



Applications Web

Hagimettaceinture

Groupe L34

Élèves :

THEVENET Louis

LEBOBE Timothé

LECUYER Simon

SABLAYROLLES Guillaume

28 Juin 2025

Table des matières

1. Projet Application Web	3
1.1. Architecture	3

1. Projet Application Web

1.1. Architecture

1.1.1. Les entités

Race id, *circuit*, date, *vehiculeType*, participants

Member idMembre, name, firstName, vehicules, subscriber

Vehicule id, *vehiculeType*, branch, model, licensePlate, date, *owner*

Circuit id, place, distance, turnNumber, spectatorNumber, name, bestTime

Meeting id, title, guests, date

Sponsoring id, *racingTeam*, *sponsor*, date, duration, investment

RacingTeam idRacingTeam, nom, classement, membres, sponsors

Sponsor id, name, investedCapital, date

Nous avons également utilisé une classe entité abstraite **Event** avec la stratégie d'héritage `InheritanceType.TABLE_PER_CLASS`.

Voici les classes qui en héritent :

Race pour sa date d'occurrence,

Vehicule pour sa date de mise en service,

Circuit pour sa date de création,

Meeting pour sa date d'occurrence,

Sponsoring pour ses dates de début et de fin,

Sponsor pour sa date de création.

Ceci nous permet de traiter toutes ces entités comme des événements pour les afficher sur notre page calendrier.

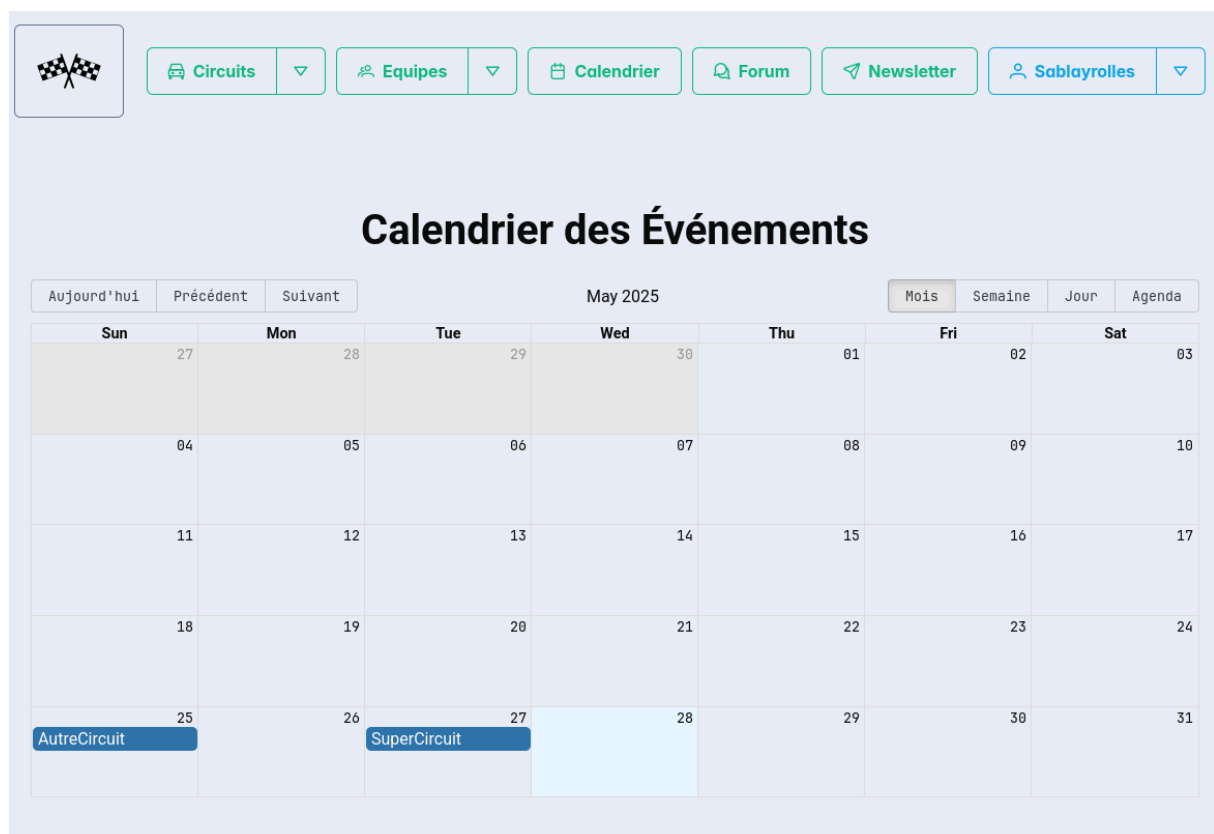


Fig. 1. – Exemple de deux créations de circuits apparaissant sur le calendrier

1.1.2. Relations

- Race
 - ManyToOne: Circuit
 - OneToOne: VehiculeType
 - ManyToMany: Member
- Member
 - OneToMany: Vehicule
 - ManyToMany: RacingTeam
- Vehicule
 - ManyToOne: VehiculeType
 - ManyToOne: Member
- Meeting
 - OneToMany: Member
- Sponsoring
 - OneToOne: RacingTeam
 - OneToOne: Sponsor
- RacingTeam
 - ManyToMany: Member
 - ManyToMany: Sponsor

Les entités du système sont fortement liées entre-elles, par exemple, une *Course* (**Race**) est organisée sur un *Circuit* donné et concerne un certain *Type de Véhicule*, elle rassemble plusieurs *Membres* participants, ce qui justifie l'utilisation d'une relation **ManyToMany** entre **Race** et **Member**. Chaque *Membre* peut posséder plusieurs *Véhicules* (**OneToMany**), et chaque véhicule appartient à un seul membre (**ManyToOne**). Les *Écuries* (**RacingTeam**) regroupent plusieurs membres et sont soutenues par plusieurs sponsors, d'où des relations **ManyToMany** avec **Member** et **Sponsor**. Les *Sponsorisations* (**Sponsoring**) relient précisément une écurie et un sponsor sur une période donnée (**OneToOne**). Enfin, les *Réunions* (**Meeting**) peuvent inviter plusieurs membres (**OneToMany**). Ces relations permettent de modéliser précisément les interactions et les dépendances entre les différents acteurs et objets du domaine.

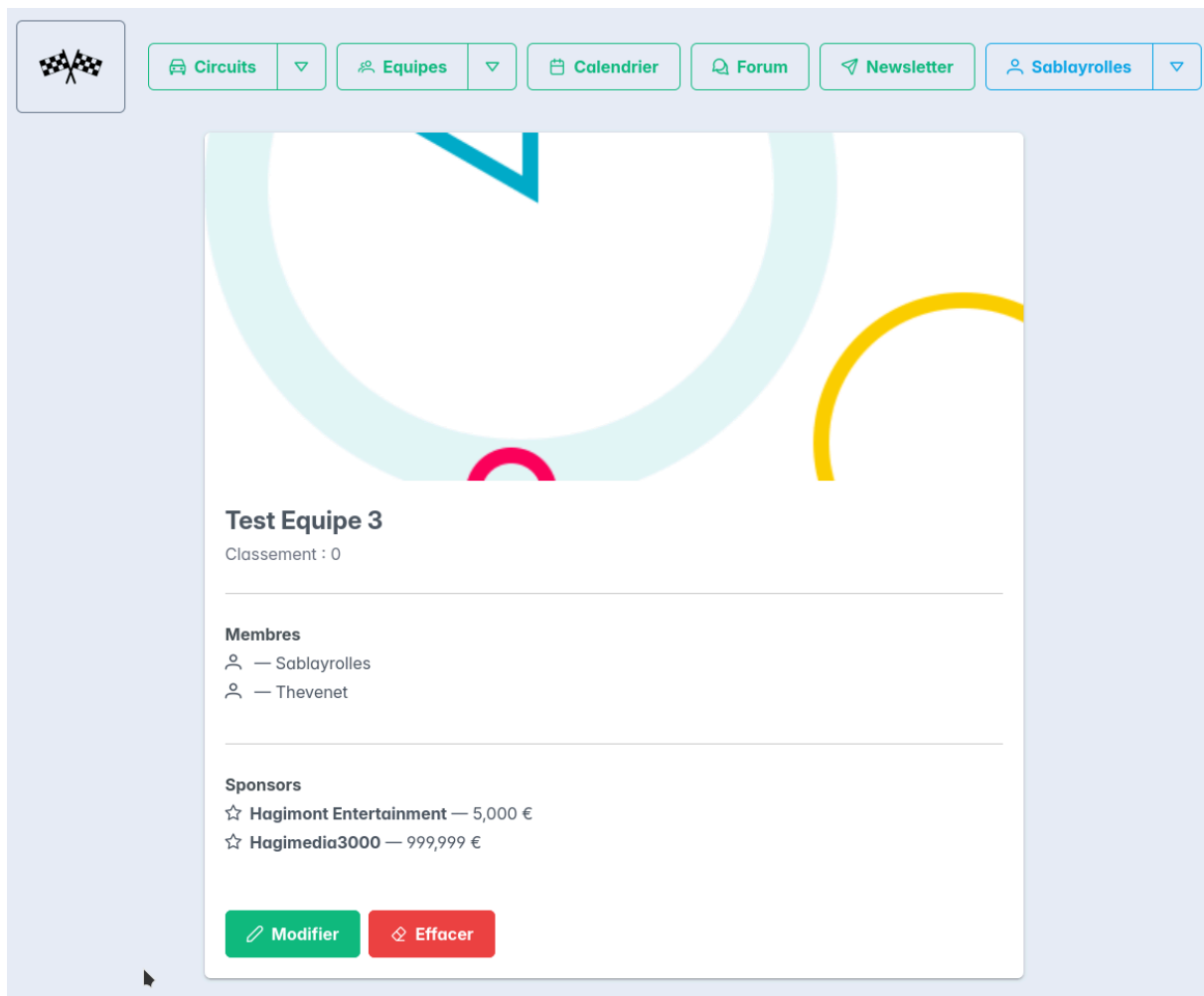


Fig. 2. – Exemple d’une équipe qui possède deux membres et deux sponsors

1.1.3. Les routes

- index
 - GET / : route index
- circuits
 - GET `/api/circuits` : afficher tous les circuits
 - GET `/api/circuits/{circuitId}` : afficher les informations du circuit `circuitId`
 - PUT `/api/circuits/{circuitId}/edit` : modifier le circuit `circuitId`
 - POST `/api/circuits/new` : créer un nouveau circuit
 - DELETE `/api/circuits/{circuitId}` : supprimer le circuit `circuitId`
- calendar
 - GET `/api/calendar` : afficher tous les événements
 - GET `/api/calendar/{date}` : afficher les événements à une date donnée
- forum
 - GET `/api/forum` : afficher tous les sujets du forum
 - GET `/api/forum/{idForumTopic}` : afficher tous les messages du sujet `idForumTopic`
 - GET `/api/forum/{idForumTopic}/consult` : afficher le titre du sujet `idForumTopic`
 - POST `/api/forum/{idForumTopic}/post` : publier un message dans le sujet `idForumTopic`
 - POST `/api/forum/post` : créer un nouveau sujet de forum

- teams
 - GET */api/teams* : afficher toutes les équipes
 - GET */api/teams/{teamId}* : afficher les informations de l'équipe *teamId*
 - PUT */api/teams/{teamId}/edit*: modifier l'équipe *teamId*
 - POST */api/teams/new* : créer une nouvelle équipe
 - DELETE */api/teams/{teamId}* : supprimer l'équipe *teamId*
- members
 - GET */api/members* : afficher tous les membres
 - GET */api/members/{memberId}* : afficher les informations du membre *memberId*
 - POST */api/members/new* : créer un nouveau membre
- inscription/connexion
 - GET */api/register/homonyms/{name}/{firstName}* : vérifier les homonymes libres
 - POST */api/register* : enregistrer un nouveau membre
 - POST */api/login* : se connecter via email et mot de passe
 - GET */api/connected* : vérifier l'état de connexion via le token JWT
- vehicules
 - GET */api/vehicules* : récupérer tous les véhicules
- sponsors
 - GET */api/sponsors* : récupérer tous les sponsors
 - POST */api/sponsors/new* : créer un nouveau sponsor