Mes réponses aux questions théoriques de devweb avancé

1. *Présentez l'architecture fondamentale d'une application ASP.NET Core. Qu'est-ce que le « kestrel », un « middleware », un « pipeline de middlewares » et un « middleware terminal » ? Quel middleware terminal utilise-t-on pour une application de « pages Razor » ? Comment créer un middleware customisé et l'utiliser dans une application ASP.NET Core ?*

Une application ASP.NET Core peut être distinguée en plusieurs composants :

Le **kestrel** qui est un serveur web qui va convertir les requètes HTTP en objet HTTPContext et qui transmettra les réponses comme des réponses HTTP

Le **pipeline de middleware** qui est décrit dans le fichier `Program.cs` et qui va permettre de traiter les requètes. Chaque middleware va faire un traitement particulier (par exemple servir des ressources statiques, vérifier que l'utilisateur·ice peut avoir accès à la ressource, router les requètes, etc).

Chaque middleware peut ainsi modifier la requète et la passer au middleware suivant de la pipeline, ou alors retourner une réponse au middleware suppérieur jusqu'a être transmit au kestrel qui s'occupera de retourner la réponse.

Le dernier middleware de la pipeline est appellé un **middleware terminal**, par exemple dans le cas de l'application présente, le middleware terminal est celui qui va gérer les page Razor (`MapRazorPages`)

Un **middleware** peut être créé comme une simple classe C# qui implémente l'interface `IMiddleware` qui va ainsi lui définir une méthode asynchrone prenant en argument le middleware suivant de la pipeline et la requète/réponse HTTP.

On peut ensuite définir une méthode d'extension pour ajouter une méthode sur l'objet ApplicationBuilder afin d'utiliser la classe de middleware.

Une fois la méthode d'exntesion créée il suffit alors de l'appeller dans le fichier Program.cs afin d'activer le middleware.

2. *Présentez la notion d'environnement d'exécution. Comment spécifier une configuration spécifique à un environnement en dehors du code ? Comment adaptez le « pipeline de middlewares » à un environnement ? Proposez un exemple concret d'adaptation.*

L'environement d'exécutions peuvent être défini dans les fichiers de configuration `appsettings.json` ou `appsettings.Development.json`. Il peut parfois être aussi préciser via des options en ligne de commande ou des variable d'environement.

La pipeline de middleware peut être adaptée en fonction des variables d'environement. Par exemple en créant une condition qui dit que si l'environement est "production" alors utiliser rediriger les pages d'erreurs interne vers la page d'erreur 500 ou encore activer HSTS.

3. *Un modèle de page remplit deux responsabilités. Citez-les et expliquez les aspects d'un modèle de page qui répondent à chaque responsabilité. Citez la classe dont hérite un modèle de page, les propriétés et les méthodes dont vous héritez. Définissez la notion de méthode de gestion et proposez quatre exemples de signatures conformes.*

Les deux responsabilités d'un PageModel sont d'actualiser le modèle métier (la base de donnée via des repository par exemple), en ce point il fonctionne comme un contrôleur.

La deuxième responsabilité est de préparer et fournir les données dont la vue a besoin (en ce point il fonctionne comme un modèle de vue).

La classe dont tous les modèles de page hérite est la classe `PageModel`. Parmis les propriétés héritée de cette classe on trouve par exemple ViewData (qui permet de passer certaines informations générales à la vue), ModelState (qui permet de vérifier l'état d'un modèle par rapport à certaines contraintes), User (qui permet de récupérer des informations sur l'utilisateur actuellement connecté).

Pour ce qui est des méthodes héritée, on trouve par exemple `RedirectToPage()` (pour rediriger vers une autre page de l'application), `StatusCode()` pour retourner une erreur avec un certain code ou encore `Page()` pour retourner la page actuelle.

Pour qu'un modèle de page soit vraiment utile il faut également lui définir certaines méthodes de gestion. Celles-ci peuvent être tel que `OnGet`, `OnPost`, `OnDelete`, `OnPut` ou des variantes asynchrone tel que `OnGetAsync`

Il est aussi possible d'utiliser des méthodes de gestion nommées tel que `OnPostRegister` qui permette d'avoir plusieurs méthodes de gestion d'un même type pour différentes fonction. En pratique, un paramètre `handler` dans l'URL afi nde pouvoir identifier quel méthode utiliser.

4. *Quel est l'objectif du « model binding » (liaison du modèle) ? Sur quoi se base-t-il pour atteindre son objectif ? Expliquez comment lier une requête à un modèle d'entrée pour les valeurs simples et des valeurs complexes. Comment vérifier la validité des données côté serveur et côté client (précisez les JavaScript nécessaires) ?*

La liaison au modèle est un processus qui récupère les données issues de plusieurs source (contenu de l'URL, paramètre de la requète, etc) pour les affecter à des cibles (méthode de gestion des pages).

Il est possible de lier les données à des types simple par exemple des strings ou des nombres. Il est possible de les lier avec des `BindProprety` ou encore en les passant comme information dans l'URL et en les récupérant comme argument dans la méthode de gestion correspondante.

Mais il est également possible de lier des données à des types plus complexe tel que des formulaire (classes) entières ou des listes d'éléments.

Il est possible de valider les différentes propriétés en utilisant des attributs de navigation tel que `[Required]`, `Range()`, etc.

La validation coté cliente se fait elle grâce à des scripts jQuery nécessaire à importer qui va permettre de récupérer automatiquement le messgae d'erreur à afficher pour chaque champ. Cette validation n'est cependant pas suffisante pour valider les données.

Il faut également vérifier le modèle avec `ModelState.IsValid` afin de vérifier coté serveur si les données sont valide.

5. *Qu'est-ce que Razor ? Quelles sont les directives qu'une vue d'un modèle de page doit mentionner pour être considérée comme telle ? Comment définir un contenu commun pour tout le site ? Comment introduire les parties spécifiques à chaque page dans ce contenu commun ? Comment définir d'autres parties spécifiques et y injecter du code ? Comment factoriser le code dupliqué au sein d'une même vue ou dans plusieurs vues ? Comment passer des données à ce code factorisé ?*

Une page razor est un couple vue razor (fichier .cshtml) et modèle de page (.cshtml.cs). Pour qu'une vue soit considérée comme telle elle doit mentionner `@page` pour indiquer qu'elle est une page, ainsi que `@model` pour précisre le modèle de page qu'elle va utiliser. Eventuellemnet elle peut aussi utiliser `@using` pour importer des espace de nom C#.

Pour qu'une partie de vue soit défini comme commune à tout le site (par exemple pour créer un header) on peut utiliser le fichier `Shared/_Layout.cshtml` qui a été automatiquement généré. Dans ce fichier y est défini une instruction `@RenderBody` qui sera là où le contenu des autres page va être placé.

Razor sait que c'est le fichier _Layout qu'il doit utilisé car il a été précisé dans le fichier `_ViewStart.cshtml`.

Il est également possible de générer plus de section que simplement le body en utilisant l'instruction `@await RenderSectionAsync` ou `@RenderSection`. Les pages pourront alors utiliser l'instruction `@section` pour préciser une section spécifique de la page.

Enfin pour factoriser le code on peut utiliser des vues partielles qui sont des vues Razor qui n'ont pas la définition `@page` et qui peuvent être intégrée dans une page Razor en utilisant une balise `<partial>`. Ces vues partielles peuvent égalemnet utilser leur propre modèle, on il faudra donc que la vue de la page crée des objets de modèle de vue partielle pour pouvoir lui passer les données nécessaires.

6. *Qu'est-ce qu'un assistant de balise ? Proposez au moins deux exemples d'assistant de balise, dont un qui génère de nouveaux éléments HTML. Comment lier un champ de saisie d'un formulaire à une propriété d'un modèle de page ou de vue ? En particulier, comment traiter les options d'une « combo box » ou d'une « list box ».*

Les assistants de balise automatise la génération de certain code HTML, par exemple le tag `asp-page` sur un élément de lien permet de directement lier une page Razor à un lien, lorsque la page sera générée cet élément se fera transformé en élément `href` qui pointera vers l'URL de la page en question.

Un autre exemple de ceci est le tag `asp-for` qui permet de récupérrer une propriété bindée par un `BindProperty` dans le modèle. C'est avec cet élément que l'on peut ainsi lier un champ de saisie de formulaire à un champ articulier dans un modèle de page (ou de vue partielle).

Pour pouvoir gérer des champs de saisie pouvant retourner plusieurs valeurs, on peut simplement les lier à des collections dans le code du modèle de page, par exemple à une liste.

7. *Qu'est-ce que « Entity Framework » ? Quel rôle joue le contexte de base de données ? Que représente un DbSet ? Qu'est-ce qui distingue un modèle de données relationnel d'un modèle métier objet ? Proposez un exemple de concret de modèle métier donnant un modèle de donnée différent. Qu'est-ce qu'une migration ? Comment migrer ? Citez au moins deux arguments utiles pour faire une migration.*

Entity Framework (EF) est un ORM (Object Relational Mapping) qui permet de faire le pont entre les objets du code C# et la base de donnée. Elle permet ainsi d'abstraire une grande partie du fonctionnement interne du stockage des données et ainsi permettre un accès et une écriture plus simple et plus sûr dans la base de donnée que si cela était fait manuellement avec des requètes SQL.

Le contexte de la base de donnée est l'objet représentant la connexion à la base de donnée, c'est par ce dernier que l'on peut communiquer avec la base de donnée depuis le code C#.

Un DbSet permet de lier une classe à une table.

La différence principale entre les modèles relationnels et objet est que les modèles relationnels (tables) fonctionne généralement sur base de clé étrangères. Ainsi si un objet est lié à plusieurs, il y aura une table qui fera la liaison entre les deux objets.

Tandis que dans un modèle relationnel, le premier objet aura simplement une liste de l'autre objet.

Une **migration** est un moyen pour permettre à la structure/schéma des données d'évoluer avec le temps. Ainsi la migration va créer les tables et éventuellement ajouter les données initales (seeding) et lors des version suivantes de l'application d'autres migrations vont s'ajouter par dessus pour modifier le schéma en se basant sur les migrations précédentes.

Cela permet ainsi de ne pas briser la compatibilité avec les versions antiérieures lors de mises à jour.

La migration se crée et s'exécute simplement en lançant une commande dans le terminal une fois dans le projet.

8. *Qu'est-ce que « Identity » ? Expliquez les notions d'authentification et d'autorisation. Comment restreindre l'accès d'une page aux utilisateurs authentifiés ? Comment restreindre l'accès d'un dossier aux utilisateurs authentifié ? Comment restreindre l'accès à une page sur base des rôles ? Comment combiner les rôles pour affiner les règles d'accès ?*

Identity est un système d'identification et d'autorisation. C'est à dire qu'il permet d'authentifier les utilisateurs (c'est-à-dire la connexion) et l'autorisation qui consiste à restreindre certaines pages à certains groupes d'utilisateur·ice·s.

Il est très simple de restreindre l'accès aux utilisateurs authentifiés, il suffit d'ajouter `[Authorize]` au dessus de la classe de modèle de page correspondante.

Il est également possible de limiter l'accès a des dossiers en le précisant dans le fichier `Program.cs` via des conventions.

Il est également possible de restreindre l'accès sur base de rôles en précisant simplement le rôle dans la clause d'autorisation : `[Authorize(Roles = "Teacher")]` ce rôle est lui-même défini.

Il est également possible de créer des Policy (politiques) afin de combiner des rôles.

9. *Présentez l'attaque « Over-Posting ». Donner les grands principes de l'attaque. Illustrez une attaque par un exemple concret. Présentez les mesures correctives en ASP.NET Core. Illustrez par un exemple concret.*

L'overposting est un type d'attaque qui consiste à envoyer plus de donnée que demandé pour modifier des propriétés qui ne sont pas sensées être modifiées. Par exemple, si un modèle de page a tous ses attributs en `[BindProperty]` ou sur base de méthodes d'action spécifiques (`OnPostUserCreate`), si l'attaquant envois plus de donnée de façon à inclure des champs correspondant aux attributs du modèle de page, ceux-ci se feront automatiquement remplacé, ce qui est déjà très problématique.

Ce problème devient encore plus problématique si des échange/modifications du modèle métier sont faites sur base de ces attributs.

Pour prévenir cette attaque on peut utiliser des modèles de vues afin de vérifier les données entrées, faire attention aux champs qui sont lié à la page et s'assurer que tous les champs non modifiable ne sont pas bindé à la vue.

10. *Présentez l'attaque « Open-Redirect ». Donner les grands principes de l'attaque. Illustrez une attaque par un exemple concret. Présentez les mesures correctives en ASP.NET Core. Illustrez par un exemple concret.*

Pour pouvoir rediriger automatiquement vers la bonne page, par exemple lorsqu'un utilisateur·ice va sur une page non autorisée, il peut être redirigé vers une page de connexion, dans la redirection l'URL vers la page non autorisée est ajoutée.

Ainsi une fois connecté, l'utilisateur est redirigé vers cette URL. Sauf que si un attaquant crée une fausse URL en changeant cette propriété il peut faire en sorte de rediriger l'utilisateur·ice vers un site malveillant à la place.

Une manière de protéger contre ce genre d'attaque peut être de simplement vérifier que l'URL de redirection est bien présente sur le site et n'est donc pas celle d'un attaquant.

11. *Présentez l'attaque « Cross Site Scripting ». Donnez les grands principes de l'attaque. Illustrez une attaque par un exemple concret. Présentez les mesures correctives en ASP.NET Core. Illustrez par un exemple concret.*

Le **Cross Site Scripting** (ou XSS) consiste à injecter du code malveillant (le plus souvent du JavaScript) dans une page web de façon à ce que celui-ci soit compris comme du code à exécuter par le navigateur.

Par exemple si un attaquant entre dans un espace commentaire non protéger quelque chose comme `<script>alert("bonjour tout le monde");</script>`, le code sera ajouté à la page mais sera interpreté comme du JavaScript par le navigateur.

ASP.NET protège par défault contre ce genre d'attaque car tout code HTML est "escaped" par défault de manière à ne pas être interprétée comme du code HTML par les navigateurs.

12. *Présentez l'attaque par Injection Sql. Donnez les grands principes de l'attaque. Illustrez une attaque par un exemple concret. Présentez les mesures correctives en ASP.NET Core. Illustrez par un exemple concret.*

Une attaque par injection SQL consiste à écrire du code SQL dans un champs de formulaire par exemple de façon à ce que celui-ci soit simplement ajouté dans une requète SQL de façon à écrire une requète SQL qui n'est pas sensée être exécutée.

C'est une attaque très grave car elle affecte directement la base de donnée mais est facilement évitable, dans le cas présent avec Entity Framework, le framework protège naturellement contre ce genre d'attaques. D'autres frameworks qui utilisent du code SQL eux utilise des requètes préformattées qui empèche de créer des requètes malicieuses.

13. *Présentez l'attaque « Cross Site Request Forgery ». Donnez les grands principes de l'attaque. Illustrez une attaque par un exemple concret. Présentez les mesures correctives en ASP.NET Core. Illustrez par un exemple concret.*

L'attaque **Cross Site Request Forgery** (ou CSRF, ou XSRF) consiste à faire un script qui consiste à faire exécuter du code malicieux pour créer une requète vers le site depuis un autre site qui lui est malicieux.

Ainsi lorsque l'utilisateur·ice connecté accède au site malicieux, ce dernier va demander au navigateur d'exécuter une requète vers le site déjà connecté pour y faire une opération. Le navigateur le fait et l'opération s'exécute contre le grès de l'utilisateur·ice.

Pour protéger contre ce type d'attaque on peut protéger les cookies de session de façon à les protéger pour qu'il ne soit valable que sur le site en question et sur aucun autre. Une autre manière de protéger est d'utiliser un jeton anti-CSRF.

14. *Présentez l'attaque « Man in The Middle ». Donnez les grands principes de l'attaque. Illustrez une attaque par un exemple concret. Présentez les mesures correctives en ASP.NET Core. Illustrez par un exemple concret.*

L'attaque de **man in the middle** consiste à intercepter une requète pour en extraire les informations sensibles (données de connexion par exemple) pour la rejouer au serveur et ainsi gagner un accès non autorisé.

Pour éviter ces attaques il suffit de s'assurer que les utilisateur·ice·s soient toujours sur une connexion sécurisée (SSL/TLS) et certifiée par une autorité de certification, de cette manière un attaquant ne pourra pas faire l'intermédiaire car il ne pourra pas déchiffrer les requètes.

15. *Présentez l'attaque « Deny of Service ». Donnez les grands principes de l'attaque. Illustrez une attaque par un exemple concret. Présentez les mesures correctives en ASP.NET Core. Illustrez par un exemple concret*

Une attaques par **Déni de Service** (DoS) consiste à submerger le serveur de requètes frauduleuses de façon à l'empécher de traiter de véritables requètes.

Cela peut être simplement un script qui exécute énormément de requètes à la seconde de façon à rendre le site indisponible. Ou encore un botnet qui va faire la même chose depuis plein de machines différentes pour rendre le blocage plus complexe. Il existe encore d'autres types qui sont plus sournois tel qu'envoyer des flux de données très lentement de façon à faire en sorte que le serveur doivent gérer trop de connexions de sessions simultanées, etc.

Protéger contre ce type d'attaque est compliqué et ne peut généralement par être fait qu'au niveau de l'application. Mais il est tout de même possible de déjouer les attaques les plus basiques à l'aide d'un système de Rate Limtiting qui va bloquer les requètes au bout d'un certain débit depuis une machine.

Il est aussi possible d'utiliser des systèmes plus perfectionnés tel que Cloudflare, fail2ban ou Crowdsec afin de protéger l'application.