



Backend

Martijn Koch

////////////////////////////////////

////////////////////

Na overleg met het development team binnen Greenhouse is de keuze aan de backend kant op het PHP framewor Laravel komen te vallen. Omdat de backend veel functionaliteiten gaat krijgen is er gekozen voor een groot framework zoals Laravel. Er had gekozen kunnen worden voor een micoframework zoals SlimPhp of Lumen. Deze microframeworks zijn gericht op het maken van een API maar ook met een groter framework als Laravel is dit goed te doen. Tevens is Laravel op moment ook het meest populaire PHP framework (A. Kothari , 2019) en er is veel informatie over te vinden.

Hieronder worden een aantal backend hoofdzaken kort toegelicht met hiervan de bijbehorende code snippets. Voor alle backend code kan de volgende Github link worden bezocht.



Gebruiker registreren

De eerste stap in het ontwikkelen van de applicatie was om een nieuwe streamer te registreren. Niet zomaar iedereen kan een account aanmaken en dit zal alleen door een admin kunnen worden gedaan Er is gebruik gemaakt van Laravel passport voor de beveiliging van de applicatie. Laravel passport zorgt voor Oauth 2 authentication en het is een losse dependency van laravel met Composer is geïnstalleerd. Bij het aanmaken van een nieuwe user wordt een access token meegegeven. Het testen van de API is gedaan door middel van Postman.

```
/**
 * Register user
 */
public function register(Request $request)
{
    $validatedData = $request->validate([
        'name'=>'required|max:55',
        'email'=>'email|required|unique:users',
        'password'=>'required|confirmed'
    ]);

    $validatedData['password'] = bcrypt($request->password);

    $user = User::create($validatedData);
    $accessToken = $user->createToken('authToken')->accessToken;

    return response(['user'=> $user, 'accessToken'=>$accessToken]);
}
```

```
1  {
2  "user": {
3      "name": "John",
4      "email": "test5@user.com",
5      "updated_at": "2019-12-23 14:18:18",
6      "created_at": "2019-12-23 14:18:18",
7      "id": 3
8  },
9  "accessToken":
10     "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOiIiXWianRjIjoieWYiWTM0ZmY2ZjNmOTAyOGQ0NzQyZGE4NDkyMTc2YzkyMTBjZmExZTRkM2Y4ZGJhNDNlOTlhdD0mNmM1NmNkOTQ0TR1ZWY3ZWYyY5YjgiLCJpYXQ0IjE1NzcxMTA2OTg5Im51ZiIEMTU3NzExMDY5OCwiZXhwIjojA4NzMzMdk4LCJzdWI0IiIiIiwic2NvGvczIjpbXXZl.
eFXefEPFCZKGi2WNPA6Y7WgCEpuoKRBHqDSCilBmcB2PuvuAJa99mV13k2_9D62rla1UMxTxdZ7PUjQsfnWJqrdkR_Nad0cNkgUha6L6fX6cF993wFFE3klb0eReyzMk6ARn
32EfUfX4awuQ683c7K7Xok1qXgpwriAeqYLKHC8Q0WUChEA14m3EXnwOdWozrIYw7iNlQApN7IjqLeWflsdHqum7TBuvcaA0Klb07RWMc9JVBxgbSgZuEKsa21P2YvSphlWssMY
iyo2qX34nn0V2jfNUCbWRFpRfTCnuo5q-mw47iWl6ZuAQ2f1-K991HA0Ng80ZiH0LA42JfJai-BG4sNONBvq-R0YRoik-4GLuStioLd0LQy4b-4gASySEbiqExWYLTg-VMI1w
RhSGzgnrr3t_ss0c-fsRGepUwMtbbdP2YlnoJ0sdQoP1e8cZ9mYEZEDrTQxqWpoK4FFEdirwA25fs_hL4ePiG83GeM3AU0FELVm2p1fzJ-NUSAIFopLJtpvY2w_Qbc0hhqph
0SXgMUJ23IDwDw_bkXSWe47zJaZr6fNbU068SkrrJ5dpmK3BR6Uw20Hyd41HUCRM6mQbZ7_z_uiy_djwIMF83fu-IWs0wEnxG6XrZWNoLN7EP3hf0HFSXNkBSrJ8cXaqZ8U1l
ETnd80vw-uPKw"
```

```

47  /**
48   * Post a new game
49   */
50  public function store(Request $request)
51  {
52      // Required data
53      $this->validate($request, [
54          'name' => 'required',
55          'image' => 'image|mimes:jpeg,png,jpg,gif,svg|max:2048'
56      ]);
57
58      if(!$request->hasFile('image')) {
59          return response()->json(['upload_file_not_found'], 400);
60      }
61      $file = $request->file('image');
62      if(!$file->isValid()) {
63          return response()->json(['invalid_file_upload'], 400);
64      }
65
66      //Get original file name
67      $originalName = $file->getClientOriginalName();
68      // Get just filename
69      $filename = pathinfo($originalName, PATHINFO_FILENAME);
70      // Get the ext
71      $extension = $file->getClientOriginalExtension();
72      // Filename to store
73      $fileNameToStore = $filename.'_'.time().'.'.$extension;
74      // Path to store
75      $path = $file->storeAs('/images', $fileNameToStore);
76      $file_path = 'images/';
77      // Move the image
78      $file->move($file_path, $fileNameToStore);
79
80      // Create the new game
81      $game = new Game;
82      $game->name = $request->input('name');
83      $game->image = $path;
84
85      $game->save();
86
87      //Response message
88      $response = [
89          'success' => true,
90          'data' => $game,
91          'message' => 'Game stored successfully.'
92      ];
93      return response()->json($response, 200);
94  }

```

Reclame toevoegen

Een nieuwe reclame toevoegen lijkt veel op het toevoegen van een nieuwe game. Echter moet er hier geen image worden toegevoegd, maar een video file. Onder de rules wordt gekeken of de file wel toegestaan is. Aan de hand van de ingevoerde data wordt de filenaam gegenereerd. Deze krijgt de video game naam, het scenenummer en de naam van de klant. Mocht de klant een nieuwe reclame hebben voor een video game scene, dan kan deze gemakkelijk worden overschreven en ook de streamer hoeft niets goed te zetten in zijn streaming software. Als je Toto als voorbeeld neemt: dan is er dagelijks wel een voetbalwedstrijd en de tekst moet vaak worden vernieuwd. Het is dan alleen een kwestie van de nieuwe te uploaden en de content op de stream past zich vanzelf aan.

```
22  /**
23   * Store a ad
24   */
25  public function store(Request $request)
26  {
27      //Rules
28      $rules = [
29          'gamename' => 'required',
30          'game_id' => 'required',
31          'scenenummer' => 'required',
32          'clientname' => 'required',
33          'file' => 'required|mimetypes:video/x-ms-asf,video/x-flv,video/mp4,application/x-mpegURL,video/MP2T,
video/3gpp,video/quicktime,video/x-msvideo,video/x-ms-wmv,video/avi',
34      ];
35
36      //Error messages
37      $messages = [
38          'gamename' => 'A game name is required',
39          'game_id' => 'A game id is required',
40          'scenenummer' => 'A scene number is required',
41          'clientname' => 'A client name is required',
42          'file' => 'A file is required'
43      ];
44
45      //Validate the data
46      $validator = $this->validate($request, $rules, $messages);
47
48      //Handle file uoload
49      $file = $request->file('file');
50
51      // Get the ext
52      $extension = $file->getClientOriginalExtension();
53      $fileNameToStore = $request->input('gamename').'-'.
54          $request->input('scenenummer').'-'.
55          $request->input('clientname').'-'.
56          $extension;
57      $path = $file->storeAs('/videos', $fileNameToStore);
58      $file_path = 'videos/';
59      $file->move($file_path, $fileNameToStore);
60
61      // Store the data
62      $ad = new Ad;
63      $ad->gamename = $request->input('gamename');
64      $ad->game_id = $request->input('game_id');
65      $ad->scenenummer = $request->input('scenenummer');
66      $ad->clientname = $request->input('clientname');
67      $ad->file = $path;
68
69      $ad->save();
70
71      //Response message
72      $response = [
73          'success' => true,
74          'data' => $ad,
75          'message' => 'Ad stored successfully.'
76      ];
77
78      return response()->json($response, 200);
79  }
```


One to many relation

Een belangrijk onderdeel van de API is om een one to many relation op te stellen. Zo moet een game meerdere reclames kunnen hebben. Een game heeft meerdere scenario's wanneer een bepaalde reclame kan worden afgespeeld. In Laravel is het opzetten van een one to many relatie vrij gemakkelijk. In de model van de game kan worden vastgesteld, dat een game meerdere reclames en scènes heeft.

```
15 // Create the one to many relationship for ads
16 public function ads() {
17     return $this->hasMany(Ad::class);
18 }
19
20 // Create the one to many relationship for scenes
21 public function scenes() {
22     return $this->hasMany(Scene::class);
23 }
```

API

De API heeft verschillende paden. Deze paden kunnen door de front-end worden aangesproken. Vladimir Sidorenko (2017) heeft een Best Practices RESTful API Design guide. Deze guide is gevolgd om de definities van de Api paden aan te duiden. Zo zijn een aantal korte regels voor API paden:

Simplicity
Fast performance
Fully-featuring
Reliability
Scalability

```
22 // User controller
23 Route::post('/register', 'Api\AuthController@register');
24 Route::post('/login', 'Api\AuthController@login');
25
26 // Game controller
27 Route::get('/games', 'Api GameController@index');
28 Route::get('/games/mostpopular', 'Api GameController@mostPopular');
29 Route::get('/games/{id}', 'Api GameController@show');
30 Route::get('/games/{id}/ads-one', 'Api GameController@showAdsOne');
31 Route::get('/games/{id}/ads-two', 'Api GameController@showAdsTwo');
32 Route::get('/games/{id}/ads-three', 'Api GameController@showAdsThree');
33 Route::get('/games/{id}/scene-one', 'Api GameController@showSceneOne');
34 Route::get('/games/{id}/scene-two', 'Api GameController@showSceneTwo');
35 Route::get('/games/{id}/scene-three', 'Api GameController@showSceneThree');
36 Route::post('/games', 'Api GameController@store');
37 Route::delete('/games/{id}', 'Api GameController@destroy');
38
39 //Ad controller
40 Route::post('/advertisements', 'Api AdController@store');
41
42 //Scene controller
43 Route::post('/scenes', 'Api SceneController@store');
```

Bronnen

Bronnenlijst

Laravel. Version 6.6.1 [Framework] - The PHP Framework For Web Artisans. (2019) Geraadpleegd van <https://laravel.com/>

Kothari, A. (2019). Top 7 PHP Frameworks for Better Application Development. Geraadpleegd van <https://geekflare.com/php-frameworks/>

Postman (2019). Version 5.5.4 [Software]. Geraadpleegd van <https://www.getpostman.com/>

Laravel Passport. Version 8.0 [Library]. (2019) Geraadpleegd van <https://laravel.com/docs/5.8/passport>

Composer. Version 1.8.5 [Software]. (2019) Geraadpleegd van <https://getcomposer.org/>

Guzzle, PHP HTTP client. Version 6.5 [Library]. (2019) Geraadpleegd van <http://docs.guzzlephp.org/en/stable/>

Sidorenko , V. (2017, 1 oktober). RESTful API Design: Best Practices. Geraadpleegd op 31 december 2019, van <https://gearheart.io/blog/restful-api-design-best-practices/>