**Стайлгайд по разработке SPA на Laravel и Vue.JS**

*SPA* (англ. Single Page Application – Одностраничное приложение) - это веб-приложение или веб-сайт, использующий единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript, обычно посредством AJAX.

*Laravel* — бесплатный веб-фреймворк с открытым кодом, предназначенный для разработки с использованием архитектурной модели MVC.

*Vue.js* — JavaScript-фреймворк с открытым исходным кодом для создания пользовательских интерфейсов.

**Примечание: перед ознакомлением с данным документом в обязательном порядке необходимо прочесть официальные документации по используемому инструментарию:**

<https://laravel.com/docs/5.4> – официальная документация по Laravel 5.4 (англ.)

<https://ru.vuejs.org/v2/guide/index.html> - официальная документация по Vue.js (рус.)

Альтернативы:

<https://laravel.ru/docs/v5> - народный перевод документации по Laravel 5.x (рус.)

**1. Пояснение структуры типичного SPA-проекта.**

**1.1 Роутинг**

|  |
| --- |
| Первичный флоу проекта на Laravel/Vue.js следует логике:  1. Запрос попадает на Laravel роутер.  2. Laravel возвращает бандл Vue.js.  3. vue-router подтягивает необходимый компонент по текущему URL. |

Для построения данной структуры были выполнены следующие действия:

1. Установлен npm-пакет vue-router.

|  |
| --- |
|  |
| *app.js* |

2. В единственном блейд-файле, куда редиректят все web-роуты, подключен собранный app.js файл и прописан тег <router-view></router-view>, внутри которого рендерится соответствующий url vue-компонент.

|  |
| --- |
|  |
| *app.blade.php* |

3. Далее создаются небходимые однофайловые vue-компоненты и привязываются к url с помощью router.js.

|  |
| --- |
|  |
| *router.js* |

4. Те же самые роуты прописываются в web-роутере Laravel, который на все url возвращает тот же самый blade-файл.

|  |
| --- |
|  |
| *web.php* |

5. vue-компоненты, по необходимости, обращаются к бэкэнду с помощью запросов axios. Мы для удобства используем обертку для запросов, чтобы автоматически привязать к хедеру токен авторизованного пользователя (см. spa-template/resources/assets/js/helpers/api.js)

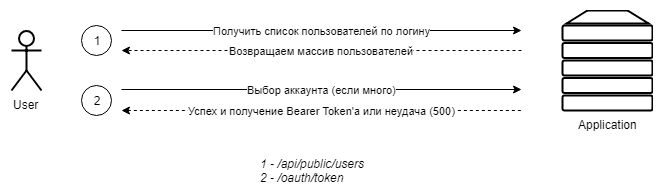
|  |
| --- |
|  |
| *users/Item.vue* |
|  |
| *api.php* |

**1.2 Пользователи и авторизация.**

Эндпоинт пользователей построен на использовании Laravel Passport.

1. В первую очередь при вводе логина и пароля проверяется наличие пользователя с подобным логином. В ответе содержится массив найденных пользователей.

2. Если пользователь один, то автоматически посылается запрос на получение токена и пользователь авторизуется в системе. Если несколько, ему необходимо выбрать аккаунт и передать для уточнения users.id.



Для распределения после авторизации пользователей на разные роуты (для ролевой дифференциации), редирект пишется в хук beforeEach.

|  |
| --- |
|  |
| *app.js* |

**2. Общие рекомендации по коду**

**2.1. Модели.**

Если один или несколько аксессоров приходится использовать повсеместно, хорошим тоном будет объявить parent-функцию *toArray()* и дополнить ее необходимыми значениями.

|  |
| --- |
| public function toArray()  {  $array = parent::toArray();  $array['pipeline'] = $this->pipeline; // это наш аксессор getPipelineAttribute()  return $array;  } |

Для реализации функции фильтрации, хорошим и проверенном вариантом будет использование scope-функций. Для этого необходимо передать в нее фильтры, пришедшие с запросом и прописать необходимые действия.

|  |
| --- |
| public function scopeFilter($query, $filters)  {  if(isset($filters['program\_id'])){  $query->where('program\_id', $filters['program\_id']);  }  if(isset($filters['not\_in\_ids']) && is\_array($filters['not\_in\_ids'])){  $query->whereNotIn('id', $filters['not\_in\_ids']);  }  if(isset($filters['incomplete']) && $filters['incomplete'] == 'true'){  $query->where('lesson\_type\_id', 1);  $query->whereHas('students', function ($query){  $query->havingRaw('COUNT(\*) < 5');  });  }  return $query;  } |

Если для какой-либо сущности необходимо создать ее историю взаимодействий, можно создать модель EntityHistory и дополнять в нее записи о каждом произведенном действии с ней. Для удобства в модели историй лучше будет сразу объявить функции добавления конкретных действий, чтобы не загромождать методы контроллера кодом добавления в сущность историй.

|  |
| --- |
| class GroupHistory extends Model  {  protected $fillable = ['value', 'group\_id'];  public static function insertStudents($group\_id, $student\_ids)  {  foreach ($student\_ids as $student\_id)  {  $student = Student::find($student\_id);  $history\_link = '<a href="/student/'.$student\_id.'">'.$student->user->name.'</a';  self::create([  'group\_id' => $group\_id,  'value' => 'Добавлен ученик '.$history\_link  ]);  }  }  } |

При разработке лога действий в типичной CRM-системе, можно столкнуться с проблемой размещения массивов. Наиболее корректным будет следующее решение:

1. Модель состоит из user\_id (актор), name (наименование действия), x\_ids (объекты действия), y\_ids (субъекты действия), value (дополнительная информация).

2. Если в x\_ids или y\_ids необходимо поместить несколько экземпляров сущности, то можно использовать синтаксис JSON-массива (“[‘1’,’2’,’3’]”).

3. В аксессорах на x\_ids и y\_ids прописываются правила подтягивания необходимой информации в зависимости от наименования действия.

|  |
| --- |
| public function getXIdsAttribute($value)  {  $return = [];  switch ($this->name){  case 'student\_create':  case 'student\_archive':  case 'student\_inarchive':  case 'student\_edit':  $student = Student::find($value);  $return = $student ? ['id' => $student->id, 'name' => $student->user->name] : ['id' => 0, 'name' => '[ученик удален]'];  break;  case 'teacher\_create':  case 'teacher\_edit':  case 'teacher\_archive':  case 'teacher\_inarchive':  $teacher = Teacher::find($value);  $return = $teacher ? ['id' => $teacher->id, 'name' => $teacher->user->name] : ['id' => 0, 'name' => '[преподаватель удален]'];  break;  case 'user\_create':  $user = User::find($value);  $return = $user ? ['id' => $user->id, 'name' => $user->name] : ['id' => 0, 'name' => '[пользователь удален]'];  break;  }  return $return ? $return : $value;  } |

**2.2. Контроллеры**

CRUD-методы: item(), items(), lightItem(), save(), delete().

item($id) – получение одного экземпляра сущности по id.

|  |
| --- |
| public function item($id, request $request)  {  $student = Student::where('id', $id)->with([  'user' => function ($q) {  $q->with('office', 'level');  },  'lesson\_type',  'program'  ])->first();  if ($student){  $student->setAppends([  'histories'  ]);  }  return $student;  } |

items(request $request) – получения списка (по указанным филтрам) (с пагинацией)

|  |
| --- |
| public function items(request $request)  {  $groups = Group::with([  'office',  'program',  'level',  'lesson\_type',  'lessons' => function ($q) {  $q->orderBy('datetime', 'asc');  },  'teacher' => function ($q) {  $q->with('user');  },  'students' => function ($q) {  $q->with('user');  $q->with('visit\_lessons');  },  'schedules'  ])  ->filter(Input::all())  ->orderBy('id', 'desc')  ->paginate(20);  return $groups;  } |

lightItem($id) – получение экземпляра с ограниченным набором параметров (как в items()) для атомарного обновления в списке.

|  |
| --- |
| public function lightItem($id, request $request)  {  $student = Student  ::where('id', $id)->with(  [  'user',  'program',  'schedules'  ]  )  ->first();  return $student;  } |

save(request $request) – объединенные методы create/update. Если в форме передается id – то обновляется существующая модель, если нет – создается новая.

|  |
| --- |
| public function save(request $request)  {  $this->validate($request, [  'office\_id' => 'required',  'program\_id' => 'required',  'level\_id' => 'required',  'lesson\_type\_id' => 'required',  'teacher\_id' => 'required',  'schedules' => 'required',  'default\_office\_cabinet\_id' => 'required',  'count\_lessons' => 'integer',  'cost' => 'integer',  'lesson\_duration' => 'numeric'  ]);  if ($request->get('id')) {  $group = Group::find($request->get('id'));  $group->update($request->all());  UserLog::insert('group\_edit', $request->get('id'));  }else{  $group = Group::create($request->all());  }  $group->updateSchedules($request->get('schedules'));  return response()->json(['status' => 'success', 'id' => $group->id], 200);  } |

delete($id) – удаление экземпляра сущности.

|  |
| --- |
| public function delete($id)  {  $group = Group::find($id);  $group->delete();  UserLog::insert('group\_delete', '', '', $id);  } |

Для оптимизации лучше всего использовать справочник, который бы подгружался после авторизации. Он объявлен в MainController.php. Если у вас имеются более-менее статичные, часто используемые сущности, добавьте их в метод data().

|  |
| --- |
| public function data(Request $request)  {  $data = [  'programs' => Program::all(),  'levels' => Level::all(),  'offices' => Office::with('cabinets')->get(),  'lesson\_types' => LessonType::all(),  'teachers' => $teachers->get(),  'tags' => TransactionTag::all()  ];  if($request->user()->hasRole('administrator')){  $data['roles'] = Role::all();  }  return $data;  } |

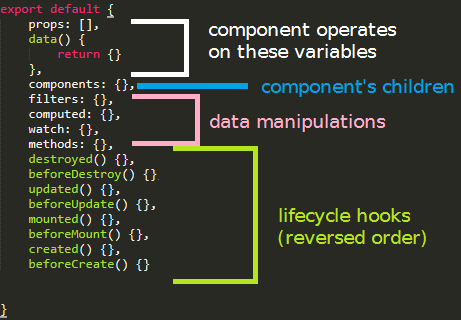
**2.3. Vue.JS**

Наименование компонентов строится по следующей логике:

Для смысловых модулей создается отдельная папка. В ней содержатся все vue-компоненты (вид списком, вид элемента, модальные окна и прочее).

|  |  |  |
| --- | --- | --- |
| Базовая структура | Структура с комплексным видом элемента и множеством функциональных модальных окон | Структура не CRUD-подобного модуля |
| **module\_name/**  **-List.vue** (вид списком)  **-Item.vue** (вид одного элемента)  **-Filter.vue** (фильтрация списка)  **-Form.vue** (модальное окно создания/редактирования элемента) | **module\_name/**  **-Item/** (блоки вида элемента)  **--Info.vue** (блок Item.vue)  **--History.vue** (блок Item.vue)  **--SomeBlock.vue** (блок Item.vue)  **-Modals/** (модальные окна)  **--SomeModal.vue** (модальное окно)  **-List.vue** (вид списком)  **-Item.vue** (вид одного элемента)  **-Filter.vue** (фильтрация списка)  **-Form.vue** (модальное окно создания/редактирования элемента) | **module\_name/**  **-SomeFolder/** (блоки)  **--SomeBlock.vue** (блок)  **-Modals/** (модальные окна)  **--SomeModal.vue** (модальное окно)  **-Index.vue** (основной вид)  **-Filter.vue** (фильтрация)  **-Form.vue**  (модальное окно создания/редактирования) |

*Примечание: модальное окно подтверждение удаления и другие нефункциональные модальные окна в виду отсутствия сложной логики помещается внутри родительского vue-компонента.*



*последовательность кода в компонентах*

Часто используемые методы лучше всего вынести в components/Common.vue, что позволит обращаться к ним через this.$common.someMethod(). Это сократит дублирование в коде. Точно так же можно создать и другие элементы, помимо Common.

Если необходимо подключить плагин без доступной обертки для Vue.js в npm, можно воспользоваться папками components и plugins.

В plugins при необходимости ложится исходник плагина, а подключается для использования в отдельном компоненте в папке components. Примеры таких решений уже можно там найти (DatePicker.vue, DateRangePicker.vue).