

# TPI

## Gestion des activités d'un apiculteur

Kevin Avdylaj – CID4B ETML

Chef de projet : M. Gaël Sonney

Experts : Nicolas Borboën et Pascal Benzonana

Lieu : ETML Vennes, Av. de Valmont 30, 1014 Lausanne

Date : Du jeudi 02 mai au lundi 03 juin 2024

Durée : 88 heures

## Table des matières

1	Analyse préliminaire.....	4
1.1	Introduction.....	4
1.2	Objectifs.....	4
1.3	Planification initiale.....	4
2	Analyse / Conception .....	5
2.1	Concept .....	5
2.2	Méthode de projet.....	5
2.3	Technologies du projet .....	6
2.3.1	Typescript .....	6
2.3.2	Node JS .....	6
2.3.3	Express JS .....	6
2.3.4	MySQL .....	6
2.3.5	Prisma (ORM) .....	7
2.3.6	Vue JS .....	7
2.3.7	Figma .....	7
2.3.8	Draw.io.....	7
2.3.9	VS code.....	7
2.3.10	Docker .....	7
2.4	Fonctionnalités.....	7
2.4.1	Authentification.....	7
2.4.2	CRUD.....	7
2.4.3	Consulter les activités par année .....	8
2.4.4	Consulter les activités d'une ruche ou d'un rucher.....	8
2.4.5	Ergonomie des interfaces.....	8
2.5	Maquette.....	8
2.5.1	Login.....	9
2.5.2	Page principale.....	9
2.5.3	Activités .....	10
2.5.4	Détails ruche.....	11

2.5.5	Détails rucher.....	12
2.6	Base de données .....	12
2.6.1	MCD.....	13
2.6.2	MLD .....	14
2.6.3	MPD .....	15
2.7	API .....	16
2.8	Stratégie de test .....	19
3	Réalisation.....	19
3.1	Remplir la base de données .....	19
3.2	Tests.....	19
3.2.1	API .....	19
3.2.2	Interface.....	19
4	Conclusion .....	20
4.1	État finale de l'application .....	20
4.2	Objectifs atteints .....	20
4.3	Objectifs non-atteints .....	20
4.4	Problèmes rencontrés.....	20
4.5	Amélioration possible .....	20
4.6	Bilan de la planification .....	20
4.7	Bilan personnel.....	20
5	Sources – Bibliographie .....	20
6	Glossaire .....	20
7	Annexes.....	21
8	Résumé .....	21

# 1 Analyse préliminaire

## 1.1 Introduction

Ce rapport contient la réalisation du début à la fin de mon travail pratique individuel (TPI) effectué à l'ETML.

Le sujet de TPI est un site web qui servira à la gestion de ruches et rucher pour un apiculteur.

Le choix du sujet du TPI n'a peu d'importance et j'ai laissé celui-ci à mon chef de projet, ce qui va nous intéresser c'est la conception d'un site web full-stack

## 1.2 Objectifs

Le but est de fournir une application web permettant la journalisation des activités pour un apiculteur.

L'application doit pouvoir :

- Gérer l'authentification d'un apiculteur
- Fournir les opérations CRUD sur un rucher, une ruche et les activités qui y sont liées
- Consulter la liste de toutes les activités par années
- Consulter la liste des ruchers et ruche d'un apiculteur

L'application doit également avoir une base de données conçue selon ce qui a été vu lors des modules ICT (104, 105, 1153).

Le code source doit être lisible et respecter les conventions de nommage standards pour le langage de programmation utilisé.

## 1.3 Planification initiale

La planification initiale est fournie en annexe

## 2 Analyse / Conception

### 2.1 Concept

Lors de la saison des abeilles, un apiculteur doit régulièrement réaliser des inspections et relever les détails propres à chaque ruche. Il doit aussi exécuter des travaux ou des activités spécifiques.

Cette application est destinée à un apiculteur qui s'occupe de plusieurs ruches et réalise les travaux nécessaires à la bonne conduite de son rucher. Un rucher est composé de plusieurs ruches. Il possède un numéro de rucher, un nom et une localisation.

Une ruche possède un numéro, une description, une couleur, l'année de naissance de la reine associée à une couleur. Un apiculteur peut posséder plusieurs ruchers.

Les activités possèdent une catégorie, une description, une durée et une date. Les catégories d'activités sont inspection, mise des hausse, extraction, traitement et nourrissage. Une activité peut être réalisée sur une ruche ou un rucher (toutes les ruches du rucher).

### 2.2 Méthode de projet

Pour ce projet je vais utiliser la méthode des 6 pas

#### ❖ **INFORMER**

Ici il va falloir s'informer sur le projet, prendre connaissance des objectifs, des outils à utiliser, etc.

Pour cette étape j'ai pris connaissance du cahier des charges et eu une discussion avec le premier expert sur le déroulement du TPI. J'ai aussi obtenu des clarifications sur le cahier des charges après avoir fait part de mes questionnements à mon chef de projet.

#### ❖ **PLANIFIER**

La phase de planification consiste simplement en la réalisation de ma planification initiale

#### ❖ **DÉCIDER**

Pour la partie « décider » je dois choisir la façon dont laquelle je vais réaliser ce projet. Les technologies utilisées pour ce projet ont déjà été décidée au

préalable lors du P\_APPRO 1 et 2. Il ne manque plus qu'à établir le modèle de base de données, la maquette du site ainsi que la stratégie de test. Une fois tout ça fait, on peut passer à la phase de réalisation

### ❖ **RÉALISER**

C'est ici qu'on commence à coder ! Il faut implémenter le backend (Base de données, API, CRUD) et le frontend (Intégration de la maquette)

### ❖ **CONTRÔLER**

Pour le contrôle, je vais effectuer les tests prévus sur l'application, relire et finaliser le rapport

### ❖ **ÉVALUER**

L'évaluation consiste à la rédaction de la conclusion de ce rapport. Conclusion qui contient tous les bilans du projet, le problème rencontré, l'état finale de l'application, etc.

## 2.3 Technologies du projet

### 2.3.1 Typescript

Typescript est le langage de programmation utilisé dans ce projet. Il permet de réaliser du code en frontend ainsi qu'en backend, me permettant d'utiliser qu'un seul langage pour tout le projet. Typescript apporte des éléments supplémentaires à javascript, notamment les types. Cela me permet de typer mes variables et de déboguer plus simplement

### 2.3.2 Node JS

Node est un environnement d'exécution pour javascript qui permet de faire du code javascript en dehors du navigateur. Il sera utilisé pour la réalisation du backend.

### 2.3.3 Express JS

Express est un Framework javascript qui aide à la réalisation d'API en fournissant les éléments de base pour leurs création (route, middlewares, etc.).

### 2.3.4 MySQL

MySQL est le SGBD que je vais utiliser dans ce projet. Il permet gérer des bases de données relationnelles, ce qui est nécessaire pour ce projet.

### 2.3.5 Prisma (ORM)

Prisma est un ORM qui supporte le Typescript. Celui-ci va me permettre de communiquer avec la base de données en ayant les types des données directement traduit en Typescript et ainsi me donné plusieurs avantages comme l'autocomplétions dans mon IDE, requêtes simplifiée, migrations, etc.

### 2.3.6 Vue JS

Vue JS et le framework frontend que j'ai choisi, il va me simplifier l'intégration du site en me permettant de créer des composants qui contiennent leur propres template (HTML), style (CSS) et script (Typescript), facilitant la création d'interface réactive

### 2.3.7 Figma

Figma est un éditeur graphique, permettant la réalisation de maquette pour site web principalement. C'est avec lui que je vais designer mon site

### 2.3.8 Draw.io

Draw.io est un site qui permet de créer plusieurs type diagrammes. Je l'ai utilisé pour schématiser ma base de données (MCD, MLD, MPD)

### 2.3.9 VS code

VS code est l'IDE que j'ai utilisé pour ce projet

### 2.3.10 Docker

Docker sera utilisé pour faire tourner la base de données, ainsi que phpMyAdmin.

## 2.4 Fonctionnalités

### 2.4.1 Authentification

L'authentification s'effectue avec un nom d'utilisateur et un mot de passe, une fois authentifié l'utilisateur possède tous les droits sur l'application. Un utilisateur non-authentifié n'a accès à aucune fonctionnalité.

### 2.4.2 CRUD

Les opérations CRUD devront être établie pour les ruchers, ruches et activités.

### 2.4.3 Consulter les activités par année

L'application doit permettre de pouvoir consulter toutes les activités, en filtrant par année.

### 2.4.4 Consulter les activités d'une ruche ou d'un rucher

Les activités liées à une ruche ou un rucher doit pouvoir être consulter depuis la page détails de celui-ci.

### 2.4.5 Ergonomie des interfaces

Une maquette du site doit être réalisée dans le respect des critères UX (simplicité, cohérence, interaction, crédibilité, etc.).

## 2.5 Maquette

Pour la conception de la maquette avec Figma j'ai décidé de me basé sur le concept du design atomique.

Le design atomique (atomic design) décompose une interface en atome, molécule, organisme, puis finalement la page entière. Un atome serait par exemple une typographie ou une couleur, puis une molécule serait une combinaison d'atome, comme un bouton (texte avec une couleur en background).

Un organisme et une combinaison de molécule et la page finale une combinaison d'organisme. Pour mon TPI, j'ai simplifié ce processus en sélectionnant une typographie et une palette de couleurs puis j'ai directement créé les organismes pour produire la page. C'est suffisant pour mon utilisation.

Voici un exemple :



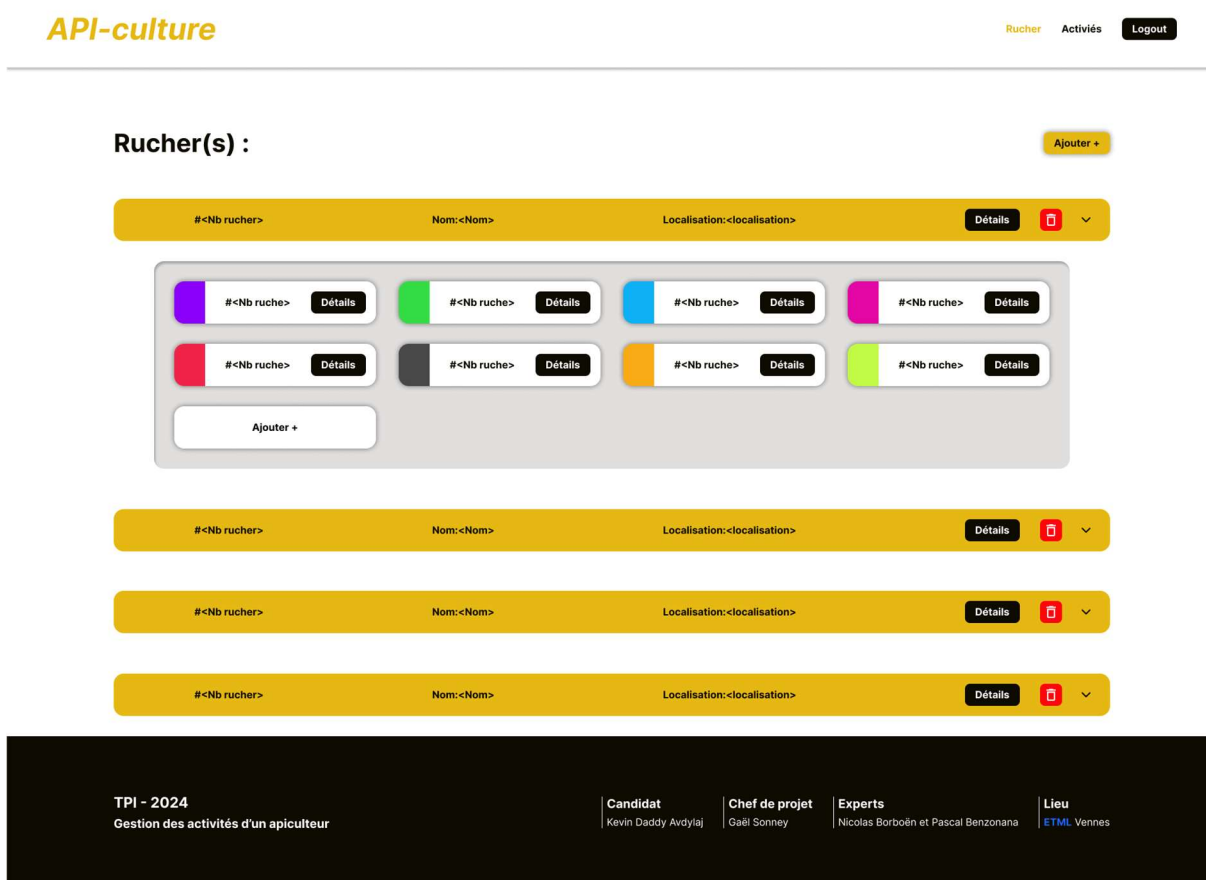
Dans mon projet j'ai choisi la police « inter » et le jaune comme couleur principale et c'est avec ces éléments que je vais créer les pages du site.



### 2.5.1 Login



### 2.5.2 Page principale








### 2.5.3 Activités

## API-culture

Rucher   **Activités**   Logout

**Activité(s):**

◀ 2024 ▶

<Catégorie>	<DD-MM-YYYY>	<HH-MM>	<a href="#">Ruche:&lt;Nb ruche&gt;</a>	Modifier		▼
Description:						
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam et auctor lectus. In eleifend ornare tincidunt. Suspendisse sollicitudin arcu leo, sed sodales dolor egestas sed. Curabitur sed turpis a mauris auctor blandit. Sed ac euismod purus.						
<Catégorie>	<DD-MM-YYYY>	<HH-MM>	<a href="#">Ruche:&lt;Nb ruche&gt;</a>	Modifier		▼
<Catégorie>	<DD-MM-YYYY>	<HH-MM>	<a href="#">Ruche:&lt;Nb ruche&gt;</a>	Modifier		▼
<Catégorie>	<DD-MM-YYYY>	<HH-MM>	<a href="#">Ruche:&lt;Nb ruche&gt;</a>	Modifier		▼
<Catégorie>	<DD-MM-YYYY>	<HH-MM>	<a href="#">Ruche:&lt;Nb ruche&gt;</a>	Modifier		▼

**TPI - 2024**  
**Gestion des activités d'un apiculteur**

**Candidat**  
Kevin Daddy Avdylaj

**Chef de projet**  
Gaël Sonney

**Experts**  
Nicolas Borboën et Pascal Benzonana

**Lieu**  
**ETML** Vennes

2.5.4 Détails ruche

API-culture

RucherActivésLogout

Rucher: <NomRucher>

Numéro: <Nb ruche>

Couleur: Jaune

Reine:

- Couleur: Bleu
- Naissance: 2024

Description:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam et auctor lectus. In eleifend ornare tincidunt. Suspendisse sollicitudin arcu leo, sed sodales dolor egestas sed. Curabitur sed turpis a mauris auctor blandit. Sed ac euismod purus.

Modifier

Activité(s)

Ajouter +

<Catégorie><DD-MM-YYYY><HH-MM>Ruche:<Nb ruche>Modifier🗑️▼

Description:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam et auctor lectus. In eleifend ornare tincidunt. Suspendisse sollicitudin arcu leo, sed sodales dolor egestas sed. Curabitur sed turpis a mauris auctor blandit. Sed ac euismod purus.

<Catégorie><DD-MM-YYYY><HH-MM>Ruche:<Nb ruche>Modifier🗑️▼

<Catégorie><DD-MM-YYYY><HH-MM>Ruche:<Nb ruche>Modifier🗑️▼

<Catégorie><DD-MM-YYYY><HH-MM>Ruche:<Nb ruche>Modifier🗑️▼

<Catégorie><DD-MM-YYYY><HH-MM>Ruche:<Nb ruche>Modifier🗑️▼

<Catégorie><DD-MM-YYYY><HH-MM>Ruche:<Nb ruche>Modifier🗑️▼

TPI - 2024

Gestion des activités d'un apiculteur

Candidat

Kevin Daddy Avdylaj

Chef de projet

Gaël Sonney

Experts

Nicolas Borboën et Pascal Benzonana

Lieu

ETML Vennes

Auteur : Kevin Avdylaj

Page 11 sur 21

[Tapez ici]

### 2.5.5 Détails rucher

#### API-culture

Rucher Activiés Logout

Nom: &lt;nom du rucher&gt;

Numéro: &lt;Nb&gt;

Localisation: &lt;localisation&gt;

Modifier

#### Activité(s)

Ajouter +

<Catégorie>	<DD-MM:YYYY>	<HH-MM>	Ruche:<Nb ruche>	Modifier		▼
Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam et auctor lectus. In eleifend ornare tincidunt. Suspendisse sollicitudin arcu leo, sed sodales dolor egestas sed. Curabitur sed turpis a mauris auctor blandit. Sed ac euismod purus.						
<Catégorie>	<DD-MM:YYYY>	<HH-MM>	Ruche:<Nb ruche>	Modifier		▼
<Catégorie>	<DD-MM:YYYY>	<HH-MM>	Ruche:<Nb ruche>	Modifier		▼
<Catégorie>	<DD-MM:YYYY>	<HH-MM>	Ruche:<Nb ruche>	Modifier		▼
<Catégorie>	<DD-MM:YYYY>	<HH-MM>	Ruche:<Nb ruche>	Modifier		▼
<Catégorie>	<DD-MM:YYYY>	<HH-MM>	Ruche:<Nb ruche>	Modifier		▼

TPI - 2024  
Gestion des activités d'un apiculteurCandidat  
Kevin Daddy AvdylajChef de projet  
Gaël SonneyExperts  
Nicolas Borboën et Pascal BenzonanaLieu  
ETML Vennes

## 2.6 Base de données

La base de données est composée de 8 tables dont une table pivot servant à lier les activités aux ruches.

Il n'y a aucun lien entre une activité et un rucher. Comme l'application d'une activité sur un rucher consiste à appliquer l'activité à toutes les ruche qui le compose, les activités sont uniquement liées aux ruches en base de données.

Une table « t\_reine » a été créer pour associer l'année de naissance d'une reine à une couleur.

Une table « t\_couleur » a été créer pour associer le nom d'une couleur à son code hexadécimale, permettant ainsi d'afficher la bonne couleur en CSS si nécessaire.

Une table « t\_catégorie » a été créée pour modéliser la liste prédéfinie des catégories d'activité disponible, évitant ainsi le risque d'entrer en base de données une catégorie qui n'existe pas.

Tous les identifiants sont des entiers non signés qui s'auto-incrémentent.

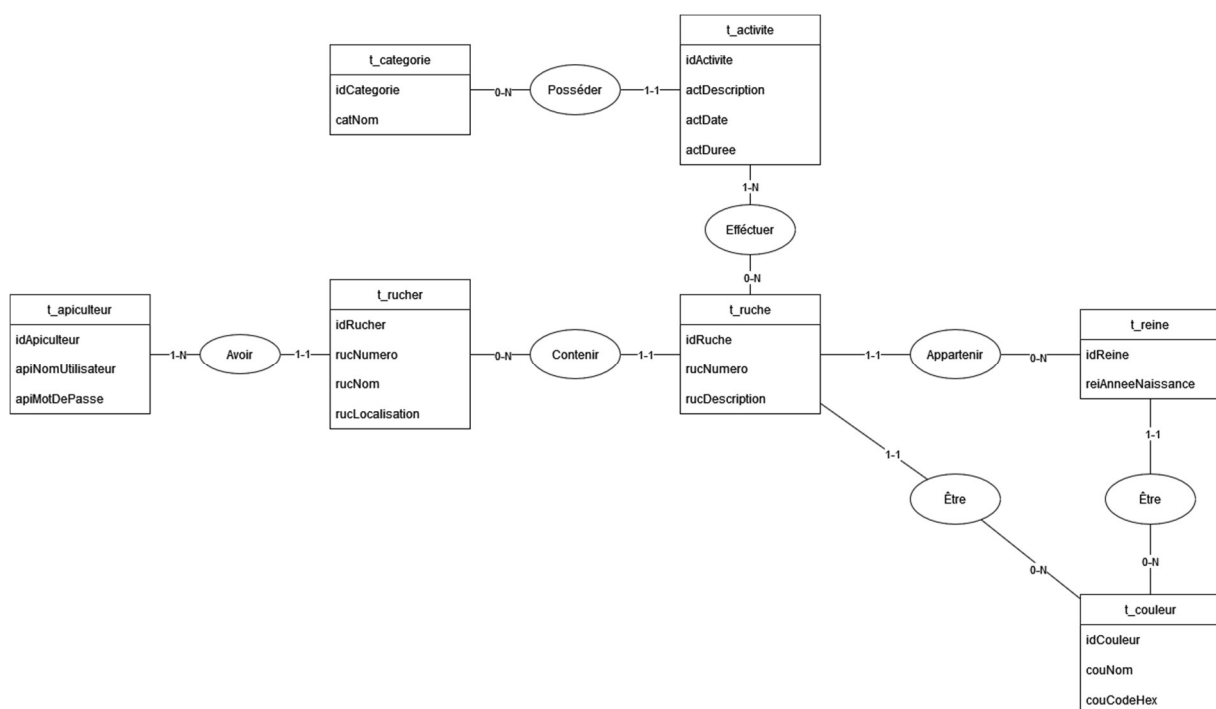
Tous les textes (sauf description) en base de données sont des varchar(255) laissant suffisamment de place ce qui va être stocké (nom d'utilisateur, mot de passe haché, etc.). Les descriptions sont stockées avec un type varchar(1000) laissant un bon paragraphe pour décrire quoi que ce soit.

L'année de naissance d'une reine est stockée avec le type « year » de MySQL qui permet de contenir une année sur le format « YYYY », parfait pour nos besoins.

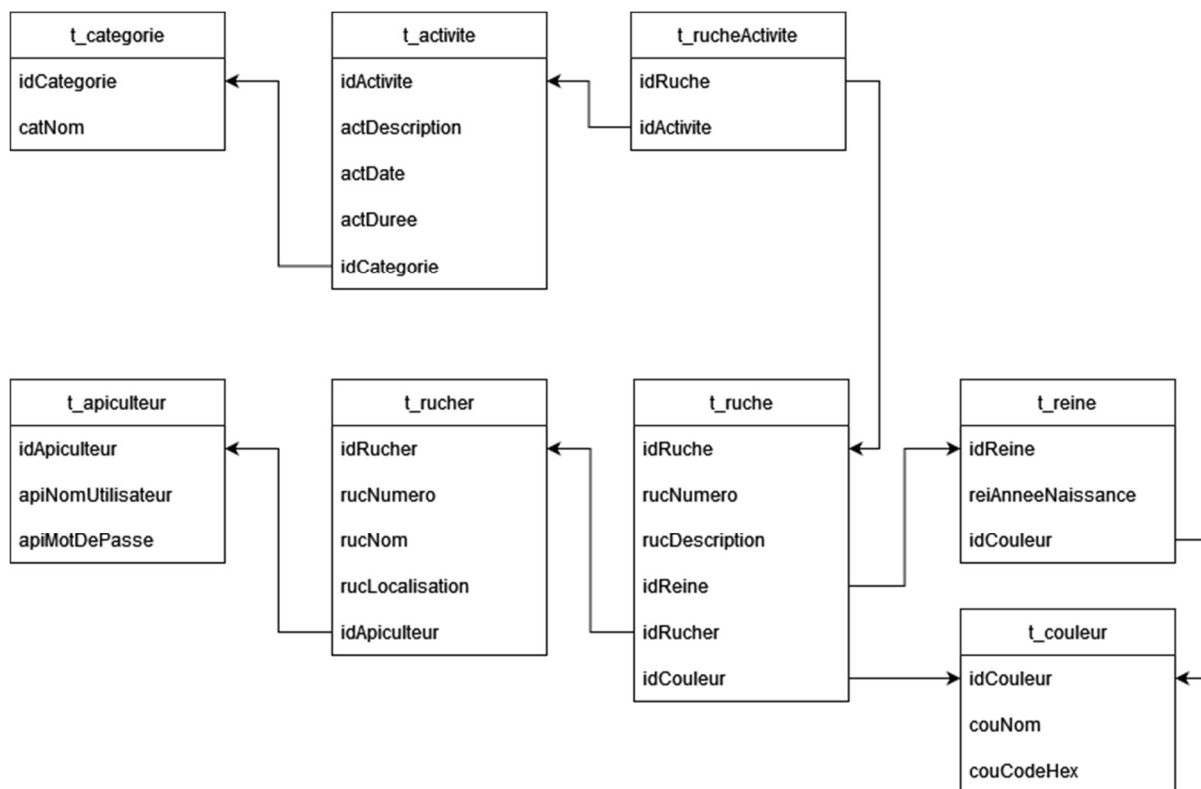
La date et la durée sont stockées avec leur type correspondant en MySQL (Date, Time).

Le MPD est réalisé avec Prisma (<https://www.prisma.io/docs/orm/prisma-schema/overview>) qui fournit sa propre façon de décrire un schéma de base de données. Prisma va également générer le code SQL pour effectuer la migration et enfin avoir une base de données MySQL avec notre schéma. (La table pivot « t\_rucheActivite » n'apparaît pas dans le schéma car ce genre de table est générée automatiquement par Prisma)

### 2.6.1 MCD



## 2.6.2 MLD



## 2.6.3 MPD

```

1  model t_categorie {
2      idCategorie Int          @id @default(autoincrement()) @db.UnsignedInt
3      catNom      String       @unique @db.VarChar(255)
4      t_activite  t_activite[]
5  }
6
7  model t_activite {
8      idActivite Int          @id @default(autoincrement()) @db.UnsignedInt
9      actDescription String    @db.VarChar(1000)
10     actDate      DateTime    @db.Date()
11     actDuree     DateTime    @db.Time()
12     categorie     t_categorie @relation(fields: [fkCategorie], references: [idCategorie])
13     fkCategorie  Int          @db.UnsignedInt
14     ruches       t_ruche[]
15 }
16
17 model t_apiculteur {
18     idApiculteur Int          @id @default(autoincrement()) @db.UnsignedInt
19     apiNomUtilisateur String  @unique @db.VarChar(255)
20     apiMotDePasse String      @db.VarChar(255)
21     t_rucher     t_rucher[]
22 }
23
24 model t_rucher {
25     idRucher      Int          @id @default(autoincrement()) @db.UnsignedInt
26     rucNumero     Int          @unique @db.UnsignedInt
27     rucNom        String       @db.VarChar(255)
28     rucLocalisation String    @db.VarChar(255)
29     apiculteur    t_apiculteur @relation(fields: [fkApiculteur], references: [idApiculteur])
30     fkApiculteur  Int          @db.UnsignedInt
31     t_ruche       t_ruche[]
32 }
33
34 model t_couleur {
35     idCouleur Int          @id @default(autoincrement()) @db.UnsignedInt
36     couNom     String       @unique @db.VarChar(255)
37     couCodeHex String      @db.VarChar(6)
38     t_reine    t_reine[]
39     t_ruche    t_ruche[]
40 }
41
42 model t_reine {
43     idReine      Int          @id @default(autoincrement()) @db.UnsignedInt
44     reiAnneNaissance Int      @db.Year
45     couleur      t_couleur    @relation(fields: [fkCouleur], references: [idCouleur])
46     fkCouleur    Int          @db.UnsignedInt
47     t_ruche      t_ruche[]
48 }
49
50 model t_ruche {
51     idRuche      Int          @id @default(autoincrement()) @db.UnsignedInt
52     rucNumero     Int          @unique @db.UnsignedInt
53     rucDescription String      @db.VarChar(1000)
54     reine         t_reine      @relation(fields: [fkReine], references: [idReine])
55     fkReine       Int          @db.UnsignedInt
56     rucher        t_rucher     @relation(fields: [fkRucher], references: [idRucher])
57     fkRucher      Int          @db.UnsignedInt
58     couleur       t_couleur     @relation(fields: [fkCouleur], references: [idCouleur])
59     fkCouleur     Int          @db.UnsignedInt
60     activites     t_activite[]
61 }

```

## 2.7 API

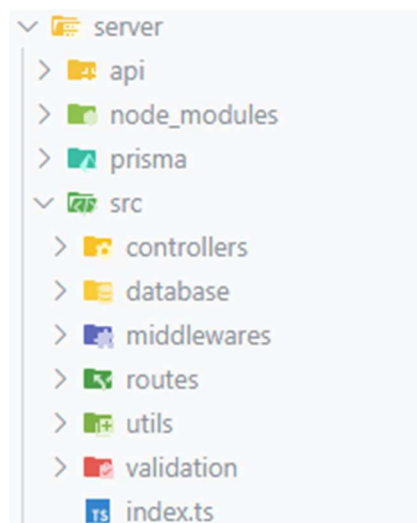
Pour permettre à notre application web d'accéder aux données dans la base de données il nous faut une API pour faire le pont entre ces deux éléments.



La communication entre la base de données et l'api sera simplifiée avec l'ORM Prisma qui s'occupera de générer les requêtes SQL et d'inférer les types des données récupérer pour profiter pleinement des capacités de Typescript.

### 2.7.1 Architecture

Pour la création de l'api j'ai établi une architecture qui va nous servir de fondement pour le code de l'API.



- **Server :**

Contient tout le code du serveur.

- **Api**

Contient des fichiers « .rest » qui sont utilisé par l'extension vscode « REST client ». Cette extension permet de décrire des requête http et de les exécuter. Je l'utilise pour tester mon api

- **Node\_modules**



Contient module et package de node.js

- **Prisma**

Ce dossier est créé automatiquement par Prisma, il contient le code SQL de toutes les migrations réalisées

- **Src**

- **Controllers**

Contient tous les contrôleurs de l'api. Les contrôleurs sont appelés dès qu'une requête les concernant a été faite et les gèrent.

- **Database**

Contient toutes les requêtes faites à la base de données pour créer, lire, modifier ou supprimer une donnée.

- **Middlewares**

Contient tous les middlewares de l'api

- **Routes**

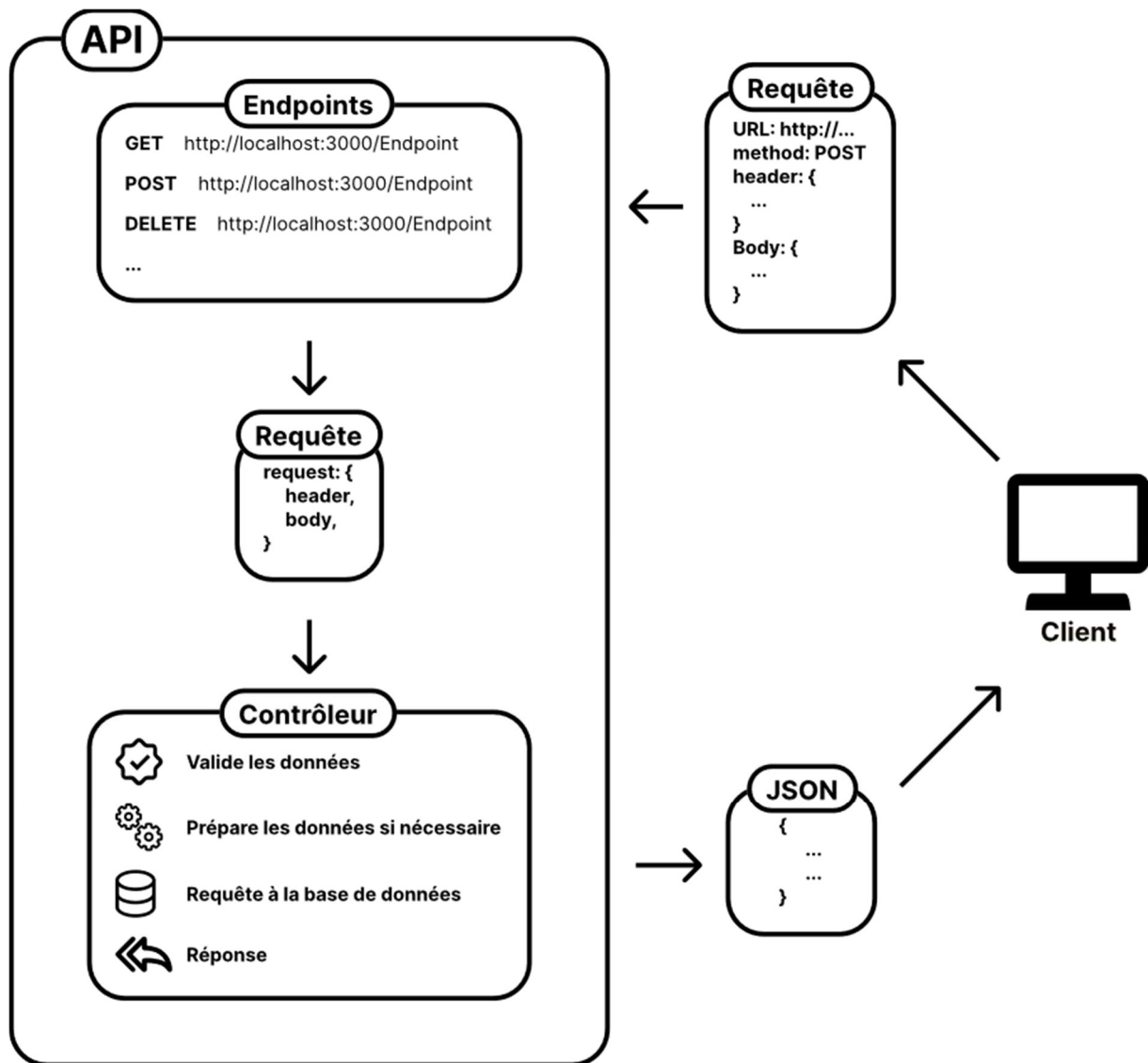
Contient toutes les routes de l'api

- **Utils**

Contient du code qui peut être utile à l'api et qui est utilisé dans plusieurs endroits de celle-ci.

- **Validation**

Contient du code pour la validation des données reçues depuis les requêtes faites à l'api.



À chaque fois que l'utilisateur va faire une requête à l'API, un contrôle lié à l'Endpoint qui a été appelé par le client va prendre en charge la requête.

Celui-ci va faire tout d'abord une validation des données reçues par le client, si celles-ci sont erronées et/ou non conformes à ce que le contrôleur attend de recevoir, une erreur sera retournée avec un message et un code http approprié.

Ensuite, le contrôleur va préparer les données reçues si nécessaire, par exemple, mettre le bon format de date sur les données reçues pour pouvoir ensuite les stocker en base de données.

Pour continuer, le contrôleur va faire la/les requêtes en base de données nécessaires et finalement retourner une réponse au client sous forme JSON.

## 2.8 Stratégie de test

Dans les points techniques évalué dans le TPI (point A14 à A20) il y a :

- L'apiculteur peut se loguer dans l'application et afficher ses ruchers et ruches.
- Les opérations CRUD sur un rucher et une ruche.
- Les opérations CRUD sur une activité.
- L'utilisateur peut afficher la liste des activités pour une année spécifique.
- Les activités concernant un rucher ou une ruche sont affichées dans les détails du rucher ou de la ruche

Je vais donc concentrer mes tests sur tous ces points.

En premier lieu, je vais tester l'API en testant chaque Endpoint et en vérifiant que l'on obtient le résultat attendu. Le teste de tous les Endpoint de l'API va me permettre de déterminer la validité des CRUD et de l'authentification.

Il va également falloir tester l'interface, pour cela je vais décrire les étapes que l'utilisateur doit effectuer et décrire le résultat attendu, c'est-à-dire, décrire les informations qu'il devrait voir à l'écran.

## 3 Réalisation

### 3.1 Remplir la base de données

### 3.2 Tests

#### 3.2.1 API

Pour tester l'api, je vais définir l'Endpoint testé, la forme que doit avoir la requête et le résultat attendu.

#### Tests

Test N°	Endpoint	Header (Requête)	Body (Requête)	Résultat attendu
---------	----------	---------------------	-------------------	---------------------

#### Résultat

Test N°	Date	Résultat du test
---------	------	------------------

#### 3.2.2 Interface

Pour tester l'interface, je vais définir les actions que l'utilisateur doit effectuer et les informations qui devrait être affichées une fois les actions réalisées

## Tests

Test N°	Actions	Résultat attendu
---------	---------	------------------

## Résultats

Test N°	Date	Résultat du test
---------	------	------------------

## 4 Conclusion

### 4.1 État finale de l'application

### 4.2 Objectifs atteints

### 4.3 Objectifs non-atteints

### 4.4 Problèmes rencontrés

### 4.5 Amélioration possible

### 4.6 Bilan de la planification

### 4.7 Bilan personnel

## 5 Sources – Bibliographie

## 6 Glossaire

- **API (Application Programming Interface)** : L'API agit comme une interface permettant la communication entre un logiciel et un service
- **Backend** : Tout ce que l'utilisateur ne voit pas. C'est ce qui se passe dans les coulisses de l'application et fourni les fonctionnalités nécessaires à son fonctionnement
- **CRUD (Create Read Update Delete)** : L'acronyme CRUD définit les opérations de base sur les données stockées. Créer une donnée, la lire, la mettre à jour ou la supprimer
- **Endpoint** : Élément de l'interface d'une API qui permet la communication avec celle-ci. Représenté par une URL sur laquelle on peut y faire une requête HTTP.
- **Environnement d'exécution** : logiciel qui s'occupe de l'exécution d'un programme pour un langage de programmation donné
- **Framework** : Structure de code sur laquelle on va construire notre application
- **Frontend** : Tout ce que l'utilisateur voit. C'est ce avec quoi l'utilisateur va interagir pour bénéficier des fonctionnalités disposées par le backend

- **IDE** (Integrated **D**evelopment **E**nvironment) : un environnement de développement et une combinaison d'outils qui facilite la création de programme (éditeur de texte, débogueur, etc.)
- **Intégration** : assemblages des différents éléments qui constitue une application web (textes, API, images, vidéos, etc.)
- **Middlewares** : Un middleware est une fonction qui s'exécute entre la requête faites par l'utilisateur au serveur et le traitement final de la requête par le serveur
- **Migrations** : transitions de schéma de base de données. Les base de données doivent changer au fil du temps pour s'adapter aux nouvelles exigences, le changement de schémas (structure de la base de données) est une migration
- **ORM** (**O**bject **R**elational **M**apping) : interface entre la base de données et le langage de programmation (orienté objet). Simplifie la communication entre les deux
- **Route** : Une route et la combinaison entre une URL, une méthode HTTP (GET, POST, PUT, DELETE, etc.) et une fonction qui sera appelé lorsqu'une requête sera effectuée sur la route.
- **SGBD** (**S**ystème de **G**estion de **B**ase de **D**onnées) : logiciel permettant la gestion d'une base de données

## 7 Annexes

## 8 Résumé