Beyond Type Checking

Building Bulletproof TypeScript Applications

DevWorld Conference 2025

Today's Journey

The Problem
Compile-time type safety

The Solution
Runtime validation

Implementation
Real-world examples

Live Demo
Runtime Validation in Action

JS Type Safety Journey

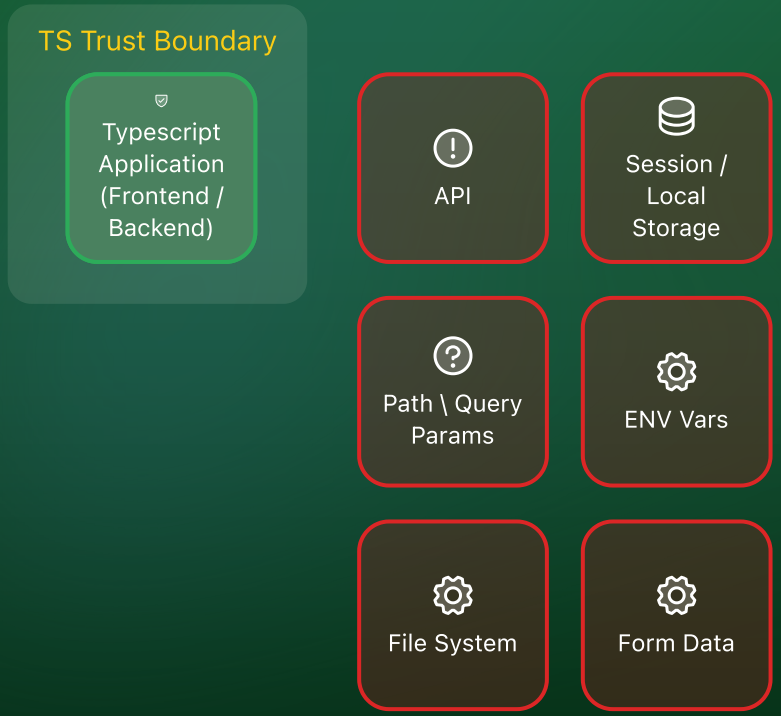
JavaScript's
"Trust Me" Era

TypeScript
Compile-Time Safety

Runtime
Safety Gap

The Trust Boundary

```
// JavaScript's "Trust Me" Era
const user = {
  id: '123',
  name: 'Alice'
} // Let's hope this is a user
```



Real-World Impact

Common Pain Points

- ❌ Type coercion errors in API responses
- ❌ Unexpected null/undefined values
- ❌ Invalid enum values from external systems
- ❌ Unable to access data due to security restrictions
- ❌ Cross-site scripting (XSS) from unvalidated data

The Cost

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

```typescript
// Example: API Response Validation
interface User {
  id: string
  email: string
  createdAt: Date
  role: 'ADMIN' | 'USER'
  preferences: { theme: 'light' | 'dark' }
}

// What we receive from API
const apiResponse: User = {
  id: '123', // ✅
  email: 'not-valid', // ❌ Invalid format
  createdAt: '2024-13-45', // ❌ Invalid date
  role: 'admin', // ❌ Wrong case
  preferences: { theme: 'blue' } // ❌ Invalid theme
}

// Runtime errors
apiResponse.email.includes('@') // 💥 Invalid email
new Date(apiResponse.createdAt) // 💥 Invalid date
apiResponse.role === 'ADMIN' // 💥 Case mismatch
```

The Runtime Validation Gap

```typescript
// Real-world example
interface LoanStatusResponse {
  id: string
  loanStatus: {
    amount: number
    currency: 'USD' | 'EUR'
    status: 'pending' | 'approved' | 'rejected'
    interestRate: number
  }
  total: number
  createdAt: Date
}
```

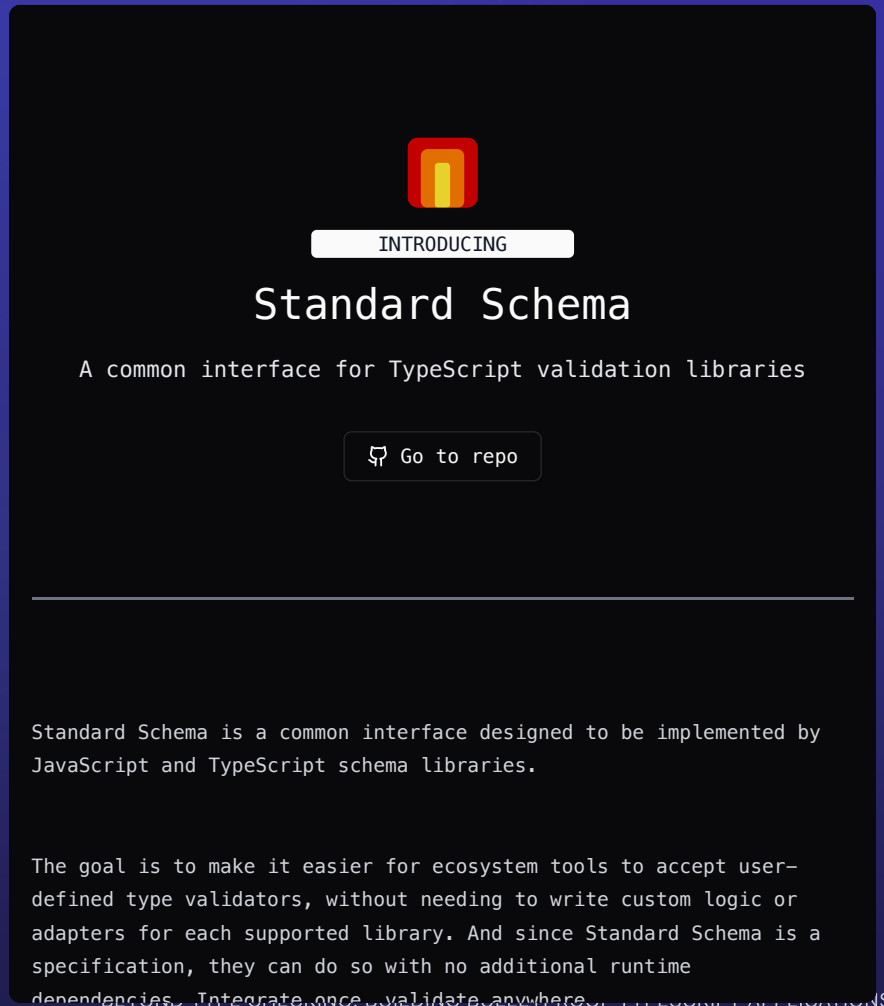
From Blind Faith to Bulletproof Validation

Building Trust Through Runtime Validation

What's Out There?

Schema Validation Libraries

- Zod
- Valibot
- Arktype



What is Standard Schema?

Collaborative Initiative

Created through collaboration between Zod, Valibot, and ArkType teams

Ecosystem Integration

Designed for seamless adoption across frameworks and tools

Universal Standard

Unified approach to schema validation across the JavaScript ecosystem

Community-Driven

Evolving best practices shaped by real-world implementation

Schema Validation

Why Zod?

- **Popular Choice:** Most widely adopted in the TypeScript ecosystem
- **Type Safety:** Seamless TypeScript integration
- **Developer Experience:** Intuitive API and excellent documentation
- **Performance:** Optimized for runtime validation
- **Ecosystem:** Rich set of utilities and community support

Schema Fundamentals

```
type Product = {
  id: string
  price: number
  variants: {
    size: "S" | "M" | "L"
  }[]
}
```

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

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

Developer Workflows

Schema-First Development

```
// 1. Define Schema
const todoSchema = z.object({
  title: z.string(),
  completed: z.boolean()
})

// 2. Define Update Schema
const updateTodoSchema = todoSchema.extend({
  title: z.string().min(1).max(100),
})

// 2. Generate Types
type Todo = z.infer<typeof TodoSchema>
type UpdateTodo = z.infer<typeof UpdateTodoSchema>
```

Implement Features

```
// 3. Implement Features
function getTodo(id: string) {
  const todo = fetch(`/api/todos/${id}`).then(res => re
  return todoSchema.parse(todo)
}

function updateTodo(id: string, data: UpdateTodo) {
  const validatedData = updateTodoSchema.parse(data)
  const updatedTodo = fetch(`/api/todos/${id}`, {
    method: 'PUT',
    body: JSON.stringify(validatedData)
  }).then(res => res.json())
  return updatedTodoSchema.parse(updatedTodo)
}
```

Schema 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 Veevalidate
const userSchema = z.object({
  username: z.string().min(3),
  email: z.string().email(),
  password: z.string().min(8)
})

const form = useForm({
  validationSchema: toTypedSchema(userSchema)
})
```

```
// API Response Validation
const apiSchema = z.object({
  id: z.string(),
  name: z.string(),
  createdAt: z.string().datetime(),
  updatedAt: z.string().datetime(),
})

const response = await fetch('/api/data')
const data = apiSchema.parse(await response.json())
```

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.

AI Data Generation

Use schemas to generate structured data with AI.

Tooling Support

API Frameworks

- H3
- Nitro
- tRPC
- Hono
- oRPC
- GQLoom
- express-zod-api

and more...

Form Libraries

- TanStack Form
- Formwerk
- Veevalidate
- Regle
- Superforms
- React Hook Form

and more...

UI Frameworks

- Qwik
- Nuxt UI
- Mage
- Primevue
- Shadcn
- Shadcn-vue
- renoun
- Nuxt Content
- Astro Content

and more...

HTTP Clients

- upfetch
- rest-client
- better-fetch
- make-service

and more...

Utilities

- T3 Env
- cachified
- UploadThing
- OpenAuth
- zod-schema-faker

and more...

Routing

- TanStack Router
- call-api
- Kitbag

and more...

Live Demo 🚀

Runtime Schema Validation

Scan QR for demo repo

Nuxt

Nitro

Zod

Vercel AI with Browser basedGemini

API Validation
Request / Response safety with Zod

Frontend Safety
Form validation with Zod

Data Generation
Generate Data using Zod and AI

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

Thank You!

Let's Build Safer Systems Together

Slides & Resources

josephanson.com | josephanson