

DOSSIER PROFESSIONNEL (DP)

Nom de naissance ▶ GRIGNARD
Nom d'usage ▶ GRIGNARD
Prénom ▶ Georges
Adresse ▶ 3 avenue Olympe de Gouges
44240 La Chapelle-sur-Erdre

Titre professionnel visé

Concepteur Développeur d'Applications

MODALITE D'ACCES :

- ☒ Parcours de formation
- ☐ Validation des Acquis de l'Expérience (VAE)

Présentation du dossier

Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel. **Ce titre est délivré par le Ministère chargé de l'emploi.**

Le DP appartient au candidat. Il le conserve, l'actualise durant son parcours et le présente **obligatoirement à chaque session d'examen.**

Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE.

Il est consulté par le jury au moment de la session d'examen.

Pour prendre sa décision, le jury dispose :

1. des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l'entretien professionnel ou de l'entretien technique ou du questionnement à partir de productions.
2. du **Dossier Professionnel** (DP) dans lequel le candidat a consigné les preuves de sa pratique professionnelle
3. des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d'un parcours de formation
4. de l'entretien final (dans le cadre de la session titre).

[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels du ministère chargé de l'Emploi]

Ce dossier comporte :

- ▶ pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ;
- ▶ un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d'un titre, d'un diplôme, d'un certificat de qualification professionnelle (CQP) ou des attestations de formation ;
- ▶ une déclaration sur l'honneur à compléter et à signer ;
- ▶ des documents illustrant la pratique professionnelle du candidat (facultatif)
- ▶ des annexes, si nécessaire.

Pour compléter ce dossier, le candidat dispose d'un site web en accès libre sur le site.



<http://travail-emploi.gouv.fr/titres-professionnels>

Sommaire

Exemples de pratique professionnelle

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

p.

- ▶ Les sessions p. 4
- ▶ L'authentification p. 8

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

p.

- ▶ Les modèles p. 11
- ▶ Les entités p. 13
- ▶ La persistance p. 15

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

p.

- ▶ Trois tiers p. 17
- ▶ MVC..... p. 19

Titres, diplômes, CQP, attestations de formation *(facultatif)*

p.

Déclaration sur l'honneur

p.

Documents illustrant la pratique professionnelle *(facultatif)*

p.

Annexes *(Si le RC le prévoit)*

p.

Activité-type 1

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

Exemple n°1 ► Les Sessions

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Introduction aux sessions

Les sessions sont utilisées pour stocker des informations utilisateur persistantes entre les requêtes HTTP dans une application web. Dans notre application C#, les sessions sont gérées à travers divers composants configurés et utilisés ensemble pour assurer une gestion efficace et sécurisée des sessions.

Configuration des Services pour notre application

Le fichier **Startup.cs** contient la méthode **ConfigureServices** qui configure les services nécessaires pour la gestion des sessions, l'authentification, et d'autre fonctionnalités :

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(Microsoft.AspNetCore.Authentication.Negotiate.NegotiateDefaults.AuthenticationScheme).AddNegotiate();
    services.AddSingleton<ValidateAuthentication>();
    services.AddRazorPages().AddJsonOptions(opts =>
    {
        opts.JsonSerializerOptions.PropertyNamingPolicy = null;
    });
    services.AddHttpContextAccessor();
    // Sliding vs Absolute Expiration (attention ! le lien ci-dessous parle du cache commun et non relatif au contexte)
    // https://stackoverflow.com/questions/13637856/net-caching-how-does-sliding-expiration-work
    services.AddSession(options =>
    {
        options.IdleTimeout = TimeSpan.FromMinutes(30);
    });
    services.AddScoped<ISessionService, SessionService>();
    // services.AddTransient<ContextUtilisateur>();

    services.AddOptions().Configure<DataWiperAppConfig>(Configuration.GetSection("ApplicationConfig"));
    services.AddHostedService<Lifetime>();
    services.AddLogging(loggingBuilder =>
    {
        //loggingBuilder.AddLog4Net("log4net.{InFix()}.config");
    });
}
```

1. Configuration des Sessions

Déconnexion automatique des sessions après une période d'inactivité pour réduire les risques de sécurité :

```
services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(30);
});
```

2. HttpContextAccessor

HttpContextAccessor, vas nous permettre d'accéder à **HttpContext**, et qui contient des informations sur la requête HTTP actuelle, y compris les sessions.

3. Injection de Dépendances

ISessionService : Interface pour le service de gestion des sessions.

SessionService : Implémentation de **ISessionService** qui sera injectée avec une durée de vie « scoped », c'est-à-dire une nouvelle instance pour chaque requête HTTP.

Gestion des Sessions dans nos contrôleurs

Les contrôleurs gèrent les sessions en injectant **IHttpContextAccessor**, qui permet d'accéder à **HttpContext** et aux sessions associées. Les services comme **SessionService** encapsulent la logique d'accès à **HttpContext** et aux sessions, ce qui permet une gestion modulaire et testable des sessions dans notre application. Les données utilisateur sont stockées et récupérées dans les sessions, et ces données sont utilisées pour initialiser des services ou personnaliser l'expérience utilisateur.

```
public ApplicationController(IHttpContextAccessor httpContextAccessor)
{
    var cacheKey = "user";
    this.sessionService = new SessionService(httpContextAccessor);
    ISession session = this.sessionService.GetSession();
    var userJson = session.GetString(cacheKey);
    SvcUtilisateurModel user = null;
    if (!string.IsNullOrEmpty(userJson))
    {
        try
        {
            // Null safety
            user = JsonConvert.DeserializeObject<SvcUtilisateurModel>(userJson);
        }
        catch (JsonSerializationException)
        {
            // pass
        }
    }
    service = new SvcApplication(user);
}
```

Gestion des informations utilisateur (Partie 2 ou 1...)

Le fichier ``ContextUtilisateur.cs`` contient plusieurs méthodes pour gérer les informations utilisateur en utilisant les sessions.

- ``GetUtilisateur(string login, string habilitation)`` : Récupère les informations de l'utilisateur basé sur le login et l'habilitation.
- ``ExtractUserRoles(PrincipalSearchResult<Principal> groups)`` : Extrait les rôles AD de l'utilisateur.
- ``SetUserSession(ISession session, SvcUtilisateurMModel user, string cacheKey)`` : Stocke les informations de l'utilisateur dans la session.
- ``userRolesParser(PrincipalSearchResult<Principal> groups, SvcUtilisateurModel user)`` : Analyse les rôles de l'utilisateur.

Ces méthodes permettent de stocker et récupérer des informations utilisateur spécifiques dans les sessions, ce qui est crucial pour l'authentification et l'autorisation (Partie 2).

Conclusion

Pour récapituler, voici un aperçu complet de la gestion des sessions dans notre application :

1. **Configuration dans Startup.cs** : Configure les services nécessaires pour l'authentification, les sessions, HttpContextAccessor, et d'autres services essentiels.
2. **Service de Gestion des Sessions (SessionService)** : Utilise IHttpContextAccessor pour accéder à HttpContext et obtenir la session actuelle.
3. **Utilisation dans les Contrôleurs** : Les contrôleurs injectent IHttpContextAccessor pour accéder à la session via SessionService. Ils récupèrent et stockent les données utilisateur dans la session.
4. **Gestion des Informations Utilisateur (ContextUtilisateur)** : Fournit des méthodes pour gérer les informations utilisateur, stocker et récupérer les données utilisateur dans la session.

Ces éléments combinés assurent une gestion efficace et sécurisée des sessions dans notre application. En utilisant **IHttpContextAccessor** et en configurant correctement les services dans **Startup.cs**, nos contrôleurs peuvent accéder et gérer les sessions de manière centralisée et cohérente.

En rédigeant ce DP, je m'aperçois que **ASP.Core 6** et plus, n'utilise plus **Startup.cs** par défaut pour un nouveau projet, mais plutôt directement **Program.cs**, ou il faut ajouter directement les Services au builder, par exemple :

```
var builder = WebApplication.CreateBuilder(args);  
var app = builder.Build();  
app.UseAuthorization();
```

DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

À l'aide d'un IDE (Visual Studio), d'un langage de programmation : C#, de .NET Core 5, des RazorPages, de TypeScript, de ce qu'on appelle l'Internet, d'un PC à 8Go de RAM, enfin et surtout du package `Microsoft.AspNetCore.Http`.

3. Avec qui avez-vous travaillé ?

Nous avons été 4 à travailler sur ce chantier, Rémi l'externe qui aura démarré le chantier, moi-même qui aura commencé mon apprentissage de C# à ses côtés, épauler par notre manager Denis, et enfin consolidé et terminer par le stagiaire Nathan.

4. Contexte

Nom de l'entreprise, organisme ou association ► Cliquez ici pour taper du texte.

Chantier, atelier, service ► Pour le **DataWiper** et **SQLTheque**

Période d'exercice ► Du **30/01/2023** au **26/01/2024**

5. Informations complémentaires (facultatif)

Cliquez ici pour taper du texte.

Activité-type 1

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

Exemple n°2 ► L'Authentification

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Introduction

L'authentification est un aspect crucial de la sécurité des applications garantissant que seuls les utilisateurs autorisés peuvent accéder aux ressources protégées. Dans notre application l'authentification est gérée à travers divers composants et services pour assurer une sécurité robuste.

Configuration

Dans **Startup.cs**, nous configurons l'authentification en utilisant le schéma **Negotiate**, qui est souvent utilisé pour l'authentification intégrée de Windows (détails du code dans la partie 1). Nous utiliserons également un Middleware, puis nous créerons un contexte utilisateur.

- Middleware de Validation d'Authentification

Le middleware **ValidateAuthentication** assure que toutes les requêtes sont faites par des utilisateurs authentifiés. Il est enregistré en tant que singleton pour garantir une instance unique partagée dans toute l'application.

```
internal class ValidateAuthentication : IMiddleware
{
    public async Task InvokeAsync(HttpContext context, RequestDelegate next)
    {
        if (context.User.Identity.IsAuthenticated)
            await next(context);
        else
            await context.ChallengeAsync();
    }
}
```

Enregistrement du middleware dans le pipeline de requêtes via :
app.AddSingleton<ValidateAuthentication>(); dans le ConfigureServices (Visible dans la partie 1)

Et l'ajout de **App.UseMiddleware<ValidateAuthentication>()** dans le **Configure**, qui lui est utilisé pour configurer le pipeline des requête HTTP de l'application. Cette méthode définit comment répond l'application au requête HTTP.

- ContextUtilisateur.cs

C'est cette classe, **ContextUtilisateur** qui va contenir les méthodes pour gérer les informations de l'utilisateur et les rôles. (la partie 1 contient la liste des méthodes)

- Les contrôleurs

Nous avons déjà évoqué la construction du contrôleur dans la première partie de cette activité, le contrôleur invoquera un service en fonction de la session de l'utilisateur, le « user » sera donné comme paramètre du service.

- Les services (domaine)

```
private readonly SvcUtilisateurModel user;
public SvcApplication(SvcUtilisateurModel user)
{
    this.user = user;
}

/// <summary>
/// Liste des applications
/// </summary>
/// <param name="colName">Nom de colonne de tri.</param>
/// <param name="sortDir">Direction du tri.</param>
/// <returns>Liste des applications, triée selon les critères (par défaut, trié
par Nom).</returns>
public override IEnumerable<SvcApplicationModel> List(string colName, string
sortDir)
{
    // 1. Modèle
    IEnumerable<Application> applications = new List<Application>();
    if (this.user != null)
    {
        if (!this.user.isAdmin && this.user.habilitation == "1")
        {
            string apps = string.Join(",", this.user.applications);
            int appsCount = this.user.applications.Count();
            if (appsCount > 0)
            {
                applications = svcEntite.Get($"SELECT A.* FROM
T{nameof(Application)} AS A "
                + $"WHERE A.{nameof(Application.Code)} IN ('{apps}')" );
            }
        }
        else
        {
            applications = svcEntite.GetAll();
        }
    }

    // 2. Tri (sans pagination)
    var applicationsTriees = (colName, sortDir) switch
    {
        (nameof(Application.Code), "desc") =>
        applications.OrderByDescending(application => application.Code),
        (nameof(Application.Code), _) => applications.OrderBy(application =>
        application.Code),
        (nameof(Application.Nom), "desc") =>
        applications.OrderByDescending(application => application.Nom),
        (_, _) => applications.OrderBy(application => application.Nom),
    };

    // 3. Résultat pour affichage
    var result = applicationsTriees.Select(application => new
    SvcApplicationModel(application));

    return result;
}
```

DOSSIER PROFESSIONNEL (DP)

Dans un premier temps nous vérifions si le user est bien identifié si oui, nous cherchons à savoir s'il est admin, ou possède les plein droits, si ce n'est pas le cas nous récupérons les données en fonction de son identité.

Conclusion

En combinant **ValidateAuthentication**, **IHttpContextAccessor**, et les configurations d'authentification dans **Startup.cs**, notre application assure une gestion sécurisée et efficace de l'authentification des utilisateurs. Ces composants travaillent ensemble pour valider les utilisateurs, gérer les sessions et vérifier les rôles et permissions, garantissant ainsi que seules les requêtes autorisées accèdent aux ressources protégées.

2. Précisez les moyens utilisés :

Identique à la première partie.

3. Avec qui avez-vous travaillé ?

Identique à la première partie.

4. Contexte

Nom de l'entreprise, organisme ou association ► Cliquez ici pour taper du texte.

Chantier, atelier, service ► DataWiper / SQLTheque

Période d'exercice ► Du 30/01/2023 au 26/01/2024

5. Informations complémentaires (facultatif)

Cliquez ici pour taper du texte.

Activité-type 2

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

Exemple n°1 ► Les modèles

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Introduction

Les modèles (ou **Data Transfer Objects, DTOs**) représentent des objets de données utilisés pour transférer des informations au sein de l'application, souvent entre les différentes couches. Ils ne contiennent pas de logique métier ni de logique de persistance et sont utilisés principalement pour véhiculer des données. Les modèles peuvent être utilisés pour des opérations d'affichage, de transfert de données entre le client et le serveur, ou entre différentes couches de l'application.

```
[DisplayName("User")]
public record SvcUserModel : SvcModel<User>, IModelWithId, IVersionable
{
    public int Id { get; set; }
    public string Version { get; set; }

    [Display(Name="Username")]
    public string UserID { get; set; }

    [Display(Name = "Admin")]
    public int Admin { get; set; }
    public SvcUserModel() { }

    public SvcUserModel(User entite)
    {
        Id = entite.Id;
        Version = entite.Version.ToString();
        UserID = entite.UserID;
        Admin = entite.Admin;
    }
    public override User ToEntity() => new(this.Id, this.Version, this.UserID,
    this.Admin);
}
```

- L'attribut **[DisplayName]**

Les attributs en C# sont un puissant mécanisme pour ajouter des métadonnées et modifier le comportement de votre code. Ils améliorent la lisibilité et la maintenabilité du code en fournissant des informations supplémentaires et en centralisant les comportements répétitifs ou les configurations. Dans notre cas l'attribut « **DisplayName** » nous permettra de personnaliser le nom d'affichage de notre classe au sein de notre application.

- La déclaration de la classe
 - Le mot clé « **record** »

En C#, les records sont une manière concise de définir des objets immuables avec des fonctionnalités intégrées pour l'égalité structurelle, le clonage avec modification, et plus encore. Dans notre cas, le modèle est un record à cause de l'héritage, si le parent est un record, l'enfant aussi.

DOSSIER PROFESSIONNEL (DP)

- Héritage : SvcModel<User>

Afin d'accéder aux comportements et aux fonctionnalités génériques pour nos modèles, cette classe comme toutes les autres, hériteront de la classe mère « SvcModel ». Par exemple l'utilisation de la méthode ToEntity() qui sera accessible pour tous les modèles.

- Interfaces : IModelWithId et IVersionable

Étant donné que nous aurons un id et un numéro de version pour chacun des objets de notre application, nous avons créé une Interface afin de pouvoir utiliser facilement/rapidement ces propriétés à tous les modèles de notre application.

- Propriétés...
- Constructeurs

Ici nous avons deux constructeurs, celui par défaut, et celui avec l'Entité « User » en tant que paramètre, cela nous permettra de convertir une entité « User » en un modèle pour notre application.

- Méthode « ToEntity »

Cette méthode convertit le SvcUserModel en une entité User. Cela est utile pour passer du modèle utilisé dans les couches de service (domaine) ou de présentation à une entité qui peut être utilisée pour la persistance.

2. Précisez les moyens utilisés :

Pour cela nous utilisons toujours le même IDE, à savoir **Visual Studio**, du framework **ASP.Net Core 5** des **RazorPages**, de **TypeScript**, enfin et surtout du package `Microsoft.AspNetCore.Http`.

3. Avec qui avez-vous travaillé ?

Cliquez ici pour taper du texte.

4. Contexte

Nom de l'entreprise, organisme ou association ► Cliquez ici pour taper du texte.

Chantier, atelier, service ► SQLTheque / DataWiper

Période d'exercice ► Du 30/01/2023 au 26/01/2024

5. Informations complémentaires (facultatif)

Cliquez ici pour taper du texte.

Activité-type 2

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

Exemple n°2 ▶ Les entités

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Introduction

Les entités sont les représentations des objets directement mappés à la base de données. Elles reflètent la structure des tables et sont utilisées par les adaptateurs de persistance pour effectuer des opérations CRUD. Tandis que l'entité se concentre sur la persistance, le modèle se concentre lui sur la présentation et l'interaction utilisateur. Cette séparation des préoccupations permet de structurer l'application de manière modulaire et maintenable, tout en assurant que les données restent synchronisées entre la base de données et l'interface utilisateur.

```
[Display(Name = "Liste des Users")]
public record User : Entite
{
    [Display(Name = "User ID")]
    public string UserID { get; init; }

    [Display(Name = "Admin")]
    public int Admin { get; init; }

    public User(int Id, string Version, string UserID, int Admin)
    {
        this.Id = Id;
        this.Version = Version;
        this.UserID = UserID;
        this.Admin = Admin;
    }

    public override bool ValueEquality(object other)
    {
        User that = other as User;
        if (object.ReferenceEquals(this, that)) return true;
        if (that is null) return false;
        return base.ValueEquality(that)
            && this.UserID == that.UserID
            && this.Admin == that.Admin;
    }

    public override string ToString() => $"User {this.Id}";
}
```

Ainsi, nos entités et nos modèles auront une forte similitude, c'est dû à la nécessité de représenter les mêmes données de manière cohérente à travers différentes couches de l'application.

2. Précisez les moyens utilisés :

Identique à la première partie.

DOSSIER PROFESSIONNEL (DP)

3. Avec qui avez-vous travaillé ?

Identique à la première partie.

4. Contexte

Nom de l'entreprise, organisme ou association ► Cliquez ici pour taper du texte.

Chantier, atelier, service ► SQLTheque / DataWiper

Période d'exercice ► Du 30/01/2023 au 26/01/2024

5. Informations complémentaires (facultatif)

Cliquez ici pour taper du texte.

Activité-type 2

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

Exemple n°3 ► La persistance des données

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

J'aurai aimé vous parler de DAO (**Data Accès Object**) mais notre application n'utilise pas de DAO à proprement dit. C'est un peu plus complexe qu'une « simple » DAO ou encore moins que l'utilisation d'un ORM type « **Entity Framework** » que nous pourrions voir en cours par exemple... 😞
Pour la persistance des données, nous utiliserons plusieurs couches :

- Un service de domaine

Les services de domaine encapsulent la logique métier de notre application. Ils orchestrent les opérations entre les modèles, les entités et la persistance des données. Ces services manipulent les objets métier, appliquent des règles métier et coordonnent l'accès aux données. **IL EST APPELÉ DEPUIS LE CONTROLLER**

- Un adaptateur d'entité

Les adaptateurs d'entité servent de pont entre les entités persistantes et les objets métier (modèles). Ils sont responsables de la conversion des entités de la base de données en objets métier et vice versa. Les adaptateurs d'entité sont souvent personnalisés pour chaque type d'entité afin de gérer les particularités de la persistance et de la transformation des données.

```
public class AdaptateurUser : AdaptateurEntite<User>
{
    public override User WithLinks(User instance) => instance;
}
```

- Un fournisseur de base de données « **DbProvider** »

Les fournisseurs de base de données offrent une abstraction pour interagir avec différents types de bases de données. Ils fournissent des fonctionnalités telles que la connexion à la base de données, l'exécution de requêtes, la gestion des transactions, etc. Chaque fournisseur de base de données peut avoir ses propres implémentations spécifiques pour gérer les détails techniques de la communication avec la base de données.

2. Précisez les moyens utilisés :

Identique à la première partie.

3. Avec qui avez-vous travaillé ?

Identique à la première partie.

4. Contexte

Nom de l'entreprise, organisme ou association ► Cliquez ici pour taper du texte.

DOSSIER PROFESSIONNEL (DP)

Chantier, atelier, service	► SQLTheque / DataWiper
Période d'exercice	► Du 30/01/2023 au 26/01/2024

5. Informations complémentaires *(facultatif)*

Cliquez ici pour taper du texte.

Activité-type 3

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

Exemple n°1 ► Architecture trois tiers

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

L'architecture trois tiers est un modèle de conception utilisé pour structurer les applications en trois couches distinctes : la couche de présentation, la couche logique métier et la couche de données. Cette séparation des préoccupations permet d'améliorer la modularité, la maintenabilité et la scalabilité de l'application.

1. Couche de Présentation

La couche de présentation est responsable de l'interaction avec l'utilisateur. Elle affiche les données et recueille les entrées de l'utilisateur.

Dans notre application, nous utilisons : « RazorPages » et « TypeScript »

2. Couche Logique Métier

La couche logique métier contient la logique de l'application. Elle traite les règles métier et les processus de gestion.

Dans notre application, nous utilisons : « ASP.NET Core » et une « API Rest »

3. Couche de Données

La couche de données gère l'accès et la manipulation des données. Elle interagit avec la base de données pour stocker et récupérer les informations nécessaires.

Dans notre application, nous utilisons : « SQL Server » et un « DBProvider »

L'architecture trois tiers permet une séparation claire des préoccupations, ce qui facilite la maintenance, la scalabilité et la sécurité des applications. En implémentant des recommandations de sécurité à chaque couche, nous protégeons notre application contre diverses menaces et assurons la protection des données utilisateurs.

2. Précisez les moyens utilisés :

Cliquez ici pour taper du texte.

3. Avec qui avez-vous travaillé ?

Cliquez ici pour taper du texte.

4. Contexte

Nom de l'entreprise, organisme ou association ► Cliquez ici pour taper du texte.

Chantier, atelier, service ► Cliquez ici pour taper du texte.

Période d'exercice ► Du Cliquez ici au Cliquez ici

DOSSIER PROFESSIONNEL (DP)

5. Informations complémentaires *(facultatif)*

Cliquez ici pour taper du texte.

Activité-type 3

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

Exemple n°2 ► *Model View Controller*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Le modèle MVC (Modèle-Vue-Contrôleur, en français) est un modèle de conception logiciel (design pattern en anglais) qui sépare une application en trois composants principaux : le Modèle, la Vue et le Contrôleur. Cette séparation des préoccupations facilite le développement, la maintenance et la testabilité des applications.

1. Modèle

Le modèle représente les données de l'application ainsi que la logique métier. Il gère les données et les règles de l'application. Voici le service « User » :

```
using Requeteur.Models;
using System.Collections.Generic;

namespace Requeteur.Domaine
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Reflection.Metadata.Ecma335;
    using Commun;
    using Commun.Domaine;
    using Requeteur.Entites;
    using Requeteur.Models;

    public class SvcUser : Svc<SvcUserModel, User>
    {
        public Boolean IsAdmin()
        {
            User user = svcEntite.Get($"SELECT * FROM T{nameof(User)} WHERE {nameof(User.UserID)} = '{System.Security.Principal.WindowsIdentity.GetCurrent().Name.Split('\\')[1]}'").Single();
            return Convert.ToBoolean(user.Admin);
        }

        public override SvcUserModel GetById(string UserID)
        {
            if(svcEntite.GetWhereValue("UserID", UserID).Count() > 0)
            {
                User user = svcEntite.GetWhereValue("UserID", UserID).Single();
                return new SvcUserModel()
                { UserID = UserID, Admin = user.Admin };
            }
            return new SvcUserModel() { UserID = UserID, Admin = 0 };
        }
    }
}
```

2. Vue

La vue est responsable de l’affichage des données à l’utilisateur. Elle présente les données du modèle dans un format adapté pour l’interface utilisateur. Voici la page razor pour l’Accueil :

```
@page
<div class="pageText">
  <h1>SQLTheque</h1>

  <h2>Scripts</h2>
  <ul>
    <li>
      <a href="#/Script">
        <span id="nbrScript"></span>
      </a>
    </li>
  </ul>
  <h2>Utilisation</h2>
  <p>L'application affiche ses données dans des tableaux. Voici les principes de
fonctionnement utiles à savoir :</p>
  <ul>
    <li>
      <p>Les entêtes de colonnes permettent le tri sur une colonne (par ordre
ascendant ou descendant), ainsi que le filtrage par saisie ou sélection des valeurs à
afficher.</p>
    </li>
    <li>
      <p>Pour modifier une donnée, double-cliquer sur la ligne : ses colonnes
modifiables s'ouvrent à la saisie.</p>
    <ul>
      <li>pour enregistrer la saisie terminée : presser la touche Entrée, ou
cliquer sur une autre ligne du tableau. Une notification vous confirmera le bon
enregistrement.</li>
      <li>pour annuler la saisie : pressez la touche Echap et/ou rafraichir la
page.</li>
    </ul>
    </li>
    <li>
      <p>Pour ajouter une nouvelle ligne de données, il suffit de la saisir dans
la ligne blanche affichée tout en bas.</p>
    </li>
    <li>
      <p>Pour supprimer une ligne, la sélectionner et pressez la touche Suppr.</p>
    </li>
  </ul>
  <p>Les pages peuvent être mises dans les favoris du navigateur pour y revenir
ultérieurement (toute saisie en cours non enregistrée sera toutefois perdue).</p>
  <p>Certaines pages ne permettent que la consultation, elles se remarquent
visuellement par l'absence de ligne blanche en bas.</p>
</div>
<script type="module">
  import * as page from "../js/AccueilController.js";
  var controler = page.AccueilController.instance;
  controler.init();
</script>
```

Le fichier TypeScript qui génère le code JS pour constituer le contenu « réel » de la razor page :

```
/// <reference types="jquery" />

import * as Application from "../Commun/Application.js";
import * as AjaxPage from "../Commun/AjaxPageController.js";
import * as Requeteur from "../Requeteur.Models.js";
import header = Requeteur.Accueil model ressources;

export class AccueilController extends AjaxPage.AjaxPageController {

    // Singleton
    private static _instance: AccueilController;
    public static get instance(): AccueilController {
        if (!AccueilController._instance) {
            AccueilController._instance = new AccueilController();
        }
        return AccueilController._instance;
    }

    constructor() {
        super();
        Application.startMethod('gridController.constructor()');
        this.title = Requeteur.Accueil_model_ressources.get("_PageTitle_");
        this.defaultFetchQuery = "/GetById?id=0";
        Application.endMethod();
    }

    init(): Promise<void> {
        Application.startMethod('AccueilController.init()');
        var result = new Promise<void>((resolve, _reject) => {
            Application.getData(this.fetchUri())
                .then(data => {
                    var model: Requeteur.Accueil_Model = data;
                    $("#utilisateur").html(model.Utilisateur.toString());
                    $("#nbrScript").html(model.NbrScript.toString() + " " +
header.get(Application.nameof<Requeteur.Accueil_Model>("NbrScript")));
                }).catch(r => {
                    Application.showMessageBox(r, "Incident au chargement");
                });

                resolve();
            });
        Application.endMethod();
        return result;
    }
}
```

DOSSIER PROFESSIONNEL (DP)

3. Contrôleur

Le contrôleur gère les requêtes utilisateur et les interactions. Il récupère les données du modèle, les traite si nécessaire, et passe les résultats à la vue pour l’affichage. Voici le contrôleur « User » :

```
[Route("/[controller]")]
[ApiController]
public class UserController : FacadeWeb
{
    private static readonly Lazy<ILog> _log = new(() =>
log4net.LogManager.GetLogger(System.Reflection.MethodBase.GetCurrentMethod().DeclaringType))
;

    protected override ILog Log => log.Value;

    private readonly SvcUser service = new();

    /// <summary>
    /// Accueil
    /// </summary>
    /// <param name="id">Identifiant (inutilisé).</param>
    /// <returns>Modèle.</returns>
    [HttpGet]
    [Route("GetByUserID")]
    public IActionResult GetByUserID(string userID) => this.GetWhereValue("UserID", userID,
service);

    [HttpGet]
    [Route("IsAdmin")]
    public Boolean IsAdmin() => service.IsAdmin();
}
```

Le modèle MVC permet une séparation claire des préoccupations, facilitant le développement et la maintenance des applications. En intégrant des recommandations de sécurité à chaque niveau du modèle MVC, nous assurons la protection de notre application contre diverses menaces tout en offrant une expérience utilisateur fiable et sécurisée.

2. Précisez les moyens utilisés :

Cliquez ici pour taper du texte.

3. Avec qui avez-vous travaillé ?

Cliquez ici pour taper du texte.

4. Contexte

Nom de l’entreprise, organisme ou association ► Cliquez ici pour taper du texte.

Chantier, atelier, service ► Cliquez ici pour taper du texte.

Période d’exercice ► Du Cliquez ici au Cliquez ici

5. Informations complémentaires (facultatif)

Cliquez ici pour taper du texte.

DOSSIER PROFESSIONNEL (DP)

Titres, diplômes, CQP, attestations de formation

(facultatif)

Intitulé	Autorité ou organisme	Date
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.

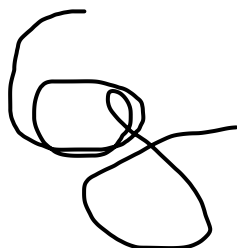
DOSSIER PROFESSIONNEL (DP)

Déclaration sur l'honneur

Je soussigné(e)*Georges GRIGNARD*..... ,
déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je
suis l'auteur(e) des réalisations jointes.

Fait à ..*Saran*..... le ..*17/07/2024*.....
pour faire valoir ce que de droit.

Signature :



DOSSIER PROFESSIONNEL (DP)

Documents illustrant la pratique professionnelle

(facultatif)

Intitulé
Cliquez ici pour taper du texte.

ANNEXES

(Si le RC le prévoit)