Dossier de projet

Titre Professionnel CDA

Projets présentés – BddCompare – Web

DarkDominion – Mobile

Candidat CDA - Rio CLEMENT

Sommaire

Table des matières

Liste des compétences du référentiel

1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

Maquetter une application.

Développer une interface utilisateur de type desktop.

Développer des composants d'accès aux données.

Développer la partie front-end d'une interface utilisateur web.

Développer la partie back-end d'une interface utilisateur web.

2. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité

Concevoir une base de données.

Mettre en place une base de données.

Développer des composants dans le langage d'une base de données.

3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement.

Concevoir une application.

Développer des composants métier.

Construire une application organisée en couches.

Développer une application mobile.

Préparer et exécuter les plans de tests d'une application.

Préparer et exécuter le déploiement d'une application.

Résumé du projet BddCompare en anglais

BddCompare was a project requested by Softia which is a french company at « Le Kremlin-Bicêtre ».
BddCompare's goal is to make database comparisons easier. With it you can access the desired database comparisons via an Internet URL, without having to install database management software in the user's environment.

To achieve this task, the application retrieves the connection lines to connect to the databases of the desired machines; in a principle of "Single Page Application" (SPA); all the accomplishment of these tasks are carried out on a single page.

The user, through the application, can:

- Store the connection strings,

- View the schematics / tables and data of each of the machines in a range of 25 recovered data (recovered data changeable according to the machine RAM)

- Compare the data and create an Excel of this comparison.

This application in a security context should be handled with care, the only people who should have the right to connect to this application must have an administrator role, therefore there is an authentication system implemented with the application to avoid leakage of sensitive data. It's best to use this application in a LAN / PAN configuration for the company that use it.

LAN = Local Area Network

PAN = Personal Area Network

I- Description du cahier des charges du projet BddCompare

I-1.Contexte

Le projet BddCompare est un projet qui a été demandé par la société Softia. Ce projet a pour but de réaliser une comparaison des données simplifiées sans avoir à interagir avec un logiciel de bases de données qui induit en général la nécessité de copier les données d'une machine à une autre.

La société Softia a été créée il y a de cela 20 ans par Alain RAKOTONANAHARY. En 2007, Softia a fini par déménager à Kremlin-Bicêtre suite à son passage de startup à PME. En 2015, il fait son premier million d'euros de chiffre d'affaires et enfin, en 2020, il atteint la barre des 30 collaborateurs. Softia Ingénierie est une ESN, la société est sur deux secteurs d'activité qui sont le FORFAIT et la RÉGIE.

Le projet a été fait à la demande du ministère de l'éducation Français qui est le client qu'on référera tout au long du projet.

I-2.Fonctionnalités principales de BddCompare

- Création de comptes clients
- Gestion de l'identification et des paramètres par cookies de sessions
- Création de lignes de connections postgresql vers les bases de données de plusieurs machines
- Partitionnement des données récupérées sur une base de données ciblé
- Comparaison des données ciblées
- Création d'un Excel des données ciblées

I-3.Technologies, Outils

Outils de maquetting : Figma, Canva

Canva servait pour donner des idées générales de pages de manière très simplifié. Nous avons une personne de l'équipe préférant l'utiliser plutôt que figma ce qui fait que nous devions l'utiliser tout de même.

Figma a servi principalement à créer des maquettes visuelles et éventuellement préparer l'aspect mobile du site. De plus avec figma nous avons créé des composants visuels important pour le site.

Front-end: Bootstrap5, css, express js

Bootstrap5 propose une très large librairie css qui a permis d'accélérer la production et la mise en place des pages pour le site. Express js est un Framework javascript avec lequel on a intégré le bootstrap 5.

Back-end: Javascript, express js, mongoDb, pgsql

Javascript et express js ont une très large proposition de librairies qu'on a utilisé pour mettre en place les communications des bases de données nécessaires à l'application.

Tests unitaires : Javascript-Mangoose -Mocha

Pour les tests unitaires nous avons utilisés des scripts qui testent nos fonctions et contrôleurs dans notre code et nous avons aussi utilisés Mocha pour compléter.

Tests fonctionnels : CodeceptJS

Pour les tests fonctionnels de type IHM CodeceptJS était le choix le plus simple pour représenter un utilisateur en machine via sélénium.

Logiciels gestion du cadre de travail : Gitlab – (ticketing via SCRUM)

Les pushs et la réalisation des projets se faisait à travers plusieurs branches parallèles à la main lié à des « issues » vérifiées par le développeur sénior une fois push avant d'être merge sur la main. La réalisation des « issues » se faisait principalement autour de réunions où on discuter des fonctionnalités et leurs réalisations pour le projet BddCompare.

I-4.Cadre de travail

Program Increment Planning de 8 semaines (40 jours)

Nous avions que 40 jours pour produire une application opérationnelle avec des indications très changeantes de la cliente où il fallait être « agile » par rapport à sa demande. Le chef de projet a donc décidé de partir sur un Program Increment Planning pour effectuer le projet. A chaque début de semaine le lundi et le mardi nous faisions des comptes rendus de la semaine, des objectifs et issues à réaliser avant les deadlines courtes pour chaque semaine. Pendant ces deux jours on devait aussi réaliser une étude des demandes de la cliente, une réflexion sur la veille technologique et une réflexion sur la sécurité. Nous devions regarder régulièrement le site de l'owasp https://owasp.org/ et les mises à jour du langage et framework javascript https://developer.mozilla.org/fr/docs/Web/JavaScript.
L'un des points importants dans toutes ces recherches était la praticabilité de l'application. Il fallait que l'application soit avant tout pratique pour la cliente car la sécurité serait principalement gérée par le réseau interne de l'entreprise de la cliente. Mais nous devions tout de même regarder s'il y avait des possibilités de récupérer ou d'insérer du code malicieux à l'intérieur des inputs utilisées pour l'application.

Tout l'objectif du PIPlanning était de
1 : planifier le lundi et le mardi des objectifs à réaliser en communiquant avec la cliente,
2 : Mettre en œuvre le plan avant la fin de la semaine,
3 : Evaluer la solution mis en œuvre vers la fin de semaine entre jeudi et vendredi,
4 : Ajuster en inspectant et modifiant la solution mis en œuvre le vendredi avant de recommencer la semaine.

Il fallait tenir ce rythme pendant les 40 jours jusqu'à la finalisation de l'application.
Et au bout de ces 40 jours nous avons réalisés la totalité des demandes de la cliente en demande de design et de fonctionnalités. Nous lui avons ajouté par rapport à ses demandes des tests unitaires et fonctionnelles.

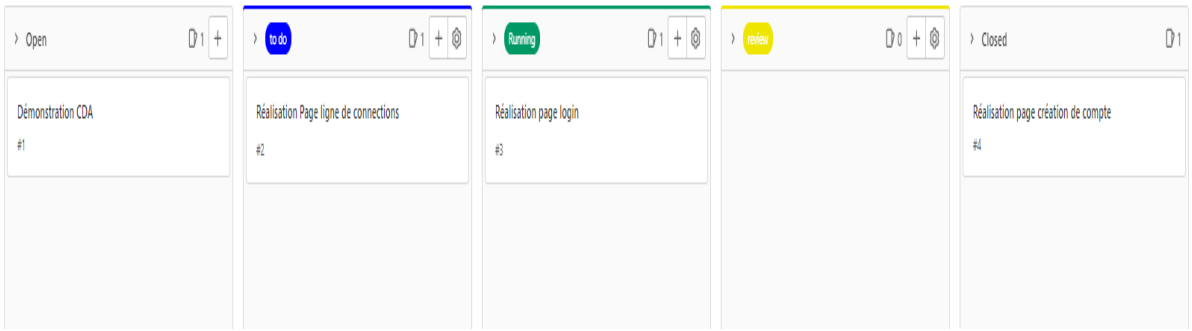
Le projet aura rapporté environ 12 000€ à l'entreprise.

II- Spécification fonctionnelles et techniques du projet BddCompare

II-1. Organisation des demandes de la cliente via les issues Gitlab

Afin de préparer le projet par rapport au cahier des charges le développeur senior a décidé d'organiser les tâches et les divisés dans des issues gitlab qui peuvent par la suite se changer en des branches qui partiront de la main pour y développer les fonctionnalitées demandées.

Voici un exemple de l'organisation de board de l'entreprise simplifié :



Nous avons les tickets open qui correspond aux demandes de la cliente sur toute l'application.
Les tickets to do qui ont été mis pour la semaine avec les deadlines correspondantes à récuperer et à réaliser pour les développeurs.
Les tickets running avec le développeur assigné pour signaler aux autres développeurs que la fonctionnalité et actuellement prise en charge.
Les ticket review pour signaler que le ticket a été réalisé et doit être vérifié par le développeur senior pour vérifier s'il n'y a aucune régression et si le ticket peut être mergé.

Les tickets closed pour signaler que le ticket est fini.

Nous devions réaliser ceci toutes les semaines tout le long du projet, chaque issues générées une branche avec son nom afin de faciliter sa gestion pour le développeur senior et le chef de projet.

La convention de nommage suivantes a été appliquée pour le code :

- Noms des classes, variables, méthodes en anglais
- Nom des classes, variables, méthodes en Camel Case
- Commentaires en anglais

En base de données :

- Les tables et noms de champs définir en anglais
- Les noms de colonnes en Camel Case
- Toutes les tables sont en minuscule

Le projet prend la forme d'un MVC (Modèle-Vue-Contrôleur).

- Le modèle (Model) : c'est la représentation des données de l'application. Il peut s'agir de données stockées dans une base de données ou de données dynamiques récupérées via des API.

- La vue (View) : c'est la partie visible de l'application, l'interface utilisateur. Elle est construite à partir des modèles de données et permet aux utilisateurs d'interagir avec l'application.
- Le contrôleur (Controller) : c'est le cerveau de l'application, il gère les interactions entre la vue et le modèle. Il écoute les événements déclenchés par l'utilisateur dans la vue, interagit avec le modèle pour récupérer ou mettre à jour les données, et met à jour la vue en conséquence.

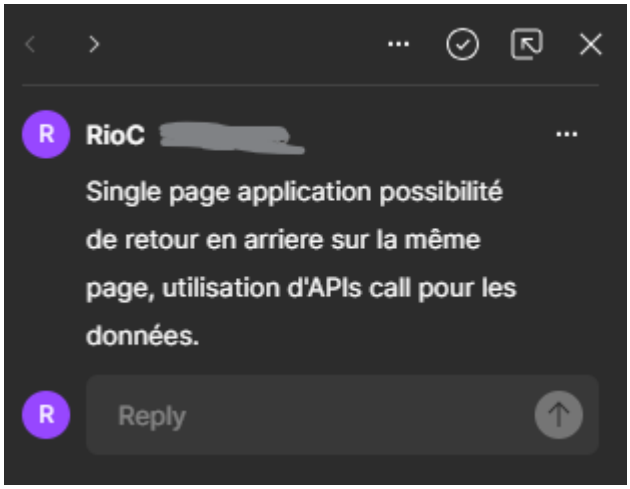
Ces trois parties sont représentées respectivement par les services, les composants et les directives:

- Les services (Model) : ils fournissent les données à l'application. Ils peuvent se connecter à une base de données ou à une API pour récupérer les données.
- Les composants (View) : ils sont responsables de la présentation des données à l'utilisateur. Ils récupèrent les données depuis les services et les présentent dans la vue.
- Les directives (Controller) : elles permettent de lier les composants et les services entre eux. Elles réagissent aux événements déclenchés par l'utilisateur dans la vue et interagissent avec les services pour récupérer ou mettre à jour les données

Le modèle MVC a été convenu pour la partie création de compte et création de lignes de connexion. Afin de simplifier l'interface nous avons ensuite décidé de créer une SPA qui regroupera divers événements javascripts stockant les variables nécessaires afin de rester sur la même page et traiter des requêtes api et modifier la page selon les données récupérées. Ces évenements javascript ont été réaliser en AJAX pure sans Jquery, afin d'éviter le téléchargement d'une librairie qui aurait pu alourdir le projet et le temps des requêtes.

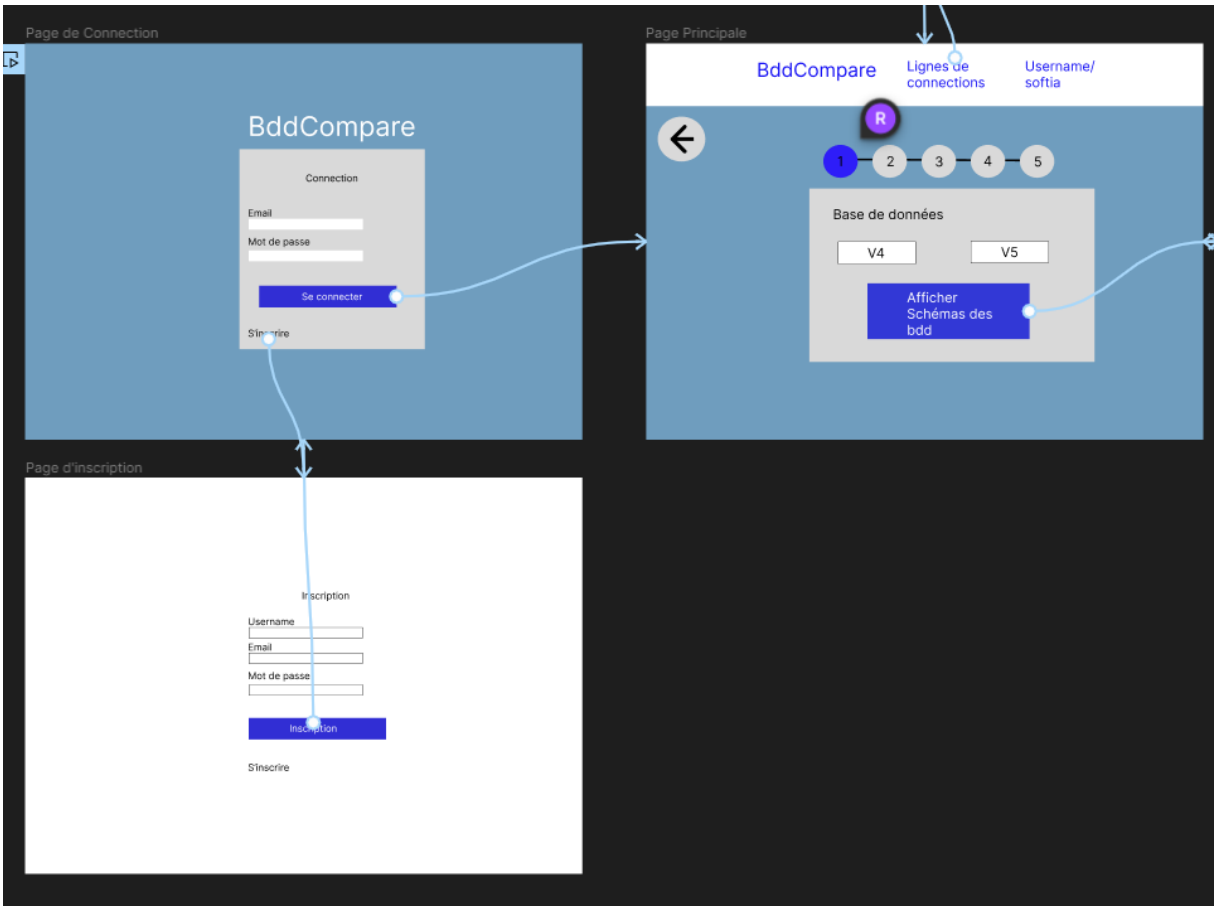
II-2. Maquetting de l'application via figma

Design finale de la maquette figma :

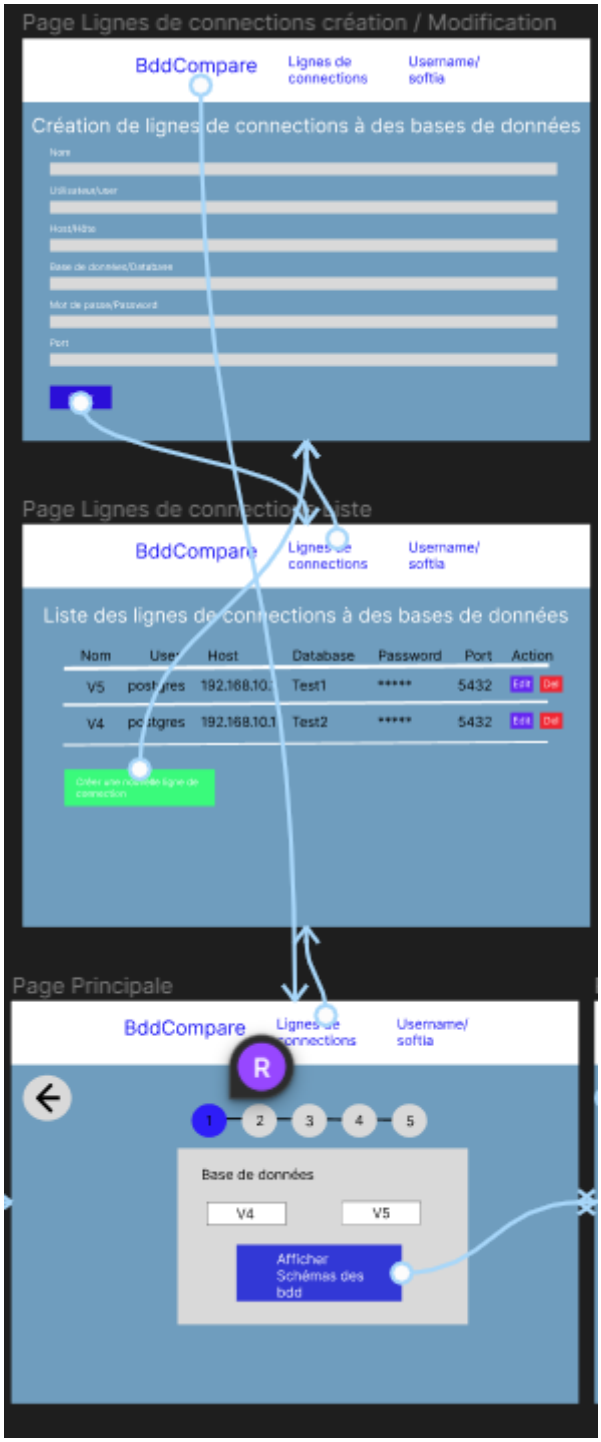
Pour réaliser l'application, l'équipe autour du projet BddCompare a principalement utilisé figma, comme on peut le voir sur le schéma ci-dessus, on a un design prototype qui aidera dans la réalisation de l'application et pour la finalisation du design finale. Les flèches indiquent les transitions entre les pages et il faut aussi prendre en compte les commentaires qui ont une forme de bulle avec des lettres  . Souvent le dev senior détaille un peu plus ce qu'il faut réaliser avec la page ou un élément en particulier. A savoir ici par exemple vu que c'est une « SPA » soit une single page application pour la page principale. Ici il fallait que toutes les réalisations des étapes se fassent via des call apis. 

Figma est un logiciel très intéressant pour sa rapidité de création de design / components / parcours utilisateur, si on suit l'application bddCompare quand on lance la présentation de l'application on aura une version demo avec les pages présentes.

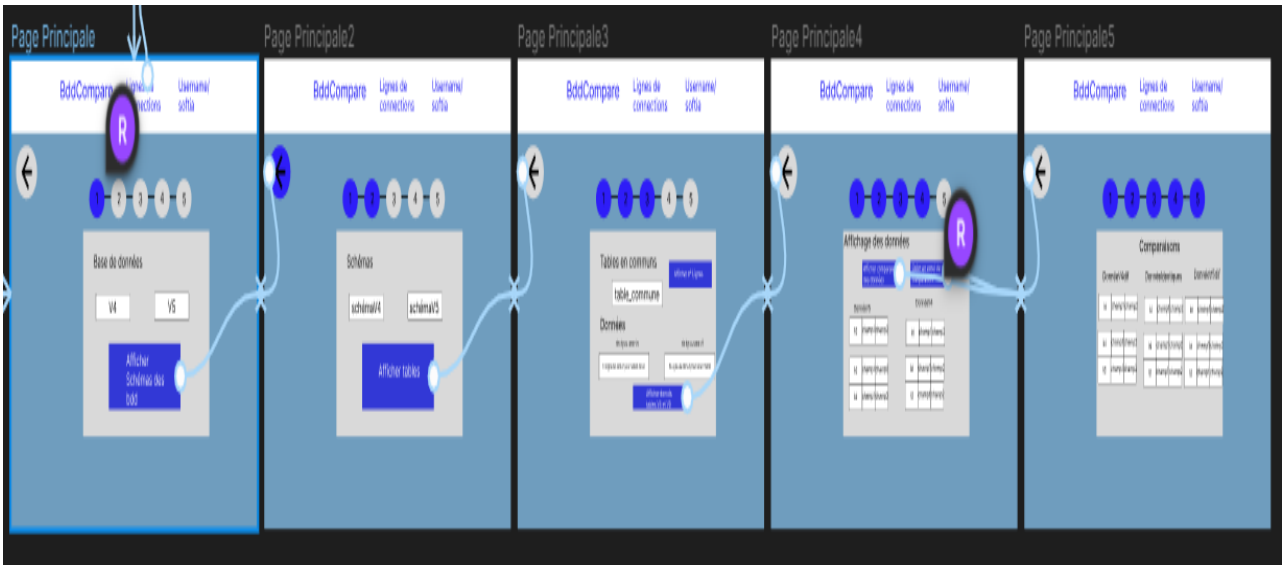
Via le figma on peut voir que le parcours utilisateur ressemble à peu prés ceci on commence par une page de connections qui propose une inscription simple et une connexion après entrée de l'email et d'un mot de passe.



Le figma démontre la demande de la création et modification de lignes de connections dans d’autres pages



Il démontre aussi l’importance de devoir suivre une single page application dans ces 5 pages détaillées avec des commentaires pour diriger les développeurs.



II-3. Démonstration des fonctionnalités présentes

A – Création de comptes clients et gestion de l'identification des sessions

Pour mettre en place les demandes techniques de la cliente nous nous sommes dirigées sur le Framework expressJS pour avoir une technologie qui propose des requêtes asynchrones plus dirigées vers le web. Le développeur senior a choisi le Framework express js car le squelette d'express js est très léger et permet une meilleure flexibilité et créativité quant à le deadline donné. Le souhait du développeur sénior aurait été d'utilisé react si nous avions plus de temps afin de permettre une éventuelle application mobile dans le futur.

Technologie Framework : 

Pour suivre ce Framework nous avons pris pour la création de comptes et la création de lignes de connections MongoDB qui est géré par librairie mangoose dans javascript et express js.

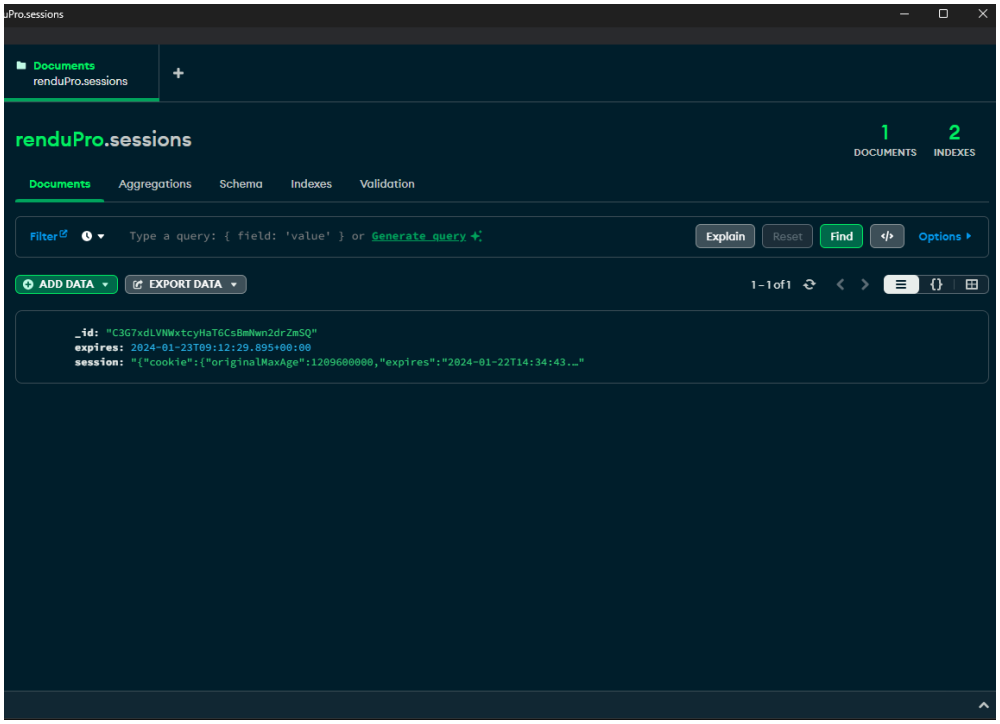
Nous avons choisi mongoDB pour ces deux éléments car la cliente ne savait pas si elle voulait ajouter un rôle au compte, ou si elle voulait ajouter d'autres éléments par rapport à la création de lignes de connections. MongoDB étant une base de données NoSQL nous permettait d'avoir la flexibilité et la rapidité dans la génération des données quand aux demandes changeantes de la cliente par rapport à la base de données.

Pour mieux illustrer les demandes de création du compte client, regardons-le model pour un utilisateur :

BddCompare > database > models > user.model.js > ...

```
// librairie javascript pour l'utilisation de mongoDB
const mongoose = require('mongoose');
// librairie pour le cryptage en Bcrypt
const bcrypt = require('bcrypt');
const schema = mongoose.Schema;

const userSchema = schema({
    // Stockage local pour la création d'un cookie de session
    local: {
        email: {type : String, unique: true, required:true } ,
        password: { type: String, required: true },
    },
    username: String
});

// cryptage du mot de passe
userSchema.statics.hashPassword = async (password) => {
    try{
        const salt = await bcrypt.genSalt(10);
        return bcrypt.hash(password, salt);
    }catch(e){
        throw e
    }
};

userSchema.methods.comparePassword = function(password) {
    return bcrypt.compare(password, this.local.password);
};

// Initialisation du model pour réutilisation du code
const User = mongoose.model('user', userSchema);

module.exports = User;
```

Sur le morceau de code nous pouvons voir l'ajout des librairies de la ligne 1 à 4 et l'initialisation du schéma à la ligne 5. Nous avons un stockage en local de l'email et password car nous utilisons aussi la libraire « passport » qui fonctionne avec express générant des cookies de session qui expire au bout d'une certaine durée et qui sont stockées en base de données pour donner un suivi des connections à l'éventuel admin de l'application. Les cookies de session aident à identifier un utilisateur principalement par son id.



En matière de sécurité nous avons préféré utiliser le bcrypt plutôt que le sha256 pour ramener une couche de sécurité au détriment d'un peu de rapidité, vu qu'il n'y a pas énormément de connections qui sera faite dans l'application on est parti sur bcrypt car la perte de temps pour le hachage bcrypt du mot de passe est plutôt minime quant à la sécurité qu'elle procure par rapport au sha256.

Cette fonctionnalité est testé et prise en charge par les documents présent dans les répertoires « codeceptjs/ » et « BddCompare/test/user.test.js ».

```
async function createVerifyDeleteUser(email, password, username) {
    mongoose.connect('mongodb://rio:dossierPro@127.0.0.1:27017/renduPro', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    }).then( async function() {
        try {
            await deleteUser(email);
            // Créez un utilisateur
            await createUser(email, password, username);

            // Vérifiez le mot de passe
            await verifyPassword(email, password);

            // Supprimez l'utilisateur
            await deleteUser(email);
          } catch (error) {
            console.error('Erreur lors de l\'exécution des fonctions :', error.message);
          } finally {
            exit();
          }
        }).catch( err => {
            console.log("fail");
            console.log(err);
    });

    // await mongoose.disconnect();
}
```

Dans le document user.test.js Nous avons créé des fonctions qui nous donne des prompts sur la création, la vérification du hachage du mot de passe sous bcrypt et la suppression de cette donnée par la même occasion.
Il y a de plus des tests fonctionnels effectués via codecept.js qui viennent tester la fonctionnalité de création de compte et de login.

```
async function createUser(email, password, username) {
    try {
        // Vérifiez si l'utilisateur existe déjà
        // await bddConect();
        const existingUser = await User.findOne({ 'local.email': email });
        if (existingUser) {
        console.log('Cet utilisateur existe déjà.');
        return;
        }

        // Créez un nouvel utilisateur
        const hashedPassword = await User.hashPassword(password);
        const newUser = new User({
        local: {
            email,
            password: hashedPassword,
        },
        username,
        });

        await newUser.save();
        console.log('Utilisateur créé avec succès.');
    } catch (error) {
        console.error('Erreur lors de la création de l\'utilisateur :', error.message);
    } finally {
        // mongoose.disconnect();
        return // Fermez la connexion à la base de données
    }
}
```

Une fonction type createUser ressemble à peu près ceci où on va tester les fonctions présentes de l'application dans un script qui respectera un scénario donné.

Les tests fonctionnelles et unitaires étant un ajout non demandé par la cliente ils n'ont pas été réalisé sur les autres fonctionnalités pour ne pas retarder la réalisation du run très court pour le déploiement de l'application (environ 40 jours).

Ici on voit le déroulement d'un scenario de création de compte et de login de la feature « user » testés. Il va premièrement sur la page puis ensuite clique sur les éléments de la page afin de simuler le comportement d'un utilisateur.

codeceptjs > tests > creation_test.js > ...

```
Feature('user');

Scenario('creation_de_compte', async ({I}) => {
  I.amOnPage("http://localhost:3000/");
  I.click('#yourUsername');
  I.click('.card-body');
  I.click('Créer un compte');
  I.click('/html/body/div/div/div/form/div[1]/input');
  I.fillField('/html/body/div/div/div/form/div[1]/input', 'testIt');
  I.click('/html/body/div/div/div/form/div[2]/input');
  I.fillField('/html/body/div/div/div/form/div[2]/input', 'testtest');
  I.click('/html/body/div/div/div/form/div[3]/input');
  I.fillField('/html/body/div/div/div/form/div[3]/input', 'testtest@gmail.com');
  I.click('.btn');
});
```

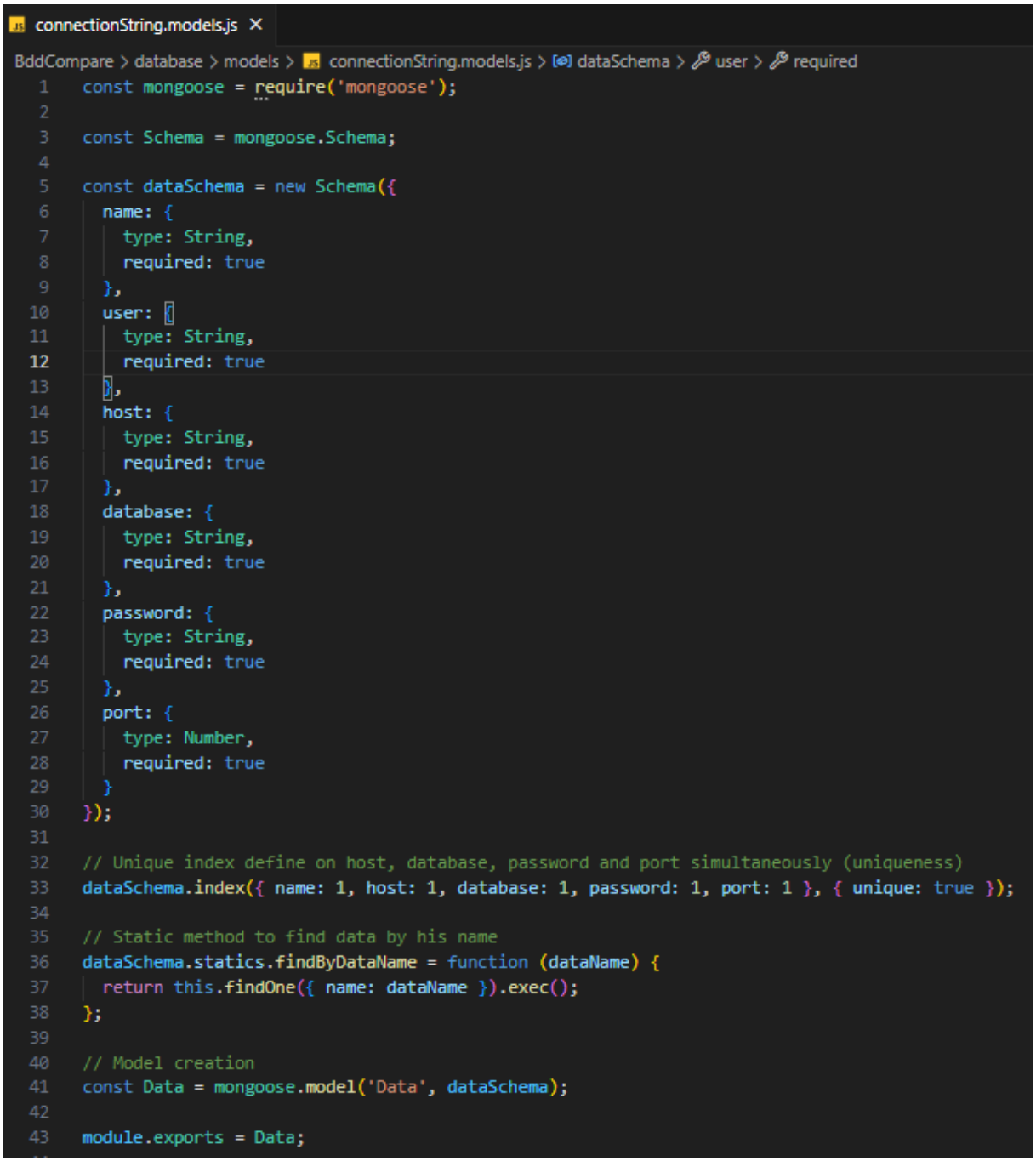
codeceptjs > tests > login_test.js > ...

```
Feature('user');

Scenario('login', async ({I}) => {
    I.amOnPage("http://localhost:3000/");
    I.click('//*[@id="yourUsername"]');
    I.fillField('//*[@id="yourUsername"]', 'testtest@gmail.com');
    I.click('//*[@id="yourPassword"]');
    I.fillField('//*[@id="yourPassword"]', 'testtest');
    I.click('/html/body/main/div/section/div/div/div/div[2]/div/form/div[4]/button');
});
```

B – Création de lignes de connections postgresql

L'application doit créer une interaction entre deux machines qui disposent toutes les deux postgresql et qui ont les ports ouverts pour l'utilisation de leur base de données pgsql respectives.

Les informations des lignes de connections sont stockées en base de données via un model mangoose appelé « connectionString.js ». Ceci est semblable à la création de comptes, tous les éléments sauf port sont récupérés sous forme de string, et pour éviter de commettre l'erreur d'écrire deux fois la même ligne de connections nous avons ajouté un comportement d'index à l'ensemble des éléments.



Le model est organisé de tel manière à pouvoir être utilisé par la librairie pg est à changer chacune des requêtes qui seront semblable à des appels d'API afin de récupérer les données.

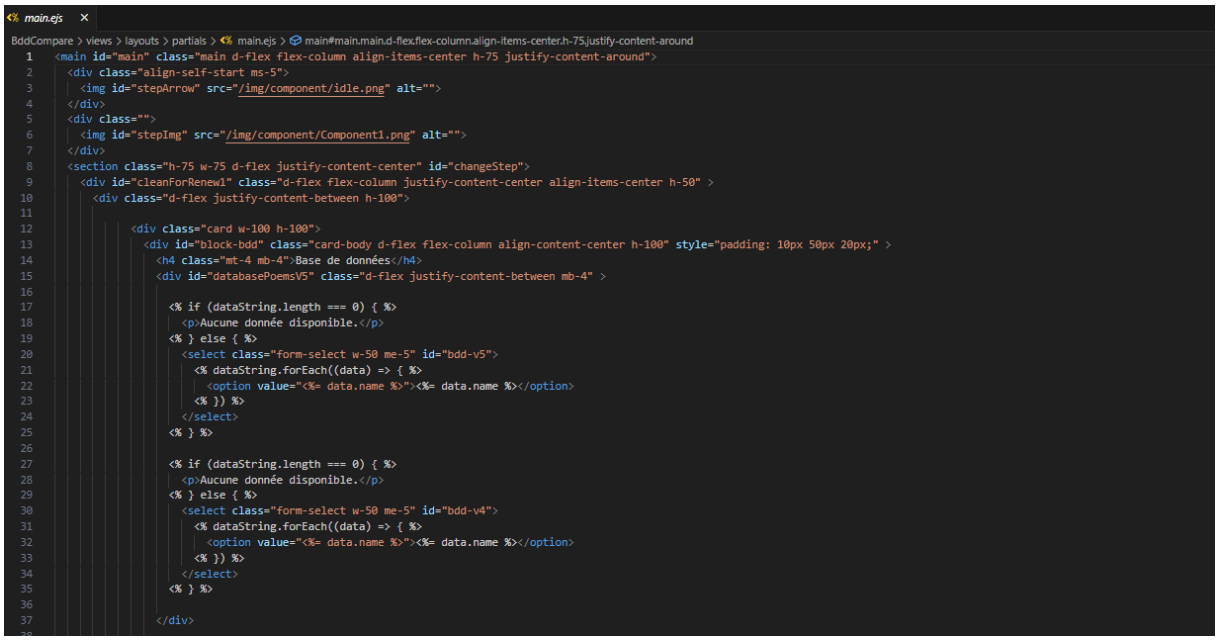
schemaV4.routes.js

BddCompare › routers › schemas › schemaV4.routes.js › router.post('/schemasV4') callback

```
const functionMods = require('../../private/functions/functionsMods');
const express = require('express');
const router = express.Router();

const { Pool } = require('pg');
const ejs = require('ejs');

// ----------schémas ---------- //
router.post('/schemasV4', async (req, res) => {
    const data = req.body;
    const poolChoose = await functionMods.getDataByName(data.bdd);
    const poolDecided = new Pool({
      user: poolChoose.user,
      host: poolChoose.host,
      database: poolChoose.database,
      password: poolChoose.password,
      port: poolChoose.port, // le port par défaut pour PostgreSQL est 5432
    });

    const client = await poolDecided.connect();
    try {
      const result = await client.query("SELECT schema_name FROM information_schema.schemata WHERE schema_name NOT LIKE 'pg_%' AND schema_name NOT IN ('public', 'information_schema') ORDER BY schema_name ASC");
      // console.log(res);
      const schemata = result.rows;
      // console.log(schemata);
      poolDecided.end;
      res.send(schemata);

    } catch (err) {
      console.error(err);
      poolDecided.end;
      res.send('Error ' + err);
    }
  });

  module.exports = router;
```

Ici nous pouvons voir l'interaction de la base de données pgsql à partir d'une requête SELECTE avec les données qui aideront à réaliser la fonctionnalité demandée. Sur cette image on a récupéré les schémas des bases de données concerné dans Pool qui aura été récupéré via le « main.ejs ».

Les données envoyées dans la route api sont récupérées via l'interface web à partir de la page « main.ejs » qui aura récupéré les données des lignes de connections au préalable dans les conditions présentes dans le document. (<% dataString.forEach((data)) %> pour les lignes de connections qui seront récupérées).



La page prenant un principe de single page application se transformera au fur et à mesure via des scripts javascript `<%- include('scriptsJS') %>` `<%- include('scriptsBotstr') %>`.

Voici à quoi ressemble un document type servant à l'adaptation de la page par rapport aux requêtes demandées :

getSchemasV4.js M

BddCompare > public > js > schemas > getSchemasV4.js > getSchemaV4

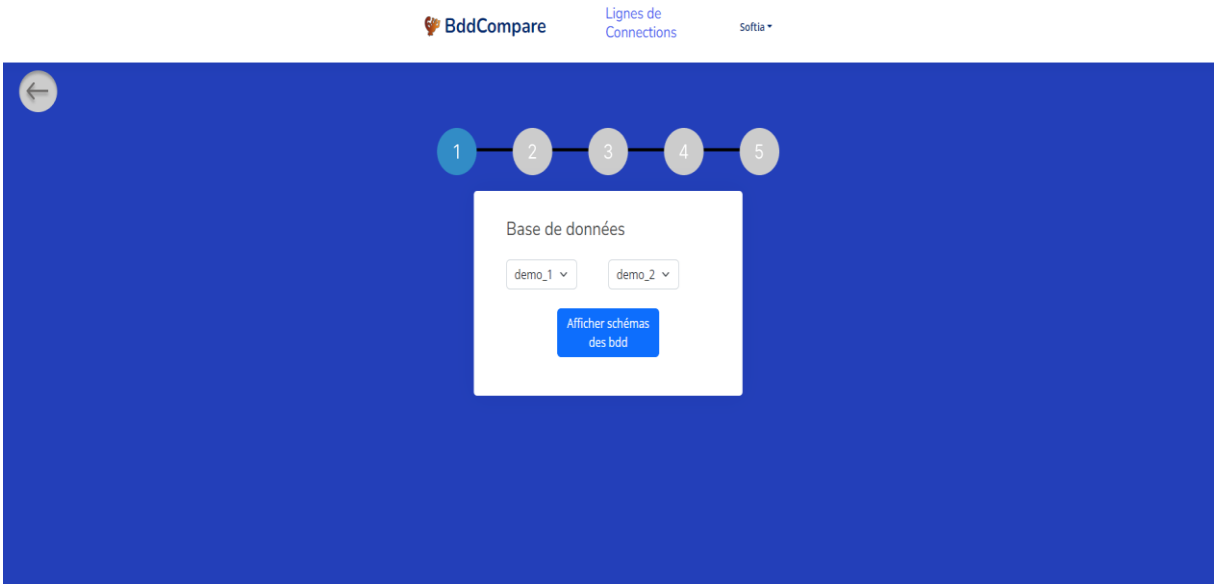
```
const bddV4 = document.getElementsByClassName("displayBddV4")[0];

console.log(keepBddNameV5);

async function getSchemaV4(){

  await getSchemaV5();

    const allSchemaSelect = document.createElement("select");
    allSchemaSelect.id = "allSchemaV4";
    allSchemaSelect.className = 'form-select';
    const divSchemV4 = document.getElementById("schemaV4");
    divSchemV4.appendChild(allSchemaSelect);

    var xhr = new XMLHttpRequest(); // création d'un objet XMLHttpRequest
    const url = "/api/schemasV4";
    const data = {bdd: keepBddNameV4};
    xhr.open("POST", url, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    xhr.onreadystatechange = function() {
      if (this.readyState === 4 && this.status === 200) {
        // fonction appelée lorsque la requête est réussie

        var data = JSON.parse(this.responseText);

        var schemas = data;
        // récupération des données reçues

        var schemasList = document.getElementById('allSchemaV4'); // sélection du conteneur

        schemas.forEach(function(schema) {
          // création d'un élément <li> pour chaque utilisateur
          var option = document.createElement('option');
          option.className = "schema-v4";
          option.value = schema.schema_name;
          option.innerText = schema.schema_name;
          // ajout de l'élément <li> au conteneur
          schemasList.appendChild(option);
        });
      } else if (this.readyState === 4) {
        // fonction appelée en cas d'erreur lors de la requête
        alert('Une erreur est survenue.');
      }
    };

    const jsonData = JSON.stringify(data);

    xhr.send(jsonData);


}

bddV4.addEventListener("click", getSchemaV4);

```

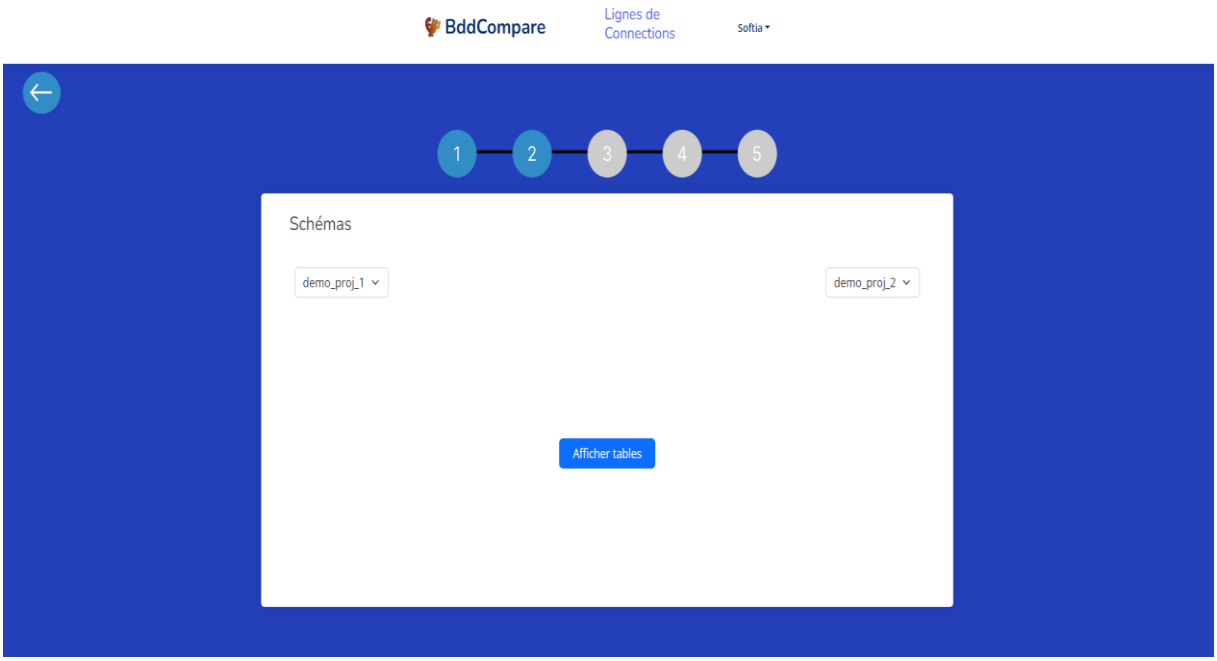
Comme on peut le voir via un appel ajax qui prendra en charge la route API qui contient la requête par rapports aux éléments qu'on a sélectionnées, le script changera l'aspect de la page et ajoutera/modifiera des éléments afin d'afficher les données demandées.

C – Partitionnement des données, comparaison et création d'excel

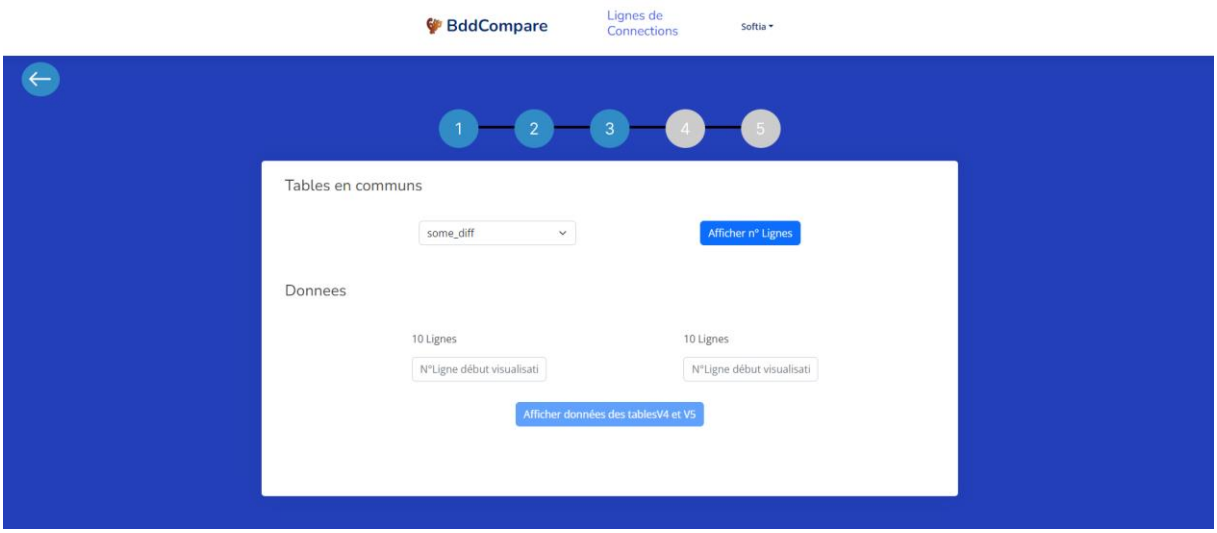
Une fois les données récupérées via les différents appels API de l'application

Il faut donc sélectionnées les deux bases de données que l'on veut comparer et afficher leurs schémas



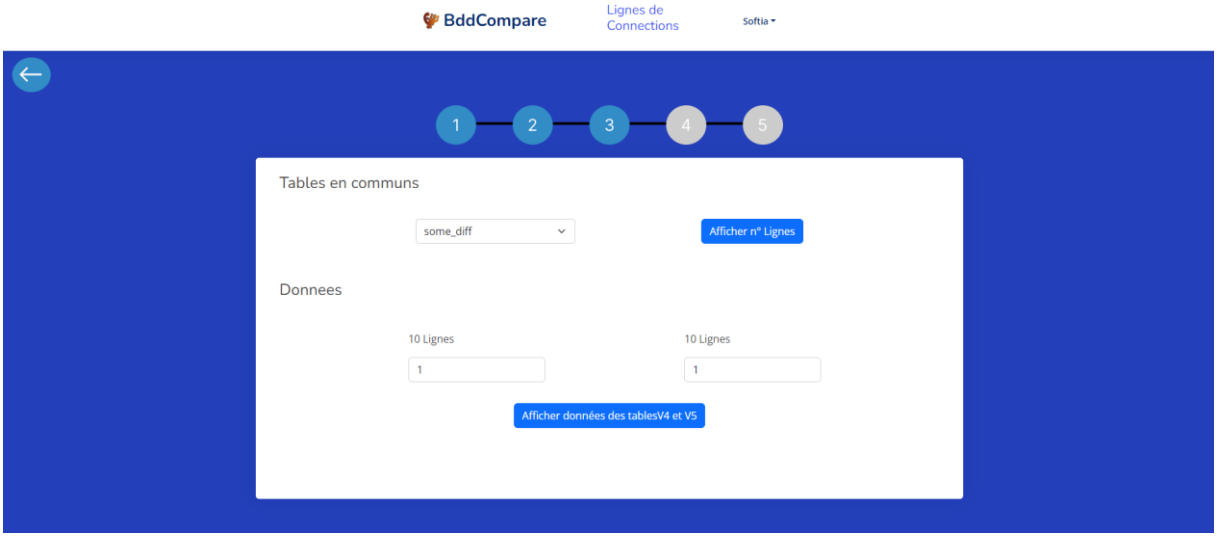


Il faudra continuer le parcours utilisateur jusqu'à ce qu'on arrive au partitionnement des données à afficher.



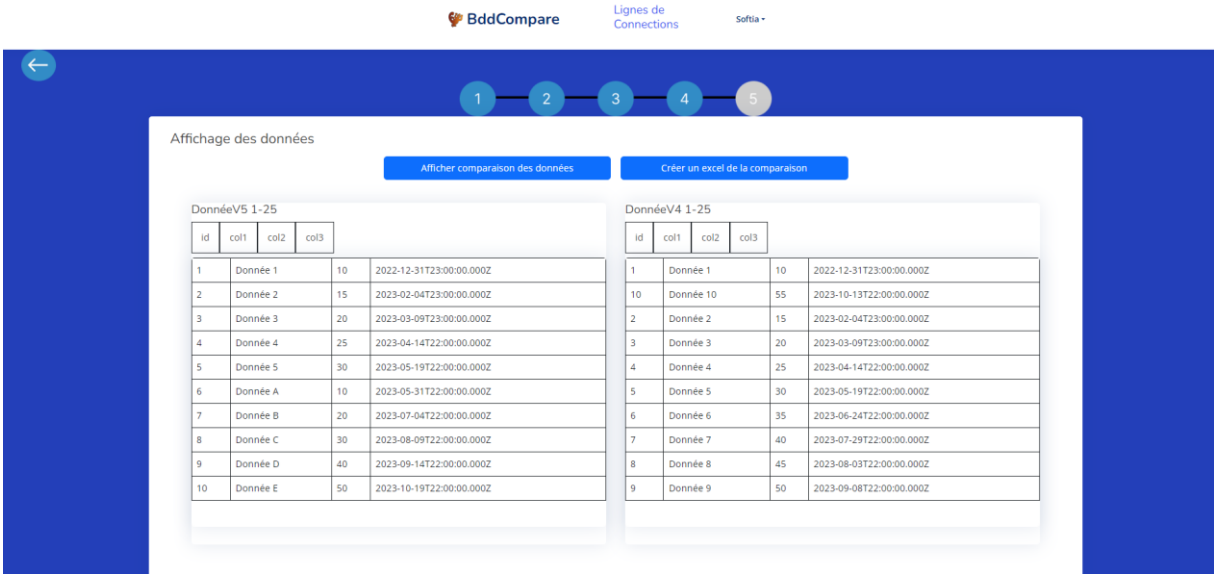
Ici on peut voir les tables en communs entre les deux machines et le nombre de ligne à gauche de la table some_diff de demo_1 et à droite le nombre de lignes de la table some_diff de demo_2.

Une fois que nous avons remplis dans la « N de ligne début de visualtisation » le numéro de ligne à laquelle on veut que l'affichage des données commence on peut cliquer le bouton d'affichage des données.



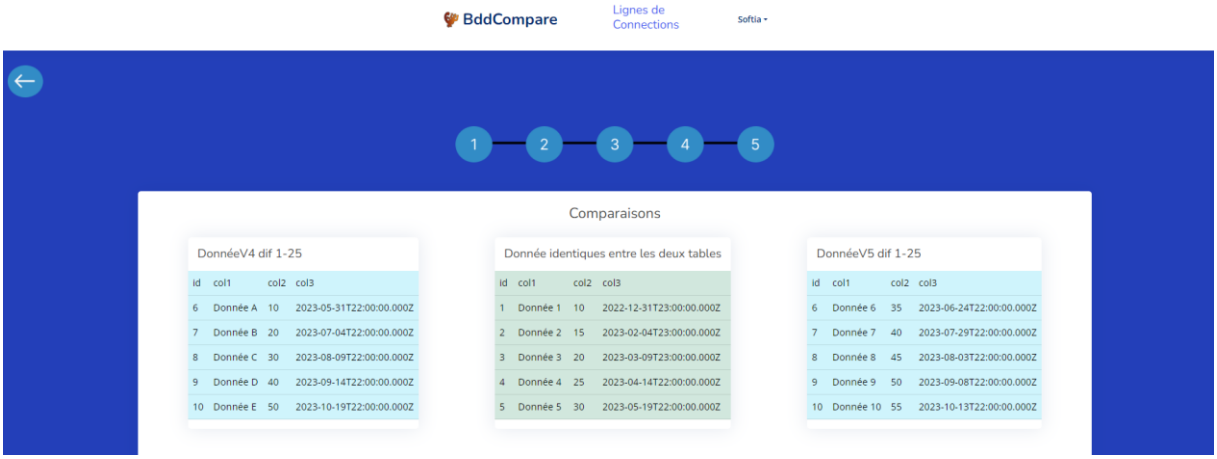
Ici nous avons choisi de commencer à 1.

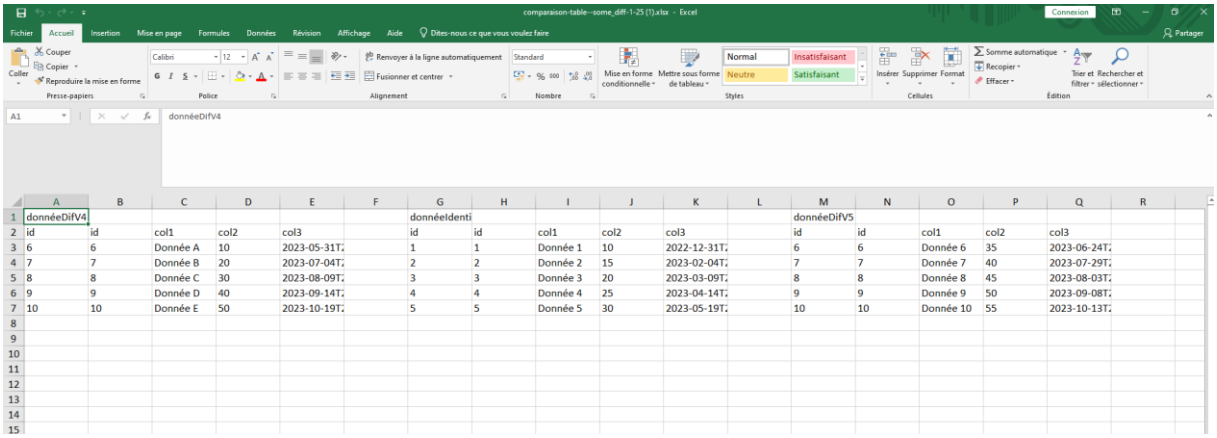
Une fois que nous cliquons sur ce bouton nous obtenons cette page.



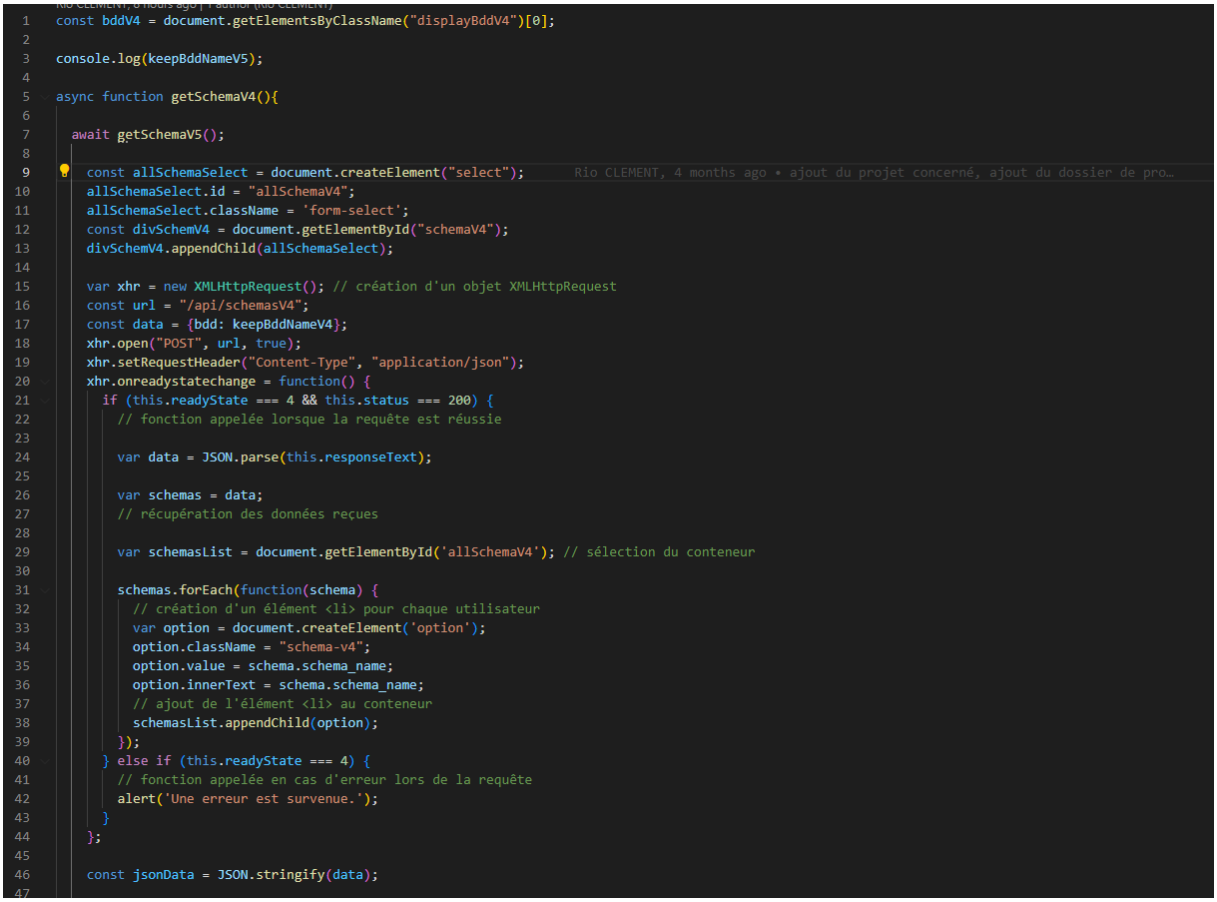
Il sera possible pour l'utilisateur de choisir entre un affichage brut de la comparaison des données ou bien de tout simplement créer un excel de cette comparaison.

Voici le résultat des deux boutons :

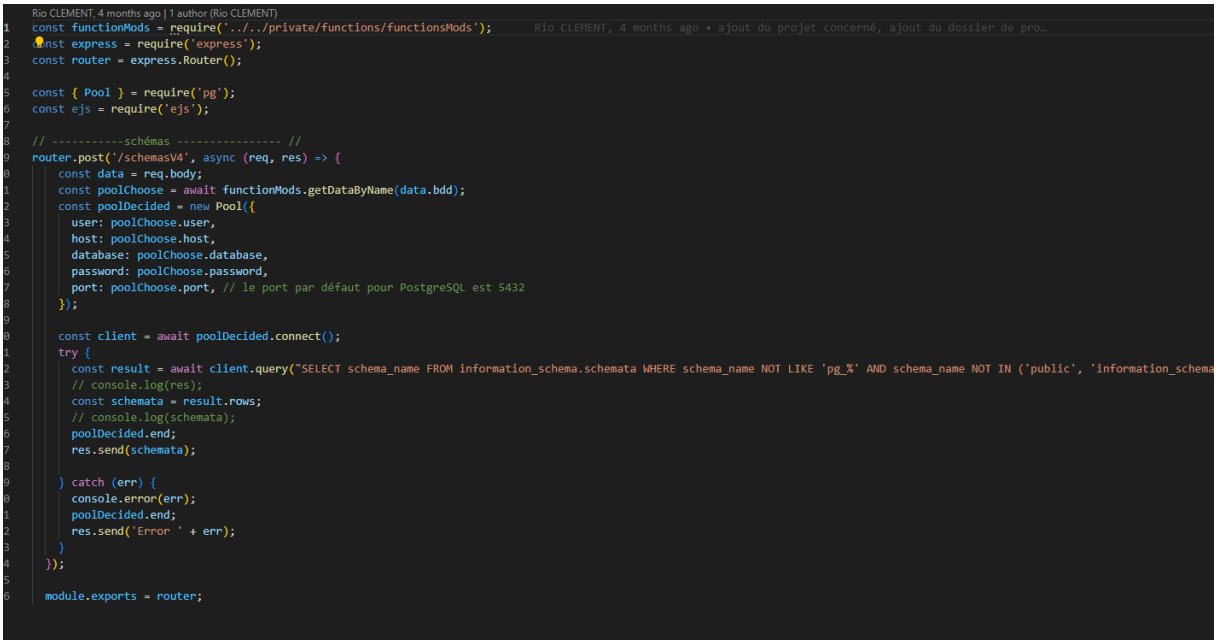




Les données entrées comme on peut le voir sont reliés aux balises selectes de l'application, on a une constante communication avec les routes API via les classes et les id pour aider à la formulation des requêtes pgsql pour pouvoir récupérer les données désirées.



```
const bddV4 = document.getElementsByClassName("displayBddV4")[0];

console.log(keepBddNameV5);

async function getSchemaV4(){

  await getSchemaV5();

    const allSchemaSelect = document.createElement("select");      Rio CLEMENT, 4 months ago • ajout du projet concerné, ajout du dossier de pro…
    allSchemaSelect.id = "allSchemaV4";
    allSchemaSelect.className = 'form-select';
    const divSchemV4 = document.getElementById("schemaV4");
    divSchemV4.appendChild(allSchemaSelect);

    var xhr = new XMLHttpRequest(); // création d'un objet XMLHttpRequest
    const url = "/api/schemasV4";
    const data = {bdd: keepBddNameV4};
    xhr.open("POST", url, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    xhr.onreadystatechange = function() {
      if (this.readyState === 4 && this.status === 200) {
        // fonction appelée lorsque la requête est réussie

        var data = JSON.parse(this.responseText);

        var schemas = data;
        // récupération des données reçues

        var schemasList = document.getElementById('allSchemaV4'); // sélection du conteneur

        schemas.forEach(function(schema) {
          // création d'un élément <li> pour chaque utilisateur
          var option = document.createElement('option');
          option.className = "schema-v4";
          option.value = schema.schema_name;
          option.innerText = schema.schema_name;
          // ajout de l'élément <li> au conteneur
          schemasList.appendChild(option);
        });
      } else if (this.readyState === 4) {
        // fonction appelée en cas d'erreur lors de la requête
        alert('Une erreur est survenue.');
      }
    };

    const jsonData = JSON.stringify(data);
```



```
Rio CLEMENT, 4 months ago | 1 author (Rio CLEMENT)
const functionMods = require('../../private/functions/functionsMods');      Rio CLEMENT, 4 months ago • ajout du projet concerné, ajout du dossier de pro…
const express = require('express');
const router = express.Router();

const { Pool } = require('pg');
const ejs = require('ejs');

// ----------schémas ---------------- //
router.post('/schemasV4', async (req, res) => {
    const data = req.body;
    const poolChoose = await functionMods.getDataByName(data.bdd);
    const poolDecided = new Pool({
      user: poolChoose.user,
      host: poolChoose.host,
      database: poolChoose.database,
      password: poolChoose.password,
      port: poolChoose.port, // le port par défaut pour PostgreSQL est 5432
    });

    const client = await poolDecided.connect();
    try {
      const result = await client.query("SELECT schema_name FROM information_schema.schemata WHERE schema_name NOT LIKE 'pg_%' AND schema_name NOT IN ('public', 'information_schema
      // console.log(res);
      const schemata = result.rows;
      // console.log(schemata);
      poolDecided.end;
      res.send(schemata);

    } catch (err) {
      console.error(err);
      poolDecided.end;
      res.send('Error ' + err);
    }
  });

  module.exports = router;
```

Comme on peut le voir dans cette requête ajax xhr, on envoit les éléments de la classe dans la req.body pour la post '/schémaV4' pour ensuite avoir une modification de la requête en direct.
Ce principe est réutilisé pour les tables et les données pour ensuite les comparés selon leurs chaines de caractères avec un regex qui fera une itération sur les plages de données ciblées.

III- Description de la veille effectuée BddCompare

Pour BddCompare la veille effectuée principale étaient des recherches quand à l'utilisation des fonctionnalités ajax sans jquery https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest et les bonnes pratiques. Jquery étant devenu non recommandé, il fallait utiliser du ajax pur et ce qui a été réalisé pour ce projet.

On peut ajouter que parmi les sites à visiter régulièrement il y avait aussi le site de l'OWASP

https://owasp.org/www-project-top-ten/ par rapport aux dernières failles éventuelles à corriger pour l'application concernant javascript et express js.

Résumé du projet DarkDominion Web/Mobile en anglais

DarkDominion Web/Mobile was a project requested by ETNA which is a french company and school at « Ivry-sur-Seine ».
DarkDominion's goal is to be a simplified version of the very complex game "Villainous".

To achieve this task, the application made simple characters which goal is to either have twenty gold or four monster summons in the board. If the player manage to reach this goal he wins the game. This is little easter egg to the character "Prince Jean in Villainous" whose goal is to make twenty gold in order to win.

The user, through the application, can:

- Manage a deck of card,
- Play cards based of the situation
- Play a turn based game
- And have a lot of fun

This application is encrypted with pck key in order to protect the code from malicious person and make the assets of the game private. The pck key is encrypted with openssl in hex32 which generate a unique salt at the beginning and random characters after for a total of 256 bits in hexadecimal.

The target and audience of this game was primarily casual players who want a bit of complexity without overdoing it. Dark Dominion is not suited for children of less than 14 years old, because the game is a little bit complex to understand. You have to read the rules or watch the walkthrough in order to play it.

IV- Description du cahier des charges de DarkDominion Web et Mobile

IV-1.Contexte

Le projet **DarkDominion** est un projet qui a été demandé par la société ETNA. Ce projet a pour but de créer une version mobile et web du jeu de plateau « Villainous » mais en simplifié.

La société ETNA a été créée il y a de cela 20 ans par Fabrice BARDECHE vice-président exclusif de IONIS Education Group. Elle est venue d'un constat que l'informatique se maîtrise vraiment en la pratiquant.

L'audience ciblé pour ce jeu est principalement les adolescents et adultes étant donnés la complexité du jeu sur lequel il est basé.

IV-2.Fonctionnalités principales de DarkDominion

- Possibilité d'afficher un tutoriel, modification dans le menu
- Jouer a un jeu de cartes sur PC, Android et Web
- Test unitaires

IV-3.Technologies, Outils

Langage Godot / Gdscript

Le langage pour développer DarkDominion est le gdscript qui est très similaire au python et simple dans sa formulation et utilisation.

Outils de maquetting : Inspiration du plateau de jeu Villainous

Le plateau de jeu Villainous représenté la base du jeu dans ses concepts et dans sa réalisation. Nous avons pris l'inspiration de certains personnage et nous avons ensuite crée de l'originalité et de la simplicité dans le gameplay.

Front-end: Civitai

Pour l'aspect visuel et artistique du jeu nous avons particulièrement utilisé une IA open source qui s'appelle Civitai

Tests unitaires : GUT

Gut est une librairie intégré à godot permettant de réaliser des tests unitaires.

Logiciels gestion du cadre de travail : Gitlab – (ticketing via SCRUM)

Les pushs et la réalisation des projets se faisait à travers plusieurs branches parallèles à la main lié à des « issues » vérifiées par le développeur sénior une fois push avant d'être merge sur la main. La réalisation des « issues » se faisait principalement autour de réunions où on discuter des fonctionnalités et leurs réalisations similaire au projet BddCompare.

IV-4.Cadre de travail

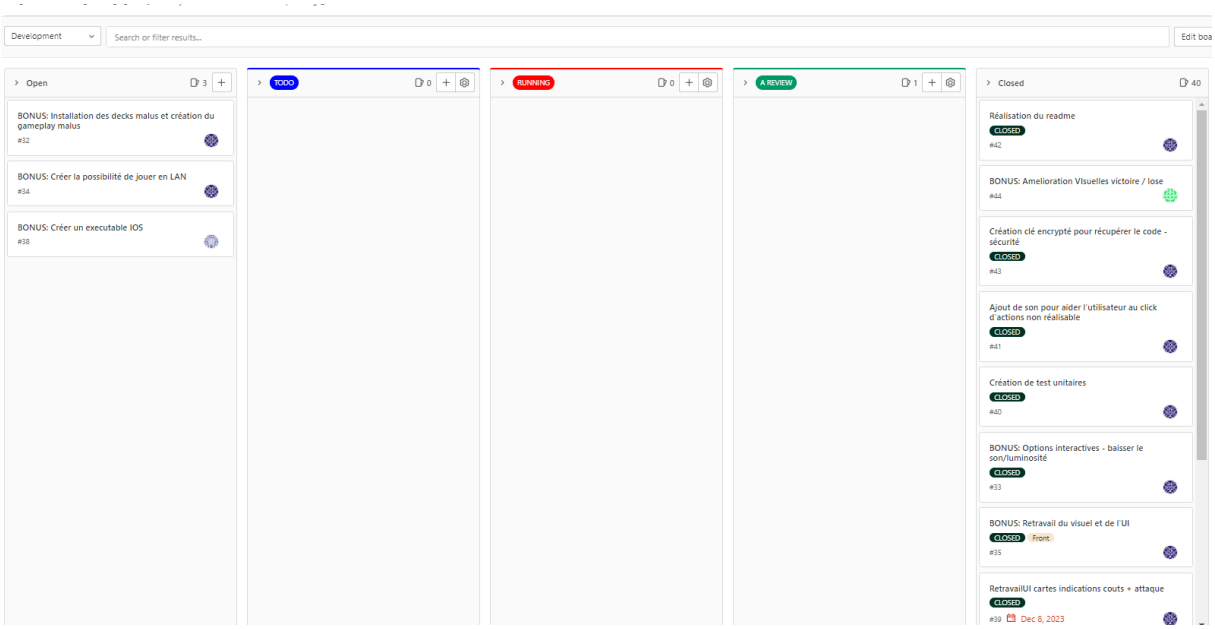
SCRUM – Culture Agile

V- *Spécification fonctionnelles et techniques du projet DarkDominion*

V-1. Organisation Gitlab similaire à BddCompare

L'organisation pour DarkDominion reprend le principe de création d'issues avec des branches paralléles.

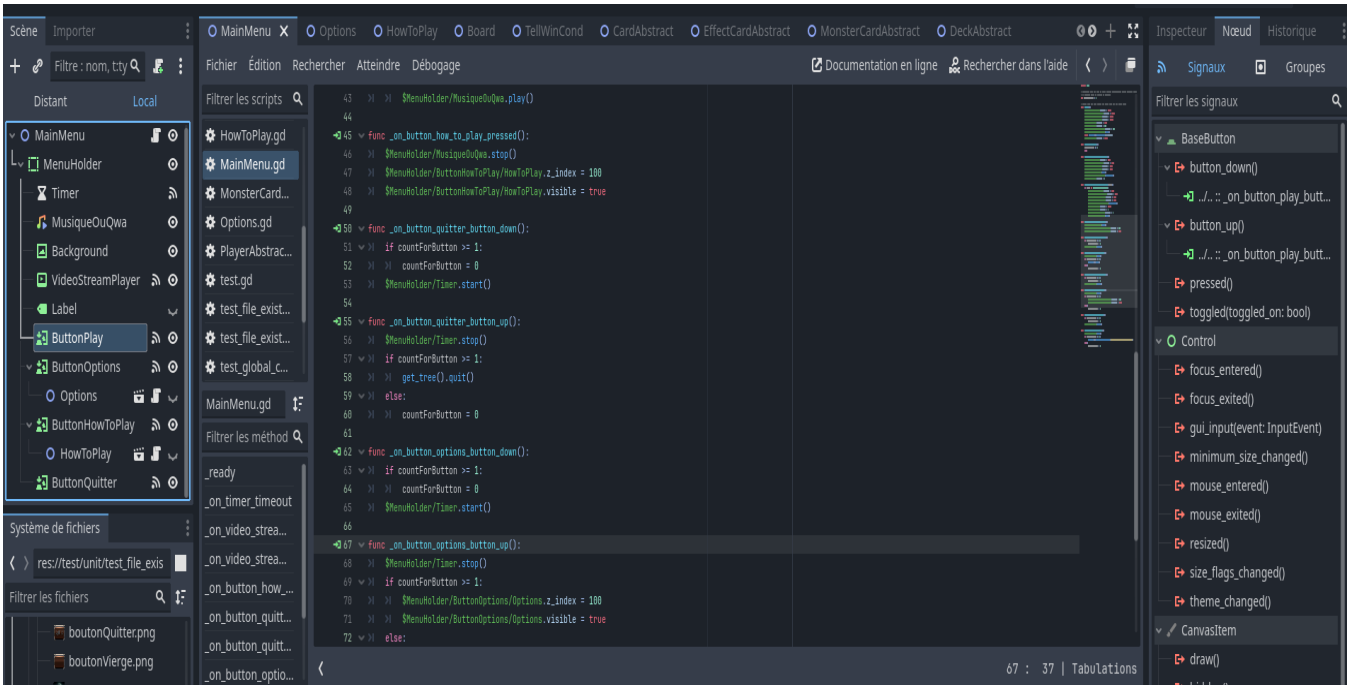
Voici une image du board du repos de BddCompare, ou on peut voir les icônes de personnes assignées



Vu que la forme beta du projet est terminé nous pouvons voir qu'il ne reste que des tickets bonus et la majorité des tickets sont closed.
Nous avons utilisés Godot qui est un moteur de jeu qui s'organise par node , et qui est principalement un codage sous principe de POO. Les nodes aidents dans la réalisation des scènes qui sont la partie visuelle de godot, et les scripts sont en gd qui sont la partie logique du jeu qui elles sont en gdscript semblables au python.

V-2. Fonctionnalités du menu

Pour le menu nous avons 4 boutons simples qui donne au joueur le choix de modifier le son dans les options, d'apprendre comment jouer et de jouer tout simplement. Avec une gestion d'évènements et de signaux avec Gdscript on peut indiquer à l'application d'avoir un certain comportement.

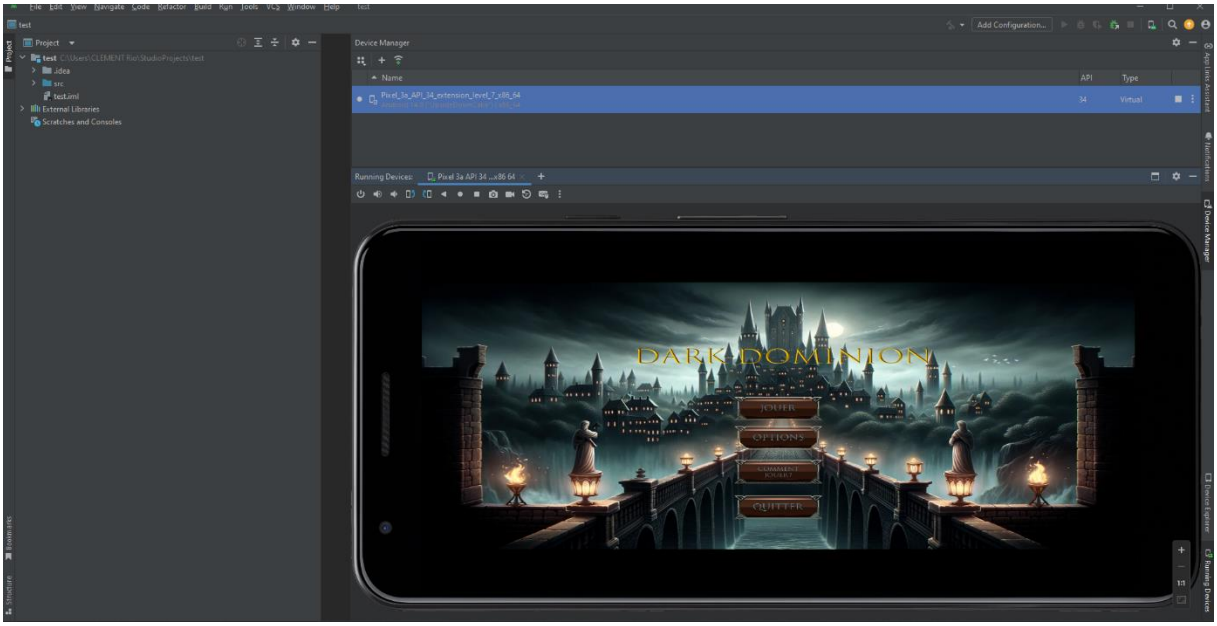


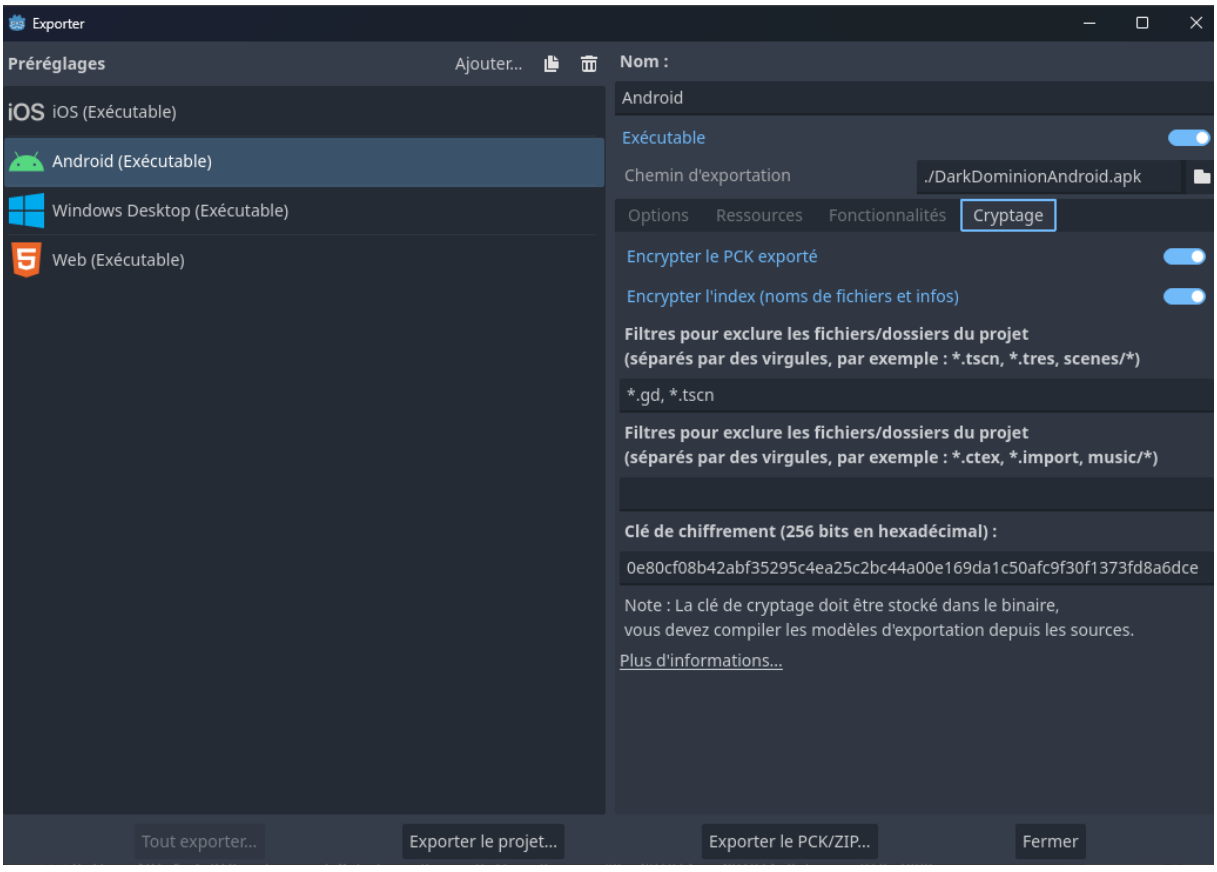
Sur ce morceau de code on peut voir l'état pressed du button qui est géré déclenchant des événements comme quitter le jeu représenter par get_tree().quit pour la fonction _on_button_quitter_button_up() .

V-2 Présence multiplateformes

Il est actuellement possible de lancer le jeu sur trois plateformes différentes

En tant qu'application mobile android, cela est possible par la technologie d'exportation simplifié de godot qui change ces attributs pour créer une apk qui peut être crypté et sécurisé si on le désire.





En tant qu'executable windows en reprenant le même principe. Et pour finir avec une version actuellement présente sur le web déployé sur itch.io https://rio-clement.itch.io/dark-dominion.

Le jeu a avant tout était pensé pour un jeu android avec un format de 800 de largeurs pour 300 de longeurs qui correspond à la taille lambda d'un écran android.

Un dossier crypté cachera et rendra principalement le pck inutilisable rendant impossible la récupération de fichier à partir de l'éxecutable.

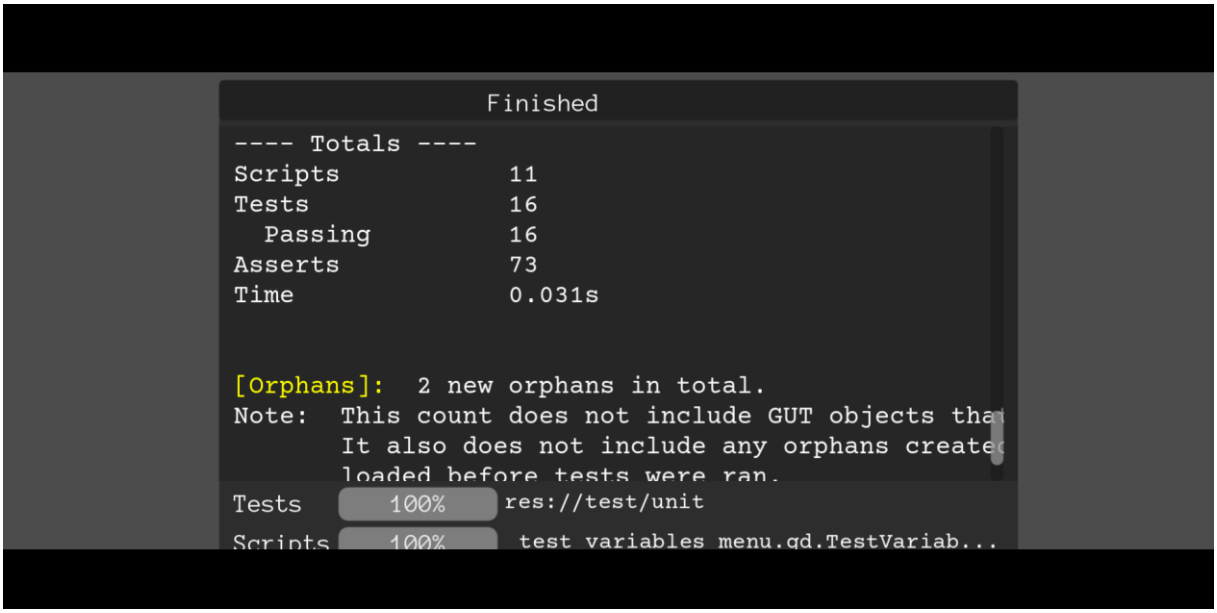
V-3 Tests unitaires

Il est possible d'effectuer des test unitaires via GUT sur Godot

```gdscript
extends GutTest

class TestFileBoard:
    extends GutTest

    func before_each():
        gut.p('-- Verification existence fichiers board --')

    func test_assert_file_exist_sprite_bouton():
        gut.p('-- Verification existence des fichiers sprites boards --')
        assert_file_exists('res://Scenes/Board/Sprite/actionbtn1.png')
        assert_file_exists('res://Scenes/Board/Sprite/actionbtn2.png')
        assert_file_exists('res://Scenes/Board/Sprite/actionbtn3.png')
        assert_file_exists('res://Scenes/Board/Sprite/actionbtn4.png')
        assert_file_exists('res://Scenes/Board/Sprite/boardcleaned.png')
        assert_file_exists('res://Scenes/Board/Sprite/boardv2.png')
        assert_file_exists('res://Scenes/Board/Sprite/boardv2clear.png')
        assert_file_exists('res://Scenes/Board/Sprite/choseActions1.png')
        assert_file_exists('res://Scenes/Board/Sprite/choseActions2.png')
        assert_file_exists('res://Scenes/Board/Sprite/choseActions3.png')
        assert_file_exists('res://Scenes/Board/Sprite/choseActions4.png')
        assert_file_exists('res://Scenes/Board/Sprite/coinfill.png')
        assert_file_exists('res://Scenes/Board/Sprite/forestForBoard.png')

    func test_assert_file_exist_sprite_player():
        gut.p('-- Verification existence des fichiers sprites players boards--')
        assert_file_exists('res://Scenes/Board/Player/Sprite/picMainAnta2.png')
        assert_file_exists('res://Scenes/Board/Player/Sprite/picMainAnta.png')
        assert_file_exists('res://Scenes/Board/Player/Sprite/picMainChara2.png')
        assert_file_exists('res://Scenes/Board/Player/Sprite/picMainChara.png')

    func test_assert_file_exist_music():
```

Dans ce script on s'assure de la bonne existence et la présence dans le path des fichiers désirés.

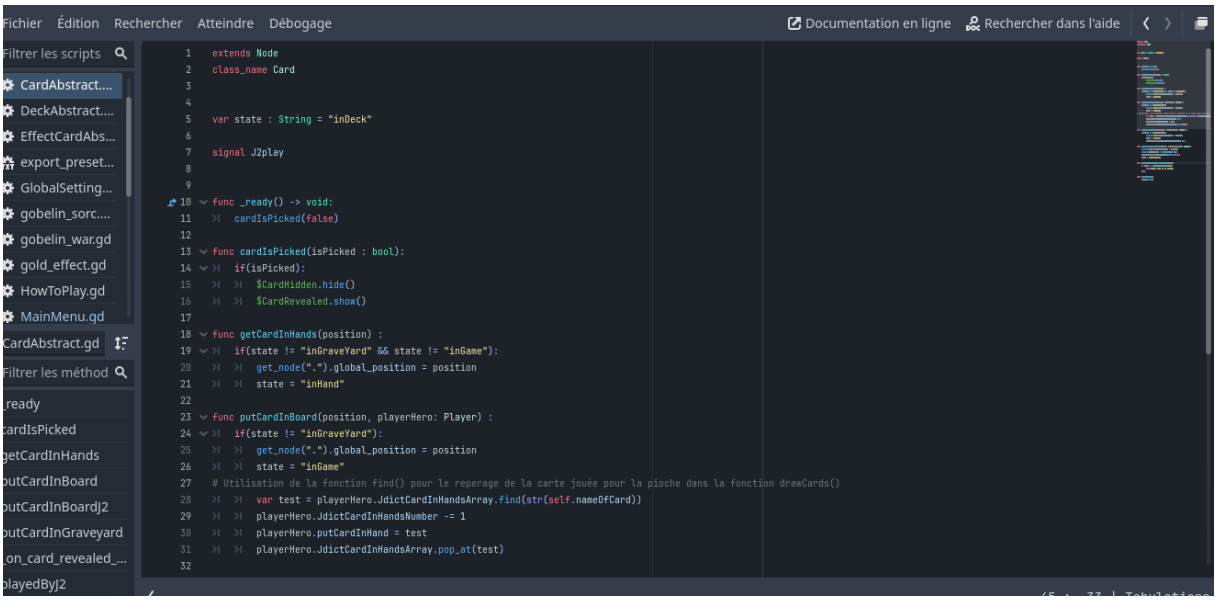
Une fois les tests lancées on a une interface qui nous indique le nombre de tests qui ont réussis et le nombre de tests qui ont ratés

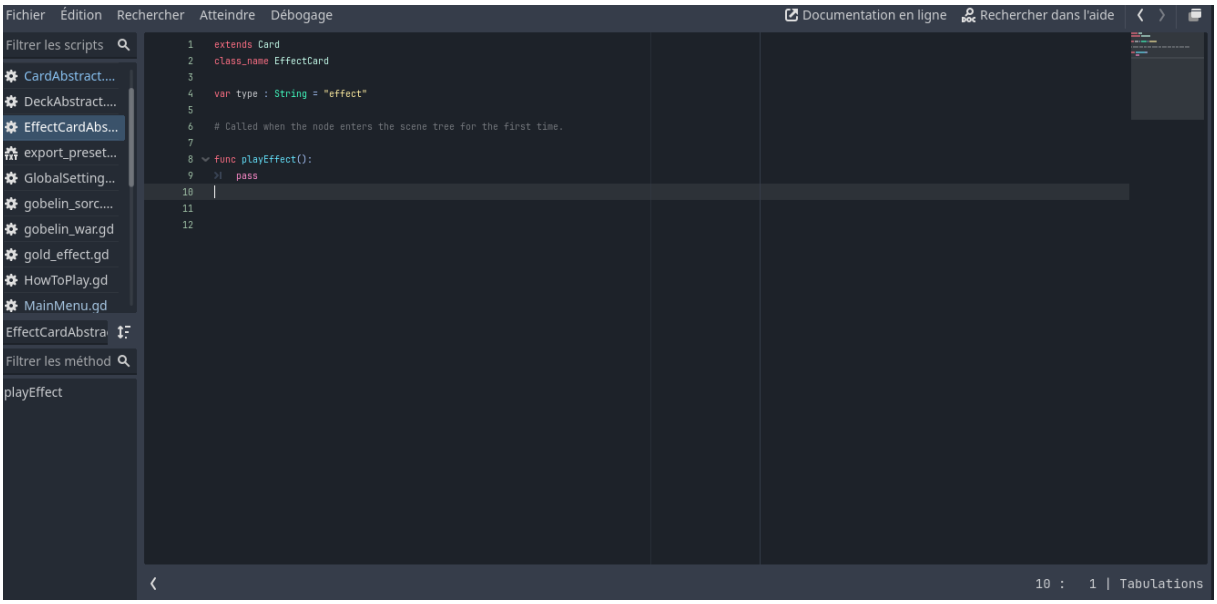


Il ajoute de plus les nœuds orphelins qui peuvent correspondre à des fuites mémoires pour aider à la bonne pratique lors de la formulation du code, en libérant la mémoire alloué lors de l'utilisation d'un nœud.

VI- Description de la veille effectuée DarkDominion

La veille principale a été la recherche sur la bonne pratique et la bonne utilisation des classes pour le code. Pour godot une bonne pratique est de réutiliser les classes et les fonctions sous forme de components. Toutes les cartes dans godot sont un héritage de la classe CardAbstract.gd par exemple faisant qu'il reprenne le code et les propriétés de cet élément.





On le voit avec le « extends Card » . Ce principe s'applique aussi au niveau des scènes qui sont un élément pour gérer la partie visuelle du jeu.