Hochschule Konstanz
Fakultät Informatik

WEB TECHNOLOGIES

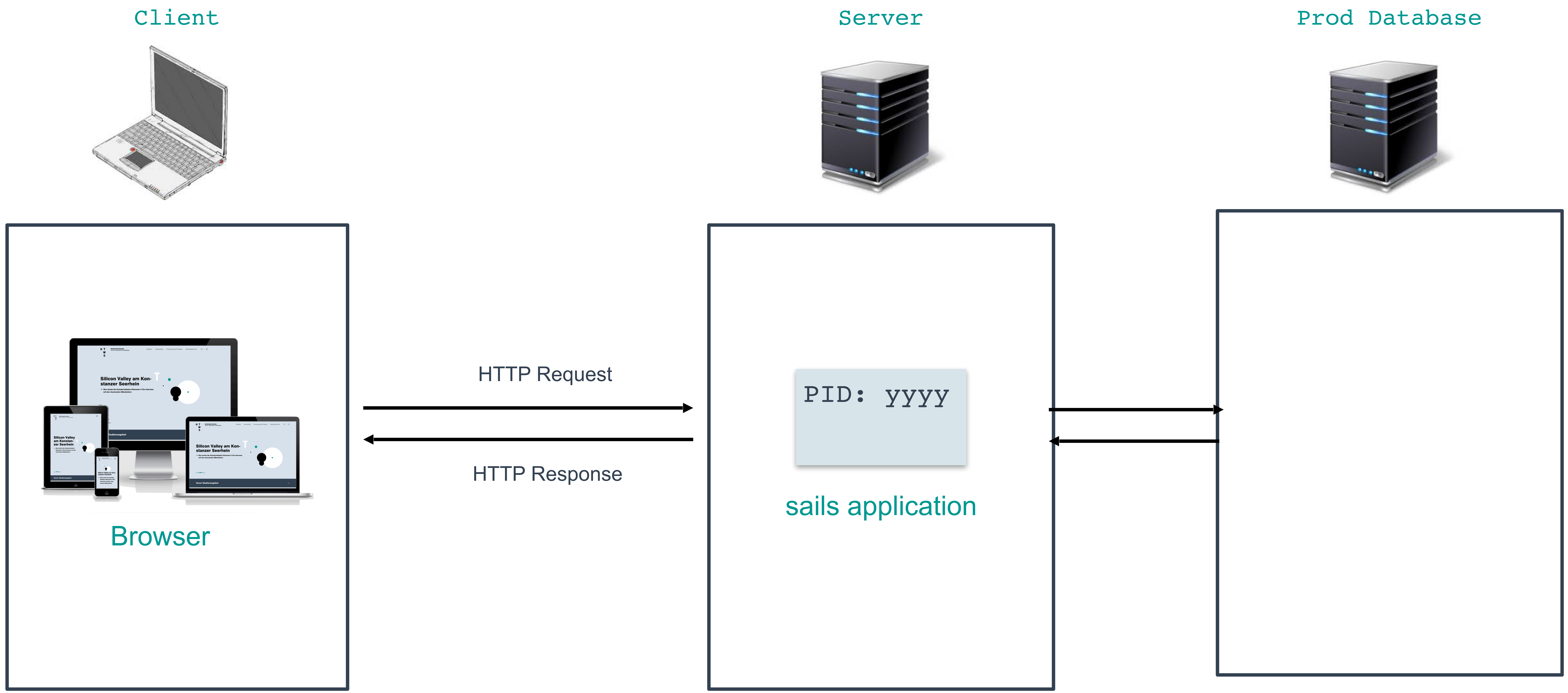
Prof. Dr. Markus Eiglsperger
Sommersemester 2023

Hochschule Konstanz
Fakultät Informatik

INPUT VALIDATION AND FILE UPLOAD

File Upload

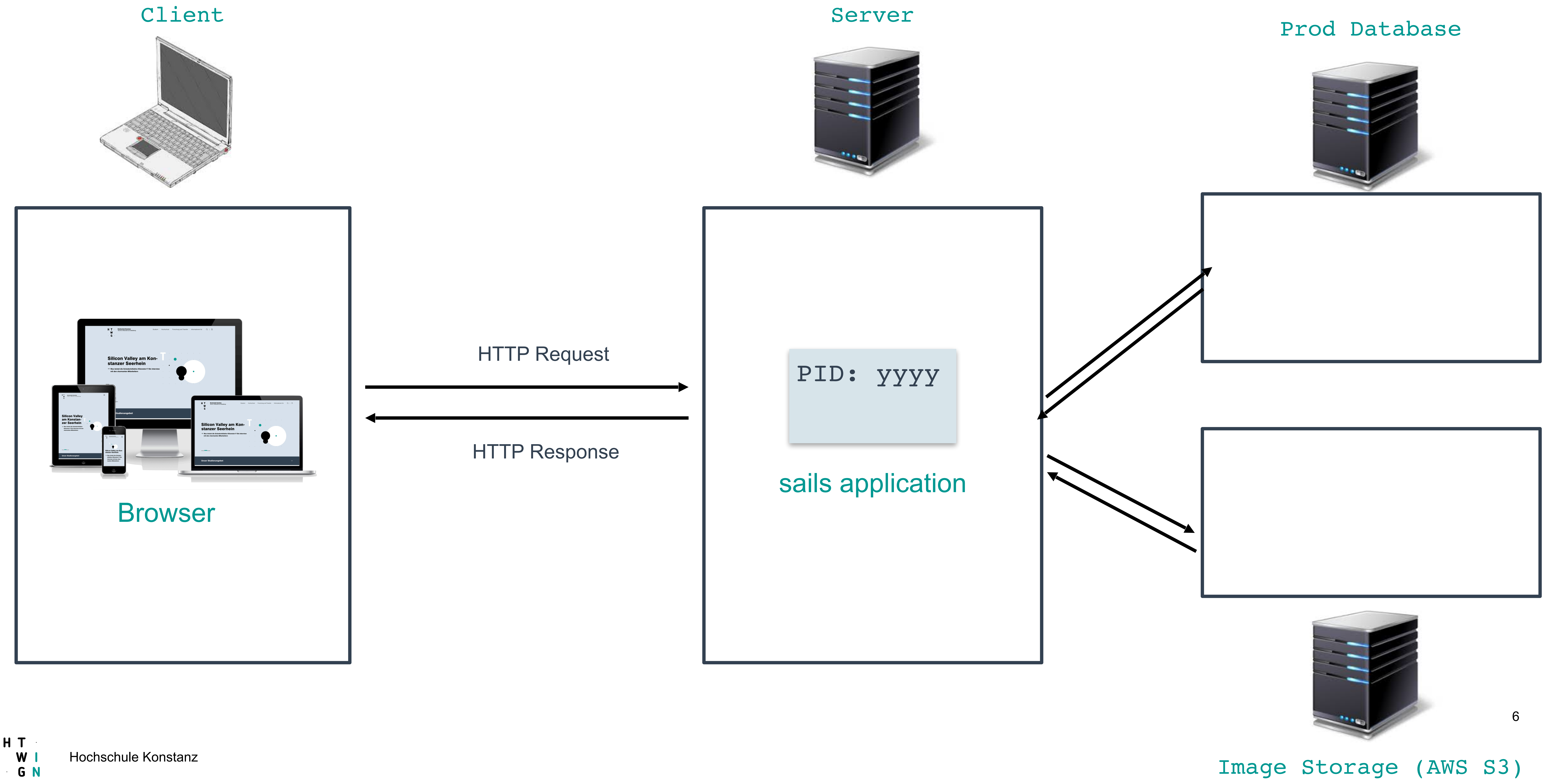
Production Setup with Database



Storing Binary Data

- Storing binary data like images in a database is not efficient
- File System is an alternative
  - Easy solution, especially in local development
  - If a WebApp is deployed to multiple servers, file can not be stored locally, a file server is needed
  - File System access may not be possible on deployment platform
  - File System access may not be stable on deployment platform
- Object Storage is alternative which offers cost effective solution
  - Object Storages use buckets to store data in files.
  - We use AWS S3, plenty of alternatives exists
  - We use an API to access S3
  - Install client: `npm install skipper-s3 --save`

Production Setup with Database and Image Storage



File Upload: Client side (Browser)

- Large data like images, videos, documents must be uploaded as multipart document:

```
<form action="…" method="post" enctype="multipart/form-data">
```

- To define a file upload field, use the input type file

```
<input type="file" class="form-control" name="image">
```

https://sailsjs.com/documentation/concepts/file-uploads

File Upload: Server side (Controller)

The Controller must implement two steps for file upload:

- **Object Storage: The uploaded file will be uploaded in the object storage with a unique name.**
- **Database Update: The name of the file will be stored in the database along with the entity it belongs to**
  - **In our example we add a column `image` to the `Meal` Entity, where we store the image name.**

File Upload: Server side (Controller)

- With the `req.file` method, the content of the file to upload can be accessed.
- With the `upload` method the file can be stored in the object storage
  - In `params` we can specify where to credentials and connection details
  - The `callback` parameter must be a function, which is called once the upload is finished. The logic to update the database is implemented in this function

```
let params = {
     adapter: require('skipper-s3'),
      key: 'AKIAXCTWFCIPLLVPH4NK',
      secret: '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',
      bucket: 'wetebucket',
      region: 'us-west-2'
    };

let callback = async function (err, uploadedFiles) {
 …
};

req.file(‘image’).upload(params, callback)
```

Example, which stores filename in DB

```
let callback = async function (err, uploadedFiles) {
      if (err) {
        return res.serverError(err);
      } else {
        sails.log("Uploaded!")
      }
      let fname = require('path').basename(uploadedFiles[0].fd);
      await Meal.updateOne({ id: req.params.id }).set({ image:fname });
    };
```

Check if upload was successful

Extract filename

Update DB row for meal with filename

Display image in website

```
<% if(meal.image) { %>
                <div class="row">
                    <div class="col-2">
                        Bild
                    </div>
                    <div class="col-10">
                        <img src="https://wetebucket.s3.us-west-2.amazonaws.com/<%= meal.image %>">
                    </div>
                </div>
<% } %>
```

Validation

Problem

- How to handle invalid user input?
  - Not all input values make sense
  - Application should not allow that they enter the system
    - Usability
    - Robustness
    - Security

Learning Objectives

- Students know where to apply validation in web applications and how to implement it in Sails.
  - Students understand why validation is important in the frontend and in the backend.
  - Students can implement backend validation in Sails.

Validation

- Client-Side
  - Form
  - JavaScript
- Server-Side
  - Part of Application code

Are both necessary?

- Never trust the client
- Input validation on on each layer

Server Side

Validation of Model fields

- Validation rules of models are executed whenever `create()` or `update()` is called
- If Validation fails, error is thrown

```
module.exports = {

  attributes: {
    emailAddress: {
      type: 'string',
      required: true,
      unique: true,
      isEmail: true,
      maxLength: 200,
      example: 'mary.sue@example.com'
    },

    emailStatus: {
      type: 'string',
      isIn: ['unconfirmed', 'change-requested', 'confirmed'],
      defaultsTo: 'confirmed',
      description: 'The confirmation status of the user\'s email address.',
      extendedDescription:
`…`
    },
```

Validation Rules

Name of Rule	What It Checks For	Notes On Usage
custom	A value such that when it is provided as the first argument to the custom function, the function returns true.	Example
isAfter	A value that, when parsed as a date, refers to a moment after the configured JavaScript Date instance.	isAfter: new Date('Sat Nov 05 1605 00:00:00 GMT-0000')
isBefore	A value that, when parsed as a date, refers to a moment before the configured JavaScript Date instance.	isBefore: new Date('Sat Nov 05 1605 00:00:00 GMT-0000')
isBoolean	A value that is true or false	isBoolean: true
isCreditCard	A value that is a credit card number.	
isEmail	A value that looks like an email address.	isEmail: true
isHexColor	A string that is a hexadecimal color.	isHexColor: true
isIn	A value that is in the specified array of allowed strings.	isIn: ['paid', 'delinquent']
isInteger	A number that is an integer (a whole number)	isInteger: true
isIP	A value that is a valid IP address (v4 or v6)	isIP: true
isNotEmptyString	A value that is not an empty string	isNotEmptyString: true
isNotIn	A value that is not in the configured array.	isNotIn: ['profanity1', 'profanity2']
isNumber	A value that is a Javascript number	isNumber: true
isString	A value that is a string (i.e. typeof(value) === 'string')	isString: true
isURL	A value that looks like a URL.	isURL: true
isUUID	A value that looks like a UUID (v3, v4 or v5)	isUUID: true
max	A number that is less than or equal to the configured number.	max: 10000
min	A number that is greater than or equal to the configured number.	min: 0
maxLength	A string that has no more than the configured number of characters.	maxLength: 144
minLength	A string that has at least the configured number of characters.	minLength: 8
regex	A string that matches the configured regular expression.	regex: /^[a-z0-9]$/i

See: `https://sailsjs.com/documentation/concepts/models-and-orm/validations`

Error Handling in Controller

```
  new: async function (req, res) {
    sails.log.debug("Create new category....")
    res.view('pages/category/new', { "message": "", "name": "", "ordernumber": "" })
  },

  create: async function (req, res) {
    sails.log.debug("Create new category....")
    Category.create(req.allParams()).then(() => {
      res.redirect('/category');
    }).catch(
      (err) => {
        sails.log.debug("Error: " + err.message)
        res.view('pages/category/new', { "message": err.message, "name": 
req.body.name, "ordernumber": req.body.ordernumber })
      }
    );
  },
```

https://sailsjs.com/documentation/concepts/models-and-orm/errors#?usage-errors

Error Handling in View

```
<div class="container">
    <form action="/category" method="post">
       <div>
            <div class="form-group">
                <label class="col-form-label-lg">Name</label>
                <input type="text" class="form-control" name="name" maxlength="20"
value="<%= name %>">
            </div>
            <div class="form-group">
                <label class="col-form-label-lg">Reihenfolge</label>
                <div class="alert alert-danger"><%= message %></div>
                <input type="number" class="form-control" name="ordernumber" min="0"
max="100" type="number" value="<%= ordernumber %>">
            </div>
        </div>
        <button type="submit" class="green-button mt-4">Hinzufügen</button>
    </form>
</div>
```

Client Side

Validation in HTML Forms - Required

Use the required attribute to indicate that a field must be completed in order to pass validation.

```
<input required>
```

Source: https://riptutorial.com/html/example/2259/input-validation

Validation in HTML Forms - Minimum / Maximum Length

Use the minlength and maxlength attributes to indicate length requirements. Most browsers will prevent the user from typing more than max characters into the box, preventing them from making their entry invalid even before they attempt submission.

```
<input minlength="3">
<input maxlength="15">
<input minlength="3" maxlength="15">
```

Source: https://riptutorial.com/html/example/2259/input-validation

Validation in HTML Forms - Specifying a range

Use min and max attributes to restrict the range of numbers a user can input into an input of type number or range

```
Marks: <input type="number" size="6" name="marks" min="0" max="100" />
Subject Feedback: <input type="range" size="2" name="feedback" min="1" max="5" />
```

Source: `https://riptutorial.com/html/example/2259/input-validation`

Validation in HTML Forms - Match a Pattern

For more control, use the pattern attribute to specify any regular expression that must be matched in order to pass validation. You can also specify a title, which is included in the validation message if the field doesn't pass.

```
<input pattern="\d*" title="Numbers only, please.">
```

Source: https://riptutorial.com/html/example/2259/input-validation

Validation in Vuetify

- Vuetify provides components to assemble forms
- Vuetify provides convenient methods to validate form values

```
<v-sheet class="mx-auto">
    <v-form>
        <v-text-field v-model="name" :rules="rules" label="Name"></v-text-field>
        <v-text-field v-model="address" :rules="rules" label="Adresse"></v-text-field>
    </v-form>
</v-sheet>
```

Rules contains an array of validation rules

See: https://vuetifyjs.com/en/components/forms/

Validation in Vuetify

- If a **rule** returns a `string` or `false` a validation error is shown

```
data() {
    return {
      name: "",
      address: "",
      rules: [
        val => {
          const specialChars =
          '[`!@#$%^&*()_+-=[]{};\':"\\|<>/?~]/';
          if (specialChars
            .split('')
            .some((specialChar) => val.includes(specialChar))) {
              return "Der Name enthält Sonderzeichen! ";
            } else {
            return true;
          }
        },
      ],
    };
  },
```

See: https://vuetifyjs.com/en/components/forms/

Vue Validation Libraries

Vuetify provides integration for validation libraries:

Vee-validate:

vee-validate documentation can be found at: https://vee-validate.logaretm.com/v4/

Vuelidate

vuelidate documentation can be found at: https://vuelidate-next.netlify.app/