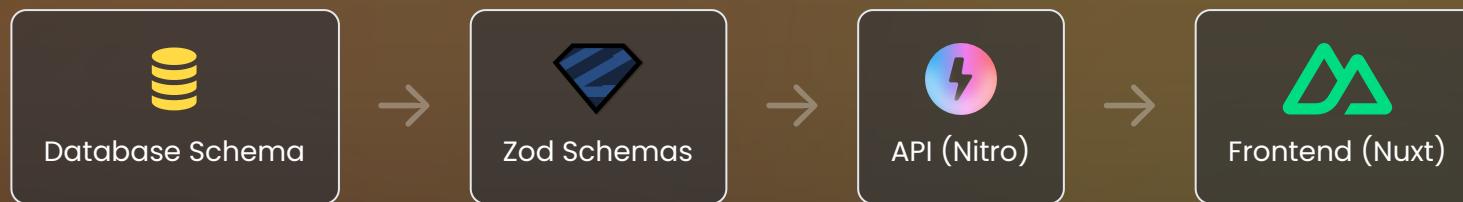
## Schema Parity

Single source of truth across all layers

# Live Demo 🚀

## End-to-End Type Safety

Scan QR for demo repo



**API Validation**  
Request/Response safety

**Frontend Safety**  
Form & API client types

**Schema Generation**  
DB → Zod schemas

# Thank You!

**Let's Build Safer Systems Together**

✉ josephanson.com | 🐾 josephanson

Slides & Resources:  
[josephanson.com/talks/beyond-type-checking](https://josephanson.com/talks/beyond-type-checking)

