Dossier Projet pour le titre **Développement Web Web Mobile - 2025 / 2026**

# My Formula One Tracker

Par **Elise LIAUTAUD**

# Sommaire

**Introduction** ..... **2**
**Présentation du projet** ..... **3**
**Cahier des charges** ..... **4**
Parties prenantes ..... 4
Périmètre fonctionnel ..... 5
Besoins fonctionnels et techniques ..... 6
Contraintes ..... 7
Matrice de Risque ..... 8
Fiche MVP (Minimum Viable Product) ..... 10
**Gestion du projet** ..... **11**
**Analyse et conception** ..... **13**
Choix techniques ..... 13
Maquettage ..... 14
**Développement** ..... **15**
Fonctionnalités ..... 15
Sécurité ..... 17
Accessibilité ..... 18
Explication de code : Calendrier annuel (Annexe n°3) ..... 19
Explication de code : Sauvegarde des résultats de courses dans la base de données (Annexe n°4) ..... 20
**Déploiement** ..... **21**
**Tests et validations** ..... **22**
**Difficultés rencontrées** ..... **23**
**Bilan et perspectives** ..... **24**
**Conclusion** ..... **25**
**Annexes** ..... **26**
Annexe 1 : MLD (Modèle Logique de Données) ..... 26
Annexe 2 : Échantillons Figma ..... 27
Annexe 3 : Code Calendrier annuel ..... 31
Annexe n°4 : Code Sauvegarde résultats de course ..... 35

# Introduction

Depuis un certain temps, j'ai la volonté de créer un site personnalisable permettant de suivre les événements et l'avancement de la saison de Formule 1 tout au long de l'année. Ce projet est pensé comme une plateforme à la fois informative et ludique, destinée à enrichir l'expérience des fans en y ajoutant des fonctionnalités interactives et personnelles.

Sa réalisation s'inscrit dans le cadre de ma formation *Développeur Web et Web Mobile,* dans laquelle ce projet final individuel constitue une étape obligatoire. Il représente ainsi une opportunité concrète de mettre en œuvre mes compétences en développement web front-end et back-end.

# Présentation du projet

Le site poursuit plusieurs objectifs :

- **Gestion de compte utilisateur** : Création de compte, modification des informations personnelles, gestion de tokens (connexion, déconnexion)
- **Suivi de l'évolution de la saison de Formule 1** en ayant accès aux informations clés : résultats des courses, classements pilotes et écuries au championnat, statistiques, grille actuelle
- **Interface claire et immersive** : inspirée de la direction artistique officielle de la F1, avec un design responsive et moderne, et des créations visuelles originales
- **Expérience utilisateur personnalisée** : possibilité de créer ses propres classements (pilotes, écuries, circuits) pour refléter ses préférences (Hors MVP)
- **Souvenirs de Grand Prix** : espace personnel de commentaires sur chaque course.

Le site s'adresse principalement à des personnes intéressées par la Formule 1, qu'il s'agisse de passionné·es régulier·es ou de spectateur·trices plus occasionnel·les, souhaitant suivre l'évolution de la saison actuelle. Le public visé partage les caractéristiques suivantes :

- **Aisance avec les interfaces web** (navigation, création de compte, interaction avec des contenus dynamiques)
- **Intérêt pour les données sportives** : résultats, statistiques, classements
- **Volonté de personnalisation** : expression de ses propres opinions ou préférences via des classements et des commentaires

Bien que le projet ne vise pas une audience massive, il est conçu pour offrir une expérience engageante et personnalisée à toute personne familière avec le sport automobile et à l'aise avec l'environnement numérique.

Persona type :

- Identité : Lucie, 27 ans
- Profil : Fan de Formule 1 depuis 2018, suit la majorité des courses.
- Comportement : Consulte les classements, lit des actus F1, aime donner son avis sur les pilotes, adore les statistiques.
- Besoins :
    - Accès facile au calendrier annuel
    - Créer un classement "de cœur"
    - Consulter les fiches pilotes et écuries rapidement
    - Avoir accès à un espace pour ses sauvegarder ses avis et émotions de chaque Grand Prix

# Cahier des charges

## Parties prenantes

| Partie prenante | Rôle dans le projet |
|---|---|
| **Commanditaire, Jury de formation** | Définition des attentes pédagogiques<br><br>Evaluation du projet final |
| Elise Liautaud (Moi) :<br><br>**Cheffe de projet**<br><br>**Développeuse Front-end & Back-end**<br><br>**Designer UI/UX** | Respect du planning et des spécifications<br><br>Développement technique (front et back), mise en oeuvre des fonctionnalités<br><br>Gestion de la qualité technique, sécurité, performances |

| | |
|---|---|
| **Utilisateur·trice** | Prise en compte de leurs besoins lors de l'analyse fonctionnelle<br><br>Cible principale lors des tests et ajustements UX |

## Périmètre fonctionnel

Le périmètre a été défini en tenant compte des ressources disponibles (temps, compétences, budget, personnel…) afin de garantir un projet réaliste et fonctionnel dans les délais impartis.

Fonctions **incluses** dans le périmètre :

- Création et gestion de compte utilisateur·trice (Inscription, connexion, déconnexion)
- Affichage du calendrier de la saison
- Affichage des détails de Grand Prix
- Affichage des résultats de course (par grandprix, par pilote, par écurie à l'année)
- Consultation de fiches détaillées (pilotes // écuries // Grands Prix)
- Espace personnel utilisateur : Commentaires sur les courses, classements personnels et personnalisables (pilotes, écuries, circuits)
- Formulaire de contact (Hors MVP)
- Pages annexes (RGPD, Confidentialité)

Fonctions **exclues** du périmètre :

- Pari avec de l'argent réel ou virtuel à visée monétaire
- Fonctionnalités sociales complexes (chat en direct, messagerie privée, groupes)
- Application mobile

## Besoins fonctionnels et techniques

| Partie prenante | Besoin exprimé | Fonctionnel | Non fonctionnel |
|---|---|---|---|
| **Compte et Accès** | | | |
| Utilisateur·trice | **Gestion de compte (Inscription, Connexion, Déconnexion)** | ✔️ | |
| Utilisateur·trice /Client·e | **Sécurité des données (authentification, protection, hachage de mot de passe avant stockage)** | | ✔️ |
| **Interface** | | | |
| Utilisateur·trice | **UX clair, navigation intuitive** | ✔️ | |
| Utilisateur·trice | **Interface Responsive ordinateur / mobile / tablette** | | ✔️ |
| **Performance** | | | |
| Utilisateur·trice | **Informations à jour (résultats, planning...)** | ✔️ | |
| Utilisateur·trice | **Chargement rapide, bonnes performances** | | ✔️ |
| **Contenu** | | | |
| Utilisateur·trice | **Accès à l'emploi du temps annuel** | ✔️ | |

| | | | |
|---|---|---|---|
| Utilisateur·trice | **Accès aux résultats** | ✔️ | |
| Utilisateur·trice | **Création et affichage de commentaires PERSONNELS** | ✔️ | |
| Client·e/Utilisateur·trice | **Formulaire de contact fonctionnel** | ✔️ | |
| Client·e | **Respect des délais de production** | | ✔️ |

## Contraintes

| Type de contrainte | Détail | Impact sur le projet |
|---|---|---|
| **Front-end** | **React / TypeScript** | Développement rapide, composants UI réutilisables |
| **Back-end** | **Node.js / Express** | Traitement des formulaires, Gestion des utilisateurs et commentaires et préparations de requêtes SQL |
| **Base de données** | **SQL** (**PostgreSQL, Neon**) | Stockage structuré des comptes, données sportives, résultats de courses… |
| **RGPD** | Politique de confidentialité | Affichage du bandeau cookies, consentement explicite requis pour le suivi |

| | | |
|---|---|---|
| **Accessibilité** | **Conformité RGAA** : contrastes, navigation clavier, textes alternatifs | Meilleure accessibilité, inclusivité, tests réguliers à prévoir |
| **Sécurité** | HTTPS, validation des entrées, protection XSS/SQLi, rôles utilisateurs | Sécurisation côté client et serveur, gestion des accès et données sensibles |
| **Navigateurs supportés** | Chrome, Firefox, Safari, Edge | Tests de compatibilité nécessaires pour éviter les erreurs d'affichage |
| **Performance** | Chargement optimisé des images, lazy loading | Chargement rapide, meilleure UX sur mobile, économie de ressources |

Ces contraintes techniques ont été identifiées afin d'assurer un développement structuré, sécurisé et accessible du projet, en cohérence avec les bonnes pratiques du développement web.

## Matrice de Risque

Légende :

- Probabilité : Faible (1), Moyenne (2), Élevée (3)
- Gravité : Faible (1), Moyenne (2), Élevée (3)
- Criticité = Probabilité x Gravité

| Risque | Probabilité (1-3) | Gravité (1-3) | Criticité | Plan d'action / Prévention |
|---|---|---|---|---|
| Non-conformité RGPD | 1 | 2 | **2** | Vérification des formulaires, mentions légales, gestion des données utilisateurs |
| Retard de livraison | 1 | 3 | **3** | Travail régulier |
| Oubli d'une fonctionnalité | 2 | 2 | **4** | Travail réfléchi, Relecture de la note d'intention |
| Perte de motivation, fatigue mentale | 2 | 2 | **4** | Planification souple, pauses, étapes motivantes |
| Faille de sécurité (XSS, injection SQL...) | 2 | 3 | **6** | password_hash, requêtes préparées |

# **Fiche MVP** (Minimum Viable Product)

| Fonctionnalité Retenues | Justification |
|---|---|
| **Inscription/Connexion/Déconnexion** | Associer un espace personnel à chaque utilisateur, activer la personnalisation |
| **Planning** | Suivre l'évolution de la saison et anticiper les prochains événements |
| **Résultats des Grands Prix** | Vue essentielle sur le déroulement réel de la saison |
| **Données statistique de circuit** | Données statiques, affichage simple |
| **Grille actuelle** | Vue essentielle à un instant T de la grille, avec statistiques en fonction des résultats inscrits dans la BDD |
| **Création et Affichage de commentaire** | Revue personnelle de chaque grand prix |

**Fonctionnalités exclues du MVP :**

- **Classements personnelles :** Non prioritaire, implémentation longue
- **Formulaire de contact** : non prioritaire dans une première version non destinée à être déployée publiquement
- **Barre de recherche** : utile à terme, mais non indispensable pour une version initiale avec navigation simple

**Parcours utilisateur du MVP :**

1. L'utilisateur·trice accède au site et crée un compte via le formulaire d'inscription.
2. Après connexion, il/elle accède à un accueil.
3. Il/elle peut consulter :

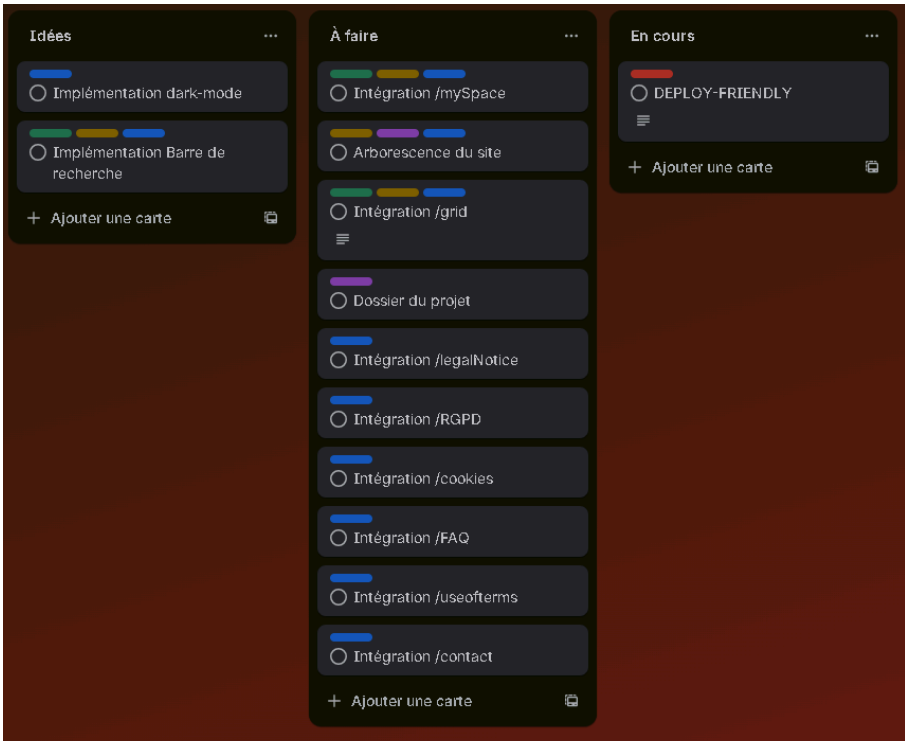
- Le planning complet de la saison de F1
- Les résultats des Grands Prix passés
- Le classement
- La grille de départ actuelle

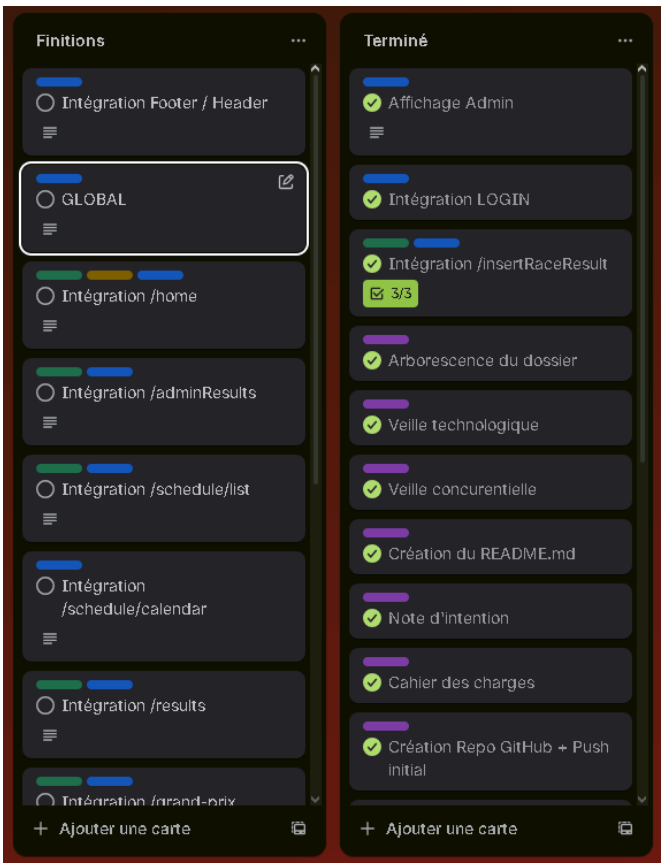
4. L'utilisateur·trice peut créer des commentaires, via une interface dédiée, pour personnaliser son expérience, via la page du Grand Prix concerné.

# Gestion du projet

Pour gérer mon projet, j'utilise **Trello**, un outil qui me permet d'appliquer la méthode **KANBAN**. Pour cela, j'ai mis en place **cinq colonnes** dans lesquelles mes tâches évoluent :

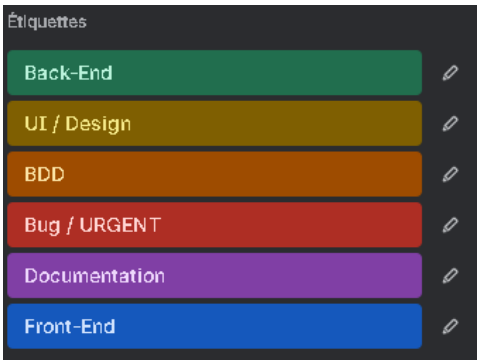
- **Idées** : Fonctionnalités hors-MVP
- **A faire** : Tâche à commencer / en attente
- **En cours** : Tâches en cours de création
- **Finitions** : Tâches non terminées mais dont les finitions sont hors MVP (à traiter en second plan)
- **Terminées** : Tâches finalisées



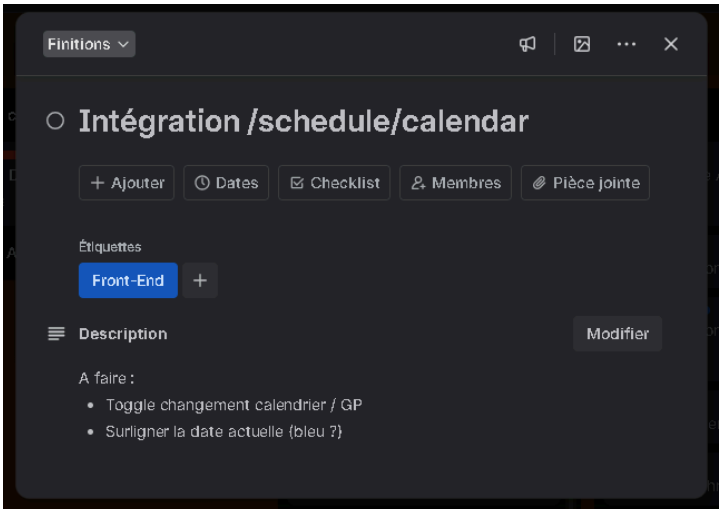



J'ai également créé des **étiquettes colorées** afin d'identifier rapidement le type de tâche associé à chaque carte. Elles sont réparties en six catégories :

- **Vert** : Back-end
- **Jaune** : UI / Design (création des dessins originaux)
- **Orange** : Base de données (peuplement, logique)
- **Rouge** : BUG / URGENT
- **Violet** : Documentation (Dossier annexe du projet)
- **Bleu** : Front-end



Les tâches sont organisées en **cartes**, chacune correspondant à une **page du site web**. Dans la partie *Description* de chaque carte, je détaille précisément les tâches à effectuer. Une fois une tâche réalisée, je peux simplement **supprimer la ligne correspondante**.




# Analyse et conception

## Choix techniques

Le projet repose sur une architecture composée d'un front-end (UI/UX), d'un back-end (interaction avec la base de données) et d'une base de données assurant la structure et le stockage des informations.

Pour le développement du back-end, Node.js est utilisé afin de permettre l'exécution de JavaScript côté serveur, couplé au framework Express pour la gestion des routes et des requêtes HTTP. Cette combinaison permet une séparation claire et structurée entre le front-end et le back-end, facilitant la maintenabilité du projet.

Le front-end est développé à l'aide du framework React, qui repose sur une architecture par composants et un système de rendu dynamique. Le langage TypeScript a été choisi afin de renforcer la fiabilité du code grâce au typage statique, limitant ainsi les erreurs potentielles lors du développement. Pour la gestion des styles, SASS est utilisé afin de structurer le CSS en fichiers dédiés à chaque composant ou page, tout en centralisant les variables (couleurs, tailles, espacements, etc.).

Les données sportives du projet étant fortement liées entre elles (par exemple, un Grand Prix est associé à une date, un circuit, une grille de départ et des commentaires

utilisateurs), le choix d'une base de données relationnelle s'est imposé. La solution retenue est Neon, une plateforme PostgreSQL serverless, permettant de créer les tables, de gérer les relations et de stocker les données de manière fiable et structurée (Voir annexes n°1).

## Maquettage

Le travail de maquettage a été une étape essentielle afin de définir l'identité visuelle du site avant la phase de développement. Plusieurs outils ont été utilisés pour construire une charte graphique cohérente, réaliser des maquettes complètes et visualiser le rendu final de l'application.

Dans un premier temps, une palette de couleurs a été définie à l'aide d'un générateur de couleurs. Celle-ci s'articule autour du rouge iconique de la discipline, associé à sa couleur complémentaire pour mettre en valeur les éléments importants et les effets de survol.

Le choix des typographies repose sur trois critères principaux : une apparence sérieuse et professionnelle, une licence libre de droits et une variété suffisante de graisses. La police *Playfair Display* a été retenue pour les titres et sous-titres, tandis que *Barlow* est utilisée pour les textes courants, afin d'assurer une bonne lisibilité.

Playfair Display

My Formula One tracker

Barlow

My Formula One tracker

L'iconographie provient de la bibliothèque ***React Icons***, qui offre un large choix d'icônes gratuites, cohérentes et faciles à intégrer. À cela s'ajoutent des créations graphiques originales réalisées avec InDesign et Procreate, apportant une dimension personnelle et distinctive au projet.

Une fois la charte graphique définie, l'outil Figma a été utilisé pour concevoir des maquettes haute fidélité. Celles-ci couvrent un grand nombre de pages du site et prennent en compte trois formats d'écran (ordinateur, tablette et mobile). Les composants Figma ont permis de gérer efficacement la redondance des éléments récurrents (header, footer, etc.), facilitant les mises à jour globales.

Enfin, Procreate a également été utilisé dans un second temps pour retravailler certains visuels, améliorer le rendu graphique de certaines pages et intégrer des illustrations originales (Voir annexe n°2).

Figma complet : My F1 Tracker Figma

# Développement

## Fonctionnalités

Le site propose plusieurs fonctionnalités principales, réparties en trois catégories selon le type de compte utilisé lors de la navigation.

Lorsqu'un utilisateur ne possède pas de compte, le site permet tout de même l'accès à plusieurs pages de consultation :

- Page **Programme** : permet de consulter le programme annuel de l'année en cours, affiché sous forme de calendrier ou de liste.
- Page **Résultat** : permet de consulter les résultats de l'année en cours, avec la possibilité d'utiliser des filtres de recherche (boutons et menus déroulants) afin d'afficher un type de résultat spécifique :
    - Classement des pilotes
    - Classement des écuries
    - Résultat par Grand Prix
    - Podiums de l'année en cours
    - Résultats de l'année d'une écurie
    - Résultats de l'année d'un pilote
- Page **Grand Prix** : accessible depuis le calendrier, permet de consulter les détails d'un Grand Prix, incluant l'emploi du temps du week-end, les classements et les informations du circuit.
- Page **Grille** : permet de consulter la grille de la saison en cours (écuries et leurs deux pilotes titulaires), ainsi que diverses statistiques de l'année telles que les points, podiums, victoires et abandons (DNF).
- Pages **d'Inscription** et de **Connexion** : permettent à l'utilisateur de :
    - créer un compte à l'aide d'un nom d'utilisateur, d'une adresse e-mail et d'un mot de passe (stocké de manière hachée dans la base de données), avec une case à cocher pour l'acceptation des conditions générales d'utilisation.

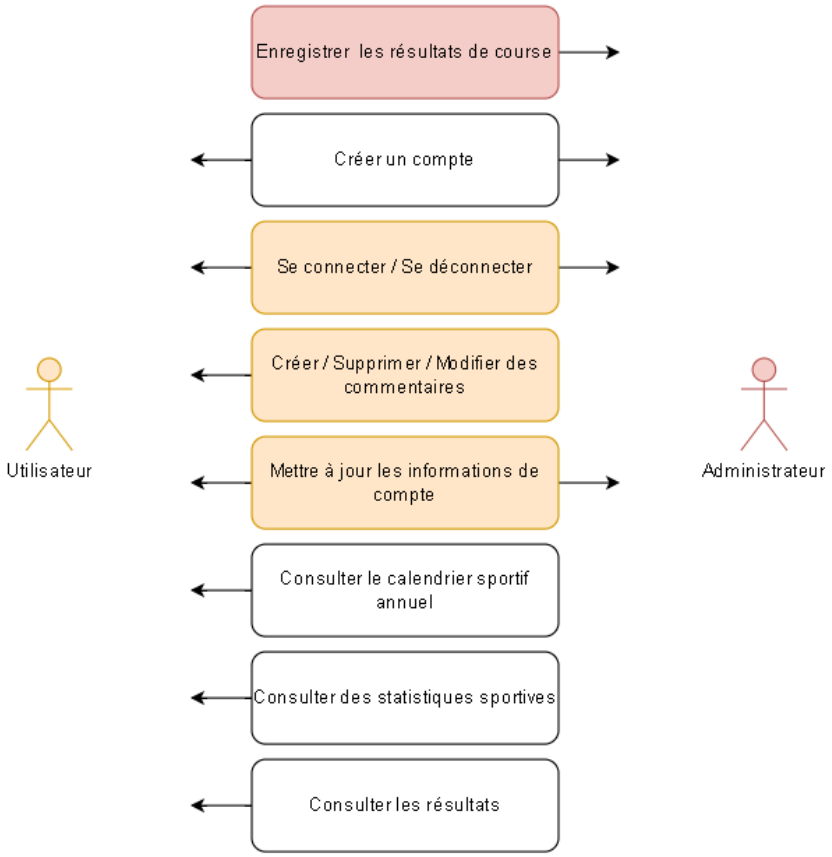
- se connecter à un compte existant à l'aide d'une adresse e-mail et d'un mot de passe.

Lorsqu'un utilisateur est inscrit et connecté, il accède à des fonctionnalités supplémentaires liées à la gestion de son compte :

- Page **Paramètres de compte** : permet de modifier les informations personnelles du compte, telles que le mot de passe, le nom d'utilisateur ou l'adresse e-mail.
- Page **Grand Prix** : permet la gestion de commentaires associés à un Grand Prix (création, modification et suppression). Ces commentaires sont visibles uniquement par l'utilisateur concerné.

Enfin, lorsqu'un utilisateur dispose d'un compte administrateur, il accède à une fonctionnalité spécifique :

- Page **Administrateur** : permet l'insertion et la gestion des résultats de course pour chaque Grand Prix déjà disputé au cours de la saison en cours.



# Sécurité

La sécurité d'une application de ce type vise à protéger les données des utilisateurs, à garantir leur intégrité et à empêcher les accès non autorisés. Elle concerne à la fois le front-end, le back-end et la base de données.

Un point central de la sécurité repose sur l'authentification et la gestion des comptes. Le système d'inscription et de connexion est géré à l'aide de tokens (JSON Web Tokens). Lorsqu'un utilisateur crée un compte ou se connecte avec une adresse e-mail et un mot de passe valide, un token est généré par le contrôleur. Celui-ci sert de clé d'authentification : après validation par le back-end, il permet l'accès aux pages et fonctionnalités nécessitant un compte utilisateur. Le token permet également de différencier les différents types de comptes (utilisateurs et administrateurs).

Le stockage des mots de passe passe obligatoirement par un hachage avant leur enregistrement dans la base de données. Pour cela, la bibliothèque bcrypt est utilisée afin de garantir un niveau de sécurité élevé et d'éviter toute conservation de mots de passe en clair.

Il est également primordial de valider et de nettoyer les données avant leur stockage en base de données. Les informations issues des formulaires d'inscription ainsi que les commentaires font l'objet de plusieurs vérifications et sont encodées et échappées afin de prévenir les injections SQL, les failles XSS (cross-site scripting) ou toute tentative de surcharge. Dans le même objectif, l'ensemble des requêtes vers la base de données est préparé.

Les fonctionnalités sensibles, telles que l'ajout des résultats de course ou la suppression de commentaires, sont sécurisées par l'utilisation des tokens. Aucune action critique n'est accessible via de simples liens publics, ce qui limite les risques de manipulation non autorisée.

Enfin, la sécurité de l'application est renforcée par l'utilisation du middleware Helmet. Celui-ci permet de configurer automatiquement plusieurs en-têtes HTTP de sécurité, contribuant à la protection contre certaines attaques courantes telles que le XSS ou l'interprétation incorrecte du type de contenu par le navigateur.

La sécurité a été intégrée dès la conception du projet à travers une gestion rigoureuse de l'authentification, des droits d'accès et de la protection des données.

## Accessibilité

L'accessibilité vise à rendre l'application utilisable par le plus grand nombre, y compris les personnes en situation de handicap, mais aussi les utilisateurs confrontés à des contraintes techniques. Elle contribue, entre autres, à améliorer l'expérience utilisateur globale.

Un point majeur de l'accessibilité, permettant l'utilisation des lecteurs d'écran et de la navigation au clavier, repose sur la sémantique HTML. L'utilisation de balises adaptées à leur contenu, ainsi que le respect de la hiérarchie des éléments, permet aux technologies d'assistance de comprendre efficacement la structure de la page et facilite ainsi la navigation des utilisateurs au sein du site web.

L'application est également entièrement navigable sans souris. À l'aide de la touche Tab pour se déplacer entre les éléments interactifs (<a>, <button>, etc.) et de la touche Entrée pour interagir, un utilisateur ne pouvant pas utiliser de souris peut accéder à l'ensemble des informations. Un focus visible permet par ailleurs d'identifier clairement l'élément actuellement sélectionné.

Puis, le contraste et la lisibilité constituent également des points clés pris en compte dès la phase de conception de la maquette. Le choix des couleurs de fond et de texte garantit un contraste suffisant, tandis que la taille et la graisse des polices assurent une lecture confortable pour tous les utilisateurs. De plus, les informations ne sont pas transmises uniquement par la couleur : elles sont accompagnées de texte et d'icônes explicites lorsque cela est nécessaire.

Concernant les contenus visuels, un attribut alt est ajouté aux images afin de fournir une description textuelle. Cela permet aux utilisateurs de lecteurs d'écran, ainsi qu'aux personnes disposant d'une connexion limitée, d'accéder au contenu visuel de manière alternative. Une autre bonne pratique est de réduire la qualité des images en fonction de leur taille, et de les convertir en format WEBP pour réduire leur poids.

L'accessibilité implique également la possibilité d'utiliser l'application sur différentes tailles d'écran. Dans cette optique, le projet intègre un design responsive, pensé dès la phase de maquettage sur Figma, avec trois formats principaux : desktop, tablette et mobile. Cette approche permet de prendre en compte les contraintes des petits écrans dès la conception.

Enfin, une fonctionnalité de traduction a été mise en place afin de rendre le site accessible en français et en anglais. Pour cela, le système de traduction i18n est utilisé. Son intégration au projet repose sur la création d'un fichier i18n.ts pour la configuration globale, ainsi que de fichiers JSON contenant les textes traduits. Dans les composants React, le hook useTranslation() et la fonction t("key") sont utilisés directement dans les balises HTML. Un

bouton de changement de langue permet à l'utilisateur de basculer facilement entre les deux langues.

L'accessibilité a ainsi été prise en compte dès la conception du projet, notamment à travers l'utilisation de balises sémantiques, et une attention particulière portée à un design responsive. Toutefois, certaines améliorations pourraient encore être envisagées, notamment par la réalisation de tests plus approfondis.

## Explication de code : Calendrier annuel (Annexe n°3)

La fonctionnalité front-end majeure du site est le calendrier annuel, qui affiche le programme de manière dynamique sous forme de calendrier, à l'aide de composants React.

Tout d'abord, je crée une page ScheduleCal.tsx, qui correspond à l'affichage de la page lorsque l'utilisateur clique, entre autres, sur l'onglet « Programme » dans le header. Dans cette page, un appel à l'API est effectué via un fetch (l. 39) afin de récupérer les informations stockées dans la base de données (table schedule). Les données reçues sont ensuite stockées dans un useState() (l. 43), ce qui permet à React de déclencher un re-render automatique dès que les données deviennent disponibles. Tant que celles-ci ne sont pas chargées, une animation de chargement s'affiche à la place, grâce à la gestion d'un état loading (l. 37).

La fonction contenant le fetch (fetchSchedule) est appelée à l'intérieur d'un useEffect() (l. 52) afin d'éviter l'exécution du code à chaque render. Pour générer l'affichage des mois, je crée ensuite un tableau d'une longueur de 12 (l. 54), correspondant aux douze mois de l'année. L'affichage dynamique de la page est alors construit à l'aide de balises HTML comprenant le titre, les boutons de navigation et une div contenant une boucle map. À chaque itération, cette boucle récupère les informations des Grands Prix (dates, programme, circuit, etc.) ayant lieu durant le mois concerné (l'index du tableau correspondant au numéro du mois) les stocke dans un tableau dédié (l. 77), puis exécute le composant du mois correspondant (l. 91).

Je crée ensuite le composant MonthCalendar.tsx, qui reçoit depuis son composant parent le numéro du mois, l'année, ainsi que le tableau des Grands Prix associés. Afin de gérer simplement et efficacement le nombre de jours par mois, j'utilise la bibliothèque externe **date-fns**, qui fournit plusieurs fonctions utiles : startOfMonth (date du premier jour du mois, l. 22), endOfMonth (date du dernier jour du mois, l. 23) et eachDayOfInterval (création d'un tableau de dates comprises entre deux bornes). Grâce à ces fonctions, l'ensemble des jours du mois est stocké dans un tableau.

Pour gérer l'indentation visuelle du calendrier en fonction du jour de la semaine sur lequel débute le mois, j'initialise également un tableau d'une longueur variable, déterminée par le jour de la semaine du premier jour du mois (l. 28). Le composant retourne ensuite les balises HTML nécessaires à l'affichage : une div principale, le nom du mois (provenant d'un fichier JSON), ainsi qu'une div contenant deux boucles map. La première génère des div vides correspondant à l'indentation, tandis que la seconde génère un composant enfant pour chaque jour du mois.

Le composant DayCalendar.tsx reçoit de son parent la date du jour ainsi que le tableau des Grands Prix du mois. À partir de ces données, je récupère le numéro du jour du mois (l. 24) et le compare au numéro du jour de chaque Grand Prix présent dans le tableau (l. 26). En cas de correspondance (stockée dans une constante race) les informations de l'événement sont récupérées (l. 28). Le composant retourne alors une balise HTML <a> cliquable, accompagnée d'un élément qui s'affiche au survol grâce à un useState() (l. 20) et aux événements onMouseEnter et onMouseLeave (l. 32). En l'absence de correspondance, le composant affiche simplement une div contenant le numéro du jour.

Lorsqu'un jour correspondant à un Grand Prix est survolé, le composant HoverCalendar.tsx s'affiche. Il s'agit d'une simple balise <article> qui présente dynamiquement les informations reçues de son composant parent, telles que le nom de l'événement, la date de début, la date de fin et le mois.

Ainsi, cette implémentation permet d'obtenir un calendrier annuel affichant l'ensemble des mois et des jours, ainsi que les Grands Prix issus des données de la table schedule de la base de données. L'utilisateur peut accéder rapidement aux informations clés de chaque événement grâce à un effet de survol clair et intuitif.

## Explication de code : Sauvegarde des résultats de courses dans la base de données (Annexe n°4)

Côté back-end, la sauvegarde des résultats de course après chaque Grand Prix a représenté un défi important.

Le principe est le suivant : une page administrateur contient un formulaire complexe, conçu notamment à l'aide de SortableJS, une bibliothèque JavaScript permettant de créer des listes dont les éléments sont déplaçables. Il est ainsi possible de générer une liste de vingt éléments représentant les pilotes, puis de les ordonner selon le résultat de la course. Chaque élément de la liste contient également des champs permettant de renseigner différentes informations (disqualification, temps de course…). Le principal enjeu est ensuite, après la

validation du formulaire, de stocker l'ensemble de ces données de manière cohérente dans la base de données.

Dans le contrôleur addRaceResult, les informations du formulaire ainsi que le numéro du Grand Prix (round) sont récupérées depuis req.body par déstructuration (l. 4). L'ensemble du traitement est ensuite encapsulé dans un bloc try/catch, afin de pouvoir intercepter d'éventuelles erreurs et retourner un message explicite au front-end. Les données reçues sont structurées sous la forme d'un tableau d'objets, chacun représentant un pilote avec sa position et les informations liées à sa course.

Une boucle for...of (l. 8) permet alors de parcourir ce tableau. À chaque itération, le code vérifie que toutes les données obligatoires sont bien présentes (l. 9) et applique une règle métier simple : si un pilote est disqualifié, ses points sont automatiquement mis à zéro.

Les requêtes SQL sont ensuite préparées à l'aide d'un map (l. 13), qui génère une requête INSERT INTO pour chaque pilote. Ces requêtes permettent d'enregistrer l'ensemble des résultats dans la table correspondante. Grâce à Promise.all() (l. 20), toutes les insertions sont exécutées de manière concurrente. Une réponse avec un statut HTTP 201 est enfin envoyée afin d'indiquer au front-end que les résultats ont bien été enregistrés (l. 22).

Cette approche permet d'insérer l'ensemble des vingt lignes de résultats en une seule opération logique, ce qui réduit à la fois la complexité du code et la consommation de ressources, tout en garantissant de bonnes performances.

# Déploiement

Le déploiement du site a été réalisé afin de rendre le projet accessible en ligne et confronter son fonctionnement à un environnement de production. Il s'est fait via deux différents services : Vercel pour le front-end, et Render pour l'API.

Le front-end a été déployé sur la plateforme **Vercel**, spécialisée dans l'hébergement d'applications front-end, notamment celles développées avec le framework React. Ce service permet une mise en ligne rapide, et un déploiement automatique à chaque mise à jour du dépôt GitHub. Le forfait gratuit est suffisant pour les besoins du projet, tout en garantissant de bonnes performances, et une expérience utilisateur fluide et agréable.

Le back-end a été, lui, déployé sur **Render**, qui permet l'hébergement d'API Node.js. Cette solution a été choisie également pour sa simplicité de configuration et sa compatibilité avec les technologies utilisées dans le projet. Là aussi, le forfait gratuit a été utilisé, ce qui implique cette fois-ci certaines contraintes, notamment un temps de latence d'une cinquantaine de secondes au démarrage du serveur après une période d'activité.

Ce choix de séparer le déploiement du front-end et du back-end reflète une architecture moderne, facilitant la maintenance et les évolutions futures de l'application. Malgré les limitations liées au budget, cette solution reste adaptée à un projet pédagogique.

# Tests et validations

Les tests et validations permettent de prévenir les régressions et d'assurer la fiabilité des fonctionnalités développées.

Des tests fonctionnels ont été réalisés tout au long de la phase de développement. Il s'agit des tests les plus importants, car ils permettent de vérifier l'état des fonctionnalités en cours d'implémentation, qu'il s'agisse des formulaires (inscription, connexion), des affichages (calendrier, résultats), de la gestion des commentaires ou des accès restreints. Les parcours utilisateurs ont également été testés selon les trois contextes possibles : utilisateur non connecté, utilisateur connecté et administrateur connecté.

Des tests manuels ont également été effectués à l'aide d'un navigateur afin de vérifier la navigation, le design, les messages d'erreur et l'expérience utilisateur globale. Ces tests permettent notamment de valider le bon fonctionnement des messages d'erreur, des boutons, des liens et des formulaires. Ils ont été réalisés sur différents formats d'écran afin de vérifier le comportement responsive du site.

Des tests de validation des données ont été mis en place pour s'assurer du bon traitement des informations issues des formulaires. Ces tests permettent de vérifier que les données valides sont correctement prises en compte par le front-end et stockées par le back-end dans la base de données. En cas de données incorrectes, des messages d'erreur clairs sont affichés, les informations ne sont pas enregistrées et l'utilisateur est guidé de manière compréhensible.

Pour la mise en place de ces tests, plusieurs outils ont été utilisés :

- Le **navigateur web** a permis de tester l'application en conditions réelles, de vérifier les formulaires et les parcours utilisateurs, ainsi que la lisibilité, le responsive et certains aspects de l'accessibilité (navigation au clavier via la touche Tab).
- L'extension **Thunder Client**, intégrée à Visual Studio Code, a été utilisée pour tester les routes de l'API et vérifier les réponses du serveur sans passer par l'interface utilisateur.
- Des tests unitaires ont également été réalisés avec **Jest**, de manière ciblée sur le userController.

- L'outil **Lighthouse** a permis de vérifier les performances de base, ainsi que certains critères d'accessibilité et de bonnes pratiques.
- L'outil **WAVE** est spécialisé coté accessibilité, a été utilisé pour supprimer les erreurs empêchant certains utilisateurs de profiter d'une bonne expérience d'utilisation.
- L'utilisation régulière des **logs console** constitue une bonne pratique pour suivre et corriger les erreurs tout au long du développement.

Grâce à une utilisation réfléchie des tests, les principales fonctionnalités de l'application ont pu être validées. Plusieurs ajustements ont néanmoins été nécessaires, notamment l'amélioration de messages d'erreur initialement peu explicites, une optimisation du formulaire de saisie des résultats de course, jugé trop énergivore, ainsi qu'une meilleure gestion du design responsive. Le site est désormais stable pour les cas d'usage prévus. Il est toutefois important de souligner que des améliorations restent possibles, notamment une couverture plus complète des cas extrêmes et une automatisation des tests lors des mises à jour.

# Difficultés rencontrées

Durant le processus de développement, plusieurs difficultés ont été rencontrées, nécessitant de prendre du recul et de réfléchir à des solutions adaptées.

Tout d'abord, le choix du type de base de données a constitué un véritable casse-tête. Les données du projet étant facilement organisables sous forme de tables liées entre elles par des clés étrangères, l'utilisation d'une base de données relationnelle s'est rapidement imposée. Cependant, ne disposant initialement que des connaissances nécessaires pour relier une API Node.js à une base de données NoSQL (MongoDB), j'ai dû consacrer du temps à l'apprentissage de la connexion et de l'utilisation d'une base de données SQL. J'ai finalement opté pour **Neon**, une plateforme serverless permettant de gérer une base de données PostgreSQL. À partir de cette solution, j'ai pu créer les tables ainsi que leurs relations de manière efficace.

Ensuite, une difficulté importante a concerné la compréhension du fonctionnement de l'authentification et de la restriction d'accès côté front-end. Après la génération du token par jsonwebtoken dans le contrôleur, j'ai eu du mal à saisir que la vérification de ce token devait être effectuée côté client afin d'adapter l'affichage de la vue en fonction de l'état de connexion de l'utilisateur. Il a donc été nécessaire de stocker le token dans un état global, accessible à l'ensemble de l'application, et de vérifier régulièrement sa validité afin de retirer

les droits en cas d'expiration. Pour répondre à cette problématique, j'ai intégré **Zustand**, qui permet, entre autres, la création de hooks globaux via le fichier authStore.tsx. Cette solution permet à l'interface de prendre en compte en permanence l'état d'authentification de l'utilisateur et d'afficher les éléments appropriés.

Enfin, la fonctionnalité de mot de passe oublié a posé problème lors de la mise en production du site. En phase de développement, j'avais mis en place **nodemailer**, permettant l'envoi d'un e-mail contenant un lien temporaire associé à un token afin de réinitialiser le mot de passe. Toutefois, ce système ne fonctionne pas sur les projets déployés via le service **Render**. Cette fonctionnalité a donc été temporairement désactivée, dans l'attente de la mise en place d'une solution compatible avec un environnement de production.

Certaines difficultés ont ainsi pu être surmontées, notamment celles liées à la base de données et à l'authentification, permettant d'aboutir à un site fonctionnel. D'autres problématiques nécessitent toutefois davantage de recul et de recherches avant de pouvoir être résolues de manière pérenne.

# Bilan et perspectives

Ce projet m'a permis de renforcer considérablement mes compétences en création de sites web, notamment avec le framework React et Node.js/Express, mais aussi mes compétences en maquettage et en design. J'ai également acquis de nouvelles connaissances concernant les contraintes liées à la mise en production, à l'accessibilité, à la sécurité, ainsi qu'à la gestion de projet de manière générale.

L'application peut évoluer, notamment grâce à l'implémentation de nouvelles fonctionnalités. Tout d'abord, la fonctionnalité de mot de passe oublié pourrait être réintégrée, dans la mesure où elle est essentielle dans un projet web comportant des comptes utilisateurs. Ensuite, certaines fonctionnalités exclues du MVP, par choix de priorisation, sont naturellement des axes d'amélioration, tels que les classements personnels (à l'aide de SortableJS) ou un système de favoris (pilotes, Grands Prix, écuries…).

Enfin, si le site est amené à prendre de l'ampleur, il serait envisageable d'opter pour un déploiement payant du back-end, afin d'améliorer les performances et de réduire la frustration des utilisateurs.

# Conclusion

Finalement, ce projet a permis de mettre en œuvre l'ensemble des compétences acquises durant la formation DWWM, tant sur les aspects techniques que méthodologiques, de la conception à la mise en production. Chaque étape a nécessité une réflexion éclairée.

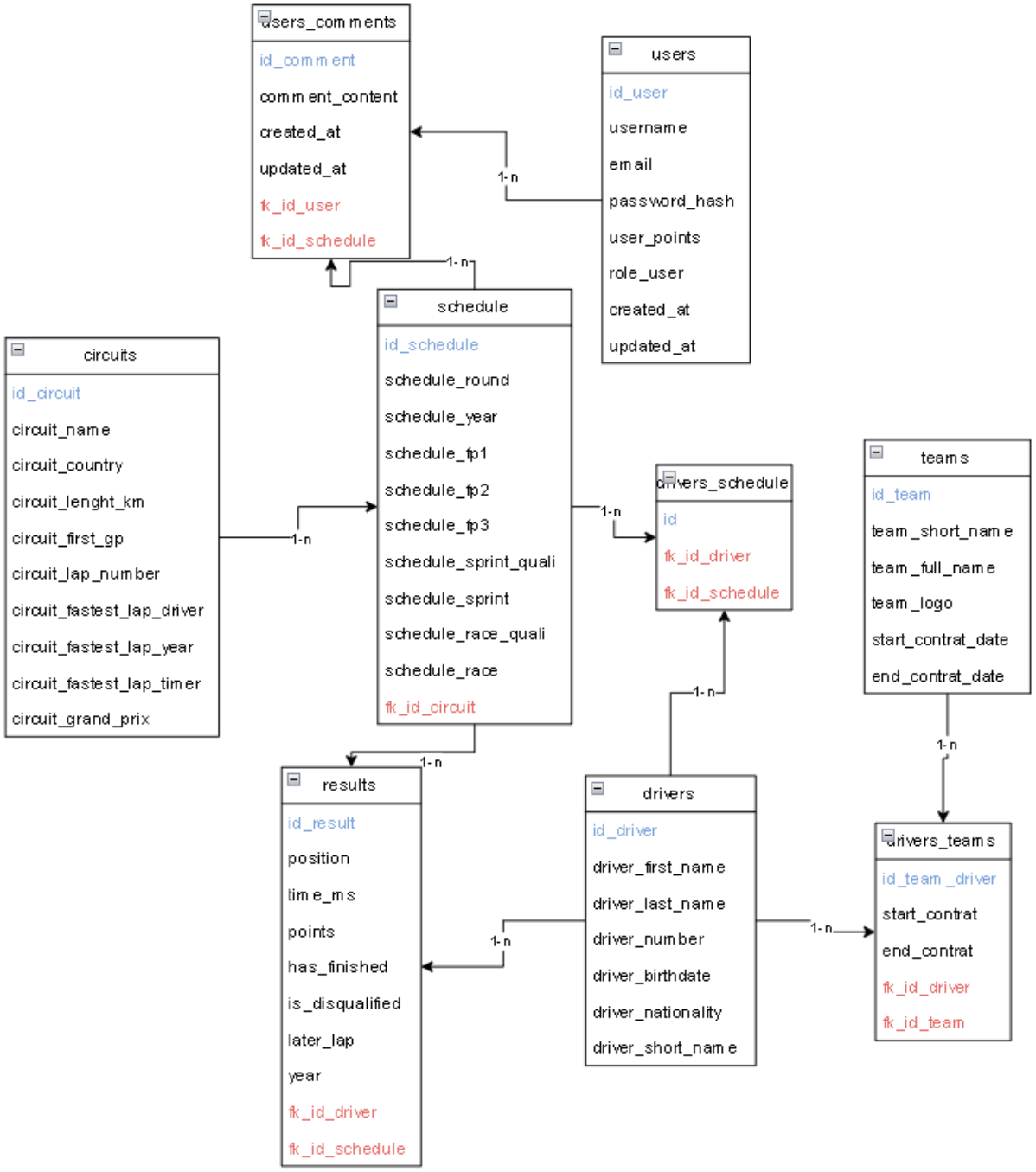
Le développement du site a mobilisé des compétences variées, notamment en front-end avec React, en back-end avec Node.js et Express, ainsi qu'en gestion des données, sécurité et expérience utilisateur. Les tests ont permis d'assurer la fiabilité des fonctionnalités principales, tandis que les problèmes rencontrés ont contribué à approfondir la compréhension des enjeux d'un projet web complet.

Pour conclure, l'application répond aux objectifs définis dans le périmètre du MVP et constitue une base fonctionnelle et évolutive.

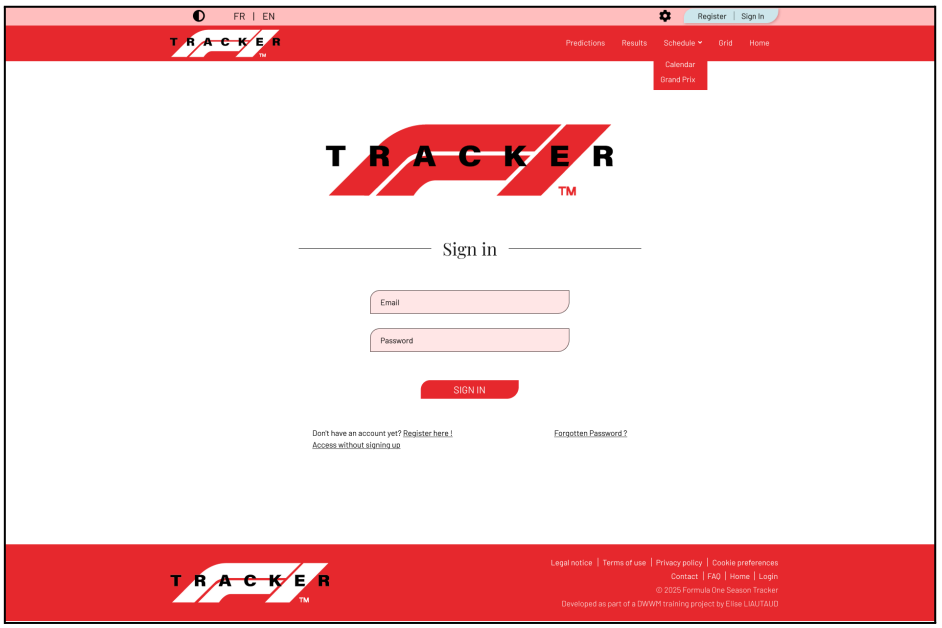
Au-delà du projet en lui-même, cette expérience servira de référence pour de futurs projets, en particulier dans un contexte professionnel. Elle devient une base de travail réutilisable et perfectible pour des projets à venir.

# Annexes

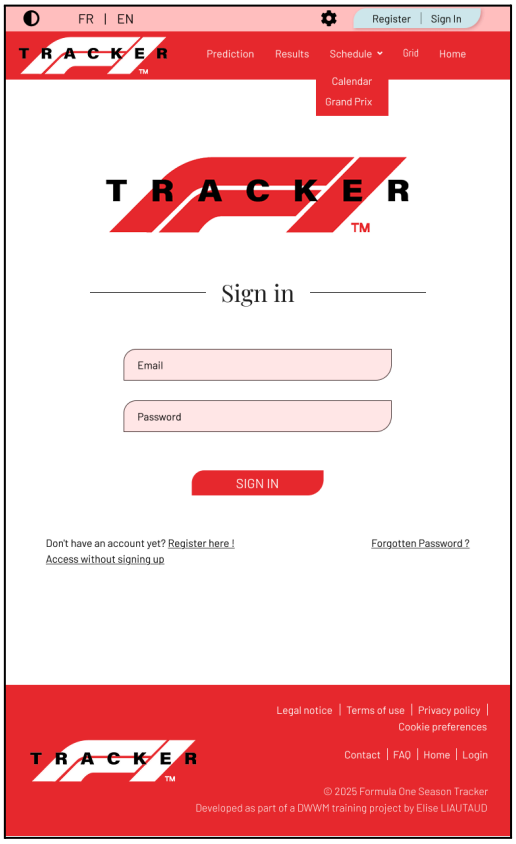
## Annexe 1 : MLD (Modèle Logique de Données)



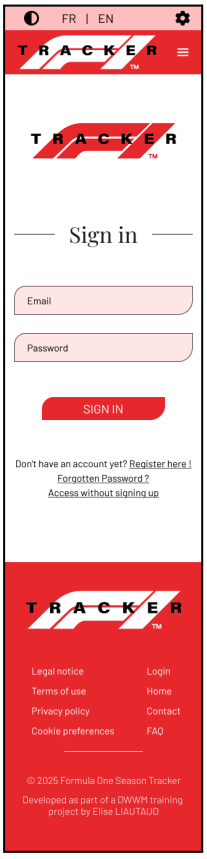
# Annexe 2 : Échantillons Figma



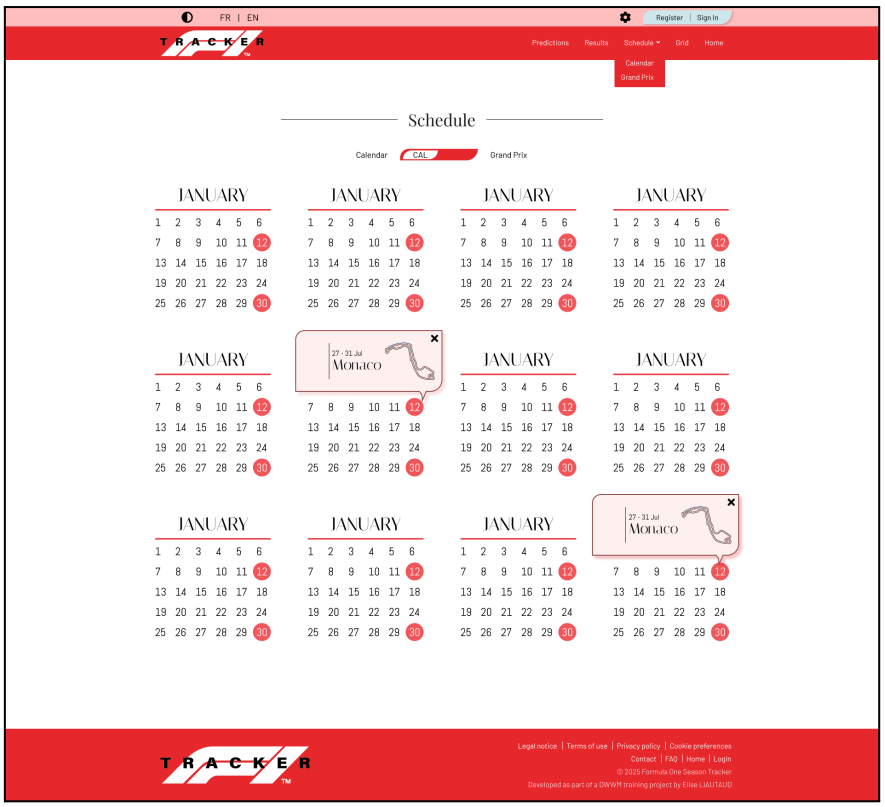

Page de connexion (1900px)




Page de connexion (768px)




Page de connexion (320px)




Page de calendrier annuel (1900px)


Page de Grand prix (1900px)


Maquette Procreate page de Grand Prix (desktop)

Lien maquette fimga :
https://www.figma.com/design/RVRjye5E1jK6KznSd850pv/f1-tracker-2026-v2?node-id=147-804&t=YuUkD4U7IvRkMZSp-0

# Annexe 3 : Code Calendrier annuel

## ScheduleCal.tsx

```tsx
import { useCallback, useEffect, useState } from "react"
import { useTranslation } from "react-i18next"
import { Link } from "react-router-dom"

import Title from "../components/repeat/H1Title"
import Month from "../components/repeat/MonthCalendar"
import Loading from "../components/global/Loading"

interface raceProp {
    id_schedule: number,
    schedule_round: number,
    schedule_fp1_start: Date,
    schedule_race_start: Date,
    circuit_name: string,
    circuit_grand_prix: string
}

interface gpOfTheMonthProp {
    round: number,
    circuit_name: string,
    fp1_day: number
    race_day: number,
    month: number,
    name: string
}

const ScheduleCal = () => {
    const { t } = useTranslation()
    const year = new Date().getFullYear()

    const [ loading, setLoading ] = useState<boolean>(true)
    const [ error, setError ] = useState<string|null>(null)
    const [ races, setRaces ] = useState<raceProp[]>([])

    const fetchSchedule = useCallback( async () => {
        try {
            setLoading(true)

            const res = await fetch(`${import.meta.env.VITE_API_URL}/races/schedule/${year}`)
            if(!res.ok) throw new Error("Error fetch circuits")

            const data = await res.json()
            setRaces(data.allRacesofTheYear)
                // eslint-disable-next-line @typescript-eslint/no-explicit-any
        } catch (error: any) {
            setError(error.message)
        }finally{
            setLoading(false)
        }
    }, [year ])

    useEffect( () => { fetchSchedule() }, [fetchSchedule])

    const months = Array.from({ length: 12 })

    if(loading) return (
    <main id="loading-main">
        <Loading loading={loading}/>
    </main>
    )
    if(error) return (<p>{error}</p>)
```

```
    return (
        <main id="schedule">
            <Title title={t('schedule')}/>

            <div>
                <Link to='/schedule/calendar' className="shadowed-button selected-filter">{t('calendar')}</Link>
                <Link to='/schedule/list' className="shadowed-button">Grand Prix</Link>
            </div>

            <div id="cal-container">
                {months.map( ( _, i ) => {

                    // Récolte des gp du mois
                    const gpOfTheMonth:gpOfTheMonthProp[] = []

                    races.forEach((race) => {
                        const gpMonth = new Date(race.schedule_race_start).getMonth()
                        if(gpMonth === i) gpOfTheMonth.push({
                            round: race.schedule_round,
                            race_day: new Date(race.schedule_race_start).getDate(),
                            circuit_name: race.circuit_name,
                            fp1_day: new Date(race.schedule_fp1_start).getDate(),
                            month: new Date(race.schedule_fp1_start).getMonth(),
                            name: race.circuit_grand_prix
                        })
                    })

                    return <Month year={2025} key={i} month={i} gpOfTheMonth={gpOfTheMonth} />
                })}
            </div>
        </main>
    )
}

export default ScheduleCal
```

## MonthCalendar.tsx

```tsx
import { startOfMonth, endOfMonth, eachDayOfInterval } from "date-fns"
import { useTranslation } from "react-i18next";

import Day from "./DayCalendar";

type MonthProp = {
    month : number, // 0 = jan...
    year : number,
    gpOfTheMonth: {
        round: number,
        race_day: number,
        circuit_name: string,
        fp1_day: number,
        month: number,
        name: string
    }[]
}

const Month = ({ month, year, gpOfTheMonth }: MonthProp) => {
    const { t } = useTranslation();

    const start = startOfMonth(new Date(year, month))
    const end = endOfMonth(new Date(year, month))

    let firstDayDate = start.getDay()
    if (firstDayDate === 0) firstDayDate = 7

    const indentArray = Array.from({ length : firstDayDate-1 })

    const days = eachDayOfInterval({ start, end })

    return(
        <div className="month-container">
            <h2>{t(`months.${month}`).toUpperCase()}</h2>
            <div className="line"/>
            <div className="days-container">
                {indentArray.map( (_, i) => (
                    <div key={i} />
                ) )}

                {days.map((el, i) => <Day dayOfMonth={el} gpOfTheMonth={gpOfTheMonth} key={i} />)}

            </div>
        </div>
    )
}

export default Month
```

## DayCalendar.tsx

```tsx
import { Link } from 'react-router-dom'
import { useState } from 'react'

import HoverCalendar from './HoverCalendar'

type dayProp = {
    dayOfMonth: Date,
    gpOfTheMonth: {
        round: number,
        race_day: number,
        circuit_name: string,
        fp1_day: number,
        month: number,
        name: string
    }[]
}

const Day = ({ dayOfMonth, gpOfTheMonth }: dayProp) => {

    const [ isHovered, setIsHovered ] = useState(false)
    const handleMouseEnter = () => {setIsHovered(true)}
    const handleMouseLeave = () => {setIsHovered(false)}

    const dayNum = dayOfMonth.getDate()

    const gpEntry = gpOfTheMonth.find(({ race_day }) => race_day === dayNum) // recherche de date de GP --> jour du gp ou undefined

    const race = gpEntry ? gpOfTheMonth.find(el => el.round === gpEntry.round) : undefined // si gpEntry existe, race = round du gp

    return race ? (
        <div className="gp-link-hover-container" key={dayNum} >
            <Link to={`/grand-prix/${race.round}`} key={dayNum} className="highlighedDay" onMouseEnter={handleMouseEnter} onMouseLeave={handleMouseLeave}>
                {dayNum}
            </Link>
            {isHovered && <HoverCalendar name={race.name} dateStart={race.fp1_day} dateEnd={race.race_day} month={race.month}/>}
        </div>
    ) : (
        <div key={dayNum} className='commonDay'>
            {dayNum}
        </div>
    );
}

export default Day
```

## HoverCalendar.tsx

```tsx
import { useTranslation } from "react-i18next"

type hoverCalendarProp = {
    name: string,
    dateStart: number
    dateEnd: number
    month: number
}

const HoverCalendar = ({ name, dateStart, dateEnd, month }: hoverCalendarProp) => {

    const { t } = useTranslation()

    return(
        <article className="hoverCalendar-article">
            <div className="line-vertical" />
            <div>
                <p>{dateStart} - {dateEnd} {t(`months-mini.${month}`)}</p>
                <h3>{t(`country.${name}`)}</h3>
            </div>
            <img src={`/circuits/png/${name}.png`} alt={`track ${t(`country.${name}`)}`} />
        </article>
    )
}

export default HoverCalendar
```

# Annexe n°4 : Code Sauvegarde résultats de course

```
exports.addRaceResult = async (req, res) => {
    const { formData, round } = req.body

    try {

        for (const el of formData){
            if(!el.time && el.has_finished && !el.later_lap) return res.status(400).json({ message: `data missing for position n°${el.position}` })
            if(el.is_disqualified) el.points = 0
        }

        const queries = formData.map( el =>
            db.none(
                'INSERT INTO results (fk_id_driver, fk_id_schedule, position, points, has_finished, time_ms, year, is_disqualified, later_lap) VALUES($1, $2, $3, $4, $5, $6, $7, $8, $9)',
                [el.id_driver, round, el.position, el.points, el.has_finished, el.time, new Date().getFullYear(), el.is_disqualified, el.later_lap]
            )
        )

        await Promise.all(queries);

        res.status(201).json({ message: "results-inserted" });

    } catch (error) {
        res.status(500).json({ message: error.message })
    }
}
```