**Présentation personnelle, parcours :**

Suite à l’obtention de mon BTS assistant de gestion en 2017, j’ai décidé d’entamer une reconversion dans le domaine du numérique. Pour cela à côté d’un apprentissage constant en autodidacte sur des technologies liées au domaine du numérique. J’ai débuté par un stage en tant que responsable web marketing dans entreprise spécialisé dans la production d’objets publicitaire personnalisés. Ensuite j’ai souhaité entamer une formation de chef de projet digital en alternance, sans trouver d’alternance. C’est lorsque j’avais débuté une formation en ligne via Openclassroom que j’ai alors compris que ces 2 années de recherches étaient centrées sur le numérique et que c’est le développement web qui attisait le plus ma curiosité. C’est ainsi que j’ai rejoint l’organisme Simplon en intégrant cette formation intensive de développeur web. Simplon met l’avant sur la pédagogie active, c’est via des projets essentiellement pratique réalisé aussi bien groupe ou en autonomie qu’on développait nos compétences. Un mélange efficace entre autodidacte et partage constant de connaissance, l’entraide au sein de cette promotion était la pierre angulaire de la formation.

**Présentation de Simplon :**

**Résumé du projet :**

UniArts est un projet né bien avant le début de la formation, sa source provient de ma volonté d’offrir plus de visibilités aux artistes amateurs en mettant un place un collectif qu’ils pourraient rejoindre comme bon leur semble. Ils disposeront d’un back-office permettant de présenter leurs projets, publier des liens vers les plateformes on peut suivre leurs activités. En tant qu’admin je réaliserais des articles présentant les artistes ayant rejoint le collectif.

**Etude concurrentiel :**

La naissance de ce projet n’a pas été alimenté à la suite d’une étude concurrentiel approfondie. Néanmoins j’ai beaucoup apprécié la mise en place du réseaux social collabortive pour des artistes du mon entier du nom de « Daisie » mis en place par l’actrice Maisie Williams

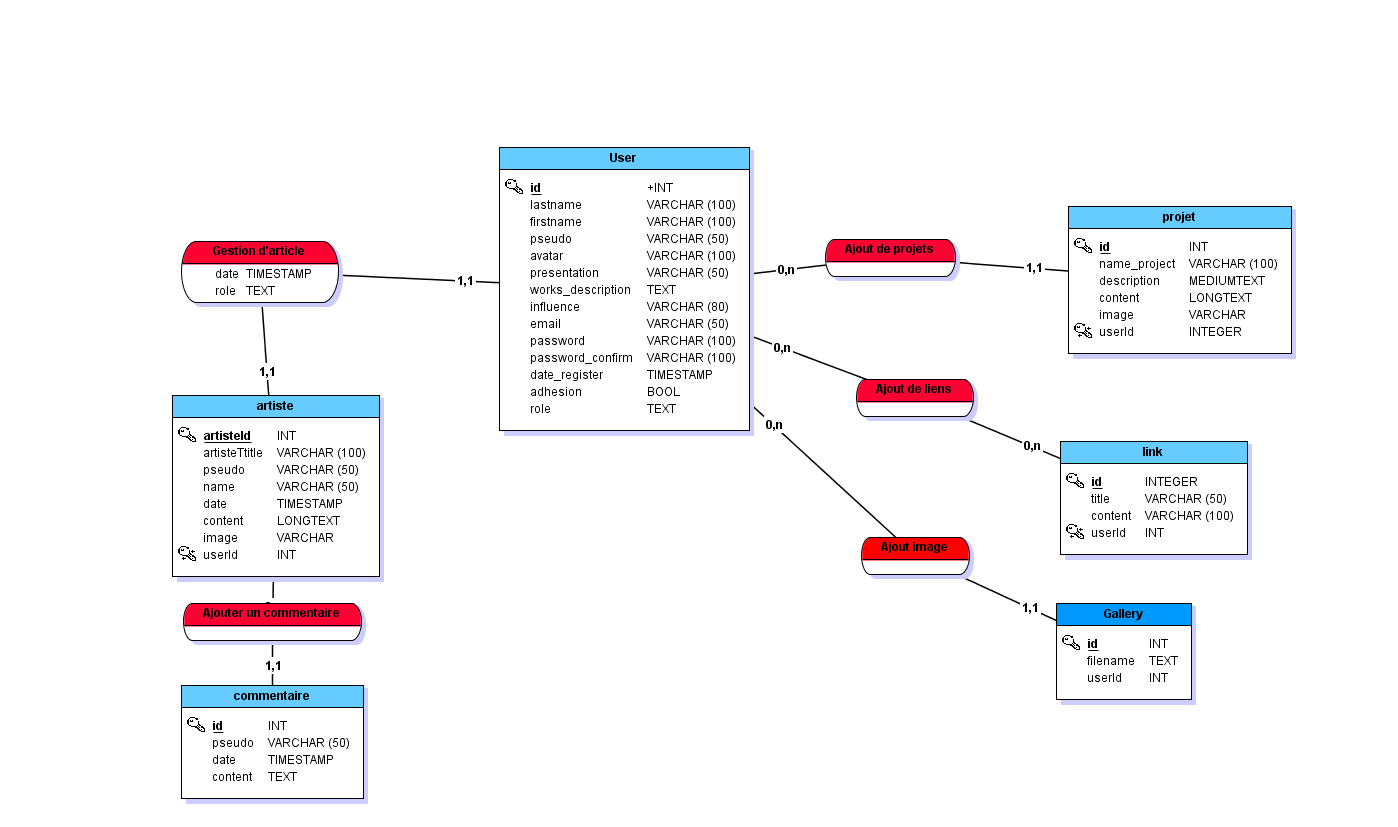
**Présentation des spécificités :**

**Les uses cases** :

* **Les artistes (users)** : Les artistes pourront au sein du système gérer leurs projets (add, update et delete), ajouter des liens externes pour permettre aux visiteurs de savoir où se diriger pour les soutenir ou en découvrir davantage sur ce qu’ils font. Pour cela, ils ont besoin d’adhérer au collectif en s’inscrivant via le formulaire d’inscription, l’inscription sera d’abord vérifiée par l’admin.

* **L’administrateur (users)**: l’admin pourra réaliser les articles de présentations des artistes et les gérer (add/update/delete), sur un back office il gérera également les demandes d’adhésions des artistes au collectif.
* **Les visiteurs** : les visiteurs pourront lire les articles de présentation et également les commenter, il l’en est de même pour les projets des artistes.

**Le mcd :**



Il s’agit d’une représentation d’une base de données facilement compréhensible du fonctionnement du système de base de données.

J’ai défini dans un premier temps une table Users, la différence entre l’administrateur et l’user classique qui représente l’artiste se situe au niveau du champs rôle. Le champs rôle contient deux valeurs : « Admin » et « Artistes », selon le rôle qu’aura l’user la redirection ne serait pas identique.

* Un user ayant le rôle « Admin » pour gérer la création/la modification/la suppression des articles de présentation des artistes. Il gère également les demandes d’adhésions au collectif.

* L’user qui aura le rôle « artiste » accèdera au back-office lui permettant de gérer la création/la modification/la suppression de projets. Il a également la possibilité d’ajouter des liens externes.

**Les wireframes & les maquettes :**

Pour la réalisation de ce projet j’ai adopté la méthode mobile first en procédant à la réalisation, en priorisant sur l’aspect mobile du projet j’ai conçu les wireframes de la version mobile/tablette et dekstop de la page home, le back-office du user et ainsi que son profil. Pour la réalisation des maquettes j’ai utilisé le logiciel justimind pour le grand nombre de fonctionnalité qu’il propose et son accessibilité très intuitive. Il en est de même pour les maquettes.

**La charte graphique** :

**La typographie**: Comme pour l’ensemble du design du site web, j’ai souhaité rester sur des polices sobres mais permettant tout de même d’attirer l’œil du visiteur.

* **LES TITRES**: les titres de type h1 ou h2 auront la font Teko, une police à la structure sobre se mariant correctement aux titre.
* **LES PARAGRAPHE**: pour garder cette synergie, la police font est utilisé pour les paragraphes.
* **LES MENU & BUTTONS :** les menus avec la police **source sans pro.**

**Les couleurs :**

Les deux couleurs principales ont été choisi pour garder cette cohérence dans la sobriété du site, la dernière couleur a été choisi afin de faire ressortir certains éléments du site pour les rendre plus visibles (tels que le menu, etc)

**Descriptif technique du projet :**

**Méthode et organisation de travail :**

Pour la réalisation de ce projet au vu de la charge considérable de la mise en place de celui-ci, l’utilisation de nombreux supports ont été primordial quant à la manière d’appréhender ce projet. Dans un premier temps j’ai procédé à la mise en place d’un Trello dont les colonnes sont organisées de la manière suivante :

* Backlog : Elle contient l’ensemble de mes tâches
* A faire : Elle contient mes tâches à réaliser
* En cours : Elle concerne mes tâches en courts
* Bug/à revoir : Elle concerne mes bugs en termes de code ou l’amélioration des spécificités du projet

J’ai défini un code couleur permettant d’établir des ordres de priorités sur les tâches, généralement tâche concerne une fonctionnalité.

En plus de cela, afin de garder un suivi constant de l’évolution du projet, je versionnais régulièrement sur Ghithub.

Au court de la conception du projet, j’éditais en parallèle l’aspect back et front du site web. Lorsqu’une fonctionnalité était opérationnelle côté serveur à la suite de plusieurs tests via potsman, je procédais à la mise en place du front de cette même fonctionnalité.

**Environnement de développement :**

**Technologies utilisées :**

**Implémentation des fonctionnalités primordiales du site :**

**Le Back :**

**Mise en écoute du serveur :**

**Middleware :**

* **Express :**
* **Bodyparser :**
* **Mongoose :**
* **Cors :**
* **BaererToken :**
* **Dotenv :**
* **Multer :**
* **Path :**

**Type de bdd :**

Pour la réalisation de ce projet j’ai opté pour l’utilisation de base de données noSql de type document. Mon choix s’est porté sur ce type de données pour sa flexibilité, car elle me permet facilement de m’adapter à différents cas possibles d’utilisation de mon site web, ainsi que pour sa performance.

**Les models/schémas :** Grâce à l’utilisation de mongoose, cette dépendence/middleware facilite le traitement des données de type noSQL, notamment avec mongodb. Il permet dans le cas présent la mise en place de schémas/models qui permet de faciliter la gestion de base de données qu’on souhaite recevoir. Ainsi j’ai pu mettre en place différents models (user, projects, links, articles, comments…)

**Les contrôleurs :** on insère dans les fichiers contrôleur les function call-back des routes où on effectue les requêtes. J’ai adopté cette méthode afin d’alléger et découper mon code au niveau du serveur qui commençait à gagner en volume et donc manquer en visibilité. Ces contrôleurs effectuent donc la logique des routes définis en appelant les méthodes définis dans le fichier contrôleur.

**Contrôleurs Auth :**

Le contrôleur Auth contient la logique d’inscription et de connexions de mes routes **register/login.**  Le contrôleur est passé en fonction callback de deux routes respectives afin « d’importer » les méthodes que j’ai conçu pour la connexion et l’inscription. Afin de sécuriser la phase de connexion, j’ai procédé à la méthode la jwt (json web token), pour cela dans un premier temps j’ai créé un fichier caché qui contient mes variables d’environnement, dans le cas présent la clé secrète utile à l’utilisation de token. L’utilisation de package **dotenv** va permettre de trouver ce fichier caché, pour pouvoir ensuite grâce au **process.env** récupérer ma variable d’environnement.

**Contrôleur User :**

Le contrôleur User contient l’ensemble des méthode liés à l’utilisateur(l’artiste), Le contrôleur contient la méthode getById permettant de retourner toutes les informations de l’utilisateur.

**Contrôleur Project :** Le contrôleur Project contient toute la logique de la gestion de projet de l’utilisateur. Passé en fonction callback de l’ensemble des routes spécifiques à la gestion de projet, j’importe dans chacune d’entre elles les différentes méthodes permettant à l’user de créer ses projets. Mettant en place ainsi un CRUD complet

**Contrôleur Link :** Le contrôleur Project contient toute la logique de la gestion de projet des généré par l’utilisateur.

**Contrôleur Profile-User :** Le contrôleur ProfileUser contient les méthodes liées à la gestion du profil, plus particulièrement la gestion d’avatar avec la fonctionnalité d’ulpoad des images et de tous les autres champs liés au profile de l’utilisateur.

**LES ROUTES :**

**user/register :**

* Cette route permet aux utilisateurs (artistes) de procéder à l’adhésion au collectif en remplissant ce formulaire d’inscription. J’ai mis en place la gestion des cas d’erreurs au sein du contrôleur.
* Les cas d’erreurs :

**user/login :**

* Cette route permet à l’utilisation de s’authentifier à la suite de son inscription. Lors de la connexion, un jeton sera généré à l’aide de jwt. Celui-ci contiendra l’id du user, son rôle ainsi que la clé secrète qui est situé dans un fichier caché récupérable grâce à module dotenv.

**BACK-OFFICE(USER):**

**Gestion de projets:**

* **user/addProject:** La route addProject permet à l’utilisateur de créer/poster des projets. Je procède à la mise en place de la méthode **jwt.verify** afin de vérifier l’existence du token, dans le cas où un token n’est pas reconnu le serveur génère **un status 401**. Dans le cas contraire, je procède à l’instance du projet. Lorsqu’un projet est créé grâce à la méthode save de mongodb, si le rôle « Artiste » contenu dans le token est bien reconnu j’update par la même occasion l’User en ajoutant au champs projet l’id du projet nouvellement créé. Ainsi, dès la création d’un projet celui-ci sera automatiquement lié à l’utilisateur qu’il l’aura créé.
* **user/getAllProjects:** getAllProjects a pour but de retourner l’ensemble des projets liés à un utilisateur, l’objectif étant de pouvoir les afficher côté client pour qu’il soit visible pour tout les utilisateur du site.
* **user/updateProject:** Cette route permet à l’utilisateur de modifier son projet grâce à la métode updateOne de mongodb, en spécifiant dans le set les champs que l’utilisateur pourra modifier.
* **user/deleteProject:**  La route deleteProject permet à l’utilisateur de supprimer ses projets.

**Gestion des liens:**

* **user/addLink:** La route addLink permet à l’utilisateur de créer/poster des liens. Tout comme la méthode addProjet, je procède à la mise en place de la méthode **jwt.verify** afin de vérifier l’existence du token, dans le cas où un token n’est pas reconnu le serveur génère **un status 401**. Dans le cas contraire, je procède à l’instance d’un lien Lorsqu’un lien est créé grâce à la méthode save de mongodb, si le rôle « Artiste » contenu dans le token est bien reconnu j’update par la même occasion l’User en ajoutant au champs projet l’id du lien nouvellement créé. Ainsi, dès la création d’un lien celui-ci sera automatiquement lié à l’utilisateur qu’il l’aura créé.
* **user/updateLink:** Cette route permet à l’utilisateur de modifier les liens que l’utilisateur aura créé grâce à la métode updateOne de mongodb, en spécifiant dans le set les champs que l’utilisateur pourra modifier.
* **user/getAllLinks:** getAllLinks a pour but de retourner l’ensemble des liens liés à un utilisateur, l’objectif étant de pouvoir les afficher côté client pour qu’il soit visible pour tous les utilisateur du site.
* **user/deleteLink:** La route deleteLink permet à l’utilisateur de supprimer ses liens.

**Gestion de profil :**

* **user/updateAvatar :** Cette route permet au user de mettre à jour le champs avatar spécifié dans le model. En effet, lors de la phase d’inscription définis dans le contrôleur auth, le champs avatar ne fait pas partie des champs requis pour procéder à l’inscription.

Il faut donc à la suite de l’inscription du user, update celui-ci en mettant à jour le champs avatar. Pour cela j’assigne dans un premier temps à une variable **avatar** « req.file.filname » afin que lorsque la requête effectuer, d’enregistrer le nom du fichier téléchargé. Dans le header je précise **l’autorisation** où sera spécifié le **jeton d’authentification** du user. La méthode « findOneAndUpdate » de mongodb permet de rechercher l’id décodé du user contenu dans le token, et « $set » « modifier » le champs avatar qui contiendra le filename du fichier enregistrer.

* **user/getAvatar :** La route getAvatar permet de retourner l’avatar du user créé grâce à l’id du user.

L’upload des fichiers :

var multer = require('multer');

const storage = multer.diskStorage({

    destination: (req, file, cb) => {

        cb(null, './uploads');

    },

    filename: (req, file, cb) => {

        cb(null, file.originalname);

    } // Si il trouve que le nom d'origine existe déjà, en cas doublons

})

Pour procéder à l’upload de fichier j’ai utilisé le middleware multer qui permet d’upload des fichiers sur le serveur, dans le cas de mon projet les fichiers sont essentiellement des images (jpeg et png). La fonction ci-dessus permet dans un premier temps de vérifier l’existence du dossier upload. Dans un second temps, elle vérifie l’existence du fichier via le nom d’origine du fichier, si le nom du fichier existe déjà celui-ci n’est pas enregistré afin d’éviter les doublons d’images.

**Le Front :**

**Mise en place d’une application react :**

**Organisation de l’application (components, routage) :**

1. **Le component Register user :**

Afin de récupérer la valeur des inputs du formulaires d’inscription, j’ai mis en place les states suivantes **name, pseudo, email, password, des chaînes de caractères vides qui se rempliront dès qu’un user remplira les champs souhaités.**

* **La méthode handleChange() :** Cette méthode me permet de cibler la valeurs des attribut name de chaque input. Ensuite pour rendre cette méthode dynamique, je mets à jour le state indiquant que **l’attribut name est une clé** et qu’il faut en récupérer la valeur par value.
* **La méthode handleSubmit() :**

Cette méthode accueille la requête fetch qui permettra de communiquer avec le serveur. Ayant indiqué côté serveur que je souhaité traiter des donnés **JSON, ainsi** que la méthode **POST** et en indiquant dans le serveur. Enfin les données récupérées par la méthode **handleChange**, celles-ci seront changées en chaîne JSON.

Les requêtes Fetch fonctionnent avec **les promesses javascript,** au sein de la première promesse j’effectue dans un premier temps **la remontée des cas d’erreurs.** La gestion de cas d’erreurs étant définis côtés Front, je remonte celle-ci côté clients afin de l’afficher aux utilisateurs. Pour cela j’utilise la méthode switch afin de définir des conditions lorsqu’une erreur est reconnu par le serveur.

* **Status 401** : affiche à l’utilisateur que son mot de passe ne respecte pas la regex défini.
* **Status 400 :** affiche à l’utilisateur l’impossibilité de s’incrire sans le remplissage de l’ensemble des champs.
* **Status 200 :** la requête est correcte, l’utilisateur pourra être rediriger sur le formulaire de connexion.

Enfin la seconde promesse permettra de retourner les données au format json.

J’applique l’événement **onChange** sur chacun des champs afin d’effectuer la logique de la méthode **handleChange**. L’événement **onSubmit** sur le formulaire afin de procéder à la soumission de la requête.

1. **Le component Login User :**

Dans la promesse de la requêtes fetch du component login afin de procéder à la gestion du token côté front, au moment de la connexion tout comme côté serveur, le token affecté au user sera transférer dans le localStorage. Afin de l’utiliser dans certaines situations du user. Ainsi l’utilisateur sera autorisé à être redirigé sur la partie back office de son profil.

1. **Mise en place du back office User côté Front :**

Dans un premier temps, grâce à l’utilisation du token j’ai pu définir que seul les user disposant d’un token (incluant donc l’id et le rôle du user), peuvent avoir accès à la route/component « **Office user ».**

**Organisation du back-office user :**

Le component **OfficeUser** contient plusieurs **components enfants :**

* **OfficeUser :** Office user correspond au back office de l’utilisateur, au chargement de ce component je récupère grâce à la propriété getItem du localStorage le token de l’utilisateur. Ainsi les visiteurs ne pourront pas avoir accès à ce component. OfficeUser est découpé en trois autres components
* **OfficeProfile :** OfficeProfile représente la partie gestion de profile du back-office front.

Au chargement du component grâce à la méthode **componentDidMount**, je mets en place une requête **fetch** qui va dans une première promesse envoyer **la reponse** au format json. Dans la seconde promesse je vérifie dans une condition si l’utilisateur dispose ne dispose pas d’avatar afin de lui affecter un avatar par défault en **setState** le state avatar qui correspond au src de la balise image. Dans le cas où l’utilisateur dispose déjà d’un avatar, je concatène la l’adresse où se situe les images enregistrées avec la réponse qui côté serveur correspond au **filename** actuel de l’image qui représente l’avatar.

La méthode HandleChangeAvatar permet via une requête fetch de procéder à l’upload d’image pour l’avatar côté front. Etant un upload de fichier, j’assigne à une variable data la méthode javascript formData. La propriété append de formData permet d’indiquer avec une clé et une valeur la donnée qu’on souhaite envoyé, étant un inputfile je précise que je cible la propriété files de l’input en y récupérant qu’n seul fichier. Enfin dans la première promesse j’envoyé la réponse au format json, ensuite dans la seconde je récupère la réponse de l’update et comme celle-ci contient le filname de l’image, je setState l’adresse du dossier uploads avec la réponse de l’ulpoad.

* **OfficeProject :**
* **OfficeLinks :**
* **WhiteBoard :**