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

End-to-end validation

JS Type Safety Journey

JavaScript's
"Trust Me" Era

TypeScript
Compile-Time Safety

Runtime
Safety Gap

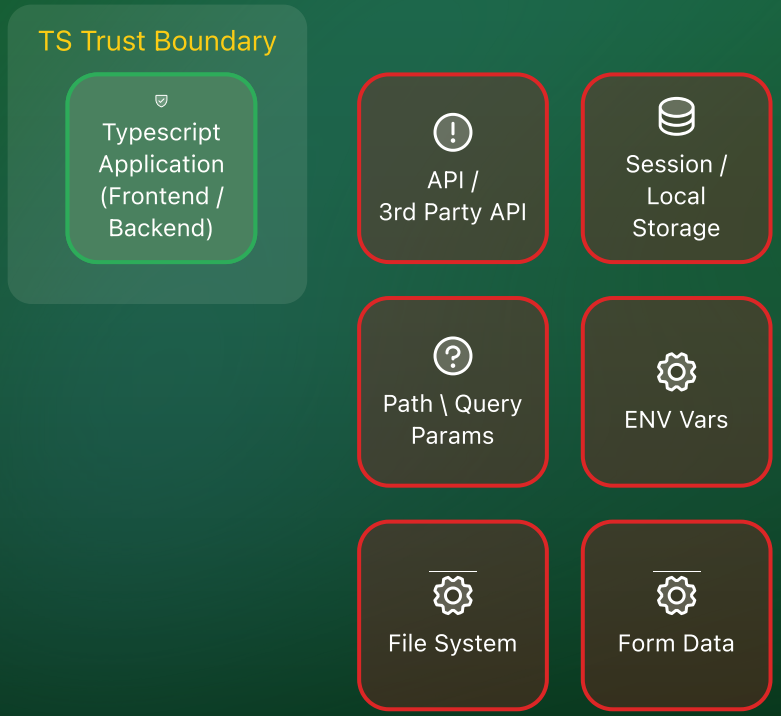
From Blind Faith to Bulletproof Validation

Building Trust Through Runtime Validation

Learn how to protect your TypeScript applications from runtime type errors

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 = {
  id: '123', // ✅
  email: 'not-valid', // ❌ Invalid format
  createdAt: '2024-13-45', // ❌ Invalid date
  role: 'admin', // ❌ Wrong case
  preferences: { theme: 'blue' } // ❌ Invalid theme
} as User // 🚨 Type assertion!

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

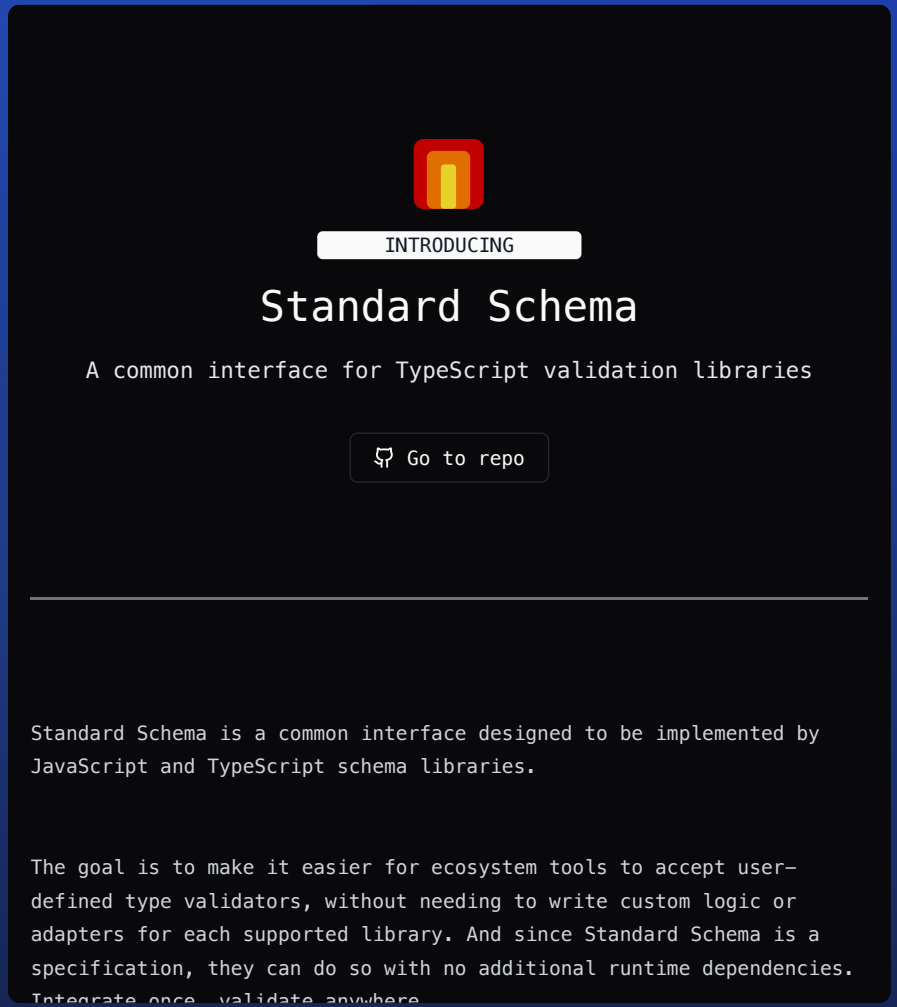
The Runtime Validation Gap

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

What's Out There?

Schema Validation Libraries

- Joi
- Valibot
- Yup
- Zod
- Arktype



Why Choose a library that supports Standard Schema? 🎯

- Backed by Zod, Valibot, ArkType
- Growing ecosystem adoption
- Industry standardization
- Shared best practices

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) // ❌ Error
  console.log(result) // type Product
}
catch (error) {
  console.error(error)
}
```

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({
  data: z.array(z.object({
    id: z.string(),
    name: z.string(),
  })),
  meta: z.object({
    page: z.number(),
    total: z.number()
  })
})
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

Form Libraries

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

UI Frameworks

- Qwik
- Nuxt UI
- Mage
- Shadcn
- renoun

HTTP Clients

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

Utilities

- T3 Env
- cachified
- UploadThing
- OpenAuth

Routing

- TanStack Router
- call-api

Live Demo 🚀

End-to-End Type Safety

Scan QR for demo repo

Nuxt

Nitro

Zod

Vercel AI
with Browser
basedGemini

API Validation
Request/Response safety

Frontend Safety
Form & API client types

Schema Generation
Generate Data using Schema and AI

Developer Workflows 👩‍💻

Schema-First Development

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

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
const getTodo = (id: string) => {
  const todo = db.getTodo(id)
  return todoSchema.parse(todo)
}

const updateTodo = (id: string, data: UpdateTodo) => {
  const updatedTodo = db.updateTodo(id, data)
  return updateTodoSchema.parse(updatedTodo)
}
```

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

Joseph Anson

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

josephanson.com josephanson

Thank You! 🎉

Let's Build Safer Systems Together

josephanson.com | josephanson

Slides & Resources:
josephanson.com/talks/beyond-type-checking