Réaliser la partie front responsive

Table des matières

I. Contexte

Durée (en minutes) : 120

Environnement de travail :

- Ordinateur avec Windows, MacOS ou Linux
- Figma
- Visual Studio Code

Prérequis : connaître Git et avoir des bases de HTML, SASS, JS, et Bootstrap

Objectifs de la partie :

- Apprendre à créer un projet
- Interagir avec Git
- Savoir réaliser le visuel d'une page web

Contexte

Dans un cadre professionnel, il est fondamental de savoir créer des pages web agréables à regarder, fluides et dynamiques. Pour cela, vous avez besoin de vous entourer des bons outils, et de savoir les faire fonctionner ensemble. C'est ce que nous allons faire dans ce cours : nous allons aborder le début de la réalisation de la partie *front* d'un site web sans framework JS.

Dans ce cours, nous allons réaliser la partie *front responsive* de notre application. L'objectif est de coder la partie visuelle pour le visiteur. Nous commencerons donc par la création du projet sur VS Code, nous gérerons ensuite notre code source avec Git et nous l'hébergerons sur Gitub. Nous allons voir comment gérer notre visuel avec le framework CSS Bootstrap. Nous n'utiliserons pas de framework JavaScript : nous utiliserons un module de routage déjà codé.

II. L'initialisation du projet

A. La création du projet et l'envoi sur Git

Définition **Un projet sur VS Code**

Visual Studio Code est un éditeur de texte. Pour lui, un projet est finalement uniquement un dossier sur votre ordinateur. Pour créer votre projet, vous avez juste à choisir où mettre votre dossier sur votre machine, à ouvrir ce dossier avec VS Code, et vous pourrez commencer à travailler.

Méthode **Git**

Nous allons utiliser l'intégration de Git sur VS Code pour gérer notre projet avec Git. Le but est d'initialiser le repository Git en local, de créer le repository distant, et de faire des allers-retours entre le local et le distant pour avoir une copie de notre repository sur GitHub.

B. L'ajout de la navigation

Le framework JavaScript

Le but d'un framework JavaScript est de vous donner un **cadre de travail**, c'est-à-dire de vous guider lors du développement de la partie front de votre application. Vous aurez des outils de routage, de style, de sécurité, etc.

Depuis quelques années, certains développeurs décident de se passer de framework JavaScript pour coder l'application eux-mêmes à la main, en utilisant toute la puissance de ce langage : on appelle ça le **VanillaJS**.

Nous n'allons pas utiliser de framework JavaScript pour ce projet, mais nous allons explorer JavaScript pour voir ce qu'il est possible de faire uniquement avec ce langage.

Définition **La navigation**

La navigation, en développement *front-end*, fait référence à la manière dont les utilisateurs interagissent avec une application web, en se déplaçant d'une page à une autre ou en explorant différentes sections de l'interface utilisateur. Sur un site statique (uniquement du HTML), on parcourt les fichiers du site et on affiche tels quels les fichiers HTML du site. Ce n'est pas idéal, car le HTML est un langage peu puissant et on sera vite limité dans notre développement.

Exemple **La limite du HTML**

L'un des problèmes courants est que HTML ne propose pas nativement de mécanisme permettant d'inclure un menu ou un contenu récurrent dans plusieurs pages. Par conséquent, pour afficher le même menu sur chaque page, il est nécessaire de **dupliquer** le code HTML correspondant dans chaque fichier HTML individuel. Cela signifie qu'en cas de modification du menu, il faudra mettre à jour chaque occurrence du code dupliqué, ce qui peut rapidement devenir fastidieux et propice aux erreurs. Pour cela, nous mettons généralement en place un système de routage.

Définition **Le routage**

Le routage, en développement front-end, fait référence à la gestion de la navigation entre différentes pages ou vues d'une application web du côté client, c'est-à-dire dans le navigateur de l'utilisateur. Il permet de définir des règles et des comportements pour gérer les transitions entre les différentes URL ou routes de l'application.

Lorsque l'utilisateur clique sur un lien ou effectue une action qui modifie l'URL, le routage détecte cet événement et déclenche une fonction ou un gestionnaire défini par le développeur. Ce gestionnaire est responsable de mettre à jour l'état de l'application et d'afficher la vue correspondant à l'URL actuelle.

Le routage utilise une **table de correspondance** qui associe des URL spécifiques à des vues ou à des composants. Lorsqu'une URL est détectée, le routage consulte cette table pour trouver la vue ou le composant correspondant à l'URL. Il peut également extraire des paramètres de l'URL, tels que des identifiants ou des valeurs, et les transmettre à la vue ou au composant approprié.

Dans notre cas, nous allons utiliser un système de routage basique codé en JavaScript. Il est composé de 3 fichiers :

- Le fichier **Route.js** définit une classe Route qui représente une route de l'application. Chaque route a une URL, un titre, un chemin vers un fichier HTML, un chemin vers un fichier JavaScript facultatif.
- Le fichier **allRoutes.js** crée un tableau « *allRoutes* » contenant toutes les routes de l'application. Chaque route est créée en utilisant la classe « *Route* » avec les paramètres appropriés. Il définit également la variable « *websiteName* », qui représente le nom du site web.
- Le fichier **router.js** importe la classe Route et les variables « *allRoutes* » et « *websiteName* » du fichier allRoutes.js. C'est lui qui contient la logique de routage.

Le système utilise une structure de routes qui associe des URL à des informations sur les fichiers HTML et JavaScript correspondants à chaque vue. Les routes sont définies dans un fichier allRoutes.js, où chaque route est créée à l'aide d'une classe Route. Cette classe contient des propriétés telles que l'URL, le titre et les chemins vers les fichiers HTML et JavaScript de la vue.

Lorsque l'application est chargée ou lorsqu'un événement de routage se produit (comme un clic sur un lien), le système de routage récupère l'URL actuelle et recherche la route correspondante. Cela se fait en utilisant une fonction `getRouteByUrl` qui compare l'URL avec les URL définies dans les routes. Si une correspondance est trouvée, les informations de la route sont utilisées pour charger le contenu de la vue associée.

Pour charger le contenu d'une vue, le système utilise la fonction `LoadContentPage`. Cette fonction récupère le chemin du fichier HTML de la vue correspondante à partir de la route, puis utilise la fonction `fetch` pour obtenir le contenu HTML du fichier. Une fois le contenu HTML obtenu, il est injecté dans l'élément approprié du DOM de la page.

Si la route spécifie également un chemin vers un fichier JavaScript, ce fichier est chargé dynamiquement en tant que script dans le document, ce qui permet d'ajouter des fonctionnalités spécifiques à cette vue.

En plus de charger le contenu de la vue, le système met également à jour le titre de la page en utilisant le titre de la route et le nom du site web, afin de fournir une expérience de navigation cohérente.

Fichier Route.js :

```
export default class Route {
    constructor(url, title, pathHtml, pathJS = "") {
      this.url = url;
      this.title = title;
      this.pathHtml = pathHtml;
      this.pathJS = pathJS;
    }
}
```

Fichier allRoutes.js :

```
import Route from "./Route.js";

//Définir ici vos routes
export const allRoutes = [
    new Route("/", "Accueil", "/pages/home.html"),];

//Le titre s'affiche comme ceci : Route.titre - websitename
export const websiteName = "Quai Antique";
```

Fichier Router.js :

```
import Route from "./Route.js";
import { allRoutes, websiteName } from "./allRoutes.js";

// Création d'une route pour la page 404 (page introuvable)
const route404 = new Route("404", "Page introuvable", "/pages/404.html");

// Fonction pour récupérer la route correspondant à une URL donnée
const getRouteByUrl = (url) => {
  let currentRoute = null;
  // Parcours de toutes les routes pour trouver la correspondance
  allRoutes.forEach((element) => {
    if (element.url == url) {
      currentRoute = element;
    }
  });
  // Si aucune correspondance n'est trouvée, on retourne la route 404
  if (currentRoute != null) {
    return currentRoute;
  } else {
    return route404;
  }
};

// Fonction pour charger le contenu de la page
const LoadContentPage = async () => {
  const path = window.location.pathname;
  // Récupération de l'URL actuelle
```

```
  const actualRoute = getRouteByUrl(path);
  // Récupération du contenu HTML de la route
  const html = await fetch(actualRoute.pathHtml).then((data) => data.text());
  // Ajout du contenu HTML à l'élément avec l'ID "main-page"
  document.getElementById("main-page").innerHTML = html;

  // Ajout du contenu JavaScript
  if (actualRoute.pathJS != "") {
    // Création d'une balise script
    var scriptTag = document.createElement("script");
    scriptTag.setAttribute("type", "text/javascript");
    scriptTag.setAttribute("src", actualRoute.pathJS);

    // Ajout de la balise script au corps du document
    document.querySelector("body").appendChild(scriptTag);
  }

  // Changement du titre de la page
  document.title = actualRoute.title + " - " + websiteName;
};

// Fonction pour gérer les événements de routage (clic sur les liens)
const routeEvent = (event) => {
  event = event || window.event;
  event.preventDefault();
  // Mise à jour de l'URL dans l'historique du navigateur
  window.history.pushState({}, "", event.target.href);
  // Chargement du contenu de la nouvelle page
  LoadContentPage();
};

// Gestion de l'événement de retour en arrière dans l'historique du navigateur
window.onpopstate = LoadContentPage;
// Assignation de la fonction routeEvent à la propriété route de la fenêtre
window.route = routeEvent;
// Chargement du contenu de la page au chargement initial
LoadContentPage();
```

Complément **Pourquoi tout ça ?**

Grâce à ce système, nous avons un fichier **index.html** qui contient le header et le footer de la page. Nous chargeons le contenu de la page dans la balise `<main id="main-page">`. Ainsi, si nous devons un jour modifier le menu ou le footer, il faudra le modifier uniquement dans un seul fichier, pratique !

Attention

Dans ce cours, le but n'est pas de comprendre parfaitement cette partie de routage : nous devons nous concentrer sur le style de notre application. Les compétences principales requises pour ce cours sont en HTML et CSS, donc pas de panique !

Lorsqu'on fait du développement, c'est normal de devoir ignorer des morceaux de code que nous ne comprenons pas, cela fait partie de l'apprentissage. Et même après des années de développement, on ne comprend pas toujours tout. Le principal est de comprendre pourquoi on utilise un outil, de savoir comment l'utiliser, et avec ces informations nous pourrons avancer.

Méthode

C. L'ajout de Bootstrap

Le style d'un site

Pour gérer le style d'un site, nous devons utiliser le langage **CSS**. Le but est d'appliquer des styles à des éléments HTML pour les organiser, ou pour changer leurs identités visuelles. Pour nous aider, des frameworks CSS existent.

Définition **Un framework CSS**

Un framework CSS est un ensemble de styles prédéfinis, de composants et de règles de conception réutilisables qui facilitent le développement web en fournissant une base de styles et de fonctionnalités prêts à l'emploi.

On pourrait résumer en disant que ce sont des feuilles CSS codées pour nous, pour que nous n'ayons pas à tout reprendre à zéro à chaque étape. Elles aident aussi à cadrer dans la production de notre HTML et de notre CSS, pour respecter des normes standardisées.

Bootstrap

Bootstrap est l'un des frameworks CSS les plus populaires et convient parfaitement aux débutants dans le développement web. Il fournit une base solide pour la création de sites web réactifs et attrayants, avec une grande facilité d'utilisation.

Bootstrap offre un ensemble complet de styles prédéfinis et de composants réutilisables, tels que des grilles de mise en page, des formulaires, des boutons, des barres de navigation, des modals et bien plus encore. Ces composants peuvent être simplement intégrés à vos pages web en ajoutant des classes CSS appropriées à vos éléments HTML, ce qui vous permet de construire rapidement et facilement des interfaces cohérentes.

Méthode **Installer Bootstrap**

Vous pouvez installer Bootstrap de différentes manières. Elles sont décrites dans la documentation de Bootstrap (Bootstrap[1]).

1. Téléchargez les fichiers Bootstrap à partir du site officiel et intégrez-les manuellement dans votre projet en incluant les fichiers CSS et JavaScript.
2. Utilisez un gestionnaire de paquets, comme npm, pour installer Bootstrap à partir de la ligne de commandes dans votre projet.
3. Utilisez un CDN (*Content Delivery Network*) en ajoutant les liens des fichiers CSS et JavaScript de Bootstrap directement dans votre page HTML en utilisant les URL fournies par le CDN.

Nous allons utiliser npm pour notre projet. Pour télécharger et installer Node.js et npù, vous pouvez vous rendre sur ce site : node JS[2].

Méthode **L'installation de Boostrap sur un projet**

Nous allons ici décrire les étapes pas à pas pour installer Bootstrap sur un projet.

Avant de commencer, assurez-vous d'avoir Node.js installé sur votre système. Node.js inclut npm par défaut, ce qui facilitera l'installation de Bootstrap. Vous pouvez vérifier si Node.js est installé en ouvrant votre terminal (ou invite de commande) et en tapant la commande `node -v`. Si Node.js est installé, vous verrez la version installée affichée dans le terminal. Sinon, rendez-vous sur le site officiel de Node.js (node JS[3]) pour télécharger et installer la dernière version.

1 https://getbootstrap.com/docs/5.2/getting-started/introduction/
2 https://nodejs.org/en/download
3 https://nodejs.org/

Vous devez maintenant ouvrir un terminal de commande à la racine de votre projet. Vous pouvez installer Bootstrap en utilisant la commande `npm install bootstrap`. Cela téléchargera les fichiers de Bootstrap et leurs dépendances dans le dossier « *node_modules* ». Le dossier « *node_modules* » est un répertoire créé lors de l'installation de dépendances *via* npm, contenant les bibliothèques et les modules externes nécessaires pour le fonctionnement du projet.

Une fois l'installation terminée, vous pouvez inclure Bootstrap dans votre projet en important les fichiers CSS et JS. Il suffit d'inclure les fichiers JS et CSS de Bootstrap, qui sont désormais présents dans le dossier « *node_module* ».

Si votre fichier « *index.html* » est à la racine du projet, ce code devrait fonctionner :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" href="node_modules/bootstrap/dist/css/bootstrap.min.css">
    <title>Document</title>
</head>
<body>
<button class="btn btn-primary">Cliquez ici</button>
    <script src="node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Si le bouton est bleu avec la mise en forme de Bootstrap, vous avez réussi à inclure Bootstrap ! Bravo !

Complément

Pas facile de choisir la méthode pour intégrer Bootstrap. Pour vous aider, voici quelques points positifs et négatifs de chaque méthode. En fonction de votre projet, à vous de choisir avec ces arguments en tête :

- Installation manuelle en téléchargeant les fichiers Bootstrap :
  - Points forts : contrôle total sur les fichiers utilisés, possibilité de personnaliser les styles et de les intégrer facilement dans votre projet.
  - Points faibles : requiert une gestion manuelle des mises à jour et des dépendances.
- Utilisation d'un gestionnaire de paquets (comme npm) :
  - Points forts : gestion des dépendances simplifiée, facilité des mises à jour grâce à la gestion automatisée des packages.
  - Points faibles : peut nécessiter une configuration initiale plus complexe, surtout si vous n'utilisez pas déjà un gestionnaire de paquets.
- Utilisation d'un CDN (Content Delivery Network) :
  - Points forts : facilité d'utilisation, aucun téléchargement ou intégration de fichiers requis, mise en cache optimisée pour une meilleure performance.
  - Points faibles : dépendance externe, nécessite une connexion internet pour charger les fichiers Bootstrap, moins de contrôle sur les versions utilisées et les mises à jour.

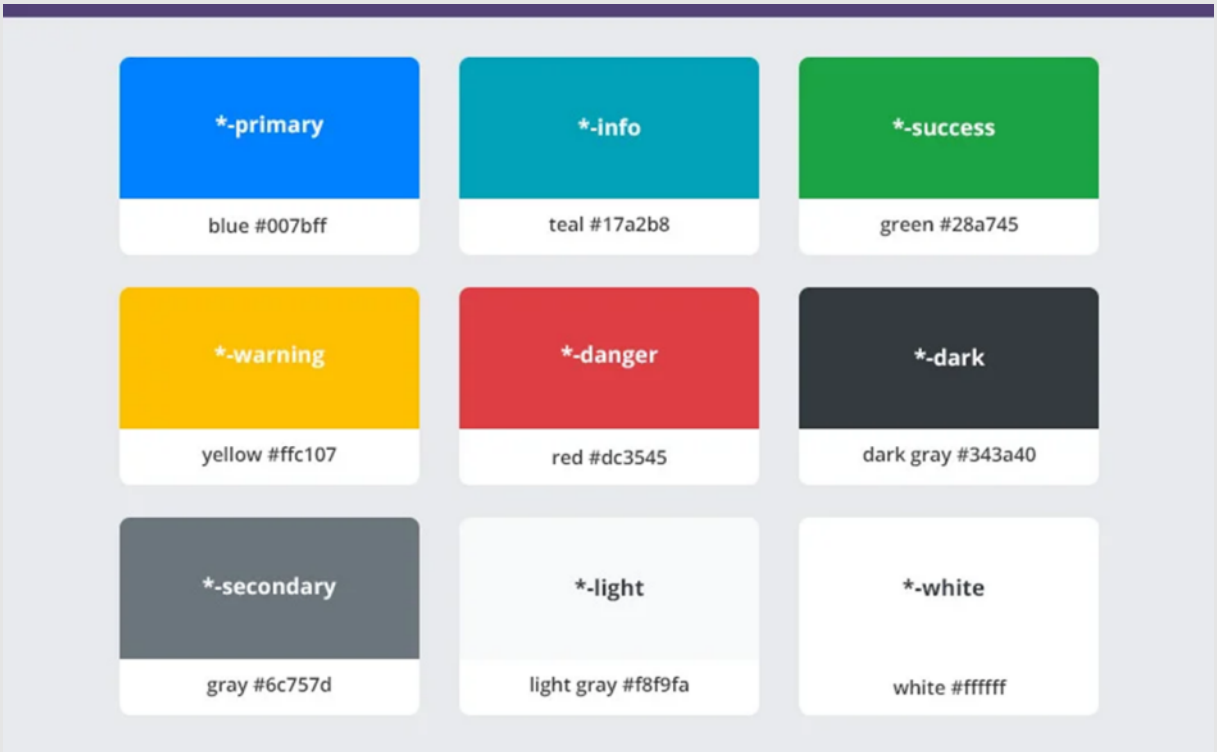
D. La customisation de Bootstrap avec Sass

Méthode **Customiser Bootstrap**

Par défaut, Bootstrap embarque un certain style et donc une certaine charte graphique. Par exemple, Bootstrap définit certaines couleurs. Mais grâce à Sass, nous avons la possibilité de modifier ces couleurs par défaut.

Qu'est-ce que Sass ? C'est un préprocesseur pour CSS. C'est-à-dire que ce langage offrant bon nombre de nouvelles fonctionnalités sera compilé et interprété par la suite en CSS. Il permet d'utiliser des règles imbriquées, des *mixins* (ensemble de règles CSS réutilisables), ainsi que des variables. Bootstrap étant codé en Sass, il est prévu que nous puissions surcharger ces variables, afin d'écraser ces couleurs par défaut et de les remplacer par nos couleurs personnalisées.

Nous pouvons personnaliser d'autres éléments sur Bootstrap, voici le lien pour comprendre comment cela fonctionne : Bootstrap[1].



Bootstrap colors

Source : BitDegree[2]

Méthode **Changer les couleurs de Bootstrap**

Pour personnaliser les couleurs de Bootstrap avec Sass, il est recommandé de créer un fichier « *_custom.scss* ».

Vous devez avoir un fichier « *main.scss* », qui sera le fichier principal de votre site. Ce fichier main importera ce fichier custom, qui lui importe Bootstrap. Nous pouvons aussi choisir d'installer et d'importer Bootstrap icons pour pouvoir afficher de jolies icônes.

```
1 @import url("/node_modules/bootstrap-icons/font/bootstrap-icons.css");
2 @import 'custom';
```

1 https://getbootstrap.com/docs/5.3/customize/overview/#overview
2 https://www.bitdegree.org/learn/storage/media/images/7383b588-563f-4117-82bd-4c866b5490bd.jpg

Votre fichier « *_custom.scss incluera Bootstrap* », après avoir surchargé les variables par défaut :

```
$primary: #906427;
$secondary: #B6AC97;
$dark:#392C1E;
$black:  #292222;
$font-family-sans-serif: "Montserrat", sans-serif !default;

@import "../node_modules/bootstrap/scss/bootstrap";
```

Ainsi, la compilation de votre fichier « *main.scss* » sera un fichier nommé « *main.css* ». Il contiendra tout Bootstrap, Bootstrap icons et votre surcharge de Bootstrap.

III. Le codage du visuel

A. Le header

Coder le header

Notre environnement de travail étant prêt, nous pouvons commencer à coder notre application. Nous pouvons commencer par le header de l'application qui contient le menu de navigation.

En version mobile, c'est un menu « *burger* », qui affiche ses liens au clic. En mode desktop, c'est une liste horizontale de liens.

Pour créer cette barre de navigation, nous pouvons utiliser le composant navbar de Bootstrap.

Méthode **Le composant navbar de Bootstrap**

Vous pouvez voir des exemples de navbar directement sur la documentation de Bootstrap (Bootstrap[1]). En HTML, une navbar Bootstrap basique ressemble à ça :

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Mon Site</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Accueil</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Fonctionnalités</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Prix</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

Comment fonctionne-t-elle ?

1 https://getbootstrap.com/docs/5.3/components/navbar

- `navbar` : déclare le composant de la barre de navigation.
- `navbar-expand-lg` : la barre de navigation s'étendra (mode horizontal) à partir du point de rupture « *lg* » (large).
- `navbar-light` et `bg-light` : définissent la couleur de la barre de navigation.
- `navbar-brand` : représente généralement le logo ou le nom du site.
- `navbar-toggler` : bouton utilisé pour afficher/masquer le contenu de la navbar en mode responsive (écrans plus petits).
- `collapse navbar-collapse` : conteneur pour les éléments qui seront réduits dans les vues mobiles.
- `navbar-nav` : classe de base pour la navigation.
- `nav-item` et `nav-link` : définissent les éléments et les liens de la barre de navigation.

Méthode **Notre header**

En appliquant les styles demandés par la maquette, nous obtenons ce header :

```
  <header>
       <nav class="navbar navbar-expand-lg bg-dark" data-bs-theme="dark">
           <div class="container-fluid">
             <a class="navbar-brand" href="/">Quai Antique</a>
             <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
               <span class="navbar-toggler-icon"></span>
             </button>
             <div class="collapse navbar-collapse" id="navbarSupportedContent">
               <ul class="navbar-nav ms-auto mb-2 mb-lg-0">
                 <li class="nav-item">
                   <a class="nav-link" href="/">Accueil</a>
                 </li>
                 <li class="nav-item">
                   <a class="nav-link" href="/galerie">Galerie</a>
                 </li>
                 <li class="nav-item">
                   <a class="nav-link" href="#">La carte</a>
                 </li>
                 <li class="nav-item">
                   <a class="nav-link" href="#">Les réservations</a>
                 </li>
                 <li class="nav-item">
                   <a class="nav-link" href="#">Mon compte</a>
                 </li>
                 <li class="nav-item">
                   <a class="nav-link" href="#">Connexion/Deconnexion</a>
                 </li>
               </ul>
             </div>
           </div>
         </nav>
    </header>
```

B. Le footer

Réaliser le footer

Réaliser le footer sera plus simple que le header, nous n'allons pas utiliser de composant Bootstrap. En soit, ce sont juste 3 blocs qui s'affichent différemment en fonction de la taille de notre écran. Sur mobile, nous avons le bloc « *nos horaires* » qui prend toute la largeur, et les 2 blocs « *adresse* » et « *mail* » qui prennent la moitié de l'écran. Sur desktop, ces 3 blocs prennent respectivement un tiers de l'écran. Bootstrap Grid nous permet de gérer ce genre de problématique très facilement.

Méthode **Bootstrap Grid**

Dans le système de grille de Bootstrap, une ligne est divisée en 12 colonnes. Donc, pour définir une colonne qui prend toute la largeur de l'écran, vous utilisez la classe `col-12`.

Voici un exemple :

```
<div class="row">
  <div class="col-12">
    Ceci est une colonne qui prend toute la largeur de l'écran.
  </div>
</div>
```

Si vous voulez qu'une colonne occupe la moitié de l'écran, vous lui attribuez 6 des 12 colonnes disponibles. Pour ce faire, vous utilisez la classe `col-6`.

Voici comment cela se fait :

```
<div class="row">
  <div class="col-6">
    Ceci est une colonne qui prend la moitié de l'écran.
  </div>
</div>
```

Dans ce cas, la colonne occupera la moitié de l'espace disponible, peu importe la taille de l'écran, car aucune classe de point de rupture spécifique (comme `-sm`, `-md`, `-lg`, `-xl`, `-xxl`) n'a été utilisée. Si vous voulez que la largeur de la colonne change en fonction de la taille de l'écran, vous pouvez alors utiliser ces classes de points de rupture.

Méthode **Le point de rupture (breakpoint)**

Par exemple, si vous voulez qu'une colonne occupe toute la largeur de l'écran (12 colonnes) sur les petits appareils, mais seulement la moitié de l'écran (6 colonnes) sur les appareils de taille moyenne et plus grande, vous pouvez le faire comme suit :

```
<div class="row">
  <div class="col-12 col-md-6">
    Sur les petits écrans, je prends toute la largeur. Sur les écrans de taille moyenne et
plus, je prends la moitié de la largeur.
  </div>
</div>
```

Dans cet exemple, la classe `col-12` signifie que la colonne occupera toute la largeur (12 sur 12 colonnes) sur tous les appareils avant le premier point de rupture spécifié (ici `md`). La classe `col-md-6` signifie que, dès que vous atteignez une taille d'écran moyenne (largeur d'écran ≥ 768 px), la colonne n'occupera que la moitié de l'écran (6 sur 12 colonnes).

Notre footer

Voici le footer que nous obtenons :

```
<footer class="bg-dark text-white text-center footer">
     <div class="row">
         <div class="col-12 col-lg-4">
             <h3 class="text-primary">Nos horaires</h3>
             <p>Du mardi au dimanche  12:00-14:00  18:00-23:00</p>
         </div>
         <div class="col-6 col-lg-4">
             <p>Quai Antique <br/>
                 5 rue de la route <br/>
                 31150 Ville <br/>
                 04 01 01 01 01 <br/>
             </p>
         </div>
         <div class="col-6 col-lg-4">
             <p>test</p>
         </div>
     </div>
 </footer>
```

Méthode

Complément **Offset**

Lorsque vous utilisez l'offset, vous pouvez créer des espaces vides (marges) à gauche des colonnes pour obtenir la mise en page désirée. Cela peut être utile lorsque vous avez besoin de créer des mises en page asymétriques où certaines colonnes sont placées différemment par rapport à la structure de la grille standard.

L'offset est utilisé en ajoutant des classes de la forme suivante : `offset-{taille}-X`, où :

- `{taille}` est la taille de l'écran pour laquelle l'offset sera appliqué (par exemple xs, sm, md, lg, ou xl). Ces classes permettent de spécifier à quelles tailles d'écran l'offset doit être appliqué.
- `X` est le nombre de colonnes que vous souhaitez décaler. Par exemple, `offset-md-3` décale une colonne de 3 colonnes sur les écrans de taille moyenne (md).

Exemple

```
<div class="container">
  <div class="row">
    <div class="col-md-6 offset-md-3">
      <!-- Contenu ici -->
    </div>
  </div>
</div>
```

Dans cet exemple, `col-md-6` signifie que la colonne occupe 6 colonnes de large sur les écrans de taille moyenne (md), et `offset-md-3` signifie que la colonne est décalée de 3 colonnes vers la droite sur les écrans de taille moyenne (md). Cela crée une marge à gauche de 3 colonnes, ce qui centre le contenu dans la grille.

C. La page d'accueil

Méthode **Réaliser le bloc bigtitle**

Pour commencer, notre page d'accueil, nous pouvons réaliser le bloc bigtitle. Voici comment le réaliser :

```html
<div class="hero-scene text-center text-white">
    <div class="hero-scene-content">
        <p>Le chef Arnaud Michant vous invite au</p>
        <h1>Quai Antique</h1>
        <button class="btn btn-primary">Réserver</button>
    </div>
</div>
```

```scss
.hero-scene{
    position: relative;
    &::before{
        content:"";
        position: absolute;
        top:0;
        left:0;
        width:100%;
        height:100%;
        background-image: url(../images/FondHeroScene.jpg);
        background-size: cover;
        filter: brightness(0.3);
    }

    *{
        //Positionner les éléments devant le ::before
        position:relative;
    }

    .hero-scene-content{
        height: 300px;
        display: flex;
        flex-direction: column;
        align-items: center;
        justify-content: center;
    }
}
```

Méthode **Réaliser le contenu de la page Home**

Le contenu de notre page Home est constitué de 3 blocs, eux-mêmes composés de 3 lignes. La première étant le titre du bloc, dans la deuxième, nous avons 2 blocs prenant respectivement la moitié de la largeur de la ligne, et enfin la dernière ligne contient un bouton. Nous pouvons donc construire notre structure html ainsi :

```html
 <article>
      <div class="container p-4">
          <h2 class="text-center text-primary">Bienvenue au quai Antique</h2>
          <div class="row row-cols-2 align-items-center">
              <div class="col">
                  <p class="text-justify">
                      Bienvenue sur le site de notre restaurant gastronomique situé au cœur de la magnifique ville de Chambéry. Le chef Arnaut Michant vous invite à découvrir un voyage culinaire mémorable.
                  </p>
              </div>
              <div class="col">
```

```
                    <img class="w-100 rounded" src="../images/chef.jpg"/>
                </div>
            </div>
            <div class="text-center pt-4">
                <a href="#" class="btn btn-primary">Nos menus</a>
            </div>
        </div>
    </article>
```

Le deuxième bloc (« *Nos produits* ») inverse le texte et l'image. Il nous suffit donc d'inverser l'image et le texte dans le code pour les afficher dans le bon ordre. Pour changer la couleur de fond, nous pouvons ajouter la classe `bg-dark` et `text-white`, pour colorer le fond en gris foncé, et avoir le texte en blanc. Nous obtenons donc ceci pour le deuxième bloc :

```
<article class="bg-black text-white">
        <div class="container p-4">
            <h2 class="text-center text-primary">Des produits frais</h2>
            <div class="row row-cols-2 align-items-center">
                <div class="col">
                    <img class="w-100 rounded" src="../images/saumon.jpg"/>
                </div>
                <div class="col">
                    <p class="text-justify">
                        Découvrez la richesse des saveurs avec nos plats élaborés à partir de produits frais et de saison, cueillis localement.
                    </p>
                </div>
            </div>
            <div class="text-center pt-4">
                <a href="/galerie" class="btn btn-primary">Voir la galerie</a>
            </div>
        </div>
    </article>
```

Il nous suffit pour le dernier bloc de reprendre le premier, en modifiant les textes et l'image.

D. La page galerie

Méthode **Le bloc bigtitle**

Pour commencer notre page galerie, nous pouvons reprendre le bloc bigtitle fait pour la page Home, en enlevant les éléments non nécessaires. Nous avons ce résultat :

```
<div class="hero-scene text-center text-white">
    <div class="hero-scene-content">
        <h1>Galerie</h1>
    </div>
</div>
```

Méthode **La grille d'images**

La grille d'images est assez simple : en mobile, nous avons 2 images par ligne, et en desktop, nous avons 3 images par ligne. Nous allons utiliser la classe `row-cols-n`.

La classe `row-cols-n` de Bootstrap est utilisée pour définir combien de colonnes (où n est le nombre de colonnes) sont placées dans une ligne. Pour définir 2 éléments par ligne en mobile et 3 en desktop, vous pouvez utiliser la classe `row-cols-2` pour les petits appareils et `row-cols-lg-3` pour les appareils de taille moyenne (`lg`) et plus grands.

Voici le rendu que nous aurions :

```
<div class="row row-cols-2 row-cols-lg-3">
        <div class="col p-3">
                <img src="../images/food.jpg" class="rounded w-100"/>
        </div>
<!– Ajouter d’autres images –>
</div>
```

Dans cet exemple, sur les petits écrans, chaque ligne aura 2 éléments (grâce à `row-cols-2`). Ensuite, lorsque la taille de l'écran atteint le point de rupture « *lg* » (≥ 992 px), chaque ligne contiendra 3 éléments (grâce à `row-cols-lg-3`).

Méthode **Afficher le titre au survol**

Maintenant que nos images sont affichées, il nous faut afficher le titre au survol de l’image. Pour pouvoir travailler, nous pouvons modifier la structure HTML de notre image.

Voici le rendu que nous aurions :

```
<div class="col p-3">
            <div class="image-card text-white">
                <img src="../images/food.jpg" class="rounded w-100"/>
                <p class="titre-image">Titre</p>
            </div>
        </div>
```

L’objectif est maintenant d’afficher le titre au milieu de l’image, nous pouvons appliquer ce style pour cela :

```
.image-card{
    position:relative;

    .titre-image{
        position: absolute;
        top:50%;
        left:50%;
        transform: translate(-50%, -50%);
    }
}
```

Maintenant, il nous faut masquer le titre par défaut, nous pouvons lui mettre l’opacité à 0 par défaut. Lorsque nous survolons le bloc « *image-card* », nous, devons changer l’opacité du titre à 1, et ajouter un filtre sur l’image pour la foncer. Nous pouvons aussi ajouter une transition pour que l’animation soit fluide. Nous obtenons donc ce code final pour nos images :

```
.image-card{
    position:relative;
    &:hover{
        .titre-image{
            opacity: 1;
        }
        img{
            filter: brightness(0.3);
        }
    }

    img{
        transition: all 0.3s;
    }

    .titre-image{
```

```
        position: absolute;
        top:50%;
        left:50%;
        transform: translate(-50%, -50%);
        opacity: 0;
        transition: all 0.3s;
    }
}
```

IV. L’essentiel

Dans ce cours, nous avons vu comment réaliser la partie front responsive d'un projet. Nous avons commencé par créer le projet dans Visual Studio Code et par l'envoyer sur Git. Ensuite, nous avons abordé le concept de framework JavaScript et son utilité pour guider le développement front-end. Nous avons exploré le routage en développement front-end et comment il permet de gérer la navigation entre les pages. Nous avons réalisé notre projet sans framework JavaScript, pour voir comment réaliser un mini framework js à la main.

Pour faciliter le style de notre application, nous avons utilisé Bootstrap, un framework CSS populaire, et appris comment l'installer et le personnaliser. Nous avons réalisé le header et le footer de notre application en utilisant les composants de Bootstrap. Enfin, nous avons créé la page d'accueil et la page galerie en utilisant les grilles Bootstrap pour organiser les éléments.

V. Pour aller plus loin

A. Pour aller plus loin

- Vous pouvez réaliser la page « *la carte* » du projet.
- Vous pouvez chercher à ajouter des animations à l’arrivée des blocs, le chargement de la page n’est pas très fluide ; en CSS et JS, vous pouvez rendre l’arrivée de la page plus fluide.