DOCUMENTATION TECHNIQUE

Td Temp / Génie Climatique

Nicolas Leroy - ECF Study

session décembre 2021

A. spécifications techniques

Pour Le front-end :

- React.js
- Material ui V5 (frameworks ui pour react)
- Bcrypt Js
- DevExtreme React (librarie react pour les graphiques)
- React-Paparse (librarie react pour convertir le csv en json)
- axios (librairie pour les requêtes ajax)
- react router dom

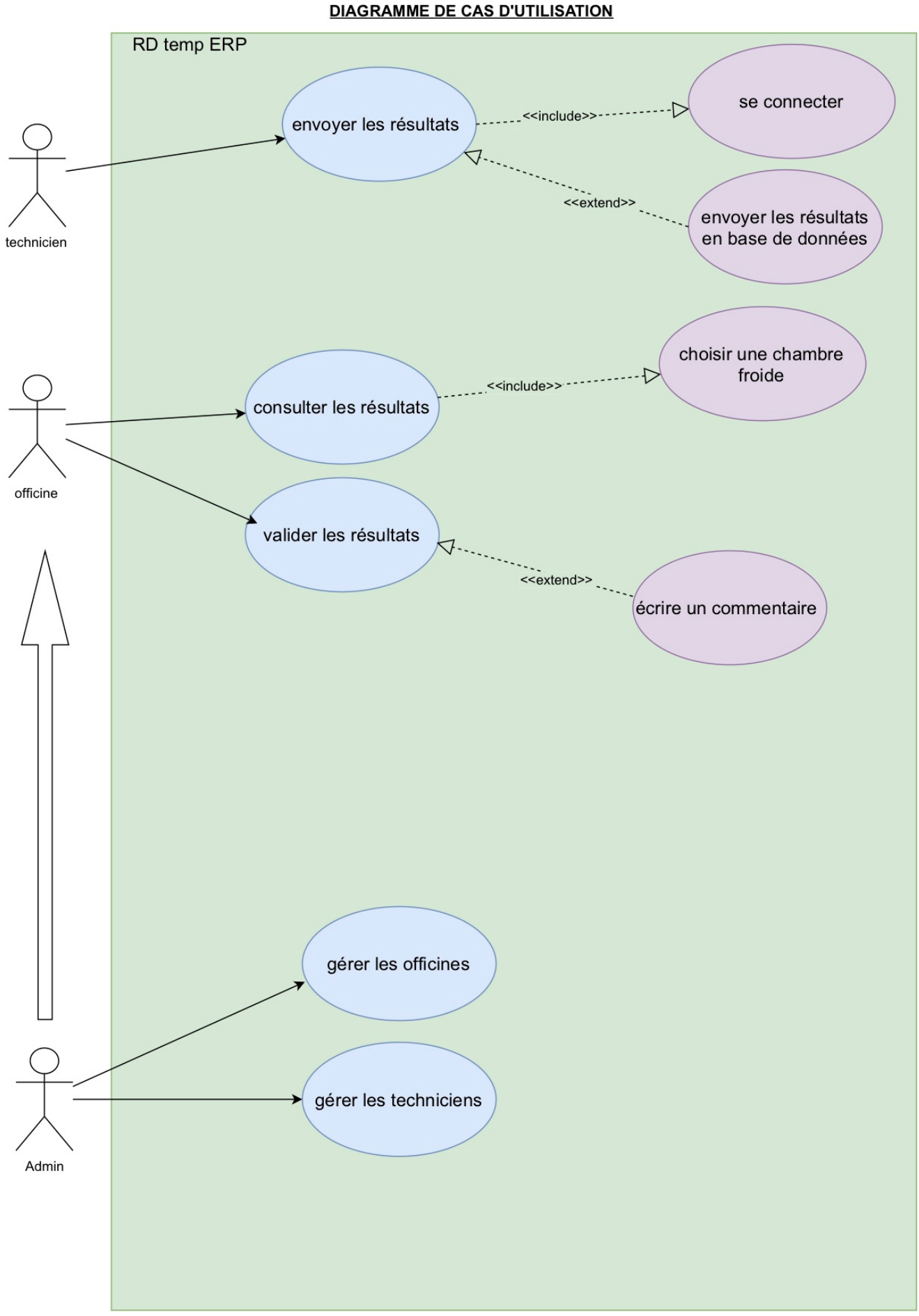
Pour Le back-end :

- Symfony
- Api - Platform

Pour Le déploiement :

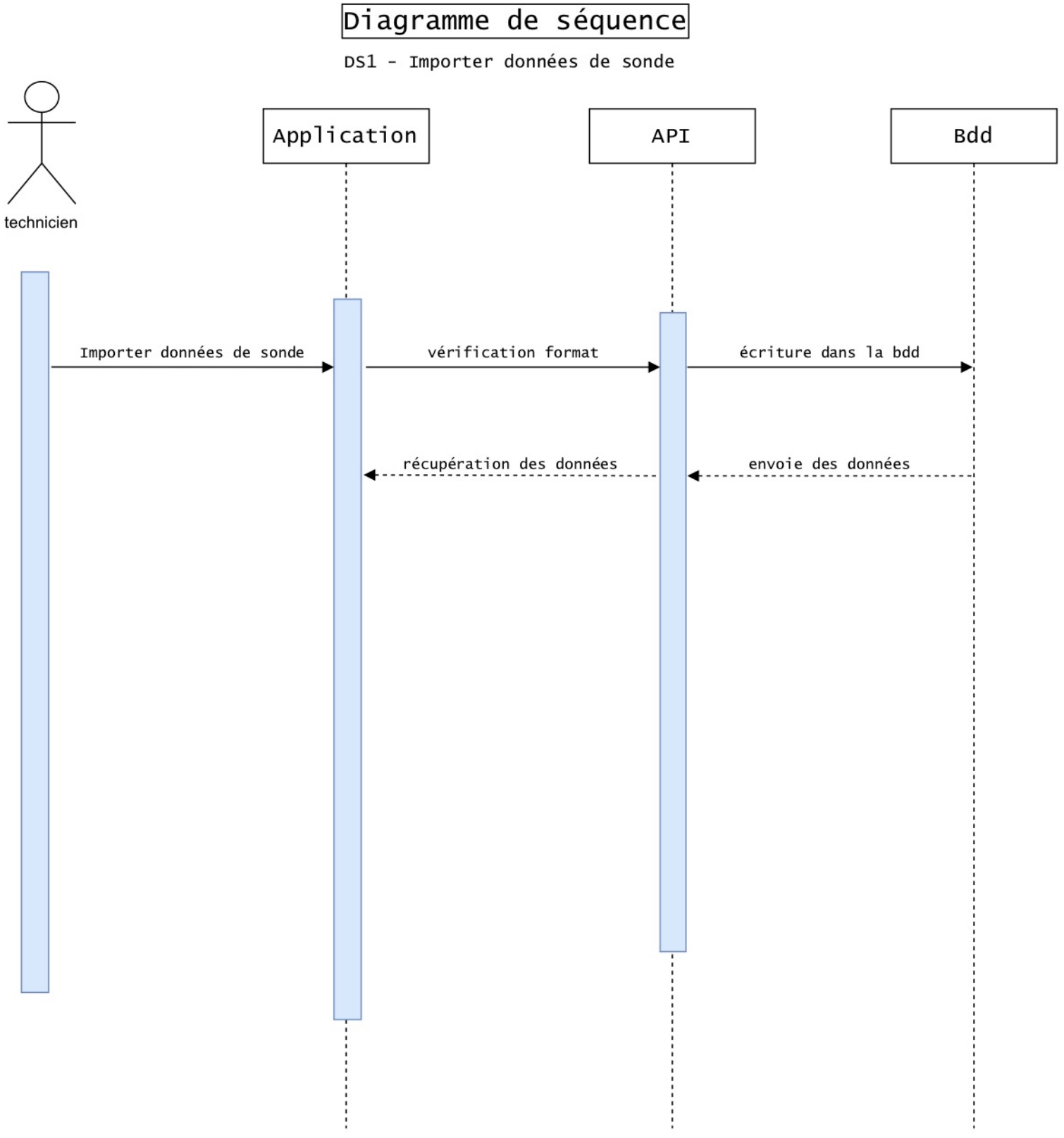
- Heroku
- freesqldatabase (Apache/2.4.18 (Ubuntu) - Version de PHP : 7.0.33)

B. Diagramme de Cas d'utilisation



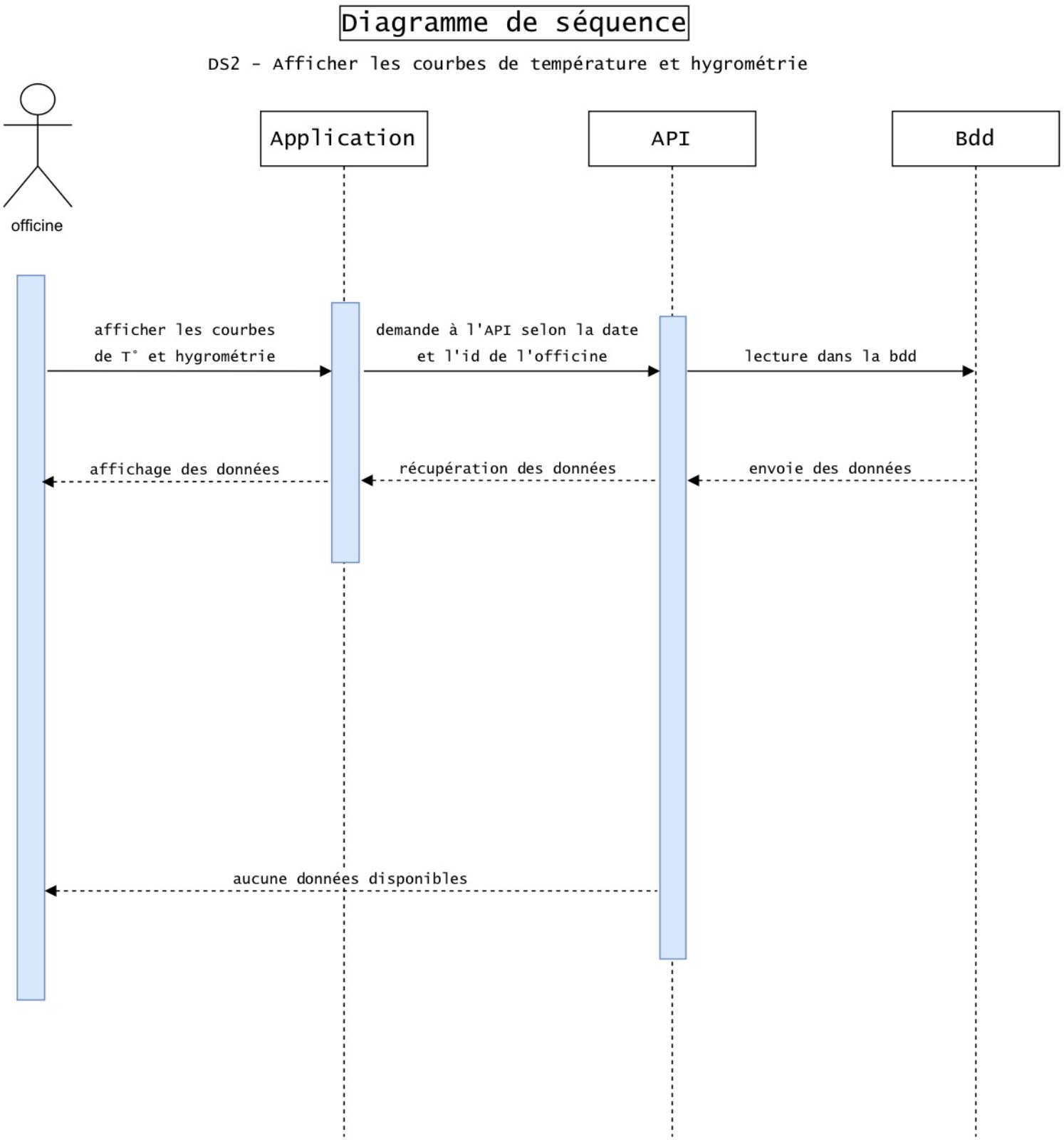
B. Diagramme de séquences

01. DS1 - Importer données de sonde



B. Diagramme de séquences

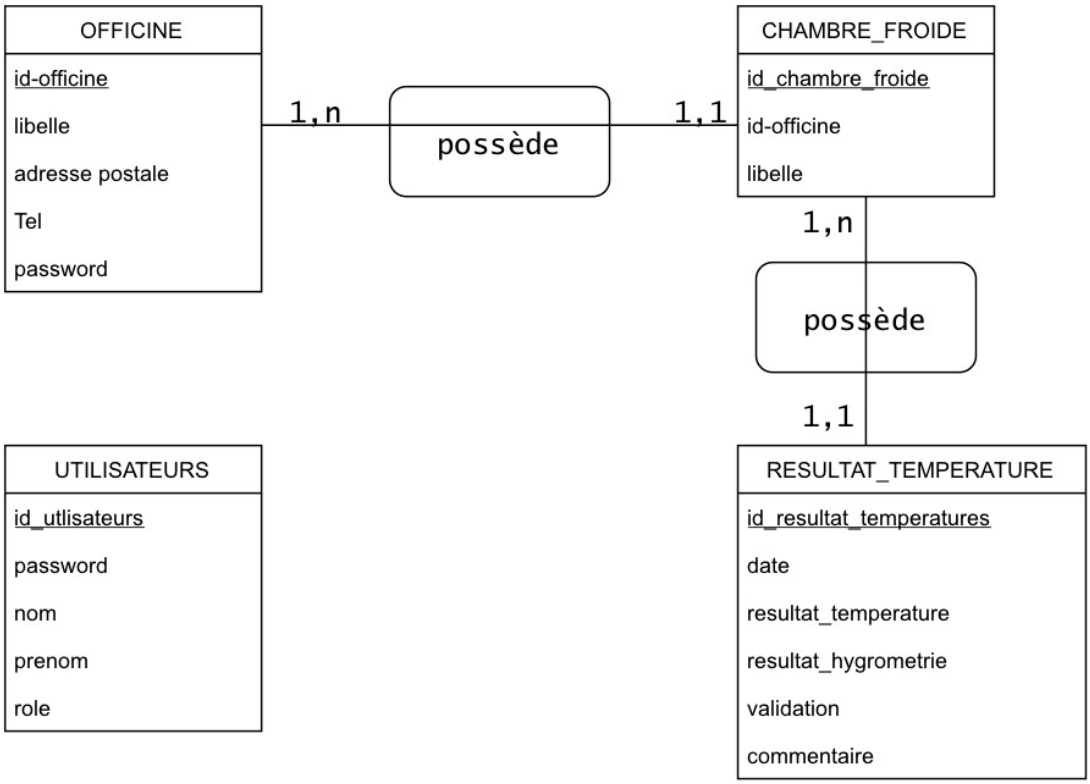
02. DS2 - Afficher les courbes de température et hygrométrie



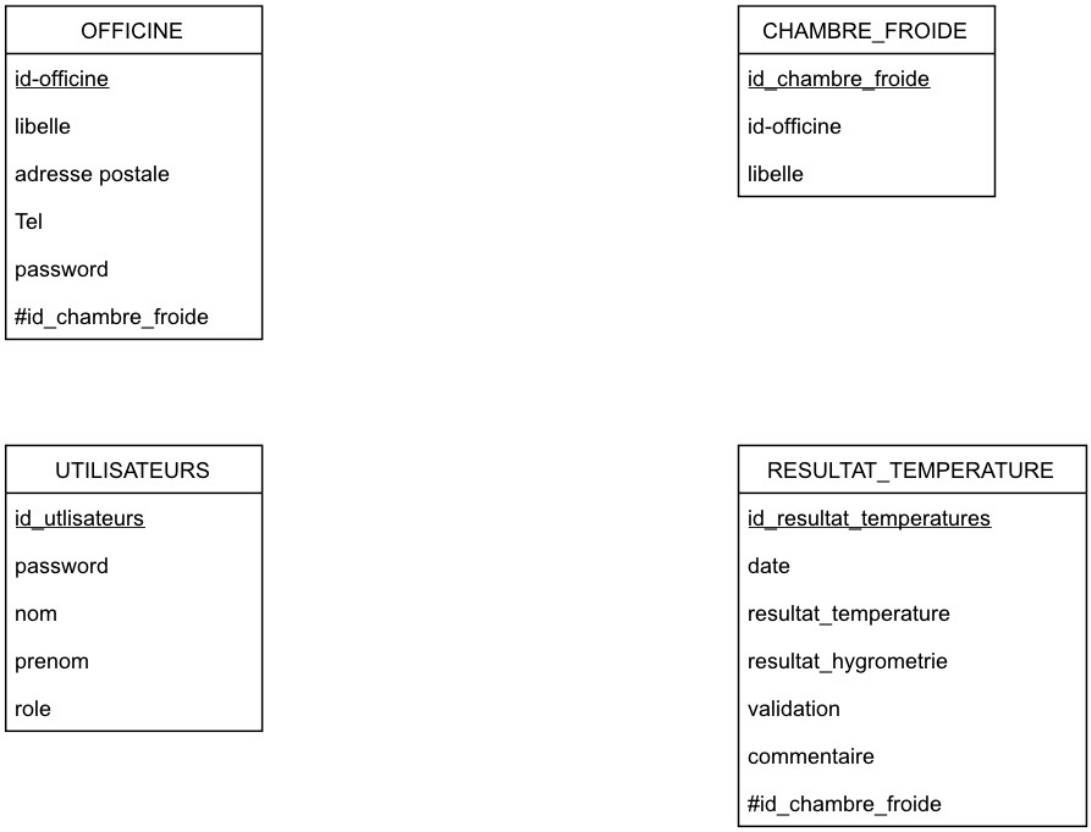
DOCUMENTATION TECHNIQUE

B. Diagramme de Classe

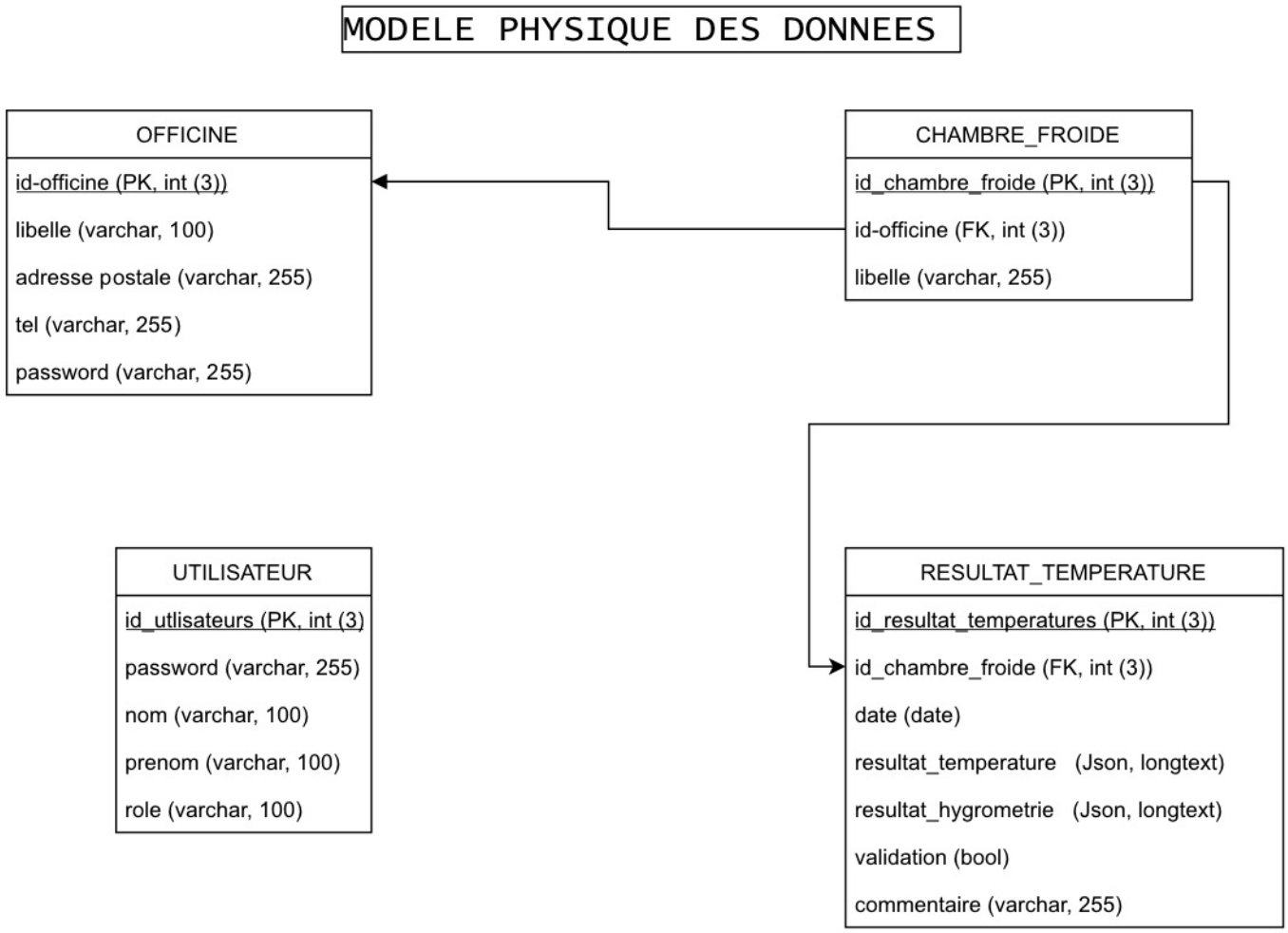
MODELE CONCEPTUEL DES DONNEES



MODELE LOGIQUE DES DONNEES



B. Diagramme de Classe (suite)



B. Pratiques de sécurité mises en place

1. hashage du mot de passe

J'ai décidé de hasher le mot de passe entre le moment où le mot de passe était saisi et celui où il est envoyé vers l'api pour être enregistré en base de donnée. Comme indiqué en ligne 14, 15 et 16 de cet extrait de code, je charge les variables nécessaires en utilisant les fonctions que possède la librairie Bcrypt.

Ensuite, lors de l'exécution de la méthode POST pour envoyer les données vers l'API, je hash à ce moment le mot de passe que j'avais préalabelement stocker dans une variable au fur et à mesure de sa saisie grâce à la gestion de state que fourni React.js. Il n'apparaïtra jamais de manière clair pendant tout le processus de création.

```
14  const axios = require("axios");
15  const bcrypt = require("bcryptjs");
16  const salt = bcrypt.genSaltSync(10);
17
18  function CreateOfficine() {
19    const center = { display: "flex", justifyContent: "center" };
20    const [nom, setNom] = useState("");
21    const [prenom, setPrenom] = useState("");
22    const [password, setPassword] = useState("");
23    const [createDone, setDreateDone] = useState(false);
24
25    const create = () => {
26      if (nom !== "" && prenom !== "" && password !== "") {
27        axios.defaults.headers.common["Authorization"] = "Bearer " + jwt;
28        axios({
29          method: "post",
30          url: url.utilisateurs,
31          data: {
32            nom: nom,
33            prenom: prenom,
34            password: bcrypt.hashSync(password, salt),
35            role: roles.technicien,
36          },
37        })
```

2. sécurisatioin de l'API

Comme indiqué également dans cet extrait de code à la ligne 27, toutes les requêtes vers l'API sont autorisées grâce au token JWT qu'on ajoute dans le header de ces mêmes requêtes. Ce token a été délivré par Symfony (et le bundle Lexik jwt-authentication-bundle. On peut alors délivrer à ce token entre autre un temps de validité précise (attention car même si l'utilisateur n'existe plus son jeton délivrera encore un accès valide). Il nous permet, en plus de valider l'accès à notre API, de vérifier qui se connecte à notre API.