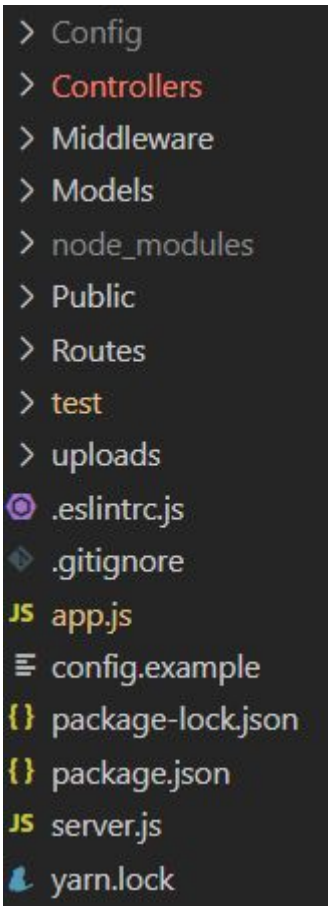
Architecture MVC, mais non, model et contrôleur



sécurité

Voici les failles les plus communes :

- Injection SQL

Commençons par écrire une route de base pour créer un utilisateur dans la base de données:

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

const app = express();



app.use(express.json());

app.post('/user', (req, res) => {

  User.create({

    username: req.body.username,
```

```
    password: req.body.password
  }).then(user => res.json(user));
});
```

Ensuite, vous voudrez vous assurer que vous validez l'entrée et signalez toutes les erreurs avant de créer l'utilisateur:

```
// ...rest of the initial code omitted for simplicity.
const { body, validationResult } = require('express-validator');

app.post('/user', [
  // username must be an email
  body('username').isEmail(),
  // password must be at least 5 chars long
  body('password').isLength({ min: 5 })
], (req, res) => {
  // Finds the validation errors in this request and wraps them in an object with handy functions
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  User.create({
    username: req.body.username,
    password: req.body.password
  }).then(user => res.json(user));
});
```

Désinfection

Parfois, la réception d'une entrée dans une requête HTTP ne consiste pas seulement à s'assurer que les données sont dans le bon format, mais aussi qu'elles sont exemptes de bruit .

validator.js fournit une poignée de désinfectants qui peuvent être utilisés pour prendre en charge les données qui entrent.

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

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


const app = express();

app.use(express.json());


app.post('/comment', [

  body('email')

    .isEmail()

    .normalizeEmail(),

  body('text')

    .not().isEmpty()

    .trim()

    .escape(),

  body('notifyOnReply').toBoolean()

], (req, res) => {

  // Handle the request somehow

});
```

- Attaque par force brute (ou Brute Force)

La première chose à faire est donc d'obliger ses utilisateurs à choisir un mot de passe suffisamment solide. Selon la recommandation de la CNIL, il faut :

- au moins 8 caractères
- au moins une lettre majuscule
- au moins une lettre minuscule
- au moins un chiffre
- au moins un caractère spécial

- Cross-Site Scripting (ou XSS)

react est proteger de base

```
 <p className="post__excerpt" dangerouslySetInnerHTML={{ __html:
DOMpurify.sanitize(excerpt) }} />
```

- Cross-Site Request Forgery (ou CSRF)

contrôle par token

```
// check, decode and add token to sent request
module.exports = (req, res, next) => {
  try {
    const token = req.headers.authorization.split(' ')[1];
    const decoded = jwt.verify(token, process.env.JWT_PASSWORD);
    req.dataToken = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ err });
```

```
    }

};
```