# Rapport de Veille Technologique • Tab Magiques

Lussandre Lederrey - Corentin Batard - Allan Maccrez

---

## Sommaire



---

## Partie 1 : Languages de programmation

### Introduction

La programmation est le cœur de chaque application. Le choix du langage détermine la performance, la facilité de développement et la maintenance. Dans notre projet, le langage définira la logique métier de notre application.

### Présentation des Solutions

Les langages considérés sont : **Python**, **JavaScript**, **PHP** et **Java**.

### Comparaison des Solutions

- **Python** : Facile à apprendre et à lire, offre une grande modularité, et dispose d'une riche bibliothèque standard. C'est très populaire pour le développement web.
- **JavaScript** : Principalement utilisé côté client, mais Node.js permet également une utilisation côté serveur. Il est asynchrone et basé sur des événements.
- **PHP** : Spécifiquement conçu pour le développement web. Il est intégré à de nombreux systèmes de gestion de contenu.
- **Java** : Orienté objet, portable et performant, il est largement utilisé dans les grandes entreprises pour les applications d'entreprise.

### Conclusion

Nous avons choisi **Python** pour sa simplicité, sa modularité et sa popularité dans le développement web.

---

## Partie 2 : Système de gestion de base de données

### Introduction

Le stockage et la gestion des données sont essentiels à notre application pour suivre les inscriptions et les adhérents. Le choix de la base de données influencera la performance et la fiabilité de l'application.

### Présentation des Solutions

Les systèmes de gestion de bases de données considérés sont : **MySQL**, **SQLite** et **PostgreSQL**.

### Comparaison des Solutions

- **MySQL** : Populaire, fiable, rapide et facile à utiliser. Il prend en charge de grandes bases de données.
- **SQLite** : Base de données légère, ne nécessite pas de processus serveur séparé. Idéal pour les applications plus légères.
- **PostgreSQL** : Open-source, extensible et conforme à SQL. Il offre des fonctionnalités avancées.

### Conclusion

Nous avons opté pour **MySQL** pour sa popularité, sa fiabilité et sa capacité à gérer de grandes bases de données.

# Partie 3 : Librairie d’accès aux bases de données (ORM)

## Introduction

Les ORM (Object-Relational Mapping) permettent une interaction simplifiée entre l'application et la base de données en mappant les objets du code aux enregistrements de la base de données. Pour notre projet, cela facilitera la gestion des inscriptions et des informations des adhérents.

## Présentation des Solutions

Les ORM considérés sont : **Django ORM**, **SQLAlchemy**, **Sequelize**, **Doctrine**, et **Hibernate**.

## Comparaison des Solutions

- **Django ORM** : Fourni avec le framework Django, il permet une intégration transparente avec les modèles et vues Django.
- **SQLAlchemy** : Flexible et puissant pour Python, offre des modèles de données déclaratifs.
- **Sequelize** : Un ORM basé sur des promesses pour Node.js, supportant de multiples bases de données.
- **Doctrine** : Populaire pour les applications PHP, il offre une abstraction de la base de données.
- **Hibernate** : ORM pour Java, il est mature et largement utilisé dans les applications d'entreprise.

## Conclusion

**Django ORM** a été notre choix pour son intégration sans heurt avec le reste de notre stack technologique et sa facilité d'utilisation avec Python.

---

# Partie 4 : Framework web côté client

## Introduction

Le framework côté client détermine la réactivité et l'interface utilisateur de l'application web. Pour notre projet, cela influencera l'expérience utilisateur lors de la gestion des inscriptions.

## Présentation des Solutions

Les frameworks considérés sont : pur **HTML/CSS/JS** (peut-être avec **TailWind**), **Vue.js**, **Angular**, et **React.js**.

### Comparaison des Solutions

- **HTML/CSS/JS (TailWind)** : Approche sans framework, offrant une flexibilité totale. TailWind offre des classes utilitaires pour un design rapide.
- **Vue.js** : Framework JavaScript progressif, il est facile à intégrer et léger.
- **Angular** : Framework complet de Google, il offre une grande boîte à outils mais a une courbe d'apprentissage plus raide.
- **React.js** : Librairie de Facebook pour la construction d'interfaces utilisateurs, elle est modulaire et performante.

### Conclusion

Nous avons décidé de rester avec une approche **pure HTML/CSS/JS**, potentiellement avec TailWind, pour sa simplicité et pour éviter une surcharge de technologies.

---

## Partie 5 : Framework web côté serveur

### Introduction

Le framework côté serveur est responsable de la logique serveur, de la gestion des requêtes et des réponses. Dans notre projet, cela gère les inscriptions et les interactions avec la base de données.

### Présentation des Solutions

Les frameworks considérés sont : **Django**, **Express.js**, **Spring Boot**, et **Laravel**.

### Comparaison des Solutions

- **Django** : Framework Python, il offre un ORM intégré, une interface admin et une architecture MTV (Model-Template-View).
- **Express.js** : Framework minimaliste pour Node.js, il est rapide et non-opinioné.
- **Spring Boot** : Basé sur Java, il est robuste et offre une multitude d'outils pour le développement d'entreprise.
- **Laravel** : Framework PHP, il est élégant, complet, et vient avec son propre ORM (Eloquent).

### Conclusion

**Django** a été choisi pour son ensemble complet d'outils, son ORM intégré et son adéquation avec notre choix de langage de programmation, Python.

---

# Partie 6 : Gestion de versions / Conteneurisation et déploiement

## Introduction

La gestion des versions est cruciale pour suivre les modifications du code, collaborer et déployer. L'utilisation de conteneurs garantit que l'application fonctionne de manière uniforme dans différents environnements.

## Présentation des Solutions

Les outils considérés pour la gestion des versions et la conteneurisation sont : **GitHub**, **GitLab**, et **Docker**.

## Comparaison des Solutions

- **GitHub** : Plateforme populaire pour l'hébergement de code, avec des outils intégrés pour la collaboration et la CI/CD.
- **GitLab** : Semblable à GitHub, mais avec une approche plus intégrée pour la CI/CD.
- **Docker** : Permet de créer des conteneurs pour les applications, garantissant une exécution cohérente.

## Conclusion

Nous avons choisi **GitHub** pour l'intégration avec Jira et le CI/CD sans entrer de coordonnées bancaires. **Docker** est également envisagé pour assurer la conteneurisation.

---

# Conclusion Globale

Lors de l'élaboration de notre projet pour le club d’escalade Senescalade, nous avons dû effectuer des choix technologiques mûrement réfléchis pour garantir la pertinence et l'efficacité de notre solution. Après une étude comparative approfondie des technologies disponibles :

- Pour la **librairie d'accès aux bases de données (ORM)**, nous avons opté pour **Django ORM**, principalement pour son intégration

harmonieuse avec le framework Django et sa compatibilité native avec Python, notre langage de programmation principal.
- Sur le front-end, c'est-à-dire le **framework web côté client**, nous avons décidé de privilégier la simplicité et la flexibilité en optant pour une approche **pure HTML/CSS/JS**, potentiellement renforcée par TailWind. Cette décision vise à éviter l'ajout inutile de complexité tout en offrant une expérience utilisateur optimale.
- En ce qui concerne le **framework web côté serveur**, notre choix s'est porté sur **Django**. Sa nature complète, combinée à un ORM intégré et à une interface d'administration prête à l'emploi, a fait de lui un choix évident, d'autant plus qu'il s'intègre parfaitement avec notre choix de langage, Python.

En somme, les choix technologiques réalisés pour ce projet ont été guidés par la recherche d'une solution robuste, intégrée et efficace, tout en veillant à maintenir une simplicité qui favorise la maintenance et l'évolutivité à long terme.