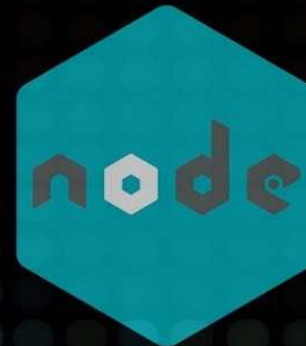


Node.js API Authentication

JSON Web Tokens



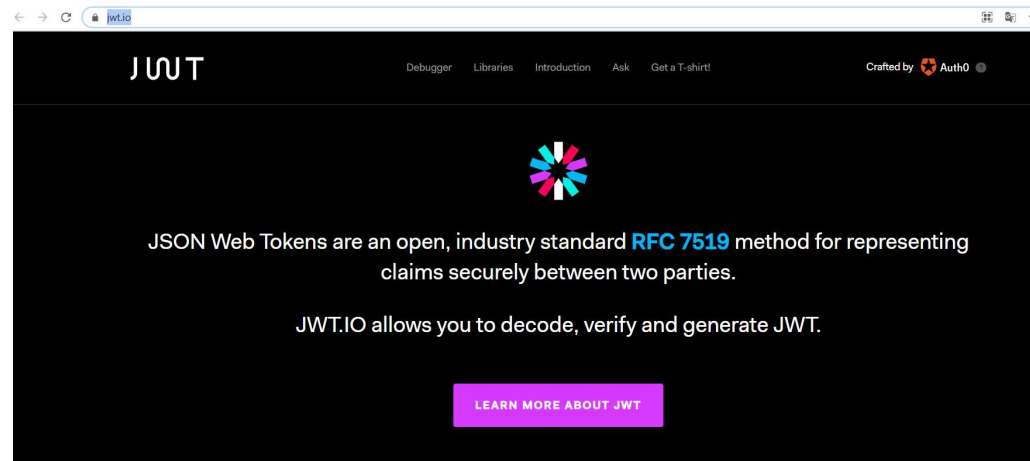
JWT



Utilisation de JWT

- Pour sécuriser les routes on utilise le JSON Web Token (JWT) qui est une méthode standard pour l'échange de données sécurisées entre deux parties.
- Le site officiel de JWT :

<https://jwt.io/>



Comment ça marche ?

- Le client envoie son login/mot de passe au serveur,
- Le serveur vérifie que les informations passées sont valides.
- Le serveur génère alors un token signé décomposé en 3 parties séparées par un point :
 - Le **header** ;
 - Le **payload** ;
 - La **signature**.

Le header

- Le **header** détermine le type de token : JWT et l'algorithme de chiffrement **HS256** (HMAC avec du SHA-256) ou **RS256** (signature RSA avec du SHA-256).

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

Le payload

- Le payload concerne la partie informations de l'utilisateur comme son id, son nom d'utilisateur, etc.

```
{  
  "sub": "1234567890",  
  "name": "Mohamed Tounsi",  
  "iat": 1516239022  
}
```

La signature

- La signature, résultant des 2 premières parties et d'une clé secrète,
- Le jeton est souvent signé à l'aide de toute méthode de signature sécurisée (par exemple, un algorithme à clé asymétrique tel que HMAC SHA-256 ou un système à clé publique asymétrique, tel que comme RSA).

Le token 1/2

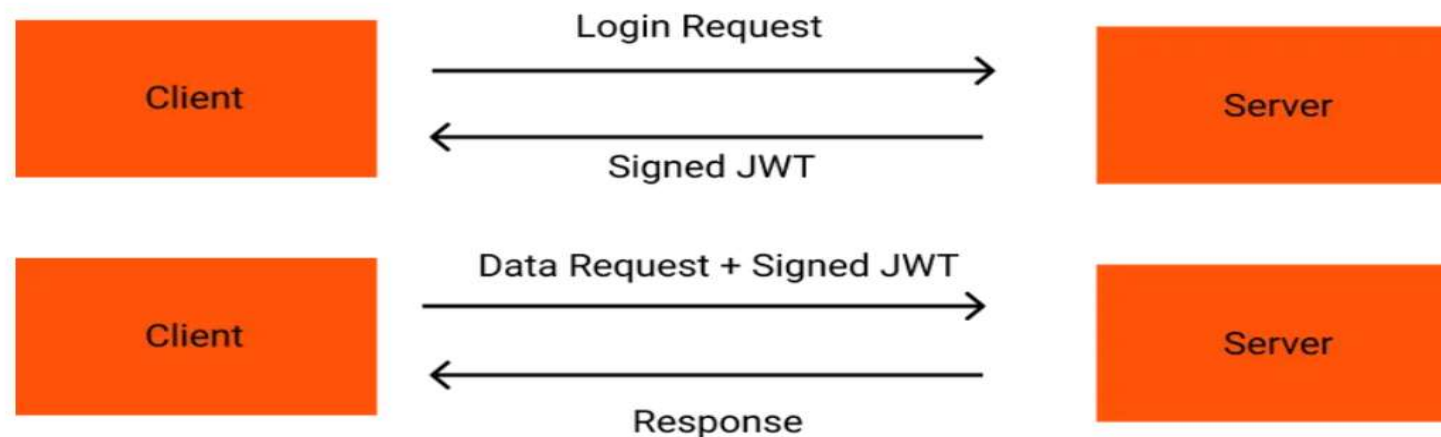
- Les 2 premières parties sont encodées en base64. On obtient alors un token sous la forme suivante (3 parties séparées par un point) :
- eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
- Dans notre API, l'utilisateur devra posséder un token pour accéder aux routes protégées. Pour se connecter, il devra saisir ses identifiants (login et mot de passe) sur une route de type POST. Cette dernière retournera un jeton, le fameux JWT. Celui-ci devra être renseigné dans le header de la requête pour accéder aux routes protégées.

Le token 2/2

- Une fois ce token généré, le serveur l'ajoute dans l'entête de la réponse.
- Le client récupère ensuite ce token et le stocke de son côté en cookie par exemple. À la requête suivante le client n'a plus qu'à reprendre ce token et l'inclure dans l'entête de cette dernière.
- Côté serveur on vérifie la validité du token, s'il est valide on donne l'accès à la ressource demandée et dans le cas contraire on lui renvoie une erreur lui expliquant qu'il n'a pas les droits pour y accéder. Voilà pour la théorie.

Comment fonctionne JWT?

- JWT fonctionne comme un protocole bidirectionnel dans lequel une demande est effectuée et la réponse est générée à partir d'un serveur.
- avec JWT, lorsque le client envoie une demande d'authentification au serveur, il renverra un jeton JSON signé au client, qui comprend toutes les informations sur l'utilisateur avec la réponse.
- Le client enverra ce jeton avec toutes les demandes suivantes. Ainsi, le serveur n'aura pas à stocker d'informations sur la session.



Installation de JWT

`npm install jsonwebtoken --save`

- La bibliothèque Jsonwebtoken est une implémentation de la RFC 7519 qui fournit diverses méthodes pour générer des JWT, vérifier les JWT, etc.
- Après avoir installé ces modules, le dossier node_modules sera créé.

bcrypt


- bcrypt est une fonction de hachage de mot de passe basée sur le chiffrement.
- Le hachage effectue une transformation unidirectionnelle sur un mot de passe, transformant le mot de passe en une autre chaîne, appelée mot de passe haché.
- Le hachage est appelé à sens unique car il est pratiquement impossible d'obtenir le texte original à partir d'un hachage.
- De plus il permet d'incorporer un « salt » pour se protéger contre les attaques de « rainbow table ».
- Une rainbow table est une structure de données pour retrouver un mot de passe à partir de son empreinte.
- L'ajout d'un sel (salt) consiste à concaténer une chaîne aléatoire au mot de passe avant de l'envoyer dans la fonction de hachage. Le sel est stocké avec le mot de passe dans la base de données.

La chaîne de hachage

Une chaîne de hachage bcrypt est de la forme :
\$2b\$[cost]\$[22 character salt][31 character hash]

Par exemple:

\$2a\$10\$N9qo8uLOickgx2ZMRZoMyeljZAgcfl7p92ldGxad68LJZdL17lhWy


Alg Cost Salt Hash

Avec :

\$2a\$: L'identifiant de l'algorithme de hachage (bcrypt)

10: Facteur de coût (2¹⁰ ==> 1,024 rounds)

N9qo8uLOickgx2ZMRZoMye: 16-byte (128-bit) salt, base64-encoded à 22 caractères

ljZAgcfl7p92ldGxad68LJZdL17lhWy: 24-byte (192-bit) hash, base64-encoded à 31 caractères

Installation de bcrypt

```
npm install bcrypt --save
```

Etude de cas



A file explorer view showing a project structure. The root directory contains several folders and files. The 'middlewares' folder is expanded, showing 'auth.js'. The 'models' folder is also expanded, showing 'User.js'. The 'node_modules' folder is collapsed. The 'public' folder is collapsed. The 'routes' folder is expanded, showing 'authentification.is'. The root directory also contains a '.env' file and an 'app.js' file.

- ▼ middlewares
 - JS auth.js
- ▼ models
 - JS User.js
- > node_modules
- > public
- ▼ routes
 - JS authentification.is
- ⚙ .env
- JS app.js

Création du projet et des dépendances

```
npm init -y
```

```
npm i express jsonwebtoken cors mongoose dotenv bcrypt body-parser --save
```

app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const cors=require('cors');
require("dotenv").config();
const app = express();
var loginRoutes=require("./routes/authentication")
app.use('/api/auth', loginRoutes);

app.use(bodyParser.urlencoded({ extended: true }))
app.use(bodyParser.json())
app.use(cors());
```



```
mongoose.connect(process.env.DATABASE,{
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useCreateIndex: true
}) .then(() => {console.log("MongoDB connected to the database");
}).catch(err => {
console.log('Could not connect to the database. Exiting now...', err);
process.exit();
});
```

```
// Ecoute des requêtes
app.listen(3003, () => {
  console.log("Server is listening on port 3003"); });
```

.env

ENV=DEVELOPMENT

DATABASE=mongodb://localhost:27017/baseExemple

SECRET=accesssecret

models/User.js

```
const mongoose = require("mongoose");
var userSchema = mongoose.Schema({
  name:{
    type:String,
    required:"nom is required"
  } ,
  email:{
    type:String,
    required:"Email is required",
    unique:true
  } ,
  password:{
    type:String,
    required:"password is required"
  }
});
module.exports = mongoose.model('user', userSchema)
```

middlewares/auth.js

```
const jwt =require('jsonwebtoken');
```

```
module.exports=async(req,res,next)=>{
```

```
    const token = req.header('x-auth-token');
```

```
    if(!token)
```

```
    return res.status(401).json({msg:'pas de token, opération non autorisée'})
```

```
    try{
```

```
        var decoded=jwt.verify(token,process.env.SECRET);
```

```
        req.user=decoded;
```

```
        next();
```

```
    }catch(err){
```

```
        res.status(401).json({msg:"token non valide"})
```

```
    }
```

```
}
```

routes/authentication.js

```
var express = require('express');
var router = express.Router();
var User = require ('../models/User');
const bcrypt= require ("bcrypt");
const auth =require ('../middlewares/auth');
const jwt =require('jsonwebtoken');
//http://emailregex.com/ javascript
const EMAIL_REGEX=/^((([^\<>()\[\]\\\.,;:\s@""]+)(\.[^\<>()\[\]\\\.,;:\s@""]+)*|("[.+")@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|((([a-zA-Z\-\0-9]+\.[a-zA-Z]{2,}))$)/
//password Contenir au moins 8 caractères,numéro,1 caractère majuscule,seulement 0-9a-zA-Z
const PASSWRD_REGEX= /^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])[0-9a-zA-Z]{3,}$/
```

```
/**
 * @route Get api/auth/login
 * @desc Login user
 */
router.post("/login", async (req, res) => {
  const {email, password} = req.body;
  //simple validation
  if (!email || !password) {
    return res.status(400).json({msg: 'Email or password non saisies'})
  }
  if (!EMAIL_REGEX.test(email)) {
    return res.status(400).json({'erreur': 'Email non valide'});
  }
  try {
    //vérifier l'existence de l'utilisateur
    const user = await User.findOne({email});
    if (!user) return res.status(400).json({'erreur': 'utilisateur non existant'});
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({msg: 'mot de passe incorrect'})
    const token = jwt.sign({ id: user._id }, process.env.SECRET, { expiresIn: 3600 });
  }
});
```

```
if (!token) throw Error('Couldnt sign the token');
res.status(200).json({
  token,
  user: {
    id: user._id,
    name: user.name,
    email: user.email
  }
});
} catch (e) {
  res.status(400).json({ msg: e.message });
}

});
```

```
/**
 * @route   POST api/auth/register
 * @desc    Register new user
 * @access  Public
 */
router.post("/register", async(req, res)=>{
  const{name,email,password}=req.body;
  //simple validation
  if(!email||!password){
    return res.status(400).json({msg:'Email or password non saisies'})
  }
  if(!EMAIL_REGEX.test(email)){
    return res.status(400).json({'erreur':'Email non valide'});
  }
  if(!PASSWRD_REGEX.test(password)){
    return res.status(400).json({'erreur':'password non valide'});
  }
  try{
    const user=await User.findOne({email});
    if(user)return res.status(400).json({'erreur':'utilisateur existe déjà'})
  }
```

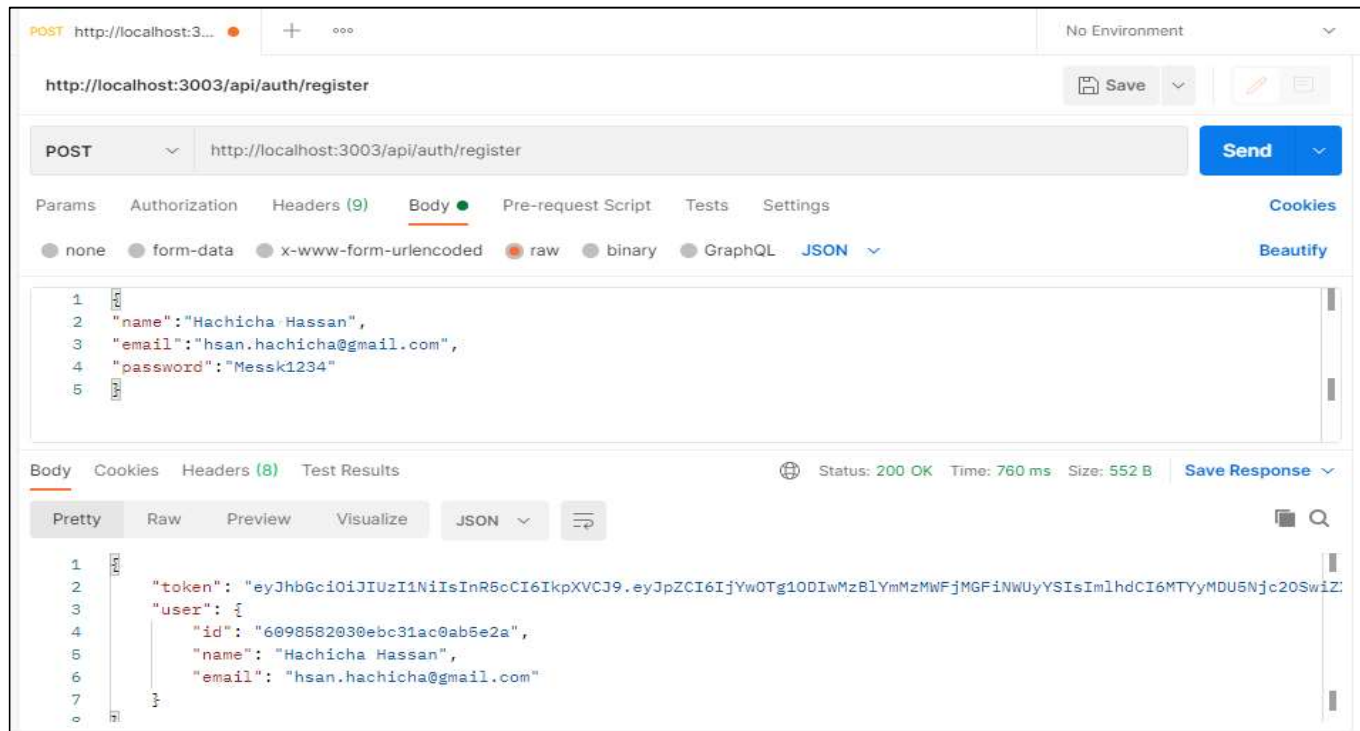


```
const salt=await bcrypt.genSalt(10);
const hash=await bcrypt.hash(password,salt);
const newuser=new User({
  name:name,
  email:email,
  password:hash,
});
const savedUser=await newuser.save();
if(!savedUser) return res.status('user ne peut pas etre enregistré')
const token = jwt.sign({ id: savedUser._id }, process.env.SECRET, {
  expiresIn: 3600
});
res.status(200).json({
  token,
  user: {
    id: savedUser.id,
    name: savedUser.name,
    email: savedUser.email
  }
})
```

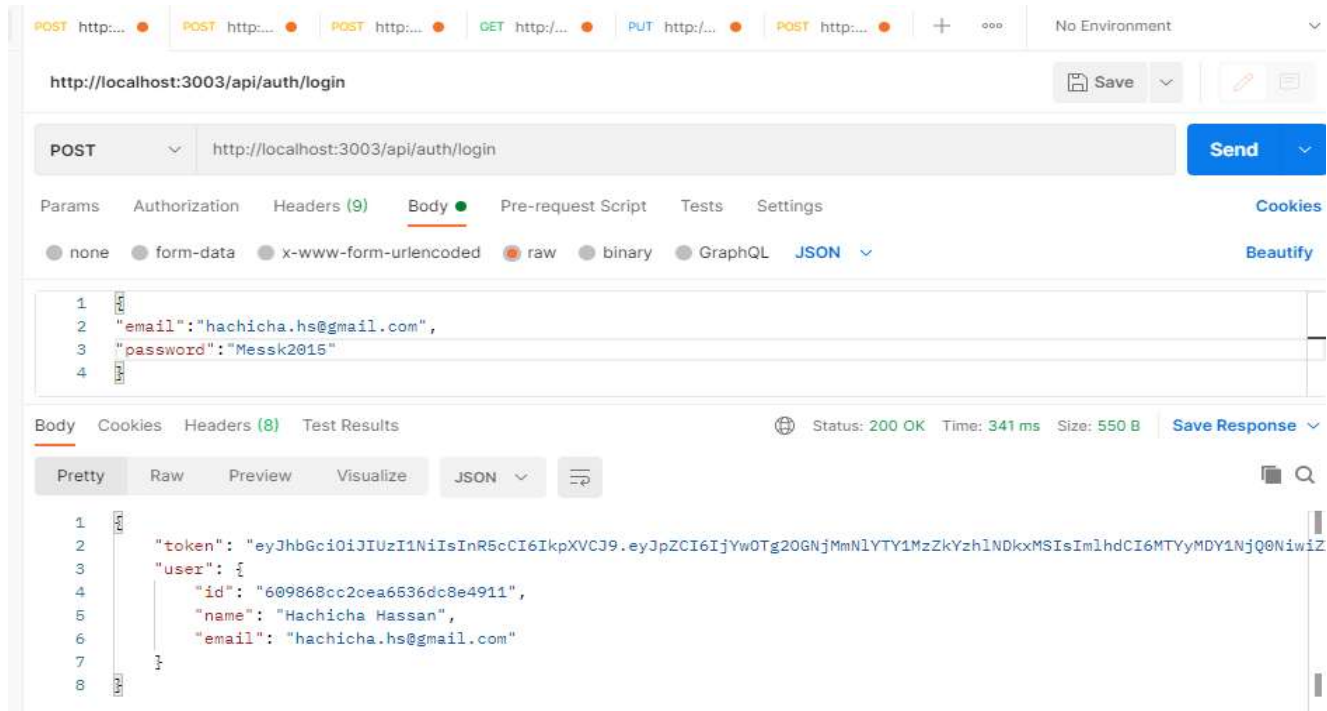
```
});  
    }catch(err){  
        res.json({  
            message:"Erreur de connexion à la base de données"+err,  
        })  
    }  
}  
);  
  
module.exports = router;
```

S'enregistrer

<http://localhost:3003/api/auth/register>



<http://localhost:3003/api/auth/login>



Détenir le token

- Copier la valeur initialement générée par le token et la copier dans Headers au niveau de « Authorization ».

Params	Authorization	Headers (12)	Body ●	Pre-request Script	Tests	Settings
<input checked="" type="checkbox"/>	Content-Type					application/x-www-form-urlencoded
<input checked="" type="checkbox"/>	Authorization					bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYwOTN...

Sécuriser les pages avec le token

- Désormais chaque route est sécurisé avec le token en faisant appel à « auth ».

Par exemple :

```
var auth = require('../middlewares/auth');
```

```
.....
```

```
router.post('/', auth, async (req, res) => {
```

```
.....
```