Documentation technique Arcadia

Réflexion initiale

Pour ce projet Arcadia j'ai décidé d'utiliser le Framework Symfony car c'est le Framework sur lequel j'ai appris pendant ma formation. Et il a l'avantage de pouvoir gérer facilement l'aspect admin que ce soit la création de CRUD ou bien son Dashboard et même tout l'aspect sécurité grâce a ses nombreux bundles (Back-end). Et j'ai pu réaliser ma partie Front-end en même temps grâce au template Twig.

Environnement de travail

Environnement :

-PHP >= 8.1

-MYSQL >= 5.7 (WAMP)

-Symfony 5.8.16

-Bootstrap 5.3

-Easyadmin 4.10 VSCode

Extensions VSCODE:

-PHP Intelephense

-Twig Language 2

-DotENV

-phpfmt PHP formatter

-Material Icon Theme

-PHP Getter & Setter

-PHP Namespace Resolver

-YAML

-PHPDoc Comment Prettier - Code formatter

Installation du projet Arcadia :

Avoir défini php dans les variables d'environnement et avoir Xdebug d'activé.

Installer composer

Installer symfony cli

Vérifier les requirements :

symfony check:requirement

Créer un nouveau projet symfony

symfony new my_project –webapp

Relancer l'indexation dans VSCODE : CTRL+MAJ+P -> Intelephense: index

Lancer un serveur symfony server:start

Installer webpack Encore 2.1:

composer require symfony/webpack-encore-bundle

npm install

Pour compiler en dev

npm run dev

Pour compiler en prod

npm run build

Pour compiler en temps réel les changements

npm run watch

Webpack Encore : SASS et Bootstrap:

npm install sass-loader sass --save-dev

Modifier webpack.config.js et ajouter .enableSassLoader()

npm install bootstrap --save-dev

Webpack Encore : Images

npm install file-loader --save-dev

Modifier webpack.config.js et ajouter

```
.copyFiles({
    from: './assets/images',
    to: 'images/[path][name].[ext]',
    })
```

Easyadmin:

Installation du bundle easyadmin

composer require easycorp/easyadmin-bundle

Installation admin dashboard

php bin/console make:admin:dashboard

Vich/uploader-bundle

Installation vich/uploader composer require vich/uploader-bundle

Modifier le fichier config/packages/vich_uploader.yaml mappings: zoo: uri_prefix: /images/nom_dossier upload_destination: '%kernel.project_dir%/public/images/zoo' namer: Vich\UploaderBundle\Naming\SmartUniqueNamer

Modifier l'entité

use Vich\UploaderBundle\Mapping\Annotation as Vich;

#[Vich\Uploadable]

#[Vich\UploadableField(mapping: 'nom_dossier', fileNameProperty: 'imageName', size: 'imageSize')] private ?File $imageFile = null;

```
#[ORM\Column(nullable: true)]
private ?string $imageName = null;

#[ORM\Column(nullable: true)]
private ?int $imageSize = null;

#[ORM\Column(nullable: true)]
private ?\DateTimeImmutable $updatedAt = null;

 /**
 * If manually uploading a file (i.e. not using Symfony Form) ensure an instance
 * of 'UploadedFile' is injected into this setter to trigger the update. If this
 * bundle's configuration parameter 'inject_on_load' is set to 'true' this setter
 * must be able to accept an instance of 'File' as the bundle will inject one here
 * during Doctrine hydration.
 *
 * @param File|\Symfony\Component\HttpFoundation\File\UploadedFile|null $imageFile
 */
public function setImageFile(?File $imageFile = null): void
{
    $this->imageFile = $imageFile;
```

```
    if (null !== $imageFile) {
        // It is required that at least one field changes if you are using doctrine
        // otherwise the event listeners won't be called and the file is lost
        $this->updatedAt = new \DateTimeImmutable();
    }
}

public function getImageFile(): ?File
{
    return $this->imageFile;
}

public function setImageName(?string $imageName): void
{
    $this->imageName = $imageName;
}

public function getImageName(): ?string
{
    return $this->imageName;
}

public function setImageSize(?int $imageSize): void
```

```
{
    $this->imageSize = $imageSize;
}


public function getImageSize(): ?int
{
    return $this->imageSize;
}
```

Modèle conceptuel de donnée

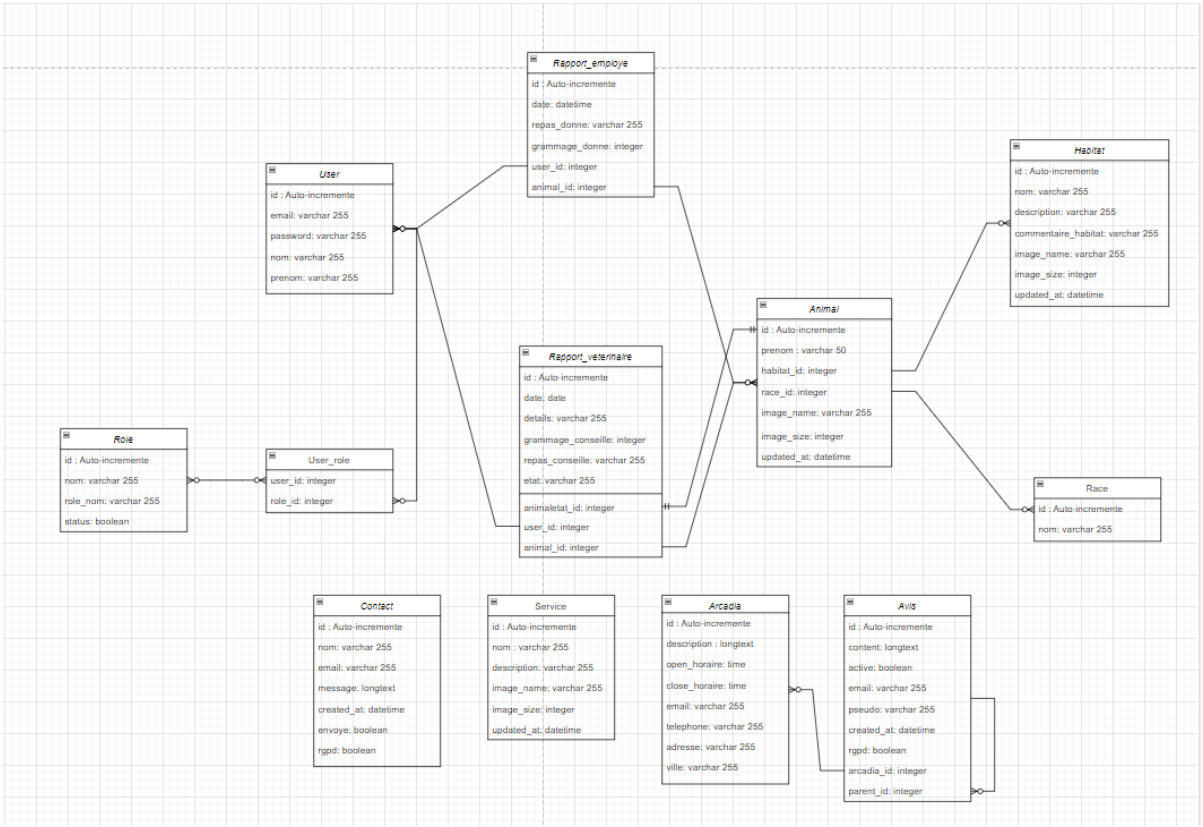
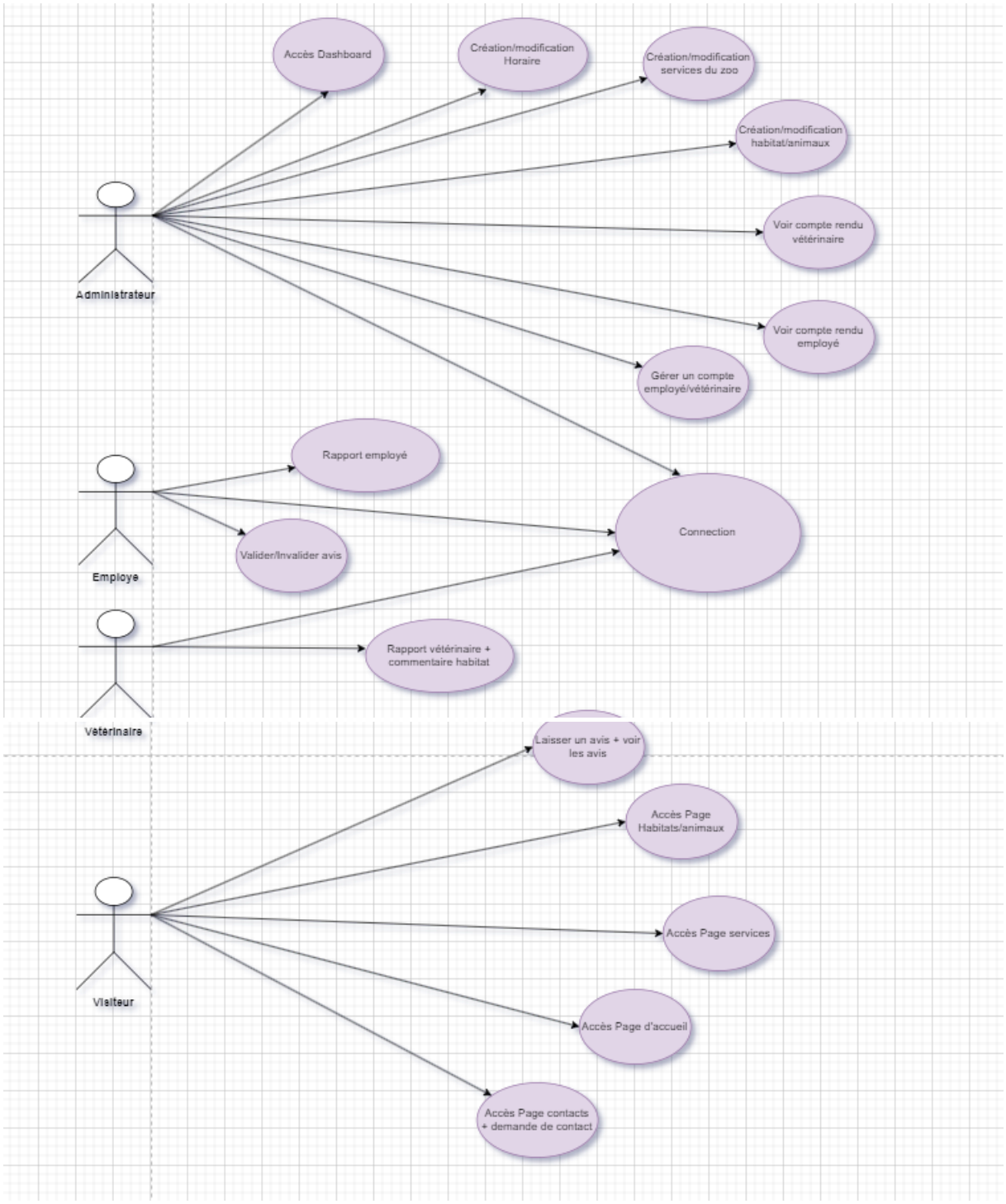


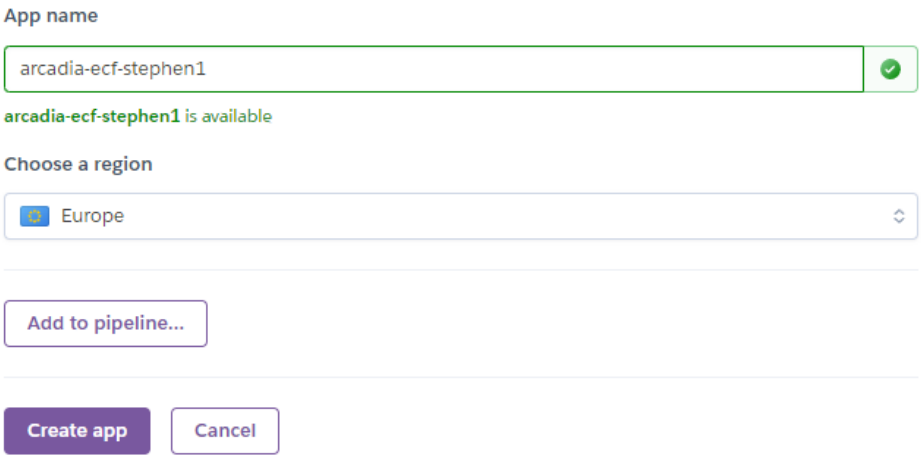
Diagramme de cas d’utilisation



Déploiement

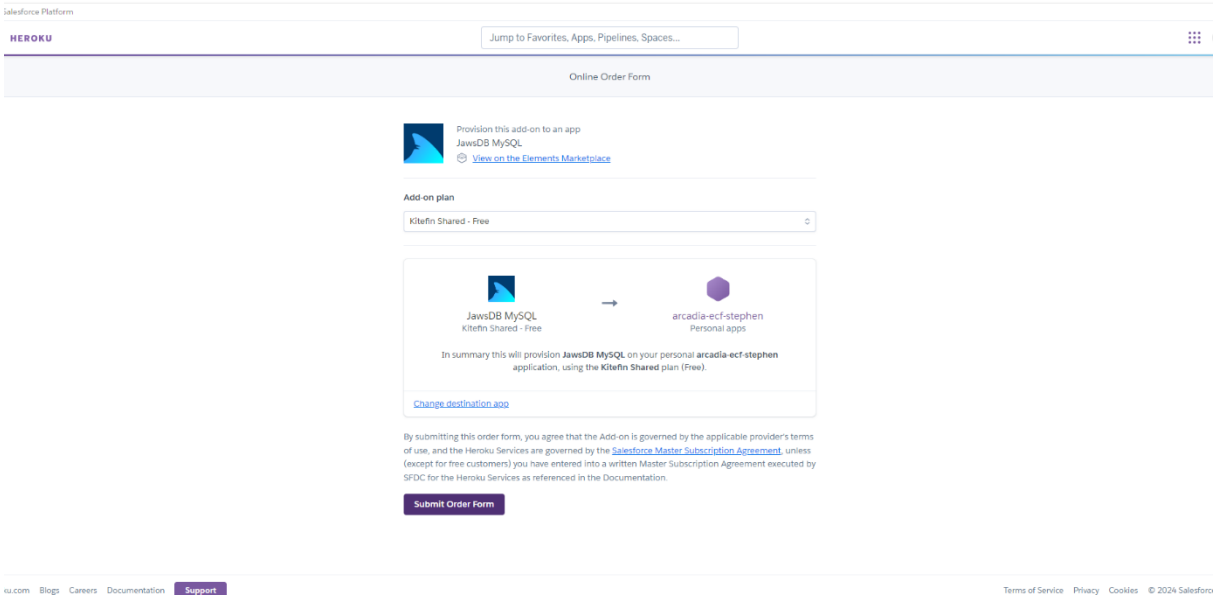
Pour le déploiement j'ai fait mon déploiement avec Heroku.

Après avoir crée mon compte et m'être connecté j'ai pu crée mon application sur heroku.

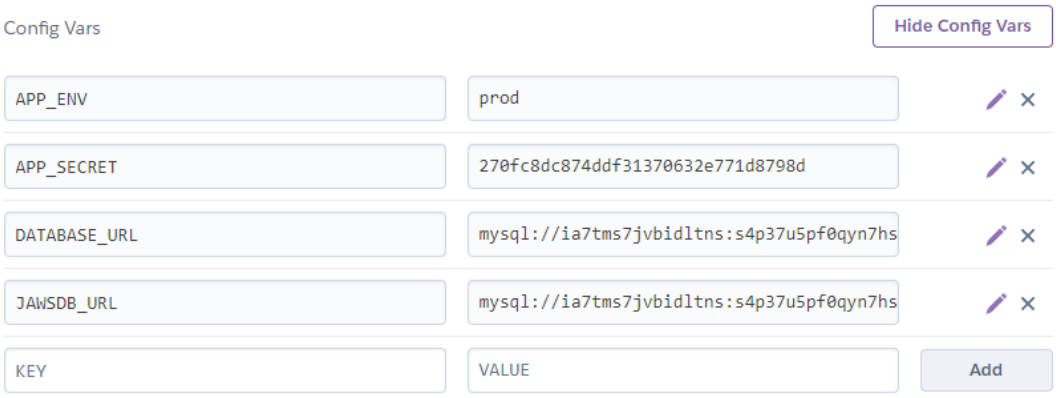


J'ai lié mon compte github à mon application créée sur le site d'heroku.

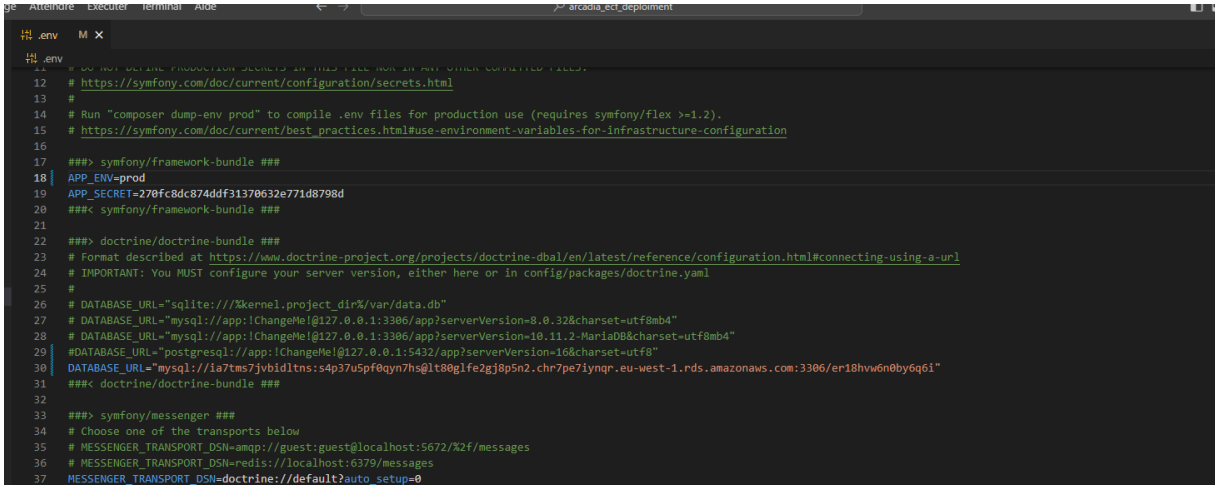
Installé une base de données



Configurer mes variables

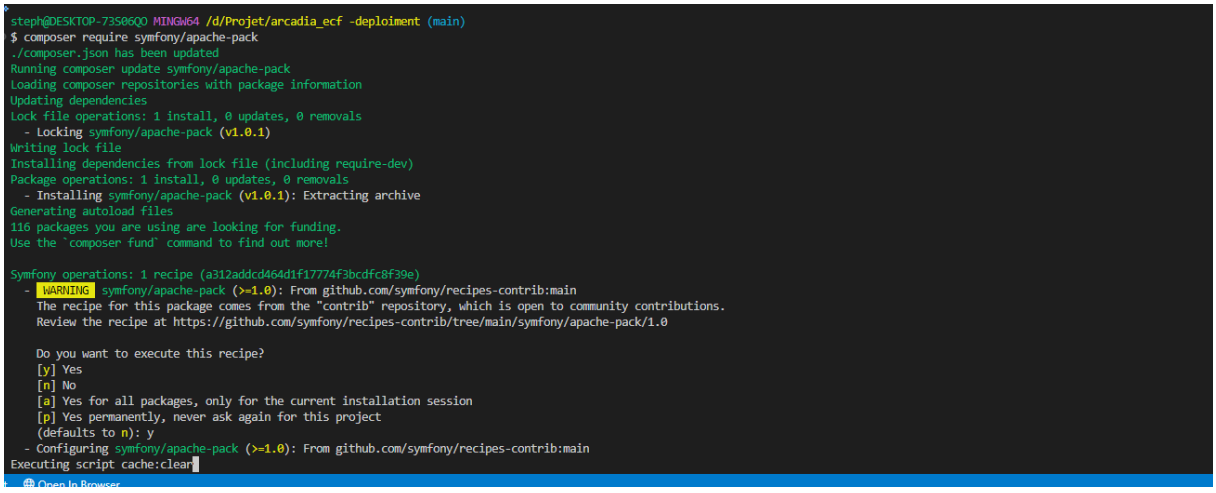


J’ai configuré ces mêmes variables dans mon fichier .env de mon projet.

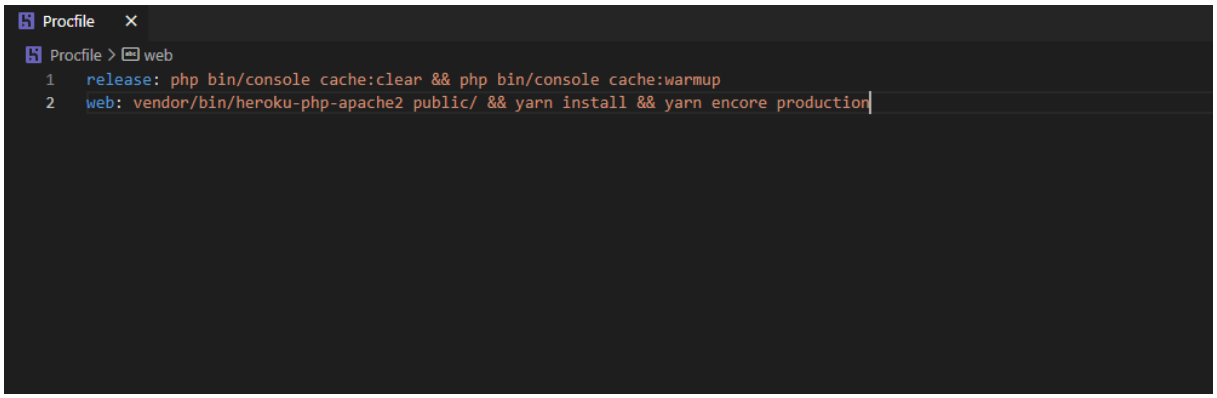


J’ai créé ensuite un fichier .htaccess avec la commande :

Composer require symfony/apache-pack.



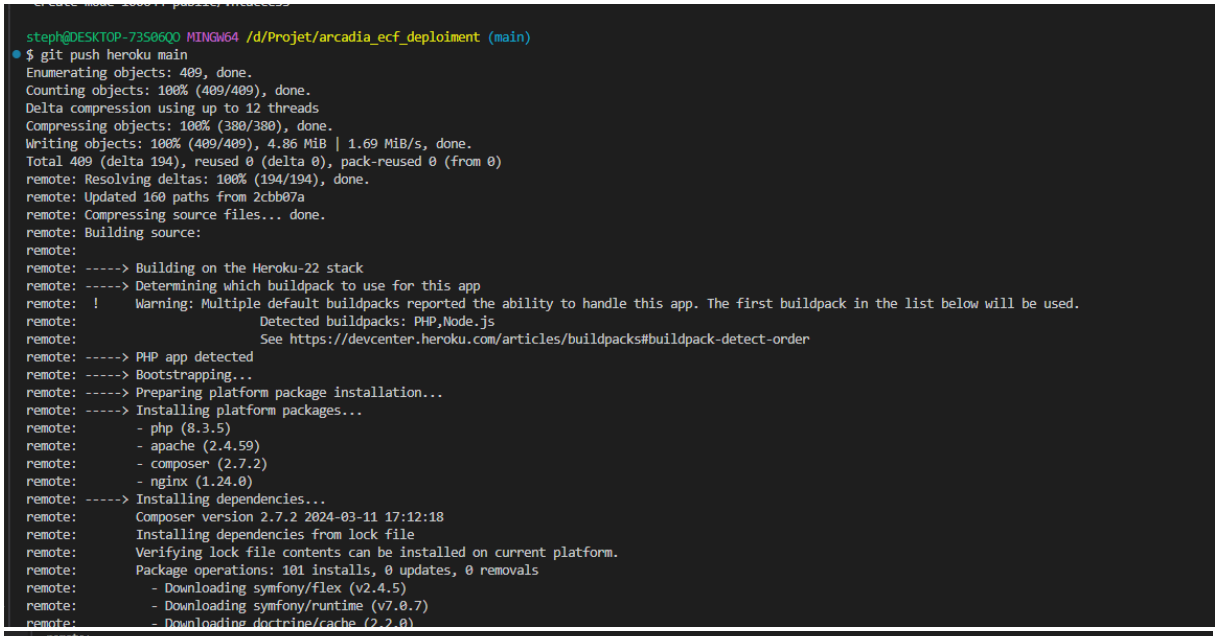
Puis le fichier Procfile.

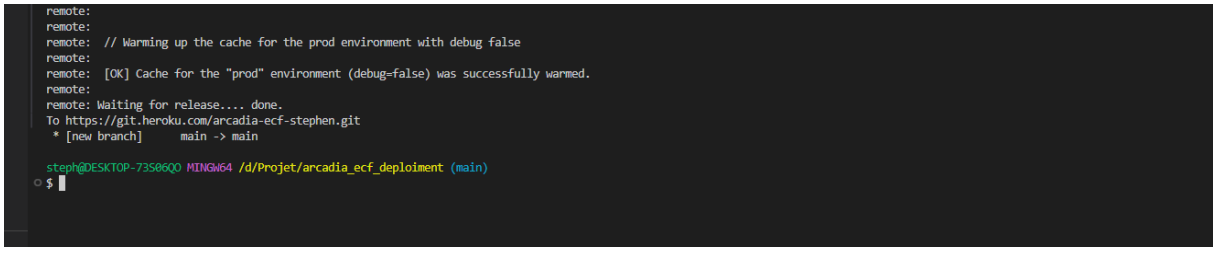


Une fois ce processus effectué j'ai fait une merge de mon dossier mis a jours sur la branch main de mon github.

J'ai lié mon compte github à mon application créée sur le site d'heroku.

Et j'ai pu faire la commande « git push heroku main » pour push mon repository git sur heroku.

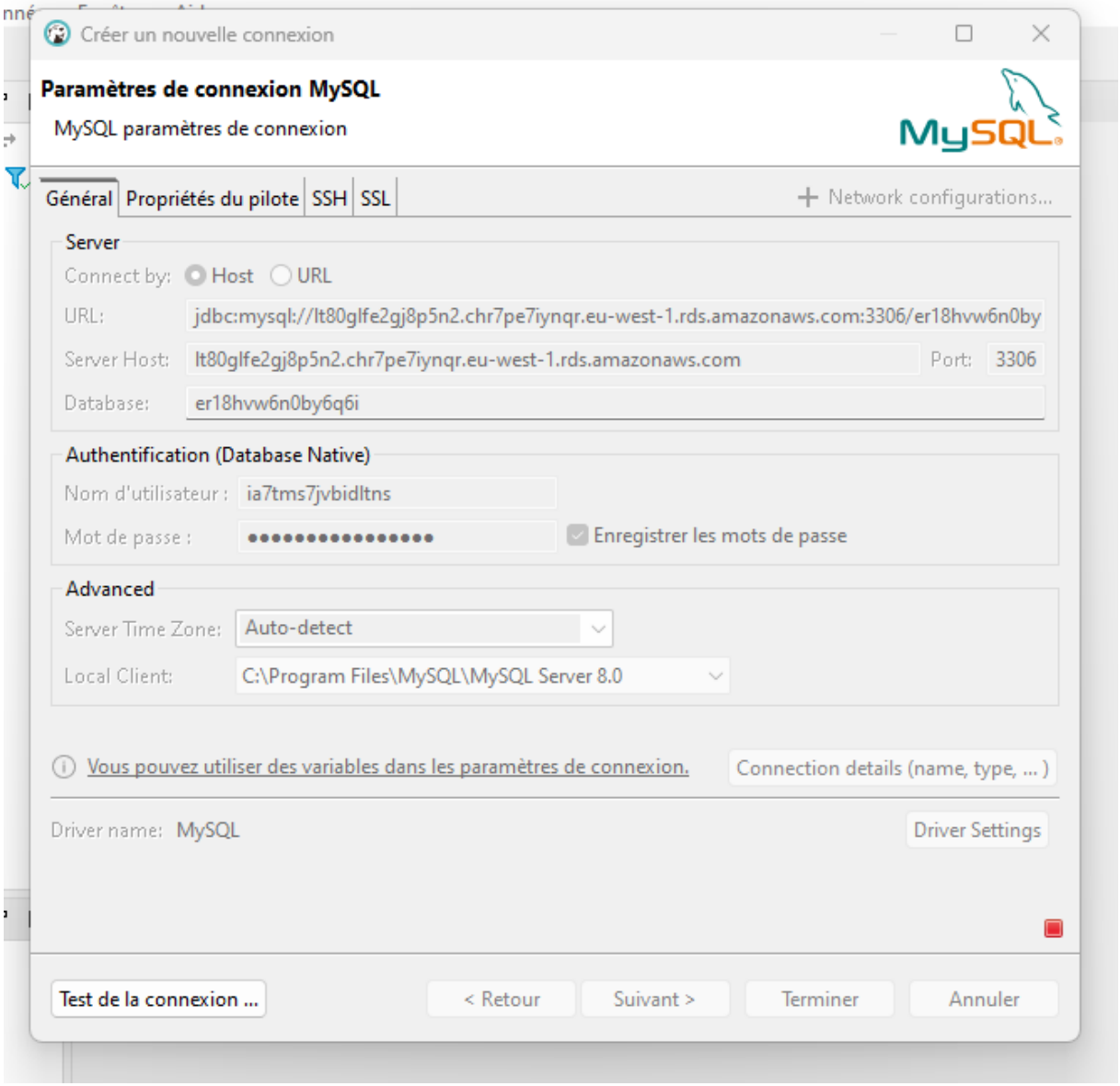




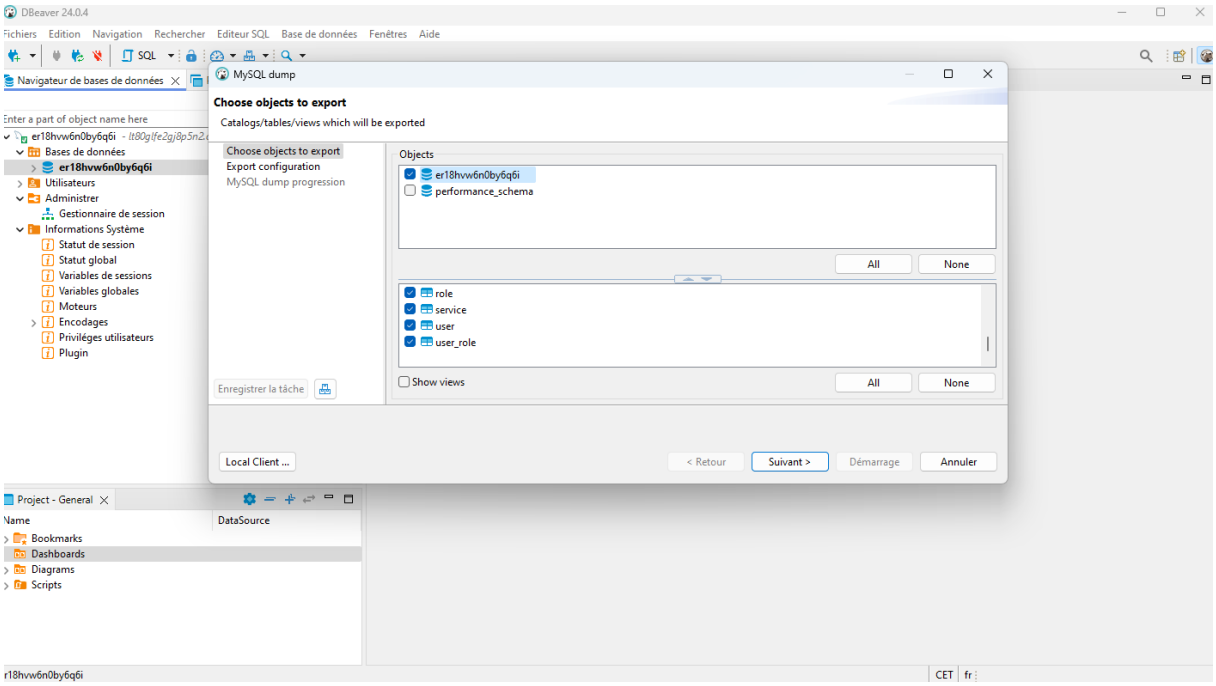
Mon application est en ligne !

Pour gérer la base de données distante afin de pouvoir créer un administrateur j'ai utilisé le logiciel DBeaver.

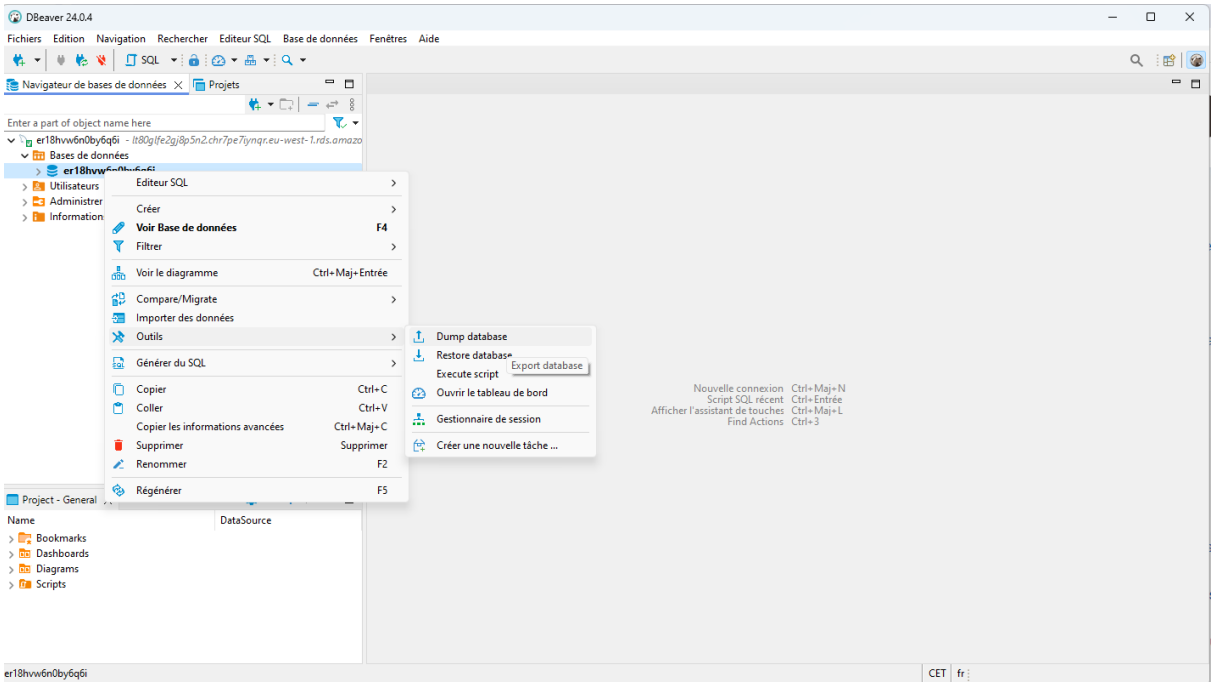
Je me suis connecté à la base grâce aux informations récupéré sur heroku.



J’ai exporté ma base de données



J’ai fait un dump de ma base pour avoir mes tables.



Puis j'ai pu intégrer toutes les données.