

Node.js

API HTTP

Pierre KRAEMER – kraemer@unistra.fr

Objectifs :

- connaître les bases du fonctionnement de Node.js
- développer des API HTTP qui exposent des données
- utiliser des bibliothèques pour :

- le traitement des requêtes HTTP

→ **express**

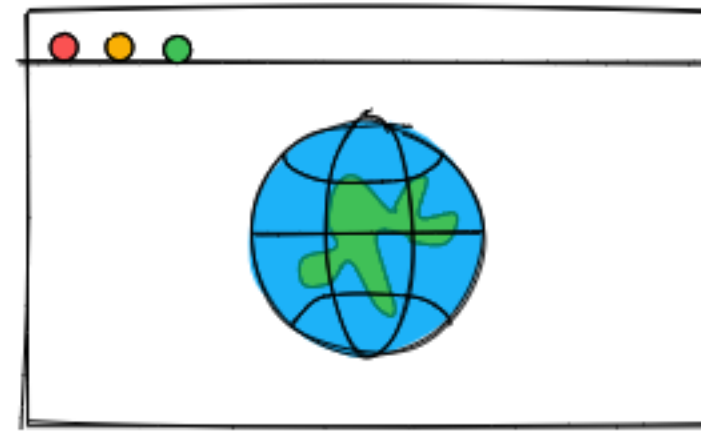
- le lien avec une base de données



sequelize



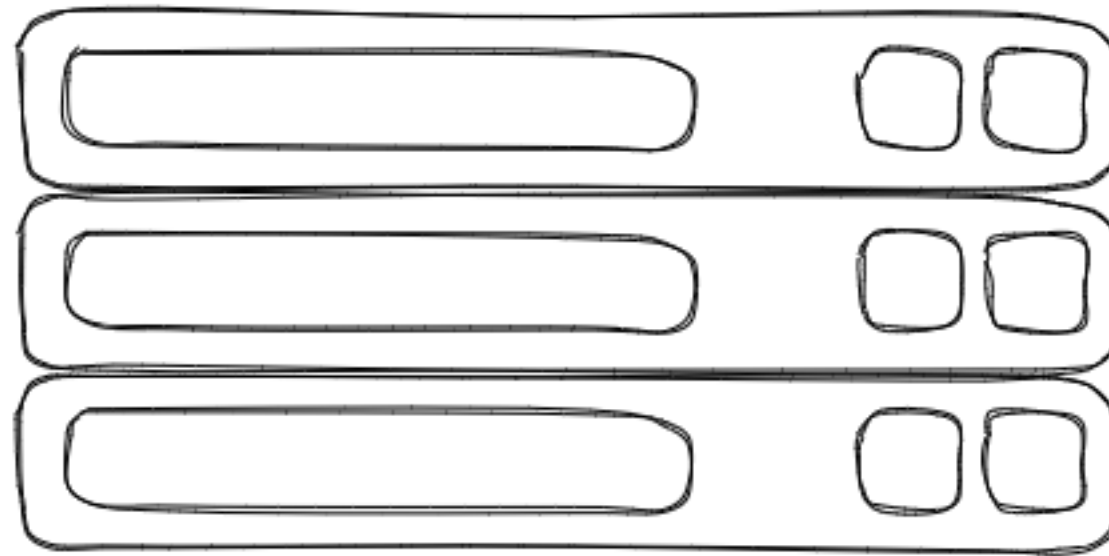
HTTP client



HTTP
request

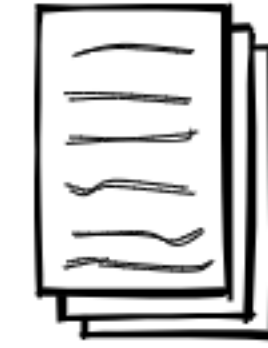
{GET, POST, DELETE, ...}
http://domain/path/to/ressource
[headers]
[some data]

HTTP server



```
[  
  { "name": "Pierre" },  
  { "name": "Adrien" },  
  { "name": "Joey" }  
]
```

HTTP
response



HTML document,
JS file,
image,
...

SQL
statement



DB server

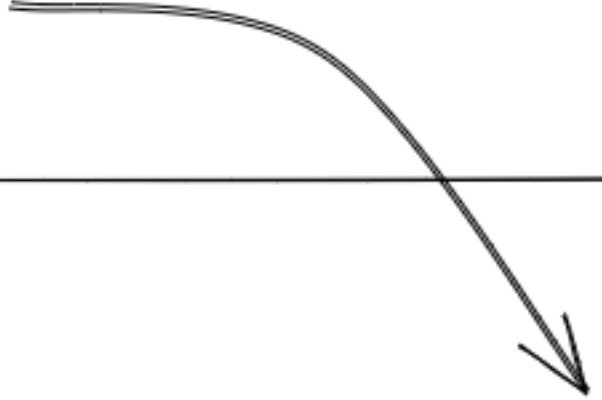
data



Modules Node.js

obj.js

```
const obj = {  
  text: 'youpi',  
  value: 42  
};  
  
const print_obj = () => { console.log(obj); };  
const update_value = v => { obj.value = v; };  
  
module.exports = {  
  print_obj,  
  update_value  
};
```



singleton

test.js

```
const m = require('./obj');  
  
print_obj();  
update_value(12);  
print_obj();
```

express

```
const express = require('express');  
const app = express();  
const port = 3000;  
  
app.get('/', (req, res) => {  
  res.send('Hello World!');  
});  
  
app.listen(port, () => {  
  console.log(`Example app listening on port ${port}`);  
});
```

enregistre un nouveau middleware qui n'est déclenché qu'aux conditions suivantes :

- > méthode HTTP GET
- > path est égal à '/'

lance le serveur HTTP
sur le port spécifié

enregistre ce middleware pour toutes les requêtes dont le chemin commence par '/'

indique à express qu'il est temps de regarder si d'autres middlewares correspondent à la requête en cours

on peut enrichir l'objet req et récupérer ses propriétés dans un middleware suivant

```
const express = require('express');
const app = express();
const port = 3000;

app.use('/', (req, res, next) => {
  console.log(`request ${req.method} from ${req.ip}`);
  next();
});

app.get('/hello', (req, res, next) => {
  req.name = req.query.name || 'unknown';
  next();
});

app.get('/hello', (req, res) => {
  res.send(`bonjour ${req.name}`);
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
});
```

effectue une réponse (texte) à destination du client

req

Error handling

```
app.get('/hello', (req, res, next) => {  
  if (!req.query.name)  
    return next({message: 'Please give a name'});  
  req.name = req.query.name;  
  next();  
});  
  
app.get('/hello', (req, res) => {  
  res.send(`bonjour ${req.name}`);  
});  
  
app.use((err, req, res, next) => {  
  return res.status(500).send(err.message);  
});  
  
app.listen(port, () => {  
  console.log(`Example app listening on port ${port}`);  
});
```

appeler next avec
un paramètre...

...déclenche l'appel du prochain
middleware de gestion d'erreur

sequelize



```
const { Sequelize, DataTypes, Model } = require('sequelize');
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: 'data.sqlite'
});

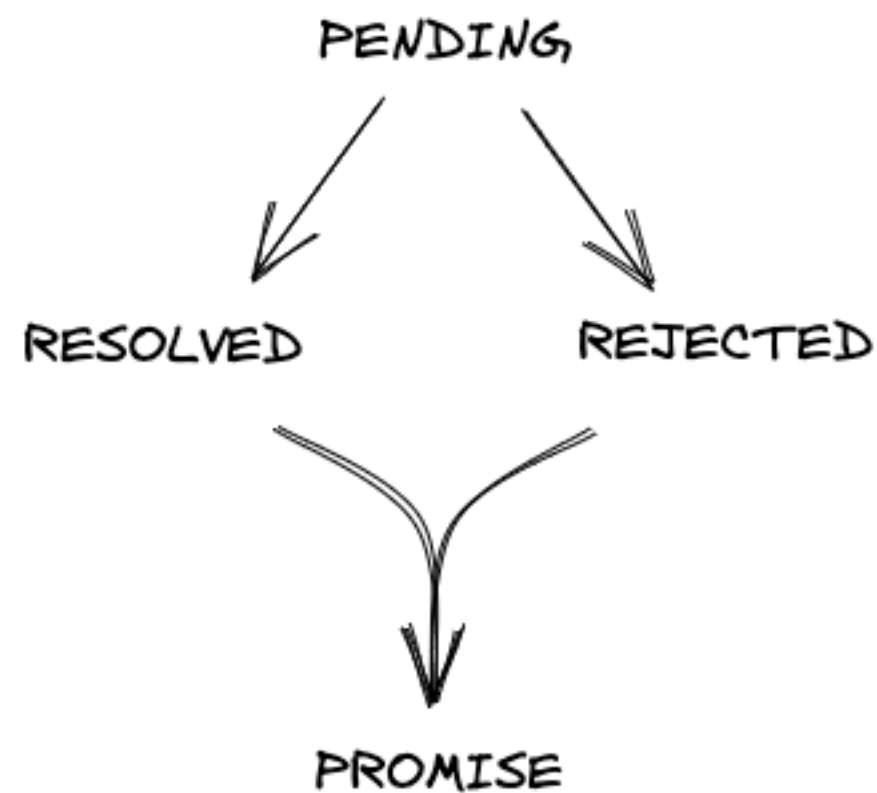
class User extends Model {}

User.init({
  firstName: {
    type: DataTypes.STRING,
    allowNull: false
  },
  lastName: {
    type: DataTypes.STRING
  }
}, {
  sequelize,
  modelName: 'User'
});

sequelize.sync();
```


Promises

obtenir tout de suite un objet qui représente une valeur disponible plus tard



configurer le code à exécuter :
- quand la valeur est disponible
OU
- quand la demande a échoué

enchaîner les demandes différées

sequelize



`findAll`
`findByPk`



Model querying

`create`
`save`
`destroy`



Model instances