

Documentation technique

- Documentation technique
 - API REST
 - Arborescence
 - app/Models
 - app/DataAccessObject
 - app/Controllers
 - app/System
 - public
 - storage
 - bootstrap.php
 - Structure
 - Base de données
 - Requête de génération de planning
 - Explication
 - Test de performance
 - Tests unitaires Postman
 - Format de code
 - Syntaxe
 - Composer
 - Librairies
 - PHP dotenv
 - PHPMailer
 - Dompdf
 - Fichier iCalendar
 - Endpoints
 - Headers
 - Listes des endpoints
 - PWA
 - Vue
 - Arborescence
 - Composants
 - npm
 - Librairies
 - AlertifyJS
 - Axios
 - Vuex
 - Vue Router
 - BootstrapVue
 - Signature Pad
 - FullCalendar
 - Moment.js
 - reCAPTCHA
 - Tests Katalon Recorder

- [Table des figures](#)

API REST

Arborescence

```
api
├── app
│   ├── Models
│   ├── DataAccessObject
│   ├── Controllers
│   └── System
├── public
├── storage
├── vendor
├── .env
└── bootstrap.php
```

app/Models

Le dossier Models contient les modèles de l'API REST. Chaque modèle est une représentation objet de sa table de base de données correspondante. La création de ces modèles me permet d'utiliser les données de ma base de données de manière objet. Exemple de la classe **Dog** représentant la table **dog** de la base de données :

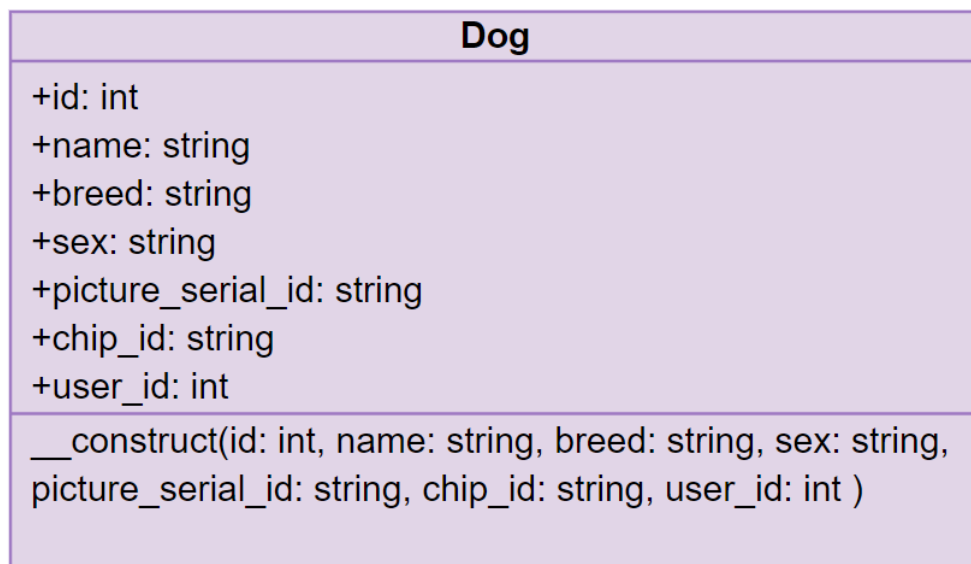
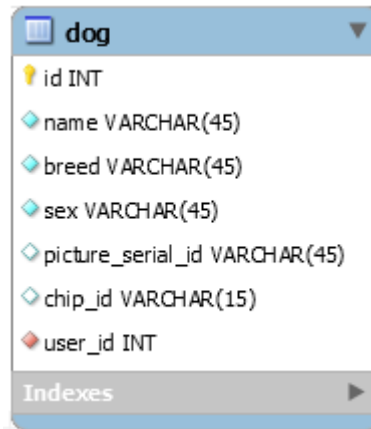


Fig.1 - Dog Model



The screenshot shows a database table named 'dog' with the following fields:

Field	Type
id	INT
name	VARCHAR(45)
breed	VARCHAR(45)
sex	VARCHAR(45)
picture_serial_id	VARCHAR(45)
chip_id	VARCHAR(15)
user_id	INT

Below the fields, there is a section for 'Indexes' with a right-pointing arrow.

Fig.2 - Table dog

app/DataAccessObject

Le dossier DataAccessObject contient les data access object (DAO) de l'API REST. Ces DAO contiennent toutes les méthodes permettant un CRUD sur sa table de base de données correspondante. Les méthodes des DAO fonctionnent de manière à créer ou récupérer des modèles afin de respecter un maximum la structure objet de l'API REST. Exemple de la classe **DAODog** :

DAODog
-db: PDO
<pre>__construct(db: PDO) +findAll(): Dog[] +find(id: int): Dog +findByUserId(userId: int): Dog +findBySerialId(serial_id: string): Dog +insert(dog: Dog): int +update(dog: Dog): int +delete(dog: Dog): int</pre>

Fig.3 - Data Access Object Dog

app/Controllers

Le dossier Controllers contient les contrôleurs de l'API REST. Comme leurs noms l'indiquent, le but des contrôleurs est de contrôler les différents cas d'utilisation et d'autorisation d'accès en utilisant, s'il le faut, les DAO afin de communiquer avec la base de données et en retournant les différents codes HTTP et messages en format JSON. Exemple de la classe **DogController** :

DogController
-DAODog: DAODog -DAOUser: DAOUser
__construct(db: PDO) +getAllDogs(): string +getDog(id: int): string +createDog(dog: Dog): string +updateDog(dog: Dog): string +deleteDog(id: int): string +uploadDogPicture(): string +downloadDogPicture(serial_id: string): string -validateDog(dog: Dog): bool

Fig.4 - Dog Controller

Dans ce dossier résident également les contrôleurs `ResponseController` et `HelperController`. Le `ResponseController` permet de retourner toutes les différentes réponses HTTP. Le `HelperController` permet l'utilisation de méthode dite d'aide et qui n'aurait pas leur place dans un contrôleur basique. Classes `ResponseController` et `HelperController` :

ResponseController	HelperController
notFoundAuthorizationHeader(): string «static» unauthorizedUser(): string «static» notFoundResponse(): string «static» successfulRequest(result: string): string «static» successfulRequestWithoutJson(result: string): string «static» unprocessableEntityResponse(): string «static» permanentScheduleAlreadyExist(): string «static» successfulCreatedRessource(): string «static» invalidDateFormat(): string «static» invalidTimeFormat(): string «static» dateOverlapProblem(): string «static» timeOverlapProblem(): string «static» chronologicalDateProblem(): string «static» invalidCodeDayFormat(): string «static» invalidDocumentTypeFormat(): string «static» invalidEmailFormat(): string «static» invalidLogin(): string «static» uploadFailed(): string «static» imageFileFormatProblem(): string «static» packageNumberFormatProblem(): string «static»	validateDateFormat(date: string): bool «static» validateTimeFormat(time: string): bool «static» validateCodeDayFormat(code_day: string): bool «static» validateChornologicalTime(firsttime: string, secondtime: string): bool «static» generateApiToken(): string «static» generateRandomString(): string «static» validateDocumentTypeFormat(document_type: string): bool «static» validateEmailFormat(email: string): bool «static» validatePackageNumber(package_number: int): bool «static» sendMail(message: string, emailRecipient: string): void «static» pngTojpegConverter(filename: string): void «static» storeConditionsRegistration(filename: string, package_number: int, date: string, signature_base64: string, userfirstname: string, userlastname: string): void «static» getDefaultDirectory(): string «static»

Fig.5 - ResponseController et HelperController

app/System

Le dossier System contient la classe **DatabaseConnector** qui permet la connexion à la base de données en récupérant les variables d'environnements *PHP dotenv* et la classe **Constants** permettant de stocker les différentes constantes de l'API REST. Classes **DatabaseConnector** et **Constants** :

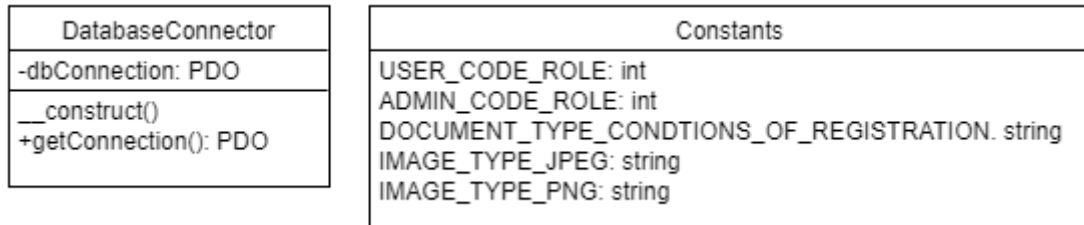


Fig.6 - Classes DatabaseConnector et Constants

public

Le dossier public contient les différents fichiers d'entrée de l'API REST. Les fichiers d'entrée récupèrent le **verb HTTP** d'une requête HTTP afin de pouvoir exécuter les bonnes méthodes des contrôleurs. Ces fichiers s'occupent également d'attribuer les headers et le body si nécessaire.

storage

Dossier contenant les différents fichiers uploadés de l'API REST, comme les documents PDF ou les photos de chiens par exemple.

bootstrap.php

Fichier de bootage de l'API REST inclus dans tous les fichiers d'entrée, celui-ci permet de :

- Charger les dépendances PHP du dossier vendor
- Charger les variables d'environnements
- Créer la connexion à la base de données

Structure

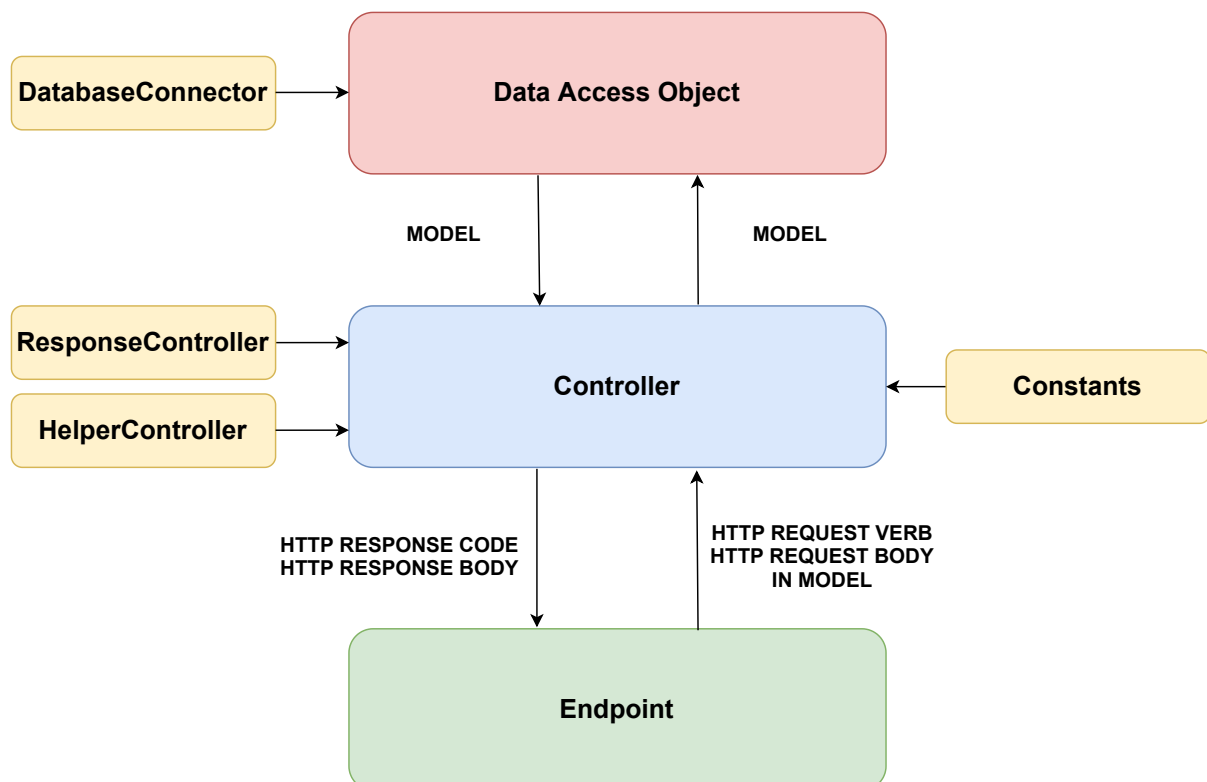


Fig.7 - Structure de l'API REST

Base de données

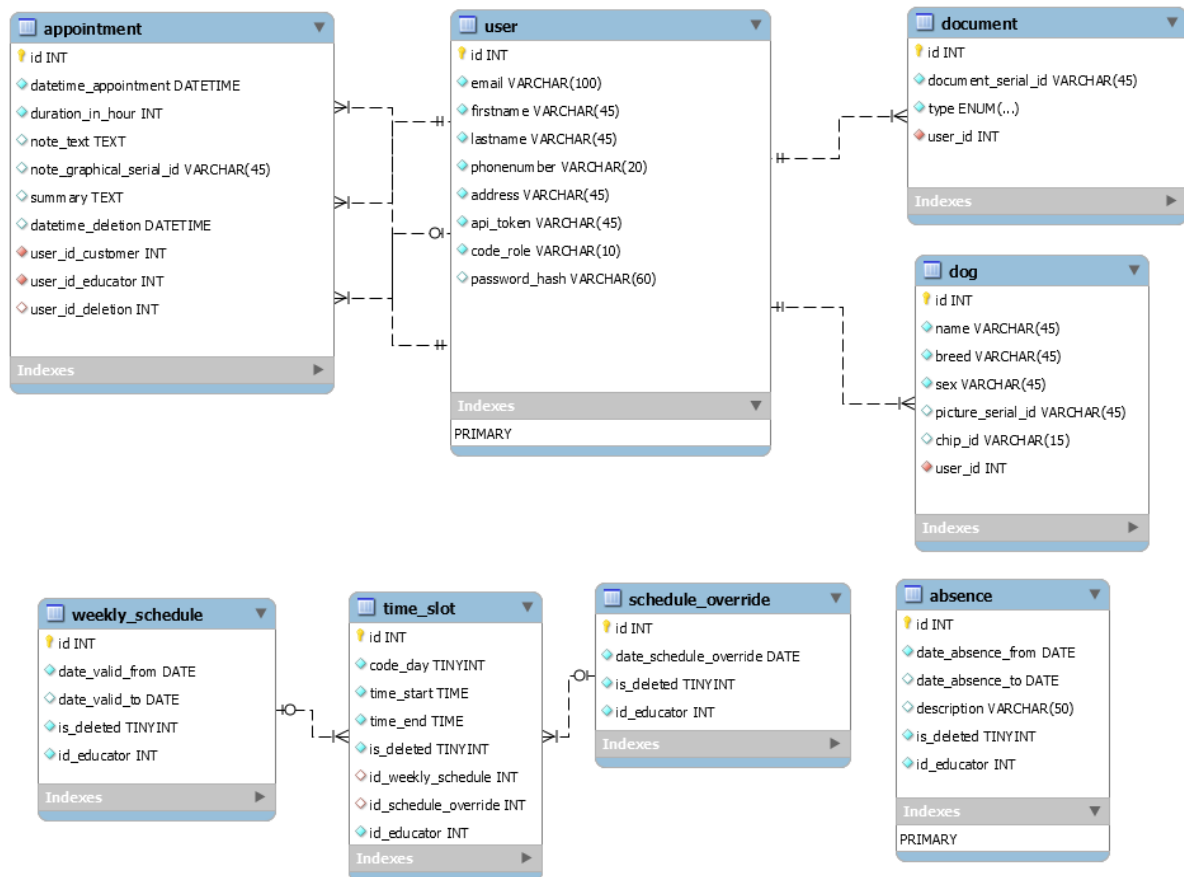


Fig.8 - Base de de données de l'API REST

La base de données que j'ai développée et utilisée se décompose en deux parties. La partie supérieure, donc les tables : `appointment`, `user`, `document` et `dog` concernent toutes les données en lien avec les clients de l'application. Tandis que la partie inférieure, donc les tables : `weekly_schedule`, `time_slot`, `schedule_override` et `absence` concernent les données de planning des éducateurs canins de la société.

La table **appointment** contient les informations des rendez-vous entre un éducateur canin et un client :

appointment

NOM	TYPE	NOT NULL	DESCRIPTION
datetime_appointment	DATETIME	X	la date ainsi que l'heure du rendez-vous entre un client et un éducateur canin.
duration_in_hour	INT	X	La durée en heure du rendez-vous.
note_text	TEXT		Les notes textuelles du rendez-vous.
note_graphical_serial_id	VARCHAR		L'identifiant de série de la note graphique uploadé sur le serveur du rendez-vous.
summary	TEXT		Le résumé d'un rendez-vous.
datetime_deletion	DATETIME		La date ainsi que l'heure de la suppression du rendez-vous.

user_id_customer	INT	X	L'identifiant de l'utilisateur de type client du rendez-vous.
user_id_educator	INT	X	L'identifiant de l'utilisateur de type éducateur canin du rendez-vous.
user_id_deletion	INT		L'identifiant de l'utilisateur ayant supprimé le rendez-vous.

Remarques

Les données des champs `datetime_deletion` et `user_id_deletion` sont uniquement ajoutées lors de la suppression non définitive du rendez-vous par un éducateur canin ou un client.

La table `user` contient les informations des différents utilisateurs :

user			
NOM	TYPE	NOT NULL	DESCRIPTION
email	VARCHAR	X	L'adresse e-mail de l'utilisateur.
firstname	VARCHAR	X	Le prénom de l'utilisateur.
lastname	VARCHAR	X	Le nom de famille de l'utilisateur.
phonenummer	VARCHAR	X	Le numéro de téléphone de l'utilisateur.
address	VARCHAR	X	L'adresse du domicile de l'utilisateur.
api_token	VARCHAR	X	Le token d'authentification de l'utilisateur.
code_role	VARCHAR	X	Le code du rôle de l'utilisateur (client, administrateur, autre).
password_hash	VARCHAR		Le hash du mot de passe de l'utilisateur.

Remarques

La table ne contient pas de sel pour le mot de passe, car depuis PHP 7.0.0, il est conseillé d'utiliser le sel généré par défaut.

La table `document` contient les informations des documents des utilisateurs :

document			
NOM	TYPE	NOT NULL	DESCRIPTION
document_serial_id	VARCHAR	X	L'identifiant de série du document uploadé sur le serveur.
type	ENUM	X	Le type du document (conditions d'inscription, poster, autre).
user_id	INT	X	L'identifiant de l'utilisateur propriétaire du document.

La table **dog** contient les informations des chiens des utilisateurs :

dog			
NOM	TYPE	NOT NULL	DESCRIPTION
name	VARCHAR	X	Le nom du chien.
breed	VARCHAR	X	La race du chien.
sex	VARCHAR	X	Le sexe du chien.
picture_serial_id	VARCHAR	X	L'identifiant de série de la photo du chien uploadé sur le serveur.
chip_id	VARCHAR		L'identifiant de la puce sous-cutanée du chien.
user_id	INT	X	L'identifiant de l'utilisateur propriétaire du chien.

La table **weekly_schedule** contient les informations des calendriers hebdomadaires des éducateurs canins :

weekly_schedule			
NOM	TYPE	NOT NULL	DESCRIPTION
date_valid_from	DATE	X	La date de début du calendrier hebdomadaire.
date_valid_to	DATE		La date de fin du calendrier hebdomadaire.
is_deleted	TINYINT	X	Le statut du calendrier hebdomadaire.
id_educator	INT	X	L'identifiant de l'éducateur canin propriétaire du calendrier hebdomadaire.

Remarques

Le système s'assure qu'il n'y ait pas de chevauchement entre les différentes lignes non supprimées de cette table (**date_valid_from** et **date_valid_to**) appartenant au même éducateur canin. Une seule ligne non supprimée d'un éducateur canin peut avoir comme valeur **null** le champ **date_valid_to**. Cette ligne correspondra donc à l'unique calendrier permanent d'un éducateur canin.

La table **schedule_override** contient les informations des exceptions d'horaires des éducateurs canins :

schedule_override			
NOM	TYPE	NOT NULL	DESCRIPTION
date_schedule_override	DATE	X	La date de l'exception d'horaire.
is_deleted	TINYINT	X	Le statut de l'exception d'horaire.
id_educator	INT	X	L'identifiant de l'éducateur canin propriétaire de

l'exception d'horaire.

Remarques

Le système s'assure qu'il n'y ait pas deux fois la même date entre les différentes lignes non supprimées de cette table (`date_schedule_override`) appartenant au même éducateur canin.

La table `time_slot` contient les informations des créneaux horaires des éducateurs canins :

time_slot			
NOM	TYPE	NOT NULL	DESCRIPTION
code_day	TINYINT	X	Le code correspondant à un jour de la semaine du créneau horaire.
time_start	TIME	X	L'heure de début du créneau horaire.
time_end	TIME	X	L'heure de fin du créneau horaire.
is_deleted	TINYINT	X	Le statut du créneau horaire.
id_weekly_schedule	INT		L'identifiant du calendrier hebdomadaire.
id_schedule_override	INT		L'identifiant de l'exception d'horaire.
id_educator	VARCHAR	X	L'identifiant de l'éducateur canin propriétaire du créneau horaire.

Remarques

Le système s'assure qu'il n'y ait pas de chevauchement entre les différentes lignes non supprimées de cette table (`time_start` et `time_end`) pour le même parent (`id_weekly_schedule` ou `id_schedule_override`) appartenant au même éducateur canin. Le système s'assure qu'un créneau horaire appartienne à un calendrier hebdomadaire OU à une exception d'horaire, il ne doit pas appartenir aux deux ni à aucun.

La table `absence` contient les informations des vacances des éducateurs canins :

absence			
NOM	TYPE	NOT NULL	DESCRIPTION
date_absence_from	DATE	X	La date de début des vacances.
date_absence_to	DATE		La date de fin des vacances.
description	VARCHAR		La description des vacances.
is_deleted	TINYINT	X	Le statut des vacances.
id_educator	INT	X	L'identifiant de l'éducateur canin propriétaire des vacances.

Remarques

La valeur du champ `date_absence_to` peut être `null` si le service est suspendu pour un moment.

Requête de génération de planning

Explication

Une des fonctionnalités les plus importantes de mon travail de diplôme est la génération de planning pour un éducateur canin. En effet, j'ai développé des tables de ma base de données en sorte de permettre aux éducateurs canins de la société d'éditer leurs propres horaires de la manière la plus flexible possible. Les éducateurs canins peuvent créer des calendriers hebdomadaires (weekly schedules) ayant une date de début et une date de fin (à noter qu'un seul et unique calendrier hebdomadaire peut ne pas avoir de date de fin afin de le rendre permanent sur une durée de une année suivant le jour actuel). Chaque calendrier hebdomadaire peut avoir des créneaux horaire (time slots) afin de spécifier les données hebdomadaires de celui-ci.

En plus de cela, les éducateurs canins peuvent créer des exceptions d'horaire (schedule override) afin d'écraser les potentiels créneaux horaires d'un calendrier hebdomadaire d'une journée par les siens. Donc, similaire aux calendriers hebdomadaires, l'exception d'horaire peut également avoir des créneaux horaires.















































Pour finir, les éducateur canins ont la possibilité de créer des plages de vacances qui devront permettre d'ignorer tous les créneaux horaire inclus dans celles-ci.

Afin d'être plus clair, imaginons les données suivantes :

- Calendrier hebdomadaire du 31 mai au 04 juillet
 - Deux créneau horaire tous les lundis
 - Deux créneau horaire tous les mercredis
 - Deux créneau horaire tous les jeudis
 - Deux créneau horaire tous les vendredis
- Exception d'horaire le 03 juin
 - Un créneau horaire dans la journée
- Exception d'horaire le 08 juin
 - Deux créneaux horaires dans la journée
- Exception d'horaire le 19 juin
 - Un créneau horaire dans la journée
- Exception d'horaire le 02 juillet
 - Un créneau horaire dans la journée
- Vacances du 28 juin au 04 juillet

Le résultat sans le traitement ressemblerais à cela :

Juin 2021


















Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
31  	1	2  	3   	4  	5	6
7  	8  	9   	10  	11  	12	13
14  	15	16  	17  	18  	19 	20
21  	22	23  	24  	25  	26	27
28  	29	30  	1  	2   	3	4

 Créneau horaire d'un calendrier hebdomadaire
  Créneau horaire d'une exception d'horaire
  Vacances

Fig.9 - Planning sans traitement

Le résultat avec le traitement réalisé dans ma requête ressemblerais à cela :

Juin 2021

Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
31  	1	2  	3 	4  	5	6
7  	8 	9 	10  	11  	12	13
14  	15	16  	17  	18  	19 	20
21  	22	23  	24  	25  	26	27
28	29	30	1	2	3	4

 Créneau horaire d'un calendrier hebdomadaire
  Créneau horaire d'une exception d'horaire

Fig.10 - Planning avec traitement

Test de performance

Afin de réaliser cette fonctionnalité, j'ai développé durant mon travail de diplôme, une requête que je pourrais considérer de touffu. En effet, afin d'arriver à ses fins, ma requête contient trois sous-requêtes et interpelle un total de six tables différentes. J'ai alors voulu vérifier sa capacité à s'adapter à un nombre conséquent de données en réalisant un test de performance de celle-ci. Pour réaliser ce test de performance, j'ai ajouté les données suivantes à l'aide d'un script dans ma base de données :

- 100 000 utilisateurs
- Pour chacun de ces utilisateurs, création d'un calendrier hebdomadaire avec 10 créneaux horaires par jour pour le mois de juillet (2021-07-01 au 2021-07-31)
- Pour chacun de ces utilisateurs, création d'une exception d'horaire avec 5 créneaux horaires le 15 juillet (2021-07-15)
- Pour chacun de ces utilisateurs, création de vacances pour la dernière semaine de juillet (2021-07-26 au 2021-07-31)
- Pour chacun de ces utilisateurs, création d'un rendez-vous (2021-07-15 08:00:00)

Une fois les données insérées, j'ai alors exécuté ma requête de génération de planning (que vous pouvez retrouver dans le [DAOTimeSlot](#)) dans l'outil Workbench afin d'y générer un **Visual Explain Plan**. Un **Visual Explain Plan** est une représentation du traitement effectué sur une base de données d'une requête SQL. Voici le **Visual Explain Plan** généré :

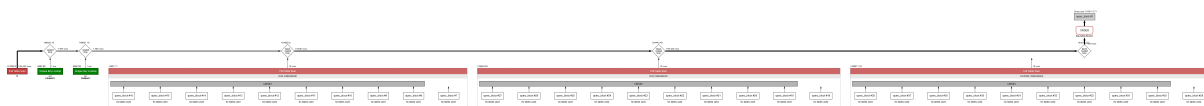


Fig.11 - Visual Explain Plan de la requête de génération de planning

Avec ces données, ma requête a pris environ 18 minutes afin de générer le planning d'un des éducateurs canins. À l'heure actuelle cette requête n'est pas prête à traiter une grande quantité de données mais je compte, dans le futur, l'optimiser voir la modifier afin que celle-ci soit utilisable avec beaucoup de données.

Tests unitaires Postman

Afin de tester l'API REST, j'ai utilisé l'outil Postman qui m'a permis d'exécuter des scripts de test pour chaque endpoint de mon API REST. Ces tests sont réalisables en JavaScript en utilisant la bibliothèque proposée par Postman [pm](#). Tous les tests unitaires de mon API REST sont identifiables grâce à un code qui leur est propre dans l'annexe [postman_unit_tests.md](#).

Format de code

[U S E - G A 1]

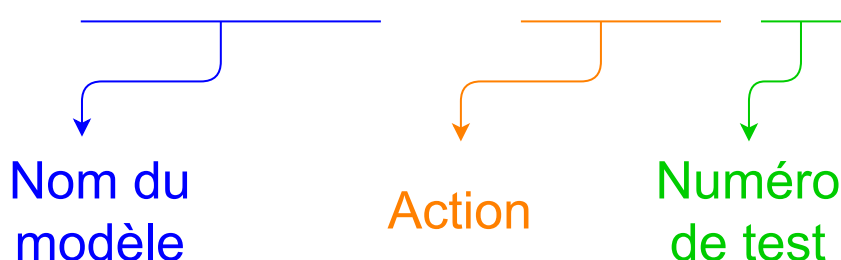


Fig.12 - Format de code des tests unitaires

Définition

Code des noms de modèles

Modèle	CODE
User	USE
Dog	DOG
Document	DOC
Absence	ABS
WeeklySchedule	WEE
ScheduleOverride	SCH
TimeSlot	TIM
Appointment	APP

Code des actions

Action	CODE
Get all	GA
Get one	GO
Create one	CO
Update one	UO
Delete one	DO
Connection	C
Get user authenticated	GUA
Update password of the authenticated user	UPAU
Upload dog picture	UDP
Download dog picture	DDP
Download document	DD
Upload note graphical	UNG
Download note graphical	DNG

Syntaxe

Postman propose à ses utilisateurs la possibilité d'écrire des scripts de test pour tester les différentes requêtes de leurs API. Afin de rédiger ces tests, j'ai dû utiliser une certaine syntaxe JavaScript proposée par Postman. Lors de la réalisation de mes tests unitaires avec Postman, j'ai utilisé la fonction `pm.test` afin de tester tous les cas d'utilisation et d'exception de mon API REST. La fonction `pm.test` prend en premier paramètre une chaîne

de caractères qui apparaîtra dans la sortie des résultats du test afin d'identifier le test. Le deuxième paramètre est une fonction qui doit retourner un booléen pour indiquer si le test a réussi ou échoué.

Dans tous mes tests unitaires, j'ai utilisé la fonction `pm.test` avec la fonction `pm.response.to.have.status(status)` afin de tester si le cas d'utilisation de l'endpoint en question retournait le bon code HTTP. En effet, cette ligne retourne `true` si le code d'état de la réponse est identique à son paramètre de fonction et `false` si ce n'est pas le cas. Par exemple, afin de tester que l'endpoint de création d'utilisateur `POST api/v1/users` retourne bien le code HTTP `201`, j'ai réalisé le test :

```
pm.test("Right code for successful created ressource", function () {  
  pm.response.to.have.status(201);  
});
```

Pour les endpoints nécessitant de retourner un message, j'ai également testé que celui-ci soit correct en utilisant la fonction `pm.expect(element1).to.eql(element2)` qui permet de tester que le premier élément corresponde bien au second. Par exemple, afin de tester que l'endpoint de création d'utilisateur `POST api/v1/users` avec une adresse e-mail ne respectant pas le bon format retourne bien le code HTTP `400` ainsi que le message d'erreur `Format d'adresse email invalide.`, j'ai réalisé les tests :

```
pm.test("Right code for invalid email format", function () {  
  pm.response.to.have.status(400);  
});  
pm.test("Right message for for invalid email format", function () {  
  const responseJson = pm.response.json();  
  pm.expect(responseJson.error).to.eql("Format d'adresse email invalide.");  
});
```

Pour les tests unitaires nécessitant le test de plusieurs données de corps de requêtes différentes, j'ai utilisé la fonctionnalité d'importation de CSV proposée par Postman. En effet, Postman permet de sélectionner un fichier CSV contenant différentes données afin de procéder à plusieurs itérations de test avec celles-ci. Par exemple, afin de tester que l'endpoint de création de vacances `POST api/v1/absences` avec une date ne respectant pas le bon format retourne bien le code HTTP `400` ainsi que le message d'erreur `Format de date invalide => (YYYY-MM-DD).`, j'ai réalisé les tests :

```
pm.test("Right code for invalid date_to format", function () {  
  pm.response.to.have.status(400);  
});  
pm.test("Right message for invalid date_to format", function () {  
  const responseJson = pm.response.json();  
  pm.expect(responseJson.error).to.eql("Format de date invalide => (YYYY-MM-DD).");  
});
```

Ce test est réalisé avec les différentes données du fichier CSV disponibles à la fin de l'annexe `unit_tests.md`.

Pour les tests unitaires nécessitant un test sur l'en-tête d'autorisation, j'ai utilisé la fonction `pm.request.to.have.header(header)`. Par exemple, afin de tester que l'endpoint de suppression d'un document `DELETE api/v1/documents` avec un api token appartenant à un client retourne bien le code HTTP `403` ainsi que le message d'erreur `Vous n'avez pas les permissions.`, j'ai réalisé les tests :

```
pm.test("Authorization header is present", () => {
  pm.request.to.have.header("Authorization");
});
pm.test("Authorization header is false", function () {
  pm.response.to.have.status(403);
});
pm.test("Right message for access without permission", function () {
  const responseJson = pm.response.json();
  pm.expect(responseJson.error).to.eql("Vous n'avez pas les permissions.");
});
```

Pour les tests unitaires retournant une réponse JSON spécifique, j'ai effectué un test afin de vérifier que sa structure soit respectée avec la fonction `pm.response.to.have.jsonSchema(jsonSchema)`. Par exemple, afin de tester que l'endpoint de récupération d'un chien spécifique `GET api/v1/dogs/{dogId}` avec un api token appartenant à un éducateur canin retourne bien le code HTTP `200` ainsi que la réponse JSON au bon format, j'ai réalisé les tests :

```
pm.test("Authorization header is present", () => {
  pm.request.to.have.header("Authorization");
});
pm.test("Authorization header is right", function () {
  pm.response.to.have.status(200);
});
pm.test("The data structure of the response is correct", () => {
  pm.response.to.have.jsonSchema({
    "type": "object",
    "properties": {
      "id" : {"type" : "integer"},
      "name" : {"type" : "string"},
      "breed" : {"type" : "string"},
      "sex" : {"type" : "string"},
      "picture_serial_id" : {"type" : ["string", "null"]},
      "chip_id" : {"type" : ["string", "null"]},
      "user_id" : {"type" : "integer"}
    },
    "required":
    ["id", "name", "breed", "sex", "picture_serial_id", "chip_id", "user_id"]
  })
});
```

Composer

Composer permet de gérer les dépendances PHP de mon API REST. En premier lieu, Composer permet de générer un fichier nommé `composer.json`. Ce fichier est un moyen pour Composer de rechercher les différentes dépendances que mon projet PHP doit télécharger. Le fichier `composer.json` vérifie également la compatibilité des versions des dépendances de mon projet, c'est-à-dire que si j'utilise un paquet obsolète, Composer me le fera savoir afin d'éviter tout problème. Afin d'installer un paquet comme Dompdf, j'ai dû exécuter la commande suivante dans mon invite de commandes :

```
$ composer require dompdf/dompdf
```

Après avoir exécuté ce type de commande, mon projet contient maintenant les fichiers `composer.json` et `composer.lock` ainsi que le dossier `vendor`. Comme expliqué auparavant, `composer.json` est comme un guide correspondant aux versions de dépendance que Composer **doit** installer tandis que `composer.lock` est une représentation exacte des versions de dépendance qui **ont** été installées. Le dossier `vendor` quant à lui, contient tous les paquets et dépendances installés du projet.

Une fois nos paquets et dépendances installés, il faut maintenant pouvoir les inclure dans un de nos scripts PHP. Pour ce faire, Composer nous facilite la tâche en générant un fichier de chargement automatique nommé `autoload.php`. En incluant ce fichier dans mon fichier `bootstrap.php` qui lui-même étant inclus dans chaque points d'entrée de mon API REST, je peux accéder quand je le souhaite aux différents paquets et dépendances de mon projet. Par exemple, si je souhaite utiliser le paquet Dompdf dans un de mes scripts PHP, il me suffit d'écrire la ligne `use Dompdf\Dompdf` afin d'y accéder.

Librairies

PHP dotenv

Installé avec la commande :

```
$ composer require vlucas/phpdotenv
```

J'ai utilisé Dotenv afin de créer et charger des variables d'environnement accessibles par l'intégralité de mon API REST. Son fonctionnement se déroule de la manière suivante :

Création d'un fichier `.env` à la racine du projet contenant les variables d'environnement de l'application.

```
DB_HOST= "dbhost"
DB_PORT= "3306"
DB_DATABASE= "database"
DB_USERNAME= "username"
DB_PASSWORD= "password"

SMTP_HOST= "smtphost"
SMTP_USERNAME= "smtpusername"
SMTP_PASSWORD= "smtppassword"

GOOGLE_RECAPTCHA_SECRET_KEY = "secretKey"
```


Chargement de ces variables d'environnement dans le fichier `bootstrap.php`.

```
...  
use Dotenv\Dotenv;  
$dotenv = new Dotenv(__DIR__);  
$dotenv->load();  
...
```

Utilisation de ces variables d'environnement de la manière suivante :

```
$host = getenv('DB_HOST');
```

[Documentation de Php dotenv](#)

PHPMailer

Installé avec la commande :

```
$ composer require phpmailer/phpmailer
```

J'ai utilisé PHPMailer afin d'envoyer des e-mails depuis le contrôleur `HelperController` de la manière suivante :

```
public static function sendMail(string $message, string $subject, string  
$emailRecipient, string $attachmentFilePath = null)  
{  
    // Création d'une instance de la classe PHPMailer en activant les exceptions  
    $mail = new PHPMailer(true);  
  
    // Chargement du template avec le sujet et le message de l'e-mail  
    $body = HelperController::loadMailTemplate($subject, $message);  
    try {  
        // Paramètres du serveur  
        $mail->SMTPDebug = SMTP::DEBUG_SERVER;  
        $mail->Host = getenv('SMTP_HOST');  
        $mail->SMTPAuth = true;  
        $mail->Username = getenv('SMTP_USERNAME');  
        $mail->Password = getenv('SMTP_PASSWORD');  
        $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;  
        $mail->Port = 587;  
  
        // Destinataire  
        $mail->setFrom('noreply@douceurdechien.com', 'Douceur de Chien');  
        $mail->addAddress($emailRecipient);  
    }
```

```
//Pièce jointe
if (!is_null($attachmentFilePath)) {
    $mail->addAttachment($attachmentFilePath);
}

//Contenu
$mail->isHTML(true);
$mail->CharSet = 'UTF-8';
$mail->Encoding = 'base64';
$mail->Subject = $subject;
$mail->AddEmbeddedImage("../resources/image/logo_douceur_de_chien.png", "logo-image", "logo_douceur_de_chien.png");
$mail->Body = $body;

$mail->send();
} catch (Exception $e) {
    echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}
}
```

[Documentation de PHPMailer](#)

Dompdf

Installé avec la commande :

```
$ composer require dompdf/dompdf
```

J'ai utilisé Dompdf afin de créer les conditions d'inscription depuis le contrôleur `HelperController` de la manière suivante :

```
public static function storeConditionsRegistration(string $filename,int
$package_number,string $date, string $signature_base64,string $userfirstname,
string $userlastname)
{
    $dompdf = new DOMPDF();

    ob_start();
    include
HelperController::getDefaultDirectory()."resources/template/conditions_registratio
n.php";
    $contents = ob_get_clean();

    $dompdf->loadHtml($contents);
    dompdf->render();
    $output = $dompdf->output();

    file_put_contents(HelperController::getDefaultDirectory()."storage/app/conditions_
```

```
registration/".$filename.".pdf", $output);  
}
```

1. Création d'une instance de la classe **DOMPDF**
2. Début de la temporisation de sortie permettant d'inclure les différentes données dans le template HTML et CSS **conditions_registration.php**
3. Récupération du contenu HTML du tampon de sortie et fermeture de sa session
4. Chargement du contenu HTML avec l'instance **DOMPDF**
5. Transformation du HTML en PDF
6. Récupération du PDF sous forme de données
7. Écriture de ces données dans un fichier PDF stocké sur le serveur

[Documentation de Dompdf](#)

Fichier iCalendar

Lors de la création d'un rendez-vous entre un client et un éducateur canin, mon API REST envoie un e-mail au client afin de lui fournir les informations du rendez-vous. En plus des informations du rendez-vous, j'ai créé une fonctionnalité permettant de générer un fichier ICS afin de l'inclure dans l'e-mail. Un fichier ICS est un format de fichier pour iCalendar. Ces fichiers ayant comme extension **.ics** permettent d'importer dans un calendrier des données de calendrier. Ce format étant une norme internationale, de nombreux calendriers numériques tels que les calendriers de Microsoft, Google et Apple sont capables de supporter ce format de fichier.

Pour générer ce fichier, je procède de la même manière que la création des conditions d'inscription avec Dompdf. C'est-à-dire qu'une fois la création du rendez-vous effectué, je vais effectuer les étapes suivantes :

```
public static function sendMailWithICSFile($start_datetime, $end_datetime,  
$educator_fullname, $customer_email , $filename)  
{  
    ob_start();  
    include  
HelperController::getDefaultDirectory()."resources/template/iCalendar_appointment.  
php";  
    $contents = ob_get_clean();  
  
    $temp = tmpfile();  
    fwrite($temp, $contents);  
  
    $tmpfile_path = stream_get_meta_data($temp)['uri'];  
    $new_tmpfile_path = sys_get_temp_dir(). '\\'. $filename;  
    rename($tmpfile_path, $new_tmpfile_path);  
  
    HelperController::sendMail("Message de l'e-mail", "Sujet de l'e-  
mail", $customer_email, null, $new_tmpfile_path);  
  
    unlink($new_tmpfile_path);  
}
```

1. Début de la temporisation de sortie permettant d'inclure les différentes données dans le template PHP

`iCalendar_appointment.php` :

```
header('Content-type: text/calendar; charset=utf-8');
header('Content-Disposition: attachment; filename=' . $filename);

$now = new DateTime('NOW');

echo "BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//douceurdechien/handcal//NONSGML v1.0//FR
CALSCALE:GREGORIAN
METHOD:PUBLISH
BEGIN:VEVENT
UID:" . md5(time()) . "
DTSTAMP;TZID=/Europe/Berlin:" . gmdate("Ymd\THis",$now->getTimestamp() + 60*60*2)
. "
DTSTART;TZID=/Europe/Berlin:".gmdate("Ymd\THis",$start_datetime->getTimestamp() +
60*60*2). "
DTEND;TZID=/Europe/Berlin:".gmdate("Ymd\THis",$end_datetime->getTimestamp() +
60*60*2). "
SUMMARY:Rendez-vous Douceur de Chien
LOCATION:701 Avenue de la Bigorre, 31210 Montréjeau, France
ORGANIZER:MAILTO:douceurdechien@douceurdechien.com
END:VEVENT
END:VCALENDAR";
```

1. Récupération du contenu du tampon de sortie et fermeture de sa session
2. Écriture du contenu dans un fichier temporaire
3. Renommage du fichier temporaire
4. Envoi d'un e-mail avec le fichier temporaire au format iCalendar
5. Suppression du fichier temporaire

[Informations sur le format iCalendar \(RFC2445\)](#)

Endpoints

Headers

Les en-têtes que j'ai utilisés dans les différents points d'entrée de mon API REST sont :

- Access-Control-Allow-Origin
- Content-Type
- Access-Control-Allow-Methods
- Access-Control-Max-Age
- Access-Control-Allow-Headers
- Content-Length

Listes des endpoints

- POST api/v1/users
 - POST api/v1/connection
 - GET api/v1/users
 - GET api/v1/users/educators
 - GET api/v1/users/{idUser}
 - GET api/v1/users/me
 - PATCH api/v1/users/{idUser}
 - PATCH api/v1/users/me/changePassword
 - DELETE api/v1/users/{idUser}
-

- POST api/v1/dogs
 - GET api/v1/dogs
 - GET api/v1/dogs/{idDog}
 - PATCH api/v1/dogs/{idDog}
 - DELETE api/v1/dogs/{idDog}
 - POST api/v1/dogs/uploadPicture
 - GET api/v1/dogs/downloadPicture/{serial_number}
-

- POST api/v1/documents
 - GET api/v1/documents
 - GET api/v1/documents/{idDocument}
 - PATCH api/v1/documents/{idDocument}
 - DELETE api/v1/documents/{idDocument}
 - GET api/v1/documents/downloadDocument/{serial_number}
-

- POST api/v1/absences
 - GET api/v1/absences
 - GET api/v1/absences/{idAbsence}
 - PATCH api/v1/absences/{idAbsence}
 - DELETE api/v1/absences/{idAbsence}
-

- POST api/v1/weeklySchedules
 - GET api/v1/weeklySchedules
 - GET api/v1/weeklySchedules/{idWeeklySchedule}
 - DELETE api/v1/weeklySchedules/{idWeeklySchedule}
-

- POST api/v1/scheduleOverrides
 - GET api/v1/scheduleOverrides
 - GET api/v1/scheduleOverrides/{idScheduleOverride}
 - DELETE api/v1/scheduleOverrides/{idScheduleOverride}
-

- POST api/v1/timeSlots
- GET api/v1/timeSlots
- GET api/v1/timeSlots/{idTimeSlot}

- `DELETE api/v1/timeSlots/{idTimeSlot}`

-
- `POST api/v1/appointments`
 - `GET api/v1/appointments`
 - `GET api/v1/appointments/{idTimeSlot}`
 - `PATCH api/v1/appointments/{idTimeSlot}`
 - `DELETE api/v1/appointments/{idTimeSlot}`
 - `POST api/v1/appointments/uploadNoteGraphical`
 - `POST api/v1/appointments/downloadNoteGraphical/{serial_number}`
 - `GET api/v1/plannings/{idEducator}`

Pour plus d'informations, rendez-vous dans l'annexe `endpoints.md`.

PWA

Vue

Arborescence

```
pwa
├── node_modules
├── public
├── src
│   ├── assets
│   │   ├── css
│   │   └── img
│   ├── components
│   │   ├── About.vue
│   │   ├── Administration.vue
│   │   └── ...
│   ├── plugins
│   │   └── bootstrap-vue.js
│   ├── Router
│   │   └── index.js
│   ├── App.vue
│   ├── main.js
│   ├── store.js
│   └── constants.js
```

Composants

Pour réaliser ma SPA (Single page application) Vue de mon travail de diplôme, j'ai réalisé différents composants. La majorité des composants que j'ai développés représentent chacun une des pages de mon application. Par exemple, la page d'accueil de mon application est le composant `home.vue`, la page de connexion de l'application est le composant `connection.vue`, etc.

En plus des composants représentant chacun une des pages de mon application, j'ai essayé de réaliser quelques composants permettant l'intégration d'une fonctionnalité ou d'un affichage redondant.

Un composant Vue est organisé de la manière suivante :

- Une partie HTML entre les balises `<template>` `</template>`
- Une partie JavaScript entre les balises `<script>` `</script>`
- Une partie CSS entre les balises `<style>` `</style>`

Pour la partie HTML, Vue permet l'utilisation de différents outils très pratiques permettant de faciliter l'élaboration de code HTML avec du JavaScript. Le premier outil est la syntaxe dite "Mustache" (les doubles accolades). Exemple :

```
<h1>Rendez-vous du client {{ lastname }}</h1>
```

Cette syntaxe permet une liaison de données entre le JavaScript et le HTML. Par rapport à cet exemple, la balise "Mustache" sera remplacée par la valeur de la propriété Vue `lastname` et elle sera également mise à jour à chaque fois que la propriété `lastname` subira une modification.

Le deuxième outil que j'ai utilisé sont les directives proposées par Vue : `v-if`, `v-for`, `v-bind`, `v-model` et `v-on`.

`v-if` et `v-for` permettent de réaliser des tests et des boucles dans le code HTML avec des propriétés Vue. Exemple :

```
<button v-if="auth">Déconnexion</button>
<ul>
  <li v-for="element in elements">
    {{ element.text }}
  </li>
</ul>
```

`v-bind` pouvant être abrégé en `:` permet d'effectuer une liaison unidirectionnelle d'une propriété Vue avec un attribut HTML. Contrairement à la directive `v-model` qui elle permet une liaison bidirectionnelle. En effet, `v-model` permet de modifier la valeur d'entrée en modifiant les données liées et inversement, tandis que `v-bind` lui permet uniquement de modifier la valeur d'entrée en modifiant les données liées. Exemple :

```
<input v-model="inputModel"></input>
<input v-bind:value="inputBind"></input> ou <input :value="inputBind"></input>
```

`v-on` pouvant être abrégé en `@` permet d'exécuter une fonction JavaScript lors du déclenchement d'événements DOM. Exemple :

```
<button v-on:click="function"></button> ou <button @click="function"></button>
```

Pour ce qui est de la syntaxe JavaScript proposée par Vue, je l'ai utilisée de la manière suivante :

```
import { otherComponent } from "component"; //Importation de composant si
nécessaire
export default {
  components: {
    otherComponent,
  },
  name: "MyComponent", //Le nom du composant actuel
  data() { //Fonction contenant les variables du composant actuel
    return {
      message : "Lorem Ipsum",
      ...
    };
  },
  methods: {}, //Méthodes du composant actuel
  computed: {}, //Propriétés calculées du composant actuel
  created() {}, //Code JavaScript exécuté après que l'instance ait été créée
  mounted() {}, //Code JavaScript exécuté juste après que l'instance ait été
montée
  destroyed() {}, //Code JavaScript exécuté après que l'instance de Vue ait été
détruite
};
```

Quant à la balise `<style></style>` je l'ai utilisée de manière très classique. A noter que celle-ci contient l'attribut `scoped` afin que le contenu CSS de la balise soit appliqué uniquement aux éléments du composant actuel.

Pour plus d'informations sur Vue, vous pouvez vous référer à sa très bonne documentation.

[Documentation de Vue.js](#)

npm

npm de son nom *Node Package Manager* est le gestionnaire de paquets officiel de Node.js. npm permet de gérer les dépendances JavaScript de ma PWA. Similaire à Composer qui lui génère le fichier `composer.json`, npm lui, permet de générer un fichier nommé `package.json`. Son fonctionnement en est également similaire. En effet, il permet à npm de rechercher les différentes dépendances JavaScript que ma PWA doit télécharger. Afin d'installer un paquet comme axios, j'ai dû exécuter la commande suivante dans mon invite de commandes :

```
$ npm install axios
```

Après avoir exécuté ce type de commande, mon projet contient maintenant les fichiers `package.json` et `package-lock.json` ainsi que le dossier `node_modules`. Les fichiers `package.json` et `package-lock.json` ont le même fonctionnement que les fichiers `composer.json` et `composer.lock`. Le dossier `node_modules` quant à lui, contient tous les paquets et dépendances installés du projet.

Librairies

AlertifyJS

Plugin vue-alertify installé avec la commande :

```
$ npm install vue-alertify
```

Et importé dans le fichier `vue-alertify.js` de la manière suivante :

```
import Vue from "vue";  
  
import VueAlertify from "vue-alertify";  
Vue.use(VueAlertify);
```

Lui-même importé dans le fichier `main.js` de la manière suivante :

```
import "../plugins/vue-alertify";
```

J'ai utilisé les notifications d'[AlertifyJS](#) afin d'avertir l'utilisateur lors des différentes actions de celui-ci comme l'ajout d'un chien pour un utilisateur par exemple :

```
this.$alertify.success("Chien ajouté avec succès");
```

[Documentation de vue-alertify](#)

Axios

Installé avec la commande :

```
$ npm install axios
```

Et importé de manière globale dans le fichier `main.js` de la manière suivante :

```
import Axios from "axios";  
Vue.prototype.$http = Axios;
```

J'ai utilisé le client HTTP basé sur les promesses Axios afin de réaliser des requêtes HTTP asynchrone à mon API REST depuis ma PWA. Par exemple, j'ai utilisé Axios afin d'ajouter un chien à un utilisateur avec l'endpoint `POST api/v1/dogs` dans le composant `CustomerInformation.vue` de la manière suivante :

```
createDogForCustomerByUserId(dogName, dogBreed, dogSex, dogChipId, userId) {  
  const params = new URLSearchParams();
```

```
params.append("name", dogName);
params.append("breed", dogBreed);
params.append("sex", dogSex);
params.append("chip_id", dogChipId);
params.append("user_id", userId);
const config = {
  headers: {
    "Authorization" : this.$store.state.api_token,
  },
};
this.$http
  .post(this.$API_URL + "dogs/", params, config)
  .then((response) => {
    //Traitement de la réponse
  })
  .catch((error) => {
    //Traitement de l'erreur
  });
}
```

[Documentation d'Axios](#)

Vuex

Installé avec la commande :

```
$ npm install vuex
```

J'ai utilisé Vuex afin de créer le gestionnaire d'état de mon application. Ce gestionnaire d'état nommé **store** par Vuex me permet de stocker de manière centralisée l'api token, le rôle et l'identifiant de l'utilisateur authentifié afin que ces données soient accessibles par l'intégralité de mes composants dans le but de permettre à l'utilisateur authentifié d'accéder à ses fonctionnalités. Dans son principe, Vuex fonctionne de la même manière que Vue Router. En effet, j'ai créé un fichier **store.js** contenant l'instance de mon **store**. Cette instance permet de gérer l'état de mon application en y plaçant les différentes actions et mutations qui devront être faites sur celui-ci. L'instance **store** de mon application est divisée en différentes parties :

- **state** contenant l'état de l'application.
- **mutations** permettant de modifier l'état de l'application.
- **actions** permettant un fonctionnement similaire aux mutations à la différence qu'au lieu de modifier l'état, elles appellent les mutations et peuvent contenir des opérations asynchrones.
- **getters** permettant de réaliser des propriétés calculées.

Par exemple, afin d'authentifier l'utilisateur et de ce fait modifier l'état de mon application, j'ai utilisé Vuex dans le fichier **store.js** de la manière suivante :

```
import Vuex from "vuex";
import router from "./router";
import { CUSTOMER_CODE_ROLE, ADMIN_CODE_ROLE } from "./constants.js";
```

```
Vue.use(Vuex);

export default new Vuex.Store({
  state: {
    api_token: null,
    code_role: null,
    user_id: null
  },
  mutations: {
    authUser(state, userData) {
      state.api_token = userData.api_token;
      state.code_role = userData.code_role;
      state.user_id = userData.user_id;
    },
    ...
  },
  actions: {
    login({ commit }, authData) {
      const params = new URLSearchParams();
      params.append("email", authData.email);
      params.append("password", authData.password);
      Vue.prototype.$http
        .post(Vue.prototype.$API_URL + "connection/", params)
        .then((res) => {
          localStorage.setItem("api_token", res.data.api_token);
          localStorage.setItem("code_role", res.data.code_role);
          localStorage.setItem("user_id", res.data.user_id);
          commit("authUser", {
            api_token: res.data.api_token,
            code_role: res.data.code_role,
            user_id: res.data.user_id,
          });
          Vue.prototype.$alertify.success("Vous êtes connecté");
          if (this.state.code_role == ADMIN_CODE_ROLE) {
            router.push("/administration");
          }
          if (this.state.code_role == CUSTOMER_CODE_ROLE) {
            router.push("/customer_information");
          }
        })
        .catch((error) => {
          Vue.prototype.$alertify.error(error.response.data.error);
        });
    },
    ...
  },
  getters: {
    ifAuthenticated(state) {
      return state.api_token !== null;
    },
    ...
  }
});
```

L'action `login` va permettre de faire un appel à l'endpoint de connexion de l'API REST `api/v1/connection` avec Axios. Si l'endpoint c'est bien déroulé, l'état de l'application se verra subir une mutation et le local storage se verra être chargé. Une fois mon `store` paramétré dans l'instance `Vuex.Store`, il est nécessaire de l'inclure en tant qu'option dans l'initialisation de l'instance principale de Vue disponible dans le fichier `main.js` :

```
import store from "./store";

new Vue({
  store,
  render: (h) => h(App)
}).$mount("#app");
```

[Documentation de Vuex](#)

Vue Router

Installé avec la commande :

```
$ npm install vue-router
```

J'ai utilisé Vue router afin de créer mon application monopage. Pour intégrer Vue router à Vue, j'ai créé le fichier `index.js` dans le dossier router. Ce fichier contient mon instance Router contenant toutes les routes de ma SPA. Chacune de mes routes sont configurées de sorte à avoir : un chemin d'accès, un nom, un composant de référence et une méthode de vérification afin d'accéder à celles-ci si cela est nécessaire. Par exemple, la route permettant d'afficher la page d'administration des éducateurs canins est configurée dans le fichier `router.js` de la manière suivante :

```
import Router from "vue-router";
import store from "./store";
import Administration from "../../components/Administration.vue";
import { ADMIN_CODE_ROLE } from "./constants.js";

Vue.use(Router);

export default new Router({
  routes: [
    {
      path: "/administration",
      name: "administration",
      component: Administration,
      beforeEnter(to, from, next) {
        if (store.state.api_token && store.state.code_role ==
ADMIN_CODE_ROLE) {
          next();
        } else {
```

```
        next("/");
      }
    },
    ...
  ]
});
```

Dans cette route ainsi que dans toutes les routes pour accéder aux pages des éducateurs canins, je teste à l'aide de la fonction `beforeEnter` de Vue router si l'utilisateur est un administrateur (éducateur canin) ou un client. Si celui-ci est un administrateur, alors je le laisse accéder au composant. Dans le cas contraire, je le redirige vers la page d'accueil de l'application. Une fois les différentes routes paramétrées dans l'instance `Router`, il est nécessaire de l'inclure en tant qu'option dans l'initialisation de l'instance principale de Vue disponible dans le fichier `main.js` :

```
import router from "./router";

new Vue({
  router,
  render: (h) => h(App)
}).$mount("#app");
```

[Documentation de Vue Router](#)

BootstrapVue

Plugin bootstrapVue installé avec la commande :

```
$ npm install vue bootstrap bootstrap-vue
```

Et importé dans le fichier `bootstrap-vue.js` de la manière suivante :

```
import Vue from "vue";

import BootstrapVue from "bootstrap-vue";
import "bootstrap/dist/css/bootstrap.min.css";
import "bootstrap-vue/dist/bootstrap-vue.css";
Vue.use(BootstrapVue);
```

Lui-même importé dans le fichier `main.js` de la manière suivante :

```
import "./plugins/bootstrap-vue";
```

J'ai utilisé le plugin BootstrapVue afin de faciliter l'intégration et la compatibilité de Bootstrap avec Vue. En effet, certaines fonctions de Bootstrap nécessitant JQuery et Poppers.js, peuvent entrer en conflit avec Vue. BootstrapVue va justement permettre de convertir la plupart de ces fonctions dans le but de les rendre compatibles avec Vue afin qu'elles fonctionnent comme prévu. Pour ce qui est de la syntaxe de BootstrapVue, elle est très similaire voire plus lisible que Bootstrap.

[Documentation de BootstrapVue](#)

Signature Pad

Plugin vue-signature-pad installé avec la commande :

```
$ npm install vue-signature-pad
```

Et importé dans le fichier `bootstrap-vue.js` de la manière suivante :

```
import Vue from "vue";  
  
import VueSignaturePad from "vue-signature-pad";  
Vue.use(VueSignaturePad);
```

Lui-même importé dans le fichier `main.js` de la manière suivante :

```
import "../plugins/vue-signature-pad";
```

J'ai utilisé le plugin vue-signature-pad afin de faciliter l'intégration et l'utilisation de la librairie JavaScript [Signature Pad](#) avec Vue. Le plugin permet d'utiliser le composant `<VueSignaturePad />` en lui fournissant si on le souhaite, différents paramètres et d'utiliser différentes fonctions décrites dans la documentation du plugin.

[Documentation de vue-signature-pad](#)

FullCalendar

Composant FullCalendar installé avec la commande :

```
$ npm install @fullcalendar/vue
```

Et importé dans les composants souhaitant l'utiliser de la manière suivante :

```
import FullCalendar from "@fullcalendar/vue";  
import dayGridPlugin from "@fullcalendar/daygrid";  
import timeGridPlugin from "@fullcalendar/timegrid";
```

```
import interactionPlugin from "@fullcalendar/interaction";
import frLocale from "@fullcalendar/core/locales/fr";
```

J'ai utilisé FullCalendar afin d'afficher les différents calendriers de mon application nécessitant d'être chargés avec des événements comme les rendez-vous de l'éducateur canin par exemple. FullCalendar permet une intégration et une utilisation optimales avec Vue en fournissant un composant qui correspond exactement aux fonctionnalités de l'API standard de FullCalendar. FullCalendar permet de personnaliser l'affichage de son calendrier ainsi que de ses fonctionnalités de manière très complète. Voici un exemple de la configuration effectuée pour l'affichage des créneaux horaires disponibles d'un éducateur canin :

```
<FullCalendar ref="fullCalendar" :options="calendarOptions" />
```

```
export default {
  components: {
    FullCalendar,
  },
  name: "Calendar",
  data() {
    return {
      calendarOptions: {
        plugins: [dayGridPlugin, timeGridPlugin, interactionPlugin],
        initialView: "dayGridMonth",
        headerToolbar: {
          left: "prev,next today",
          center: "title",
          right: "dayGridMonth,timeGridWeek,timeGridDay",
        },
        height: "auto",
        locale: frLocale,
        eventDisplay: "block",
        allDaySlot: false,
        slotMinTime: "06:00:00",
        slotMaxTime: "20:00:00",
        events: [],
        eventBackgroundColor: "green",
        eventClick: this.handleDateClick,
      },
    };
  },
}
```

Toutes ces options ainsi que bien d'autres sont disponibles dans la documentation de FullCalendar

[Documentation de FullCalendar pour Vue](#)

Moment.js

Installé avec la commande :

```
$ npm install moment
```

Et importé dans les composants souhaitant l'utiliser de la manière suivante :

```
import moment from "moment";  
moment.locale("fr-ch");
```

J'ai utilisé la librairie moment.js afin de formater et manipuler les différentes dates de mon application. Voici un exemple de traitement que j'ai réalisé avec moment.js afin de formater une date :

```
formatDate(value) {  
  if (value) {  
    return moment(value).format("dddd DD MMMM YYYY"); //Exemple de résultat :  
    lundi 21 juin 2021  
  }  
}
```

[Documentation de Moment.js](#)

reCAPTCHA

Composant Google reCAPTCHA installé avec la commande :

```
$ npm install vue-recaptcha
```

Et importé dans le composant `inscription.vue` de la manière suivante :

```
import VueRecaptcha from "vue-recaptcha";
```

J'ai utilisé le composant vue-recaptcha afin d'implémenter une vérification lors de l'inscription à l'aide du système de détection automatisée d'utilisateur reCAPTCHA de Google. Pour intégrer reCAPTCHA à mon application, je me suis rendu sur la documentation officielle de Google afin de récupérer la clef d'intégration côté client. Une fois ma clef récupérée, j'ai configuré le composant vue-recaptcha de la manière suivante :

```
<vue-recaptcha  
  ref="recaptcha"  
  @verify="onVerify"  
  sitekey="clientKey"  
  :loadRecaptchaScript="true"  
>  
</vue-recaptcha>
```



```
export default {  
  name: "Inscription",  
  components: {  
    VueRecaptcha  
  },  
  methods: {  
    onVerify(response) {  
  
    },  
  },  
}
```

Ici, la méthode `verify` émet la réponse qui représente le token à envoyer à l'API REST via l'endpoint `POST api/v1/users` afin de procéder à la vérification côté serveur.

[Documentation de vue-recaptcha](#) [Documentation reCAPTCHA pour l'intégration côté client](#)

Afin de valider la vérification d'utilisateur avec reCAPTCHA côté client, il est nécessaire d'envoyer une requête HTTP au service reCAPTCHA. Pour ce faire, j'appelle la méthode suivante dans mon UserController lors de l'inscription :

```
public static function reCAPTCHAvalidate(string $userResponseToken)  
{  
  $curl = curl_init();  
  curl_setopt($curl, CURLOPT_URL,  
    "https://www.google.com/recaptcha/api/siteverify");  
  curl_setopt($curl, CURLOPT_POST, 1);  
  curl_setopt($curl, CURLOPT_POSTFIELDS, http_build_query(  
    array(  
      'secret' => getenv('GOOGLE_RECAPTCHA_SECRET_KEY'),  
      'response' => $userResponseToken  
    )  
  ));  
  curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);  
  
  $response = json_decode(curl_exec($curl));  
  
  return $response->success;  
}
```

[Documentation reCAPTCHA pour l'intégration côté serveur](#)

Tests Katalon Recorder

Afin de tester les fonctionnalités de mon application WEB développée avec Vue, j'ai suivi les conseils de mon professeur de diplôme et j'ai réalisé des scripts de test avec Katalon Recorder. Utilisant principalement Chrome comme navigateur, je me suis rendu sur le chrome web store afin de télécharger l'extension Katalon Recorder. L'interface graphique de l'extension ressemble à cela :

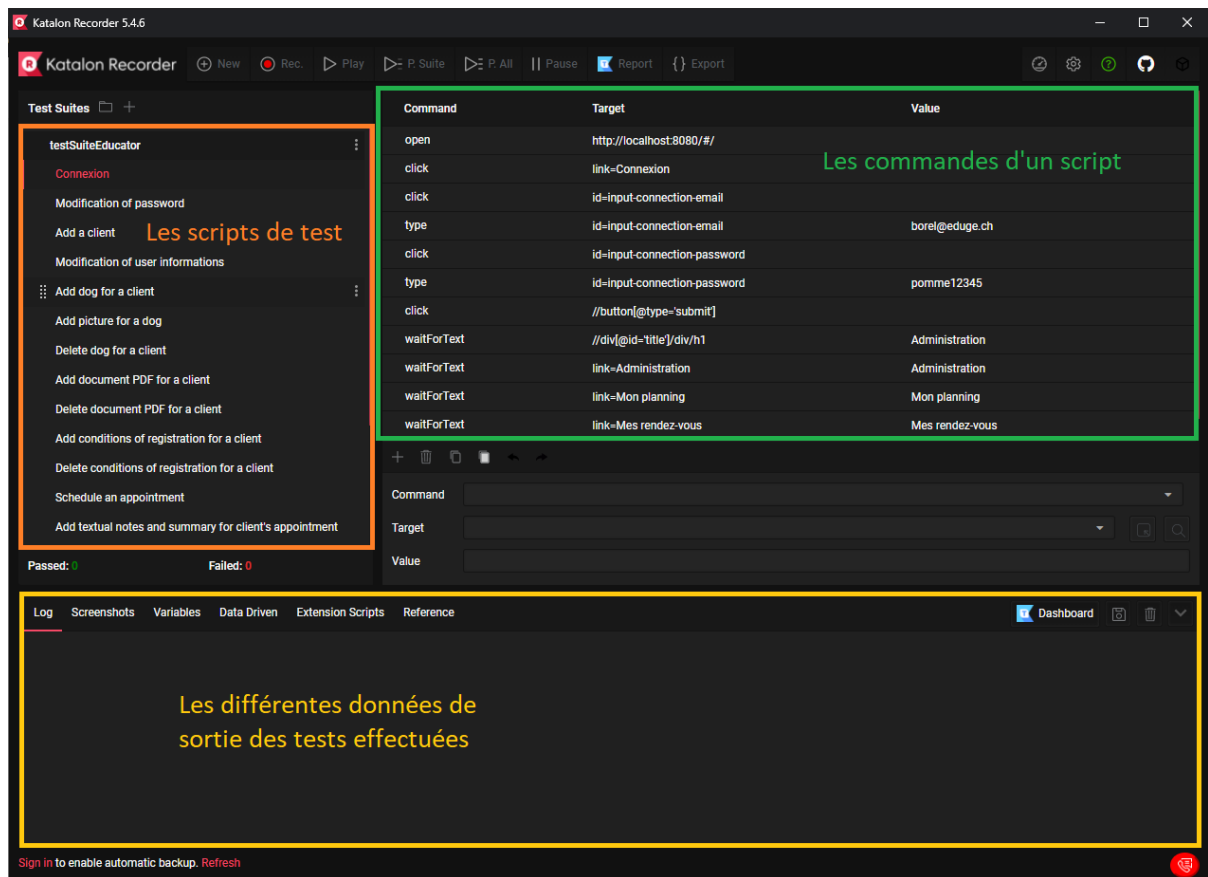


Fig.13 - Interface graphique de Katalon Recorder

Afin de tester le maximum de fonctionnalités avec Katalon Recorder, j'ai développé un script de test par fonctionnalité. Les scripts de test contiennent des lignes d'action qui vont se dérouler les unes après les autres. Chaque ligne est configurée de la manière suivante :

- Une commande désignant une action que le script doit exécuter, comme un click sur un bouton par exemple.
- Une cible permettant au script de savoir sur quel élément il doit exécuter l'action, comme un élément avec l'identifiant HTML `btnInscription` par exemple.
- Et une valeur si c'est nécessaire en cas de réalisation d'action d'écriture par exemple.

Grâce à cette interface, j'ai pu développer et utiliser des scripts de test. Voici un exemple du script de test de connexion de l'éducateur canin :

Connexion de l'éducateur canin

Commande	Cible	Valeur	Description
open	http://localhost:8080/#/		Ouverture de la page d'accueil de l'application
click	link=Connexion		Click sur le bouton de connexion de la barre de navigation
click	id=input-connection-email		Click sur la zone de saisie pour l'adresse e-mail
type	id=input-connection-email	borel@eduge.ch	Écriture de l'adresse e-mail "borel@eduge.ch" dans la zone de saisie pour l'adresse e-mail

click	id=input-connection-password		Click sur la zone de saisie pour le mot de passe
type	id=input-connection-password	poire54321	Écriture du mot de passe "poire54321" dans la zone de saisie pour le mot de passe
click	//button[@type='submit']		Click sur le bouton de connexion du formulaire
waitForText	//div[@id='title']/div/h1	Administration	Vérification de la présence du titre de composant "Administration"
waitForText	link=Administration	Administration	Vérification de la présence du lien "Administration" dans la barre de navigation
waitForText	link=Mon planning	Mon planning	Vérification de la présence du lien "Mon planning" dans la barre de navigation
waitForText	link=Mes rendez-vous	Mes rendez-vous	Vérification de la présence du lien "Mes rendez-vous" dans la barre de navigation

Table des figures

Numéro de figure	Figure
1	Dog Model
2	Table dog
3	Data Access Object Dog
4	Dog Controller
5	ResponseController et HelperController
6	Classes DatabaseConnector et Constants
7	Structure de l'API REST
8	Base de de données de l'API REST
9	Planning sans traitement
10	Planning avec traitement
11	Visual Explain Plan de la requête de génération de planning
12	Format de code des tests unitaires
13	Interface graphique de Katalon Recorder