PARIS TYPESCRIPT 2025

FULLSTACK FORM VALIDATION WITH ZOD

MORE EXPLICITLY…

- An open **exploration** on how to **streamline** form related developments

MORE EXPLICITLY…

- An open **exploration** on how to **streamline** form related developments
- Take advantage of schema validation (Zod)

ABOUT ME

- Senior freelance front-end developer
- Working for Elephantastic (OSINT) & NuxtLabs
- Vue.js Paris co-organizer

INTRODUCTION

- An example of (bad) form…

FORM REQUIREMENTS

- Data type validation
- Dynamic form

FORM REQUIREMENTS

- Data type validation
- Dynamic form

And of course… Validation on back-end & front-end !

SETUP !

- Nuxt app (meta-framework on the top of Vue)
  - *Server routes*

SETUP

- Nuxt app (meta-framework on the top of Vue)
  - *Server routes*

Note: we could have used any setup (monorepo, packages, …)

SETUP

- Let’s start with a very simple form 🌱 :

```
<form>
  <input v-model="formState.name" />
  <div v-if="formValidationState.errors.name">{{ formValidationState.errors.name }}</div>
</form>
```

BACKEND

```
export default defineEventHandler(async (event) => {
  const body = await readBody(event)
  if (!body.name) {
    throw createError({
      statusCode: 400,
      statusMessage: 'Name is required',
    })
  }
  return 'All good'
})
```

FRONTEND

- A bit more work:
  - validate on the runtime (at each modification of the form)
  - update the UI after with the form validation state (to display errors, etc.)

FRONTEND VALIDATION

```
<template>
  <form>
    <input v-model="formState.name" />
    <div v-if="formValidationState.errors.name">{{ formValidationState.errors.name }}</div>
  </form>
</template>

<script lang="ts">
const formState = reactive({
  name: ''
});

const formValidationState = computed(() => {
  const errors: Record<string, string> = {};
  if (!formState.name) errors.name = "Name is required";
  return {
    success: Object.keys(errors).length === 0,
    errors
  }
})
</script>
```

- Let’s make it more complex 💪

FORM REQUIREMENTS

- Data type validation

DATA TYPE VALIDATION

- Something which could work on backend and frontend

DATA TYPE VALIDATION

- Schema & validation library!
  - Zod
  - Yup
  - Joi
  - io-ts
  - Vine (Node only)

DATA TYPE VALIDATION

- Zod (we will see later why 👀)

SCHEMA & VALIDATION

- Principle:
  - being able to define the data type of a value, an object…
  - Validating the form content according that schema

SCHEMA & VALIDATION

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

const userSchema = z.object({
  name: z.string().min(1),
  email: z.string().min(1).email()
})

const formState = {
  name: 'John Doe',
  email: 'john@doe.com'
};

userSchema.parse(formState)
```

DATA TYPE VALIDATION - BACKEND

```
import { z } from 'zod'

const userSchema = z.object({
  name: z.string().min(1),
  email: z.string().email(),
})

export default defineEventHandler(async (event) => {
  const result = await readValidatedBody(event, body => userSchema.safeParse(body))
  if (!result.success)
    throw result.error.issues

  // (User object is validated and typed!)
  return result.data
})
```

DATA TYPE VALIDATION – FRONTEND

```
<template>
  <form>
    <input v-model="formState.name" />
    <div v-if="formErrors?.fieldErrors?.name">{{ formErrors?.fieldErrors?.name?.[0] }}</div>
    <input v-model="formState.email" />
    <div v-if="formErrors?.fieldErrors?.email">{{ formErrors?.fieldErrors?.email?.[0] }}</div>
    <input type="submit" value="Save" :disabled="!formValidationState.success"/>
  </form>
</template>

<script setup lang="ts">
import { z } from 'zod';

const userSchema = z.object({
  name: z.string().min(1, "Name is required"),
  email: z.string().min(1, "Email is required").email("Invalid email")
})
const formState = ref({
  name: '',
  email: ''
});

const formValidationState = computed(() => userSchema.safeParse(formState.value))
const formErrors = computed(() => formValidationState.value.error?.flatten())
</script>
```

DATA TYPE VALIDATION

- Btw, same schema used on back and front… 🤔

DATA TYPE VALIDATION

- Let's do some refactor 🤓 !

```ts
// schemas > validatorForm.ts

import { z } from "zod";

export function useFormSchema() {
  return z.object({
    name: z.string().min(1, "Name is required"),
    email: z.string().min(1, "Email is required").email("Invalid email")
  })
}
```

DATA TYPE VALIDATION - BACKEND

```
import { z } from ‘zod'
import { useFormSchema } from '~/shemas/validatorForm';


export default defineEventHandler(async (event) => {
  const userSchema = useFormSchema()
  const result = await readValidatedBody(event, body => userSchema.safeParse(body))
  if (!result.success)
    throw result.error.issues

  // (User object is validated and typed!)
  return result.data
})
```

DATA TYPE VALIDATION – FRONTEND

```
<template>
  <form>
    <input v-model="formState.name" />
    <div v-if="formErrors?.fieldErrors?.name">{{ formErrors?.fieldErrors?.name?.[0] }}</div>
    <input v-model="formState.email" />
    <div v-if="formErrors?.fieldErrors?.email">{{ formErrors?.fieldErrors?.email?.[0] }}</div>
    <input type="submit" value="Save" :disabled="!formValidationState.success"/>
  </form>
</template>

<script setup lang="ts">
import { useFormSchema } from '~/shemas/validatorForm';

const userSchema = useFormSchema()

const formState = ref({});

const formValidationState = computed(() => userSchema.safeParse(formState.value))
const formErrors = computed(() => formValidationState.value.error?.flatten())
</script>
```

DATA TYPE VALIDATION - FRONTEND

- But the front-end still coupled to the schema shape 🧐 :

```
<template>
  <form>
    <input v-model="formState.name" />
    <div v-if="formErrors?.fieldErrors?.name">{{ formErrors?.fieldErrors?.name?.[0] }}</div>
    <input v-model="formState.email" />
    <div v-if="formErrors?.fieldErrors?.email">{{ formErrors?.fieldErrors?.email?.[0] }}</div>
    <input type="submit" value="Save" :disabled="!formValidationState.success"/>
  </form>
</template>
```

DATA TYPE VALIDATION - FRONTEND

- Compute the frontend from the schema 😀 ?

DATA TYPE VALIDATION - FRONTEND

- Need to add more information in the schema (type of component, labels…)

DATA TYPE VALIDATION - FRONTEND

- For example:

```
export const shape: FormShape[] = [
  {
    label: 'Name?',
    _key: "name",
    value: '',
    rules: z.string().min(1),
    type: 'input'
  },
  {
    label: 'Email?',
    _key: "email",
    value: '',
    rules: z.string().min(1).email(),
    type: 'input'
  },
];
```

DATA TYPE VALIDATION - FRONTEND

- For example:

```
export const shape: FormShape[] = [
  {
    label: 'Name?',
    _key: "name",
    value: '',
    rules: z.string().min(1),
    type: 'input'
  },
  {
    label: 'Email?',
    _key: "email",
    value: '',
    rules: z.string().min(1).email(),
    type: 'input'
  },
];
```

DATA TYPE VALIDATION - FRONTEND

- Then how can we validate the forms?

DATA TYPE VALIDATION - FRONTEND

- For example:

```
export function useFormSchema(formShape: FormShape) {
  const shape: FormShape[] = [
    {
      label: 'Name?',
      _key: "name",
      value: '',
      rules: z.string().min(1),
      type: 'input'
    },
    // ...
  ];

  // Rebuild the schema from the form shape
  const schemaObject: Record<string, ZodTypeAny> = {};
  formShape.forEach((field) => {
    schemaObject[field._key] = field.rules;
  });
  const schema = z.object(schemaObject);

  return { shape, schema };
}
```

DATA TYPE VALIDATION - FRONTEND

- For example:

```
export function useFormSchema() {
  const shape: FormShape[] = [
    {
      label: 'Name?',
      _key: "name",
      value: '',
      rules: z.string().min(1),
      type: 'input'
    },
    // ...
  ];

  // Rebuild the schema from the form shape
  const schemaObject: Record<string, ZodTypeAny> = {};
  formShape.forEach((field) => {
    schemaObject[field._key] = field.rules;
  });
  const schema = z.object(schemaObject);

  return { shape, schema };
}
```

DATA TYPE VALIDATION – FRONTEND

- For example:

```
export function useFormSchema(formShape: FormShape) {
  const shape: FormShape[] = [
    {
      label: 'Name?',
      _key: "name",
      value: '',
      rules: z.string().min(1),
      type: 'input'
    },
    // ...
  ];

  // Rebuild the schema from the form shape
  const schemaObject: Record<string, ZodTypeAny> = {};
  formShape.forEach((field) => {
    schemaObject[field._key] = field.rules;
  });
  const schema = z.object(schemaObject);

  return { shape, schema };
}
```

DATA TYPE VALIDATION - FRONTEND

- Then generating the form becomes a breeze 🏝 :

```
<template>
  <form>
    <template v-for="field in formShape">
      <div v-if="field.type === 'input'">
        <input v-model="field.value" />
        <div>{{ formErrors?.[field.key] }}</div>
      </div>
      <div v-else-if="field.type === 'radio'">
        <!-- radio button input -->
      </div>
      <!-- other type of inputs -->
    </template>
  </form>
</template>
```

DATA TYPE VALIDATION - FRONTEND

- Or, more elegantly 💅 :

```
<!-- SchemaForm.vue - a generic component accepting slots -->
<template>
  <form>
    <template v-for="field in formSchema">
      <slot :name="field.type" :error="formErrors?.[field._key]" v-bind="field" />
    </template>
  </form>
</template>
```

DATA TYPE VALIDATION - FRONTEND

- Or, more elegantly 💅 :

```
<!-- `MyForm.vue`, which uses `SchemaForm` component with custom inputs -->
<template>
  <SchemaForm :data :form-schema :form-state @update-form="updateForm">
    <template v-slot:input="{ _key, label, value, error }">
      <Input :error :label :value @update:model-value="updateForm(_key, $event)" />
    </template>
    <template v-slot:radio="{ _key, legend, label, options, value, error }">
      <Radio :error :legend :value :options @update:model-value="updateForm(_key, $event)" />
    </template>
    <!-- Other inputs... -->
  </SchemaForm>
</template>
```

FORM REQUIREMENTS

- Data type validation ✅

FORM REQUIREMENTS

- Data type validation ✅
  - Form logic mutualised between back & front
  - Form logic decoupled from the rest of the code

FORM REQUIREMENTS

- Data type validation ✅
  - Form logic mutualised between back & front
  - Form logic decoupled from the rest of the code
- Dynamic form?

DYNAMIC FORM

DYNAMIC FORM

- Form shape depends on form inputs 🥸

DYNAMIC FORM

- Form shape depends on form inputs 🥸

=> form shape and schema will be computed from the form inputs.

DYNAMIC FORM

```
export function schemaComputer(formData: FormData, formShape: FormShape) {
  let shape: FormShape[] = [
    // ...
  ];

  // Form shape based on the `disease` input
  switch (formData.disease) {
    case ('covid'): shape = shape.concat(CovidShape)
    case ('gastro'): shape = shape.concat(GastroShape)
    // ...
  }

  const schemaObject: Record<string, ZodTypeAny> = {};
  formShape.forEach((field) => { schemaObject[field._key] = field.rules; });
  const schema = z.object(schemaObject);

  return { shape, schema };
}
```

DYNAMIC FORM

```
export function schemaComputer(formData: FormData, formShape: FormShape) {
  let shape: FormShape[] = [
    // ...
  ];

  // Form shape based on the `disease` input
  switch (formData.disease) {
    case ('covid'): shape = shape.concat(CovidShape)
    case ('gastro'): shape = shape.concat(GastroShape)
    // ...
  }

  const schemaObject: Record<string, ZodTypeAny> = {};
  formShape.forEach((field) => { schemaObject[field._key] = field.rules; });
  const schema = z.object(schemaObject);

  return { shape, schema };
}
```

DYNAMIC FORM – BACKEND

```
import { schemaComputer } from '~/schemaComputer';

export default defineEventHandler(async (event) => {
  const body = await readBody(event);
  const schema = schemaComputer(body);
  const result = z.safeParse(schema);

  if (!result.success)
    throw result.error.issues;

  return result.data;
});
```

DYNAMIC FORM – BACKEND

```
import { schemaComputer } from '~/schemaComputer';

export default defineEventHandler(async (event) => {
  const body = await readBody(event);
  const schema = schemaComputer(body);
  const result = z.safeParse(schema);

  if (!result.success)
    throw result.error.issues;

  return result.data;
});
```

DYNAMIC FORM - FRONTEND

- Again, a bit more complicated

$$\begin{tikzcd}
 & & \text{Form shape} \\
\text{Form input change} \arrow[r] & \text{Schéma computation} \arrow[ur] \arrow[dr] & \\
 & & \text{Form validation}
\end{tikzcd}$$

FORM REQUIREMENTS

- Data type validation ✅
  - Form logic mutualised between back & front
  - Form logic decoupled from the rest of the code
- Dynamic form ✅

CONCLUSION

- Proof a concept, but…

CONCLUSION

- Proof a concept, but…

- Beyond validation, schema can help architecting our forms with a great separation of concerns.

WHICH SCHEMA LIBRARY CHOOSE?

- Yup : less build-in schemas (promises, functions…)
- Joi : no static type inference
- io-ts : very functional
- Vine : Node only

ZOD TYPE INFERENCE

```
const userSchema = z.object({
  name: z.string(),
  email: z.string().email()
})

type User = z.infer<typeof userSchema>

// type User = {
//   name: string
//   email: string
// }
```

ZOD TYPE INFERENCE

```
const userSchema = z.object({
  name: z.string(),
  email: z.string().email()
})

type User = z.infer<typeof userSchema>

// type User = {
//   name: string
//   email: string
// }
```

RESOURCES

Demo

Slides

THANK YOU