SpotYnov API - Gestion de Projet

Ce document vise à fournir une explication détaillée de l'organisation et de la gestion du projet **SpotYnov API** réalisé par Théo COLLIOT-MARTINEZ, Marianne CORBEL, Tony DE DONATO et Nils VERNERET, dans le cadre du cours de **développement API**.

Le dépôt GitHub du projet ainsi que la procédure d'installation et les documentations fonctionnelles et techniques peuvent être trouvés ici.

1. Choix des technologies et environnements de travail

API

Par souci de simplicité et de rapidité de développement et puisqu'il s'agit d'une technologie connue de l'intégralité des membres du groupe, l'API a été réalisée sous **Express JS** (node v22). Afin d'ajouter de la rigueur au développement par le biais du typage strict, **TypeScript** a été ajouté au projet.

L'authentification utilisateur se fait au moyen de **tokens JWT** valides pendant 60 minutes, conformément au cahier des charges.

IHM (Interface web)

L'interface web a été réalisée en Vue 3, technologie appropriée à un petit projet et permettant un développement rapide. De la même manière que pour l'API, Javascript a été remplacé par **TypeScript**.

L'utilisation de stores a été faite via **Pinia**, et les tokens JWT de l'API sont stockés en **localstorage**.

2. Outils de gestion de projet

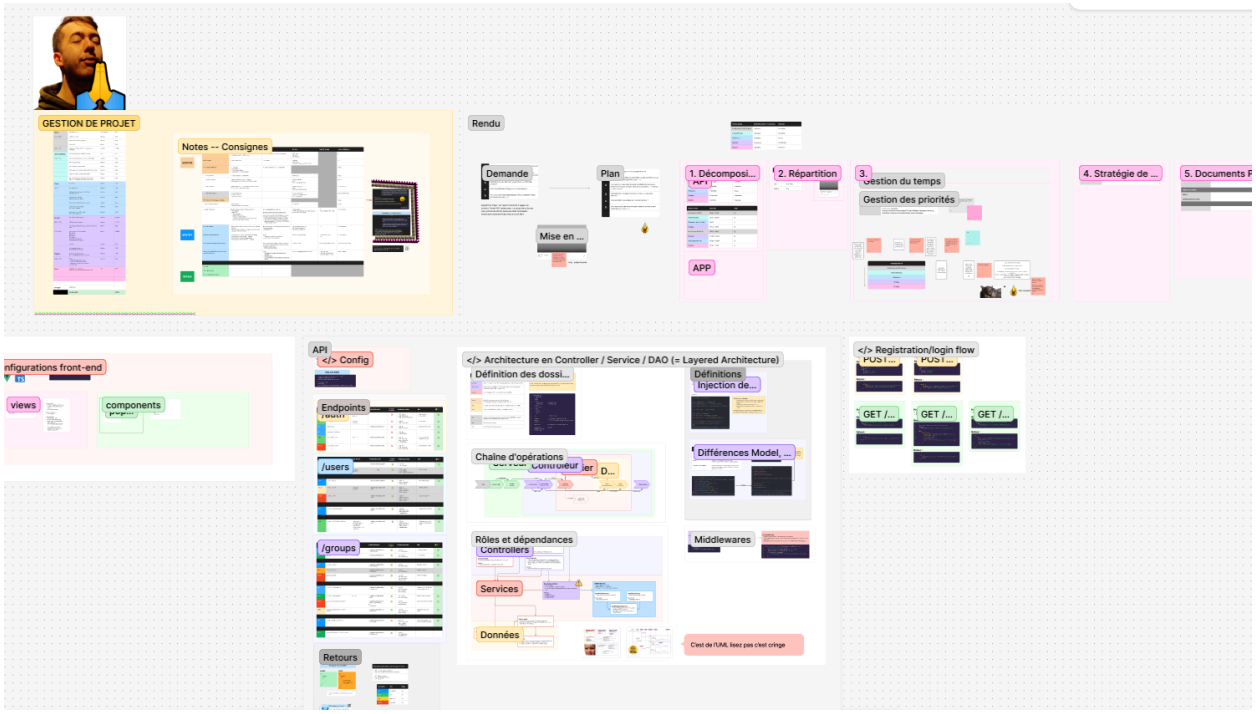
Versioning

Le versioning a été réalisé via **Git**, et le code-source du projet a été hébergé sur **GitHub**.

Toutes les fonctionnalités ont été réalisées sur des branches dédiées, préfixées du nom du développeur concerné.

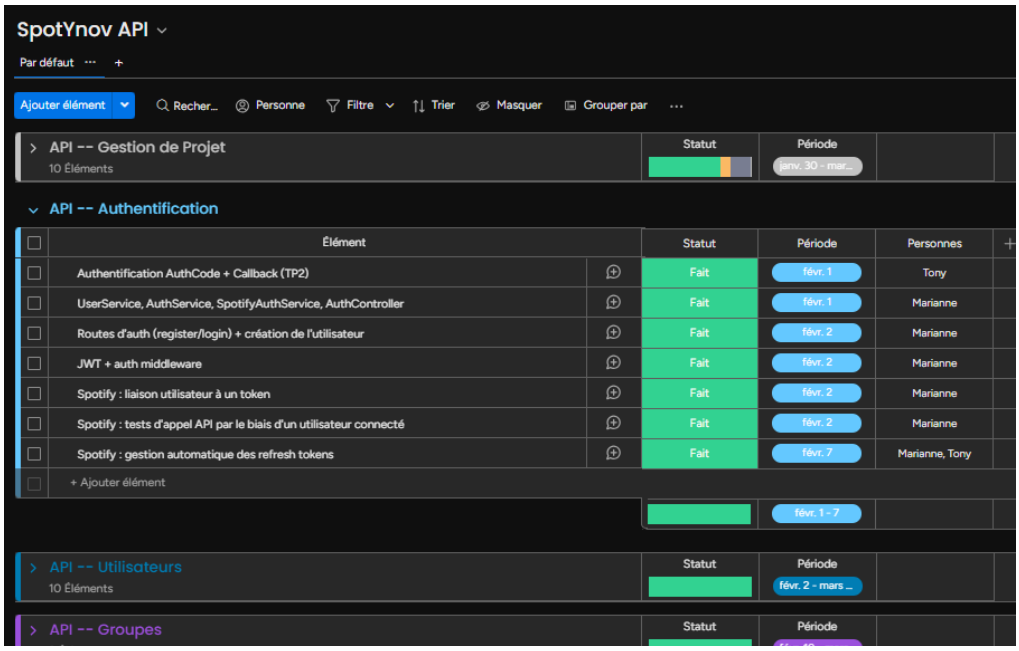
Gestion de projet

Toutes les étapes de réalisation du projet ont été réfléchies et documentées sur un canvas **Figma** accessible à toute l'équipe de développement et modifiable en temps réel.



Suivi des tâches

La séparation du projet en groupe de tâches, leur attribution et le suivi des délais de développement a été réalisé dans un premier temps sur le canvas **Figma** commun, puis transposé au propre sur un tableau **Monday**.



3. Décomposition du projet

Le projet a été décomposé comme suit :

Parties
Analyse des consignes et documentation
Architecture de l'API
Liaison à Spotify
Authentification
Utilisateurs
Groupes
Playlists
Bonus : IHM
Bonus : Conteneurisation

Le tableau précis des tâches de chaque partie se trouve ici (Monday, export excel).

Si l'architecture du projet a été pensée pour l'intégration de tests unitaires, la contrainte de temps ne nous a pas permis de confortablement nous lancer dans une approche de TDD (*Test-Driven Development*) ou d'inclure des tests unitaires au planning de développement initial.

Les tests ont donc été réalisés à la main au fur et à mesure de la création de fonctionnalités sur des branches séparées, et ce, avant leur bascule sur la branche principale.

4. Répartition de la charge de travail

La totalité des membres du groupe étant en rythme d'alternance ou en stage lors de la période de réalisation du projet, la répartition de la charge de travail a été faite de sorte à ce que chacun puisse confortablement concilier projets de cours et vie professionnelle.

De plus, puisque cette composition de groupe réalise d'autres projets de cours en parallèle, l'attribution des tâches au sens large se fait en fonction des affinités de chacun. Par conséquent, nous ne sommes pas à 25% de participation chacun, mais on y gagne du temps !

Ceci est compensé par une forte documentation du travail réalisé et un suivi précis du travail en cours, afin que chacun puisse suivre l'avancement de chaque projet.

Le travail a donc été partagé comme suit :

Analyse des consignes et documentation **Conception de l'architecture**	Tous
Développement de l'API	Marianne / Théo
Bonus : Développement de l'IHM	Tony / Nils
Bonus : Conteneurisation	Tony

5. Gestion du temps et des priorités

Nous avons décidé dès le départ que l'IHM serait partie intégrante du rendu. Cette approche nous a permis de diviser l'équipe en deux et de réaliser le développement des deux parties de manière concurrente, en estimant en conséquence les délais de réalisation.

La décomposition du projet énoncée au-dessus a également été réalisée dans un ordre de dépendance des éléments : par exemple, la partie **Utilisateur** dépendait de l'**Authentification**, et la partie **Groupe** dépendait de la réalisation préalable de celle **Utilisateur**.

Estimations du temps de développement

API	**Est.**
Analyse des consignes et documentation	0.5 sem.
Architecture de l'API	0.5 sem.
Liaison à Spotify	0.5 sem.
Authentification	0.5 sem.
Utilisateurs	1 sem.
Groupes	2 sem.
Playlists	1 sem.

Développement concurrent	**Est.**
Bonus : Conteneurisation	0.5 sem.
Bonus : IHM	3-4 sem.

Rétrospective

A peu près vers le milieu de projet, l'architecture de l'API a été revue pour intégrer pleinement la programmation orientée objet et l'injection de dépendance au projet, ceci afin de réaliser des tests pour l'API du projet fil rouge de Tony et Marianne !

Malgré ces changements, qui ont réussi à se limiter à seulement trois jours de développement, les délais ont été respectés et nous étions confortablement en avance de quelques jours côté API. Les deux tableaux suivants peuvent être retrouvés ici (Figma).

Enchaînement "idéal"	Estimation (total = 6 semaines)	Total réel
Documentation et analyse des consignes	0.5 semaine	0.5 semaine
Architecture de l'API	0.5 semaine	1 semaine
Spotify	0.5 semaine	1j
Authentification	0.5 semaine	1 semaine
Utilisateurs	1 semaine	3 jours
Groupes	2 semaines	2 semaines
Playlists	1 semaine	1 semaine
Bonus : IHM	3-4 semaines	3 semaines
Bonus : Conteneurisation	1 jour	1 jour

Enchaînement réel	Intervalle	Total
Analyse des consignes et documentation	28/02 - 30/01	3j
Architecture de l'API	28/02 - 30/01	3j
Spotify	02/02	1j
Authentification	02/02 - 07/02	6j
Utilisateurs (non-Spotify)	10/02	1j
Groupes	10/02 - 15/02	6j
Architecture OOP + DI	17/02 - 19/02	3j
Groupes	23/02 - 02/03	8j
Utilisateurs (Spotify)	02/03 - 03/03	2j
Playlists	05/03 - 11/03	7j

7. Analyse du sujet

Cahier des charges

Une analyse détaillée du cahier des charges a été réalisée afin de consigner toutes les vérifications et documentations nécessaires au bon déroulement du projet.

Cela nous a permis d'anticiper la segmentation des différentes tâches et de détailler un maximum de fonctionnalités avant de commencer le développement.

Le tableau détaillé peut être retrouvé ici (Figma).

Parties	Attentes	Notes	Routes	Spotify
FT4 - Liaison Spotify	• Scope utilisateur à bien définir pour couvrir tous les scénarios • OAuth2 ? • Où stocker le token spotify si l'Auth Bearer du header est déjà occupé par le token du Service ?	Stockage du dernier token de chaque utilisateur Spotify en JSON et **pas en base** • **Refresh token ?**	GET: /spotify/auth/authcode ? GET: /spotify/auth/implicit ? GET: spotify/auth/callback ?	• ?
FT6 - Personnalité de l'utilisateur	• Attrait pour la dance (entier de 0 à 10), voir **danceability** • Agitation (tempo moyen écouté), voir **tempo** • Instrumental ou non, voir **instrumentalness** • Attitude plutôt positive ou négative, voir **valence**	Etapes : • Récupérer les titres likés d'un utilisateur • Analyser les titres de l'utilisateur	GET: /users/[id]/personnality	• user-l • Ø
FT7 - Synchronisation des players d'un groupe		Impossible de modifier le playback state d'un utilisateur autre que celui connecté donc itérer sur les tokens des members d'un groupe puis : • Synchonisation des players	GET: /groups/sync/	user-mod
FT8 - Créer une playlist à partir des titres les plus écoutés d'un utilisateur		Etapes : • Récupérer les 10 musiques les plus jouées d'un utilisateur • Créer une playlist • Itérer sur les 10 musiques récupérées et les ajouter à la playlist	POST: /users/[id]/playlists/from-top-tracks/	• user-t • playli • playli

Définitions des routes

Une fois l'analyse du cahier des charges terminée, nous avons pu définir toutes les routes nécessaires au projet, leur paramètres, leur retour et les autorisations nécessaires à leur accès. Une première version de la documentation Swagger a été réalisée directement après la définition des routes et a pu servir de base à l'API. Une version 2.0 puis 2.1 ont été rédigées lors du développement subséquent.

Le tableau détaillé des routes peut être retrouvé ici (Figma).

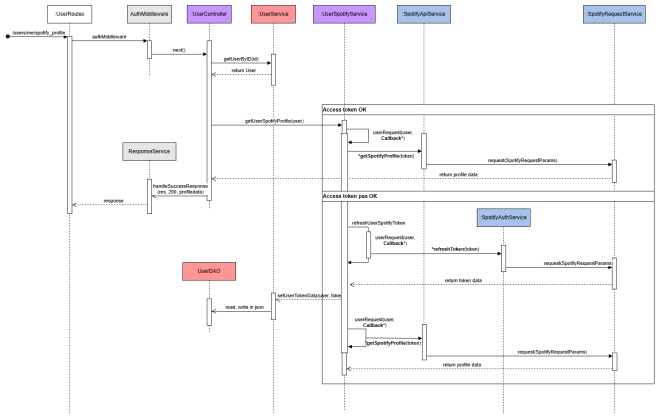
/auth

Méthode	Route	Body Param	Authentification	Accessible par Admin ?	Response codes	But	✅ // ❓
POST	/auth/register	• username • password		❌	• **201** : Created • **400** : Bad request • **409** : Conflict (already exists)	• Créer utilisateur • Génère token	✅
POST	/auth/login	• username • password		❌	• **200** : OK • **401** : Unauthorized	• Génère token	✅
GET	/auth/spotify		• **utilisateur (authenticateUser)**	❌	• **200** : OK • **401** : Unauthorized	• Spotify AuthCode OAuth2	✅
GET	/auth/spotify/callback			❌	• **200** : OK • **<Spotify Errors>**		✅
POST	/auth/spotify/link	• data : {...}	• **utilisateur (authenticateUser)**	❌	• **200** : OK • 400 : Bad request • **401** : Unauthorized • **<Spotify Errors>**	• Lie le token Spotify à l'utilisateur connecté	✅
DELETE	/auth/spotify/unlink		• **utilisateur (authenticateUser)**	❌	• **200** : OK • 400 : Bad request • **401** : Unauthorized	• Supprime le token Spotify	✅

Documentation des fonctionnalités-clés

Certaines fonctionnalités-clés complexes ont été documentées en détail en amont de leur réalisation.

Un diagramme de séquence a été réalisé pour décrire le processus de consommation d'un refresh token Spotify, et peut être retrouvé ici (Figma).



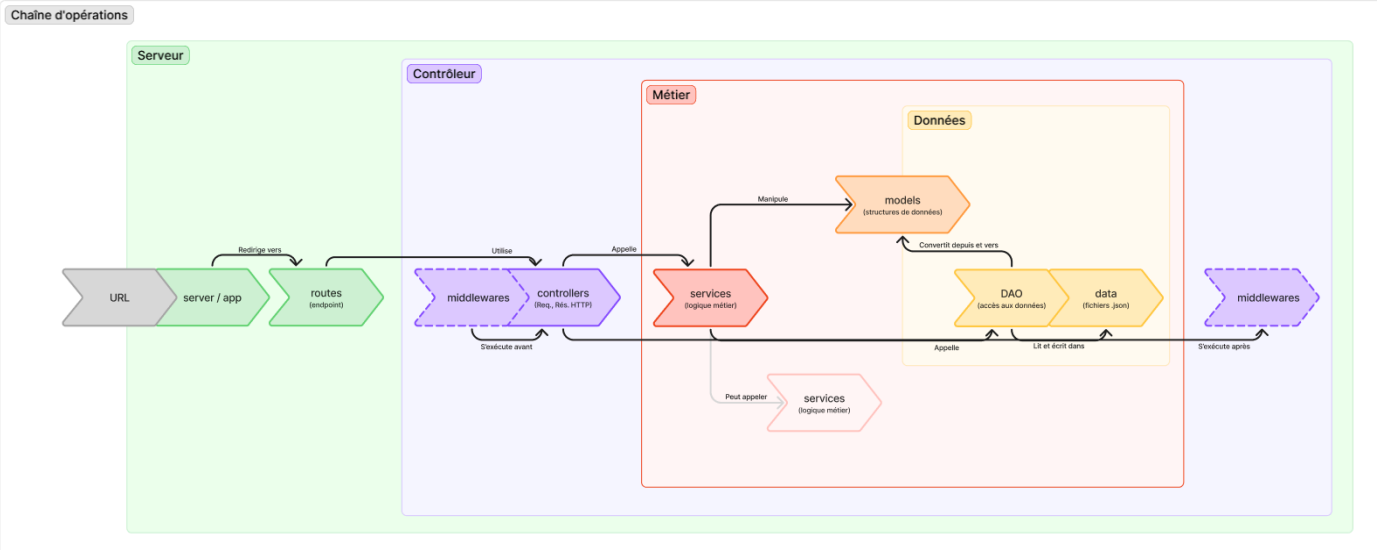
De même, tout le processus de registration / login a été documenté ici (Figma).

Définition de l'architecture

Comme précisé précédemment, ce projet a pu servir de test à une architecture dite "layered". Nous avons tenté d'appliquer un maximum de bonnes pratiques dans celle-ci, dont les principes de **Responsabilité Unique,** de **Séparations des Préoccupations** et celui d'**Ouverture/Fermeture.**

Le contrôle des dépendances et le cloisonnement des préoccupations ont pu être renforcés par de l'**injection de dépendance en cascade** entre *controllers* et *services*, ainsi qu'entre *services* et *DAO*.

Si cette implémentation n'est pas parfaite, elle a nécessité une grande rigueur et beaucoup, beaucoup de réflexion. Puisqu'il s'agissait d'une implémentation opérée par une partie du groupe, il a été nécessaire de documenter un maximum l'API afin qu'elle reste accessible et compréhensible au reste des membres.



Toute cette documentation (qui sort du spectre de la gestion de projet mais peut valoir le coup d'oeil) peut être retrouvée ici (Figma).