CÉGEP DU VIEUX MONTRÉAL

Document de conception

Audience : application web pour gestion d'horaire

Réalisé par
Dany Viens

Présenté à
Jean-Christophe Demers

Dans le cadre du cours 420-C61-VM
« Projet Synthèse »

Le 4 mars 2021

Maquette

Les maquettes ont été élaborées avec l'application balsamiq grâce à la période d'essaie gratuite de 30 jours.

Figure I : Écran de connexion

L'écran de connexion affichera des messages d'erreurs si des informations non conformes sont saisies par l'usager

Figure II : Écran d'enregistrement

L'écran affichera les messages d'erreur appropriés. La validation des entrées sera effectuée dans le back-end.

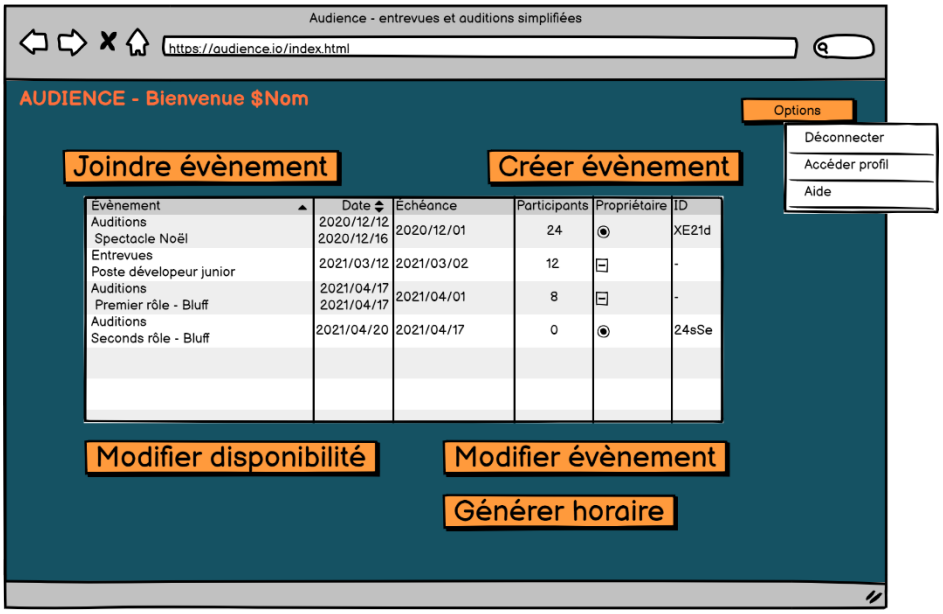

Figure III : Page d'accueil

Affiche les évènements créés par l'usager ou ceux où l'usager à déjà donné ces disponibilités.

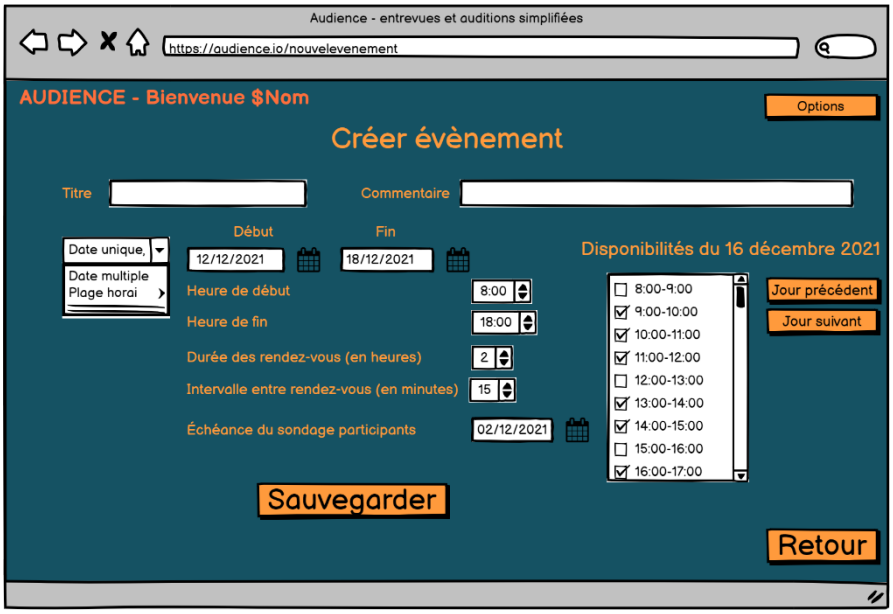

Figure IV : Page création d'évènement

Création d'un évènement affichage des différents paramètres possible.

Alpha : tous les paramètres ne pourront pas être implémentés dans le délai fourni.

Figure V : Page joindre évènement

Pour mieux utiliser l'espace vertical, il sera possible de consulter l'entièreté des journées dans un fil continu. L'usager doit cocher les plages horaires où il est disponible.

BONUS : ajouter fonction pour indiquer qu'un horaire est préférentiel. À être tenu en compte dans l'algorithme du calcul d'un horaire optimal.

Figure VI : Écran de connexion

Affichera un tableau affichant, heure par heure, l'horaire généré selon les disponibilités fournies par les participants.

BONUS : Une version avancée pourrait être connectée à un API de GMAIL permettant d'envoyer une confirmation à chaque participant. Néanmoins, ceci dépasse largement le temps dont je dispose cette session.

Patrons de conceptions

Stratégie (Strategy)

L'encapsulation est l'un des principes de programmation les plus fondamentaux. D'isoler les décisions de conception les unes des autres permet d'obtenir un code qui est plus modulable et plus facile d'entretien. Pour résoudre le problème d'optimisation d'horaire de mon projet, j'ai choisi d'implémenter le patron de conception « strategy » dans la conception d'un l'algorithme génétique d'intelligence artificielle.

L'algorithme génétique consiste, en simplifiant le concept, d'une collection de quatre catégories de fonctions : algorithme générique, sélection, croisement et mutation. L'objectif serait de rendre, pour chacune des quatre fonctions, une série d'algorithmes interchangeable pour résoudre le problème. Par exemple, il serait possible d'utiliser deux méthodes différentes pour la sélection des chromosomes dans l'algorithme en choisissant une stratégie différente lors de la création de l'algorithme.

L'algorithme génétique aurait un lien vers la classe abstraite *SelectionStrategy* avec la fonction abstraite « selectionner() ». Des stratégies complètes pourraient être implémentées dans des classes dérivées définissant ce que pourrait accomplir la fonction « selectionner () ». Ces classes pourraient être *RouletteSelectionStrategy* ou bien encore *ValueProportionalSelectionStrategy*.

Les deux avantages principaux de ce patron de conception sont qu'il sera possible de tester rapidement plusieurs options pour déterminer les stratégies optimales pour résoudre le problème. L'autre avantage est que le code qui sera produit pourra être réutilisé aisément pour résoudre d'autres types de problèmes étant donné qu'il sera très flexible sur les approches à utiliser.

Un exemple UML d'application propre à notre problème est présenté dans la **Figure XIII** (p.6).

Object d'accès aux données - DAO - *Data access object*

Il s'agit d'un patron de conception d'architecture logicielle. Il s'agit de regrouper toutes les fonctions donnant accès à la base de données dans une classe à part s'assurer de respecter le modèle d'encapsulation des données. Ainsi, il sera beaucoup plus aisé de maintenir un site web, une application où un logiciel si l'ont doit modifier le mode de stockage des données persistantes.

Les autres classes feront toujours référence aux fonctions des classes DAO. On pourrait changer le contenu des fonctions des classes DAO sans affecter le fonctionnement du programme.

En ce qui concerne notre travail, nous avons choisi d'utiliser Node.JS et le *framework* Express pour programmer notre serveur back-end. Bien que nous n'utiliserons pas des classes dans cette structure, nous ferons référence à des objets et à des fichiers pour exporter les fonctions en lien avec notre base de données.

Par exemple, nous encapsulerons les méthodes visant à chercher les usagers dans un fichier User.js. Bien que ces méthodes seront utilisées dans d'autres fichiers, la logique interne de ces fonctions sera rassemblée au même endroit, améliorant ainsi la facilité de maintenance et la lisibilité d'intention dans notre code.

Un exemple d'application propre à notre problème est présenté dans la **Figure XII** (p.7).

UML

L'ensemble des documents est disponible en annexe sous format .PDF et dans le format .MDJ originel. Star UML a été utilisé pour concevoir les diagrammes de cas d'usage, de classes et de séquences.

Diagramme de cas d'usage

Un diagramme différent est présenté pour chaque type de page disponible dans l'application web.

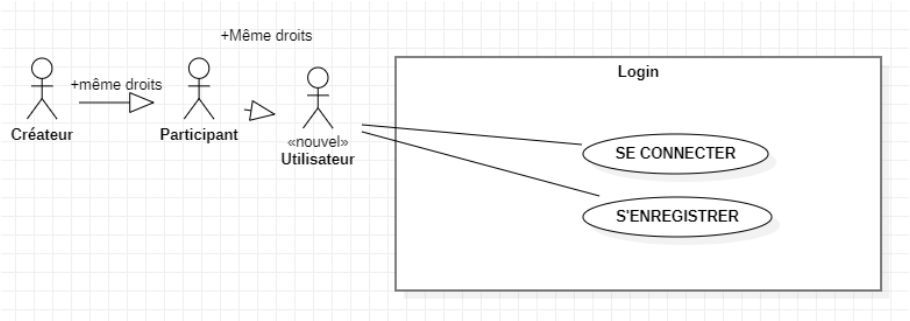

Figure VII : Diagramme des cas d'usage - login

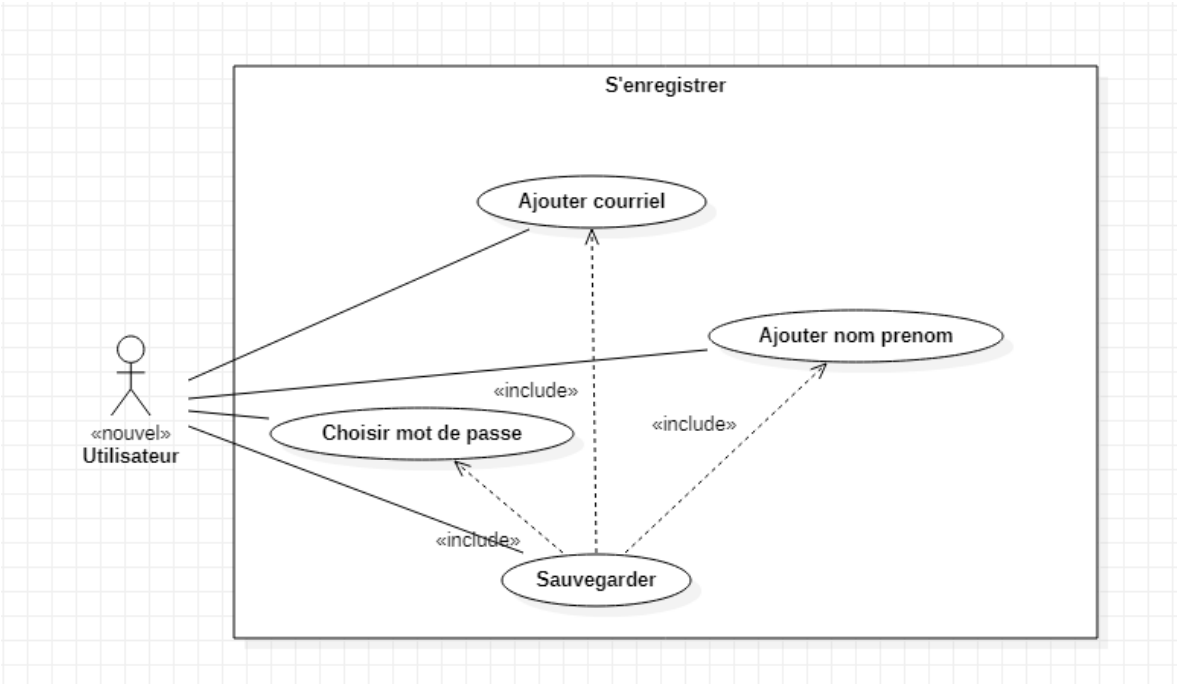

Figure VIII : Diagramme des cas d'usage : enregistrement

La figure XIV présente le diagramme de séquence pour le cas d'usage « s'enregistrer ».

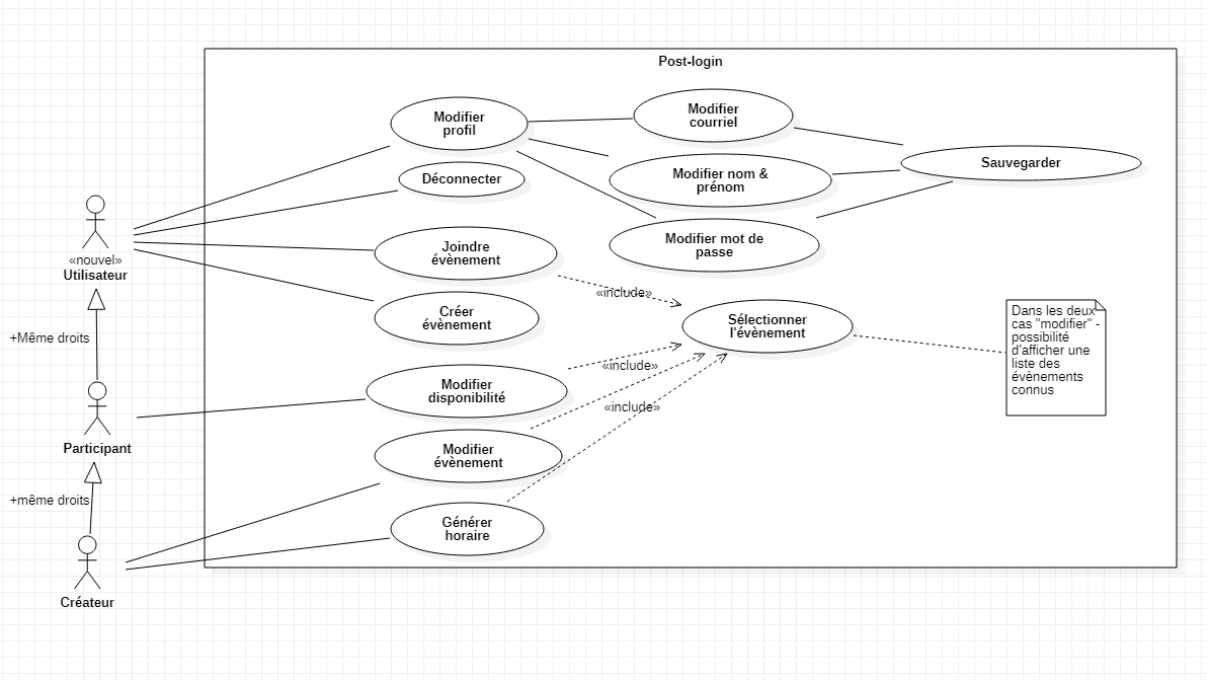

Figure IX : Diagramme des cas d'usage : écran principal

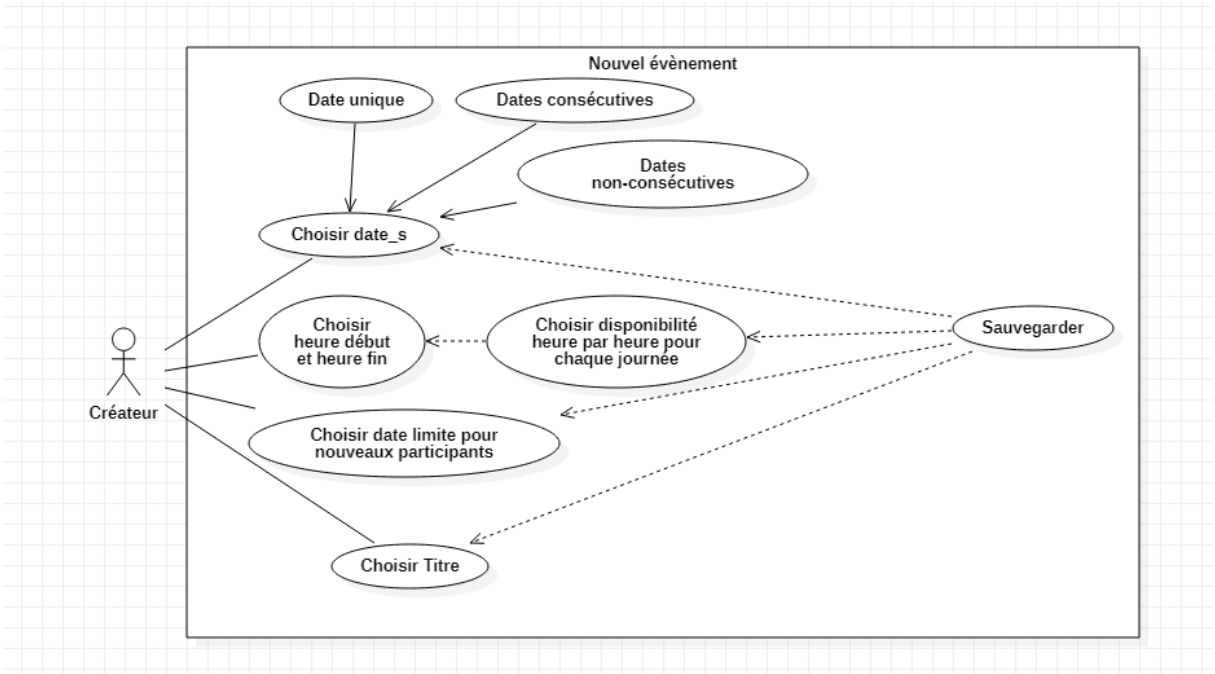

Figure X : Diagramme des cas d'usage : ajout d'évènement

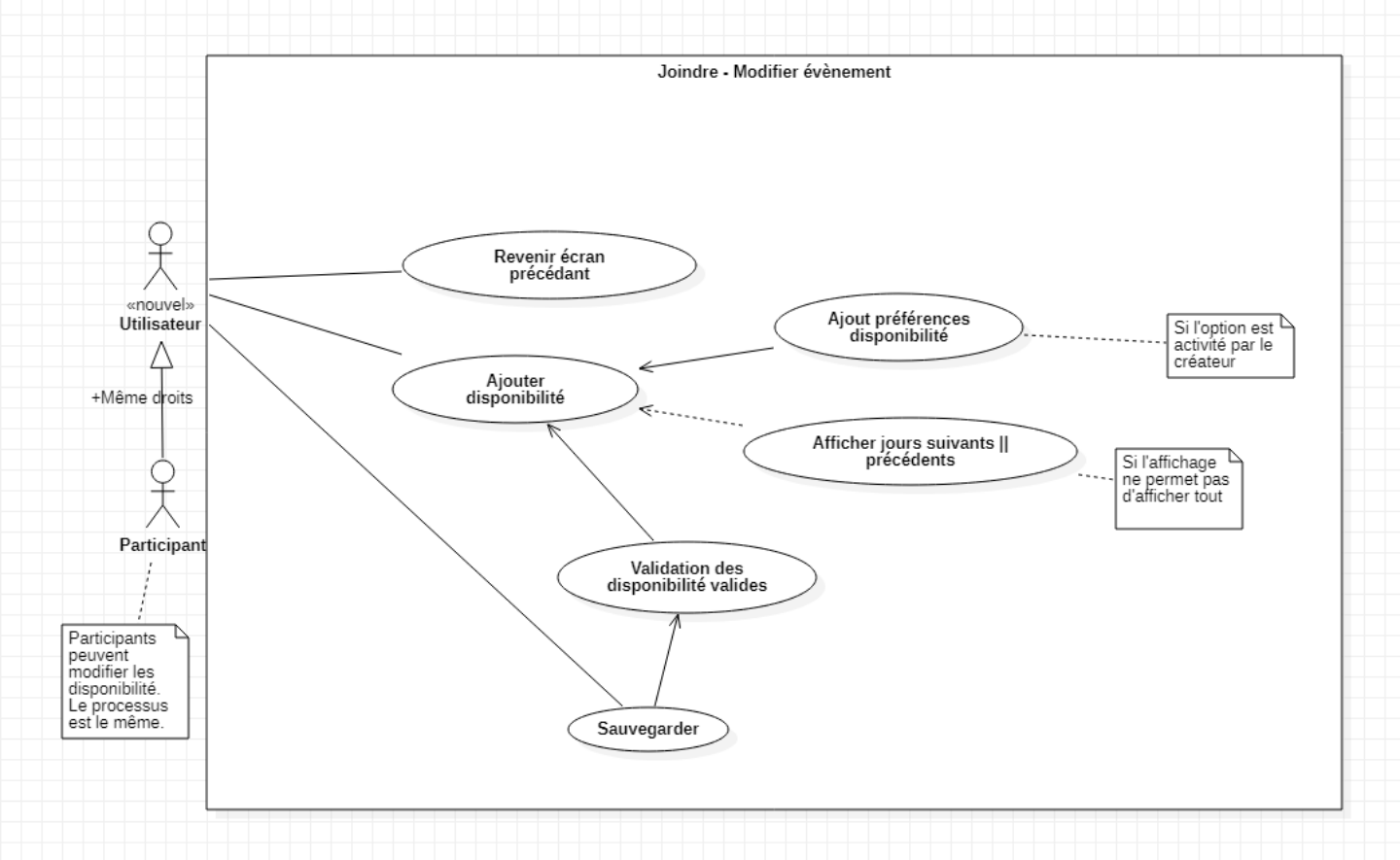

Figure XI : Diagramme des cas d'usage : joindre évènement

La figure XV présente le diagramme de séquence pour le cas d'usage « joindre évènement ».

Diagramme de classes

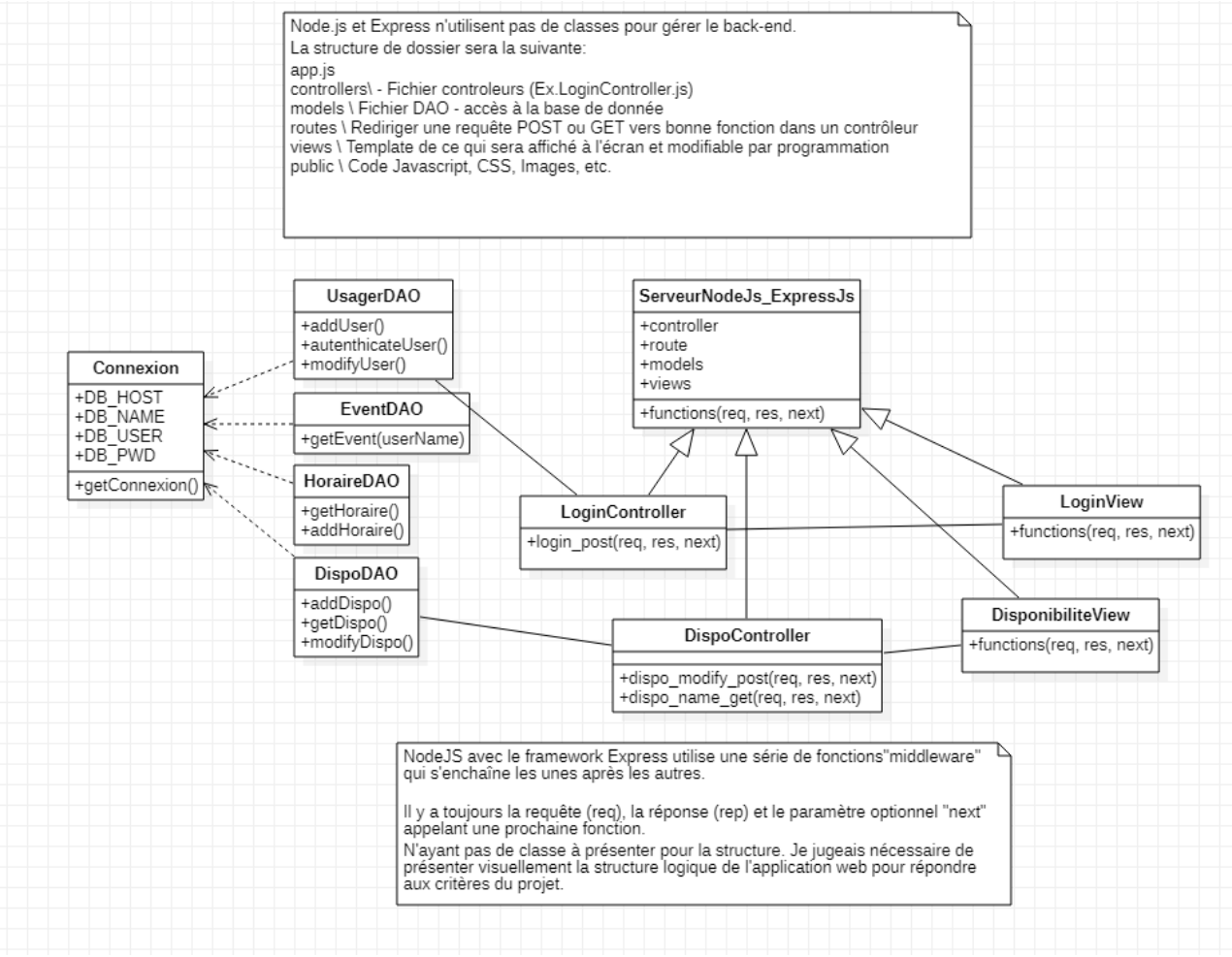

Figure XII : Diagramme de classe : webapp

Tel qu'indiqué dans les notes encadrées dans le diagramme, la structure de Node.js comme back-end n'utilise pas de classse comme d'autres language back-end tel le PHP. La structure de Node.JS utilise un thread unique où chaque demande (request) de l'utilisateur est traitée comme un évènement (event). Elles sont placées dans un Event Queue qui sera traité selon le principe FIFO (first in, first out).

Concrètement dans la structure de NodeJs, le programmeur ne fait que coder des fonctions, faisant référence à d'autres fonctions (callback). NodeJS est considéré comme un langage de script.

Il est donc difficile de faire un diagramme de classes détaille dans un environnement sans classes. Néanmoins, je tenais à faire une démonstration de la structure de dossier du projet. De plus, j'ai voulu mettre en évidence l'utilisation du patron de conception « objets d'accès aux données » (DAO).

N'étant pas le but du cours de projet final, Node.JS et Express sont le sujet de mon sujet de recherche pour le cours ***Veille technologique*** . Dans le but de mieux comprendre les composantes de la technologique, empruntée sans l'utiliser aveuglément.

Nous avons survolé brièvement Node.js dans le cours ***Web III*** est l'un des défis fondamentaux du projet est de perfectionner l'apprentissage de cette technologie au courant de la session.

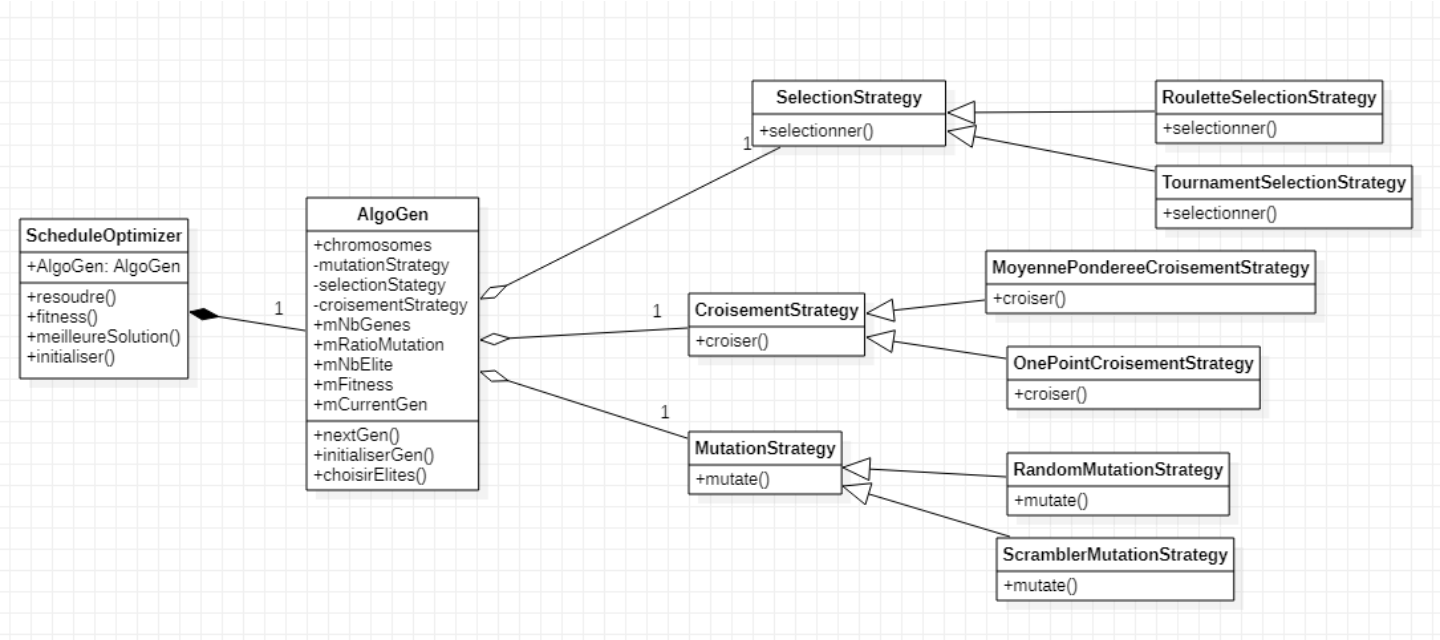

Figure XIII : Diagramme de classe : algorithme génétique

Le projet utilisera une approche orientée objet pour implémenter un algorithme génétique pour solution le problème d'optimisation d'horaire.

La **Figure XIII** fait la présentation du patron de conception « Strategy » présenté à la page cinq du présent document.

Diagramme de séquences

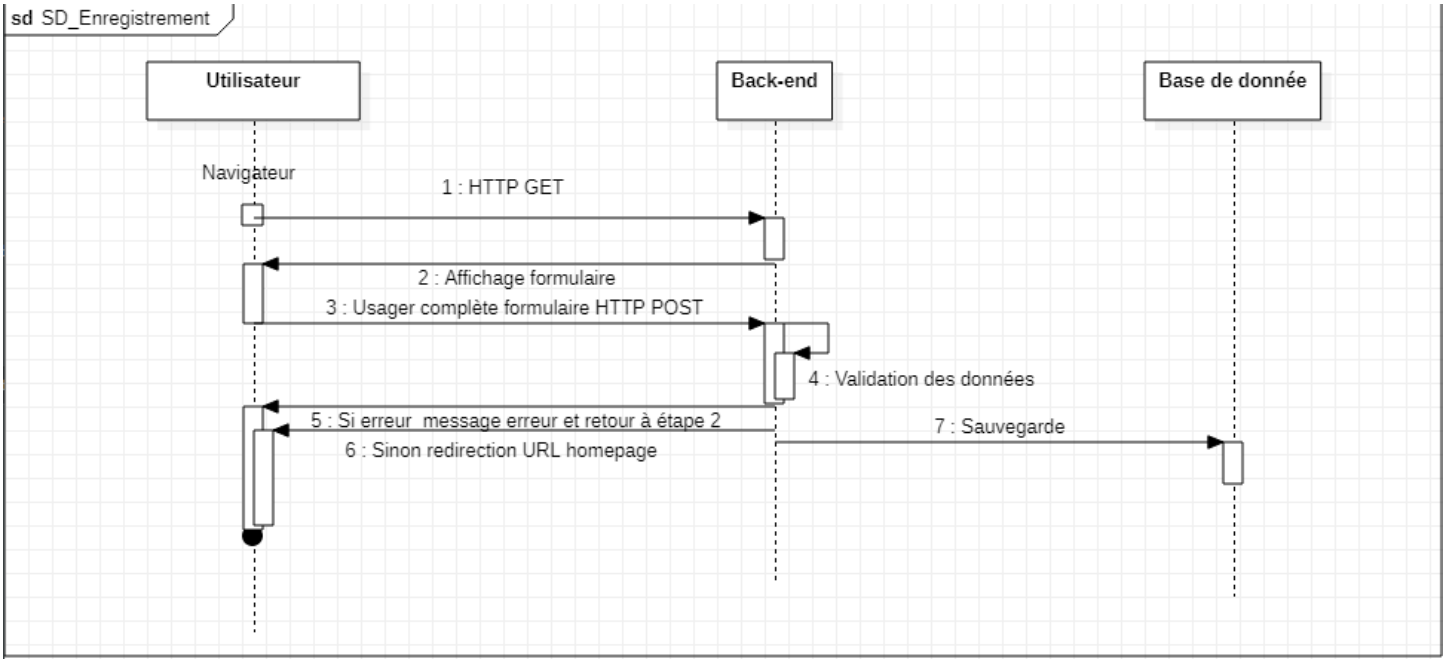

Figure XIV : Diagramme de séquences : enregistrement usager

Si la validation (4) décèle des erreurs. L'affichage du formulaire sera généré de nouveau (2) en affichant une description des erreurs.

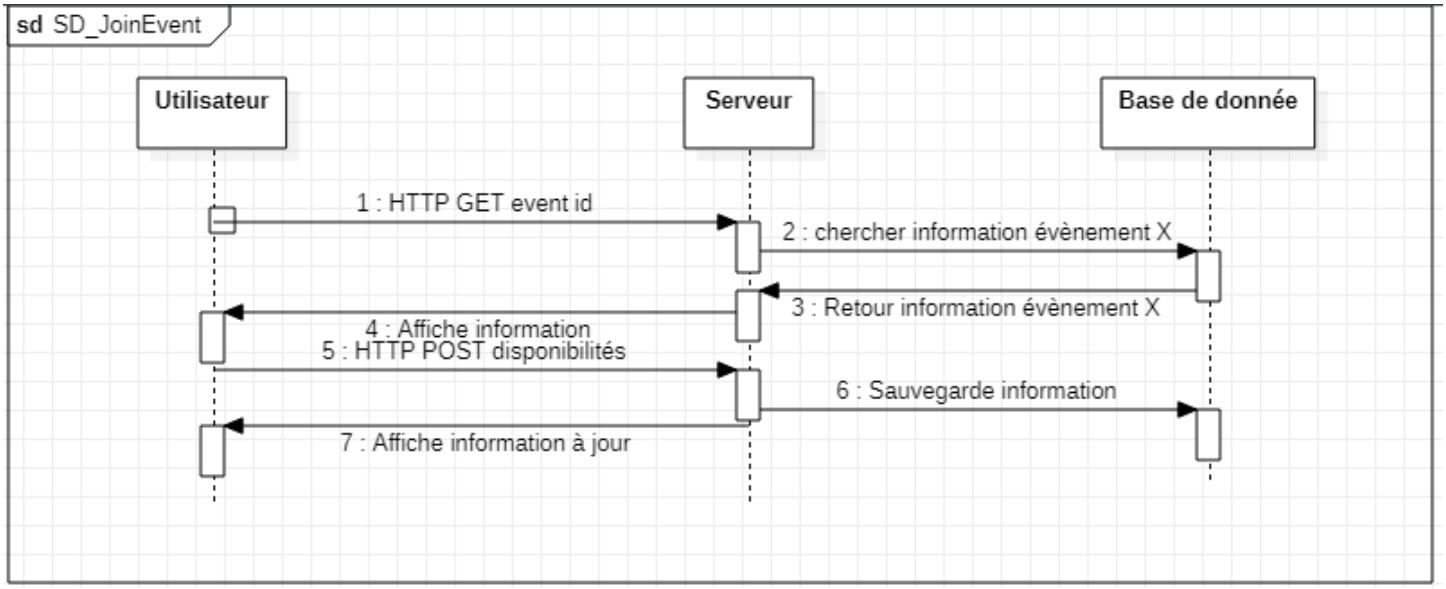

Figure XV : Diagramme de classe : joindre évènement

Comme il s'agira de choix d'horaire à cocher, il n'y a pas de validation du formulaire dans cette séquence

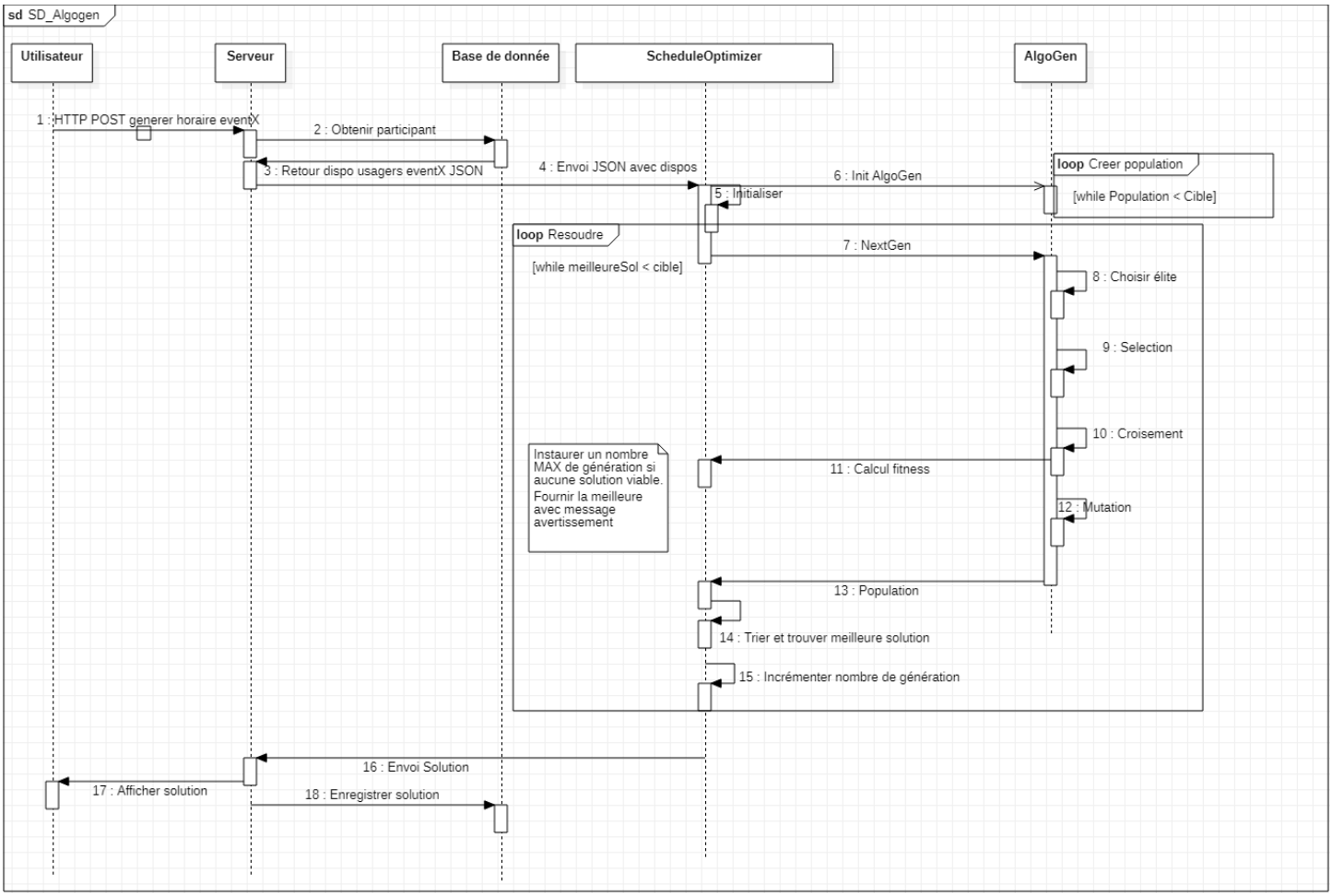

Figure XVI : Diagramme de classe : générer horaire

Schéma de données externes

Une copie du schéma, ainsi qu'un lien vers l'outil de dessin utilisé est disponible en annexe.

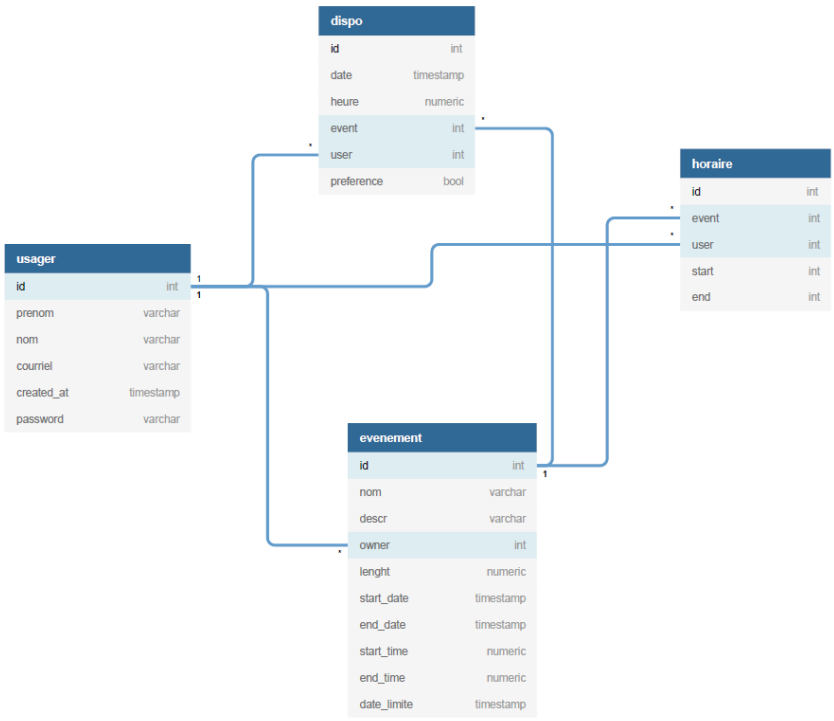

Figure XVII : Schéma de la base de donnée

Le choix définitif de la technologie n'est pas encore choisi au moment de remettre ce document. Le langage choisi sera soit PosgreSQL ou bien MongoDB. Ayant moins d'expérience dans l'utilisation de MongoDB, nous avons une préférence vers cette technologie dans le but d'améliorer notre connaissance d'une technologie.

Dans les deux solutions envisagées, il est possible d'être directement les données sous format JSON. Il s'agit du standard qui sera utilisé pour communiquer entre le *front-end*, le *back-end* et le script d'algorithme génétique.

Description des données conservées :

Usager :

- Id
- Nom
- Prénom
- Courriel
- mot de passe
- date de création.

Évènement :

- id, nom
- description
- propriétaire (id usager)
- durée
- date de début et date de fin
- heure de début et heure de fin (pour chaque journée)
- date limite (pour donner les disponibilités.

Disponibilité :

- id
- date
- heure
- évènement (id évènement)
- usager (id usager)
- préférence

Pour chaque choix de case horaire fait par un usager, une entrée sera ajoutée dans la base de données.

Voici à quoi pourrait ressembler la table « dispo » rassemblant les disponibilités d'un usager :

ID	date	heure	Event [FK]	User [FK]	preference
1	1er mars 2020	1400	12	78	True
2	1er mars 2020	1500	12	78	False
3	1er mars 2020	1500	12	24	False
4	2 mars 2020	1000	12	78	False
5	2 mars 2020	1600	14	79	True
6	2 mars 2020	1600	14	231	False

Un modèle alternatif un format JSON pour stocker les disponibilités dans une même entrée :

```
[{
        "id":"1",
        "evenement":"24",
        "usager":"76",
        "disponibilité": {
            "14 mars":[
             {"0800":"0","preference":"false"},
             {"0900":"1","preference":"true"},
             {"1000":"1","preference":"true"}  ],
            "15 mars": [
                // données ici…. ]
  }  }
```

Une rangée serait utilisée par rendez-vous confirmé par usager, pour chaque évènement.

Horaire :

- id
- évènement (id évènement)
- usager (id évènement)
- Début et fin de son rendez-vous.

Encore une fois, un modèle JSON pourrait être utilisé pour structurer les horaires générés dans un même objet. MongoDB et PostgreSQL étant en mesure de travailler avec cette structure de donnée très flexible