| Dokumentacja projektu | | AI1 |
|---|---|---|
| **Autor** | Krystian Burbano-Marek, 117783 | **Data oddania** |
| **Kierunek, rok** | Informatyka, II rok, st. stacjonarne (3,5-l) | 15.06.2022 |
| **Specjalizacja** | – | |
| **Grupa** | LAB 1 | **Ocena** |
| **Temat projektu** | Aplikacja randkowa | |

1. Tematyka projektu:

Aplikacja randkowa Hobbest to aplikacja w której użytkownik może znaleźć inne osoby o podobnych do niego zainteresowaniach poprzez system polubień. Aplikacja stawia na zainteresowania i hobby innych użytkowników. W przeciwieństwie do innych aplikacji randkowych Hobbest nie pozwala użytkownikom na pokazanie swojej twarzy innym. Pozwala to zapobiec ocenianiu użytkowników poprzez patrzenie na wygląd i pominięcie jego zainteresowań i cech charakteru. Gdy dwoje użytkowników wyrazi swoje wzajemne zainteresowanie odblokowuje się możliwość ich komunikacji poprzez wbudowany w aplikację system czatu.

2. Opis wykorzystanych narzędzi:

- 2.0.1. PHP - interpretowany, skryptowy język programowania zaprojektowany do generowania stron internetowych i budowania aplikacji webowych w czasie rzeczywistym.

  - https://www.php.net/

  Licencja: PHP License

- 2.0.2. Composer - Composer to system zarządzania pakietami dla języka PHP, dostępny jako aplikacja wiersza poleceń, która dostarcza i standaryzuje format zarządzania zależnościami skryptami i bibliotekami.

  - https://getcomposer.org/

  Licencja: MIT

- 2.0.3. Laravel – framework do aplikacji internetowych napisany w języku PHP bazujący na wzorcu architektonicznym Model-View-Controller.

  - https://laravel.com/

  Licencja: MIT

- 2.0.4. jQuery – lekka biblioteka programistyczna dla języka JavaScript, ułatwiająca korzystanie z JavaScriptu.

  - https://jquery.com/

  Licencja: MIT

- 2.0.5. React.js – biblioteka języka programowania JavaScript, która wykorzystywana jest do tworzenia interfejsów graficznych aplikacji internetowych.

  - https://pl.reactjs.org/

  Licencja: MIT

- 2.0.6. Symfony – framework dla aplikacji internetowych napisany w języku PHP bazujący na wzorcu projektowym MVC.

  - https://symfony.com/

  Licencja: MIT

- 2.0.7. Bootstrap – biblioteka CSS, rozwijana przez programistów Twittera, wydawany na licencji MIT

  - https://getbootstrap.com/

  Licencja: MIT

- 2.0.8. Visual Studio Code – darmowy edytor kodu źródłowego z kolorowaniem składni dla wielu języków, stworzony przez Microsoft, o otwartym kodzie źródłowym.

  - https://code.visualstudio.com/

  Licencja: MIT dla kodu, własnościowa dla wersji binarnych.

- 2.0.9. PostgreSQL, także Postgres – obok MySQL i SQLite, jeden z najpopularniejszych otwartych systemów zarządzania relacyjnymi bazami danych.

  - https://www.postgresql.org/

  Licencja: PostgreSQL license

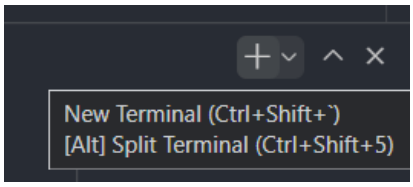
2.1 Opis licencji:

- 2.1.1. PHP license - https://www.php.net/license/3_01.txt
- 2.1.2. MIT - https://choosealicense.com/licenses/mit/
- 2.1.3. PostgreSQL license - https://www.postgresql.org/about/licence/

3. Instrukcja uruchomienia aplikacji:

- 3.1. Pobranie wszystkich narzędzi wymienionych w punkcie 2.*
- 3.2. Wypakowanie aplikacji do nowego folderu.
- 3.3. Otwarcie nowego folderu w programie Visual Studio Code
- 3.4. Konfiguracja aplikacji

Po pobraniu potrzebnych narzędzi i skonfigurowaniu bazy danych w dolnym prawym rogu włączamy nowy terminal poprzez kliknięcie w znak „+”:




Żeby aplikacja dobrze działała, przed uruchomieniem jej należy wprowadzić poniższe komendy

```
composer require --no-interaction
-
npm install
```

- 3.5. Konfiguracja połączenia z bazą danych

Po pobraniu potrzebnych narzędziu uruchamiamy Visual Studio Code i przechodzimy do miejsca gdzie mamy zapisaną aplikację i otwieramy plik .env. W tym pliku zmieniamy pola:

```
DB_CONNECTION=={nazwa sterownika do połączenia z bazą}
DB_HOST={adres bazy danych}
DB_PORT={port bazy danych}
DB_DATABASE={nazwa bazy danych}
DB_USERNAME={adres bazy danych}
DB_PASSWORD= ={hasło z bazy danych}
```

W moim przypadku konfiguracja wygląda w sposób następujący:

```
DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=hobbest_db
DB_USERNAME=postgres
DB_PASSWORD=
```

Tworzenie tabel i wypełnianie ich danymi zawarte jest w punkcie 5.1.

- 3.6 Uruchomianie aplikacji:

Po otwarciu terminala i poprawnym skonfigurowaniu aplikacji i bazy możemy wpisać komendę:

```
php artisan serve
```

Po jej wpisaniu można się już połączyć z aplikacją pod adresem poniżej:

```
php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
[Tue Jun 14 12:54:39 2022] PHP 8.1.2 Development Server (http://127.0.0.1:8000) started
```

4. Omówienie kodu:

- 4.1 Seedery
- 4.1.1: UserSeeder

Przykładowy seed zwykłego użytkownika:

```
                [
                    'name' => 'John',
                    'email' => 'john@email.com',
                    'password' => Hash::make('1234'),
                    'description' => "A green hunting cap squeezed the top of
the fleshy balloon of a head. The green earflaps, full of large ears and uncut
hair and the fine bristles that grew in the ears themselves",
                    'gender' => 'male',
                    'pref_gender' => 'female',
                    'admin' => false,
                    'created_at' => now(),
                    'updated_at' => now(),
                ],
```

Przykładowy seed administratora:

```
                [
                    'name' => 'Magdalena',
                    'email' => 'magdalena@email.com',
                    'password' => Hash::make('1234'),
                    'description' => "Her skin was a rich black that would
have peeled like a plum if snagged but then no one
                    would have thought of getting close enough to Mrs.
Flowers",
                    'gender' => 'female',
                    'pref_gender' => 'male',
                    'admin' => true,
                    'created_at' => now(),
                    'updated_at' => now(),
                ],
```

Jedynym seederem w aplikacji jest UserSeeder. W aplikacji randkowej najważniejszym jest by byli użytkownicy dlatego do aplikacji zostało dodanych czternastu unikalnych użytkowników.

- 4.2. Logika biznesowa i utrwalenie informacji w bazie
- 4.2.1. Rejestracja

Użytkownik ma możliwość stworzenia w aplikacji darmowego konta z którego może dowolnie korzystać z tego co aplikacja ma do zaoferowania. Kod odpowiedzialny za tworzenie nowego konta wygląda następująco:

```
    protected function validator(array $data)
    {
        return Validator::make($data, [
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'email', 'max:255',
'unique:users'],
            'password' => ['required', 'string', 'min:8', 'confirmed'],
            'description' => ['required', 'string', 'max:255'],
            'gender' => 'required|gender_diff',
            'pref_gender' => 'required|gender_diff',
        ], ['gender_diff' => Lang::get('auth.failed')]);
    }
```

Po sprawdzeniu czy podane informacje od użytkownika zgadzają się z przyjętym standardem aplikacja tworzy dla użytkownika konto i zapisuje jego informacje w bazie danych.

```
    protected function create(array $data)
    {
            return User::create([
                'name' => $data['name'],
                'email' => $data['email'],
                'password' => Hash::make($data['password']),
                'description' => $data['description'],
                'gender' => $data['gender'],
                'pref_gender' => $data['pref_gender'],
                'admin' => false,
            ]);
    }
```

- 4.2.2. Przeglądanie profili innych użytkowników

```
    public function index()
    {
        $id = auth()->user()->id;
        $user = User::getPairs($id);
        $chats = Chat::getChats($id);
        if ($user != null) {
            $u_id = $user['id'];
            $name = $user['name'];
            $gender = $user['gender'];
            $description = $user['description'];
        } else {
            $u_id = null;
            $name = 'No pairs left!';
            $gender = '';
            $description = 'There are no potential pairs left!';
        }
        session(['u_id' => $u_id]);
        session(['name' => $name]);
        return view('layouts.app', ['name' => $name, 'gender' =>
 $gender, 'description' => $description, 'chats' => $chats]);
    }
```

Dzięki funkcji index pokazywani są na stronie inni użytkownicy. Natomiast za system polubień odpowiada funkcja like:

```
public function like()
    {
        $id = auth()->user()->id;
        $u_id = session('u_id');
        $name = session('name');
        if (isset($_POST['like'])) {
            if (session('u_id') != null) {
                Interests::likeUser($id, $u_id);
                $paired = Interests::pairExists($id, $u_id);
                if ($paired == true) {
                    Chat::makeChat($u_id, $id);
                    Chat::makeChat($id, $u_id);
                    return redirect('app')->with('flash_message_success',
Lang::get('pairs.paired'));
                } else {
                    return redirect('app')->with('flash_message_success',
Lang::get('pairs.liked').$name);
                }
            } else {
                return redirect('/app')->with('flash_message_error',
Lang::get('pairs.failed'));
```

```
            }
        }
    }
```

Funkcja index sprawdza czy w bazie znajduje się jeszcze jakiś użytkownik który spełnia warunki na pokazanie się w aplikacji. Jeśli znajdzie się taki użytkownik to zostanie wyświetlone jego imię, płeć, oraz opis jego zainteresowań. Jeśli aplikacja nie znajdzie żadnego użytkownika zostanie wypisana na ekranie stosowna informacja o takiej sytuacji.

Funkcja like zostaje wywołana po wciśnięciu przez użytkownika guzika LIKE znajdującego się pod kartą z informacjami o innym użytkowniku. Po jej wywołaniu do bazy zostaje wysłana informacja o polubieniu nowego użytkownika. Jeśli właśnie polubiony użytkownik już wcześniej polubił użytkownika zalogowanego to zostanie wypisany komunikat o nowej parze i w prawej stronie aplikacji w sekcji czatu zostanie odblokowana możliwość komunikacji między tymi dwoma użytkownikami. Jeśli zostanie polubiony jakiś użytkownik nie zostanie on ponownie wyświetlony osobie która go polubiła.

W aplikacji zostały użyte takie funkcje jak m.in. likeUser, pairExists, getPairs, getChats. Poniżej zostaną po krótce opisane:

- 4.2.3. likeUser

```
    public static function likeUser($id, $u_id)
    {
        Interests::create([
            'user1_id' => $id,
            'user2_id' => $u_id
        ])->save();
    }
```

Prosta funkcja która tworzy nowy wpis o zainteresowaniu jednego użytkownika drugim.

- 4.2.4. pairExists

```
    public static function pairExists($id, $u_id)
    {
        $chatExists = false;
        $count = Interests::where(function (Builder $query) use ($id) {
                return $query->where('user1_id', $id)
                    ->orWhere('user2_id', $id);
            })
            ->where(function (Builder $query) use ($u_id) {
                return $query->where('user1_id', $u_id)
                    ->orWhere('user2_id', $u_id);
            })
            ->count();
        if ($count > 1) {
            $chatExists = true;
        }
        return $chatExists;
    }
```

Funkcja ta sprawdza czy dwoje użytkowników wyraziło swoje wzajemne zainteresowanie. Zwracana jest wartość true jeśli zainteresowanie istnieje a w przeciwnym wypadku false.

- 4.2.4. getPairs

```
public static function getPairs($id)
    {
        $user = User::select(['id', 'name', 'gender', 'description'])
            ->where('id', '!=', $id)
            ->whereNotIn('id', Interests::select('user2_id')
                ->where('user1_id', $id))
            ->whereNotIn('pref_gender', User::select('pref_gender')
                ->where('id', $id))
            ->get();
        if (sizeof($user) > 0) {

            return $user[rand(0, sizeof($user) - 1)];
        } else return null;
    }
```

Do funkcji podawane jest ID użytkownika dla którego funkcja ma znaleźć dostępne osoby do wyświetlenia w aplikacji. Warunkiem na wyświetlenie użytkownika1 w aplikacji używanej przez użytkownika2 jest by użytkownik2 nie wyraził jeszcze zainteresowaniem użytkownikiem1 oraz preferowana płeć użytkownika2 powinna się zgadzać płcią użytkownika1. Na końcu zwracany jest losowo 1 użytkownik spośród wszystkich możliwych.

- 4.2.5. getChats

```
    public static function getChats($id){
        return Chat::select('users.name', 'chats.sender_id')
        ->join('users', 'users.id', '=', 'chats.sender_id')
        ->where('chats.receiver_id', $id)
        ->where('chats.sender_id', '!=', $id)
        ->get('users.name', 'chats.sender_id')->toArray();
    }
```

Do funkcji podawane jest ID użytkownika dla którego funkcja ma znaleźć dostępne czaty. Zwracana jest tablica ID użytkowników z którymi dostępne są czaty oraz ich nazwy.

- 4.2.6. System czatu

Po utworzeniu pary dwóm użytkownikom odblokowuje się ich wzajemna komunikacja. System wyświetlania czatu jest w funkcji getMessages a wysyłanie wiadomości w createMessage.

- 4.2.7. getMessages

```
    public static function getMessages($sender_id, $receiver_id){
        $chat = Chat::getChatId($sender_id, $receiver_id);
        $sender_chat_id = $chat[0]->id;
        $chat = Chat::getChatId($receiver_id, $sender_id);
        $receiver_chat_id = $chat[0]->id;
        return Message::where('chat_id', $sender_chat_id)
        ->orWhere('chat_id', $receiver_chat_id)
        ->orderBy('created_at', 'ASC')
        ->get();
    }
```

Funkcja przyjmuje ID dwóch użytkowników pomiędzy odbywa się konwersacja. Zwraca wszystkie wiadomości które sobie wzajemnie wysłali.

- 4.2.8. createMessage

```
    public static function createMessage($sender_id, $receiver_id, $body){
        $chat = Chat::getChatId($sender_id, $receiver_id);
        $chat_id = $chat[0]->id;
        Message::create([
            'body' => $body,
            'read' => false,
            'created_at' => now(),
            'chat_id' => $chat_id,
        ])->save();
    }
```

Podobnie jak w funkcji getMessages funkcja przyjmuje ID dwóch użytkowników pomiędzy którymi odbywa się konwersacja w czacie, oraz dodatkowo przyjmuje treść wiadomości. Potem poprzez funkcję getChatId znajdywane jest ID czatu użytkowników a następnie tworzona jest nowa wiadomość która jest zapisywana w bazie danych.

- 4.2.8. getChatId

```
public static function getChatId($sender_id, $receiver_id){
        return Chat::where('sender_id', $sender_id)
        ->where('receiver_id', $receiver_id)
        ->get('id');
    }
```

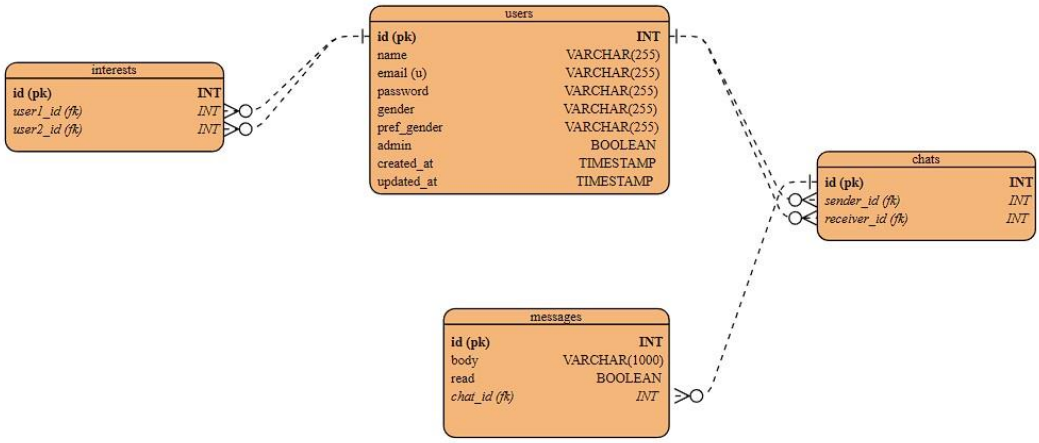
Funkcja przyjmuje ID dwóch użytkowników pomiędzy którymi odbywa się konwersacja i szuka w bazie danych ID czatu pasującego do ID tych użytkowników.

### 4.3. Routing

Po wpisaniu komendy php artisan route:list wyświetlane są trasowania aplikacji:

```
GET|HEAD  admin ..... AdminController@login
POST      admin ..... AdminController@login
GET|HEAD  admin/dashboard ..... AdminController@dashboard
GET|HEAD  admin/logout ..... AdminController@logout
GET|HEAD  admin/register ..... admin.register› AdminController@register
POST      admin/register ..... AdminController@register
GET|HEAD  admin/settings ..... AdminController@settings
GET|HEAD  admin/tables ..... AdminController@tables
GET|HEAD  admin/tables/deleteuser/{id} ..... AdminController@deleteUser
GET|HEAD  admin/tables/edituser/{id} ..... AdminController@editUser
POST      admin/tables/edituser/{id} ..... AdminController@editUser
GET|HEAD  register ..... register› Auth\RegisterController@showRegistrationForm
POST      register ..... Auth\RegisterController@register
```

## 5.Omówienie bazy danych




Baza składa się z czterech tabel z czego główną jest tabela users. Odpowiada ona za wszelkie informacje o użytkowniku jakie są potrzebne do działania aplikacji. W tabeli interests są gromadzone informacje o zainteresowaniu jednego użytkownika drugim. Tabela chats jest potrzebna do połączenia tabeli użytkowników z wiadomościami jakie są miedzy nimi prowadzone. Tabela messages ma w sobie zgromadzone wiadomości między użytkownikami oraz stan wiadomości (czyli czy jest lub nie jest odczytana przez drugiego użytkownika).

5.1 Migracje

Migracja to nazwa procesu który pozwala na bezkonfliktowe tworzenie i modyfikacje struktury bazy danych. Można ją przeprowadzić komendą : php artisan make:migration {nazwa migracji}

W pliku migracyjnym możemy zdefiniować parametry naszej nowej tabeli np. user_table:

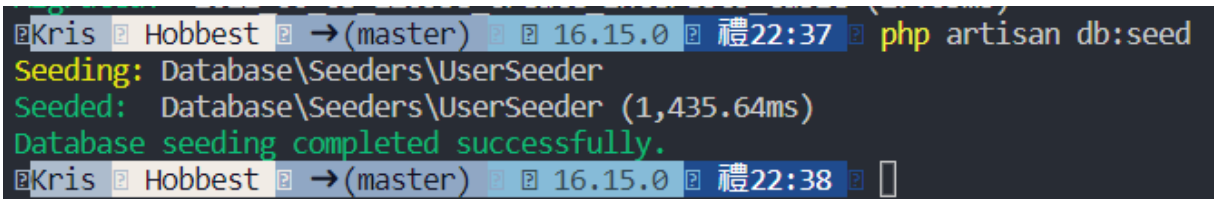
```php
public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->string('description', 1000);
            $table->string('gender');
            $table->string('pref_gender');
            $table->rememberToken()->nullable();
            $table->boolean('admin');
            $table->timestamps();
        });
    }
```

Żeby zatwierdzić migrację należy wpisać komendę php artisan migrate lub

php artisan migrate:fresh

```
Kris  Hobbest  →(master)  16.15.0  禮22:37  php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (34.17ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (21.92ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (33.40ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated:  2019_12_14_000001_create_personal_access_tokens_table (23.41ms)
Migrating: 2022_06_03_083846_create_chat_table
Migrated:  2022_06_03_083846_create_chat_table (25.48ms)
Migrating: 2022_06_03_101020_create_messages_table
Migrated:  2022_06_03_101020_create_messages_table (31.85ms)
Migrating: 2022_06_03_120958_create_interests_table
Migrated:  2022_06_03_120958_create_interests_table (27.63ms)
Kris  Hobbest  →(master)  16.15.0  禮22:37
```

Żeby wypełnić tabele danymi z seedera należy wpisać php artisan db:seed


Kris Hobbest →(master) 16.15.0 禮22:37 php artisan db:seed
Seeding: Database\Seeders\UserSeeder
Seeded: Database\Seeders\UserSeeder (1,435.64ms)
Database seeding completed successfully.
Kris Hobbest →(master) 16.15.0 禮22:38