

B



Dossier Projet - GreenFoot Application / Mark3ts.io VF

Ricardo Martinho



Suivre

DOSSIER PROJET

Ricardo Martinho | 2024



Dossier Projet

M1 Bac +4 Titre Concepteur

Développeur D'Applications Niveau 6

Ricardo Martinho da Cruz



Index

Prologue	4
Résumé	5
Liste Compétences Professionnelles	6
Part I GreenFoot	
Concept	7
Besoins et Exigences Fonctionnelles	8
Planification et Gestion du Projet	9
Git Versioning, Workflow et Repository	10
Architecture Système	12
Stack & Environnement Technologique	13
UML & Merise - Modélisation Base de données	14
Maquettage UX/UI	16
Composants d'Accès aux Données SQL	17
Le script de création de la base de données	18
Sécurité Auth	19
Réalisations Personnelles I	22
Réalisations Personnelles II	26
Développer les composants d'appels API - Promesse Fetch	28
Injections SQL, et Validation Données d'Entrée	31
JEU D'ESSAI (Blocage d'un utilisateur par l'admin)	32
Déploiement Continue	34
Docker, conteneurisation et répartition des services	35
DevOps, IaC et Configuration des Proxies	36
Part II Mark3ts	
Concept	37
Besoins et Exigences Fonctionnelles	38
Architecture MVT	38
Environnement Technologique - Stack & Système	40
HATEOAS - Hypermedia Driven Application	40
HTMX vs SPA	41
Maquettage des interfaces utilisateur	42
UML & Merise - Modélisation Base de données MPD	43
Couche de Persistance	43
Du MLD au ORM	43
Composants d'accès au données SQL	45
Requêtes API Htmx - Event Listeners	45
Sécurité - Authentification	46
Réalisation Personnelle III	48
Git Workflow, Tests Automatisés	54
Déploiement & DevOps mark3ts.io	56
IaC et Configuration des Proxies	57
Conclusion	60
Documentation et références utilisées	61
Annexes	62

Liste Compétences Professionnelles

Les deux projets documentés techniquement dans ce dossier suivent la liste des compétences professionnelles suivantes :

1. Développer une application sécurisée

- Installer et configurer son environnement de travail en fonction du projet
- Développer des interfaces utilisateur
- Développer des composants métier
- Contribuer à la gestion d'un projet informatique

2. Concevoir et développer une application sécurisée organisée en couches

- Analyser les besoins et maquetter une application
- Définir l'architecture logicielle d'une application
- Concevoir et mettre en place une base de données relationnelle
- Développer des composants d'accès aux données SQL

3. Préparer le déploiement d'une application sécurisée

- Préparer et exécuter les plans de tests d'une application
- Préparer et documenter le déploiement d'une application
- Contribuer à la mise en production dans une démarche DevOps

GreenFoot

Interface Calculatrice Empreinte Carbone

Concept

Cette interface est une application personnalisée permettant aux utilisateurs de calculer et suivre, sur une base annuelle, leur empreinte carbone en mesurant leurs émissions de CO₂ à l'aide de formulaires et de visualisations graphiques. L'application intègre également un réseau social et une page dédiée aux dons vers une cause écologique. L'administrateur a accès à une plateforme de gestion (back office) où il peut surveiller les utilisateurs, les types d'émissions, les dons et les publications du réseau social.





Besoins et Exigences Fonctionnelles

MVP: minimum viable product

En tant que visiteur:

- Accéder à la page d'accueil
- Créer un compte

En tant que utilisateur:

- Se connecter à son compte (authentification sécurisée).
- Ajouter ses dépenses de carbone ainsi que ses véhicules personnels.
- Surveiller ses dépenses carbone via des graphiques et des analyses quantitatives.
- Modifier ou supprimer son compte.
- Partager ses publications dans un réseau social interne.
- Pouvoir suivre d'autres utilisateurs et *like*er ses publications.
- Faire des dons à une cause écologique.

En tant qu' administrateur:

- Créer ou modifier le type des dépenses de carbone sous forme d'activité (avec la valeur des émissions).
- Modérer les utilisateurs (les bloquer ou supprimer).
- Modérer les publications sur les réseaux.
- Suivre les dons.
- Analyser via des graphiques les comportements écologiques des utilisateurs d'une façon générale.

GREENFOOT BACKLOG						
ID	EN TANT QUE ...	JE VEUX ...	PRIORITY	SPRINT	STATUS	
27	Admin	ajouter / modifier / supprimer un type d'activité	Haute	1	Terminé	
6	Utilisateur	Me connecter à l'aide de mon pseudo ou adresse email et mon mot de passe	Haute	1	Terminé	
7	Utilisateur	réinitialiser mon mot de passe à l'aide de mon adresse mail	Moyenne	1	Terminé	
10	Utilisateur	modifier mon compte	Moyenne	1	Terminé	
1	Visiteur	avoir accès à la page d'accueil	Haute	1	Terminé	
3	Visiteur	créer un compte	Haute	1	Terminé	
9	Utilisateur	enregistrer mes véhicules personnels dans mon compte	Moyenne	2	Terminé	
11	Utilisateur	supprimer mon compte	Moyenne	2	Terminé	
12	Utilisateur	voir sur mon tableau de bord mon score (consommation en CO2) pour l'année en cours	Haute	2	En cours	
16	Utilisateur	chercher d'autres utilisateurs à l'aide de leur pseudo	Basse	2	Terminé	
19	Utilisateur	ajouter une dépense carbone	Haute	2	Terminé	
2	Visiteur	avoir accès à une page de pédagogie sur l'empreinte carbone	Basse	2	Terminé	
16	Utilisateur	suivre ou ajouter en "ami" d'autres utilisateurs et avoir accès à leur tableau de bord	Basse	3	Terminé	
25	Admin	modérer les comptes utilisateurs	Basse	4	Terminé	
29	Admin	modérer les posts des utilisateurs	Basse	4	En cours	
17	Utilisateur	mettre des posts en lien avec l'empreinte carbone	Basse	4	Terminé	
23	Utilisateur	modifier / supprimer mon post	Basse	4	Terminé	
18	Utilisateur	likez les posts des autres utilisateurs	Basse	4	Terminé	
8	Utilisateur	renseigner mes habitudes de consommation lors de l'initialisation de mon compte (chauffage + divers)	Basse	5	Pas commencé	
8	Utilisateur	renseigner mes habitudes de consommation lors de l'initialisation de mon compte (déplacement)	Basse	5	Pas commencé	
13	Utilisateur	voir sur mon tableau de bord mes dernières activités ayant générée du CO2	Moyenne	5	En cours	
14	Utilisateur	voir sur mon tableau de bord un graphique représentant mon évolution de consommation	Moyenne	5	En cours	
4	Utilisateur	voir le montant de la cagnotte en cours	Moyenne	5	Terminé	
20	Utilisateur	visualiser le graphique selon le type de dépense carbone	Basse	5	En cours	
22	Utilisateur	faire une donation	Basse	5	Terminé	



Git Versioning, Workflow et Repository

En pré-développement, l'équipe a choisi d'utiliser **Git** comme système de contrôle de version pour notre projet et de créer une repository sur la plateforme cloud **Github** comme interface de collaboration et de gestion de projet.

Le groupe a adopté la pratique appelée CI ou Intégration Continue comme méthode idéale pour l'intégration du code de chaque développeur au repo commun. Le dépôt GitHub a été configuré avec une **branche principale main (production)** et une **branche de développement dev (pre-production)**, avec certaines spécifications :

- La branche principale *main* n'accepte que les émargements de code venant de la branche *dev* et avec une pull request sécurisée.
- Pour intégrer du code sur la branche *dev*, la branche individuelle ou branche *feature* créée par chaque développeur doit soumettre une pull request sécurisée.
- Chaque pull request et l'intégration de code qui en résulte nécessitent une revue de code (*code review*) approuvée par au moins deux développeurs autres que le créateur de la branche en question.
- Chaque pull request doit avoir tous ses tests unitaires, d'intégration ou end-to-end approuvés, sinon le *merge* du code ne peut pas être effectuée.

The screenshot shows a dark-themed GitHub interface. At the top, there are two approval notifications: one from 'aliciacqt' and another from 'AnaisCav', both marked as approved '2 days ago'. Below these is a merge commit from '096benjaminbenoit' that merged commit 'ae78932' into the 'dev' branch '2 days ago'. It indicates '3 checks passed'. A large callout box at the bottom states 'Pull request successfully merged and closed' and 'You're all set—the fix/all branch can be safely deleted.' Buttons for 'View details' and 'Revert' are also visible.

dynamic user initials in up-right corner (#72)	dev	3 months ago	...
Test, compile and push client and server to production #5: Commit e965335 pushed by aliciacqt		4m 14s	
UI ux/user initials	ui-ux/userInitials	3 months ago	...
test frontend #6: Pull request #72 opened by aliciacqt		35s	
test staging 3	dev	3 months ago	...
Test, compile and push client and server to production #4: Commit 83f380d pushed by AnaisCav		4m 8s	
test staging 2	dev	3 months ago	...
Test, compile and push client and server to production #3: Commit c95fbec5 pushed by AnaisCav		4m 29s	
test staging	dev	3 months ago	...
Test, compile and push client and server to production #2: Commit bcfab1c pushed by AnaisCav		4m 33s	



Par sécurité, chaque développeur devait suivre ces étapes :

1. Localement faire un pull de la branch dev avec:

```
git pull origin dev
```

2. Résoudre les éventuels conflits
3. Fusionner la dernière version de l'environnement de développement distant sur sa branche individuelle:

```
git checkout feature/example
```

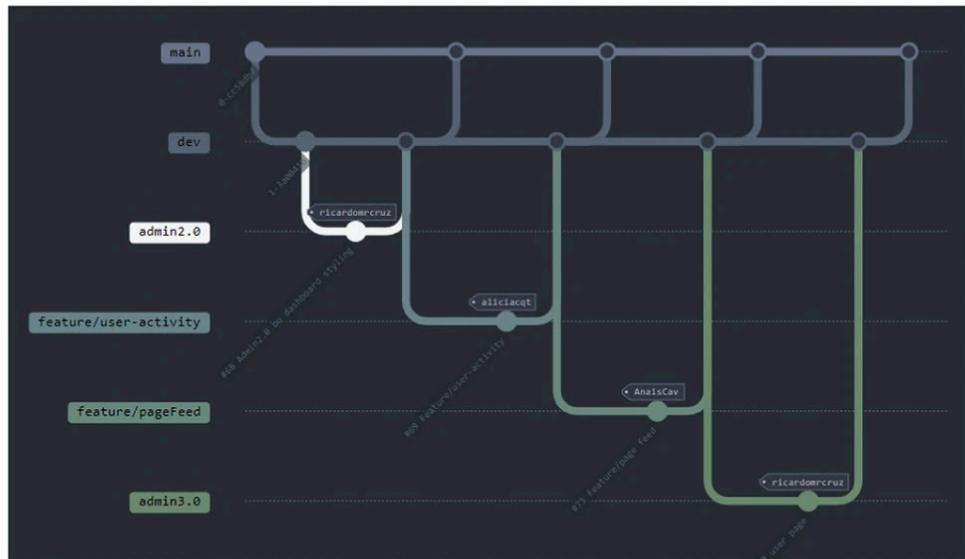
```
git rebase origin dev ou git merge origin dev
```

4. Seulement après ces étapes, faire un push sur le repo remote et éventuellement créer une pull request vers la branche de développement pour sa mise à jour.

```
git push origin feature/example
```

5. Tester la nouvelle feature directement sur l'appli hébergée (dans notre cas sur un VPS) dans un sous-domaine de pré-production ou *staging*. Si aucun conflit n'est détecté, alors faire une pull request de la branche *dev* vers la branche de production *main*.

Ce diagramme représente graphiquement le versioning de notre repo, avec l'exemple continu des branches personnelles qui sont mises à jour avec la dernière version avant la réalisation d'une pull request vers la branche dev. Ce pull request déclenche les tests automatisés sur GitHub Actions.





Architecture Système

Cette application présente une architecture moderne et **découplée en couches**, une approche couramment enseignée et pratiquée dans le développement web moderne. L'architecture de cette application repose sur trois couches distinctes :

Couche Présentation (Frontend):

Interface utilisateur construite avec React

Next.js pour le rendu côté serveur (SSR), optimisant les performances et le SEO

Communication avec l'API via GraphQL et Apollo Client

Couche Métier/API (Backend):

Gestion de la logique métier avec Node.js et Express

Exposition des endpoints via Apollo Server (GraphQL)

Validation et typage fort avec TypeGraphQL

Définition des entités et leurs relations via les décorateurs TypeORM

Gestion des accès aux données via TypeORM (ORM)

Couche Persistance (ORM / Data):

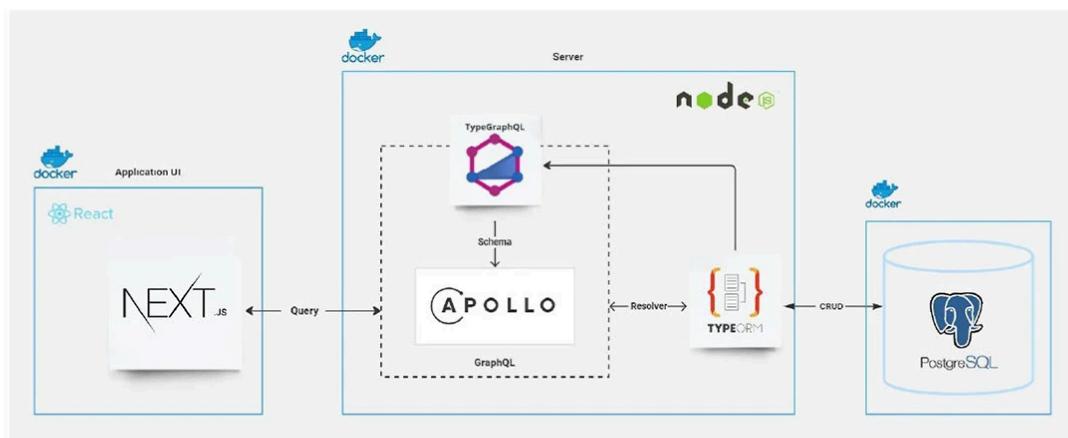
TypeORM pour le mapping objet-relationnel

Définition des entités et leurs relations via décorateurs

Gestion des accès aux données

Base de données relationnelle PostgreSQL

L'application adopte une **architecture conteneurisée** via Docker, permettant une isolation des services et une gestion cohérente des dépendances. Le déploiement est orchestré par Docker Compose sur un VPS Linux Ubuntu, assurant ainsi une mise en production standardisée des différents serveurs et services (frontend, backend, base de données).





Stack & Environnement Technologique

TypeScript. Language superset typé de JavaScript, utilisé dans l'ensemble du projet. Il offre des types statiques optionnels et des fonctionnalités orientées objet, avant d'être transpilé en JavaScript vanilla pour l'exécution, améliorant ainsi la maintenabilité et la robustesse du code.

Backend

Node.js. Environnement d'exécution (*runtime*) JavaScript côté serveur, base de notre backend. Efficace pour les applications en temps réel grâce à son modèle non bloquant et asynchrone.

Apollo Server. Serveur web GraphQL intégré à Express (framework web pour Node Js), créant une API auto-documentée pour notre graphe de données. Particularité de travailler sur **un seul endpoint /graphql** et avec une architecture se basant sur des resolvers et des queries, et non sur des routes comme une API REST traditionnelle.

GraphQL. Langage de requête API permettant des aux clients de demander exactement les données nécessaires.

TypeORM. ORM pour Node.js et techniquement responsable pour la modélisation de notre base de données et ses relations.

TypeGraphQL. Créer des schémas GraphQL avec TypeScript, complémentant TypeORM et Apollo Server. Assure la typage de données et une couche de sécurité.

PostgreSQL. Système de gestion de base de données relationnelle pour stocker les données de l'application.

Frontend

React. Bibliothèque JavaScript pour construire notre interface utilisateur frontend avec des composants réutilisables. Vaste écosystème et docs.

Next.js. Framework React connue par ses features comme le rendu côté serveur (SSR) qui améliore la performance du frontend et le SEO.

Tailwind CSS. Framework CSS utilitaire pour créer rapidement des interfaces utilisateur personnalisées.



Tests

Jest. Framework de test JavaScript pour les tests unitaires et d'intégration du code frontend et backend.

Playwright. Framework pour les tests end2end, permettant de tester l'application sur différents navigateurs.

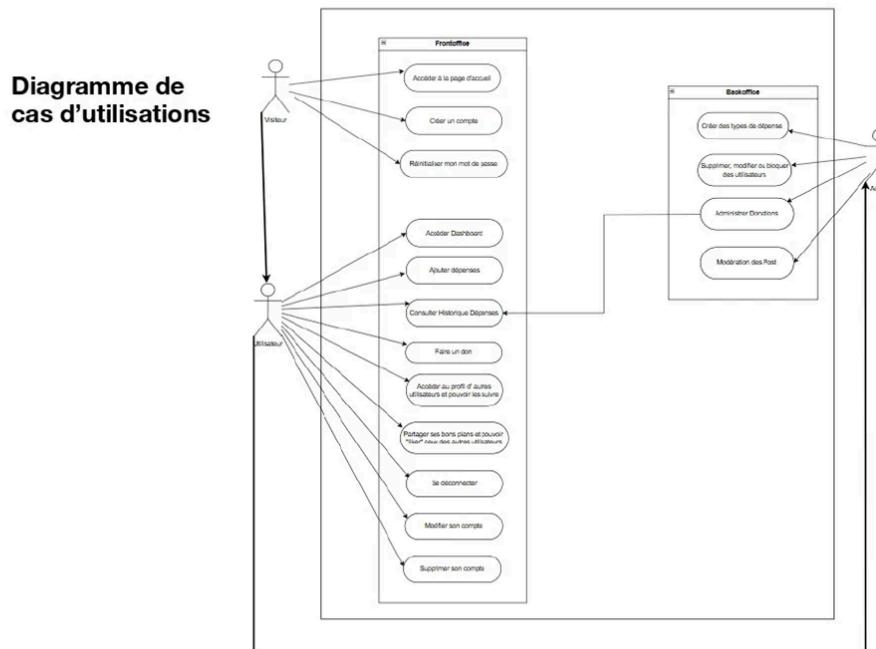
Déploiement

Docker. Plateforme de conteneurisation assurant la cohérence entre les environnements de développement et de production.

Nginx. Serveur web/proxy inverse haute performance, excellent pour la diffusion de contenu statique et load balancing (utilisé en parallèle avec Caddy).

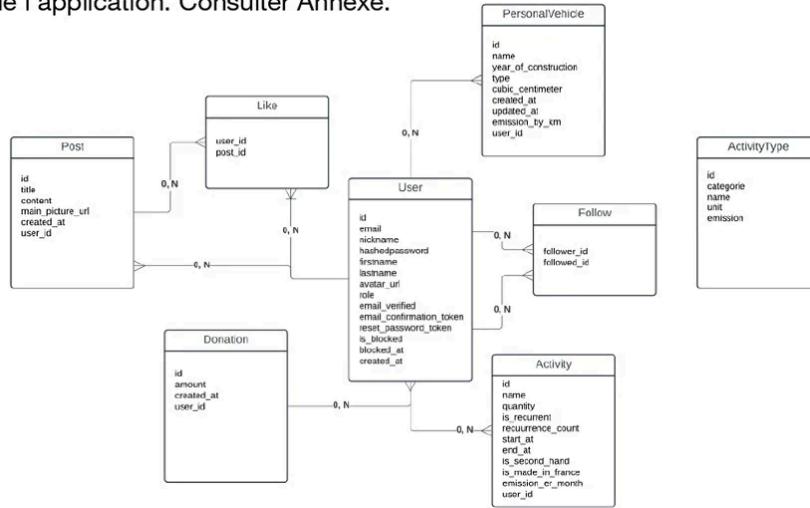
UML & Merise - Modélisation Base de données

L'équipe a collaboré pour modéliser la base de données en utilisant UML (Unified Modeling Language) et le méthode Merise. Ces méthodes offrent une représentation graphique progressive de la structure des données, du concept abstrait à l'implémentation concrète. Cette approche structurée est cruciale pour développer notre ORM, définir les tables et leurs relations, ainsi que pour concevoir une API cohérente. Les diagrammes (**MCD**, **MLD**, **MPD**) illustrent l'évolution du modèle, du conceptuel au physique, guidant efficacement le développement. **Les diagrammes sont disponibles en annexe.**

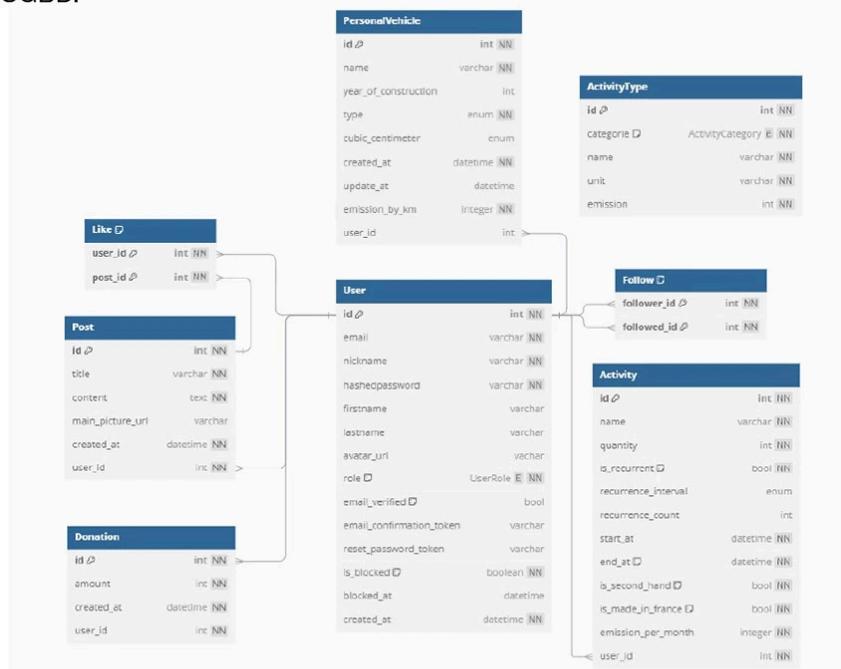




MLD Modèle Logique de Données. Le MLD, élaboré à partir de l'analyse des besoins clients, structure les données de l'application en définissant les tables principales et leurs relations. Ce modèle inclut la gestion des utilisateurs, des publications, des véhicules personnels, et des activités, reflétant les fonctionnalités clés de l'application. Consulter Annexe.



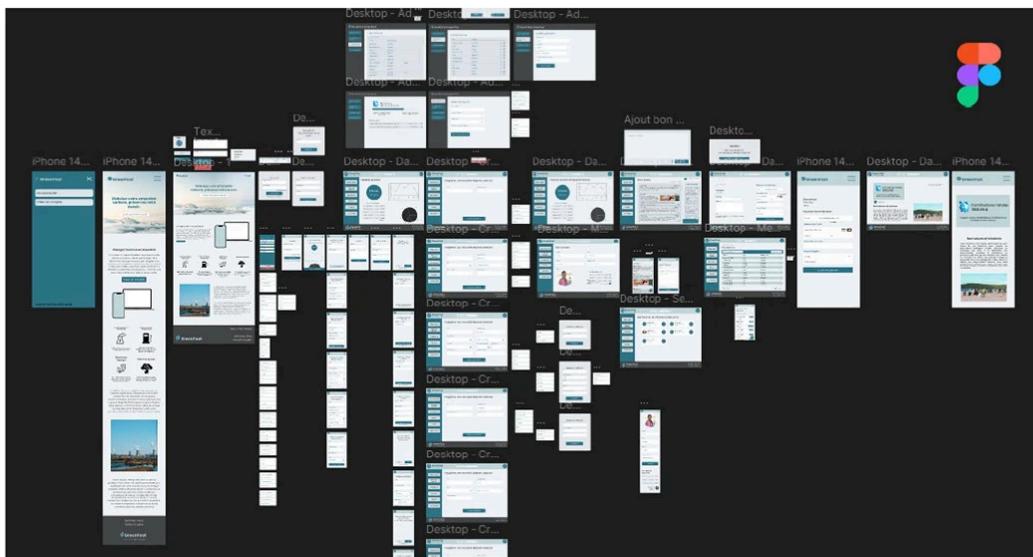
MPD Modèle Physique de Données Le MPD, dérivé du MLD, définit la structure concrète de la base de données (en SQL dans ce cas) avec des tables spécifiques (User, Post, PersonalVehicle, Activity, etc.) et leurs champs détaillés. Il précise les **types de données** (int, varchar, datetime, bool, enum) et les **contraintes** (NN pour NOT NULL). Les **relations entre tables** sont matérialisées par des clés étrangères (ex: user_id dans Post). Cette représentation est directement implantable dans un SGBD.





Maquettage UX/UI

La conception de notre UI, en versions desktop et responsive, ainsi que l'élaboration de la **charte graphique** et la sélection typographique, ont été réalisées collaborativement. Notre palette chromatique, composée de teintes vertes et bleues, a été choisie pour évoquer la sérénité et symboliser un équilibre écologique entre terre et mer, alignée avec notre vision d'un avenir durable. L'utilisation de Figma comme plateforme de prototypage nous a permis de créer itérativement les wireframes et les maquettes fidèles. La maquette et le prototype UX sont également inclus dans les annexes.



Poppins often gets used by startups

Geometric sans serif typefaces have been a popular design tool ever since these actors took to the world's stage. Poppins is a new comer to this long tradition.

Light	Regular	Medium	Semi Bold	Bold	Extra Bold
Aa	Aa	Aa	Aa	Aa	Aa

Type designed by Indian Type Foundry & Jonny Pinhorn

fontpair

#E9F2F6
Greenfoot

#D9E4EA
Greenfoot

#C0D6D8
Greenfoot

#2D7487
Greenfoot

#262828
COOLORS



TypeORM - Composants d'Accès aux Données SQL

Notre modèle de données et schémas sont fusionnés ensemble lors de l'utilisation des bibliothèques **TypeORM** et **Type GraphQL** sous forme de classes. TypeORM est responsable de la définition des entités avec `@Entity()` et dans la même classe, type-graphql assure la définition des types dans un schéma avec `@ObjectType()`. Les classes entités sont définies dans un script individuel, normalement accompagné des objets input type et de leurs relations.

Différents décorateurs comme `@PrimaryGeneratedColumn()` ou `@Column()` définissent les colonnes de notre table et d'autres décorateurs comme `@OneToMany()` établissent la relation avec d'autres tables.

Au niveau du schéma, le décorateur `@ObjectType()` définit la classe comme type GraphQL, ce qui permet d'utiliser des décorateurs comme `@Field()`. La librairie **class-validator** fournit des décorateurs de validation supplémentaires, tels que `IsEmail()` et `IsStrongPassword()`, qui ajoutent une couche de validation plus approfondie en complément de TypeGraphQL.

```

@Entity()
@ObjectType()
class User extends BaseEntity {
    password: string;

    @BeforeInsert()
    async hashPassword() {
        this.hashedPassword = await hash(this.password);
    }

    @Field()
    @PrimaryGeneratedColumn()
    id: number;

    @Field()
    @Column()
    email: string;

    @Field()
    @Column()
    nickname: string;

    @Column()
    hashedPassword: string;

    @Field()
    @Column({ enum: UserRole, default: UserRole.User })
    role: UserRole;

    @CreateDateColumn()
    @Field()
    createdAt: string;

    @OneToMany(() => Activity, (activity) => activity.user)
    @Field(() => [Activity], { nullable: true })
    activities?: Activity[];

    @OneToMany(() => PersonalVehicle,
    (personalVehicle) => personalVehicle.user)
    @Field(() => [PersonalVehicle], { nullable: true })
    personalVehicles?: PersonalVehicle[];
}

...
}

```

```

@InputType()
export class NewUserInput {
    @IsEmail()
    @Field()
    email: string;

    @Length(2, 30)
    @Field()
    nickname: string;

    @Field()
    @IsStrongPassword()
    password: string;
}

@InputType()
export class UpdateUserInput {
    @Field({ nullable: true })
    firstName?: string;

    @Field({ nullable: true })
    lastName?: string;

    @Length(2, 30)
    @Field({ nullable: true })
    nickname?: string;

    @Length(2, 30)
    @Field({ nullable: true })
    @IsEmail()
    email?: string;

    @Field({ nullable: true })
    avatarUrl?: string;
}

export default User;

```



La dernière figure représente un extrait du modèle User, certaines de ses colonnes et le schéma adjacent. On y visualise aussi, comme exemple, sa [relation one-to-many](#) vers la table activités, qui représentent les dépenses carbone individuelles à chaque utilisateur, essentielles au calcul total de son empreinte carbone. Du côté droit, on visualise un exemple des input types, qui assurent la validation des types de données pour les formulaires de notre application.

Le script de création de la base de données

Le script de configuration `db.ts` est stocké dans la racine du dossier `src` du côté backend. Ce fichier de config contient certaines propriétés comme les variables d'environnement, et les entités importées et définies dans un objet `DataSource`. La propriété `synchronize: true` assure que la base de données est automatiquement créée si elle n'existe pas. Selon la doc, cette propriété devrait être depuis désactivée en production pour éviter des pertes de données en prod. La bdd est ensuite initialisée dans le script de lancement du serveur Apollo avec une fonction `initialize()`. Un système de migration était configuré et mis en place pour éviter des conflits en production.

```
//db.ts
import { DataSource } from "typeorm";
import env from "./env";
import User from "./entities/User";
import ActivityType from "./entities/ActivityType";
import Activity from "./entities/Activity";
import { Follow } from "./entities/Follow";
import PersonalVehicle from "./entities/PersonalVehicle";
import Post from "./entities/Post";
import Like from "./entities/Like";
import Report from "./entities/Report";

const db = new DataSource({
  type: "postgres",
  host: env.DB_HOST,
  port: env.DB_PORT,
  username: env.DB_USER,
  password: env.DB_PASS,
  database: env.DB_NAME,
  entities: [User, ActivityType, Activity,
    Follow, PersonalVehicle, Post, Like, Report],
  synchronize: true, ▲
  logging: env.NODE_ENV === "test",
});

export default db;
```



```
//index.ts
const app = express();
const httpServer = http.createServer(app);
const { SERVER_PORT: port } = env;

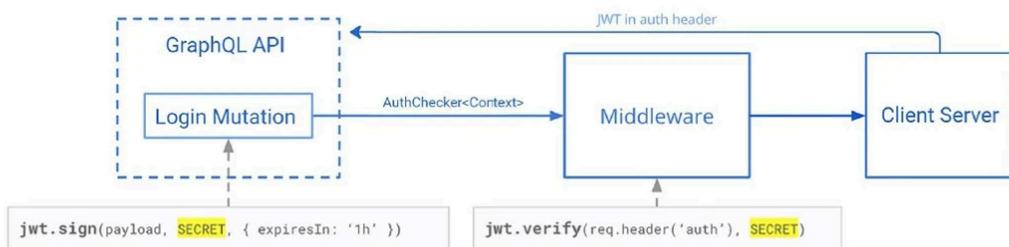
schemasBuilt.then(async (schema) => {
  await db.initialize();
  const server = new ApolloServer<Context>({
    schema,
    plugins: [ApolloServerPluginDrainHttpServer({ httpServer })],
  });
  await server.start();
  app.use(
    "/",
    cors<cors.CorsRequest>({
      credentials: true,
      origin: env.CORS_ALLOWED_ORIGINS.split(","),
    }),
    express.json(),
    expressMiddleware(server, {
      context: async ({ req, res }) => ({ req, res }),
    })
  );
  await new Promise<void>((resolve) => httpServer.listen({ port },
  resolve));
  console.log(`⚡ Server ready at http://localhost:${port}/`);
});
```



Sécurité Auth

Cette application utilise un système d'authentification sécurisé basé sur des [JSON Web Tokens ou JWT](#), et géré par le middleware d'Apollo Server. Une clé privée et unique `JWT_PRIVATE_KEY` est stockée dans le fichier `.env` de notre backend essentiel à l'encryption de ce tokens.

Un fichier `auth.ts` contient l'`authChecker`, une fonction qui extrait le JWT soit du header `Authorization`, soit du cookie d'une requête API, puis vérifie le JWT avec la clé privée stockée dans le fichier `.env`. Si la vérification réussit, la fonction récupère l'ID de l'utilisateur à partir du token, puis vérifie le rôle de l'utilisateur courant dans la base de données.



```
//auth.ts

export const authChecker: AuthChecker<Context> = async ({ context }, roles: string[] = []) => {
  const { headers = {} } = context.req ?? {};
  const tokenInCookie = cookie.parse(headers.cookie ?? "").token;
  const tokenInAuthHeaders = headers.authorization?.split(" ")[1];

  const token = tokenInAuthHeaders ?? tokenInCookie;
  if (typeof token !== "string") return false;

  const decoded = (await jwt.verify(token, env.JWT_PRIVATE_KEY)) as any;
  if (!decoded?.userId) return false;

  const currentUser = await User.findOneByOrFail({ id: decoded?.userId });
  if (currentUser === null) return false;

  context.currentUser = currentUser;

  return roles.length === 0 || roles.includes(currentUser.role);
};
```



Cette fonction contient un élément important dans la sécurisation des opérations de notre application GraphQL, appelée [Context](#). Ce context assure la vérification de types des données à chaque requête HTTP, mais aussi de l'état d'authentification de l'utilisateur courant. Cette interface définit une dépendance qui est par la suite injectée dans les resolvers pour la vérification de la présence d'un token valide bien comme le rôle de l'utilisateur au moment d'une requête.

```
//types.ts

export interface Context {
  req: express.Request;
  res: express.Response;
  currentUser?: User;
}
```

Cet objet est ensuite inséré dans le script de démarrage du serveur pour qu'il soit utilisé de manière asynchrone dans les resolvers d'Apollo Server.

```
//db.ts

const server = new ApolloServer<Context>({
  schema,
  plugins: [ApolloServerPluginDrainHttpServer({ httpServer })],
});
```

Pour que l'authentification de l'utilisateur soit vérifiée par [jwt.verify\(\)](#), il est crucial qu'un token d'authentification soit créé avec [jwt.sign\(\)](#) au moment du login et de l'ouverture de la session utilisateur. Ce token est stocké dans un cookie [HTTP-only](#) du navigateur de l'utilisateur, avec une période de validité (30 jours par exemple). Le cookie est configuré comme sécurisé (pour HTTPS seulement) normalement avec l'attribut [SameSite](#) pour prévenir les attaques CSRF.

```
expressMiddleware(server, {
  context: async ({ req, res }) => ({ req, res }),
})
```



```
//UserResolver.ts

@Mutation(() => String)
async login(@Arg("data") data: LoginInput, @Ctx() ctx: Context) {
  let findUser = await User.findOneBy({ email: data.emailOrNickname });
  if (findUser === null) {
    findUser = await User.findOneBy({ nickname: data.emailOrNickname });
    if (findUser === null) throw new GraphQLError("Invalid Credentials");
  }
  const passwordVerified = await verify(
    findUser.hashedPassword,
    data.password
  );
  if (!passwordVerified) throw new GraphQLError("Invalid Credentials");

  if (findUser.isBlocked === true)
    throw new GraphQLError("This account has been suspended.");

  const token = jwt.sign(
    {
      userId: findUser.id,
    },
    env.JWT_PRIVATE_KEY,
    { expiresIn: "30d" }
  );

  ctx.res.cookie("token", token, {
    httpOnly: true,
    maxAge: 30 * 24 * 60 * 60 * 1000,
    secure: env.NODE_ENV === "development",
  });

  return token;
}
```

À partir de ce point, on peut maintenant utiliser le décorateur `@Authorized()` pour les resolvers qui nécessitent une authentification via le token JWT (stocké dans un cookie ou transmis autrement). On peut également spécifier des autorisations de rôle spécifiques, comme l'Admin (pour les fonctions de back office par exemple). De plus, on peut passer le contexte comme argument aux fonctions pour déclencher une erreur si ce token n'est pas présent ou valide lors de la requête du client.

```
//UserResolver.ts

@Authorized([UserRole.Admin])
@Mutation(() => [String])
async toggleBlockUser(
  @Ctx() ctx: Context,
  @Arg("userIds", () => [Int]) ids: number[]
): Promise<string[]> {

  if (!ctx.currentUser) throw new GraphQLError("You must be authenticated");
  ...
}
```

Note: Cette API utilise implicitement des **requêtes préparées** via l'ORM (TypeORM dans ce cas). Les méthodes comme `User.findOneBy({ id })` traduisent les opérations en requêtes SQL paramétrées, ce qui aide à prévenir les injections SQL sans nécessiter de préparation manuelle des requêtes par le développeur.

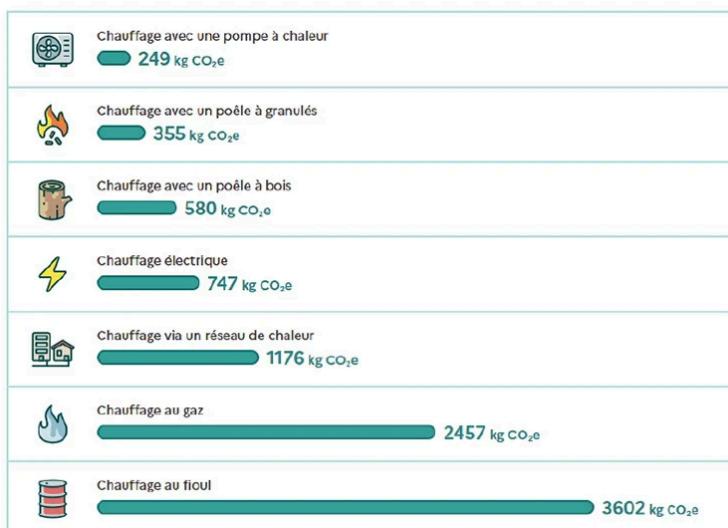


Réalisations Personnelles I

1. Types Des Dépenses CO₂ | ORM & Entité

Sur ce projet, j'étais essentiellement responsable du côté backoffice de l'application, dont l'une des principales fonctionnalités est la capacité **d'ajouter, modifier et supprimer** les types d'émission de carbone qui seront potentiellement utilisés et émis par les activités quotidiennes des Français.

Pour cela, nous avons basé nos émissions sur des données gouvernementales publiées par l'**ADEME** (Agence de la Transition Écologique). Voir annexe. Normalement, ces données sont ajoutées via un script pour pré-peupler la base de données plus facilement. Un autre objectif futur potentiel serait de connecter l'application à l'API de l'ADEME pour un rafraîchissement récurrent et actuel de ces données.



Exemples de données de l'ADEME relatifs aux systèmes de chauffage

Pour ce document, nous allons nous focaliser plutôt sur les entités, les fonctions CRUD et l'accès aux composants du côté frontend relatifs aux types d'émission, appelés dans notre application comme [Activity Types](#).

Pour définir les types d'activités nécessaires, j'ai commencé par créer une classe entité [ActivityType](#) avec toutes les colonnes nécessaires, en respectant les données de l'ADEME. Dans un type d'émission, on aura besoin de certains éléments qui sont parfois énumérés sur différents [Enums](#), comme on le voit dans une des prochaines figures.



Ces éléments incluent la catégorie d'activité, l'unité de mesure, et dans le cas d'un véhicule, les attributs spécifiques tels que l'année de fabrication, le type de carburant, entre autres. Toutes ces propriétés influencent la valeur spécifique d'émission qui sera ultérieurement insérée par l'administrateur dans le backoffice. Cette valeur sera ensuite prise en compte par un resolver responsable du calcul de dépense carbone individuelle d'un l'utilisateur lambda.

```
// entities/ActivityType.ts

export enum Unit {
  Weight = "grammes de CO2",
  PerUnit = "g CO2 par unité",
  Distance = "g CO2 par Km",
  Area = "g CO2 par m² par an",
  Energy = "g CO2 par kWh",
  Volume = "g CO2 par litre",
  Monetary = "g CO2 par € dépensé",
}
```

```
// entities/ActivityType.ts

export enum Category {
  Car = "Voiture",
  Plane = "Avion",
  Bus = "Bus",
  Metro = "Metro Tram",
  Train = "TGV",
  Moto = "Moto",
  Boat = "Bateau",
  Heating = "Chauffage",
  Cooling = "Climatisation",
  Lighting = "Éclairage",
  Appliances = "Appareils ménagers",
  Water = "Eau",
  Food = "Alimentation",
  Waste = "Déchets",
  Clothing = "Vêtements",
  Electronics = "Électronique",
  Services = "Services",
  Leisure = "Loisirs",
  Renewables = "Énergies renouvelables",
  Others = "Autres",
}
```

Sur le script des entités (models), on réalise aussi des classes dédiées au typage des données d'entrée (inputs) avec des décorateurs TypeGraphQL (Annexe 2) comme sécurité de nos opérations CRUD dans notre backend.

```
123  You, last month | 2 authors (Ricardo Martinho and one other)
124  @InputType()
125  export class ActivityTypeInput {
126    @Field()
127    name: string;
```

Activity Type Input - Typage de données d'entrée



2. Types Des Dépenses CO₂ | Backend CRUD

Suite à la construction du *model* `ActivityType`, notre serveur Apollo Server nécessite de réaliser des fonctions CRUD pour que ces fonctions soient accessibles par le client front. Sur Apollo Server, ces fonctions sont réalisées sur des **mutations** et **queries** présentes dans nos classes resolvers. On passe à la construction de la classe `@Resolver()` pour la création des `@Query()` pour les opérations READ.

```
//resolvers/ActivityTypeResolver.ts

@Resolver(ActivityType)
class ActivityTypeResolver {

  //QUERIES
  @Query(() => [ActivityType])
  async getActivitiesTypes() {
    return ActivityType.find();
  }

  @Query(() => [ActivityType])
  async getActivitiesTypesPagination(
    @Arg("offset", () => Int, { nullable: true, defaultValue: 0 })
    offset: number,
    @Arg("limit", () => Int, { nullable: true, defaultValue: 9 })
    limit: number
  ) {
    return ActivityType.find({
      skip: offset,
      take: limit,
    });
  }

  @Query(() => ActivityType)
  async getActivityTypesById(@Arg("id", () => Int) id: number) {
    const activityType = await ActivityType.findOne({
      where: { id },
    });
    if (!activityType) throw new GraphQLError("not found");
    return activityType;
  }

  ...
}

}
```

getActivitiesTypes() pour récupérer tous les types de dépenses

getActivitiesTypesPagination() pour récupérer les types de dépenses par groupes de 9 pour la pagination

getActivityTypesById() pour récupérer un type de dépense par son ID (essentiel pour les slugs du router Next.js)

Les décorateurs sur les fonctions de notre classe sont essentiels dans l'architecture et dans le schéma GraphQL de notre API. Au démarrage du serveur, ce schéma sera maintenant visible sur **Apollo Studio**, une plateforme d'Apollo Server pour la **documentation** et le **test des requêtes** GraphQL.



Les mutations suivent le même schéma en utilisant le décorateur `@Mutation()` pour identifier des opérations de création, mise à jour ou suppression (CUD). Dans cet exemple, on prend la fonction mutation `createActivityType()` pour ajouter un nouveau type de dépense à notre base de données :

```
@Authorized([UserRole.Admin])
@Mutation(() => ActivityType)
async createActivityType(
  @Arg("data", { validate: true }) data: ActivityTypeInput
) {
  const newActivityType = new ActivityType();
  Object.assign(newActivityType, data);

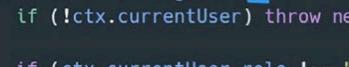
  return await newActivityType.save();
}
```

⚠ Point Sécurité - Role Based Access Control (RBAC system)

Suite à l'implémentation d'un système d'authentification, schéma nécessite des décorateurs spécifiques pour assurer la sécurisation des requêtes API. Les décorateurs `@Authorized()` et l'utilisation de la propriété `Context` sont strictement nécessaires pour la vérification des permissions et la gestion des exceptions. Le décorateur accepte un tableau de rôles (par exemple `[UserRole.Admin]`), tandis que le `@Ctx()` offre une approche plus dynamique, permettant une gestion plus fine des autorisations.

L'exemple suivant illustre que même en cas d'accès non autorisé au backoffice (par piratage, glitch ou bug), cette mutation reste protégée. L'authChecker vérifie la présence d'un token JWT avec le rôle admin avant son exécution, et le middleware empêche ainsi toute action non autorisée.







```
@Authorized([UserRole.Admin])
@Mutation(() => [String])
async toggleBlockUser(
  @Ctx() ctx: Context,
  @Arg("userIds", () => [Int]) ids: number[]
): Promise<string[]> {
  if (!ctx.currentUser) throw new GraphQLError("You must be authenticated");

  if (ctx.currentUser.role !== "admin") {
    throw new GraphQLError("You do not have permission to this operation.");
  } wActivityType.save();
} (...)
```



Réalisations Personnelles II

3. Types Des Dépenses CO₂ | Table et Formulaire d'Ajout

L'administrateur de l'application a la responsabilité de **monitored**, **d'ajouter**, de **modifier** et de **supprimer** les types de dépenses carbone du côté backoffice.

Pour cela, il est nécessaire de créer, côté frontend (dans ce cas, Next.js), une route dédiée à la visualisation de tous les types d'activités dans un tableau, ainsi qu'un formulaire d'ajout d'activité.

Pour cette réalisation, je m'appuie sur l'utilisation de :

- Composants React Next.js en TSX
- Requêtes GraphQL avec l'aide de Codegen
- Tailwind CSS pour le style
- Composants esthétiques de Flowbite pour l'interface utilisateur

Page Dashboard BackOffice

Arborisation React : Router

La création d'une route dans Next.js est simple : il suffit de créer un fichier dans le dossier 'pages'. Next.js est intégré avec un système de fichiers qui détecte en temps réel les modifications dans ce dossier (*node fs module*) et les incorpore automatiquement dans la structure de routage de l'application.



```

import LayoutAdmin from "@/layouts/layout-admin";
import TableActivities from "@/components/backoffice/table-admin-activities";

export default function Activities() {
  return (
    <LayoutAdmin>
      <div className="m-auto w-4/5">
        <h2 className="text-2xl font-semibold mt-5">Liste Type Activités</h2>
        <br />
        <p>
          Ici, vous avez le choix de visualiser les types d'activités que
          l'utilisateur peut enregistrer sur son compte avec leurs émissions de
          CO2.{" "}
        </p>
        <p>Vous pouvez les visualiser, les modifier ou les supprimer.</p>
      </div>
      <TableActivities />
    </LayoutAdmin>
  );
}

```

On voit la structure de l'interface UI du back-office dans les composants React en haut, comme l'importation d'un layout visuel, ainsi que d'autres composants comme la table "activities" qu'on va créer pour les types de dépenses CO₂.

Pour le design de notre table, on utilise un tableau Flowbite. Je trouve personnellement cette librairie efficace et intuitive par la facilité de juste copier-coller des éléments HTML sans besoin d'installer des dépendances et sans polluer le serveur client avec du code boilerplate.

The screenshot shows the Flowbite UI library interface. On the left, there's a sidebar with various components listed: Modal, Navbar, Pagination, Popover, Progress, Rating, Sidebar, Skeleton, Speed Dial, Spinner, Stepper, Tables (which is currently selected), Tabs, Timeline, Toast, Tooltips, and Typography. The main content area has a title "Table head (sortable)" and a subtitle "This example can be used to show the head of the table component with sortable icons." Below this is a dark-themed table component with three rows of data: Apple MacBook Pro 17" (Silver, Laptop, \$2999, Edit), Microsoft Surface Pro (White, Laptop PC, \$1999, Edit), and Magic Mouse 2 (Black, Accessories, \$99, Edit). At the bottom of the table component, there's an "HTML" tab which displays the generated HTML code:

```


| Product name          | Color  | Category    | Price  |      |
|-----------------------|--------|-------------|--------|------|
| Apple MacBook Pro 17" | Silver | Laptop      | \$2999 | Edit |
| Microsoft Surface Pro | White  | Laptop PC   | \$1999 | Edit |
| Magic Mouse 2         | Black  | Accessories | \$99   | Edit |


```

flowbite.com/docs/components/tables | librairie UI



Développer les composants d'appels API - Promesse Fetch

La connexion à l'API du côté client est centrée sur l'utilisation des hooks GraphQL. Les queries et mutations GQL permettent une optimisation des requêtes API plus performantes.

```
//src/graphQL/AdminTableActivities.gql
query GetActivitiesTypesPagination($limit: Int, $offset: Int) {
  getActivitiesTypesPagination(limit: $limit, offset: $offset) {
    category
    emissions
    id
    name
    unit
  }
}
```

On utilise la librairie **Codegen** qui stocke nos requêtes GraphQL dans un système de surveillance (file system). Celui-ci génère un schéma de hooks typés qui sont ensuite utilisés comme promesses dans les composants React. Dans notre composant *TableActivities*, ces hooks retournent une **propriété data** qui stocke les données de la réponse API, un **statut de chargement (loading)** qui affiche un message 'Chargement...' pendant la récupération des données, ainsi qu'un **statut d'erreur** qui gère l'affichage des messages d'erreur en cas de conflit avec l'API.

```
import {
  useDeleteActivityTypeMutation,
  useGetActivitiesTypesPaginationQuery,
} from "@/graphQL/generated/schema";

const PAGE_SIZE = 8;

export default function TableActivities() {
  ...
  const { data, loading, error } = useGetActivitiesTypesPaginationQuery({
    variables: {
      limit: PAGE_SIZE,
      offset: page * PAGE_SIZE,
    },
  });

  useEffect(() => {
    if (data) {
      setNotEndPage(data.getActivitiesTypesPagination.length === PAGE_SIZE);
    }
  }, [data]);

  if (loading) {
    return (
      <p className="mt-3 text-center justify-center align-middle m-auto">
        Chargement...
      </p>
    );
  }

  if (error) {
    return <p>Error: {error.message}</p>;
  }
  const activities = data?.getActivitiesTypesPagination || [];
}
```



Le prochain pas sera d'injecter les données de la réponse API avec un *map* de la variable *activities* dans le tableau html de notre composant. Cette requête *gql* est spécifique parce qu'elle utilise un système de pagination *offset-limit* de 8 en 8 résultats par requête. Ça évite le mode *scroll* qu'était évité dans l'UX du côté backoffice.

NOM	CATÉGORIE	EMISSIONS	UNITÉ	ACTIONS
t-shirt	Vêtements	6430	g CO2 par unité	
paire de chaussures	Vêtements	15000	g CO2 par unité	
jean	Vêtements	25900	g CO2 par unité	
pull	Vêtements	56700	g CO2 par unité	
manteau	Vêtements	101000	g CO2 par unité	
Apple	Electronique	62000	g CO2 par unité	
Huawei	Electronique	71000	g CO2 par unité	
Repas au Restaurant	Services	5500	grammes de CO2	

Formulaire d'Ajout Type de Dépense Carbone

On applique la même logique pour le formulaire d'ajout de types de dépenses. On crée un nouveau script [TSX](#) sur le routeur de Next.js, puis un composant React dans le dossier "components" pour que ce soit modifiable avec un formulaire Flowbite. À l'inverse de notre dernière query, on utilise une [mutation GraphQL](#) pour l'opération *CreateActivityType()* afin d'envoyer les données vers le serveur. On type notre hook avec CodeGen et on insère la promesse dans la fonction qui gère l'envoi du formulaire *handlesubmit()*.

```
mutation CreateActivityType($data: ActivityTypeInput!) {
  createActivityType(data: $data) {
    category
    emissions
    id
    name
    unit
    vehicleAttributes {
      fuelType
      motoEngine
      vehicleDecade
      vehicleType
    }
  }
}
```



```
(...)
export default function newActivityType() {
  const [error, setError] = useState({ message: "", errorInput: "" });
  const router = useRouter();

  const { id } = router.query;

  // create mutation
  const [createActivityType] = useCreateActivityTypeMutation();

  const handleSubmit = async (e: FormEvent<HTMLFormElement>) => {
    setError({ message: "", errorInput: "" });
    e.preventDefault();
    const formData = new FormData(e.target as HTMLFormElement);
    const formJSON: any = Object.fromEntries(formData.entries());

    // validation sanitisation
    const name = formJSON.name.trim();
    if (!/^([A-Za-z\s]+)$/.test(name) || name.length > 50 || name === "") {
      setError({
        message:
          "Le champ nom ne doit pas être vide et doit comporter que des lettres. Max 50 caractères.",
        errorInput: "name",
      });
      return;
    }
  (...)
```

```
(...)
if (
  !/^d*\.\d*$/ .test(formJSON.emissions) ||
  formJSON.emissions.length > 50 ||
  formJSON.emissions === ""
) {
  setError({
    message:
      "Le champ emissions ne doit pas être vide et doit comporter que des chiffres",
    errorInput: "emissions",
  });
  return;
}

const object: ActivityTypeInput = {
  name: formJSON.name as string,
  category: formJSON.category as string,
  emissions: parseFloat(formJSON.emissions as string),
  unit: formJSON.unit as string,
  vehicleAttributes: {
    vehicleType: formJSON.vehicleType as string,
    fuelType: formJSON.fuelType as string,
    vehicleDecade: formJSON.vehicleDecade as string,
  },
};

try {
  await createActivityType({ variables: { data: object } });
  setError({ message: "", errorInput: "" }); // error reset
  router.push('/admin/activities');
} catch (e) {
  setError({
    message:
      "Une erreur est survenue. Assurez-vous de bien remplir tous les champs.",
    errorInput: "general",
  });
  console.error("Error : ", error);
};

return (
  <LayoutAdmin>
    <form className="max-w-3xl mx-auto mt-3 p-5" onSubmit={handleSubmit}>
      <div>
        <h1 className="text-3xl font-bold text-reef">
          Créer Nouveau Type d'Activité
        </h1>
    (...)
```



⚠ Point Sécurité

Injections SQL, et Validation Données d'Entrée

Dans les formulaires, plus spécifiquement dans les champs input, il est strictement essentiel de mettre en place des pratiques comme la **sanitisation**, la **validation** et le **typage des données d'entrée**. Ces pratiques permettent une couche de sécurité contre les injections SQL, les injections de commandes système, et les attaques basées sur la manipulation d'URLs.

The screenshot shows a form with two fields: 'Nom' and 'Emissions CO₂'. Below each field is a red error message. The 'Nom' field has the message: 'Le champ nom ne doit pas être vide et doit comporter que des lettres. Max 50 caractères.' The 'Emissions CO₂' field has the message: 'Le champ emissions ne doit pas être vide et doit comporter que des chiffres.'

Dans notre formulaire, je souligne plusieurs fois **l'importance du typage des données d'entrée**, dans ce cas dans l'objet form qui est traité par notre hook GraphQL, comme la nécessité d'un parsing en numérique pour la valeur d'émissions. La limitation des caractères spéciaux avec des **expressions régulières (regex)** et la **limitation du nombre total de caractères** sont également importantes. Un système d'affichage d'erreurs spécifiques est aussi crucial pour l'accessibilité.

The screenshot shows a 'Créer Nouveau Type d'Activité' (Create New Activity Type) form and a table below it. The form fields include 'Nom' (Repas au Restaurant), 'Catégorie' (Services), 'Emissions CO₂' (5500), and 'Unité Mesure' (grammes de CO₂). The 'Confirmer' button is highlighted with a blue arrow pointing to the table. The table lists activities: Huawei (Électronique, 71000, g), Repas au Restaurant (Services, 5500, gr). Navigation buttons at the bottom are '< Précédente', 'Page 4', and 'Suivante >'.



JEU D'ESSAI (Blocage d'un utilisateur par l'admin)

Les données en entrée : l'action de l'administrateur sur l'application

L'administrateur s'aperçoit d'un comportement inapproprié d'un utilisateur nommé Julien Eco, dont l'adresse e-mail est julien9012@app.com.

L'administrateur décide de prendre une action immédiate et se dirige vers le back-office, plus précisément vers la liste des utilisateurs. Cette liste contient un bouton de blocage qu'il décide d'activer. L'utilisateur Julien est maintenant bloqué.

<input type="checkbox"/>		MarcLeVert	marc601234@app.com
<input type="checkbox"/>		LilaFleurie	lilaflleurie@app.com
<input type="checkbox"/>		admin	admin@app.com
<input type="checkbox"/>		ClaireNature	claire5678@app.com
<input type="checkbox"/>		JulienEco	julien9012@app.com

Étes-vous certain de vouloir bloquer ce(s) élément(s)? Attention, cette action est irréversible.

Oui, bloquer Annuler

04/03/2024

12/03/2024

12/03/2024

12/03/2024

Les données attendues : comment l'application est censée réagir à cette action de l'administrateur

L'application devrait, à partir du moment où le blocage est déclenché, empêcher l'accès de cet utilisateur à notre application. Dans ce cas, si l'utilisateur est connecté, son état devrait passer à `isBlocked = True`.

```

switch (user.isBlocked) {
  case false:
    user.isBlocked = true; ⚡
    user.blocked_at = new Date();
    await user.save();

    return "User blocked and logged out of his account.";

  case true:
    user.isBlocked = false;
    await user.save();

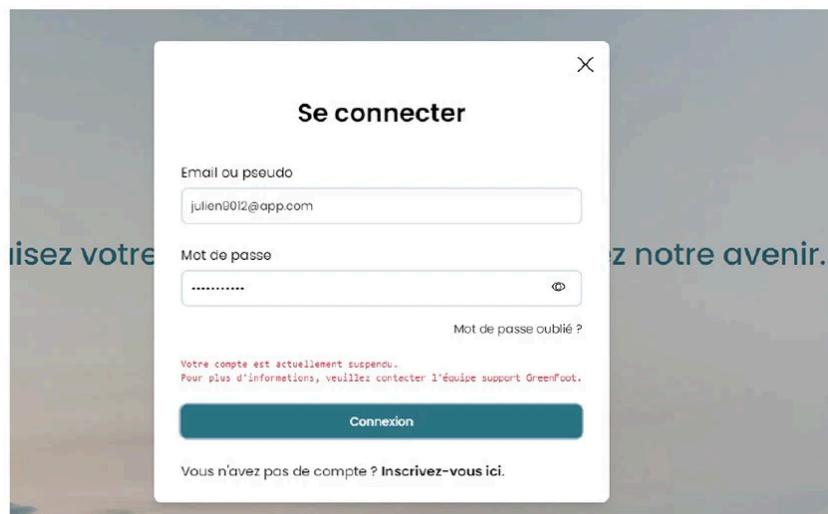
    return `${user.nickname} account has been unblocked.`;
}
  
```



Les données obtenues: comment l'application réagit réellement

L'administrateur vérifie tout suit qui l'état de l'utilisateur a changé, maintenant l'utilisateur est indiqué comme bloqué sur le backoffice. Si l'utilisateur est loggé sur l'application au moment d'une action pertinente elle sera redirigée vers le formulaire de login, elle essayera de se logger et il va se séparer avec une message informatif à cette suspense de compte.

Sur la modal de connection (réponse utilisateur bloqué):



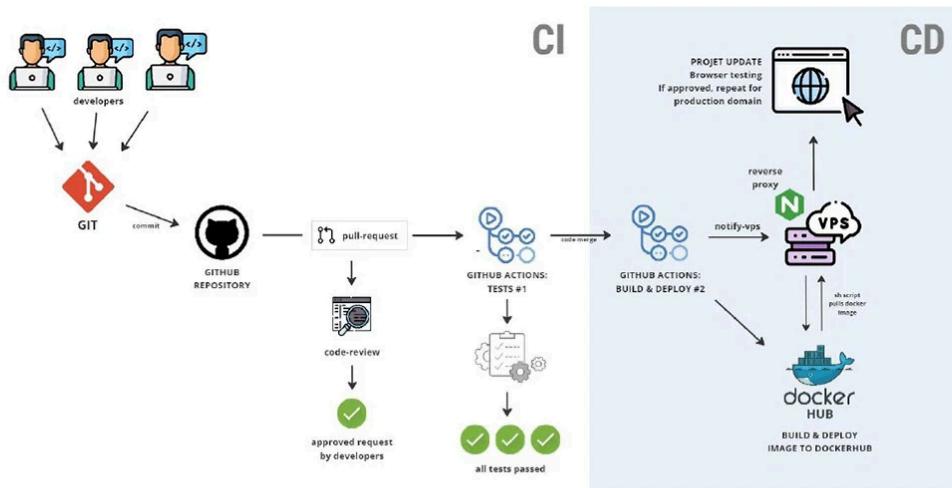
Votre compte est actuellement suspendu.
Pour plus d'informations, veuillez contacter l'équipe support GreenFoot.

Sur backoffice (réponse admin):

<input type="checkbox"/>	 JulienEco julien9012@app.com	! Compte suspendu depuis: 2024-10-07	12/03/2024	User	 
	 JulienEco julien9012@app.com	! Compte suspendu depuis: 2024-10-07			



Déploiement Continue



Chaque développeur de l'équipe *push* son code depuis sa machine vers le dépôt GitHub dans [une branche individuelle et clairement identifiée](#). Une *pull request* est ensuite créée vers une branche de staging préalablement configurée (permissions de *merge*).

Lors de la création de la *pull request*, GitHub Actions déclenche automatiquement les *jobs* liés aux tests unitaires, d'intégration (*jest*) et end-to-end (*playwright*). Concrètement, GitHub Actions exécute notre application dans des machines virtuelles appelées *runners* pour vérifier la fiabilité de nos tests.

Une fois les tests validés et la *pull request merged* avec succès, GitHub Actions suit l'ordre des jobs définis dans nos workflows GitHub (fichiers YAML) et déclenche la construction de l'image de notre projet sur DockerHub. Dans ce cas, elle effectue la construction de l'image pour l'environnement de staging ou pré-production qui sera déployée sur un sous-domaine de test (**dev.1123-xxx-x.wns.wilders.dev**). Le dernier job de notre workflow GitHub notify le VPS pour déclencher un script shell d'arrêter, et faire un *pull -> build* de la nouvelle image Docker disponible sur le repository d'image.

Après avoir testé l'application sur le sous-domaine de staging, nous pouvons répéter le processus pour la branche de production afin de mettre à jour notre site principal. Consulter annexe pour le diagramme de la pipeline CI/CD.



Docker, conteneurisation et répartition des services

La conteneurisation de notre projet avec Docker devient essentielle tant pour le workflow entre les développeurs de l'équipe que pour la répartition des services de notre application. Cet avantage est particulièrement important étant donné que les membres travaillent sur différents systèmes d'exploitation.

La répartition des services, plus précisément liée à différents environnements et fichiers Docker spécifiques, sera cruciale pour le déclenchement des conteneurs Docker pendant l'intégration continue et lors de l'exécution des conteneurs dans le serveur virtuel (VPS).

The screenshot shows two terminal windows side-by-side, each displaying a Docker Compose configuration file.

Left Terminal (Development Environment):

```
docker compose dev.yml
2023-11-wns-bleu-g3 > docker-compose.dev.yml > {} volumes > {} devDB
Bertrand Robert, 5 days ago | 3 authors (Anais and others) docker-compose.yml - The Compose spec
1 < services:
2   db:
3     Click to collapse the range. ros:15-alpine
4       stop_grace_period: 0s
5     environment:
6       POSTGRES_PASSWORD: ${DB_PASS}
7       PGUSER: ${DB_USER}
8     healthcheck:
9       test: ["CMD-SHELL", "pg_isready"]
10      interval: 2s
11      timeout: 5s
12      retries: 10
13     volumes:
14       - devDB:/var/lib/postgresql/data
15     backend:
16       stop_grace_period: 0s
17       env_file: backend/.env.dev
18     environment:
19       DB_HOST: db
20       DB_PASS: ${DB_PASS}
21       DB_USER: ${DB_USER}
22     JWT_PRIVATE_KEY: ${JWT_PRIVATE_KEY}
23     NODE_ENV: ${NODE_ENV}-development
24     SMTP_HOST: ${SMTP_HOST}
25     SMTP_PORT: ${SMTP_PORT}
26     SMTP_USER: ${SMTP_USER}
27     SMTP_PASS: ${SMTP_PASS}
28     FRONTEND_URL: ${FRONTEND_URL}
29     EMAIL_FROM: ${EMAIL_FROM}
30     CORS_ALLOWED_ORIGINS: ${CORS_ALLOWED_ORIGINS}
31     depends_on:
32       db:
33         condition: service_healthy
34     image: greenfoot/greenfoot-hack-dev
35
```

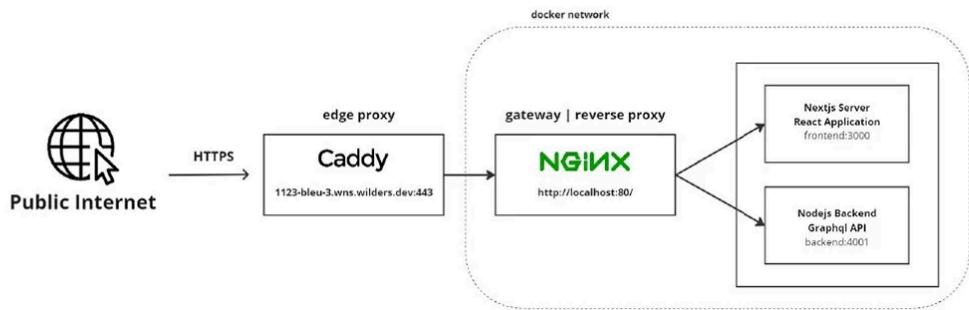
Right Terminal (Production Environment):

```
docker compose production.yml
2023-11-wns-bleu-g3 > docker-compose.production.yml > {} services
Bertrand Robert, 5 days ago | 3 authors (Benjamin Benoit and others) docker-compose.yml - The Compose spec
1 db:
2   Click to collapse the range. image: postgres:15-alpine
3     stop_grace_period: 0s
4   environment:
5     POSTGRES_PASSWORD: ${DB_PASS}
6     PGUSER: ${DB_USER}
7   healthcheck:
8     test: ["CMD-SHELL", "pg_isready"]
9     interval: 2s
10    timeout: 5s
11    retries: 10
12   volumes:
13     - prodDB:/var/lib/postgresql/data
14
15 backend:
16   stop_grace_period: 0s
17   env_file: backend/.env
18   environment:
19     DB_HOST: db
20     DB_PASS: ${DB_PASS}
21     DB_USER: ${DB_USER}
22     JWT_PRIVATE_KEY: ${JWT_PRIVATE_KEY}
23     NODE_ENV: ${NODE_ENV}
24     SMTP_HOST: ${SMTP_HOST}
25     SMTP_PORT: ${SMTP_PORT}
26     SMTP_USER: ${SMTP_USER}
27     SMTP_PASS: ${SMTP_PASS}
28     FRONTEND_URL: ${FRONTEND_URL}
29     EMAIL_FROM: ${EMAIL_FROM}
30     CORS_ALLOWED_ORIGINS: ${CORS_ALLOWED_ORIGINS}
31     depends_on:
32       db:
33         condition: service_healthy
34     image: greenfoot/greenfoot-hack
35
```

```
2023-11-wns-bleu-g3 > .github > workflows > %o deploy-production.yml
AnaisCav, 3 months ago | 2 authors (Anais and one other)
1 name: test, compile and push client and server to production
2
3 on:
4   push:
5     branches: ["main"] A
6   workflow_dispatch:
7
8 jobs:
9   test-client:
10    uses: WildCodeSchool/2023-11-wns-bleu-g3/.github/workflows/tests-front.yml@main
11
12 e2e-tests:
13    uses: WildCodeSchool/2023-11-wns-bleu-g3/.github/workflows/e2e.yml@main
14
15 integration-tests:
16    uses: WildCodeSchool/2023-11-wns-bleu-g3/.github/workflows/integration-tests.yml@main
17
18 build-and-push-server:
19   needs: Anais, 4 months ago · Feature/page mon compte (#33) ...
20   |
21   - integration-tests
22   - e2e-tests
23   uses: WildCodeSchool/2023-11-wns-bleu-g3/.github/workflows/build-backend.yml@main
```



DevOps, IaC et Configuration des Proxies



L'infrastructure de tout le système au moment du déploiement sur le VPS s'appuie sur l'utilisation de **proxies** et de **gateways** entre le réseau Docker, qui contient nos services, et le Web.

L'ensemble de nos services fonctionne dans un réseau ou network Docker, où NGINX agit comme un routeur. En tant que reverse proxy et gateway, il unifie les différents points d'accès de nos services (frontend:3000, backend:4001) vers une unique interface HTTP sur le port 80, facilitant ainsi la gestion du trafic interne.

L'accès public est géré par Caddy, comme edge proxy, qui expose notre application à l'internet en HTTPS, et est responsable de générer automatiquement les certificats SSL/TLS.

```

gateway/nginx.conf
2023-11-wns-bleu-g3 > gateway > N nginx.conf
Benjamin Benoit, 4 months ago | 1 author (Benjamin Benoit)
1  daemon off;
2  events {
3  }
4  http {
5      server {
6          listen 80;
7
8          location /graphql {
9              proxy_pass http://backend:4001;
10         }
11
12         location / {
13             proxy_pass http://frontend:3000;
14         }
15     }
16 }

```

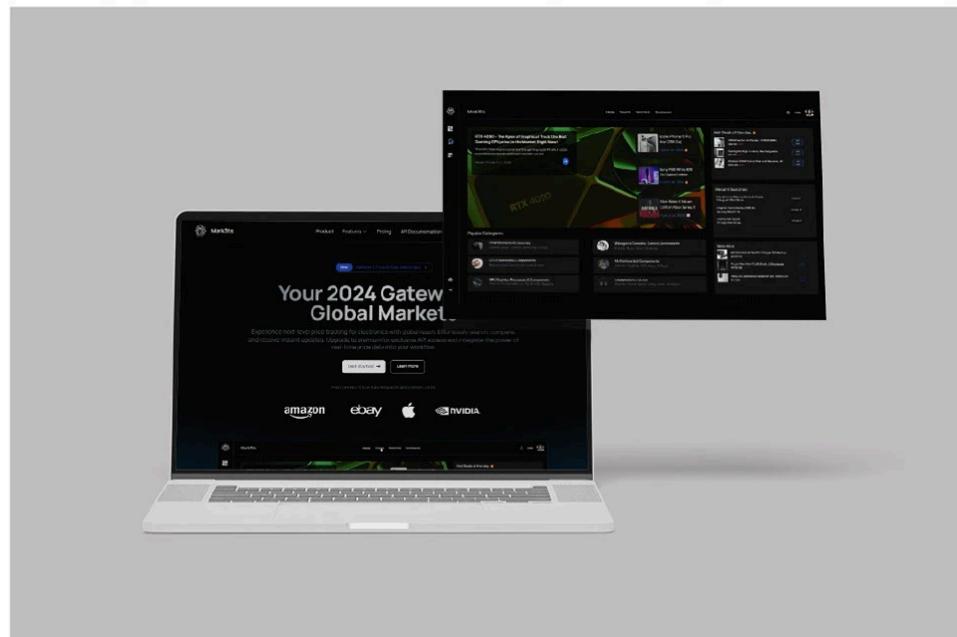


Mark3ts

SAAS Interface Web Crawling Ecommerce

Concept

Cette interface utilisateur avancée ouvre les portes à un vaste univers de données e-commerce, englobant prix, images et descriptions détaillées des produits. La plateforme se décline en trois niveaux d'accès : Starter (gratuit) pour les débutants, Professional (payant) offrant des fonctionnalités étendues, et Premium pour une expérience plus complète. Grâce à un panneau de contrôle intuitif et à une interface utilisateur moderne, les utilisateurs peuvent facilement utiliser un moteur de recherche par scraping, piloter des robots de crawling (*bots*) et explorer des ensembles de données réels du marché e-commerce.





Besoins et Exigences Fonctionnelles

Depuis le début, j'ai défini les fonctionnalités principales de mon application pour qu'elle soit viable (**MVP** Minimum Viable Product) :

- Développer un moteur de recherche de produits e-commerce
 - Permettre aux utilisateurs de rechercher des produits spécifiques
 - Afficher les résultats avec les prix et les liens vers les sites marchands
- Implémenter un tableau de bord utilisateur
 - Afficher un résumé des recherches récentes et des produits suivis
 - Consulter son historique de recherche
- Mettre en place un système de gestion sécurisée des comptes utilisateurs
 - Permettre l'inscription et la connexion authentifiée des utilisateurs
 - Offrir différents plans d'abonnement avec des fonctionnalités spécifiques
 - Protéger les données personnelles des utilisateurs
- Créer une interface utilisateur réactive et moderne
 - Assurer une expérience fluide sur desktop et mobile
 - Implémenter des mises à jour dynamiques sans recharge de page

J'ai également identifié plusieurs fonctionnalités avancées pour de futures itérations, qui permettront d'améliorer l'expérience utilisateur et d'étendre les capacités de la plateforme.

Bonus:

- Fournir une API pour les développeurs
 - Permettre l'accès programmatique aux données de produits et de tendances
 - Gérer les clés API pour les utilisateurs premium
- Implémenter des fonctionnalités de personnalisation
 - Permettre aux utilisateurs de définir des alertes de prix
 - Offrir des recommandations basées sur l'historique de recherche

Architecture MVT

L'architecture MVT (Modèle-Vue-Template) de cette application, basée sur une stack FastAPI Htmx, offre une alternative efficace aux applications à page unique (SPA) traditionnelles. En combinant un moteur de templates côté serveur avec Htmx côté UI, elle permet des interactions dynamiques sans nécessiter un framework JavaScript complexe.



Cette approche, tout en maintenant une séparation claire des responsabilités, facilite le développement d'applications web réactives avec un excellent temps de chargement initial (SEO) et une complexité réduite côté client, tout en permettant une organisation flexible des opérations CRUD.

Modèle (SQLModel + CRUD)

Utilise SQLModel pour la définition des modèles et la validation de données combinant les avantages de SQLAlchemy et Pydantic.

Implémente des opérations CRUD pour la gestion efficace des données crawlées et des informations utilisateurs dans package dédié.

Gère la logique métier, incluant le traitement des données de scraping et la recherche de produits e-commerce dans un package dédié.

Vue (Fonctions de route FastAPI)

Exploite FastAPI en intégrant Jinja2 pour créer des endpoints API entre la couche de persistance et la couche de présentation.

Implémente une authentification sécurisée via JWT pour protéger les routes sensibles avec des injections de dépendances liées aux middlewares sur les routeurs.

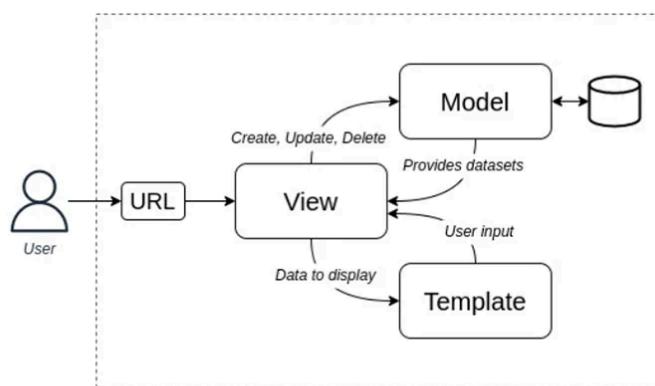
Prépare et distribue les données pour le rendu des templates Jinja ainsi que le traitement des erreurs à afficher dans le DOM.

Template (Jinja2 avec Htmx)

Utilise les fonctionnalités de Jinja2 pour le rendu et le découpage des templates Html.

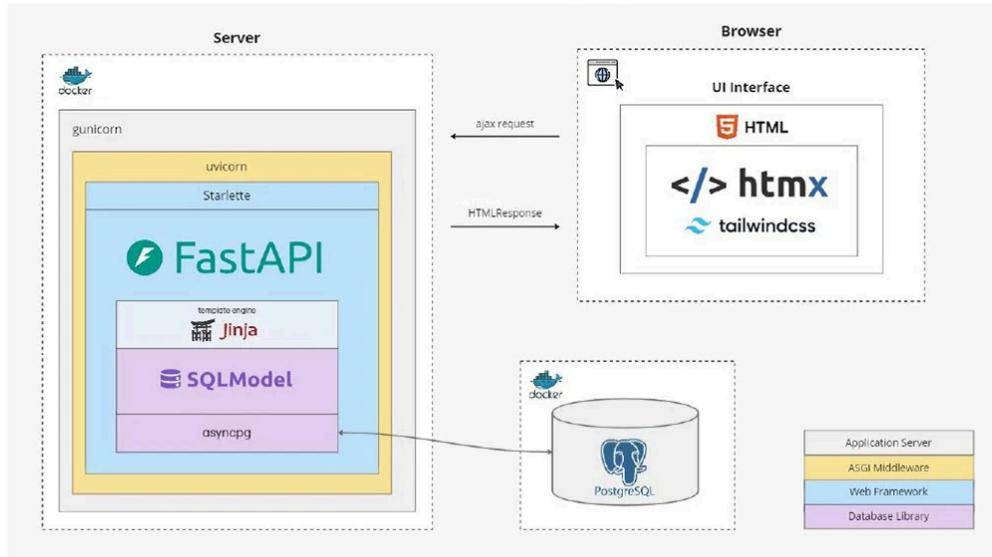
Intègre Htmx comme framework frontend pour effectuer des requêtes HTTP asynchrones vers le serveur et mettre à jour dynamiquement le contenu des pages via des attributs html personnalisés.

Créer une interface utilisateur réactive et performante en html et TailwindCSS, optimisée pour l'architecture monolithique et server-side de l'application.





Environnement Technologique - Stack & Système



HATEOAS - Hypermedia Driven Application

Cette application monolithique est une "*Hypermedia Driven Application*" suivant l'architecture **HATEOAS** (*Hypermedia as the Engine of Application State*). L'interface serveur, implémentée en API RESTful, communique avec la couche de présentation via des templates HTML rendus dynamiquement. Le comportement du user est alors piloté ou '*driven*' par des contrôles html rendus du côté serveur, au contraire de l'approche SPA qui gère d'états de données dans un serveur dédié (React, Angular, etc).

Cette approche s'appuie sur **Htmx**, une solution idéale pour l'éco-conception d'architectures web. Cette librairie vous donne accès à requêtes ajax, transitions CSS, websockets et *server sent events* directement dans le Html, en utilisant des attributs, vous permettant ainsi de construire des interfaces utilisateur modernes avec la simplicité et la puissance de l'hypertexte. Htmx est léger ~14k min.gz'd, sans dépendances, extensible et a permis de réduire la taille des bases de code de 67% par rapport à React (retiré de la doc officielle htmx.org).

L'architecture utilise **FastAPI**, un framework web asynchrone et performant pour créer des APIs avec des annotations Python standard. L'API exploite **SQLModel**, combinant l'ORM SQLAlchemy et Pydantic pour la validation des données. Le frontend intègre htmx avec html et Tailwind CSS. Les templates html sont générés par le moteur Jinja2 intégré à FastAPI.

HTMX vs SPA

Le choix technologique de cette application est justifié par la nature de notre service, qui nécessite une haute performance des réponses au client. L'application repose sur l'exécution de requêtes concurrentes et parallèles, où un frontend basé sur un serveur JavaScript dans un style SPA côté client deviendrait une contrainte pour cette recherche de performance. L'utilisation de Htmx pour cette stack a été décidée en considérant ses avantages significatifs:

Chargement initial plus rapide : htmx ne nécessite pas le téléchargement et l'exécution d'un large framework JavaScript au démarrage.

Mise à jour partielle : htmx permet de mettre à jour seulement des parties spécifiques de la page, sans recharger l'ensemble du contenu ou de l'application. UI découpé sur des *layouts*, *components*, *fragments* et *partials*.

Utilisation des capacités du navigateur : htmx s'appuie sur les fonctionnalités natives du navigateur pour les requêtes ajax HTTP et les mises à jour du DOM.

Réduction de la complexité : Il simplifie le développement en éliminant le besoin de gérer un état complexe côté client.

Performance : Avec moins de JavaScript et un serveur dédié à exécuter et builder du côté client, les interactions peuvent être plus rapides, surtout sur des appareils moins puissants.

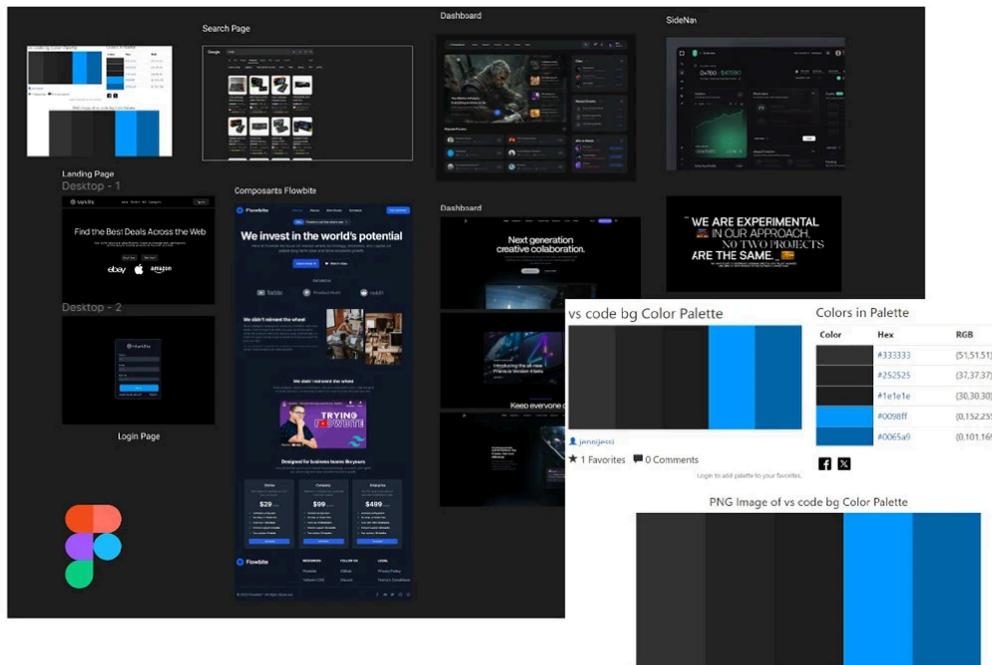
Exemple d'un composant **HTMX** : le bouton *Home* de la *navbar* de la page d'accueil effectue une requête GET au endpoint **/main_dashboard** du router web en demandant un template Ninja qui remplace ou swap l'élément **#main_section**.

```
<div class="space-x-6 ml-10 justify-center">
  <button
    id="home_btn"
    hx-get="/main_dashboard"
    hx-target="#main_section"
    hx-trigger="click"
    hx-swap="innerHTML transition:true"
    hx-select-oob="#home_btn, #search_btn, #watchlist_btn"
    class="underline underline-offset-4"
  >
    Home
  </button>
```

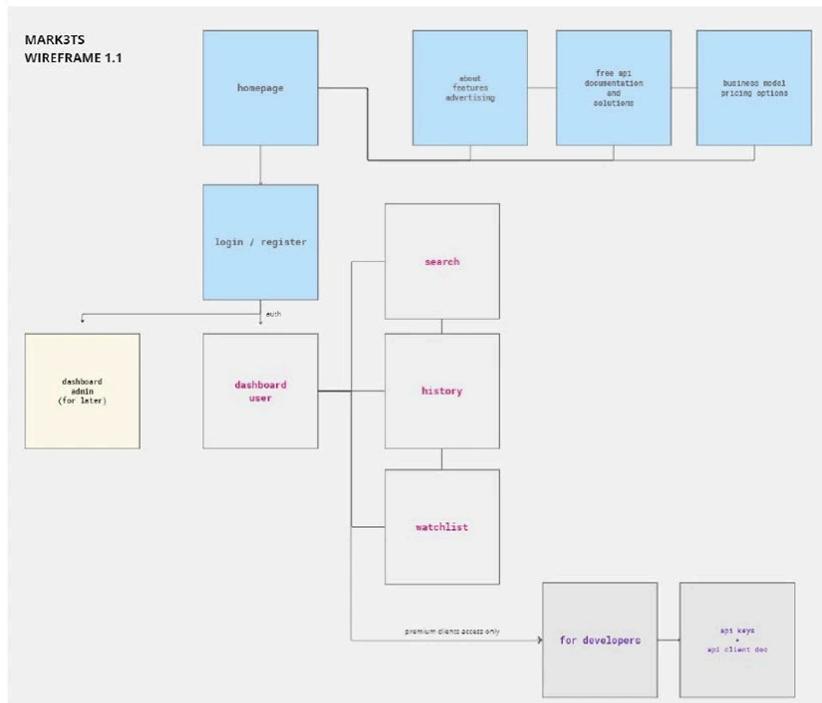


Maquettage des interfaces utilisateur

MoodBoard et Maquette Flowbite



Wireframe (Arborisation UI)





UML Merise - Modélisation Base de données MPD

MPD Modèle Physique de Données Le MPD, dérivé de l'analyse des besoins clients et du MLD, définit la structure concrète de la base de données. Ce modèle détaille les tables (users, subscriptions, crawler_bots, search) avec leurs champs spécifiques, types de données précis (uuid, varchar, boolean) et relations. Il intègre la gestion des utilisateurs, des recherches manuelles et automatisées (via des crawler bots), ainsi qu'un système de souscriptions, probablement lié à un service de paiement comme Stripe, reflétant la nature SaaS de l'application. Cette représentation est directement implantable dans un SGBD. Voir Annexe.



Couche de Persistance

Du MLD au ORM

Suite au MLD, la dernière étape de l'UML avant son intégration au développement, une conscience technologique est nécessaire de la part du développeur concernant les outils à utiliser pour la construction de la logique des données de l'application. On parle notamment de : stack, bibliothèque de validation, décorateurs, type hints, etc.

Notre structure de données s'appuie sur SQLModel, une bibliothèque émergente combinant SQLAlchemy (notre ORM) et Pydantic pour la validation. Cela permet de modéliser nos modèles et schémas de validation dans un même script, simplifiant ainsi notre approche.



The screenshot shows a file tree on the left and code snippets on the right.

File Tree:

```

app
  ├── __pycache__
  ├── .pytest_cache
  ├── api
  ├── core
  └── crud
    ├── __pycache__
    ├── __init__.py
    ├── crawler.py
    ├── search.py
    └── user.py
  └── db
    ├── __pycache__
    ├── __init__.py
    ├── session.py
    └── utils.py
  └── migrations
  └── models
    ├── __pycache__
    ├── __init__.py
    ├── base.py
    └── models.py
  └── scraper

```

Code Snippets:

```

# ----- SEARCH MODELS IN INPUTS -----
# ----- SEARCH MODELS IN INPUTS -----


class SearchBase(SQLModel):
    search_name: str = Field(default=None)
    result: dict = Field(sa_column=Column(JSON))
    is_watchlist: bool = Field(default=False)
    user_id: Optional[UUID] = Field(foreign_key="users.id")

class SearchCreate(SearchBase):
    pass

class SearchUpdate(SearchBase):
    result: Optional[dict] = None
    is_watchlist: Optional[bool] = None

class Search(IdMixin, TimestampMixin, SearchBase, table=True):
    __tablename__ = "search"
    user: User = Relationship(back_populates="searches")

```

Modèle 'Search' (table bdd) sur models.py

models.py sur l'arborescence FastAPI

En soi, Python offre déjà l'utilisation des *type hints* tels que `str`, `bool` ou `dict` pour le typage de données. SQLAlchemy va plus loin en combinant la validation des données (par exemple, `max_length`, `min_length`) avec un typage fort, notamment pour les types complexes comme `datetime` ou `UUID`, ce qui renforce la robustesse du code lors du développement.

Les champs hybrides, tels que `Field()`, permettent à SQLAlchemy d'optimiser la structuration des données. On retrouve principalement l'influence de SQLAlchemy dans la construction des tables et la définition de leurs relations, ce qui facilite la modélisation de schémas de base de données complexes.

```

# User Model
class User(IdMixin, TimestampMixin, UserBase, table=True):
    __tablename__ = "users"
    searches: list["Search"] = Relationship(back_populates="user")

    model_config = ConfigDict(arbitrary_types_allowed=True)

```

Dans cet exemple, on voit qu'un utilisateur peut avoir plusieurs recherches `list["Search"]`. En utilisant l'argument `back_populates="user"`, cela signifie que le modèle `Search` contient un attribut `user` correspondant. Aussi simplement que ça, on établit une relation bidirectionnelle **One-to-Many** entre `Users` et `Searches`.



SQLAlchemy - Composants d'accès au données SQL

Le module CRUD de notre application implémente les opérations de base de données pour chaque entité du modèle, en utilisant les modèles de validation Pydantic et des fonctions SQLAlchemy. Cette couche d'abstraction utilise les sessions asynchrones pour gérer les **transactions ACID** (atomiques ou "tout ou rien") de façon sécurisée, avec une gestion robuste des erreurs.

```
async def create_user(session: AsyncSession, user: UserCreate) -> User:
    db_user = User(**user.model_dump())
    try:
        session.add(db_user)
        await session.commit()
        await session.refresh(db_user)
    return db_user
    except IntegrityError:
        session.rollback()
```

Requêtes API Htmx - Event Listeners

Les requêtes vers l'API sont gérées directement dans le navigateur grâce aux attributs HTMX (comme hx-get, hx-post, etc.) qui agissent comme des écouteurs d'événements sur les éléments HTML. Les différents attributs html communiquent avec des routes FastAPI via des endpoints qui retournent une réponse en html.

Aucun JSON, GraphQL ou XML n'est utilisé dans ce projet ; il s'agit purement de HTML pour la communication API.

```
<!-- register new user form -->
<form id="register_form"
      hx-post="/register/"
      hx-target="#register_form"
      hx-target-error="#error-container"
      hx-swap="innerHTML"
      class="mt-5 max-w-sm mx-auto">

    <h1 class="text-white text-2xl">Register</h1>
    <div class="mt-4">
      <label for="username" class="block mb-2 text-sm font-med
```

Dans cet exemple, les données remplies dans le formulaire d'inscription sont envoyées à notre routeur web. En cas de succès, le routeur accède aux fonctions CRUD pour enregistrer le nouvel utilisateur. En cas de mauvais remplissage du formulaire, il renvoie un fragment html contenant un message d'erreur, qui sera affiché dans l'élément html avec l'id #error-container. Les formulaires restent typés selon les données d'entrée, avec un système de [validation basé sur des expressions régulières \(regex\)](#).



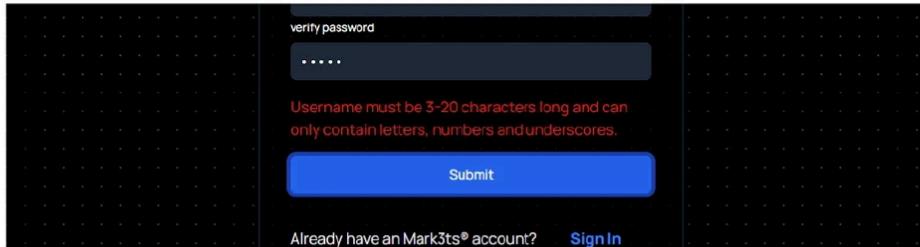
```
@router.post("/register/", response_class=HTMLResponse)
async def register(
    request: Request,
    username: str = Form(...),
    email: str = Form(...),
    password: str = Form(...),
    verify_password: str = Form(...),
    session: AsyncSession = Depends(get_session)):
```

```
if not validate_input(password, PASSWORD_PATTERN):
    return HTMLResponse(
        content=<p class='text-red-600 '>Passwords must be at least 8
            characters long and include both letters and numbers.</p>",
        status_code=422,
    )
```

L'utilisation de la fonction `create_user`, vu dans la page précédente, est importée du module CRUD et mise en œuvre dans les routes :

```
await create_user(session, user)
response = templates.TemplateResponse({"request": request}, name="login.html")
response.headers["HX-Location"] = "/signin"
return response
```

Voici un exemple d'erreur due à un mauvais remplissage du formulaire :



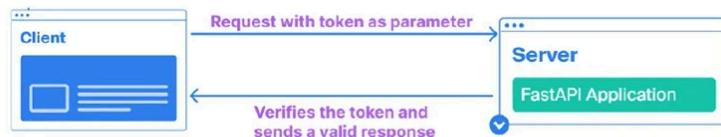
Sécurité - Authentification

Ce serveur présente une approche de sécurité similaire à la première application, avec un système d'authentification basé sur les JSON Web Tokens (JWT).

Dans FastAPI, la pratique courante est d'implémenter un script '`security.py`' dans un package 'core' de notre application, aux côtés d'autres dépendances et fichiers de configuration. Ce package `core` contient généralement nos classes de gestion de l'authentification ainsi que des dépendances liées à la sécurité.

La classe **AuthHandler** est essentielle pour la création et la vérification des tokens JWT dans les requêtes API avant le passage par un middleware. Pour implémenter ce système d'authentification, plusieurs bibliothèques et techniques ont été utilisées :

- 1. Jose : Pour la création et la vérification des tokens JWT.
- 2. Passlib : Pour le hachage sécurisé des mots de passe.
- 3. SQLAlchemy : Pour l'interaction avec la base de données.



Cette classe définit les différentes étapes nécessaires à la construction de notre système de sécurisation, telles que :

Le décodage d'un token JWT

```

def decode_token(self, token):
    try:
        payload = jwt.decode(token, self.SECRET_KEY, algorithms=self.ALGORITHM)
        return payload["sub"]
    except jwt.ExpiredSignatureError:
        raise RequiresLoginException()
    
```

La création d'un token d'accès

```

# jwt token creation
def create_access_token(
    self, subject: Union[str, Any], expires_delta: timedelta = None
) -> str:
    if expires_delta:
        expire = datetime.now(datetime.timezone.utc) + expires_delta
    else:
        expire = datetime.datetime.utcnow() + datetime.timedelta(days=1)
    payload = {
        "iat": datetime.datetime.utcnow(),
        "exp": expire,
        "sub": str(subject),
    }
    return jwt.encode(payload, self.SECRET_KEY, algorithm="HS256")
    
```



La hachage et vérification de mot de passe

```
def get_hash_password(self, plain_password):
    return self.pwd_context.hash(plain_password)

def verify_password(self, plain_password, hash_password):
    return self.pwd_context.verify(plain_password, hash_password)
```

L'authentification de l'utilisateur

```
async def authenticate_user(self, email:str , password:str ):
    async with SessionLocal() as session:
        try:

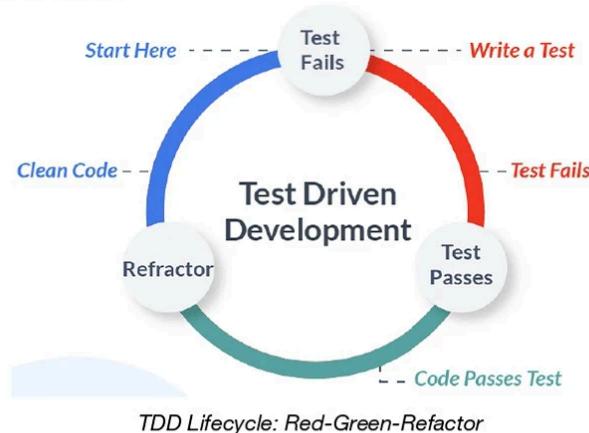
            query = select(User).where(User.email == email)
            result = await session.execute(query)
            user = result.scalar_one_or_none()
```

Ces éléments démontrent la mise en place d'un système d'authentification robuste, utilisant des techniques modernes de sécurité pour protéger les données des utilisateurs et assurer un accès sécurisé à l'application.

Réalisation Personelle III

Préparer et exécuter les plans de tests d'une application

Pour les tests, cette application a suivi l'approche **TDD (Test Driven Development)**. Le développement des fonctions CRUD et des routes a été guidé par les tests, permettant un **clean code** optimisé dès le départ. Bien que cette méthode puisse sembler ralentir le développement initial, elle s'est avérée extrêmement bénéfique sur le long terme. Le TDD a non seulement réduit les bugs, mais a aussi facilité les modifications ultérieures et la maintenance du code.





Pour les **tests d'intégration et unitaires**, nous utilisons la bibliothèque

Pytest. Les tests s'exécutent dans un conteneur Docker simulant l'environnement de l'application sur une porte isolée :8001. La configuration de pytest active un système de découverte des tests (*file system*), cherchant des fichiers Python préfixés par "`test_`". Elle implémente des marqueurs et des fixtures, et exécute les tests dans une session `asyncio` (asynchrone) de FastAPI. Le lancement des tests se fait via des scripts shell : un pour l'environnement local, l'autre pour la pipeline CI/CD.

Certains tests, notamment ceux effectuant des requêtes HTTP vers des sites externes comme Amazon, sont ignorés pendant le CI/CD. Cette approche est due à des restrictions de réseau ou de sécurité des serveurs de GitHub. Pour contourner cette problématique, nous utilisons des **Mocks** de résultats de scraping ou alors nous effectuons des **captures d'écran** (test artifacts) régulières pendant le scraping, tant dans les tests locaux qu'en production, qui pourront être éventuellement surveillés dans une cloud.

```
services:
  postgres-test:
    profiles: ["test"]
    <<: *postgres-base
    env_file: "./app/.env.local"
    environment:
      - POSTGRES_HOST=postgres-test
      - GUNICORN_WORKERS=1
    networks:
      - test

  test-all:
    profiles: ["test"]
    <<: *app-base
    container_name: test-all
    command: >
      bash -c "
        echo 'Starting app server in background...' &&
        (uvicorn api.server:app --host 0.0.0.0 --port 80
        echo 'Waiting for server to start...' &&
        sleep 10 &&
        echo 'Running tests...' &&
        python -m pytest -s -vv
      "
    build:
      context: ./app
      args:
        ENV: test
    env_file: "./app/.env.local"
    environment:
      - POSTGRES_HOST=postgres-test
      - GUNICORN_WORKERS=1
      - PYTHONPATH=/
    volumes:
      - ./app:/app
    ports:
      - "8001:8001"
    depends_on:
      postgres-test:
        condition: service_healthy
    networks:
      - test

  postgres-dev:
```

Configuration docker-compose environment des tests

```
> templates
> test_artifacts
< tests
  < crud
    _init_.py
    test_crawlers.py
    test_searches.py
    test_users.py
  < end2end
    _init_.py
    conftest.py
    test_login.py
    test_new_crawler.py
  < routes
    < v1
      _init_.py
      test_user.py
      _init_.py
      conftest.py
      _init_.py
      conftest.py
    < web\routes
      > __pycache__
```

Arborisation Pytest



Exemple d'un test d'intégrité qui crée et enregistre un utilisateur authentifié, effectue un web crawl vers un site e-commerce, avec données qui sont automatiquement sauvegardées en base de données. Ce test vérifie spécifiquement les fonctions CRUD liées à la sauvegarde d'une recherche, en utilisant des '**mocks**', dans ce cas plus précisément un '**stub**' (donnée simulée) pour le scraping, ce qui permet d'isoler et de valider la logique de persistance indépendamment du fonctionnement du scraper lui-même.

```
# this test tests the crud functions of saving a search while mocking the scraped data
# if this test fails, then the issue is assured to not come from the scraper
async def test_create_search(session: AsyncSession):
    user = UserCreate(email="test@example.com", hashed_password=fakepwd())
    created_user = await create_user(session, user)

    search = SearchCreate(
        search_name="skyrim",
        result=[
            {
                "data": [
                    {
                        "title": "Bethesda The Elder Scrolls V : Skyrim - édition spéciale",
                        "price": "37,88 €",
                        "rating": "4,7 sur 5 étoiles",
                        "img": "https://m.media-amazon.com/images/I/61t01r0rXYL._AC_UL320_.jpg",
                    },
                    {
                        "title": "Good Loot- The Elder Scrolls V Skyrim Puzzle, GDL24676, Multicolor",
                        "price": "21,99 €",
                        "rating": "4,6 sur 5 étoiles",
                        "img": "https://m.media-amazon.com/images/I/71U+2J6IgVL._AC_UL320_.jpg",
                    },
                ],
                user_id=created_user.id,
            }
        ],
    )
    created_search = await save_search(session, search)

    assert created_search.id is not None
    assert created_search.user_id is not None
    assert created_search.user_id == created_user.id
    assert created_search.search_name is not None
    assert created_search.result is not None
    assert created_search.search_name == "skyrim"
    assert created_search.result == search.result
    assert created_search.created_at is not None
    assert created_search.updated_at is not None
```



app/tests/crud/test_searches.py

Ce test [`test_create_search_wscraper`](#) va au-delà d'une simple vérification unitaire en intégrant plusieurs composants nécessaires de l'application. Il valide non seulement les fonctions CRUD, mais aussi la fiabilité du scraper dans des conditions réelles. En effectuant des requêtes HTTP et en crawlant des pages HTML, parfois sécurisées, il éprouve la robustesse du système face aux défis du web réel. L'utilisation du marqueur `@pytest.mark.skip()` permet d'exclure ce test du pipeline CI et des GitHub Actions, évitant ainsi les problèmes liés aux restrictions réseau dans ces environnements (*github test runners*).



```
# this test tests both the crud funtions
# while being dependent on the scraper to generate the data (skipped on the ci for the moment since it dont
# work with github actions)
@ pytest.mark.skipif(os.getenv('RUN_SCRAPER_TESTS') != 'true', reason="TEST LOCALLY - SKIPPED IN CI
ENVIRONMENT")
async def test_create_search_wscraper(session: AsyncSession):
    user = UserCreate(email="test@example.com", hashed_password=fakepwd())
    created_user = await create_user(session, user)

    query = "skyrim deluxe edition ps5"

    scraped_results = await asyncio.wait_for(market_scrape(query), timeout=500)

    search = SearchCreate(
        search_name=query,
        result={"data": scraped_results},
        user_id=created_user.id,
    )

    created_search = await save_search(session, search)

    assert created_search.id is not None
    assert created_search.user.id is not None
    assert created_search.user.id == created_user.id
    assert created_search.search_name is not None
    assert created_search.search_name == "skyrim deluxe edition ps5"
    assert created_search.result is not None
    assert created_search.created_at is not None
    assert created_search.updated_at is not None
```



Les **tests end-to-end** utilisent l'API synchrone de **Playwright** - ironiquement, la même dépendance utilisée pour notre scraper - ce qui évite le build d'un navigateur de test dédié. Cette librairie est intégrée à pytest via l'utilisation de *fixtures* qui permettent un cycle de vie et une page browser (navigateur *Chromium*) individuelle pour chaque test. Ces *fixtures* permettent également l'enregistrement des vidéos et captures d'écran dans les tests end-to-end. Ces tests sont sauvegardés dans un dossier dédié aux **artifacts de test**.

Prenons l'exemple d'un test end-to-end lié à l'enregistrement d'un nouvel utilisateur. Le test visite d'abord la page signin qui contient un formulaire de login et effectue une capture d'écran. Pour accéder au formulaire d'enregistrement d'un nouvel utilisateur, il faut cliquer sur le bouton '*New to Mark3ts? Create an Account*'. Ce bouton déclenche une première requête ajax htmx qui retourne un composant html en réponse : le formulaire d'enregistrement.

```
@pytest.mark.e2e
def test_registration_flow(self, page):
    """Test the complete registration flow including form display and submission"""
    page.goto("http://localhost:8001/signin")
    page.wait_for_load_state("networkidle")

    # Take screenshot of initial state
    page.screenshot(
        path=f"{self.screenshots_dir}/registration_start_{datetime.now().strftime('%Y%m%d_%H%M%S')}.png",
        full_page=True
    )

    # Clicks the registration button to switch modal and verify reg form appears
    page.click("#register_form_btn")
    expect(page.locator("#register_form")).to_be_visible()
```



app/tests/end2end/test_login.py



Ensuite, le test remplit le formulaire d'enregistrement avec certaines données, prend une nouvelle capture d'écran, et soumet le formulaire. Si le formulaire est envoyé sans erreur, le formulaire de login s'affiche automatiquement. Dans le cas contraire, le test échoue et grâce à la capture vidéo du navigateur de test (Chromium) et le logs de pytest, le testeur pourra analyser l'erreur précise survenue dans le formulaire. Ce test permet non seulement de tester la couche de présentation (requêtes Htmx), mais aussi le routeur Fast API ainsi que la couche de persistance (opérations CRUD).

```
# Clicks the registration button to switch modal and verify reg form appears
page.click("#register_form_btn")
expect(page.locator("#register_form")).to_be_visible()

# fills the reg form
page.fill("input[name='username']", "new_account")
page.fill("input[name='email']", "new_account@gmail.com")
page.fill("input[name='password']", "slVtc4rs1")
page.fill("input[name='verify_password']", "slVtc4rs1")

# takes screenshot
page.screenshot(
    path=f"{self.screenshots_dir}/registration_filled_{datetime.now().strftime('%Y%m%d_%H%M%S')}.png",
    full_page=True
)

page.click("#register_new_user")
page.wait_for_timeout(4000)

# if the form is submitted without error the user should be able to see the login form displayed
expect(page.locator("#login_form")).to_be_visible()      You, last week • stable pytest config
```



app/tests/end2end/test_login.py

L'approche **TDD (Test-Driven Development)** ou développement piloté par les tests garantit une couverture approfondie pour chaque composant logiciel, qu'il s'agisse de classes ou de packages, dans la logique métier et la structure du serveur. Le processus TDD suit un cycle caractéristique : il débute par un **échec total** des tests, évolue vers des **échecs partiels** au fur et à mesure du développement du package, et termine avec la **validation de tous les tests** une fois le code factorisé. Cette méthode assure non seulement la qualité du code, mais guide aussi son développement.

Dans notre cas, nous vérifions minutieusement les conditions des tests en local. De plus, nous analysons les captures d'écran générées par Playwright pour les tests end2end et le scraper, ce qui nous permet de valider visuellement le succès des requêtes HTTP externes. Cette double vérification - fonctionnelle via les tests et visuelle via les captures d'écran - renforce la fiabilité du système, particulièrement pour les interactions avec des sources externes.



```
scripts > [run-test-local.sh]
You, 5 hours ago | 1 author (You)
1 pytest_args=$*
2 docker compose -f docker-compose.yml run --rm app-test $pytest_args
3 docker compose -f docker-compose.yml --profile test down --volumes-c
```

script shell de déclenchement des tests

Extrait des logs de test en méthodologie TDD

```
scripts > [run-test-local.sh]
You, 5 hours ago | 1 author (You)
1 pytest_args=$*
2 docker compose -f docker-compose.yml run --rm app-test $pytest_args
3 docker compose -f docker-compose.yml --profile test down --volumes-c
```

Phase de refonte

```
0.17 setup tests/crud/test_users.py::test_get_nonexistent_user
0.16 setup tests/crud/test_searches.py::test_create_search
0.15 setup tests/crud/test_users.py::test_get_user_by_email
0.14 setup tests/crud/test_searches.py::test_update_search_watchlist_status
0.14 setup tests/crud/test_users.py::test_create_user
=====
short test summary info
=====
FAILED tests/crud/test_searches.py::test_create_search - NameError: name 'save_search' is not defined
FAILED tests/crud/test_searches.py::test_create_wscraper - pydantic_core.ValidationException: 1 validation error for SearchCreate
    name*: name 'save_search' is not defined
FAILED tests/crud/test_searches.py::test_update_search_watchlist_status - NameError: name 'save_search' is not defined
=====
time="2024-08-17T04:08:20+02:00" level=warning msg="G:\desktop\mark3ts\docker-compose.yml: 'version' is obsolete"
[+] Running 2/2
  ✓ Container mark3ts-postgres-test-1 Removed
  ✓ Network mark3ts_test Removed
```

Phase de refonte

```
asyncio: mode=auto
collected 23 items

tests/crud/test_crawlers.py::test_create_crawler PASSED
tests/crud/test_crawlers.py::test_create_crawler_w_search PASSED
tests/crud/test_crawlers.py::test_create_crawler_w_pydantic_validation_errors PASSED
tests/crud/test_crawlers.py::test_change_crawler_status PASSED
tests/crud/test_crawlers.py::test_delete_crawler PASSED
tests/crud/test_searches.py::test_create_search PASSED
tests/crud/test_searches.py::test_create_search_wscraper Using User Agent: Mozilla/5.0 (Windows NT
PASSED
tests/crud/test_users.py::test_create_user PASSED
tests/crud/test_users.py::test_create_duplicate_user PASSED
tests/crud/test_users.py::test_get_user PASSED
tests/crud/test_users.py::test_get_nonexistent_user PASSED
tests/crud/test_users.py::test_get_user_by_email PASSED
tests/crud/test_users.py::test_get_nonexistent_user_by_email PASSED
tests/crud/test_users.py::test_update_user PASSED
tests/crud/test_users.py::test_update_nonexistent_user PASSED
tests/crud/test_users.py::test_delete_user PASSED
tests/crud/test_users.py::test_delete_nonexistent_user PASSED
tests/end2end/test_login.py::TestLoginFlow::test_login_form_exists PASSED
tests/end2end/test_login.py::TestLoginFlow::test_registration_flow PASSED
tests/routes/v1/test_user.py::TestUserRouter::test_create_user PASSED
tests/routes/v1/test_user.py::TestUserRouter::test_get_user PASSED
tests/routes/v1/test_user.py::TestUserRouter::test_update_user PASSED
tests/routes/v1/test_user.py::TestUserRouter::test_delete_user PASSED
```



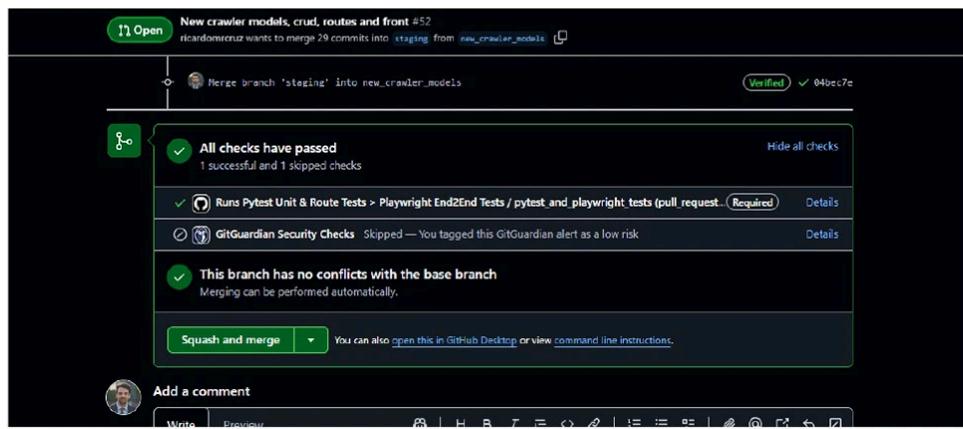
Git Workflow, Tests Automatisés

Similaire à la première application abordée dans ce document, nous utilisons **Git** pour gérer les versions de code de notre projet via des commits entre notre machine locale et le dépôt distant sur Github. Deux branches sont préalablement configurées : **une branche de pré-production (staging)** et **une branche de production (main)**, qui acceptent les fusions de code uniquement via des pull-requests. Lors d'une PR, GitHub Actions déclenche les jobs liés aux [tests unitaires, d'intégration et end-to-end](#).

Ces tests sont exécutés dans des machines virtuelles appelées runners pour vérifier leur fiabilité. Une fois les tests validés et la pull request fusionnée avec succès, GitHub Actions suit l'ordre des jobs et déclenche la [construction de l'image Docker](#) de notre app sur Docker Hub. Dans ce cas, elle construit l'image de l'environnement de staging qui sera hébergée sur un sous-domaine de pré-production. Si ce processus se déroule avec succès, la même logique est répétée pour mettre à jour la branche de production et les conteneurs de notre application sur le VPS du projet.

Exemple d'une pull request de dev vers production (*staging-to-main*) à être exécuté dans un runner github:

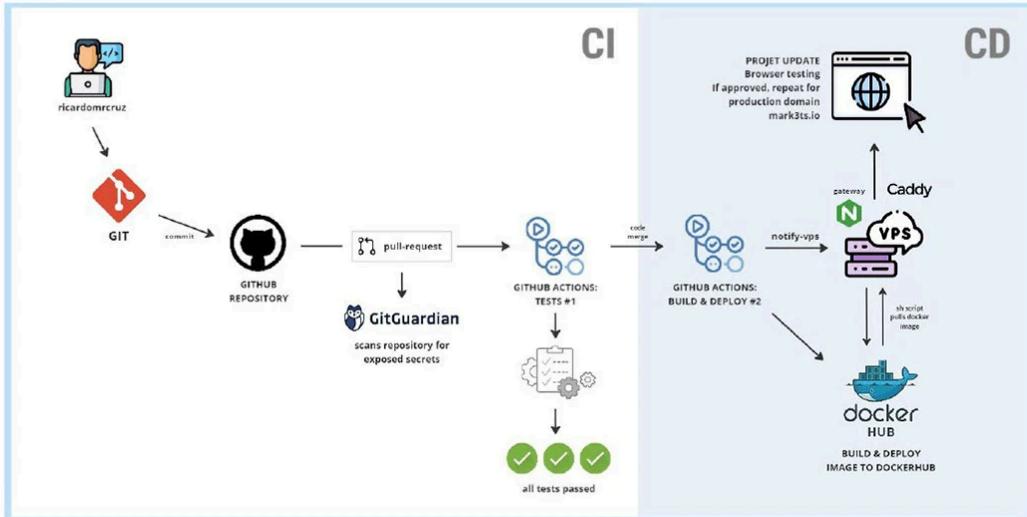
Requires statuses must pass before merging

B

Github Actions Pipeline



Pipeline complet CI/CD (consulter annexe)



Déploiement & DevOps mark3ts.io

Après la construction de l'image de notre application sur le dépôt Docker Hub, préalablement configuré dans notre repository GitHub, GitHub Actions déclenche un job pour **Notifier le VPS**.

Ce système de notification repose sur l'installation du package **webhook** et la configuration d'un endpoint dans le fichier `/etc/webhook.conf`. Lorsque GitHub Actions termine son exécution avec succès, il déclenche cet endpoint qui, à son tour, exécute un script shell de déploiement automatique sur notre serveur.

fichier config webhook

```
GNU nano 6.2                               /etc/webhook.conf *
```

```
[{"id": "upd-prod", "execute-command": "root/apps/mark3ts/prod/mark3ts/scripts/deploy-prod.sh", "command-working-directory": "/root/apps/mark3ts/prod/mark3ts/scripts"}]
```

Sur le VPS, notre projet contient déjà les variables d'environnement configurées, incluant le PORT nécessaire pour communiquer avec notre service de serveur HTTP (dans ce cas, [NGINX](#)), ainsi que les variables d'environnement (ex: `.env.production`).

La notification reçue par le VPS **déclenche Docker** avec un script de déploiement qui :

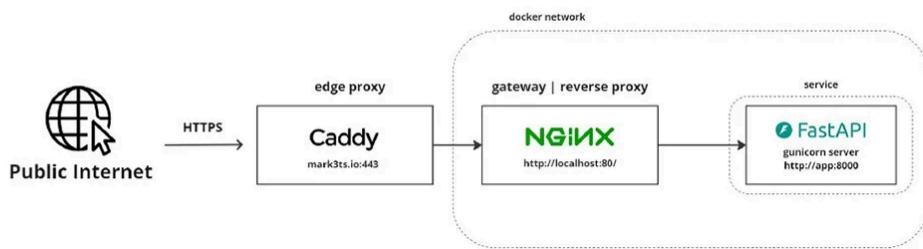
1. Récupère la dernière image depuis Docker Hub
2. Arrête et supprime le conteneur existant (s'il y en a un)
3. Démarré un nouveau conteneur avec la nouvelle image
4. Vérifie le bon fonctionnement de l'application

scripts/deploy-staging.sh (pre-production)

```
run-test-local.sh M deploy-staging.sh X market1.py M
scripts > deploy-staging.sh
You, 2 weeks ago | 1 author (You)
1 #!/bin/sh
2 git fetch origin && git reset --hard origin/main && git clean -f -d && \
3 docker compose -f docker-compose.staging.yml down && \
4 docker compose -f docker-compose.staging.yml pull && \
5 docker compose -f docker-compose.staging.yml --env-file app/.env.staging up -d "$@";
```



IaC et Configuration des Proxies



L'infrastructure du système sur le VPS suit une architecture similaire à l'application Greenfoot, basée sur des **proxies et des services**. Le conteneur de notre application tourne dans le même réseau Docker, où **NGINX** agit comme *reverse proxy* ou *gateway*. Il redirige le trafic du port 80 vers le conteneur de l'application qui écoute sur le port 8000. Cette configuration permet une gestion efficace du routage tout en maintenant une exposition unique via le port standard HTTP.

Le service bénéficie de la sécurisation **HTTPS** mise en place par **Caddy**, un serveur web agissant comme *edge proxy* qui expose notre application publiquement grâce à la configuration DNS préalable de notre domaine mark3ts.io en production. Il gère automatiquement les certificats SSL/TLS pour l'ensemble des applications du serveur.

`etc/caddy/Caddyfile (config)`

```

root@vmit979334:/apps/main ~ + -
GNU nano 6.2                                     /etc/caddy/Caddyfile *
# The Caddyfile is an easy way to configure your Caddy web server.
#
# Unless the file starts with a global options block, the first
# uncommented line is always the address of your site.
#
# To use your own domain name (with automatic HTTPS), first make
# sure your domain's A/AAAA DNS records are properly pointed to
# this machine's public IP, then replace ":80" below with your
# domain name.

mark3ts.io {
    reverse_proxy localhost:8080
}

ops.mark3ts.io {
    reverse_proxy localhost:9000
}

# Refer to the Caddy docs for more information:
# https://caddyserver.com/docs/caddyfile

```

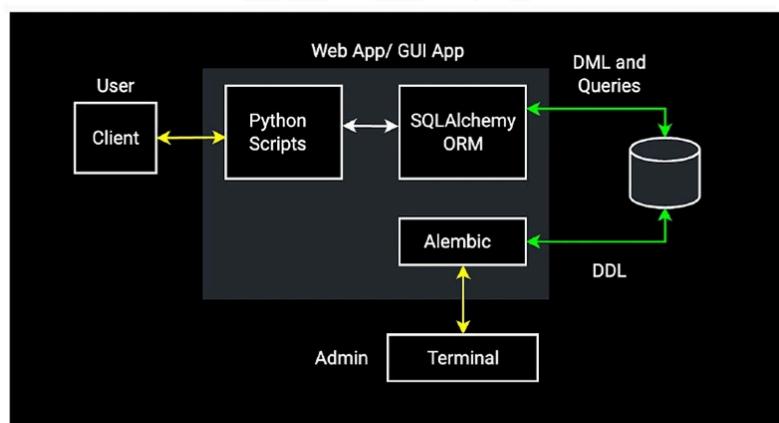


Intégrité des données et migrations automatisées

L'intégrité des données et les [migrations automatisées](#) sont essentielles lors du déploiement sécurisé des nouvelles versions de notre application, particulièrement lors de modifications importantes de notre ORM et base de données. Les migrations permettent la gestion des versions de notre base de données.

Dans l'application mark3ts, un système automatisé a été intégré au démarrage du serveur FastAPI pour prévenir toute perte catastrophique de données en production due aux conflits entre versions de développement et de production. Pour cela, nous utilisons [Alembic](#), un toolkit Python de migrations SQL conçu pour SQLAlchemy.

Implementation d'Alembic sur le système



Arborescence Alembic (migrations folder)

```

migrations
├── __pycache__
│   └── ...
└── versions
    ├── __pycache__
    │   └── ...
    ├── 445d6a748fb_fixed_crawler_missing_relationship.py
    ├── 8870723116b5_add_cascade_delete_to_search.py
    ├── a622974a6819_added_avatar_column_to_user_model.py
    ├── a3230113ec8c_base_migration.py
    ├── e1c76cb93ec2_added_crawler_model_updates_to_search_.py
    ├── env.py
    └── README
        └── script.py.mako
    └── models

```



The screenshot shows the Docker Compose logs for a service named 'mark3ts-app-dev-1'. The log output is as follows:

```
# alembic revision --autogenerate -m "add_cascade_delete_to_search"
INFO [alembic.runtime.migration] Context impl PostgresImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.autogenerate.compare] Detected type change from TIMESTAMP() to TIMESTAMP(timezone=True) on 'crawler.last_run'
INFO [alembic.autogenerate.compare] Detected type change from TIMESTAMP() to TIMESTAMP(timezone=True) on 'crawler.next_run'
INFO [alembic.autogenerate.compare] Detected removed foreign key (crawler_id)(id) on table search
INFO [alembic.autogenerate.compare] Detected added foreign key (crawler_id)(id) on table search
Generating /app/migrations/versions/8870723116b5_add_cascade_delete_to_search.py ... done
#
```

Ex. Commandes Alembic CLI pour la génération des migrations en dev

Alembic est installé et configuré avec une connexion asynchrone à la base de données. Alembic dispose d'outils **CLI** permettant de générer, à tout moment du développement, une nouvelle version identifiée de notre base de données, ainsi que d'effectuer des **downgrades** et **upgrades** entre ces versions. Un système d'automatisation des migrations a été mis en place pour éviter la mise à jour manuelle vers la version la plus récente en production.

Le gestionnaire de contexte *lifespan* orchestre l'initialisation de la base de données au démarrage de l'application FastAPI. Il utilise Alembic pour charger la configuration depuis *alembic.ini* et exécute la **commande upgrade** vers la dernière révision. Cette commande identifie les migrations pendantes et exécute séquentiellement leurs **opérations DDL** ou *data definition language* (création de tables, modifications de schéma) dans des transactions atomiques, assurant ainsi la mise à jour cohérente du schéma de la base de données à chaque déploiement.

```
@asynccontextmanager
async def lifespan(app: FastAPI):
    logging.info("Starting to create database tables and upgrading migrations...")
    try:
        # await create_db_and_tables(engine)
        # no need to create db and tables, migrations will be run and upgraded by alembic
        alembic_cfg = Config("alembic.ini")
        command.upgrade(alembic_cfg, "head") △
        logging.info("Database migrations applied successfully")
        logging.info("Database tables created successfully")
    except Exception as e:
        logging.error(f"Error creating database tables: {e}")
    yield
    logging.info("Application shutting down")
```

Système de gestion du cycle de vie dans le script de démarrage du serveur FastAPI app/api/server.py



Documentation et références utilisées

1. Apollo Studio - Documentation officielle :
<https://www.apollographql.com/docs/studio/>
2. TypeORM - Documentation officielle : <https://typeorm.io/>
3. TypeGraphQL - Documentation officielle :
<https://typegraphql.com/docs/introduction.html>
4. Mozilla Developer Network (MDN) - Documentation TypeScript :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/TypeScript
5. Next.js - Documentation officielle : <https://nextjs.org/docs>
6. React - Documentation officielle :
<https://fr.reactjs.org/docs/getting-started.html>
7. Flowbite Components - Documentation :
<https://flowbite.com/docs/components/>
8. FastAPI - Documentation officielle : <https://fastapi.tiangolo.com/>
9. Pydantic - Documentation officielle :
<https://pydantic-docs.helpmanual.io/>
10. HTMX - Documentation officielle : <https://htmx.org/docs/>
11. Tailwind CSS - Documentation officielle : <https://tailwindcss.com/docs>

Projets GitHub de référence

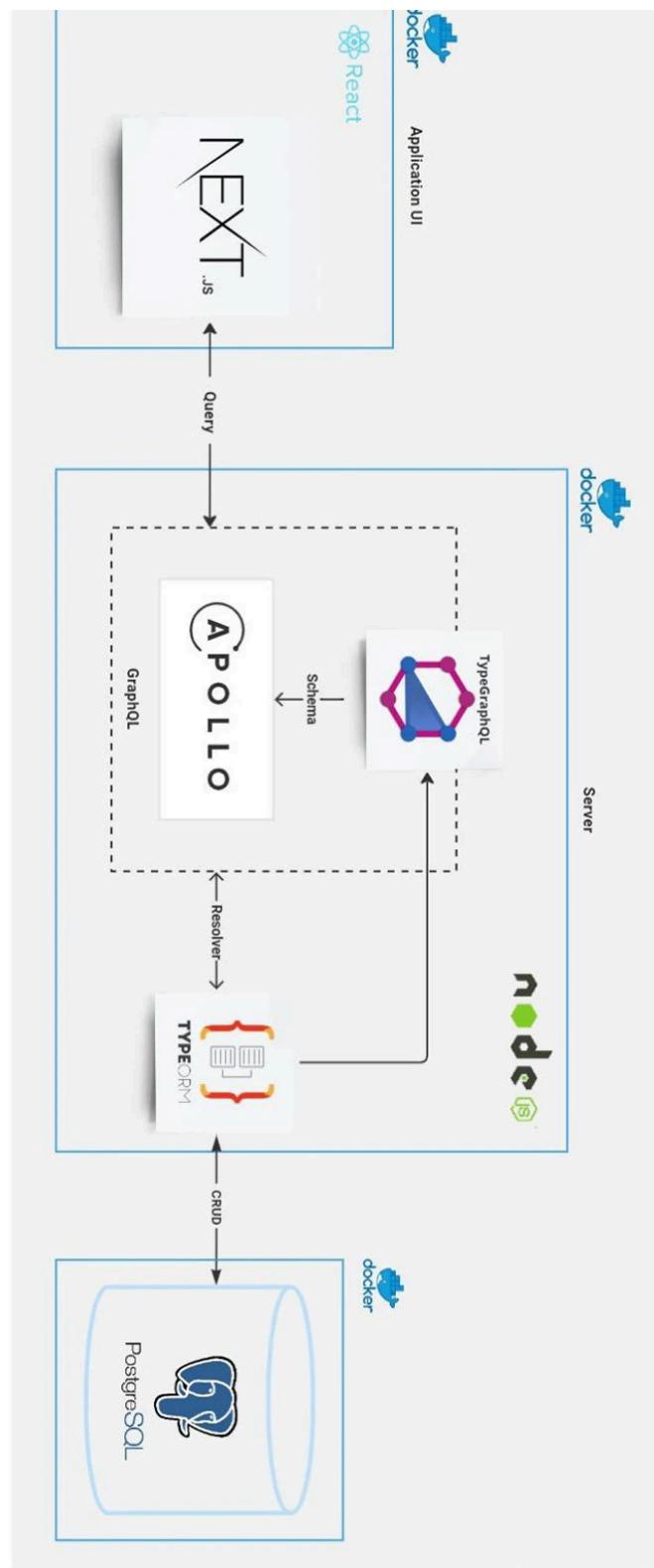
1. The Good Corner (ComicScrip) Pierre Genthon Instructor WCS :
<https://github.com/ComicScrip/the-good-corner-nov23>
2. Data Scraping API (ETretyakov) :
<https://github.com/ETretyakov/data-scraping-api>
3. FastAPI Async SQLAlchemy Demo (ThomasAitken) :
<https://github.com/ThomasAitken/demo-fastapi-async-sqlalchemy>
4. FastAPI Snippets (LTMullineux) :
<https://github.com/LTMullineux/fastapi-snippets>
5. Noted - FastAPI (jod35) : <https://github.com/jod35/Noted-FastAPI>
6. HTMX FastAPI Login (eddyizm) :
https://github.com/eddyizm/HTMX_FastAPI_Login



Annexes

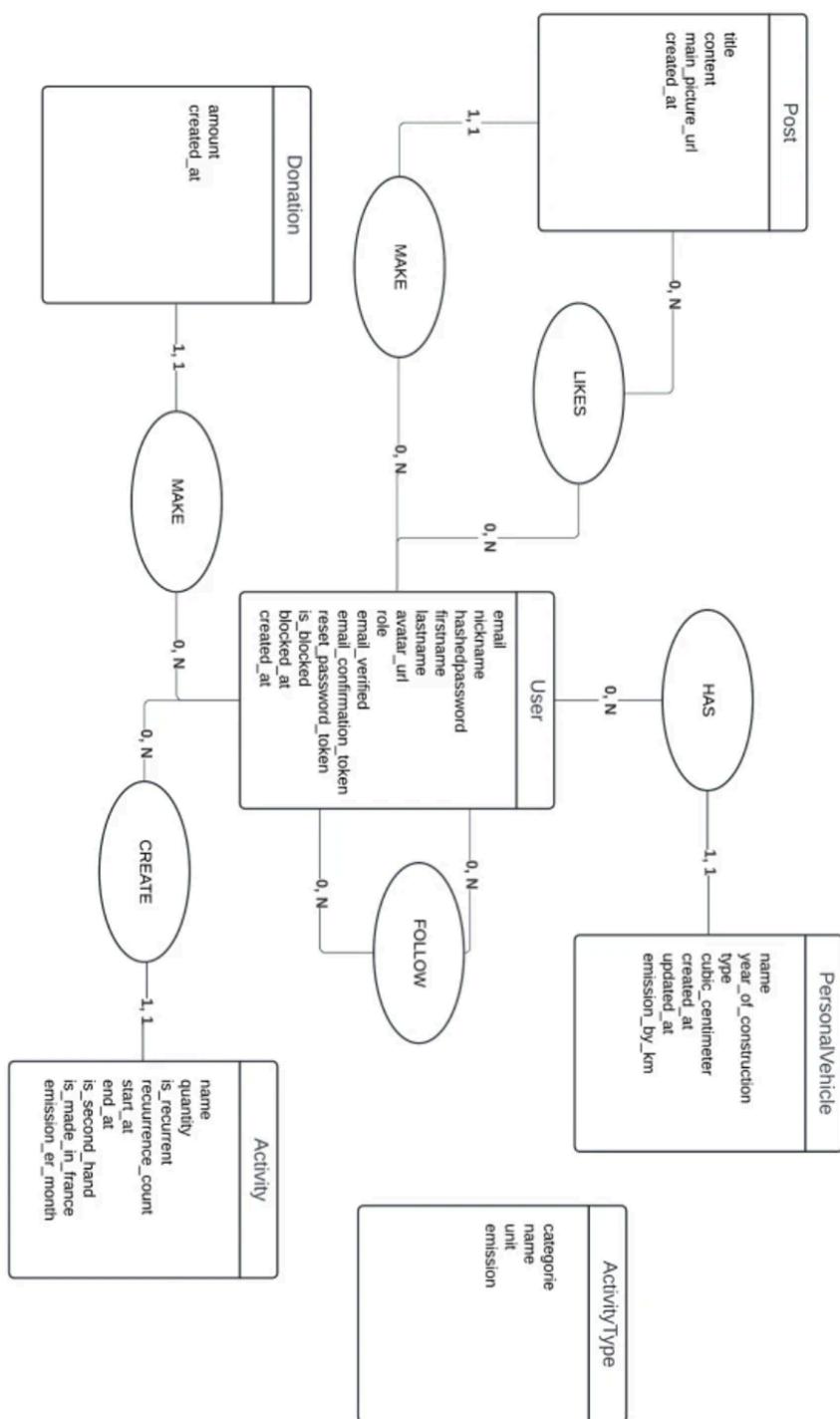
GreenFoot

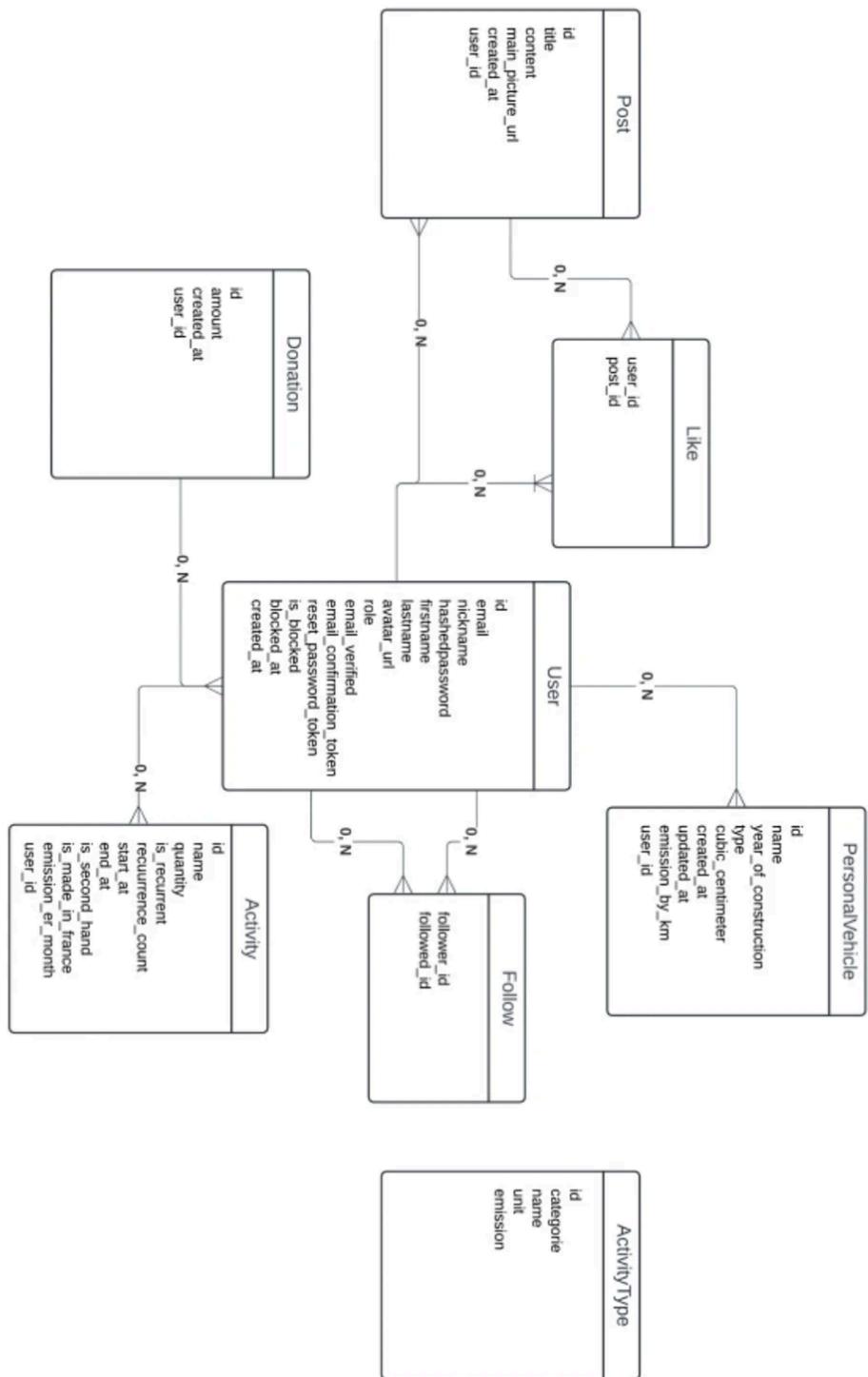
Archi
Système





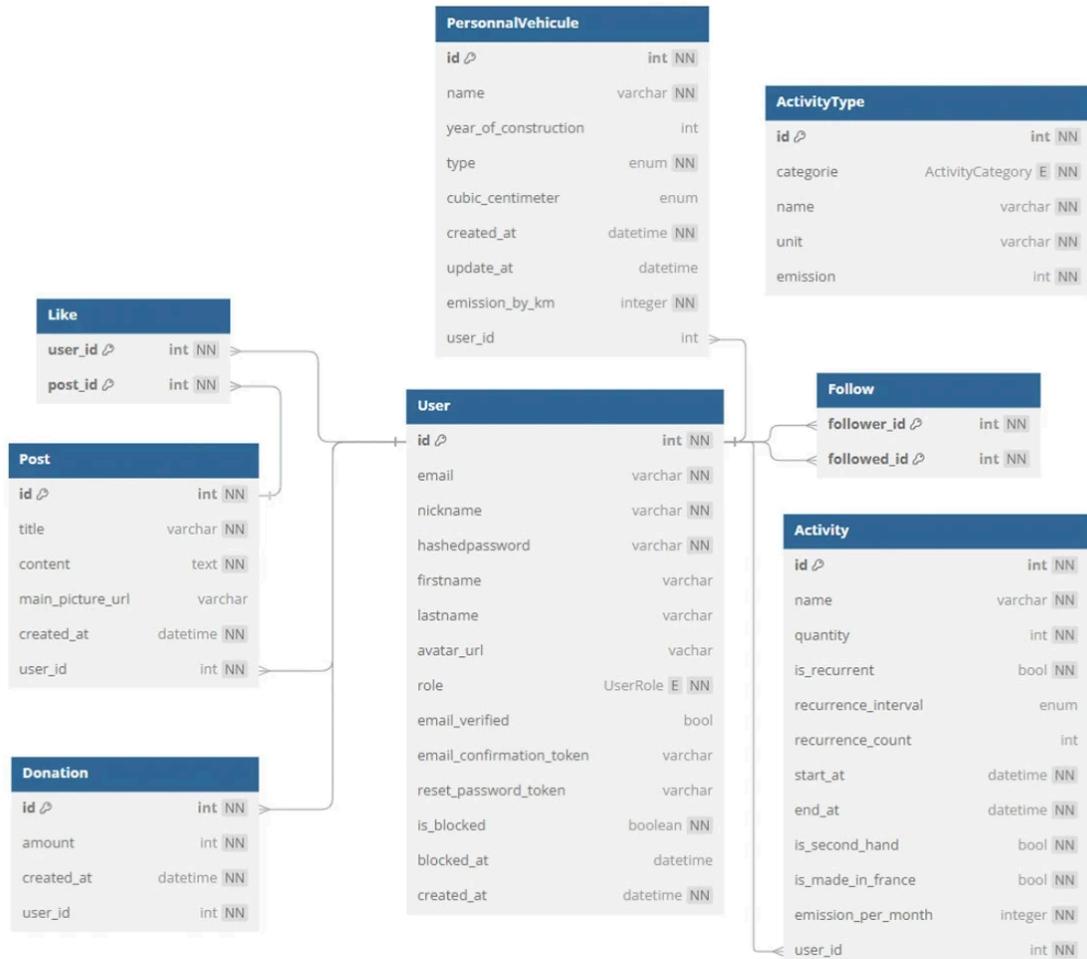
GreenFoot MCD (modélisation de la BDD)







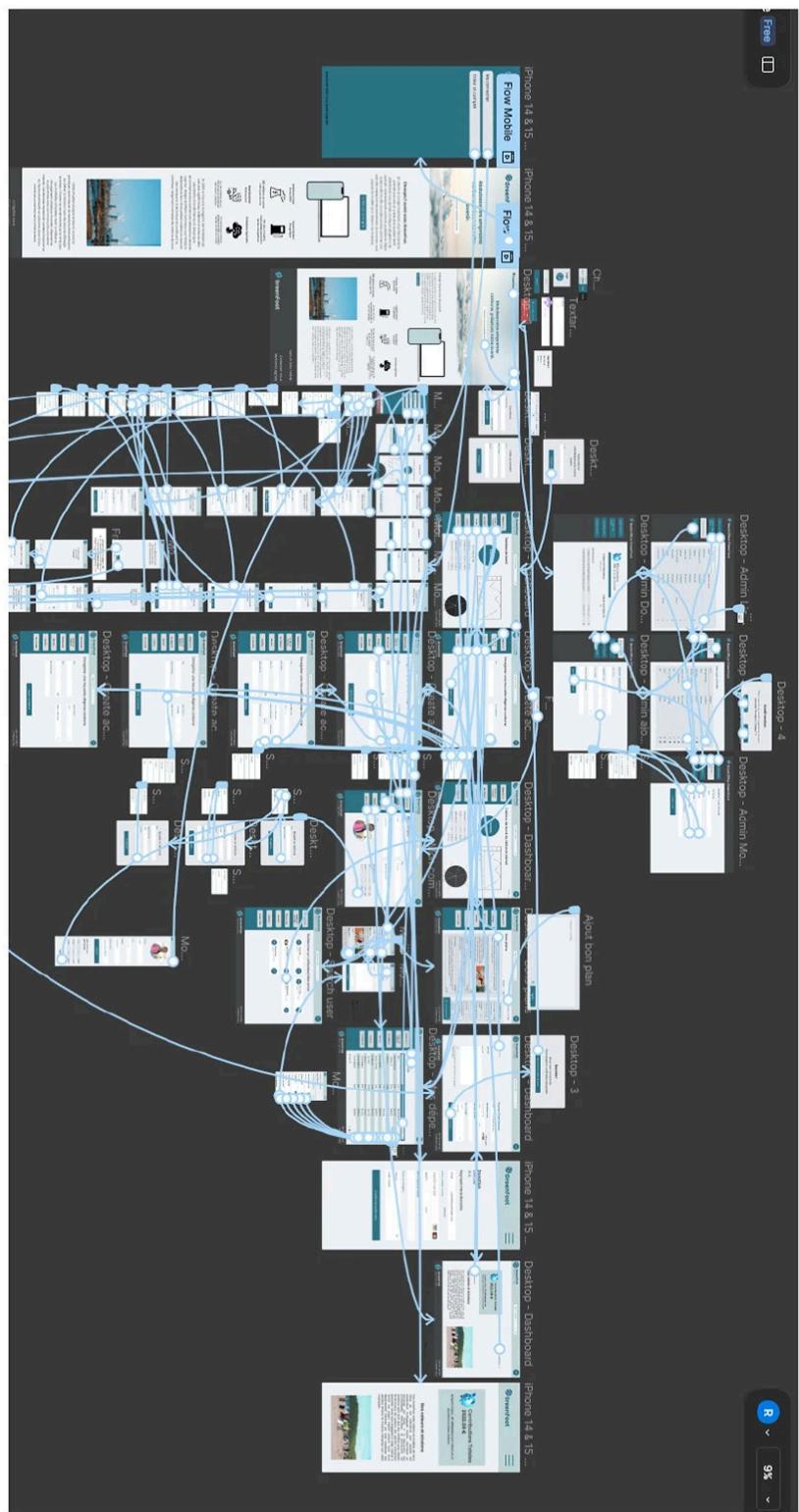
GreenFoot MPD

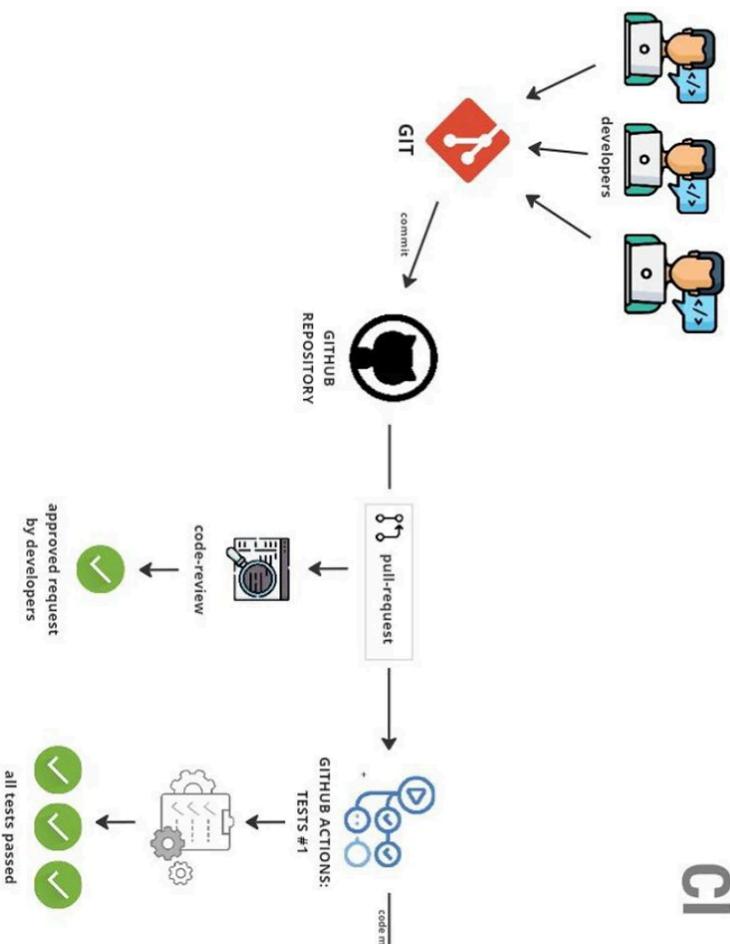
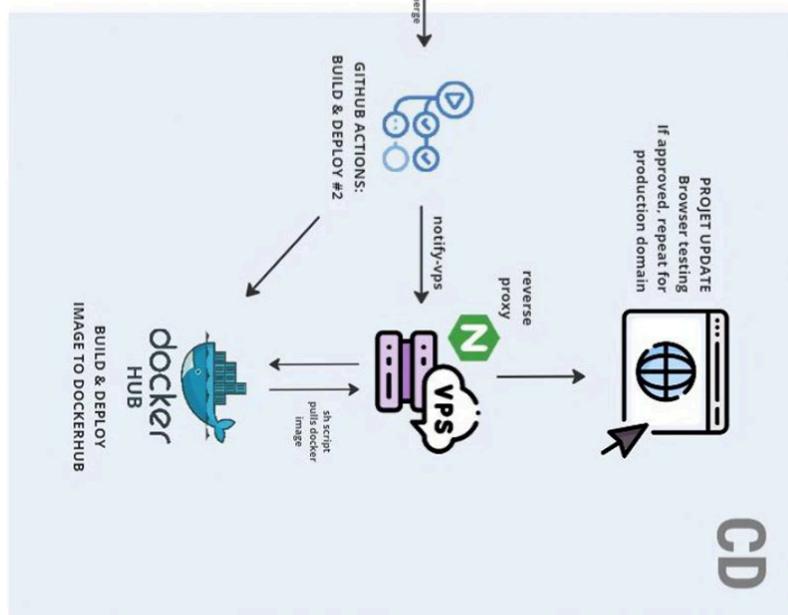


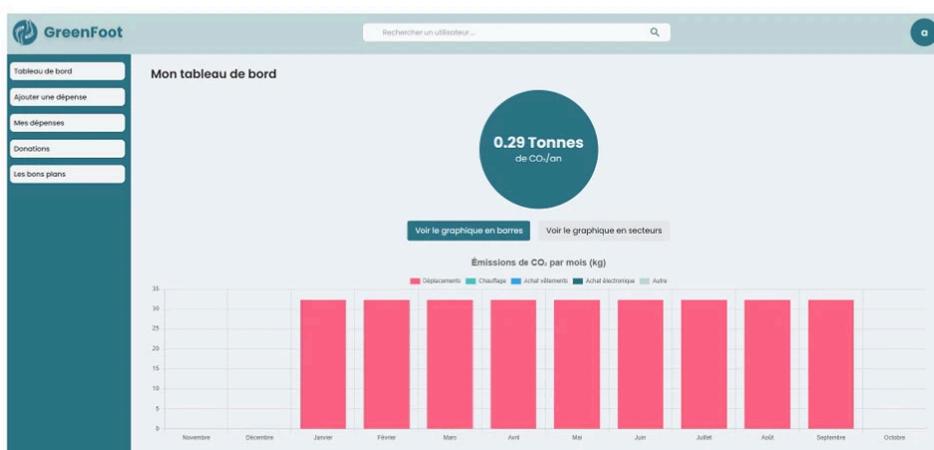
B



B



B**CI****CD**

B

Bon plans

Gretaaaaa 10/10/2024

Essayer de tendre vers le zéro déchet

Arriver ou zéro déchet, un objectif ambitieux mais pas impossible à atteindre... Pour y arriver, nous vous recommandons de vous fixer des objectifs atteignables :

- évitez les produits très emballés : repérez le magasin de vente en vrac le plus proche de chez vous pour vos achats ;
- préférez les produits réutilisables aux produits jetables : essayez les couches lavables pour vos enfants, prévoyez couverts et verres lavables pour vos pique-niques et vos réceptions, pensez à avoir en permanence un cabas pour faire vos courses, apportez une gourde recyclable au bureau ;
- vous vous sentez dans l'obligation d'utiliser du plastique ? lancez-vous dans la conception de produits d'entretien, voire de crèmes cosmétiques en suivant attentivement les recettes proposées sur la toile ;
- réutilisez, transformez, améliorez : conservez par exemple les boîtes d'emballage de vos produits pour y mettre vos bijoux, vos accessoires, les jeux de vos enfants etc.

Lilafleurie 10/10/2024

Plus de transports en commun ou de vélo

Les transports constituent la première source d'émissions de gaz à effet de serre en France (27,8 % des émissions totales du pays en 2012). La quasi-totalité (92 %) provient du transport

Partagez un bon plan

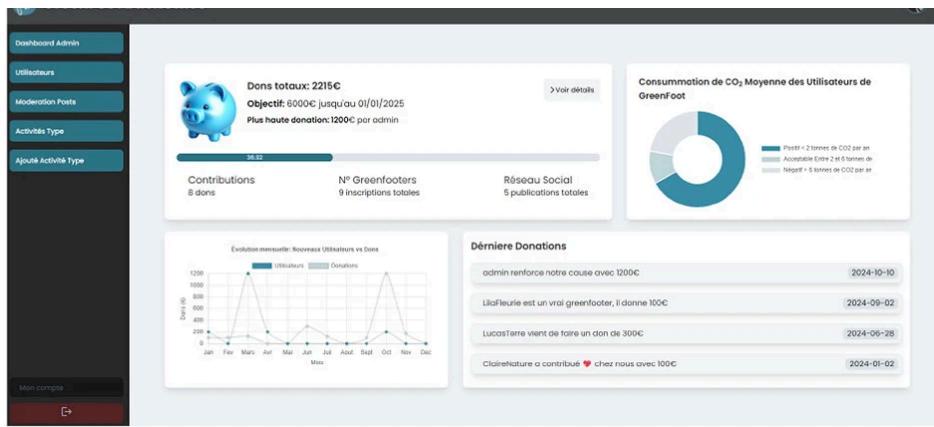
Trier par ▾ Filtrez ▾

Les posts les plus likés

Alimentation : moins de viande, plus de légumes locaux et de saison La viande rouge est plus émettrice que la viande blanche (volaille) du fait de ... Afficher tout ▾

Essayer de tendre vers le zéro déchet Arriver ou zéro déchet, un objectif ambitieux mais pas impossible à atteindre....

B



Créer Nouveau Type d'Activité

You have the possibility to create a new activity type. Make sure to fill in the fields for the category, the CO₂ emission value, and the measurement unit, ensuring they are acceptable by the ecological community.

Nom:	Barbecue à la maison
Catégorie:	Alimentation
Emissions CO ₂ :	144
Unité Mesure:	grammes de CO ₂

Confirmer **Annuler**

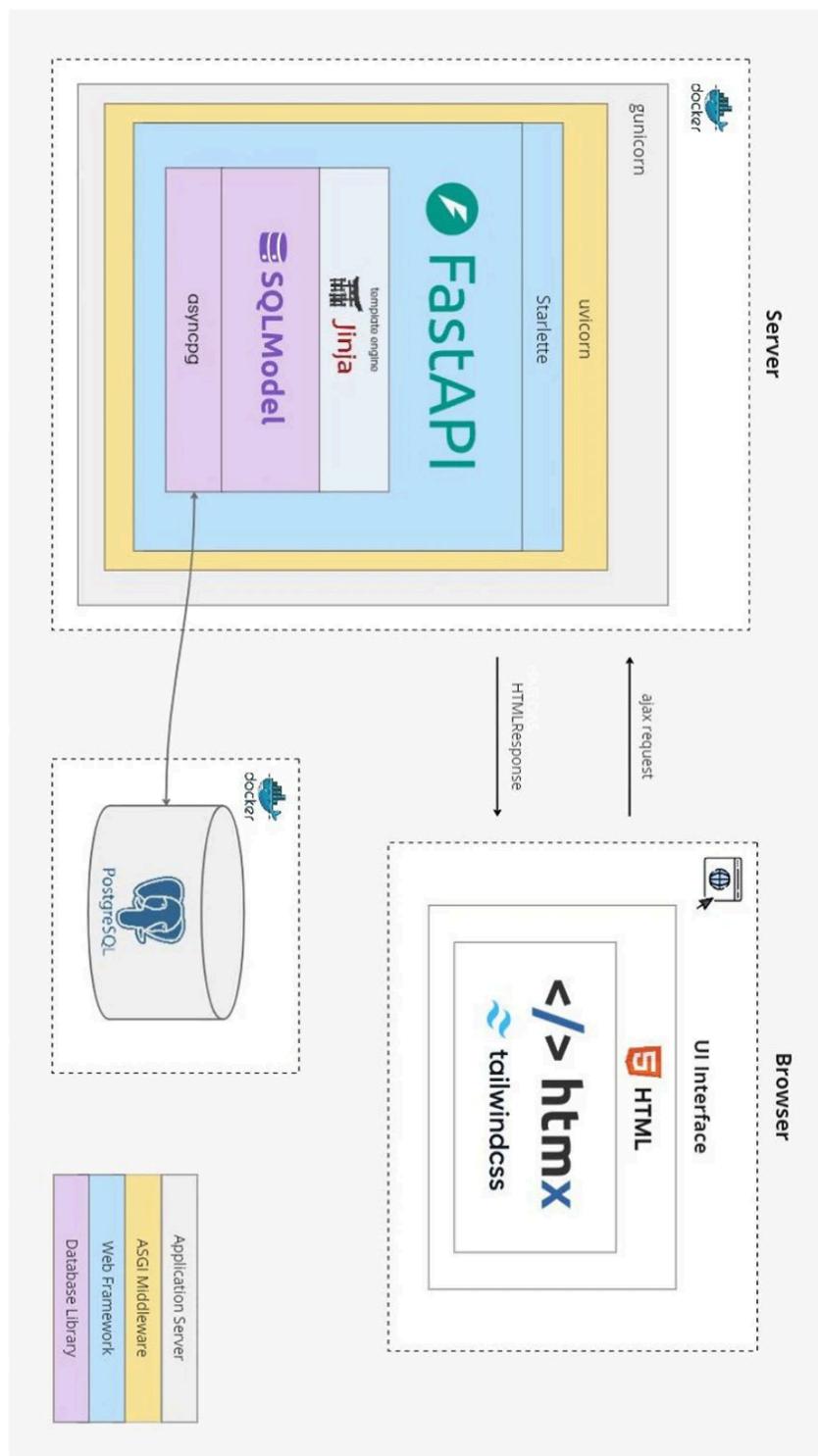
Utilisateurs

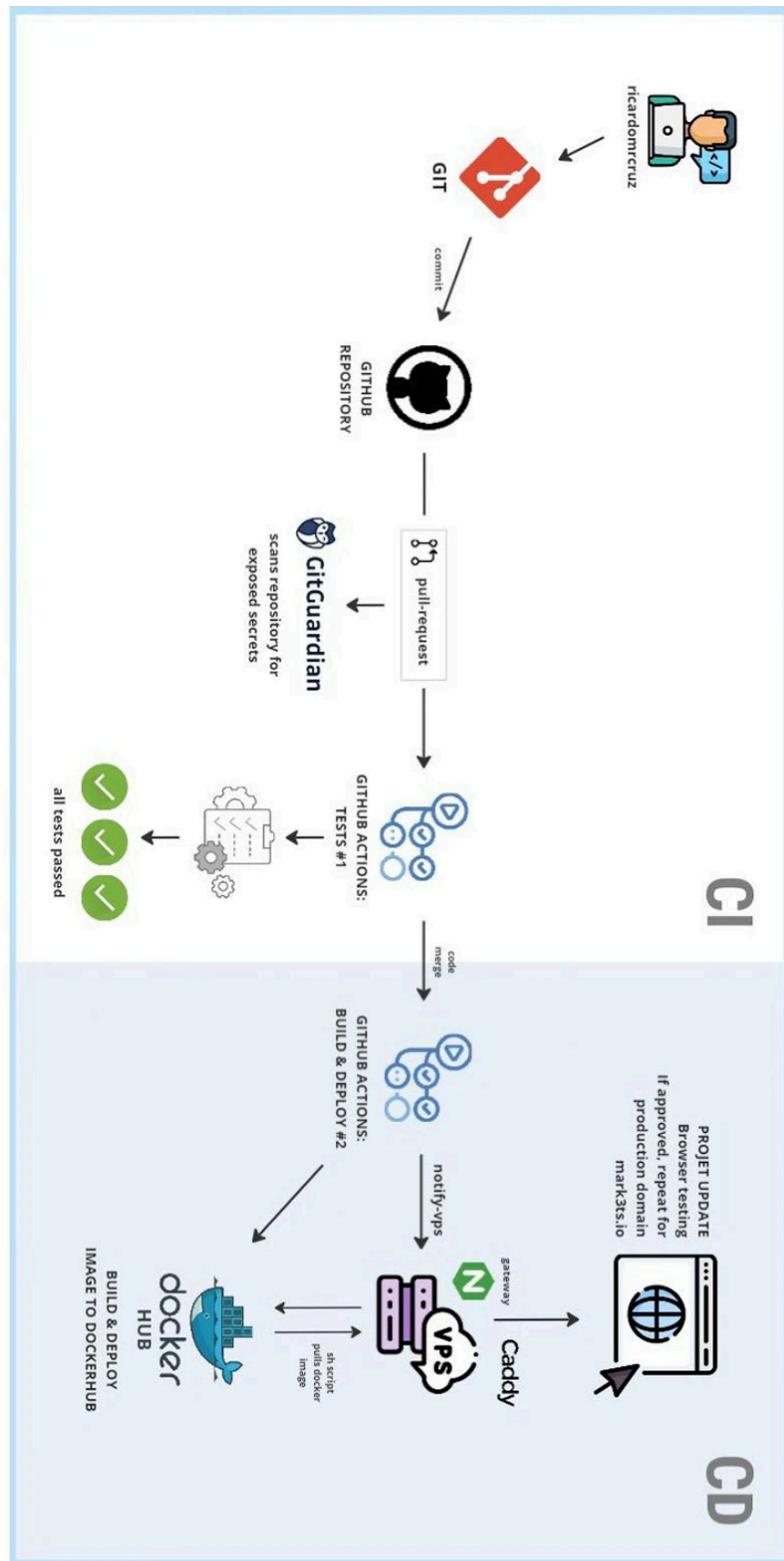
Action	NAME	INSCRIPTION	ROLE	ACTION
<input type="checkbox"/>	Gretaaaaa greenGretaaa@opp.com	01/01/2024	user	
<input type="checkbox"/>	MarcLevert marco0254@opp.com	12/03/2024	user	
<input type="checkbox"/>	LilaFleurie lilaFleurie@opp.com	20/04/2024	user	
<input type="checkbox"/>	admin admin@opp.com	10/10/2024	admin	
<input type="checkbox"/>	ClaireNature claire6576@opp.com	12/03/2024	user	
<input type="checkbox"/>	JulienCC julien@opp.com	12/03/2024	user	

+ Précédente **Page 1** **Suivante >**



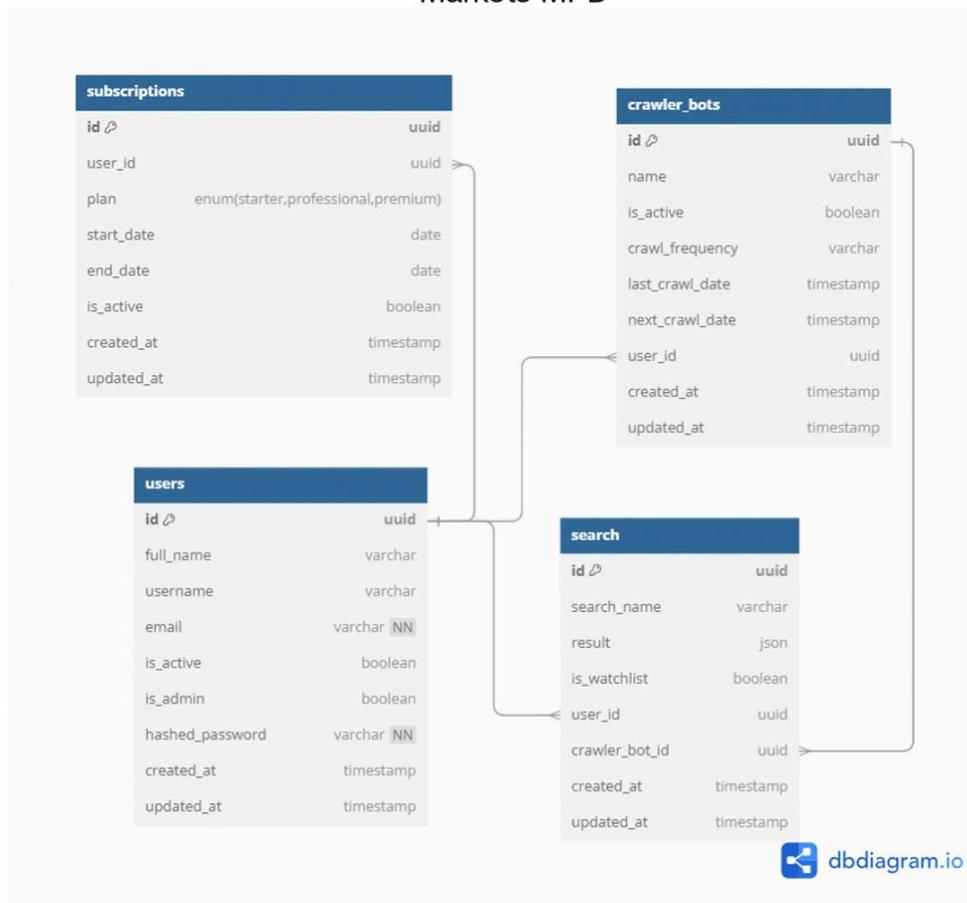
Part 2 - Mark3ts Archi Système



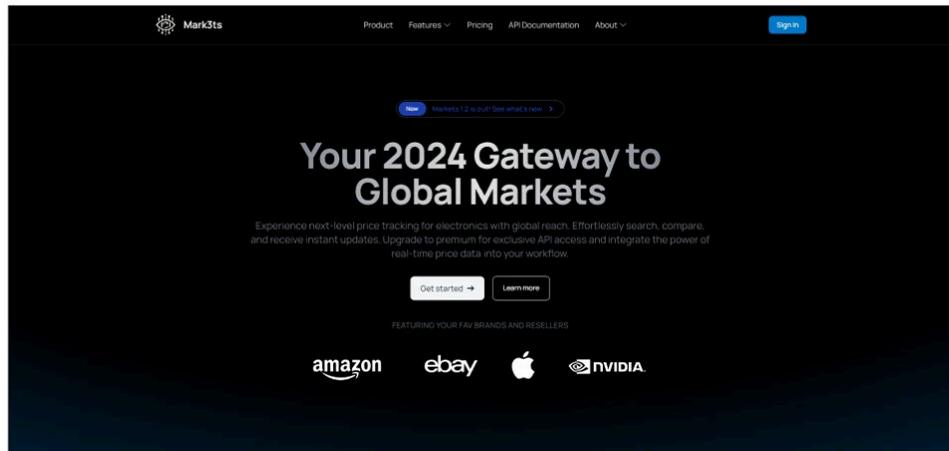
B



Mark3ts MPD



Captures d'écran UI Mark3ts



B

Stay Ahead of the Game

From savvy shoppers to data-driven enterprises, mark3ts offers tailored pricing plans to help you catch the best deals and stay ahead in the fast-paced world of real-time price intelligence.

Starter	Professional	Premium
Ideal for newcomers seeking to explore our app's capabilities before committing to a purchase. Perfect for casual users.	Tailored for professionals requiring instant access to comprehensive market data tools. Enhances daily workflows.	Designed for developers and engineers looking to seamlessly integrate our powerful scraping tools into their applications.
\$0 /month	\$29 /month	\$99 /month
<ul style="list-style-type: none"> ✓ Up to 10 queries per month ✓ Access to basic product data ✓ Simple market price search ✓ Limited feature set ✓ No watchlist available 	<ul style="list-style-type: none"> ✓ Up to 400 monthly queries ✓ CSV/JSON data downloads ✓ Email ticker notifications ✓ No API access provided ✓ Basic AI tool features 	<ul style="list-style-type: none"> ✓ Up to 1000 monthly queries ✓ 3 API keys with documentation ✓ Real-time data and tools ✓ Advanced AI market analysis ✓ Premium technical support
Get started	Get started	Get started

Charte
Graphique

vs code bg Color Palette



jennijessi

1 Favorites 0 Comments

Colors in Palette

Color	Hex	RGB
#333333	(51,51,51)	
#252525	(37,37,37)	
#1e1e1e	(30,30,30)	
#0098ff	(0,152,255)	
#0065a9	(0,101,169)	

Login to add palette to your favorites.



74

B

Mark3ts

RTX 4090 - The Apex of Graphics! Track the best Gaming GPU price in the Market Right Now!

Recent Prices from \$450

Hot Deals of the day

- Toshiba 55-inch C350 4K Fire TV: \$449.99 HOT
- Asus Vivobook Pro QLED 16-inch: \$1,299.99 (Save \$500)
- JBL Tune Buds: \$49.99 (Save \$10)

Recent Searches

- Headphones Silence Mode No Cable: 2 August 2024 09:44
- Graphic Cards Nvidia RTX 4080: 26 July 2024 11:10
- Laptop Mac Apple: 19 July 2024 02:44

Watchlist

- MSI GeForce RTX 4070 Ti Super 16G Ventus: \$899.99
- Krups Machine à Café Gran, 2 Expressos: \$259.99
- Inové 2K Camera Surveillance WiFi Intérieure: \$179.99

Popular Categories

- Smartphones & Accessories
- Videogame Consoles, Games & Accessories
- CPU Processors & Components
- Motherboards & Components
- GPU Graphics Processors & Components
- Headphones & Sound

Mark3ts

GPU Graphics Processors & Components

Geographical location

Location: United States

Zip Code:

Localization

Domain: amazon.com

Language: en-US

Advanced Filters

Condition: All

Price List:

Product Grid

Image	Name	Condition	Price
	MSI Gaming GeForce RTX 3060 12GB 16Gbps GDDR6 192-bit HDMI DP PCIe 4.0x16	New	\$284.51
	ASUS ProArt GeForce RTX™ 4060 Ti 16GB OC Edition GDDR6 Graphics Card (PCIe 4.0 x16)	New	\$529.99
	AMD Ryzen 7 5700G 8-Cores, 16-Threads Unlocked Desktop Processor with Wraith Stealth cooler	New	\$167.25
	MSI GAMING GeForce GTX 1060 6GB GDDR5 192-bit HDCP Support DirectCU Dual Fan V2	New	\$339.99

Mark3ts

New Used Under \$50

Product Grid

Image	Name	Condition	Price
	SAMSUNG Galaxy S24 Ultra Cell Phone, 256GB A SmartPhone, Unlocked Android...	New	\$1,043.55
	Motorola Moto G50 I 2023 Unlocked Made for US 4/64GB 48MP Camera Infrared	New	\$149.99
	OnePlus 12R 8GB RAM+128GB Dual-SIM US Factory Unlocked Android...	New	\$449.99
	SAMSUNG Galaxy A35 5G A Series Cell Phone, 128GB Unlocked Android...	New	\$313.99
	SAMSUNG Galaxy A15 5G A Series Cell Phone		
	OnePlus Nord N20 5G I Unlocked Dual-SIM		
	Moto G Play 2023 5G Dual SIM Unlocked		
	Xiaomi Redmi Note 15 5G Dual SIM Unlocked		

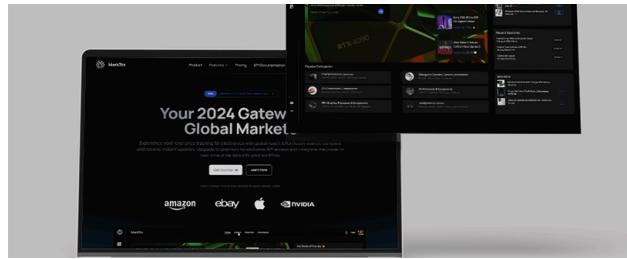
75



Ricardo Martinho

Suivre

B



Automated Food Safety

B

Que pensez-vous de ce projet ?

Commenter

PROPRIÉTAIRE



Ricardo Martinho
Strasbourg, France

Suivre

Message

DISCIPLINES

Education

Conception de sites Web

Création d'applications

Dossier Projet - GreenFoot Ap...

1 like 2 views 0 comments

Publié le : 19 décembre 2024

documentation

fullstackdevelopment

nodejs

typescript

python

Ecology

© All Rights Reserved

Signaler

**Conçu pour les créatifs**[Tester Behance Pro](#)[Trouver l'inspiration](#)[Se faire embaucher](#)[Vendre des ressources de création](#)[Vendre des prestations en indépendant](#)**Trouver des talents**[Publier une offre d'emploi](#)[Designers graphiques](#)[Photographes](#)[Monteurs vidéo](#)[Webdesigners](#)[Illustrateurs](#)**Behance**[À propos de Behance](#)[Adobe Portfolio](#)[Télécharger l'application](#)[Blog](#)[Carrières chez Behance](#)[Centre d'aide](#)[Nous contacter](#)[Termes de recherche populaires](#)[Connexion](#)**Réseaux sociaux** [Instagram](#) [Twitter](#) [Pinterest](#) [Facebook](#) [LinkedIn](#)

B



Français

CONDITIONS D'UTILISATION

Confidentialité

Communauté

Préférences en matière de cookies

Ne pas vendre ni partager mes données personnelles