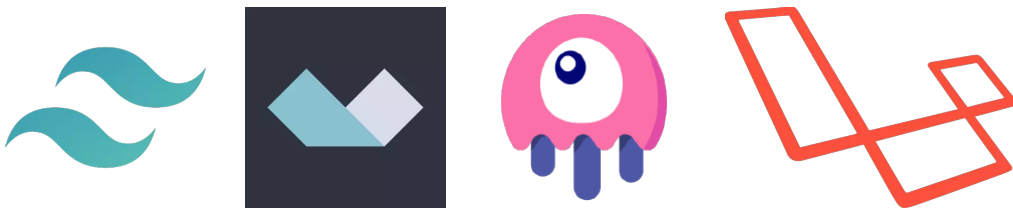


# Documentation de Podz

Application web de publication de podcasts

# Podz



Projet TPI - 2022

Samuel Roland, SI-MI4A

# Table des matières

• Analyse préliminaire	5
◦ Introduction	5
◦ Glossaire	5
◦ Objectifs	5
◦ Planification initiale	
• Analyse / Conception	
◦ Concept	
▪ Technologies utilisées	
▪ Outils d'aide	
▪ Base de données: MCD	
▪ Base de données: MLD	
▪ Maquettes	
◦ Stratégie de test	
▪ Où sont écrits les tests ?	
▪ Prérequis pour lancer les tests	
▪ Comment lancer les tests ?	
◦ Planification	
◦ Dossier de conception	
▪ Upload d'un fichier audio pour la création d'un épisode	
▪ Eléments réutilisables	
• Réalisation	
◦ Dossier de réalisation	
◦ Résultats des tests effectués	
▪ Couverture des tests	
◦ Liste des documents fournis	
• Conclusions	
◦ Objectifs atteints / non-atteints	
◦ Bilan personnel	
◦ Suites possibles au projet	
• Annexes	
◦ Résumé du rapport du TPI	
◦ Sources – Bibliographie	
◦ Remerciements	
◦ Journal de travail	
◦ Manuel d'Installation	
◦ Archives du projet	

# Analyse préliminaire

---

## Introduction

Podz est une application web de publication de podcasts, pour le projet de TPI de Samuel Roland en SI-MI4A. Les auteurs peuvent créer des podcasts, publier des épisodes, planifier la publication d'épisodes dans le futur et les cacher. L'application est basée sous Laravel 9 et ne part pas d'un projet existant.

## Glossaire

- **BDD**: Behavior Driven Development
- **CSS**: Cascading Style Sheets
- **HTML**: Hypertext Markup Language
- **IDE**: Integrated Development Environment
- **MCD**: Modèle Conceptuel de Données
- **MLD**: Modèle Logique de Données
- **MVC**: Modèle Vue Contrôleur
- **PHP**: PHP Hypertext Preprocessor
- **POO**: Programmation orientée objet
- **TALL**: TailwindCSS - AlpineJS - Livewire - Laravel : stack de 4 frameworks web

## Objectifs

Voici la liste des objectifs à atteindre, tirée du cahier des charges:

### Fonctionnalités générales (reprises d'un ancien projet)

- Création de comptes utilisateurs.
- Authentification des utilisateurs.
- Il n'y aura pas de partie back-office ni de rôle administrateur.

Ces fonctionnalités sont implémentées par Jetstream, je n'ai donc pas eu besoin de m'en occuper.

### Fonctionnalités détaillées selon le type d'utilisateur

- En tant que visiteur (personne non authentifiée) :
  - Consultation de la liste des podcasts.
  - Consultation du détail d'un podcast : épisodes
  - Ecoute d'un épisode d'un podcast.
- En tant qu'utilisateur authentifié, en plus des fonctionnalités accessibles à tout visiteur :
  - Création d'un nouveau podcast, édition d'un de ses podcasts existant.
  - Sur l'un de ses podcasts :
    - Affichage de la liste des épisodes avec toutes les données liées.
    - Ajout d'un nouvel épisode.
    - Edition d'un épisode.
    - Suppression d'un épisode.

### En plus de cela, le travail sera évalué sur les 7 points spécifiques suivants:

1. Modélisation des données pertinentes (types, tailles, associations).
2. Respect du modèle MVC.
3. Ergonomie de l'interface utilisateur.
4. Gestion des erreurs de saisie des utilisateurs.
5. Respect des normes d'écriture de code.
6. Utilisation d'un SCM type git avec commits atomiques, petits et fréquents.
7. Lecture audio du podcast bien réalisée.

## Planification initiale

La planification rendue le premier jour était faite sur un document séparé et avec une mise en page peu pratique, j'ai donc repris les données et j'ai changé l'affichage pour plus de lisibilité.

ID	Sprint	Titre	Estimé
#1	S1	Réunion avec expert 1 et chef de projet	2h
#3	S1	Mettre en place Laravel dans un nouveau repository	2h
#22	S1	Faire la planification	3h
#18	S1	Créer le MCD et documenter les spécificités	3h
#2	S1	Coder les migrations Laravel et générer le MLD	3h
#4	S1	Faire les maquettes et commencer la documentation avec le canva	3.5h
#10	S1	Documentation quotidienne Sprint 1	4h
#5	S2	Consultation de la liste des podcasts	3h
#6	S2	Création d'un nouveau podcast, édition d'un de ses podcasts existant	5h
#9	S2	Affichage de la liste des épisodes avec toutes les données liées (auteur)	4h
#7	S2	Ajout d'un nouvel épisode	5h
#11	S2	Documentation quotidienne Sprint 2	4h
#21	S3	Consultation du détail d'un podcast (visiteur)	3h
#8	S3	Edition d'un épisode	2h
#19	S3	Suppression d'un épisode	2h
#12	S3	Documentation quotidienne Sprint 3	4h
#20	S4	Ecoute d'un épisode d'un podcast	2h
#15	S4	Refactorisations du code et amélioration de la qualité générale	4h
#14	S4	Finalisation de la documentation	8h
#13	S4	Documentation quotidienne Sprint 4	4h

ID	Sprint	Titre	Estimé
<a href="#">#17</a>	S5	Relecture finale de tous les documents	3h
<a href="#">#16</a>	S5	Préparation du rendu et impression documents finaux	4h

# Analyse / Conception

---

## Concept

TODO ???? retirer ?

L'application tourne en PHP sur un serveur Apache. Elle utilise une base de données MySQL pour stocker ses données.

## Technologies utilisées

J'ai choisi la stack **TALL** (*TailwindCSS* - *AlpineJS* - *Livewire* - *Laravel*) pour ce projet, car je suis à l'aise avec ces 4 frameworks et parce qu'ils permettent d'être productif pour développer une application web.

### Petits aperçus de ces frameworks

- **Laravel**: un framework PHP basé sur le modèle MVC et en POO. Laravel donne accès à beaucoup de classes et fonctions très pratiques, d'avoir une structure imposée, d'avoir des solutions simples aux récurrents (traductions, authentification, gestion des dates, ...). Tout ceci simplifie beaucoup le développement d'applications web en PHP une fois qu'on ait à l'aise avec les bases.
- **Livewire**: un framework pour Laravel permettant de faire des composants fullstack réactifs. L'idée est d'utiliser la puissance de Blade et du PHP pour avoir des parties réactives sur le frontend (normalement codé en Javascript) sans devoir coder des requêtes AJAX.
- **AlpineJS**: un petit framework Javascript relativement simple à apprendre, utilisée ici pour gérer certaines interactions que Livewire ne permet pas, ou qui concernent des états d'affichage (là où des requêtes sur le backend seraient inutiles). Les composants s'écrivent inline (sur les balises HTML directement). Très pratique pour afficher un dropdown, faire une barre de progression, ...
- **TailwindCSS**: un framework CSS, concurrent de Bootstrap mais centré autour des propriétés CSS (en ayant des classes utilitaires - "utility-first") au lieu de tourner autour de composants. C'est très puissant pour construire rapidement des interfaces, en écrivant quasiment jamais de CSS pur et pour faire du responsive c'est très pratique parce qu'on peut préfixer toutes les classes par **md:** par ex. afin dire que la classe ne s'applique que sur les écrans medium et au dessus.

Divers:

- **Jetstream**: Un starter Kit Laravel mettant en place les fonctionnalités d'authentification, tels que la connexion, la création de compte, la gestion du compte et beaucoup d'autres. L'option Livewire a été utilisée.

## Outils d'aide

Pour m'aider dans mon développement, j'ai utilisé différents outils, ils ne sont pas indispensables mais peuvent être très utiles:

- [Clockwork](#): paquet Composer et extension web pour debugger les performances, les requêtes SQL, voir le temps d'exécution, ... Le paquet Composer est déjà installé.

Path Controller	Status	Time Database		Performance	Models	Database	Views	
GET /livewire/livewire.js?id=c69dLivewireJavaScriptAssets@source	200	85 ms 0 ms						
GET /PodcastController@index	200	97 ms 8 ms						
GET /PodcastController@index	200	98 ms 10 ms						
GET /PodcastController@index	200	113 ms 11 ms						
GET /PodcastController@index	200	97 ms 10 ms						

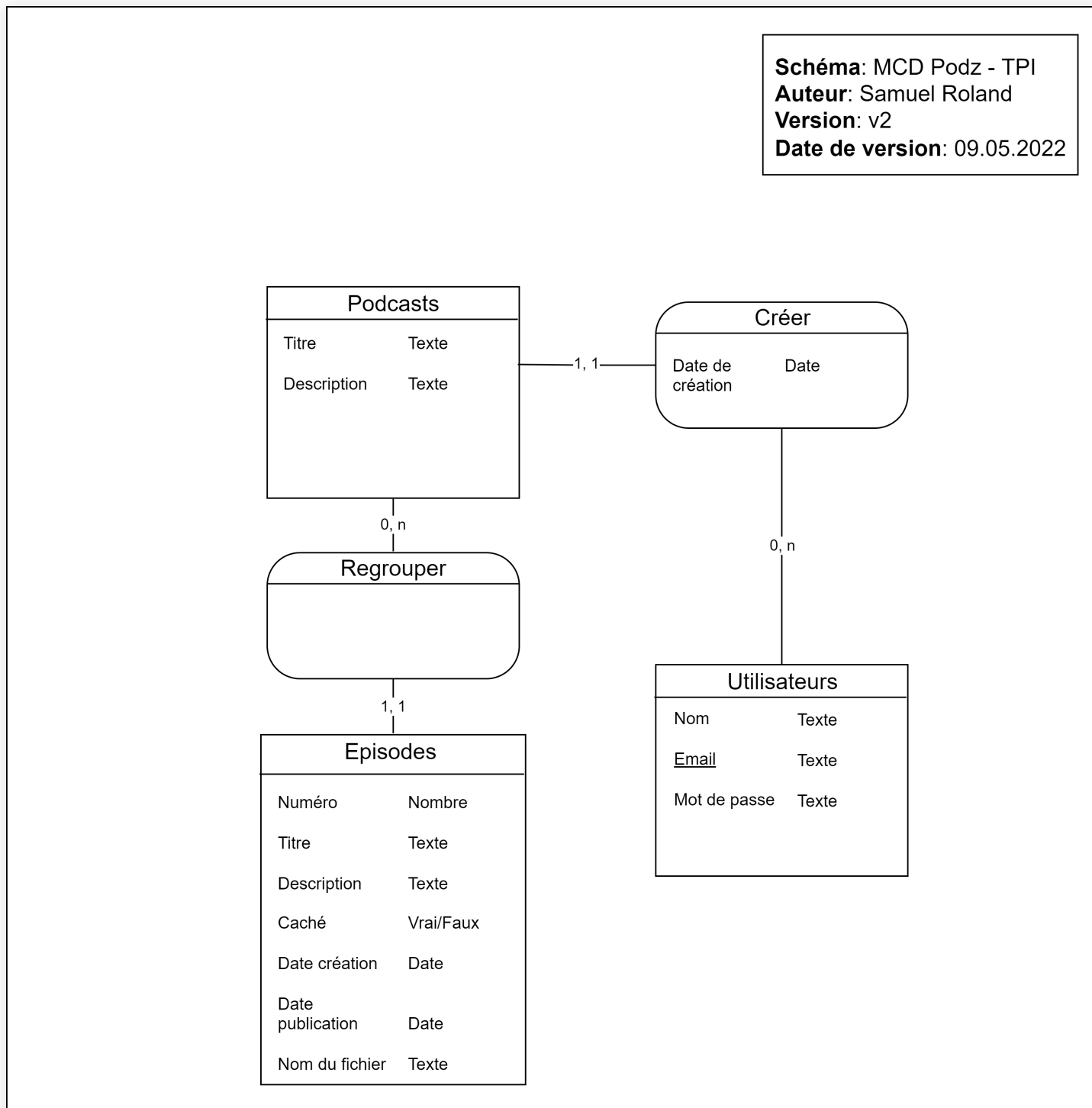
  

Model	Query	Duration
-	SELECT * FROM `sessions` WHERE `id` = 'cHVH85y2eM31mVVFDFEigzfVol9jFwoajJTG0JFZ' LIMIT 1 DisableBrowserCache.php:19	1.41 ms
Podcast	SELECT * FROM `podcasts` PodcastController.php:13	0.34 ms
Episode	SELECT * FROM `episodes` WHERE `episodes`.`podcast_id` in (1, 2, 3, 4, 5) ORDER BY `number` DESC PodcastController.php:13	0.44 ms
User	SELECT * FROM `users` WHERE `users`.`id` in (1, 2, 3) PodcastController.php:13	0.34 ms
Episode	SELECT * FROM `episodes` WHERE `episodes`.`podcast_id` = 1 and `episodes`.`podcast_id` IS not NULL and `released_at` < '2022-05-28 09:50:41' and `hidden` = 0 ORDER BY `number` DESC index.blade.php:29	0.38 ms
Episode	SELECT * FROM `episodes` WHERE `episodes`.`podcast_id` = 2 and `episodes`.`podcast_id` IS not NULL and `released_at` < '2022-05-28 09:50:41' and `hidden` = 0 ORDER BY `number` DESC index.blade.php:29	0.38 ms
Episode	SELECT * FROM `episodes` WHERE `episodes`.`podcast_id` = 3 and `episodes`.`podcast_id` IS not NULL and `released_at` < '2022-05-28 09:50:41' and `hidden` = 0 ORDER BY `number` DESC index.blade.php:29	0.36 ms
Episode	SELECT * FROM `episodes` WHERE `episodes`.`podcast_id` = 4 and `episodes`.`podcast_id` IS not NULL and `released_at` < '2022-05-28 09:50:41' and `hidden` = 0 ORDER BY `number` DESC index.blade.php:29	0.31 ms

- [Laravel Valet](#): fait tourner des serveurs web avec Nginx les rendant accessibles via des domaines en .test. Ce qui me permet de faire tourner mon serveur sous [podz.test](#) sans avoir besoin de me soucier de démarrer et d'arrêter ce serveur ni de gérer plusieurs ports quand plusieurs serveurs sont allumés. Fonctionne pour MacOS, mais des forks pour [Windows](#) et [Linux](#) existent également. Attention à bien suivre la procédure d'installation pour ne pas être coupé d'internet à cause du DNS local mal configuré.

```
[sam@sx]~/code/podz% valet links
+-----+-----+-----+-----+
| Site   | SSL | URL                               | Path                               |
+-----+-----+-----+-----+
| apdebug | X   | https://apdebug.test             | /home/sam/code/apdebug          |
| podz   | X   | https://podz.test                | /home/sam/code/podz             |
+-----+-----+-----+-----+
```

# Base de données: MCD



## Spécificités dans Episodes:

- Les combinaisons du Numéro et du podcast lié, ainsi que le titre et le podcast lié, sont uniques (exemple: on ne peut pas avoir 2 fois un épisode 4 du podcast "Summer stories", et on ne peut pas avoir 2 fois un épisode nommé "Summer 2020 review" du podcast "Summer stories").
- La date de création est définie par la date de création de l'épisode sur la plateforme (avec l'upload du fichier), peu importe ses autres informations (la publication ou l'état caché n'a pas d'influence sur cette date). Cette date ne change jamais et est affichée qu'à l'auteur.

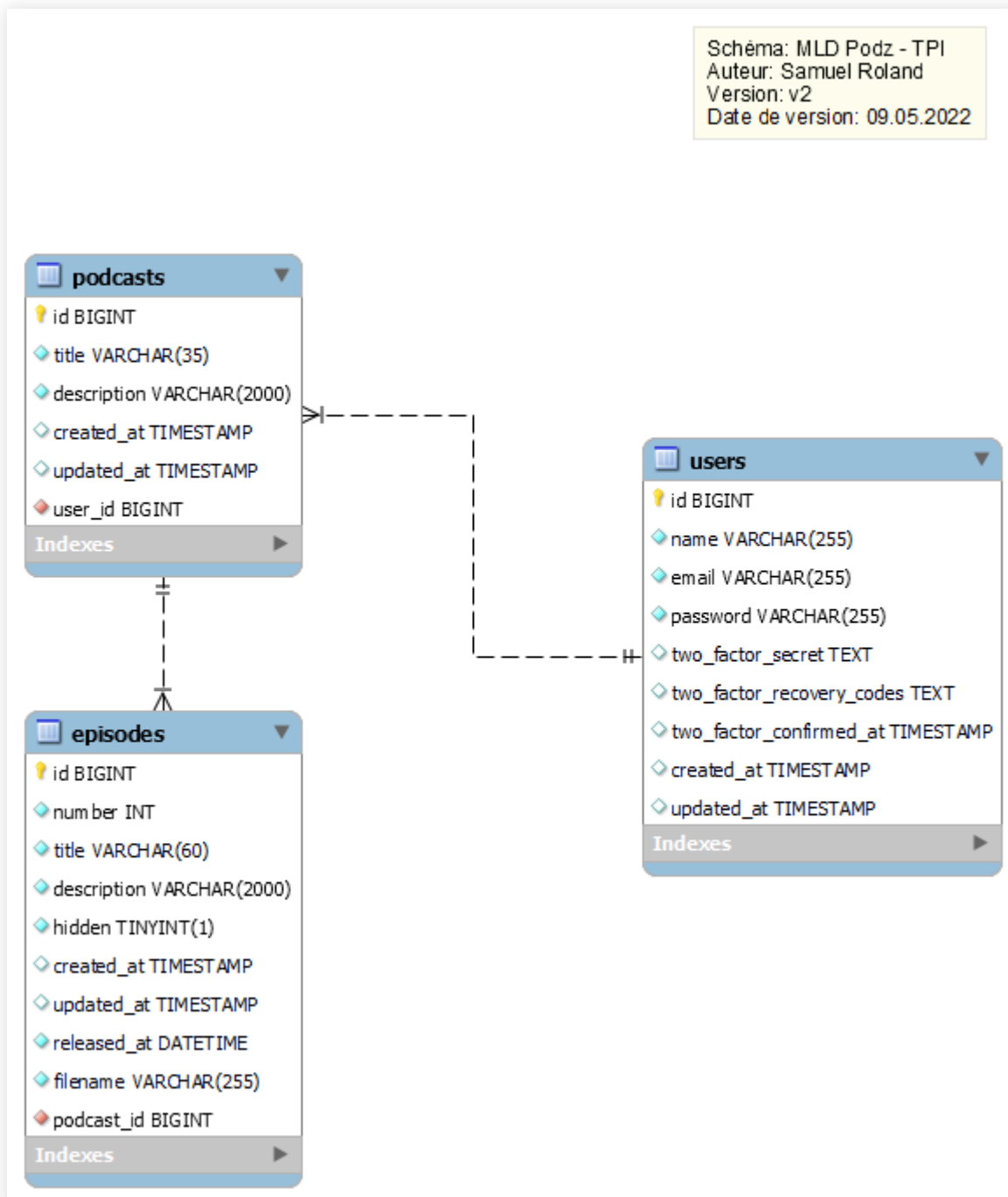


- La date de publication peut être dans le passé ou mais aussi dans le futur. Si elle est dans le futur, l'épisode n'est pas encore publié (jusqu'à la date définie). Ceci permet de programmer dans le futur une publication.
- Le champ Caché est par défaut à Faux et n'a pas d'effet dans ce cas. S'il est Vrai, l'épisode ne sera pas visible dans les détails du podcast.

**Spécificités dans Podcasts:**

- La combinaison du titre et de l'auteur est unique. Exemple: Michelle ne peut pas publier 2 podcasts s'appelant "My story", par contre Michelle et Bob peuvent chacun publier 1 podcast nommé "My story".

## Base de données: MLD



Ce MLD n'a pas été fait à la main mais a été rétro-ingéniéré depuis la base de données, après avoir codé les migrations. Certains champs sont créés par une migration générée par Jetstream, je n'en ai pas besoin mais je ne vais pas les retirer au risque de casser certaines parties existantes. Ce MLD omet volontairement les tables générées par Laravel et propres à chaque application Laravel ([sessions](#), [migrations](#), ...), une partie provient de migrations créées par Jetstream. Ne vous étonnez donc pas de trouver d'autres tables dans la base de données, je ne les utilise pas directement.

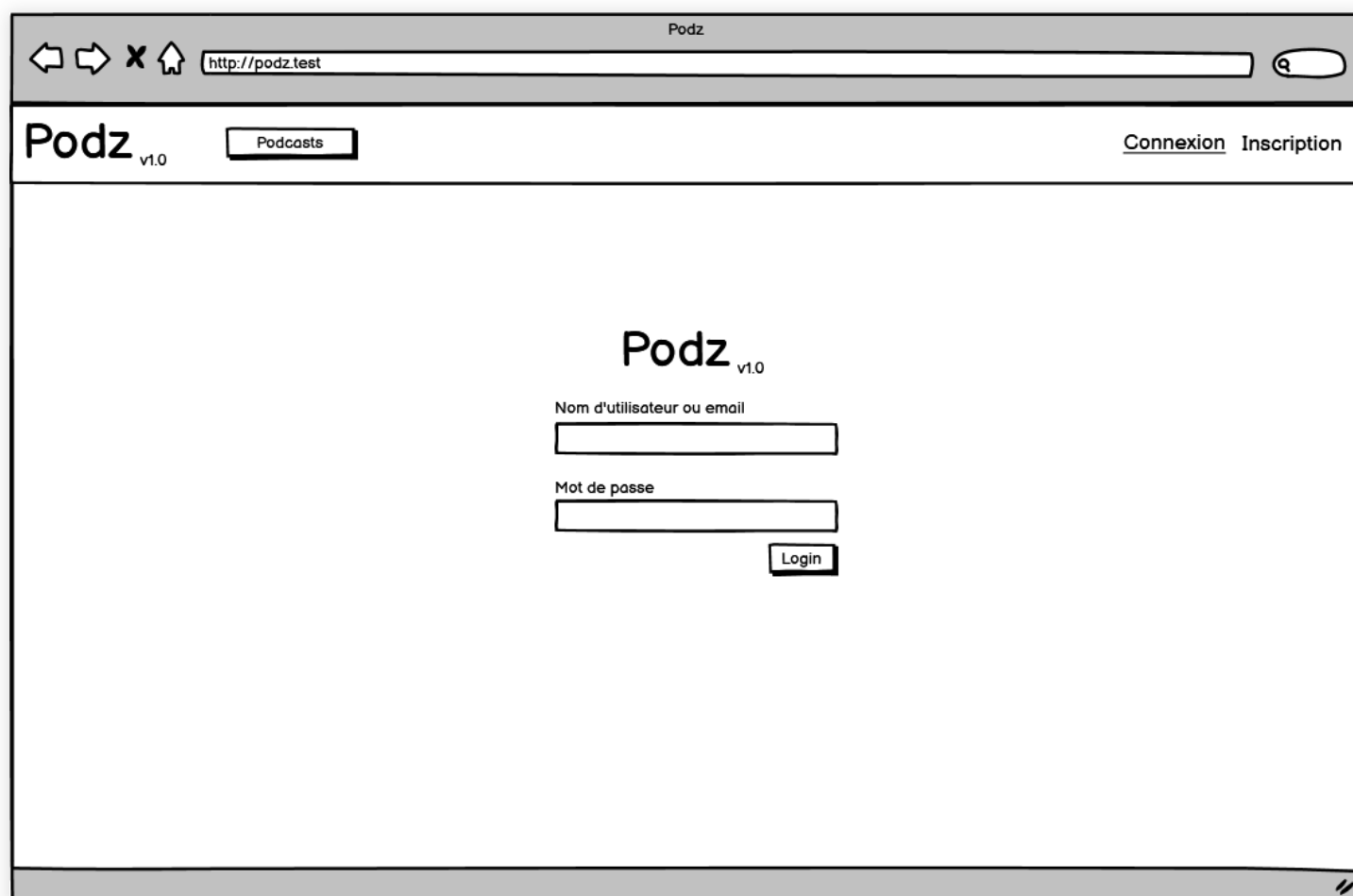
todo: documenter spécificités. Les champs [created\\_at](#) et [updated\\_at](#) sont gérés automatiquement par Laravel, je n'utilise que le [created\\_at](#) en lecture seulement.

# Maquettes

Pour pouvoir utiliser les fonctionnalités requises, voici la liste complète des pages nécessaires et leur maquette:

- Page Connexion
- Page Inscription
- Page Liste des podcasts
- Page Page Détails d'un podcast
  - Vue visiteur
  - Vue Détails et édition pour auteur
- Page Création d'un podcast

## Page Connexion



Maquette de la page de connexion (Page Connexion) pour Podz v1.0.

Le navigateur affiche l'URL `http://podz.test`.

Le header de la page contient le logo **Podz** v1.0, un bouton **Podcasts**, et des liens [Connexion](#) et [Inscription](#).

Le contenu principal de la page est centré et comprend :

- Le logo **Podz** v1.0.
- Le label "Nom d'utilisateur ou email" suivi d'un champ de saisie.
- Le label "Mot de passe" suivi d'un champ de saisie.
- Un bouton **Login**.

Page Inscription

Podz

Podcasts

Connexion Inscription

Podz

Username

Email

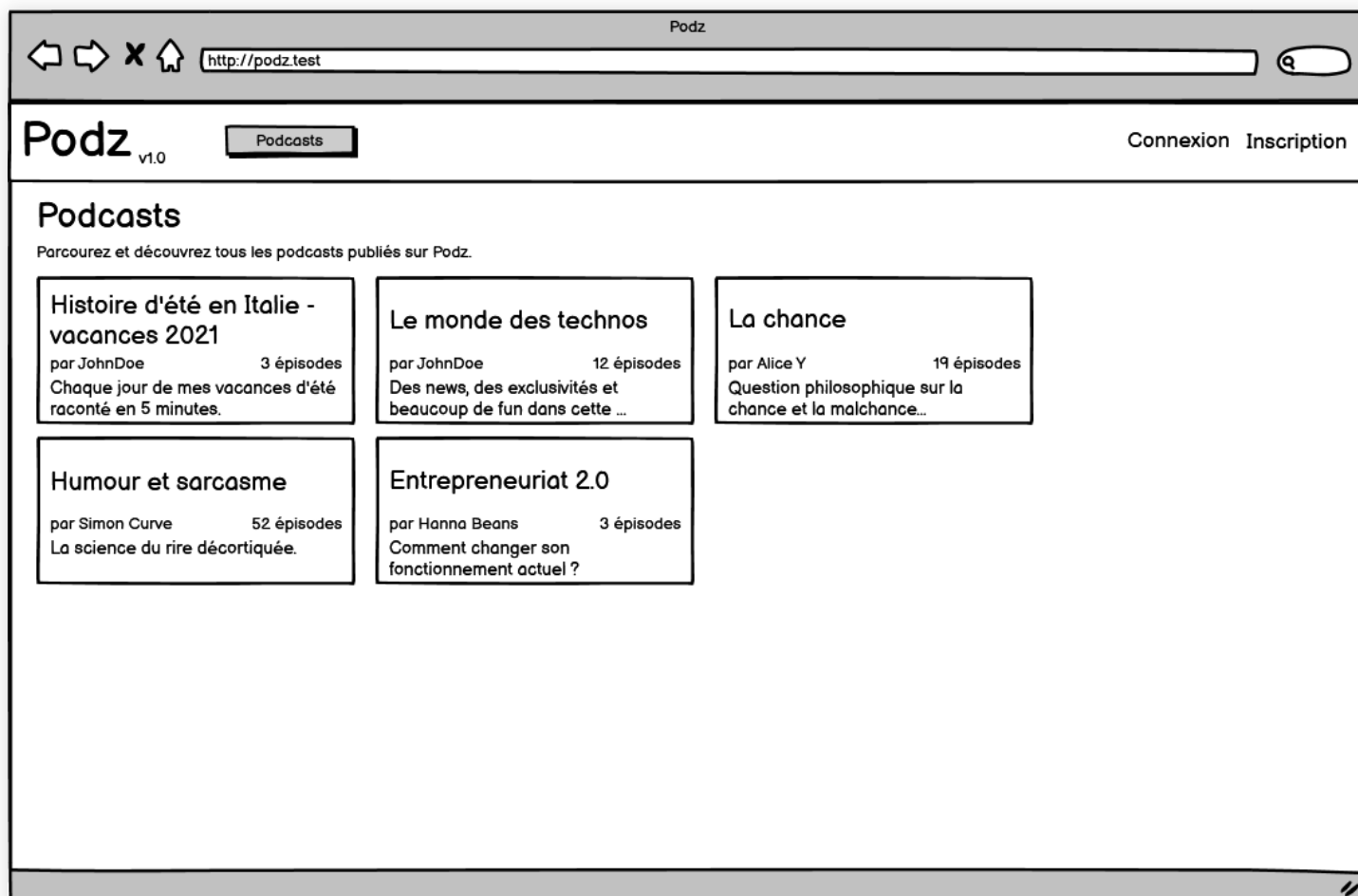
Password

Confirm Password

Register

## Page Liste des podcasts

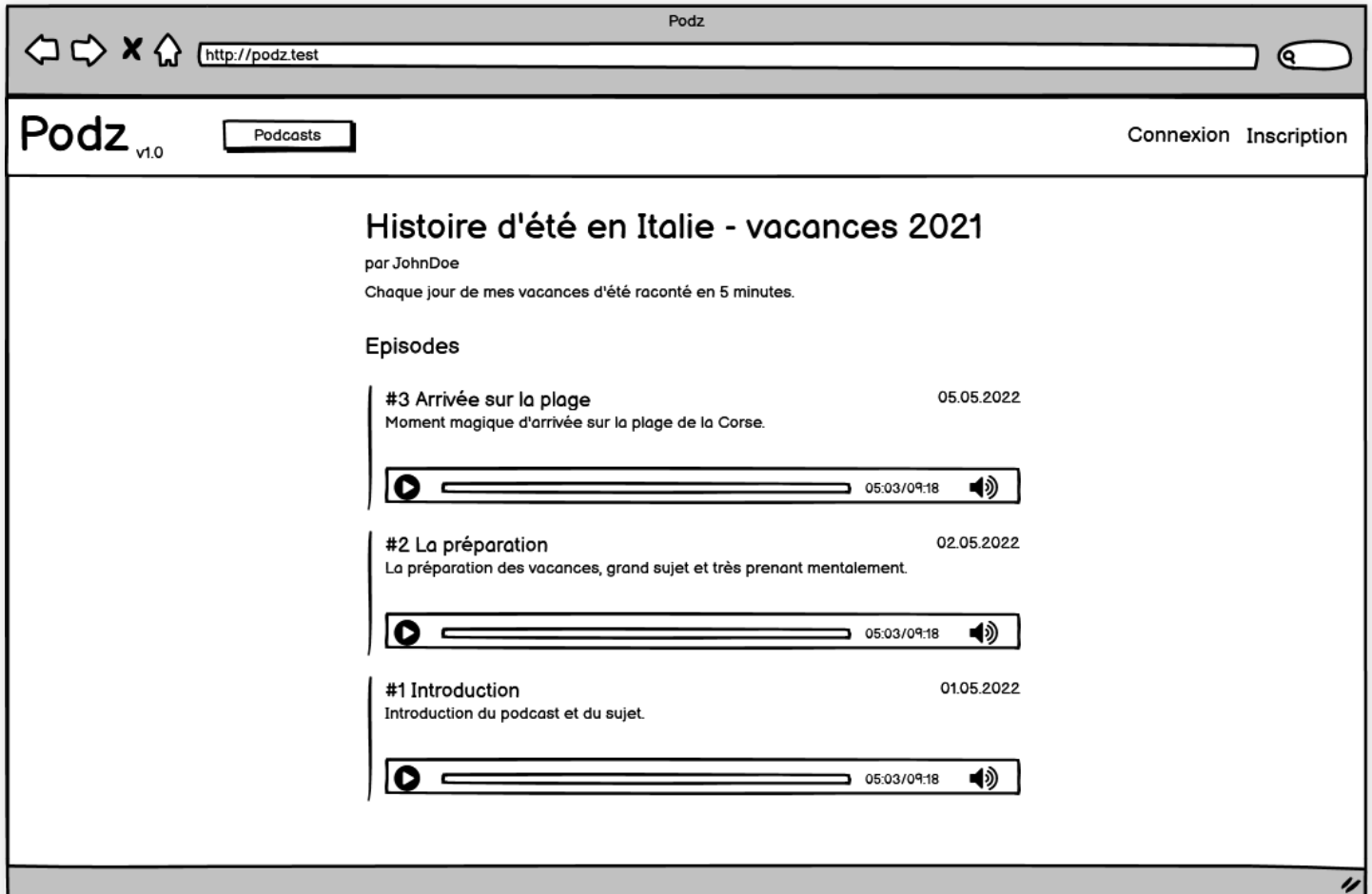
Cette page est visible publiquement et c'est la page par défaut de l'application, on y accède également via le bouton Podcasts en haut à gauche. On peut cliquer sur un podcast pour accéder à ses détails.



## Page Détails d'un podcast

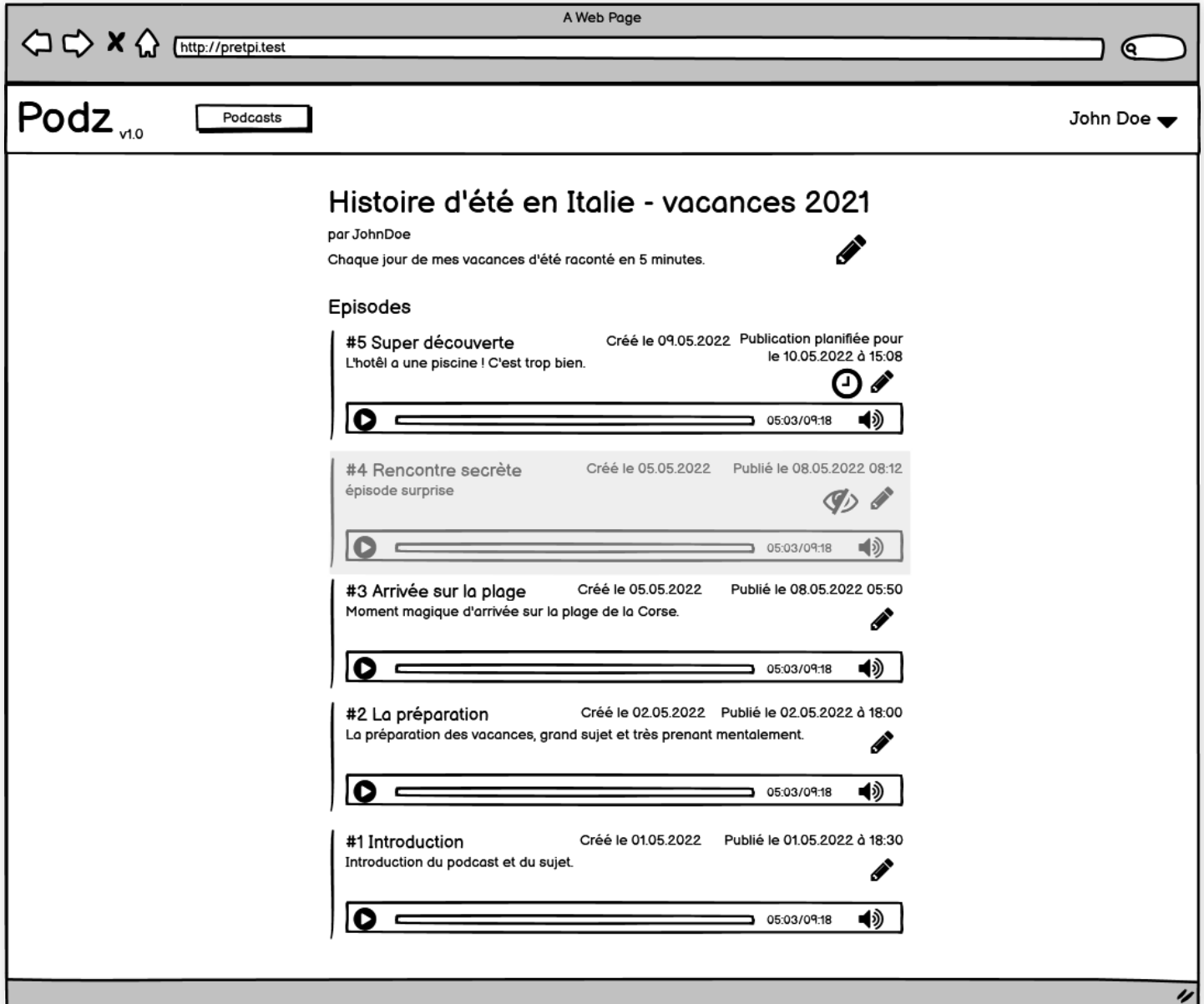
### Vue visiteur

Les visiteurs ne voient que les épisodes qui sont visibles et qu'une partie de leurs informations. Ils ne voient que le numéro, le titre, la description, l'audio et la date (mais sans l'heure et la minute de publication).



## Vue Détails et édition pour auteur

L'auteur voit toutes les informations de ses podcasts contrairement au visiteur. L'auteur a une vue visiteur sur les podcasts qui ne lui appartiennent pas. Nous sommes le 09.05.2022 dans cette maquette, l'épisode 4 est caché et le 5 est planifié pour le 10.05.2022 à 15:08. L'épisode 4 est caché parce que l'auteur a décidé après coup de le remettre en privé. Voici l'apparence de la page quand un auteur la charge.



Quand l'auteur clique sur les icônes d'édition, des formulaires s'affichent pour les éléments sélectionnés afin de permettre l'édition ou la suppression. Ici l'auteur crée un 5ème épisode planifiée qui ne sera publié que le lendemain à 15h08. On peut éditer plusieurs éléments à la fois, il n'y aura pas de conflit.

A Web Page

http://pretpi.test

Podz

v1.0

Podcasts

John Doe

## Histoire d'été en Italie - vacances 2021

par JohnDoe

Chaque jour de mes vacances d'été raconté en 5 minutes.

Enregistrer

### Nouvel épisode

#5  ☐ Caché

Fichier audio (mp3, m4a)

Aperçu 05:03/09:18

### Episodes

#4  ☒ Caché

Aperçu 05:03/09:18

#3 Arrivée sur la plage Créé le 05.05.2022 Publié le 08.05.2022 à 05:50  
Moment magique d'arrivée sur la plage de la Corse.

05:03/09:18



## Page Création d'un podcast

Simple formulaire pour créer un nouveau podcast, avec affichage des erreurs en dessous des champs si jamais les valeurs rentrées sont invalides.

The screenshot shows a web browser window with the address bar displaying 'http://pretpi.test'. The page title is 'A Web Page'. The website header includes the logo 'Podz v1.0', a navigation menu with 'Podcasts' selected, and a user profile 'John Doe' with a dropdown arrow. The main content area is titled 'Créer un podcast' and contains a form with two fields: 'Titre' (Title) and 'Description'. The 'Titre' field is a small text input, and the 'Description' field is a larger text area. A 'Créer' (Create) button is located at the bottom right of the form. The browser window has a standard toolbar with back, forward, and search icons.

Podz v1.0 Podcasts John Doe ▼

### Créer un podcast

Titre

Description

Créer

# Stratégie de test

Cette section concerne la manière dont est testé Podz durant le projet et à la fin. Je teste manuellement les fonctionnalités dans son navigateur (Firefox) et écrit aussi des tests automatisés avec PHPUnit (un framework PHP de tests). La plupart des fonctionnalités sont couvertes par ces tests automatisés et quand cela n'est pas le cas, je regarde à la main si cela fonctionne. Les factories et le seeder écrits sont également très utile pour les tests.

La stratégie de développement est le BDD (Behavior Driven Development). Cela consiste à écrire des tests qui testent le comportement avant de coder, s'assurer que le test plante, puis développer jusqu'à que le test passe. Ensuite on peut refactoriser pour augmenter la qualité tout en s'assurant que cela fonctionne.

Toute la suite de tests est lancée très fréquemment (plusieurs fois par jour) pour s'assurer qu'une nouvelle fonctionnalité n'a pas cassé une autre en chemin.

## Où sont écrits les tests ?

Tous les tests se trouvent dans le dossier `tests` à la racine du repository. Le dossier `Feature` contient les tests fonctionnels, `Unit` les tests unitaires et `Jetstream` les tests créés par Jetstream (ces derniers ont été déplacés de `Feature` afin de ne pas les exécuter constamment).

## Prérequis pour lancer les tests

Il est nécessaire d'avoir mis en place le projet et d'avoir l'extension PHP SQLite.

Avant chaque test, on retourne à l'état initial grâce au trait `RefreshDatabase`. Puis le seeder `DatabaseSeeder` s'exécute grâce au `$seed` défini à `true`. Ces 2 configurations sont faites dans `tests/TestCase.php`, ce qui permet au final que tous les tests sont lancés sur une base de données propre et remplie.

Afin de ne pas impacter la base de données de développement, les tests sont lancés sur une base de données SQLite en mémoire. Voici les lignes en bas du fichier de configuration de PHPUnit `phpunit.xml`, qui redéfinit 2 variables d'environnement permettant d'avoir une base de données en RAM.

```
<env name="DB_DATABASE" value=":memory:"/>
<env name="DB_CONNECTION" value="sqlite"/>
```

## Comment lancer les tests ?

Il y a différentes manières de lancer les tests dans un terminal dans le dossier du projet:

- `./vendor/bin/phpunit`
- `php artisan test`
- `phpunit` (si phpunit a été installé séparément)

Les tests en dehors du dossier `tests/Unit` et `tests/Feature` ne seront pas lancés. Pour exécuter les tests de Jetstream si besoin, il faut lancer `php artisan test tests/Jetstream`.

Vous pouvez passer des paramètres à `phpunit` (aussi possible pour la commande `php artisan test`).

### Exemples:

1. pour exécuter seulement 1 test nommé `podcasts_page_exists` :  
`php artisan test --filter podcasts_page_exists`
2. pour exécuter une classe de tests donnée:  
`php artisan test tests/Feature/PodcastsTest.php`
3. pour exécuter les tests d'un dossier:  
`php artisan test tests/Unit`

Je recommande de configurer un raccourci dans votre IDE pour lancer les tests. J'ai utilisé ce réglage de raccourci dans VSCode pour lancer tous les tests lors d'un `ctrl+t ctrl+t`

```
{
  "key": "ctrl+t ctrl+t",
  "command": "workbench.action.terminal.sendSequence",
  "args": {
    "text": "php artisan test\u000D"
  }
}
```

## Planification

La liste des tâches est la même qu'au départ, les estimations n'ont pas été modifiées, l'ordre est le même qu'il y avait dans les colonnes Todo sur GitHub au début du projet. Afin de comparer ce qui avait été prévu et ce qui s'est réellement passé finalement, j'ai rajouté quelques colonnes. Tout le tableau est ordonné par la date d'achèvement des tâches, ce qui explique que ce n'est pas exactement le même ordre que la planification initiale.

ID	Sprint	Terminé dans	Titre	Estimé	Passé	Delta	Achèvement
<a href="#">#14</a>	S4	-	Finalisation de la documentation	8h	h	8h	-
<a href="#">#15</a>	S4	-	Refactorisations du code et amélioration de la qualité générale	4h	h	4h	-
<a href="#">#16</a>	S5	-	Préparation du rendu et impression documents finaux	4h	h	4h	-
<a href="#">#17</a>	S5	-	Relecture finale de tous les documents	3h	h	3h	-
<a href="#">#20</a>	S4	-	Ecoute d'un épisode d'un podcast	2h	h	2h	-

ID	Sprint	Terminé dans	Titre	Estimé	Passé	Delta	Achèvement
#1	S1	S1	Réunion avec expert 1 et chef de projet	2h	3.5h	-1.5h	03.05.2022
#3	S1	S1	Mettre en place Laravel dans un nouveau repository	2h	1h	1h	03.05.2022
#22	S1	S1	Faire la planification	3h	7.5h	-4.5h	03.05.2022
#2	S1	S1	Coder les migrations Laravel et générer le MLD	3h	1h	2h	03.05.2022
#18	S1	S1	Créer le MCD et documenter les spécificités	3h	1.5h	1.5h	03.05.2022
#4	S1	S1	Faire les maquettes et commencer la documentation avec le canva	3.5h	4h	-0.5h	06.05.2022
#5	S2	S1	Consultation de la liste des podcasts	3h	h	3h	06.05.2022
#10	S1	S1	Documentation quotidienne Sprint 1	4h	3.5h	0.5h	06.05.2022
#9	S2	S2	Affichage de la liste des épisodes avec toutes les données liées (auteur)	4h	h	4h	12.05.2022
#6	S2	S2	Création d'un nouveau podcast, édition d'un de ses podcasts existant	5h	h	5h	13.05.2022
#11	S2	S3	Documentation quotidienne Sprint 2	4h	h	4h	19.05.2022
#21	S3	S3	Consultation du détail d'un podcast (visiteur)	3h	h	3h	22.05.2022
#7	S2	S4	Ajout d'un nouvel épisode	5h	h	5h	23.05.2022
#12	S3	S4	Documentation quotidienne Sprint 3	4h	h	4h	23.05.2022
#8	S3	S4	Edition d'un épisode	2h	h	2h	23.05.2022
#19	S3	S4	Suppression d'un épisode	2h	h	2h	23.05.2022

ID	Sprint	Terminé dans	Titre	Estimé	Passé	Delta	Achèvement
#13	S4	S4	Documentation quotidienne Sprint 4	4h	h	4h	28.05.2022

## Dossier de conception

### Résumé des podcasts

Sur la page Podcasts, il y a un résumé des descriptions des podcasts, qui se limitent à 150 caractères (+3 petits points), puisque la description est trop longue pour être affichée entièrement et l'utilisation de `text-overflow: ellipsis` en CSS sur plusieurs lignes n'est pas très simple. Raccourcir en PHP était donc l'autre solution. Un attribut `summary` de la classe `Podcast` permet de récupérer facilement ce résumé. Si la description est plus courte que la limite, la description est utilisée.

**Visibilité des épisodes** Pour qu'un épisode soit visible publiquement il faut que sa date de publication soit dans le passé et que son état Caché soit Faux. Si cette condition n'est pas vraie, l'épisode n'est visible que par l'auteur.

### Traduction

Pour que les messages d'erreurs soient en français. J'utilise le système d'internationalisation de Laravel et j'ai défini le français comme langue par défaut et l'anglais comme langue de repli ("fallback language") au cas où quelque chose n'aurait pas été traduit en français. J'ai dupliqué le fichier `lang/fr/validation.php` à partir `lang/en/validation.php` et j'ai traduit les quelques messages d'erreurs que j'utilisais.

### Routes

J'ai suivi les conventions des noms et URLs des routes comme pour les contrôleurs ressources (je n'en ai pas utilisé dans ce projet).

## # Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

*Tiré de la documentation de Laravel*

## Upload d'un fichier audio pour la création d'un épisode

J'ai décidé de fixer la taille maximum d'upload de fichiers à 150MB. Cette limite est fixée dans l'application, au niveau de la validation à la création d'un épisode. Ces 2 paramètres dans la configuration de PHP (fichier `php.ini`) doivent être augmentées au dessus de 150MB: `upload_max_filesize` et `post_max_size`.

## Éléments réutilisables

### Le composant Field

Un composant Blade permettant d'abstraire les éléments communs à tous les champs de formulaire: l'affichage du label, le design basique, l'affichage des erreurs de validations.

Propriétés du composant

Nom	Type	Requis	Description
<code>name</code>	String	Oui	Le nom technique du champ, utilisé pour l'attribut <code>name</code> de l'input et par le <code>@error()</code> et par la fonction <code>old()</code> .
<code>label</code>	String	Non	Nom du label au dessus du champ.
<code>type</code>	String	Non	Type de l' <code>&lt;input&gt;</code> . Par défaut <code>text</code> . Si <code>textarea</code> est donné, une balise <code>&lt;textarea&gt;</code> est utilisée à la place.
<code>placeholder</code>	String	Non	Un placeholder qui est ajouté directement sur le champ.

Nom	Type	Requis	Description
<code>cssOnField</code>	String	Non	Des classes CSS qui sont ajoutées directement sur le champ.

Tous les autres attributs non reconnus sont transférés à la `div` racine du composant, ce qui permet d'ajouter du style ou d'autres attributs HTML. Tous les attributs commençant par `wire:model` sont ajoutés au champ pour permettre l'utilisation de ce composant avec Livewire.

Exemple d'utilisation:

```
<form action="{ route('podcasts.store') }}" method="POST">
<x-field label="Title" name="title"></x-field>
<x-field label="Description" type="textarea" name="description"></x-field>
<x-field label="Date de naissance" type="date" name="user.date"></x-field>
[...]
```

Un autre exemple d'utilisation dans le cas d'un formulaire géré par Livewire:

```
<div>
  <x-field
    wire:keyup.enter="update"
    placeholder="Rentrez un titre court et marquant."
    label="Title" name="podcast.title"
    wire:model.lazy="podcast.title">
  </x-field>
  <x-field
    label="Description" type="textarea"
    name="podcast.description" wire:model.lazy="podcast.description">
  </x-field>
  @csrf
  <button wire:click.prevent="update" class="btn mt-1">Enregistrer</button>
</div>
```

## Classes CSS et couleurs

J'ai défini 3 nouvelles couleurs Tailwind, qu'on peut utiliser partout où les couleurs sont utiles avec TailwindCSS ( `border-green` , `text-lightblue` , ...)

```
//Extrait de tailwind.config.js
colors: {
  'green': '#0d9488',
  'blue': '#0d1594',
  'lightblue': '#0d159414',
}
```

Il y a aussi des classes CSS qui peuvent être utilisées pour avoir un design commun à travers l'interface:

- `text-info` : pour les messages d'informations
- `btn` : pour les boutons

# Réalisation

## Dossier de réalisation

### Structure du repository

Certains dossiers de Laravel moins pertinents ont été remplacés par des `...`. Seulement les dossiers et les fichiers à la racine sont affichés. Uniquement ceux que j'ai utilisé sont définis.

podz	Racine du repository
└─ app	
└─ Actions	
└─ Fortify	
└─ Jetstream	
└─ Console	
└─ Exceptions	
└─ Http	
└─ Controllers	Les classes contrôleurs
└─ Livewire	
└─ Middleware	
└─ Models	Les classes modèles
└─ Providers	
└─ View	Les classes des vues, pour les composants Blade
└─ Components	
└─ ...	
└─ config	Les fichiers de configuration globaux
└─ database	Tout ce qui concerne la gestion de la base de données
└─ factories	Les factories pour créer des données fictives
└─ migrations	Les migrations pour définir la structure des tables
└─ seeders	Les seeders pour remplir la base de données avec les factories
└─ docs	Dossier pour stocker les éléments de documentations (MCD, MLD)
└─ imgs	Les images utilisées dans cette documentation
└─ models	Les exports des maquettes
└─ sources	Les fichiers source binaires des maquettes, MCD et MLD
└─ lang	Les fichiers de langues
└─ en	
└─ fr	Certaines traductions en français
└─ public	
└─ resources	Toutes les ressources utiles à générer nos vues
└─ css	Style CSS global écrit dans app.css
└─ js	Javascript global écrit dans app.js
└─ markdown	
└─ views	
└─ api	
└─ auth	
└─ components	
└─ layouts	Contient le gabarit app.blade.php
└─ livewire	Les vues pour Livewire
└─ podcasts	Vues pour les podcasts
└─ profile	
└─ vendor	
└─ jetstream	Les vues de Jetstream



	└ ...	
	└ routes	Configuration des routes dans web.php
	└ storage	Espace de stockage dédié
	└ app	Dossier ciblé par le disque "local"
	└ public	Dossier publiquement accessible et ciblé par le disque "public"
	└ testing	Fichiers audios de tests pour le développement
	└ clockwork	
	└ ...	
	└ logs	Emplacement de laravel.log
	└ tests	Tests automatisés
	└ Feature	Tests fonctionnels
	└ Jetstream	Tests créés par Jetstream
	└ Unit	Tests unitaires
	.editorconfig	
	.env.example	Fichier .env d'exemple
	.gitattributes	
	.gitignore	
	.styleci.yml	
	artisan	Le CLI artisan
	composer.json	Liste des paquets Composer requis
	composer.lock	Liste des paquets Composer installées et leur version
	package-lock.json	Liste des paquets NPM installées et leur version
	package.json	Liste des paquets NPM requis
	phpunit.xml	Fichier de configuration de PHPUnit
	README.md	
	tailwind.config.js	Configuration de Tailwind
	webpack.mix.js	Configuration du build JS et CSS avec Webpack pour Mix

## Résultats des tests effectués

Cette capture montre le résultat des tests exécutés le YYYY à YYYY. Tous les tests passent.

```
[sam@sx]~/code/podz% ./vendor/bin/phpunit
PHPUnit 9.5.20 #StandWithUkraine

..... 44 / 44 (100%)

Time: 00:01.310, Memory: 50.50 MB

OK (44 tests, 174 assertions)
```

Voici la liste complète des tests, les noms devraient permettre d'avoir une idée de ce qui est testé et quels cas sont couverts.

### 1. Tests\Unit\EpisodeTest

1. path is well built

### 2. Tests\Unit\PodcastTest

1. podcasts summary is correctly extracted

1. podcasts summary doesnt extract when description length is already good
3. get next number really gives next number

### 3. Tests\Feature\EpisodeCreationTest

1. podcast details page uses episode creation component
2. podcast details page doesnt use episode creation if not author
3. episode creation works
4. data are correctly validated
5. audio file type is validated
6. default value of the episode are set
7. publishing fails silently if forbidden
8. publishing 2 episodes with same title in a podcast is not possible

### 4. Tests\Feature\EpisodeDeletionTest

1. episode deletion works
2. episode deletion is only authorized to the author

### 5. Tests\Feature\EpisodeUpdateTest

1. podcast details page uses episode update component
2. podcast details page doesnt use episode update if not author
3. episode update works
4. data are correctly validated
5. datetime value is set after mount
6. update fails silently if forbidden
7. updating title to another episode title in the same podcast fails

### 6. Tests\Feature\PodcastCreationTest

1. create a podcast page exists
2. create a podcast page is guarded
3. store route is guarded
4. podcast creation works
5. podcast is not created on invalid request
6. new podcast button is present
7. new podcast button doesnt exist as visitor

### 7. Tests\Feature\PodcastDetailsTest

1. podcasts details page exists
2. podcast info component is included in the page
3. all information are displayed for the author
4. a message is displayed when no episode is published
5. prefix text of future release date is displayed correctly for author
6. release date displays only date for the public
7. future episodes are not publicly visible
8. past hidden episodes are nt visible for the public

9. only required info are displayed publicly

## 8. Tests\Feature\PodcastUpdateTest

1. podcast details page contains update component
2. podcast details page doesnt contain update component as visitor and as non author
3. details can be updated
4. details must be valid

## 9. Tests\Feature\PodcastsTest

1. podcasts page exists
2. the page has title and description
3. all podcasts are displayed with their data

# Couverture des tests

Comme les tests sont écrits et exécutés en PHP, les tests ne peuvent que tester le comportement backend. Les interactions frontend ne peuvent pas être testées avec les outils actuels.

Pour la plupart des fonctionnalités, j'ai suivi cette ordre pour décider des tests à écrire et de leur contenu:

1. D'abord écrire un test pour vérifier que la page existe ou que le composant testé est bien chargé dans une des pages.
2. Ensuite tester le comportement idéal (toutes les données valides) pour s'assurer que les données gérées ont bien été modifiées.
3. Puis tester les validations des données.
4. Et finalement valider les permissions de visibilité ou d'accès (ex: être sûr qu'un visiteur ne peut pas modifier un épisode ou ne peut pas voir d'épisode s'il est invisible).

## Ce que les tests ne couvrent pas:


- Validation de la taille maximale d'upload d'un fichier pour la création d'épisode

Les tests manuels ont permis de vérifier que cela fonctionnait. Un test manuel avec un fichier mp3 de 170Mo a été fait plusieurs fois afin de vérifier la limite de 150Mo. En voici la démonstration:

## Nouvel épisode

#41

Est dolores esse teqwer

Caché ☐15/04/1970, 03:13 

Sed adipisci in consqwer

Fichier audio (.mp3, .ogg ou .opus). Pas plus de 150MB.

Parcourir...

3 Hour Zen Meditation M..., 🎧071C-WZKW2Hq2fks.m4a

*Le fichier ne doit pas dépasser les 150000 KB.*

**Publier**

## Liste des documents fournis

- Ce rapport de projet nommé "Documentation de Podz"
- Journal de travail
- README: contient la documentation d'installation du projet

# Conclusions

## Objectifs atteints / non-atteints

Tous les objectifs fixés au départ ont été atteints.

Objectif	Atteint ?
En tant que visiteur (personne non authentifiée) :	
• Consultation de la liste des podcasts.	Oui
• Consultation du détail d'un podcast : épisodes	Oui
• Ecoute d'un épisode d'un podcast.	Oui
En tant qu'utilisateur authentifié, en plus des fonctionnalités accessibles à tout visiteur :	
• Création d'un nouveau podcast, édition d'un de ses podcasts existant.	Oui
Sur l'un de ses podcasts :	
• Affichage de la liste des épisodes avec toutes les données liées.	Oui
• Ajout d'un nouvel épisode.	Oui
• Edition d'un épisode.	Oui
• Suppression d'un épisode.	Oui

## Bilan personnel

J'ai eu beaucoup de plaisir à développer Podz, surtout avec l'écriture des tests. Contrairement à mon Pré-TPI où je n'avais pas pu terminer le développement et la documentation, je suis plutôt content d'avoir réussi à finir toutes les fonctionnalités demandées dans les temps et d'avoir pu faire correctement la documentation.

Comme durant mon Pré-TPI, j'ai eu de la peine avec l'upload de fichiers, parce que je n'arrivais pas à écrire des tests correctement, je devais tester à la main et cela devenait vite chronophage. Grâce à l'aide M. Hurni mon chef de projet, j'ai pu changer de stratégie pour ces tests

## Suites possibles au projet

Pour la suite du projet.....

# Annexes

---

## Résumé du rapport du TPI

### Situation de départ

Le but du projet est de développer une application web avec Laravel de publication de podcasts. Pour les auteurs, il doit être possible de créer et modifier leurs podcasts, et créer, éditer et supprimer des épisodes dans leurs podcasts. Les épisodes doivent pouvoir être publiés dans le futur et caché par l'auteur si besoin. Le projet est parti de rien (il ne s'appuie pas sur un autre projet). J'ai choisi d'appeler l'application Podz.

Les critères spécifiques demandaient de faire une modélisation des données pertinentes, de respecter les principes du modèle MVC, d'avoir une interface utilisateur propre et utilisable. Il était aussi demandé de suivre les normes d'écriture de code, d'utiliser un système de versionning en faisant des petits commits atomiques et fréquents. Les épisodes devaient aussi être correctement écoutables dans les navigateurs.

### Mise en oeuvre

En plus de l'utilisation du framework Laravel, j'y ai ajouté Livewire, AlpineJS et TailwindCSS. Ces 4 frameworks que j'avais utilisé en stage et pour des projets personnels forment la stack TALL et sont régulièrement utilisé dans l'écosystème Laravel.

Pour ne pas avoir à développer la connexion et la création de compte, j'ai utilisé le starter kit Jetstream qui mettait déjà tout en place. J'ai fait mon MCD et MLD de ma base de données. J'ai réfléchi aux différentes pages nécessaires pour utiliser les fonctionnalités requises et j'ai fait des maquettes pour chacune des pages. La page de détails d'un podcast a en fait plusieurs vues, selon si l'on est visiteur ou auteur, et en tant qu'auteur on peut ouvrir ou fermer les formulaires pour modifier des épisodes ou les informations du podcast. Une fois cette analyse terminée, j'ai développé l'une après l'autre toutes les fonctionnalités demandées, tout en suivant ma planification. J'ai eu un peu d'avance au départ sur le premier sprint (j'ai avancé une tâche du sprint 2 au sprint 1) puis comme la création d'épisode avec l'upload de fichiers était plus complexe que je l'imaginais, j'ai eu un peu de retard sur mon planning, mais j'ai réussi à rattraper le retard à la fin et tout finir dans les temps.

La particularité de mon TPI par rapport à d'autres élèves est que j'ai écrit de nombreux tests automatisés pour m'assurer que la majeure partie du comportement de mon application était correct et restait fonctionnel tout le long du projet. J'ai utilisé PHPUnit pour écrire ces tests. L'écriture a pris un peu de temps tout au long du projet, mais ils ont permis d'accélérer la validation du fonctionnement, j'ai ainsi pu éviter beaucoup d'essais à la main puisque j'avais confiance sur le fait que mon backend fonctionne. Il restait bien sûr à s'assurer que tout fonctionne comme prévu dans mon navigateur mais cela était plus rapide à déterminer.

### Résultats

Toutes les fonctionnalités demandées ont pu être implémentées et testées. La création d'un podcast se fait sur une page dédiée, tandis que l'édition d'un podcast, la création, modification et suppression d'épisodes se font toutes dans la même page Détails d'un podcast. Je n'ai pas eu de difficultés particulières à designer mon application, je n'ai donc pas eu besoin d'utiliser de template.

## Sources – Bibliographie

Pour résoudre mes différents problèmes j'ai utilisé StackOverflow et les documentations officielles des 4 frameworks que j'utilise:

- [Documentation de Laravel](#)
- [Documentation de Livewire](#)
- [Documentation de AlpineJS](#)
- [Documentation de TailwindCSS](#)

J'ai aussi utilisé le site [Mozilla Developer Network](#) comme référence pour le HTML et le CSS.

- **Icônes:** les icônes ont été copié-collées (en SVG) depuis [heroicons.com](#), elle sont publiées sous licence MIT.
- [Liste des Types de médias, par l'IANA](#). Cette ressource m'a été utile pour trouver les types MIME des fichiers audios .ogg, .opus, et .mp3 pour la validation lors de la création d'épisode.

## Remerciements

J'aimerais remercier M. Hurni pour les retours et les conseils techniques qu'il m'a apporté au Pré-TPI et au TPI qui m'ont permis de progresser avec Laravel en général et l'écriture de tests. J'espère avoir pu utiliser au mieux ces feedbacks et continuer de m'améliorer continuellement sur Laravel et les autres frameworks à l'avenir, pour produire du code de qualité et maîtriser de plus en plus ces technologies.

Je remercie aussi Gatien Jayme pour sa relecture de ma documentation.

## Journal de travail

Le journal est disponible en document séparé ou directement sur Github [en Markdown](#) ou [en PDF](#).

## Manuel d'Installation

Toutes les informations nécessaires à l'installation du projet se trouve dans le [README](#).

## Archives du projet

- [podz-code-samuel-roland.zip](#)