DOSSIER DE PROJET

expeditionbrewery.com

Projet : API RESTful, application web pour micro-brasserie	Début du projet 28/11/2022
Nicholas GILLESPIE *École O'Clock*	Fin du projet 26/12/2022

SOMMAIRE

INTRODUCTION **4**

COMPÉTENCES DU RÉFÉRENTIEL **5**
- Activité type 1 5
- Activité type 2 6

RÉSUMÉ DU PROJET **7**
- Introduction 7
- Objectif du projet 7

CAHIER DES CHARGES **8**
- Expression des besoins 8
- Objectifs 9
- MVP 1.0 9
- Version 2.0 envisagée 10
- Utilisateurs 10
- User stories 10
- Routes 11
- API endpoints 11
- Wireframes 12

SPÉCIFICATIONS TECHNIQUES **15**
- Technologies et méthodologies 15
- Versioning 17
- Architecture du Projet 18
- Circulation de la données 19
- MVC 19
- API RESTful 22
- Modèles de données 23
- Sécurité de l'application 25

GESTION DE PROJET **27**
- Présentation de l'équipe et rôles 27
- Organisation de travail et sprint 27
- Outils utilisés 28

RÉALISATION DU CANDIDAT **29**
- Création maquettes 29
- Mise en place de l'architecture du projet 29
- Classes utilitaires CSS - Définition des design tokens 30
- Création BDD & connection application 33
- Création de l'API 34
- Création du gestionnaire d'erreur global 36
- Définition des erreurs à travers l'application 39

- Authentification et autorisation 40
- Interaction front / back-end (AJAX) 44

PRÉSENTATION DU JEU D'ESSAI 46

DESCRIPTION DE LA VEILLE 50

DESCRIPTION D'UNE SITUATION DE TRAVAIL 51

EXTRAIT DE SITE ANGLOPHONE 53

CONCLUSION 55

INTRODUCTION

Après cinq ans en tant que Commando Marines dans les forces britanniques, spécialisé dans la gestion des données, j'ai travaillé dans le secteur de la communication, en lien avec l'industrie du sport.

En 2020 avec l'arrivée du Covid, ainsi que des raisons personnelles, j'ai décidé de poursuivre mon intérêt pour le secteur de la communication et plus particulièrement le développement de sites internet.

Après m'être intéressé à cette spécialité à titre personnel pendant plusieurs années, j'ai décidé de suivre une formation Développeur Web & Web Mobile PHP. Cela a entraîné en moi une soif de connaissances JavaScript.

Dans le cadre de la fin de ma formation chez O'Clock, et en vue du Titre Professionnel DWWM, j'ai dû réaliser au sein d'une équipe un projet dit d' "Apothéose". L'objectif étant de développer un site web de A à Z, en un mois.

Ce dossier rassemble diverses notions relatives au projet et des détails de mes réalisations personnelles.

COMPÉTENCES DU RÉFÉRENTIEL

Décrit plus en détail dans les chapitres suivants de ce document, ce projet contient un front et back-end. Il utilise une combinaison de technologies HTML, CSS, JavaScript, Node.js et MongoDB.

Activité type 1 Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

---

1. Maquetter une application

   Les wireframes sont le résultat d'une rencontre d'un peu plus d'une heure que j'ai eue avec les deux entrepreneurs pour qui ce projet a été développé. Pendant ce temps, ils m'ont présenté différents sites et éléments qui leur plaisaient. Recueillant leurs préférences, j'ai réalisé des wireframes pour les trois pages principales du site.

2. Réaliser une interface utilisateur web statique et adaptable

   L'intégration de ce site Web s'est faite en utilisant la combinaison de la méthodologie CUBE CSS, la fonction 'clamp' de CSS et les unités de mesures viewport. Cela a permis de créer des pages algorithmiquement réactives, s'adaptant en synchronisme avec la largeur d'écran de l'utilisateur. L'utilisation de media queries a également été faite.

3. Développer une interface utilisateur web dynamique

   Qu'il s'agisse de rendre dynamique un menu coulissant sur des écrans de petites tailles ou pour faire des appels AJAX communiquant des données entre le front-end et la base de données, JavaScript a été utilisé.

5. Créer une base de données

   Bien que seulement dans la première année de leur aventure entrepreneuriale, cette jeune brasserie a déjà brassé six bières à ce jour.
   L'objectif de ce site consiste à répertorier et présenter leurs produits. Pour cela, nous avons utilisé une base de données. Compte tenu de la jeunesse de l'entreprise et la nécessité de flexibilité, entre autres, il a été décidé d'utiliser la base de données MongoDB.

6. Développer les composants d'accès aux données

   L'inclusion des données, leur consultation et manipulation a été effectuée utilisant la bibliothèque de modélisation de données d'objet (ODM) Mongoose, permettant un développement rapide et simple des interactions avec la base de données MongoDB.

7. Développer la partie back-end d'une application web ou web mobile

   Veillant à sécuriser leur site internet et son accès, la mise en place d'un système d'accès ainsi que la sécurisation des différentes routes a été implémenté. Pour cela, une combinaison de cryptage de mot de passe et de système de signature cryptée a été effectuée. En combinaison avec la recommandation de sécurité de l'OWASP, diverses mesures ont été prises afin de minimiser les risques sécuritaires.

RÉSUMÉ DU PROJET

Introduction

Expedition Brewery est une start-up qui s'est lancée, fin 2022, dans le commerce de vente de bières aux particuliers et commerces aux alentours des Yvelines.

D'abord influencés par la scène craft anglo-saxonne, ils se distinguent de leurs concurrents par un processus de brassage entièrement fait à la main, par des ingrédients sélectionnés par leurs plus grands soins.

Afin de se faire connaître, ils utilisent toutes les techniques de communications (bouche à oreille, flyers, marchés de noël ci-contre).

Aujourd'hui ils tiennent un site Wordpress qui manque de professionnalisme et n'est pas dès plus concluant.

Le principal désir des gérants d'Expedition Brewery est d'avoir un site à leur image et de permettre aux visiteurs de découvrir leur activité, les méthodes de brassage des bières qu'ils emploient, ainsi que consulter les différents produits.

Objectif du projet

Développer un site web permettant de présenter le catalogue de bière pour une micro-brasserie avec parti catalogue administrable. Fonctionnalités: ajout, mise à jour, suppression avec authentification admin.

CAHIER DES CHARGES

Expression des besoins

- Les clients souhaitent un site leur permettant de présenter leur produits et d'en informer les visiteurs qui ils sont, ce qu'ils font et quels sont leurs produits.

- Chacune de leurs bières est brassée avec des ingrédients différents, produisant des goûts très variés. L'un de leurs souhaits est de rendre cela apparent.

- Ils souhaitent pouvoir créer de nouveaux produits à même leur site, les mettre à jour puis pouvoir les supprimer si nécessaire.

- Ils souhaitent permettre à leurs clients un moyen de les contacter directement via le site Web.

- Ils souhaitent à même la page regroupant toutes leurs bières un système de filtrage, permettant aux clients de trier les bières par critères préférentiel.

- Ils ont deux types de clients. Leurs clients principaux, étant les pubs, bars et restaurants; puis l'individu lambda. Ils souhaitent pouvoir partager avec leurs clients principaux les détails de leurs produits pour leur permettre une utilisation et un affichage potentiel de données à même leurs sites Web.

- Ils souhaitent afficher à même leur site une carte affichant les différents endroits où les clients lambda peuvent trouver leurs produits.

- Ils aimeraient avoir une section événements, reflétant des collaborations qu'ils auraient avec des lieux commerciaux ou individus, tels des artistes.

- Ils souhaitent, quand ils en auront la capacité, vendre leurs bières directement à même leur site.

Objectifs

Les objectifs principaux sont:

- Avoir un site Web fonctionnel divisé en quatre sections principales :
  à propos de nous, nos bières, page de bière individuelle, contactez-nous

- La page regroupant toutes les bières, affichant toutes leurs bières.

- La page bière individuelle, affichant les caractéristiques qui rendent la bière unique.

- La page contact, permettant aux visiteurs du site de les contacter.

- Permettre aux visiteurs d'accéder aux mentions légales.

- Une interface administrateur contrôlée, permettant la création, mise à jour et suppression des bières présentés à même le site.

MVP 1.0

Étant donné le délai de réalisation d'un mois, nous avons opté pour une réalisation de l'application répartie sur plusieurs versions. De plus les difficultés techniques et humaines rencontrées nous ont induit à évoluer en nous basant sur la méthode du MVP (Minimum Viable Product).

La version 1.0 est donc un MVP avec les fonctionnalités suivantes :

- Développement d'une API regroupant les données concernant leurs bières.

- Création des quatre sections principales sous forme de pages:
  à propos de nous, nos bières, page de bière individuelle, contactez-nous.

- Interface administrateur contrôlée permettant la création, mise à jour et suppression des bières présentés à même le site.
  Formulaire de connexion avec authentification.

- Formulaire de contact.

Version 2.0 envisagée

- Inclure dans la page des bières une option de filtrage et de tri.
  Permettant au visiteur de filtrer en utilisant des critères préférés tels que le style de bière, le pourcentage d'abv, trier par ordre alphabétique, etc.

- Inclure une carte indiquant les endroits où les clients peuvent trouver leurs produits.

Utilisateurs

Types d'usagers:

L'application est destinée aux personnes intéressées par la bière artisanale.
Dans ce projet, nous pouvons définir 2 types d'usagers :

- visiteur
- administrateur

User stories

En tant que	Je souhaite pouvoir	Afin de
Visiteur	Accéder/visualiser la landing page	Comprendre le sens/but du site
	Découvrir les méthodes employées par l'entreprise	Décider si cela correspond à mes préférences, attentes
	Visualiser les différents produits proposés par l'entreprise	Voir qu'elle produit pourrait m'intéresser
	Voir les spécificités de chaque produit	Afin de distinguer les différences et de voir laquelle peut répondre à mes préférences
	Pouvoir communiquer avec l'entreprise en cas de question	Pour recevoir une réponse clarifiant les questions
Administrateur	Créer un nouveau produit	L'afficher sur le site Web afin que les visiteurs puissent le voir
	Mettre à jour un produit	Mettre à jour toute information pertinente si nécessaire
	Supprimer un produit	Le supprimer du site Web

Routes

Page	Route
home	/
beers	/beers
beer	/beers/:slug
contact	/contact
mentions légales	/legal-notice
login	/login
admin	/admin
admin create beer	/admin/create
admin update beer	/admin/:slug
admin delete beer	/admin

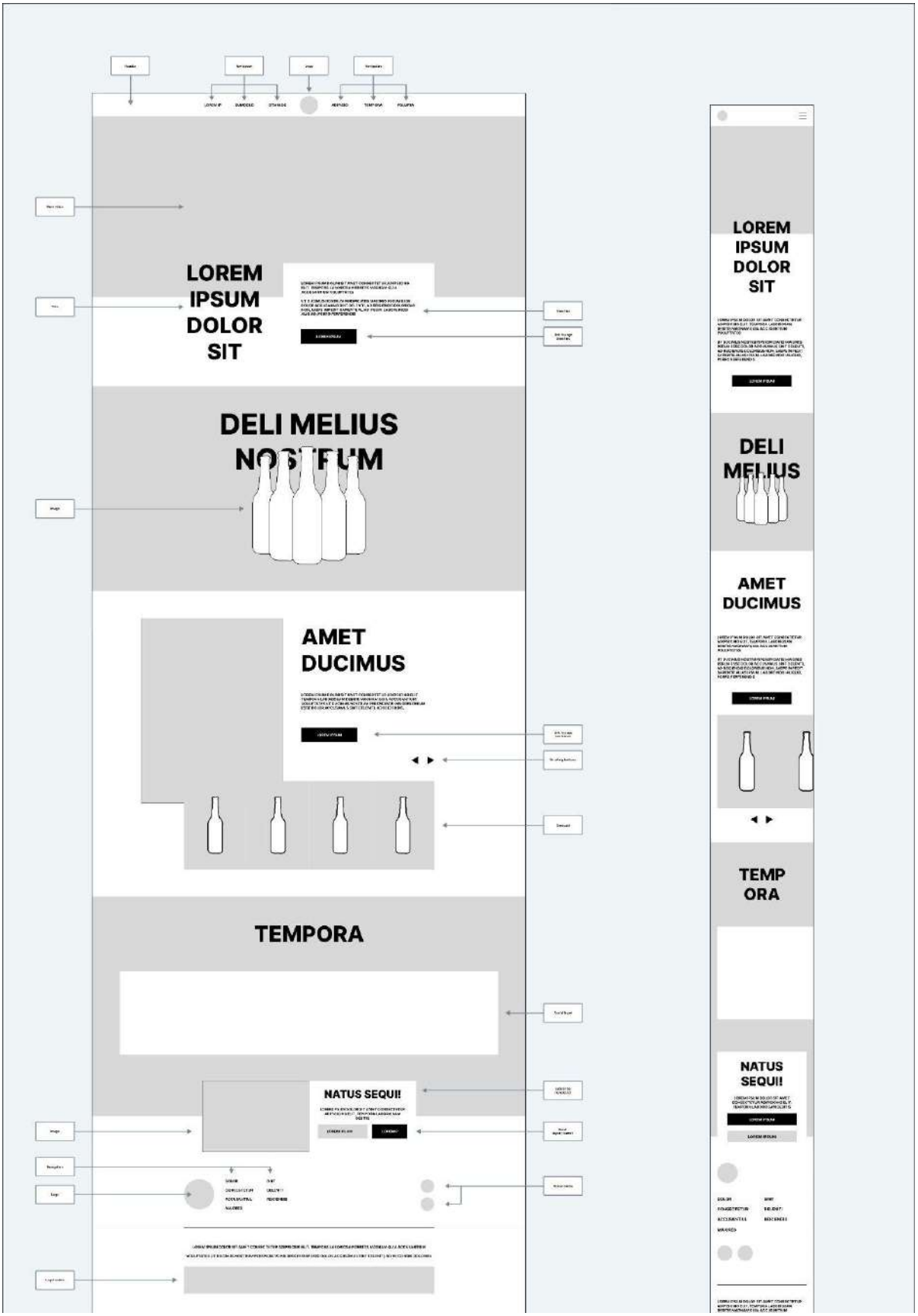
API endpoints

Resource	URL	Method	Action
beer	/api/v1/beers/	POST	Create beer
	/api/v1/beers/	GET	Get all beers
	/api/v1/beers/:slug	GET	Get beer
	/api/v1/beers/:slug	PATCH	Update beer
	/api/v1/beers/:slug	DELETE	Delete beer
user	/api/v1/users/login	POST	User login

Wireframes

Page accueil - desktop

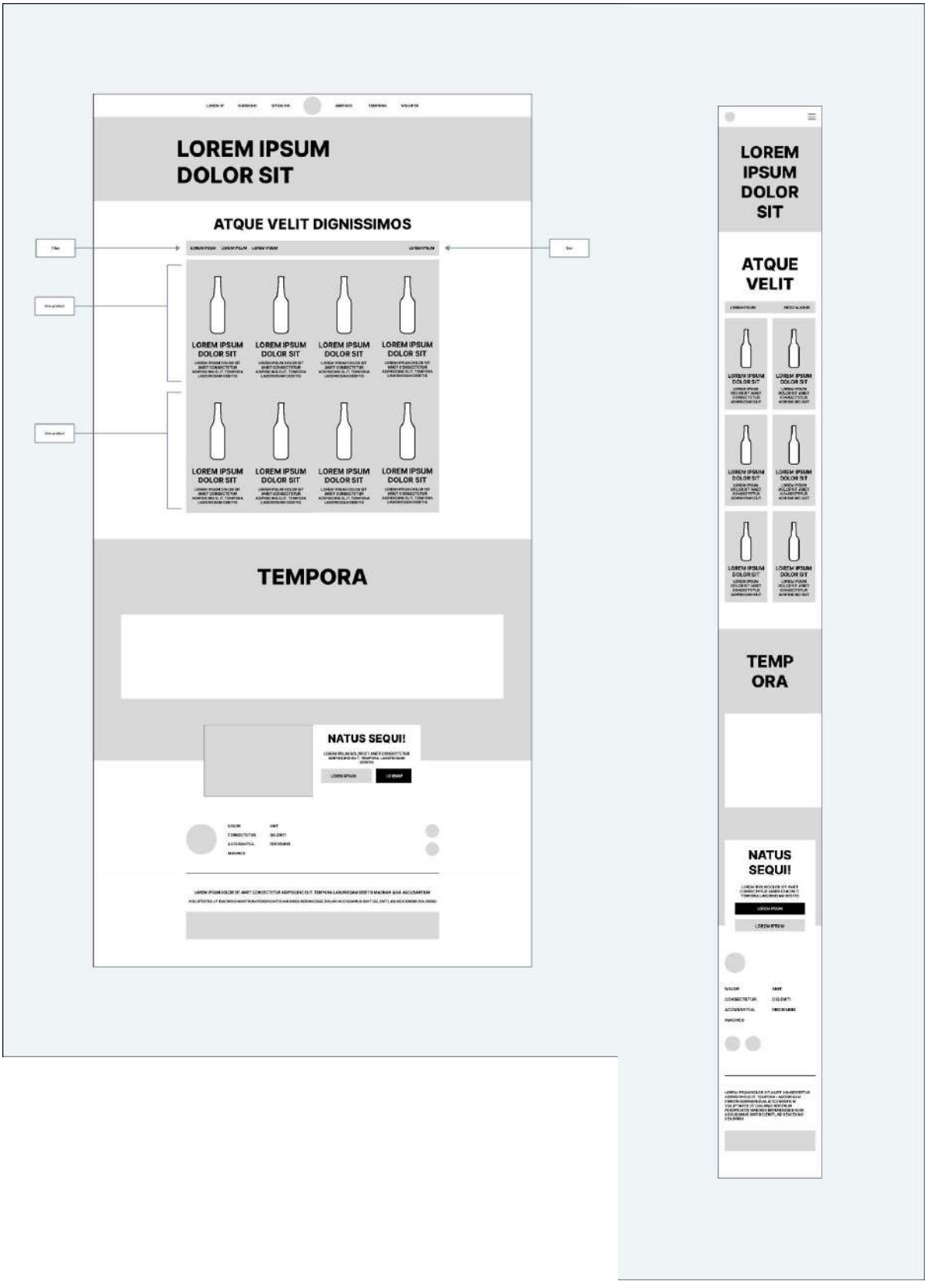
Page accueil - Mobile



Wireframes

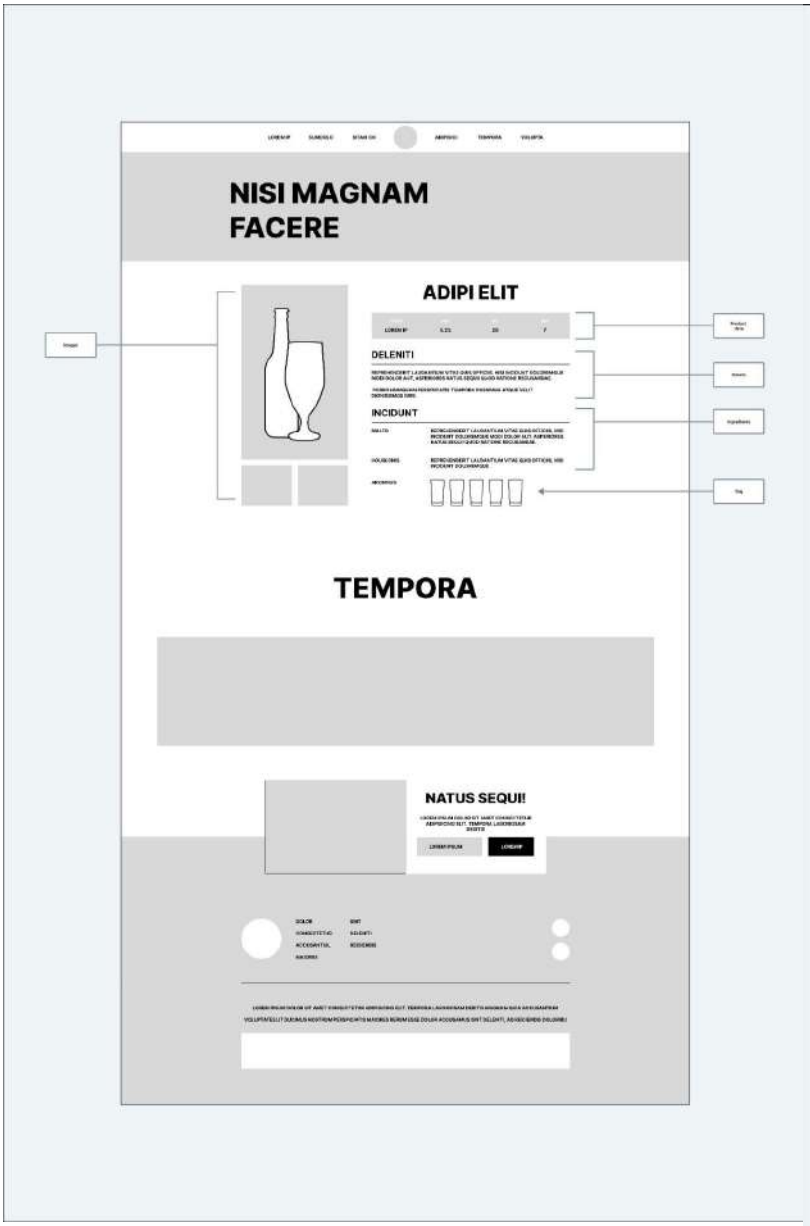
Page bières - desktop

Page bières - Mobile

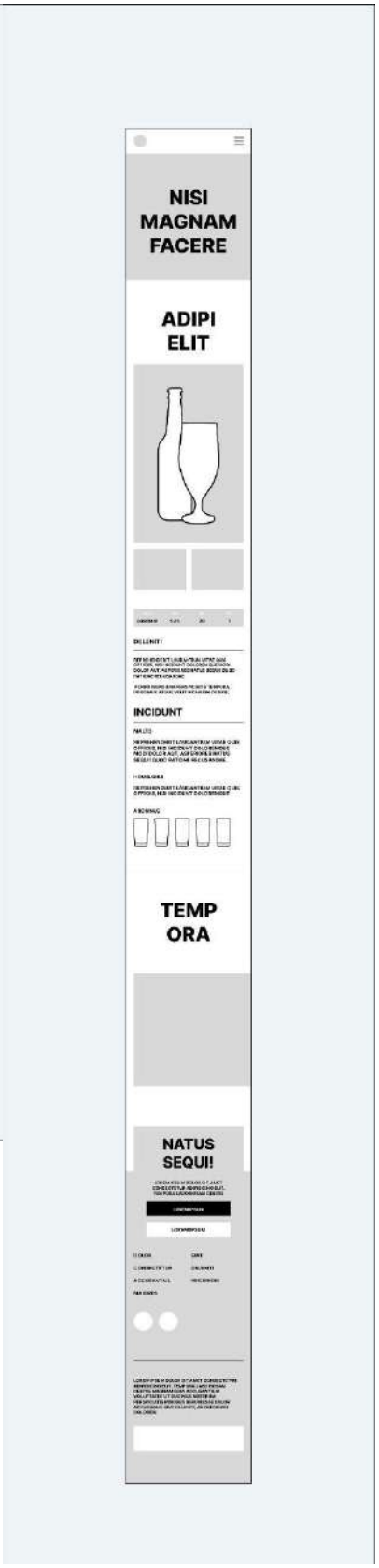


Wireframes

Page bière individuel - desktop



Page bière individuel - Mobile



SPÉCIFICATIONS TECHNIQUES

Technologies et méthodologies

Les technologies et méthodologies utilisées dans le cadre de ce projet sont listées ci-dessous.

Front-end	Technologies

HTML5, une version plus verbeuse que la précédente, permet d'optimiser le référencement des sites, et comporte plus de fonctionnalités.

CSS3 signifie Cascading Style Sheet level 3. Il permet de structurer, de styliser et de formater des pages Web.

JavaScript est un langage de script qui permet de créer du contenu mis à jour dynamiquement, de contrôler le multimédia, d'animer des images et à peu près tout le reste.

Front-end	Méthodologies

CUBE CSS est une méthodologie axée sur la simplicité, le pragmatisme et la cohérence. CUBE signifie «Composition Utility Block Exception» et est une méthodologie indépendante de l'outil. Il suit une approche descendante des éléments de style et passe des styles globaux aux styles de bas niveau et spécifiques.

Le système de conception web.dev aide à maintenir la cohérence en fournissant une structure CSS, des jetons de conception, des règles de typographie et des règles d'espacement, ainsi qu'une bibliothèque complète de composants.

Back-end Technologies

Node.js est un environnement d'exécution JavaScript basé sur le moteur JavaScript V8 open source de Google. Node.js permet d'utiliser JavaScript côté serveur du développement Web afin de créer des applications réseau rapides et hautement évolutives pour alimenter le back-end des sites Web ou des applications Web.

Express est un framework Node.js minimal qui contient un ensemble de fonctionnalités robustes et très utiles telles que le routage complexe, la gestion simplifiée des requêtes et des réponses, l'ajout de middleware, le rendu côté serveur, etc.

MongoDB est une base de données basée sur des documents NoSQL. Son schéma flexible facilite l'évolution et le stockage des données d'une manière simple pour les programmeurs. Il utilise un format de données similaire à JSON, appelé BSON, stockant les données dans des objets structurés ou non structurés appelés documents. Ces documents sont regroupés en collections.

Mongoose est une bibliothèque de modélisation de données d'objet (ODM) pour MongoDB permettant un développement rapide et simple des interactions de base de données MongoDB. Il offre des fonctionnalités pour la modélisation des données, l'application des schémas, la validation des modèles et la manipulation des données.

Postman est une plate-forme API permettant aux développeurs de concevoir, construire, tester et itérer leurs API

Back-end Méthodologies

JSend est une spécification qui établit certaines règles sur la façon dont les réponses JSON des serveurs Web doivent être formatées. Cette norme répond en créant un nouvel objet combinant les données demandées et un message d'état informant l'utilisateur si la demande a été un succès, un échec ou une erreur.

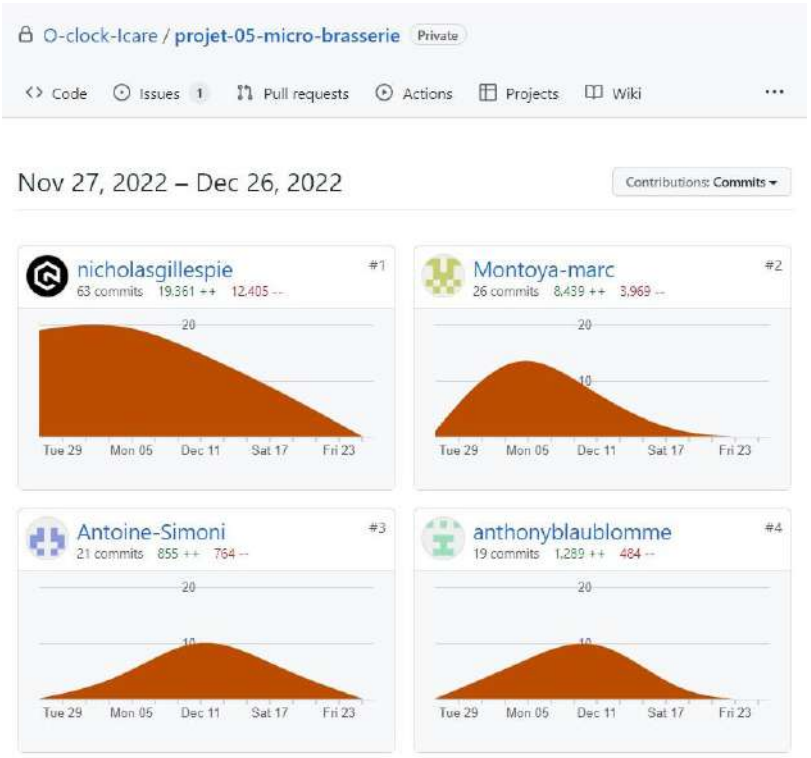
Versioning

Le versioning de ce projet a été réalisé en utilisant en combinaison Git et GitHub. Ces outils nous ont permis de travailler en collaboration, chacun travaillant sur des éléments spécifiques tout en nous permettant en même temps de joindre ces éléments quand fonctionnelles.

Chaque membre du groupe travaillait sur ses tâches au sein de sa propre branche.

Lorsqu'une fonctionnalité était fonctionnelle, celle-ci était alors transférée de la branche de l'individu pour finalement être fusionnée à la branche principale, nommée ‘main’.

Les autres membres du groupe pouvaient alors extraire cette version mise, en effectuant un ‘pull’, à jour tout en ne créant généralement pas de conflits dans les versions de code.



Si un conflit de versions survenait, une révision du code serait nécessaire. Cela consiste à analyser les deux versions du code et décider laquelle est pertinente.

Fonctionner de cette manière permet de monter un projet en collaboration.

Architecture du Projet

Cette application Web est un site Web dynamique rendu côté serveur, structuré selon le modèle architectural MVC (Model–View–Controller), qui consomme une API (Application Program Interface) suivant l'architecture REST.

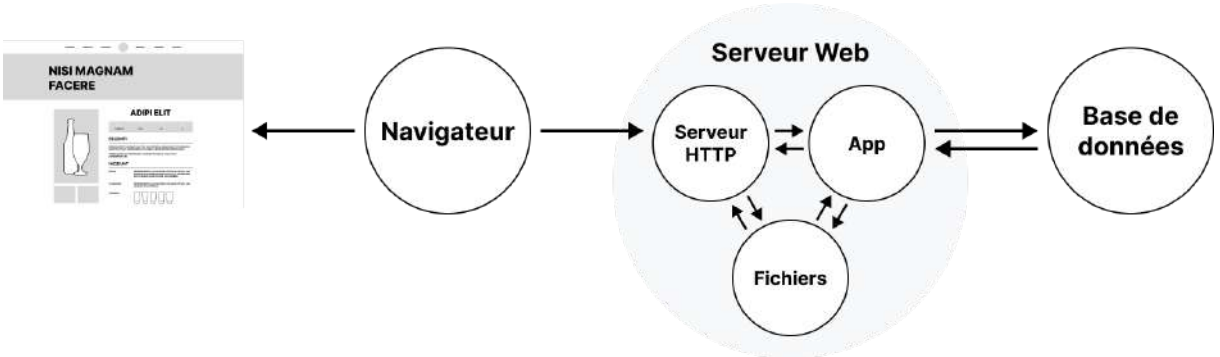
Le processus de rendu du site Web peut être expliqué de la manière suivante:
Chaque fois qu'un utilisateur, côté navigateur, fait une requête, le serveur HTTP, en collaboration avec l'application, traite la requête, récupère les données de la base de données, qui sont ensuite insérées dans un template, assemblées, et renvoyées au client.

Composants:

- Navigateur

- Serveur Web
    - Serveur HTTP: Logiciel capable de communiquer avec le navigateur à l'aide de requêtes et de réponses.
    - Fichiers: HTML, CSS, images, etc.
    - Application: Récupère les données de la base de données et insère ces données dans des modèles prédéfinis, en fonction de la demande reçue du serveur.

- Base de données



Circulation de la données

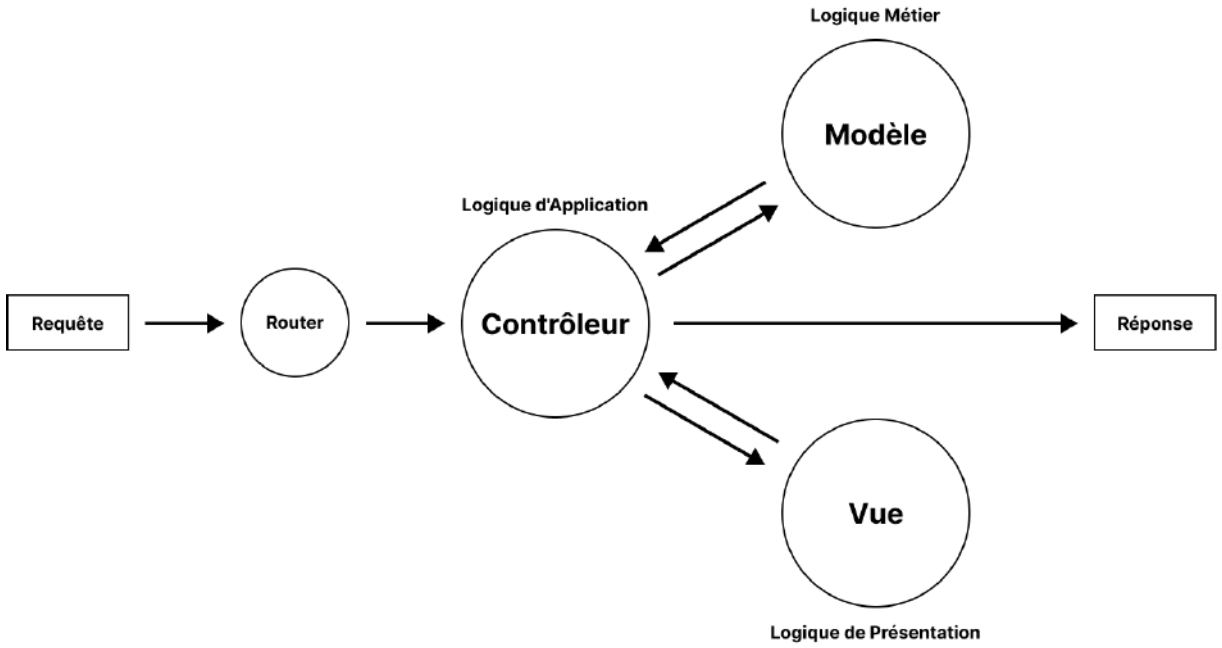
Le début du cycle requête-réponse se produit lorsqu'une requête arrive sur le serveur. À partir de là, un objet de requête et de réponse est créé. Cet objet est ensuite utilisé et traité dans toute l'application pour générer des réponses significatives.
La manipulation de la requête/réponse est traitée par des middlewares, qui sont des fonctions exécutées entre la réception et l'envoi de la réponse.

La circulation des données se fait de manière quelque peu linéaire, parcourant chaque middleware au fur et à mesure pour atteindre sa destination finale qui est une réponse renvoyée au client, complétant le cycle requête / réponse.

MVC

Ce projet suit l'architecture MVC (Modèle, Vue, Contrôleur). L'architecture MVC a pour objectif de séparer les préoccupations. Séparer la logique métier, la logique applicative et la logique de présentation.

- Modèles prennent en charge la logique métier
- Contrôleurs de logique d'application
- Vues de la logique de présentation



Plus en détail, la partie contrôleur doit gérer les demandes de l'application, l'interaction avec les modèles et renvoyer les réponses au client. C'est la logique d'application; c'est le code qui gère la gestion des requêtes et des réponses.

beerController.js

```
async getAllBeers(req, res, next) {
  // prepare query & execute
  const query = Beer.find();
  const result = await query;
  // return response
  res.status(201).json({
    status: 'success',
    results: result.length,
    data: { beers: result },
  });
},

async getBeer(req, res, next) {
  // prepare query & execute
  const query = Beer.findOne({ slug: req.params.slug }, '-_id name slug
  description image imageCover style abv ibu ebv malts hops price');
  const result = await query;
  // if no result, throw error
  if (!result) return next(new AppError('No beer found with that ID', 404));
  // return response
  res.status(201).json({
    status: 'success',
    data: { beer: result },
  });
},
```

La partie modèle concerne tout ce qui concerne les données de l'application et la logique métier; donc le code qui ne se préoccupe que de l'implémentation de l'application, c'est le code qui résout le problème métier.

beerModel.js

```
const { Schema, model } = mongoose;

const beerSchema = new Schema({
  name: {
    type: String,
    required: [true, 'A beer must have a name'],
    unique: true,
    trim: true,
    maxlength: [20, 'A beer name must have less or equal than 20 characters'],
    minlength: [3, 'A beer name must have more or equal than 3 characters'],
  },
  slug: String,
  description: {
    type: String,
    required: [true, 'A beer must have a description'],
    trim: true,
  },
  imageCover: {
    type: String,
    default: 'image beers.png',
  },
  images: [String],
  style: {
    type: String,
    required: [true, 'A beer must have a style'],
  },
});

/* MODEL //////////////////// */
const Beer = model('Beer', beerSchema);
```

Et puis la partie vue s'occupe de la partie interface graphique. C’est la logique de présentation.

beer.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="/css/index.css">
  <script defer src="/js/slideout.js"></script>
  <title><%= page %> - Expedition Brewery</title>
</head>

<%- include('partials/_header'); -%>

<h2 class="title-page"><%= page %></h2>

<section class="[ wrapper region ][ beer ]" data-region="large">
  <div class="grid">
    <% data.forEach(function(beer){ %>
      <figure>
        <a href="/beers/<%= beer.slug%>">
          <% if (beer.imageCover) { %>
            <img src="../img/beers/<%= beer.imageCover%>" alt="<%= beer.name%>">
          <% } %>
          <p><%= beer.name %></p>
        </a>
      </figure>
    <% }); %>
  </div>
</section>

<%- include('partials/_footer'); -%>
```

L'utilisation d'une telle architecture permet d'écrire des applications plus modulaires, ce qui facilite la maintenance et l'évolutivité.

API RESTful

REST (Representational State Transfer) est un modèle architectural pour la création d'API Web qui a pour but de standardiser et faciliter la consommation d'API. Ce modèle suit divers principes.

- Séparation de l'API en ressources logiques auxquelles sont associées des données. Dans le cadre de ce projet : beers, users.

- Exposition des URL d'une manière structurées, basées sur les ressources, permettant divers actions tel CRUD. Les endpoints ne doivent contenir que nos ressources et non les actions qui peuvent être effectuées sur celles-ci.

beerRoutes.js

```
/* ROUTER ////////////////////// */
const router = express.Router();

/* ROUTE ////////////////////// */
router
  .route('/')
  .get(cw(beerController.getAllBeers))
  .post(cw(beerController.createBeer));

router
  .route('/:slug')
  .get(cw(beerController.getBeer))
  .patch(cw(beerController.updateBeer))
  .delete(cw(beerController.deleteBeer));

/* EXPORT ////////////////////// */
export default router;
```

- Utilisation de la méthode HTTP appropriée en rapport à l'action visée et non l'URL.
  Create : POST
  Read : GET
  Update: PUT / PATCH
  Delete : DELETE

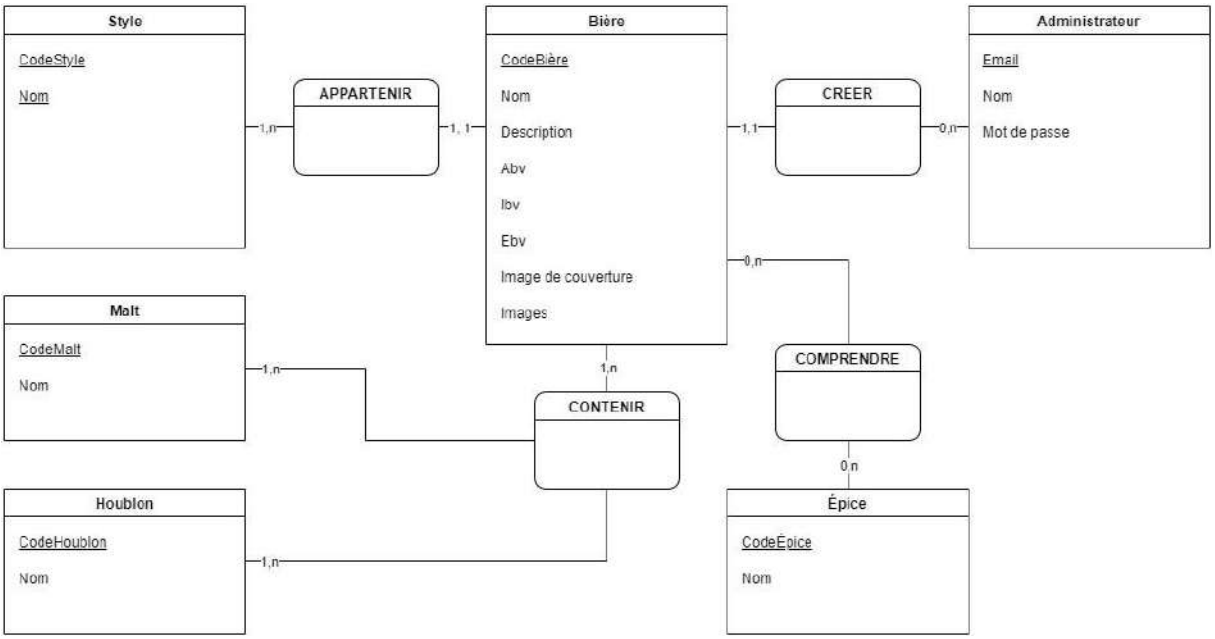
- Transfert de données en format JSON suivant une norme de formatage.
  La norme de formatage suivie dans ce projet est JSend. Cela consiste à envelopper l'objet et lui ajouter un message d'état afin d'informer le client si la requête a été un "success", "fail" ou "erreur".

- L'API doit être 'stateless'.
  "Stateless" signifie que tous les états doivent être gérés sur le client. Le serveur ne devrait pas avoir à se souvenir des requêtes précédentes.

Modèles de données

Un modèle de données est une représentation visuelle de la manière dont les données sont stockées, ainsi que de la manière dont les données sont liées les unes aux autres.

Le modèle de données conceptuel (MCD) explique ce que le système doit contenir en ce qui concerne les données et comment elles sont liées. Il représente la logique métier de l'application.



MongoDB effectue les enregistrements de données dans des documents. Ces documents utilisent le format de données BSON; un format similaire au JSON. Cela permet aux données d'être structurées différemment que dans une base de données relationnelle.

Entres autres, les documents peuvent incorporer en eux-mêmes des données. Lorsque cela est fait, cela se dit "embedding data" (intégration de données). Ces schémas sont appelés modèles "dénormalisés" par opposition au terme connu de modèles “normalisés”; décrivant les relations faites par référence.

L'un des avantages de l'incorporation de documents, valable dans certaines circonstances, pas toutes, est que cela peut réduire le nombre de requêtes qu'une base de données doit effectuer pour récupérer des données. Notion à prendre en compte sachant que chaque requête a, à la fois, un coût en ressources et un coût en temps.

MongoDB recommande d'intégrer des documents afin de simplifier les requêtes et d'améliorer les performances globales des celles- ci.

Cependant, il faut être conscient que l'intégration de données dans un seul document peut, avec le temps, devenir un document volumineux, entraînant une latence supplémentaire pour les opérations de lecture. De plus, l'ajout de données à un document sans limite peut créer ce qui est appelé un “unbounded document” (document illimité). Ceci encourt le risque de dépasser le seuil de document BSON de 16 Mo.

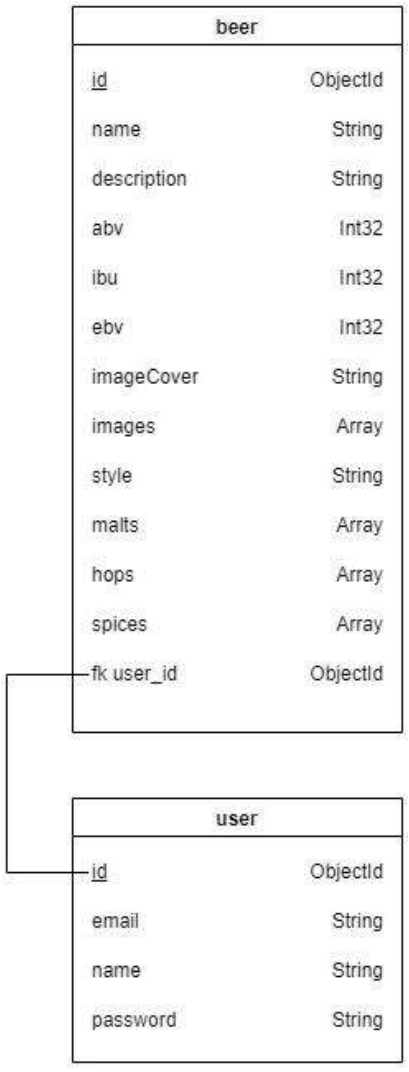
Dans le cas de ce projet, sachant qu'une bière ne contiendra qu'une quantité limitée de malts, de houblon ou d'épices, il est judicieux d'intégrer ces trois entités à même l’entité beer.

Modèle logique de données (MLD)

- beers (
  #id,
  name,
  description,
  abv,
  ibu,
  ebv, imageCover,
  style,
  images,
  malts,
  hops,
  spices,
  #user_id
)

- user (#id, email, name, password)

Modèle physique de données (MPD)



La représentation physique est ensuite utilisée pour développer les schémas de ces entités. *Aperçu du schéma de la bière visible page 35*.

Sécurité de l’application

- Brute-force Attacks (BFA)
  Lorsque l'attaquant essaie de deviner un mot de passe en essayant des millions de mots de passe aléatoires jusqu'à ce qu'il trouve le bon.
  Moyens mis en oeuvre pour contrer cela :
  - Rate limiting, qui limite le nombre de requêtes provenant d'une seule IP (en utilisant le middleware: express-rate-limit).
  - Bcrypt, sécurise et ralentit le processus de demande de connexion.

- Denial of Service Attacks (DOS)
  Lorsque l'attaquant submerge le serveur de requêtes à tel point qu'il tombe en panne et que l'application se déconnecte.
  - Rate limiting, qui limite le nombre de requêtes provenant d'une seule IP (en utilisant le middleware: express-rate-limit).
  - Limitant la quantité de données pouvant être envoyées dans la requête body post/patch; en spécifiant les options dans le body parser (express.json({ limit: '10 kb' })).

- Cross-Site Scripting Attacks (XSS)
  Lorsque l'attaquant essaie d'injecter des scripts dans notre page pour exécuter un code malveillant. Risque potentiel d'accéder au local storage du client.
  - Définition des en-têtes HTTP de sécurité (à l'aide du middleware helmet (middleware comprenant une collection d’autres middlewares).
  - Stockage JWT dans le cookie HTTPOnly (res.cookie('jwt', token, cookieOptions);).
  - Désinfection des données entrées par l'utilisateur à l'aide du middleware xss-clean, nettoyant toute entrée utilisateur malveillante.

- NoSQL Query Injection
  Lorsque l'attaquant injecte des ‘query expressions’ qui résulterait par une réponse de la BDD retournant ‘true’; problème majeur si cela concerne une demande d'authentification.
  - Définir un bon schéma et des types Mongoose.
  - Assainir les données entrées par l'utilisateur à l'aide du middleware express-mongo-sanitize. Ce middleware examine la requête body, string & params et filtre tous les symboles qui pourraient être des opérateurs MongoDB.

- HTTP Parameter Pollution (HPP)
  Lorsque l'attaquant exploite un duplicate field passé dans les paramètres.
    - Suppression des champs en double passés dans les paramètres à l'aide de hpp (HTTP parameter pollution).

- Utilisez HTTPS
  Risque potentiel de menace externe; exposition de communications, vol de jetons, etc.

GESTION DE PROJET

Présentation de l'équipe

L'équipe de ce projet est composée de quatre développeurs:

- Nicholas GILLESPIE: Product owner & lead developer back-end
  Responsable de la définition et de la conception du projet
  Responsable du développement back-end ainsi que des choix techniques

- Anthony BLAUBLOMME: Lead dev front & scrum master
  Supervise la partie front du développement
  Supervise l'avancement global du projet et des tâches effectuées

- Marc MONTOYA: Developer back-end
  Assistance back-end

- Antoine SIMONI: Developer front-end
  Assistance front-end

Organisation de travail

Ce projet est basé sur le principe de la méthode agile scrum. Le projet a été divisé en 4 sprints.

Sprint	Équipe	Description
0	Complète	Définition du cahier de charges, documents divers (wireframe, MCD, MDL, schéma, routes, etc.).
1	Front	Développement de l'intégration. Installation des modules requis. Implémentation des routes.
	Back	Mise en place du la structure du projet. Création de la BDD. Développement de l'API. Configuration du gestionnaire d'erreurs et implémentations sécuritaires.
2	Front	Continuation de l'intégration. Dynamisation d'éléments. Développement des fonctionnalités CRUD.

2	Back	Développement du système d'authentification et d'autorisation ainsi que la fonctionnalité d'import de fichiers.
3	Front	HS raison médicale.
	Back	Dernier rush pour finaliser certaines fonctionnalités. Résolution de problèmes.

Outils utilisés

Communication
- Slack
- Discord

Documentation
- Google Drive
- Google Docs

Gestion de Projet
- Trello

Versioning
- Git
- GitHub

Développement
- VS Code
- Postman

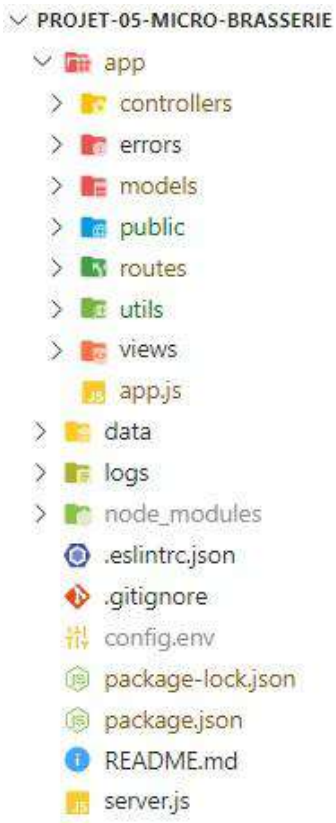
RÉALISATION DU CANDIDAT

Création maquettes

Suite à une réunion durant laquelle les clients m'ont présenté divers éléments à leur goût, j'ai créé une maquette, visible page 12, pour la page accueil, bières et bière unique.

Mise en place de l'architecture du projet

J'ai mis en place la structure de fichiers du projet.



server.js

```
/* IMPORT MODULE ////////////////////// */
import http from 'http';
import './app/utils/loadEnv.js';

/* IMPORT UNCAUGHT EXCEPTION ////////////////////// */
import './app/errors/uncaughtException.js';

/* IMPORT APP ////////////////////// */
import app from './app/app.js';

/* START SERVER ////////////////////// */
const server = http.createServer(app);
const port = process.env.PORT || 3000;
server.listen(port, () => {
  console.log(`App running on http://localhost:${port}`);
});

/* UNHANDLED REJECTION ////////////////////// */
process.on('unhandledRejection', (err) => {
  console.error('UNHANDLED REJECTION! 💥 Throwing error...');
  console.error(err);
  throw err;
});
```

Classes utilitaires CSS - Définition des design tokens

Je souhaitais que ce projet soit construit en utilisant la méthodologie CUBE CSS en combinaison avec des design tokens. La combinaison de ces deux technologies, en plus de l'utilisation des viewport measures, permet la création de site Web réactif de manière algorithmique; réduisant considérablement la nécessité des media queries.

Définition des design tokens ainsi que variables spatiales.

```
:root {
  /* VIEWPORTS ////////////////////// */
  /* 280px & 1360px */
  /* --min: 17.5rem; */
  --max: 85rem;

  /* SCALES ////////////////////// */
  /* type scale */
  --type--2: clamp(0.69rem, calc(0.69rem + 0.04vw), 0.72rem);
  --type--1: clamp(0.83rem, calc(0.81rem + 0.12vw), 0.90rem);
  --type-0: clamp(1.00rem, calc(0.96rem + 0.22vw), 1.13rem);
  --type-1: clamp(1.20rem, calc(1.13rem + 0.36vw), 1.41rem);
  --type-2: clamp(1.44rem, calc(1.33rem + 0.55vw), 1.76rem);
  --type-3: clamp(1.73rem, calc(1.56rem + 0.82vw), 2.20rem);
  --type-4: clamp(2.07rem, calc(1.84rem + 1.17vw), 2.75rem);
  --type-5: clamp(2.49rem, calc(2.16rem + 1.64vw), 3.43rem);

  /* space scale */
  --space-3xs: clamp(0.25rem, calc(0.23rem + 0.11vw), 0.31rem);
  --space-2xs: clamp(0.50rem, calc(0.48rem + 0.11vw), 0.56rem);
  --space-xs: clamp(0.75rem, calc(0.71rem + 0.22vw), 0.88rem);
  --space-s: clamp(1.00rem, calc(0.96rem + 0.22vw), 1.13rem);
  --space-m: clamp(1.50rem, calc(1.43rem + 0.33vw), 1.69rem);
  --space-l: clamp(2.00rem, calc(1.91rem + 0.43vw), 2.25rem);
  --space-xl: clamp(3.00rem, calc(2.87rem + 0.65vw), 3.38rem);
  --space-2xl: clamp(4.00rem, calc(3.83rem + 0.87vw), 4.50rem);
  --space-3xl: clamp(6.00rem, calc(5.74rem + 1.30vw), 6.75rem);

  /* One-up pairs */
  --space-3xs-2xs: clamp(0.25rem, calc(0.14rem + 0.54vw), 0.56rem);
  --space-2xs-xs: clamp(0.50rem, calc(0.37rem + 0.65vw), 0.88rem);
  --space-xs-s: clamp(0.75rem, calc(0.62rem + 0.65vw), 1.13rem);
  --space-s-m: clamp(1.00rem, calc(0.76rem + 1.20vw), 1.69rem);
  --space-m-l: clamp(1.50rem, calc(1.24rem + 1.30vw), 2.25rem);
  --space-l-xl: clamp(2.00rem, calc(1.52rem + 2.39vw), 3.38rem);
  --space-xl-2xl: clamp(3.00rem, calc(2.48rem + 2.61vw), 4.50rem);
  --space-2xl-3xl: clamp(4.00rem, calc(3.04rem + 4.78vw), 6.75rem);

  /* SIZES & SPACINGS////////////////////// */
  /* sizes */
  --measure: 75ch;
  --line-height: 1.4;
  --line-height-small: 1;

  /* spacings */
  --gutter: var(--space-s);
  --20: var(--space-2xs);
  --40: var(--space-l);
  --60: var(--space-xl);
  --80: var(--space-2xl);
  --120: calc(var(--space-xl) * 2);
```

Classes utilitaires consommant les variables spatiales.

```
/* WRAPPER */
.wrapper {
  max-inline-size: var(--wrapper-max-inline-size, 85rem);
  margin-inline-start: auto;
  margin-inline-end: auto;
  padding-inline-start: var(--gutter);
  padding-inline-end: var(--gutter);
  position: var(--wrapper-position, relative);
}

/* REGION */
.region {
  padding-block-start: var(--region-padding-start, var(--gutter));
  padding-block-end: var(--region-padding-end, var(--gutter));
}
.region[data-region='large'] {
  --region-padding-start: var(--space-xl);
  --region-padding-end: var(--space-xl);
}

/* CLUSTER */
.cluster {
  display: flex;
  flex-wrap: wrap;
  gap: var(--cluster-gap, var(--gutter));
  justify-content: var(--cluster-justify-content, flex-start);
  align-items: var(--cluster-align-items, center);
}
.cluster[data-cluster='space-between'] {
  justify-content: space-between;
  align-content: space-between
}
.cluster[data-cluster='column'] {
  flex-direction: column;
  justify-content: space-between;
  --cluster-align-items: flex-start;
}

/* BOX */
.box {
  padding: var(--space-s);
  border: var(--border);
  color: var(--color-dark);
  background-color: var(--color-light);
}

/* STACK */
.stack {
  display: flex;
  flex-direction: column;
  justify-content: flex-start;
}
```

Mes collègues n'étaient pas familiers avec les design tokens ou la méthodologie CUBE CSS. En plus de les diriger vers la documentation web.dev fournie par les développeurs Google Chrome, prêcheurs de ces méthodes, j'ai passé 1 journée à former les deux développeurs front-end participant à ce projet, sur cette technologie.

Utilisation des classes utilitaires CSS dans le document HTML.

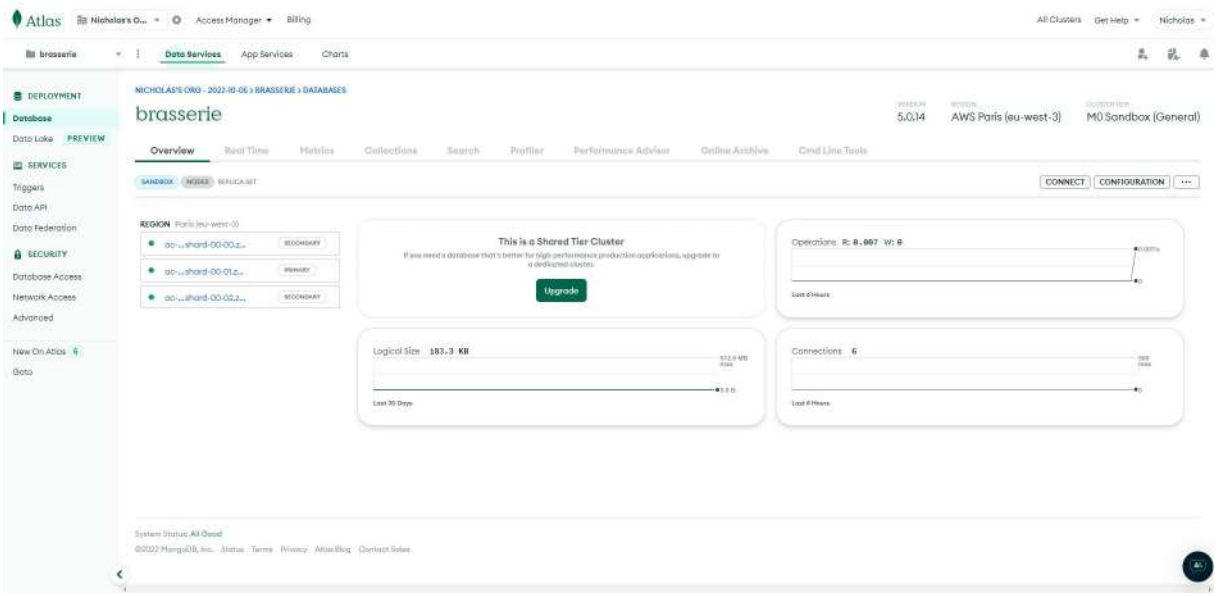
Vous remarquerez peut-être des parenthèses entourant les classes. Ceci est utilisé pour distinguer les classes utilisées pour la composition, l'utilité, les blocs et les exceptions. HTML, étant un langage permissif, ignore simplement les valeurs qu'il ne comprend pas, permettant à l'utilisateur humain de l'utiliser à des fins de distinction.

accueil.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="/css/index.css">
  <script defer src="/js/script.js"></script>
  <title>Expedition Brewery</title>
</head>

<%- include('partials/_header'); -%>

<section class="[ cover ][ b-hero ]"></section>

    <section class="[ wrapper ][ b-about-us ]" data-wrapper='medium'>
      <div class="sidebar">
        <h1>Qui sommes nous ?</h1>
        <div class="flow">
          <p id="about-us">Jeune brasserie fondée en 2022, Expedition Brewery commence deux ans plus tôt
          dans les rêves de deux brasseurs passionnés. D'abord influencés par la scène craft anglo-saxonne,
          nous voulons faire redécouvrir au plus grand nombre la diversité gustative, souvent oubliée, de
          ce breuvage à la fois simple et complexe. </p>
          <p>Aujourd'hui nous expérimentons et créons des recettes originales, sans compromis sur le gout,
          pour une expérience de dégustation unique.</p>
          <a class="button" href="/contact">Contactez-nous</a>
        </div>
      </div>
    </section>

    <section class="b-bottles">
      <div class="wrapper region stack" data-wrapper='medium'>
        <h2>Notre Catalogue</h2>
        <img src="../img/image beers.png" alt="image des bières">
      </div>
    </section>

    <section class="b-about-beers">
      <div class="wrapper" data-wrapper='x-large'>
        <div class="sidebar">
          <img class="frame" src="../img/biereverre2.png" alt="image de la brasserie" width="300px">
          <div class="flow">
            <h2>Nos bières</h2>
            <p>Notre processus de brassage est encore entièrement manuel. Nous fabriquons des bières
            modernes en suivant les mêmes étapes que les brasseurs d'autrefois, afin de laisser s'exprimer
```

Création BDD & connection application

- Création du projet et de la base de données via l'interface MongoDB Atlas.



- Connexion entre la base de données et l'application Web utilisant le lien de connexion stocké côté serveur dans un fichier de configuration.

```
/* MODULE //////////////////// */
import { MongoClient } from 'mongodb';
import './loadEnv.js';

/* DATABASE //////////////////// */
const uri = process.env.DATABASE_URI.replace(
  '<NAME>',
  process.env.DATABASE_NAME,
).replace(
  '<USER>',
  process.env.DATABASE_USER,
).replace(
  '<PASSWORD>',
  process.env.DATABASE_PASSWORD,
);

/* CREATE MONGO CLIENT //////////////////// */
const client = new MongoClient(uri);

/* EXPORT //////////////////// */
export default client;
```

```
/* DATABASE CONNECTION //////////////////// */
import './utils/dbCon.js';
```

Création de l'API

- Définition de l'URL des bières de l'API.

```
/* MOUNT ROUTE ////////////////////// */
app.use('/', viewRouter);
app.use('/api/v1/beers', beerRouter);
```

- Implémentation des différentes méthodes HTTP sur les routes des bières.

```
/* ROUTER ////////////////////// */
const router = express.Router();

/* ROUTE ////////////////////// */
router
  .route('/')
  .get(cw(beerController.getAllBeers))
  .post(cw(beerController.createBeer));

router
  .route('/:slug')
  .get(cw(beerController.getBeer))
  .patch(cw(beerController.updateBeer))
  .delete(cw(beerController.deleteBeer));

/* EXPORT ////////////////////// */
export default router;
```

- Implémentation du contrôleur de bière.

```
async getAllBeers(req, res, next) {
  // prepare query & execute
  const query = Beer.find();
  const result = await query;
  // return response
  res.status(201).json({
    status: 'success',
    results: result.length,
    data: { beers: result },
  });
},

async getBeer(req, res, next) {
  // prepare query & execute
  const query = Beer.findOne({ slug: req.params.slug }, '-_id name slug
  description image imageCover style abv ibu ebv malts hops price');
  const result = await query;
  // if no result, throw error
  if (!result) return next(new AppError('No beer found with that ID', 404));
  // return response
  res.status(201).json({
    status: 'success',
    data: { beer: result },
  });
},
```

- Définition du schéma de la bière. Création du modèle.

```
/* SCHEMA ////////////////////// */
const { Schema, model } = mongoose;

const beerSchema = new Schema({
  name: {
    type: String,
    required: [true, 'A beer must have a name'],
    unique: true,
    trim: true,
    maxlength: [20, 'A beer name must have less or equal than 20 characters'],
    minlength: [3, 'A beer name must have more or equal than 3 characters'],
    // validate: [validator.isAlpha, 'Beer name must only contain characters']
  },
  slug: String,
  description: {
    type: String,
    required: [true, 'A beer must have a description'],
    trim: true,
  },
  imageCover: {
    type: String,
    default: 'image beers.png',
    required: [true, 'A beer must have a cover image'],
  },
  images: [String],
  style: {
    type: String,
    required: [true, 'A beer must have a style'],
  },
  abv: {
    type: Number,
    required: [true, 'A beer must have a abv'],
  },
  ibu: {
    type: Number,
    required: [true, 'A beer must have a ibu'],
  },
  ebv: {
    type: Number,
    required: [true, 'A beer must have a ebv'],
  },
  malts: [String],
  hops: [String],
  createdAt: {
    type: Date,
    default: () => Date.now(),
    immutable: true,
  },
  updatedAt: Date,
});
```

- Création, via l'API, utilisant l’interface Postman, des enregistrements de la base de données.
- Tests pour garantir toutes les fonctions d'exploitation, CRUD.

Création du gestionnaire d'erreur global

Afin de fournir une expérience utilisateur sûre et sensée, il est impératif de gérer les erreurs. Pour cela, il est nécessaire de différencier les deux types d'erreur :

- Erreurs opérationnelles, qui dépendent de l'utilisateur, du système ou du réseau (ex. itinéraire invalide, saisie de données invalides, etc.).
  Ces erreurs doivent être gérées afin de permettre le fonctionnement normal d'une application.

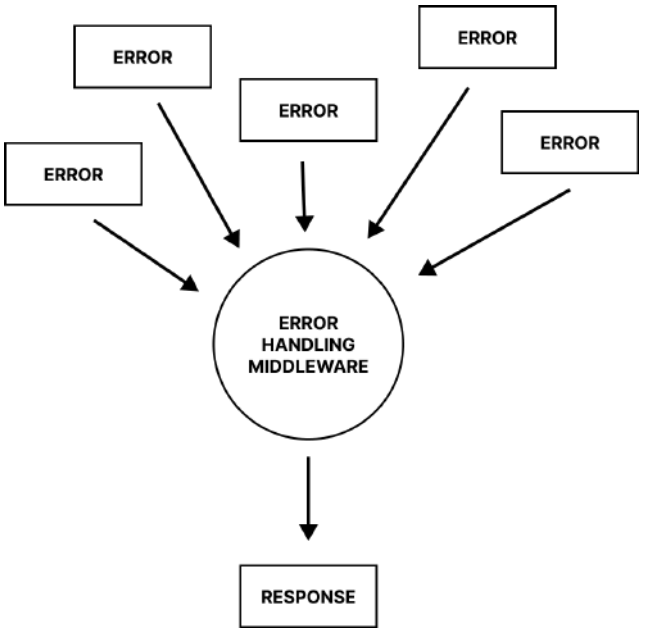
- Les erreurs de programmation, qui sont des bugs introduits lors du développement (ex. lecture de variables indéfinies, passage d'un type incorrect, utilisation de await sans async, etc.).

Selon le type, ils doivent être traités d'une certaine manière. Cela pourrait être en envoyant une réponse appropriée au client, en lui faisant savoir comment réagir à la situation.

Dans d'autres cas, cela peut signifier de réessayer l'opération, d'ignorer potentiellement l'erreur ou, dans le pire des cas, de planter le serveur.

La manière la plus appropriée de gérer ces erreurs consiste à créer un middleware global de gestion des erreurs, fonctionnant de la même manière qu'un entonnoir, qui sépare finalement les deux types et renvoie les réponses en conséquence.

Ainsi, peu importe si une erreur provient d'un gestionnaire de route, d'un validateur de modèle ou d'un autre endroit, ces erreurs se retrouvent toutes dans un middleware central de gestion des erreurs.



Le développement d'un middleware de gestion d'erreurs pour cette application s'est fait de la manière suivante :

- Définir un middleware de gestion des erreurs en passant 4 arguments dans une fonction middleware (err, req, res, next). Express reconnaît naturellement une telle structure en tant que middleware de gestion des erreurs, puis l'appelle en cas d'erreur.

- L'application des paramètres d'erreur par défaut aux propriétés statusCode et status des objets d'erreur dans l'éventualité où aucun n'a été définie auparavant dans l'application.

- Renvoie une réponse conformément à la propriété statusCode de l'erreur.

De plus, à partir de la classe innée d'Express, Error, une classe AppError a été créée. Cette classe est utilisée pour traiter les erreurs opérationnelles.

Dans les situations où des erreurs opérationnelles sont attendues (ex. ID incorrect pour une demande, information ou format de données incorrect, message procédural, etc.), une instance de la classe AppError peut être instanciée.

Lors de l'instanciation de la nouvelle erreur, 2 valeurs sont transmises:
- un message
- un statusCode.

Celui-ci serait ensuite transmis au gestionnaire d'erreurs en l'appelant à l'aide de next(), précédé de return. Cette configuration envoie l'erreur directement au middleware de gestion des erreurs pour qu'elle soit traitée.

Classe AppError

```
/* EXPORT ////////////////////// */
export default class AppError extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
    this.status = `${statusCode}`.startsWith('4') ? 'fail' : 'error';
    this.isOperational = true; /* isOperation property added to seperate Operational errors from Programming ones */
    Error.captureStackTrace(this, this.constructor); /* create .stack property on a target object */
  }
}
```

Gestionnaire d'erreurs

```
/* IMPORT DIFFERENT ERROR RESPONSES ////////////////////// */
import sendErrorDev from './sendErrorDev.js';
import sendErrorProd from './sendErrorProd.js';
import ErrorProdFxs from './sendErrorProdFxs.js';

/* EXPORT //////////////////////// */
export default (err, req, res, next) => {
  let error = Object.assign(Object.create(err), { ...err });
  error.statusCode = err.statusCode || 500;
  error.status = err.status || 'error';
  // difference between development and production
  if (process.env.NODE_ENV === 'development') {
    sendErrorDev(error, req, res);
  } else if (process.env.NODE_ENV === 'production') {
    if (error.name === 'CastError') error = ErrorProdFxs.handleCastErrorDB(error);
    if (error.code === 11000) error = ErrorProdFxs.handleDuplicateFieldsDB(error);
    if (error.name === 'ValidationError') error = ErrorProdFxs.handleValidationErrorDB(error);
    if (error.name === 'JsonWebTokenError') error = ErrorProdFxs.handleJsonWebTokenError(error);
    if (error.name === 'TokenExpiredError') error = ErrorProdFxs.handleTokenExpiredError(error);
    sendErrorProd(error, req, res);
  }
};
```

Gestionnaire d'erreurs positionné dans le stack

```
/* UNHANDLED ROUTE ////////////////////// */
app.all('*', (req, res, next) => {
  next(new AppError(`Can't find ${req.originalUrl} on this server`, 404));
});

/* ERROR HANDLING ////////////////////// */
app.use(globalErrorHandler);

/* EXPORT ////////////////////// */
export default app;
```

Définition des erreurs à travers l'application

- Itinéraires non gérés
- Contrôleurs
- Fonctions d'authentification
- Middlewares utilitaires
- Etc.

Itinéraires non gérés
app.js

```
/* MOUNT ROUTE ////////////////////// */
app.use('/', viewRouter);
app.use('/api/v1/beers', beerRouter);

/* UNHANDLED ROUTE ////////////////////// */
app.all('*', (req, res, next) => {
  next(new AppError(`Can't find ${req.originalUrl} on this server`, 404));
});

/* ERROR HANDLING ////////////////////// */
app.use(globalErrorHandler);
```

beerController.js

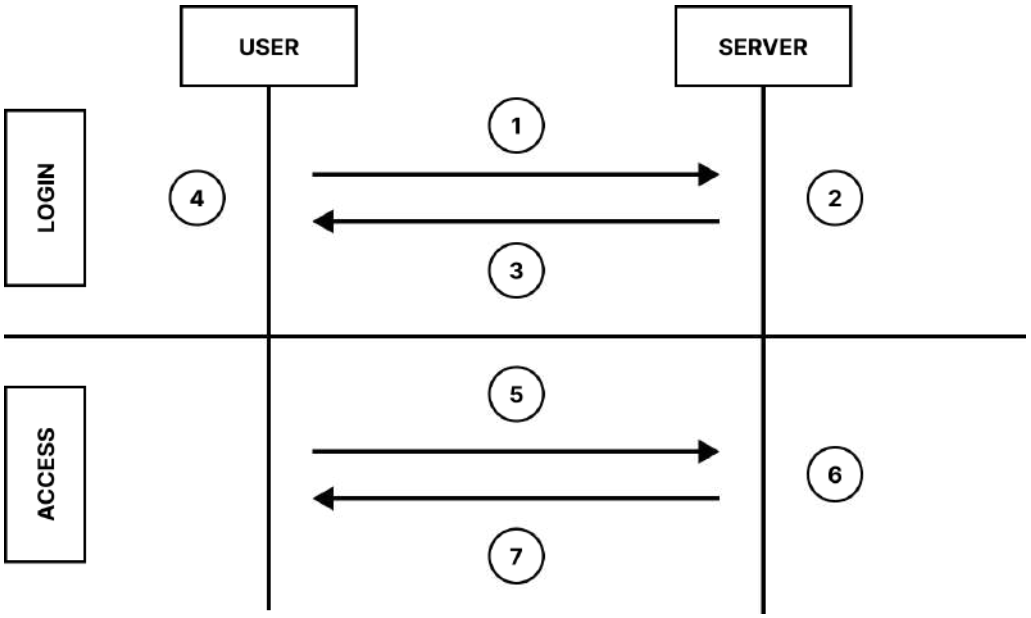
```
async updateBeer(req, res, next) {
  // prepare query & execute
  /* [options.new=false] «Boolean» if true, return the modified document rather than the original
  [options.runValidators] «Boolean» if true, runs update validators on this command.
  Update validators validate the update operation against the model's schema XXX */
  if (req.file) req.body.imageCover = req.file.filename;
  const query = Beer.findOneAndUpdate({ slug: req.params.slug }, req.body, {
    new: true,
    runValidators: true,
  });
  const result = await query;
  // if no result, throw error
  if (!result) return next(new AppError('No beer found with that ID', 404));
  // return response
  res.status(201).json({
    status: 'success',
    data: { beer: result },
  });
},

async deleteBeer(req, res, next) {
  // prepare query & execute
  const query = Beer.findOneAndDelete({ slug: req.params.slug });
  const result = await query;
  // if no result, throw error
  if (!result) return next(new AppError('No beer found with that ID', 404));
  // return response
  /* 204: No Content. In a REST API, 204 is the code to use if the action was processed successfully,
  but there is no content to return. XXX */
  res.status(204).json({
    status: 'success',
    data: null,
  });
},
```

Authentification et autorisation

Pour ce projet, les JSON Web Tokens (JWT) sont utilisés comme méthode d'authentification. En plus d'être une approche moderne et simple, les JWT sont également stateless, ce qui, pour ce projet, répond au principe de stateless poursuivi.
Contrairement aux alternatives d'authentification les plus largement utilisées, les JWT ne nécessitent pas d'enregistrer les détails de sessions sur le serveur.

Les JWT fonctionnent de la manière suivante :

1. L'utilisateur envoie une post request avec e-mail/nom d'utilisateur et mot de passe.
2. L'application vérifie si l'utilisateur existe et si le mot de passe est correct.
3. Si true, un JWT unique est créé pour cet utilisateur, à l'aide d'une chaîne secrète stockée sur le serveur.
4. Ce JWT unique est renvoyé à l'utilisateur qui est stocké dans un cookie ou un stockage local.
5. Chaque fois que l'utilisateur souhaite ensuite accéder à une route protégée, il envoie son JWT accompagné d'une requête.
6. Notre application vérifie ensuite que le JWT est valide.
7. Si true, les données demandées seront envoyées au client. Sinon, une erreur sera renvoyée.



L'ensemble de ce processus nécessite que toutes les communications soient effectuées via HTTPS afin d'éviter les fuites potentielles de mots de passe ou de JWT.

Un JWT est essentiellement une chaîne codée composée de 3 parties :

1. Header
2. Payload
3. Signature

Le header est une métadonnée sur le jeton. Le payload est constitué de données que nous choisissons d'encoder dans le jeton. La signature est créée à l'aide du header, du payload et d'un secret choisi, enregistré sur le serveur.
Les 3 combinés avec l'algorithme JWT est ce qui est utilisé pour créer un jeton.

JWT documentation: https://jwt.io/introduction

Le processus de vérification s'effectue en vérifiant le header et le payload, en le combinant avec le secret enregistré sur le serveur et en créant un jeton de test. Le jeton de test est ensuite comparé au jeton d'origine. S'ils sont identiques, l'authentification est accordée.

Si les 2 jetons ne sont pas identiques, cela signifierait qu'une entité a trafiqué les données. Mais n'ayant pas accès au secret enregistré côté serveur, le jeton ne correspondra jamais au jeton d'origine, ce qui entraîne donc un refus d'authentification.

Pour l'inscription et la vérification, le projet utilise le package npm jsonwebtokens. L'inscription se fait à l'aide de la fonction jwt.sign() et la vérification à l'aide de la fonction jwt.verify().

jsonwebtoken documentation: https://www.npmjs.com/package/jsonwebtoken

En conformité avec les directives sécuritaires, mentionnées page 25, pour contrer les attaques XSS (Cross Site Scripting), les JWT sont stockés dans un cookie HTTPOnly. Cela fait en sorte que le navigateur ne peut que recevoir et envoyer le cookie, mais ne peut en aucun cas y accéder ou le modifier.

Afin de pouvoir accéder aux cookies, le middleware « cookie-parser » est nécessaire. Ce package analyse tous les cookies de la demande entrante et nous permet de les accéder.

Cette procédure d'autorisation est associée à l'utilisation d'un middleware personnalisé nommé "protect" présent dans le authController. Ce middleware, positionné sur les routes, avant le contrôleur menant à une action de redirection de page, vérifie si l'utilisateur existe et s'il possède un token. Si les deux résultats à ces deux questions sont vrais, l'accès est accordé en laissant l'objet req / res continuer dans le middleware stack.
Si ce n'est pas le cas, une erreur est renvoyée, renvoyant un message à l'utilisateur l'informant de l'action à entreprendre.

Création du JWT et fonction de login *authController.js*

```
/* SIGN TOKEN ////////////////////// */
const signToken = id => { return jwt.sign({ id }, process.env.JWT_SECRET, { expiresIn: process.env.
JWT_EXPIRES_IN }); };

/* EXPORT ////////////////////// */
export default {
  async login(req, res, next) {
    // 1. SELECT / CHECK REQUIRED DATA WAS PROVIDED
    const { email, password } = req.body;
    if (!email || !password) return next(new AppError('Please provide email and password', 400));
    // 2. CHECK IF USER EXISTS AND PASSWORD IS CORRECT
    const query = User.findOne({ email }).select('+password'); // '+' used to select specific field
    const result = await query;
    // if no result, throw error
    if (!result || !(await result.correctPassword(password, result.password))) return next(new AppError
    ('Incorrect email or password', 401));
    // 3. CREATE TOKEN
    const token = signToken(result._id);
    // 4. CONFIGURE COOKIE & OPTIONS
    const cookieOptions = {
      expires: new Date(Date.now() + process.env.JWT_COOKIE_EXPIRES_IN * 24 * 60 * 60 * 1000),
      httpOnly: true, // cookie cannot be accessed or modified in any way by the browser (XSS protection)
    };
    if (process.env.NODE_ENV === 'production') cookieOptions.secure = true;
    // 5. SEND COOKIE
    res.cookie('jwt', token, cookieOptions);
    // 6. SEND RESPONSE
    res.status(200).json({
      status: 'success',
      token,
    });
  },
```

Fonction protect, gestionnant l'accès aux routes *authController.js*

```
async protect(req, res, next) {
  // 1. CHECK IF TOKEN EXISTS (located in header). IF YES, CREATE TOKEN VARIABLE
  let token;
  // check if token is provided in header or cookie (cookie is used for browser requests)
  if ( req.headers.authorization && req.headers.authorization.startsWith('Bearer') ) {
    token = req.headers.authorization.split(' ')[1];
  } else if (req.cookies.jwt) {
    token = req.cookies.jwt;
  }
  if (!token) return next(new AppError('You are not logged in! Please log in to get access.', 401));
  // 2. VERIFY TOKEN
  const decoded = await promisify(jwt.verify)(token, process.env.JWT_SECRET);
  // if verification fails, errors possible are:
  // - JsonWebTokenError (token invalid)
  // - TokenExpiredError (token expired)
  // 3. CHECK IF USER STILL EXISTS
  const query = User.findById(decoded.id)
  const result = await query;
  if (!result) return next(new AppError('The user belonging to this token does no longer exist.', 401));
  // 4. GRANT ACCESS TO PROTECTED ROUTE
  req.user = result; // add user to request object
  next();
},
```

Mise en place du protect middleware positionné sur une route sélectionnée

viewRoutes.js

```
/* ROUTE ////////////////////// */
router.route('/').get(cw(viewController.getPageHome));
router.route('/beers').get(cw(viewController.getPageAllBeers));
router.route('/beers/:slug').get(cw(viewController.getPageBeer));
router.route('/contact').get(cw(viewController.getPageContact));

router
  .route('/login')
  .get(cw(viewController.getLoginForm))
  .post(cw(viewController.postPageLogin));

router
  .route('/admin')
  .get(
    cw(authController.protect),
    cw(adminController.getAdminPage));

router
  .route('/admin/create')
  .get(
    cw(authController.protect),
    cw(adminController.getAdminFormPage));

router
  .route('/admin/:slug')
  .get(
    cw(authController.protect),
    cw(adminController.getAdminFormPageUpdate))
  .delete(cw(adminController.deleteBeer));

/* EXPORT ////////////////////// */
export default router;
```

Interaction front / back-end (AJAX)

Afin de permettre à l'administrateur de créer, mettre à jour et/ou supprimer des bières, une communication front/back-end était nécessaire. Cela a été fait en faisant des appels AJAX à l'API.

Pour cela, j'ai développé une fonction personnalisée nommée fetchCrud, visible page suivante, facilitant l'utilisation de la méthode native fetch().

admin-update.js

```
/* ELEMENTS - (els) ////////////////////////////////////// */
const formEl = document.querySelector('form');

/* FUNCTIONS - (fx) ////////////////////////////////////// */
const updateBeer = async beer => {
  try {
    const slug = window.location.href.match(/[^/]+$/)[0];
    const res = await fetchCrud(`/api/v1/beers/${slug}`, 'PATCH', beer);
    // if not success return early and do nothing
    if (res.status !== 'success') return;
    // if success redirect to admin page after 1 second
    removeError();
    displaySuccess('Beer updated successfully!');
    setTimeout(() => window.location.replace('/admin'), 1000);
  } catch (err) {
    // if error, log it to the console
    console.error(err.message);
    removeError();
    displayError(err.message);
  }
};

/* EVENT HANDLERS - (eh) ////////////////////////////////// */
const handleSubmit = e => {
  e.preventDefault();
  // using optional chaining (?.) to avoid errors if the element is not found
  const form = new FormData();
  form.append('name', e.target.name?.value);
  form.append('slug', slug);
  form.append('description', e.target.description?.value);
  form.append('style', e.target.style?.value);
  form.append('ebv', e.target.ebv?.value);
  form.append('abv', e.target.abv?.value);
  form.append('ibu', e.target.ibu?.value);
  if (e.target.imageCover?.files[0]) form.append('imageCover', e.target.imageCover?.files[0]);
  if (document.querySelectorAll('#images')) {
    const images = document.querySelectorAll('#images');
    images.forEach(image => {  if (image.files[0]) form.append('images', image.files[0]) } );
  }
  updateBeer(form);
};

formEl.addEventListener('submit', handleSubmit);
```

La fonction personnalisée fetchCrud.js, applique conditionnellement les champs requis en correspondance avec la méthode HTTP soumise ainsi que le type de body.

fetchCrud permet à l'utilisateur de soumettre directement l'url, la méthode et le corps en arguments sans avoir à configurer des informations supplémentaires concernant les headers ou le body.

```
/* DATA - (dt) ////////////////////////////////////////// */
const REQUEST_TIMEOUT_SEC = 5;

/* FUNCTIONS - (fx) ///////////////////////////////////////// */
// timeout function to reject the promise after sec seconds with an error message
const timeout = sec => new Promise((_, reject) => {
  setTimeout( () => reject(Error(`method timed out. Please try again later...`)), sec * 1000 );
});

// fetchCrud function which accepts a 3 parameters (url, body, method)
// use for all CRUD operations. If 'GET' or 'DELETE' method, the body and headers are not required
// check if body contains instance of FormData(). If so then set the Content-Type header to 'undefined'
otherwise set the Content-Type header to 'application/json'
const fetchCrud = async (url, method, body) => {
  body = body instanceof FormData ? body : JSON.stringify(body);
  const headers = body instanceof FormData
  ? undefined
  : { 'Content-Type': 'application/json' };

  try {
    const req = body
      ? fetch(url, { method, headers: headers, body, })
      : fetch(url, { method });
    const res = await Promise.race([req, timeout(REQUEST_TIMEOUT_SEC)]);
    // since deleteBeer is a DELETE method, there is no body to parse as json so we can skip this step for
    DELETE methods and just return the response object (res) to the calling function (deleteBeer) to handle
    the response object (res) there instead of here in fetchCrud function (see deleteBeer function)
    if (method === 'DELETE') return res;
    const data = await res.json();
    if (!res.ok) throw Error(data.message);
    return data;
  } catch (err) {
    if (err.message === 'Failed to fetch')
      err.message = `Unable to reach the server. Please check your internet connection...`;
    throw err;
  }
};
```

PRÉSENTATION DU JEU D'ESSAI

J'ai consacré une attention particulière au traitement des erreurs. Comme mentionné dans la section 'Création du gestionnaire d'erreur global' (page 36), les erreurs sont séparées en deux types ; les erreurs opérationnelles et celles de programmation.

Il est tout particulièrement nécessaire que le site informe l'utilisateur, quand des erreurs opérationnelles surviennent, comment agir. Pour cela, il est nécessaire de comprendre au préalable quelles erreurs potentielles peuvent survenir et de mettre en œuvre les réponses appropriées.

Pour l'administrateur créant ou mettant à jour une bière, les données saisies doivent répondre à certaines conditions.

Selon l'input, la saisie des données peut être requise ou non. Elle peut nécessiter un type de données spécifique. Potentiellement accepter uniquement des lettres ou des chiffres, éventuellement les deux. Pour certains champs, des caractères spéciaux peuvent aussi être acceptés. Certains exigent une longueur maximale ou minimale, etc.

Pour assurer le bon fonctionnement de cette fonctionnalité, j'ai testé la fonctionnalité en insérant des entrées correspondant à toutes les possibilités auxquelles je pouvais penser.

```
const beerSchema = new Schema({
  name: {
    type: String,
    required: [true, 'A beer must have a name'],
    unique: true,
    trim: true,
    minlength: [3, 'A beer name must have more or equal than 3 characters'],
    maxlength: [20, 'A beer name must have less or equal than 20 characters'],
    validate: {
      validator: (val) -> {
        // regex accepting french characters
        const regex = /^[a-zA-Z0-9À-ÿ\s-]+$/;
        return regex.test(val);
      },
      message: 'A beer name must contain only letters and numbers',
    },
  },
  slug: String,
  description: {
    type: String,
    required: [true, 'A beer must have a description'],
    trim: true,
    minlength: [10, 'A beer description must have more or equal than 10 characters'],
    maxlength: [1000, 'A beer description must have less or equal than 1000 characters'],
    validate: {
      validator: (val) => {
        // regex accepting french characters, '«»', ''', '-' and '.'
        const regex = /^[a-zA-Z0-9À-ÿ\s-«»''-.]+$/;
        return regex.test(val);
      },
      message: 'A description name must contain only letters and numbers',
    },
  },
  style: {
    type: String,
    required: [true, 'A beer must have a style'],
    trim: true,
    minlength: [3, 'A beer style must have more or equal than 3 characters'],
    maxlength: [20, 'A beer style must have less or equal than 20 characters'],
    validate: {
      validator: (val) => {
        // regex accepting only letters and french characters, ''' and '.'
        const regex = /^[a-zA-ZÀ-ÿ\s-''.]+$/;
        console.log('style validator:', regex.test(val));
        return regex.test(val);
      },
      message: 'A beer style must contain only letters',
    },
  },
  malts: {
    required: [true, 'A beer must have a malt'],
    type: Array,
    trim: true,
    validate: {
      validator: (val) -> {
```

Tableau des conditions à remplir ci-dessous

Value	Required	Type	Caractéristiques	Length		Comment
				min	max	
name	True	String	Alphanumeric values, including french accents, '-', ’' and '"	3	20	
slug	auto-generated	String	Auto-generated from name			auto-generated
description	True	String	Alphanumeric values, including french accents, '«»', ’', '-' and '.'	50	500	
style	True	String	Alphabetical values, including french accents, ’' and '.'	3	20	
malts	True	Array	Alphabetical values, including french accents			
hops	True	Array	Alphabetical values, including french accents			
ebv	True	Number	Numerical values, including ',' and '.'			
abv	True	Number	Numerical values, including ',' and '.'			
ibu	True	Number	Numerical values, including ',' and '.'			
cover image	True	String	Auto-generated from name + ‘cover’ + date.now()			auto-generated
2nd image		String	Auto-generated from name + ‘1’ + date.now()			auto-generated
3rd image		String	Auto-generated from name + ‘2’ + date.now()			auto-generated

Input	Données saisies	Retour attendu	Retour obtenu	Ecart
name		A beer must have a name	A beer must have a name	CORRECT
	t	A beer name must have more or equal than 3 characters	A beer name must have more or equal than 3 characters	CORRECT
	test!	A beer name must contain only letters and numbers	A beer name must contain only letters and numbers	CORRECT
	test & test	A beer name must contain only letters and numbers	A beer name must contain only letters and numbers	CORRECT
	this is a valid test - but has too many characters	A beer name must have less or equal than 20 characters	A beer name must have less or equal than 20 characters	CORRECT
	this is a valid test			CORRECT
description		A beer must have a description	A beer must have a description	CORRECT
	less than fifty characters	description must have more or equal than 50 characters	description must have more or equal than 50 characters	CORRECT
	less than fifty characters with a weird character !&#	A description must have more or equal than 50 characters	A description must contain only letters and numbers	INVESTIGATE
	>500 chars	A beer description must have less or	A beer description must have less or	CORRECT

		equal than 500 characters	equal than 500 characters	
	Session Pale Ale « légère » et rafraîchissante aux subtiles notes d'abricot et d'agrumes.			CORRECT
style		A beer must have a style	A beer must have a style	CORRECT
	ip	A beer style must have more or equal than 3 characters	A beer style must have more or equal than 3 characters	CORRECT
	style which is too long	A beer style must have less or equal than 20 characters	A beer style must have less or equal than 20 characters	CORRECT
	ipa#	A beer style must contain only letters	A beer style must contain only letters	CORRECT
	ipa2	A beer style must contain only letters	A beer style must contain only letters	CORRECT
	valid test			
image cover		A beer must have a cover image	A beer must have a cover image	CORRECT
	mimetype !== image	Not an image! please upload only images	Not an image! please upload only images	CORRECT
	.png			CORRECT

DESCRIPTION DE LA VEILLE

Description de la veille : constante.

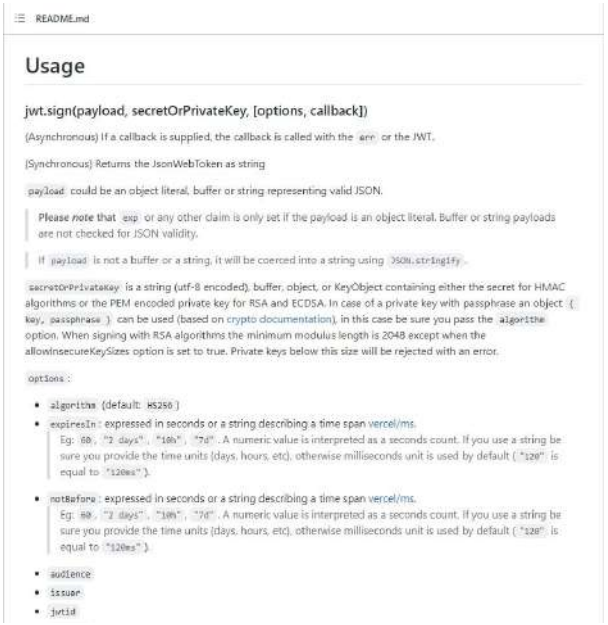
Tout au long du projet, j'ai eu besoin d'effectuer des recherches. Qu'il s'agisse de lire de la documentation sur le type de header requis pour quels appels AJAX, des explications sur les messages d'erreur ou sur la manière d'implémenter les JWT, je recherche en permanence des informations relatives au développement.

Un exemple de recherche que j'ai faite pour ce projet était de comprendre les JWTs. Leur structure, comment y intégrer des données et les utiliser à des fins d'authentification.

Comme mentionné dans le chapitre Authentification et autorisation (page 40), la création du jeton s'est faite en utilisant la fonction sign() provenant du package npm jsonwebtoken.

Cette fonction, comme on peut le voir dans la page d'informations du package, a même le site Web npm, prend trois paramètres.

- payload
- secretOrPrivateKey
- [options, callback]


README.md

Usage

jwt.sign(payload, secretOrPrivateKey, [options, callback])

(Asynchronous) If a callback is supplied, the callback is called with the `err` or the JWT.

(Synchronous) Returns the JsonWebToken as string

`payload` could be an object literal, buffer or string representing valid JSON.

> Please *note* that `exp` or any other claim is only set if the payload is an object literal. Buffer or string payloads are not checked for JSON validity.

> If `payload` is not a buffer or a string, it will be coerced into a string using `JSON.stringify`.

`secretOrPrivateKey` is a string (utf-8 encoded), buffer, object, or KeyObject containing either the secret for HMAC algorithms or the PEM encoded private key for RSA and ECDSA. In case of a private key with passphrase an object `{ key, passphrase }` can be used (based on crypto documentation), in this case be sure you pass the `algorithm` option. When signing with RSA algorithms the minimum modulus length is 2048 except when the allowInsecureKeySizes option is set to true. Private keys below this size will be rejected with an error.

`options`:

- `algorithm` (default: `HS256`)
- `expiresIn`: expressed in seconds or a string describing a time span vercel/ms.
  > Eg: `60`, `"2 days"`, `"10h"`, `"7d"`. A numeric value is interpreted as a seconds count. If you use a string be sure you provide the time units (days, hours, etc), otherwise milliseconds unit is used by default (`"120"` is equal to `"120ms"`).
- `notBefore`: expressed in seconds or a string describing a time span vercel/ms.
  > Eg: `60`, `"2 days"`, `"10h"`, `"7d"`. A numeric value is interpreted as a seconds count. If you use a string be sure you provide the time units (days, hours, etc), otherwise milliseconds unit is used by default (`"120"` is equal to `"120ms"`).
- `audience`
- `issuer`
- `jwtid`

C'est à même le payload que nous pouvons inclure des données qui peuvent être utilisées à des fins d'authentification. Dans la signature du jeton utilisé dans ce projet, c'est l'identifiant de l'utilisateur qui est passé.

Dans le champ secretOrPrivateKey est passé un secret tenu côté serveur. Ceci est utilisé, en combinaison avec la formule de Auth0, créateur des JWT, qu'est généré le token.
Additionnellement, la durée de vie du jeton peut être définie dans le paramètre options.

Ceci est ensuite utilisé en combinaison avec la fonction de décryptage verify(), qui permet au processus de confirmer son authenticité.
https://github.com/auth0/node-jsonwebtoken

DESCRIPTION D'UNE SITUATION DE TRAVAIL

L'erreur sur laquelle j'ai le plus bloqué était due à une erreur que j'ai moi-même créé… Comme mentionné dans le chapitre Interaction front / back-end (AJAX) (page 44), j'ai créé une fonction personnalisée pour aider au processus d'appels AJAX.

Cette fonction permet de soumettre directement l'url, la méthode HTTP et le body en arguments pour conditionnellement placer les headers en fonction de la méthode et du body.

J'ai lu que lors du passage d'un formulaire, devenu nécessaire afin de permettre le transfert d'une image, à même la fonction fetch(), il fallait définir les headers à { 'Content-Type': 'multipart/form-data' }. J'ai donc utilisé un opérateur ternaire pour définir les en-têtes en fonction de si le corps était une instance de FormData ou pas.

```
const headers = body instanceof FormData
   ? { 'Content-Type': 'multipart/form-data' }
   : { 'Content-Type': 'application/json' };
```

Dès lors, lors de l'utilisation de Postman, la fonction fetchCrud() fonctionnait. J'ai pu transmettre les données du formulaire, les images, sans problème. Mais, une fois à même le site Web, la fonctionnalité ne fonctionnait pas. Ceci m'a pris un temps certain à dénouer. Le message d'erreur reçu, donc à même le site, était 'MULTIPART: BOUNDARY NOT FOUND'.

Lors de ma recherche pour une réponse, sur des sites Web, j'ai lu que des personnes rencontraient des problèmes dû à leur version d'Axios, des problèmes de compatibilités entre différents outils, des situations qui n'étaient pas tout comme la mienne.

J'ai donc persisté et, après un temps certain à essayer de trouver et résoudre le problème à même le code, je suis finalement tombé sur un article posté dans Stack Overflow indiquant qu'en fait la requête pouvait elle-même comprendre ces besoins et définir son propre statut. Une information qui allait à l'encontre de ce que j'avais compris initialement, d'où l'idée qui ne m'a même pas traversé l'esprit.

J'ai donc défini l'en-tête à undefined, et cela fonctionnait.

Les méthodes employées pour rechercher des résultats dans le navigateur étaient généralement une copie du message d'erreur. De plus, je filtre les résultats par date, en visant des résultats dans la dernière année.

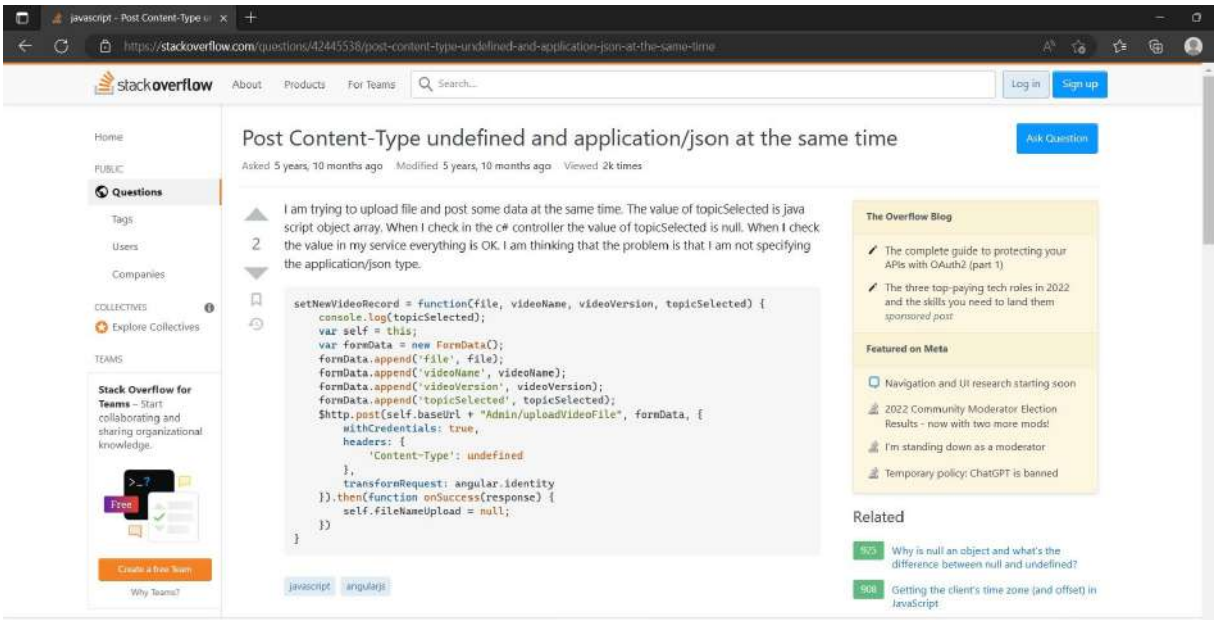
Alternativement, si un résultat ne correspond pas exactement, je recherche un terme coïncidant avec le sujet, tel que “setting headers for form data ajax request”.

95% de toutes mes recherches se font en anglais. L'anglais étant ma première langue aide beaucoup.

Je joins l'extrait de l'article me permettant de résoudre ce problème à la page suivante (page 53).

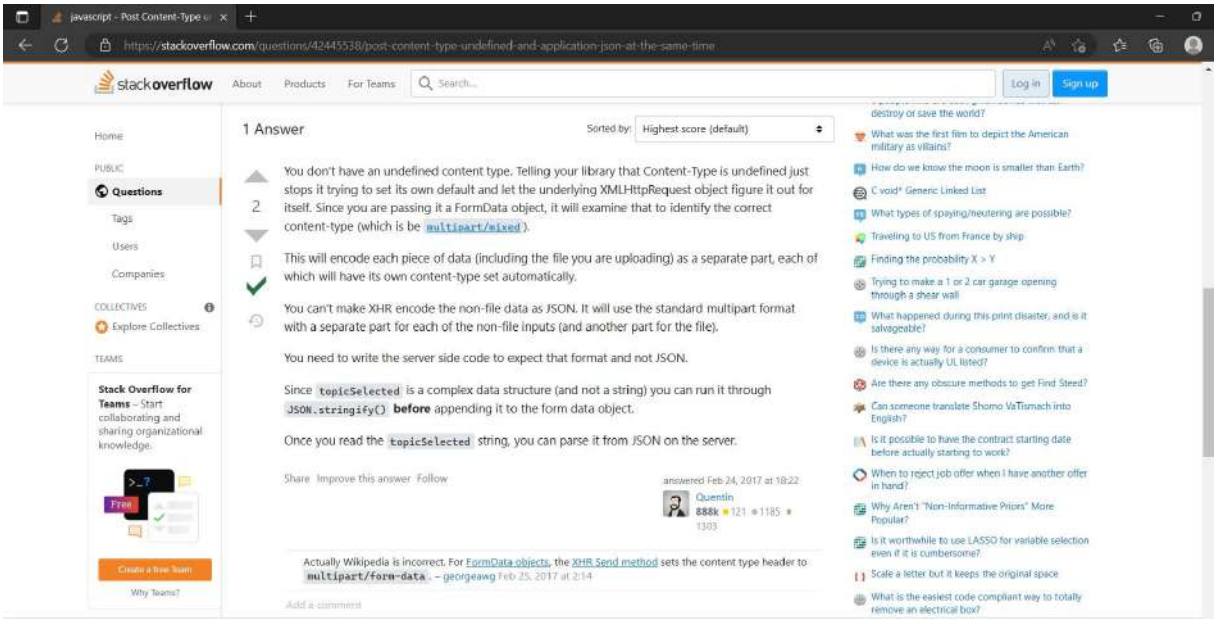
EXTRAIT DU SITE ANGLOPHONE

Ceci est la question de l'article qui m'a aidé à résoudre mon problème.
https://stackoverflow.com/questions/71609670/adding-headers-to-ajax-request-fails



L'auteur de la question mentionne qu'il essaie de télécharger un fichier et des données en même temps. Il indique qu'il rencontre des problèmes de transfert de données et pense que le problème est dû au fait qu'il ne spécifie pas le type de contenu d'en-tête.

Capture d'écran de la réponse m'ayant aidé à résoudre mon problème.



L'auteur de la réponse indique qu'en fait le type d'en-tête n'est pas indéfini. Il informe que le fait d'indiquer le type de contenu à “undefined” empêche la librairie de définir sa propre valeur par défaut. Il indique que l’objet XMLHttpRequest peut comprendre par lui-même la valeur qu’il lui est nécessaire.

```
// fetchCrud function which accepts a 3 parameters (url, body, method)
// use for all CRUD operations. If 'GET' or 'DELETE' method, the body and headers are not required
// check if body contains instance of FormData(). If so then set the Content-Type header to
'multipart/form-data' otherwise set the Content-Type header to 'application/json'
const fetchCrud = async (url, method, body) => {

  body = body instanceof FormData ? body : JSON.stringify(body);
  const headers = body instanceof FormData
    ? { 'Content-Type': 'multipart/form-data' }
    : { 'Content-Type': 'application/json' };

  try {
    const req = body
      ? fetch(url, { method, headers: headers, body, })
      : fetch(url, { method });
    const res = await Promise.race([req, timeout(REQUEST_TIMEOUT_SEC)]);
    // since deleteBeer is a DELETE method, there is no body to parse as json so we can skip this
    step for DELETE methods and just return the response object (res) to the calling function
    (deleteBeer) to handle the response object (res) there instead of here in fetchCrud function
    (see deleteBeer function)
    if (method === 'DELETE') return res;
    const data = await res.json();
    if (!res.ok) throw Error(data.message);
    return data;
  } catch (err) {
    if (err.message === 'Failed to fetch')
      err.message = `Unable to reach the server. Please check your internet connection...`;
    throw err;
  }
};
```

Version originale de ma fonction fetchCrud - code erroné

De plus, il indique qu’en transmettant un objet FormData, la requête examinera l’identité de la valeur et définira son content-type en concordance.

Grâce à ces informations, j'ai pu comprendre que tout ce que j'avais à faire était de laisser la requête de récupération définir elle-même son type de contenu d'en-tête.

```
// fetchCrud function which accepts a 3 parameters (url, body, method)
// use for all CRUD operations. If 'GET' or 'DELETE' method, the body and headers are not required
// check if body contains instance of FormData(). If so then set the Content-Type header to
'undefined' otherwise set the Content-Type header to 'application/json'
const fetchCrud = async (url, method, body) => {
  body = body instanceof FormData ? body : JSON.stringify(body);
  const headers = body instanceof FormData
  ? undefined
  : { 'Content-Type': 'application/json' };

  try {
    const req = body
      ? fetch(url, { method, headers: headers, body, })
      : fetch(url, { method });
    const res = await Promise.race([req, timeout(REQUEST_TIMEOUT_SEC)]);
    // since deleteBeer is a DELETE method, there is no body to parse as json so we can skip this
    step for DELETE methods and just return the response object (res) to the calling function
    (deleteBeer) to handle the response object (res) there instead of here in fetchCrud function
    (see deleteBeer function)
    if (method === 'DELETE') return res;
    const data = await res.json();
    if (!res.ok) throw Error(data.message);
    return data;
  } catch (err) {
    if (err.message === 'Failed to fetch')
      err.message = `Unable to reach the server. Please check your internet connection...`;
    throw err;
  }
};
```

Version modifiée

CONCLUSION

Cette situation a été très enrichissante pour moi. Le temps de ce projet m'a permis de pratiquer de nombreuses facettes du développement web et de monter en compétences.

Le temps passé ces derniers mois, en plus de travailler sur ce projet, a encore accru mon désir de développer et d'approfondir mes connaissances ainsi que de poursuivre mon orientation dans le secteur du développement web.

BACK TO TOP