CORS - JWT

CORS

CORS
Partie 1

CORS

- CORS signifie "Cross-Origin Resource Sharing"
- Il est question de requêtes "multi-orgine" ou "cross-origin" en anglais

- Exemple :
  - Soit la page `domaine-a.com/index.html`
  - Toutes les requêtes ciblant le même domaine (same-origin) sont autorisées (chargement d'image, de CSS, de JS, requêtes Ajax)
  - Toutes les requêtes ciblant un autre domaine (cross-origin) sont contrôlées par des CORS pour des raisons de sécurité.
  - Par défaut, dans un navigateur, une requête Ajax ciblant un autre domaine (`domaine-b.com`) sera rejetée, sauf si ce dernier l'y autorise

- Le mécanisme des CORS utilise les entêtes HTTP.

EXEMPLE

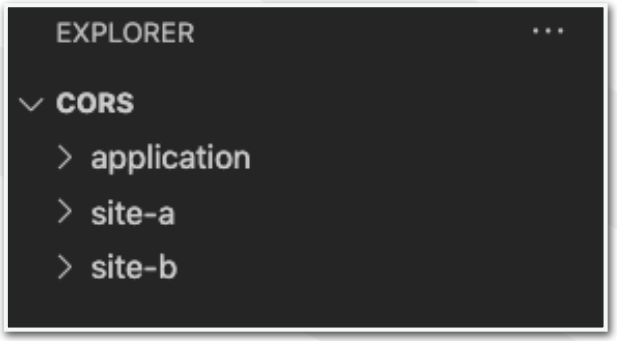
- Voici un petit exemple pour se rendre compte du problème
- Nous allons créer un premier site (site-a) avec :
  - Un fichier `data.php` qui retourne l'heure
  - Un fichier `index.html` doté d'un bouton permettant de faire une requête Ajax vers `data.php`

- Nous allons ensuite crée un second site (site-b) avec juste un fichier `index.html` qui va tenter de faire la même chose que celui de site-a en utilisant le `data.php` du site-a.

- Enfin, nous créerons une application Node.JS qui elle aussi récupérera l'heure à partir de `data.php`.

EXEMPLE

- Créer un nouveau projet `cors`
- Mettre en place la structure de dossier suivante :



- Avec un Terminal (CMD), se placer dans le dossier `site-a`
- Créer les fichiers `index.html` et `data.php`

EXEMPLE

site-a/index.html

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Site A</title>
7   </head>
8   <body>
9       <h1>Site A</h1>
10      <button id="getTimeBtn">Heure</button>
11      <span id="result"></span>
12      <script src="https://code.jquery.com/jquery-3.5.1.min.js"
            integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
            crossorigin="anonymous"></script>
13      <script>
14          $('#getTimeBtn').on('click',(evt) => {
15              $.get('data.php',(data) => {
16                  $('#result').html(data);
17              })
18          })
19      </script>
20  </body>
21  </html>
```

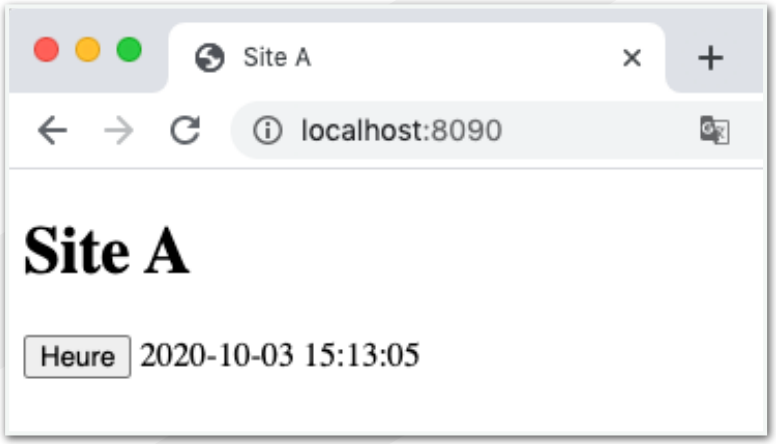
- Pour la ligne 12 cherchez "jquery CDN" avec Google pour obtenir la balise complète.
- Les CDN mettent à disposition des ressources dont des bibliothèques comme jQuery.

EXEMPLE

site-a/data.php

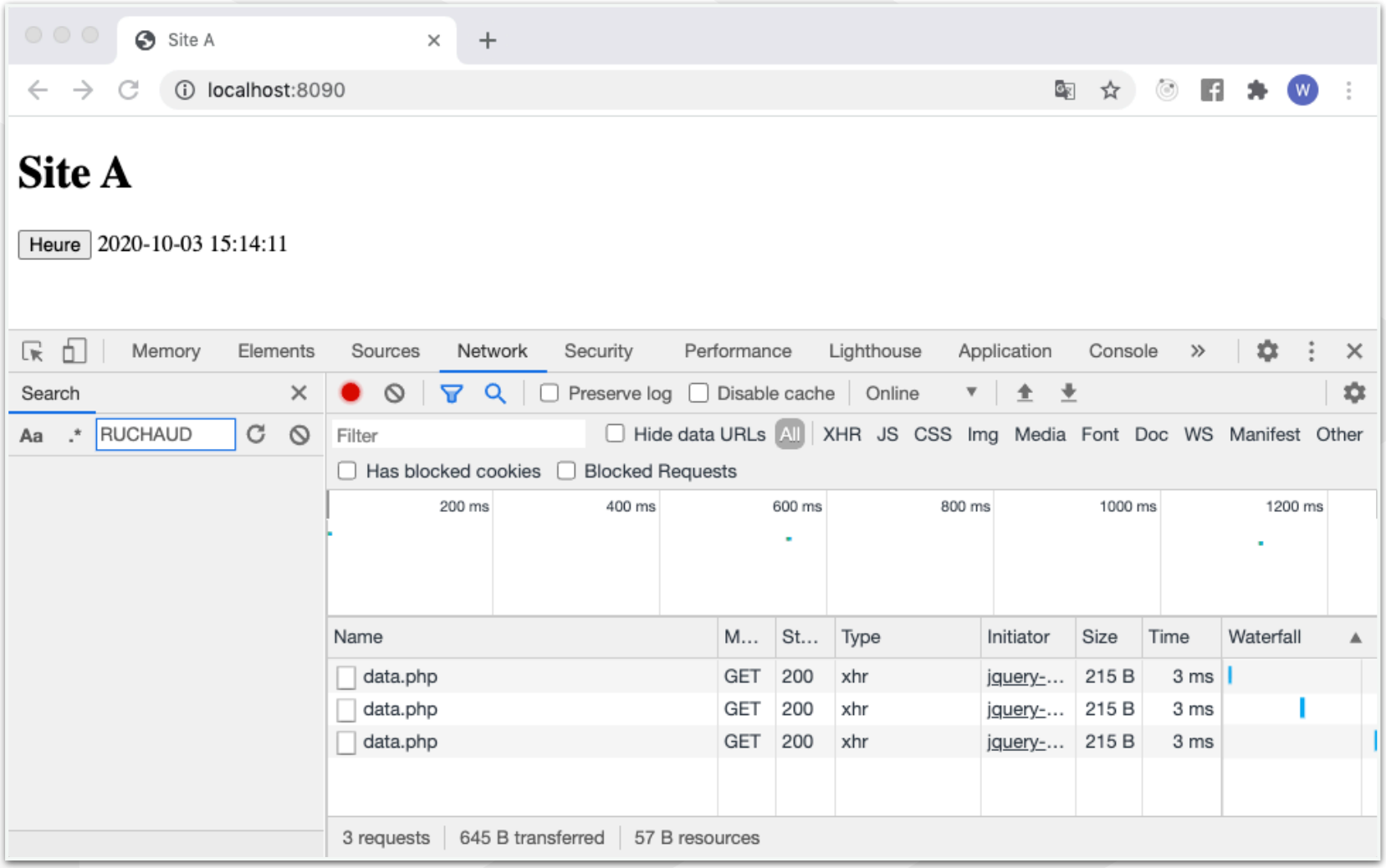
```php
<?php
    echo date("Y-m-d H:i:s");
```

- Il s'agit juste d'affiche/renvoyer l'heure.
- Ouvrir un terminal
  - Bien se placer dans le dossier `site-a`
  - Taper la commande : `php -S 127.0.0.1:8090` (changer de port s'il est déjà pris)
- Avec votre navigateur aller sur http://localhost:8090
- Cliquer sur le bouton, à chaque fois l'heure doit changer.



EXEMPLE

- Ouvrir le debugger du navigateur, onglet Network
- Appuyer à nouveau sur le bouton "Heure", à chaque fois on voit passer la requête Ajax (autant d'appel à `data.php`).



EXEMPLE

- Copier le fichier `index.html` dans le dossier `site-b` (mais pas le fichier `data.php`)
- Faites les corrections suivantes (lignes indiquées en jaune)

site-b/index.html

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Site B</title>
7   </head>
8   <body>
9       <h1>Site B</h1>
10      <button id="getTimeBtn">Heure</button>
11      <span id="result"></span>
12      <script src="https://code.jquery.com/jquery-3.5.1.min.js"
            integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
            crossorigin="anonymous"></script>
13      <script>
14          $('#getTimeBtn').on('click',(evt) => {
15              $.get('http://localhost:8090/data.php',(data) => {
16                  $('#result').html(JSON.stringify(data));
17              })
18          })
19      </script>
20  </body>
21  </html>
```

Requête cross-origin

EXEMPLE

- Ouvrir un second terminal
  - Dans Visual Studio Code le bouton + permet d'en lancer un nouveau en gardant le premier actif (et la poubelle d'en supprimer un)
  - La liste déroulante permet de naviguer de l'un à l'autre

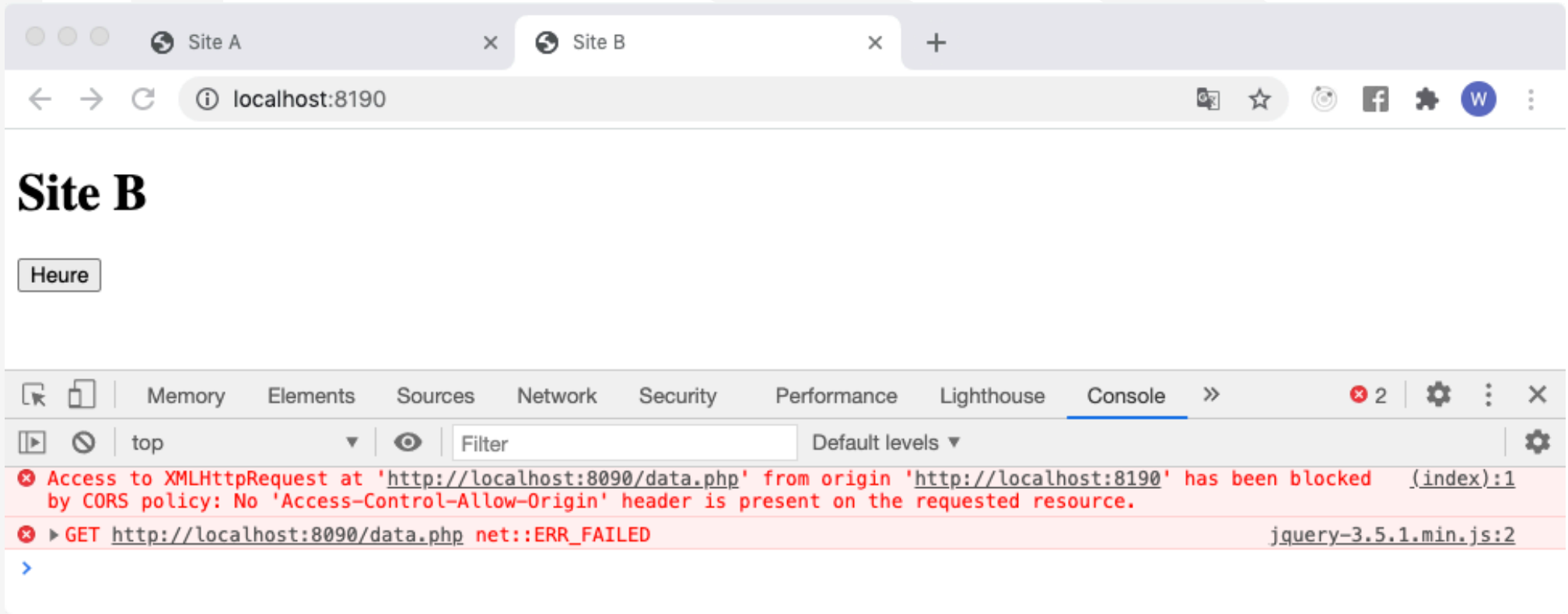
- Se placer dans le dossier `site-b`
- Lancer la commande : `php -S 127.0.0.1:8190`

- Il doit bien y avoir deux serveurs Web PHP qui tournent simultanément (un en 8090 et l'autre en 8190).
- Ouvrir http://localhost:8190 avec votre navigateur, Site B doit apparaître
- Appuyer sur le bouton heure, normalement rien ne se passe.

- Normal !

EXEMPLE

- D'un point de vue Web, localhost:8090 et localhost:8190 sont deux domaines différents, autant que lemonde.fr et google.com.
- La requête Ajax qui part de localhost:8190 à destination de localhost:8090 est rejetée par les CORS.
- Ouvrir la console du navigateur pour le constater.



EXEMPLE

- Ouvrir un troisième terminal
- Se placer dans le dossier `application`
- Créer une application Node.JS
- Installer `axios` une bibliothèque qui permet de faire des requêtes HTTP en JavaScript.
- Créer le fichier `index.js`

application/index.js

```
1  const axios = require('axios');
2
3  axios.get('http://localhost:8090/data.php')
4  .then((result) => {
5      console.log("Heure ",result.data);
6  })
7  .catch((err) => {
8      console.log("Erreur ",err.message)
9  })
```

EXEMPLE

- Exécuter `index.js`
- Constater que l'heure apparaît.

```
Pibook:application william$ node index.js
Heure  2020-10-03 15:24:55
Pibook:application william$
```

- Pourquoi apparaît-elle quand c'est Node.JS et pas quand c'est une requête Ajax ?
- Nous sommes dans deux contextes différents d'une requête HTTP
  - Le problème des CORS est propre au navigateur, pas à JavaScript
  - Avec Node.JS nous développons des applications "de bureau"
  - Avec PHP, dans cet exemple, nous avons créé des applications Web "de navigateur"

- C'est normal qu'une application de bureau puisse requêter tous les domaines contrairement à une page dans un navigateur. Une application de bureau n'a pas d'URL d'origine.

CORS
Partie 2

CAS INSPIRÉ DU RÉEL

- Pourquoi parle de ce problème si une application Node.JS n'a pas ce genre de souci.
- Voici un scénario inspiré réel :
  - J'utilise une application pour savoir combien de temps je consacre à chaque tâche dans mes activités : Clockify (véridique)
  - Clockify existe sous 3 formats :
    - Application Web
    - Application de bureau (Windows et MacOS)
    - Application Mobile
  - Evidemment les données sont liées à mon compte et donc disponibles sur les 3 applications en permanence.
  - Cela sous-entends qu'une API REST qui gère les données chez Clockify est consommées par les 3 types d'application.

CAS INSPIRÉ DU RÉEL

- L'interface Web principale de Clockify est https://clockify.me/tracker
- L'API est basée sur https://global.api.clockify.me/
- C'est donc un domaine différent.

- Normalement, les requêtes Ajax de l'interface vers l'API ne devraient pas fonctionner (CORS)
- Comment ont-ils fait ?
- Ils ont justement paramétré les CORS pour que ce soit possible.
- Cela se fait sur le serveur, c'est lui qui précise quelles requêtes cross-origin il accepte de servir.

OUVERTURE DES CORS

- Deux options sont possibles pour ouvrir les CORS
  - * : toutes les origines sont autorisées DANGEREUX en production
  - la liste exacte des domaines autorisés

- En ce qui nous concerne :

site-a/data.php

```php
<?php
    header("Access-Control-Allow-Origin: *");
    echo date("Y-m-d H:i:s");
```

- À présent le site-b doit avoir accès aux données du site-a et affiche la date et l'heure.

NODE.JS ET EXPRESS

- Nous avons mis au point une API REST avec Node.JS et Express.
- Cette API pourra potentiellement fournir des données pour une application Web (PHP/JavaScript)
- Il est donc important de savoir configurer les CORS pour pouvoir lui donner accès, même si elle est d'un domaine différent.

- Avant de voir comment modifier les CORS avec Express, couper site-a et modifier site-b pour qu'il requête la route /profs de notre API REST (vérifier les ports avant).

site-b/index.html

```
14  $('#getTimeBtn').on('click',(evt) => {
15      $.get('http://localhost:8080/api/v1/profs',(data) => {
16          $('#result').html(JSON.stringify(data));
17      })
18  })
```

EXPRESS ET CORS

- Pour gérer les CORS avec Express, il faut installer le middleware dont le nom est le plus original au monde : cors

  `npm install cors`

- Ajouter ces quelques lignes (marquées en bleu)

/index.js (extrait)

```
1   const express = require('express');
2   const bodyParser = require('body-parser');
3   const cors = require('cors');
4
5   const morgan = require('morgan');
6   const path = require('path');
7   const rfs = require('rotating-file-stream');
8
9   const expresOasGenerator = require('express-oas-generator');
10
11  const profsRouter = require('./routers/profsRouter').router;
12  const messagesRouter = require('./routers/messagesRouter').router;
13
14  const app = express();
15
16  expresOasGenerator.handleResponses(app, {});
17  app.use(cors());
```

- Toutes les origines sont autorisées pour toutes les routes (dangereux)

SPÉCIFIER UNE ORIGINE

- Pour spécifier les origines autorisées :

/index.js (extrait)

```
14  const app = express();
15
16  expresOasGenerator.handleResponses(app, {});
17
18  let whitelist = ['http://localhost:8090'];
19  let corsOptions = {
20      origin : function( req, callback ) {
21          if(whitelist.indexOf(req) !== -1) {
22              callback(null, true);
23          } else {
24              callback(null,false);
25          }
26      }
27  }
28
29  app.use(cors(corsOptions));
```

- Il suffit maintenant de les lister dans whitelist
- Possibilités de ne le faire qu'au niveau d'une route, au niveau des méthodes (GET, POST)…

JWT

JWT
Partie 1

PROTÉGER LES ACCÈS

- Toutes les API REST ne sont pas publiques.
- Il est nécessaire de pouvoir mettre en place des routes nécessitant une authentification.
- Nous allons mettre en place JWT (JSON Web Token) qui permet d'assurer une authentification par jeton.
- JWT est une façon simple et standardisée (RFC 7519) de gérer des jetons d'authentification.

JWT

- Un jeton JWT est composé de 3 parties, toutes encodées en base 64 :
  - Une entête (header)
  - Un contenu ou chargement (**payload**)
  - Une signature

- Un jeton JWT se génère à l'aide d'une **clé privée** (chaine longue et complexe préférée)

- Principales caractéristiques :
  - L'entête précise l'algorithme utilisé pour la signature et le type de jeton
  - Le payload contient les informations, comme par exemple l'id de l'utilisateur (ne pas mettre d'information sensible), la durée de validité
  - La signature est obtenue par une concaténation de l'entête et du payload chiffrée avec la clé privée.

JWT

- Un jeton JWT est généré par un serveur d'authentification qui vérifie les credentials (login + mot de passe).
- Il peut être utilisé tout au long de sa durée de vie.
  - Il n'y a pas de révocation possible.
  - Une fois obtenu il n'est plus nécessaire de recontacter le serveur d'authentification.

- Une route "protégée" à l'aide de JWT se basera sur l'incorporation d'un jeton valide dans les entêtes de la requête HTTP.
- Un jeton est valide si :
  - Sa date d'expiration n'est pas dépassée.
  - Sa signature correspond à l'entête et le payload cryptés avec la clé privée.

- Ainsi pour valider un jeton JWT, il est juste nécessaire de connaître la clé privée.

JWT

- JWT n'est pas un système parfait
  - Pas de révocation possible : si un compte doit être bloqué, il ne le sera vraiment que lorsque tous les jetons de ce compte auront expiré.
  - En cas de modification de mot de passe, un jeton généré avec l'ancien sera valide jusqu'à son expiration
  - Pas de prolongation possible : le jeton embarque sa durée de validité. Il faudra se reconnecter en cas d'expiration.

- Néanmoins, JWT est un standard grandement utilisé, notamment quand plusieurs ressources différentes doivent se référer à une même authentification.

PROTÉGER LES ACCÈS

- Notre objectif : mettre en place une route nécessitant d'être authentifié.
- Pour cela nous allons :
  - Rajouter à nos profs un mot de passe
  - Rajouter une entité message (une simple chaîne de caractères)
  - Mettre en place une route `/messages` pour servir cette entité (POST et GET seulement)
  - Protéger cette route avec JWT (GET seulement)
  - Rajouter une route `/auth` pour s'authentifier

AJOUTER LE CHAMP MOT DE PASSE

- Ajouter le champ mot de passe dans la table de la base de donnée (VARCHAR 512)
- Ajouter le mot de passe dans le modèle Sequelize des profs.
- Ajouter le mot de passe dans les opérations `add()` et `update()` du contrôleur.
- Vérifier que tout continue de fonctionner avec le mot de passe en clair.

- Évidemment un mot de passe ne doit pas être stocké en clair.
- Nous allons utiliser la bibliothèque bcrypt pour obtenir un hash salé du mot de passe.
- Exécuter : `npm install bcrypt`

UTILISER BCRYPT

- Dans le contrôleur `profsController.js` ajouter le `require` pour utiliser bcrypt.

/controllers/profsController.js (extrait)

```
1   const {profsModel} = require('../models');
2   const bcrypt = require('bcrypt');
3
```

- Puis dans `add()`, faire appel à bcrypt. Placer la partie qui enregistre les données à l'intérieur d'un callback de bcrypt.

/controllers/profsController.js (extrait)

```
44          bcrypt.hash(motdepasse,5,(err,encrypted) => {
45              profsModel.create({nom,prenom,bureau,motdepasse:encrypted})
46              .then((data)=>{
47                  res.status(201).json({
48                      'status':'sucess',
49                      'message': 'Prof ajouté'
50                  })
51              })
```

- Penser à l'accolade et à la parenthèse à ajouter après le `.catch()` pour fermer le tout.
- Faire la même chose pour `update()`.

UTILISER BCRYPT

- Tester en insérant un prof et en faisant un update d'un existant.
- Le champ mot de passe dans la base de donnée doit ressembler à une chaîne alphanumérique assez longue.

- Dans la capture seul un mot de passe est hashé.

	id	nom	prenom	bureau	motdepasse
Éditer Copier Supprimer	1	Ruchaud	William	210	
Éditer Copier Supprimer	3	Gaudin	Hervé	217	
Éditer Copier Supprimer	4	Durousseau	Christophe	214	titi
Éditer Copier Supprimer	5	Frigui	Kamel	220	$2b$05$b6olwXg/v46oeKRm1jQD2uXe2juCZ5j4a6fNSsN5xrz...

AJOUTER UNE ENTITÉ MESSAGE

- C'est une route de cette entité qui sera protégée par authentification.
- Cette entité ne contient qu'un seul champ (message) en plus de son id.
- Créer le modèle qui va avec.
- Créer le contrôleur puis le routeur qui vont avec et tester (routes GET et POST suffiront).

/models/messages.js

```
1  module.exports = (sequelize,DataTypes) => {
2      return sequelize.define('messages', {
3          text: {
4              type : DataTypes.STRING
5          }
6      })
7  }
```

AJOUTER UNE ENTITÉ MESSAGE

> /connect.js

/connect.js

```
20  const profsModel = require('./models/profsModel')(sequelize,DataTypes);
21  const messagesModel = require('./models/messagesModel')(sequelize,DataTypes);
22
23  sequelize.sync();
24
25  module.exports = {
26      db : sequelize,
27      profsModel : profsModel,
28      messagesModel : messagesModel
29  }
```

AJOUTER UNE ENTITÉ MESSAGE

> /controllers/messagesController.js

/controllers/messagesControllers.js (1/2)

```
1   const {messagesModel} = require('../connect');
2
3   module.exports = {
4       getAll : function(req,res) {
5           messagesModel.findAll()
6           .then((data) => {
7               res.status(200).json(
8                   {
9                       "status":'success',
10                      "data": data
11                  }
12              )
13          })
14      },
```

AJOUTER UNE ENTITÉ MESSAGE

- La suite du contrôleur

/controllers/messagesControllers.js (2/2)

```
16      add: function (req,res) {
17          let { text } = req.body;
18          if(text == null) {
19              res.status(409).json({
20                  'status':'error',
21                  'message': 'Données incomplète pour message'
22              })
23              return;
24          }
25
26          messagesModel.create({text})
27          .then((data)=>{
28              res.status(201).json({
29                  'status':'sucess',
30                  'message': 'Message ajouté'
31              })
32          })
33          .catch((err)=> {
34              res.status(404).json({
35                  "status" : 'error',
36                  "message" : err.message
37              })
38          })
39      }
40  }
```

AJOUTER UNE ENTITÉ MESSAGE

Le routeur

/routers/messagesRouter.js

```
1   const express = require('express');
2   const messagesController = require('../controllers/messagesController');
3
4   function buildRoutes() {
5       let router = express.Router();
6
7       router.route('/messages/').get(messagesController.getAll);
8       router.route('/messages/').post(messagesController.add);
9       return router;
10  }
11
12  exports.router = buildRoutes();
```

AJOUTER UNE ENTITÉ MESSAGE

/index.js

/index.js

```
 9  const expresOasGenerator = require('express-oas-generator');
10
11  const profsRouter = require('./routers/profsRouter').router;
12  const messagesRouter = require('./routers/messagesRouter').router;
13
14  const app = express();
15
```

/index.js

```
39  app.use(morgan('combined', { stream: accessLogStream}));
40
41
42  app.use('/api/v1',profsRouter);
43  app.use('/api/v1',messagesRouter);
44
45  app.get('/', (req,res) => {
46      res.send("API REST avec Express");
47  })
48
49  expresOasGenerator.handleRequests();
```

JWT
Partie 2

JSON WEB TOKEN

- Installer JSON Web Token

  `npm install jsonwebtoken`

- Créer le fichiers `/utils/jwt.utils.js` (et le dossier `/utils`)
- `JWT_SIGN_SECRET` doit être une chaîne complexe, pas forcément celle qui figure ici. L'obtenir avec un générateur de mot de passe sur le Web.

/utils/jwt.utils.js

```
1   const jwt = require('jsonwebtoken');
2
3   const JWT_SIGN_SECRET = "%D*F-JaNdRgUkXp2s5v8y/B?E(H+KbPeShVmYq3t6w9z$C&F)
        J@NcQfTjWnZr4u7";
4
5   module.exports = {
6       generateTokenForUser : function(user) {
7           return jwt.sign({
8               profId : user.id
9           },
10          JWT_SIGN_SECRET,
11          {
12              expiresIn:'1h'
13          })
14      }
15  }
```

JSON WEB TOKEN

- Ajouter la route `/auth` au routeur profs.

/routers/profsRouter.js

```
1   const express = require('express');
2   const profsController = require('../controllers/profsController');
3
4   function buildRoutes() {
5       let router = express.Router();
6
7       router.route('/profs/').get(profsController.getAll);
8       router.route('/profs/:id').get(profsController.getById);
9       router.route('/profs/').post(profsController.add);
10      router.route('/profs/:id').delete(profsController.delete);
11      router.route('/profs/:id').put(profsController.update);
12
13      router.route('/auth').post(profsController.login);
14
15      return router;
16  }
17
18  exports.router = buildRoutes();
19
```

JSON WEB TOKEN

- Rajouter `jwt.utils.js` dans les `require` de `/controllers/profsController.js`

/controllers/profsController.js

```
1   const {profsModel} = require('../connect');
2   const bcrypt = require('bcrypt');
3   const jwtUtils = require('../utils/jwt.utils');
4
5   module.exports = {
6       getAll : function(req,res) {
7           profsModel.findAll()
8           .then((data) => {
9               res.status(200).json(
10                  {
11                      "status":'success',
12                      "data": data
13                  }
14              )
15          })
16      },
```

JSON WEB TOKEN

- Rajouter la méthode login

/controllers/profsController.js (1/2)

```
112     login: function(req,res) {
113         let { nom, motdepasse } = req.body;
114         if(nom == null || motdepasse == null) {
115             res.status(409).json({
116                 'status' : 'error',
117                 'message' : 'Données incomplètes pour authentification'
118             });
119             return;
120         }
```

JSON WEB TOKEN

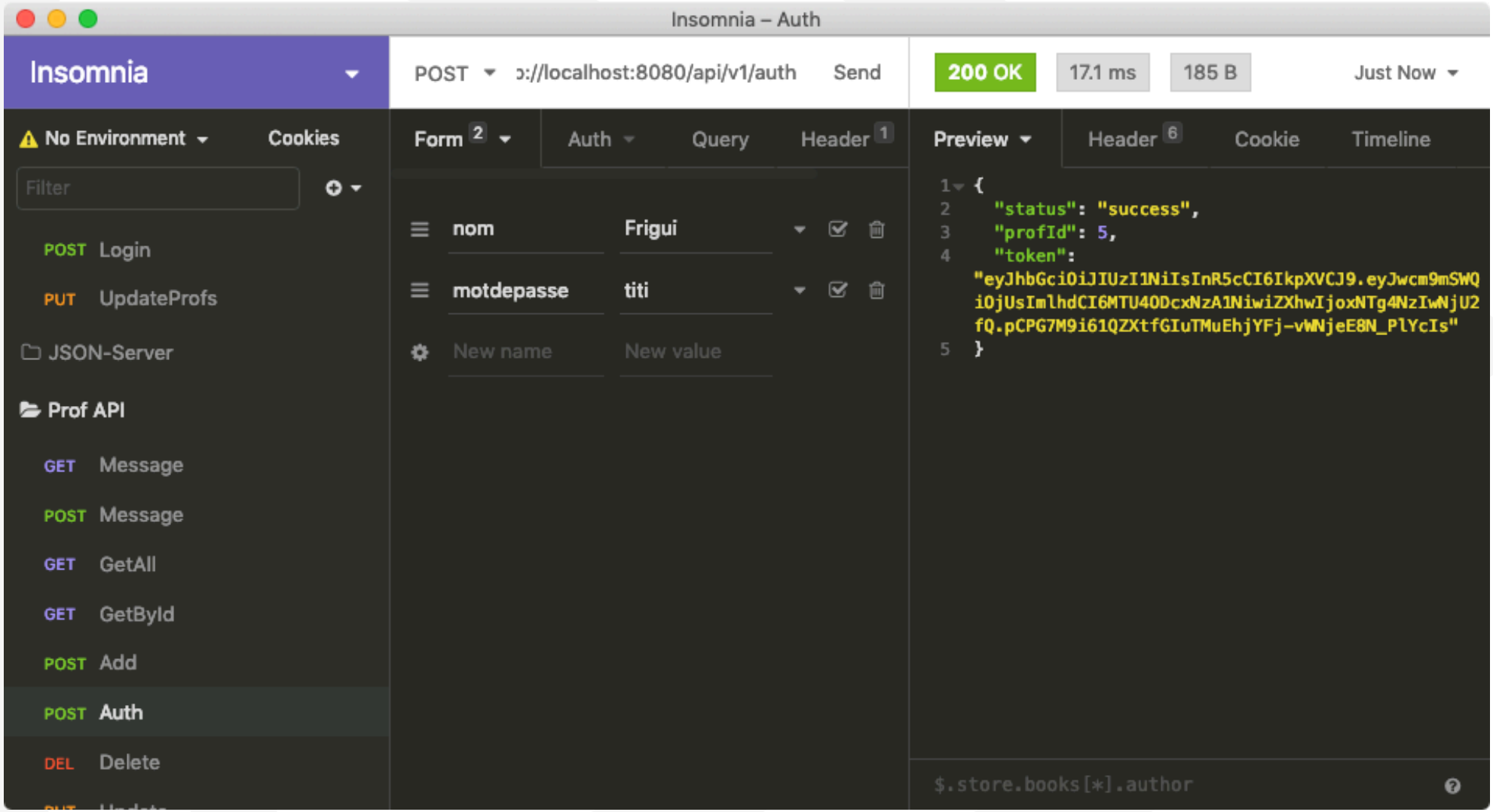
- Rajouter la méthode login

/controllers/profsController.js (2/2)

```
121         profsModel.findOne({where: {nom:nom}})
122         .then((profFound) => {
123             if(profFound) {
124                 bcrypt.compare(motdepasse,profFound.motdepasse, (err, resBcrypt) => {
125                     if(resBcrypt) {
126                         res.status(200).json({
127                             'status' : 'success',
128                             'profId' : profFound.id,
129                             'token' : jwtUtils.generateTokenForUser(profFound)
130                         })
131                         return;
132                     } else {
133                         res.status(403).json({
134                             status : 'error',
135                             message : 'donnees de connexion invalides'
136                         })
137                     }
138                 })
139             } else {
140                 res.status(403).json({
141                     status : 'error',
142                     message : 'donnees de connexion invalides'
143                 })
144             }
145         })
146     }
```

JSON WEB TOKEN

- Dans Insomnia ajouter une route POST vers `/api/v1/auth`
- Y placer dans le body en form-url-encoded le `nom` et le `motdepasse` en clair d'un prof.



JSON WEB TOKEN

Dans `/utils/jwt.utils.js` ajouter les fonctions suivantes :

/utils/jwt.utils.js

```
15      parseAuthorization : function(authorization) {
16          if(authorization.indexOf('Bearer') !== 0) return null;
17          return (authorization != null) ? authorization.replace('Bearer ','') : null;
18      },
19
20      getUserId: function(authorization) {
21          let userId = -1;
22          let token = module.exports.parseAuthorization(authorization);
23          if(token != null) {
24              try {
25                  let jwtToken = jwt.verify(token, JWT_SIGN_SECRET);
26                  if(jwtToken != null) {
27                      userId = jwtToken.userId;
28                  }
29              } catch(err) {}
30          }
31          return userId;
32      }
```

JSON WEB TOKEN

- Dans `/controllers/messagesController.js` ajouter les lignes suivantes :

/controllers/messagesController.js

```
1   const {messagesModel} = require('../connect');
2   const jwtUtils = require('../utils/jwt.utils');
3
4   module.exports = {
5       getAll : function(req,res) {
6           let headerAuth = req.headers['authorization'];
7           let profId = jwtUtils.getUserId(headerAuth);
8           if(profId==-1) {
9               res.status(403).json({
10                  status: 'error',
11                  message: 'acces interdit'
12              })
13              return;
14          }
15          messagesModel.findAll()
16          .then((data) => {
17              res.status(200).json(
18                  {
19                      "status":'success',
20                      "data": data
21                  }
22              )
23          })
24      },
25
```

JSON WEB TOKEN

- Dans Insomnia créer une nouvelle requête GET vers `/messages`
- Placer dans le header un champ `authorization` avec pour valeur `Bearer` plus le token obtenu lors du `/login` (séparés par un espace et sans les guillemets)