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

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 lier
- 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 vue lors des modules ICT (104, 105, 1153).

Le code source doit être lisible et respecte 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 nourrissement. 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 rencontrer, 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 débugger plus simplement

2.3.2 Node JS

Node et 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

La maquette du site est fournie en annexe

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 créer 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-incrémente.

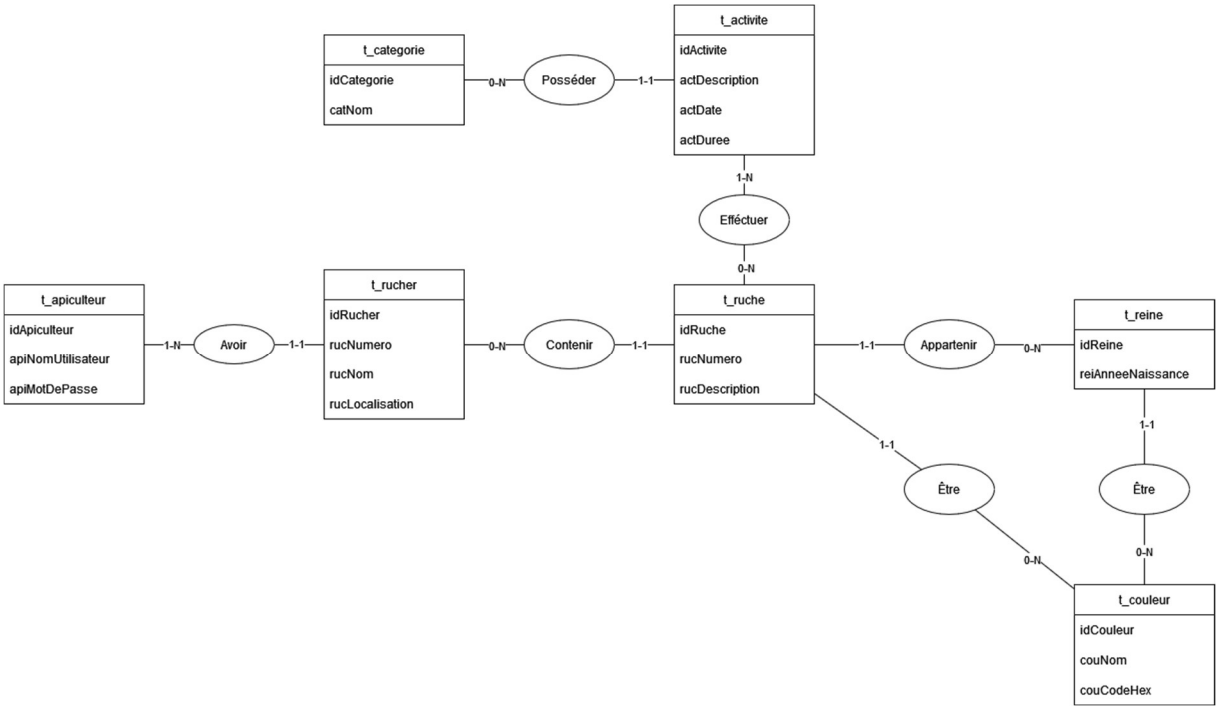
Tous les textes (sauf description) en base de données sont des varchar(255) laissant suffisamment de place ce qui va être stocké (nom d'utilisateur, mot de passe haché, etc.). Les descriptions sont stockées avec un type varchar(1000) laissant un bon paragraphe pour décrire quoi que ce soie

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.

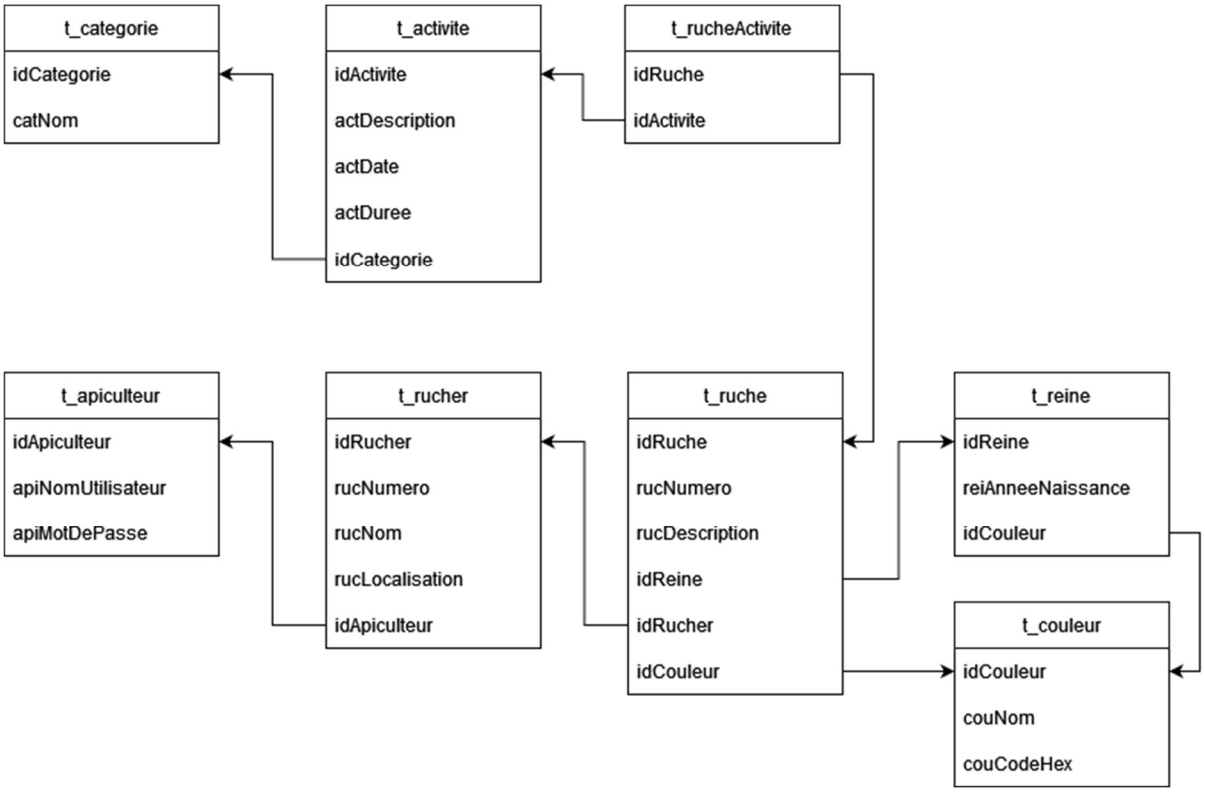
La date et la durée sont stocké avec leur type correspondant en MySQL (Date, Time).

Le MPD est réaliser avec Prisma (https://www.prisma.io/docs/orm/prisma-schema/overview) qui fournis ça 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 sont générée automatiquement par Prisma)

2.6.1 MCD



2.6.2 MLD


t_categorie
idCategorie
catNom
t_activite
idActivite
actDescription
actDate
actDuree
idCategorie
t_rucheActivite
idRuche
idActivite
t_apiculteur
idApiculteur
apiNomUtilisateur
apiMotDePasse
t_rucher
idRucher
rucNumero
rucNom
rucLocalisation
idApiculteur
t_ruche
idRuche
rucNumero
rucDescription
idReine
idRucher
idCouleur
t_reine
idReine
reiAnneeNaissance
idCouleur
t_couleur
idCouleur
couNom
couCodeHex

2.6.3 MPD

```
model t_categorie {
  idCategorie Int          @id @default(autoincrement()) @db.UnsignedInt
  catNom      String       @unique @db.VarChar(255)
  t_activite  t_activite[]
}

model t_activite {
  idActivite     Int         @id @default(autoincrement()) @db.UnsignedInt
  actDescription String      @db.VarChar(1000)
  actDate        DateTime    @db.Date()
  actDuree       DateTime    @db.Time()
  categorie      t_categorie @relation(fields: [fkCategorie], references: [idCategorie])
  fkCategorie    Int         @db.UnsignedInt
  ruches         t_ruche[]
}

model t_apiculteur {
  idApiculteur      Int        @id @default(autoincrement()) @db.UnsignedInt
  apiNomUtilisateur String     @unique @db.VarChar(255)
  apiMotDePasse     String     @db.VarChar(255)
  t_rucher          t_rucher[]
}

model t_rucher {
  idRucher        Int          @id @default(autoincrement()) @db.UnsignedInt
  rucNumero       Int          @unique @db.UnsignedInt
  rucNom          String       @db.VarChar(255)
  rucLocalisation String       @db.VarChar(255)
  apiculteur      t_apiculteur @relation(fields: [fkApiculteur], references: [idApiculteur])
  fkApiculteur    Int          @db.UnsignedInt
  t_ruche         t_ruche[]
}

model t_couleur {
  idCouleur  Int       @id @default(autoincrement()) @db.UnsignedInt
  couNom     String    @unique @db.VarChar(255)
  couCodeHex String    @db.VarChar(6)
  t_reine    t_reine[]
  t_ruche    t_ruche[]
}

model t_reine {
  idReine          Int       @id @default(autoincrement()) @db.UnsignedInt
  reiAnneNaissance Int       @db.Year
  couleur          t_couleur @relation(fields: [fkCouleur], references: [idCouleur])
  fkCouleur        Int       @db.UnsignedInt
  t_ruche          t_ruche[]
}

model t_ruche {
  idRuche        Int          @id @default(autoincrement()) @db.UnsignedInt
  rucNumero      Int          @unique @db.UnsignedInt
  rucDescription String       @db.VarChar(1000)
  reine          t_reine      @relation(fields: [fkReine], references: [idReine])
  fkReine        Int          @db.UnsignedInt
  rucher         t_rucher     @relation(fields: [fkRucher], references: [idRucher])
  fkRucher       Int          @db.UnsignedInt
  couleur        t_couleur    @relation(fields: [fkCouleur], references: [idCouleur])
  fkCouleur      Int          @db.UnsignedInt
  activites      t_activite[]
}
```

2.7 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 doits 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

5 Sources – Bibliographie

6 Glossaire

- **API** (**A**pplication **P**rogramming **I**nterface) : 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** (**C**reate **R**ead **U**pdate **D**elete) : L'acronyme CRUD définie les opérations de base sur les données stockées. Créer une donnée, la lire, la mettre à jour où 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 vas interagir pour bénéficier des fonctionnalités disposées par le backend

- **IDE** (**I**ntegrated **D**evelopment **E**nvironment) : un environnement de développement et une combinaison d'outils qui facilite la création de programme (éditeur de texte, débuggeur, 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 Résume