BTS SIO 2023

☐ Administration des systèmes et des réseaux (E5 – SISR)

ou

■ Conception et développement d'applications (E5 – SLAM)

PAGE DE PRÉSENTATION DU DOSSIER

N° d'inscription[1] : |0_|1_|9_|4_|7_|0_|8_|2_|9_|6_| |9_|

NOM : DESCHAMPT

Prénom : Thomas

Date de passage[1] : 31/05/2023	**Heure de passage[1] :** 12**h**00

CATEGORIE CANDIDAT [2] (UNE CASE A COCHER)	
■ Scolaire	❑ Ex-scolaire
❑ Apprenti	❑ Ex-apprenti
❑ Formation professionnelle continue	❑ Ex-formation professionnelle continue
❑ Expérience professionnelle 3 ans	

[1]Informations communiquées sur votre convocation envoyée en mars-avril 2022

[2] Informations communiquées sur votre confirmation d'inscription

Tampon de l'établissement

BTS SERVICES INFORMATIQUES AUX ORGANISATIONS **SESSION 2023**

Épreuve E5 - Conception et développement d'applications (option SLAM)

ANNEXE 7-1-B : Fiche descriptive de réalisation professionnelle (recto)

DESCRIPTION D'UNE RÉALISATION PROFESSIONNELLE	**N° réalisation : 02**
Nom, prénom : Deschampt Thomas	**N° candidat :** 01947082969
Épreuve ponctuelle ☒ **Contrôle en cours de formation** ☐	**Date :** 31/05/2023

Organisation support de la réalisation professionnelle
Laboratoire GSB

Intitulé de la réalisation professionnelle
PROJET Service Web

Période de réalisation : 15/03/2023 au 27/04/2023 **Lieu :** EPSI Lyon
Modalité : ☒ **Seul(e)** ☐ **En équipe**

Compétences travaillées
- ☒ Concevoir et développer une solution applicative
- ☐ Assurer la maintenance corrective ou évolutive d'une solution applicative
- ☒ Gérer les données

Conditions de réalisation[5] (ressources fournies, résultats attendus)
Ressources fournies : description du contexte, expression des besoins, exemple de page

Résultats attendus : Réalisation d'une solution Frontend web MVC qui utilise les API développées lors de la première réalisation, sécurisation de certaines pages via un formulaire d'authentification pseudo/mot de passe (informations stockées sur une base de données locale), format de sortie json

Description des ressources documentaires, matérielles et logicielles utilisées[6]

SGBD : SQL Server et SQL Management Studio
Environnement de développement : Visual Studio
Bibliothèque de développement : .NET
Langages : HTML, CSS, Javascript, C#, Jquery
Gestion de version : Github
Tests des comportement API :Tests unitaires et tests d'intégrations manuels via navigateur web

Modalités d'accès aux productions[7] et à leur documentation[8]
Base de données et application Frontend hébergées en local via Visual Studio
https://github.com/ThomasDeschampt/BTS_PPE
- Seul le répertoire FrontCours est concerné, il regroupe l'ensemble des fichiers de la solution web

[5] En référence aux *conditions de réalisation et ressources nécessaires* du bloc « Conception et développement d'applications » prévues dans le référentiel de certification du BTS SIO.
[6] Les réalisations professionnelles sont élaborées dans un environnement technologique conforme à l'annexe II.E du référentiel du BTS SIO.
[7] Conformément au référentiel du BTS SIO « *Dans tous les cas, les candidats doivent se munir des outils et ressources techniques nécessaires au déroulement de l'épreuve. Ils sont seuls responsables de la disponibilité et de la mise en œuvre de ces outils et ressources. La circulaire nationale d'organisation précise les conditions matérielles de déroulement des interrogations et les pénalités à appliquer aux candidats qui ne se seraient pas munis des éléments nécessaires au déroulement de l'épreuve.* ». Les éléments peuvent être un identifiant, un mot de passe, une adresse réticulaire (URL) d'un espace de stockage et de la présentation de l'organisation du stockage.
[8] Lien vers la documentation complète, précisant et décrivant, si cela n'a été fait au verso de la fiche, la réalisation professionnelle, par exemples service fourni par la réalisation, interfaces utilisateurs, description des classes ou de la base de données.

Contexte et besoins :

Dans le cadre de ses activités, le laboratoire Galaxy Swiss Bourdin a pour habitude de travailler avec des médecins implantés dans différents départements, chacun pouvant avoir une spécialité. Ainsi, le GSB utilise une base de données listant tous les médecins pour faciliter ses différents travaux. Cette base est vouée à évoluer en permanence compte tenu de l'arrivée de nouveaux médecins et du départ en retraite de certains d'entre eux, mais aussi à cause des déplacements de médecins vers de nouveaux départements.

Pour répondre à ses besoins, le GSB a déjà contacté une entreprise de services informatiques qui a été chargée de fournir au laboratoire un Service Web répondant à toutes leurs contraintes. Désormais, la demande du GSB est de lui fournir une application Front qui lui permettra de visualiser directement, depuis son navigateur, les informations en utilisant le Service Web.

Solution à mettre en place :

Pour satisfaire les attentes du GSB, l'application web va devoir respecter un certain nombre de contraintes.

Dans un premier temps, il faut que l'application fasse appel aux API (*Application Programming Interface*) du Web Service créé lors de la première réalisation ; l'application permettra donc aussi bien la visualisation que la modification, la suppression et l'ajout de données. L'utilisateur pourra aussi télécharger un fichier JSON qui contient les informations affichées à l'écran.

L'application permettra aussi de retrouver une liste de médecins en fonction d'un département ou bien en fonction de son nom.

Le fonctionnement de la solution Front repose sur la technologie .NET version 4.8 ainsi que des langages web de base tels que HTML, CSS et JavaScript. Les requêtes de récupération et d'envoi de données en direction de la base se feront donc par le biais d'API qui utilise le header HTTPS.

Pour ce qui est de la sécurisation, il faudra disposer d'un Token particulier pour pouvoir appeler certaines API ; dans le cas contraire, l'utilisateur sera renvoyé vers une erreur 403 "Accès refusé".

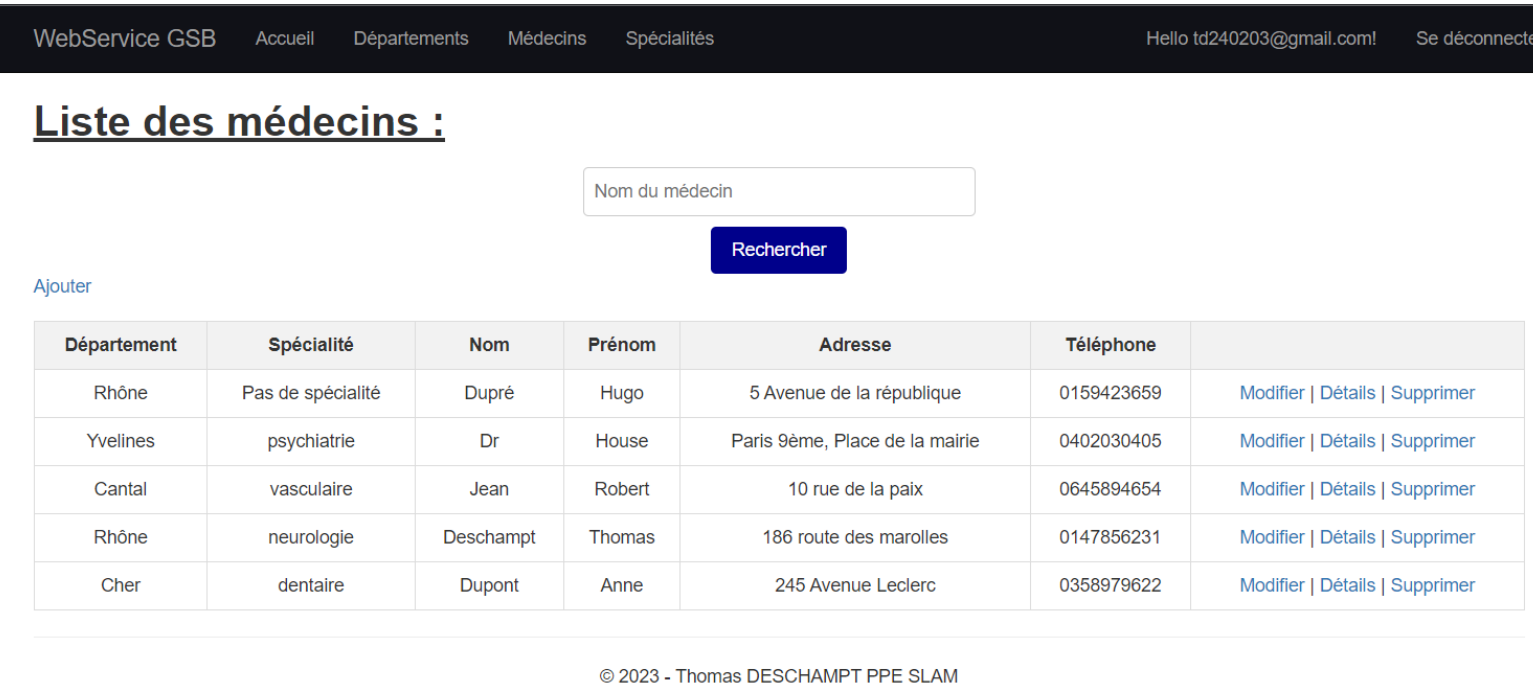
De plus, l'accès à certaines pages de l'application sera privé ; par exemple, la page avec la liste des départements et leurs médecins associés sera accessible à tous alors que la page pour modifier un médecin sera privée et nécessitera de se connecter avec des identifiants valides.

Réalisations :

Pour débuter cette réalisation, j'ai dans un premier temps créé un nouveau projet web ASP .NET au sein de la même solution que pour la première réalisation. Ce qui nous permettra de pouvoir interagir avec les éléments développés précédemment. Ensuite, il est important de ne pas oublier d'ajouter des références vers l'ORM (O*bject-relational mapping)* et le modèle depuis notre projet Front de manière à bien rendre possible les interactions.

L’avantage de .NET est qu’à la création d’un projet, le Framework va générer de nombreux fichiers qui vont nous permettre d’avoir une base de départ structurée et fonctionnelle, et donc de ne pas commencer à partir de zéro. Cela nous permet ainsi de gagner du temps au lancement d’un nouveau projet ; on pourra se concentrer sur les nouvelles fonctionnalités à mettre en place et pas sur la totalité de l'infrastructure de l’application.
À la création, on retrouve notamment différentes pages comme la page d’accueil, la page de connexion, la page contact mais aussi les contrôleurs, les scripts et fichiers de style qui leur sont associés.

Avant de passer aux explications des réalisations faites lors de ce projet, voici pour mieux se rendre compte des objectifs du projet, une page de la solution une fois terminée.



Contrôleurs :

Tout d’abord, on va pouvoir ajouter 3 contrôleurs à notre projet, un pour chaque table de la base de données que l’on va utiliser (médecin, département et spécialité). Pour créer ces contrôleurs on utilisera le contexte de données du data_model de notre ORM. Il est aussi judicieux d’autoriser les actions asynchrones sur les contrôleurs, de manière à améliorer la fluidité et la réactivité de la solution front. Les actions asynchrones vont permettre à l’application de fonctionner normalement même si cette dernière est en attente d’une réponse suite à l’appel d’une API alors que dans le cas où les actions sont effectuées de manière synchrone, l’application sera bloquée pendant le temps d’attente.

Néanmoins, une fois le contrôleur créé, on peut voir que même si ce dernier est bien asynchrone avec le “async” de la deuxième ligne, le contrôleur ne passe pas par les API pour recevoir les données mais utilise ses propres méthodes.

L'objectif est donc de modifier les contrôleurs pour qu'ils fassent appel aux API, la solution front ne doit pas réaliser d'opération par elle-même mais seulement demander au Web Service de les effectuer.

Pour le getter, qui renvoie le détail des données d'un médecin précis en fonction de son identifiant, il faut dans un premier temps vérifier si un identifiant a bien été envoyé avec l'appel de l'API, et dans le cas contraire, un message d'erreur sera renvoyé.
Ensuite, j'ai associé à la variable url, l'adresse de l'API que l'on va appeler, ici : "https://localhost:44345/api/medecins/"+id ; qui correspond à l'url du Web Service et l'API désirée suivi de l'identifiant du médecin passé comme paramètre.

On crée ensuite un objet HttpClient pour envoyer la requête. Dans le bloc "using", on appelle la méthode "GetAsync" pour envoyer la demande des données du médecin à l'API et attendre sa réponse avec le mot-clé "await". Si la réponse n'est pas "SuccessStatusCode", cela signifie qu'il y a eu une erreur suite à la demande, et une exception est levée par le serveur.

Si la réponse ne génère pas d'erreur, on extrait les données en utilisant la méthode "ReadAsAsync". Enfin, la méthode renvoie la vue "Détails" avec les données du médecin demandé.

```
// GET: medecins/Details/5
0 références
public async Task<ActionResult> Details(int? id)
{

    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }

    // url de l'api
    string url = "https://localhost:44345/api/medecins/" + id;

    using (HttpClient client = new HttpClient())
    {
        HttpResponseMessage response = await client.GetAsync(url);

        if (!response.IsSuccessStatusCode)
            throw new Exception();

        var medecin = await response.Content.ReadAsAsync<medecin>();
        return View(medecin);
    }

}
```

J'ai ensuite répété cette opération pour chacune des actions que notre contrôleur peut être amené à effectuer (afficher, modifier, ajouter et supprimer), puis il a aussi fallu faire ces mêmes modifications pour les contrôleurs Départements et Spécialités.

Recherche :

Pour la recherche en fonction d'un nom de médecin, qui n'est pas prise en charge par défaut lors de la création du contrôleur, j'ai fait son implémentation de zéro, avec pour objectif d'appeler une fois encore l'API mise en place dans le Backend.

L'un des enjeux principaux de la recherche d'un médecin est la gestion des exceptions qui peuvent nous renvoyer des erreurs si elles sont mal gérées. Par exemple, dans le cas où la recherche ne retourne aucun médecin, l'application va nous renvoyer par défaut vers une page d'erreur. L'intérêt de la gestion de l'exception serait ici de renvoyer l'utilisateur vers une page l'informant qu'aucune correspondance n'a été trouvée.

Pour ce faire, on va procéder aux vérifications de base dans le contrôleur, c'est-à-dire vérifier si un nom a bien été renseigné et que le formulaire n'a pas été envoyé vide puis après cela j'ai adapté la page médecin. Cette page s'adapte ainsi en fonction du nombre de médecins qu'elle reçoit : elle affichera un tableau si un ou plusieurs résultats lui sont retournés mais n'affiche qu'un message "Aucun médecins n'a été trouvé " si l'API ne lui retourne pas de résultat.

Ensuite pour la recherche, j'ai créé un formulaire qui va appeler l'action SearchMedecins du contrôleur medecins par le biais de la méthode GET.

```
@using (Html.BeginForm("SearchMedecins", "medecins", FormMethod.Get))
{
    <input type="text" name="nom" placeholder="Nom du médecin">
    <button type="submit">Rechercher</button>
}
```

Sécurisation :

Après avoir configuré l'appel des API, il faut désormais sécuriser leur accès. En effet, il ne serait pas approprié de laisser n'importe qui pouvoir apporter ses modifications à la base de données. C'est pour cela que l'on va configurer certaines pages comme interdites d'accès si l'utilisateur n'est pas authentifié.

A sa création, le projet .NET a lui-même généré une méthode permettant de créer un compte et de s'identifier sur l'application Front. Les mots de passe et identifiants étant stockés après avoir été hachés sur une base de données. Ainsi, il nous suffit de faire appel à la méthode d'authentification en précisant [authorize] qui redirigera l'utilisateur vers la page de connexion s'il essaye d'accéder à l'API sans s'être authentifié.

Dans notre cas, [authorize] à un deuxième rôle, il permet de faire appel à la sécurisation via Token JWT mise en place lors de la première réalisation. Ainsi, il nous faut ajouter un nouvel entête au sein du header HTTPS de manière à intégrer le token à la requête pour que le serveur valide notre demande.

```
client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
client.DefaultRequestHeaders.Add("Authorization", "Bearer " + ReadToken());
```

Téléchargement des données:

Enfin la dernière méthode que j'ai mise en place donne la possibilité de pouvoir télécharger un fichier JSON qui va contenir toutes les informations présentes à l'écran, cette méthode permettra de télécharger la liste des médecins, la liste des départements et la liste des spécialités.

Ici, dans le cas des départements, on va appeler l'API qui retourne la liste des départements puis on va vérifier si l'appel s'est déroulé avec succès. Si c'est le cas, alors on va extraire le contenu de la réponse pour le convertir en texte JSON. Ensuite on va désérialiser le texte (cela nous permet de transformer le texte en objet pour d'éventuels futurs traitements), l'encoder en ASCII pour enfin terminer par renvoyer le fichier JSON sous la forme d'un fichier téléchargeable par l'utilisateur.

```
//GET : departements/Downaload
0 références
public FileContentResult DownloadJson()
{
    var client = new HttpClient();
    client.BaseAddress = new Uri("https://localhost:44345/api/departements/");
    var response = client.GetAsync(client.BaseAddress).Result;

    if (response.IsSuccessStatusCode)
    {
        var json = response.Content.ReadAsStringAsync().Result;
        var departements = JsonConvert.DeserializeObject<IEnumerable<departement>>(json);
        var bytes = Encoding.ASCII.GetBytes(json);
        return File(bytes, "application/json", "liste_departements.json");
    }
    else
    {
        throw new Exception("Une erreur est survenue lors de la récupération des données");
    }
}
```

Après avoir effectué toutes les tâches relatives à la sécurisation et aux interactions avec le Web Service, on peut maintenant se pencher sur les modifications fonctionnelles et visuelles à mettre en place sur la solution Front.

Visuel :

J'ai décidé de ne pas aborder les modifications de style que j'ai réalisées, car ces modifications sont assez répétitives et peu intéressantes dans le cadre de ce rapport. Malgré tout, il y a quand même eu des modifications fonctionnelles, par exemple pour la page détail d'un département (qui n'affiche à l'origine que le nom, l'identifiant et la région du département) j'ai ajouté de nouveaux éléments qui vont permettre d'afficher les médecins associés à ce département.

Dans un premier temps, j'ai mis en place une vérification pour savoir s'il y avait bien au moins un médecin associé au département, si ce n'est pas le cas alors le tableau ne s'affichera pas, ce qui évitera d'avoir un tableau vide affiché sur la page.

Les premières balises <tr> comportent les données “statique” du tableau, le nom des colonnes.
Ensuite, la boucle foreach permet d’ajouter une nouvelle ligne pour chaque médecin associé, donc pour chaque médecin contenu dans Model.medecins.
Pour chaque item, on va afficher la valeur de ses champs par exemple, @Html.DisplayFor(modelItem => item.nom_med) affiche la valeur du champ nom_med de l’objet médecin actuellement dans la boucle foreach.