IPIES.EDU.PL - dokumentacja techniczna

## Spis treści

* [Spis treści](#spis-treści)
* [Instalacja i konfiguracja](#instalacja-i-konfiguracja)
  + [Wymagania](#wymagania)
  + [Proces instalacji i konfiguracji ( WSL / Linux)](#X6233750ff641b2080f8283b73547c74cc99b1ee)
* [Ogólny zarys działania aplikacji](#ogólny-zarys-działania-aplikacji)
  + [Dlaczego SPA?](#dlaczego-spa)
  + [Przykład interakcji między systemami](#przykład-interakcji-między-systemami)
* [Backend](#backend)
  + [Struktura modeli](#struktura-modeli)
    - [Migracje - struktura bazy danych](#migracje---struktura-bazy-danych)
    - [Modele - relacje i pole $fillable](#modele---relacje-i-pole-fillable)
  + [Mapa relacji dla naszego projektu:](#mapa-relacji-dla-naszego-projektu)
  + [Struktura adresów i kontrolerów API](#struktura-adresów-i-kontrolerów-api)
    - [Adresy](#adresy)
    - [Kontrolery](#kontrolery)
* [Frontend](#frontend)
  + [Komponenty](#komponenty)
  + [Routing](#routing)
* [Todo naszego projektu](#todo-naszego-projektu)
* [Użyte paczki](#użyte-paczki)
  + [Front](#front)
  + [Backend](#backend-1)

IPIES jest systemem który pozwala na testowanie wiedzy uczniów. Jest on napisany przy użyciu Vue.js i Laravel-a.

## Instalacja i konfiguracja

### Wymagania

* Dowolna SQL-owa baza danych (MariaDB, PostgreSQL)
* PHP 7.x
  + php-xml
  + Rozszerzenie wybranej bazy danych ( np.: php-mysql )
* Composer
* npm
* git ### Proces instalacji i konfiguracji ( WSL / Linux)

1. Klonujemy repozytorium do wybranego folderu

* git clone https://github.com/jkarpiu/jakas\_strona\_do\_testow.git   
  cd jakas\_strona\_do\_testow

1. Instalujemy paczki PHP:

* composer install

1. Instalujemy paczki Javascript-a:

* npm install

1. Kopiujemy domyślny plik .env (plik z informacjami o środowisku)

* cp .env.example .env

1. Generujemy klucz szyfrowania aplikacji

* ./artisan key:generate

1. Podajemy dane naszej bazy danych edytując wybranym edytorem plik .env ``` bash vi .env \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* [...]  
    
   DB\_CONNECTION=mysql // rodzaj (mysql, pgsql, sqlite, sqlsrv)  
   DB\_HOST=127.0.0.1 // host bazy   
   DB\_PORT=3306 // port na którym działa serwer bazy  
   DB\_DATABASE=nazwa\_bazy\_danych  
   DB\_USERNAME=użytkownik  
   DB\_PASSWORD=hasło  
    
   [...]
* ```

1. Wykonujemy migracje

* ./artisan migrate

1. Generujemy klucze, które potem będą używane w procesie autoryzacji użytkowników

* ./artisan migrate

1. Tworzymy dowiązania pozwalające odczytywać pliki potrzebne do działania niektórych elementów strony bash ./artisan storage:link
2. Kopiujemy pliki zawierające listę z pytaniami egzaminacyjnymi
3. Zapisujemy te pytania oraz podstawowe informacje do bazy bash ./artisan db:seed --class=DzialySeeder ./artisan db:seed --class=ee08\_seeder //wbrew nazwie to nie tylko ee08 ./artisan db:seed --class=SchoolSedder
4. Tworzymy jedną paczkę ze wszystkich napisanych komponentów Vue bash npm run dev
5. Uruchamiamy serwer ./artisan serve
6. Przechodzimy w naszej przeglądarce na adres 127.0.0.1:8000

## Ogólny zarys działania aplikacji

### Dlaczego SPA?

Nasza aplikacja jest napisana w oparciu o model SPA, co oznacza że cały HTML wczytywany jest tylko raz, reszta danych przesyłanych między przeglądarką, serwerem to tylko czysty JSON. Zaletami takiego rozwiązania są:  
- skrócenie czasu ładowania poszczególnych podstron - większa kontrola nad tym co widzi użytkownik - mniejsze zużycie np.: danych mobilnych

Rozwiązanie to ma też swoje wady takie jak np.: - Zwiększony początkowy czas ładowania strony - dodatkowe czynności podczas wymiany danych

### Przykład interakcji między systemami

Jako praktyczny przykład działania tego założenia w naszej aplikacji weźmy uruchomienie testu niezadanego przez nauczyciela: 1. Użytkownik klika w “Losuj 40 pytań” 2. Vue router zmienia aktualny widok na komponent testu z odpowiednim parametrem. 3. Komponent wysyłá zapytanie o pytania do API Laravel-a, z danymi dotyczącymi testu 4. Laravel przekierowuje dane z adresu do odpowiedniej funkcji 5. Funkcja sprawdza, czy test jest zadany przez nauczyciela, czy nie. Nasz nie jest więc ustawienia testu są domyślne np.: dla EE.09 mamy 60 min, próg zdania 50 % i wylosowane 40 pytań 6. Tworzona jest sesja testu w tabeli activeTests, ma ona swój unikalny klucz i tam także zapisany jest “deadline” naszego testu. 7. Laravel losuje 40 pytań z bazy. 8. Funkcja zwraca jako odpowiedź treść, id pytań odpowiedzi (bez informacji która jest poprawna) oraz informacje o sesji testu. 9. Vue wyświetla pobrane pytania do rozwiązania oraz zapisuje do zmiennej informacje o sesji. 10. Po rozwiązaniu testu Vue wysyła listę zaznaczonych odpowiedzi do innego adresu api, a Laravel jeżeli wszystko się zgadza zwraca do Vue wyniki oraz listę poprawnych odpowiedzi. 11. Vue wyświetla pobrane dane.

## Backend

### Struktura modeli

#### Migracje - struktura bazy danych

Pliki definiujące strukturę bazy danych można znaleźć w folderze database / migrations

migrations  
├── 2014\_10\_12\_000000\_create\_users\_table.php  
├── 2014\_10\_12\_100000\_create\_password\_resets\_table.php  
├── 2019\_08\_19\_000000\_create\_failed\_jobs\_table.php  
├── 2020\_11\_16\_123051\_create\_przedmioties\_table.php  
├── 2020\_11\_16\_152658\_create\_klasies\_table.php  
├── 2020\_11\_16\_153352\_create\_pytanias\_table.php  
├── 2020\_11\_16\_155033\_create\_odpowiedzis\_table.php  
├── 2020\_11\_16\_162705\_create\_dzialies\_table.php  
├── 2020\_11\_16\_184600\_create\_wynikis\_table.php  
├── 2020\_11\_20\_225022\_create\_groups\_models\_table.php  
├── 2020\_11\_20\_225136\_create\_schools\_models\_table.php  
├── 2020\_11\_20\_225147\_create\_cities\_models\_table.php  
├── 2020\_11\_21\_191436\_create\_active\_tests\_table.php  
├── 2020\_11\_25\_193907\_create\_regions\_models\_table.php  
├── 2020\_11\_25\_194125\_create\_school\_types\_table.php  
├── 2020\_11\_25\_200305\_create\_schools\_models\_users\_table.php  
├── 2020\_12\_04\_230933\_create\_groups\_model\_user\_table.php  
├── 2020\_12\_04\_232610\_create\_group\_invitations\_table.php  
├── 2020\_12\_05\_153040\_create\_group\_posts\_table.php  
├── 2020\_12\_05\_153216\_create\_group\_attachments\_table.php  
├── 2020\_12\_05\_153902\_create\_attachment\_types\_table.php  
├── 2020\_12\_06\_113646\_create\_teacher\_tests\_table.php  
├── 2020\_12\_06\_114333\_create\_users\_teacher\_tests\_table.php  
└── 2020\_12\_07\_131419\_create\_comments\_table.php

Każdy z tych plików ma raczej podobną strukturę, najważniejszą częścią jest funkcja up(). Zawiera ona definicje kolumn danej tabeli np.: w pliku comments\_table.php

Schema::create('comments', function (Blueprint $table) {  
 $table->id();  
 $table->timestamps();  
 $table->longtext('tresc');  
 $table->integer('group\_post\_id');  
 $table->integer('user\_id');  
});

Na przykładzie 4 linijki możemy zobaczyć że tworzona jest kolumna o nazwie treść i typie longText. Konwencja w Laravel-u mówi że kolumna zawierająca klucz obcy powinna mieć nazwę składającą się z nazwy tabeli do której ten klucz się odnosi i słówka id. Całość zapisana ma być w notacji węgierskiej. Relacje same w sobie definiowane są w plikach modeli, ale to w większych szczegółach opisane jest poniżej. Oprócz tego możemy tu także zobaczyć dwie funkcje nie przyjmujące parametru: id oraz timestamps. Funkcja id definiuje klucz podstawowy z domyślnymi SQL-ymi parametrami (AUTO-INCREMENT i UNIQUE), a timestamps dodaje dwa pola, jedno z datą utworzenia rekordu, a drugie z datą ostatniej edycji.

#### Modele - relacje i pole $fillable

W głównym folderze app możemy znaleźć pliki odpowiedzialne za definicje relacji i innych rzeczy związanych z odnoszeniem się do bazy danych w kodze

app  
├── activeTests.php  
├── attachmentType.php  
├── citiesModel.php  
├── comments.php  
├── Dzialy.php  
├── groupAttachment.php  
├── GroupInvitation.php  
├── groupPost.php  
├── groupsModel.php  
├── Odpowiedzi.php  
├── przedmioty.php  
├── Pytania.php  
├── regionsModel.php  
├── schoolsModel.php  
├── schoolType.php  
├── teacherTest.php  
├── User.php  
└── wyniki.php

W każdym z tych plików znajdziemy mniej więcej podobną strukturę z główną klasą definiującą model w niej np.: w pliku groupPost.php

protected $fillable = ['author\_id', 'groups\_model\_id', 'content', 'title', 'active'];  
  
public function comments(){  
 return $this->hasMany(comments::class);  
}  
public function group () {  
 return $this -> belongsTo(groupsModel::class );  
}

Zmienna $fillable definiuje tutaj pola, które możemy ‘wypełniać’ z poziomu samej aplikacji. a funkcje definiują tutaj nasze relację.

### Mapa relacji dla naszego projektu:

Możemy tutaj zobaczyć że np.: model użytkownika potrafi być połączony z modelem grup kilkoma relacjami. Wynika to z tego że użytkownik może być zarówno nauczycielem jak i uczniem, ale tylko nauczyciele mogą ‘administratorami’ grupy, i tylko uczniowie mogą być jej członkami (przynajmniej na chwile obecną).

### Struktura adresów i kontrolerów API

#### Adresy

Adresy ap zdefiniowane są w pliku /routes/api.php możemy tam zobaczyć że każdy z zdefiniowanych adresów ma mniej więcej podobną budowę: php Route::get('miasta', 'basicServicesController@miasta'); w której to po słówku route definiowany jest rodzaj akcji ( post / get), później w nawiasach zobaczymy adres oraz kontroler i funkcje do jakiej się odwołujemy. Możemy też zobaczyć że niektóre z funkcji mają middleware ‘auth.api’. Nie pozwala on niezalogowanym użytkownikom na dostęp do danej części aplikacji, a później w kontrolerze daje nam dostęp do obiektu z danymi zalogowanego użytkownika.

#### Kontrolery

Pliki zawierające większość naszych kontrolerów możemy znaleźć w folderze /app/Htpp/controllers. Znajdują się tam następujące pliki:

app/Http/controllers  
├── addQuestionController.php  
├── AuthController.php  
├── basicServicesController.php  
├── CommentsController.php  
├── Controller.php  
├── groupsController.php  
├── HomeController.php  
├── randQuestionController.php  
├── SpaController.php  
├── testsController.php  
└── userController.php

Weźmy za przykład plik basicServicesController.php. Na początku pliku zdefiniowany jest namespace dla kontrolerów, później do naszego pliku importujemy potrzebne modele oraz dodatkowe paczki, modele mają przedrostek App np.:

use App\citiesModel;

importuje model miast do naszego pliku.

Poniżej zdefiniowana jest klasa naszego kontrolera, a w niej poszczególne metody do których odnosimy się w pliku routes/api.php Np: funkcja miasta:

public function miasta(Request $request)  
 {  
 return response()->json(  
 regionsModel::with('cities')->find($request['id'])->cities  
 );  
 }

Przyjmuje parametr $request, który jest obiektem zawierającym wszystkie dane wysłane do danej ścieżki dowolną metodą ( np.: post, get ). Dalej funkcja ta zwraca odpowiedź w formacie json, w której to zawarte są wszystkie miasta, w danym regionie o danym id. W podobny sposób działą większość naszych kontrolerów, jednak zwykle mają więcej ‘ciekawej logiki’ związanej z czasem, czy użytkownikiem.

## Frontend

### Komponenty

Frontend naszej aplikacji jest podzielony na na zamknięte osobne komponenty. Każdy taki komponent zawiera własne style, skrypty (które muszą być pisane obiektowo oczywiście) i szablon strony, który jest napisany w HTML-u z kilkoma specyficznymi dla Vue dodatkami. wszystkie nasze komponenty znajdują się w folderze resources/js/components.

components  
├── AddDzial.vue  
├── App.vue  
├── Groups  
│ ├── Classrooms.vue  
│ ├── Groups.vue  
│ └── OneClassroom.vue  
├── GroupsMain.vue  
├── Loading.vue  
├── Login.vue  
├── PageNotFound.vue  
├── Pytanka  
│ ├── JednoPytanie.vue  
│ └── Wyniki.vue  
├── PytankaCopy.vue  
├── Register.vue  
├── SchoolSelector.vue  
├── SetTest.vue  
├── TestsLists.vue  
├── Tests.vue  
├── test.ts  
├── UpcomingTests.vue  
├── UserMenu.vue  
├── Welcome.vue  
└── WynikiZapisane.vue

Jako przykładowy komponent do analizy weźmy JednoPytanie.vue. Plik ten podzielony jest trzema tagami: template, script i style. Ciekawszymi rzeczami w template są te związane typowo z vue. Np.: możemy przypisać wartość dowolnego parametru do zmiennej zdefiniowanej wewnątrz tagu script. Przykładem tutaj może być div wyświetlający jedną odpowiedź dla naszego pytania:

<div  
 class="ans"  
 :style="calculateColor(odpowiedz.id)"  
 @click="selectAnswer(odpowiedz.id)"  
>  
 <li>  
 <p style="margin: 0; padding: 0; margin-left: 5px">  
 {{ odpowiedz.tresc }}  
 </p>  
 </li>  
</div>

Gdzie do parametru style przypisana jest wartość, którą zwraca funkcja calculateColor:

calculateColor: function (id) {  
 if (this.answered == null) {  
 if (this.check == id) {  
 return "background-color: #0061c9;";  
 }  
 return " ";  
 } else if (  
 this.answered.zaznaczana.id == 1 &&  
 id == this.answered.poprawna.id  
 ) {  
 return "background-color: #007c02;";  
 } else if (this.answered.poprawna.id == id) {  
 return "background-color: #007c02;";  
 } else if (  
 !this.answered.zaznaczana.poprawna &&  
 id == this.answered.zaznaczana.id  
 ) {  
 return "background-color: darkred;";  
 }  
 }

Która to wybiera nam kolor tła w zależności od tego, czy odpowiedź jest poprawna, zaznaczona, czy też niepoprawna.

W naszym projekcie często też używaliśmy paczki axios wewnątrz skryptów, aby kontaktować się z API. Np.: w komponencie WynikiZapisane.vue (konwencje w nazewnictwie nie są naszą mocną stroną) istnieje metoda getWynik:

getWyniki: function() {  
 console.log("test");  
 axios  
 .get("/api/wyniki")  
 .catch(err => {  
 console.log(err.response);  
 })  
 .then(res => {  
 console.log(res.data);  
 this.wyniki = res.data;  
 });  
 },

Gdzie axios pobiera dane z adresu /api/wyniki, a następnie w funkcji then zapisuje je do zmiennej this.wyniki zapisanej w obiekcie data. Do tej zmiennej później możemy się odnieść w szablonie komponentu. ### Routing Routing-iem w naszej aplikacji zajmuje się Vue-Router. Wszystkie możliwe adresy ‘webowe’ dostępne są w zmiennej stałej routes znajdującej się w pliku resources/js/app.js. Weźmy za przykład adres losowanie40

{  
 path: '/losowanie40',  
 name: 'losowanie40',  
 component: Pytanka,  
 props: { ilosc: 40 }  
},

Atrybuty path i name raczej nie wymagają wyjaśnienia. Atrybut component określa jaki komponent będzie głównym komponentem tego adresu, a props definiuje zmienne jakie chcemy podać temu komponentowi. Komponenty są importowane na górze pliku ( więcej o komponentach poniżej ). Głównym komponentem w środku którego wyświetlane są inne komponenty jest app, zdefiniowano to w stałej router.

## Todo naszego projektu

* Lepsza walidacja formularzy
* W pełni działające logowanie z Google
* Załączniki do postów grupach
* Możliwość zadawania zadań grupom
* Możliwość oddania projektu z plikiem
* Edytory typu office przy oddawaniu projektu (OnlyOffice?)
* Możliwość zadania testu kilku grupom naraz
* Więcej rodzajów pytań (otwarte, prawda i fałśz, połącz opcje)
* Możliwość ustawienia kolejności pytań
* Wyświetlanie nadchodzących testów
* Możliwość sprawdzenie swoich odpowiedzi z poziomu podstrony wyniki
* Typ zadania w którym można pisać kod (cos w stylu clash of code)

## Użyte paczki

### Front

* Vue
* Vuex
* Vue-router
* Bootstrap
* JQuery
* Fontawesome
* vue-countdown
* moment.js
* tiptap editor
* vue slick caousel
* vue spinners css
* vuelidate
* Laravel-mix
* typescript

### Backend

* Laravel
* Laravel-passport
* Laravel-socialite
* Laravel-er-diagram-generator