Programmation Web – Avancé

JavaScript & Node.js

Partie 26 : Authentification sécurisée et autorisation d'un utilisateur via JWT

Version 2020

Attribution – Partage dans les Mêmes Conditions 4.0 International (CC BY-SA 4.0)

Presentation template by SlidesCarnival

Introduction à l'authentification sécurisée d'un utilisateur via JWT

En route vers une authentification « stateless »...

Différents moyens d'authentification ?

- Stateful authentication (session management)
  - User session data : côté serveur (avec cookie géré par Express)
  - Inconvénients & Avantages ?

- Stateless authentication
  - User session data : côté client
  - Inconvénients & Avantages ?
  - Exemple : JWT

JWT

- JSON Web Token (JWT) : [90.]
- JWT : **xxxxx.yyyyy.zzzzz**
  - Header : encodé
  - Payload : encodé, pas crypté !
  - Signature : cryptée !

Mise en place de l'authentification et autorisation via JWT

Scénario nominal :

- A) Frontend : Login & authentification : si pas de JWT enregistré, demande à une Web API d'authentifier l'utilisateur.
  NB : Si JWT enregistré, récupération de la session.
- B) Backend : Si authentification OK, création d'un JWT et renvoi au frontend

Mise en place de l’authentification et autorisation via JWT

- C) Frontend : Si nécessaire, enregistrement du token côté client & accès à une secure route côté client seulement si JWT / user authentifié

Frontend: affichage du composant de Login suite à une action d'un utilisateur et submit

Demo : Monolith SPA with JWT auths

Email

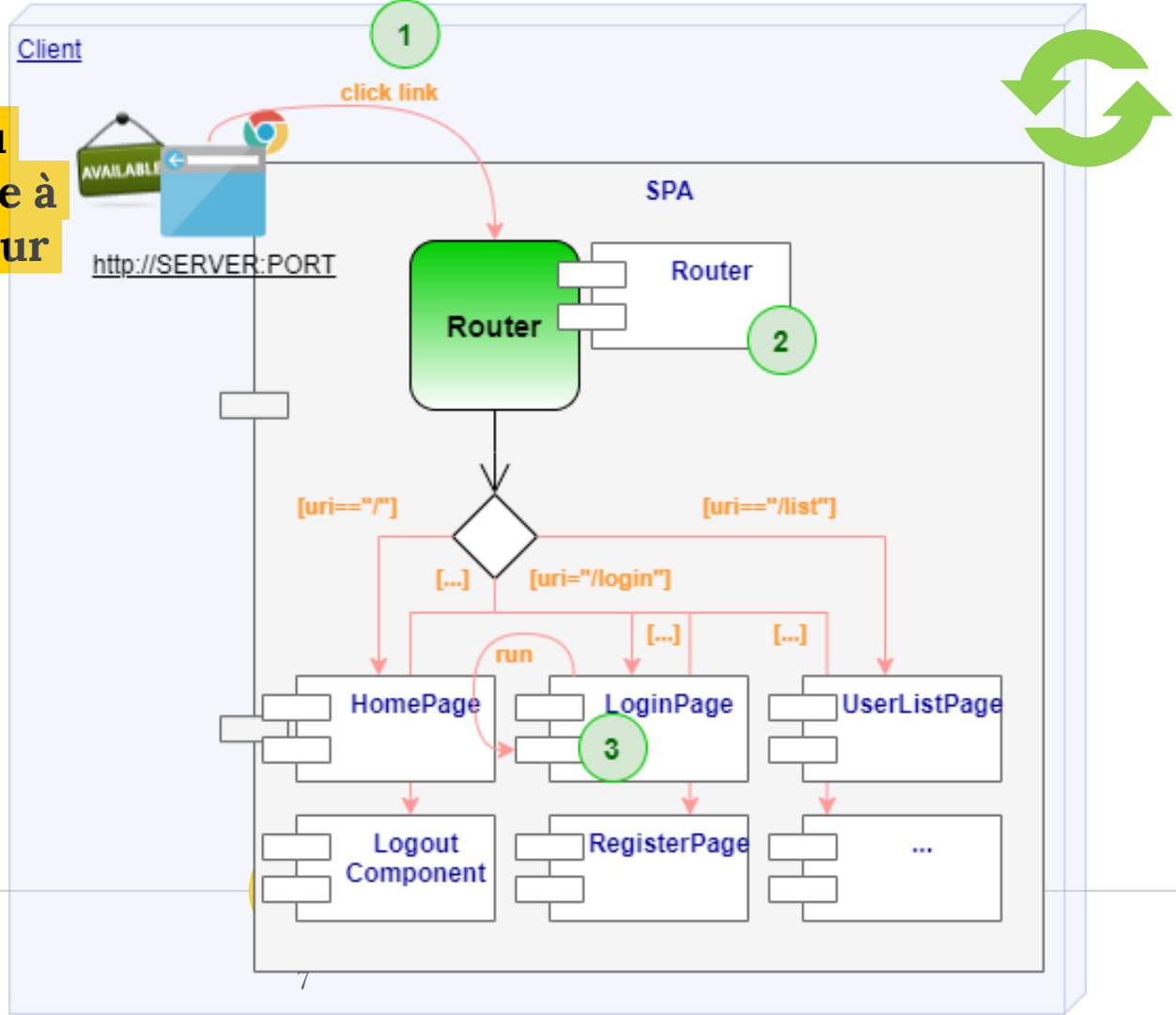
teacher@vinci.be

Password

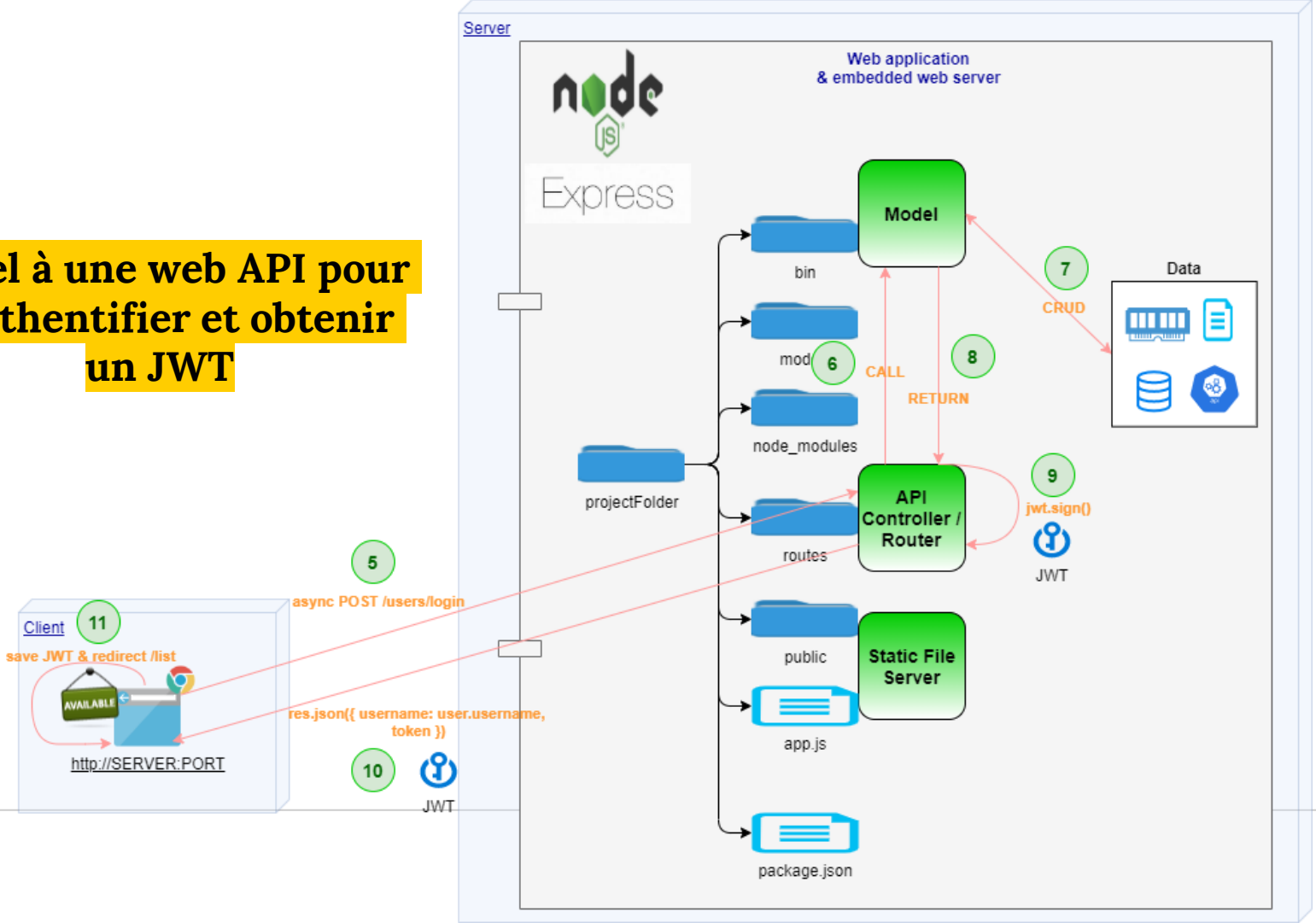
•••••••

Submit

4



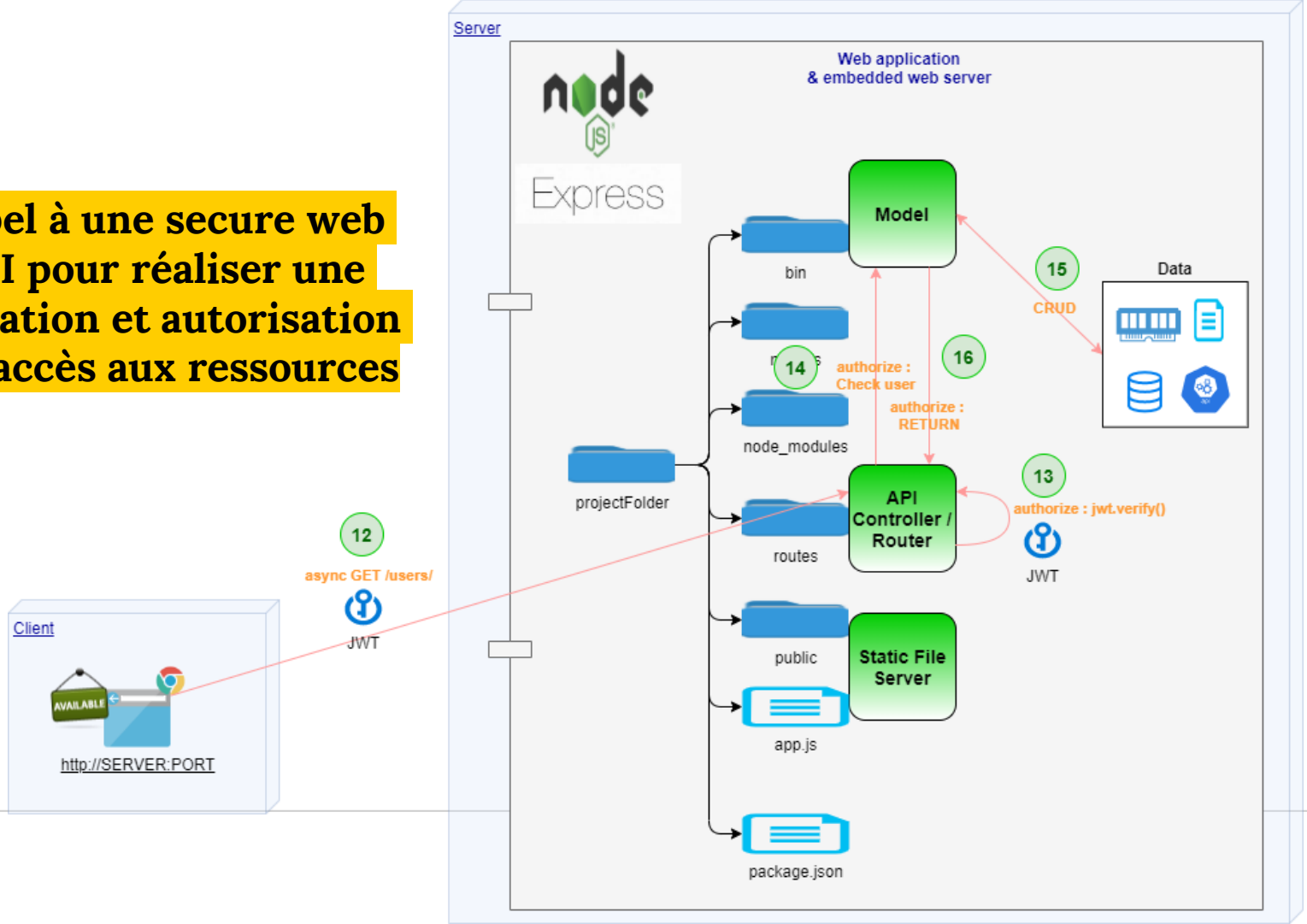
Appel à une web API pour s'authentifier et obtenir un JWT



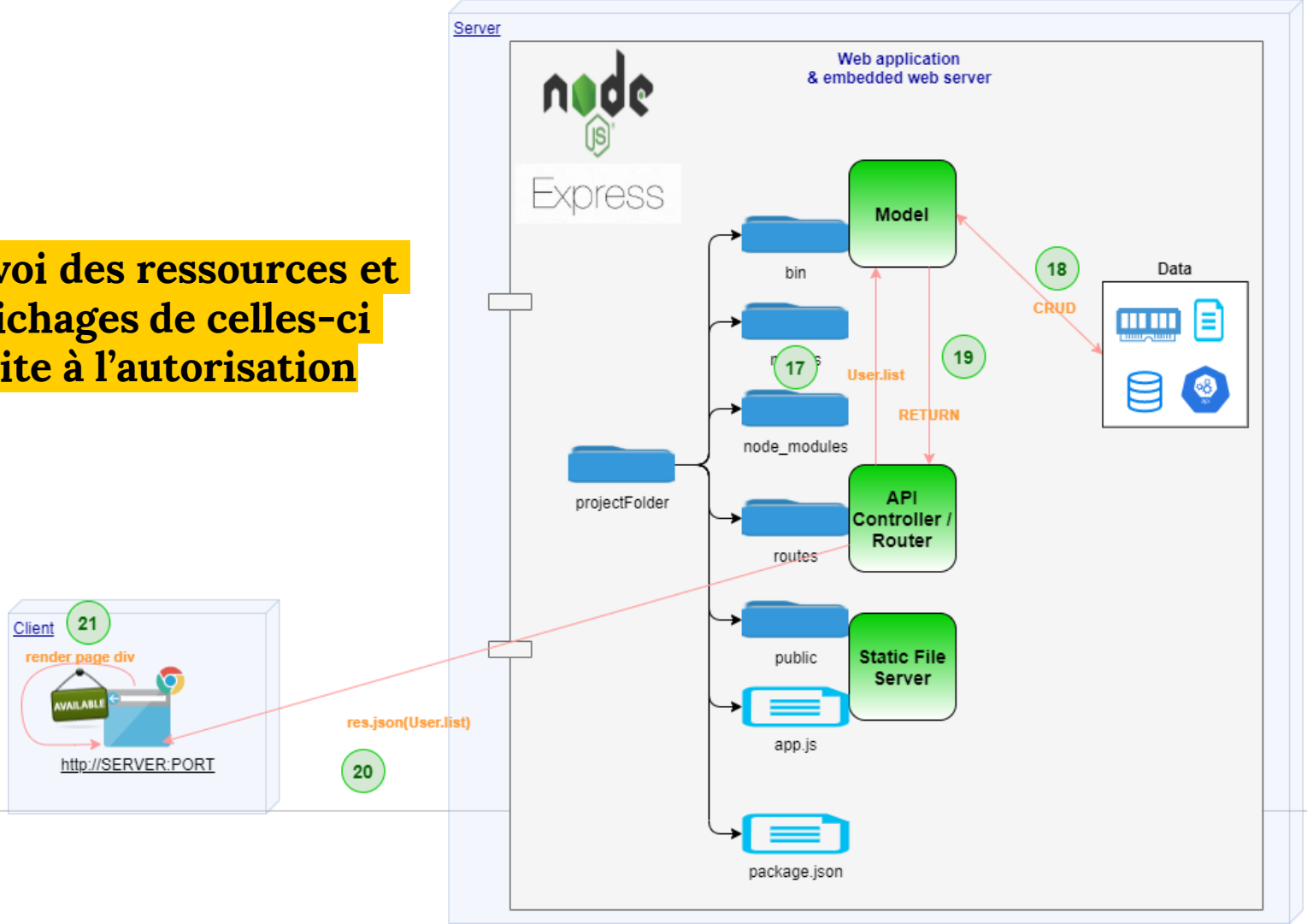
Mise en place de l'authentification et autorisation via JWT

- D) Frontend : exécution du composant associé à la Secure route suite à l'appel à une Secure Web API en envoyant le JWT
- E) Backend : Autorisation & Opération sur les ressources si JWT valide et renvoi au frontend des données ou un code d'erreur
- F) Frontend : Si code d'erreur, demande de login, sinon, éventuel affichage de nouvelles données

Appel à une secure web API pour réaliser une opération et autorisation de l'accès aux ressources



Renvoi des ressources et affichages de celles-ci suite à l'autorisation



JWT côté backend

- JWT middleware : **jsonwebtoken** [91.]
- Installation : **npm install jsonwebtoken**

JWT côté backend

- Utilisation du middleware :

```
const jwt = require("jsonwebtoken");
const jwtSecret = "jkjJ1235Ohno!";
const LIFETIME_JWT = 24 * 60 * 60 * 1000; // 10;// in seconds // 24 * 60 * 60
* 1000 = 24h
```

- jwtSecret : connu uniquement par le serveur permettant :
  - de signer un token
  - de vérifier la signature d'un token

JWT côté backend : Création d'un token via jwt.sign()

```
router.post("/login", function (req, res, next) {
  let user = new User(req.body.email, req.body.email, req.body.password);
    user.checkCredentials(req.body.email, req.body.password).then((match) => {
    if (match) {
      jwt.sign(
        { username: user.username },
        jwtSecret,
        { expiresIn: LIFETIME_JWT },
        (err, token) => {
          if (err) return res.status(500).send(err.message);
          return res.json({ username: user.username, token });
        }
      );
    } else return res.status(401).send("bad email/password");
  });
});
```

JWT côté backend : Vérification d'un token via jwt.verify()

```
/**
 * Authorize middleware to be used on the routes to be secured
 */
const authorize = (req, res, next) => {
  let token = req.get("authorization");
  if (!token) return res.status(401).send("You are not authenticated.");

  jwt.verify(token, jwtSecret, (err, token) => {
    if (err) return res.status(401).send(err.message);
    let user = User.getUserFromList(token.username);
    if (!user) return res.status(401).send("User not found.");
    // authorization is completed, call the next middleware
    next();
  });
};
```

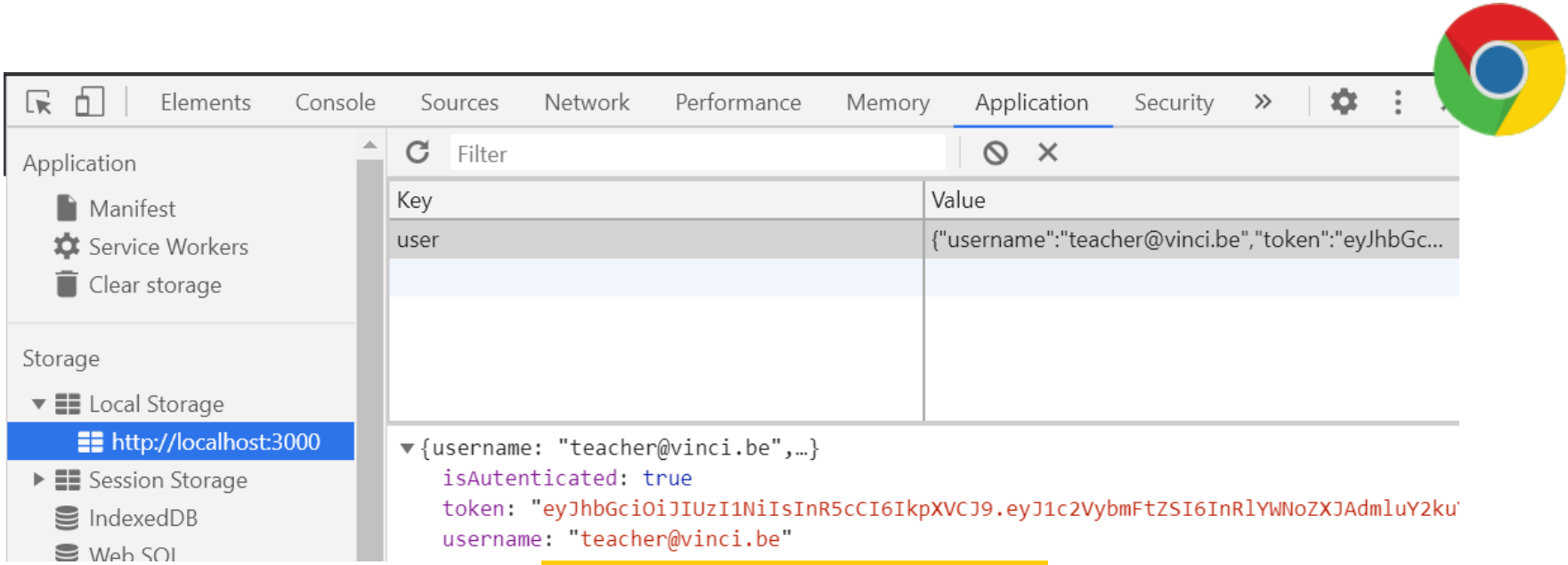
Sécurisation des API

- Vérification d'un token lors d'une requête (user authorization)
  - Utilisation d'un Middleware à passer aux « secure routes »
  - Possibilté de passer plusieurs middlewares à une **route.METHOD()**

```
/* GET user list : secure the route with JWT authorization */
router.get("/", authorize, function (req, res, next) {
  return res.json(User.list);
});
```

JWT côté client

- Gestion de sessions côtés client soit via cookies ou via le web storage
- Web storage plus moderne
- Visulalisation du token dans le localStorage et décodage via jwt.io [90.]



Visualisation d'un token dans le Local Storage

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
1c2VybmFtZSI6InRlYWNoZXJAdmluY2kuYmUiLCJ
pYXQiOjE1OTYxOTQyODYsImV4cCI6MTY4MjU5NDI
4Nn0.8rJab_P3bGtRPpz3GrBXUCqCMTWBKUAiG1t
SZ_2NKP0
```

Décodage du token via via jwt.io [90.]

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "username": "teacher@vinci.be",
  "iat": 1596194286,
  "exp": 1682594286
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

JWT côté client : Appel à une Secure Web API

- Ajout du token aux **Authorization headers** au sein de vos appels asynchrones

```
const UserListPage = () => {
  const user = getUserSessionData();
  if (!user) RedirectUrl("/error", "Resource not authorized. Please login.");
  fetch("/api/users", {
    method: "GET",
    headers: {
      Authorization: user.token,
    },
  })
    .then(// deal with the API response
```

SPA & Secure Routes avec JWT

- **DEMO : SPA monolithique avec JWT auths (authentication & authorization)**

  MyCMS possède maintenant une Secure Route API pour **/users/**

  - Tester les appels via REST Client :
    - http://localhost:3000/api/users sans token
    - Avec token
  - Full secure, sans password sauvegardé :)

SPA & Secure Routes avec JWT

- **DEMO : SPA monolithique avec JWT auths (authentication & authorization)**
  - A vous de tester : quid si durée de vie du JWT est de 10sec ?
  - Quid si token est modifié côté client ?