

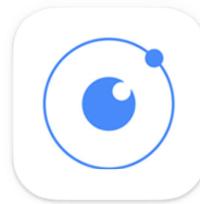
# Formation Mobile Hybride

## Ionic Cordova

# Correction des exercices

`git checkout step1`

## Ionic View



<https://docs.ionic.io/tools/view/>

Permet de visualiser rapidement le résultat sur son téléphone, sans devoir builder son app pour le téléphone

# Ionic View

[Installer et s'enregistrer](#)



[S'authentifier au niveau du projet](#)

ionic login

[Créer une nouvelle app](#)

ionic link → <https://apps.ionic.io/apps/>

[Vérifier l'APP ID](#)

/ionic.config.json → "app\_id"

[Envoyer le code vers Ionic View](#)

ionic upload

# Authentification

login

Email

Password

**LOGIN**

**SIGN UP**

signup x

Email

Password

**SIGN UP**

# Modals

Transition de la page Login => Signin

ModalController

Gestion des erreurs

ToastController

Spinner qui s'affiche à l'authentification

LoadingController

# Authentification avec JWT

## Web Token

Pas de cookie ni de session serveur.

Applicable pour tout type de plateforme et tous devices

## JSON Web Token

JSON Web Tokens are an open, industry standard [RFC 7519](#) method for representing claims securely between two parties.



Header encodé en base64

Contenu encodé en base64

Encodage du header + contenu

Le header contient un objet javascript indiquant l'algorithme utilisé pour "hasher" le contenu.

*Exemple : { « typ »: « JWT », « alg »: « HS256 » }*

⇒ **eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9** (en base64)

Le contenu est un simple objet avec les informations à échanger entre le client et le serveur

*Exemple : { « iss »: « ekino.com », « name »: « John Doe », « admin »: true }*

⇒ **eyJiaXNzIjogImVraW5vLmNvbSIsICJuYW1lIjogIkpvG4gRG9IiwgImFkbWluljogdHJ1ZSB9** (en base64)

La signature est une concaténation du header et du contenu puis encodé avec l'algorithme défini dans le header.

HMACSHA256(

**eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9 + « . »**

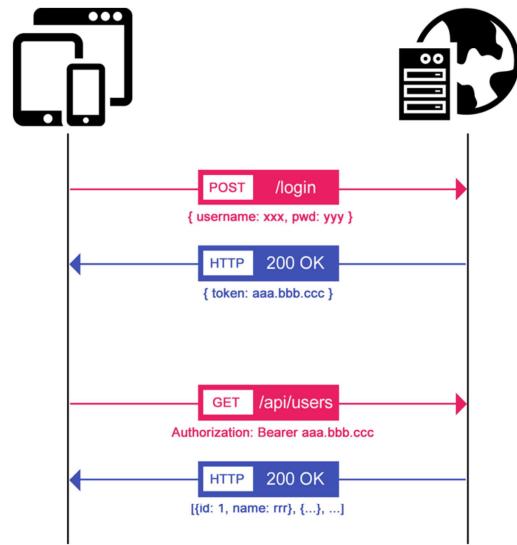
+ **eyJiaXNzIjogImVraW5vLmNvbSIsICJuYW1lIjogIkpvG4gRG9IiwgImFkbWluljogdHJ1ZSB9,**

« secret key »

)

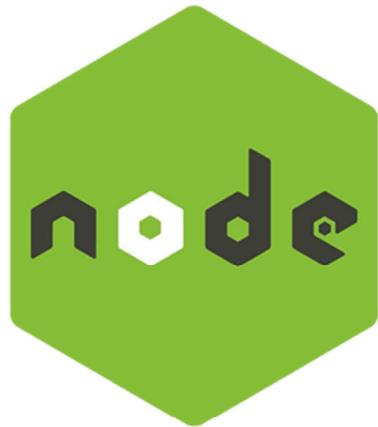
En savoir plus: <https://www.ekino.com/introduction-aux-json-web-tokens/>

## Authentification avec JWT



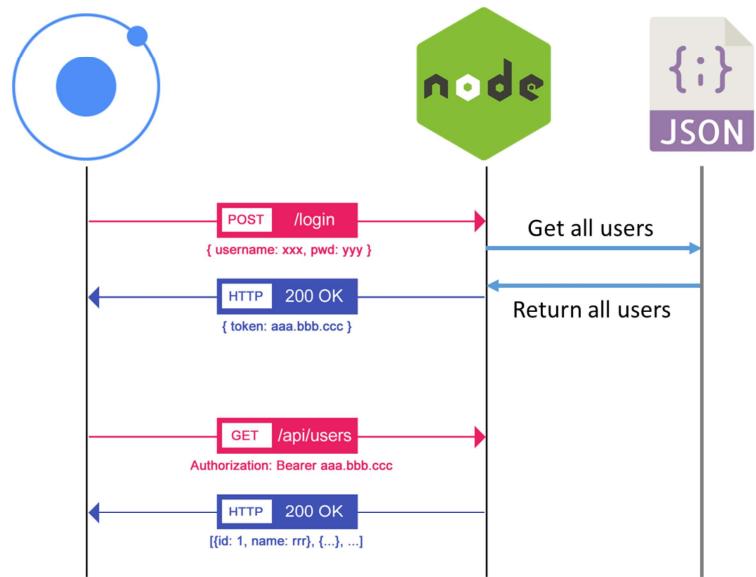
## Authentification avec JWT

Rapide survol de NodeJS



**Node.js** est une plateforme logicielle libre et événementielle en JavaScript orientée vers les applications réseau qui doivent pouvoir monter en charge. Elle utilise la machine virtuelle V8 et implémente sous licence MIT les spécifications CommonJS.

## Authentification avec JWT



## Authentification avec JWT

```
git checkout step2
```

## Authentification avec JWT

### [Installer JWT](#)

```
npm install angular2-jwt --save
```

### [Installer les paquets pour le serveur node \(en local\)](#)

```
npm install nodemon -g
```

```
npm install body-parser express fs jsonwebtoken --save
```

### [Lancer le serveur en local sur le port 4300](#)

```
nodemon
```

## Authentification avec JWT

[Tester son serveur avec Postman](#)



# Authentification avec JWT

## Les routes et la sécurité (authguard)



olegwn commented on 4 May

@Barryrowe right to the point. It's our pain to implement route guards with Ionic 3 :( Works fine for small apps, but for production ready - nope



3

## Authentification avec JWT

### app.module.ts

```
import { JwtHelper, AuthConfig, AuthHttp } from "angular2-jwt";
import { Http, HttpClientModule, RequestOptions } from "@angular/http";
import { Storage, IonicStorageModule} from "@ionic/storage";

// Auth Factory
export function authHttpServiceFactory(http: Http, options: RequestOptions, storage: Storage) {
  const authConfig = new AuthConfig({
    tokenGetter: () => storage.get('jwt'),
  });
  return new AuthHttp(authConfig, http, options);
}

imports: [
  HttpClientModule,
  IonicStorageModule.forRoot({
    name: 'myapp',
    driverOrder: ['sqlite', 'indexeddb', 'websql']
  }),
  providers: [
    JwtHelper,
    {
      provide: AuthHttp,
      useFactory: authHttpServiceFactory,
      deps: [Http, RequestOptions, Storage]
    }
]
```

# REVISION

# Exercices

## Consignes

Créer 2 providers « endpoints » et « auth ». Le 1<sup>er</sup> contiendra les 3 différentes urls de connexion au serveur (login, signup, getauth) qui seront appelées par auth afin d'établir la stratégie d'authentification.

Le provider « auth » sera utilisé par loginPage pour s'authentifier, signupPage pour créer l'utilisateur, app.component pour vérifier si l'utilisateur est connecté et tabsPage pour afficher l'e-mail de l'utilisateur courant dans le menu et la gestion du logout.

**git checkout step3**

### Référence

<https://golb.hplar.ch/p/JWT-Authentication-with-Ionic-2-and-Spring-Boot>

Merci de votre attention