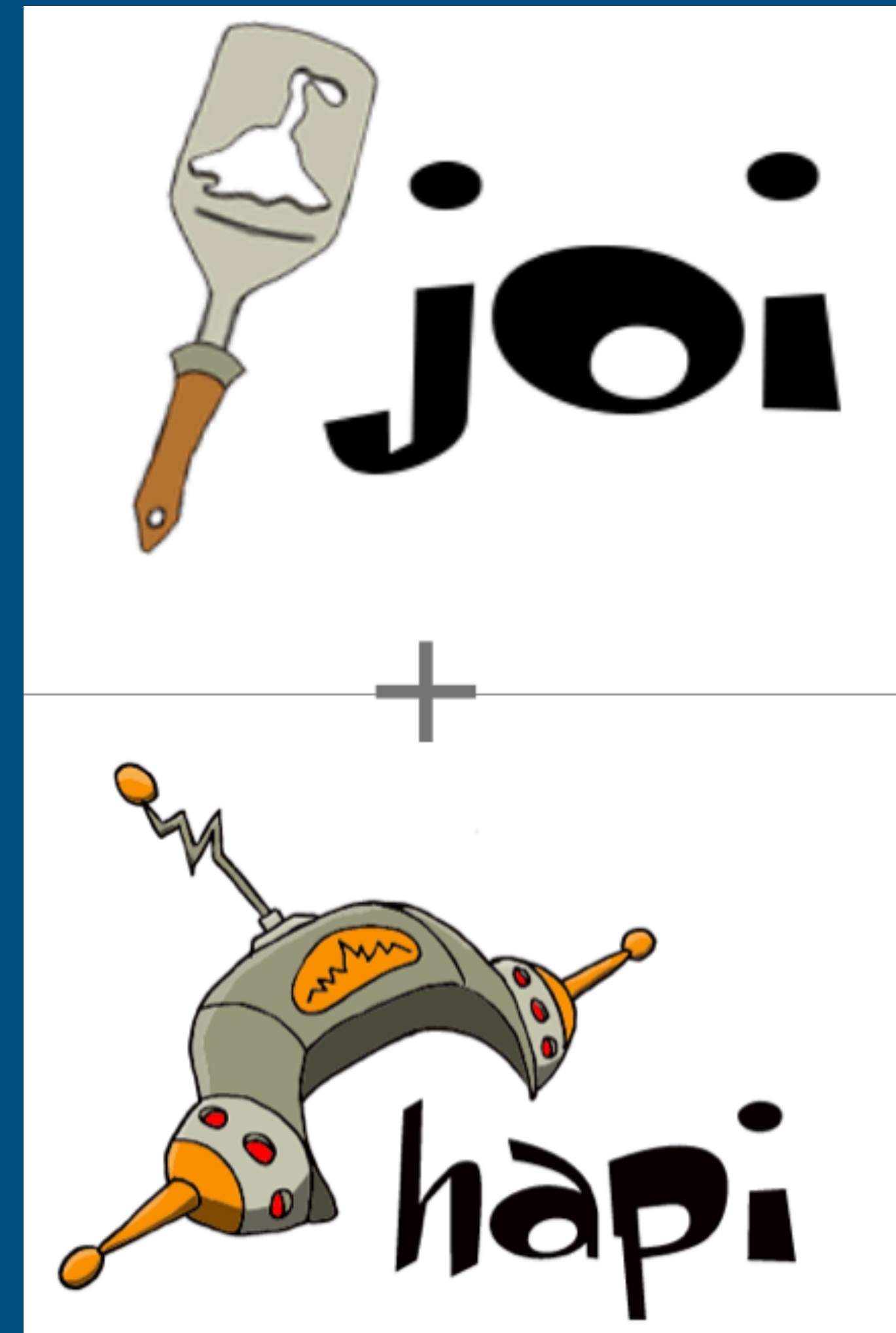


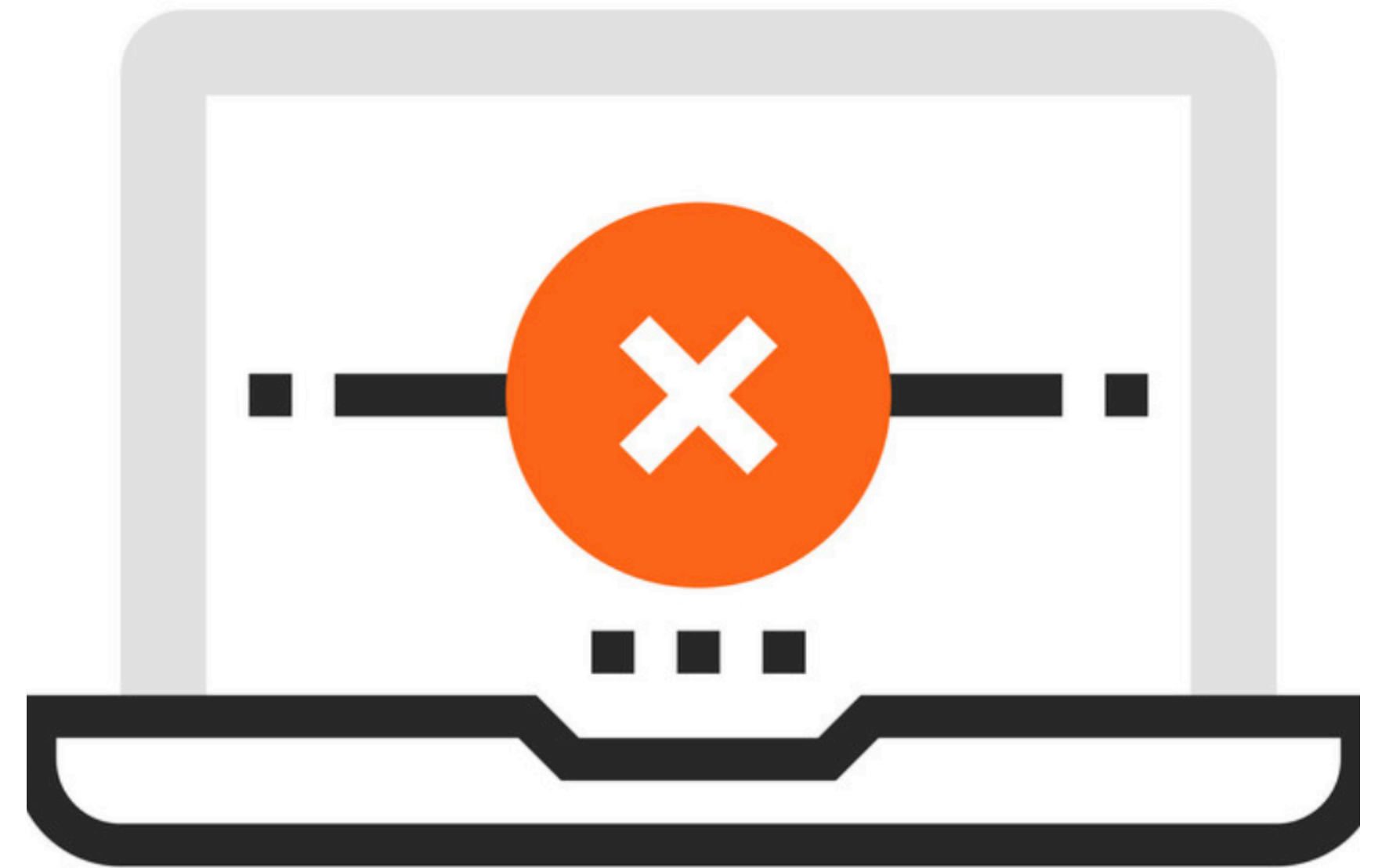
# Joi + Hapi



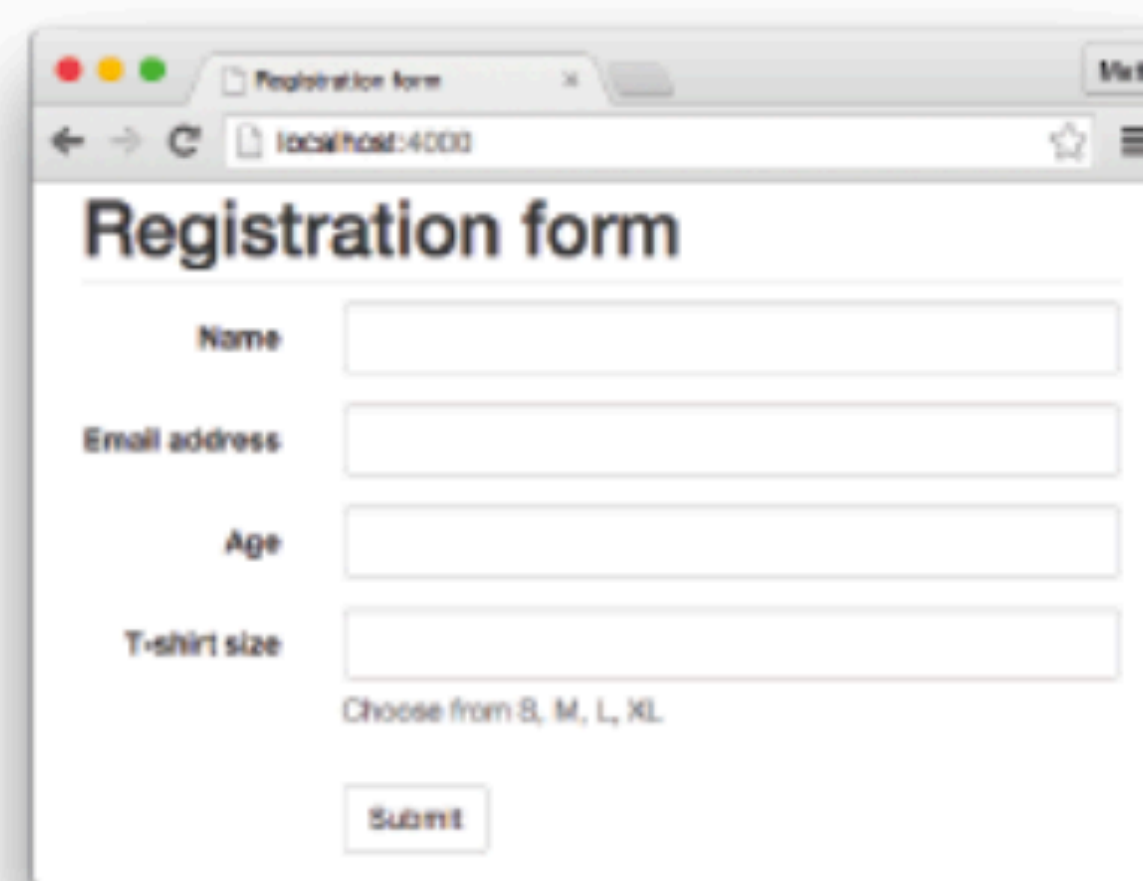
Full Stack Web Development

# Agenda

- Error Handling UI Strategies
- Joi Installation
- Joi in Donation



## The basic form

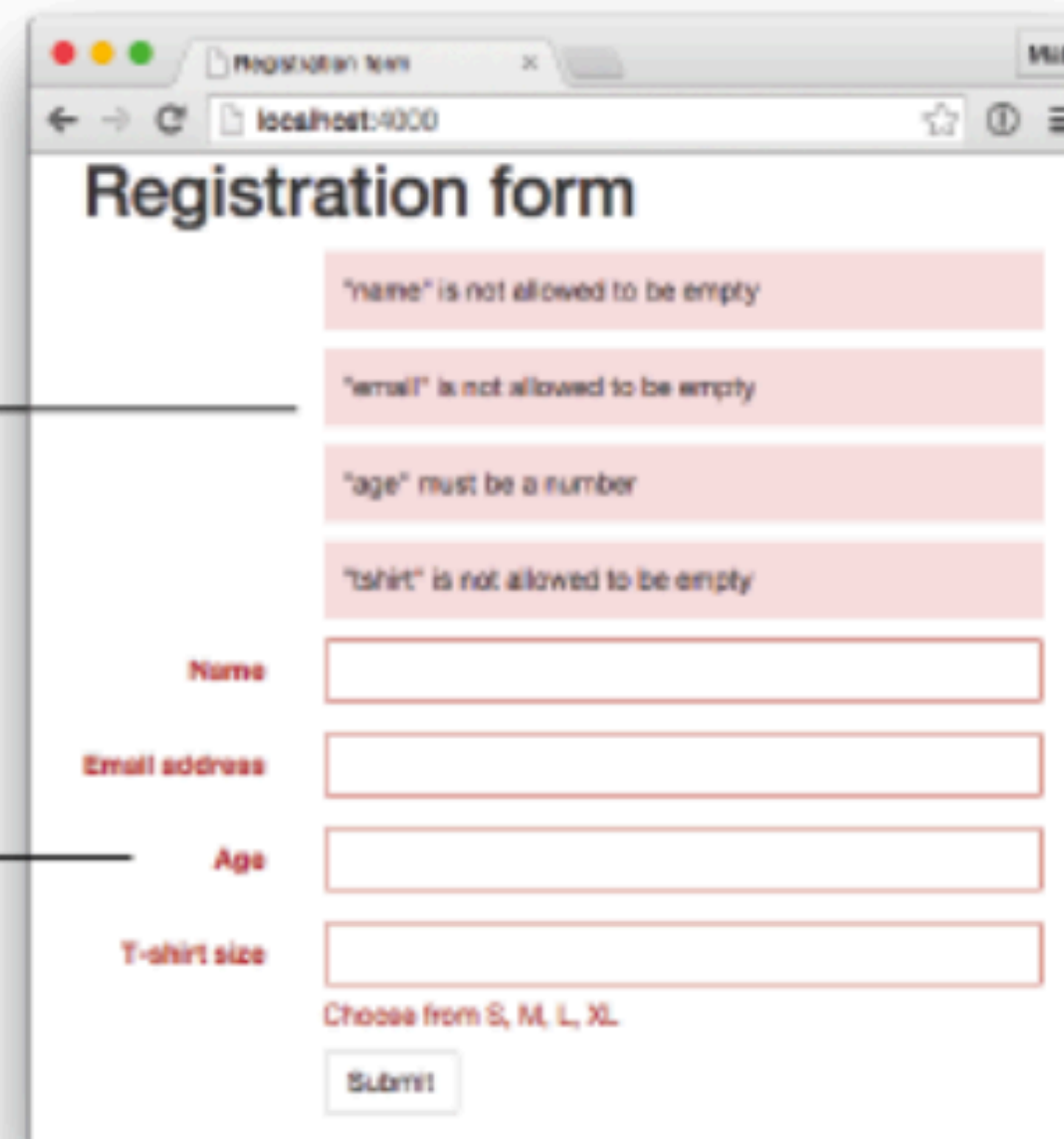


A screenshot of a web browser window showing a registration form. The browser's address bar displays 'localhost:4000'. The form is titled 'Registration form' and contains four input fields: 'Name', 'Email address', 'Age', and 'T-shirt size'. Below the 'T-shirt size' field, there is a text label 'Choose from S, M, L, XL'. A 'Submit' button is located at the bottom of the form.

## The form with errors

All errors are listed at the top

Fields with errors are highlighted in red



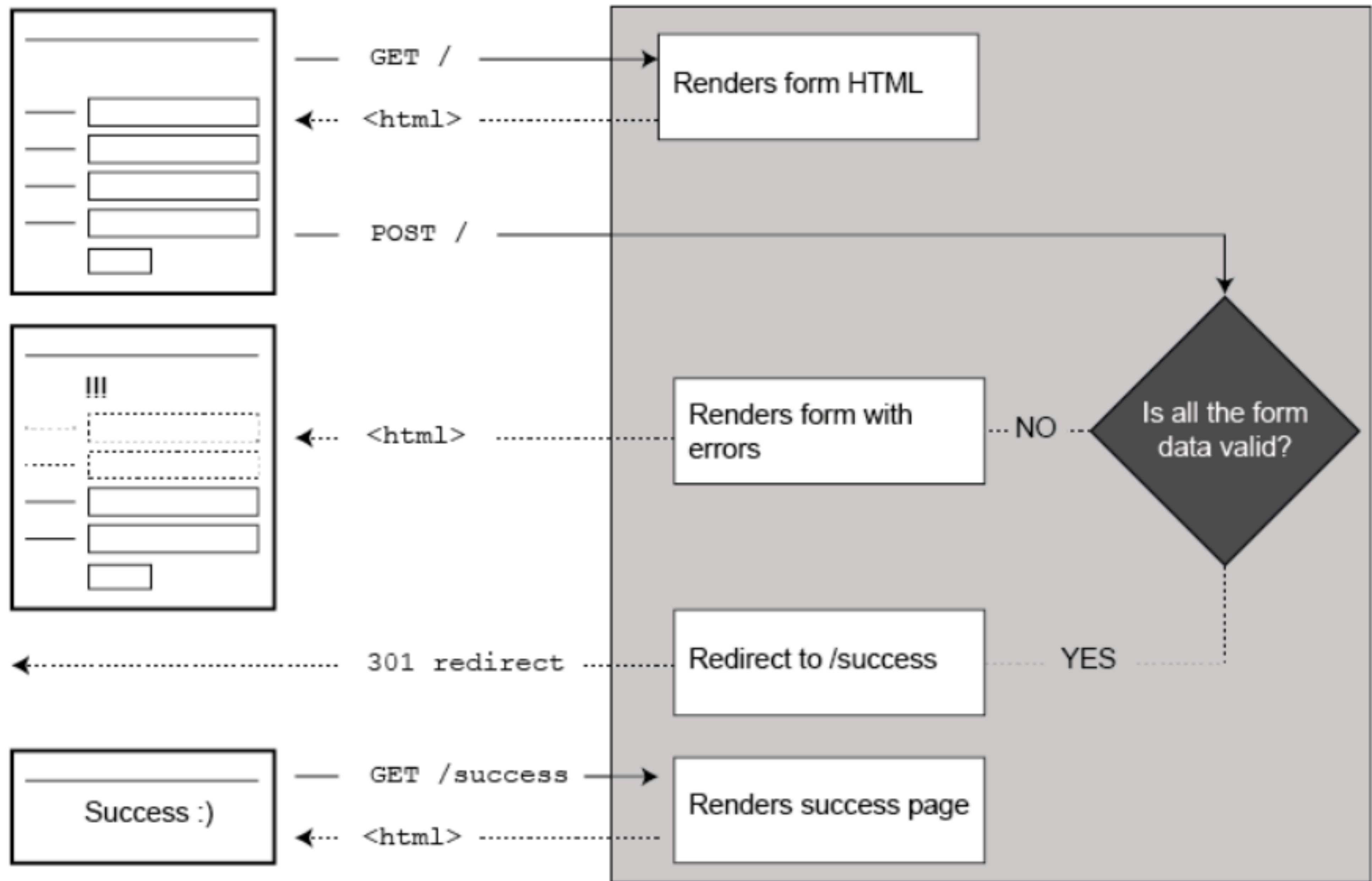
A screenshot of the same registration form, but with validation errors. At the top of the form, four error messages are listed in red boxes: '"name" is not allowed to be empty', '"email" is not allowed to be empty', '"age" must be a number', and '"shirt" is not allowed to be empty'. The input fields for 'Name', 'Email address', 'Age', and 'T-shirt size' are all highlighted with red borders. The 'Submit' button remains at the bottom.

## The success page



A screenshot of a web browser window showing the success page. The browser's address bar displays 'localhost:4000/success'. The page is titled 'Registration form' and contains the text 'Thanks, we'll be in touch soon'.

- Potential Error Reporting Strategy



# Install Joi

```
npm install joi
```

**server.js**

```
import Joi from "joi";

...
await server.register(Vision);
await server.register(Cookie);
server.validator(Joi);
...
```

- Install and configure Joi validation engine.

The screenshot shows the GitHub repository page for 'sideway/joi'. At the top, it indicates the repository is 'Public' and shows statistics: 176 watches, 1.4k forks, and 18.1k stars. Navigation tabs include Code, Issues (95), Pull requests (1), Actions, Projects, Security, and Insights. Below these, there are buttons for 'master' (4 branches), '241 tags', 'Go to file', 'Add file', and a green 'Code' button. The main content area displays a commit history table with columns for file changes, commit messages, and timestamps. The right sidebar contains an 'About' section describing Joi as 'The most powerful data validation library for JS', with tags for 'javascript', 'schema', 'validation', and 'hapi'. It also lists links for 'Readme', 'View license', and 'Code of conduct', along with statistics: 18.1k stars, 176 watching, and 1.4k forks. Below this is a 'Releases' section showing '241 tags' and a 'Packages' section stating 'No packages published'.

File	Commit Message	Time
.github/workflows	chore: replace travis with github actions	last month
benchmarks	Delete package-lock.json	2 years ago
browser	Remove process and null-loader	6 months ago
lib	Support wrapping of strings inside arrays. ...	last month
test	Support wrapping of strings inside arrays. ...	last month
.eslintignore	Use helper in tests (1st part)	2 years ago
.gitignore	Setup for browser build publishing	2 years ago
API.md	Support wrapping of strings inside arrays. ...	last month
LICENSE.md	Update deps	15 months ago
README.md	Update README.md	15 months ago
package.json	17.5.0	last month




# Joi in Playtime

## db/joi-schema.js

```
import Joi from "joi";

export const UserSpec = {
  firstName: Joi.string().required(),
  lastName: Joi.string().required(),
  email: Joi.string().email().required(),
  password: Joi.string().required(),
};
```

 Playtime

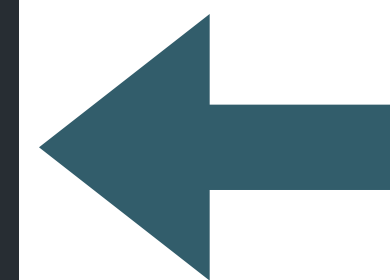
[Log in](#) [Sign up](#)

### Sign up

**Name**

**Email**

**Password**



Define a schema to describe user signup information

## Sign up

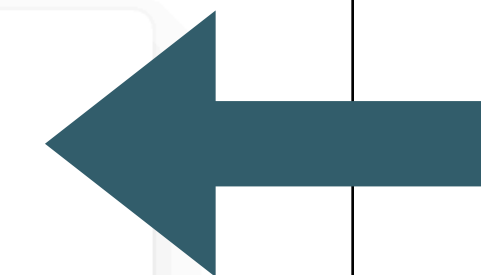
**Name**

**Email**

**Password**

There was a problem...

- "email" must be a valid email



Additional  
Error panel



## Sign up

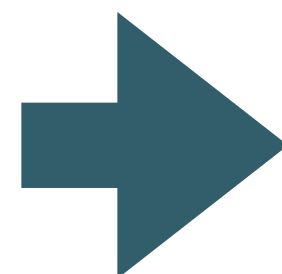
Name

Email

Password

There was a problem...

- "email" must be a valid email



### error.hbs

```
{{#if errors}}
  <div class="box content">
    <p> There was a problem... </p>
    <ul>
      {{#each errors}}
        <li>{{message}}</li>
      {{/each}}
    </ul>
  </div>
{{/if}}
```



## layout.hbs

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>{{title}}</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.3/css/bulma.min.css">
    <script src="https://use.fontawesome.com/releases/v5.15.4/js/all.js"></script>
  </head>
  <body>
    <div class="container">
      {{{content}}}
      {{> error }}
    </div>
  </body>
</html>
```

## error.hbs

```
{{#if errors}}
  <div class="box content">
    <p> There was a problem... </p>
    <ul>
      {{#each errors}}
        <li>{{message}}</li>
      {{/each}}
    </ul>
  </div>
{{/if}}
```

- Define a schema to describe user signup information

- **payload:** This defines a schema which defines rules that our fields must adhere to.

```
validate: {  
  payload: UserSpec,  
  options: { abortEarly: false },  
  failAction: function(request, h, error) {  
    return h.view("signup-view", { title: "Sign up error", errors: error.details }).takeover().code(400);  
  },  
},
```

There was a problem...

- "firstName" is not allowed to be empty
- "lastName" is not allowed to be empty
- "email" is not allowed to be empty
- "password" is not allowed to be empty

- **failAction:** This is the handler to invoke if one or more of the fields fails the validation

- Equip the handlers with validate objects
- These objects will drive the Joi validation engine, generating errors if there are any violations

There was a problem...

- "firstName" is not allowed to be empty
- "lastName" is not allowed to be empty
- "email" is not allowed to be empty
- "password" is not allowed to be empty

## accounts-controller.js

```
import { UserSpec, } from "../models/joi-schemas.js";

signup: {
  auth: false,
  validate: {
    payload: UserSpec,
    failAction: function (request, h, error) {
      return h.view("signup-view", { title: "Sign up error" }).takeover().code(400);
    },
  },
  handler: async function (request, h) {
    const user = request.payload;
    await db.userStore.addUser(user);
    return h.redirect("/");
  },
},
```

## validate Property in Hapi handler

```
validate: {
  payload: {
    firstName: Joi.string().required(),
    lastName: Joi.string().required(),
    email: Joi.string()
      .email()
      .required(),
    password: Joi.string().required()
  },
  failAction: function(request, h, error) {
    return h
      .view('signup', {
        title: 'Sign up error',
        errors: error.details
      })
      .takeover()
      .code(400);
  },
},
```

- payload: This defines a schema which defines rules that our fields must adhere to.
- failAction: This is the handler to invoke if one or more of the fields fails the validation



enable validation for this handler

the joi schema - tied to payload from request

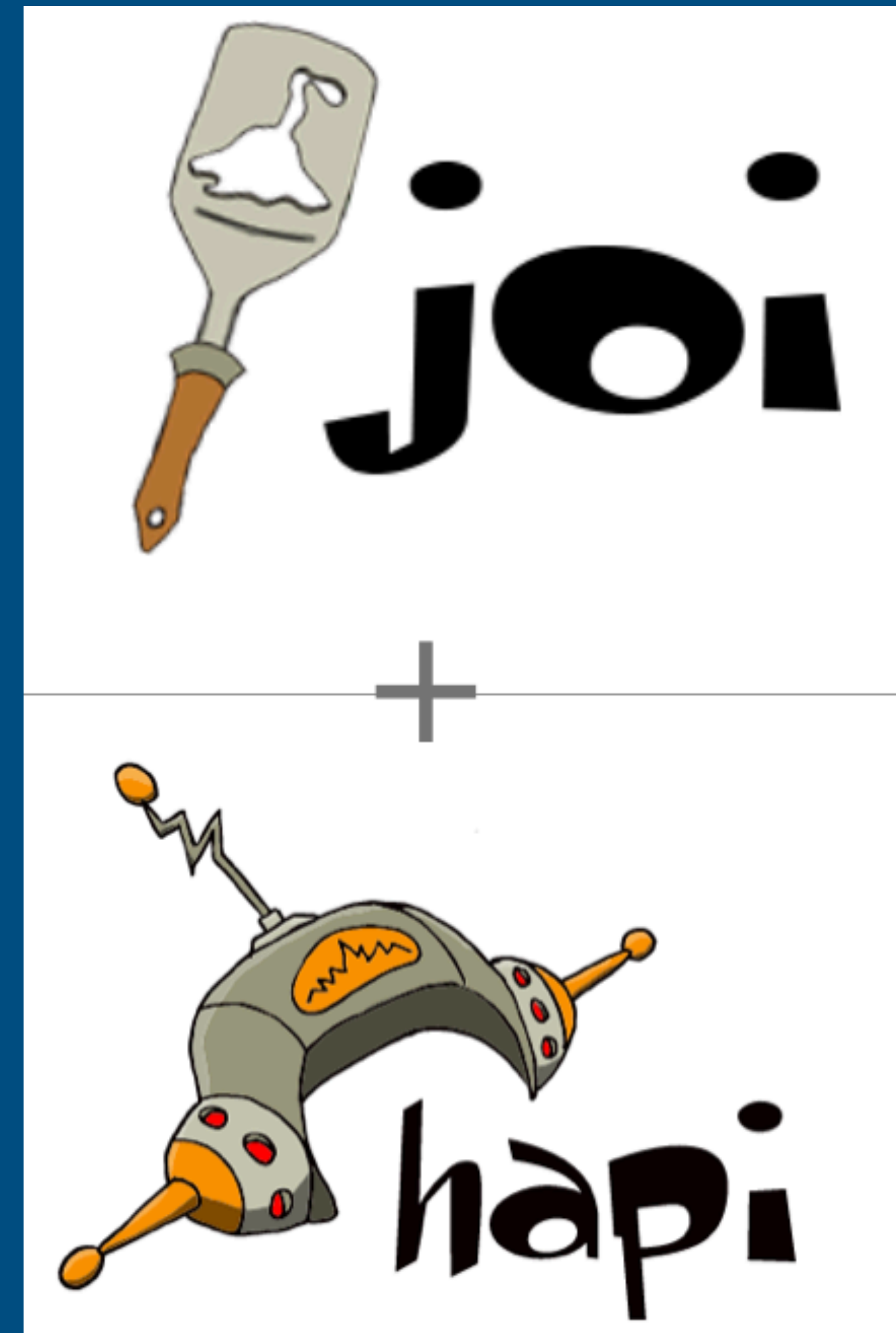
handler if validation fails

the main handler for register, only validated payload gets here

signup handler

```
signup: {  
  auth: false,  
  validate: {  
    payload: UserSpec,  
    options: { abortEarly: false },  
    failAction: function (request, h, error) {  
      return h.view("signup-view", {  
        title: "Sign up error",  
        errors: error.details  
      }).takeover().code(400);  
    },  
  },  
  handler: async function (request, h) {  
    const user = request.payload;  
    await db.userStore.addUser(user);  
    return h.redirect("/");  
  },  
},
```

# Joi + Hapi



Full Stack Web Development