Using Validation Middleware

Mark Scott
AUTHOR

@tripletdad99

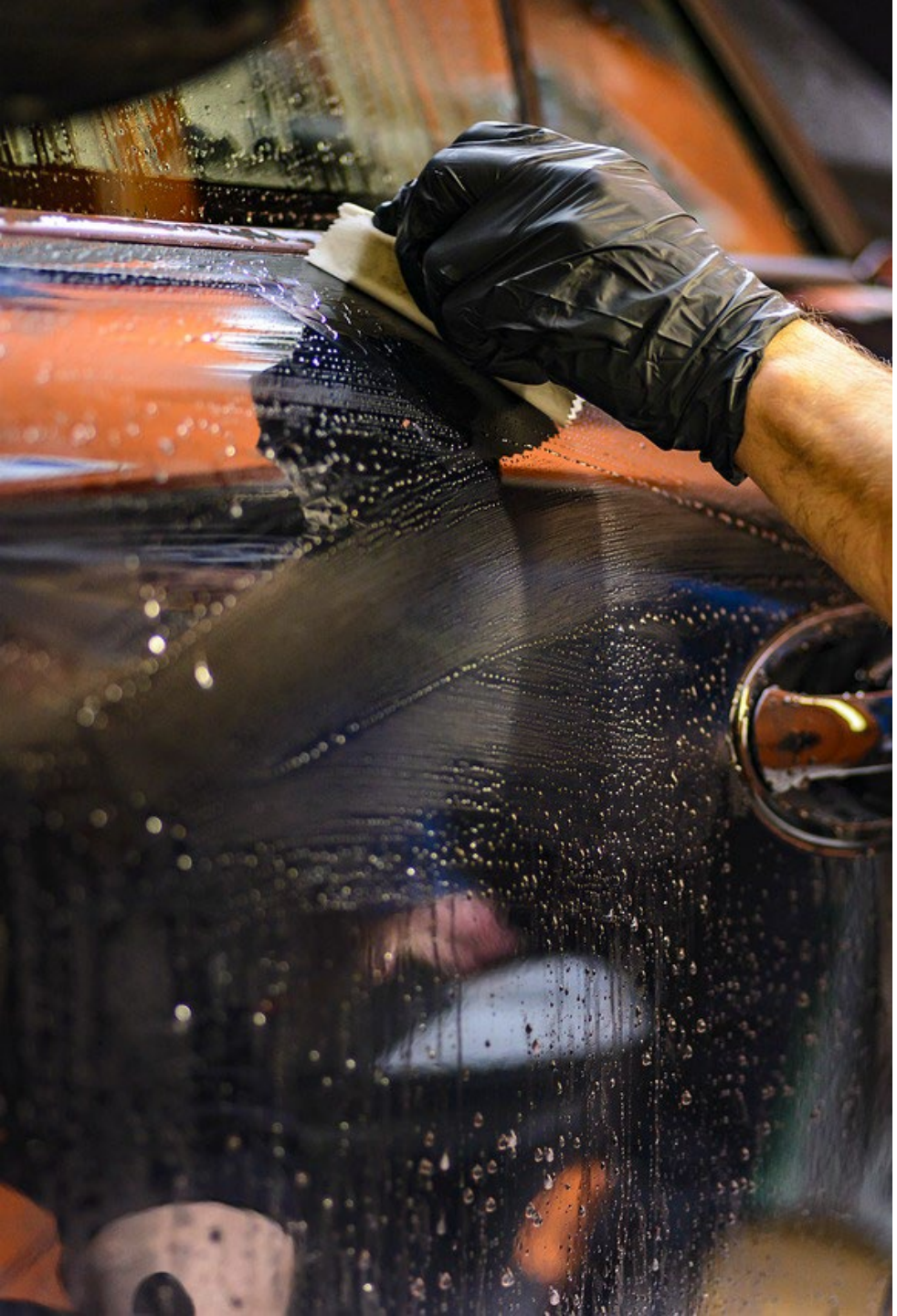
Introduction

Built-in mongoose validators

Introduction to middleware

Development of custom mongoose validators

Handling errors



Polish off the Express API Server

- Add validation
- Add error handling

Built-in Validators

SchemaTypes

- String
- Number
- Date
- Boolean
- Mixed
- ObjectId
- Array

Built-in Validators

String

Number

Date

Boolean

Mixed

ObjectId

Array

required

enum, match, minlength, maxlength

min, max

String – Required

```
// Required Validator Example

const customerSchema = new mongoose.Schema({
     name:      { type: String, required: true },
     address:   String,
     city:      String,
     state:     String,
     country:   String,
     zipCode:   Number,
     createdOn: Date,
     isActive:  Boolean
});

// After the schema is defined – via the path API
customerSchema.path('city').required(true, 'Oops! Supply a city.');

// Signature = required(required, [message])
```

String – Match and Enum

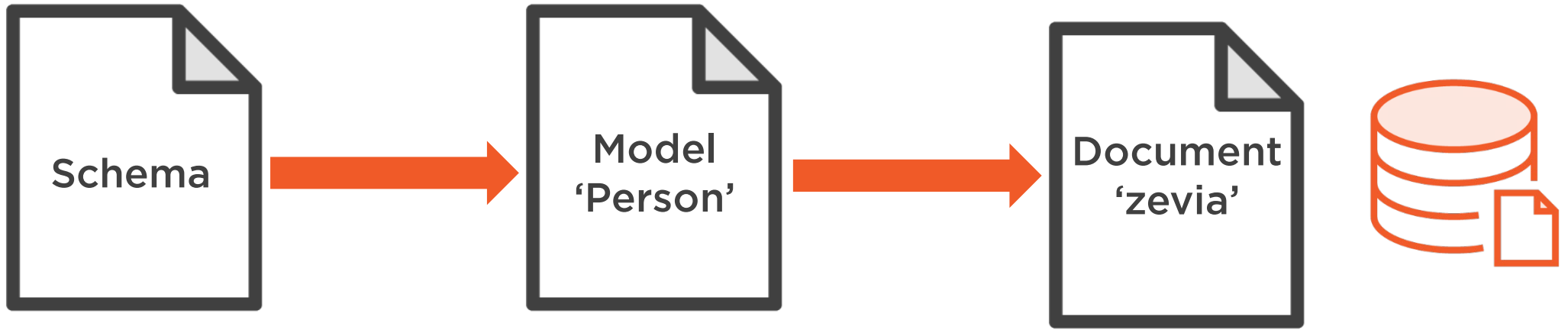
```
// String – Match Validator Example
const reMatch = /[a-zA-Z]/;

const customerSchema = new mongoose.Schema({
    name: { type: String,
            required: true,
            match: reMatch },
     // abbreviated...
});

// String – Enum Validator Example
const impediments = ['none', 'minor', 'blocking', 'severe'];

const standupSchema = new mongoose.Schema({
    // abbreviated...
    impediment: { type: String,
                  required: true,
                  enum: impediments }
});
```

Number – Min and Max

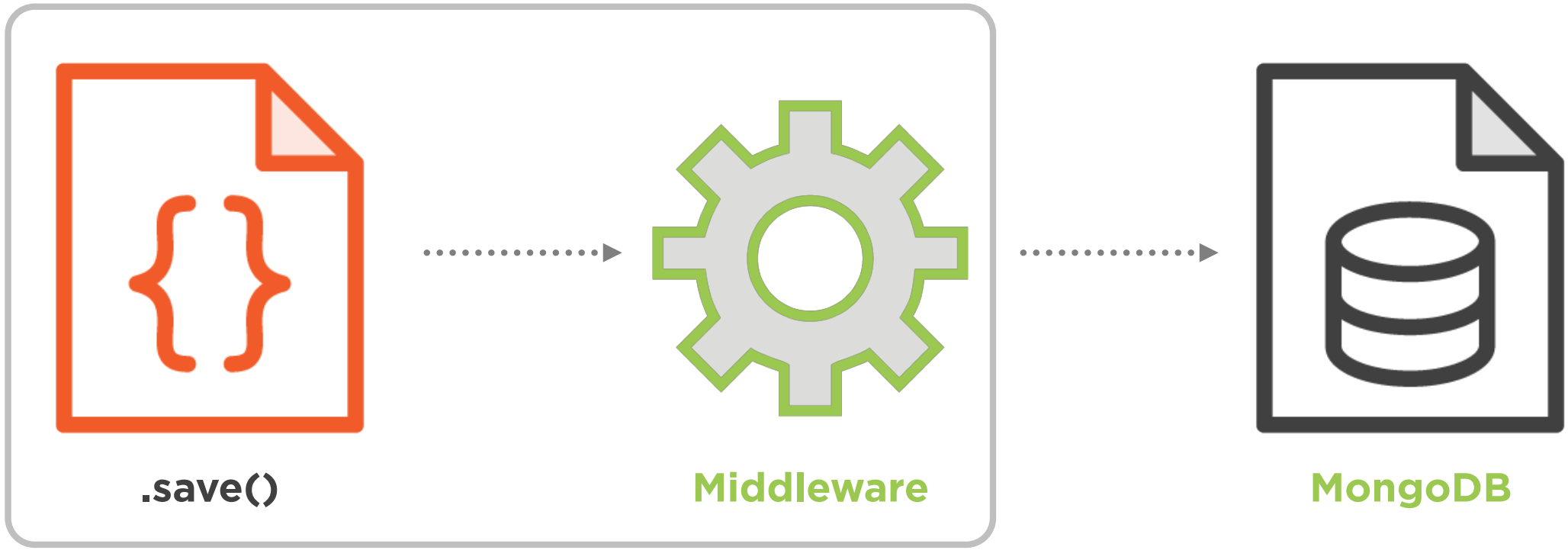
```
// Customers must receive at least a 5% discount
const customerSchema = new mongoose.Schema({
     name: String,
     // ...
     discount: { type: Number, min: 5 }
});

// Customers not allowed more than 60% discount
const customerSchema = new mongoose.Schema({
     name: String,
     // ...
     discount: { type: Number, max: 60 }
});

// Customers allowed a discount between 5% and 60% only
const customerSchema = new mongoose.Schema({
     name: String,
     // ...
     discount: { type: Number, min: 5, max: 60 }
});
```

Introduction to Middleware

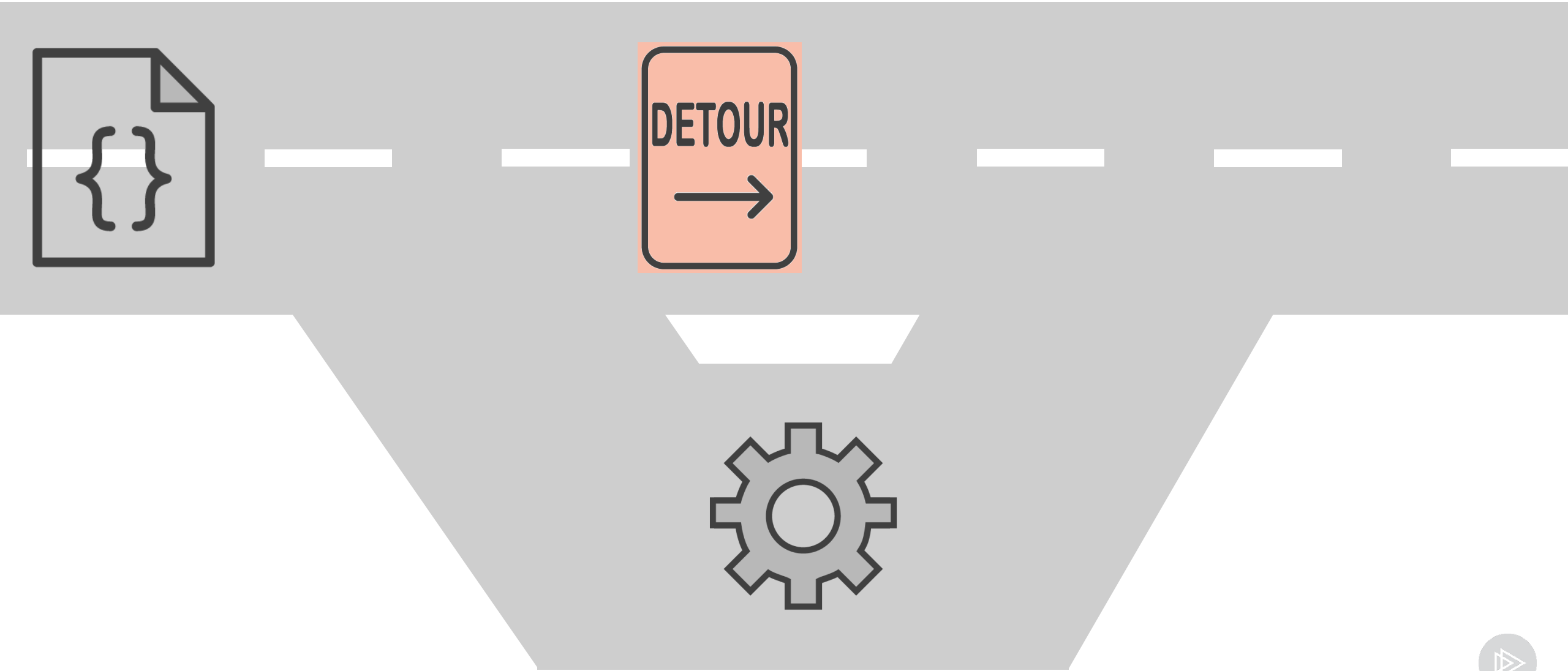
```
const personSchema = new mongoose.Schema({
    firstName: String,
    lastName:  String
});

const Person = mongoose.model('Person', personSchema);

const zevia = new Person({
    firstName: 'Zevia',
    lastName = 'Cola'
});
```



Middleware



Middleware Detour



Middleware Example

```
// Middleware execution flow example...
const personSchema = new Schema({
   firstName: { type: String, required: true },
   lastName:  { type: String, required: true },
   status:    { type: String, required: true, default: 'Alive' }
});

// Build a model from the person schema
const Person = new mongoose.model('Person', personSchema);

// New document instance of a Person model
const newPerson = new Person( { firstName: 'Summer', lastName: 'Heat' } );

// Save the document... Internal validation (required) kicks off now
newPerson.save(function (err) {
  if (err) return handleError(err);
  // saved the person document!
});
```

Flow of Execution

```
// Save the document... Internal validation (required) kicks off now
newPerson.save(function (err) {
  if (err) return handleError(err);
  // saved the person document!
});
```

Custom Validators

Built-in Validators

String

Number

Date

Boolean

Mixed

ObjectId

Array

required

enum, match, minlength, maxlength

min, max

Custom Validator Example

```
// Custom validation - method signature = validate(obj, [errorMsg])
const sizeValidator = [
    function (val) {
        return (val.length > 0 && val.length <= 50)
    },
    // Custom error text...
    'String must be between 1 and 50 characters long'
];

const personSchema = new mongoose.Schema({
    firstName: { type: String, required: true, validate: sizeValidator },
    lastName:  { type: String, required: true, validate: sizeValidator },
    status:    { type: String, required: true, default: 'Alive' }
});
```

Handling Validation Errors

Custom Validator Example

```
// Custom validation - method signature = validate(obj, [errorMsg])
const sizeValidator = [
    function (val) {
        return (val.length > 0 && val.length <= 50)
    },
    // Custom error text...
    'String must be between 1 and 50 characters long'
];

const personSchema = new mongoose.Schema({
    firstName: { type: String, required: true, validate: sizeValidator },
    lastName:  { type: String, required: true, validate: sizeValidator },
    status:    { type: String, required: true, default: 'Alive' }
});
```

Validation Error Example

```
// New document instance of a Person model
const newPerson = new Person(
    {
        firstName: ‘Some obnoxiously long name that is more that 50 characters
long...’,
        lastName: ‘Crazy’
    }
);

// Save the person document now – validate and check for errors
newPerson.save( function (err) {
    if (err) {
        console.log('Sorry, there was an error: ' + err);
    }
    else {
        console.log('Your person was saved!');
    }
});
```

Error Object Example

```
{
   "errors": {
      “firstName": {
         "message": "String must be between 1 and 50 characters long",
         "name": "ValidatorError",
         "properties": {
            "message": "String must be between 1 and 50 characters long",
            "type": “user defined",
            "path": “firstName",
            "value": "Some obnoxiously long name that is more that 50 characters long..."
        },
        "kind": “user defined",
        "path": “firstName",
        "value": "",
        "$isValidatorError": true
      }
   },
   // truncated for brevity sake...
}
```

Error Message Templates

{PATH} {VALUE} {TYPE} {MIN} {MAX}

```
const mongoose = require('mongoose');

mongoose.Error.message.Number.min = "{PATH} is too low! Must be at least {MIN}.";
```

```
msg.Number.min = "Path '{PATH}' ({VALUE}) is less than minimum allowed value ({MIN}).";
```

Custom Validator Example

```
// Custom validation - method signature = validate(obj, [errorMsg])
const sizeValidator = [
    function (val) {
        return (val.length > 0 && val.length <= 50)
    },
    // Custom error text...
    '{PATH} must be between 1 and 50 characters long'
];

const personSchema = new mongoose.Schema({
    firstName: { type: String, required: true, validate: sizeValidator },
    lastName:  { type: String, required: true, validate: sizeValidator },
    status:    { type: String, required: true, default: 'Alive' }
});
```

Adding Validation

Standup – No Validation

```
const standupSchema = new mongoose.Schema({
    teamMemberId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'teamMembers‘
    },
    teamMember: { type: String },
    project: { type: String },
    workYesterday: { type: String },
    workToday: { type: String },
    impediment: { type: String },
    createdOn: { type: Date, default: Date.now }
})
```

Standup – Required Validator

```
const standupSchema = new mongoose.Schema({
    teamMemberId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'teamMembers‘
    },
    teamMember: { type: String, required: true },
    project: { type: String , required: true },
    workYesterday: { type: String , required: true },
    workToday: { type: String , required: true },
    impediment: { type: String , required: true, default: ‘None’ },
    createdOn: { type: Date, default: Date.now }
})
```

Required String Custom Validator

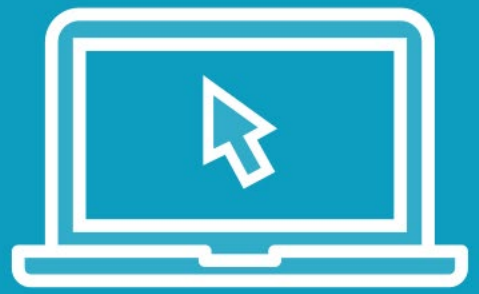
```
const requiredStringValidator = [
    function (val) {
        let testVal = val.trim()
        return (testVal.length > 0)
    },
    // Custom error text
    'Please supply a value for {PATH}'
]
```

```
const standupSchema = new mongoose.Schema({
    teamMemberId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'teamMembers‘
    },
    teamMember: { type: String, required: true },
    project: { type: String , required: true },
    workYesterday: {
        type: String,
        required: true,
        validate: requiredStringValidator
    },
    workToday: { type: String , required: true },
    impediment: { type: String , required: true, default: ‘None’ },
    createdOn: { type: Date, default: Date.now }
})
```

TeamMember – Size Custom Validator

```
const sizeValidator = [
    function (val) {
        let testVal = val.trim()
        return (testVal.length > 0 && testVal.length <= 50)
    },
    // Custom error text...
    '{PATH} must be between 1 and 50 characters long'
]

const teamMemberSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        validate: sizeValidator
    }
})
```

Demo



Add validation to the demo application

Start with required validation

- Custom validators in place (not used)
- Test in Postman

Add custom validator to schema

- standup.js
- teamMember.js

Summary

Started with required validation only

- Test in Postman

Add custom validators to the schema

- standup.js
- teamMember.js

Home work

- Build a custom validator for the project schema