DOSSIER PROJET

FERGUENIS Bérenger

Développeur Web et Web mobile

Studi

Projet restaurant

Quai Antique

Studi

DIGITAL EDUCATION

SOMMAIRE

Le projet restaurant

Afin de valider l'acquisition des compétences requises pour chacune des Activités Types du Titre Professionnel de Développeur Web et Web Mobile. L'école du groupe STUDI réalise une Évaluation passée en Cours de Formation(Livret ECF).
Pour cet ECF le sujet était de créer une application pour un projet de restaurant en respectant les 8 compétences du titre professionnel.

Pour la partie front-end il y a 4 compétences à développer en intégrant les recommandations de sécurité à savoir le maquettage, l'interface utilisateur web statique et dynamique adaptable ainsi qu'une interface avec une solution de gestion de contenu.

Pour la partie back-end il y a également 4 autres compétences à développer en intégrant les recommandations de sécurité comme créer une base de données, développer les composants aux données, développer la partie back-end d'une application web et web mobile puis élaborer et mettre en œuvre des composants dans une application de gestion de contenu.

Le projet restaurant est une application web vitrine pour le Quai Antique situé à Chambery. Le Chef Arnaud Michant est un passionné des produits et producteurs de la Savoie. Le Quai Antique proposera au déjeuner comme au diner une expérience gastronomique, à travers une cuisine sans artifice. Lors de l'inauguration de son deuxième établissement, le Chef Michant a pu constater l'impact positif que pouvait avoir un bon site web sur son chiffre d'affaires.

J'ai donc pour mission d'honorer les envies et contraintes que me propose ce projet.
Le sujet comprend des User Stories qui doivent être respectées, tout en servant de guide pour structurer efficacement le projet et ses différentes étapes jusqu'à son déploiement.

Résumé du projet restaurant

Le projet restaurant, Quai Antique, est une application web vitrine. L'utilisateur peut consulter les différents plats, menus, photos et faire une réservation. L'utilisateur peut créer un compte afin de gagner du temps lors de ses prochaines réservations avec un système de pré-remplissage lors de la création du compte pour le nombre de convives et les allergies mentionnées. C'est une particularité du site car cela permettra aux utilisateurs réguliers du restaurant de gagner du temps lors de la complétion du formulaire de réservation.

Environnement humain du projet restaurant

C'est un projet que j'ai réalisé seul pour une durée approximative de 2 mois d'une moyenne de deux à quatre heures par jour.
Lors des blocages ou incompréhensions techniques pendant la durée de ce projet, j'ai pu obtenir de l'aide auprès des différents formateurs au sein de l'école STUDI, tel que les cours, le forum sur la plateforme ou bien même lors de live qu'organisaient certains formateurs. J'ai également pu me renseigner auprès des différentes solutions que propose la documentation de l'environnement que j'ai choisis à savoir le framework Symfony. Le site Stackoverflow m'a été d'une grande aide également ainsi que des salons Discord qu'ont organisé certains camarades de ma promotion pour l'entraide de l'Évaluation en Cours de Formation (ECF).

Environnement technique du projet restaurant

J'ai utilisé le framework Symfony pour démarrer mon projet car il offre une solution sécurisée, rapide et moins fastidieuse que le PHP pur. J'ai créé l'application sur la version Symfony 5.4 LTS (Long-term support), qui offre une documentation précise, une bibliothèque de gestion de bases de données Doctrine et un système en ligne de commande pour générer des entités, des contrôleurs et des formulaires.

Symfony propose également une partie Front via son moteur de template Twig et ainsi réduire les risques d'injections de code malveillant avec son système d'échappement automatique des variables. J'ai aussi choisi d'utiliser Twig pour sa facilité d'utilisation, sa flexibilité et son intégration facile avec les autres fonctionnalités Symfony.

Pour le Front-end, j'ai utilisé HTML5 et CSS3, ainsi que Bootstrap v5.3.0 pour sa responsivité et ses composants prédéfinis. Pour une meilleure organisation de mon CSS, j'ai opté pour le préprocesseur SASS et sa syntaxe SCSS qui permet l'utilisation des variables de Bootstrap. Pour la partie dynamique, j'ai choisi le langage Javascript et j'ai utilisé Webpack Encore pour un rendu plus qualitatif.

En ce qui concerne la partie Back-end, j'ai utilisé PHP 8.2.1. J'ai créé ma base de données sur alwaysdata en utilisant le langage SQL via Doctrine et PHPMyAdmin pour visualiser les données. Pour le déploiement, j'ai choisi le service PaaS (Platform As A Service) Heroku pour sa flexibilité et son utilisation simple, ainsi que sa suite logicielle, comme PHP et Composer, et l'utilisation de Git pour le déploiement. Travailler avec le framework Symfony pour ce projet a été comme une évidence pour moi du projet restaurant, Quai Antique.

Conception du projet restaurant

1. Mise en place avant l'initialisation du projet

a) Création d'un Google Drive
Google met à disposition un système de stockage à distance gratuit pour différents projets professionnels ou personnel. Lorsque je devrais conserver les différents fichiers dont je vais avoir besoin pour ce projet, Google Drive me permettra de les conserver en toute sécurité et d'éviter de tout perdre si un incident survenait dans mon ordinateur pendant le développement du projet restaurant. J'ai donc importé les photos que contiendra le site, les différents diagrammes, la charte graphique, le wireframe et les liens pour le GitHub et Trello. Puisqu'il s'agit d'un projet pour une Évaluation en Cours de Formation (ECF), j'ai donc choisi de mettre le Drive en partage pour les personnes ayant l'accès à l'url du Drive. J'aurais très bien pu le mettre en accès privé également.

b) Création d'un Trello
Voici un service de gestion de projet que j'utilise énormément lors du développement de l'application. Le Trello me permet d'avoir une organisation très structurée. C'est un système de cartes et de listes. Il est possible de détailler ou non chaque tâche ou liste. Trello m'offre la fonctionnalité de voir l'avancement de mon projet dans un visuel clair et précis. Je l'utilise du début jusqu'à la fin du projet restaurant.

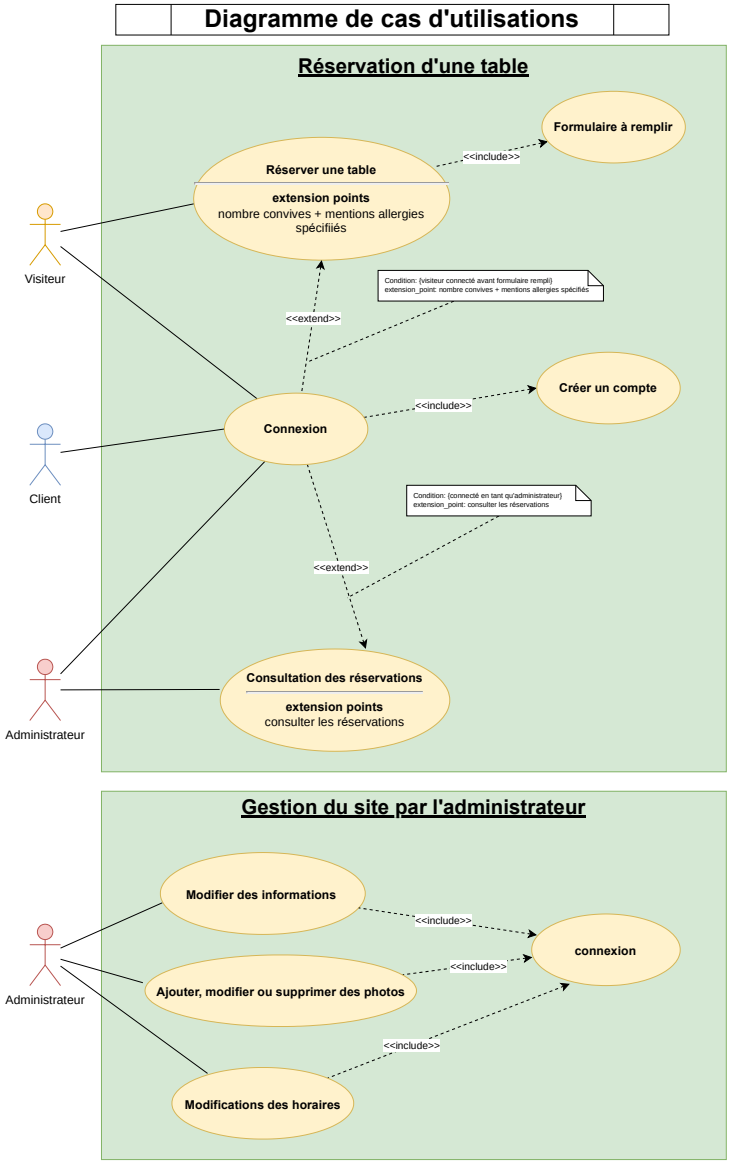
c) Méthode MoSCOW
Dans le Trello, je fais appel à la méthode MoSCOW qui n'est autre que l'acronyme pour Must have this, Should have this if at all possible, Could have this if it does not affect anything et Won't have this time but would like in the future. Traduit en Français cela donne ceux qui doit être fait, devrait être fait dans la mesure du possible, pourrait être fait dans la mesure où cela n'a pas d'impact sur les autres et ne sera pas fait cette fois mais sera fait plus tard.

Pour chaque étiquette dans mon Trello, j'assigne une méthode en fonction des priorités que je vais avoir besoin. Cela me permet d'être en mesure de savoir si une tâche doit être effectué en urgence ou pas.

2. Les diagrammes UML

a) Diagramme de cas d'utilisation

De manière à définir les fonctionnalités de l'application. Le diagramme de cas d'utilisation me permettra d'avoir un premier visuel sur les accès selon les différents utilisateurs du site. Pour le cas du projet restaurant, Quai Antique, il y a trois utilisateurs. Le visiteur qui a un accès sur toutes les fonctionnalités du site sauf la partie d'auto-complétions du formulaire de réservation pour le nombre de convives et les allergies mentionnées. Le client, qui a les mêmes accès qu'un visiteur mais il bénéficie cette fois-ci de l'auto-complétions ainsi que d'un résumé des réservations en cours ou déjà effectués. Le dernier utilisateur est l'administrateur. Il a un accès sur toute la partie administrative du site. Il peut modifier, supprimer ou ajouter du contenu telles que des photos, des plats, des menus et il a la possibilité de consulter les réservations à venir. Je créé le diagramme de cas d'utilisation depuis l'application Draw.io. Ce logiciel est facile à prendre en main et dispose d'un onglet réservé à la phase de conception UML (Unified Modeling Language).



b) Diagramme de séquence

En vue d'illustrer un cas précis, celui d'une réservation. Le diagramme de séquence me permet d'avoir un visuel sur toutes les interactions entre les différents acteurs, comme l'utilisateur visiteur ou client avec la base de données lors de la réservation. J'utilise le logiciel Draw.io, en respectant le langage de modélisation graphique orienté objet. Lorsque le visiteur arrive dans l'espace de réservation, la base est interrogée puis fait le lien pour voir si un compte est déjà créé. En l'occurrence non pour le cas précis du visiteur. Il n'y aura donc pas besoin d'afficher les données du nombre d'invités et les allergies mentionnées. Une fois que le visiteur soumet le formulaire rempli, la base de données est interrogée pour voir si il reste des places disponibles ce jour ou si une heure n'est pas déjà réservée. Pour le cas du client, la base de données est interrogée, il reconnait l'utilisateur grâce à son identifiant et affiche le nombre d'invités et les allergies mentionnées.

	Diagramme de séquence	

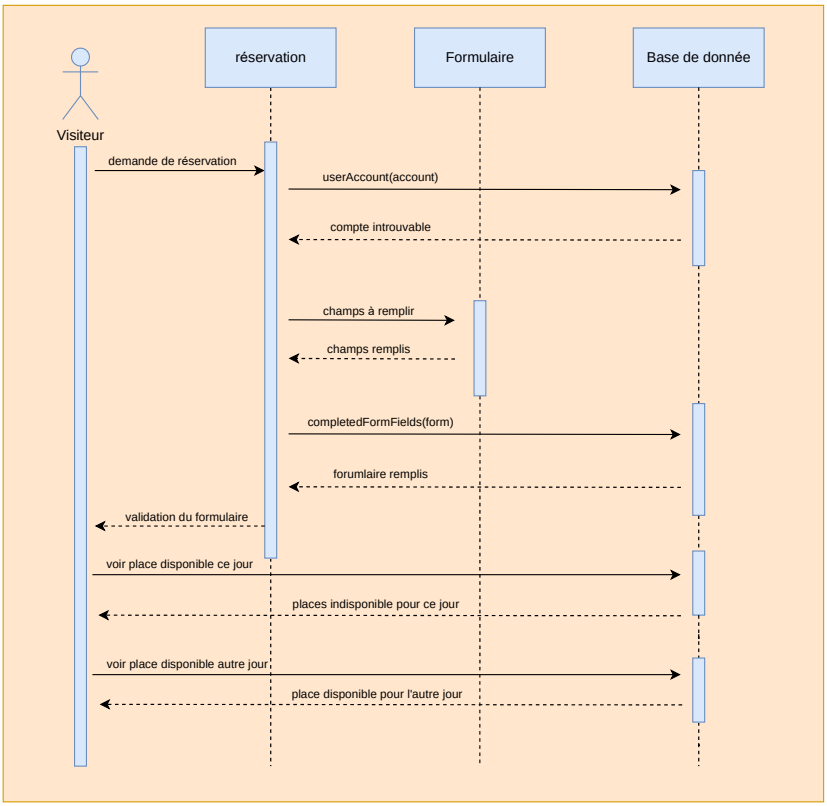
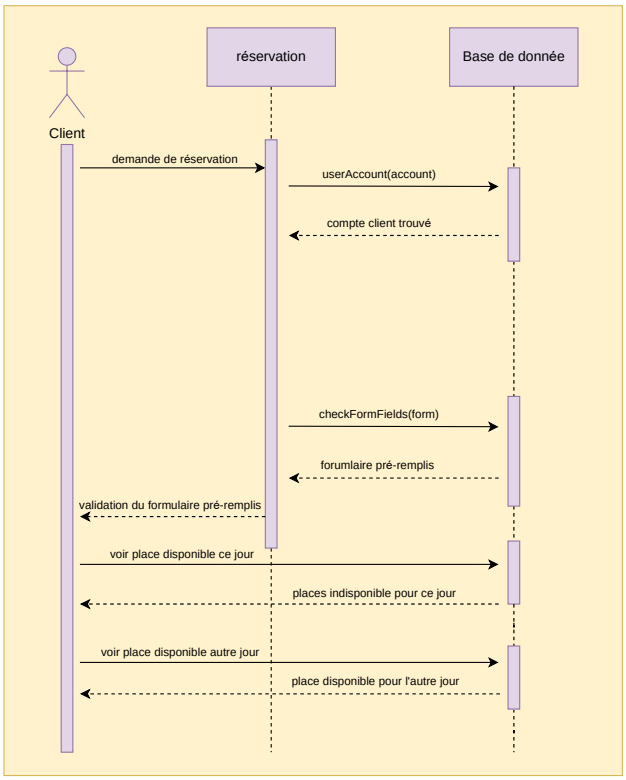


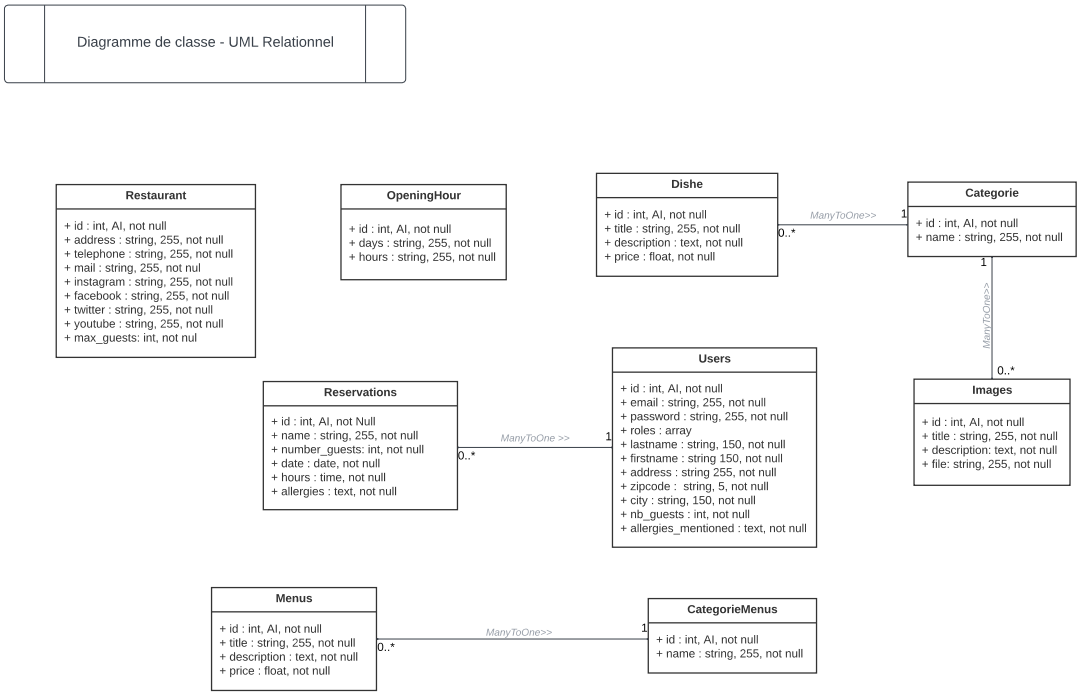
	Diagramme de séquence	



c) Diagramme de classe

Le diagramme de classe me permet de voir les relations entre les différentes entités, leurs attributs et leurs méthodes. Pour le cas du restaurant, Quai Antique, je commence par créer la classe Users qui va contenir tout ce dont aura besoin l'utilisateur lors de son inscription sur le site. Ces informations sont ensuite répertoriées dans la base de données. Dans le diagramme, chaque classe possèdes des propriétés avec différents types. Pour une visibilité publique, j'assigne le préfixe + devant chaque propriété. L'entité Users est en relation avec l'entité Reservations. Entre chaque relation je dispose des cardinalités (0..*, 1).

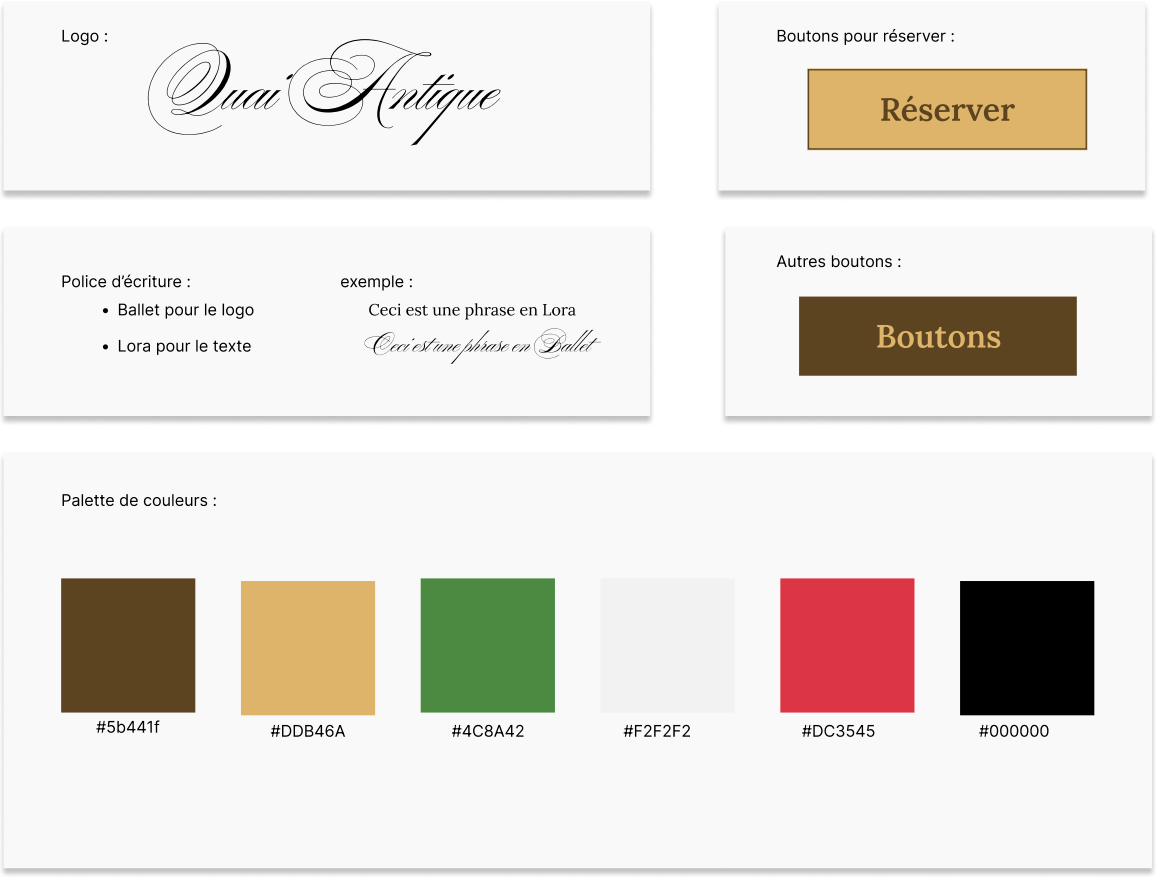
Par exemple pour le cas de la relation entre un utilisateur et une réservation, Une réservation peut être faite par un utilisateur et un utilisateur peut faire zéro ou infini de réservations. C'est ce qu'on appel une relation ManyToOne. Je fais en sorte de séparer chaque classe dont les relations ne sont pas forcément liées.



Ce diagramme de classe va être ma principale base lorsque je vais faire la création de mes entités dans le framework Symfony. Les relations vont être aussi affichées dans ma base de données avec les clés primaires pour les identifiants de chaque entité et les clés étrangères qui vont être associés pour chaque relation. L'application web open-source PHPMyAdmin me permet d'avoir une interface graphique clairement fournit pour examiner cela.

3. Charte graphique

À l'aide du site Figma, je réalise la charte graphique pour la police d'écriture du logo ainsi que pour l'ensemble de la police d'écriture du contenu. Je n'oublie pas le côté gastronomique que prône le restaurant. J' apporte une touche élégante et épuré tout en gardant un aspect convivial pour tout type de clientèle. Il est important de conserver l'esprit chaleureux qu'offre la Savoie et pour cela, je suis partie sur une palette de couleur représentant les spécialités savoyardes avec une nuance de jaune-marron, un peu couleur sable pour les parties clair du site et une nuance de brun foncé voir marron-foncé pour les parties plus sombres du site. Le mélange des deux arborent un côté savoyard et chaleureux. Pour les parties de validation, je respecte un code de couleur vert tirant sur un vert bouteille rappelant la couleur des bouteilles de vins blanc de la région. Pour tout ce qui est erreurs ou invalidités je suis partie sur un rouge cerise évoquant la touche dessert.

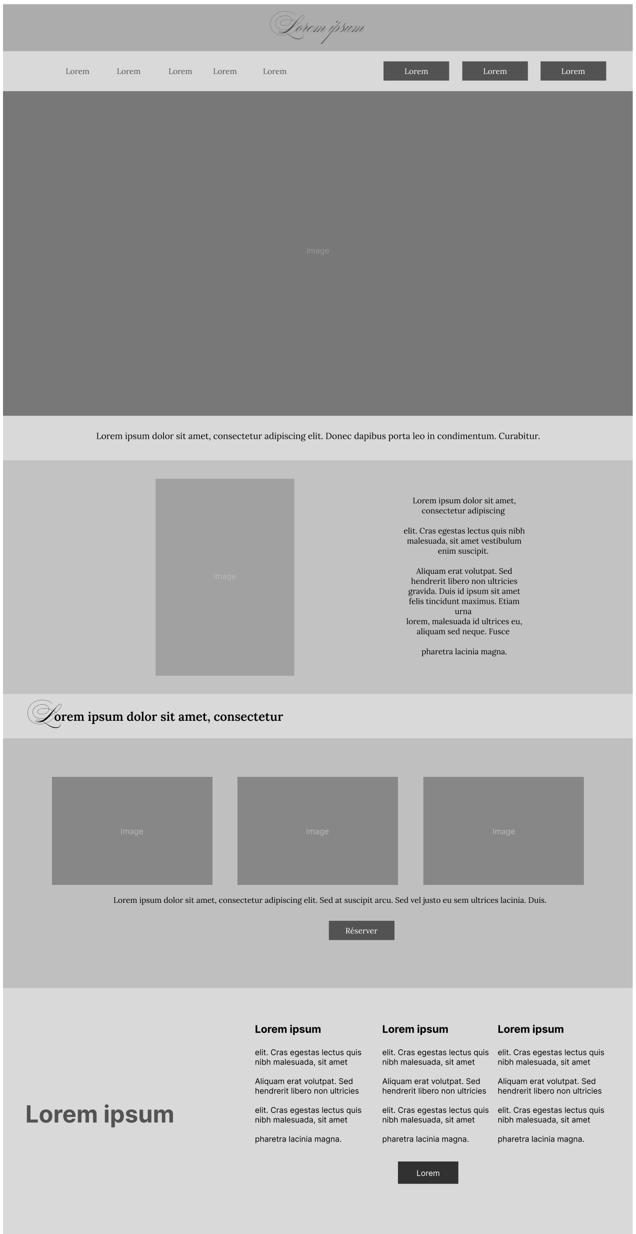


4. Références et inspirations

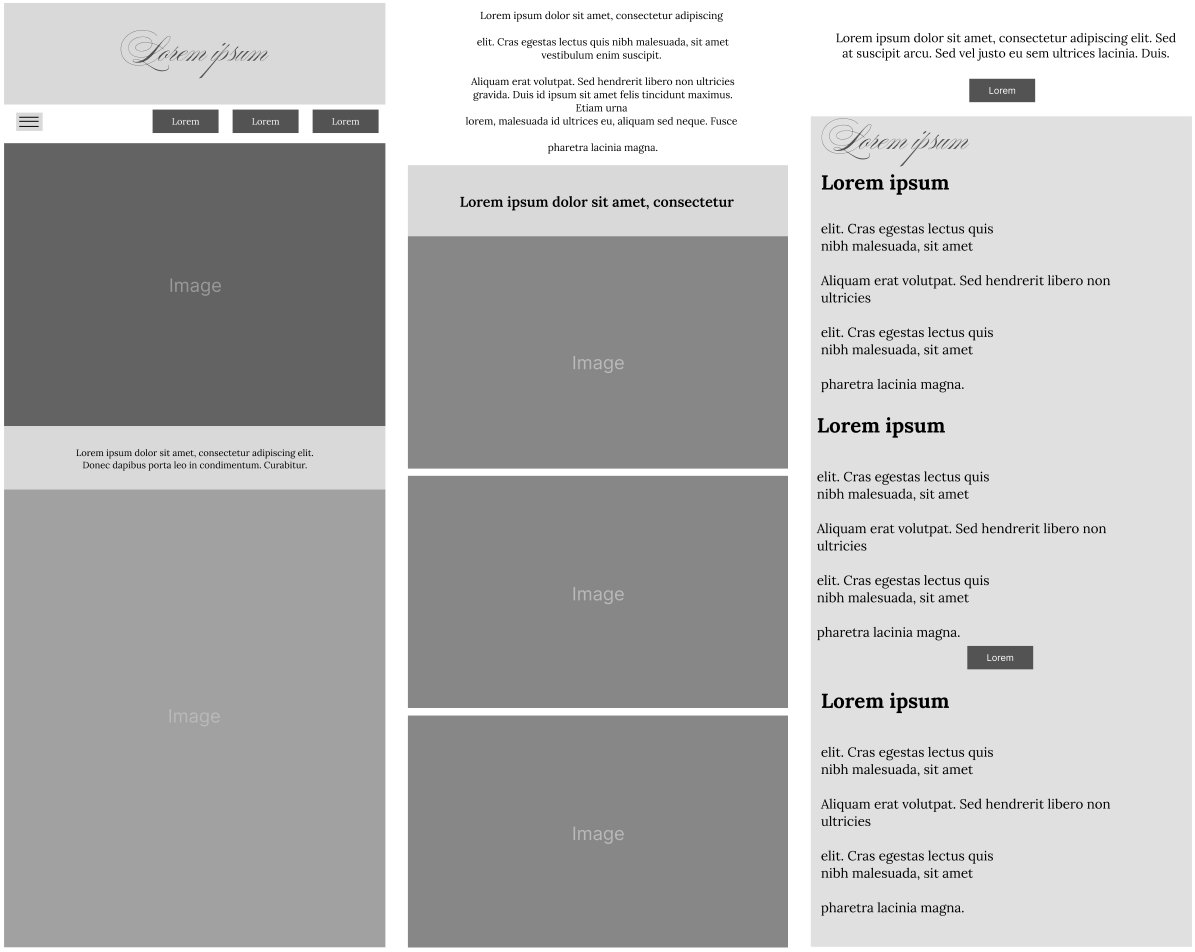
Je pioche le plus de références possible en visitant d'autres sites de restaurants déjà fonctionnels et complet.
Dès qu'un site web vitrine d'un restaurant m'inspire, je n'hésite pas non plus à aller voir le code dans la console du navigateur.
J'essaie de garder une cohérence avec des sites gastronomique pour voir leur style et pouvoir les utiliser dans ma maquette.

5. Les Wireframes

Pour le maquettage, je réalise tout le travail avec l'outil Figma. Le Wireframe me permet d'avoir une visualisation préliminaire sur les différents besoins du Chef Michant pour le restaurant Quai Antique.
Je laisse volontairement le schéma en noir et blanc car il peut y avoir des changements qui peuvent survenir au cours du développement pour la partie front-end, notamment en CSS. Je propose deux types de Wireframes, l'un pour le Desktop donc tout ce qui est visible depuis un écran d'ordinateur et une autre pour le Mobile lorsque l'application va être visualisée depuis un smartphone.
N'ayant pas encore défini chaque paragraphe ou titre pour le site, j'utilise du Lorem ipsum. C'est un excellent moyen de gagner du temps sur les blocs de paragraphe lors de la conception du site. Je peux également voir ce que cela rend avec la police d'écriture choisit dans la charte graphique.
Voici dans sa version Desktop le Wireframe pour la page d'accueil du site.



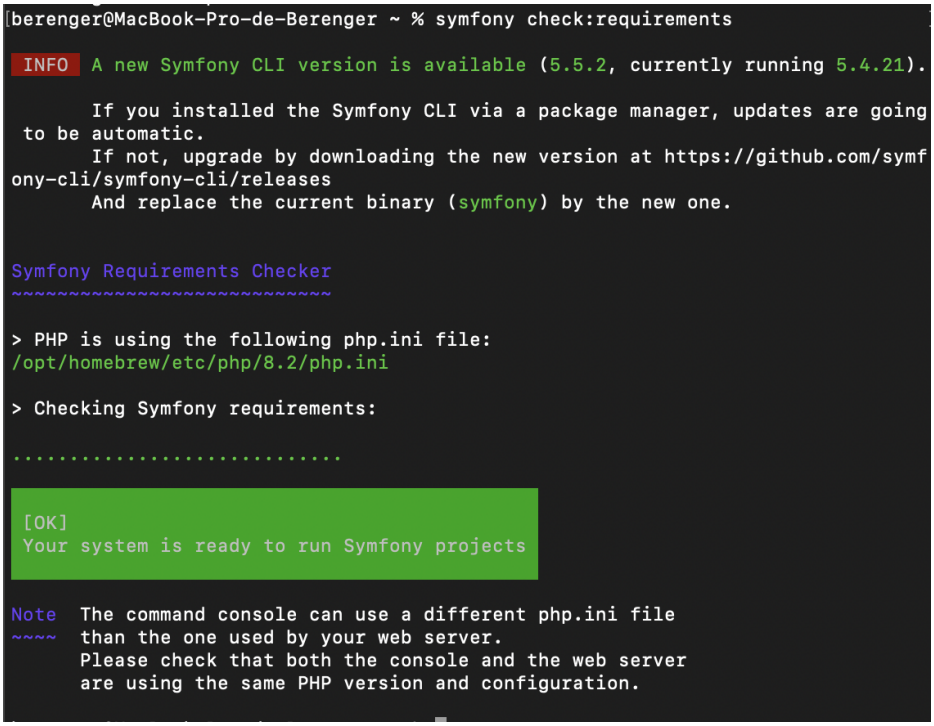
Et voici dans sa version Mobile divisé en trois du Wireframe pour la page d’accueil du site.



Initialisation du projet restaurant

1. Les prérequis

Afin de bien démarrer la partie technique du projet, je m'assure d'avoir tous les prérequis nécessaires avec le framework Symfony.
Avec la documentation Symfony, dans l'onglet d'installation, Symfony me montre les versions et outils à avoir.
Depuis le terminal, en ligne de commande, je peux voir quelle est ma version du langage PHP d'installé sur mon ordinateur. Je peux voir aussi si je dispose bien de Composer, l'outil de gestion de dépendances pour PHP. Composer me facilitera la mise en place du projet Symfony en automatisant l'installation des dépendances requises et en garantissant leur compatibilité avec le framework.
Je peux également observer si tous les prérequis nécessaires à l'exécution de l'application Symfony sont satisfaites sur le serveur où l'application est déployée avec la commande.



Cette commande effectue une série de tests pour s'assurer que la version de PHP installée sur le serveur répond aux exigences minimales de Symfony.

Avant même d'en avoir l'utilité, je vérifie aussi si node.js est installé. Node.js n'est pas une composante native de Symfony mais plutôt une technologie externe basée sur le moteur Javascript V8 de Google qui peut être utilisée avec Symfony pour améliorer les fonctionnalités de développement front-end de l'application en exécutant du code Javascript. Avec Node.js d'installé, je peux ainsi me servir du gestionnaire packages NPM (Node Package Manager) qui me permet d'automatiser le processus de compilation des fichiers CSS, Javascript et autres, d'installer des modules et dépendances tiers.

2. Création du nouveau projet Symfony

Depuis mon terminal, je m'assure d'être dans le bon dossier ou je veux créer le projet Symfony. Je choisi la version LTS(Long-term support) de Symfony car elle me garantit jusqu'à fin 2024 un support de sécurité à long terme en réduisant le risque de failles de sécurité et assurant la compatibilité avec les nouvelles versions de PHP.
Je me renseigne auprès du services releases de Symfony pour un détails des versions de Symfony.
Voulant disposer d'un squelette d'application Symfony prêt à l'emploi déjà préconfiguré avec les packages couramment utilisés pour le développement web, tels que Twig pour le rendu des templates, Doctrine ORM pour la gestion de la base de données, je décide d'introduire l'option « --webapp ».

```
[berenger@MacBook-Pro-de-Berenger htdocs % symfony new ecf_quai_antique --version="5.4.*"--webapp ]
```

Symfony me créer automatiquement plusieurs dossiers dont un dossier caché .git à la racine du projet pour initialiser un dépôt Git afin de suivre les modifications apportées et gérer la synchronisation du code. Il créé une branche master qui va être ma base sur le lequel chaque nouvelle branche apportent des modifications au projet vont être fusionnées. La branche master me sert également pour le déploiement de l'application.

Dans la foulé, je crée un dépôt distant à l'aide de la plateforme web GitHub. Le dépôt distant dit Repository m'assure un stockage, partage et une collaboration au projet restaurant, Quai Antique. Une fois qu'une tâche est accomplie, que les branches ont fusionné côté local, il me suffit de pousser « push » les modifications apportées sur Github côté distant. Ainsi le code du projet va être vue par d'autre développeurs voulant reprendre mon code à condition que le Repository soit public.

3. L'environnement de développement et les extensions

Pour l'IDE (Integrated Development Environment), j'utilise Visual Studio Code (VSCode). Cet outil va être mon principal logiciel pour les phases d'écriture du code, de test, débogue et de développement du projet restaurant.
Je n'oublie pas que VSCode me fournit une multitude d'extensions pour me simplifier certaines tâches.
J'utilise principalement le plugin PHP Intellisense pour améliorer la fonctionnalité d'autocomplétions du code PHP en fournissant des suggestions de code plus précises et plus rapides. Pour une assistance à la programmation du moteur de template Twig, j'utilise le plugin Twig et Twig Language 2. La colorisation syntaxique et l'autocomplétions aident énormément pour l'écriture du code.

4. Création du fichier readme.md

À la racine de mon projet dans l'IDE, je crée un fichier readme.md. Ce fichier est essentiellement prévu sur les démarches à suivre pour l'exécution en local du projet restaurant, Quai Antique. Plusieurs informations vont être listées dans ce fichier. Ainsi, d'autres développeurs peuvent suivre les étapes depuis le readme qui est en visible depuis le dépôt distant de la plateforme web GitHub.
Généralement je liste chaque étape au fur et mesure que le projet avance. Une fois qu'une étape est terminée, j'écris la démarche à suivre dans le readme.

Organisation par branche avec Git

Pour conserver un projet cadré et structuré, je crée des nouvelles branches pour chaque partie différente lors du développement. Par exemple pour l'installation de nouveaux outils ou bibliothèque dans le projet Symfony, je créé une nouvelle branche qui commence toujours par feat, pour features.

```
feat/template
```

Sur chaque changement apporté à la partie template par exemple, je suis positionné sur cette branche, ainsi je peux modifier le contenu que je souhaite sans casser ce qui va être sur la branche master ou d'autres branches. Il me suffit de fusionner la branche sur laquelle je viens d'effectuer des modifications avec la branche master.

Mise en place d'autres outils

1. Webpack Encore

Afin d'apporter de l'efficacité et des performances au projet, j'installe l'outil de build Webpack Encore. Symfony met à ma disposition une formidable documentation. En utilisant Webpack Encore, je peux simplifier le processus de création de l'application web, améliorer les performances et optimiser la taille du fichier final, ce qui améliore l'expérience utilisateur en réduisant les temps de chargement.
Cela créer un fichier webpack.config.js à la racine du projet afin d'injecter les différents outils que me permet de fournir Webpack Encore.

2. Préprocesseur SASS, SCSS

Comme Webpack Encore me fournit une facilité à la compilation, j'installe le préprocesseur SASS qui me permet d'écrire des feuilles de style CSS plus facilement et efficacement en ajoutant des fonctionnalités telles que les variables, les fonctions, les mixins et bien d'autres. J'opte pour une syntaxe alternative, le SCSS.
Le SCSS se rapproche plus du langage CSS. C'est surtout un choix purement personnel et une question d'habitude.
Pour chaque modification apporté au projet dans les fichiers SCSS, il me suffit depuis le terminal d'exécuter la commande suivante :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % npm run build
```

Ou bien

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % npm run watch
```

Cette dernière commande m'évite de réécrire la ligne de commande pour la compilation à chaque modification SCSS ou Javascript enregistré, mais doit être arrêter avant de refermer l'IDE.

3. Postprocesseur PostCSS et autoprefixer

Une fois de plus la documentation Symfony m'offre la possibilité d'obtenir dans le projet Symfony un outil me permettant d'appliquer des transformations automatisées sur du code CSS. Dans le projet il est surtout utile à la gestion des préfixes de navigateurs avec le plugin Autoprefixer pour un rendu visuel identique sur les tous les navigateurs.
Le fichier postcss.config.js est créer à la racine du projet. J'insère le code suivant à l'intérieur du fichier :

postcss.config.js

```
postcss.config.js > ...
1   module.exports = {
2       plugins: {
3           // include whatever plugins you want
4           // but make sure you install these via yarn or npm!
5           // add browserslist config to package.json (see below)
6           autoprefixer: {}
7       }
8   }
```

4. Référencer des images

Dans la continuité de l'amélioration pour l'expérience utilisateur, référencer des images à partir d'un modèle fournit des images pertinentes et de haute qualité pour l'illustration des pages du site web. Je peux également rendre un processus de gestion des images plus efficace en permettant la réutilisation des images et en évitant la duplication des images pour chaque page.
Dans le fichier webpack.config.js je mets le code pour le package du file-loader :

```
// enables Sass/SCSS support
.enableSassLoader()
.enablePostCssLoader()
.copyFiles({
    from: './assets/images',
    to: 'images/[path][name].[hash:8].[ext]'
})
```

Ce code a pour objectif de copier le nom des fichiers généré dans le code HTML depuis la balise img assets/images vers le répertoire public/build/images. J'y inclus un hachage basé sur le contenu. Si un fichier d'image est modifié à l'avenir, le hachage sera également modifié, ce qui signifie que le nom de fichier de l'image dans le code HTML changera également.
Voici un exemple dans le code HTML :

```
<img class="w-100 reveal" src="{{ asset('build/images/restaurant-salle-01.png') }}" alt="salle restaurant">
```

5. Bootstrap 5

Le framework open-source front-end Bootstrap m'offre la possibilité d'obtenir un côté responsives et adaptatifs avec une bibliothèque complète de styles CSS préconçus. Son système de class efficace, intuitif et son style m'assure un meilleur rendu au point vu de l'expérience utilisateur.
La documentation sur le site de Bootstrap est très bien conçue et rapide à prendre en main. En suivant les étapes sur la documentation de Symfony, il me suffit de copier les codes suivants dans les dossiers d'assets pour le Javascript et le SCSS :

```
// any CSS you import will output into a single css file (app.css in this case)
import './styles/app.scss';

// or, specify which plugins you need:
import { Tooltip, Toast, Popover } from 'bootstrap'

// start the Stimulus application
import './bootstrap';
```

Pour compiler le tout je fais la commande : npm run build

Création du premier Controller

1. Première route pour la page d'accueil

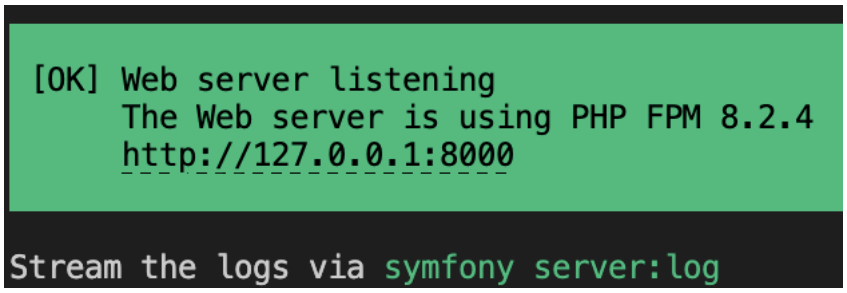
Les éléments clé de l'architecture de Symfony sont les Controller. À l'aide de Symfony CLI (Symfony Command Line Interface) qui me permet de faciliter le développement en ligne de commande comme:

symfony console make :controller

Je crée directement le fichier HomeController pour ma page d'accueil du site. Pour vérifier son bon fonctionnement, je redirige la route vers un chemin dans l'url qui finit par « / »
Avec le lancement du serveur Symfony :

symfony serve -d

J'ai un premier visuel en local sur le port suivant :



2. Méthode render()

Cette méthode permet d'avoir le rendu du fichier html.twig :

```
return $this->render('home/index.html.twig',
```

Dans le dossier templates figure le fichier base qui sert généralement de modèle de base pour toutes les autres pages du site web. Ce fichier comporte les éléments communs à toutes les pages du site, tels que la structure de base, les en-têtes, les pieds de page, les liens vers les fichiers Javascript et CSS.
Le chemin dans la méthode **render()** renvoi à un dossier nommé « home » qui est composé d'un fichier intitulé « index.html.twig ». Dans ce template figure le code en HTML 5 et les class Bootstrap ou les class créer avec la méthode de structuration HTML appelé BEM (Block Element Modifier).

Mise en place de la base de données

1. Création du compte alwaysdata

La plateforme d'hébergement web alwaysdata m'offre des services de serveurs virtuels privés(VPS), d'hébergement mutualisé et une gestion de base de données. Je crée un compte sur ce service afin d'avoir une interface utilisateur conviviale avec PHPMyAdmin, une sécurité et sa compatibilité avec PHP et d'autres technologies.

2. Création de la base de données

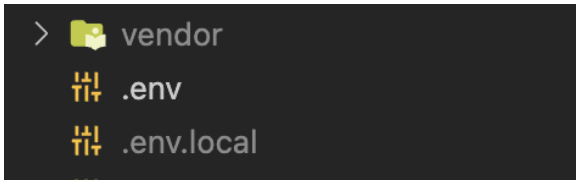
J'utilise le Symfony CLI pour la création de base données avec doctrine depuis le terminal de VSCode. Doctrine me permet de travailler avec la base de données de manière orienté objet. Doctrine m'est utile pour le mapping objet-relationnel(ORM) qui me permet de faire le lien entre mon code dans l'application et ma base de données.

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % symfony console d:d:c
```

le D pour Doctrine, D pour Database et C pour Create.
Lorsque la commande est lancée, j'observe la présence d'une base de données sur la plateforme web alwaysdata, comprenant une table vide portant le nom du projet Symfony.

3. Changer les variables d'environnement

À la racine de mon projet il y a un fichier intitulé .env qui stocke les variables d'environnement de l'application. Ici vont être entreposé le nom, mot de passe et serveur de la base de données. Cependant comme le fichier .env est versionné, c'est-à-dire que git enregistre les modifications apportées à ce fichier dans le dépôt local et distant. Pour contourner cela je créé un second fichier toujours à la racine du projet depuis VSCode que je nomme .env.local qui celui-ci ne va pas versionné.



Le fichier .env.local est grisé donc pas pris en compte par git.
Comme j'utilise la plateforme web alwaysdata avec l'interface PHPMyAdmin, dans la page d'accueil du serveur dans l'onglet Database server, ce trouve toutes les informations essentielles pour changer le **DATABASE_URL** dans le fichier .env.local.
Il faut modifier le type de serveur à savoir MariaDB qui est le même système de gestion de base de données que MySQL excepté que MariaDB offre une sécurité et des performances supplémentaire.
Il faut modifier également le nom de l'utilisateur de la base de données, le mot de passe, le host qui est l'adresse du serveur de la base de données.
Ce sont des données confidentielles pour le projet donc c'est pour cela que je créé un second fichier pour les variables d'environnement.

Création des entités

Cette étape est très importante car elle détermine précisément les tables et relations dans la base de données utilisée par l'application Symfony. J'utilise également le système de Symfony CLI en ligne de commande.

1. Création de l'entité Users

Je commence par créer l'entité « Users » avec la commande suivante :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % symfony console make:user
```

Je me base sur le diagramme de classe - UML Relationnel que j'avais créé précédemment. Il me suffit de suivre les étapes proposées. Le **make :user** génère automatiquement une nouvelle classe utilisateur avec des propriétés de base tel que l'identifiant, l'adresse email, le mot de passe et le rôle. Il me suffit par la suite de taper la commande en suivant les étapes de :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % symfony console make:entity
```

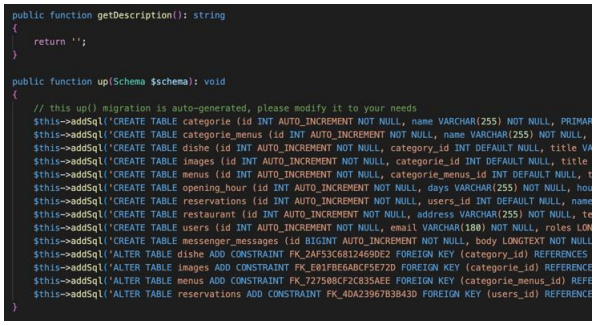
pour générer d'autres propriétés tel que le nom, le prénom, l'adresse, le code postal, la ville, le nombre d'invités et les allergies mentionnées comme l'indique le diagramme de classe - UML Relationnel. Lors de l'inscription d'un utilisateur, il est impératif de suivre la consigne du Chef Michant, qui stipule que l'utilisateur doit pouvoir bénéficier d'une fonctionnalité d'autocomplétion pour le nombre d'invités et les allergies mentionnées, à condition qu' il ait rempli ces informations dans le formulaire.

2. Créations des autres entités

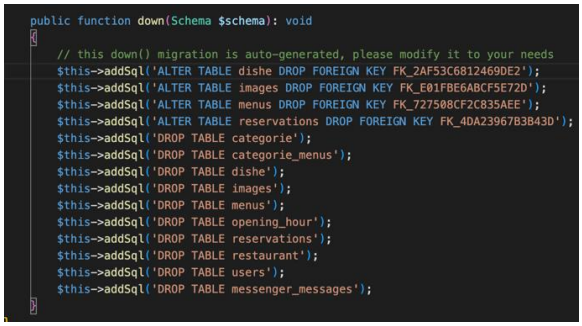
Je continu le même procéder pour chaque table de mon digramme de classe.
Je n'oublie pas la relation ManyToOne entre l'entité Reservations et l'entité Users, ainsi qu'entre Dishe et Categorie, Images et Categorie puis Menus et CategorieMenus qui sont aussi des relations ManyToOne.

3. Les migrations

Afin de refléter l'état actuel de la base de données, la commande utilisant **DoctrineMigrationsBundle** génère une classe de migration. Cette migration est stockée dans le dossier migrations de l'application Symfony.



```
public function getDescription(): string
{
    return '';
}

public function up(Schema $schema): void
{
    // this up() migration is auto-generated, please modify it to your needs
    $this->addSql('CREATE TABLE categorie (id INT AUTO_INCREMENT NOT NULL, name VARCHAR(255) NOT NULL, PRIMAR
    $this->addSql('CREATE TABLE categorie_menus (id INT AUTO_INCREMENT NOT NULL, name VARCHAR(255) NOT NULL,
    $this->addSql('CREATE TABLE dishe (id INT AUTO_INCREMENT NOT NULL, category_id INT DEFAULT NULL, title VA
    $this->addSql('CREATE TABLE images (id INT AUTO_INCREMENT NOT NULL, categorie_id INT DEFAULT NULL, title
    $this->addSql('CREATE TABLE menus (id INT AUTO_INCREMENT NOT NULL, categorie_menus_id INT DEFAULT NULL, t
    $this->addSql('CREATE TABLE opening_hour (id INT AUTO_INCREMENT NOT NULL, days VARCHAR(255) NOT NULL, hou
    $this->addSql('CREATE TABLE reservations (id INT AUTO_INCREMENT NOT NULL, users_id INT DEFAULT NULL, name
    $this->addSql('CREATE TABLE restaurant (id INT AUTO_INCREMENT NOT NULL, address VARCHAR(255) NOT NULL, te
    $this->addSql('CREATE TABLE users (id INT AUTO_INCREMENT NOT NULL, email VARCHAR(180) NOT NULL, roles LON
    $this->addSql('CREATE TABLE messenger_messages (id BIGINT AUTO_INCREMENT NOT NULL, body LONGTEXT NOT NULL
    $this->addSql('ALTER TABLE dishe ADD CONSTRAINT FK_2AF53C6812469DE2 FOREIGN KEY (category_id) REFERENCES
    $this->addSql('ALTER TABLE images ADD CONSTRAINT FK_E01FBE6ABCF5E72D FOREIGN KEY (categorie_id) REFERENCE
    $this->addSql('ALTER TABLE menus ADD CONSTRAINT FK_727508CF2C835AEE FOREIGN KEY (categorie_menus_id) REFE
    $this->addSql('ALTER TABLE reservations ADD CONSTRAINT FK_4DA23967B3B43D FOREIGN KEY (users_id) REFERENCE
}
```



```
public function down(Schema $schema): void
{
    // this down() migration is auto-generated, please modify it to your needs
    $this->addSql('ALTER TABLE dishe DROP FOREIGN KEY FK_2AF53C6812469DE2');
    $this->addSql('ALTER TABLE images DROP FOREIGN KEY FK_E01FBE6ABCF5E72D');
    $this->addSql('ALTER TABLE menus DROP FOREIGN KEY FK_727508CF2C835AEE');
    $this->addSql('ALTER TABLE reservations DROP FOREIGN KEY FK_4DA23967B3B43D');
    $this->addSql('DROP TABLE categorie');
    $this->addSql('DROP TABLE categorie_menus');
    $this->addSql('DROP TABLE dishe');
    $this->addSql('DROP TABLE images');
    $this->addSql('DROP TABLE menus');
    $this->addSql('DROP TABLE opening_hour');
    $this->addSql('DROP TABLE reservations');
    $this->addSql('DROP TABLE restaurant');
    $this->addSql('DROP TABLE users');
    $this->addSql('DROP TABLE messenger_messages');
}
```

À l'aide des commandes SQL générées par l'API de création de schéma de Doctrine, deux méthodes sont contenues par la classe « AbstractMigration ». Dans la méthode **up()**, chaque commande « CREATE TABLE » définit les colonnes de la nouvelle table ainsi que le type de données et contraintes pour chaque colonne. Pour le méthode **down()**, les opérations inverses sont effectuées, ce qui signifie que les tables sont supprimées. Les contraintes de clé étrangère sont supprimées en premier pour éviter toute erreur.

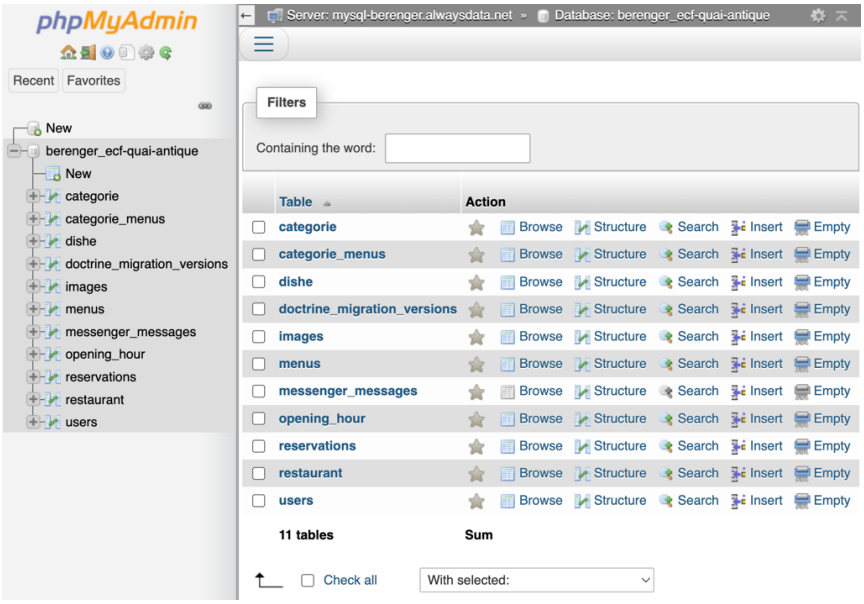
4. Appliquer les migrations

Pour pouvoir exécuter les migrations de doctrine vers la base de données alwaysdata, je procède toujours avec le Symfony CLI en ligne de commande :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % symfony console d:m:m
```

D pour Doctrine, M pour Migrations et le dernier M pour Migrate.
Cette commande parcoure toutes les migrations en attente et exécute dans l'ordre en mettant à jour la structure de la base de données alwaysdata.

Voici une vue des tables depuis PHPMyAdmin de la base données :



Voici une vue des tables de la base de données depuis la ligne de commande MySQL :

```
Connection id:    26843629
Current database: berenger_ecf-quai-antique

mysql> show tables;
+-----------------------------------------+
| Tables_in_berenger_ecf-quai-antique |
+-----------------------------------------+
| categorie                               |
| categorie_menus                         |
| dishe                                   |
| doctrine_migration_versions             |
| images                                  |
| menus                                   |
| messenger_messages                      |
| opening_hour                            |
| reservations                            |
| restaurant                              |
| users                                   |
+-----------------------------------------+
11 rows in set (0.22 sec)

mysql>
```

Réalisation des tests unitaires

En vue de m'assurer que chaque classe fonctionne correctement, je réalise des tests unitaires depuis l'outils PHPUnit. Je crée des cas de test qui vérifient que les méthodes, les fonctions ou les classes fonctionnent comme prévu.
Je vérifie en tout premier si PHPUnit est bien installé avec la commande :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % phpunit --version

PHPUnit 10.0.12 by Sebastian Bergmann and contributors.
```

Je vérifie également si figure le fichier phpunit.xml.dist depuis la racine du projet.
Je continu avec le système que me propose Symfony CLI en ligne de commande pour créer les tests unitaires avec la commande :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % symfony console make:unit-test
```

Je renomme chaque fichier de test que je crée avec le nom de la table en premier suivit par UnitTest.
Exemple : UsersUnitTest

Pour chaque test je crée toujours trois public function :
. un pour vérifier si le test est vrai = **testIsTrue()**
. un pour vérifier si le test est faux = **testIsFalse()**
. un pour vérifier si le test est vide = **testIsEmpty()**

Pour chaque public function je reprends les fonctions « set » qui sont répertorié dans le fichier des entity.
Exemple pour le « lastname » dans le fichier entity « Users » :

```php
public function setLastname(string $lastname): self
{
    $this->lastname = $lastname;

    return $this;
}
```

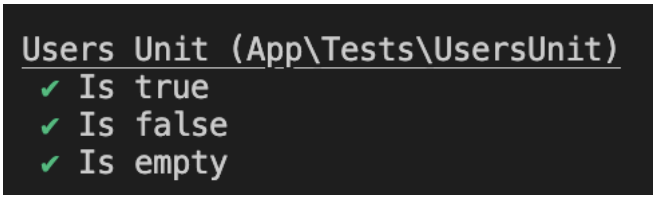
Repris dans la fonction **testIsTrue()** du fichier de test unitaire de « Users » (UsersUnitTest) :

```php
public function testIsTrue()
{
    $users = new Users();

    $users->setEmail('true@mail.fr')
          ->setPassword('password')
          ->setLastname('nom')
          ->setFirstname('prénom')
          ->setAddress('adresse')
          ->setZipcode('73000')
          ->setCity('chambéry')
          ->setNbGuests(8)
          ->setAllergiesMentioned('aucune');
```

Une fois les codes écrit, je vérifie si les tests fonctionnent correctement. La commande :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % php vendor/bin/phpunit --testdox
```

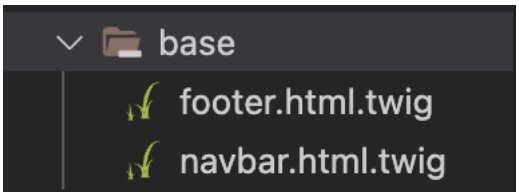
me permet d'avoir un format de documentation lisible et facile à comprendre grâce au testdox.
Voici un exemple de test unitaire réussit au format testdox :



PARTIE FRONT - La page d'accueil

1. Les Pages HTML

Depuis le dossier templates, je crée un sous-dossier intitulé « base ». À l'intérieur de ce dossier je crée deux fichiers, un fichier « navbar.html.twig » pour la barre de navigation et un fichier « footer.html.twig » pour le pied de page. Il est stipulé par le Chef Michant que le pied de page doit être sur chaque page du site. Je décide de faire de même pour la barre de navigation.



Dans le fichier « base.html.twig » j'inclue les deux fichiers du dossier « base » dans la balise <body>.

```
<body>
    {% include "base/navbar.html.twig" %}
    {% block body %}{% endblock %}
    {% include "base/footer.html.twig" %}
    {% block javascripts %}
        {{ encore_entry_script_tags('app') }}
    {% endblock %}
</body>
```

Désormais, pour chaque fichier twig créé dans le dossier templates, va être incrusté la barre de navigation en haut de la page et le pied de page.
Entre les deux il va toujours y avoir le block body.

Ensuite je crée un sous-dossier intitulé « home » dans le dossier « templates ». Ce dossier contient trois fichiers twig. Un des trois fichiers a pour nom « index.html.twig ». Dans ce fichier figure le modèle de base qui est inclue. Le fichier « index.html.twig » hérite du modèle de « base.html.twig ». Ensuite entre le block body est inclue un fichier pour une partie de la page d'accueil et l'autre pour une deuxième partie. J'ai volontairement séparé ces fichiers pour m'éviter d'avoir un seul fichier trop lourd.

templates > home > index.html.twig

```
1   {% extends "base.html.twig" %}
2
3   {% block body %}
4   {% include "home/cover.html.twig" %}
5   {% include "home/preview-galleries.html.twig" %}
6   {% endblock %}
7
```

J’inclus dans le Controller « HomeController » à la méthode **render()** le chemin ou va être redirigé la route que je nomme « app_home » avec comme url « 127.0.0.1:8000/. »

```
class HomeController extends AbstractController
{
    #[Route('/', name: 'app_home')]
return $this->render('home/index.html.twig',
```

Le fichier « cover » et « preview-galleries.html.twig » ont un schéma classique html. On retrouve les class avec la méthode de structuration HTML appelé BEM (Block Element Modifier) ainsi que les class prédéfinies du framework Bootstrap. La particularité que m’offre Bootstrap est son côté responsive grâce à ses class :

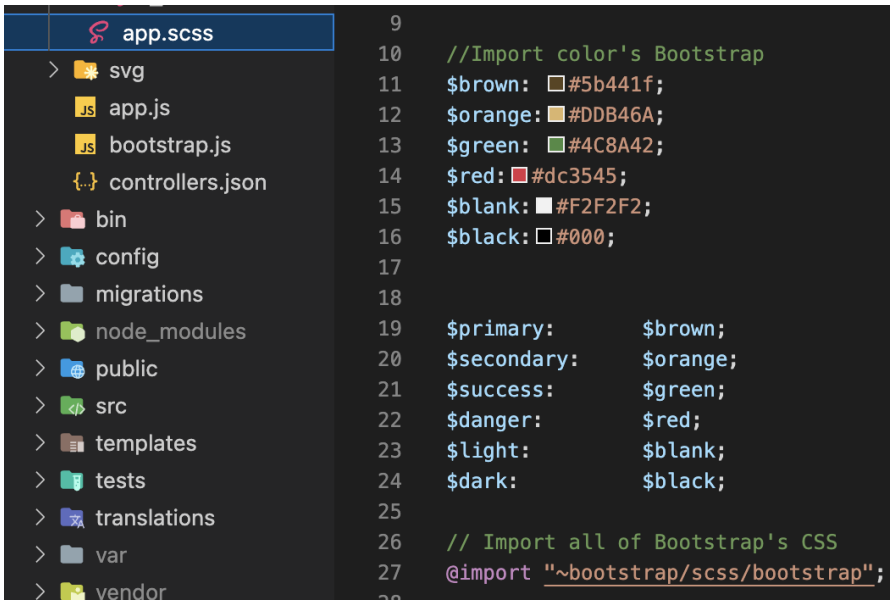
```
<div class="card col-md-12 col-sm-12 col-lg-4 col-xl-4 rounded-0 border-0">
```

Ce système extrêmement simplifié m’évite de surcharger mes feuilles de style CSS ou plus particulièrement SCSS avec le préprocesseur.

2. Les fichiers SASS (SCSS)

a) L’ architecture SASS

Pour suivre les bonnes pratiques d'organisation des fichiers Sass, j'importe en premier lieu les variables de couleurs proposées par Bootstrap et les adapte aux couleurs de la charte graphique. Je modifie également le nom de certaines variables pour pouvoir utiliser des class Bootstrap avec des couleurs prédéfinies. Ensuite, j'importe les fichiers CSS nécessaires en spécifiant le chemin approprié.



Pour suivre les conventions de Sass, je crée un dossier « scss » dans le dossier « styles », qui contient des sous-dossiers pour chaque type de code. J'applique les bonnes pratiques en créant un dossier « atoms » pour les styles de boutons de l'application, un dossier « base »

pour le reset CSS, un dossier « components » pour chaque partie du site (comme la page de connexion, la page de la carte, etc.), et enfin un dossier « utils » pour les variables et mixins.

b) Mise en œuvre fidèle de la maquette
Je m'efforce de suivre fidèlement la mise en page des éléments telle qu'elle est présentée sur la maquette, afin de garantir une cohérence stylistique sur l'ensemble de la réalisation. Pour un visuel sur mon navigateur web en local, j'exécute la commande suivante :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % symfony serve -d
```

Cette commande me permet de lancer le serveur web intégré de Symfony en arrière-plan. L'utilité de « -d », qui signifie « détaché », donne la possibilité de laisser le serveur en marche pendant que je continue à travailler sur le code de l'application. Je peux aussi continuer à utiliser le terminal sans interrompre le serveur.

```
[OK] Web server listening
     The Web server is using PHP FPM 8.2.4
     http://127.0.0.1:8000

Stream the logs via symfony server:log
```

c) Compilation des fichiers CSS
Lorsque je sauvegarde mon travail avec « command + s », les fichiers SCSS doivent être compilés en fichiers CSS lisibles par le navigateur.

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % npm run build
```

Pour m'éviter d'exécuter cette commande à chaque fois, j'utilise la commande suivante :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % npm run watch
```

PARTIE FRONT - Les autres pages

1. Création d'autres Controllers

Pour associer chaque élément du menu de la barre de navigation à une URL spécifique dans Symfony, je crée un Controller dédié pour chaque page. En définissant ainsi les routes associées à chaque Controller, Symfony est capable de diriger les requêtes HTTP vers le Controller approprié et donc d'afficher la page correspondante.

```
#[Route('/la-carte', name: 'app_dishes')]
```

Exactement, tout comme le premier Controller « HomeController », les autres Controllers dans Symfony génèrent également une réponse HTTP en utilisant la méthode **render()** qui renvoie un template Twig pour produire la page HTML à afficher. Cette méthode permet d'injecter des données dynamiques dans la page à partir du Controller, en utilisant le système de variables de Twig.

```
return $this->render('dishes/index.html.twig',
```

J'utilise la directive {% extends %} en haut de chaque fichier Twig pour étendre un template de base et y inclure des blocs spécifiques. Je prends une habitude d'avoir recours aux méthodes de structuration HTML BEM, ainsi qu'aux class prédéfinies du framework Bootstrap dans les fichiers du dossier templates de l'application.

2. Utilisation des variables Twig

Dans le but d'éviter une surcharge et une répétition du code dans les fichiers Twig, j'utilise les variables fournies par Twig avec une syntaxe pour les instances de classe des entités associées qui contiennent les propriétés. Par exemple, si j'affiche le nom de chaque catégorie dans un fichier Twig, j'utiliser la syntaxe Twig `{{ categorie.name }}`. Ainsi, j'ai accès à la propriété **«** name » de chaque objet « Category » pour afficher le nom de chaque catégorie. Cela permet de générer dynamiquement la page HTML en itérant sur la collection des catégories dans le contrôleur et en passant les informations nécessaires à la vue.

J'utilise des boucles pour parcourir des variables et itérer sur chaque objet dans une collection, ce qui me permet d'afficher leur contenu de manière dynamique dans mon code.

```
{% for categorie in categories %}
    <div class="row mb-2 mt-5 text-primary">
        <div class="col"></div>
        <div class="col-xl-4 col-md-4 ms-sm-5 ms-5"><h3 class ="container--category-fonts container--categorie-firstletter">{{ categorie.name }}</h3></div>
        <div class="col-xl-6 col-md-6 "></div>
    </div>
    {% for dishe in dishes %}
    {% if dishe.category is same as(categorie) %}
    <div class="row mb-1 reveal">
        <div class="col"></div>
        <div class="col-xl-5 col-md-5 col-sm-5 col-5 text-primary container--dishes-title">{{ dishe.title }}</div>
        <div class="col"></div>
        <div class="col"></div>
    </div>
    <div class="row dishes mb-3 reveal">
        <div class="col"></div>
        <div class="col-xl-5 col-md-5 col-sm-5 col-5 container--dishes-description text-primary">{{ dishe.description }}</div>
        <div class="col text-secondary header--menu-font">{{ dishe.price }}€</div>
        <div class="col"></div>
    </div>
    {% endif %}
    {% endfor %}
{% endfor %}
```

3. La police d’écriture

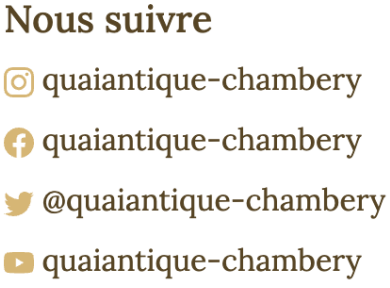
J’importe depuis le site web de Google fonts, la même police d’écriture que celle choisit depuis la charte graphique et le Wireframe dans la balise <head> de mon fichier « base.html.twig ».

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Quai Antique{% endblock %}</title>
        <link rel="icon" href="data:image/svg+xml,<svg xmlns=%22http://www.w3.org/2000/svg%22 viewBox=%220 0 128
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
        <link href="https://fonts.googleapis.com/css2?family=Lora:ital,wght@0,400;0,500;0,600;0,700;1,400;1,500;1
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
        <link href="https://fonts.googleapis.com/css2?family=Ballet:opsz@16..72&display=swap" rel="stylesheet">
        {# Run `composer require symfony/webpack-encore-bundle` to start using Symfony UX #}
        {% block stylesheets %}
            {{ encore_entry_link_tags('app') }}
        {% endblock %}
    </head>
    <body>
        {% include "base/navbar.html.twig" %}
        {% block body %}{% endblock %}
        {% include "base/footer.html.twig" %}
        {% block javascripts %}
            {{ encore_entry_script_tags('app') }}
        {% endblock %}
    </body>
</html>
```

4. Intégration d’images SVG

a) <u>Les symboles</u>
Depuis le site web de Bootstrap, je télécharge les symboles directionnels que je vais avoir besoin sur les pages du site afin de guider l’utilisateur et bénéficier d’une meilleure expérience utilisateur. C’est un fichier SVG (Scalable Vector Graphics), que je redimensionne sans perdre en qualité lors du responsive.

b) <u>Les icônes</u>
Dans le cadre de la conception du pied de page, des icônes pour les réseaux sociaux sont nécessaires. Je les télécharge également depuis le site Bootstrap. Ces icônes permettent aux utilisateurs d'identifier et de cliquer sur les réseaux sociaux correspondants s'ils souhaitent les suivre ou les consulter.



5. Les chemins d'accès

Dans la liste des différents menus qui compose la barre de navigation, figure les fonctions **path()** Twig remplaçant l'attribut « href », en vue de générer une URL correspondant à une route.

```
<div class="header--menu-font mx-auto">
    <ul class="navbar-nav">
        <li class="nav-item ms-5">
            <a class="header--menu-hover text-primary nav-link" aria-current="page" href="{{ path('app_dishes')}}">La carte</a>
        </li>
```

J'utilise cette même fonction **path()** sur les symboles SVG lorsqu'un utilisateur clique sur une flèche directionnelle pour se rendre à une autre page du site web.

Gestion de l'authentification et autorisation

Pour permettre à un utilisateur de se connecter à un compte afin d'obtenir un formulaire prérempli pour le nombre de convives et les allergies mentionnées lorsqu'il fait une réservation, la commande suivante génère les fichiers nécessaires pour mettre en place un système d'authentification :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % symfony console make:auth
```

En plus de gagner du temps en automatisant la création de classes et de fichiers, cette commande créer un fichier « SecurityController.php » avec un système de login et log out.

Dans le dossier « src/Security », je décommente depuis le fichier « UsersAuthenticatore.php », dans le « public function onAuthenticationSuccess », la redirection lorsqu'une connexion est valide et je commente l'exception :

src > Security > UsersAuthenticator.php >

```php
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
        return new RedirectResponse($targetPath);
    }

    // For example:
    return new RedirectResponse($this->urlGenerator->generate('app_home'));
    // throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
}
```

Une fois cette étape accomplie , je redirige l'utilisateur connecté vers la page d'accueil de l'application Quai Antique. Je modifie quelques class et phrases depuis le HTML du fichier « UsersAuthenticator.php » dans le templates en vue de conserver une cohérence avec les thèmes, les couleurs et le style.

L'API 'IntersectionObserver'

Afin d'avoir un rendu dynamique sur l'interface web pour l'expérience utilisateur, l'API « IntersectionObserver » utilisant du Javascript permet d'ajouter un attrait visuel avec des effets sur les éléments lors du défilement de la page.
Le code suivant crée une instance de l'API **IntersectionObserver()** pour détecter quand des éléments avec la classe « reveal » entrent dans la zone d'intersection avec la racine de la page web. Lorsque la proportion de l'élément visible est supérieure à 0,1, la classe « reveal-visible » est ajoutée à l'élément et l'observation de l'élément est arrêtée.

assets > JS app.js >

```
const ratio = .1
const options = {
    root: null,
    rootMargin: "0px",
    threshold: ratio
}

const handleIntersect = function (entries, observer) {
    entries.forEach(function (entry) {
        if (entry.intersectionRatio > ratio) {
            entry.target.classList.add('reveal-visible')
            observer.unobserve(entry.target)
        }
    })
}

const observer = new IntersectionObserver(handleIntersect, options)
document.querySelectorAll('.reveal').forEach(function (r){
    observer.observe(r)
})
```

J'ai découvert cette API sur le site web de Mozilla. Une documentation est très bien fournie pour mettre en place cette API dans le projet.

Création du formulaire d'inscription

Pour une expérience utilisateur cohérente, la création d'un compte est généralement nécessaire avant de pouvoir se connecter à une application.
La commande suivante génère automatiquement un formulaire de création de compte, qui va être personnalisé en fonction du code graphique que je propose pour l'application Quai Antique :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % symfony console make:registration-form
```

La commande offre différentes options, comme l'envoi d'un e-mail à l'utilisateur après son enregistrement, que je choisis de ne pas utiliser.

Symfony crée un dossier « Form » contenant un fichier nommé « RegistrationFormType.php ». Ce fichier contient le formulaire que je modifie selon mes besoins.
L'objet **$builder** est utilisé pour définir les champs du formulaire de création de compte. J'utilise la méthode **add()** qui me permet d'ajouter un champ de formulaire en spécifiant le type de champ et les options qui lui sont associées. Lorsque le formulaire est soumis, le **$builder** utilise des options globales de PHP (**POST**, **GET**, etc,).

src > Form > RegistrationFormType.php >

```php
public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
        ->add('email', EmailType::class, ['attr' => ['class' => 'form-control mb-3'], 'constraints' =>
        [ new NotBlank(['message' => 'Veuillez saisir une adresse email'])], 'label' => false])
        ->add('lastname',TextType::class, ['attr' => ['class' => 'form-control mb-3'], 'label' => false])
        ->add('firstname', TextType::class, ['attr' => ['class' => 'form-control mb-3'], 'label' => false])
        ->add('address', TextType::class, ['attr' => ['class' => 'form-control mb-3'], 'label' => false])
        ->add('zipcode', TextType::class, ['attr' => ['class' => 'form-control mb-3'],  'label' => false])
        ->add('city', TextType::class, ['attr' => ['class' => 'form-control mb-3'], 'label' => false])
        ->add('nbGuests', IntegerType::class, ['attr' => ['min' => 2, 'max' => 8,
        'class' => 'form-control container--form-fonts mb-3 text-primary',
        'placeholder'=>'exemple: 4'], 'label' => false])
        ->add('allergiesMentioned', TextType::class, ['attr' =>
        ['class' => 'form-control mb-3 container--form-fonts text-primary',
        'placeholder'=>'exemple: fruits de mer'], 'label' => false,])
        ->add('agreeTerms', CheckboxType::class, ['attr' => ['class' => 'form-check-input '],
            'mapped' => false,
            'constraints' => [
                new IsTrue([
                    'message' => 'Vous n\'avez pas accepté les conditions d\'utilisation',
                ]),
            ], 'label' => false
        ])
```

Une fois qu'un utilisateur est inscrit depuis le formulaire valide de l'application web, le code suivant récupère les données entrées par l'utilisateur pour le nombre d'invités et les allergies mentionnées. Si l'utilisateur est connecté, ses informations sont mises à jour avec les données récupérées et la méthode **flush()** est appelée sur l'objet **$manager** pour enregistrer les modifications dans la base de données.

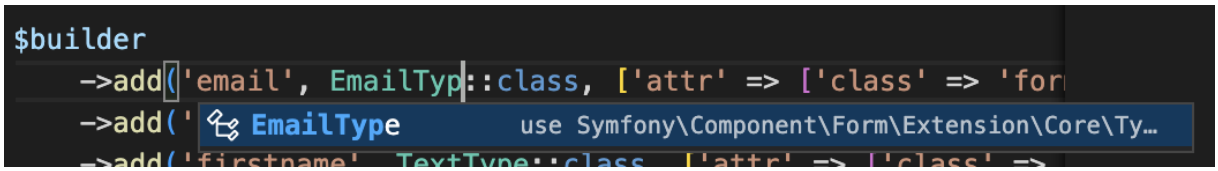
src > Controller > RegistrationController.php

```php
$user = new Users();
$form = $this->createForm(RegistrationFormType::class, $user);
$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {
    //récupère les données pour le nombre d'invités
    $nbGuests = $form->get('nbGuests')->getData();
    $allergiesMentioned = $form->get('allergiesMentioned')->getData();

    // si l'utilisateur est connecté, mettre à jour ses informations
    if ($user) {
        $user->setNbGuests($nbGuests);
        $user->setAllergiesMentioned($allergiesMentioned);
        $this->manager->flush();
    }
```

Pour le bon fonctionnement du code dans les fichiers Controllers, Form, Entity ou autres. Symfony utilise un système d'importation de classes et des n**amespace** avec l'instruction **use** qui sont nécessaires. Ces dépendances m'évitent de spécifier le chemin complet pour les classes appelées dans un fichier à chaque fois.
voici un exemple des dépendances utilisées avec le fichier « RegistrationFormType.php » :

```php
<?php

namespace App\Form;

use App\Entity\Users;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\PasswordType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Validator\Constraints\IsTrue;
use Symfony\Component\Validator\Constraints\Length;
use Symfony\Component\Validator\Constraints\NotBlank;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\IntegerType;

class RegistrationFormType extends AbstractType
```

L'extension « PHP Intelephense » depuis mon VSCode m'aide pour trouver les bonnes dépendances en haut de mes fichiers :



L’ utilisateur connecté

1. les rôles

Avec l’annotation « [ORM\Column] », la propriété **$roles** est stockée automatiquement dans une colonne de la table de la base de données grâce à Doctrine. La valeur **[‘ ROLE_USER‘]** est initialisée.

src > Entity > Users.php

```php
#[ORM\Column]
private array $roles = ['ROLE_USER'];
```

C’est en décommentant l’accès suivant pour la route de l’URL **/profile** et par le système d’authentification de Symfony en se basant sur un utilisateur enregistré dans une base de données que cela permet de sécuriser l’accès à certaines parties de l’application web.

config > packages > security.yaml

```yaml
- { path: ^/profile, roles: ROLE_USER }
```

Il y a deux rôles attribués dans mon application, un pour l’utilisateur connecté et un pour l’administrateur :

config > packages > security.yaml

```yaml
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/profile, roles: ROLE_USER }
```

2. les conditions

En fonction du rôle de l’utilisateur dans le projet restaurant, Quai Antique. J’utilise dans le fichier ou l’on retrouve les menus de la barre de navigation, une condition Twig.
Le code vérifie si l’utilisateur connecté possède le rôle **‘ROLE_ADMIN’** ou **‘ROLE_USER’** et avec la fonction **is_granted()** comme aide je redirige l’utilisateur connecté.

templates > base > navbar.html.twig

```twig
    <!--login & book-->
    <div class="d-grid gap-2 d-md-flex justify-content-md-end">
{% if is_granted('ROLE_ADMIN') %}
  <a href="{{ path('admin')}}"><div class="header--menu-btn me-md-2 ">Administration</div></a>
  <a href="{{ path('app_logout')}}"><div class="header--menu-btn me-md-2">Se déconnecter</div></a>
  <a href="" class="text-decoration-none"><div class="header--btn-booking me-md-2">Réserver</div></a>
{% elseif is_granted('ROLE_USER') %}
    <a href="{{ path('app_account_users')}}"><div class="header--menu-btn me-md-2 ">Vos réservations</div></a>
  <a href="{{ path('app_logout')}}"><div class="header--menu-btn me-md-2">Se déconnecter</div></a>
  <a href="" class="text-decoration-none"><div class="header--btn-booking me-md-2">Réserver</div></a>
{% else %}
  <a href="{{ path('app_register')}}"><div class="header--menu-btn me-md-2">S'inscrire</div></a>
  <a href="{{ path('app_login')}}"><div class="header--menu-btn me-md-2">Se connecter</div></a>
  <a href="" class="text-decoration-none"><div class="header--btn-booking me-md-2">Réserver</div></a>
{% endif %}
</div>
    <!--login & book-->
```

L’utilisateur est redirigé vers une page en fonction de son rôle.
Pour l’administrateur qui détient le rôle admin, il y a un accès à la gestion de contenu administration depuis un bouton ainsi qu’un autre bouton pour se déconnecter.

Pour le l’utilisateur qui a le rôle user, il a un accès à un bouton de déconnexion ainsi qu’à un bouton pour l’accès à son contenu de réservations afin qu’il consulte les réservations déjà effectuées.

Formulaire de réservation

1. Les contraintes du Chef

Le Chef Michant souhaite pour une réservation, que les utilisateurs soient informés du nombre de places disponibles par tranche de 15 minutes, entre l'ouverture et la fermeture du restaurant, tout en respectant un seuil maximum de convives par jour.

2. Le Controller de réservation

Je crée un Controller qui stock toutes les conditions, fonctions et contraintes nécessaires pour assurer le bon déroulement d'une réservation pour l'utilisateur.

3. Le FormType pour la réservation

Le code du formulaire de réservation utilise des fonctions anonymes pour récupérer les entités à afficher dans les champs du formulaire pour le nombre d’invités et les allergies mentionnées lorsqu’un utilisateur est déjà connecté.

src > Form > ReservationsFormType.php >

```
$builder
    ->add('name', EntityType::class,
    ['class' => Users::class, 'query_builder' => function
    (EntityRepository $er){return $er->createQueryBuilder('u')->orderBy('u.lastname', 'ASC');},
    'choice_label' => 'lastname', 'attr' => ['class' => 'form-control mb-3 form-title_style text-primary',
    'placeholder'=>'exemple: Dupont'], 'label' => false])
```

L'objet **EntityRepository** est fourni par le système de gestion d'entités de Doctrine, qui est utilisé par Symfony pour gérer la persistance des données. La fonction anonyme passe ensuite comme argument de la méthode **query_builder**, permettant de définir une requête personnalisée afin de récupérer les entités à afficher dans le champ correspondant.

4. Le fichier Service

Pour m'éviter une surcharge du code dans le fichier « ReservationsController.php », je crée un dossier « Service » qui contient un fichier « ReservationService.php ». Les objets qui sont dans ce fichier sont instanciés et injectés dans le « ReservationsController.php ».

5. Les conditions, fonctions et contraintes

a) Vérification du formulaire soumis

Dans un premier temps je crée un nouvel objet pour la réservation. Cet objet représente une réservation et stocke les données saisies dans le formulaire. Ensuite le « ReservationsFormType.php » est associé à l'objet **$reservations**. La méthode **$reservationForm->handleRequest($request)** est appelée pour traiter la requête http associé au formulaire. Cette méthode vérifie si le formulaire est bien soumis et si les données saisies sont valides. Elle met à jour l'état de l'objet **$reservations** avec les données saisies dans le formulaire.
Enfin si le formulaire est soumis et valide alors le code contenu dans le bloc « if » est exécuté. Dans ce cas, l'objet **$reservations** est récupéré à partir du formulaire en utilisant la méthode **$reservationForm->getData()**. Cet objet est ensuite ajouté à la base de données pour sauvegarder la réservation.

src > Controller > ReservationsController.php

```php
//création d'une réservation
$reservations = new Reservations();

//création du formulaire
$reservationForm = $this->createForm(ReservationsFormType::class, $reservations);

//traite de la requete du formulaire
$reservationForm->handleRequest($request);

//Si le formulaire est soumis et valide, alors on ajoute l'objet $reservations dans la BDD
if ($reservationForm ->isSubmitted() && $reservationForm->isValid()){
    $reservations = $reservationForm->getData();
```

b) La date, l'heure et le seuil

Le code suivant vérifie les conditions pour une réservation. La date et l'heure de la réservation sont récupérées à partir de l'objet **$reservations**. Ensuite, le jour de la semaine correspondant à la date est vérifié, en respectant les horaires du restaurant. si celui-ci est un dimanche alors j'attribue 0, si c'est un autre jour de la semaine, alors c'est 1. Un message **flash()** est envoyé en cas d'erreur.
Ensuite, le code vérifie si le nombre total de convives pour cette date est supérieur ou égal à celui que j'ai mentionné, en l'occurrence 40. Si c'est le cas, cela signifie que le restaurant a

atteint sa capacité maximale pour cette journée et comme pour la date et l'heure vérifié, un message via message **flash()** est envoyé au client en cas d'erreur. Sinon, si le nombre de places disponibles est calculé en soustrayant le nombre total de convives pour cette date à la capacité maximal du restaurant, soit 40.
Si le nombre de places disponibles est compris entre 1 et 4 inclus, un message **flash()** d'avertissement au couleur de la variable **$warning** de Bootstrap est envoyé à l'utilisateur pour l'en informer. En revanche, si le nombre de places disponibles pour cette journée est égale à 0, dans ce cas-là, un message d'erreur de la couleur que j'ai attribué pour la variable **$danger** de Bootstrap est envoyé au client aussi. Sinon, si aucune des actions précédentes est entreprise alors le processus de création de réservation continue.

src > Controller > ReservationsController.php

```php
if ($reservationForm ->isSubmitted() && $reservationForm->isValid()){
    $reservations = $reservationForm->getData();

    // Récupérer la date et l'heure depuis l'objet $reservations
    $date = $reservations->getDate();
    $hour = $reservations->getHours();

    // Vérifie que la date est différente de dimanche (0 pour dimanche, 1 pour lundi, etc.)
    $dayOfWeek = (int)$reservations->getDate()->format('w');
    if ($dayOfWeek === 0) {
        $this->addFlash('error', 'Le restaurant est fermé le dimanche.');
        return $this->redirectToRoute('app_reservations');
    }

    // Vérifie si le nombre total de convives pour cette date est supérieur ou égal à 40
    $totalGuests = $reservationService->countGuestsByDate($date);
    if ($totalGuests + $reservations->getNumberGuests() > 40) {
        $this->addFlash('error', 'Le nombre total de convives pour cette journée a atteint sa limite de 40. Veuillez choisir une autre journée.');
        return $this->redirectToRoute('app_reservations');
    }else {
        $availablePlaces = 40 - $totalGuests;
        if ($availablePlaces > 0 && $availablePlaces < 5) {
            $this->addFlash('warning', 'Il ne reste plus que ' . $availablePlaces . ' places disponibles pour cette journée.');
        } elseif ($availablePlaces == 0) {
            $this->addFlash('error', 'Il n\'y a plus de places disponibles pour cette journée. Veuillez choisir une autre journée.');
            return $this->redirectToRoute('app_reservations');
        }
    }
```

Dans le fichier « ReservationService.php », je créé les fonctions pour trouver les réservations par date et heure, le nombre d'invités par date, les places disponibles par dates.

Avant tout, je défini un constructeur qui prend en compte des classes PHP en paramètre d'entrée.
L'objet **$manager** implémente l'interface **EntityManagerInterface** afin d'interagir avec la base de données. L'objet **$security** implémente la classe **Security** pour gérer les autorisations d'accès des utilisateurs connectés à l'application. L'objet **$reservationRepository** implémente la classe **ReservationsRepository** pour récupérer les données à partir de la base de données pour les réservations effectuées. L'opérateur **$this** assigne chaque propriétés des classes correspondantes.

src > Service > ReservationService.php

```php
class ReservationService
{
    private $security;
    private $manager;
    private $reservationRepository;


    public function __construct(EntityManagerInterface $manager,
    Security $security, ReservationsRepository $reservationRepository)
    {
        $this->manager = $manager;
        $this->security = $security;
        $this->reservationRepository = $reservationRepository;

    }
```

La méthode publique **findReservationByHOur()**, prend en compte deux paramètres d'entrée (**$hour** et **$date**). La méthode retourne un tableau d'objets de type **Reservations**. La méthode **createQueryBuilder()** permet de créer un nouvel objet de la classe **QueryBuilder()** de Doctrine. Je spécifie les colonnes que je souhaite récupérer dans la requête à l'aide de la méthode SQL **select()** qui a comme alias 'r' pour la table des Reservations. J'utilise **from(Reservations::class, 'r')** pour indiquer la table à partir de laquelle je veux récupérer les données. Afin de spécifier les conditions de filtrage des résultats, j'utilise **where()** et **andWhere()** pour récupérer les réservations qui ont une date égale à **$date** et une heure égale à **$hour**. Les conditions sont définies en utilisant **setParameter()** pour attribuer les valeurs aux paramètres de la requête. Enfin, j'appelle la méthode **getQuery()** pour finaliser la construction de la requête. Les résultats sont envoyés sous forme d'un tableau d'objets de type **Reservations**.

src > Service > ReservationService.php

```php
public function findReservationsByHour($hour, $date): array
{
    $qb = $this->manager->createQueryBuilder();
    $qb->select('r')
       ->from(Reservations::class, 'r')
       ->where('r.date = :date')
       ->andWhere('r.hours = :hour')
       ->setParameter('date', $date)
       ->setParameter('hour', $hour);

    $query = $qb->getQuery();
    $reservations = $query->getResult();

    return $reservations;
}
```

Le code suivant prend en paramètre l'objet **$date** qui implémente l'interface **\DateTimeInterface** de la méthode publique **countGuestsByDate()**, retourne un entier représentant le nombre total d'invités pour la date spécifiée. Afin de formater la date au format « Y-m-d » et de la stocker dans la variable **$formattedDate**, la méthode **format** est appelée sur l'objet **$date**. Ensuite, la méthode **createQueryBuilder()** est appelée sur l'objet **$this->reservationRepository** pour créer une requête de type **select** sur la table des réservations. Cette méthode est utilisée pour sélectionner la somme totale de tous les invités dans les réservations pour la date spécifiée. **getQuery()** est utilisé pour obtenir l'objet **Querybuilder** générer à partir de la requête et la méthode **getSingleScalarResul()** appelée pour exécuter la requête et récupérer le nombre total d'invités pour la date spécifiée.

src > Service > ReservationService.php

```php
public function countGuestsByDate(\DateTimeInterface $date): int
{
    $formattedDate = $date->format('Y-m-d');
    $reservations = $this->reservationRepository->createQueryBuilder('r')
        ->select('SUM(r.numberGuests)')
        ->where('r.date LIKE :date')
        ->setParameter('date', $formattedDate.'%')
        ->getQuery()
        ->getSingleScalarResult();

    return $reservations ?: 0;
}
```

Le code pour la méthode publique **getAvailablePlacesByDate()** prend en paramètre d'entrée l'objet **DateTime** qui représente la date pour laquelle le nombre de places disponibles doit être calculé. Je commence par formater la date en utilisant **$date->format('Y-m-d')** afin d'obtenir une chaîne de caractères représentant la date au format 'Y-m-d' (année-mois-jour).Je crée ensuite un nouvel objet de la classe **QueryBuilder()** en utilisant **createQueryBuilder()** pour construire la requête SQL.
Je sélectionne la somme des nombres d'invités avec l'alias 'reservedPlaces' depuis la table de **Reservations** utilisant un alias 'r' pour la représenter. Je récupère la date des réservations qui ont une date égale à la date spécifiée dans la requête. Je définie la valeur du paramètre **$formattedDate** qui est attribué au paramètre ' :date'. Je finalise la construction de la requête et exécute la requête en appelant **getSingleScalarResult()** pour qu'elle renvoie une valeur unique donc la somme des nombres d'invités. Je soustrais le nombre total de places disponible, à savoir 40, au résultat. La méthode retourne le nombre de places disponibles calculé.

src > Service > ReservationService.php

```php
public function getAvailablePlacesByDate(\DateTime $date): int
{
    $formattedDate = $date->format('Y-m-d');
    $qb = $this->manager->createQueryBuilder();
    $qb->select('SUM(r.numberGuests) as reservedPlaces')
    ->from(Reservations::class, 'r')
    ->where('r.date = :date')
    ->setParameter('date', $formattedDate);

    $query = $qb->getQuery();
    $result = $query->getSingleScalarResult();

    $totalPlaces = 40;
    $availablePlaces = $totalPlaces - (int)$result;

    return $availablePlaces;
}
```

c) Garantir que chaque réservation est unique
Dans le « ReservationsController.php », je vérifie si d'autres reservations existe déjà pour une même date et une heure différente.

src > Controller > ReservationsController.php

```php
// Vérifie si une autre réservation existe pour la même date et heure différente
$conflictingReservations = $reservationService->findReservationsByDateAndDifferentHour($date, $hour);

if (!empty($conflictingReservations)) {
    $this->addFlash('error', 'Vous avez déjà une réservation pour cette journée. Veuillez choisir une autre heure.');
    return $this->redirectToRoute('app_reservations');
}

// Vérifie si une autre réservation existe pour la même heure
$conflictReservations = $reservationService->findReservationsByHour($hour, $date);

if (!empty($conflictReservations)) {
    $conflictingHours = [];
    foreach ($conflictReservations as $conflictReservation) {
        $conflictingHours[] = $conflictReservation->getHours()->format('H:i');;
    }
    $this->addFlash('error', 'La plage horaire de ' . $hour->format('H:i') . ' est déjà réservée. Veuillez choisir une autre plage horaire.
    Les heures déjà réservées sont : ' . implode(', ', $conflictingHours) . '.');
    return $this->redirectToRoute('app_reservations');
}
```

Ce code effectue deux vérifications pour s'assurer qu'une réservation est unique avant de l'ajouter à la base de données.
La première vérification utilise la méthode **findReservationsByDateAndDifferentHour()** du « ReservationService.php » pour rechercher d'autres réservations pour la même date mais à une heure différente. Si d'autres réservations sont trouvées, un message d'erreur est ajouté et l'utilisateur est redirigé vers la page de réservation.

src > Service > ReservationService.php

```php
public function findReservationsByDateAndDifferentHour(\DateTime $date, \DateTime $hour): array
{
    return $this->manager->getRepository(Reservations::class)
        ->createQueryBuilder('r')
        ->where('r.date = :date')
        ->andWhere('r.hours != :hour')
        ->setParameter('date', $date)
        ->setParameter('hour', $hour)
        ->getQuery()
        ->getResult();
}
```

La deuxième vérification utilise la méthode **findReservationsByHour()** toujours depuis « ReservationService.php » pour rechercher d'autres réservations pour la même heure. Si d'autres réservations sont trouvées, un message d'erreur est ajouté avec les heures déjà réservées et l'utilisateur est redirigé vers la page de réservation.

Si aucune réservation conflictuelle n'est trouvée, la réservation est ajoutée à la base de données et l'utilisateur est redirigé vers la page de réservation avec un message success.

d) Comparaison des heures par rapport à la base de données
Le code suivant vérifie si toutes les heures d'ouverture pour une journée spécifique ont été réservées en comparant les heures d'ouverture disponibles dans la base de données avec les heures déjà réservées pour la journée en question.
Si toutes les heures d'ouverture ont été réservées, le code affiche un message d'erreur à l'utilisateur et le redirige vers la page de réservation.

src > Controller > ReservationsController.php

```php
// Vérifie si toutes les heures pour cette journée ont été réservées
$allOpeningHours = $openingHourRepository->findAll();
$reservedHours = $reservationService->findReservationsByDate($date);
$availableHours = array_diff($allOpeningHours, $reservedHours);

if (empty($availableHours)) {
    $this->addFlash('error', 'Toutes les heures pour cette journée ont été réservées. Veuillez choisir une autre journée.');
    return $this->redirectToRoute('app_reservations');
}
```

e) La méthode persist() et flush()

Le code suivant est une méthode qui prend un objet **Reservations** en argument et l'enregistre dans la base de données.

Avant l'enregistrement, la méthode utilise le service **Security** pour récupérer l'utilisateur actuel connecté et l'associe à la réservation en cours. Ensuite, la méthode utilise l'objet **EntityManager** pour ajouter la nouvelle réservation à la base de données en appelant la méthode **persist()** et enfin, elle applique les changements en appelant la méthode **flush()**.

src > Service > ReservationService.php

```php
public function persistReservation(Reservations $reservations): void
{

    $user = $this->security->getUser();
    $reservations->setUsers($user);
    $this->manager->persist($reservations);
    $this->manager->flush();

}
```

La fonction **persistReservation()** est injectée dans le fichier **ReservationsController** puis retourne un message **flash()** en cas de succès en redirigeant le tout sur l'URL « app_reservations ».

src > Controller > ReservationsController.php

```php
$reservationService->persistReservation($reservations);

$this->addFlash('success', 'Votre réservation a été enregistrée avec succès !
Vous pouvez faire une autre réservation ou voir vos réservations dans l\'onglet "Vos réservations"');
return $this->redirectToRoute('app_reservations');
}

return $this->render('reservations/index.html.twig', [
    'reservationForm' => $reservationForm->createView(),
    'restaurants' => $restaurantRepository->findBy([]),
    'openinghours' => $openingHourRepository->findBy([]),
    'availablePlaces' => $availablePlaces,
]);
```

Fixtures et utilisation de FakerPHP

Avec les fixtures je créer des données préétablies pour alimenter ma base de données de façon fictives. Je vérifie si les relations fonctionnent correctement entre les tables de la base de données avec les identifiants depuis l'interface de PHPMyAdmin sur la plateforme web alwaysdata.
Je peux également le voir en ligne de commande avec le langage SQL mais pour gagner du temps lors des tests pendant la phase de développement de l'application web du projet restaurant, Quai Antique, je préfère m'orienter vers une interface que me propose PHPMyAdmin.

1. Installation des fixtures

La commande suivante installe le paquet « orm-fixtures » comme une dépendance de développement pour le projet :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % composer require --dev orm-fixtures
```

config > bundles.php

```
Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle::class => ['dev' => true, 'test' => true],
```

En utilisant Doctrine, ce paquet fournit une bibliothèque de classes et d'outils pour me générer des données de test dans ma base de données.
Pour pouvoir me générer de faux utilisateurs et de fausses données pour tester l'application, j'utilise la bibliothèque PHP, « FakerPHP ».
Comme pour le paquet « orm-fixtures », la commande suivante m'installe une dépendance de développement pour le projet en utilisant Composer :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % composer require fakerphp/faker --dev
```

Sur le GitHub de « FakerPHP » une documentation est fournie ainsi que les méthodes utilisées pour le bon fonctionnement de la bibliothèque PHP « FakerPHP ».

2. Création de faux utilisateurs et de l'admin

Je crée un nouveau fichier que je nomme « UsersFixtures.php » dans le dossier « DataFixtures ». Je définis un constructeur en initialisant la propriété privée **$encoder** de type **UserPasswordHasherInterface** qui me fournit des méthodes pour le hachage et la vérification des mots de passe pour les utilisateurs.
Le code suivant utilise la fonction **load()** de la classe **Fixture** pour créer et persister des données de test dans la base de données. Il crée un utilisateur administrateur, un utilisateur standard et une réservation en utilisant la bibliothèque **Faker** pour générer des données aléatoires réalistes. Le mot de passe est hashé pour les utilisateurs en utilisant l'interface **UserPasswordHasherInterface**. Enfin, les données sont persistées en utilisant l'objet **EntityManager**.

src > DataFixtures > UsersFixtures.php

```php
public function load(ObjectManager $manager): void
{

    $admin = new Users();
    $admin->setEmail('admin@mail.fr')
        ->setLastname('Dumilly')
        ->setFirstname('Alfred')
        ->setAddress('124 Rue de la République')
        ->setZipcode('73000')
        ->setCity('Chambéry')
        ->setNbGuests(0)
        ->setAllergiesMentioned('aucune');
        $password = $this->encoder->hashPassword($admin, 'password');
        $admin->setPassword($password)
        ->setRoles(['ROLE_ADMIN']);
        $manager->persist($admin);

        $faker = FakerFactory::create('fr_FR');

        $users = new Users();
        $users->setEmail($faker->email());
        $users->setLastname($faker->lastName());
        $users->setFirstname($faker->firstName());
        $users->setAddress($faker->streetAddress());
        $users->setZipcode(str_replace(' ', '', $faker->postcode()));
        $users->setCity($faker->city());
        $users->setNbGuests($faker->numberBetween(2, 8));
        $users->setAllergiesMentioned($faker->words(5, true));
        $password = $this->encoder->hashPassword($users, 'secret');
        $users->setPassword($password);
        $users->setRoles(['ROLE_USER']);
        $manager->persist($users);

//User éffectue une réservation
$reservations = new Reservations();
$reservations->setName($faker->lastname());
$reservations->setNumberGuests($faker->numberBetween(2, 8));
$reservations->setDate($faker->datetime('Y_m_d'));
$reservations->setHours($faker->datetime('H_i_s'));
$reservations->setAllergies($faker->words(5, true));
$reservations->setUsers($users);
$manager->persist($reservations);

$manager->flush();
```

3. fonctionnalité de Bundle getReference() et addReference()

Dans les fixtures, je décide de créer les plats, menus et les noms des catégories sans passer par **Faker** puisque j'ai besoin d'avoir un rendu clair même si ce n'est qu'une phase de test. J'ai besoin de savoir si les plats sont bien associés aux catégories et de même pour les menus.

La méthode **addReference()** du code suivant est appelée pour enregistrer une référence à une instance de la catégorie avec une clé. Elle est ensuite utilisée avec la méthode **getReference()** dans les « DishesFixtures.php ».

src > DataFixtures > CategoriesFixtures.php

```php
public function load(ObjectManager $manager): void
{
    //a la carte
    $category = new Categorie();
    $category->setName('Entrées');
    $manager->persist($category);
    $this->addReference('entrées', $category);
```

src > DataFixtures > DishesFixtures.php

```php
public function load(ObjectManager $manager): void
{
    //entrées
    $dishe = new Dishe();
    $dishe->setTitle('Soupe à l'oignon');
    $dishe->setDescription('Soupe à l'oignon, crouton aillés, gratiné de parmesan');
    $dishe->setPrice(16);
    $category = $this->getReference('entrées');
    $dishe->setCategory($category);
    $manager->persist($dishe);
```

J'utilise le même procédé avec la méthode **addReference()** et **getReference()** pour les « CategorieMenusFixtures.php ».

4. Charger les fixtures dans la base de données

Pour charger les données de fixtures dans ma base de données, Le bundle **DoctrineFixturesBundle** de Symfony me permet de créer des classes de fixtures et de les charger dans la base de données avec la commande :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % Symfony console doctrine:fixtures:load
```

Une fois la commande exécuté, Symfony supprime toutes les données existantes dans la base de données pour charger les nouvelles créées dans les fichiers de fixtures.

La bibliothèque VichUploader

1. Installation de VichUploader

En utilisant le bibliothèque PHP open source **VichUploader**, je gère facilement les téléchargements des fichiers dans l'application web du projet restaurant, Quai Antique. **VichUploader** est utile principalement pour stocker les fichiers images dans un endroit bien spécifique que je modifie dans le fichier de configuration de l'extension **VichUploader**.

Le gestionnaire de dépendance Composer m'installe **VichUploader** au projet :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % composer require vich/uploader-bundle
```

config > bundles.php

```
Vich\UploaderBundle\VichUploaderBundle::class => ['all' => true],
```

Le **['all' => true]** signifie que **VichUploader** est exécuté en phase de développement et également en phase de production.

2. Configuration de VichUploader

config > packages > vich_uploader.yaml

```yaml
vich_uploader:
    db_driver: orm

    mappings:
        galerie_images:
            uri_prefix: '%galerie_images%'
            upload_destination: '%kernel.project_dir%/public%galerie_images%'
            namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
            delete_on_update: false
            delete_on_remove: false

```

Le paramètre **upload_destination** spécifie l'emplacement sur le système de fichiers où les fichiers téléchargés sont stockés. Cet emplacement est défini à partir de la variable **kernel.project_dir**, qui pointe vers le répertoire racine du projet Symfony, suivi de « /public » et de la variable **galerie_images**.

3. Modification des chemins pour les images

La fonction **vich_uploader_asset()** de **VichUploaderBundle** dans la balise HTML <img> du templates de Twig génère l'URL de l'image à partir de l'objet d'entité et du nom de l'attribut qui stocke l'image. Cette URL est ensuite utilisée comme source pour l'élément « img » dans la page web.

templates > galeries > index.html.twig

```
<img src="{{ asset('uploads/galeries/' ~ image.file) }}" class="card-img w-100 rounded-0" alt="...">
```

4. Compression des images

Afin d'améliorer les performances de l'application web et pour une meilleur expérience utilisateur, je compresse toutes les images en passant par le service de compression d'images en ligne **TinyPNG**.

L'interface utilisateur avec une solution de gestion de contenu - Easy Admin 4

1. Installation de Easy Admin 4

Pour permettre à un administrateur d'un site de pouvoir créer, lire, modifier et supprimer des éléments de l'application (CRUD), Symfony offre avec **Easy Admin 4** une facilité de création d'interface d'administration avec le **Bundle** Symfony. La documentation fournie par le framework est très détaillée et facile à comprendre.
Je passe par Composer pour qu'il soit installé dans les dépendances.

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % composer require easycorp/easyadmin-bundle
```

config > bundles.php

```
EasyCorp\Bundle\EasyAdminBundle\EasyAdminBundle::class => ['all' => true],
```

2. Création du tableau de bord

Avec la commande suivante, cela créer un tableau de bord d'administration pour l'application. La commande génère automatiquement le code nécessaire pour créer le tableau de bord en utilisant une configuration simple et intuitive basée sur des fichiers **YAML**.

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique %  symfony console make:admin:dashboard
```

J'utilise une class Bootstrap pour un meilleur rendu depuis le templates Twig du tableau de bord d'administration.

templates > admin > dashboard.html.twig

```
1   {% extends "@EasyAdmin/layout.html.twig" %}
2
3   {% block content %}
4       <div class="jumbotron">
5           <h1 class="display">Administration</h1>
6           <p class="lead">Bienvenue dans l'interface d'administration du Quai Antique</p>
7       <a href="{{ path('app_home')}}">
8       <button class="btn btn-primary" type="button">Retour accueil</button>
9       </div></a>
10  {% endblock %}
```

3. Sécuriser le backoffice par une connexion

Je configure l'URL « /admin » pour qu'elle soit accessible qu'aux administrateurs qui ont l'adresse e-mail « admin@mail.fr » et le mot de passe « password ». Pour cela, j'effectue des modifications dans le fichier « DashboardController.php » afin de rediriger les utilisateurs non autorisés vers une autre la page de connexion au compte.

src > Controller > Admin > DashboardController.php

```
#[Route('/admin', name: 'admin')]
```

Il est important de souligner que l'accès au tableau de bord administratif est actuellement possible grâce aux fixtures. Tant que le rôle « admin » est défini dans les fixtures pour le compte associé à l'adresse e-mail « admin@mail.fr » et au mot de passe « password », l'accès est autorisé. Toutefois, lors du déploiement de l'application sur une autre base de données, il va être nécessaire de créer le compte administrateur et de lui attribuer le rôle « admin » manuellement.

4. Création des CRUDController avec Easy Admin 4

Pour gérer les opérations CRUD (Create, Read, Update, Delete) sur une entité j'exécute la commande suivante :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % symfony console make:admin:crud
```

Lorsque j'exécute cette commande, une liste d'entités est affichée, me permettant de choisir celle pour laquelle je souhaite créer le CRUD correspondant. Chaque contrôleur CRUD (CRUDController) contient une méthode statique appelée **getEntityFqcn()** qui est définie dans une classe appropriée. De plus, il y a également une méthode **configureFields()** qui me permet de configurer les champs à afficher et à éditer dans les différentes pages liées aux opérations CRUD.

Chaque élément du tableau représente un champ à configurer. Ils sont créés à l'aide de différentes classes de champs fournies par **EasyAdminBundle**, telles que **IdField**, **TextField**, etc.

La méthode **new()** est appelée sur chaque classe de champ pour créer une instance de ce champ. On lui passe en paramètre le nom du champ dans l'entité, suivi d'un label optionnel qui spécifie le libellé à afficher pour ce champ dans l'interface utilisateur.

Afin d'améliorer l'expérience utilisateur pour l'administrateur, j'incorpore des icônes provenant de la bibliothèque **FontAwesome**.

src > Controller > Admin > DashboardController.php

```
yield MenuItem::linkToCrud('Clients inscrits', 'fas fa-user', Users::class);
```

voici un exemple avec l'entité Users :

src > Controller > Admin > UsersCrudController.php

```php
class UsersCrudController extends AbstractCrudController
{
    public static function getEntityFqcn(): string
    {
        return Users::class;
    }


    public function configureFields(string $pageName): iterable
    {
        return [
            IdField::new('id','Identifiant')->hideOnForm(),
            TextField::new('email')->hideOnForm(),
            TextField::new('password')->hideOnIndex()->hideOnForm(),
            TextField::new('lastname', 'Nom')->hideOnForm(),
            TextField::new('firstname', 'Prénom')->hideOnForm(),
            TextField::new('city', 'ville')->hideOnForm(),
        ];
    }
```

Il est également possible de définir une fonction personnalisée pour modifier les droits de l'administrateur, tels que limiter sa capacité à supprimer ou modifier un compte client.

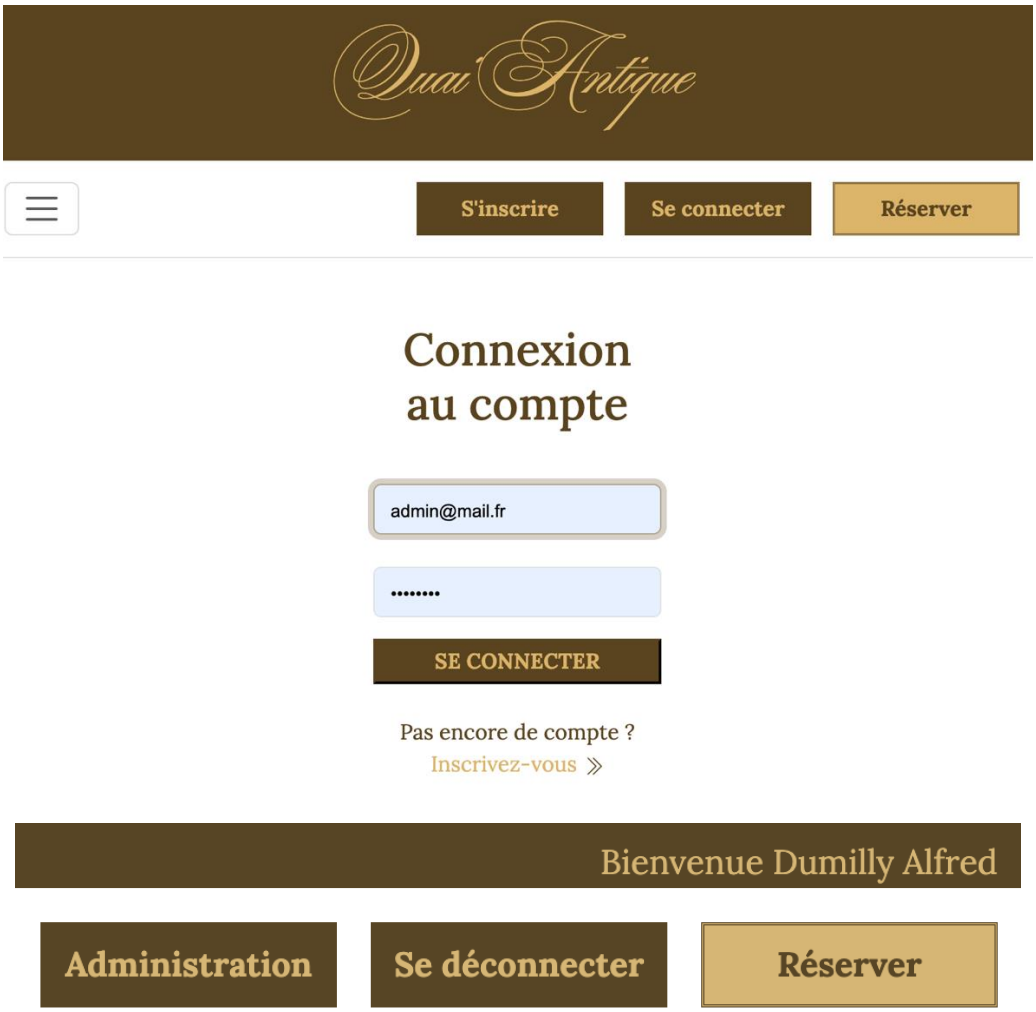
voici un exemple pour l'entité Users :

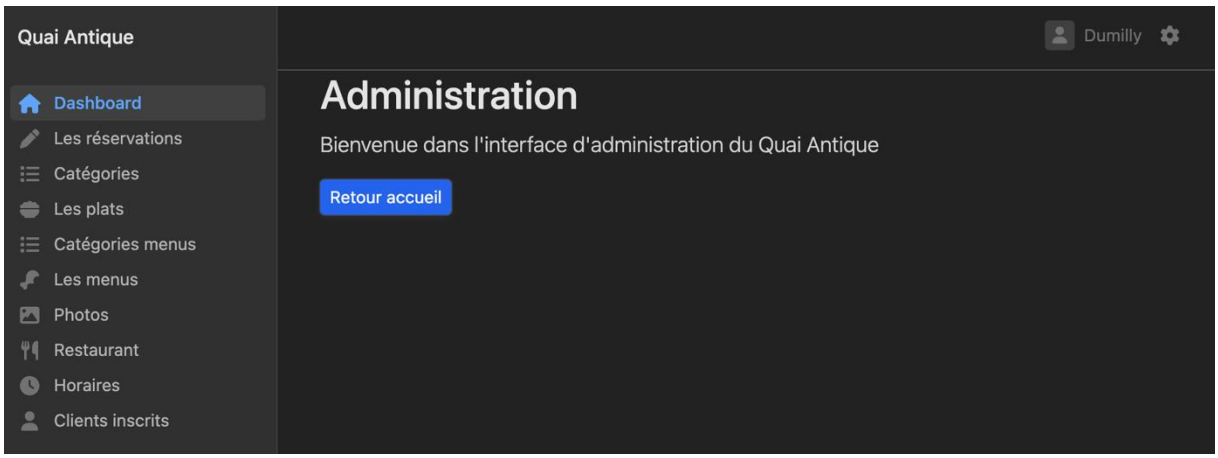
```php
public function configureActions(Actions $actions): Actions
{
    return $actions
        ->disable(Action::DELETE, Action::NEW)
        ->disable(Action::DELETE, Action::EDIT)
        ->disable(Action::DELETE, Action::DELETE);
}
```

Mise en œuvre des composants de l'application de gestion de contenu

1. Accès au tableau de bord

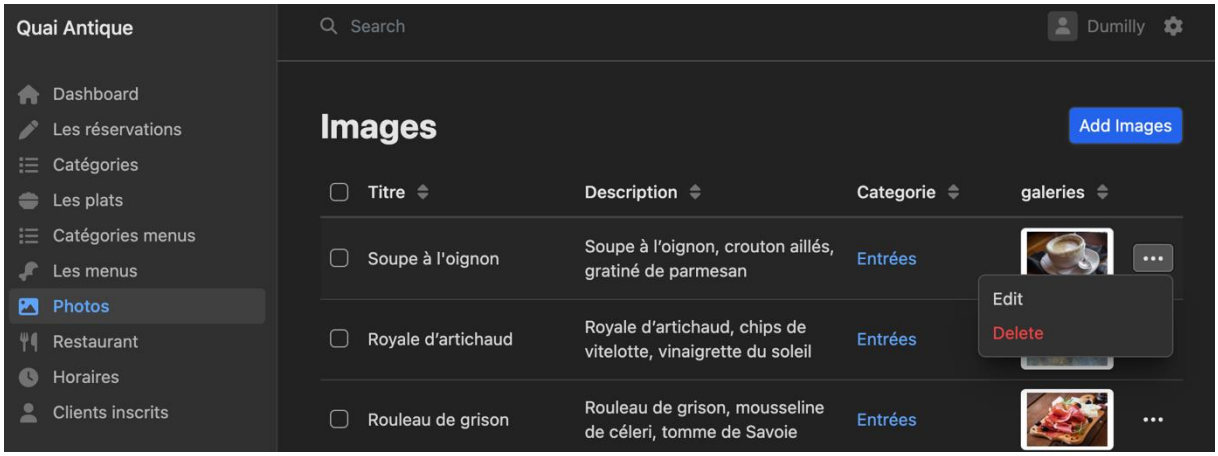
En environnement local, j'ai un accès au tableau de bord à partir du compte administrateur, me permettant ainsi d'effectuer des modifications ou de consulter les réservations en toute simplicité.



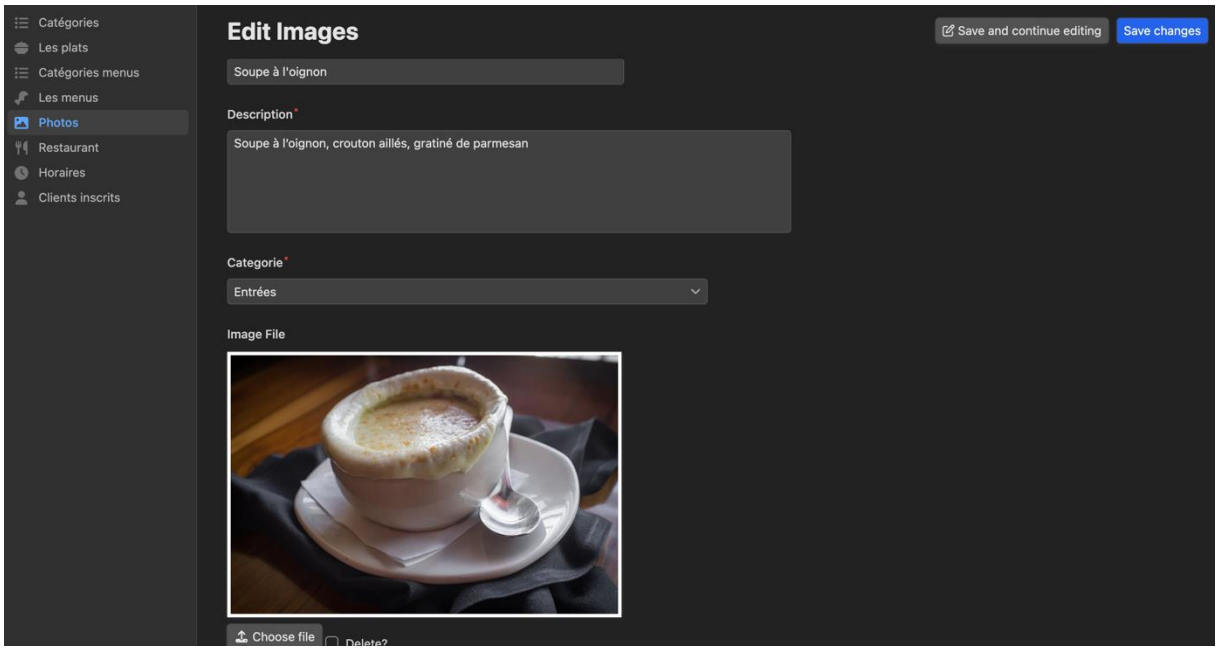


2. Application du CRUD depuis le tableau de bord

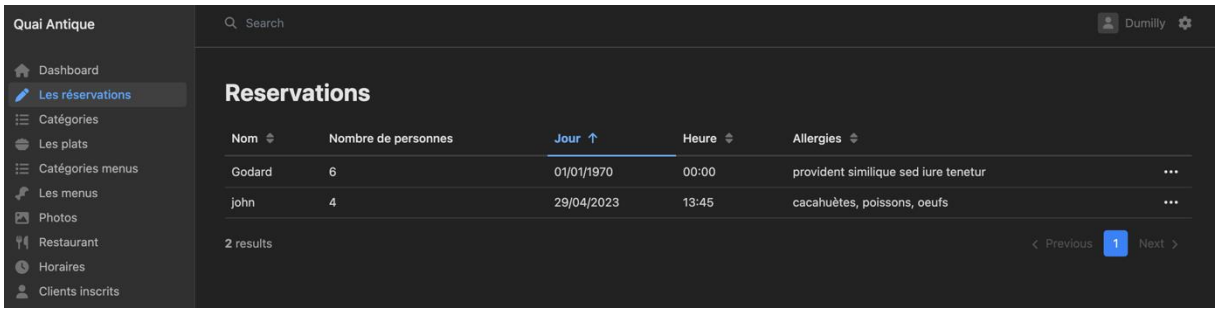
Depuis le tableau de bord, l'administrateur peut éditer des images qui sont dans l'application :



Conformément aux directives du Chef Michant, l'administrateur peut également avoir la possibilité de modifier le titre et la description, comme indiqué.



L'administrateur a la possibilité de visualiser les prochaines réservations à venir, ce qui lui permet de planifier les préparatifs avec son équipe :



L'optimisation du référencement avec le fichier sitemap.xml

1. Protocole respecté

Pour améliorer la visibilité et le classement du site web du restaurant Quai Antique sur les moteurs de recherche, j'accède à la page « sitemap.org » pour consulter le protocole à respecter. Cela permettra de fournir une meilleure expérience de recherche et gagner en visibilité.

2. Controller sitemap.xml

Je crée le controller "SitemapController.php" avec la commande suivante :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % Symfony console make:controller SitemapController
```

Ce controller génère dynamiquement le contenu du fichier « sitemap.xml » en fonction des URLs spécifiées, puis renvoie ce contenu en tant que réponse http avec le type « text/xml ».

J'attribue des contraintes pour le locale et le format à l'attribue #[Route] :

src > Controller > SitemapController.php

```php
#[Route('/sitemap.xml', name: 'app_sitemap', requirements: [
    '_locale' => 'en|fr',
    '_format' => 'xml',
])]
```

La méthode **index()** récupère le nom d'hôte de la requête, génère une liste d'URLs pour le sitemap en utilisant les noms de routes spécifiés, crée le contenu du sitemap à partir d'un template Twig, configure les en-têtes de la réponse avec le type de contenu « text/xml », et renvoie cette réponse contenant le sitemap généré.

src > Controller > SitemapController.php

```php
public function index(Request $request, Hostname $hostname): Response
{
    $hostname = $request->getSchemeAndHttpHost();

    $urls = [];
    $urls[] = ['loc' => $this->generateUrl('app_home')];
    $urls[] = ['loc' => $this->generateUrl('app_dishes')];
    $urls[] = ['loc' => $this->generateUrl('app_menus')];
    $urls[] = ['loc' => $this->generateUrl('app_galeries')];
    $urls[] = ['loc' => $this->generateUrl('app_chef')];
    $urls[] = ['loc' => $this->generateUrl('app_brigades')];
    $urls[] = ['loc' => $this->generateUrl('app_register')];
    $urls[] = ['loc' => $this->generateUrl('app_login')];
    $urls[] = ['loc' => $this->generateUrl('app_reservations')];

    $response = new Response(
        $this->renderView('sitemap/index.html.twig', [
            'urls' => $urls,
            'hostname' => $hostname,
        ]),
        200
    );

    $response->headers->set('Content-type', 'text/xml');

    return $response;
}
```

3. Boucle dans le fichier Twig

Le code dans le fichier template génère un fichier XML structuré selon les spécifications du protocole sitemap, où chaque URL est représentée par une balise « url » contenant la balise « loc » avec l'URL complète.

templates > sitemap > index.html.twig

```
<?xml version="1.0" encoding="UTF-8"?>

<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">

    {% for url in urls %}
        <url>
            <loc>{{ hostname }}{{ url.loc }}</loc>
        </url>
    {% endfor %}

</urlset>
```

À l'intérieur de la balise « urlset », une boucle « for » itère sur chaque élément de la variable « urls ».
Pour chaque élément de « urls », génère une balise « url » contenant une balise « loc » qui représente l'URL de la page dans le sitemap.

Déploiement du site sur Heroku

1. les prérequis

Pourquoi utiliser la plateforme Heroku pour déployer l'application ?

Tout d'abord, Heroku me fournit un environnement de développement intégré basé sur le navigateur, ce qui facilite grandement la gestion de mes applications. De plus, Heroku prend en charge le provisionnement des ressources, la mise en place des serveurs et la configuration automatique de l'infrastructure, ce qui me permet de me concentrer sur le développement de mon application sans me soucier des détails techniques.

En utilisant Heroku, j'ai également accès à la gestion de bases de données et à des outils de surveillance avancés. Cela me permet de stocker et de manipuler facilement les données de mon application, ainsi que de déboguer et d'optimiser ses performances de manière efficace.

Étant donné que j'utilise le langage PHP et le framework Symfony, Heroku est parfaitement compatible avec mes choix technologiques. Je n'ai donc pas à m'inquiéter de problèmes de compatibilité lors du déploiement de mon application.

Heroku me fournit un article sur « devcenter.heroku.com » pour l'installation et l'optimisation de mon application déployé.

Je m'assure de l'avoir correctement installé :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique %  heroku --version
heroku/8.1.3 darwin-x64 node-v16.19.0
```

Étant utilisateur de **Mac**, si je constate que je n'ai pas la version d'Heroku d'installée, je peux facilement l'installer en utilisant la commande suivante :

brew tap heroku/brew && brew install heroku

Cette commande utilise **Homebrew**, un gestionnaire de paquets populaire sur **Mac**, pour ajouter le référentiel Heroku et installer Heroku sur mon système.

Pour assurer la sécurité de mon compte Heroku, je m'inscris sur la plateforme. Lors de la connexion à mon compte, j'ai la possibilité d'optimiser la sécurité en utilisant l'application iOS Salesforce Authenticator.

Pour me connecter, j'ai deux options, soit je peux utiliser la commande « heroku login » dans mon terminal. Cela ouvre une interface de connexion où j'entre mes identifiants et utilise l'application Salesforce Authenticator pour une authentification à deux facteurs ou bien alternativement, je me connecte directement depuis la plateforme Heroku. J'accède à la page de connexion où je saisis mes identifiants, puis je suis invité à utiliser l'application Salesforce Authenticator pour l'authentification à deux facteurs.

En utilisant l'application Salesforce Authenticator lors de la connexion à mon compte Heroku, j'ajoute une couche de sécurité supplémentaire en plus de mes identifiants traditionnels, renforçant ainsi la protection de mon compte contre les accès non autorisés.

2. Création de l’application Heroku

Je m’assure d’être sur le bon dossier du projet restaurant, Quai Antique, puis j’exécute la commande suivante pour créer l’application Heroku.

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % heroku create quai-antique-chambery
```

Une fois que j'ai créé mon application sur Heroku, la plateforme me fournit une URL par défaut. Cependant, si je souhaite personnaliser cette URL, je peux le faire directement depuis l'application Heroku. Dans l'onglet « Settings », j'ai la possibilité de modifier le nom de mon application et d'ajuster ainsi son URL en conséquence. Cette fonctionnalité me permet de choisir un nom plus significatif et en accord avec le projet restaurant, rendant ainsi l’application plus identifiable et facilement accessible pour les utilisateurs.

Pour faciliter le déploiement de l'application, Heroku utilise Git en liant le référentiel (repository) local avec le dépôt Heroku sur la branche principale (master).

3. Création du fichier procfile

Une fois que j'ai choisis le nom de mon application sur la plateforme PaaS (Platform as a Service) Heroku, je crée un fichier de configuration appelé « Procfile ». Ce fichier est utilisé pour déclarer les différents types de processus nécessaires à l'exécution de l’application, qui est basée sur PHP avec Apache comme serveur HTTP.

Pour créer le « Procfile », j'ajoute les lignes correspondantes à chaque type de processus que je souhaite exécuter. Par exemple, si j'ai besoin d'un processus web pour gérer le trafic HTTP, je peux inclure une ligne indiquant « web: heroku-php-apache2 public/ » dans le « Procfile ».

Une fois que j'ai ajouté toutes les lignes nécessaires pour mes processus, je sauvegarde le fichier « Procfile ». Lorsque je vais déployer l’application sur Heroku, la plateforme va lire ce fichier pour comprendre quels processus doivent être exécutés. Elle utilisera ensuite les commandes spécifiées dans le « Procfile » pour démarrer les processus correspondants dans l'environnement d'exécution d'Heroku.

4. Configuration de l’environnement dev/prod

a)Définir la variable d’environnement
En utilisant la commande en ligne, je défini la variable d'environnement sur « prod », ce qui indique que l'environnement de l'application sur Heroku est configuré pour fonctionner en mode production.

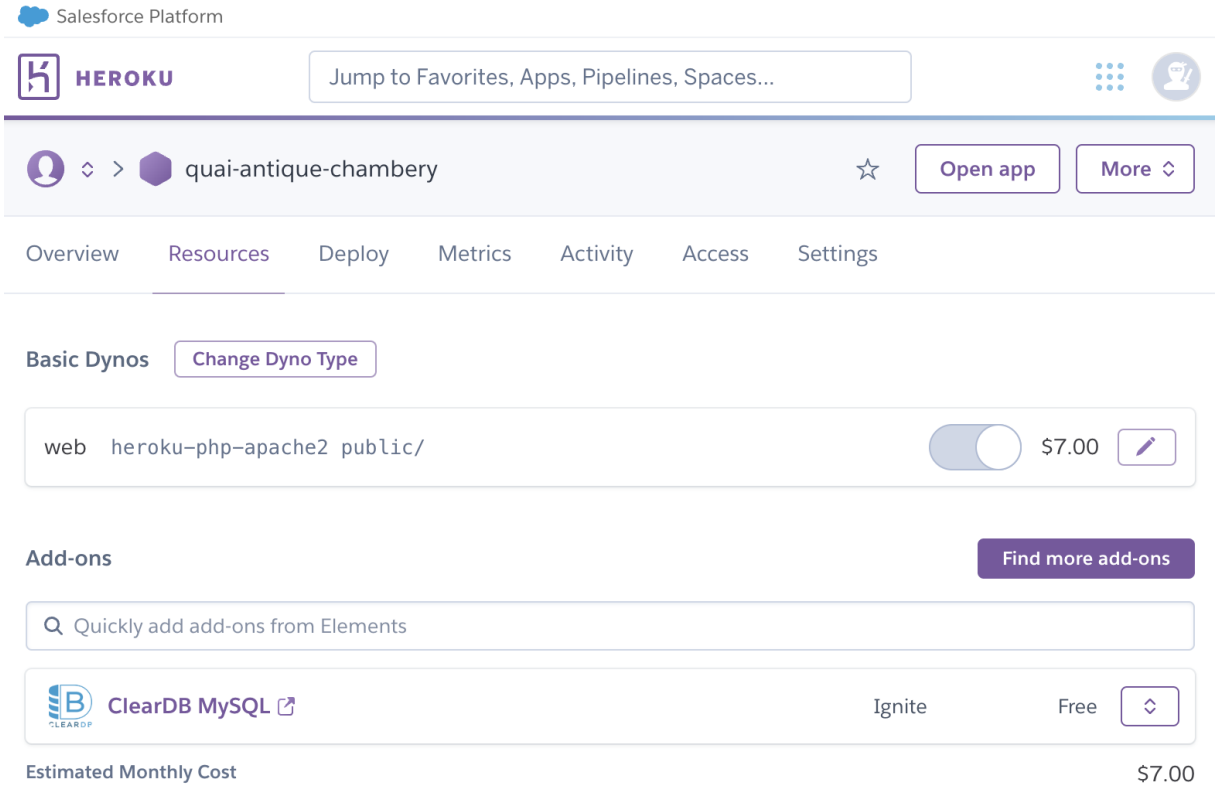
```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % heroku config:set APP_ENV=prod
```

Avec la commande suivante, je vérifie si la variable d'environnement est correctement définie en mode production :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % heroku config:get APP_ENV
prod
berenger@MacBook-Pro-de-Berenger ecf-quai-antique %
```

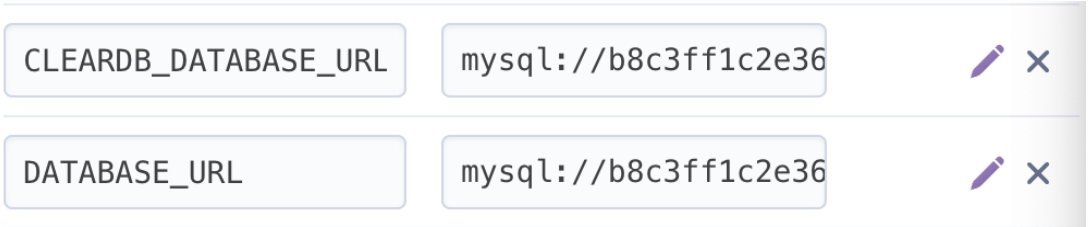
b)Définir la DATABASE URL

Depuis l'interface de la plateforme Heroku, j'ajoute l'add-on **ClearDB** MySQL qui me permet de bénéficier d'un environnement sécurisé pour la base de données MySQL.



c)Définir ses propres variables d'environnement

Toujours depuis l'interface de la plateforme Heroku, j'ai la possibilité de configurer mes propres variables d'environnement.



Pour cela, il me suffit de copier la ligne « mysql » du **CLEARDB_URL** vers le **DATABASE_URL**. Afin de vérifier si tout est correct, j'utilise la commande suivante :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % heroku config:get DATABASE_URL
mysql://b8c3ff1c2e3660:[redacted]-east-06.cleardb.net/heroku_65b301d6ee32807?reconnect=true
berenger@MacBook-Pro-de-Berenger ecf-quai-antique %
```

Maintenant, j'ai accès à un nom d'utilisateur, un mot de passe, un hôte et le nom de la base de données, ce qui me permet d'injecter des données dans cette base de données.

5. Exécuter les migrations

Pour mettre à jour le schéma de base de données de l'application hébergée sur Heroku, j'utilise la commande suivante :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % heroku run php bin/console doctrine:migrations:migrate
```

Cette commande exécute les migrations de doctrine, ce qui applique les modifications correspondantes au schéma de la base de données. Ainsi, je m'assure que ma base de données est à jour avec la dernière structure de schéma définie dans les migrations.

Je vérifie directement en ligne de commande MySQL si toutes les tables ont bien été créées dans la base de données fournie par Heroku. Cela me permet de confirmer que la structure de la base de données est conforme à mes attentes.

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % mysql -u b8c3ff1c2e3660 -p[redacted] -h us-cdbr-east-06
.cleardb.net
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 181220362
Server version: 5.6.50-log MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show databases;
```

```
Connection id:    181223063
Current database: *** NONE ***

+--------------------------+
| Database                 |
+--------------------------+
| information_schema       |
| heroku_65b301d6ee32807 |
+--------------------------+
2 rows in set (5,59 sec)
```

```
mysql> connect heroku_65b301d6ee32807;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Connection id:    181226439
Current database: heroku_65b301d6ee32807
```

```
mysql> show tables;
+-----------------------------------+
| Tables_in_heroku_65b301d6ee32807 |
+-----------------------------------+
| categorie                         |
| categorie_menus                   |
| dishe                             |
| doctrine_migration_versions       |
| images                            |
| menus                             |
| messenger_messages                |
| opening_hour                      |
| reservations                      |
| restaurant                        |
| users                             |
+-----------------------------------+
11 rows in set (0,10 sec)
```

6. Rajouter le buildPack pour nodeJS

L'application qui contient l'URL « https://quai-antique-chambery.herokuapp.com/ » affiche des erreurs 500 sur la page. La raison en est que le dossier « public », qui contient le build, est exclu de Git (gitignore), ce qui signifie que Heroku n'y a pas accès. Afin de résoudre ce problème, je compile les assets en production sur le serveur Heroku.

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % heroku run bash
```

En utilisant la commande « heroku run bash », je me connecte en SSH au serveur sur lequel mon application est hébergée sur Heroku.
Pour ajouter le buildpack Node.js, j'exécute les commandes suivantes :
- « heroku buildpacks « : Je vérifie la liste des buildpacks configurés pour mon application.
- « heroku buildpacks:add heroku/nodejs » : J'ajoute le buildpack Node.js à mon application.
- « heroku config:set NPM_CONFIG_PRODUCTION=false » : J'utilise cette commande pour définir la variable d'environnement **NPM_CONFIG_PRODUCTION** sur « false ». Cela indique à Heroku de ne pas ignorer les packages de développement lors de l'installation des dépendances.

Une fois ces commandes exécutées, je consulte les variables d'environnement et les buildpacks configurés pour l'application dans les paramètres de l'interface de la plateforme Heroku.

Je procède à la vérification de la version de Node.js et npm, puis, à la racine de mon projet, j'ajoute les lignes suivantes dans le fichier package.json :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % node -v
v19.8.1
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % npm -v
9.5.1
```

package.json

```
"engines": {
    "node": "v19.6.*",
    "npm": "9.4.*"
  },
```

Ces lignes spécifient les versions minimales requises de Node.js et npm pour l'application. Cela garantit que Heroku utilise les versions spécifiées lors du déploiement du projet.

7. Configuration du server web avec Symfony - Apache : (.htaccess)

Pour spécifier des directives spécifiques pour la configuration du serveur web Apache et gérer certains aspects du comportement du site, je crée un fichier « .htaccess » à partir de la racine de du projet. Ce fichier permet de définir des règles de réécriture d'URL, des redirections, des paramètres de sécurité, des restrictions d'accès aux fichiers, des configurations de cache, et d'autres personnalisations pour le fonctionnement du site. Il est lu par le serveur Apache lors du traitement des requêtes pour le répertoire où il est placé et ses sous-répertoires. Cela me permet d'ajuster la configuration du serveur Apache de manière spécifique au projet sans avoir à modifier la configuration globale du serveur.

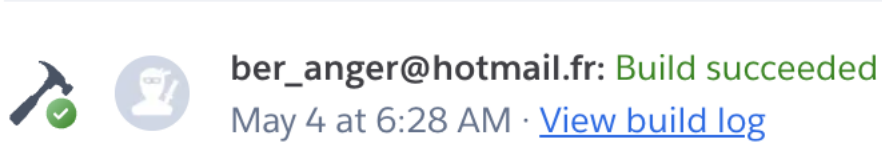
```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % composer require symfony/apache-pack
```

8. Déploiement sur Heroku

Après avoir vérifié que tout est correct, que les derniers commits ont été poussés sur la branche master, je suis prêt à déployer mon application Symfony sur Heroku en utilisant la commande suivante :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % git push heroku master
```

Depuis l'interface de la plateforme Heroku, je vérifie si le processus de build de mon application a réussi.



Attribution du rôle admin

Le Chef Michant souhaite que le maître d'hôtel du restaurant soit l'administrateur du site. Pour permettre l'accès au tableau de bord de l'application en tant qu'administrateur depuis l'application déployé, la première étape consiste à créer le compte pour l'administrateur puis de lui attribuer le rôle « admin ». Ainsi, j'utilise une commande SQL en ligne pour récupérer l'identifiant du maître d'hôtel dans la base de données et lui attribuer le rôle d'administrateur.

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % mysql -u b8c3ff1c2e3660 -p[redacted] -h us-cdbr-east-06
.cleardb.net
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 181220362
Server version: 5.6.50-log MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show databases;
```

```
Connection id:    181223063
Current database: *** NONE ***

+--------------------------+
| Database                 |
+--------------------------+
| information_schema       |
| heroku_65b301d6ee32807 |
+--------------------------+
2 rows in set (5,59 sec)
```

```
mysql> connect heroku_65b301d6ee32807;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Connection id:    181226439
Current database: heroku_65b301d6ee32807
```

```
mysql> show tables;
+---------------------------------+
| Tables_in_heroku_65b301d6ee32807 |
+---------------------------------+
| categorie                       |
| categorie_menus                 |
| dishe                           |
| doctrine_migration_versions     |
| images                          |
| menus                           |
| messenger_messages              |
| opening_hour                    |
| reservations                    |
| restaurant                      |
| users                           |
+---------------------------------+
11 rows in set (0,10 sec)
```

Après avoir accédé à la base de données à l'aide de commandes SQL et je navigue vers la base de données appropriée. Ensuite, je recherche les tables et sélectionne spécifiquement la table des utilisateurs.

```
mysql> select * from users;
```

Après avoir identifié l'ID de l'utilisateur qui doit être promu administrateur, j'exécute la commande suivante :

```
mysql> UPDATE `users` SET `roles`='["ROLE_ADMIN"]' WHERE `id`=24;
```

Désormais, je constate que l'utilisateur ayant l'identifiant 24 possède le rôle d'administrateur, lui permettant ainsi de gérer les images, les titres, les descriptions et les réservations à partir du tableau de bord de l'application. Le maître d'hôtel n'aura qu'à se connecter et accéder à la section d'administration de l'application.

Vérification du bon fonctionnement de l'application

Je vais maintenant simuler l'expérience d'un client souhaitant consulter les plats et menus, voir les photos des plats, créer un compte, réserver une table et vérifier ses réservations à venir. Je vais effectuer plusieurs tests de réservation, tels que réserver les jours de fermeture pour vérifier si les messages d'avertissement s'affichent correctement. Je vais également effectuer des réservations multiples sans compte client pour vérifier si le nombre de places est effectivement limité par jour.

Tout se déroule parfaitement comme en local, il ne me reste plus qu'à partager le lien de l'application avec le Chef Michant. Bien sûr, il peut y avoir quelques détails à peaufiner, mais si je réponds aux exigences du Chef Michant, je peux considérer que ses souhaits et contraintes ont été respectés avec succès.

Difficultés rencontrées lors du développement du projet

Lors de la réalisation de ce projet, j'étais encore en phase d'apprentissage. Plus je pratiquais, plus je comprenais le framework Symfony et les avantages qu'il offrait. Au départ, j'ai été confronté à de nombreuses erreurs que je ne comprenais pas, mais avec le temps, j'ai fini par les comprendre.

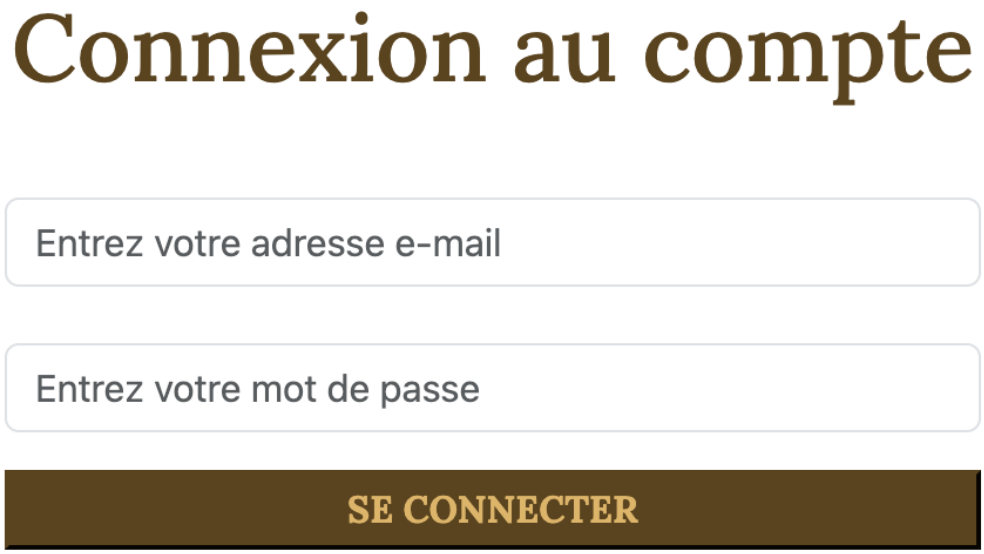
À plusieurs reprises, je me suis engagé sur des chemins qui n'étaient pas forcément les bons. Par exemple, j'ai commencé le projet en effectuant un test sur la base de données en utilisant le logiciel **MAMP** pour configurer mon environnement de développement local. Cependant, j'ai réalisé que la version de MySQL utilisée par le logiciel **MAMP** était inférieure à celle utilisée pour le projet du restaurant, Quai Antique. J'ai alors compris que je devais opter pour un autre schéma de stockage de ma base de données.

C'est grâce à la formation que je suivais qu'un formateur a dispensé des cours sur la plateforme alwaysdata. J'ai donc décidé de m'orienter vers cette plateforme pour héberger ma base de données en local.

Fonctionnalité la plus représentative : L’authentification

1. Jeu d’essai

Afin de faciliter la saisie du formulaire de réservation, ainsi que la consultation des réservations passées ou à venir, l'utilisateur doit s'authentifier en utilisant son adresse e-mail et son mot de passe.



Le mot de passe est stocké dans la base de données, mais pour garantir la sécurité des utilisateurs enregistrés, il est hashé, c'est-à-dire qu'il est chiffré de manière spécifique pour qu’il ne puisse pas apparaître en clair.

id	email	roles	password	lastname	firstname	address	zipcode	city	nb_guests	allergies_mentioned
14	admin@mail.fr	["ROLE_ADMIN"]	$2y$13$bvHK0qago8jdA/2rya32J…	Dumilly	Alfred	124 Rue de la République	73000	Chambéry	0	aucune
15	rlaine@camus.com	["ROLE_USER"]	$2y$13$6nSnsxkRceKFrIxehmBDF…	Boulanger	Suzanne	74, rue Claire Guillou	05074	Dupont	3	harum ut possimus consectetur rem
16	johndoe@mail.fr	["ROLE_USER"]	$2y$13$VrJBMxtwIlcpHrXs6e5GZe…	doe	john	11 rue du général	73000	Chambéry	8	fruit de mer

En utilisant le **MakerBundle** de Symfony et la commande CLI suivante :

```
berenger@MacBook-Pro-de-Berenger ecf-quai-antique % symfony console make:auth
```

Cela me permet donc de générer rapidement un système d'authentification dans mon application Symfony.

Lorsque j'exécute cette commande, elle me guide à travers une série d'options pour configurer mon système d'authentification. Je choisis le type d'authentification que je souhaite mettre en place, comme l'authentification basée sur les formulaires ou l'authentification par token. Je spécifier l'entité que j'utilise pour représenter les utilisateurs.

Une fois que j'ai fourni ces informations, la commande **make:auth** génère automatiquement le code nécessaire pour mettre en place l'authentification dans mon application Symfony. Cela inclut la création des classes, des contrôleurs, des vues et des fichiers de configuration nécessaires.

Grâce à la commande **make:auth**, je créer rapidement un système d'authentification robuste dans mon application Symfony. Cela me permet de gagner du temps en évitant d'écrire manuellement tout le code requis et de bénéficier d'une structure cohérente grâce aux conventions de Symfony.

Le contrôleur suivant gère les actions de connexion et de déconnexion des utilisateurs. Il récupère les éventuelles erreurs de connexion, le nom d'utilisateur précédemment saisi, ainsi que d'autres données liées aux restaurants et aux horaires d'ouverture, puis les transmet à la vue appropriée pour l'affichage.

src > Controller > SecurityController.php

```php
class SecurityController extends AbstractController
{

    #[Route(path: '/connexion', name: 'app_login')]
    public function login(
    AuthenticationUtils $authenticationUtils,
    RestaurantRepository $restaurantRepository,
    OpeningHourRepository $openingHourRepository): Response
    {
        // if ($this->getUser()) {
        //     return $this->redirectToRoute('target_path');
        // }

        // get the login error if there is one
        $error = $authenticationUtils->getLastAuthenticationError();
        // last username entered by the user
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render('security/login.html.twig', [
            'last_username' => $lastUsername,
            'error' => $error,
            'restaurants' => $restaurantRepository->findBy([]),
            'openinghours' => $openingHourRepository->findBy([])
        ]);
    }

    #[Route(path: '/logout', name: 'app_logout')]
    public function logout(): void
    {
        throw new \LogicException('This method can be blank.');
    }
}
```

La classe suivante **UsersAuthenticator** définit la logique d'authentification personnalisée pour le formulaire de connexion. Elle récupère les informations d'identification de l'utilisateur, crée un objet **Passport** correspondant et gère les redirections après une authentification réussie.

src > Security > UsersAuthenticator.php

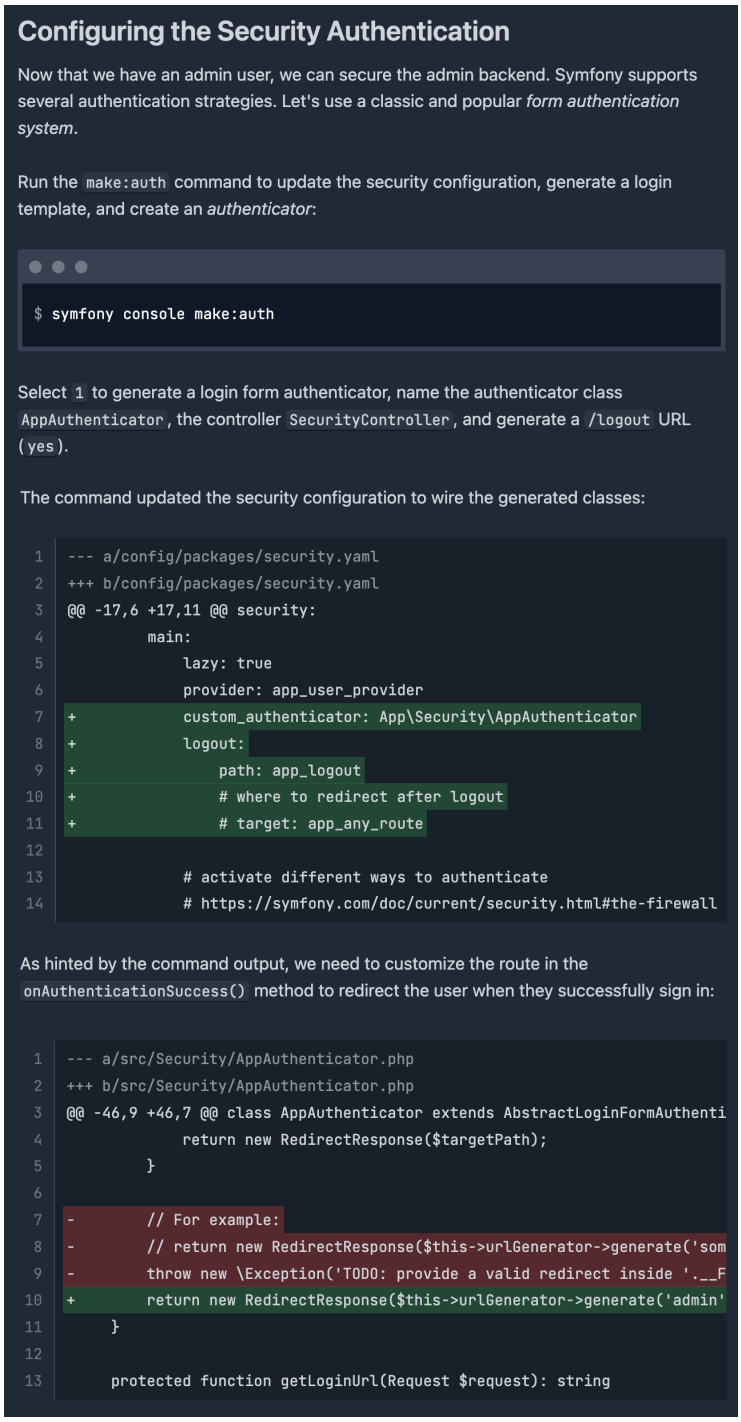
```php
class UsersAuthenticator extends AbstractLoginFormAuthenticator
{
    use TargetPathTrait;

    public const LOGIN_ROUTE = 'app_login';

    public function __construct(private UrlGeneratorInterface $urlGenerator)
    {
    }

    public function authenticate(Request $request): Passport
    {
        $email = $request->request->get('email', '');

        $request->getSession()->set(Security::LAST_USERNAME, $email);

        return new Passport(
            new UserBadge($email),
            new PasswordCredentials($request->request->get('password', '')),
            [
                new CsrfTokenBadge('authenticate', $request->request->get('_csrf_token')),
            ]
        );
    }

    public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
    {
        if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
            return new RedirectResponse($targetPath);
        }

        // For example:
        return new RedirectResponse($this->urlGenerator->generate('app_home'));
        // throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
    }

    protected function getLoginUrl(Request $request): string
    {
        return $this->urlGenerator->generate(self::LOGIN_ROUTE);
    }
}
```

2. Veille effectuée sur les vulnérabilités de sécurité

Il est fort recommandé de s'informer sur la sécurité et les vulnérabilités lié à l'informatique. C'est pourquoi le blog Symfony (https://symfony.com/blog/category/security-advisories) en plus des sites **CVE** (Common Vulnerabilities and Exposures) ou **OWASP** (Open Web Application Security Project) offre une information continue sur la sécurité et les vulnérabilités lié directement à l'application.
La documentation de Symfony propose également des informations et des conseils en matière de sécurité concernant l'authentification des utilisateurs.

3. Recherche à partir de site anglophone, extrait du site anglophone et sa traduction

Voici un extrait de la documentation Symfony permettant d'authentification du utilisateur : (https://symfony.com/doc/current/the-fast-track/en/15-security.html#configuring-the-security-authentication)



Configuring the Security Authentication

Now that we have an admin user, we can secure the admin backend. Symfony supports several authentication strategies. Let's use a classic and popular *form authentication system*.

Run the `make:auth` command to update the security configuration, generate a login template, and create an *authenticator*:

```
$ symfony console make:auth
```

Select `1` to generate a login form authenticator, name the authenticator class `AppAuthenticator`, the controller `SecurityController`, and generate a `/logout` URL (`yes`).

The command updated the security configuration to wire the generated classes:

```
--- a/config/packages/security.yaml
+++ b/config/packages/security.yaml
@@ -17,6 +17,11 @@ security:
         main:
             lazy: true
             provider: app_user_provider
+            custom_authenticator: App\Security\AppAuthenticator
+            logout:
+                path: app_logout
+                # where to redirect after logout
+                # target: app_any_route

             # activate different ways to authenticate
             # https://symfony.com/doc/current/security.html#the-firewall
```

As hinted by the command output, we need to customize the route in the `onAuthenticationSuccess()` method to redirect the user when they successfully sign in:

```
--- a/src/Security/AppAuthenticator.php
+++ b/src/Security/AppAuthenticator.php
@@ -46,9 +46,7 @@ class AppAuthenticator extends AbstractLoginFormAuthenti
             return new RedirectResponse($targetPath);
         }

-        // For example:
-        // return new RedirectResponse($this->urlGenerator->generate('som
-        throw new \Exception('TODO: provide a valid redirect inside '.__F
+        return new RedirectResponse($this->urlGenerator->generate('admin'
     }

     protected function getLoginUrl(Request $request): string
```

Configuration de l'authentification de sécurité
Maintenant que nous avons un utilisateur administrateur, nous pouvons sécuriser le backend administratif. Symfony prend en charge plusieurs stratégies d'authentification. Utilisons un système d'authentification par formulaire classique et populaire.

Exécutez la commande make:auth pour mettre à jour la configuration de sécurité, générer un modèle de connexion et créer un authentificateur :

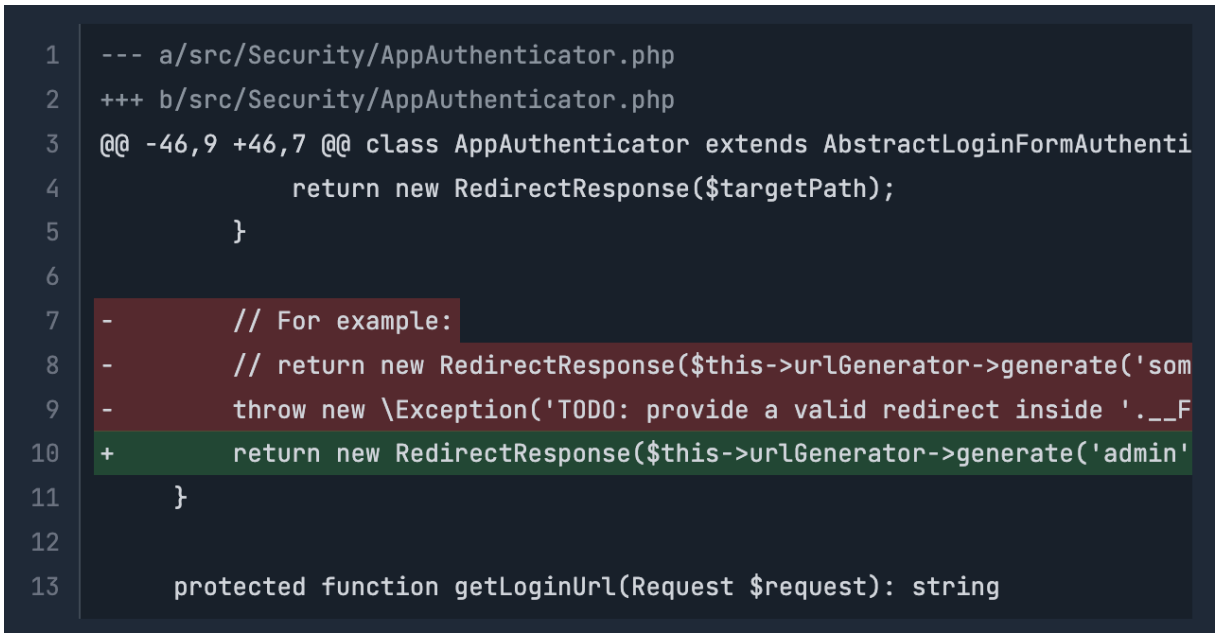
```
$ symfony console make:auth
```

Sélectionnez 1 pour générer un authentificateur de formulaire de connexion, nommez la classe d'authentificateur AppAuthenticator, le contrôleur SecurityController, et générez une URL /logout (oui).

La commande a mis à jour la configuration de sécurité pour connecter les classes générées :

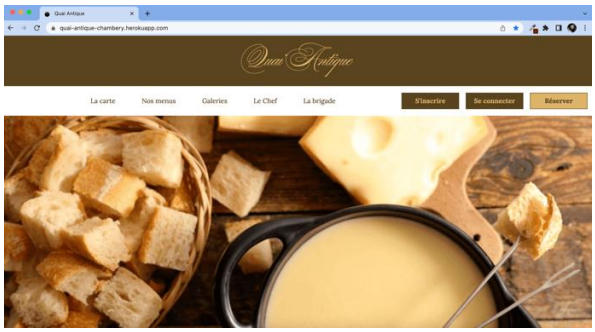
```
--- a/config/packages/security.yaml
+++ b/config/packages/security.yaml
@@ -17,6 +17,11 @@ security:
         main:
             lazy: true
             provider: app_user_provider
+            custom_authenticator: App\Security\AppAuthenticator
+            logout:
+                path: app_logout
+                # where to redirect after logout
+                # target: app_any_route

             # activate different ways to authenticate
             # https://symfony.com/doc/current/security.html#the-firewall
```

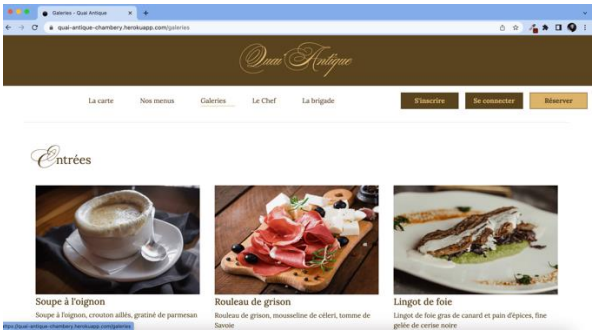
Comme indiqué par la sortie de la commande, nous devons personnaliser la route dans la méthode onAuthenticationSuccess() pour rediriger l'utilisateur lorsqu'il se connecte avec succès :

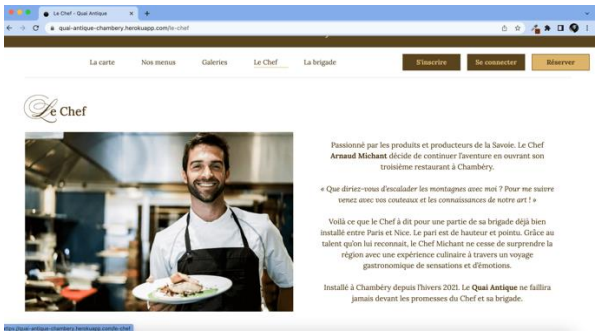


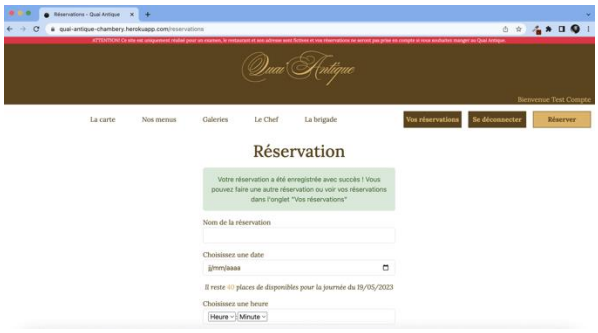
```
--- a/src/Security/AppAuthenticator.php
+++ b/src/Security/AppAuthenticator.php
@@ -46,9 +46,7 @@ class AppAuthenticator extends AbstractLoginFormAuthenti
             return new RedirectResponse($targetPath);
         }

-        // For example:
-        // return new RedirectResponse($this->urlGenerator->generate('som
-        throw new \Exception('TODO: provide a valid redirect inside '.__F
+        return new RedirectResponse($this->urlGenerator->generate('admin'
     }

     protected function getLoginUrl(Request $request): string
```

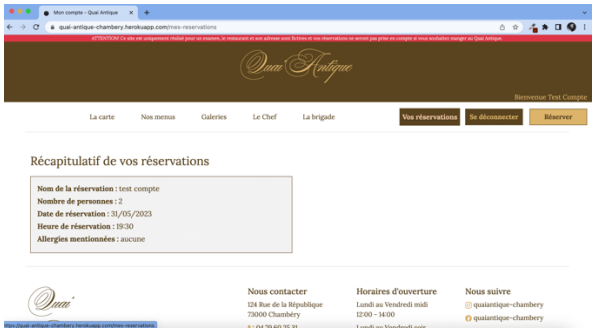
Annexes

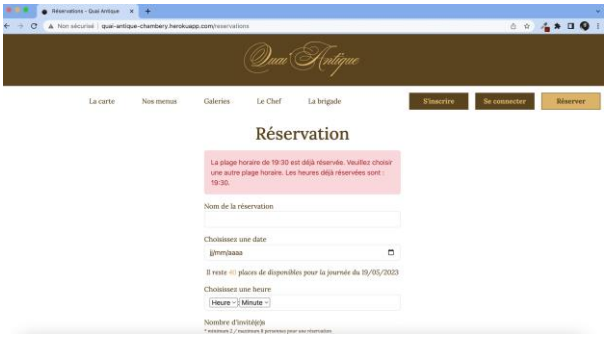

Quai Antique
La carte
Nos menus
Galeries
Le Chef
La brigade
S'inscrire
Se connecter
Réserver

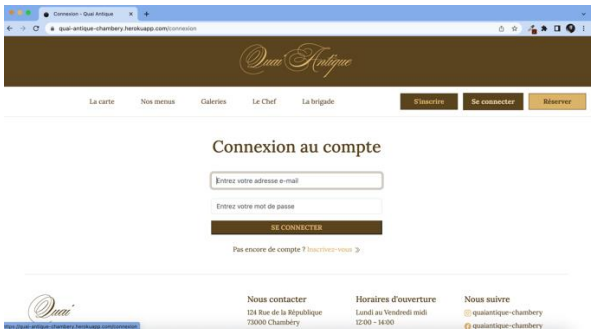

Quai Antique
Entrées
Soupe à l'oignon
Rouleau de grison
Lingot de foie

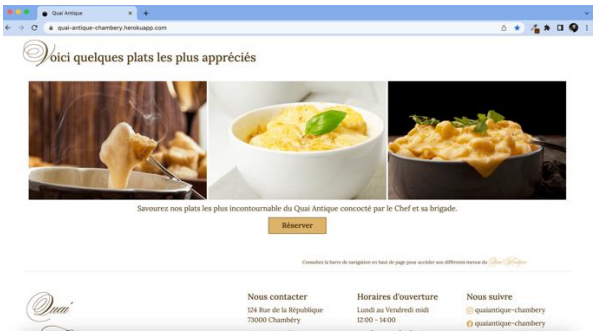

Le Chef
Passionné par les produits et producteurs de la Savoie. Le Chef Arnaud Michant décide de continuer l'aventure en ouvrant son troisième restaurant à Chambéry.


Quai Antique
Réservation
Votre réservation a été enregistrée avec succès ! Vous pouvez faire une autre réservation ou voir vos réservations dans l'onglet "Vos réservations"
Nom de la réservation
Choisissez une date
Choisissez une heure


Quai Antique
Récapitulatif de vos réservations
Nom de la réservation : test compte
Nombre de personnes : 2
Date de réservation : 31/05/2023
Heure de réservation : 19:30
Allergies mentionnées : aucune


Quai Antique
Réservation
La plage horaire de 19:30 est déjà réservée. Veuillez choisir une autre plage horaire. Les heures déjà réservées sont : 19:30.
Nom de la réservation
Choisissez une date
Choisissez une heure


Quai Antique
Connexion au compte
Entrez votre adresse e-mail
Entrez votre mot de passe
SE CONNECTER


Quai Antique
Voici quelques plats les plus appréciés
Savourez nos plats les plus incontournable du Quai Antique concocté par le Chef et sa brigade.
Réserver

Remerciements

Je souhaite exprimer ma profonde gratitude envers les formateurs de la plateforme STUDI, à savoir Thomas BUREAU DU COLOMBIER, Ala ATRASH, Damien BOITEUX, Nathan DERHORE, Gaëtan ROLÉ-DUBRUILLE, Charline LAPORTE, Antony LAPLANE, Jérémy TAUPIN, Julien MARTRES et Carlen LAURENT. Leur dévouement, leur soutien, leurs conseils techniques et leurs encouragements ont été d'une valeur inestimable tout au long du développement de l'application, en m'aidant à résoudre les problèmes rencontrés.

Je tiens également à remercier chaleureusement les apprenants de la plateforme STUDI, à savoir Rebecca GERREY, Hanan REBAIA, Julian CALEGARI, Kevin CAFOLLA et David LE GOUELLEC, pour leur précieuse aide et leurs échanges sur un Discord dédié à ce projet.

Bien que j'aie travaillé seul sur ce projet, j'ai ressenti un soutien collectif tout au long du processus.

Le projet du restaurant **Quai Antique** a représenté une avancée majeure dans mon parcours d'apprentissage, marquant une étape significative dans mon développement.

FERGUENIS Bérenger

2023