Denis, Yiqing, Armelle, Lydia

Du 14 au 19 février 2018

Résumé

Support d’accompagnement du projet

Formation à ADO.Net

Projet Final

Table des matières

[1 Introduction (Equipe) 3](#_Toc506571849)

[2 Fonctionnalités de l’application 3](#_Toc506571850)

[2.1 Réservation de chambres (Armelle) 3](#_Toc506571851)

[2.2 Gestion du compte du compte 3](#_Toc506571852)

[2.2.1 Création et modification du compte (Denis) 3](#_Toc506571853)

[2.2.2 Consultation des factures (Yiqing) 3](#_Toc506571854)

[2.3 Gestion de l’hôtel (Lydia) 3](#_Toc506571855)

[3 API Web 5](#_Toc506571856)

[4 Conclusion (Equipe) 5](#_Toc506571857)

# Fonctionnalités de l’application

## Réservation de chambres (Armelle)

A la page « Réservations », nous affichons tout d’abord le formulaire qui permet au client de remplir les informations correspondant à la réservation qu’il voudrait effectuer. Ensuite, celui-ci est redirigé vers la page « liste de chambres disponibles » qui affiche la liste de chambre disponible avec des lits supérieur ou égal au nombre de personnes saisie et s’il n’y a aucune chambre de disponible il est redirigé vers une page qui le lui indique. Ensuite, le client doit se rendre dans la page « détails » pour pouvoir afficher le détail de la chambre sélectionné (numéro de chambre, étage, bain, douche, WC, prix unitaire, prix HT) et l’utilisateur peut retourner dans la liste précédente en cliquant sur « Retour ». Ensuite, l’utilisateur pourra réserver une chambre en cliquant sur détails dans la page « Details » et sera rediriger vers la page de connexion ou d’inscription à un compte client, et après s’être connecté ou inscrit il sera dirigé vers la page qui recapitule sa réservation et qui le remercie de s’être inscrit.

Nous décrivons ci-dessous l’algorithme et choix techniques.

**Algorithme et choix techniques**

1. **Dans l’action « Index » du contrôleur**

* Affiche le formulaire qui permet au client de saisir les informations sur la réservation.

1. **Dans l’action « VerifDisponibilite » du contrôleur**

* Nous avons décidé de récupérer via une requête SQL la liste des chambres qui ne respecte pas les besoins du client et de déduire celle-ci de la liste totale des chambres correspondante aux besoins du client en termes de nombre de lits et de jour disponible (nous avons décidé d’utiliser les requêtes SQL car nous avions plusieurs jointures à faire et il est plus aisés de le faire avec des commandes SQL et si on sait exactement ce que l’on envoi au serveur). On aurait aussi pu le faire avec linq to Entities mais nous aurions chargé plus d’informations en mémoire.
* Nous avons utilisé un View Model « ReservationVM » et des ViewBag pour pouvoir passer des informations de notre contrôleur vers les vues associés. Nous avons décidé de faire ces deux choix pour appliquer les enseignements reçus et monter que le passage des informations du contrôleur vers la vue pourra se faire via plusieurs méthodes.

1. **Dans l’action « Details » du contrôleur**

* Nous avons choisi de mettre notre objet réservation dans une session car nous voulions passer les informations de cet objet de l’action « Details » à l’action « Creates » et aussi permettre au contrôleur client de pouvoir faire la redirection après que l’utilisateur qui souhaite effectuer une réservation ai créé son compte, cet objet est récupéré dans l’action « Creates » et enregistrer en base de données. L’objet Session nous a ainsi permis permettre de stocker temporairement pendant toute la durée de vie de la session) des informations transmissibles de page en page. Le mécanisme utilisé par l'objet Session pour permettre de transmettre les informations est basée sur l'utilisation de cookies, mais il est réalisé de manière transparente, ce qui signifie que l'utilisateur n'a pas à s'en soucier.

1. **Dans l’action « Creates » du contrôleur**

* Nous récupérons l’objet Réservation de la session et nous faisons une boucle sur le nombre de nuit saisie pour pouvoir enregistrer la réservation du client sur toute la durée de son séjour.
* Nous avons été obligés d’enregistrer la réservation dans la boucle car après le premier ajout dans le contexte les autres ajouts écrasaient la valeur précédente (pourtant les dates étaient différentes à chaque fois) et nous n’avons pas pu identifier la raison.
* Enfin nous supprimons notre objet réservation de la session pour permettre à un utilisateur lambda de se connecter à son espace client sans passer par le contrôleur « Réservations ».

1. **Dans l’action « VerifConnexion » du contrôleur**

* Pour qu’un utilisateur n’ai plus à se connecter quand il veut faire une réservation. Cette action vérifie si l’utilisateur est connecté. Et s’il n’est pas connecté il le revoit vers la page de connexion.

### Création et modification du compte (Denis)

De manière générale, il reste encore des points à traduire en français.

#### Clients

Il y a 2 niveaux pour le compte client : l’authentification sur l’application (pour devenir utilisateur) et l’enregistrement dans la base de données (pour devenir client).

##### Utilisateur :

###### Devenir Utilisateur :

Pour devenir s’enregistrer, le futur utilisateur doit renseigner son adresse mail et un mot de passe. Comme demandé, le **mot de passe** contient entre 6 et 100 caractères, avec au minimum 1 majuscule et 1 minuscule. **L’adresse mail** doit contenir 2 parties, séparées par un « @ », et doit être unique à chaque utilisateur. Après la création du compte utilisateur, l’utilisateur est invité à renseigner ses coordonnées (voir partie « devenir client »).

Les anciens clients peuvent devenir utilisateur en s’enregistrant normalement. Ils doivent cependant donner la même adresse mail que celle qui est associée au compte client. Ils seront redirigés vers la page de mise à jour de leur compte client afin qu’ils renseignent les éléments nouveaux.

###### Déjà utilisateur :

L’utilisateur peut aussi se « logger » s’il a déjà un compte utilisateur. S’il n’a pas de compte client, il sera redirigé pour « devenir client », sinon il sera redirigé vers la page à laquelle il voulait accéder (page précédente).

L’utilisateur ne peut pas modifier son adresse mail ; c’est un point à améliorer en utilisant temporairement l’ID pour identifier le client. Aussi le nom d’utilisateur est l’adresse mail ; on pourrait donner le moyen à l’utilisateur d’entrer un nouveau nom.

##### Client :

Concernant la gestion des clients, nous avons choisis de séparer les parties clients, adresse et téléphones. Les 2 derniers pouvant être « non renseigné », cela permet de faciliter l’affichage et la gestion des validations, particulières à chaque partie.

Nous avons choisi d’enregistrer les noms de famille et la première lettre du prénom en majuscule, cela permet de faciliter la recherche des clients par leurs noms.

###### Devenir Client :

Pour devenir client, il faut donc déjà être utilisateur (sauf cas particulier des anciens clients). En effet, nous utilisons l’adresse mail du l’utilisateur pour l’associer au compte client. Ainsi, comme il n’existe qu’un ID par client, il n’existe qu’une adresse Mail par client.

Pour s’en assurer, nous avons bloqué la création de client si **l’adresse mail** de l’utilisateur qui lance l’action est déjà dans la BDD. Ainsi il n’est pas encore possible pour un manageur, ayant une adresse unique, de créer plusieurs clients.

L’**adresse** est facultative et tout (Rue, Ville, CP) doit être renseigné pour enregistrer l’adresse ; un autre point à améliorer, il devrait être possible de renseigner seulement certains champs.

Le client peut avoir au maximum 2 **numéros de téléphone** : un portable et/ou un fixe. Il faudrait plus tard mieux gérer la liste des téléphones afin de lui permettre d’ajouter un plus grand nombre de numéros. Les numéros fixes commencent par 01 à 06 et les numéros portables par 06 ou 07. Il ne peut y avoir plusieurs téléphones ayant le même numéro ; ainsi on s’assure avant d’enregistrer que le numéro n’existe pas et que les numéros entrés par le client ne sont pas les mêmes. On s’assure aussi que les numéros sont composés de 10 chiffres (convention des numéros français). Un nouveau client ne peut pas créer un compte avec aucun numéro de téléphone. Au moins 1 numéro doit être renseigné correctement pour pouvoir créer le compte ; actuellement si un numéro est bien renseigné et que l’autre champs n’est pas vide mais mal renseigné, on empêche la création du compte. Il serait intéressant de pouvoir valider un numéro même si l’autre numéro est mal renseigné (et simplement prévenir l’utilisateur).

Les erreurs sont affichées en haut, cependant l’affichage pourrait être mieux géré.

#### Déjà Client :

Il existe 2 types de clients : ceux qui ont créé leur compte comme présenté ci-dessus, et les anciens clients déjà présents dans al BDD. Les nouveaux clients ont normalement tous les champs obligatoires renseignés (dont au moins un téléphone). Les anciens clients ne sont pas passés par la nouvelle validation et peuvent ainsi avoir des champs manquants. Dans tous les cas, il est possible de voir les informations du compte dans la rubrique « modifier son compte ».

Il faudrait aussi orienter les anciens utilisateurs vers leurs profils sans passer par « création client ».

#### Cas spécifique de la réservation en cours :

Il est possible d’effectuer une réservation sans être ni utilisateur, ni client. Dans ce cas, le client sera redirigé vers la création de la réservation, après avoir été enregistré comme client.

#### Mon compte :

L’interface permet de voir les informations de l’utilisateur (Profil), modifier le mot de passe ou encore modifier le compte.s

La modification du compte permet de mettre à jour les informations du compte. Seuls les clients peuvent y accéder ; les utilisateurs non enregistrés comme clients sont invités à saisir leurs coordonnées. Les validations sont les mêmes que pour la création d’un client à la différence que les informations liées au client (nom, prénom, etc.) sont mises à jour même si les téléphones sont invalidés par les vérifications internes (numéro déjà existant). Cela permet au client de ne pas retaper tous les champs si seulement les téléphones ne sont pas validés.

Les erreurs sont affichées en haut du formulaire avec une couleur rouge ; les autres informations sont en vert.

Aussi, les informations sont systématiquement mises à jour, qu’elles aient été modifiées ou non ; il serait intéressant de mettre à jour seulement les données modifiées par le client.

### Consultation des factures (Yiqing)

La page « Consultation des factures » permet l’utilisateur de consulter ses factures : date de la facture, date de paiement, mode de paiement ainsi que le montant total de chaque facture. L’utilisateur peux aussi consulter le détail de chaque facture (quantité, montant HT, Taux de TVA, Taux de réduction, le prix correspondant) en cliquant sur un bouton « Détail ». Ensuit l’utilisateur peut retourner dans la liste précédente en cliquant sur « Retour »

Pour avoir l’accès à ses factures, le client est invité de se connecter s’il a déjà un compte, ou bien créer un compte pour un nouveau client.

Cette page est également accessible depuis un sous-menu dans « Mon Compte » pour l’utilisateur authentifié.

Nous décrivons ci-dessous l’algorithme et choix techniques.

**Algorithme et choix techniques**

Nous avons ajouté un contrôleur « FacturesController ».

1. **Dans l’action « Index » du contrôleur**

* Si l’utilisateur n’est pas connecté :

La page Index va réorienter vers la page « Login /Register ».

* Si l’utilisateur est connecté :

La page Index va réorienter vers la page « Afficher ».

1. **Dans l’action « Afficher » du contrôleur**

Nous identifions **Id** de client lié au compte connecté. La page « Afficher » afficher une liste de facture trié par Id, par année et par date décroissante. L’action Afficher prends en paramètres « AnneeSelected » pour triée les factures selon l’année. Par défaut, on lui affecte l’année « 2018 » pour qu’elle affiche les factures de l’année en cours.

Pour récupérer la liste d’année, nous avons créé une liste en ajoutant l’année en cours (2018), et puis ajouté les autres l’année dans lesquelles le client a eu des factures. La liste d’année est enregistrée et envoyé à la vue Afficher dans un ViewBag.

Pour envoyer l’année sélecté dans la select List, nous avons été obligés de créer un View Model FactureVM, qui contient ce paramètre « AnneeSelected » comme une propriété.  Nous avons utilisé le tag-helper pour envoyer l’année choisi de la vue au contrôleur.

1. **Dans l’action « Details » du contrôleur**

L’action Details prends deux paramètres (id de la facture et l’année sélectionné). L’Id de la facture nous permets d’afficher toutes les lignes de cette facture (la quantité, le montant HT, le taux TVA, le taux de réduction et le prix de chaque ligne facture). Nous récupérons le paramètre l’année sélectionné et renvoyer à l’action « Affichage », cela nous permet de retourner dans la liste précédant.

1. **L’accès depuis un sous-menu de menu Mon Compte**

Nous ajouton également un accès à la page consultation factures dans le menu « Mon Compte ». Le client peut consulter ses factures en cliquant sur un bouton « Mes Factures » qui lui réorienter vers la page Index de factures.

## Gestion de l’hôtel (Lydia)

Cette section se focalise sur la gestion interne de Grand Hôtel. Un menu Gestion interne est ajouté dans l’application web. Ce dernier contient deux pages : la liste des clients et la liste de chambres.

### Liste des clients

La page « Liste des clients », contient la liste de tous les clients de Grand Hôtel (id, civilité, nom, prénom, email), leurs nombres totaux de réservations effectuées, ainsi que leurs nombres de réservations en cours. L’affichage se fait selon la première lettre du nom de famille des clients.

Pour réaliser cette page, nous ajoutons un contrôleur « CLientsController » dans le dossier « Controller » de la solution ASP.Net de l’application web. Nous générons simultanément toutes ses vues dans un sous-dossier « Clients » du dossier « View ». Le nom du sous-dossier permet de lier le contrôleur à ses vues.

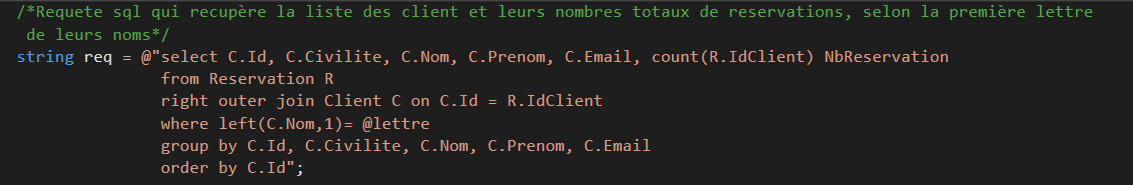
Dans l’action « Index » de ce contrôleur, les données sont récupérées du model « Client, Reservation », les traitements sont faits, et une réponse est envoyée à la vue « Index » de ce même contrôleur sous forme d’une liste des objets « Client ».

Nous décrivons ci-dessous l’algorithme et les choix techniques.

**Algorithme et choix techniques**

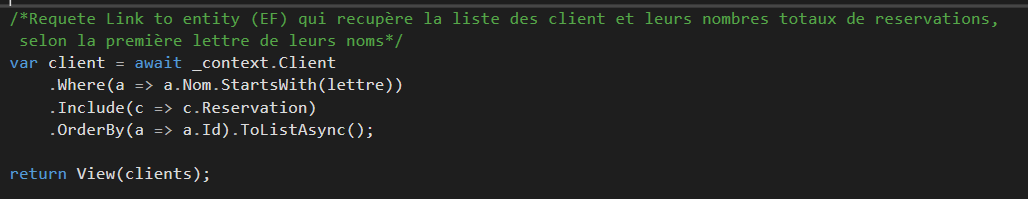
1. **Dans l’action « Index » du contrôleur**

* L’action « Index » prend en paramètre un caractère, par défaut on lui affecte la première lettre de l’alphabet « A ». Cela permet d’éviter d’une part de passer un caractère « null » et d’autre part, d’afficher une page vide à l’ouverture.
* On récupère les données des clients (**id, civilité, nom, prénom, email, total de réservations effectuées**) selon la première lettre de leurs noms, à l’aide d’une requête SQL (ADO.NET approche SQLCommande), la lettre est passée en paramètre. Ce choix est fait afin d’optimiser le chargement des informations en mémoire.

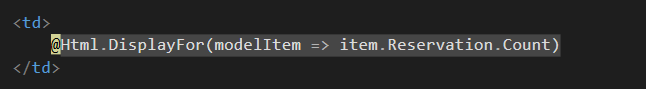


On pourrait faire ce traitement facilement dans l’approche « Database First avec Entity Framework ».

En effet, un client possède une propriété Reservation, ainsi pour récupérer son nombre total de réservation, on charge la liste des réservations en appelant la méthode Include() :

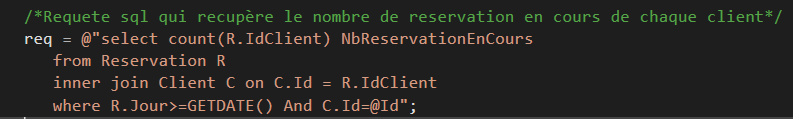


Dans la vue, on compte le nombre total avec la propriété Reservation.Count.



L’inconvénient de cette méthode réside dans la surcharge d’un grand nombre de données en mémoire, si on a un nombre élevé de clients et de réservations. Cela affectera la performance de l’application.

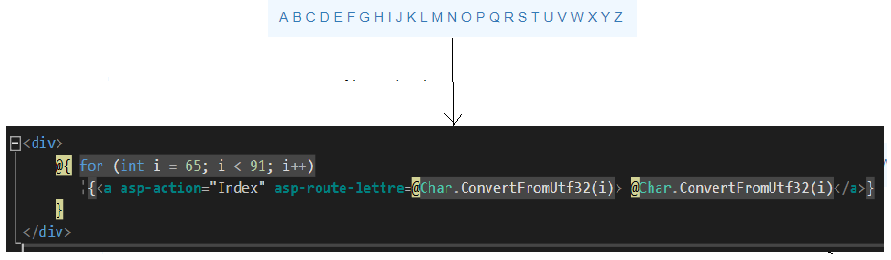
* On récupère une connexion à l’aide de la méthode  \_context.Database.GetDbConnection() : cette dernière permet à l’application de se connecter à la base de données. Elle est placée dans une instruction « using ». A la fin de son bloc, l’instruction « using » appelle la méthode « dispose » qui fermer automatiquement la connexion.
* On crée une commande (avec « SqlCommand »), on affecte à cette commande le code SQL et la connexion.
* On crée un paramètre (avec « SqlParameter ») et on l’ajoute à la commande.
* On ouvre la connexion.
* La commande définie ci-dessous, est exécutée et placée dans un objet de la classe « SqlDataReader »
* On récupère les lignes de résultat une par une, et on ajoute une autre requête SQL qui récupère **le nombre de réservations en cours** de chaque client.



Chacune de ces lignes est placée dans un objet « Client ». Le nombre de réservations en cours est affecté à une propriété de cet objet. On ajoute ensuite cet objet à une liste et celle-ci est envoyée à la vue au moyen de la méthode « View ».

1. **Dans la vue « Index » du contrôleur**

* Dans la vue, on ajoute un bloc de code « Razor » permettant de générer les 26 liens hypertextes correspondant aux lettres de l’alphabet. Ces lettres sont générées à partir de leur code ASCII au moyen de la méthode « Char.ConvertFromUtf32() ».



* On récupère la liste des clients en utilisant le « Model Client ».
* Le moteur « Razor » combine cette vue avec la page de disposition pour fournir la page « Liste des clients ».

### Liste des chambres

Dans la page « Liste des chambres », on affiche les 21 chambres de Grand Hôtel (numéro, étage, nombre de lits, Tarif). Les tarifs sont révisés au premier janvier de chaque année. On filtre les chambres selon leur état d’occupation (occupée ou libre) à la date du jour. Toutes les chambres sont affichées par défaut.

De la même manière que pour les clients, on ajoute un contrôleur « ChambresController » dans le dossier « Controller » avec toutes ses vues dans un sous-dossier « Clients » du dossier « View ». L’algorithme et les choix techniques sont similaires à ceux de la section [2.3.1](#_Page_liste_des). Cette partie ne sera pas détaillée.

**Algorithme et choix technique**

1. **Dans l’action « Index » du contrôleur**

* Action <<Index>> prend en paramètre une chaîne de caractères.
* On récupère les données des Chambres avec une requête SQL. Le filtrage sur l’état d’occupation des chambres se fait de la manière suivante : par défaut, on affiche toutes les chambres et leur tarif. Si le paramètre passé à l’action index prend la valeur ″Occupe″, on utilise le code SQL qui affiche la liste des chambres occupées et si le paramètres est égale à ″Libre″, on utilise le code SQL qui affiche la liste des chambres libres. Cette méthode nous oblige à insérer trois requêtes SQL, pour optimiser le code, nous avons placé ces requêtes dans trois vues (SQL) créées dans la base de données. Nous avons ensuite effectué une sauvegarde de la base c’est le « SauvegardeGrandHotel.bak ».
* La liste des chambres obtenue est affectée à un modèle de vue « ChambreVM » et est envoyée à la vue au moyen de la méthode « View ».

1. **Dans la vue « Index » du contrôleur**

* On réalise le filtrage en utilisant un formulaire.



* On récupère la liste des chambres en utilisant le « Model ChambreVM ».
* Le moteur « Razor » combine cette vue avec la page de disposition pour générer la page « Liste des Chambres ».

Cette partie de notre application fonctionne bien. Nous avons ajouté une fonctionnalité « Détails » qui permet à l’utilisateur de voir les détails de chaque client ou de chaque chambre. On pourrait ajouter plus de fonctionnalité comme Modifier, supprimer, les données d’un Client (d’une nouvelle chambre), nous ne l’avons pas fait par mesure de sécurité. Nous avons limité l’accès à la base de données car pour l’instant le menu Gestion Interne est accessible par n’importe quel utilisateur authentifié.

# API Web (Denis)

La Web-API permet de voir la liste des clients, créer un client, trouver un client selon l’ID ou son nom et de supprimer un client. Cependant l’affichage des informations clients selon l’ID n’est pas encore abouti : il reste vraisemblablement un problème dans la désérialisation de la réponse. Ainsi nous avons choisi de vous présenter les informations pour l’instant sans téléphone.

Les messages d’erreurs, passés dans les requêtes ne sont aussi pas encore assez clair.

# Conclusion (Equipe)

Dans l’ensemble, l’application répond aux besoins du client. Il reste cependant plusieurs détails à améliorer concernant l’efficacité des requêtes, la gestion des autorisations ou encore les cas particuliers.

L’API-Web/ConsoleTest ne sont prêtes que partiellement par manque de temps. En effet plus des 2/3 des fonctions répondent aux besoins du client. Cependant il reste le dernier point à terminer afin d’avoir une API finalisée.

Nous avions prévu de faire des tests unitaires mais vu l’importance de l’interaction avec le client, les tests sont difficiles à mettre en œuvre.