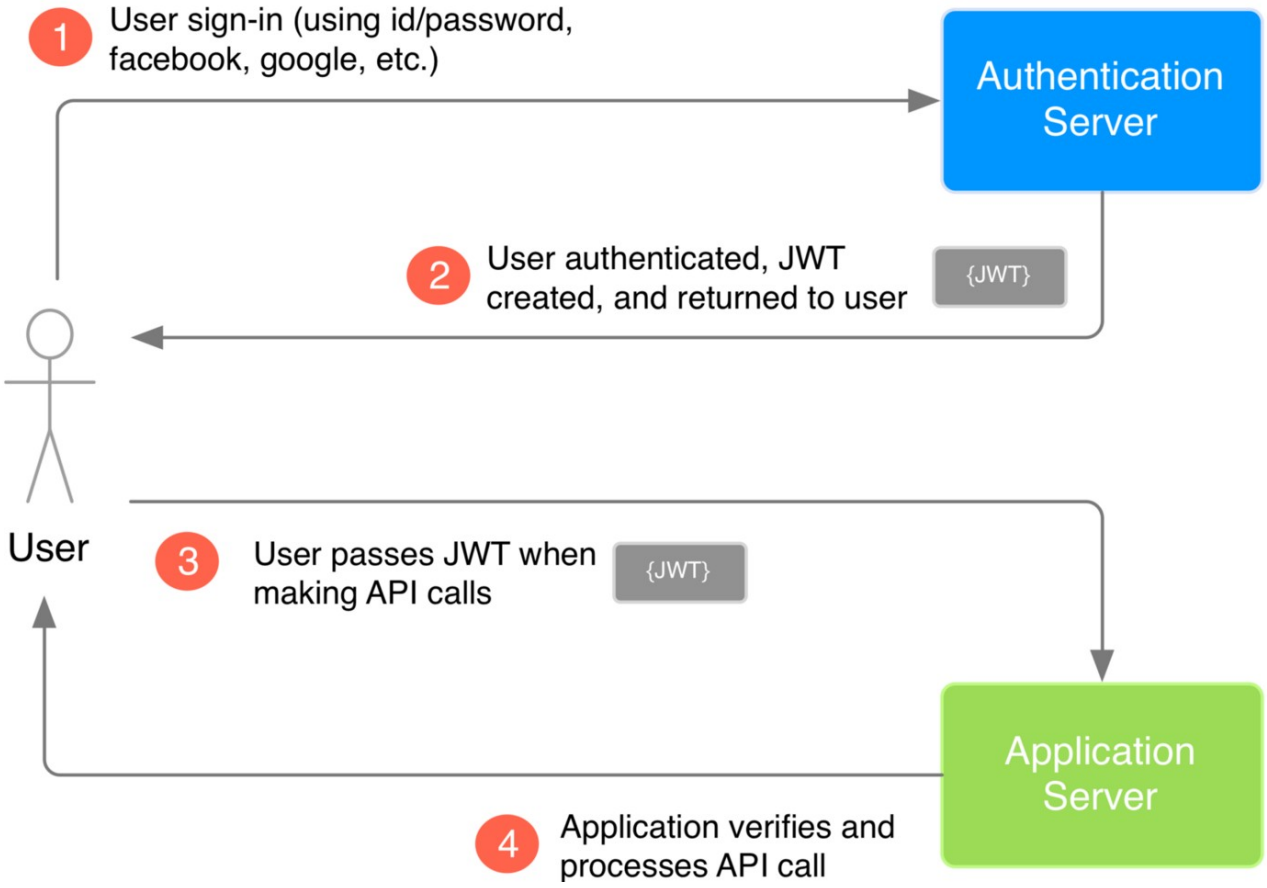
TP: Authentification et autorisation basées sur des jetons

Objectifs

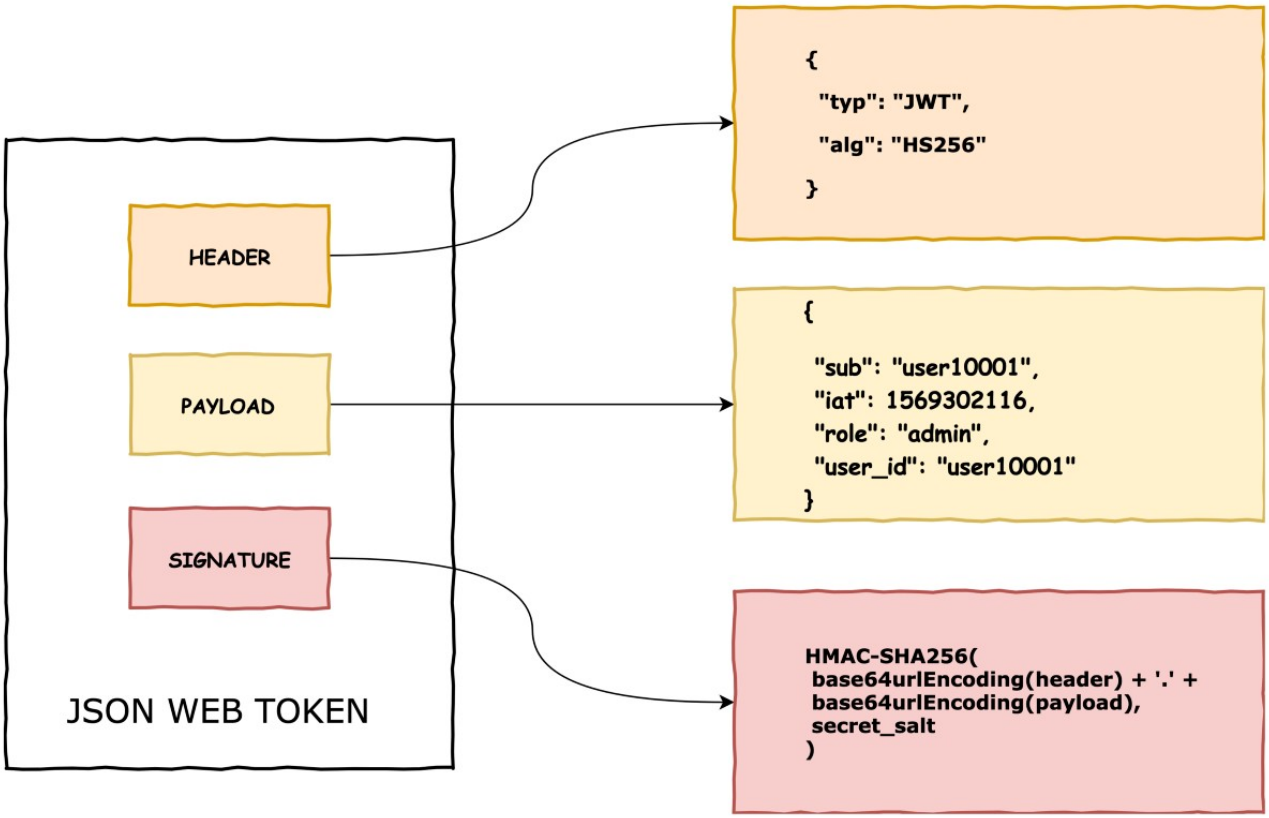
Dans ce TP, nous allons créer un exemple d'API Node.js Express REST qui prend en charge l'authentification basée sur des jetons avec JWT (JSONWebToken).

Authentification basée sur des jetons

Par rapport à l'authentification basée sur la session qui doit stocker la session sur le cookie, le grand avantage de l'authentification basée sur le jeton est que nous stockons le JSON WEB TOKEN (JWT) côté client: stockage local pour le navigateur, et préférences partagées pour Android…



Il y a trois parties importantes d'un JWT: **Header**, **Payload**, **Signature**. Ensemble, ils sont combinés en une structure standard: **header.payload.signature**.



Le client attache généralement JWT dans l'en-tête d'autorisation (authorization) avec le préfixe (Bearer) :

```
Authorization: Bearer [header].[payload].[signature]
```

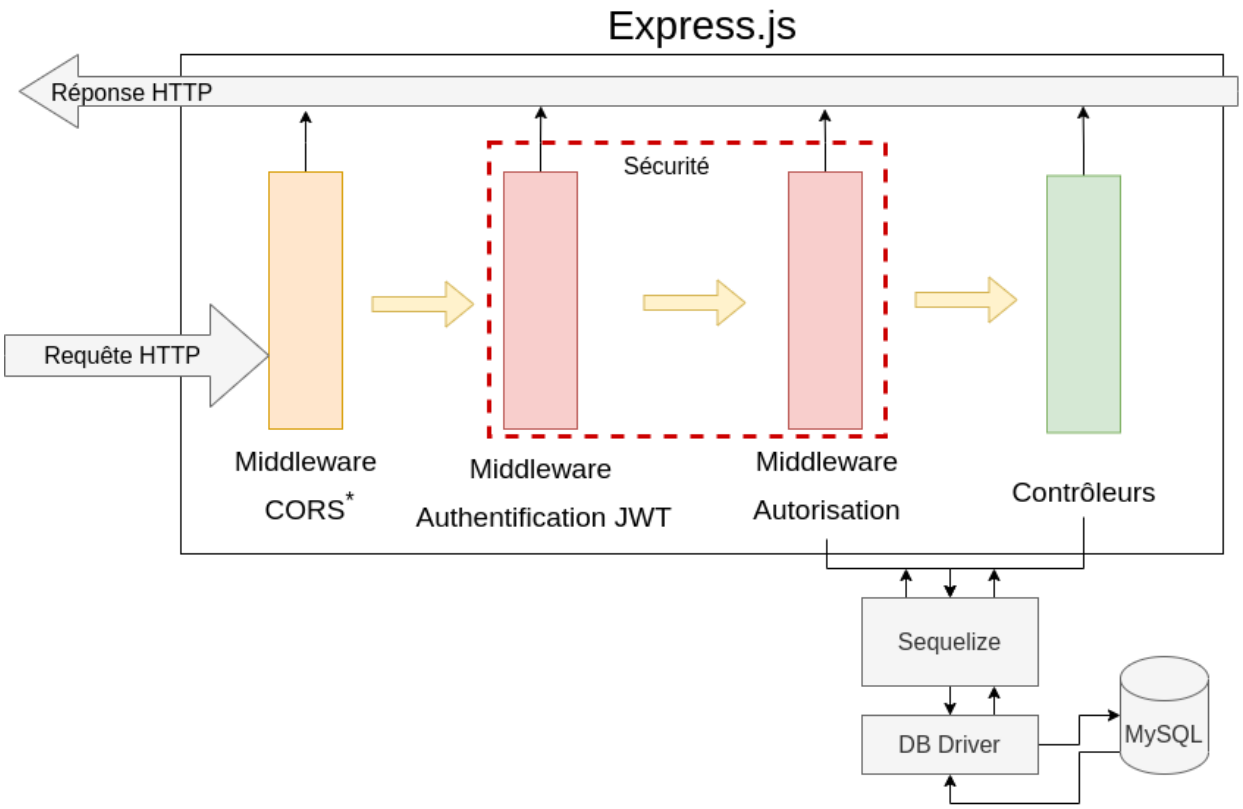
Ou uniquement dans l'en-tête **x-access-token**:

```
x-access-token: [header].[payload].[signature]
```

Voici les API que nous devons fournir:

Méthodes	URLs	Actions
POST	/auth/signup	créer un nouveau compte utilisateur
POST	/auth/signin	connecter un compte utilisateur
GET	/test/all	récupérer du contenu public
GET	/test/user	accéder au contenu de l'utilisateur enregistré

Vous pouvez avoir un aperçu de notre application Node.js Express JWT avec le schéma ci-dessous:



*Cross-origin resource sharing (CORS) est un mécanisme qui permet d'accéder à des ressources restreintes sur une page Web à partir d'un autre domaine en dehors du domaine à partir duquel la première ressource a été servie

Via les routes express, la requête HTTP qui correspond à une route sera vérifiée par le middleware CORS avant d'arriver à la couche de sécurité.

La couche de sécurité comprend:

- Middleware d'authentification JWT: vérifiez l'inscription, vérifiez le jeton
- Middleware d'autorisation: vérifiez les rôles de l'utilisateur avec un enregistrement dans la base de données (**Cette partie sera ignorée dans ce TP. Cependant, cela doit être pris en compte lors de la création d'applications où les utilisateurs sont divisés en catégories (utilisateurs normaux, administrateurs, etc.)**)

Si ces middlewares génèrent une erreur, un message sera envoyé en réponse HTTP.

Les contrôleurs interagissent avec la base de données MySQL via Sequelize et envoient une réponse HTTP (token, informations utilisateur…) au client.

Créer une application Node.js

Créez un dossier pour le projet, puis initialisez l'application Node.js avec un fichier **package.json:**

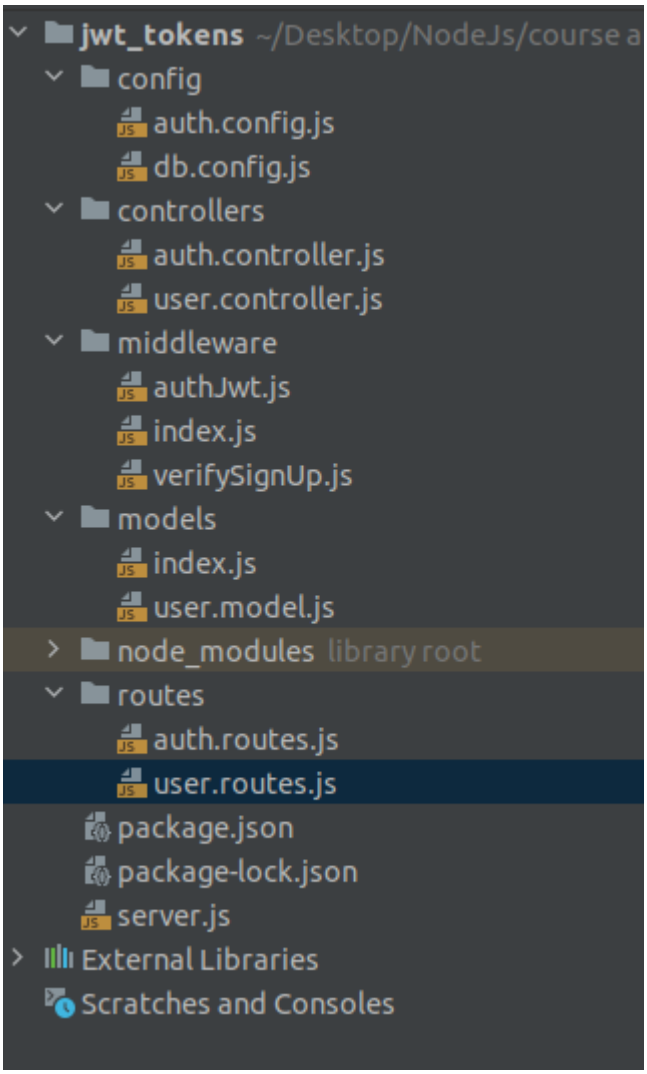
```
$ npm init –yes
```

Nous devons installer les modules nécessaires : **express**, **cors**, **body-parser**, **sequelize**, **mysql2**, **jsonwebtoken** et **bcryptjs**.

Exécutez la commande:

$ npm install express sequelize mysql2 body-parser cors jsonwebtoken bcryptjs –save

Nous devons ensuite créer la structure des fichiers illustrée dans la figure ci-dessous:



$ mkdir config

$ mkdir controllers

$ mkdir middleware

$ mkdir models

$ mkdir routes

$ touch config/auth.config.js

$ touch config/db.config.js

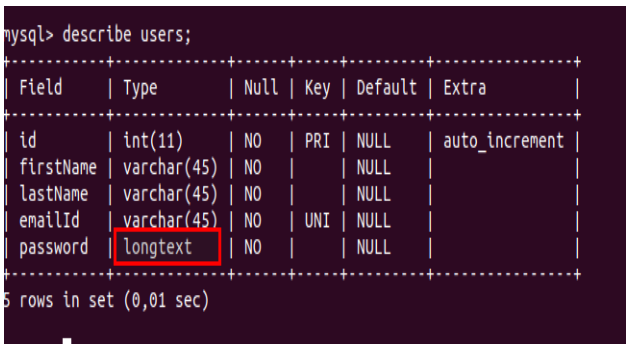
```
$ touch controllers/auth.controller.js
$ touch controllers/user.controller.js
$ touch middleware/authJwt.js
$ touch middleware/index.js
$ touch middleware/verifySignUp.js
$ touch models/index.js
$ touch models/user.model.js
$ touch routes/auth.routes.js
$ touch routes/user.routes.js
$ touch server.js
```

- **config**
  - *db.config.js:* configurer la base de données MySQL et sequelize
  - *auth.config.js*: configurer la clé d'authentification
- **routes**
  - *auth.routes.js:* inscription et connexion (login) POST
  - *user.routes.js:* OBTENIR des ressources publiques et protégées GET
- **middlewares**
  - *verifySignUp.js:* vérifier si l'email existe
  - authJwt.js: vérifier le jeton (token)
- **controllers**
  - *auth.controller.js:* gérer les actions d'inscription et de connexion
  - *user.controller.js:* renvoie le contenu public et protégé
- **models**
  - *user.model.js:* modèle Sequelize

Base de données

Nous allons utiliser la base de données utilisée dans TP4: **bdd_node_1**.

IMPORTANT: assurez-vous que le mot de passe de la colonne est "longtext" afin de stocker le texte chiffré. Sinon, téléchargez **db.sql** de la semaine 8 (cours-info) et remplacez l'ancienne base de données par celle du fichier **db.sql** (en refaisant "**source db.sql**").



```
mysql> describe users;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| id        | int(11)     | NO   | PRI | NULL    | auto_increment |
| firstName | varchar(45) | NO   |     | NULL    |                |
| lastName  | varchar(45) | NO   |     | NULL    |                |
| emailId   | varchar(45) | NO   | UNI | NULL    |                |
| password  | longtext    | NO   |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
5 rows in set (0,01 sec)
```

Semaine 8 (Partiel devrait être envoyé cette semaine):

La date limite pour présenter le Partiel est le 12 novembre 18H

- TD- Authentification des comptes utilisateurs -partie 2-
- db.sql (à télécharger avant de commencer avec le TD) ⇒ ***bdd_node_1***
- TP 8: Ajout de l'authentification des utilisateurs au projet du Partiel **(IMPORTANT : CE TP EST NOTÉ)**
- Ressources:
  - http://www.passportjs.org/packages/passport-local/ (Passport local)
  - http://www.passportjs.org/packages/passport-google-oauth2/ (Passport Google Oauth2)
  - http://www.passportjs.org/docs/facebook/ (Passport Facebook)
  - https://www.youtube.com/watch?v=KIE9RAOI9KA (HOW TO: Implement Facebook Login OAuth in Web App)
  - https://www.youtube.com/watch?v=Q0a0594tOrc (NodeJS & Express - Google OAuth2 using PassportJS)

Sujets abordés:

- Authentification des utilisateurs à l'aide de Passport