TELECOM Nancy

Rapport du Projet Pluridisciplinaire d'Informatique Intégrative

Michael ENESCU
Emilien PETITEAU
Thomas BALDUZ
Promotion 2023

Responsables du module :
Sebastien DA SILVa
Gerald OSTER

29 Mars 2021 — 11 Juin 2021

Table des matières

Chapitre 1

Introduction au sujet

1.1 Contexte du projet

Ce projet a été réalisé dans le cadre du module de P2I (Projet Pluridisciplinaire d'informatique integrative) durant la première année de formation du cycle ingénieur sous statut étudiant à TELECOM Nancy.

Le concours Mines-Télécom rassemble plusieurs milliers d'élèves pour passer des épreuves écrites et orales afin de pouvoir accéder à de nombreuses écoles d'ingénieur.

Cependant, la grande quantité d'informations doit pouvoir être efficacement gérée par une base de données adaptée. Un site internet qui facilite l'affichage des informations essentielles et de représentations graphiques doit également être créé.

Ce projet permettra à l'ensemble du groupe d'étudier la création d'une base de données relationnelle, sa transposition dans un Système de Gestion de Base de Données (SGBD), l'implantation automatique des données et la réalisation de requêtes SQL pertinentes pour interroger la base de données. Il nous permettra également de mettre en œuvre une application web native et d'étudier son fonctionnement.

Le travail se décompose globalement en 3 parties :

- La manipulation des données fournies pour les rendre utiles à l'utilisateur
- La création d'une application web permettant de consulter la base de données produite
- La gestion de projet en équipe

1.2 Organisation du document

Dans le chapitre 2, nous effectuons une présentation générale du système de sélection des concours ainsi qu'une explication du fonctionnement standard d'une application web sous la forme d'un état de l'art.

Dans le chapitre 3, nous présentons la manière dont l'équipe s'est organisée et avec quels outils, ce chapitre est destiné à la gestion de projet.

Dans le 4ème chapitre, il est question de la conception et la réalisation de la base de données.

Dans le chapitre 5, nous présentons l'application que nous avons réalisés, avec les choix faits afin d'obtenir les meilleurs résultats et compromis.

Dans le 6ème et dernier chapitre, nous réalisons un bilan complet du projet avec un point de vue global et individuel de chaque membre de l'équipe projet.

1.3 Précisions légales

Ce projet est destiné à un usage purement scolaire, ainsi, les images présentes, notamment les images de test, ne sont pas destinées à la publication et ne sont pas toutes libres de droit.

Cependant, ce caractère strictement scolaire nous autorise à les inclure en accord avec :

- Code civil : articles 7 à 15, article 9 : respect de la vie privée
- Code pénal : articles 226-1 à 226-7 : atteinte à la vie privée
- Code de procédure civil : articles 484 à 492-1 : procédure de référé
- Loi n78-17 du 6 janvier 1978 : Informatique et libertés, Article 38

Chapitre 2

Etat de l'art

Tout d'abord, le système d'enseignement supérieur français permet aux étudiants ayant obtenus le baccalauréat de s'inscrire à une classe préparatoire aux grandes écoles. Pour une durée de un à trois ans, cet enseignement, généralement hébergé dans les lycées, vise à proposer à l'étudiant une formation exigeante et intensive pour ensuite intégrer une école. [11]

D'autre part, les données fournies sont stockées sous un format .xlsx ou .csv. Ce sont deux extensions de nom de fichier pour tableur utilisé par la suite Microsoft Office, une suite bureautique créée par l'entreprise Microsoft en 1992. [12]

Enfin, la consultation de données par l'utilisateur est permise par la création d'une application web.

Le thème du projet est très intéressant car il nous fait découvrir la chaîne de réalisation d'une application web : en partant de données brutes, nous arrivons à offrir à l'utilisateur des données porteuses de sens et facilement consultables. Ce sujet nous permettra aussi de découvrir différentes types de langages.

2.1 Concours scientifique

Il existe trois catégories distinctes de classes préparatoires : littéraires, économiques et enfin scientifiques. Le sujet s'intéresse ici aux classes préparatoires scientifiques. Cette catégorie utilise le modèle du concours pour procéder à la sélection des étudiants. Notamment, il existe un nombre limité de concours regroupant des banques d'écoles. Les plus connues sont Mines et Ponts, Centrale ou encore CCINP. Chaque banque érige ses propres critères de sélection. En 2020, 26976 élèves se sont inscrits aux différents concours scientifiques. [6]

Les données fournies par le projet s'apparentent à une épreuve type Mines-Ponts, soit une sélection en deux temps. La première phase se compose d'un ensemble d'épreuves écrites. Les candidats ayant été sélectionnés deviennent alors "admissibles". La deuxième phase est l'épreuve orale, à la suite de laquelle les étudiants sont "admis" ou "recalés".

2.2 Application web

Pour définir ce qu'est une application web, nous devons comprendre comment fonctionne Internet. Internet, soit un réseau de réseaux, se fonde sur le principe d'encapsulation, soit le processus d'inclusion d'une abstraction dans une autre. Internet possède en effet une architecture composée de différentes couches indépendantes, incluses les unes dans les autres à la manière des poupées russes. C'est grâce à cette architecture que nous pouvons communiquer avec les autres membres d'un réseau.

Ainsi, la première de ces couches est l'application, soit l'application layer. Cette couche concerne les applications que chaque utilisateur d'Internet utilise quotidiennement. Dans l'Internet des années 70, on a vu l'apparition des mails et du transfert de fichiers, puis Internet est devenu indispensable dans les années 90 avec l'apparition d'une nouvelle application : le web. Depuis, de nombreuses applications sont apparues comme le streaming de vidéos, les jeux en ligne ou plus récemment les réseaux sociaux.

L'application web est une application de type Client-Serveur. C'est une architecture où tous les hôtes sont réparties en deux catégories distinctes : d'un côté les clients, d'un autre côté les serveurs. Les clients sont les ordinateurs ou téléphones qu'on utilise directement, tandis que les serveurs sont des ordinateurs constamment connectés qui fournissent l'information aux clients. [4]

Venons-en maintenant à la définition de l'application web. Il s'agit d'un programme ne nécessitant aucune installation, et utilisable sur toute machine disposant d'un navigateur et d'une connexion internet. Elle permet de se connecter sur des sites web qui leur fournissent des informations à travers le protocole HTTP pour Hypertext Transfer Protocol. Les informations que le serveur envoie au client sont des fichiers aux différents formats comme HTML ou CSS. Selon le protocole HTTP, un client qui veut consulter un fichier sur un serveur doit envoyer une requête bien définie.

Prenons un exemple. Si le client souhaite consulter la page HTML qui s'appelle Bienvenue (Bienvenue.html) sur un site qui s'appelle www.meteo.com, il devra envoyer la requête suivante : GET www.meteo.com/Bievenue.html. Le mot-clé GET signifie que

l'utilisateur demande une information au serveur. Par contre, si le client veut envoyer une information au serveur, il utilisera le mot-clé POST. Le protocole HTTP définit aussi de nombreux codes de statut que le serveur va envoyer au client pour lui dire comment la requête s'est passée. Si tout s'est bien passé, il va envoyer un code 200 Ok au client. Au contraire, si la page qu'a demandé le client est introuvable, le serveur va renvoyer le code 404 NOT FOUND. Il existe de nombreux autres codes de statut. Pour gérer toutes ces commandes HTTP, plusieurs applications ont été créées. Du côté client, Chrome, Safari ou Firefox sont les navigateurs les plus connus. Du côté serveur, les applications les plus connues sont Apache et NGINX.

Une application web peut donc être séparée en deux catégories, ou deux points de vue différents. D'un côté, nous avons le front-end qui correspond à tout ce qui est en contact avec l'utilisateur du site. De l'autre côté, nous avons le back-end qui correspond au fonctionnement du site pour lui envoyer des réponses depuis le serveur.

2.2.1 Front-End

Plus précisément, le front-end est le code qui est exécuté sur la machine du client. Ce code se lance sur le navigateur du client et crée l'interface utilisateur. Un des objectifs principaux des développeurs front-end est de permettre une interaction fluide avec le client. En d'autres mots, le front-end d'une application web doit être intuitive et facile à utiliser. Ceci peut paraître simple, mais peut se compliquer rapidement dès que les supports changent. Une application développée sur un téléphone requiert une implantation différente que celle d'un ordinateur. Ainsi, les sites web doivent aujourd'hui fonctionner sur de multiples supports, avec des tailles d'écrans qui diffèrent d'un appareil à l'autre.

Pour assurer ce service, trois principaux types de code sont utilisées : HTML, CSS et JavaScript [5]. HTML est un langage de balise, il représente la structure du contenu de la page web. C'est lui qui contient le texte brut. CSS est un un langage de classe et s'occupe de la mise en page du code HTML. Enfin, JavaScript gère le code quis era exécuté sur l'ordinateur client.

2.2.2 Back-End

Parallèlement au front-end, le back-end est le code qui tourne sur le serveur, qui reçoit les requêtes clients pour renvoyer les données appropriées au client. Le back-end inclue aussi la base de données, qui stocke toutes les données de l'application comme les pages HTML.

Dans le cadre de ce projet, nous générons les pages de manière dynamique, c'est-à-dire que le rendu côté client dépend des informations données par l'utilisateur. Pour cela, nous utilisons le framework web Flask, utilisant Python3 [3]. Flask contient un controleur qui s'occupe de récupérer les pages HTML grâce à une vue, de remplir les informations manquantes grâce à un modèle, pour enfin rendre le code complété à un routeur pour l'envoyer au client. Différents langages existent pour formuler une requête à une base de données. Lors de ce projet, nous utilisons le langage SQLite.

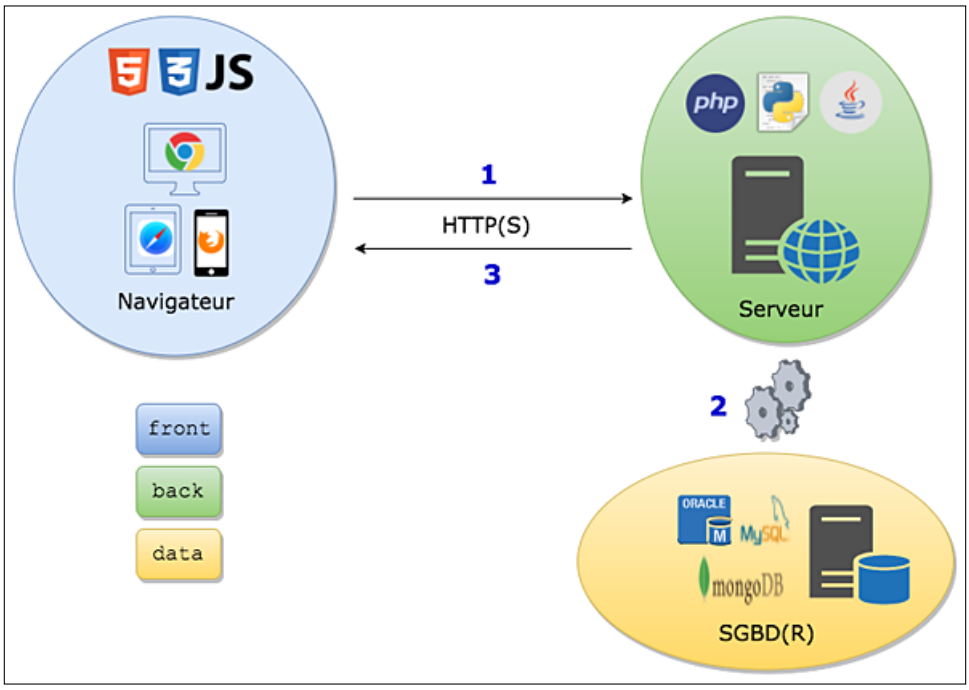

FIGURE 2.1 – Schéma d'une application web dynamique

Chapitre 3

Gestion du projet

3.1 Équipe projet

3.1.1 Composition de l'équipe

L'équipe projet est constituée de quatre étudiants de première année à l'école TELECOM Nancy. Emilien PETITEAU, Michael ENESCU et Thomas BALDUZ provenant tous de CPGE. Étant tous de niveau globalement équivalent, les tâches seront réparties équitablement et dans tous les domaines concernés par ce projet durant toute sa durée.

3.1.2 Organisation de l'équipe

Lors de ce projet, nous essaierons d'organiser une réunion par semaine afin de s'imposer un rythme suffisant pour envisager le reste du sujet avec calme. De plus, ces réunions nous permettront de jalonner notre avancement et d'éviter ainsi l'effet tunnel.

Pour collaborer entre les membres de l'équipe projet, nous utiliserons l'outil Gitlab présenté en cours. La rédaction du rapport se fera quant à elle sur Overleaf.

Pour la partie codage, chaque membre est libre d'utiliser son propre IDE tant que les versions des langages utilisés sont à jour. Chacun de nous utilisera le système d'exploitation Windows, et Michael utilisera de plus Unix pour vérifier que nos algorithmes fonctionnent sur tout OS.

3.2 Analyse du projet

3.2.1 Objectif SMART

Le management par objectifs consiste à identifier des objectifs quantitatifs et/ou qualitatifs sur une période définie.

	Critère	Indicateur	Application au projet
S	Spécifique	L'objectif est défini clairement.	Guidé par le sujet et les réunions d'équipe projet
M	Mesurable	L'objectif est mesurable, par des indicateurs chiffrés ou des livrables.	Application web permettant la visualisation des données, la recherche de candidats et autres. Rapport LaTeX sur la réalisation du produit, soutenance.
A	Atteignable	L'objectif doit être motivant sans être décourageant et doit apporter un plus par rapport au lancement du projet.	L'application doit paraître professionnelle, originale, avec différentes fonctionnalités. On doit cependant savoir choisir ce qu'on implémente pour gérer convenablement notre temps.
R	Réaliste	L'objectif est réaliste au regard des compétences et de l'investissement de l'équipe projet.	Objectifs réalisables de semaine en semaine sur des domaines connus (Python) ou facilement pris en main (LateX) ou bien à découvrir (HTML, CSS, js).
T	Temporellement défini	Il doit être inscrit dans le temps, avec une date de fin et des jalons.	Date de départ : 29 mars 2021 Projet sectionné en livrable intermédiaire.(différentes questions par semaine, fin des questions, rapport rédigé, présentation orale) Echéances redéfinies chaque semaine pour la semaine suivante, afin de gérer au mieux le temps Date de rendu : 11 juin 2021

3.2.2 Analyse des risques : matrice SWOT

La matrice SWOT permet d'évaluer les risques liés au déroulement du projet mais aussi de mettre en avant les qualités de l'équipe et les opportunités.

	Critère	Application au projet
S	Forces	-Bonne cohésion d'équipe -Bonne communication -Compétences informatique aguerries pour Emilien et Michael (spé info)
W	Faiblesses	-Beaucoup de travail en autonomie a été nécessaire pour comprendre des concepts n'ayant pas été traités en cours -Difficile de maintenir une activité régulière -On sort tous de classe préparatoire -Pas d'expérience en HTML, CSS et JS -Gestion du temps
O	Opportunités	-Aide possible venant des autres intervenants de l'école (professeurs, 2A, 3A) -Se démarquer en produisant une réalisation originale
T	Menaces	-Temps limité -Période d'examen et vacances -Attention à ne pas s'inspirer des autres groupes

3.3 Organisation du projet

Du fait de la spécificité du projet de ne pas avoir de fin, il est important de s'imposer un planning pour assurer une avancée continue. L'équipe se rend ainsi prête à accueillir la masse de travail supplémentaire donnée plus tard. Le projet a donc été segmenté en plusieurs étapes qui représentent chacune des check-points à atteindre.

Voici la répartition attendue sur l'étendue de la durée du projet, du 29 mars au 7 juin (nous ignorions la date de rendu précise à ce moment-là).

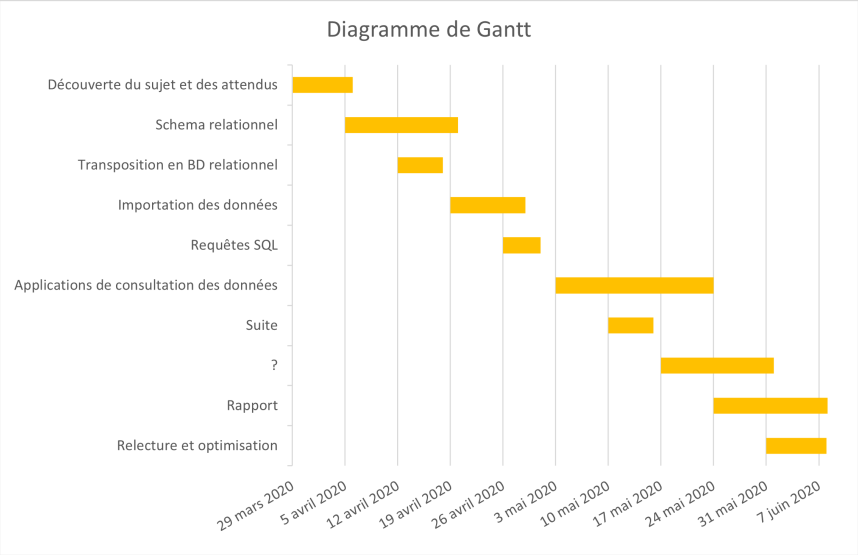

FIGURE 3.1 – Diagramme de Gantt prévisionnel du projet

Il nous a été jugé important de comparer cette prévision pré-projet à ce qui s'est vraiment passé. Nous pouvons alors remarquer les étapes qui ont été mal pré-visionnées et en tirer des conclusions sur l'organisation pré-projet en général.

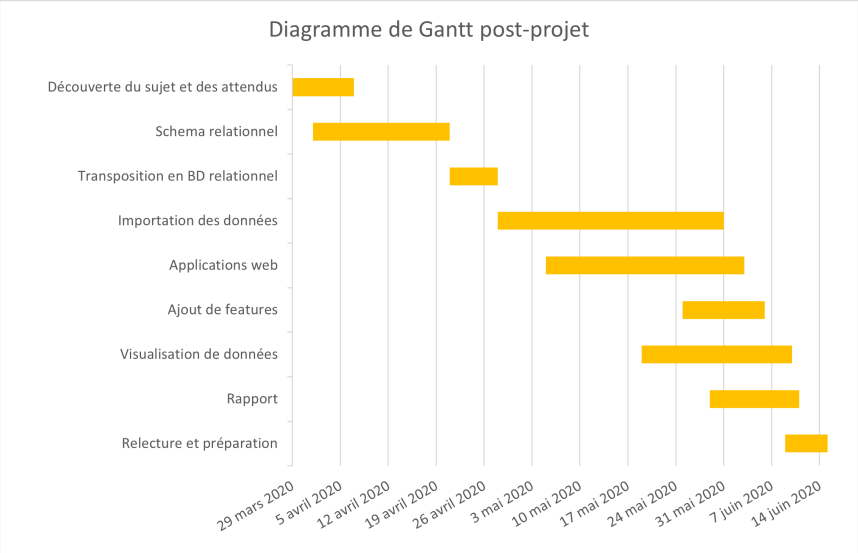

FIGURE 3.2 – Diagramme de Gantt post-projet

On peut donc voir que l'avancée du projet ne s'est pas passé comme prévu. La progression a été beaucoup moins linéaire que prévu. Ce qui est remarquable est l'accumulation des étapes vers début juin. Cette concentration de travail dans une période courte est à éviter car elle ne permet pas l'élaboration d'un code sain, c'est-à-dire performant et stable. Nous avons en effet perdu du temps sur des petits bugs que nous n'arrivions plus à détecter car nous faisons plusieurs tâches en même temps.

3.4 Comptes-rendus des réunions

3.4.1 Compte-rendu 1

Motif : Lancement du projet	**Lieu : Plateforme vidéo Discord**
Équipe projet : - Emilien PETITEAU - Michael ENESCU - Thomas BALDUZ	Date : 31 mars 2021 Durée : 15min Horaire : 10h30

Ordre du jour :
-Analyse du sujet
-Définition des premiers objectifs
-Répartition du travail

Mise au point :
-Découverte du sujet
-On constate que le projet semble fait pour être réalisé de façon linéaire.
-Accord sur la prochaine réunion et sur les taches à réaliser.

Décisions prises / Todo-List :
-Réaliser chacun de notre côté un schéma de base relationnelle en accord avec l'énoncé, l'objectif étant de les comparer et d'en choisir un (avec ou sans modifications) pour la suite du projet.

Date de la prochaine réunion : 7 avril 2021

3.4.2 Compte-rendu 2

Motif : Premier jet pour la base de donnée	**Lieu : En présentiel à Telecom Nancy**
Équipe projet : - Emilien PETITEAU - Michael ENESCU - Thomas BALDUZ	Date : 7 avril 2021 Durée : 2h Horaire : 16h

Ordre du jour :

-Mettre en commun les schémas de base relationnelles, en choisir un pour ensuite le compléter en 3ème forme normale et le normaliser

Mise au point :

-Le modèle retenu est celui de Michael
-Plusieurs questions se posent concernant les attributs des tables :

- Le code postale et l'adresse sont-ils liés ? (dans ADMIS_MP)
- Peut-on renommer les attributs ?
- Arrondissement est-il lié à Ville ?

Décisions prises / Todo-List :

-Vérifier le schéma relationnel chacun de son côté, notamment les liens
-Demander de l'aide à un professeur sur des questions précises
-Chercher une manière d'intégrer les avec les filières et les villes avec les pays

Date de la prochaine réunion : 21 avril 2021

3.4.3 Compte-rendu 3

Motif : Basa de données	Lieu : En présentiel à Telecom Nancy
Équipe projet : - Emilien PETITEAU - Michael ENESCU - Thomas BALDUZ	Date : 21 avril 2021 Durée : 40min Horaire : 14h

Ordre du jour :

-Trouver la version finale de la base de donnée

Mise au point :

-L'équipe met en commun les modifications trouvées pour la base de donnée.
-Discussion sur l'organisation pour la suite du projet, pour une possible répartition des tâches à venir.

Décisions prises / Todo-List :

-Chercher les outils à utiliser pour remplir la base de donnée sera suffisant, la semaine suivante est chargée.

Date de la prochaine réunion : 28 avril 2021

3.4.4 Compte-rendu 4

Motif : Organisation pour la suite	Lieu : Plateforme vidéo Discord
Équipe projet : - Emilien PETITEAU - Michael ENESCU - Thomas BALDUZ	Date : 28 avril 2021 Durée : 1h Horaire : 16h

Ordre du jour :

-S'organiser pour l'implémentation de la base de donnée et la suite du projet

Mise au point :

-On décide de l'organisation pour les prochaines semaines :

-Thomas : faire la moulinette pour récupérer les données des fichiers .csv et .xlsx
-Michael : implémenter la base de donnée et insérer les données dans la base de donnée
-Emilien : Commencer la création de l'application web avec laquelle on visualisera les données

Décisions prises / Todo-List :

-Avancer le plus possible dans nos tâches respectives
-Essayer d'avancer la base de données en particulier puisqu'elle est un élément important du site web.

Date de la prochaine réunion : 5 mai 2021

3.4.5 Compte-rendu 5

Motif : Implémentation et application	**Lieu : Plateforme vidéo Discord**
Équipe projet : - Emilien PETITEAU - Michael ENESCU - Thomas BALDUZ	Date : 5 mai 2021 Durée : 1h Horaire : 18h30

Ordre du jour :

-Suivi du progrès depuis la dernière séance

Mise au point :

-La base de donnée est implémentée mais les données ne sont pas insérées.
-La moulinette a malheureusement pris un peu de retard.
-Le site a été commencé, mais travailler avec la base de donnée va devenir nécessaire pour la suite.

Décisions prises / Todo-List :

-Travailler sur la moulinette pour récupérer les données et ensuite remplir la base de donnée.

Date de la prochaine réunion : 14 mai 2021

3.4.6 Compte-rendu 6

Motif : Application web	Lieu : Plateforme vidéo Discord
Équipe projet : - Emilien PETITEAU - Michael ENESCU - Thomas BALDUZ	Date : 14 mai 2021 Durée : 1h Horaire : 20h

Ordre du jour :

-On commence le site web

Mise au point :

-La plus grosse partie de la base de donnée est remplie. On peut maintenant sérieusement commencer à travailler sur le site web.
-Cependant, quelques tables posent encore problème, il faudrat travailler à les régler.

Décisions prises / Todo-List :

-Emilien et Michael travaillent à terminer la base de données.
-Thomas travaille sur le front-end du site.

Date de la prochaine réunion : 29 mai 2021

3.4.7 Compte-rendu 7

Motif : Application web et rapport	**Lieu : En présentiel à Telecom Nancy**
Équipe projet : - Emilien PETITEAU - Michael ENESCU - Thomas BALDUZ	Date : 29 mai 2021 Durée : 1h Horaire : 16h

Ordre du jour :

-Concentration sur l'application web.

Mise au point :

-La base de donnée est entièrement remplie. Des améliorations pourront être effectuées afin de rendre l'opération plus rapide. La durée d'insertion actuelle est entre 3 et 4 minutes. L'insertion des données pourrait être plus modulaire en fonctions des filières et permettre l'existence d'une nouvelle.
-Thomas a créé la page d'accueil du site, unanimement acceptée. -Emilien a commencé le rapport.

Décisions prises / Todo-List :

-Appliquer le style du site aux autres pages.
-Développer les fonctionnalités du site. Notamment l'affichage des tables en premier lieu.
-Réfléchir à avancer davantage le rapport.

Date de la prochaine réunion : 3 juin 2021

3.4.8 Compte-rendu 8

Motif : Journée PPII	Lieu : Plateforme vidéo Discord
Équipe projet : - Emilien PETITEAU - Michael ENESCU - Thomas BALDUZ	Date : 3 juin 2021 Durée : 7h Horaire : 8h

Ordre du jour :

-Essayer de terminer le site.

Mise au point :

-Emilien s'est occupé principalement des graphiques du site dans l'onglet statistiques à l'aide de la librairie JavaScript chart.js.
-Thomas a avancé significativement le rapport et optimisé l'insertion dans la base de donnée. Elle est par la même occasion moins dépendante des filières.
-Michael a aidé Emilien à créer les graphiques en proposant des requêtes SQL qui permettent une collection des données facilitée. Il a également travaillé sur beaucoup de points divers du site.

-Bien que l'insertion des donnée dans la base de donnée dépend moins des filières présentes, les notes des candidats ne sont pas positionnées de la même façon suivant les filières. Dans le cas où une filière est ajoutée, on ne peut donc pas prévoir la disposition des informations dans les documents à exploiter.

Décisions prises / Todo-List :

Les objectifs de cette semaine sont particulièrement consacrés à la rédaction du rapport, du moins à sa finalisation.
Il faut cependant s'occuper des derniers détails du site.

Dernière réunion.

Chapitre 4

Implémentation de l'application

4.1 Préambule

Afin de lancer la création de la base de données, nous devons lancer un certain nombre de fichiers dans un certain ordre. Tout d'abord, *Création-base.py* créé la base de donnée. Nous y avons rentré manuellement le nom de chacune des tables et éléments et avons défini le type de chaque éléments tout en définissant les clés primaires. Afin de s'assurer de l'intégrité de la base de donnée, nous avons définis des élément, notamment les clés primaires, les identifiant comme étant *Uniques*.

Une fois la base de donnée crée, nous la remplissions de façon automatisée colonnes par colonnes grâce au fichier *remplissage.py*. Celui-ci est un regroupement du code créé par chacun d'entre nous permettant un remplissage automatisé de tous les fichiers. Il détecte ainsi automatiquement le nom du fichier et connaissant sa structure incorpore à la base de donnée la colonne souhaitée. Il nous semble tout de même qu'il ne soit pas possible de rajouter un autre ensemble de fichiers avec une structure identique aux fichiers que nous possédons avec cette méthode étant donnée qu'ils n'ont pas tous la même structure : 2 fichiers de nom de structure identique possèdent les même éléments dans des colonnes différentes. Cela rend l'ajout d'une nouvelle filière impossible. Si cependant les fichiers possédaient tous le même ordre de colonnes, le rajout d'une autre classe par exemple serait possible. Il aurait été plus judicieux d'effectuer le remplissage par nom de colonnes, les noms étant identiques.

Nous avons ainsi créé et complété la base de donnée. Afin de travailler dessus, nous avons décidé de créer une application web notamment grâce à *Flask*. Le fichier python en question se trouve dans le dossier *Application*. Cette application web utilise du *HTML*, *CSS* et *js*. Nous avons une page d'accueil permettant d'accéder à toutes les fonctionnalités de notre projet par son menu. Tout d'abord, nous avons inclus l'option de parcourir manuellement la base de donnée. Ainsi, on peut cliquer sur le nom de chaque table afin d'accéder à son contenu. Par la suite, dans *requêtes* nous pouvons lancer la requête SQL de notre choix et obtiendrons la table correspondant à notre requête. Notre site contient

aussi un certain nombre de statistiques intéressantes notamment un calcul de données statistiques que l'on représente sous forme de diagramme, entre autre la provenance des personnes admises dans une école.

4.2 Création et remplissage de la base de donnée

4.3 Création théorique

Tout d'abord, nous avons effectué un certain nombre de simplifications de la base de donnée. Ainsi, nous avons retiré des informations redondantes comme la civilité étant à la fois représenté sous forme de nombres et d'un libelle, ne conservant que le libelle.

Par la suite, nous avons mis sous troisième forme normale [2] la structure de cette base de données. Afin de réaliser ce procédé avec plus de simplicité nous avons utilisé un outil informatique de création et de visualisation des bases de données qu'est dbdiagram.io. Nous avons donc tout d'abord mis notre base de donnée sous la première forme normale en s'assurant de l'unicité de l'information de chaque donnée, puis sous deuxième forme normale s'assurant que chaque élément d'une sous table dépendent des clefs primaires avant de mettre sous troisième forme normale où l'on a retiré la propriété de transitivité d'association d'éléments d'une base de donnée.

Par la suite, nous avons dénormalisé cette structure de donnée en regroupant un certain nombre de tables d'oraux et en retirant la table Bonification par soucis de simplicité et ajouté bonification dans les tables de résultats Écrits et Oraux.

4.3.1 Création informatique

La création de la table correspond au script python *creation_base.py*.

La librairie *SQLite3* de python est utilisé. Il n'y a aucun algorithme à détailler dans cette partie : chaque table de la base de donnée est successivement créée dans celle-ci à l'aide de requêtes *CREATE TABLE* faites à la main.

```
    cursor.execute('''CREATE TABLE IF NOT EXISTS concours
  (
  code_concours integer PRIMARY KEY,
  libelle_concours text,
  voie text,
  FOREIGN KEY (code_concours) REFERENCES inscription (code_concours)

)''')

```

4.3.2 Remplissage

Le remplissage des tables correspond au script python *remplissage.py*.

Divers librairies python sont utilisés dans ce script. Les plus importantes sont cependant *SQLite3* et *pandas*. La librairie *pandas* nous a permis de traiter les fichiers *.xlsx* et *.csv* fournis.

Le principe général du code est de générer une requête qui remplie l'intégralité d'une table en une requête. Il s'agit donc de concaténer chaque élément de la table à la requête.

La première partie du code est l'initialisation d'un dictionnaire qui contient l'ensemble des données fournies dans le *dow-master*. Il s'agit pour chaque fichier du répertoire de regarder son nom et son extension pour utiliser une variante d'une fonction de la librairie *pandas*. Cette fonction permet de récupérer les informations et les insérer dans la variable *dico*. Des variantes sont nécessaires car les formats des fichiers sont légèrement différents.

Pour les tables qui peuvent être intégralement remplies à l'aide d'un seul fichier et qui contiennent un élément par ligne du fichier, telle que la table *Candidat*, remplie à l'aide du seul fichier *Inscription.xlsx*, le remplissage est assez direct. On utilise le dictionnaire créé en début de programme pour récupérer les données sous forme d'un tableau facile à manipuler. On construit ensuite la requête en itérant sur chaque élément de de tableau. On sélectionne directement par la chaîne de caractère concaténée les colonnes qui nous intéressent pour remplir la base de donnée. On exécute alors la requête et on confirme les changements avec *c.execute("COMMIT;")*.

D'autres tables peuvent également être remplies à l'aide d'un seul fichier, mais le nombre d'éléments contenu dans la table est très inférieur au nombre de lignes de celui-ci. Cela est du au fait que les informations sont dupliquées. La table *csp* en est un parfait exemple. On utilise pour ces cas là les dictionnaires. Les clés du dictionnaire vont correspondre alors aux données de la clé primaire de la table à remplir. On parcourt alors l'ensemble des données du fichier en remplissant ainsi le dictionnaire. Chaque ligne du fichier apporte soit une nouvelle information, qui devient alors une nouvelle clé du dictionnaire ou est une information déjà récupérée qui ne change pas les données du dictionnaire. A partir du dictionnaire ainsi généré, on peut aisément construire la requête de la même façon que vu précédemment. Il n'y a plus qu'à l'exécuter.

```
    with app.app_context():
        df = dico["Inscription"]
        tab = df.to_numpy()
        c = get_db().cursor()

        dic = {}
        for row in tab:
            dic[row[45]] = row[46]
```

```
            dic[row[47]] = row[48]
        req = "INSERT INTO csp (cod_csp, lib_csp) VALUES "
        i = len(dic)
        for x in dic:
            req += f"(\"{x}\", \"{dic[x]}\")"
            i -= 1
            if i > 0: req += ", "
        req  += ";"
        c.execute(req)
        c.execute("COMMIT;")

```

Certaines tables nécessitent plusieurs fichiers différents pour êtres remplies. Ces fichiers sont le plus souvent d'un format identique ou très similaire. Ce sont en résumé les mêmes fichiers qu'on retrouve pour différentes filières. Quand les formats sont exactement les mêmes, on utilise une fonction particulière pour récupérer les données qui nous intéressent en fonction du nom du fichier. C'est le cas de la table *Resultats-Oraux*. La requête à effectuer est ensuite générée en récupérant le colonnes intéressantes pour toutes les lignes des tableaux. Le tout est effectué dans la même boucle. Le code est légèrement complexifié pour permettre à la requête créée d'avoir une syntaxe correcte. Il y a aussi le cas où les formats diffèrent légèrement, quand les colonnes à extraire sont différentes entre filières. C'est la situation de la table *CMT_Oraux_Spe* (inclue dans *Resultats-Oraux*) avec les fichiers *Classes_YY_CMT_spe_XXXX.xlsx*. On remplit alors la table en utilisant directement le nom des fichiers à utiliser. Des boucles différentes ici les plus simples à gérer. Cependant, à cause du format de ces fichiers, l'ajout d'une nouvelle filière ne sera pas complètement compatible avec notre programme à cause de ces variations qu'il est impossible de prévoir.

```
with app.app_context():
    c = get_db().cursor()
    c.execute("DELETE FROM CMT_Oraux_Spe;")
    req = "INSERT INTO CMT_Oraux_Spe (Numerodinscription, QCM_info_phy, Maths,
   Entretien_MT, QCM_Anglais) VALUES "

    df = dico["Classes_MP_CMT_spe_XXXX"]
    tab = df.to_numpy()
    for row in tab:
        req += f"(\"{row[0]}\", \"{row[25]}\", \"{row[26]}\", \"{row[27]}\", \"{row
   [28]}\"), "

    df = dico["Classes_PC_CMT_spe_XXXX"]
    tab = df.to_numpy()
    for row in tab:
        req += f"(\"{row[0]}\", \"{row[24]}\", \"{row[25]}\", \"{row[26]}\", \"{row
   [27]}\"), "

    df = dico["Classes_PSI_CMT_spe_XXXX"]
    tab = df.to_numpy()
    for row in tab:
        req += f"(\"{row[0]}\", \"{row[25]}\", \"{row[26]}\", \"{row[27]}\", \"{row
   [28]}\"), "

    df = dico["Classes_PT_CMT_spe_XXXX"]
    tab = df.to_numpy()
    for row in tab:
        req += f"(\"{row[0]}\", \"{row[24]}\", \"{row[25]}\", \"{row[26]}\", \"{row
```

```
    [27]}\"), "
25
26      df = dico["Classes_TSI_CMT_spe_XXXX"]
27      tab = df.to_numpy()
28      i = len(tab)
29      for row in tab:
30          req += f"(\"{row[0]}\", \"{row[24]}\", \"{row[25]}\", \"{row[26]}\", \"{row
    [27]}\")"
31          i -= 1
32          if i > 0: req += ", "
33      req += ";"
34      c.execute(req)
35      c.execute("COMMIT;")
36
```

Les candidats ATS ont en outre nécessité un traitement particulier du à leur absence au sein de fichiers qui ne leur étaient pas dédiés. Leur traitement n'a cependant pas apporté de problèmes majeur.

Nous avons fait en sorte que ce script de remplissage fonctionne sur les systèmes sous Linux et Windows à l'aide de la libraire *os* de python. Cette particularité s'applique également pour le fichier *app.py* qui fait fonctionner l'application web.

4.4 Application web

4.4.1 Navigation dans le site

La page d'accueil ne contient pas d'informations particulières à part les contacts. Elle sert à naviguer dans l'ensemble de notre application grâce à la barre de menu. Sur cette barre de menu, nous trouvons les fonctionnalités de notre site :

- Nous avons un ongle statistiques, nous y retrouvons les principales statistiques que nous avons effectué : provenance par pays (étant donnée que les adresses sont fictives) des candidats admissibles et admis ainsi que la provenance par pays des candidats qui demandent une certaine école.
- Nous avons un ongle permettant de parcourir notre base de donnée.
- Une section est dédié aux requêtes SQL nous permettant d'y entrer les requêtes de notre choix.
- Un certain nombre de recherche de candidats par ID, nom, classe...

4.4.2 Création du site : partie python/HTML

Pour la création de ce site, nous avons utilisé Flask. Nous avons un certain nombre de fonctions spécialisées pour créer les différentes pages représentant chacune des tables de notre base de donnée et les pages HTML qui leur correspondent. Cette implantation est un peu lourde nécessitant beaucoup de fichiers HTML qui sont tous quasiment identiques. Cette implantation aurait pu être moins lourde notamment utilisant la méthode

utilisée pour les fonctions de recherche crées : puisque toutes les pages ont un code identique, une seule page avec des variables d'entrées différentes aurait suffit.

Tout de même pour simplifier cette implantation nous avons utilisé une fichier de base *base.html* afin d'éviter de répéter tous les headers dans tous les fichiers et n'avoir à l'écrire qu'une seule fois. [7]

Les fonctions de recherche utilisent la méthode *POST* et ne nécessitant chacune qu'une seule page HTML.[8]

```
@app.route("/recherche", methods=["POST", "GET"])
def requestt():
    if request.method == "POST":
        c = get_db().cursor()
        requ= request.form["Req"]
        rec = "SELECT" + requ
        c.execute(rec)
        return render_template("/tables/resultat_recherche.html", results= c.fetchall
    ())
    else:
        return render_template("/recherche.html")
```

Afin d'afficher l'intégralité des tables, nous avons utilisé la requête SQL suivante *SELECT name FROM SQLite master WHERE type='table'*, nous permettant d'avoir accès au nom de tables.

```
def tables(conn):
    cursor = conn.execute("SELECT name FROM sqlite_master WHERE type='table';")
    tables = [
        v[0] for v in cursor.fetchall()
        if v[0] != "sqlite_sequence"
    ]
    cursor.close()
    return tables
```

Puis vient un grand bloc de fonctions très similaires qui consistent à attribuer à une route une certaine page HTML où l'on fournit des variables représentant les requêtes SQL requises pour chaque page. Nous utilisons pour beaucoup de ces fonctions deux requêtes.

```
@app.route("/ListeEcoles")
def listeEcole():
    c = get_db().cursor()
    h = get_db().cursor()
    h.execute("PRAGMA table_info(ListeEcoles)")
    c.execute("select * from ListeEcoles")
    return render_template("/tables/ListeEcoles.html", results= c.fetchall(),
   results2= h.fetchall())
```

En effet, ne voyant pas comment importer le nom de chaque colonne en une seule requête, nous avons effectué une seconde requête passée en argument à la page HTML ne contenant que les entêtes et combine ces deux tables dans le fichier HTML.

```
<table class="table table-striped">
  <thead>
    <tr class="table-secondary">
      {% for h in results2 %}
         <th scope="col">{{ h[1] }}</th>
      {% endfor %}
    </tr>
  </thead>
  <tbody>
    {% for r in results %}
    <tr>
      {% for x in r %}
        {% if (x != "nan") %}
          <td id="idd" scope="row">{{ x }}</td>
        {% else %}
          <td scope="row"></td>
        {% endif %}
      {% endfor %}
  </tr>
  {% endfor %}
  </tbody>
</table>
```

La première boucle ici représente la création des entêtes des tables et la deuxieme boucle représente le contenu de la table.

4.4.3 CSS

Pour ce qui est de la structure des pages HTML, nous avons utilisé du code CSS pour décrire la présentation des différents éléments du site.

Toutes présentes dans le fichier style.CSS, les règles CSS controlent tous les éléments inclus dans l'application web tels que les titres(*h1, h2*), les paragraphes (*p*), les articles (*article*) ou sections (*section*).

Deux classes principales donnent de la cohérence à l'ensemble de l'application. Il s'agit des classes .header et .footer.

La classe .header permet de mettre en forme l'en-tête présente sur chaque page afin de faciliter la navigation. Elle est constituée du logo du site situé à gauche, ainsi que des différentes sections de l'application composant la balise <nav>.[9] Voici l'en-tête actuelle :

FIGURE 4.1 – En-tête de chaque page HTML

Ensuite, la deuxième classe est .footer qui contient le style du pied de page, présent lui aussi sur chaque page. La présence de ces 2 classes permet d'encadrer une page HTML quelconque en lui donnant un début et une fin, ce qui favorise l'expérience utilisateur.

Nous avons ensuite crées des classes pour divers éléments HTML afin que leur insertion dans la page soit harmonieuse et esthétiquement valable. La page d'accueil est concrètement la plus travaillée, car elle se compose de différents parties toutes gérées par des règles CSS. Nous donnons ici une capture d'écran de la page d'accueil.

FIGURE 4.2 – Page d'accueil de l'application web

4.4.4 JavaScript et graphiques

4.4.4.1 Le JavaScript dans son ensemble

D'autres part, nous avons utilisé du code JavaScript pour ajouter des fonctionnalités dynamiques à l'application web. La première fonction concerne les fichiers $Header.js$ et $Footer.js$ qui permettent d'ajouter à l'aide d'une simple ligne de code l'en-tête et le pied de page mentionnés plus haut. La ligne de code s'insère dans le fichier HTML et le script JavaScript la remplace par le contenu des fichiers Header.js et Footer.js. Les lignes de code sont les suivantes :

```
<head>
    ...
    <script src="static/js/Header.js" type="text/javaScript" defer></script>
    <script src="static/js/Footer.js" type="text/javaScript" defer></script>
    ...
</head>
<body>
    <header-component></header-component>
    ...
    <footer-component id = "contact"></footer-component>
</body>
```

4.4.4.2 Utilisation de charts.js

Les graphiques présents dans l'onglet Statistiques ont été réalisés à l'aide de la librairie Google Charts de JavaScript *chart.js*. [1]

La création des graphiques se fait à travers une fonction JavaScript *charts* qui prend en paramètre plusieurs éléments. Chaque élément contient les données requises pour afficher un graphe. La fonction est appelée quand la page est chargée. Elle s'applique sur les balises *<canvas>* présentes dans la page HTML *charts.html* dédiée à l'affichage de ces graphiques.[10] Les paramètres rentrés pour la fonction sont les dictionnaires récupérés en paramètre de la fonction *render_template* qui génère la page HTML. Ces paramètres sont des dictionnaires générés dans le fichier *app.py*, dans la fonction *charts*.

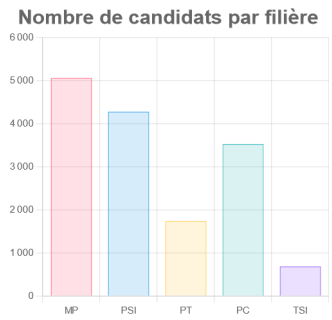

FIGURE 4.3 – Graphique présentant le nombre de candidats par filières

Enfin, nous aimerions utiliser utiliser d'autres fonctionnalités que nous offre JavaScript comme la sélection d'une filière qui afficherait la table correspondante tout en restant

sur la même page, ou encore la librairie Mappa qui permet de coupler données et carte géographique.

4.4.5 Recherches SQL ciblées

Les requêtes SQL sont à mettre sous leur forme complète et renvoie le résultat de recherches simples et complexes. Pour les recherches par nom des candidats, il est nécessaire de mettre des "" autour du nom pour recherche par nom et qu'ils soient en majuscules : ex *"LAPORTE"*. Initialement, la fonction de recherche permettait de rentrer toutes requêtes SQL que l'on souhaitait. Cependant, ceci a pour problème d'authoriser le *drop-table*, ce qui présente une faille de sécurité. Pour commencer à contourner ce problème nous avons rajouté l'obligation que la commande SQL commence par un *SELECT*. Ainsi, est concaténée à la requête SQL rentrée dans le *form* un SELECT au début de celle-ci, et la commande à rentrer dans *form* est du type : ** FROM Candidat*, reconnaissant ainsi *SELECT * FROM Candidat*

```
@app.route("/recherche", methods=["POST", "GET"])
def requestt():
    if request.method == "POST":
        c = get_db().cursor()
        requ= request.form["Req"]
        rec = "SELECT" + requ
        c.execute(rec)
        return render_template("/tables/resultat_recherche.html", results= c.fetchall
    ())
    else:
        return render_template("/recherche.html")


```

Une meilleure solution aurait cependant été de créer un système obligeant l'utilisateur à ne pouvoir sélectionner que les commander SQL permettant d'interroger la base de donnée, comme des cases pré-remplies.

4.5 Utilisation

Deux scripts ont été créés pour faciliter l'utilisation de nos productions :

- *fill_db.sh*
- *launch_app.sh*

Ces deux script se lancent à l'aide de la commande *bash* de WSL.

fill_db.sh permet de créer la base de donnée et de la remplir simultanément. Le script utilise les fichiers *creation_base.py* et *remplisssage.py*.

launch_app.sh permet de créer un environnement python, l'activer, d'y installer la librairie Flask et de lancer l'application web *app.py* en une seule commande. Il suffit ensuite

d'accéder à l'adresse 127.0.0.1 :5000 sur un navigateur pour accéder à l'application.

python3, *pip3* et *venv* doivent au préalable être installés sur WSL pur utiliser les deux scripts.

4.6 Fonctionnalités futures

Bien que ces prochaines fonctionnalités ne sont pas encore implémentées, il nous semble important de les exprimer ici pour au moins souligner leur faisabilité.

L'objectif principal de toutes ces données est de savoir quel élève intègre quelle école. Or les fichiers fournis ne nous donnent pas d'indications quant au nombre d'étudiants qu'accepte chaque école. Obtenir alors des résultats et travailler sur ceux-ci semble compromis si on ne choisit pas une méthode artificielle d'attribuer chaque élève à une école. C'est pour cela que nous avons établi la méthode suivante :

- Classer chaque élève suivant ses notes, et ceci pour chaque filière.
- Indiquer alors que chaque école prendra un nombre fixe d'élèves suivant chaque filière, soit le nombre d'élèves divisé par le nombre d'écoles pour permettre au plus grand nombre d'intégrer.
- Enfin, attribuer chaque élève à chaque école, en commençant par les mieux classés et par leur voeu le plus élevé.
- Dans l'éventualité qu'un candidat n'est pas eu son dernier voeu (car l'école avait déjà atteint son quota d'intégrés), il devra alors redoubler.

Une fois ceci effectué, nous pourrions alors produire une batterie de statistiques par écoles, notes, ou encore pays.

Chapitre 5

Bilan du projet

5.1 Bilan global du projet

Une fois le projet terminé, il est nécessaire de faire un bilan pour évaluer si les objectifs ont été atteints mais également afin de voir les points qui nous ont ralenti tout le long du travail.

Cette partie a pour objectif de mettre en exergue les difficultés rencontrées durant la réalisation de ce projet dans le but d'être plus efficace lors des prochains projets à réaliser en équipe mais aussi d'analyser les facteurs de réussite ayant permis de le faire aboutir ainsi que les bénéfices et l'expérience acquise.

Travail attendu	**Travail réalisé**
- Conception et mise en œuvre d'une base de données relationnelle -Réalisations de plusieurs composants logiciels informatiques dans les langages de programmation C, Java et/ou Python - Réalisation d'une étude bibliographique - Recherche de solution à un problème ouvert - Évaluation de la qualité des solutions proposées et des résultats obtenus - Mise en oeuvre d'une gestion de projet	- Recherches sur la création d'une base de données -Implantation longue des scripts d'importation de fichiers .xlsx due à la grande masse de données - Découverte des langages HTML, CSS et JavaScript - Création d'une application web - Exploitation de la base de données - Rédaction du rapport - Préparation à la soutenance

Justification de discordances :

Bien que nous pensons avoir répondu aux principales attentes du projet, il se pourrait que certains aspects aient mal été traités.

- Chacun de nous s'est concentré sur la résolution de problèmes singuliers et distincts entravant les attentes principales du projet. De ce fait, nous n'avons pas eu le temps d'exploiter au maximum les données fournies et aurions aimer leur donner plus de "sens".

- La phase de tests de notre réalisation n'a pas été suffisamment prise en compte tôt, ce qui résulte d'une grande quantité de code à tester d'un coup, réduisant la qualité des vérifications des solutions proposées.

	Points positifs	**Points Négatifs**	**Nouveautés**
Gestion de Projet	Équipe motivée Réunions fréquentes	Difficulté d'avancer à une vitesse régulière Mauvaise répartition des tâches	Projet reposant sur un problème ouvert Membres du groupe ne se connaissaient pas
Le livrable	L'ensemble des codes demandés ont été réalisés Prise en main de la solution proposée	Les codes n'ont pas été assez commentés Certains ont nécessités beaucoup de temps pour être mis en place	Découverte de nouveaux langages Utilisation parallèle de différents codes

5.2 Bilan des membres de l'équipe projet

Dans le but de gagner en efficacité sur les prochains projets, chaque membre de l'équipe projet aura à charge de se livrer sur ses ressentis concernant l'organisation de l'équipe ainsi que sur les objectifs du projet en lui-même.

5.2.1 Bilan personnel de Michael Enescu

Points positifs	Équipe organisée et motivée Développement des connaissances en WebBD conséquent avec cette mise en pratique situation ressemblant à un travail réel en base de donnée.
Difficultés rencontrées	Réunion en distanciel étaient moins efficaces que celles en présentiel.
Expérience personnelle/ressenti	Meilleure mise en place de la gestion de projet que celle eue en TOP. Travail relativement régulier de la part de tout le monde (mis à part une césure lors des vacances)
Axes d'amélioration	Travailler sur l'optimisation de la création de la base de donnée Recherche d'autres moyens de sécurisation de la base de donnée

5.2.2 Bilan personnel de Emilien Petiteau

Points positifs	Une ambiance agréable au sein de l'équipe projet, même si nous ne nous connaissions pas au départ Toutes les faces du projet ont pu être explorées Le sujet était assez large, j'ai beaucoup appris.
Difficultés rencontrées	Nous nous sommes parfois mal compris, ce qui nous a fait perdre du temps. On manquait d'information sur les documents fournis. Le cahier des charges n'était pas très complet. Le sujet était vaste, il y avait beaucoup à faire. En conséquence, il était fréquent de s'éloigner de nos tâches vers d'autres problèmes.
Expérience personnelle/ressenti	Un projet satisfaisant dans l'ensemble, je suis content du produit final. Les plus gros bémols à mon sens étaient des petits soucis de communication et une mauvaise gestion de git. La base de donnée était délicate à concevoir.
Axes d'amélioration	Davantage de réunions en présentiel me semblent nécessaire Communiquer un peu plus hors des réunions Une bien meilleure utilisation de git est possible Eviter de s'éparpiller dans les tâches à réaliser en se fixant des objectifs plus clairs.

5.2.3 Bilan personnel de Thomas Balduz

Points positifs	Chacun des membres de l'équipe avait le souci de bien faire et était facilement joignable Beaucoup de nouvelles connaissances acquises, notamment tout ce qui concerne le front-end d'une application web Un sentiment de réussite d'avoir trouvé une solution à chaque problème qui s'est présenté
Difficultés rencontrées	Les consignes ouvertes du sujet dures à appréhender Une perte de temps sur des petits problèmes comme une faute de frappe non détectée Parfois, deux personnes faisaient la même chose sans le savoir Un manque de deadline qui permettait difficilement de se projeter Une gestion du git assez brouillon
Expérience personnelle/ressenti	Beaucoup de nouveaux savoirs, mais j'aurais mieux profiter de ce projet et de ce qu'il pouvait offrir avec une meilleure gestion du temps Méthode SCRUM adoptée au premier projet à privilégier
Axes d'amélioration	Prévoir des réunions à fréquence fixe pour les prochains projets Laisser plus de temps à la phase de compte rendu. Utiliser les fonctionnalités de git

Bibliographie

[1] CHART.JS. *chart$_j s$*. URL : https://www.chartjs.org/.

[2] COURSALINE. *Explications formes normales sur 3 videos*. URL : https://www.youtube.com/watch?v=lHuhcuc-y7M.

[3] FLASK. *Flask tutoriel*. URL : https://flask.palletsprojects.com/en/2.0.x/.

[4] IDEEMATIC. *Application web*. URL : https://www.ideematic.com/dictionnaire-digital/application-web/.

[5] MOZILLA. *Front tuto (html, css,jss)*. URL : https://developer.mozilla.org/fr/docs/Web.

[6] SCEI. *Site du concours des cpge scientifiques*. URL : https://www.scei-concours.fr/.

[7] Tech With TIM. *Blocs$_E xtend$*. URL : https://www.youtube.com/watch?v=xIgPMguqyws&t=5s.

[8] Tech With TIM. *flask formes GET/POST*. URL : https://www.techwithtim.net/tutorials/flask/http-methods-get-post/.

[9] W3SCHOOLS. *navbar*. URL : https://www.w3schools.com/howto/howto_css_navbar_icon.asp.

[10] From Scratch - Développement WEB. *Créer des graphiques en 3 minutes avec Chart JS*. URL : https://www.youtube.com/watch?v=EwCx2GcwdMY.

[11] WIKIPEDIA. *Explciation cpge*. URL : https://fr.wikipedia.org/wiki/Classe_pr%5C%C3%5C%A9paratoire_aux_grandes_%5C%C3%5C%A9coles.

[12] WIKIPEDIA. *Histoire Mircosoft Office*. URL : https://fr.wikipedia.org/wiki/Microsoft_Office.