

Distelvinklaan 22

2660 Hoboken

+32 3 830 4105

[www.cvoAntwerpen.be](file:///E:\Bibliotheken\MIJN%20DOCUMENTEN\Opleiding%20Informatica\63F\A3%20Communicatie%20en%20organisatietechnieken\2012-2013\www.cvoAntwerpen.be)

[info@cvoantwerpen.be](mailto:info@cvoantwerpen.be)

**HBO Informatica**

**Module B3 Programmeren 4**

**TV 11727**

**Projectdossier**

**Instant Interview**

**Dieter Benoot**

**Cedric Jacobs**

**David Vlaminck**

Academiejaar 2014 – 2015

# Inhoudsopgave

[1 Inhoudsopgave 3](#_Toc421467319)

[2 Inleiding 5](#_Toc421467320)

[3 Projectvereisten 6](#_Toc421467321)

[3.1 Concept 6](#_Toc421467322)

[3.2 Concreet 6](#_Toc421467323)

[3.3 Gebruikte concepten en frameworks 6](#_Toc421467324)

[4 Analyse 7](#_Toc421467325)

[4.1 ERD 7](#_Toc421467326)

[4.2 Use cases 7](#_Toc421467327)

[4.2.1 Maken van een pdf na het selecteren van een functie 7](#_Toc421467328)

[4.2.2 Een nieuwe vraag toevoegen 8](#_Toc421467329)

[5 Indeling van het project 9](#_Toc421467330)

[5.1 Mappenstructuur 9](#_Toc421467331)

[5.2 Map app 9](#_Toc421467332)

[5.3 Map BLL 9](#_Toc421467333)

[5.4 Map DAL 9](#_Toc421467334)

[5.5 Map temp 9](#_Toc421467335)

[5.6 Map vendor 10](#_Toc421467336)

[5.7 Map web 10](#_Toc421467337)

[5.8 Composer & namespaces 10](#_Toc421467338)

[5.8.1 Aanpassing aan autoload\_real.php 10](#_Toc421467339)

[5.8.2 Toevoeging van autoload\_II.php 10](#_Toc421467340)

[6 Opbouw 11](#_Toc421467341)

[6.1 Index.php en Javascript 11](#_Toc421467342)

[6.2 MVC 11](#_Toc421467343)

[6.3 BLL 11](#_Toc421467344)

[6.4 DAL 11](#_Toc421467345)

[7 Uitbreiding 12](#_Toc421467346)

[7.1 Uitbreiding blijft mogelijk 12](#_Toc421467347)

[7.2 Use case diagram 12](#_Toc421467348)

# Inleiding

Dit document is het dossier van het project Instant Interview. Dit project is het resultaat van de opdracht die tijdens het vak Programmeren 4 (PHP) gegeven is. In dit document worden de verschillende aspecten van dit project beschreven en gedocumenteerd.

Eerst wordt een beschrijving gemaakt van de projectvereisten. Daarna volgt de analyse die ons bij dit project geholpen heeft. Vervolgens wordt de structuur van de code uitgelegd en tot slot dieper ingegaan op de concepten en hoe dit in de code uitgewerkt is.

Notities:

1. Presentatie
   1. Je stelt voor hoe je je project hebt opgebouwd waarbij je uitlegt hoe je OOP, n-tier, MVC en Tests hebt toegepast.
   2. Je geeft een demo.

# Projectvereisten

## Concept

Het concept Instant Interview werd bedacht door Cedric. Hij doet interviews met kandidaten voor een sollicitatie. Hierbij gebruikt hij een lijst met vragen die gerangschikt staan per competentie in een Excel lijst. Verder heeft hij ook manier om voor een bepaalde functie een vaste lijst vragen te laten opstellen. Maar deze manier van werken is weinig overzichtelijk, niet gebruiksvriendelijk en moeilijk te onderhouden. Bovendien wil hij dit eventueel op termijn verder uitbouwen naar een online formulier waar hij ook meteen de antwoorden van de kandidaat kan invullen tijdens het interview. Daarom hebben wij, Cedric, Dieter en ik, een deel van dit concept uitgewerkt als project voor dit vak.

## Concreet

Dit project is een PHP project met een MySQL database. Sommige van de keuzes die hier gemaakt zijn, zijn omwille van de mogelijkheden of beperkingen die daar bij horen. Zo is beslist om dit project een single page website te maken. Aangezien de gebruiker vragen zal aanvinken of keuzes maken is het niet wenselijk veel van pagina te veranderen.

De gebruiker kan naar de verschillende delen op de website gaan, maar blijft op 1 pagina. Het Instant Interview gedeelte zelf wordt ook binnen deze ene pagina opgebouwd. De gebruiker krijgt een keuzelijst te zien waaruit een functie kan geselecteerd worden. Wanneer de gebruikt een keuze maakt, worden verschillende competenties weergeven die bij deze functie horen, de anderen worden voorlopig verborgen. Er worden al een aantal vragen aangeduid, maar de gebruiker is vrij om meer vragen aan te duiden. De vragen worden zichtbaar telkens er op een competentie geklikt wordt.

De gebruiker kan alle competenties zien door op een knop te klikken. Wanneer hij op de “Rapport” knop klikt, verschijnt in een kleiner bovenliggend venster een overzicht van de vragen die aangeduid zijn. Wanneer in dat venster op de “Download PDF” knop wordt geklikt, genereert de server een pdf met de aangeduide vragen en zal de browser dit bestand downloaden.

De laatste mogelijkheid die de gebruiker heeft, is het toevoegen van een vraag. Hiervoor klikt hij op de “Voeg vraag toe” knop die een kleiner venster opent. Hier kan men een competentie uit een keuzelijst selecteren en een vraag in een tekstvak typen. De vraag wordt dan aan de geselecteerd competentie toegevoegd. Hiervoor wordt de pagina wel vernieuwd en verdwijnt de selectie van vragen.

## Gebruikte concepten en frameworks

De volgende zaken worden in dit project gebruikt:

* Bootstrap
* jQuery
* FPDF (netasign)
* Composer
* Helper klassen uit AnOrmApart
* MVC

# Analyse

## ERD

Het is essentieel voor projecten zoals deze om een ERD op te stellen. In onderstaande figuur zijn de 3 entiteiten voor dit project aanwezig.



Figuur 1: ERD

Bij het omzetten van dit model naar een database diagram krijgen we 1 tabel per entiteit en 1 extra tussentabel voor de veel op veel relatie tussen entiteiten Vraag en Functie.

## Use cases

Deze twee use cases zijn de voornaamste bewerkingen die een gebruiker zal uitvoeren op deze website.

### Maken van een pdf na het selecteren van een functie

|  |  |
| --- | --- |
| Naam | Maken van een pdf na het selecteren van een functie |
| Samenvatting | De gebruiker zal een functie selecteren en vervolgens de pdf laten genereren en dan downloaden |
| Actoren | De gebruiker, het systeem |
| Precondities | De gebruiker bevindt zich in het Instant Interview gedeelte op de website. De keuzelijst bevat de mogelijke functies. |
| Verloop | 1. De gebruiker selecteert een functie uit de keuzelijst. 2. Het systeem genereert het deel van de webpagina met competenties en vragen en selecteert de vragen die bij deze functie horen. De competenties zonder geselecteerde vragen zijn verborgen 3. De gebruiker klikt op de knop “Rapport”. 4. Een bovenliggend scherm verschijnt en bevat een overzicht van geselecteerde vragen. 5. De gebruiker klikt op “Download PDF”. 6. Het systeem genereert een PDF. 7. Het systeem stuurt het bestand door naar de browser, die het automatisch downloadt. |
| Postcondities | De gebruiker heeft het bestand op de downloadlocatie staan. |
| Alternatieve wegen | 5a. De gebruiker klikt op “Sluiten”. Het bovenliggende scherm sluit en de gebruiker kan verder vragen selecteren. |

Figuur 2: Use case Maken van een pdf na het selecteren van een functie

### Een nieuwe vraag toevoegen

|  |  |
| --- | --- |
| Naam | Een nieuwe vraag toevoegen |
| Samenvatting | De gebruiker kan een nieuwe vraag toevoegen bij een competentie |
| Actoren | De gebruiker, het systeem |
| Precondities | De gebruiker bevindt zich in het Instant Interview gedeelte op de website. |
| Verloop | 1. De gebruiker klikt op de knop “Voeg vraag toe”. 2. Een bovenliggend scherm verschijnt en bevat een keuzelijst met alle competenties. 3. De gebruiker selecteert een competentie. 4. De gebruiker typt een vraag in het tekstvak. 5. De gebruiker klikt op de knop “Voeg toe”. 6. Het systeem voegt de vraag toe in de database en geeft op de pagina weer dat de bewerking gelukt is. 7. 3 seconden laten wordt de pagina vernieuwd zodat de nieuwe vraag kan worden aangeduid. |
| Postcondities | De gebruiker bevindt zich in het Instant Interview gedeelte en de nieuwe vraag staat in de database. |
| Alternatieve wegen | 3a. De gebruiker selecteert geen competentie.  3b. Na het klikken op “Voeg toe” zal het systeem een foutmelding laten zien aan de gebruiker.  4a. De gebruiker typt geen vraag in het tekstvak.  4b. Na het klikken op “Voeg toe” zal het systeem een foutmelding laten zien aan de gebruiker.  5a. De gebruiker klikt op “Sluiten”. Terug naar de beginsituatie. |

Figuur 3: Use case Een nieuwe vraag toevoegen

# Indeling van het project

## Mappenstructuur

Het project heeft een aantal mappen om het overzicht en de structuur te bewaren.

|  |
| --- |
| * Root   + app     - controller     - models     - views   + BLL   + DAL     - Helpers   + temp   + vendor     - composer     - setasign   + web     - css     - fonts     - img     - js |

Figuur 4: Mappenstructuur

Deze worden nu kort toegelicht.

## Map app

De map app bevat de MVC structuur. Daarom zijn er ook 3 submappen controller, models en views. Deze hebben elk hun eigen namespaces respectievelijk Controller, Models en Views. De MVC structuur zelf wordt verder toegelicht in een later hoofdstuk.

## Map BLL

Deze map bevat een aantal business logic klassen en ook een aantal php pagina’s waarnaar verwezen kan worden om een taak te laten uitvoeren door de server. De klassen zitten in namespace BLL.

## Map DAL

De DAL map bevat voornamelijk klassen die nodig zijn om met de database te kunnen verbinden en er bewerkingen mee te doen. In de submap Helpers zitten helper klassen uit AnOrmApart. De klassen in de DAL map zitten in namespace DAL.

## Map temp

Wanneer een pdf bestand gegenereerd wordt, kan deze niet rechtstreeks naar de browser als download gestuurd worden omdat dit deel functionaliteit wordt aangeroepen met een POST call via ajax. Daarom worden de gegenereerde pdf’s op de server in een map temp opgeslagen. Elke keer een pdf gemaakt wordt, zullen oudere pdf’s verwijderd worden.

## Map vendor

De vendor map bevat extra packages die we in dit project nodig hebben. Er wordt gebruik gemaakt van FPDF van setasign. Dit wordt door de dependency manager composer (zie [5.8](#_Composer)) verzorgd en dus is ook een map composer bij de vendors.

## Map web

De map web bevat html elementen en de index.php waarmee de opbouw van de webpagina start. Er is de volgende onderverdeling voor submappen:

* css: bevat de bestanden met CSS voor de website. Hier zit ook het bootstrap css bestand.
* fonts: bevat de lettertypes van bootstrap.
* img: het bestand loading.gif zit in deze map om een loading animatie te kunnen gebruiken.
* js: In deze map zitten de javascript bestanden en dus ook een deel van bootstrap.

## Composer & namespaces

In dit project wordt er gebruikt gemaakt van composer, een dependency manager. Na de installatie van de standaard composer zijn de bestanden aangepast.

### Aanpassing aan autoload\_real.php

De getLoader() functie van autoload\_real.php werd aangepast, het volgende stuk code is toegevoegd zodat zelf gemakkelijk de namespaces kunnen beheerd worden in apart bestand.

|  |
| --- |
| $map = **require *\_\_DIR\_\_*** . **'/autoload\_II.php'**; **foreach** ($map **as** $namespace => $path) {  $loader->setPsr4($namespace, $path); } |

Figuur 5: aanpassing aan autoload\_real.php

### Toevoeging van autoload\_II.php

Dit bestand bevat de definitie van een aantal namespaces. Door in de bestanden telkens de autoloader te gebruiken, worden de definities van de namespaces omgezet en wordt telkens het juiste klasse bestand gebruikt.

|  |
| --- |
| **<?php**  $vendorDir = *dirname*(*dirname*(***\_\_FILE\_\_***)); $baseDir = *dirname*($vendorDir); $appDir = *realpath*($baseDir . **'/app/'**); $dalDir = *realpath*($baseDir . **'/DAL/'**); $bllDir = *realpath*($baseDir . **'/BLL/'**);  **return array**(  **'Controller\\'** => **array**($appDir . **'/controller'**),  **'Models\\'** => **array**($appDir . **'/models'**),  **'Views\\'** => **array**($appDir . **'/views'**),  **'DAL\\'** => **array**($dalDir),  **'BLL\\'** => **array**($bllDir), ); |

Figuur 6: Toevoeging van autoload\_II.php

# Opbouw

## Index.php en Javascript

### Head

De head van de index.php pagina bevat de standaard elementen die daar te verwachten zijn:

* Meta tags
* Bootstrap viewport code
* Stylesheet links naar de css bestanden
* Links naar de source bestanden van jQuery en Ajax.

### Pagina-indeling

In de body van index.php staat bovenaan een carousel element. Daaronder staat een menu waarmee je snel naar de verschillende paginaonderdelen kan gaan. Dit wordt door bootstrap gedaan en vraagt dus geen verdere uitleg. Verder staan de paginaonderdelen er zelf op, zoals een info gedeelte met wat info en eentje met contactgegevens.

### Div interviewForm

Dit div element bevat heel weinig maar heel belangrijke code, namelijk:

|  |
| --- |
| **<?php require '../app/views/JobFunctionsView.php'**; **?>** |

Figuur 7: stukje code uit index.php

Dit zal de JobFunctionsView code op deze plaats in de pagina plaatsen.

### Modals

Op deze index.php staan ook nog twee div elementen met class modal. Deze “report-modal” en “add-question-modal”. Dit zijn de bovenliggende vensters die zullen verschijnen wanneer de gebruiker een vraag wil toevoegen of een pdf wil maken van de geselecteerde vragen.

## MVC

### Models

Er zijn in dit MVC model drie entiteiten waar een model van geïmplementeerd wordt. Deze implementatie is vrij vanzelfsprekend en eenvoudig.

#### Question

|  |
| --- |
| **namespace** models;  **class** Question {  **private $id**;   **public function** setId($id) {  $this->**id** = $id;  }   **public function** getId() {  **return** $this->**id**;  }   **private $fullQuestion**;   **public function** setFullQuestion($fullQuestion) {  $this->**fullQuestion** = $fullQuestion;  }   **public function** getFullQuestion() {  **return** $this->**fullQuestion**;  }   **private $competenceId**;   **public function** setCompetenceId($competenceId) {  $this->**competenceId** = $competenceId;  }   **public function** getCompetenceId() {  **return** $this->**competenceId**;  }   **public function** \_\_construct($id, $fullQuestion, $competenceId)  {  $this->**id** = $id;  $this->**fullQuestion** = $fullQuestion;  $this->**competenceId** = $competenceId;  } } |

Figuur 8: Model Question

#### Competence

|  |
| --- |
| **namespace** Models;  **class** Competence {  **private $id**;  **public function** setId($id) {  $this->**id** = $id;  }   **public function** getId() {  **return** $this->**id**;  }   **private $name**;  **public function** setName($name) {  $this->**name** = $name;  }   **public function** getName() {  **return** $this->**name**;  }   **public function** \_\_construct($id, $name)  {  $this->**id** = $id;  $this->**name** = $name;  } } |

Figuur 9: Model Competence

#### JobFunction

|  |
| --- |
| **namespace** Models;  **class** JobFunction {  **private $id**;  **public function** setId($id) {  $this->**id** = $id;  }   **public function** getId() {  **return** $this->**id**;  }   **private $name**;  **public function** setName($name) {  $this->**name** = $name;  }   **public function** getName() {  **return** $this->**name**;  }   **public function** \_\_construct($id, $name)  {  $this->**id** = $id;  $this->**name** = $name;  } } |

Figuur 10: Model JobFunction

### Views

#### JobFunctionsView

Er zijn twee views in deze MVC. De eerste is een eenvoudige view die een keuzelijst maakt met daarin de alle job functies.

|  |
| --- |
| <**form**>  <**div class="form-group col-xs-12 col-md-6"**>  <**h2**>Selecteer hier een functie.</**h2**>  <**select class="form-control" id="jobfunction-select" onchange="***jobFunctionSelectChanged*()**"**>  <**option selected disabled hidden value=''**></**option**>  **<?php foreach** ($controller->getJobFunctions() **as** $jobFunction) { *// iterate through all the JobFunctions and make an option in the dropdown for each function* **?>** <**option  value="<?php echo** $jobFunction->getId()**?>"** >**<?php echo** $jobFunction->getName() **?>** </**option**>  **<?php** } **?>** </**select**>  </**div**> </**form**> <**div id="fetch-competences-div"**></**div**> |

Figuur 11: JobFunctionsView

Door een foreach loop te maken met een array die aan de controller gevraagd wordt, is de keuzelijst snel ingevuld met al de mogelijke opties. Deze opties krijgen als waarde de ID mee van de functie en als tekst de naam van functie.

De eerste option die geen waarde heeft en attribuut hidden krijgt, is een trucje om er voor te zorgen dat er niets in de keuzelijst staat met een waarde wanneer deze op de pagina verschijnt. Dit kan ook in javascript gedaan worden maar op deze manier is het eenvoudiger en sneller.

Op het einde wordt er ook een div element klaar gezet dat zal opgevuld worden met de competenties en vragen.

#### jobFunctionSelectChanged()

Aan de gegenereerde keuzelijst in JobFunctionsView is een script gekoppeld dat uitgevoerd wordt elke keer de gebruiker de waarde van deze keuzelijst wijzigt. Dit stukje code zal Ajax gebruiken om de volgende view op te halen en in de pagina te plaaten.

Eerst verbergt dit mogelijk de knoppen “Voeg vraag toe”, “Rapport” en “Laat alles zien”, omdat deze na het succesvol uitvoeren van dit script zichtbaar worden. De eerste keer is dit dus overbodig, maar daarna niet meer.

Er verschijnt ook een “Loading…” boodschap. Dit verschijnt zolang Ajax nog geen antwoord heeft gekregen. De ID van de geselecteerde functie wordt meegegeven naar de CompetencesView pagina (zie [6.2.2.3](#_ComptencesView)). Wanneer Ajax het antwoord terug krijgt, zal deze de “Loading…” boodschap overschrijven en de knoppen (terug) zichtbaar maken.

|  |
| --- |
| **function** *jobFunctionSelectChanged*() {  *HideButtons*();  **var** cdiv = $(**"#fetch-competences-div"**);  cdiv.html(**"<img src='img/loading.gif' width='30'/><span>Loading...</span>"**); *// making a loading effect* **var** select = **document**.getElementById(**"jobfunction-select"**);  **var** id = select.**options**[select.**selectedIndex**].**value**; *// gets the JobFunction Id* $.ajax({  **type**: **"GET"**,  **url**: **"../app/views/CompetencesView.php"**,  **data**: {**q**: id},  success: **function** (msg) {  cdiv.html(msg);$(**"#modal-report-button"**).removeClass(**"hidden"**);  $(**"#show-all-button"**).removeClass(**"hidden"**);  $(**"#add-question-button"**).removeClass(**"hidden"**);  },  error: **function** () {  alert(**"failure"**);  }  }); } |

Figuur 12: jobFunctionSelectChanged() uit script.js

#### CompetencesView

Deze view bouwt met twee loops de competenties op met bijhorende vragen en duidt de vragen die bij de geselecteerde functie horen meteen aan.

Eerst wordt de Query string verwerkt. Indien $q een waarde bevat, wordt de controller aangesproken met $q als parameter. Dit zorgt er voor dat de nodige arrays gevuld worden (zie [6.2.3.2](#_loadDataByFunction($functionId))).

Daarna wordt de eerste loop begonnen die alle competenties gaat doorlopen en per competentie een Accordion element (Bootstrap) zal maken. De titel van dit element is de naam van de competentie. Als de ID van de competentie niet in de $competencesToShow array van de controller kan gevonden worden, dan start de accordion ingetrokken. De ID van de competentie wordt ook gebruikt om de bijhorende vragen te krijgen met een controller method (selectQuestionsByCompetenceId())en om die accordion een unieke naam te geven.

Er wordt over deze selectie van vragen ook een foreach loop gemaakt die per vraag een keuzevakje met bijhorend label maakt. Het label krijgt de feitelijke vraag als inhoud en hoort bij het keuzevakje dat een unieke naam krijgt omdat de ID van de vraag er in verwerkt wordt. Bij het plaatsen van het keuzevakje wordt ook de controller array $questionsMarked nagekeken. Als de ID van de vraag in die array zit, wordt het keuzevakje al aangevinkt door het attribuut checked te plaatsen.

|  |
| --- |
| **include**(**'../../vendor/autoload.php'**);  *header*(**'Content-type: text/html; charset=UTF-8'**) ;  $controller = **new** Controller\Controller(); *parse\_str*($\_SERVER[**'QUERY\_STRING'**]); **if** (**isset**($q)) {  $controller->loadDataByFunction($q);  }**?>** <**div class="col-xs-12" id="questions"**>  <**div class="panel-group" id="accordion" role="tablist" aria-multiselectable="true"**>  **<?php foreach** ($controller->getCompetences() **as** $competence) { $questions = $controller->selectQuestionsByCompetenceId( $competence->getId()); **?>** <**div class="panel panel-primary  <?php if** (!*array\_key\_exists*($competence->getId(), $controller->getCompetencesToShow())) { **echo ' collapse'**;} **?>  " id="questionssection<?php echo** $competence->getId() **?>"**>  <**div class="panel-heading" role="tab" id="heading-<?php echo** $q; **?>"**>  <**h4 class="panel-title"**>  <**a class="collapsed" data-toggle="collapse" data-parent="#accordion"  href="#<?php echo 'competence-id-'** . $competence->getId(); **?>"**>  **<?php echo** $competence->getName(); **?>** </**a**>  </**h4**>  </**div**>  <**div id="<?php echo 'competence-id-'** . $competence->getId(); **?>" class="panel-collapse collapse" role="tabpanel"**>  **<?php foreach** ($questions **as** $question) { **?>** <**div class="input-group"**>  <**span class="input-group-addon"**>  <**input type="checkbox" class="questionsCheck" id="question-<?php echo** $question->getId(); **?>"  <?php if** (*array\_key\_exists*($question->getId(), $controller->getQuestionsMarked()))   { **echo ' checked'**; }   </**span**>  <**div class="list-group-item"**>  <**label for="question-<?php echo** $question->getId(); **?>"**>  **<?php echo** $question->getFullQuestion(); **?>**  </**label**>  </**div**>  </**div**>  **<?php** } **?>** </**div**>  </**div**>  **<?php** } **?>** </**div**> </**div**> |

Figuur 13: CompetencesView

### Controller

Deze klasse haalt data op via een DAL klasse en houdt deze bij in arrays zodat ze in views kan gebruikt worden.

|  |
| --- |
| **namespace** Controller;  **class** Controller {  **protected $questions** = **array**();  **protected $competences** = **array**();  **protected $jobFunctions** = **array**();  **protected $questionsMarked** = **array**();  **protected $competencesToShow** = **array**();  **protected $context**;   **public function** \_\_construct()  {  $this->**context** = **new** \DAL\InterviewContext();  }   **public function** getQuestions()  {  **return** $this->**questions**;  }   **public function** getCompetences()  {  **return** $this->**competences**;  }   **public function** getQuestionsMarked()  {  **return** $this->**questionsMarked**;  }   **public function** getCompetencesToShow()  {  **return** $this->**competencesToShow**;  }   **private function** getContext()  {  **return** $this->**context**;  } …  } |

Figuur 14: Controller basis

#### getJobFunctions()

Deze methode wordt gebruikt om alle jobfuncties uit de database te halen via de selectAllFunctions() methode. Met het resultaat wordt een JobFunction object gemaakt. Al deze objecten worden in een array bijgehouden en ook meteen gereturnd.

De eerste waarde in het resultaat wordt omgezet naar een getal, omdat de JobFunction ID een integer is.

|  |
| --- |
| **public function** getJobFunctions() {  *// uses a stored procedure that gets all job functions* $return = $this->getContext()->selectAllFunctions();  $result = **array**();  **foreach** ($return **as** $value) {  *array\_push*($result, **new** \Models\JobFunction(*intval*(value[**'Id'**]), $value[**'Naam'**]));  }  $this->**jobFunctions** = $result;  **return** $this->**jobFunctions**; } |

Figuur 15: getJobFunctions uit Controller

#### loadDataByFunction($functionId)

LoadDatabyFunction is een methode die 4 private methods aanspreekt. Als parameter wordt een functie ID meegegeven, die wordt doorgegeven aan loadQuestionToMark

|  |
| --- |
| **public function** loadDataByFunction($functionId) { $this->loadQuestions();  $this->loadCompetences();  $this->loadQuestionsToMark($functionId);  $this->fillCompetencesToShow(); } |

Figuur 16: loadDataByFunction uit Controller

#### loadQuestions()

Deze methode wordt gebruikt om alle vragen uit de database te halen via de selectAllQuestions() methode. Met het resultaat wordt een Question object gemaakt. Al deze objecten worden bijgehouden in de $questions array. De positie van het element is ook de ID van het element om het object sneller te kunnen opvragen.

|  |
| --- |
| **public function** loadQuestions() {$return = $this->getContext()->selectAllQuestions();  **foreach** ($return **as** $value) {  $this->**questions**[*intval*($value[**'Id'**])] = **new** \Models\Question( *intval*($value[**'Id'**]), $value[**'Vraag'**], *intval*($value[**'CompetentieId'**]));  } } |

Figuur 17: loadQuestions uit Controller

#### loadCompetences()

Deze methode wordt gebruikt om alle competenties uit de database te halen via de selectAllCompetences() methode. Met het resultaat wordt een Competence object gemaakt. Al deze objecten worden bijgehouden in de $competences array met als index de ID.

|  |
| --- |
| **public function** loadCompetences() { $return = $this->getContext()->selectAllCompetences();  **foreach** ($return **as** $value) {  $this->**competences**[*intval*($value[**'Id'**])] = **new** \Models\Competence(*intval*($value[**'Id'**]), $value[**'Naam'**]);  } } |

Figuur 18: loadCompetences uit Controller

#### loadQuestionsToMark()

Deze methode wordt gebruikt om de vragen die bij een specifieke functie horen uit de database te halen via de selectQuestionIdsFromFunction() methode. Het resultaat wordt bijgehouden in de array $questionsMarked. De objecten worden opgeslagen met als index de waarde zodat sneller kan gezocht worden in deze array.

|  |
| --- |
| **public function** loadQuestionsToMark($functionId) { $return = $this->getContext()->selectQuestionIdsFromFunction( $functionId);  **foreach** ($return **as** $value) {  $this->**questionsMarked**[*intval*($value[**'Id'**])] = i*ntval*($value[**'Id'**]);  } } |

Figuur 19: loadQuestionsToMark uit Controller

#### fillCompetencesToShow()

Wanneer de gebruiker een functie selecteert, zullen een aantal vragen standaard aangeduid worden. De bijhorende competenties moeten zichtbaar worden, de rest niet. Dat is wat deze methode gaat voorbereiden.

|  |
| --- |
| **private function** fillCompetencesToShow() {  **foreach** ($this->**questionsMarked as** $questionId) {  $question = $this->selectQuestionById($questionId);  **if** (**isset** ($question)) {  **if** (!*array\_key\_exists*($question->getCompetenceId(), $this-> **competencesToShow**)) {  $this->**competencesToShow**[$question->getCompetenceId()] = $question->getCompetenceId();  }  }  } } |

Figuur 14: Controller basis

|  |
| --- |
| **public function** getJobFunctions()  {  *// uses a stored procedure that gets all job functions* $return = $this->getContext()->selectAllFunctions();  $result = **array**();  **foreach** ($return **as** $value) {  *array\_push*($result, **new** \Models\JobFunction(*intval*( $value[**'Id'**]), $value[**'Naam'**]));  }  $this->**jobFunctions** = $result;  **return** $this->**jobFunctions**;  } |

Figuur 14: Controller basis

|  |
| --- |
| **namespace** Controller;  **class** Controller {  **protected $questions** = **array**();  **protected $competences** = **array**();  **protected $jobFunctions** = **array**();  **protected $questionsMarked** = **array**();  **protected $competencesToShow** = **array**();  **protected $context**;   **public function** \_\_construct()  {  $this->**context** = **new** \DAL\InterviewContext();  }   **public function** getQuestions()  {  **return** $this->**questions**;  }   **public function** getCompetences()  {  **return** $this->**competences**;  }   **public function** getQuestionsMarked()  {  **return** $this->**questionsMarked**;  }   **public function** getCompetencesToShow()  {  **return** $this->**competencesToShow**;  }   **private function** getContext()  {  **return** $this->**context**;  }   **public function** getJobFunctions()  {  *// uses a stored procedure that gets all job functions* $return = $this->getContext()->SelectAllFunctions();  $result = **array**();  **foreach** ($return **as** $value) {  *array\_push*($result, **new** \Models\JobFunction(*intval*($value[**'Id'**]), $value[**'Naam'**]));  }  $this->**jobFunctions** = $result;  **return** $this->**jobFunctions**;  }   **public function** loadDataByFunction($functionId)  {  *// this should only be called once per controller !!    // - loading ALL questions in $this->questions* $this->loadQuestions();  *// - loading ALL competences in $this->competences* $this->loadCompetences();  *// - loading all the questions that are tied to a function in $this->questionsMarked* $this->loadQuestionsToMark($functionId);  *// then make an array $this->competencesNotNeeded using $this->questions and $this->questionsMarked that will be used to hide competences* $this->fillCompetencesToShow();  }   **public function** loadQuestions()  {  *// uses a stored procedure that gets all questions into the questions array  // save each object at index object Id => makes it easier to search by id* $return = $this->getContext()->SelectAllQuestions();  **foreach** ($return **as** $value) {  $this->**questions**[*intval*($value[**'Id'**])] = **new** \Models\Question(*intval*($value[**'Id'**]), $value[**'Vraag'**], *intval*($value[**'CompetentieId'**]));  }  }   **public function** loadCompetences()  {  *// uses a stored procedure that gets all competences into the competences array  // save each object at index object Id => makes it easier to search by id* $return = $this->getContext()->SelectAllCompetences();  **foreach** ($return **as** $value) {  $this->**competences**[*intval*($value[**'Id'**])] = **new** \Models\Competence(*intval*($value[**'Id'**]), $value[**'Naam'**]);  }  }   **public function** loadQuestionsToMark($functionId)  {  *// uses a stored procedure that gets questions linked to a function  // save each object at index object Id => makes it easier to search by id* $return = $this->getContext()->selectQuestionIdsFromFunction($functionId);  **foreach** ($return **as** $value) {  $this->**questionsMarked**[*intval*($value[**'Id'**])] = *intval*($value[**'Id'**]);  }  }   **public function** fillCompetencesToShow()  {  *// makes an array $this->competencesToShow using $this->questions and $this->questionsMarked   // that will be used to hide competences that don't have any marked questions by default* **foreach** ($this->**questionsMarked as** $questionId) {  $question = $this->selectQuestionById($questionId);  **if** (**isset** ($question)) {  **if** (!*array\_key\_exists*($question->getCompetenceId(), $this->**competencesToShow**)) {  $this->**competencesToShow**[$question->getCompetenceId()] = $question->getCompetenceId();  }  }  }  }   **public function** selectCompetenceById($competenceId)  {  *// gets the competence object, based on the id* **if** (**isset**($competenceId)) {  **if** (*array\_key\_exists*($competenceId, $this->**competences**)) {  **return** $this->**competences**[$competenceId];  }  }  }   **public function** selectQuestionById($questionId)  {  *// gets the question object, based on the id* **if** (**isset**($questionId)) {  **if** (*array\_key\_exists*($questionId, $this->**questions**)) {  **return** $this->**questions**[$questionId];  }  }  }   **public function** selectQuestionsByCompetenceId($competenceId)  {  *// selects all questions from 1 competence, needed for the view* $v = **Array**();   **foreach** ($this->**questions as** $question) {  **if** ($question->getCompetenceId() == $competenceId) {  $v[] = $question;  }  }  **return** $v;  }   **public function** insertNewQuestion($competenceId, $question)  {  **return** $this->getContext()->InsertQuestion($question, $competenceId);  } } |

### Controller test?

## BLL

### CreateReport.php

Deze php pagina wordt gebruikt om de aangeduide vragen door te geven aan de helper klasse DataController die verder andere klassen zal aansturen om een pdf te kunnen maken.

Wanneer de gebruiker op de “Download PDF” knop klikt, wordt een stukje javascript uitgevoerd. De ID van de geselecteerde functie wordt opgezocht en er wordt een array gemaakt van ID’s van aangeduide vragen. Dit wordt in een JSON object gestoken en doorgestuurd met Ajax naar CreateReport.php. Als alles goed verloopt, wordt het resultaat later naar Download.php gestuurd.

|  |
| --- |
| $(**'#get-report-button'**).click(**function** () {  *// gets the Id of the selected function* **var** func = **document**.getElementById(**'jobfunction-select'**);  **var** jobTitle = func.**options**[func.**selectedIndex**].**value**;   *// creates an array of selected questions* **var** ids = [];  $(**"input.questionsCheck:checked"**).each(**function**() {  ids.push(**this**.**id**.substring(9));  });   *// combine into a JSON object* **var** result = {**"functionId"**: jobTitle};  result.**questionId** = ids;  **var** jsonObj = *JSON*.*stringify*(result); $.ajax({  **type**: **"POST"**,  **url**: **"../BLL/CreateReport.php"**,  **data**: jsonObj,  success: **function** (msg) {  **window**.**location** = **"../BLL/Download.php?filename="** + msg , **"\_blank"**;},  error: **function** () {  alert(**"The pdf file was not ready for download. Please try again."**);  }  }); }); |

Table 1

Deze CreateReport.php pagina decodeert het JSON object en geeft het door aan de DataController die de verdere verwerking afhandelt.

|  |
| --- |
| **include**(**'../vendor/autoload.php'**);  $dataController = **new** BLL\DataController();  *header*(**'Content-type: text/html; charset=UTF-8'**) ; $request\_body = *file\_get\_contents*(**'php://input'**); $jsondata = json\_decode($request\_body); $name = $dataController->getReport($jsondata); **echo** $name; |

Table 2

### DataController

Deze helper klasse bundelt een aantal bewerkingen die nodig zijn om een pdf te creëren.

#### getReport()

Wanneer getReport wordt aangeroepen, wordt eerst deleteOldTempFiles uitgevoerd (zie 6.3.1.2). Deze methode verwijdert oudere bestanden op de server om te verhinderen dat te veel plaats wordt in genomen door tijdelijke bestanden.

We maken een InterviewContext object aan dat we samen met een nieuw PDF object meegegven naar een CreatePdf object. Dit laatste zal instaan voor het opbouwen van de PDF, en dat is wat deze klasse juist beheert.

Eerst wordt de data, die meegegeven wordt bij het aanroepen van de method, verwerkt binnen het CreatePdf object. Daarna wordt aan de hand van de verwerkte data de database aangesproken die data uit de database haalt om de pdf te kunnen aanmaken. Vervolgens wordt de BuildPdf methode aangeroepen die de pdf zal aanmaken. Daarna wordt gekozen om de pdf op te slaan in een tijdelijke map zodat deze later kan gedownload worden. Als laatste stap wordt de naam gereturnd. Dit zal er voor zorgen dat de Download.php pagina in staat is de pdf te downloaden.

|  |
| --- |
| **namespace** BLL;  **class** DataController {  **private $tempFolder** = **'../temp/'**;  **public function** getReport($data)  {  *// clean up old files first* $this->deleteOldTempFiles();  $array = **$data**->{**'questionId'**};  $fId = **$data**->{**'functionId'**};  $context = **new** \DAL\InterviewContext();  $pdf = **new** \BLL\CreatePdf($context, **new** \BLL\PDF(),$this-> tempFolder);  $pdf->parseData($array, $fId);  $pdf->getData();  $pdf->buildPdf();  $pdf->outputToDownloadLater();  **return** $pdf->getName();  }  …  } |

#### deleteOldTempFiles

Deze methode zal de tijdelijke map beheren. Het is nodig om dit te doen omdat telkens een gebruiker een pdf wil laten genereren, deze tijdelijk op server moet gezet worden om dan rechtstreeks te downloaden. Het rechtstreeks downloaden via de Output methode van FPDF werkte niet. Deze methode zal alle bestanden in de temp map nakijken op de datum wanneer ze werden aangemaakt. Daarna wordt gekeken of deze bestanden ouder dan 1 minuut zijn, en worden die bestanden verwijderd.

|  |
| --- |
| **public function** deleteOldTempFiles() {  $x = 60; *//1 minute* $current\_time = *time*();   $files = **array**();   **if** (*sizeof*(*scandir*($this->tempFolder)) === 0) {  **return**;  }   **foreach** (*scandir*($this->tempFolder) **as** $file) {  **if** (**'.'** === $file) **continue**;  **if** (**'..'** === $file) **continue**;   $files[] = $file;  }   **foreach** ($files **as** $file) {  $file\_creation\_time = *filemtime*($this->tempFolder . $file);  $difference = $current\_time - $file\_creation\_time;   **if** ($difference >= $x) {  *unlink*($this->tempFolder . $file);  }  } } |

Table 3

### Download.php

Zoals al kort aangehaald bij de DataController, het is niet mogelijk om de pdf rechtstreeks naar de browser te laten downloaden. Daarom zetten we de pdf in een tijdelijke map temp op de server en gaan we via de Download.php pagina deze downloaden. Voor de gebruiker is er geen verschil zichtbaar, want de pagina vernieuwt niet.

Deze pagina ontvangt via query string de naam van het bestand en gaat zoeken of dat bestand in de temp map staat. Als dat zo is, worden de nodige headers ingesteld en wordt het bestand doorgestuurd.

|  |
| --- |
| **include**(**'../vendor/autoload.php'**); *parse\_str*($\_SERVER[**'QUERY\_STRING'**]); *// parses the query string and makes vars with the key => $q is created* **if** (**isset**($filename)) {  $fullPath = **'../temp/'**;  **if** (*file\_exists*($fullPath . $filename)) {  *header*(**'Content-Description: File Transfer'**);  *header*(**'Content-Type: application/octet-stream'**);  *header*(**'Content-Disposition: attachment; filename='** . $filename);  *header*(**'Content-Transfer-Encoding: binary'**);  *header*(**'Expires: 0'**);  *header*(**'Cache-Control: must-revalidate, post-check=0, pre-check=0'**);  *header*(**'Pragma: public'**);  *header*(**'Content-Length: '** . *filesize*($fullPath . $filename));  *ob\_clean*();  *flush*();  *readfile*($fullPath . $filename);  **exit**();  } } |

Table 4

… missing classes

## DAL

De DAL map bevat een submap Helpers die hier niet verder besproken wordt. De klassen die hier in staan zijn immers een kopie van de klassen uit AnOrmApart die we gebruiken in het project.

De andere klassen met bijhorende test pagina’s worden hier wel besproken.

### Provider

Deze klasse bevat de details van de server en database. Verder moet er ook een Log object worden meegegeven wanneer deze wordt gemaakt. Het is belangrijk op de charset mee in de connectionstring te definiëren. Accentletters worden niet altijd goed geïmporteerd als dit niet gebeurt.

|  |
| --- |
| **namespace** DAL;  **class** Provider **extends** \DAL\Helpers\Connection {  **public $connectionString**;   **public function** \_\_construct($log)  {  $this->**log** = $log;  $this->**databaseName** = **'InterviewDB'**;  $this->**password** = **'9uPZV)U;z\_)+'**;  $this->**userName** = **'CvoProject'**;  $this->**hostName** = **'trouw.benoot-cupers.com:3306'**;$this->**connectionString** = **"mysql:host=**$this->**hostName;dbname=**$this->**databaseName;charset=utf8"**;} } |

Table 5

### ConnectionTest.php

Deze pagina dient om de verbinding met de database te testen. Er wordt een LogApp object aangemaakt waarmee we een nieuw Provider object aanmaken. De verbinding wordt geopend en terug gesloten. De log die hierdoor gemaakt wordt, wordt dan met een loop weergegeven met html code.

Wanneer de verbinding niet lukt, krijg je foutboodschappen te zien. Als de verbinding wel lukt, krijg je de 2 entries van de log te zien met alle details.

|  |
| --- |
| **include\_once**(**'../vendor/autoload.php'**);  $log = **new** DAL\Helpers\LogApp(**'en\_US'**); $provider = **new** DAL\Provider($log); $provider->open(); $provider->close(); **?>** <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="utf-8"** />  <**title**>Connection class test</**title**> </**head**> <**body**> **<?php foreach** ($log->getBook() **as** $key => $feedback) { **?>** <**h1**>**<?php echo** $key;**?>**</**h1**>  <**p**><**b**>Name</**b**> **<?php echo** $feedback->getName();**?>**</**p**>  <**p**><**b**>Feedback</**b**> **<?php echo** $feedback->getText();**?>**</**p**>  <**p**><**b**>Error code</**b**> **<?php echo** $feedback->getErrorCode();**?>**</**p**>  <**p**><**b**>Error message</**b**> **<?php echo** $feedback-> getErrorMessage();**?>**</**p**>  <**p**><**b**>Error Code Driver</**b**> **<?php echo** $feedback-> getErrorCodeDriver();**?>**</**p**>  <**p**><**b**>Is error</**b**> **<?php echo** $feedback->getIsError();**?>**</**p**>  <**p**><**b**>Start</**b**> **<?php echo** $feedback->getStartTime();**?>**</**p**>  <**p**><**b**>End</**b**> **<?php echo** $feedback->getEndTime();**?>**</**p**> **<?php** } **?>** </**body**> </**html**> |

Table 6

### InterviewContext

De InterviewContext klasse bevat methodes die steeds een stored procedure in de database uitvoeren. Bij het aanmaken van een object van deze klasse wordt ook een nieuw ContextController object aangemaakt die telkens hetzelfde PDO object gebruikt zodat wanneer meerdere methodes worden aangeroepen met hetzelfde InterviewContext object, dit sneller wordt uitgevoerd. De tijd nodig om de CompetencesView aan te maken, wordt hierdoor ongeveer gehalveerd.

In onderstaand stukje code staat één methode die als voorbeeld dient voor de vele methode die in deze klasse uitgewerkt zijn. Deze methodes worden zeker niet allemaal gebruikt, maar kunnen op termijn, wanneer dit wordt uitgebreid, wel van pas komen.

|  |
| --- |
| **namespace** DAL;  **class** InterviewContext {  **protected $controller**;   **public function** \_\_construct()  {  $this->**controller** = **new** \DAL\ContextController();  }   **public function** getController()  {  **return** $this->**controller**;  }  **public function** selectQuestionsOnID($Ids)  {  $preparedStatement = $this->**controller**->getPDO()->prepare(**"call SelectReportData(:pListId);"**);  $preparedStatement->bindParam(**':pListId'**, $Ids, \PDO::***PARAM\_STR***);  $result = $preparedStatement->execute();  $data = $preparedStatement->fetchAll(\PDO::***FETCH\_ASSOC***);  **return** $data;  } |

Table 7

### ContextController

Dit is een helper klasse die in de InterviewContext wordt gebruikt. Om efficiënter om te gaan met resources, zal deze klasse een PDO object aanmaken en bijhouden zodat met deze controller dat object steeds opnieuw gebruikt wordt.

|  |
| --- |
| **namespace** DAL;  **class** ContextController { **private $log**;  **private $connection**;  **private $pdo**;   **public function** \_\_construct()  {  $this->**log** = **new** Helpers\LogApp(**'en\_US'**);  $this->**connection** = **new** Provider($this->**log**);  $this->**pdo** = **new** \PDO($this->**connection**->**connectionString**,  $this->**connection**->getUserName(), $this->**connection**->getPassword());  }   **public function** getLog()  {  **return** $this->**log**;  }   **public function** getPDO()  {  **return** $this->**pdo**;  } } |

Table 8

# Uitbreiding

## Uitbreiding blijft mogelijk

## Use case diagram