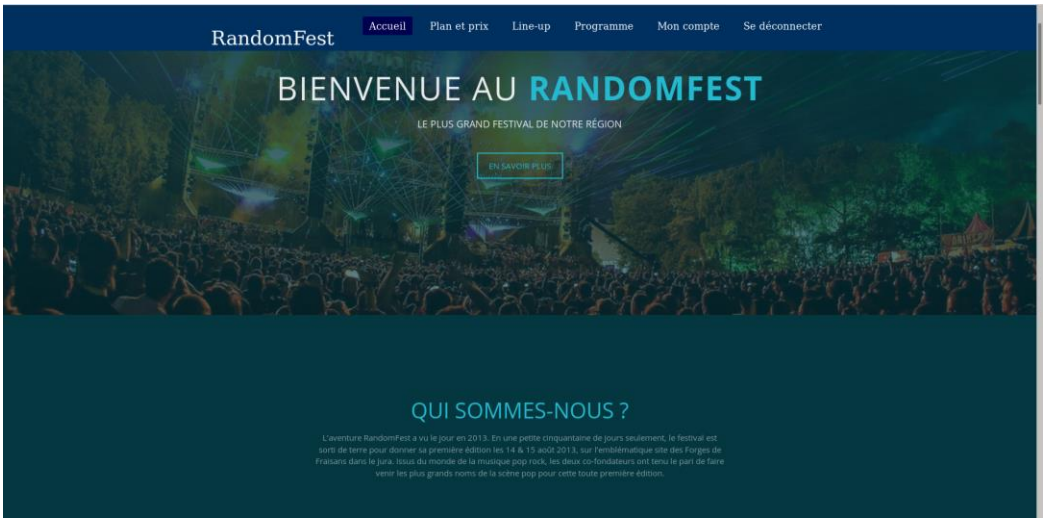
Iane Vallanzasca
Florian Golgevit
Jean-Baptiste Lanneluc

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Rapport de Projet WEB

Random Fest'

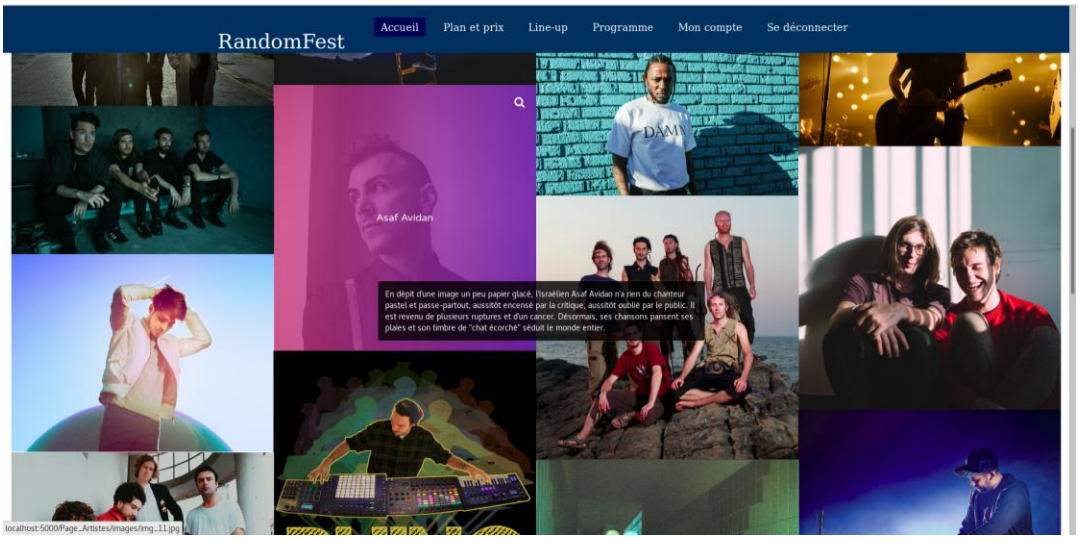


Fonctionnalités :

Notre site dispose évidemment d'une page d'accueil qui présente l'équipe, l'esprit du festival, les scènes, le lieu etc. C'est donc une page « vitrine » qui se doit d'être accueillante et esthétique afin de donner envie aux clients potentiels de venir. Elle permet également de montrer que c'est un vrai festival pour les sponsors potentiels, rassemblant beaucoup de personnes, avec notamment des photos des éditions précédentes. Elle dispose également des infos pratiques concernant le lieu, moyen d'accès, les dates, le prix, etc.

A partir de là, l'utilisateur peut naviguer comme il le souhaite entre les différentes pages du site grâce à la barre de navigation, présente sur toutes les pages. Sur cette barre se trouvent plusieurs boutons.

S'il clique sur Line-up, il ira sur la page de présentation des artistes. Cette page est très visuelle et liste tous les artistes présents sous forme d'images. L'utilisateur a accès s'il le souhaite à une brève description pour chacun d'eux, qui s'affiche lorsqu'il clique sur la photo dynamique de l'artiste (ici le curseur est sur la photo d'Asaf Avidan par exemple).



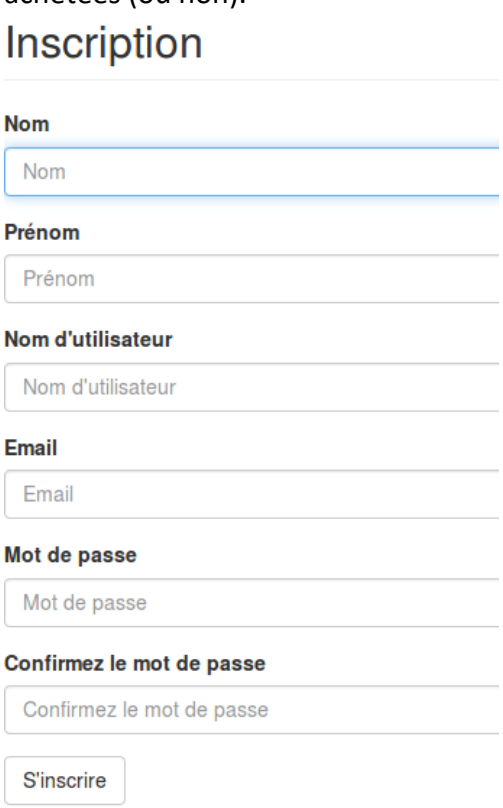
Le second bouton redirige vers une page de la programmation : cette page permet à l'utilisateur de connaître les horaires et la scène de passage de chaque artiste.

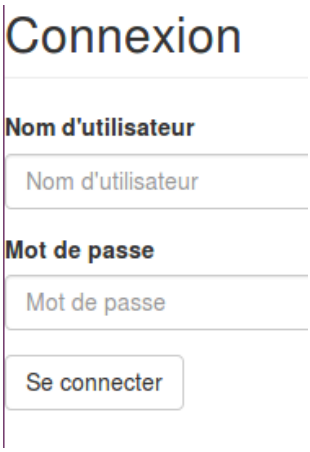
Si après avoir navigué sur notre site, l'utilisateur est intéressé par le festival, il peut acheter des places. Pour cela, il accède à la billetterie en cliquant sur le bouton correspondant. Pour y avoir accès, l'utilisateur doit avoir un compte sur le site et y être connecté. Si ce n'est pas le cas, il sera redirigé vers la page de connexion. Une fois cela fait, il accède à la billetterie. Il peut alors consulter la liste des pass disponibles. Ensuite il peut choisir la quantité et le type de billets qu'il souhaite acheter, et les ajouter à son panier. Avant de procéder au paiement, il devra rentrer les informations personnelles de la personne à qui la place est destinée. Il peut acheter ainsi des places non seulement pour lui mais pour d'autres personnes car le site n'utilise pas les informations de son compte pour les pass. Ensuite, il pourra procéder au paiement, qui se fait via l'API Stripe.

L'utilisateur peut également avoir accès à la connexion ou à l'enregistrement via la barre de navigation. La page de connexion demande un identifiant et un mot de passe. Si l'utilisateur a déjà créé un compte, il lui suffira de rentrer ses identifiants et mot de passe pour se connecter. Sinon, il aura un message d'erreur.

Pour créer un compte, l'utilisateur doit renseigner tous les champs de la page d'enregistrement, dont son nom, prénom, etc. Une fois tous les champs rempli, il peut créer son compte en cliquant sur « s'inscrire ».

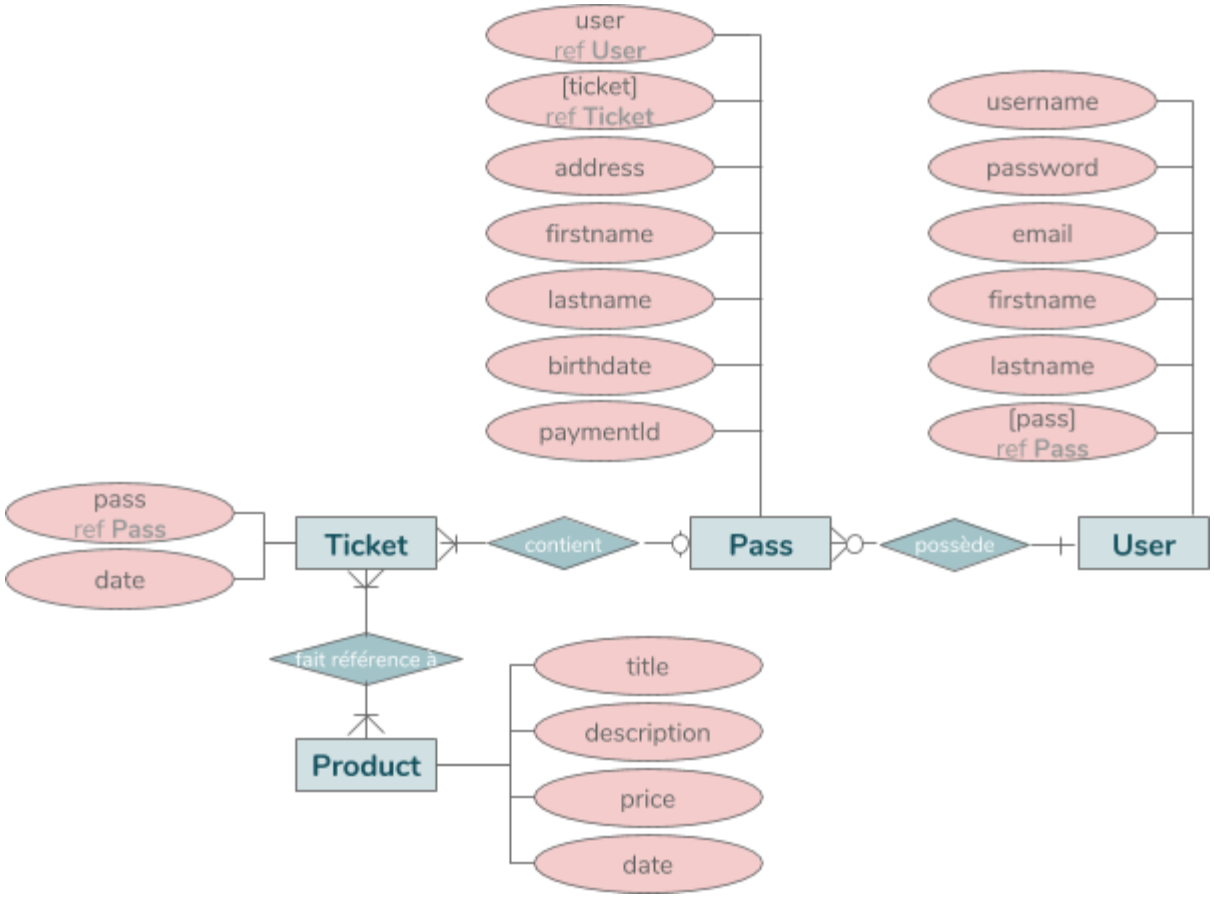
Connecté, il pourra avoir accès à ses informations personnelles et aux places qu'il a achetées (ou non).





Structure de la base de données :

Notre base de donnée est gérée par MongoDB, système NoSQL. Nous avons implémenté quatre collections :



- User stocke les informations des membres inscrits sur le site : pseudo, mot de passe, email, nom et prénom. Elle est consultée lors de la connexion d'un utilisateur et se rempli au fur et à mesure des inscriptions. Elle recense ainsi toutes les personnes inscrites sur le site. User peut également contenir des références vers les pass achetés par l'utilisateur.
- Pass modélise un pass pour le festival. La collection contient les informations personnelles du festivalier qui permettront de l'identifier à l'entrée du festival et fait référence à l'utilisateur par qui elle a été achetée, sachant qu'un utilisateur peut tout à fait acheter des places pour d'autres personnes que lui. Le pass peut être valide pour un, deux ou trois jours: cela se modélise par une liste de références à une autre collection, Ticket. Un ticket donne l'accès à un jour.
- Ticket stocke tous les tickets à la journée : cela permet de quantifier le maximum de personnes par jour autorisées sur le festival. Ses seuls attributs sont le jour de validité et une référence vers le pass dans lequel elle sera

utilisée. L'attribut pass peut être nul ou non, selon si la place est ou non disponible. La collection est initialisée à N tickets par jour ne faisant référence à aucun pass. Si le ticket ne fait référence à aucun pass, on considère qu'il est disponible et pourra être acheté.

- Product liste les pass disponibles à l'achat. Elle contient un nombre fini de documents : pass Jour 1, pass Jour 2, pass Jour 3, pass 2 jours et pass 3 jours. Elle sert à l'affichage des places disponibles dans la billetterie. Elle est mise à jour quand un type de pass est épuisé, ce qui permet d'afficher uniquement ce qui est encore disponible.

Structure du code :

Back-end :

Nous avons structuré notre code de manière à bien séparer le Back-End et le Front-End. Nous avons donc un serveur NodeJs réalisé via express (un framework de NodeJs) qui s'initialise grâce à deux fichiers, le premier appelé via la commande start de npm, définie dans notre package.json, qui constitue le coeur du serveur, avec les diverses fonctions d'écoute et de création du serveur. Le second fichier, appelé par le précédent, définit les dépendances globales du serveur, et les chemins vers lesquels il ira chercher le Front-End. Il initialise aussi les diverses fonctionnalités du site et les modules additionnels si besoin, et se connecte à la base de données mongoDB.

Ensuite, nous avons décomposé notre code suivant les fonctionnalités de chacun des fichiers. Nous avons ainsi un dossier "config" dans lequel, via passport - le module utilisé pour gérer les logins des utilisateurs - nous avons nos fonctions de login et register.

Nous avons aussi un dossier "route", dans lequel nous indexons les différentes adresses de notre site tout en appelant leur affichage Front-End correspondant. Nous décomposons cette indexation en deux fichiers : un fichier contenant toutes les routes vers des pages "vitrines" ou de simples redirections, et un fichier rassemblant les pages dédiées à l'utilisateur (page d'inscription, de connexion, espace personnel). Il faut noter que dans cette partie, nous avons aussi implanté des commandes qui s'exécutent une fois l'affichage de la page chargé. C'est ainsi principalement dans ce dossier que nous appelons et utilisons une API externe : Stripe, pour gérer notre système de paiement, une fois l'utilisateur logué et arrivé au moment du paiement, donc à la page correspondante.

Un autre dossier présent est le dossier "models", dans lequel nous avons les différents schéma de stockage des données pour les rentrer dans notre base de donnée (comme les pass, les tickets, ou les utilisateurs). C'est dans cette partie notamment que nous utilisons un module de hachage : bcrypt, qui nous permet de sécuriser l'enregistrement des mots de passe de nos utilisateur dans notre base de donnée. On retrouve enfin dans cette partie le fichier permettant la gestion du panier des utilisateurs.

Front-end:

Toute la partie front-end de notre site, c'est-à-dire la partie à laquelle l'utilisateur peut avoir accès, se trouve dans les dossier "views" et "public". En effet, les différentes pages html sont appelées par le fichier de routage dans le dossier views, puis elles-mêmes appellent des ressources css, JavaScript, JPG etc directement dans le dossier public. Ainsi, nous pouvons comme dit précédemment facilement séparer le back-end et le front-end, et le client n'a pas les droits de demander une requête sur les fichiers du back-end.

Si les pages html sont toutes dans le même dossier, elles utilisent toutes des ressources css et js différentes, qui portent souvent le même nom, puisque nous avons construit notre site en combinant les fonctionnalités intéressantes de plusieurs templates. Plutôt que de fusionner toutes les classes css dans un seul fichier, de rassembler toutes les images dans un seul dossier et de rassembler tous les scripts JS, nous les avons séparés en dossiers, un pour chaque page du site, et avons modifié l'appel par la page html.

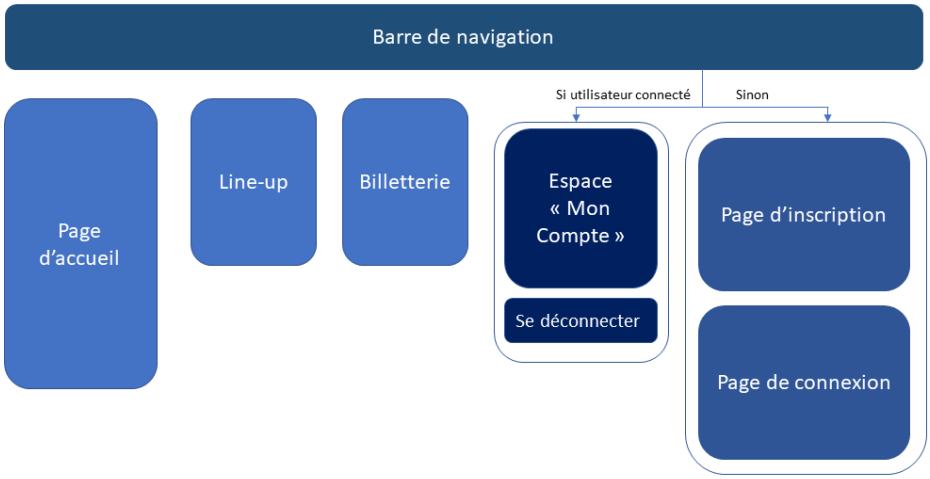
Combiner plusieurs templates en une seule page a demandé beaucoup de travail, puisque de nombreuses classes css ou scripts js entraient en conflit en raison d'un même nom, ce qui posait problème. Nous avons ainsi dû adapter les classes, et surtout en créer de nouvelles pour le texte par exemple, afin que chaque page soit uniforme, en termes de police, taille etc, et corresponde à ce que l'on recherche.

Afin d'aider l'utilisateur à naviguer sur le site, nous voulions implanter une barre de navigation. Cependant, plutôt que de devoir rajouter le code de la barre sur toutes nos pages html, nous avons trouvé un outil très adapté pour réaliser cela : le template engine Handlebars, d'Express. Alors que la plupart des gens utilisent désormais ejs ou jade comme template engine pour Express, Handlebars a l'avantage de pouvoir ajouter un en-tête automatiquement sur chaque page du site, ainsi qu'un pied de page si nécessaire, évitant ainsi des conflits de classe potentiels.

Notre barre de navigation se présente comme ci-dessus, et son code est indépendant du code de chacune des autres pages. Handlebars l'ajoute simplement en tant que barre de navigation sur chaque page html.

Nous avons voulu faire un site esthétique, et non pas seulement technique, puisque c'est un site censé attirer du monde. Chaque page dispose donc de quelques effets visuels grâce au css et à des scripts js, comme par exemple un bouton qui change de couleur lorsque le curseur passe dessus, ou la photo d'un artiste qui s'agrandit, etc. C'est cette motivation qui nous a poussé à combiner les effets de plusieurs templates différents.

Architecture du site :



Voici un schéma de l'architecture de notre site au moment où nous écrivons ce rapport :

La barre de navigation donne accès à différentes pages selon que l'utilisateur soit connecté ou non, mais l'essentiel du site reste évidemment accessible dans tous les cas, à part la boutique qui redirige sur la page de connexion si nécessaire.

Pistes d'améliorations

Nous avons énormément d'idées de fonctionnalités que nous pourrions ajouter à notre site pour le rendre plus riche, et également des idées d'améliorations que nous pourrions apporter à celles qui sont déjà existantes.

Tout d'abord, nous pourrions améliorer le système de login de plusieurs manières. Déjà, implémenter une vérification d'e-mail avec un envoi de mail automatique lors de l'inscription, qui demanderait de cliquer sur un lien pour la valider. Ensuite, nous pourrions suivant le même modèle faire un système de réinitialisation de mot de passe, via e-mail.

Ensuite, ce qu'il peut être intéressant de faire, c'est une base de données recensant tous les artistes ainsi que leur description et une image. Cela permettrait de faire appel à la base de données au lieu d'insérer ces données en brut dans le template html, ce qui serait plus propre. A partir de ça, nous aurions pu ajouter une fonctionnalité permettant de rechercher un artiste précis par style ou nom avec des filtres.

Nous pensions également à une fonctionnalité indiquant la fréquentation prévue à chaque concert : les utilisateurs auraient répondu à un genre de sondage dans leur espace client, dans lequel ils indiqueraient quels concerts les intéressent, afin d'estimer l'affluence à ce concert pour indiquer aux festivaliers qu'il vaut mieux venir en avance ou non.