Cinquin Andy
B3 – C1 – Dev

Atelier développement d'une application mobile

Projet principal (avec le rendu à l'intérieur, les screens, et les vidéos)

https://github.com/CinquinAndy/FestixAPI

https://github.com/CinquinAndy/FestixAPI/tree/master/Rendu

https://github.com/CinquinAndy/FestixAPI/tree/master/Rendu/festix

https://github.com/CinquinAndy/FestixAPI/tree/master/Rendu/festix_admin

Recontextualisation

Le sujet était le suivant :

- Réalisation d'une API de gestion de festival permettant de :
  - Se connecter / déconnecter de manière sécurisée (tout le monde)
  - Consulter artiste / évènement / utilisateur (tout le monde)
  - Ajouter / supprimer / modifier un artiste / évènement / utilisateur (Admin)

- Réalisation d'une application mobile (Smartphone / Tablette) utilisant les technologies de votre choix.
  - Celle-ci doit permettre à un festivalier de :
    - Connaître la programmation du festival.
    - Connaître la fiche de chaque artiste.
    - Se retrouver dans le festival.
  - Un administrateur doit pouvoir :
    - Modifier la programmation du festival
    - Ajouté / modifié / supprimé des fiches artistes.
    - Ajouté / modifié / supprimé des évènements.

- Sous forme de dossier présentant :
  - L'application.
  - Les choix techniques.
  - Le fonctionnement de l'application.
  - Le fonctionnement de l'API.
  - La sécurité de l'application.
  - Les futures évolutions souhaitées.
  - Le dépôt de code.
  - Les éléments facultatifs que vous avez souhaité ajouter

Le projet est un projet individuel à rendre avant le 07/01/21 - 18H00 sur MylearningBox au format PDF

Table des matières

L’application

N’ayant pas un ‘vrai’ projet, et un ‘vrai’ client, nous allons prendre en compte ici que le projet a carte blanche et que le client nous fait aveuglément confiance pour la réalisation de ce dernier.

De manière générale l’application devra permettre d’avoir le programme du festival, de la manière la plus simple et agréable possible, et disponible sur le plus de plateformes possibles

L’utilisateur sera donc invité à télécharger l’application (choix du client, de faire une application mobile pour cela), et aura la possibilité de voir les différents évènements à venir, et de consulter le programme de chacun de ces évènements avec une description de l’artiste ou du groupe présent dans chaque partie de l’évènement général.

L’administrateur lui aura accès à une autre version de l’application, avec connexion obligatoire, et sécurisée, qui permettra de saisir des données (contrairement à la version ‘visiteur’ de l’application), et qui permettra à l’administrateur, d’ajouter, modifier et supprimés chaque élément de l’application.

Beaucoup d’application et de système fonctionne comme cela actuellement, cela à plusieurs avantages, déjà les taches sont séparées et l’application visiteur s’en retrouve plus légère, donc plus rapide à téléchargée

En plus de cela, le système est davantage sécurisé, en effet, une application à de droits ‘utilisateurs’ et l’autre n’en a tout simplement pas, de ce fait, on pourrait imaginer distribué à grande échelle et au public, l’application visiteur, et l’application administrateur elle serait envoyée via des dépôts privés, et donc, par essence, moins sensibles.

De plus aucune donnée privée d’utilisateurs, etc. ne pourrait se retrouver ‘menacée’ en cas de problème sur l’application visiteur.

Les choix techniques

Au niveau purement technique, voici les choix qui ont été faits :

L'api

V1.

L'api est fait en JavaScript

(NodeJS : qui est un environnement de backend, pour faire des applications côté serveur, avec du JavaScript, on s'en sert ici pour les modules et libraires liés au fait de supportés l'envoie et la réception de requête HTTP)

(Express : qui est un environnement basé sur nodeJS avec tous les éléments nécessaires et préconfigurer pour les requêtes HTTP)

(Sequelize : qui est un ORM basé sur les promises et sur le JS pour s'occuper de la partie 'BDD' sans que le développeur ne s'en occupe, on définit les différents model et la librairie se charge du reste)

Ces choix ont été faits pour plusieurs raisons, déjà en question de performances c'est tout à fait fiable, réactif et performant, tout autant que toute autre API ,

Les différentes de réactivités entre les différents langages sont souvent, dans le cas d'API négligeable,

La seconde, est tout à fait personnels, professionnellement, en parallèle des cours, je suis en free-lance et propose mes services à différents clients, j'utilise ce projet afin de faire un POC sur le stack Sequelize / Express / PostgreSQL, au niveau du back.

Également en termes de temps, je suis à l'aise en JavaScript, et donc ce genre de stack me permet de gagner beaucoup de temps sur le développement de l'application dans sa globalité.

V2

Toujours pour des raisons de 'POC', j'ai choisi de changer de fonctionnement d'api, Sequelize était extrêmement bien et la première version d'api fut extrêmement agréable à utiliser et efficace, mais très longue à mettre en place, pas simple du tout à maintenir, la techno est à mon avis trop jeune pour être utilisable à 100% de son potentiel dans un gros projet.

(Le lien de la v1 est toujours disponible, mais ne sera pas à jour par rapport aux différentes évolutions dût aux différents besoins lors du développement de l'application mobile)

J'ai envie d'essayer de faire une api en Java avec Spring et d'autres librairies qui servent à accélérer la vitesse de développement et la vitesse de mise en place d'un projet.

Pourquoi cette technologie, pour plusieurs raisons, la première, apprendre une nouvelle façon de faire des apis, de manière professionnelle,

La seconde, ce sont des compétences dont j'ai besoin en entreprise, cela me permet donc d'être plus efficace et efficient à moyen & long terme, pour mon alternance

La troisième cela me permet de faire un POC de cette techno également.

En termes de performance beaucoup de très grosses entreprises utilisent cette technique, c'est qu'il doit y avoir une raison en termes de rapidité de développement, de sécurité, etc.

J’ai donc envie d’essayer cet api (le 29 novembre) –

Le 08 décembre :

J’ai donc essayé cette technologie, qu’est le Java, en partant de ce que j’ai appris en entreprise, mes connaissances personnelles, et ce que j’ai appris en freelance

La stack est la suite : Spring (Framework Java) , Hibernate (ORM) , Flyaway (Version control application for databases), PgSQL (BDD) , MapStruct (Mappers) , Lombok (Facilitateur de création de classes / accesseurs / méthodes basiques = equals & to string par exemple)

Cette stack à tout un tas d’avantages, tout a déjà été fait au moins une fois, énormément de docs existent, énormément de personnes sont extrêmement compétente sur la question,

cela à un paquet d’avantages, on trouve facilement 10 versions de faire pour chaque solution, avec un peu de bidouille à chaque fois, on arrive à se sortir de tous les soucis qu’on pourrait rencontrer

Cet avantage est aussi son pire défaut à mon avis, on se retrouve avec pas mal de mauvaise pratique sur le web, exemple :

Lors de mes recherches, et de la construction de l’api, j’ai eu des soucis sur les retours à faire pour activé le X-CSRF, (pour éviter les attaques XSS) , les différents sujets sur stackoverflow ou autre forum d’aide, conseillaient simplement de désactivé le X-CSRF …

(À ne pas faire du tout, s’il est là, c’est pour une raison, c’est une sécurité non négligeable !)

Également, la vitesse de développement est rapide dans le développement en lui-même, pas mal d’outils pour augmenter la productivité (Lombok par exemple, avec un simple @data, on crée tous les getters / setter / et to string de la classe)

Mais ceci est contrebalancé très largement pas l’absence totale de hotreload, le rechargement du serveur tomcat est nécessaire pour update les différents éléments de la base de données !)

Finalement je suis assez mitigé, en ayant déjà testé des API sur Symfony, Laravel, Python Flask, Express (Js), Stapi.io, Firebase … et maintenant Java …

Java est très stable, et entièrement fonctionnel, on en restera là pour le choix de technologie.

Dans un cas de vrai projet, je pense que la question se poserait très sérieusement entre Java & Sequelize, les deux stacks ont de très sérieux avantages, les deux ont de très sérieux inconvénients …

Une dernière possibilité serait d'utiliser GraphQL qui révolutionne la façon de voir une API, mais, par manque de temps, le test de cette dernière technologie n'a pas pu être fait.

L'application

Pour ce qui est de l'application, le choix a été porté sur du Flutter,

La première raison, professionnellement et personnellement, en termes de curiosité, cela me permet de faire un POC sur une simulation de vrai projet, et donc d'avoir un recul sur l'utilisation réelle des technologies choisies,

Également, nous avons eu un module Flutter juste avant à l'EPSI et j'aimerais aller plus loin sur la technologie,

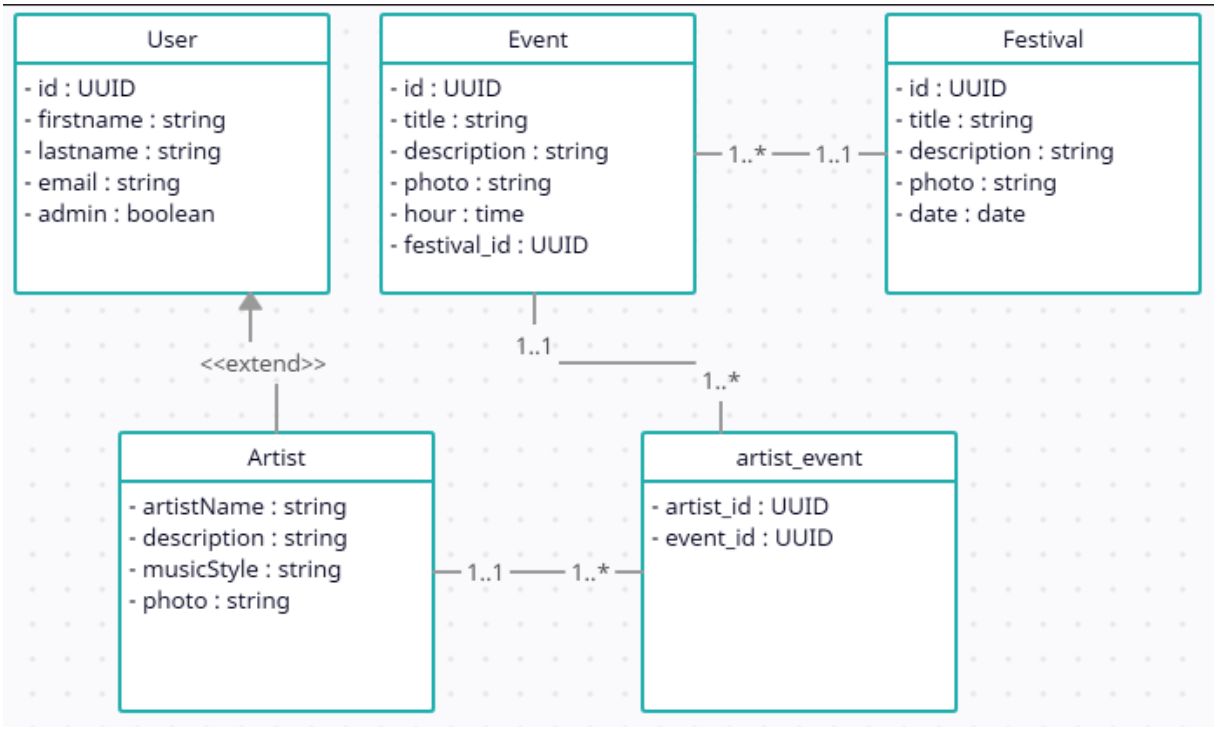
Et enfin, le choix était simple, Flutter, du Natif, ou du React Native, le flutter était, selon moi, le meilleur compromis en termes de confort de développement / rapidité de développement / performance,

React native aurait été le plus rapide, mais tout comme le natif, il aurait fallu développer une version Android et une version IOS,

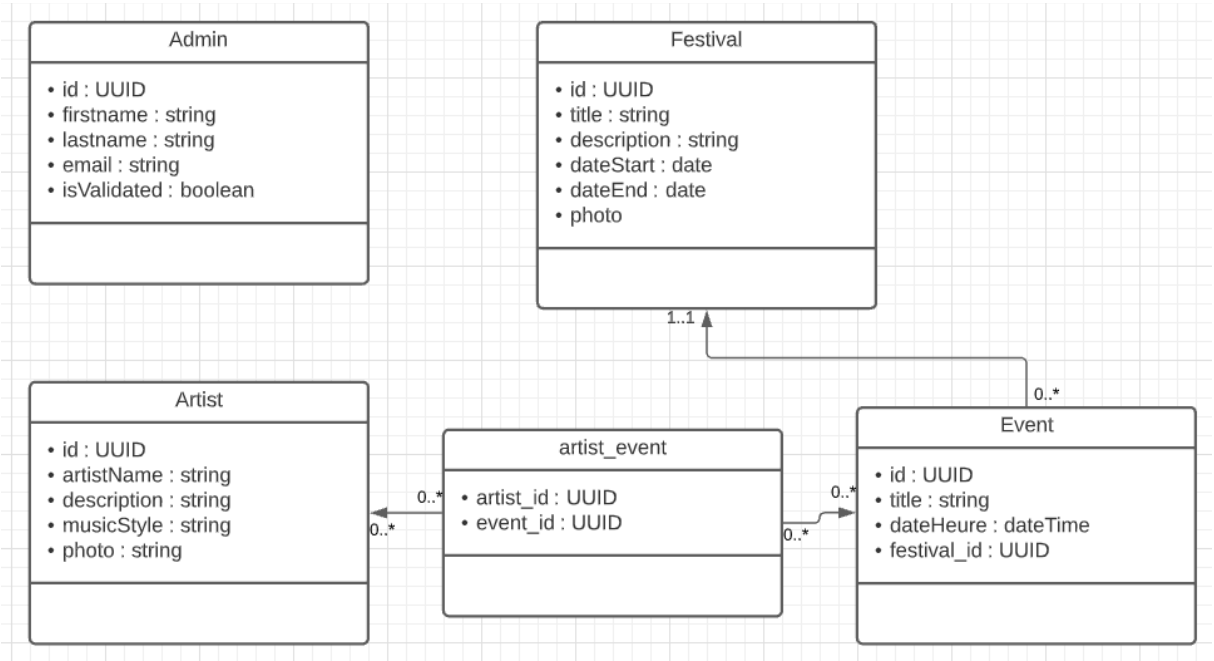
Alors que Flutter offre la possibilité de faire les deux à la fois.

Le fonctionnement de l'application

Le fonctionnement de l'api



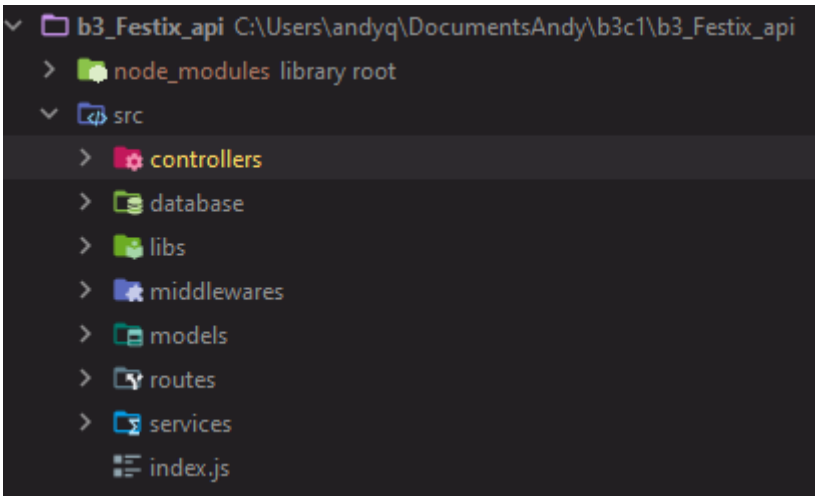
La première version du schéma uml proposé, après avoir fait la maquette, le schéma à évoluer en ceci :



Après corrections des cardinalités, et quelques changements (le login sur tous les utilisateurs n'étant plus nécessaires, il suffit d'avoir une table à part pour les admins n'ayant aucun lien, le reste, pas grand-chose ne change, mis à part quelques propriétés modifiées, supprimées et ajoutées.

Pour la version Javascript (sequelize) :

L'API est composée de plusieurs couches, nous allons les décrire ici



Tout d'abord, les contrôleurs c'est eux qui vont faire le lien entre la route, et le service, par convention les contrôleurs ont été implémentés mais n'ont pas d'utilité spécifique dans ce projet, nous pourrions très bien appelés les différents services directement

La couche database permet de gérés les migrations SQL, et de créer les model en bases de données, les différents tables et schéma de base de données

La couche libs nous permet de définir tout un tas d'utilitaires de vérifications et de test

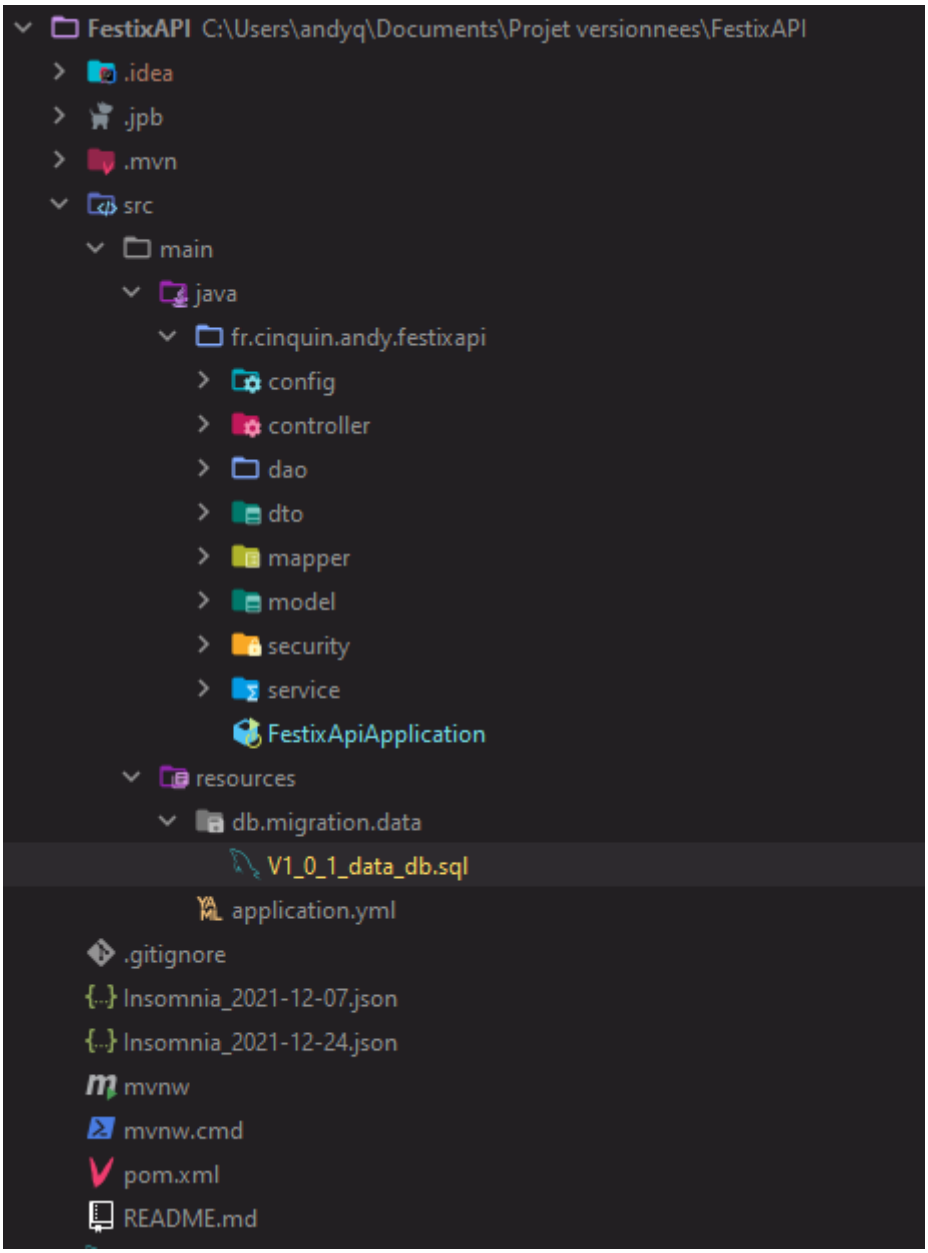
La couche middleware va être celle qui intercepte les requêtes et les vérifies (on vérifie les tokens avec cette couche)

Les model permettent de définir nos éléments et faire les liens entre eux (entre un artiste & un évènement par exemple)

Les routes sont les points d'entrées des requêtes, nous permettent de définir les différents points d'appui des différentes requêtes

La partie service s'occupe de toute la logique de chaque requête, le traitement et l'envoi des bonnes données, de tri, etc., etc., etc.

Pour la version Java spring boot :



Ici on s'intéresse aux différentes couches de l'application, dans un model plus classique à base de controller, de dao, dto, mapper, model, et service,

On utilise les Controller en temps que point d'entrée d'API (c'est ici qu'on accueillera les requêtes du style /login, /logout , /auth/register etc, etc, etc.

En paramètre on peut avoir des schémas de données, décris dans les dto (data transfert object), ces éléments seront des éléments de transitions, les liens entre différentes tables seront justes des listes d'id ou des simples id directement, et non des liens vers d'autres objets.

Les dao seront les éléments qui permettes de décrire comment on récupère nos données directement en base de données, on utilise hibernate pour ça, ce qui nous permet d'avoir tout un tas de fonctions de recherches préinstallées. (find by id, etc etc)

Les models sont les éléments qui permettes de décrire les données sous formes objets, et comment les éléments interagissent entre eux au niveau logique objets, on s'en sert également ici pour décrire le fonctionnement de notre base de données.

Les mappers sont des éléments qui permettes de faire le transfert entre model & dto, et de convertir l'un à l'autre, pour pouvoir créer automatiquement les liens objets via les dtos.

Les services eux, permettes de faire la logique, et permettes de d'utiliser et récupérés les autres couches, c'est ici que ce passe la plupart des éléments logiques et traitements de l'application, des retours utilisateurs, traitements des données, etc etc

La partie security et config sont un peu à part, et servent à définir des éléments de l'application, son fonctionnement et comment ces éléments réagiront, par exemple, ont défini le traitement de nos authentifications et tokens dans ces parties, ainsi que les règles CORS. Par exemple.

La sécurité de l'application

L'application 'Visiteur' n'a pas de sécurité particulière, l'application étant capable uniquement de récupérer des infos et pas d'en envoyer.

L'application admin & l'api, possède un système d'authentification basé sur des authentifications basiques à bases de mot de passe et de login basique également,

après avoir établi et vérifié les identifiants et mots de passe, l'api renvoie un token d'authentification (JWT) et un autre token (X-CSRF) qui permettes tous deux de vérifié la source des demandes, et de vérifié si le token n'a pas changé de source entre temps,

Le JWT sert à vérifier l'utilisateur, et s'il est connecté à un compte

Le CSRF sert, lui à vérifier que l'utilisateur ne s'est pas fait voler son token et vérifie l'identité également, en faisant une triple vérification, à la fois dans les headers de requêtes, mais également dans les cookies.

Les tokens sont vérifiés et régénérés à chaque requête sensible de l'utilisateur

Également, les mots de passe sont bien évidements chiffrés et pas en clair en base de données.

Les futures évolutions souhaitées

On pourrait imaginer un autre fonctionnement de l'api, et la possibilité pour un artiste de faire des demandes de changements de sa page, avec vérification côté administrateur des différentes données.

Le dépôt de code

Lien git API v1

https://github.com/CinquinAndy/b3_Festix_api

Lien git API v2

https://github.com/CinquinAndy/FestixAPI

Lien git Application Visiteur

https://github.com/CinquinAndy/FestixApp

Lien git Application Administrateur

https://github.com/CinquinAndy/FestixAppAdmin

Lien maquette

https://xd.adobe.com/view/fd877f12-a1cb-4dcd-852c-1f719709e7ab-a8d4/

Les éléments facultatifs que vous avez souhaité ajouter

Faire deux applications à la place d'une, question de logique et de sécurité à mon sens.

Un système d'authentification poussé, protégé contre beaucoup d'attaque, xss & injections comprises.

Une ui avec un design personnalisé et un template fait à la main et from scratch.

Une maquette adobe xd relativement poussée également.

L'activation / désactivation des comptes -> (permet de ban des autres utilisateurs)