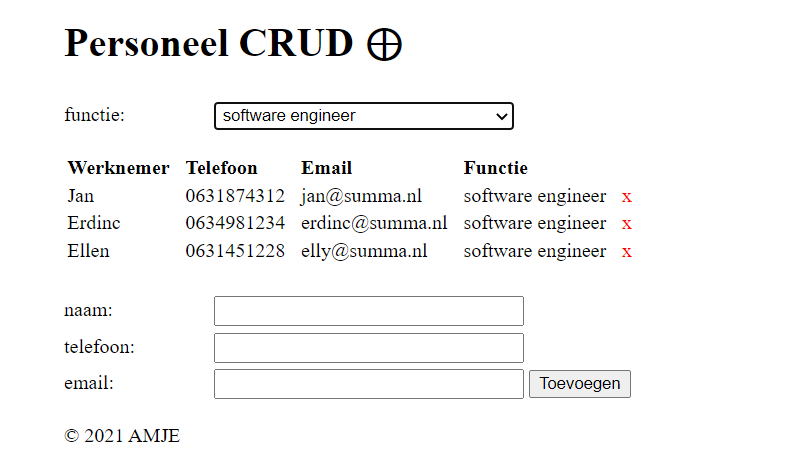
# API1: Handboek

## Inleiding

In deze herhaling vind je de stappen terug die je geleerd hebt bij het maken van een API met Laravel en het gebruik ervan in een front end applicatie.

Naast Laravel heb je in API1 ook gezien hoe je een API kunt maken met Express en PHP. Deze 2 smaken behandel ik hier niet expliciet.



## Planning

|  |  |  |  |
| --- | --- | --- | --- |
| Actie | Datum | Planning | Werkelijk |
| 1. Beschrijf Applicatie die je wilt maken en de eisen en wensen die je eraan stelt | 2-mei | 1:00u | 0:30u |
| 1. Ontwerp de api's die de applicatie nodig heeft. | 2-mei | 0:30u | 1:00u |
| 1. Ontwerp het datamodel | 2-mei | 1:00u | 1:00u |
| 1. Maak de database | 2-mei | 0:30u | 0:30u |
| 1. Maak de API's, models en route | 2-mei | 1:30u | 3:00u |
| 1. Maak een seeder voor de database | 2-mei | 1:00u | 1:00u |
| 1. Voeg logging toe | 2-mei | 1:00u | 1:00u |
| 1. Maak een featuretest voor de database | 2-mei | 1:30u | 1:20u |
| 1. Maak je applicatie | 2-mei | 2:00u | 1:50u |
| 1. Test de applicatie | 2-mei |  |  |
| 1. Lever applicatie op | 8-mei | 0:10u |  |
| Totaal |  | 10:00u | 11:10u |

## Eisen en Wensen

Er moet een applicatie voor het administreren van de medewerkers. Van medewerkers wordt de naam, het telefoonnummer, emailadres en de functie vastgelegd.

De applicatie toon een overzicht van alle medewerkers die een bepaalde functie hebben. Deze functie kun je uit een combobox kiezen.

Met de applicatie kun je ook medewerkers verwijderen en toevoegen.

**Must have**

* Create Read Delete beheerfuncties voor alle medewerkers
* Kiezen van een functie waarna alle medewerkers worden getoond met de gekozen functie.

**Shoud have**

* Update van de gegevens van een medewerker

**Could have**

**Won't have**

* -

## API ontwerp

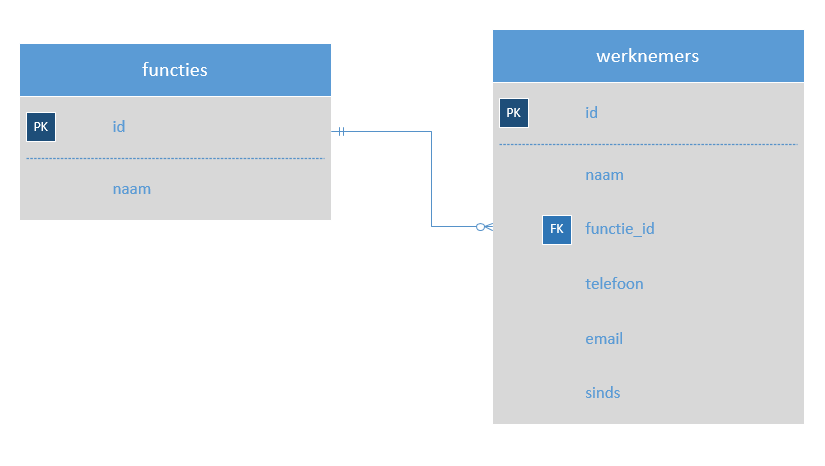
*Opmerking: Er staan hier meer api's dan nodig is op basis van de eisen en wensen (vet gedrukt is nodig). Zorg er voor dat wanneer jij een FE voor een opdracht maakt, je ook de extra api's kunt ontwikkelen.*

|  |  |  |  |
| --- | --- | --- | --- |
| methode | url | json parameters | json resultaat |
| **get** | **api/werknemers/** |  | **Alle werknemers** |
| get | api/werknemers/{id} |  | werknemer met opgegeven id |
| **post** | **api/werknemers/** | **naam, telefoon, email, functie\_id** | **gecreëerde werknemer** |
| patch | api/werknemers/{id} | naam en/of telefoon en/of email en/of functie\_id | werknemer met opgegeven id |
| **delete** | **api/werknemers/{id}** |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| methode | url | json parameters | json resultaat |
| **get** | **api/functies/** |  | **Alle functies** |
| get | api/functies/{id} |  | functie met opgegeven id |

|  |  |  |  |
| --- | --- | --- | --- |
| methode | url | json parameters | json resultaat |
| get | api/werknemers/namen/{naam} |  | Alle werknemers met de tekst naam in de naam |
| get | api/werknemers?naam={naam} |  | Alle werknemers met de tekst naam in de naam |
| get | api/werknemers?sort={veld} |  | Alle werknemers in volgorde van veld |
| get | api/functies/{id}/werknemers |  | Werknemers met de functie\_id gelijk aan id |
| get | api/functies/{id}/werknemers?sort={veld} |  | Werknemers met de functie\_id gelijk aan id in volgorde van veld |
| delete | api/functies/{id}/werknemers |  |  |

## Datamodel ontwerp



|  |  |  |  |
| --- | --- | --- | --- |
| Table | Functies |  |  |
| Kolom | Type |  |  |
| Id | tinyint(4) | Key, required | Autoincrement |
| Naam | varchar(20) | Required |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Table | Werknemers |  |  |
| Kolom | Type |  |  |
| id | tinyint(4) | Key, required | Autoincrement |
| naam | varchar(50) | Required |  |
| functie\_id | tinyint(4) |  | FK Functies |
| telefoon | varchar(20) |  |  |
| email | varchar(50) |  |  |
| sinds | date |  |  |

## Maak de database

*Hier is dat gedaan met een migration in laravel:* php artisan migrate

|  |
| --- |
| php artisan make:migration functies |
| *Migration staat in: apipersoneel\database\migrations* |
| public function up()  {      Schema::create('functies', function (Blueprint $table) {          $table->tinyInteger('id')->autoIncrement();          $table->string('naam', 20)->nullable(false)->unique();      });  }  public function down()  {      Schema::dropIfExists('functies');  } |

|  |
| --- |
| php artisan make:migration werknemers |
| *Migration staat in: apipersoneel\database\migrations* |
| public function up()  {      Schema::create('werknemers', function (Blueprint $table) {          $table->tinyInteger('id')->autoIncrement();          $table->string('naam', 50)->nullable(false);          $table->tinyInteger('functie\_id');          $table->string('telefoon', 20)->nullable(true);          $table->string('email', 50)->nullable(true);          $table->date('sinds')->nullable(true);          $table->foreign('functie\_id')->references('id')->on('functies')->restrictOnDelete();      });  }  public function down()  {      Schema::dropIfExists('werknemers');  } |

## Maak de API's, models en route

|  |
| --- |
| php artisan make:controller FunctieController --api --model=Functie |
| *Model aanpassen: staat in: apipersoneel\app\models* |
| class Functie extends Model  {      use HasFactory;      public $timestamps = false;      protected $fillable = ['naam'];      // protected $with = ['werknemers'];      public function werknemers()      {          return $this->hasMany(Werknemer::class, 'functie\_id', 'id');      }  } |
| *Route toevoegen: staat in: apipersoneel\routes\api.php* |
| Route::apiResource('functies', FunctieController::class)  ->parameters(['functies'=>'functie'])  ->only(['index', 'show']); |
| *Controller staat in: apipersoneel\app\http\controllers* |
| public function index()  {      return Functie::all();      // return Functie::with('werknemers')->get();  } |

|  |
| --- |
| php artisan make:controller WerknemerController --api --model=Werknemer |
| *Model aanpassen: staat in: apipersoneel\app\models* |
| <?php  namespace App\Models;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Database\Eloquent\Model;  class Werknemer extends Model  {      use HasFactory;      public $timestamps = false;      protected $fillable = ['naam', 'functie\_id', 'telefoon', 'email', 'sinds'];      public function functie()      {          return $this->belongsTo(Functie::class, 'functie\_id', 'id');      }  } |
| *Route toevoegen: staat in: apipersoneel\routes\api.php* |
| Route::apiResource('werknemers', WerknemerController::class);  Route::get('functies/{id}/werknemers', [WerknemerController::class, 'indexFunctie']);  Route::delete('functies/{id}/werknemers', [WerknemerController::class, 'destroyFunctie']); |
| *Controller staat in: apipersoneel\app\http\controllers* |
| <?php  namespace App\Http\Controllers;  use App\Models\Werknemer;  use Illuminate\Http\Request;  use Illuminate\Support\Facades\Validator;  class WerknemerController extends Controller  {      /\*\*       \* Display a listing of the resource.       \*       \* @return \Illuminate\Http\Response       \*/      public function index(Request $request)      {          if ($request->has('naam')){              return Werknemer::where('naam', 'like', '%'.$request->naam.'%')->get();          }          if ($request->has('sort')){              return Werknemer::orderBy($request->sort)->get();          }          return Werknemer::All();          // return Werknemer::with('functie')->get();      }      /\*\*       \* Store a newly created resource in storage.       \*       \* @param  \Illuminate\Http\Request  $request       \* @return \Illuminate\Http\Response       \*/      public function store(Request $request)      {         // dd($request->all());          $validator = Validator::make($request->all(), [              'email' => 'email',              'naam' => 'required'          ]);          if ($validator->fails()) {              //header('Content-Type: application/json');              return response('{"Foutmelding":"Data niet correct"}', 400)->header('Content-Type','application/json');          }          else return Werknemer::create($request->all());      }      /\*\*       \* Display the specified resource.       \*       \* @param  \App\Models\Werknemer  $werknemer       \* @return \Illuminate\Http\Response       \*/      public function show(Werknemer $werknemer)      {          return $werknemer;      }      /\*\*       \* Update the specified resource in storage.       \*       \* @param  \Illuminate\Http\Request  $request       \* @param  \App\Models\Werknemer  $werknemer       \* @return \Illuminate\Http\Response       \*/      public function update(Request $request, Werknemer $werknemer)      {          $validator = Validator::make($request->all(), [              'naam' => 'required',              'email' => 'email',          ]);          if ($validator->fails()) {              return response('{"Foutmelding":"Data niet correct"}', 400)->header('Content-Type','application/json');          }            $werknemer->update($request->all());          return $werknemer;      }      /\*\*       \* Remove the specified resource from storage.       \*       \* @param  \App\Models\Werknemer  $werknemer       \* @return \Illuminate\Http\Response       \*/      public function destroy(Werknemer $werknemer)      {          $werknemer->delete();      }      public function indexFunctie(Request $request, $id)      {          if ($request->has('sort'))              return Werknemer::where('functie\_id',$id)->orderBy($request->sort)->get();          return Werknemer::where('functie\_id',$id)->get();      }        public function destroyFunctie($id)      {          Werknemer::where('functie\_id', $id)->delete();      }  } |

|  |
| --- |
| Extra router als fall back |
| Route::fallback(function(){      return response()->json([          'message' => 'Page Not Found. If error persists, contact info@website.com'], 404);  }); |

## Maak een seeder voor de database

|  |
| --- |
| php artisan make:seeder FunctiesSeeder |
| <?php  namespace Database\Seeders;  use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  use Illuminate\Support\Facades\DB;  class FunctiesSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*       \* @return void       \*/      public function run()      {          DB::table('functies')->insert(['naam' => 'software engineer',]);          DB::table('functies')->insert(['naam' => 'software architect',]);          DB::table('functies')->insert(['naam' => 'tester',]);          DB::table('functies')->insert(['naam' => 'sales',]);          DB::table('functies')->insert(['naam' => 'manager',]);          DB::table('functies')->insert(['naam' => 'field engineer',]);      }  } |

|  |
| --- |
| php artisan make:seeder WerknemersSeeder |
| <?php  namespace Database\Seeders;  use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  use Illuminate\Support\Facades\DB;  class WerknemersSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*       \* @return void       \*/      public function run()      {          DB::table('werknemers')->insert([              'naam' => 'Jan',              'functie\_id' => 1,              'telefoon' => '0631874312', 'email' => 'jan@summa.nl', 'sinds' => '2002-11-01'          ]);            DB::table('werknemers')->insert([              'naam' => 'Erdinc',              'functie\_id' => 1,              'telefoon' => '0634981234', 'email' => 'erdinc@summa.nl', 'sinds' => '1989-11-01'          ]);            DB::table('werknemers')->insert([              'naam' => 'Carla',              'functie\_id' => 5,              'telefoon' => '0634120975', 'email' => 'carla@summa.nl', 'sinds' => '1993-05-01'          ]);            DB::table('werknemers')->insert([              'naam' => 'Ellen',              'functie\_id' => 1,              'telefoon' => '0631451228', 'email' => 'elly@summa.nl', 'sinds' => '1995-08-15'          ]);            DB::table('werknemers')->insert([              'naam' => 'Vijay',              'functie\_id' => 2,              'telefoon' => '0631784466', 'email' => 'vijay@summa.nl', 'sinds' => '1997-08-01'          ]);            DB::table('werknemers')->insert([              'naam' => 'Piet',              'functie\_id' => 6,              'telefoon' => '0631552341', 'email' => 'piet@summa.nl', 'sinds' => '1999-07-01'          ]);            DB::table('werknemers')->insert([              'naam' => 'Juma',              'functie\_id' => 3,              'telefoon' => '0631224550', 'email' => 'juma@summa.nl', 'sinds' => '2000-09-01'          ]);      }  } |

|  |
| --- |
| <?php  namespace Database\Seeders;  use Illuminate\Database\Console\Seeds\WithoutModelEvents;  use Illuminate\Database\Seeder;  class DatabaseSeeder extends Seeder  {      /\*\*       \* Seed the application's database.       \*       \* @return void       \*/      public function run()      {          $this->call([              FunctiesSeeder::class,              WerknemersSeeder::class,          ]);      }  } |

*NB: Alternatief is dat je een SQL script runt (niet uitgewerkt):*

|  |
| --- |
| <?php  namespace Database\Seeders;  use Illuminate\Database\Seeder;  class DatabaseSeeder extends Seeder  {      public function run()      {          exec("C:\\xampp\mysql\bin\mysql --user=root --password= --host=localhost --database personeel < ..\personeel.sql");          // sql-bestand zit 1 niveau dieper dan de projectmap          // C:\\ ipv C:\ omdat \x betekent iets      }  } |

## Voeg logging toe

|  |
| --- |
| *Controller staat in: apipersoneel\app\http\controllers* |
| …  class WerknemerController extends Controller  {  …      public function store(Request $request)      {          Log::info('werknemers toevoegen', ['ip' => $request->ip(), 'data' => $request->all()]);          $validator = Validator::make($request->all(), [              'email' => 'email',              'naam' => 'required'          ]);          if ($validator->fails()) {              Log::error("werknemers toevoegen Fout");              //header('Content-Type: application/json');              return response('{"Foutmelding":"Data niet correct"}', 400)->header('Content-Type','application/json');          }          else return Werknemer::create($request->all());      }  …      public function update(Request $request, Werknemer $werknemer)      {          Log::info('werknemers wijzigen', ['ip' => $request->ip(), 'oud' => $werknemer, 'nieuw' => $request->all()]);          $validator = Validator::make($request->all(), [              'naam' => 'required',              'email' => 'email',          ]);          if ($validator->fails()) {              Log::error("werknemers wijzigen Fout");              return response('{"Foutmelding":"Data niet correct"}', 400)->header('Content-Type','application/json');          }            $werknemer->update($request->all());          return $werknemer;      }  …      public function destroy(Werknemer $werknemer)      {          Log::info('werknemers verwijderen', ['data' => $werknemer]);          $werknemer->delete();      }  …  } |
| *NB: Kan uitgebreider*  *Log vind je standaard in bestand: storage/logs/laravel.log* |

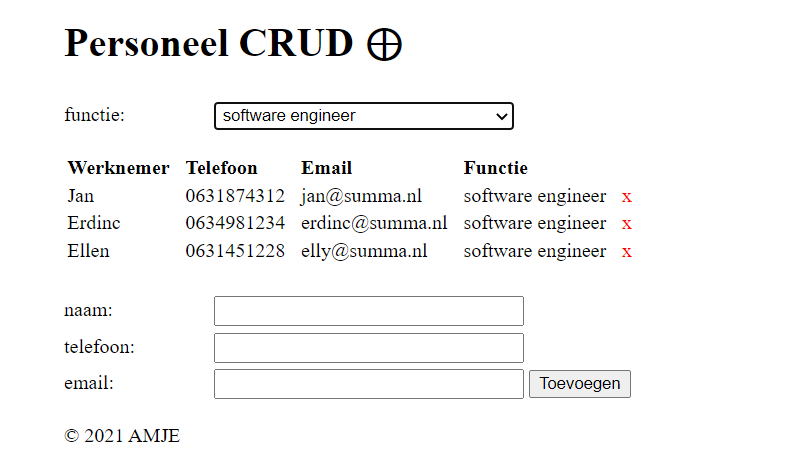
## Maak een featuretest voor de database

|  |
| --- |
| php artisan make:test PersoneelTest |
| *Feature test staat in test/feauture* |
| <?php  namespace Tests\Feature;  use Illuminate\Foundation\Testing\RefreshDatabase;  use Illuminate\Foundation\Testing\WithFaker;  use Tests\TestCase;  class PersoneelTest extends TestCase  {      /\*\*       \* Test dat api/werknemers/{id} werkt (GET)       \* @return void       \*/      public function test\_werknemer\_op\_id()      {          $response = $this->get('api/werknemers/1');          $response->assertStatus(200);          $response->assertJson(['naam'=>'Jan','email'=>'jan@summa.nl', ]);      }      /\*\*       \* Test dat api/werknemers?naam={naam} (GET)       \* @return void       \*/      public function test\_werknemer\_op\_naam()      {          $response = $this->get('api/werknemers?naam=J');          $response->assertStatus(200);          $response->assertJsonCount(3);          $response->assertJsonFragment(['naam'=>'Jan','email'=>'jan@summa.nl', ]);      }       /\*\*       \* Test dat api/werknemers werkt, om werknemer toe te voegen (POST)       \* @return void       \*/      public function test\_insert\_werknemer()      {          // Maak werknemer          $data = ['naam'=>'Karel', 'email'=> 'karel@summa.nl', 'functie\_id' => 1];          // gebruik de API          $response = $this->json('POST','api/werknemers', $data);          // stel vast dat de database in de tabel werknemers de toegevoegde werknemer heeft          $this->assertDatabaseHas('werknemers',                  ['naam'=>'Karel','email'=>'karel@summa.nl', ]);          $response->assertStatus(201);          $response->assertJson(['naam'=>'Karel','email'=>'karel@summa.nl', ]);      }       /\*\*       \* Test dat api/werknemers werkt, om werknemer toe te voegen (DELETE)       \* @return void       \*/      public function test\_delete\_werknemer()      {          $response = $this->delete('api/werknemers/7');          $response->assertStatus(200);      }  } |
| Uitvoeren van de test: php artisan test |

## Maak je front-end applicatie

### Beschrijving

De front-end applicatie is gemaakt met HTML/CSS en JavaScript



De combobox is een <select>-tag.

De werknemers staan in een tabel <table>

De gegevens van een nieuwe werknemers staan in een aantal <input>-tags

Zodra het programma is geladen:

* worden eenmalig alle functies uit de database ingelezen. Hierbij wordt de api *api/functies* gebruikt. De opgehaalde functies:
  + worden in de hulptabel gezet, zodat ze later gebruikt kunnen worden om de naam van een functie te tonen bij een werknemers (in de tabel met werknemers staat de naam van de functie).
  + Worden omsloten door een <option>-tag in de <select> tag gezet.
* Worden alle werknemers waarvan de functie\_id de waarde heeft die in de combobox is geselecteerd. Hierbij wordt de api *api/functies/{id}/werknemers* gebruikt. Van de medewerkers worden de naam, telefoon, email getoond, en de omschrijving van functie. De omschrijving wordt uit opgehaald uit de hulptabel. Er wordt hier van een hulptabl gebruik gemaakt, om te voorkomen dat er onnodig veel api-calls uitgevoerd moeten worden.

Zodra een gebruiker in de combobox een andere functie kiest, wordt de tabel met werknemers leeggemaakt en opnieuw gevuld: alle werknemers waarvan de functie\_id de waarde heeft die in de combobox is geselecteerd. Dit gebeurd, zoals hiervoor beschreven.

Wordt op de knop Toevoegen gedrukt, dan wordt een nieuwe medewerker toegevoegd. Hierbij wordt de api *api/werknemers* gebruikt i.c.m. met een POST method. De gegevens van de werknemer staan in de body van het request: naam, telefoon, email uit het input-tag, functie\_id uit de geselecteerde functie van de combobox.

### Code

|  |
| --- |
| <!DOCTYPE html>  <html lang="nl" xmlns:mso="urn:schemas-microsoft-com:office:office" xmlns:msdt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882">  <head>      <title> Personeel CRUD+</title>      <meta charset="utf-8" />      <style>          div, footer, form {margin-top: 20px;}          input, select {margin-top: 5px; width: 40%; padding: 2px;}          body  {width : 80%; max-width:600px; margin: auto; margin-top:50px;}          th,td {text-align : left; padding-right:10px;}          div {width : 100%;}          label {width: 20%;display: inline-block;}          td:last-child {color : red; cursor: pointer;}      </style>      <script src="https://unpkg.com/axios/dist/axios.min.js"></script>  </head> |
| Style is niet in een apart bestand gezet. Dat zou wel moeten!  Voor de api-calls wordt gebruik gemaakt van axios. |
| <body onload="laad()">      <h1>Personeel CRUD &oplus;</h1>      <div>          <label for="functie">functie:</label>  <select id="functie" onclick="laadFunctieWerknemers()"></select>      </div>      <br />      <table>          <tr>              <th>Werknemer</th>              <th>Telefoon</th>              <th>Email</th>              <th>Functie</th>          </tr>          <tbody id="tabelInhoud" />      </table>      <div>          <label for="naam">naam:</label><input type="text" id="naam" required /><br />          <label for="telefoon">telefoon:</label><input type="text" id="telefoon" /><br />          <label for="email">email:</label><input type="email" id="email" />          <button onclick="voegToe()" type="button">Toevoegen</button>      </div>      <footer>&copy; 2021 AMJE</footer>  </body> |
| * Function laad wordt hierna uitgewerkt: laad alle gegevens * De Id van de combobox is functie. Deze id heeft de function *laadFunctieWerknemers* nodig om de waarde van de geselecteerde <option>-tags op te halen. * De functie *laadFunctieWerknemers* leest alle werknemers waarvan de functie\_id gelijk is aan de id uit de combobox. * De body van de tabel heeft als id *tabelInhoud* . Deze id heeft *laadFunctieWerknemers* om medewerkers in te zetten. * De id's functie, naam, telefoon en email heeft de function *voegToe* nodig om de gegevens van de toe te voegen werknemer samen te stellen * De function *voegToe* voegt een nieuwe werknemer toe |
| <script>      "use strict"      const apiBasis = "http://127.0.0.1:8000/api"      const apiWerknemers = apiBasis + "/werknemers"      const apiFuncties = apiBasis + "/functies"        let functies = []      let json = null |
| * Functies bevat een array waarin alle namen van functies uit de database worden gezet. Van functie 4 wordt de naam gezet in functies[4]. Bij de werknemer staat zijn functie in de kolom *functie\_id*. Als je de naam van zijn functie wilt tonen, staat deze in *functies[functie\_id]*. |
| const laadFuncties = async () => {      const response = await axios.get(apiFuncties)      const json = await response.data      let nieuweInhoud = ''      json.map(el => {          functies[el.id] = el.naam          nieuweInhoud += `<option value="${el.id}">${el.naam}</option>`      })      document.querySelector("#functie").innerHTML = nieuweInhoud      // console.log(functies)  } |
| De function *laadFuncties* leest alle functies in. Hiervoor wordt gebruik gemaakt van een api.  Het resultaat bevat een JSON tabel waarvan de rijen in de tabel *functies* worden gezet en die bovendien in een <option>-tag worden. Alle <option>-tags worden in de combobox (<select>) met id *functie* gezet |
| const laadFunctieWerknemers = async () => {      const functie = document.querySelector("#functie").value      // console.log('selecteer Functie ', functie)      const apiFunctiesWerknemers = `${apiFuncties}/${functie}/werknemers?sort=naam`      const response = await axios.get(apiFunctiesWerknemers)      json = await response.data      toon()  } |
| De function *laadFunctieWerknemers* leest alle werknemers met een bepaalde functie in en sorteert ze op naam. Hiervoor wordt gebruik gemaakt van een api.  Het resultaat bevat een JSON tabel die in de variabele json wordt gezet. Daarna worden de werknemers met de function *toon* in weergegeven |
| const toon = () => {      let tabelInhoud = ''      json.map(el => tabelInhoud += `<tr><td>${el.naam}</td><td>${el.telefoon}</td><td>${el.email}</td>                      <td>${functies[el.functie\_id]}</td><td onclick="verwijder(${el.id})"> x </td></tr>`)      document.querySelector("#tabelInhoud").innerHTML = tabelInhoud  } |
| De function *toon*, vult een string met tabelrijen (<tr>-tags). Elke tabelrij bevat gegevens van een werknemer (*el*) en een kolom x.  Deze kolom x heeft een function *verwijder* die uitgevoerd wordt als er op geclickt wordt (*onclick*). In *el.id* staat de id van de werknemer die verwijderd wordt.  De string wordt tenslotte toegevoegd aan de tabel met id *tabelinhoud*. |
| const laad = async () => {      await laadFuncties()      await laadFunctieWerknemers()  } |
| De function laad wordt uitgevoerd nadat de site volledig is geladen. Eerst worden alle functies ingelezen met de function laadFuncties. Daarna worden de werknemers ingelezen die dezelfde functie hebben als de functie die in de combobox is geselecteerd. |
| const voegToe = async () => {      var werknemer = {};      werknemer.naam = document.querySelector("#naam").value;      werknemer.telefoon = document.querySelector("#telefoon").value;      werknemer.email = document.querySelector("#email").value;      werknemer.functie\_id = document.querySelector("#functie").value;      // console.log("voegtoe: ", JSON.stringify(werknemer))      const respons = await axios.post(apiWerknemers, JSON.stringify(werknemer), { headers: { 'Content-Type': 'application/json' } })      document.querySelector("#naam").value = ''      document.querySelector("#telefoon").value = ''      document.querySelector("#email").value = ''      await laadFunctieWerknemers()  } |
| De function *voegtoe* voegt met een POST method een nieuwe werknemer toe. Eerst wordt een JSON gemaakt met de gegevens van de werknemer. Deze JSON wordt daarna als argument aan de api doorgegespeeld. Nadat de werknemer is toegevoegd, worden de ingevulde gegevens op het scherm leeg gemaakt en wordt de tabel met werknemers opnieuw gevuld met de function *laadFunctieWerknemers*. Als het goed is, is de toegevoegde werknemer dan te zien. |
| const verwijder = async (id) => {      // console.log("verwijder: ", id)      const respons = await axios.delete(apiWerknemers+"/"+ id)      // console.log('status code', respons.status)      await laadFunctieWerknemers()  } |
| De function *verwijder* verwijdert met een DELETE method de werknemer waarvan de id in de parameter staat. Na het verwijderen wordt de function *laadFunctieWerknemers* uitgevoerd, zodat de tabel met werknemers wordt bijgewerkt: als het goed is, is de verwijderde werknemer dan niet meer zichtbaar. |
| </script>  </html> |

## Test de applicatie end-to-end

Maak testrapport

## Lever applicatie op

Demonstratie en uitleg code

## Overzicht gebruikte Laravel commando's

|  |  |
| --- | --- |
| Maken laravel project ApiPersoneel | laravel new ApiPersoneel |
| Maken database personeel | XAMPP: database apipersoneel maken  Laravel Env bevay al de juiste databasenaam |
| Maken migrations | php artisan make:migration functies  php artisan make:migration werknemers |
| Uitvoeren migration | php artisan migrate  php artisan migrate:rollback |
| Maken seeder | php artisan make:seeder FunctiesSeeder  php artisan make:seeder WerknemersSeeder |
| Alle seeders runnen | php artisan db:seed  php artisan migrate:fresh --seed |
| FeatureTest maken | php artisan make:test PersoneelTest |
| Testen uitvoeren | php artisan test |