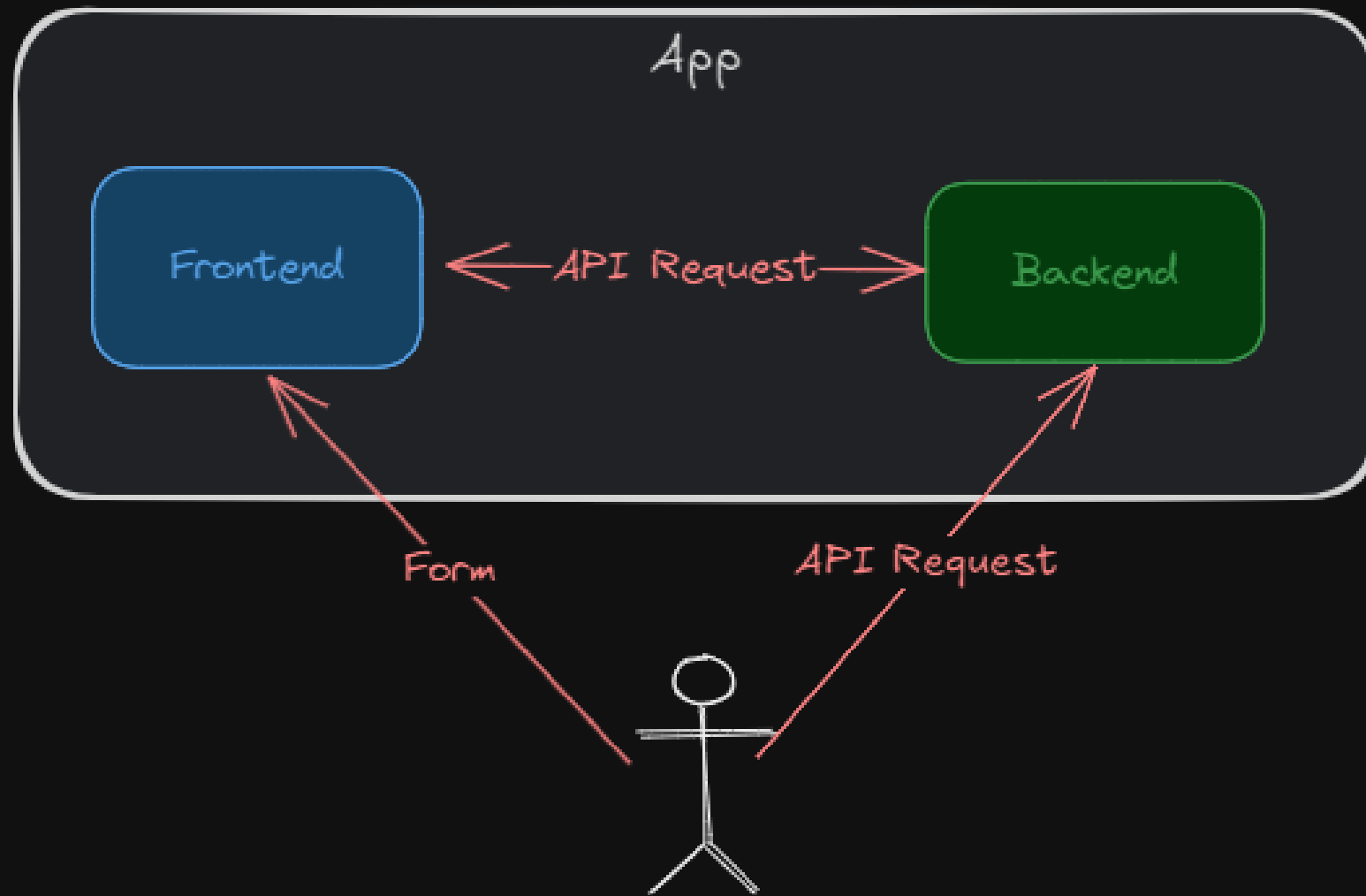


Fullstack Development

Form and validation

| Is your app safe?



Case study

- `git clone -b no-check https://github.com/fullstack-67/fv-zod.git fv-no-check`

Frontend (backend)

- `pnpm i`
- `npm run dev`

Backend pitfalls (1)

- GET /users route
 - The response has password fields.

Backend pitfalls (2)

- `POST` `/users` route

```
{  
  "firstName": "Test",  
  "lastName": "Test",  
  "email": "test@example.com",  
  "dateOfBirth": "2024-01-01",  
  "password": "1234",  
  "confirmPassword": "1234"  
}
```

- Try sending incomplete fields / wrong field names.
 - Internal error

Backend pitfalls (3)

- Trying sending wrong year calendar.
 - Data is now not consistent.

Frontend pitfalls (1)

`.env`

```
VITE_URL_DATA=/api/users_wrong
```

- Notice the incoming data
 - Key differences
 - Calendar year difference
- The wrong data causes blank/inconsistent display.

Frontend pitfalls (2)

- Try using form
 - No input validation

Wait, but I thought I used TypeScript.

- TypeScript catches compile-time errors.
- We are talking about **run-time** errors.

We want

- Advanced/maintainable/scalable data validation
- Useful error message when validation fails
- Data sanitization (backend response)
- Logic reusability in frontend and backend codes
- Seamless combatibility with `TypeScript`
- API documentation

Validation library

- Zod
- Yup
- Typebox

zod basic

Setup

- `git clone -b basic https://github.com/fullstack-67/fv-zod.git fv-basic`
- `pnpm i`
- `npm run dev`

Validation with `zod`

Setup

- `git clone -b check https://github.com/fullstack-67/fv-zod.git fv-check`

Frontend (backend)

- `pnpm i`
- `npm run dev`

Backend - schema

./src/utils/schema.ts

```
export const zUserBase = z.object({  
  // Fields  
});  
// Response  
export const zUsersRes = z.array(zUserBase.omit({ password: true }));
```

Backend - data sanitization

```
// * Endpoint: get users
app.get("/users", (req, res) => {
  res.json(zUsersRes.parse(data)); // 📌📌📌
});
```

Backend - validation middleware

```
export function validateData(schema) {  
  return (req, res, next) => {  
    try {  
      schema.parse(req.body); // 📌📌📌  
      next();  
    } catch (error) {  
      // Error logic  
    }  
  };  
}
```

Backend - data validation

./src/index.ts

```
app.post("/users", validateData(zUsersCreateReq), async (req, res, next) => {  
  // Route logic  
});
```

Frontend - schema

- `./src/utils/schema.ts`
 - Same as backend schema.

Frontend - validation

./src/hooks/useUser.ts

```
function useUsers() {  
  // ...  
  async function fetchUsers() {  
    const res = await axios.get<User[]>(URL_DATA);  
  
    // Validation from Zod  
    const result = usersSchema.safeParse(res.data); // ➡➡➡  
  
    if (!result.success) {  
      // Error logic  
    }  
  }  
}
```

Frontend - form validation

```
const FormVanilla: FC = () => {  
  // ...  
  async function sendData(e) {  
    //...  
    const result = formSchema.safeParse(values); // ➡➡➡  
    if (!result.success) {  
      // Show error message  
    }  
    // ...  
  }  
  return <div id="form"></div>; // Form stuff  
};
```

Try the form yourself

- <http://10.10.12.140:5176> (CMU Network only)
 - Try `Form Vanilla`

Form UX improvement

- "Real-time" validation
- Disable submission if input is not valid.
- Prevent double submission.
- Prevent typing during submission.
- Auto-focus the wrong input.

Real-time validation

- Use `useEffect` to trigger schema validation
- Store errors in `errors` state.
- Keep track of when user touches the form.
 - Prevent premature validation.
 - Store `touch` state

Form disable

- Keep track of `valid` state.
- Keep track of `submission` state.

Spiral out of control

- Too many states
- Too many logics
- Not reusable

Form library

- Help you handle form states and logics in a reusable manner.
 - It is essentially a custom hook.
 - Integrates seamlessly with validation library.
- Popular libraries
 - Formik
 - React Hook Form

Usage

```
import { useForm } from "react-hook-form";  
//  
const rhf = useForm<Form>({  
  // Options  
});  
  
// Observer the states  
console.log({ a_rhf: rhf, b_formState: formState, c_watch: watch() });
```

UI Control

```
<input  
  {...register("firstName")} // 📌📌📌  
  type="text"  
  id="firstName"  
  disabled={isSubmitting}  
>
```

- Check `register("firstName")` in console.
 - This gives `onBlur`, `onChange`, `ref` to HTML element.

Form validation

```
import { zodResolver } from "@hookform/resolvers/zod";
import { formSchema, type Form } from "../utils/schema";

const rhf = useForm<Form>({
  resolver: zodResolver(formSchema), // ➡➡➡
  defaultValues: getInitData(),
  mode: "onTouched",
});
```


Generating documentation

<http://10.10.12.140:5175/api-docs>