IMAC2 - Développement Web - 2020 - Victor VIALE

Rapport de projet Web

Thomas GESLIN
Julien HAUDEGOND
Cyrielle LASSARRE
Océane RIOSSET
Sarah VEYSSET

Panorama des festivals du territoire français

https://julien-haudegond.github.io/IMAC_S4_Web_Dashboard/
https://github.com/Julien-Haudegond/IMAC_S4_Web_Dashboard

Choix du thème de notre projet

Pour ce projet, nous avons, dans un premier lieu, commencer à chercher un thème convenant à chacun des membres de notre équipe. Nous avons consulté les sites donnés dans le sujet du projet et c'est sur https://www.data.gouv.fr que nous avons trouvé notre bonheur. En effet, après la sélection de cinq banques de données, c'est finalement le panorama des festivals couvrant tous les domaines culturels du territoire français qui a retenu notre attention.

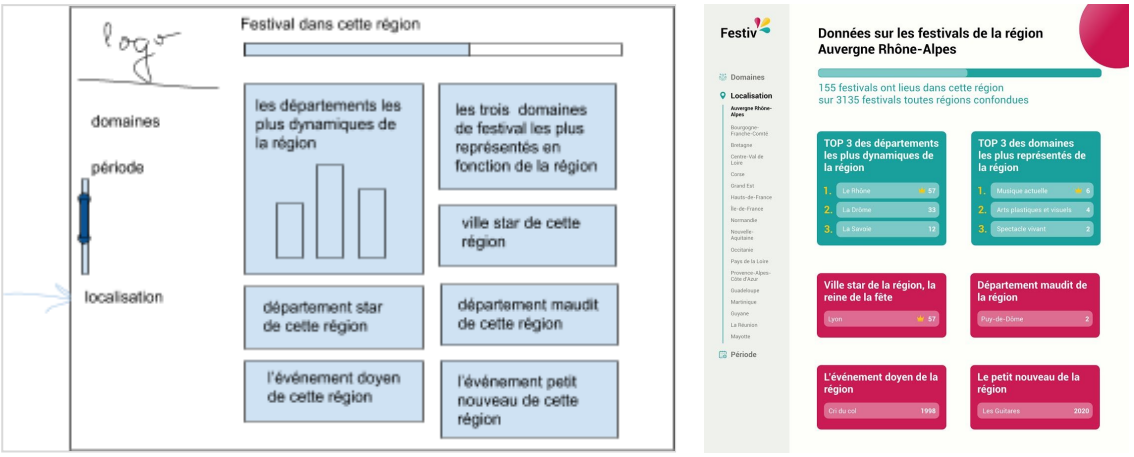
https://www.data.gouv.fr/fr/reuses/informations-cles-sur-les-festivals-dans-tous-les-domaines-culturels-et-sur-tout-le-territoire/

Ce dernier avait deux principaux avantages :

- il proposait différentes façons d'exploiter les données, ce qui nous permettait d'avoir un champ d'action plus large
- le thème nous intéressait et nous parlait

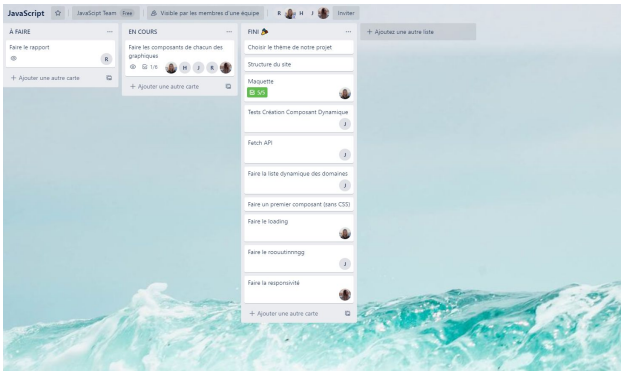
Structure et design du site

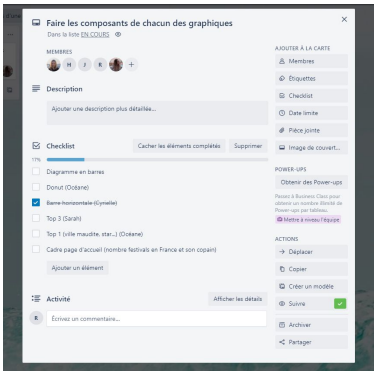
Nous avons fait une première réunion afin d'organiser et de définir la structure des pages de notre futur dashboard. Nous avons alors brainstormé sur les éléments à faire apparaître, les différents graphiques que nous pourrions faire figurer… Nous nous sommes mis d'accord sur des maquettes simplifiées et nous avons lancé la production du web design.



La page Localisation avant le design et après le design

Pour notre organisation et au vu de notre grande équipe, nous avons créé un Trello. Sur celui-ci, nous avons écrit les différents tâches et les avons distribué. Cet outil permet de voir l'avancement grâce au catégorie : 'à faire', 'en cours' et 'fini'.





PLANNING :

02/04	Recherche de thème pour notre projet
04/04	Recherches d'inspirations graphiques pour constituer un moodboard
07/04	Réunion Discord : choix des infos à afficher et design, définition de la structure du site et création du trello
10/04	Maquette designée, finie et validée par l'équipe
11/04	Création du GitHub et code de la page d'accueil et du menu
15/04	Réunion sur l'avancement du projet (maquette faite, page d'accueil et menu déroulant faits)
16/04	Code du loader et des components qui n'utilisent pas chart.js
18/04	Mise en place du routage
20/04	Point avec Victor
25/04	Hébergement du site

Justification de nos choix

1. MISE EN PLACE DU PROJET

Nous avons fait de nombreuses recherches sur la façon dont les frameworks front-end tournent ainsi que sur l'architecture nécessaire au bon fonctionnement de ce type de projet.
En s'inspirant de différentes bases et ressources trouvées en ligne, nous avons choisi les librairies suivantes :

- Empaqueteur : Parcel.js (plus simple à mettre en place que WebPack)
- Transpileur JavaScript : Babel.js
- Analyseur de code : Eslint (avec le fichier de configuration mis à notre disposition dans les TDs afin de suivre exactement la syntaxe attendue)
- Notre framework favori : Hyperapp
- Client HTTP : Axios
- Suppresseur de dossiers : Rimraf (pour nettoyer facilement le cache)
- Outil de copie : cpx (afin d'ajouter facilement le '404.html' à notre dossier de build)
- Module de visualisation : Chart.js
- Module de temps : Moment.js

Concernant la syntaxe, nous nous sommes tournés vers la syntaxe JSX.

2. ROUTAGE

Nous disposons de 4 types de pages différents :

- Home
- Domain
- Localisation
- Period

Afin de les afficher, nous avons dû mettre en place un routage. Chaque élément du menu peut être vu comme un "filtre". Ainsi, dès lors que l'utilisateur clique sur l'un de ces éléments, nous générons à la volée un composant global relatif à ce "filtre" qui va afficher tous les modules de visualisation qui lui sont associés.

Exemple : cliquer sur "Cinéma" amènera sur une page de type "Domain", c'est-à-dire qu'un composant de type "DomainContent" sera généré et contiendra alors tous les graphiques correspondants.

Avant de se lancer dans la procédure de routage, nous avions pensé à utiliser un attribut de notre état qui correspondrait à la page actuelle et dont ce dernier aurait été mis à jour par une action. Cependant, même si les tests étaient relativement concluants, un problème majeur est apparu : recharger la page entraînait un renouvellement complet de notre état et donc un retour inévitable à la "page d'accueil". De plus, d'après les recherches effectuées à ce sujet, on a cru comprendre que cette méthode allait un peu à l'encontre du framework.

Et c'est là que le routage est entré en jeu !

Pour une fois, on ne peut plus dire que toutes les routes mènent à Home..! :)

3. ACTIONS ET ETAT

Nous avons choisi de constituer notre état au chargement initial du site web (lors de la requête vers l'API) et qu'il ne soit pas modifié par la suite. Cela revient également à dire que l'utilisateur ne peut en aucun cas déclencher une action ayant une répercussion directe sur notre état.

Toute l'application est basée sur notre système de "filtres".
Pour créer une nouvelle page, nous allons passer à notre composant global uniquement les données qui lui sont utiles en les filtrant au préalable.

Les graphiques

1. LOADING BAR

Cette barre située en première ligne de notre site pour chacune des catégories Domaine, Localisation et Période est indépendante de chart.js. Elle permet de visualiser la place du champ sélectionné parmi l'ensemble des festivals du territoire français.

Elle est codée parmi les components dans un fichier dédié. Pour la réaliser, nous avons utilisé les données du nombre de festivals concernés par le champ sélectionné et le nombre de festivals au total. À partir de ces informations, nous étions capables de réaliser un produit en croix et ainsi de **déterminer le pourcentage qu'occuperait notre barre**. Ce résultat étant stocké dans la variable percent que nous passons ensuite au champ CSS width. Cette dernière étape m'a causée quelques difficultés car je ne savais pas quelle syntaxe utiliser pour faire passer ce champs. Après de nombreux essais j'ai finalement opté pour cette méthode qui me paraissait être la plus simple à mettre en oeuvre.

Afin d'**adapter le texte à la vue sélectionnée par l'utilisateur**, nous avons utilisé un switch qui couvre chacune des pages de notre site. Dans chacun des fichiers DomainContent.js, PeriodContent.js et LocalisationContent.js, nous importons le fichier précédemment décrit puis nous relions les données nécessaires à la composition du graphique.

Le style de ce component a été complété dans le fichier MainContent.css afin d'ajouter les couleurs correspondantes à la maquette, une transition et les différentes caractéristiques de la loading bar.

2. TOP 3

Ce component apparaît pour chacune des catégories Domaine, Localisation et Période. Il est indépendant de chart.js et permet de visualiser un top 3 en fonction de divers filtres : top 3 des régions les plus dynamiques, top 3 des départements les plus dynamiques ou encore top 3 des domaines les plus représentés de la région.

En plus d'un fichier dédié à ce component, celui-ci utilise **la classe utils**. Dans cette classe, une série d'instruction est donnée :

- récupération via la fonction **map()** des données que l'on veut
- suppression des doublons via **l'objet Set**
- création d'une nouvelle **Map** afin d'avoir notre donnée ainsi que son nombre d'apparition
- création d'une nouvelle **Map** afin de trier par ordre décroissant notre précédente map

Ces nombreuses étapes m'ont causé des difficultés et une perte de temps car il m'a été compliqué de trouver la syntaxe appropriée pour chaque étape.

Pour rendre ce component réutilisable nous avons utilisé un **switch** afin de changer le titre de celui-ci en fonction de nos besoins.

3. TOP 1

Ce component apparaît pour les catégories Localisation et Période. Il n'utilise pas non plus chart.js. Ce "top 1" permet de visualiser : la ville star selon la période (celle avec le plus de festivals à ce moment-là), la ville star selon la localisation (celle avec le plus de festivals selon le lieu) et le département maudit (celui qui a le moins de festivals) .

Il possède un fichier dédié nommé 'top1.js' qui utilise **la classe utils**. Dans cette classe, on utilise le même procédé que pour le top 3 sauf qu'il a fallu ajouter un élément pour que l'on puisse avoir une Map triée dans l'ordre décroissant pour la ville star et une dans l'ordre croissant pour le département maudit. Pour rendre ce component réutilisable nous avons utilisé un **switch** afin de changer le titre de celui-ci en fonction de nos besoins.

J'ai passé beaucoup de temps à réfléchir la structure pour faire les 7 top1 de base. Le problème principal est venu de la récupération de dates, qui fut trop complexe et je n'avais pas le temps de le faire au vu du reste du travail. Il était compliqué traité la date pour ne récupérer que les nombres intéressants,les dates étant du type "XXXX-XX-XX", et de les associer au nom du festival. Nous avons donc décidé de ne pas les mettre.

CHART JS

Pour avoir des graphiques dynamiques, nous avons choisi d'utiliser Chart js. Cet outil permet de créer plusieurs types de graphiques (camembert, graphiques en bar, donut…). Sa structure permet de choisir facilement les données, les couleurs, les légendes. Pour l'utiliser, il suffit d'importer chart js dans le fichier souhaité et de suivre la structure (faire un canva dans lequel on définit les éléments du graphique).

1. GRAPHIQUE DONUT

Ce component apparaît dans les catégories Domaines et Période. Il utilise chart.js et fut le premier élément de chart.js a être fait. Le donut permet de visualiser : la répartition des festivals adaptés aux enfants, la répartition en fonction de la parité des départements, la répartition en fonction des saisons, la répartition en fonction des îles de France et la répartition en fonction des environnements (mer, montagne, autre).
Le donut possède un fichier dédié, dans lequel on importe chart.js. Pour ce dernier, j'ai beaucoup regardé la doc chart.js qui est très bien faite même si cela n'a pas été facile de l'adapter à la nouvelle syntaxe js, ce qui a fait perdre pas mal de temps.

Pour son utilisation, j'ai par exemple recuperé les numéros de départements que l'on a testé sur leur parité. J'ai aussi créé des listes dans lesquelles j'ai placé les données des départements/régions dont j'avais besoin et je les comparais pour pouvoir les récupérer dans mon tableau de données pour mon donut selon les catégories.
On a passé beaucoup de temps sur ce component car il était plus complexe que prévu ainsi que sur le traitement des données qu'on lui fournissait.

2. GRAPHIQUE BAR

Ce component permet d'avoir une vue d'ensemble des données concernant les festivals suivant leur mois de début habituel ou leur thème suivant la page sélectionnée. Comme le component Donut, Bar utilise Chart.js. Son comportement est spécifié dans le fichier Bar.js et instancié dans les pages Domaines et Période. Nous avons d'abord développé le component de manière statique avant de spécifier des filtres. Enfin, pour rendre ce component dynamique, nous avons utilisé les méthodes .map et reduce afin d'automatiser en une ligne la distribution du filtre sur les données. Le plus compliqué fut de comprendre que la méthode .sort modifiait le jeu de données et n'était donc pas appropriée ici. Nous avions aussi développé un graphique de courbe mais celui-ci ne s'est finalement pas révélé utile vis-à-vis des données que nous devions traiter.

Difficultés rencontrées

1. FETCHING API ASYNCHRONE

Faire les requêtes à l'API n'a pas été un problème. Cependant, pour s'assurer que toutes les données soient bien récupérées et permettre par la suite de créer les composants sans avoir de soucis de variables non définies, ce n'était pas la même histoire ! Async/await qui ne semble pas fonctionner ?!? Après être resté bloqué sur une Runtime Regenerator Error, c'est quand même dommage, ahah ! Du coup, on a mis en place des attributs dans le state qui nous assurent qu'aucune page ne s'affiche pendant le chargement de l'API. L'avantage, c'est que ça nous a permis d'inclure facilement une page de loading ! Mais est-ce la bonne façon de procéder ?!

2. CHART.JS

La documentation de Chart.js utilise l'ancienne syntaxe JS, dépréciée aujourd'hui. Mais ce n'est pas un gros problème. Là, où c'était plus compliqué, c'est que la documentation ne prend pas du tout en compte l'utilisation de la librairie avec un framework. Du coup, adapter le code dans un composant n'a pas été évident. Une fois fait, nous pensions que tout allait bien… jusqu'à ce que l'on se rende compte que les graphs ne se mettaient pas à jour !
Il nous a fallu beaucoup chercher avant de trouver la solution que l'on utilise actuellement : appeler une fonction qui update ledit graph en le récupérant dans la liste de toutes les instances de Chart grâce à son identifiant.

3. MISE EN LIGNE

Hébergement sur un serveur ? Mutualisé, virtualisé ou dédié ? Hébergement de pages statiques ? Tant de questions ! Pour ce type de Dashboard, un simple hébergement statique après avoir buildé l'application est finalement largement suffisant. Cela semblait être rapidement réglé… enfin, avait-on parlé trop vite ? Eh oui ! Problèmes de liens relatifs vers nos images, impossibilité de rendre la page d'accueil, les rafraîchissements mènent vers des 404… Finalement, ce n'était peut-être pas si mal quand on tournait sur notre serveur, non ?! Heureusement, la persévérance nous pousse à toujours trouver des solutions (plus ou moins bonnes). Tous ces problèmes ont donc été réglés ! Ouf !! Et en prime, après tant d'heures de recherches, ce qui était encore un peu obscur à propos du routage s'est peu à peu éclairci. On remerciera quand même **rafrex** pour son "hack" très astucieux pour contourner les 404 de GitHub Pages !

Questions qui restent en suspens

1. ROUTAGE

Le Switch qui était censé englober toutes nos routes provoquait un mauvais “rafraîchissement” de celles-ci. Ainsi, les composants se surperposaient de façon aléatoire. Nous avons dû l’enlever afin de résoudre le problème. Mais quel était ce problème ?

2. MISE EN LIGNE AVEC UN SERVEUR

Nous n’avons pas bien saisi comment mettre l’application en ligne sur un serveur (non statique). Les tutoriels disponibles sur Heroku, par exemple, expliquent assez bien la mise en “lien” avec le serveur. Cependant, il faut ensuite configurer le serveur sur le port d’écoute, etc. Cette partie reste très floue.

Avis personnels

Julien

Ce projet a été très enrichissant pour moi ! Le travail d’équipe a très bien fonctionné ! J’étais légèrement plus à l’aise en JS que mes camarades donc nous avons fait de nombreux streams afin que j’essaye de les aider un peu : j’ai beaucoup aimé cela !
Quelques unes de mes questions restent cependant encore en suspens. Je viendrai peut-être t’embêter Victor pour avoir quelques informations supplémentaires, ahah !

Océane

Ce projet a été très ambitieux,. J’en avais peur car j’ai beaucoup de difficultés en programmation et je ne pratique pas le js depuis longtemps. Heureusement, la répartition des tâches étaient claires et chacun avait sa part. Julien m’a beaucoup aidé et débloqué durant le projet, ainsi que d’autres personnes. Nous avons beaucoup utilisé Discord avec partage d’écran, ce qui était très efficace. Grâce à eux, j’ai pu réussir à coder mes components. Je suis contente d’avoir travaillé sur ce projet et très fière de mon équipe.

Sarah

Personnellement, j’ai pris plaisir à travailler sur ce projet et il a été très enrichissant ! Ce plaisir est fortement lié au bon fonctionnement de notre équipe. En effet, nous nous sommes très bien répartis les tâches au départ ce qui a permis une avancée fluide et rapide

du travail. De plus, Julien, plus à l'aise en programmation, nous a guidé sur ce projet et nous a épaulé face à certaines difficultés. :)

Thomas

Ce projet a été l'occasion pour moi de sortir de ma zone de confort, étant plus à l'aise avec le HTML/CSS que le javascript. Comme Océane, Julien a été d'une grande aide en m'aidant à comprendre les parties de mon code qui me posaient problème. J'ai aussi apprécié pouvoir me référer à une maquette et un visuel déjà défini en me permettant de me fixer des objectifs intermédiaires. La communication au sein du groupe a été facile et les sessions de réunions sur Discord m'ont permis de discuter avec Océane et Julien, puisque le code des components était assez proche.

Cyrielle

J'ai trouvé que faire un projet dans cette matière était le plus efficace pour tirer bénéfice de ce que nous avions appris en TD ! En effet, nous avons pu mettre nos apprentissage à profit d'un thème que nous avions envie de traiter avec une synergie de groupe motivante qui nous pousse à aller plus loin dans nos recherches. Nous avons pu nous répartir les tâches de manière efficace et également résoudre les problèmes ensemble lorsque le besoin s'en faisait ressentir. Julien, plus à l'aise en programmation, a su apporter ses connaissances et ses conseils précieux à notre groupe. Sa patience et son approche pédagogue ont été très bénéfiques et un moteur pour notre projet. Je suis très contente et fière du travail que nous avons fourni. :)