Form Validation in React with Zod & React Hook Form

Raja Krishna

Raja Krishna

Senior Software Engineer
@LOOP

LinkedIn (/rajakrishna)

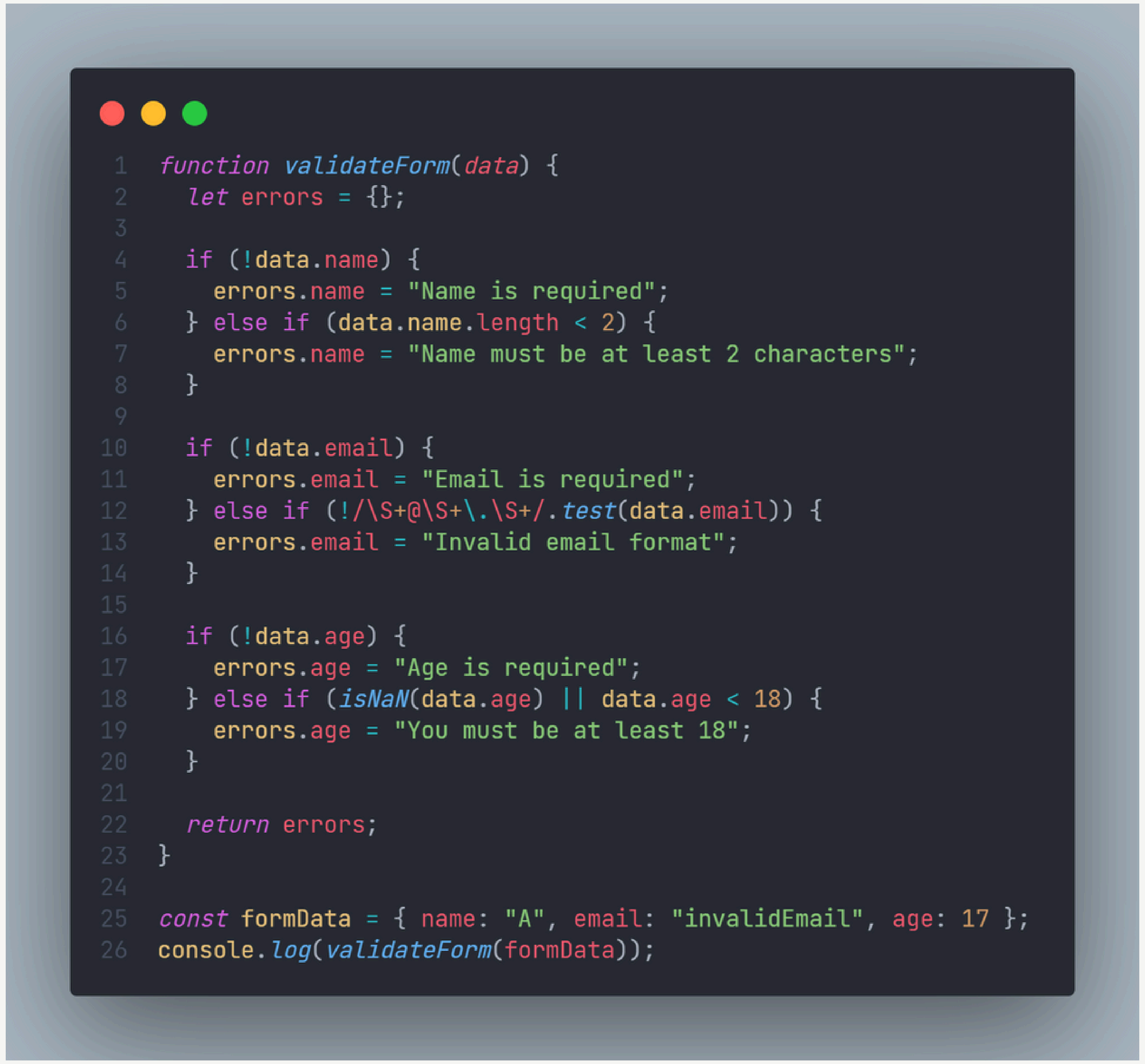
Problems with Traditional Validation

- **Common approaches and their drawbacks:**
- HTML5 validation
- Manual validation with useState
- **Problem:**
    - Managing validation rules separately from TypeScript types is painful.
    - Errors can be hard to track.
    - Performance suffers due to unnecessary re-renders.

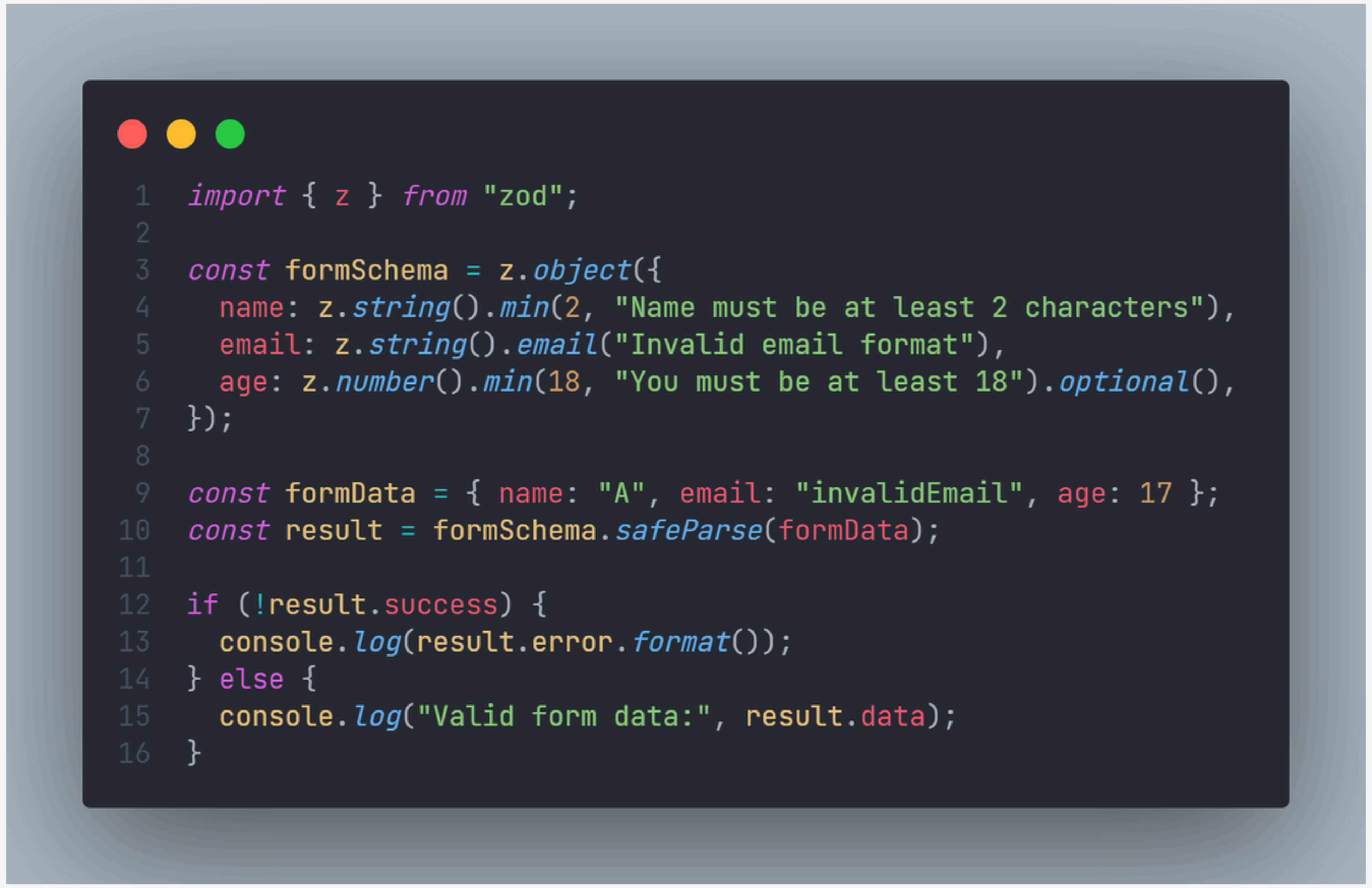
Zod & React Hook Form

- **What is Zod?**
  - A TypeScript-first schema validation library.
  - API validation, form validation, and runtime type checking
  - Works perfectly with React Hook Form.
- **Why React Hook Form?**
  - Optimized for performance (unlike useState-based validation).
  - Works well with controlled & uncontrolled components.
  - Simple API

Why Zod?



```javascript
function validateForm(data) {
  let errors = {};

  if (!data.name) {
    errors.name = "Name is required";
  } else if (data.name.length < 2) {
    errors.name = "Name must be at least 2 characters";
  }

  if (!data.email) {
    errors.email = "Email is required";
  } else if (!/\S+@\S+\.\S+/.test(data.email)) {
    errors.email = "Invalid email format";
  }

  if (!data.age) {
    errors.age = "Age is required";
  } else if (isNaN(data.age) || data.age < 18) {
    errors.age = "You must be at least 18";
  }

  return errors;
}

const formData = { name: "A", email: "invalidEmail", age: 17 };
console.log(validateForm(formData));
```

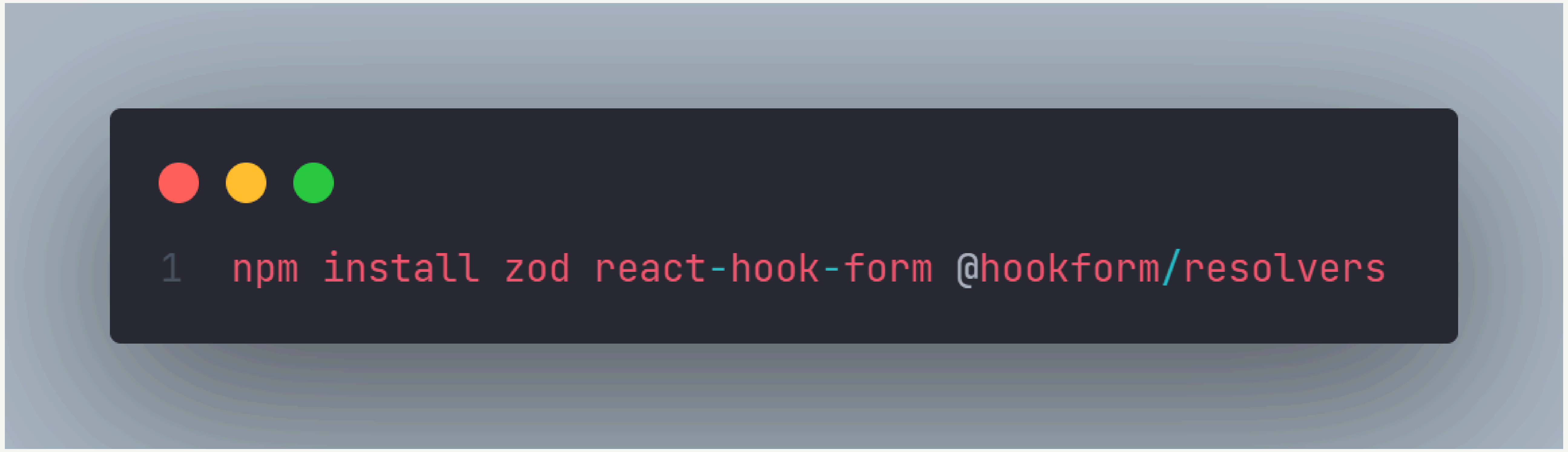


```javascript
import { z } from "zod";

const formSchema = z.object({
  name: z.string().min(2, "Name must be at least 2 characters"),
  email: z.string().email("Invalid email format"),
  age: z.number().min(18, "You must be at least 18").optional(),
});

const formData = { name: "A", email: "invalidEmail", age: 17 };
const result = formSchema.safeParse(formData);

if (!result.success) {
  console.log(result.error.format());
} else {
  console.log("Valid form data:", result.data);
}
```

RHF Resolvers

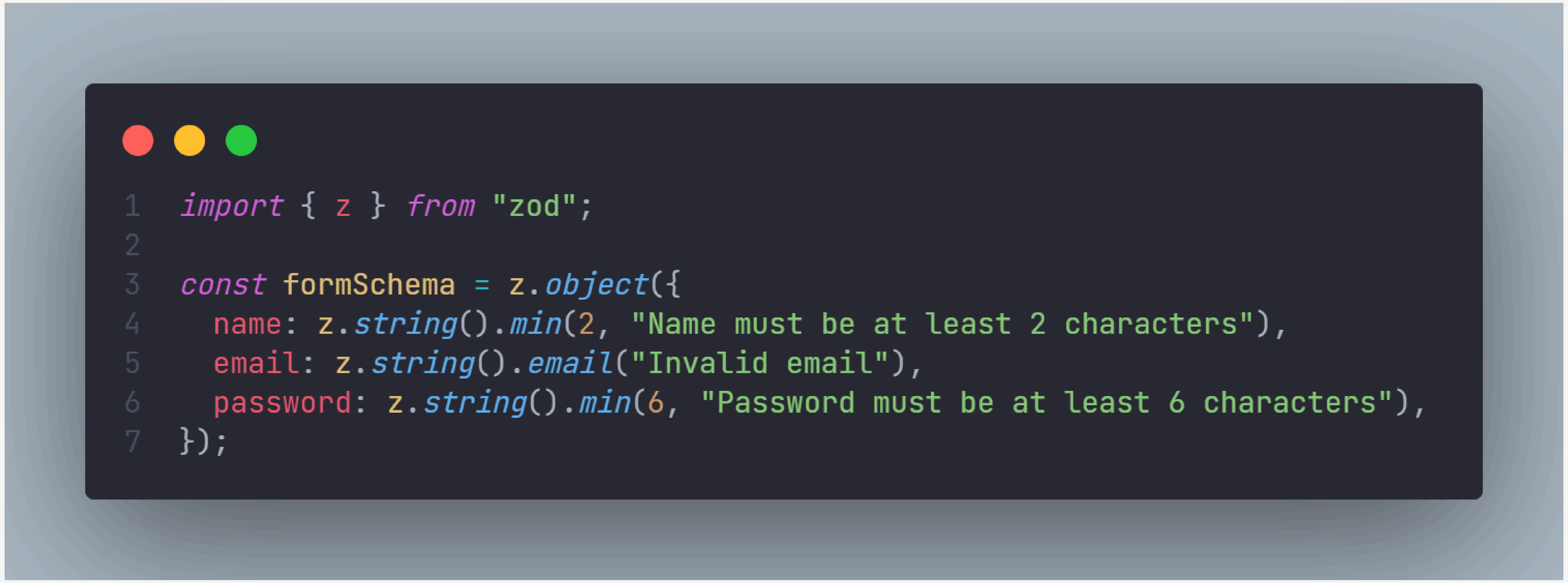
- Resolvers are functions that allow you to integrate external validation libraries (like Zod, Yup, Joi) with React Hook Form seamlessly.
- They act as an adapter between your validation schema and React Hook Form’s internal form handling.
- Resolvers ensure that validation is handled outside the form, but the errors are automatically passed into RHF’s validation flow.

Step 1: Install Dependencies



```
npm install zod react-hook-form @hookform/resolvers
```

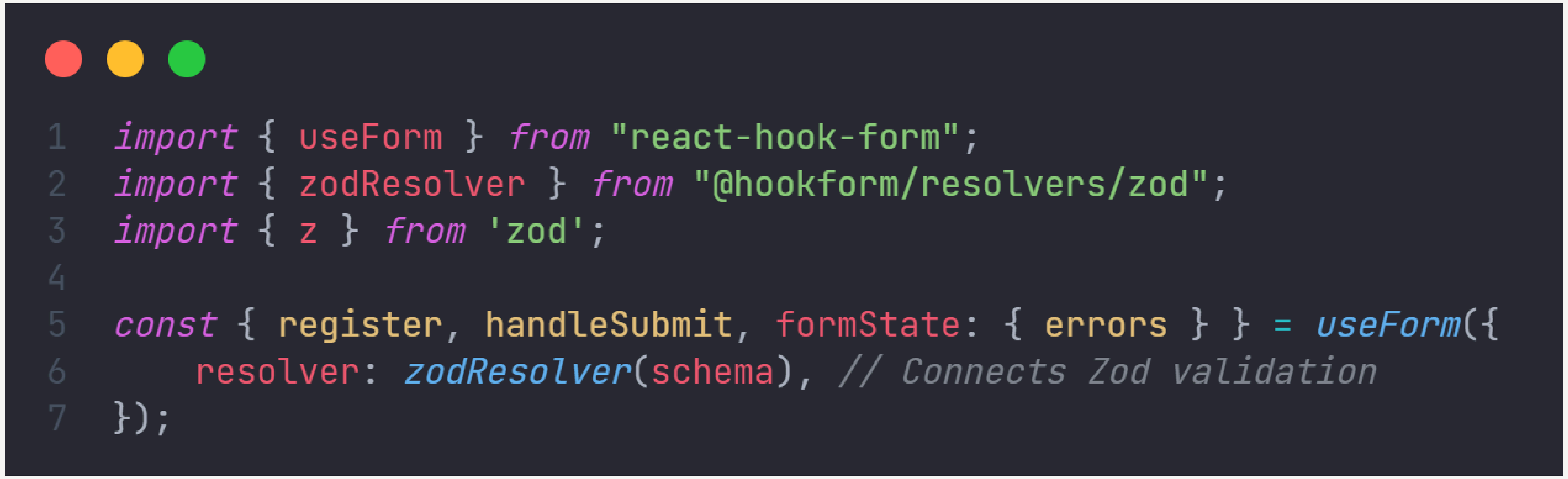
Step 2: Define a Schema with Zod



```
import { z } from "zod";

const formSchema = z.object({
  name: z.string().min(2, "Name must be at least 2 characters"),
  email: z.string().email("Invalid email"),
  password: z.string().min(6, "Password must be at least 6 characters"),
});
```

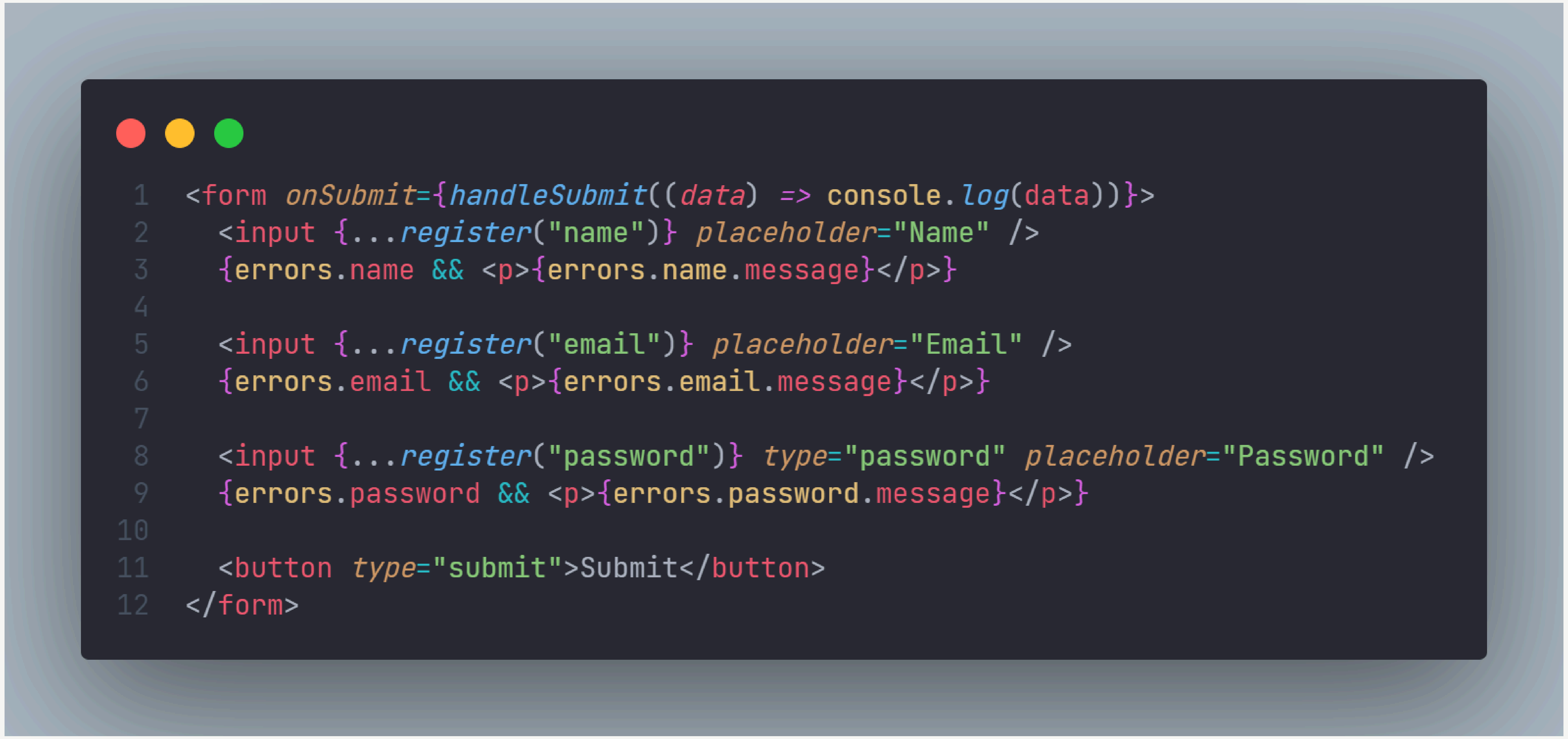
Step 3: Connecting Zod to React Hook Form

- React Hook Form (RHF) doesn’t handle validation by itself—it delegates it to a resolver.
- Resolvers act as a bridge between RHF and validation libraries like Zod, Yup, and Joi.



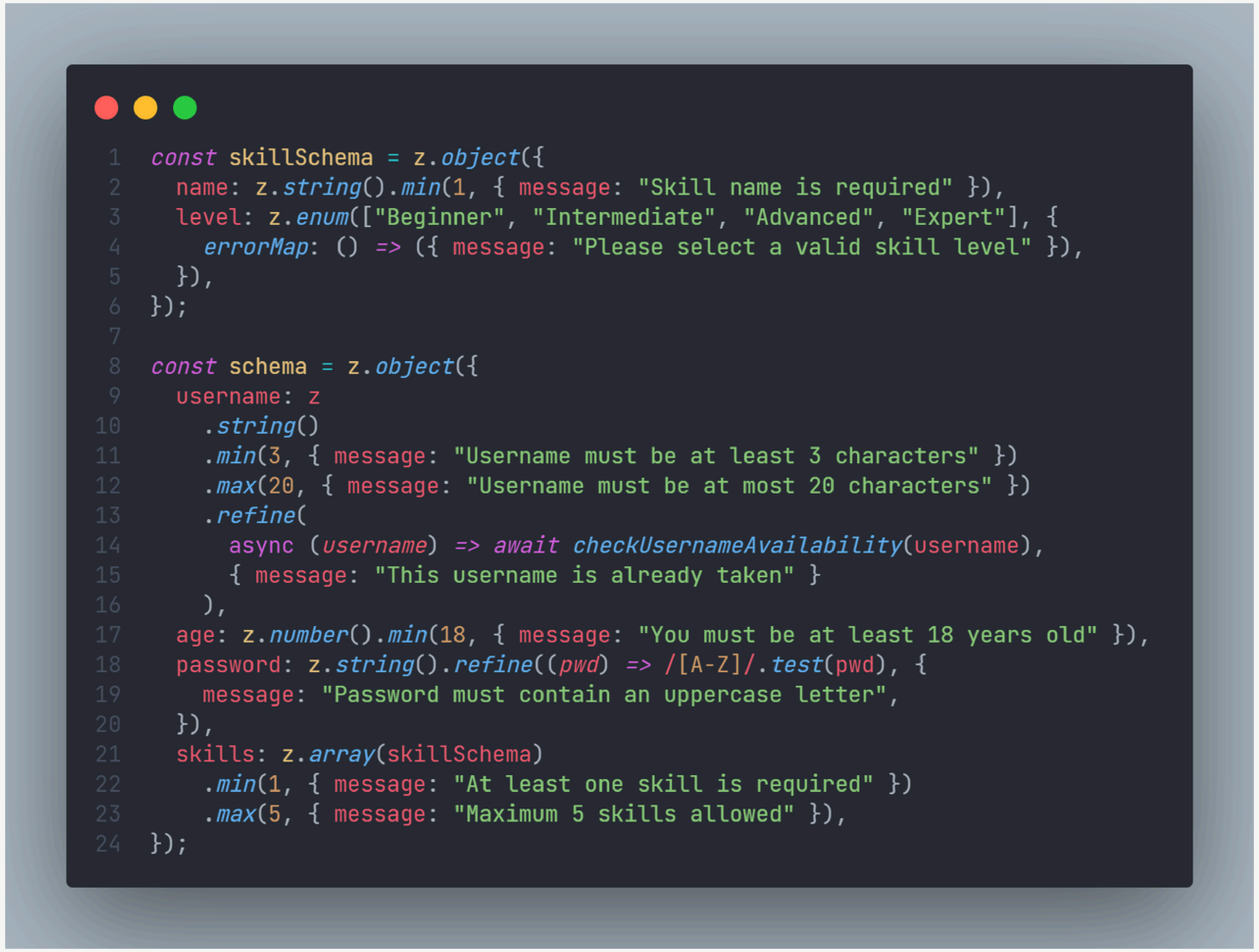
```
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { z } from 'zod';

const { register, handleSubmit, formState: { errors } } = useForm({
    resolver: zodResolver(schema), // Connects Zod validation
});
```

Step 4: Render the Form



```
<form onSubmit={handleSubmit((data) => console.log(data))}>
  <input {...register("name")} placeholder="Name" />
  {errors.name && <p>{errors.name.message}</p>}

  <input {...register("email")} placeholder="Email" />
  {errors.email && <p>{errors.email.message}</p>}

  <input {...register("password")} type="password" placeholder="Password" />
  {errors.password && <p>{errors.password.message}</p>}

  <button type="submit">Submit</button>
</form>
```

Advanced Zod Schema



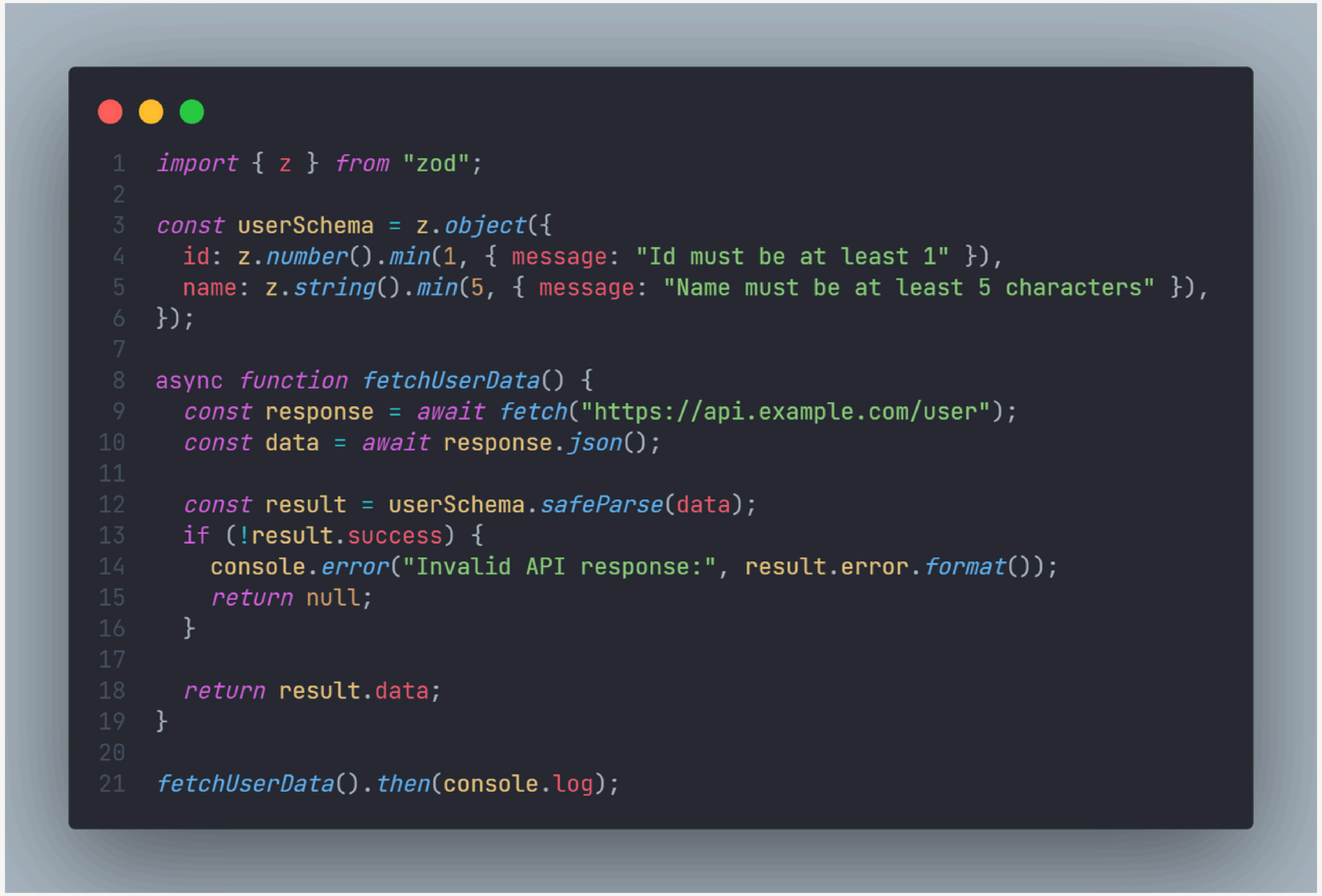
```
const skillSchema = z.object({
  name: z.string().min(1, { message: "Skill name is required" }),
  level: z.enum(["Beginner", "Intermediate", "Advanced", "Expert"], {
    errorMap: () => ({ message: "Please select a valid skill level" }),
  }),
});

const schema = z.object({
  username: z
    .string()
    .min(3, { message: "Username must be at least 3 characters" })
    .max(20, { message: "Username must be at most 20 characters" })
    .refine(
      async (username) => await checkUsernameAvailability(username),
      { message: "This username is already taken" }
    ),
  age: z.number().min(18, { message: "You must be at least 18 years old" }),
  password: z.string().refine((pwd) => /[A-Z]/.test(pwd), {
    message: "Password must contain an uppercase letter",
  }),
  skills: z.array(skillSchema)
    .min(1, { message: "At least one skill is required" })
    .max(5, { message: "Maximum 5 skills allowed" }),
});
```

Demo

Other uses cases for Zod

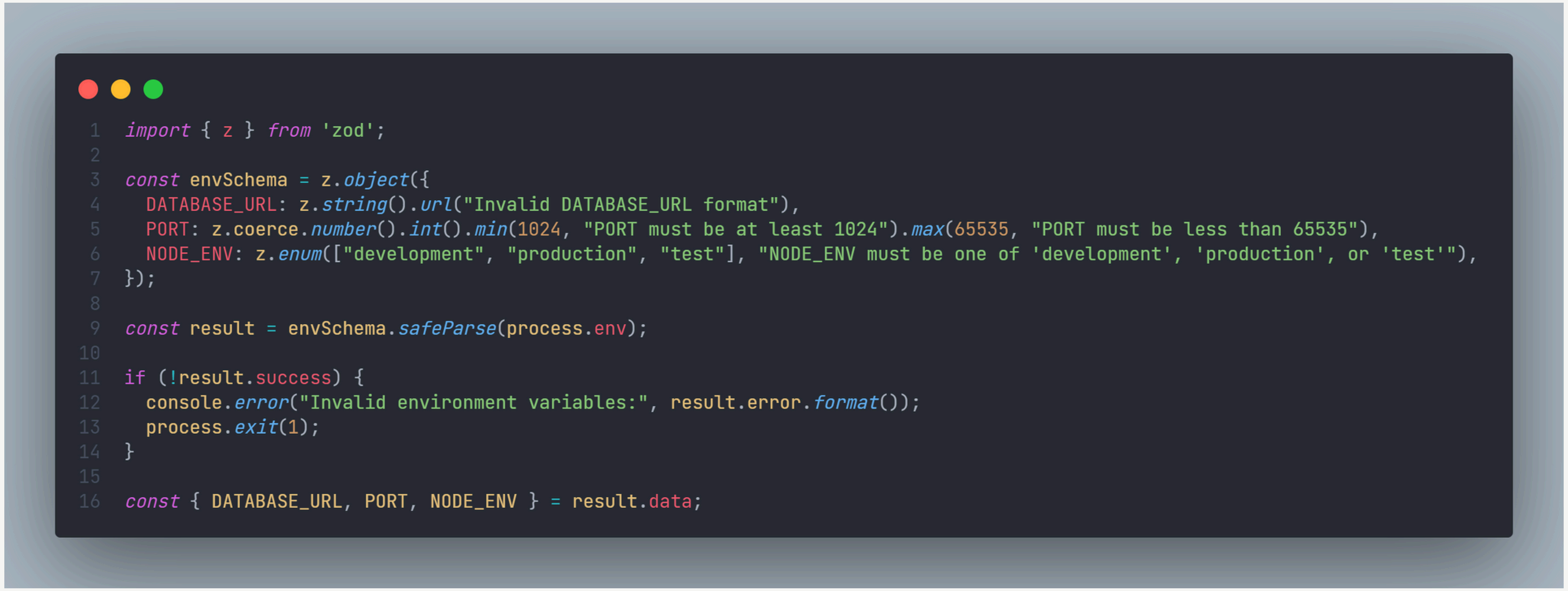
- API Validation – Ensure API requests & responses follow the expected structure.
- Env Variables – Validate process.env at startup to prevent misconfigurations.
- Database Validation – Ensure query results match expected schemas.
- State Management – Enforce structure in Redux, Zustand, or Context API.
- CLI Input Validation – Ensure correct arguments for command-line tools.
- File Uploads & JSON Parsing – Validate uploaded JSON/CSV before processing.

API Validation



```javascript
import { z } from "zod";

const userSchema = z.object({
  id: z.number().min(1, { message: "Id must be at least 1" }),
  name: z.string().min(5, { message: "Name must be at least 5 characters" }),
});

async function fetchUserData() {
  const response = await fetch("https://api.example.com/user");
  const data = await response.json();

  const result = userSchema.safeParse(data);
  if (!result.success) {
    console.error("Invalid API response:", result.error.format());
    return null;
  }

  return result.data;
}

fetchUserData().then(console.log);
```

Env Variable Validation



```javascript
import { z } from 'zod';

const envSchema = z.object({
  DATABASE_URL: z.string().url("Invalid DATABASE_URL format"),
  PORT: z.coerce.number().int().min(1024, "PORT must be at least 1024").max(65535, "PORT must be less than 65535"),
  NODE_ENV: z.enum(["development", "production", "test"], "NODE_ENV must be one of 'development', 'production', or 'test'"),
});

const result = envSchema.safeParse(process.env);

if (!result.success) {
  console.error("Invalid environment variables:", result.error.format());
  process.exit(1);
}

const { DATABASE_URL, PORT, NODE_ENV } = result.data;
```

Raja Krishna

Senior Software Engineer
@LOOP

LinkedIn (/rajakrishna)