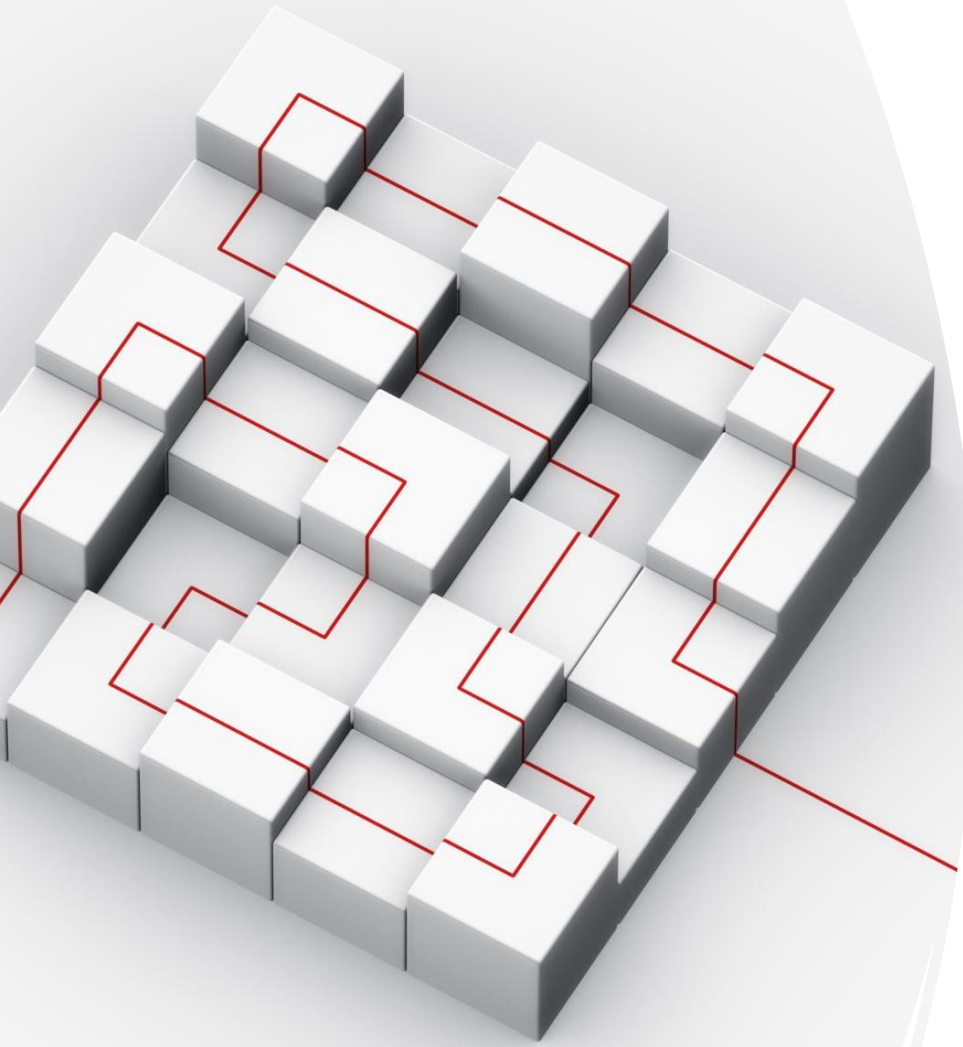
INT 161

Basic Backend Development

RESTful API

Input Validation



Unit Objectives

- After completing this unit, you should be able to:
  - Explain the importance of input validation
  - Describe how Joi works as a validation library
  - Implement Joi schemas for validating API requests
  - Integrate validation module into Express.js routes

What is Input Validation?

- Checking if the data sent to your API is correct.
- Makes sure required fields are filled.
- Checks if the format of the data is right (like email, age, etc.).
- Protects your system from bad or harmful data.

Why is Input Validation Important?

- Prevents application errors
- Enhances security (avoids injection attacks)
- Improves API reliability and client trust
- Gives helpful error messages to users.
- Protects database integrity

- Validate at FE vs Validate at BE ? & Concurrent user issues

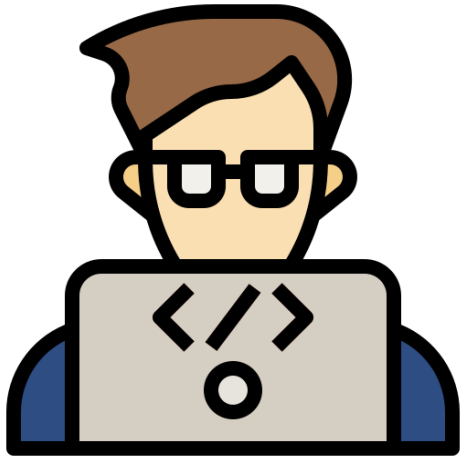
Express Request Lifecycle

Request → Middleware (Error Handler, Validation, etc.) → Route Handler → (CRUD) → Response

Manual Validation Example

```
app.post('/user', (req, res, next) => {
   const { name, email } = req.body;
   if (!name || !email.includes('@')) {
      next(new Error('Invalid input'));
   }
   next();
});
```

Practices



Create validator mw (1/2) : ./middlewares/film-validate.js

```
export const validateFilm = (req, res, next) => {
    let {title, description, releaseYear, rating, length,
        specialFeatures, actors, categories } = req.body;

    const validateErrors = [];
    title = title?.trim();
    description = description?.trim();
    releaseYear = Number(releaseYear) ;

    const currentYear = new Date().getFullYear();

    if (!title)
        validateErrors.push({title : 'Title is required'});
    else if (title.length < 4)
        validateErrors.push({title :'Title must be at least 3 characters'});
```

Create validator mw (2/2) : ./middlewares/film-validate.js

```
    if(!releaseYear || isNaN(releaseYear) || releaseYear < 1900
       || releaseYear > currentYear)
       validateErrors.push({releaseYear :
           `Release year must be a number between 1900,${currentYear}`});

    if (validateErrors.length > 0) {
         const err = new Error('Validation error');
         err.status = 400;
         err.code = 'VALIDATION_ERROR';
         err.errors = validateErrors;
         return next(err);
    }
    delete req.body['releaseYear'];   //why ?
    req.body.language_id = 1;
    req.body.release_year = releaseYear;
    next();
}
```

Use validator in film-route.js

```
  router.post('/', validateFilm, controller.create);
```

Modify Error Handler : `app.js`

```
// error handler
app.use(function (err, req, res, next) {
    const status = err.status || 500;
    const errorCode = err.code || (err.message.toUpperCase().replace(/ /g, '_'));
    res.status(status);
    res.json({
        error: errorCode,
        status: status,
        message: err.message,
        resource: req.originalUrl,
        timestamp: new Date().toLocaleString(),
        ...(err.errors && {errors: err.errors}),
    });
});
```

Modify Film - repository/service

```
update: async function (fid, data) {
    ({actors, ...film} = data);
    return await prisma.film.update({
        where: {id: fid},
        data: film,
    });
},
create: async function (data) {
    return await prisma.film.create({
        data: data,
    });
},
findByTitle: async function (title) {
    return await prisma.film.findFirst({
        where: {title: title}
    });
}
```

```
update: async function (id, data) {
    await this.getById(id);
    const sameTitleFilm = await repo.findByTitle(data.title);
    if (sameTitleFilm && sameTitleFilm.id !== id) {
        throw errResp.duplicateItem(data.title, 'Film');
    }
    return await repo.update(id, data);
},
create: async function (data) {
    const sameTitleFilm = await repo.findByTitle(data.title);
    if (sameTitleFilm) {
        throw errResp.duplicateItem(data.title, 'Film');
    }
    return await repo.create(data);
},
```

Modify Film - controller/route

```
update: async function (req, res) {
    const data = req.body;
    const id = Number(req.params.id);
    const film = await service.update(id, data);
    res.json(film);
},
create: async function (req, res) {
    const data = req.body;
    const film = await service.create(data);
    res.json(film);
},
```

```
const {validateFilm} = require("../middlewares/film-validate");

router.get('/', controller.list);
router.get('/:id', controller.get);
router.put('/:id', controller.update);
router.post('/', validateFilm, controller.create);
```

Test:

POST with <<no body>>

```
{
    "name" : "New Series Y ",
    "releaseYear" : "1885",
    "length" : 90
}
```

```
{
    ”title" : " XO",
    "releaseYear" : "1885",
    "length" : 90
}
```

```
{
    ”title" : " New Series Y",
    "releaseYear" : ”2029",
    "length" : 90
}
```

```
{
    ”title" : " New Series Y",
    "releaseYear" : ”2029",
    "length" : 90,
    ”remark" : ”Recommended"
}
```

```
{
    ”title" : ”New Series Y",
    "releaseYear" : ”2024",
    "length" : 90
}
```

Express.js Middleware Validators

- Joi
  - Schema-based
  - Powerful declarative schema validation by Hapi team
- express-validator
  - Middleware-based
  - Uses validator.js internally, integrates tightly with Express routes
- Yup
  - Schema-based
  - Inspired by Joi, commonly used with front-end frameworks (React, Formik)
- Zod
  - Schema-based + TypeScript native
  - Type-safe runtime validation and type inference

Comparison

7️⃣ Quick Comparison Table

Feature	Joi	express-validator	Yup	Zod
Schema-based	✅	❌	✅	✅
Middleware-ready	✅	✅	⚠️ (custom needed)	⚠️ (custom needed)
TypeScript support	⚠️ Partial	⚠️ Manual	⚠️ Partial	✅ Strong
Complex objects	✅	⚠️	⚠️	✅
Performance	⚠️ Medium	✅ Fast	✅ Fast	✅ Fast
Ecosystem maturity	⭐⭐⭐⭐	⭐⭐⭐	⭐⭐⭐	⭐⭐

Joi : https://joi.dev

- Pros:
  - Mature and widely used
  - Supports complex nested objects, conditions, and custom rules
  - Easy to integrate as a middleware
  - Built-in sanitization (stripUnknown)

- Cons:
  - Not TypeScript-native (needs type inference manually)
  - Slightly heavier runtime cost for very large payloads

Understanding Joi Library

- Installation:
  - npm install joi
- Basic Schema Concepts
  - Defining schemas
  - Validating objects
  - Using .validate() and .validateAsync()
- Common Joi Data Types:
  - string(), number(), boolean(), array(), object(), date()
- Useful Joi Methods:
  - .required(), .optional(), .min(), .max(), .pattern(), .valid(), .default()

Express Integration Basics

- Typical API structure (controller, routes, middleware)
- Where validation fits in request lifecycle
- Creating a Reusable Validation Middleware

```
export const validate = (schema, property = 'body') => (req, res, next) => { ... }
```

- Validating Different Request Parts
  - Request Body (req.body) (POST, PUT, PATCH)
  - Query Parameters (req.query) (filters, pagination, sorting)
  - URL Parameters (req.params) (ID-based lookups)
  - Headers (req.headers) (For custom auth tokens or API keys)

Joi schema example :

```
import Joi from 'joi';


const schema = Joi.object({
    title: Joi.string().min(4).max(100).required(),
    genre: Joi.string().valid('Action', 'Comedy', 'Drama').required(),
    released: Joi.number().integer().min(1900)
       .max(new Date().getFullYear()).required(),
    rating: Joi.number().min(0).max(10).optional(),
})
```

```
const { error, value } = schema.validate({ title: ”XO", genre: 17, rating: ”PG” });
// { Handle error if has error }
```

Generic Validation Middleware

```
export const validate = (schema, property = 'body') => {
  return (req, res, next) => {
    const { error, value } = schema.validate(req[property], {
      abortEarly: false, // show all errors
      stripUnknown: true // remove unknown fields
    });
    if (error) {
      const err = new Error('Validation error: ');
      err.errors = error.details.map(d => ({
        field: d.path.join('.'),
        message: d.message.replace(/"/g, '')
      }))
      next(err);
    }

    req[property] = value; // sanitized input
    next();
  };
};
```

Integrating Joi with express: @router

```
const {validate} = require('../validators/validate');
const {filmSchema, filmQuerySchema} =require('../validators/film-validator');

router.get('/', validate(filmQuerySchema, 'query'),controller.list);

router.get('/:id', controller.get);
router.put('/:id', controller.update);
router.delete('/:id', controller.delete);

router.post('/', validate(filmSchema, 'body'), controller.create);
```

Common Joi types

Type	Example
String	Joi.string().min(3).max(100)
Number	Joi.number().integer().min(0)
Boolean	Joi.boolean()
Date	Joi.date().iso()
Array	Joi.array().items(Joi.string())
Object	Joi.object({...})

Useful Joi Methods

Method	Description
.required()	Field must exist
.optional()	Field may be omitted
.default(value)	Apply default value
.valid(...values)	Restrict to specific values
.invalid(...values)	Restrict to specific values
.pattern(regex)	Match regex pattern
.email()	Validate email format
.when()	Conditional validation
.alphanum()	Allows only letters and digits.
.custom	Custom validation rule.

```
const schema = Joi.object({
   type: Joi.string().valid('A', 'B').required(),
   discount: Joi.number().when('type', {
      is: 'B',
      then: Joi.required(),
      otherwise: Joi.forbidden()
   })
});
```

```
const schema = Joi.object({
   password: Joi.string().custom((value, helpers) => {
      if (!/[A-Z]/.test(value)) {
         return helpers.error('any.invalid');
      }
      return value;
   }, 'Password uppercase rule')
});
```

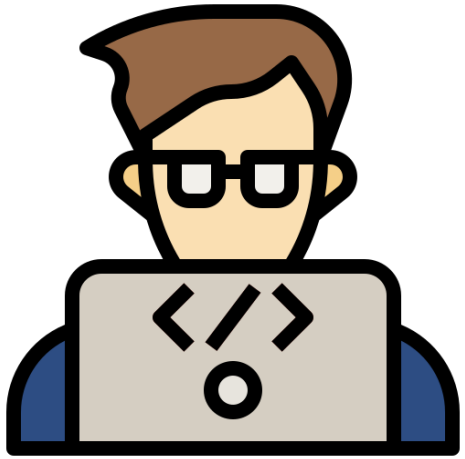
```
const schema = Joi.object({
   phone: Joi.string().pattern(/^[0-9]{10}$/)
});
```

Custom Messages

```
export const filmSchema = Joi.object({

    title: Joi.string().trim()
        .min(3).max(128).required()
        .messages({
            'string.min': 'Title >= 3 characters',
            'string.max': 'Title <= 128 characters',
            'any.required': 'Title is required',
            }),
});
```

Practices



Modify Model Film: `prisma.schema`

```
model Film {
  id                   Int             @id @default(autoincrement())
                                        @map("film_id") @db.UnsignedSmallInt
  title                String          @db.VarChar(128)
  description          String?         @db.Text
  releaseYear          Int?            @map("release_year") @db.Year
  language_id          Int             @db.UnsignedTinyInt
  original_language_id Int?            @db.UnsignedTinyInt
  rental_duration      Int             @default(3) @db.UnsignedTinyInt
  rental_rate          Decimal         @default(4.99) @db.Decimal(4, 2)
  length               Int?            @db.UnsignedSmallInt
  replacement_cost     Decimal         @default(19.99) @db.Decimal(5, 2)
  rating               FilmRating?     @default(G)
  special_features     String?
  last_update          DateTime        @default(now()) @db.Timestamp(0)
  film_actor           FilmActor[]
  film_category        FilmCategory[]

  @@index([language_id], map: "idx_fk_language_id")
  @@index([original_language_id], map: "idx_fk_original_language_id")
  @@index([title], map: "idx_title")
}
```

Modify Model Dto: `FilmDetailDto, SimpleFilmDto`

```
class FilmDetailDto {
    constructor(film = {}) {
        const {id, title, description, length, releaseYear,
            special_features,rating, film_actor, film_category} = film;
        this.id = id ?? null;
        this.title = title ?? '-';
        this.releaseYear = releaseYear ?? '-';
        this.rating = rating ?? '-';
        this.description = description ?? '-';
```

```
class SimpleFilmDto {
    constructor(film = {}) {
        const {id, title, releaseYear, rating, film_actor, film_category} = film;
        this.id = id ?? null;
        this.title = title ?? '-';
        this.releaseYear = releaseYear ?? '-';
        this.rating = rating ?? '-';
```

Create Joi Validators (1/2): ./validators/film-validator.js

```
import Joi from 'joi';
const filmRating = {G:'G', PG:'PG',PG_13:'PG-13',R:'R',NC_17:'NC-17'};
export const filmSchema = Joi.object({

   title: Joi.string().trim()
      .min(3).max(128).required()
      .messages({
         'string.min': 'Title >= 3 characters',
         'string.max': 'Title <= 128 characters',
         'any.required': 'Title is required',
         }),
   releaseYear: Joi.number().integer()
      .min(1900)
      .max(new Date().getFullYear())
      .optional(),
```

```
   rating: Joi.string().valid(...Object.keys(filmRating)).optional(),
      language_id: Joi.number().integer().min(1).optional().default(1),
      description: Joi.string().optional(),
      actors: Joi.array().items(Joi.object({})).optional(),
      length: Joi.number().integer().min(30).optional(),
      stripUnknown : true,
});
```

Create Joi Validators (2/2): ./validators/film-validator.js

```
export const filmQuerySchema = Joi.object({
    page: Joi.number().integer().min(1).default(1),
    pageSize: Joi.number().integer().min(5).max(30).default(10),
    sortBy: Joi.string()
        .pattern(/^[a-zA-Z_]+:(asc|desc)$/)
        .message('Sort must be in format field:asc or field:desc')
        .default('id:asc'),
});
```

Generic Validation Middleware: ./validators/validate.js

```
export const validate = (schema, property = 'body') => {
  return (req, res, next) => {
    const { error, value } = schema.validate(req[property], {
      abortEarly: false, // show all errors
      stripUnknown: true // remove unknown fields
    });

    if (error) {
      const err = new Error('Validation error: ');
      err.status = 400;
      err.code = 'VALIDATION_ERROR';
      err.errors = error.details.map(d => ({
        field: d.path.join('.'),
        message: d.message.replace(/"/g, '')
      }))
     console.log(error: ‘, error);
      next(err);
    }

    req[property] = value; // sanitized input
    console.log('value: ', value);
    console.log('request-property: ', req[property]);
    console.log('request-body: ', req.body);
    next();
  };
};
```

Integrating Joi with express: @ film-route

```
const express = require('express');
const router = express.Router();
const controller = require('../controllers/film-controller');
// const {validateFilm} = require("../middlewares/film-validate");
const {validate} = require('../validators/validate');
const {filmSchema, filmQuerySchema} =require('../validators/film-validator');

router.get('/', validate(filmQuerySchema, 'query'),controller.list);
router.get('/:id', controller.get);
router.put('/:id', controller.update);
router.delete('/:id', controller.delete);
// router.post('/', validateFilm, controller.create); // manual validation
router.post('/', validate(filmSchema, 'body'), controller.create);

module.exports = router;
```

Test:

POST with <<no body>>

```
{
    "name" : "New Series Y ",
    "releaseYear" : "1885",
    "length" : 90
}
```

```
{
    ”title" : " XO",
    "releaseYear" : "1885",
    "length" : 90
}
```

```
{
    ”title" : " New Series Y",
    "releaseYear" : ”2029",
    "length" : 90
}
```

```
{
    ”title" : " New Series Y",
    "releaseYear" : ”2029",
    "length" : 90,
    ”remark" : ”Recommended"
}
```

```
{
    ”title" : ”New Series Y",
    "releaseYear" : ”2024",
    "length" : 90
}
```