## **1. Protecció contra HTML Injections i XSS**

### **Mecanismes de protecció**

#### **1.1. Escapar contingut de l'usuari**

Utilitza biblioteques per escapar el contingut introduït pels usuaris abans de mostrar-lo en una pàgina web. Això evita que es processi com a codi HTML o JavaScript.

* **Biblioteca recomanada:** [DOMPurify](https://github.com/cure53/DOMPurify) (per al costat del client) o [xss-filters](https://github.com/yahoo/xss-filters) (per al servidor).

Exemple amb xss:

const xss = require('xss');

// Sanititza entrada de l'usuari

const entradaNetegada = xss(req.body.nom);

#### **1.2. Utilitza helmet per a capçaleres de seguretat**

[Helmet](https://www.npmjs.com/package/helmet) és una biblioteca que afegeix capçaleres de seguretat per prevenir atacs comuns, incloent XSS.

Instal·la helmet:

npm install helmet

Exemple d’incorporació a la teva aplicació:

const helmet = require('helmet');

app.use(helmet());

Algunes capçaleres rellevants:

* **Content Security Policy (CSP):** Defineix quines fonts de contingut són segures.
* **X-XSS-Protection:** Activa el filtre anti-XSS en navegadors.

#### **1.3. Valida i neteja les dades d'entrada**

Assegura't que només acceptes dades que compleixin els teus criteris esperats.

* **Biblioteca recomanada:** express-validator.

Exemple:

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

app.post('/formulari', [

body('nom').isAlphanumeric().withMessage('El nom ha de ser alfanumèric.'),

body('email').isEmail().withMessage('L\'email no és vàlid.')

], (req, res) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(400).json({ errors: errors.array() });

}

res.send('Formulari enviat correctament!');

});

## **2. Protecció contra SQL Injections**

### **Què és?**

Un atacant manipula consultes SQL injectant codi no desitjat, sovint mitjançant formularis o paràmetres URL.

### **Mecanismes de protecció**

#### **2.1. Utilitza consultes preparades (Parameterized Queries)**

Aquest és el mètode més efectiu per prevenir SQL Injection. No concatena dades d'usuari directament a la consulta SQL, sinó que utilitza marcadors de posició.

Exemple amb **MySQL**:

const sql = 'SELECT \* FROM usuaris WHERE usuari = ? AND contrasenya = ?';

db.query(sql, [req.body.usuari, req.body.contrasenya], (err, result) => {

if (err) throw err;

res.send(result);

});

Amb **Sequelize** (ORM):

const usuari = await Usuari.findOne({

where: {

nom: req.body.usuari,

contrasenya: req.body.contrasenya

}

});

#### **2.2. Escapar dades manualment (si no pots utilitzar consultes preparades)**

Si has de construir consultes SQL manualment, assegura't d'escapar les dades de l'usuari.

Exemple amb MySQL:

const usuari = mysql.escape(req.body.usuari);

const sql = `SELECT \* FROM usuaris WHERE usuari = ${usuari}`;