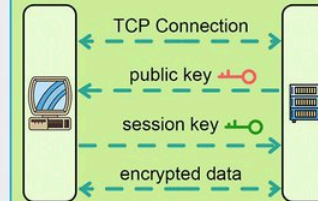


Fullstack Development

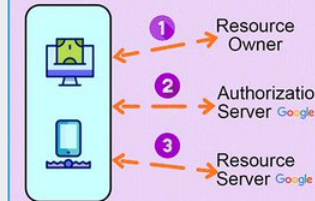
12 Tips for API Security



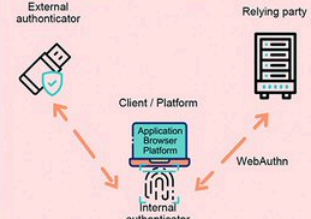
Use HTTPS



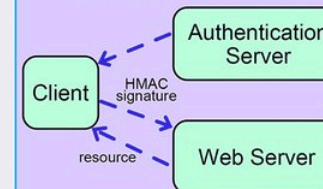
Use OAuth2



Use WebAuthn



Use Leveled API Keys



Authorization

- ✓ Can view
- ✗ Cannot modify

Rate Limiting



Design rate limiting rules based on IP, user, action group etc

API Versioning

- ✓ GET / v1 / users / 123
- ✗ GET / users / 123

Allowlist

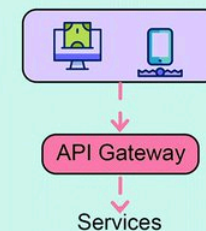


Design allowlist rules based on IP, user etc

Check OWASP API Security Risks



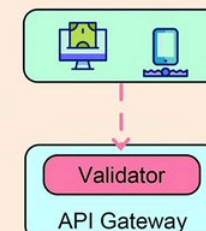
Use API Gateway



Error Handling

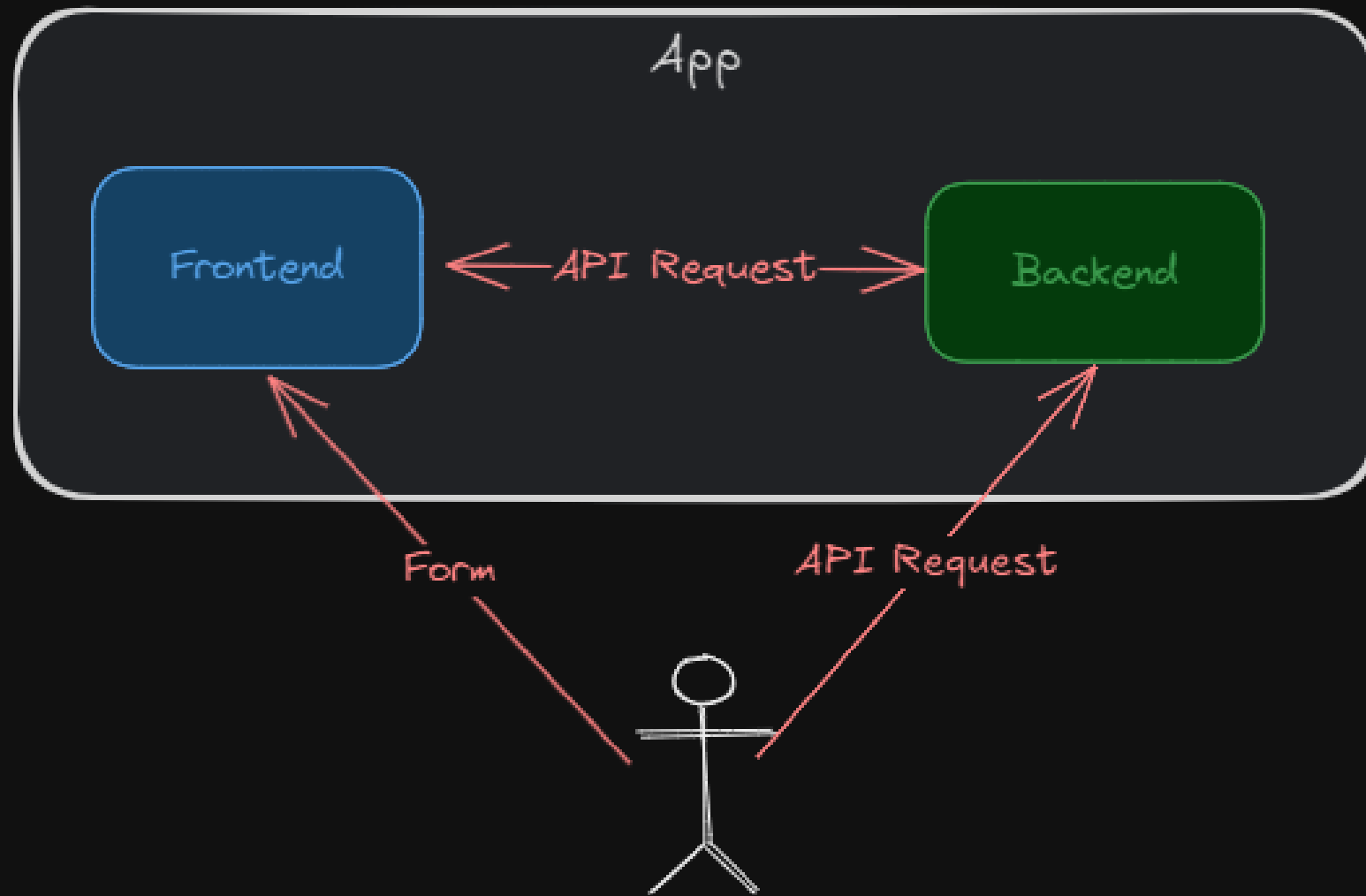
- ✓ descriptive, helpful error messages
- ✓ be empathetic
- ✗ internal stack trace
- ✗ incorrect error codes

Input Validation



Source

Form and validation



Case study

- `git clone https://github.com/fullstack-68/fv-fullstack.git`
- `git checkout -t origin/no-check`

Frontend / Backend

- `pnpm i`
- `pnpm 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)

- Change `.env` back

```
VITE_URL_DATA=/api/users
```

- 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
- Others

zod basic

Setup

- `git clone https://github.com/fullstack-68/fv-zod-basic.git`
- `pnpm i`
- `pnpm run dev`

Validation with `zod`

Setup

- Use `fv-fullstack` folder
- `git checkout -t origin/check`
 - Reinstall packages in frontend and backend.

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

<https://fs-nr.iecmu.com/>

- Try `Form Vanilla` first

Form UX improvement

- "Real-time" validation
- Disable submission button 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

Submission button disabled

- 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
- Newcommer
 - tanstack 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",
});
```

- See `mode` options

Generating documentation

| OpenAPI

What is OpenAPI?

- API description format for REST APIs.
 - Formerly Swagger Specification
- An OpenAPI file allows you to describe your entire API.
 - Available endpoints
 - Operation parameters
 - Authentication methods

What is Swagger?

- Open-source tools built around the OpenAPI Specification.
- Major Swagger tools include
 - [Swagger Editor](#) – browser-based editor
 - [Swagger UI](#) – Library for rendering OpenAPI definitions as interactive documentation

Example

- <https://fs-nr-backend.iecmu.com/api-docs>

Setup

- Use `fv-fullstack` folder.
- `git checkout -t origin/openapi`
 - Reinstall packages in the backend.

Highlight packages

```
{  
  "dependencies": {  
    // ...  
    "@asteasolutions/zod-to-openapi": "^8.1.0",  
    "swagger-ui-express": "^5.0.1",  
    "zod": "^4.1.11"  
  }  
}
```

Extending zod object

./src/openAPI.ts

```
import { extendZodWithOpenApi } from "@asteasolutions/zod-to-openapi";  
// ...  
extendZodWithOpenApi(z);
```

schema.ts

```
export const zUserBase = z.object({  
  id: z  
    .string()  
    // ...  
    .openapi({ example: nanoid() }), // 📌📌📌  
  // ...  
});
```

OpenAPI registry instance

```
import { OpenAPIRegistry } from "@asteasolutions/zod-to-openapi";  
// ...  
export const registry = new OpenAPIRegistry();
```

Documentation from `zod` schema

`./src/index.ts`

```
registry.registerPath({
  method: "get",
  path: "/users",
  responses: {
    200: {
      content: {
        "application/json": {
          schema: zUsersRes, // ➡➡➡
        },
      },
    },
  },
});
app.get("/users", (req, res) => {
  // Route logic
});
```

Route middleware

./src/index.ts

```
app.use(  
  "/api-docs",  
  swaggerUi.serve,  
  swaggerUi.setup(getOpenApiDocumentation())  
);
```

OpenAPI generator

./src/openAPI.ts

```
export function getOpenApiDocumentation() {  
  const generator = new OpenApiGeneratorV3(registry.definitions);  
  return generator.generateDocument({  
    // Options  
    openapi: "3.0.0",  
    info: {  
      version: "1.0.0",  
      title: "My API",  
      description: "This is the API.",  
    },  
    servers: [{ url: "/" }],  
  });  
}
```