

**PROJET DE FIN DE FORMATION**

**CONCEPTION/DEVELOPPEMENT D’UNE APPLICATION WEB**

**Matthieu Bourgoin**

Développeur Web/Web Mobile, option Cybersécurité

SESSION 2021-2022



**TABLE DES MATIERES**

**I/** **Introduction**

**II/ Cahier des Charges**

* II – 1 *Compétences Professionnelles couvertes par le projet*
* II – 2 *Présentation du Projet*
* II – 2 *Analyse du Projet*
* II – 3 *Objectifs du Projet*

**III/ Spécialités Fonctionnelles**

* III – 1 *Arborescence*
* III – 2 *Maquettage*
* III – 3 *Analyse Fonctionnelle*
  + *III – 3.1 Diagramme de cas d’utilisation*
  + *III – 3.2 Diagrammes d’activité*
  + *III – 3.3* *Diagrammes de séquence*

**IV/ Conception**

* IV – 1 *Modèle Conceptuel de Données (MCD)*
* IV– 2 *Modèle Logique de Données (MLD)*

**V/ Développement du Projet**

* V – 1 *Outils couvert par le Projet*
* V – 2 *Les contraintes de développement*
* V – 3 *Présentation d’une API*
* V – 4 *Le Code*
  + *V – 4.a Développement « front-end »*
  + *V – 4.b Récupération d’une API et insertion en base de données*
  + *V – 4.c Fonctionnement de l’API du Projet*

**VII/ Conclusion**

* VI – 1 *Améliorations futures*
* VII – 2 *Bilan personnel*

**VIII/ Annexes**

**Remerciements**

**Abstract**

**I. Introduction**

J’ai à un moment ou à un autre depuis ma petite enfance, possédé des cartes à collectionner. Le plus souvent ceci dès le primaire, là où les cartes *Pokémon* faisaient rage dans la cour de récré, on découvrait un univers au travers de ces petites cartes aux visuels passionnants.

Très rapidement, deux écoles se forment au fil des années : les personnes qui ont abandonné leurs cartes au profit d’un autre divertissement et les autres, qui ont petit à petit découvert qu’il existait de véritables règles,

relayant ainsi les cartes en véritable [jeu de cartes](https://www.actugaming.net/tag/jeu-de-societe/) avec ses propres règles.

*Magic the Gathering* (ou *Magic L’Assemblée* dans la langue de Molière) fait bien entendu partie de tout ceci.

Pour présenter l’histoire de *Magic the Gathering,* le jeu est sorti en 1993, créé par Richard Garfield,à l’époque étudiant, aujourd’hui professeur de mathématiques et concepteur de divers autres jeux. *Magic the Gathering* reste le jeu de cartes à collectionner le plus joué au monde encore aujourd’hui. Avec 25 ans derrière lui, le jeu de Richard Garfield n’a jamais cessé d’évoluer au fil des années, comme par exemple la création du jeu vidéo *Magic Arena* qui propose des compétitions internationales d’eSport avec des joueurs professionnels.

\*(Ajouter parcours et projets futurs) \*

\*(Décrire le choix de passer par une API pour le projet) \*

1. **Compétences Professionnelles couvertes par le Projet**



**II. Cahier des Charges**

**II.1 Présentation du Projet**

Mon projet s’articule autour de la création d’un paquet de cartes, communément appelé « deck ». Plus précisément la mise en place d’un système de « filtre » qui permettra de trier les cartes par couleurs, types, ou encore rareté par exemple. Il existe bien d’autres informations sur ces cartes, comme nous pouvons le voir sur l’image ci-dessous.

En effet, ces « filtres » permettront aux utilisateurs une recherche plus précise et plus rapide lors de leur construction. Également, certains decks axe leur stratégie autour de certaines cartes du même type, ou de même couleur.

Le but de ce projet est donc de créer une meilleure expérience pour l’utilisateur. En définitive, l’orientation du projet se tourne vers un affichage très dynamique lors du choix de certains type de carte sans rafraîchissement de la page. Nous verrons comment développer cette fonctionnalité plus tard. *(cf. Chapitre V)*



**II.2 Analyse du Projet**

**Points Forts :**

* Site en Français/Anglais.
* Outil utile pour les joueurs de Magic du format Physique et Numérique.
* Actualités quotidiennes.

**Points Faibles :**

* Sites similaires déjà existants.
* Difficulté à se démarquer.
* Nouveau site web.

**Charte Graphique :**

* En rapport avec les couleurs de Magic (Rouge, Bleu, Noir, Vert, Blanc).
* Intégration d’illustrations et d’artworks.
* Site épuré, confortable, lisible et cohérent.

**Cibles :**

* Joueurs actifs de Magic du format Physique et Numérique.
* Nouveaux joueurs ayant un goût pour les jeux de stratégies.
* Joueurs voulant partager ses créations « decklists »

\*(Rédiger sous forme de phrases) \*

**III. Spécifications fonctionnelles**

**III.1 Arborescence**

****

**arborescence**

**III.2 Maquettage**

1. **Zoning** d’une page où consulter une carte.



**Le zoning est une représentation globale d’une page qui permet de schématiser “grossièrement” une page web grâce à l’utilisation de blocs.** L’objectif est de présenter l’emplacement des zones de contenu et des grandes fonctionnalités.

C’est une première étape déterminante pour définir l’organisation générale des pages d’un projet. Elle arrive juste après la création d’une arborescence.

**Le zoning permet de présenter une** **première approche d’une page web afin d’en valider les grands axes** (zones de contenu, fonctionnalités, hiérarchie, …).

La validation des zonings constitue une étape déterminante, préalable à la définition des Wireframes. Les zonings permettent par exemple d’identifier une surcharge d’information sur la page d’accueil d’un site web.

1. **Wireframe** d’une page où consulter un deck.



Les wireframes exploitent les zonings réalisés lors de la phase précédente pour préciser chaque bloc en introduisant les contenus présents. Un wireframe est équivalent au squelette ou à la structure simple d’une page web ou de l’écran d’une application mobile. Il est beaucoup plus détaillé que le zoning.

Le wireframe est la suite logique du zoning. Chaque bloc réalisé lors de l’étape précédente se voit doté d’image(s), de texte(s) ou de vidéo(s). Ce contenu peut être fictif (ex. Lorem Ipsum) car les informations finales ne sont pas toujours connues à ce stade du projet. De plus l’étape de “wireframing” se concentre sur l’ergonomie, et utiliser de vrais textes lors de cette étape détournerait l’attention de cet objectif principal. Le wireframe n’est pas une maquette !

Un wireframe, aussi appelé “maquette fil de fer” en français, est donc une représentation filaire d’un site internet en noir et blanc, et dépourvus de couleurs, de choix de polices, de logos qui décrit la taille, l’organisation et l’emplacement précis des éléments de la page, les fonctionnalités du site, moteur de recherche, les boutons d’action (CTA) et les éléments de navigation de votre site Web sans aucune notion graphique. L’aspect visuel ne sera traité que lors de la phase suivante de maquettage.

Cette étape est d’autant plus importante lors de la [définition d’une page produite sur un site e-commerce.](https://www.imagescreations.fr/comment-optimiser-les-pages-produits-de-votre-site-e-commerce/)

[Lors d’un atelier de conception web, le Wireframe place l’ergonomie au premier plan](https://www.imagescreations.fr/atelier-de-conception-web/) (simplicité de navigation, lisibilité, …). Les wireframes permettent aux différents membres du groupe projet de se projeter et de valider définitivement les fonctionnalités attendues.

Les Wireframes sont aussi parfois utilisés dans le cadre de la rédaction de spécifications fonctionnelles pour mieux présenter chaque fonctionnalité.

Créer des wireframes permet de rendre le processus de conception itératif. Il permet de gagner du temps lors de la phase de maquettage et de design graphique. Il permet ensuite d’en gagner également lors de la phase de réalisation car les équipes de développement comprendront plus facilement le résultat final attendu. Construiriez-vous une maison sans les plans en commençant par la décoration ? Probablement, non. Et bien c’est la même chose pour un site web. C’est assez logique et l’expérience montre que cela fonctionne !

**III.3 Analyse fonctionnelle**

**III.3.1 Cas d’utilisation (UseCase)**



**CAS D’UTILISATION**

En UML, un diagramme de cas d'utilisation permet de représenter les informations des acteurs d’un système, tel qu’une application logicielle, et leurs interactions avec ce dernier.

* du système : représenté par un grand quadrilatère, avec en haut le nom du système
* d’acteurs : représentant un rôle d’un utilisateur qui interagit avec le système. Il est symbolisé par un bonhomme bâton, avec son rôle en-dessous.
* de traits : nommés Association, représentant les droits d’accès à la fonctionnalité
* de traits fléchés : nommés Generalization, représentant la notion d’héritage (transmission des droits).
* de bulles : représentant les cas d’utilisation (un cas d’utilisation équivaut à une fonctionnalité) accessible par les acteurs auxquels elles sont reliées

Dans notre cas, les trois acteurs sont des personnes avec des droits différents d’accès à certaines fonctionnalités selon leur « statut ».

En effet, notre premier acteur « le Visiteur » n’a que peu de droits, s’en suit notre second acteur « le Membre » qui a accès à plus de fonctionnalités et va hériter des droits du « Visiteur ». Enfin notre troisième acteur « l’Administrateur » va avoir accès à toutes les fonctionnalités.

Comme évoqué dans la liste ci-dessus, nous remarquons qu’une flèche d’association relie les 3 acteurs : on parle de notion **d’héritage.**

Nous pouvons donc dire qu’un cas d’utilisation est un ensemble d’événements qui se produisent quand un acteur utilise une fonctionnalité pour achever un processus.

Nous pouvons noter qu’un diagramme de cas d’utilisation montre le comportement attendu du système mais qu’il n’indique pas l’ordre dans lequel les étapes sont effectuées. (*cf. Chapitre III.3.3*)

**III.3.2 Diagrammes d’activité**

1. **Diagramme d’activité de connexion**



1. **Diagramme d’activité de construction d’un deck**



Dans le langage UML, un diagramme d'activité fournit une vue du comportement d'un système en décrivant la séquence d'actions d'un processus.

Il est extrêmement utile pour illustrer un processus ou un flux de travail entre les utilisateurs et le système.

Fondamentalement, un diagramme d’activités est un organigramme illustrant les activités exécutées par un système.

Il est composé :

* **de swimlanes** (ou lignes de vie) :  représentées par des rectangles verticaux ou horizontaux. Ils indiquent où se situe les utilisateurs et les systèmes dans lesquels l’action se déroule.
* **d’un état initial** : représenté par un petit cercle plein. C’est le point de départ du flux du diagramme d’activité.
* **d’actions** : représentées par des rectangles aux bords arrondis. Ils indiquent le processus en cours, décrit de manière textuelle à l’intérieur
* **d’accept signal** : représenté par un rectangle aux coins tronqué derrière. C’est une action qui représente l'acceptation d'un événement. Une fois l'événement reçu, le flux qui vient de cette action est terminé.
* **de send signal** : représenté par un rectangle aux coins tronqué devant. C’est une action qui crée un message et le transmet à un objet cible, où elle peut déclencher un comportement.
* **de control flow** : représentées par des traits fléchés. Ils connectent les actions entre elles et indique le flux directionnel d’une activité. Une fois l’étape terminée, le flux se poursuit avec une flèche sortante.
* **de points de décision** : représenté par un losange. C’est le symbole d’une décision qui possède au moins deux embranchements pour permettre aux utilisateurs de voir les options dont dispose l’action.
* **de flow final** : représenté par un cercle vide barré d'un « *X »*. Il signifie la fin d’un procédé spécifique. Différent de l’état final, il ne représente pas la fin de tous les flux d’activités.
* **d’état final** : représenté par un cercle vide contenant un petit cercle plein. Il marque la fin d’une activité et l’achèvement de tous les flux du diagramme.

Ici, nous allons prendre comme exemple « le diagramme d’activité d’une création de decklist ». Tout d’abord le système d’information affiche la page relative au diagramme. C’est **l’état initial** du futur processus.

Par la suite l’utilisateur choisit de « Créer une decklist », c’est une **action**. Une fois **l’activité** achevée, un « **control flow** » (ou flèche) sortant poursuit son processus vers la prochaine **action**. Le système d’information et l’utilisateur communiquent donc entre eux jusqu’à la « Vérification du format de jeu » où une **décision** interroge le système d’information.

Si la condition demandée n’est pas correcte, il renvoie l’utilisateur à une **action** précédente afin de modifier son choix ou corriger ses données saisies pour une « Demande de connexion » par exemple *(cf. Diagramme d’activité de connexion).* Si la condition est respectée, le système indique à la base de données un « Affichage des cartes » mais la base de données peut avoir une erreur et envoyer un **send signal** suivi d’un **flow final**.

S’il n’y a pas d’erreurs, le système renvoie **l’action** en cours à l’utilisateur qui renvoie lui-même à nouveau une **action**. Le système d’information interroge la base de données afin d’effectuer **l’action** qu’il lui demande.

La base de données renvoie un résultat au système. Ce dernier est à nouveau confronté à une **décision**.

En effet, il peut y avoir un nouveau cas d’erreur en base de données comme mentionné plus haut. Enfin, dans le cas contraire, le système effectue la seconde **action** qui va effectuer une « Vérification du respect des contraintes de construction ». Le système se confronte à une dernière décision, un **send signal** de « message d’erreur » qui indique un « **flow final**» ou un autre **send signal** respectant les conditions qui va « Sauvegarder le deck » et terminé le processus par un **état final**.

**III.3.3 Diagrammes de séquence**

* 1. **Diagramme de séquence de connexion**



Un diagramme de séquence est un diagramme d’interaction dont le but est de décrire comment les objets collaborent au cours du temps et quelles responsabilités ils assument.

Il décrit un scénario d’un cas d’utilisation.

Un diagramme de séquence représente donc les interactions entre objets, en insistant sur la chronologie des envois de message. C’est un diagramme qui représente la structure dynamique d’un système car il utilise une représentation temporelle.

Les objets, intervenant dans l’interaction, sont matérialisés par une « **ligne de vie** », et les messages échangés au cours du temps sont mentionnés sous une forme textuelle.

Reprenons l’exemple de la « construction d’un deck » pour faire une suite logique avec notre diagramme d’activité. *(cf. page 18)*

Nous retrouvons donc nos trois acteurs représentés par trois lignes de vie distinctes : l’utilisateur, le système d’information et la base de données.

Sur ces lignes de vie peuvent se trouver des « barres d’activation » comme sur celle du système d’information. (cf. image …) Elle est utilisée pour indiquer qu’un objet est instancié lors d’une interaction entre deux objets, en l’occurrence notre membre, notre système ou notre base de données. La longueur du rectangle indique la durée pendant laquelle les objets restent actifs.

Pour commencer la séquence, l’utilisateur envoie un message synchrone au système, représenté par une flèche continue afin de lui demander l’accès à la page « créer un deck » sur la barre d’activation. Ce dernier renvoie une réponse, représenté par une flèche discontinue à l’utilisateur et lui donne l’accès à la page.

L’utilisateur « sélectionne un format de jeu » pour pouvoir débuter la construction de son deck selon le format. Le système reçoit à nouveau un message et exécute un message réflexif, c’est-à-dire qu’il s’envoie un message à lui-même. Il est indiqué par une flèche qui commence et se termine à la même ligne de vie. (cf. séquence 4)

Le système interroge la base de données sous forme de requête, que nous verrons plus tard dans notre code, pour « récupérer les cartes et contraintes » et lui renvoie une réponse. Il affiche ensuite selon les contraintes, les cartes relatives au format de jeu. L’utilisateur peut donc commencer à sélectionner ses cartes et construire son deck selon ses choix. Une fois les cartes choisies et les contraintes respectées, il envoie un message pour « sauvegarder son deck ».

Le système envoie à son tour un message sous forme de requête à la base de données qui va traiter ce dernier et effectuer à son tour un message réflexif. Cette dernière valide la requête et renvoie le résultat au système.

Enfin, le système « vérifie les conditions » via un message réflexif également. Selon, le respect des conditions, il valide la sauvegarde du deck ou informe l’utilisateur d’une erreur, le renvoie à l’étape 3 et l’invite à reformuler son choix.

Dans l’idéal, les étapes et la sauvegarde s’exécute correctement mais nous pouvons rencontrer des cas d’erreurs et des cas alternatifs. Le cas d’erreur, comme un problème avec la base de données, met fin au scénario. Le cas alternatif quant à lui, retourne l’utilisateur à une certaine étape du diagramme.

* 1. **Diagramme de séquence de construction d’un deck**



**IV. Conception**

**IV.1 Modèle Conceptuel des Données (MCD)**

****

**MODELE CONCEPT1UEL DEs DONNEES**

Faisant partie de la boîte à outil Merise, le MCD décrit les données utilisées par le système d’information et leurs relations. Les informations sont représentées logiquement en utilisant un ensemble de règles et de diagrammes :

* Les [entités](https://www.base-de-donnees.com/entite/) (1 rectangle = 1 objet) ;
* Les [propriétés](https://www.base-de-donnees.com/propriete/) (la liste des données de l’entité) représenter par un type (VARCHAR, DATE, INTEGER) auxquels ont peut attribuer une taille de caractères ;
* Les relations qui expliquent et précisent comment les entités sont reliées entre elles avec un verbe à l’infinitif. (les ovales avec leurs « lignes » qui se rattachent aux entités) ;
* Les [cardinalités](https://www.base-de-donnees.com/cardinalites/) (les chiffres et les lettres entre les relations). Par exemple, dans mon cas, l’entité « **POST** » ne peut « commenter » qu’une fois l’entité « **CARTE** » (0,1) mais cette dernière peut être commentée par plusieurs « **POST** » (0,n)

Utilisé assez tôt en conception de base de données, le **MCD** sert de base de travail et sera ensuite utilisé par les autres outils de Merise, à savoir le [**MLD**](https://www.base-de-donnees.com/mld/). Le MCD constitue une étape très importante de la modélisation. Si cette tâche est mal réalisée, des erreurs en cascade se produiront et rejailliront sur le MLD, sur le script SQL et donc sur la base de données finale.

Le **MCD** est une donc représentation graphique qui permet facilement et simplement de comprendre comment les différents éléments sont liés entre eux.

**IV.2 Modèle Logique des Données (MLD)**

****

**MODELE LOGIQUE DEs DONNEES**

Le **MLD** ou **Modèle Logique des Données** est simplement la représentation textuelle du [**MCD**](https://www.base-de-donnees.com/mpd/). Il s’agit juste de la représentation en ligne du schéma représentant la structure de la base de données. Il n’y a pas de travail poussé à réaliser à cette étape, il s’agit juste d’appliquer quelques règles toutes simples.

On représente ainsi les données issues de la modélisation [Merise](https://www.base-de-donnees.com/merise/) sous la forme suivante :

* Chaque rectangle représente une [table](https://www.base-de-donnees.com/table/) (nommée entité en MLD) ;
* Les [attributs](https://www.base-de-donnees.com/champ/) de ces tables sont listés les uns en dessous des autres ;
* Les tables d’association qui découle des cardinalités du MLD (par exemple si deux entités ont des cardinalités 0,n entre leurs relations)
* Les [clés primaires](https://www.base-de-donnees.com/cle-primaire/) (Primary Key) sont soulignées et placées au début de la liste des tables avec un « id » ;
* Les [clés étrangères](https://www.base-de-donnees.com/cle-etrangere/) (Foreign Key) sont contenus dans les tables d’associations.

**IV.2 Script SQL (à déplacer en ANNEXES)**

****

Le langage **SQL** (Structured Query Language) est un langage informatique utilisé pour exploiter des bases de données. Il permet de façon générale la définition, la manipulation et le contrôle de sécurité de données. Pour mon projet il m’a permis d’importer tout mon MCD dans ma base de données *(cf. page 47)*

Dans la pratique, le langage **SQL** est utilisé pour créer des tables, ajouter des enregistrements sous forme de lignes, interroger une [base de données](https://www.journaldunet.com/developpeur/base-de-donnees/), la mettre à jour, ou encore gérer les droits d'utilisateurs de cette base de données.

**IV/ Développement du Projet**

**IV.1 Outils couverts par le Projet**

* Analyse fonctionnelle / Conception

****

**StarUML**

Logiciel de modélisation UML (Unified Modeling Language) open source.

Je l’ai utilisé pour tous les diagrammes de mon analyse fonctionnelle (cf. pages 18 à 23).

****

**JMerise**

JMerise est un outil de modélisation de MCD. Il permet la création du MCD et la génération du MLD et du script SQL de création des tables. (cf. pages 24 à 26)

* Langages d’interfaces web statiques et dynamiques

 **** 

**HTML5 CSS3 JavaScript PHP5**

* Logiciels d’hébergements et applications web



**XAMPP**

Logiciel permettant de mettre en place un [serveur web](https://fr.wikipedia.org/wiki/Serveur_Web) local, un [serveur FTP](https://fr.wikipedia.org/wiki/Serveur_FTP) et un [serveur de messagerie électronique](https://fr.wikipedia.org/wiki/Serveur_de_messagerie_%C3%A9lectronique).



[Application](https://fr.wikipedia.org/wiki/Application_Web) web de gestion pour les [systèmes de gestion de base de données](https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_base_de_donn%C3%A9es) [MySQL](https://fr.wikipedia.org/wiki/MySQL) et [MariaDB](https://fr.wikipedia.org/wiki/MariaDB), réalisée principalement en [PHP](https://fr.wikipedia.org/wiki/PHP).



[**https://www.github.com**](https://www.github.com)**:** site internet d’hébergement et de gestion de logiciels, utilisant le logiciel de version Git. Je l’ai utilisé pour mettre mes dossiers et mon code. Ainsi, j’ai pu les sauvegarder en ligne.



**GitHub Desktop**

Logiciel qui permet d’interagir avec GitHub en utilisant une interface. Je l’ai utilisé tout au long de la formation pour sauvegarder, partager et sécuriser mes données.

* IDE (Integrated Development Environment)



**Visual Studio Code**

Éditeur de code source autonome qui s’exécute sur Windows, macOS et Linux. Il possède beaucoup d’extensions et peut prendre en charge à peu près n’importe quel langage de programmation. Je m’en suis servi tout au long de la formation

* Maquettage / Arborescence



**Balsamiq Wireframes**

Logiciel de conception qui permet de créer des maquettes, basé sur du glisser-déposer, facile d’utilisation. Je l’ai utilisé pour faire mon maquettage.



[**https://www.gloomaps.com**](https://www.gloomaps.com): Site internet spécialisé dans la création d’organigramme. Je l’ai utilisé pour faire l’arborescence de mon site.

**IV.2 Les contraintes de développement**

⇢ Limité dans le nombre de cartes dû à la volonté d’affichage "rapide" côté client malgré des fonctions asynchrones. En effet, mon API va chercher les images sur une autre API.

⇢

**IV.3 Présentation d’une API**

Commençons par expliquer mon choix de créer une API pour mon projet.

Tout d’abord la fonctionnalité que j’ai en premier lieu décidé pour mon projet, fût de pouvoir « filtrer » les cartes par attributs afin d’aider l’utilisateur à construire plus facilement son deck. C’est-à-dire qu’il puisse décider de choisir la couleur « rouge », de type « rituel » et de rareté « mythique » par exemple.

La priorité pour moi était donc de développer un **affichage dynamique** pour filtrer les cartes afin d’améliorer l’expérience utilisateur au maximum.

Après recherche et réflexion, deux choix s’offrait à moi :

⇢ Développer mon projet avec un modèle **MVC** et utiliser la technologie **AJAX** (Asynchronous JavaScript + XML) afin d’éviter le rechargement des pages à chaque clic.

⇢ Développer mon projet avec une interface d’échange de données moderne et rapide nommée **API** grâce à plusieurs méthodes notamment les fonctions asynchrones.

J’ai donc opté pour la deuxième solution et je me suis lancé dans la création d’un système d’**API** pour mon projet.

Cela a totalement correspondu à mes attentes lors des premiers tests. J’ai pu constater que les échanges entre mon application client et mon application serveur étaient très rapides et ne nécessitait pas de rafraîchissement de la page pour afficher les cartes, puis les filtrer.

C’est donc tout naturellement que j’ai décidé de développer mon projet autour de ce système.

1. **Qu’est-ce qu’une API ?**

Une API (acronyme pour ***Application Programming Interface***) est une interface de programmation.

Commençons par une analogie.

Nous allons prendre les anciens annuaires téléphoniques pour exemple.

⇢ Un **annuaire téléphonique** permet à un **humain**, de demander une information à un autre **humain**, par **téléphone**.

Sur le web, les API jouent le même rôle que les annuaires. Sauf que le téléphone est remplacé par internet et les humains par des ordinateurs.

⇢ Une **API**, permet à un **ordinateur** de demander une information à un autre **ordinateur**, par **internet**.

1. **A quoi sert une API ?**

Les API sont de plus en plus utilisées dans le milieu professionnel car elles répondent à plusieurs besoins. Elles permettent de **moderniser les structures des sites web et applications.**

Une API offre la possibilité aux entreprises d’ouvrir leurs données à leurs clients, fournisseurs et partenaires de manière sécurisée. Tout est interconnectable plus facilement, ce qui représente un **gain de temps et d’efficacité pour les équipes de développeurs.**

1. **Comment fonctionne une API ?**

Pour développer une API, il faut un serveur et un client. **Le serveur fournit et exécute le programme de l’API**. Il attend qu’on lui envoi une requête pour lui demander des données.

**Le client est un programme distinct qui demande et reçoit les données** transmises par l’API. Ce client peut être un site web, une application, une machine, un logiciel métier, un smartphone, etc…

Il faut considérer l’API comme **une interface d’échange de données entre plusieurs systèmes**.

Le client envoie une requête de données au serveur. Le serveur va extraire et collecter des données qu’il va ensuite renvoyer au client.

**Ce type de système permet aux différentes applications d’échanger entre elles.***(cf. schéma ci-dessous)*

****

**IV.4 Le Code**

**V -4. 1 Développement de l’interface HTML/CSS**

En préambule, je tenais fortement à préciser que je suis attiré depuis toujours par le graphisme, le design ou encore le montage vidéo. En effet, en tant qu’autodidacte, j’ai développé certaines compétences sur des logiciels du pack Adobe comme Photoshop, Illustrator ou encore Premiere et j’ai pu réaliser beaucoup d’infographies et quelques montages vidéo durant mon parcours personnel comme le logo de notre groupe en début de formation. Je vais donc vous présenter la page d’accueil.

**Etape 1 :**

Tout d’abord, j’ai décidé de commencer par la création d’une barre de navigation avec un « futur » logo, une barre de recherche, deux onglets et un espace « Mon Compte ».



*(Image 1)*

**Etape 2 :**

J’ai ensuite voulu présenter les « decks les plus populaires » sur la page d’accueil. Pour cela, j’ai intégré des images que j’ai redimensionné directement dans mon code HTML ainsi que les icônes de couleurs représentant la decklist. Pour finir, j’ai décidé d’intégrer et d’indiquer le format de jeu. *(cf. diagramme d’activité, page 21)*

Ici nous prendrons le « ***Niv-Mizzet 5C*** » comme exemple.



*(image 2)*

**Etape 3 :**

Une fois, l’interface HTML développé, j’ai voulu styliser et mettre en forme mon site avec CSS que j’ai au préalable « linké » à ma page HTML grâce à la ligne de code suivante : <link rel="stylesheet" type="text/css" href="../styles/index.css" />. J’ai tout d’abord intégré différentes polices au format « woff » et « woff2 » :



J’ai ensuite intégré un fond d’écran dans mon « body » (image 1, ligne 14), balise qui représente toute la page, et que j’ai voulu positionner de manière statique grâce à la propriété « background-attachment: fixed; »  :



**Etape 4 :**

J’ai décidé ensuite de m’occuper de la barre de navigation à laquelle j’ai attribué la classe « navbar » (en rouge) dans une « div » *(cf. image 1, ligne 16).*

Nous pouvons dire que cette classe est définit « avant » *(cf. page 42, premier paragraphe)* les différentes classes situées en-dessous (en bleu) :



Côté CSS, j’ai donc stylisé grâce à Flexbox, toute ma barre de navigation en imbriquant des classes dans des classes et en utilisant différentes propriétés tels que : display: flex; / align-self: center; / justify-content: center; ou encore la taille et la police intégrée plus haut avec : font-size / font-family



*(image 3)*

Nous pouvons également remarquer que j’ai déplacé les éléments grâce aux propriétés « margin » afin de positionner les éléments de la barre de navigation de manière plus précises comme la barre de recherche ou l’onglet « Bibliothèque ».

Enfin pour expliquer un peu plus en profondeur le principe de Flexbox, il faut savoir que CSS fonctionne en « cascade », comme son nom l’indique « Cascade Style Sheet ». On peut voir par exemple sur l’image 3, que plusieurs classes se suivent *(ligne 97)* sous la formede « .navbar .logoMonCompte ». Nous devons passer par ce processus pour cibler l’élément « .logoMonCompte » car il est « l’enfant » de la « div » « .navbar » qui est le parent.

**Etape 5 :**

J’ai ensuite créé l’habillage des « des decks les plus populaires » mais nous prendrons le « ***Niv-Mizzet 5C*** » pour exemple comme évoqué en **Etape 2**.

Ici, la manipulation de Flexbox était un peu plus compliquée car plus d’éléments à insérer et de manière précise. En effet, j’ai dû tout d’abord créer 6 « div » contenant 6 classes distinctes. *(image 2, ligne 51 à 63)*

Nous retournons sur CSS et nous remarquons qu’une pseudo-classe « nth-child(1) » se situe derrière la classe « icon\_deck » *(couleur rouge, image 4)* mais elle n’apparaît pas dans mon code HTML. Cette pseudo-classe permet de de devenir « l’enfant » de la classe « icon\_deck » que l’on nomme « parent ».

Pour le reste du code CSS *(image 4)*, il est relativement similaire en termes de propriétés par rapport à ce que j’ai détaillé dans l’**Etape 4.**

On notera tout de même la propriété « hover » qui permet de déclencher une propriété lors du survol du curseur sur l’élément ciblé. Ici la propriété « opacity » qui va réduire l’opacité d’un deck afin de rendre plus interactif le site et améliorer le confort visuel de l’utilisateur.



*(image 4)*

Voici le résultat :



**IV.4. 2 Récupération d’une API et insertion en base de données**

1. **Récupération des données via une API externe**

Tout d’abord, j’ai pu récupérer sur le site MTGJSON (<https://mtgjson.com/api/v5/GRN.json>) plusieurs données de cartes grâce à une API fournit en open source encodé au format de données JSON. J’ai par la suite utilisé le navigateur Mozilla Firefox pour décoder le format JSON et avoir un résultat sous forme de tableau.



1. **Insertion des cartes en base de données**

J’ai ensuite grâce à mon MCD, importer le SQL dans **phpMyAdmin** afin de créer les tables dans la base de données. *(cf. page 36)*

Nous pouvons donc voir que toutes les tables ont été importées



Ci-dessous, j’ai également inséré 12 attributs, traduit sous forme de colonne dans ma table « carte » qui nous servira plus tard.



J’ai donc en premier lieu, créé un fichier PHP dans lequel la variable « $bdd » *(ligne 4)* permettra la connexion à ma base de données via l’objet PDO*.*



Par la suite j’ai récupéré l’url de l’API sous format JSON que j’ai inséré dans une variable *«*$pageContent*» (ligne 3)* avec la fonction « file\_get\_contents»mise en forme grâce la fonction nativePHP« json\_decode »dans la variable « $jsonData *». (ligne 5*)



Pour l’insertion des cartes dans la base de données, j’ai dû faire une boucle «foreach» *(ligne 11)* afin de pouvoir parcourir toutes les cartes*,* et les insérer une par une.

Dans cette boucle, j’ai constitué plusieurs variables pour chaque colonne de ma table et dû mettre des variables à « NULL» *(ligne 16)* car comme évoqué en introduction, les cartes ont plusieurs spécificités et ne se correspondent pas.

J’ai également dû instancier plusieurs conditions «if» *(ligne 25 à 38)* avec la fonction native « isset» afin de vérifier si certains champs étaient remplis ou non pour une nouvelle fois éviter des erreurs d’insertion.

**

Enfin, j’exécute une requête SQL pour effectuer l’insertion finale avec une requête préparée (*ligne 54 à 56*).



Nous pouvons voir maintenant que toutes les cartes ont été insérées en base de données dans la table « carte ».



1. **Fonctionnement de l’API**

**Etape 1 :**

On crée une vue HTML pour notre page de création de decklist.



J’insère les icônes des couleurs du jeu pour m’en servir de filtre de *« couleur »*, et je crée également une liste déroulante pour filtrer les cartes par *« type »*.



*(image 1)*

Nous pouvons donc voir le résultat de la vue HTML, où les icônes de couleurs sont bien présentes ainsi que la liste déroulante.



**Etape 2 :**

J’ai créé un fichier Javascript qui va me permettre de déclencher les différents évènements d’affichage que je souhaite pour mon application, c’est-à-dire filtrer les cartes de manière dynamique sans rechargement de la page, d’où l’API et les fonctions asynchrones que nous verrons plus en détails plus tard.

En premier lieu, j’ai créé des constantes où je vais stocker des url qui vont récupérer toutes les cartes dans mon fichier backend que nous verrons plus tard.



*(image 2)*

Par la suite, j’ai utilisé les fonctions natives de Javascript pour créer les évènements. Ici ce sont seulement des clics.

 

*Event « Color »* *(image 3)* *Event « Type » (image 4)*

**Etape 3 :**

Comme nous pouvons le constater il y a plusieurs couleurs *(image 3)* et nous allons choisir la couleur *« blue »* comme exemple.

Nous créons donc une variable dans notre Javascript dans laquelle nous insérons la fonction « document.querySelector » qui va nous permettre de récupérer le contenu HTML ciblé, ici *« l’id #blue » (image 1, ligne 52).*

*Par la suite, la fonction* « add.EventListener » *(image3, ligne 154*) va réagir au clic, et déclencher la fonction« showBlueCards ». *(image 5)*



*(image 5)*

**Etape 4 :**

Nous remarquons que cette fonction est une fonction *« asynchrone ».* L’idée principale de l’asynchrone est que notre script Javascript puisse continuer à s’exécuter (ici nos cartes) pendant qu’une certaine opération plus longue ou demandant une réponse est en cours (la réponse de notre API). Elle permet donc un affichage plus rapide des pages et une meilleure expérience utilisateur.

Dans cette fonction, je créé une variable « $data »*(ligne 101, image 5)*dans laquelle je stock l’url de la constante« urlU »*(image 2, ligne 8)* précédé de la fonction« fetch »qui va attendre de récupérer les ressources au travers de notre API, dans notre backend, grâce à la fonction « await »(fonctionrelative à une fonction asynchrone)

**Etape 5 :**

Nous nous retrouvons donc dans notre fichier backend, qui va permettre de vérifier si les url transmises sont « présentes » grâce à la condition « isset », elle-même dans une condition « if ». *(image 6, ligne 25)*

******

*(image 6)*

**Etape 6 :**

Une fois, l’url vérifiée, la fonction « reqBlueCards » s’exécute.



*(image 7)*

Nous remarquons, que la fonction « include »*(ligne 158),* nous permet de nous mettre en relation avec notre base de données avant n’importe quelle requête SQL.

Dans la variable « $requête », j’utilise la requête SQL « SELECT »*(ligne 160, image 7)*que l’on va stocker.

Je crée par la suite une variable « $output »qui va aller parcourir grâce à la fonction « fetchAll » toute ma table ***« carte »*** en base de données et me permettre de stocker toutes les cartes contenant la couleur « bleue » sous forme de tableau.

Enfin, une fois cette fonction exécutée et le tableau retourné, on va l’encoder en format de données JSON grâce à la fonction « json\_encode »*(image 6, ligne 26)* et le renvoyer à l’API.

**Etape 7 :**

Nous retournons dans le script Javascript, et plus précisément dans la fonction « showBlueCards ».*(image 5)*

La réponse attendue par la fonction « fetch » étant revenu, je vais la stocker dans la constante « json »*(ligne 103, image 5),* dans laquelle je vais instancier un tableau du résultat de la constante « data ».

Je crée une variable « url\_image » *(ligne 106, image 5)* qui va stocker tous les « id » des images présentes dans la base de données contenant la couleur « bleue », récupéré grâce à la boucle « for ». *(ligne 109, image 5)*

Enfin, j’affiche dans la div « zone »*(image 5, ligne 57)* de manière dynamique lors d’un clic, toutes les cartes contenant la couleur « bleue » auxquelles je donne une taille en hauteur et en largeur.

Voici le résultat :



