COMPOSANTS ACCES BDD

JOUR 2 : VALIDATION DES DONNEES

PRESENTE PAR NATACHA DESSE

AFEC

DEROULE SEMAINE

- Composants d’accès SQL
- **Validation des données**
- MongoDB
- Composants d’accès aux données NoSQL

VALIDATION DES DONNEES

LES OBJECTIFS

Savoir quel type de données vérifier

Mettre en place des règles métiers pour la vérification

Empêcher l'implémentation en base de données pour des données invalides

VALIDATION MANUELLE

NUAGE DE MOTS

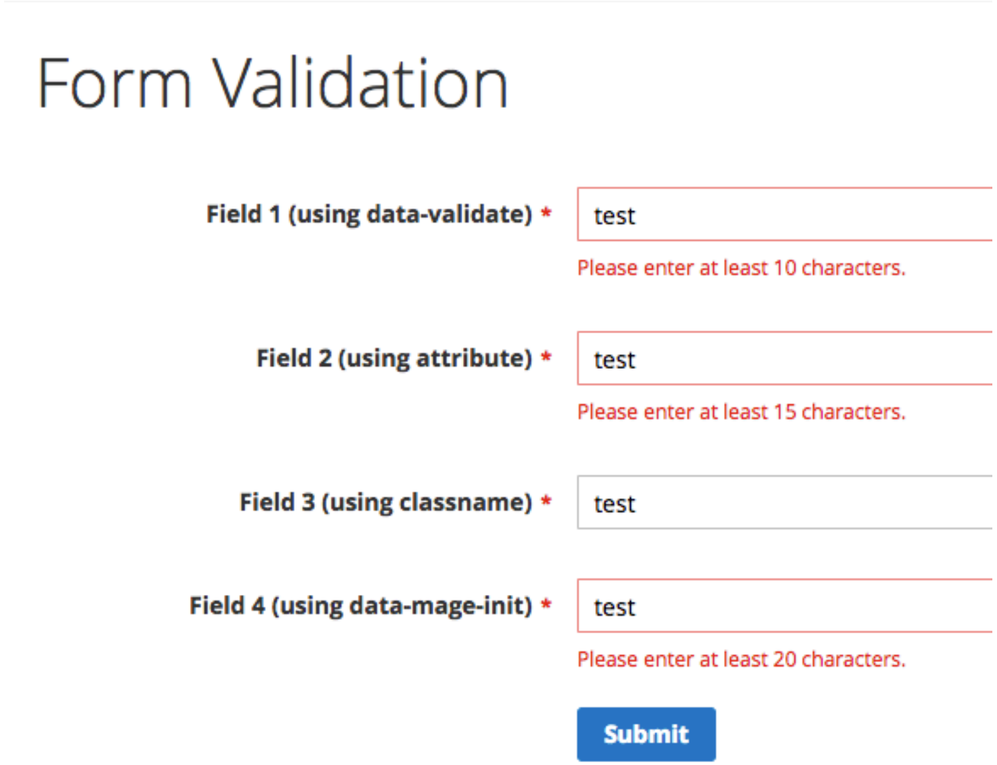
Quel données utilisateur peut on valider ?

https://www.menti.com/aldpp4773piz

POURQUOI VALIDER LES DONNEES ?

Valider les données, c'est contrôler leur qualité, leur forme et leur cohérence avant de les enregistrer ou de les utiliser

- **Éviter** les **erreurs** techniques (ex : insertion en base échouée).
- Assurer la **cohérence métier** (ex : un âge ne peut pas être négatif).
- Renforcer la **sécurité** (ex : éviter les injections ou scripts).
- **Améliorer** l'**expérience** utilisateur (erreurs claires, précises).
- **Prévenir** les attaques (XSS, injection SQL, spam, etc.).



VALIDATION COTE CLIENT VS COTE SERVEUR

Côté client	**Côté serveur**
Empêche de soumettre un formulaire invalide	Protège la base et le serveur
UX améliorée (messages immédiats)	Ne peut pas être contourné
Peut être désactivé ou contourné	Est obligatoire pour la sécurité

Conclusion : toujours valider côté serveur, même si le client le fait déjà

CONTEXTE

Exemple de données envoyées par un client :

```json
{
  "name": "Alice",
  "email": "alice@example.com",
  "age": 25
}
```

Objectif : Avant de faire :

```
connection.query('INSERT INTO users (...) VALUES (...)', ...)
```

... il faut valider que

- *name* est présent et non vide,
- *email* est valide,
- *age* est un entier positif.

TYPES DE VALIDATION

Présence :

S’assurer qu’une donnée est bien envoyée :

```
if (!user.name) {
  return res.status(400).send("Le nom est requis.");
}
```

Type :

Contrôler que le type correspond à l’attendu :

```
if (typeof user.age !== "number") {
  return res.status(400).send("L'âge doit être un nombre.");
}
```

Format :

Utilisation de regEx

```
const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
if (!emailRegex.test(user.email)) {
  return res.status(400).send("Email invalide.");
}

```

Plage de valeur

```
if (user.age < 0 || user.age > 120) {
  return res.status(400).send("L'âge doit être entre 0 et 120.");
}
```

TYPES DE VALIDATION

Longueur minimale ou maximale:

```
1 if (user.name.length < 2) {
2   return res.status(400).send("Le nom doit contenir au
  moins 2 caractères.");
3 }
4
```

Valeur autorisées (ENUM)

```
1 const roles = ["admin", "user", "guest"];
2 if (!roles.includes(user.role)) {
3   return res.status(400).send("Rôle invalide.");
4 }
5
```

ERREURS FREQUENTES

Insertion de chaînes vides

```
1 if (user.name.trim() === "") { ... }
```

Comparaison à null seulement

Ne pas oublier que undefined existe aussi :

```
1 if (user.age == null) // couvre null et undefined
```

Confiance aveugle dans l'objet req.body

Vérifier les propriétés une par une

Ne pas valider les types

Un formulaire HTML envoie parfois des chaînes :

```
1 console.log(typeof req.body.age); // souvent "string"
```

EXEMPLE

```
app.post("/users", (req, res) => {
  const user = req.body;

  // Vérification présence
  if (!user.name || !user.email || user.age == null) {
    return res.status(400).send("Champs obligatoires : nom, email, âge.");
  }

  // Types
  if (typeof user.name !== "string" || typeof user.email !== "string") {
    return res.status(400).send("Nom et email doivent être des chaînes.");
  }
  if (typeof user.age !== "number") {
    return res.status(400).send("L'âge doit être un nombre.");
  }

  // Format email
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (!emailRegex.test(user.email)) {
    return res.status(400).send("Email invalide.");
  }

  // Plage d'âge
  if (user.age < 0 || user.age > 120) {
    return res.status(400).send("Âge hors limite raisonnable.");
  }

  // Longueur du nom
  if (user.name.length < 2 || user.name.length > 50) {
    return res.status(400).send("Le nom doit contenir entre 2 et 50 caractères.");
  }

  // Si tout est bon, insertion
  connection.query(
    "INSERT INTO users (name, email, age) VALUES (?, ?, ?)",
    [user.name, user.email, user.age],
    (err, results) => {
      if (err) return res.status(500).send("Erreur base de données");
      res.status(201).json({ id: results.insertId, ...user });
    }
  );
});

```

EXERCICES

Trouvez les validations

IDENTIFIER LES CHAMPS A VALIDER

Objectif

À partir d'un ensemble de données envoyées par un utilisateur, l'objectif est d'identifier toutes les vérifications à réaliser avant d'insérer ces données en base de données avec Node.js

Consignes

Pour chaque champ, indique :

1. Les vérifications à effectuer
2. Le type attendu
3. Des contraintes ou règles métier éventuelles

Données reçues extrait de req.body


```
{
  "username": "marie27",
  "email": "marie27@gmail.com",
  "password": "123abc",
  "confirmPassword": "123abc",
  "age": 16,
  "role": "admin"
}
```

AIDE

Exemple de validation

- **Type** attendu : string
- **Validation** de **présence** : Oui, obligatoire
- **Validation** de **type** : typeof
- **Validation** de **longueur** : minimum 3 caractères, maximum 20
- **Validation** de **format** : lettres, chiffres, éventuellement underscore (^[a-zA-Z0-9_]{3,20}$)
- **Règle métier** :
  - Ne pas accepter les espaces ni les caractères spéciaux
  - Unicité à vérifier en base (pas deux utilisateurs avec le même pseudo)

REPONSE

Username

- **Type** attendu : string
- **Validation** de **présence** : Oui, obligatoire
- **Validation** de **type** : typeof username === "string"
- **Validation** de **longueur** : minimum 3 caractères, maximum 20
- **Validation** de **format** : lettres, chiffres, éventuellement underscore (^[a-zA-Z0-9_]{3,20}$)
- **Règle métier** :
  - Ne pas accepter les espaces ni les caractères spéciaux
  - Unicité à vérifier en base (pas deux utilisateurs avec le même pseudo)

Email

- **Type** attendu : string
- **Validation** de **présence** : Oui, obligatoire
- **Validation** de **type** : typeof email === "string"
- **Validation** de **format** :
  - Regex recommandée : /^[^\s@]+@[^\s@]+\.[^\s@]+$/
- **Règle métier** :
  - Vérifier l'unicité en base
  - S'assurer que l'utilisateur ne puisse pas s'enregistrer plusieurs fois avec le même email

REPONSE

Password

- **Type** attendu : string
- **Validation** de **présence** : Oui, obligatoire
- **Validation** de **type** : typeof password === "string"
- **Validation** de **longueur** : au moins 8 caractères
- Validation de **sécurité** (recommandée) :
  - 1 lettre minuscule
  - 1 lettre majuscule
  - 1 chiffre
  - 1 caractère spécial

- **Règle métier** :
  - Ne jamais stocker en clair : il doit être haché (avec bcrypt par exemple)
  - Ne pas retourner le mot de passe dans les réponses
  -

Confirm password

- **Type** attendu : string
- **Validation** de **présence** : Oui, si password est présent
- **Validation** de **cohérence** :
- Il doit correspondre exactement au champ password

- **Règle métier** :
  - Ne doit pas être stocké en base
  - Peut être comparé côté serveur, ou côté client (mais le serveur doit revalider)

REPONSE

Age

- **Type attendu** : number (attention : les formulaires envoient souvent des chaînes !)
- **Validation** de **présence** : Facultatif selon les cas, mais souvent recommandé
- **Validation** de **type** :
  - typeof age === "number" ou forcer conversion avec parseInt
- **Validation** de **valeur** :
  - Minimum : 13 (âge légal minimal d'inscription dans de nombreux pays)
  - Maximum : 120 (pour éviter les abus)

- **Règle métier** :
  - Si non requis, alors mettre une valeur par défaut (NULL ou 0)
  - Interdire les âges négatifs ou absurdes

Role

- **Type attendu** : string
- **Validation** de **présence** : Optionnelle (on peut forcer le rôle par défaut côté serveur)
- **Validation** de **type** : typeof role === "string"
- **Validation** de **valeur autorisée** :
  - Liste restreinte : ["user", "admin", "moderator"]

- **Règle métier** :
  - Ne jamais faire confiance à un rôle envoyé par un formulaire !
  - Forcer role = 'user' côté serveur, sauf pour les comptes créés manuellement par un admin.

IMPLEMENTER LA VALIDATION POUR USER

Username

- **Type** attendu : string
- **Validation** de **présence** : Oui, obligatoire
- **Validation** de **type** : typeof username === "string"
- **Validation** de **longueur** : minimum 3 caractères, maximum 20
- **Validation** de **format** : lettres, chiffres, éventuellement underscore (^[a-zA-Z0-9_]{3,20}$)
- **Règle métier** :
  - Ne pas accepter les espaces ni les caractères spéciaux
  - Unicité à vérifier en base (pas deux utilisateurs avec le même pseudo)

```
app.post("/users", (req, res) => {
  const { username } = req.body;

  // Vérifier que le champ est présent
 //Ecrire condition ici {
    return res.status(400).json({ error: "Le champ 'username'
  }

  // Vérifier que c'est une chaîne de caractères
  // Vérifier le typeof {
    return res.status(400).json({ error: "Le champ 'username'
  }

  // Supprimer les espaces inutiles
  const trimmedUsername = username.trim();

  // Vérifier que le champ n'est pas vide après trim
  //Vérifier que la longueur n'est pas égale à 0 {
    return res.status(400).json({ error: "Le champ 'username'
  }

  // Vérifier la longueur
  // Vérifier que ça soit compris entre 3 et 20 {
    return res.status(400).json({ error: "Le 'username' doit
  }

  // Vérifier le format (lettres, chiffres, underscore uniqu
  const usernameRegex = /^[a-zA-Z0-9_]+$/;
  if (!usernameRegex.test(trimmedUsername)) {
    return res.status(400).json({ error: "Le 'username' ne do
  }
```

VALIDATION AUTOMATISEE

POURQUOI UTILISER UNE BIBLIOTHEQUE DE VALIDATION ?

- **Centralise** toutes les règles dans un **schéma**
- Produit des **messages** d’**erreurs cohérents**
- **Gère** les **types**, les **formats**, les **longueurs**, les valeurs **autorisées**, etc.
- Peut **valider** des **objets imbriqués** (comme des objets user.address)
- Gère aussi la **transformation automatique** (ex : convertir "42" en number)
- Souvent compatible avec **TypeScript**

LES PRINCIPALES BIBLIOTHEQUES DE VALIDATIONS

Joi

La plus connue et la plus utlisée historiquement

- Syntaxe fluide, orientée objet
- Très complète
- Très utilisée côté back-end
- Transforme automatiquement les types (ex : string → number)

Avantages

- Grande communauté
- Bonne documentation
- Messages d'erreur personnalisables

```
1 const Joi = require("joi");
2
3 const userSchema = Joi.object({
4   username: Joi.string().alphanum().min(3).max(20).required(),
5   email: Joi.string().email().required(),
6   password: Joi.string().min(8).pattern(/[A-Z]/).required(),
7   confirmPassword: Joi.ref('password'),
8   age: Joi.number().integer().min(13).max(120).optional()
9 });
```

LES PRINCIPALES BIBLIOTHEQUES DE VALIDATIONS

Zod

Moderne, léger très utilisé en TypeScript

- Idéal si tu fais du TypeScript (type-safe + inférence)
- Syntaxe claire, orientée fonctionnelle
- Très rapide, sans dépendance externe

```zod
const { z } = require("zod");

const userSchema = z.object({
  username: z.string().min(3).max(20).regex(/^[a-zA-Z0-9_]+$/),
  email: z.string().email(),
  password: z.string().min(8),
  confirmPassword: z.string(),
  age: z.number().int().min(13).max(120).optional()
}).refine(data => data.password === data.confirmPassword, {
  message: "Les mots de passe ne correspondent pas",
  path: ["confirmPassword"]
});
```

Avantages

- Très moderne
- Parfait pour API + validation + types
- Compatible avec front et back (React, Next, Node...)

VERIFICATION DES SCHEMAS

Middleware de vérification

middlewares/validate.js

```
const validate = (schema) => (req, res, next) => {
    const { error } = schema.validate(req.body);
    if (error) {
        return res.status(400).json({ error:
error.details[0].message });
    }
    next();
};

module.exports = validate;

```

Utilisation dans la route

routes/userRoutes.js

```
const express = require('express');
const { registerUser } = require('../controllers/userController');
const validate = require('../middlewares/validate');
const { userSchema } = require('../schemas/userSchema');

const router = express.Router();

router.post('/register', validate(userSchema), registerUser);

module.exports = router;

```

LES PRINCIPALES BIBLIOTHEQUES DE VALIDATIONS

Express-validator

Middleware Express natif, pas de schéma externe

- Fonctionne sous forme de middlewares dans les routes
- Syntaxe un peu plus verbeuse
- Pas de schéma central, moins réutilisable

Avantages

- Facile à intégrer dans Express
- Pas besoin de schéma séparé
- Parfait pour débuter

```
1  const { body, validationResult } = require('express-validator');
2
3  app.post('/users', [
4    body('username').isLength({ min: 3, max: 20 }).matches(/^[a-
   zA-Z0-9_]+$/),
5    body('email').isEmail(),
6    body('password').isStrongPassword(),
7  ], (req, res) => {
8    const errors = validationResult(req);
9    if (!errors.isEmpty()) {
10     return res.status(400).json({ errors: errors.array() });
11   }
12   res.status(201).send("OK");
13 });
14
```

INCROPORATION DANS UN MIDDLEWARE

middlewares/validateUser.js

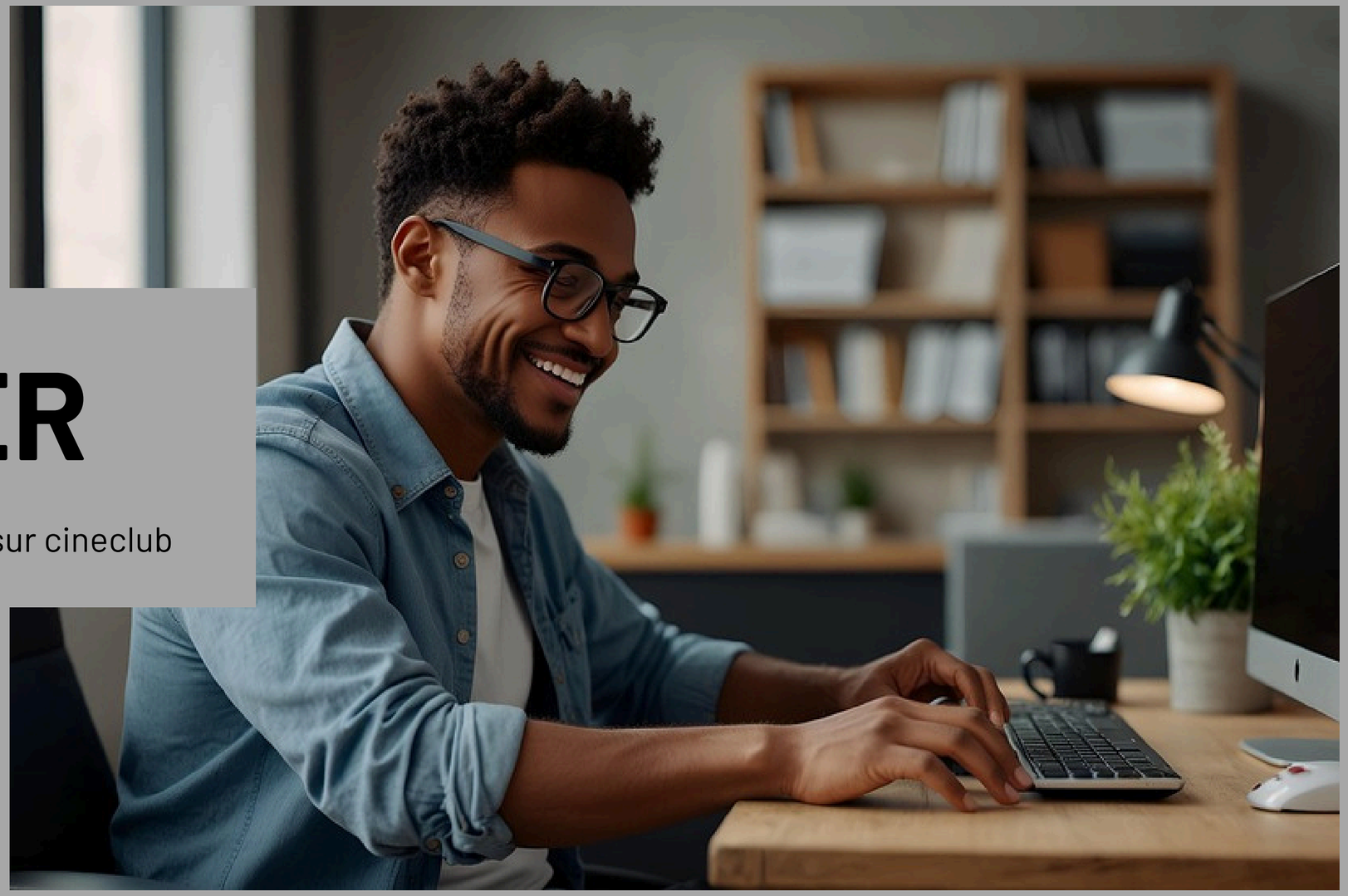
```
const { body, validationResult } = require('express-validator');

const userValidationRules = [
    body('username').isLength({ min: 3 }).withMessage('Username must be at least 3 characters long'),
    body('email').isEmail().withMessage('Must be a valid email'),
    body('password').isLength({ min: 6 }).withMessage('Password must be at least 6 characters'),
];

const validate = (req, res, next) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        return res.status(400).json({ errors: errors.array() });
    }
    next();
};

module.exports = { userValidationRules, validate };
```

routes/users.js

```
const express = require('express');
const { userValidationRules, validate } = require('./middlewares/validateUser');

const router = express.Router();

router.post('/register', userValidationRules, validate, (req, res) => {
    res.json({ message: 'User created successfully' });
});

module.exports = router;

```

ATELIER

Mettre en place les validations sur cineclub



CONSIGNES

Vous allez reprendre votre projet cineclub et mettre en place les validations nécessaires sur les requêtes POST

QUIZZ