

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](#_Toc421619701)

[2 Voorwoord 6](#_Toc421619702)

[3 Projectvereisten 7](#_Toc421619703)

[3.1 Concept 7](#_Toc421619704)

[3.2 Concreet 7](#_Toc421619705)

[3.3 Gebruikte concepten en frameworks 7](#_Toc421619706)

[4 Analyse 8](#_Toc421619707)

[4.1 ERD 8](#_Toc421619708)

[4.2 Use cases 8](#_Toc421619709)

[4.2.1 Maken van een pdf na het selecteren van een functie 8](#_Toc421619710)

[4.2.2 Een nieuwe vraag toevoegen 9](#_Toc421619711)

[5 Indeling van het project 10](#_Toc421619712)

[5.1 Mappenstructuur 10](#_Toc421619713)

[5.2 Map app 10](#_Toc421619714)

[5.3 Map BLL 10](#_Toc421619715)

[5.4 Map DAL 10](#_Toc421619716)

[5.5 Map temp 10](#_Toc421619717)

[5.6 Map vendor 11](#_Toc421619718)

[5.7 Map web 11](#_Toc421619719)

[5.8 Composer & namespaces 11](#_Toc421619720)

[5.8.1 Aanpassing aan autoload\_real.php 11](#_Toc421619721)

[5.8.2 Toevoeging van autoload\_II.php 11](#_Toc421619722)

[6 Opbouw 12](#_Toc421619723)

[6.1 Index.php en Javascript 12](#_Toc421619724)

[6.1.1 Head 12](#_Toc421619725)

[6.1.2 Pagina-indeling 12](#_Toc421619726)

[6.1.3 Div interviewForm 12](#_Toc421619727)

[6.1.4 Modals 12](#_Toc421619728)

[6.2 MVC 13](#_Toc421619729)

[6.2.1 Models 13](#_Toc421619730)

[6.2.1.1 Question 13](#_Toc421619731)

[6.2.1.2 Competence 14](#_Toc421619732)

[6.2.1.3 JobFunction 14](#_Toc421619733)

[6.2.2 Views 15](#_Toc421619734)

[6.2.2.1 JobFunctionsView 15](#_Toc421619735)

[6.2.2.2 jobFunctionSelectChanged() 15](#_Toc421619736)

[6.2.2.3 CompetencesView 16](#_Toc421619737)

[6.2.3 Controller 18](#_Toc421619738)

[6.2.3.1 getJobFunctions() 18](#_Toc421619739)

[6.2.3.2 loadDataByFunction($functionId) 19](#_Toc421619740)

[6.2.3.3 loadQuestions() 19](#_Toc421619741)

[6.2.3.4 loadCompetences() 19](#_Toc421619742)

[6.2.3.5 loadQuestionsToMark() 20](#_Toc421619743)

[6.2.3.6 fillCompetencesToShow() 20](#_Toc421619744)

[6.2.3.7 selectCompetenceById() 20](#_Toc421619745)

[6.2.3.8 selectQuestionById() 21](#_Toc421619746)

[6.2.3.9 selectQuestionsByCompetenceId() 21](#_Toc421619747)

[6.2.3.10 insertNewQuestion() 21](#_Toc421619748)

[6.2.4 Controller test 22](#_Toc421619749)

[6.3 BLL 22](#_Toc421619750)

[6.3.1 CreateReport.php 22](#_Toc421619751)

[6.3.2 DataController 23](#_Toc421619752)

[6.3.2.1 getReport() 23](#_Toc421619753)

[6.3.2.2 deleteOldTempFiles() 24](#_Toc421619754)

[6.3.3 Download.php 25](#_Toc421619755)

[6.3.4 CreatePdf 26](#_Toc421619756)

[6.3.4.1 parseData() 26](#_Toc421619757)

[6.3.4.2 getData() 27](#_Toc421619758)

[6.3.4.3 prepareData() 27](#_Toc421619759)

[6.3.4.4 buildPdf() 28](#_Toc421619760)

[6.3.4.5 addTripleLine() 28](#_Toc421619761)

[6.3.4.6 outputToDownloadLater() 28](#_Toc421619762)

[6.3.4.7 outputDirect() 29](#_Toc421619763)

[6.3.5 CreatePdfTest.php 29](#_Toc421619764)

[6.4 DAL 30](#_Toc421619765)

[6.4.1 Provider 30](#_Toc421619766)

[6.4.2 ConnectionTest.php 30](#_Toc421619767)

[6.4.3 InterviewContext 31](#_Toc421619768)

[6.4.4 ContextController 32](#_Toc421619769)

[6.4.5 AddQuestion.php 33](#_Toc421619770)

[6.4.5.1 add-question-modal 33](#_Toc421619771)

[6.4.5.2 sendQuestion() 33](#_Toc421619772)

[6.4.5.3 AddQuestion.php pagina 34](#_Toc421619773)

[7 Uitbreiding 35](#_Toc421619774)

[7.1 Uitbreiding blijft mogelijk 35](#_Toc421619775)

[7.2 Use case diagram 35](#_Toc421619776)

# Voorwoord

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.

Dit document bevat niet alle code die we hebben geschreven, maar wel de meest relevante code om duidelijk te maken wat de bedoeling is. Er wordt verondersteld dat lezers van dit document code kunnen lezen en begrijpen. Dit document laat alleen zien hoe al die code opgebouwd is tot de webapplicatie die het is geworden.

De volledige code zal in een .zip bestand bezorgd worden. Om de website te kunnen tonen, moet er wel een webserver geconfigureerd worden. Wij hebben voor dit project met PhpStorm gewerkt waar heel eenvoudig een lokale webserver kan geconfigureerd worden.

# 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 bestand. 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 David, 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 (en dus met ondersteuning voor alle soorten apparaten)
* 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. In deze use cases zit alles wat nodig was om het project te kunnen opbouwen.

### 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.

#### add-question-modal

De gebruikt klikt op de knop “Voeg vraag toe”, dit roept een javascript op dat een bovenliggend venster (modal) zichtbaar maakt. Hier op staat een tekstvak en een keuzelijst met competenties. De keuzelijst wordt met 1 functie gevuld, maar hiervoor wordt niet opnieuw naar de database gegaan. De gegevens (competenties met ID’s) staan gewoon op de pagina en kunnen dus met jQuery lus opgehaald worden. Per competentie wordt een option element aangemaakt in de keuzelijst.

|  |
| --- |
| $(**'#add-question-button'**).click(**function** () {  $(**'#add-question-modal'**).modal();   **var** select = $(**'#competence-select'**);  $(**"div[id^='questionssection']"**).each(**function** () { select.append($(**'<option></option>'**) .val(**this**.**id**.substring(16)) .html($(**this**).find(**".panel-title"**).text()));}); }); |

Figuur 8: #add-question-button klik event uit scripts.js

#### report-modal

De gebruikt klikt op de knop “Rapport” en ook nu wordt het bovenliggende venster zichtbaar gemaakt. De gebruiker krijgt eerst nog een snel overzicht te zien van alle vragen. Deze worden met 1 jQuery lus onder elkaar gezet, gemaakt met de elementen van de bestaande pagina.

|  |
| --- |
| $(**'#modal-report-button'**).click(**function** () { $(**'#report-modal'**).modal();  **var** out = **""**;  $(**"input:checked"**).parent().next(**"div"**).find(**"label"**).each(**function** () {  out += $(**this**).text() + **"</br>"**;  });  $(**"#report-body"**).html(out); }); |

Figuur 9: #modal-report -button klik event uit scripts.js

## 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 10: 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 11: 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 12: 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.

|  |
| --- |
| <**div class="row"**>  <**div id="testdiv"**></**div**> </**div**> <**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** $jobFunction) { **if**(**isset**($functionId) && $functionId == $jobFunction-> getId()){ **?>** <**option  selected value="<?php echo** $jobFunction-> getId()**?>"** >**<?php echo** $jobFunction->getName() **?>** </**option**>  **<?php** } **else** { **?>** <**option  value="<?php echo** $jobFunction->getId()**?>"** >**<?php echo** $jobFunction->getName() **?>** </**option**>  **<?php** }  } **?>** </**select**>  </**div**> </**form**> <**div id="fetch-competences-div"**></**div**> |

Figuur 13: JobFunctionsView.php

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.

Als er een waarde voor $functionId is, dan wil dit zeggen dat deze view opgeroepen is na het toevoegen van een vraag. In dat geval bevat $functionId de ID van de functie die de gebruiker heeft geselecteerd en wordt die nu ook meteen geselecteerd.

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

Figuur 14: jobFunctionSelectChanged() uit script.js

#### CompetencesView

Deze view bouwt met twee loops de elementen met 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 15: CompetencesView.php

### 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**;  }  **public function** getLog() {   **return** $this->**context**->getController()->getLog(); } …  } |

Figuur 16: 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 17: 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 18: 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 19: 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 20: 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 21: 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. Er wordt een foreach lus gemaakt met alle vragen die aangeduid gaan worden. Van elke vraag wordt de competentie ID in een array gestoken als deze nog niet aanwezig was. Zo is er na het uitvoeren van deze methode een array genaamd $competencesToShow die enkel ID’s van competenties die moeten getoond worden. Alle andere competenties zullen worden verborgen.

|  |
| --- |
| **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 22: fillCompetencesToShow uit Controller

#### selectCompetenceById()

Dit is een methode die een bepaalde competentie ophaalt wanneer de ID geweten is. Omdat de array $competences alle competenties met als index de ID heeft, is het ophalen vrij eenvoudig. Er wordt gekeken of er iets op de plaats ID staat en zo ja, wordt dit gereturnd.

|  |
| --- |
| **public function** selectCompetenceById($competenceId) { **if** (**isset**($competenceId)) {  **if** (*array\_key\_exists*($competenceId, $this->**competences**)) {  **return** $this->**competences**[$competenceId];  }  } } |

Figuur 23: selectCompetenceById uit Controller

#### selectQuestionById()

Heel gelijkaardig aan de voorgaande methode, maar voor vragen. Dit haalt dus een Question object op aan de hand van de ID op dezelfde manier.

|  |
| --- |
| **public function** selectQuestionById($questionId) { **if** (**isset**($questionId)) {  **if** (*array\_key\_exists*($questionId, $this->**questions**)) {  **return** $this->**questions**[$questionId];  }  } } |

Figuur 24: selectQuestionById uit Controller

#### selectQuestionsByCompetenceId()

Naar analogie met LINQ is dit een methode die alle vragen returnt die bij een bepaalde competentie horen. Er wordt een lus gemaakt en elke vraag wordt gecontroleerd. Heeft deze de gewenste competentie, dan wordt die in een tijdelijke array gestopt en deze wordt op het einde gereturnd.

|  |
| --- |
| **public function** selectQuestionsByCompetenceId($competenceId) { $v = **Array**();  **foreach** ($this->**questions as** $question) {  **if** ($question->getCompetenceId() == $competenceId) {  $v[] = $question;  }  }  **return** $v; } |

Figuur 25: selectQuestionsByCompetenceID uit Controller

#### insertNewQuestion()

Deze methode roept een gelijkaardige methode op in de InterviewContext om daar het toevoegen van een vraag verder te verwerken.

|  |
| --- |
| **public function** insertNewQuestion($competenceId, $question) {  **return** $this->getContext()->InsertQuestion($question, $competenceId); } |

Figuur 26: insertNewQuestion uit Controller

### Controller test

Om zeker te zijn dat de controller werkt en om tijdens het ontwikkelen ook snel te kunnen verifiëren dat het nog werkt, is er een ControllerTest pagina. Deze pagina zal uit de controller een simpele methode aanroepen. Zowel de log als het resultaat van deze methode zal getoond worden, zodat het probleem sneller kan gevonden worden.

|  |
| --- |
| **include "../../vendor/autoload.php"**; $controller = **new** \Controller\Controller(); $data = $controller->getJobFunctions(); $log = $controller->getLog(); **?>** <**html**> <**head**>  <**title**>ControllerTest</**title**> </**head**> <**body**> <**h3**>Data Loaded by the controller:</**h3**> <**pre**>  **<?php** *var\_dump*($data) **?>** </**pre**> <**h3**>Content of the logbook</**h3**> **<?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**> |

Figuur 27: ControllerTest.php

## 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."**);  }  }); }); |

Figuur 28: #get-report-button click event uit scripts.js

De 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; |

Figuur 29: CreateReport.php

### 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.

|  |
| --- |
| **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();  }  …  } |

Figuur 30: DataController.php

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.

#### 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);  }  } } |

Figuur 31: deleteOldTempFiles uit DataController

### 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.

|  |
| --- |
| **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**();  } } |

Figuur 32: Download.php

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.

### CreatePdf

Dit is een klasse die zal instaan voor aanmaken van de pdf. Bij het aanmaken van een object zijn 3 parameters vereist:

* een InterviewContext object dat gebruikt wordt om data uit de database te halen.
* een PDF object, dat overerft van FPDF, om de pdf te maken.
* de locatie van de tijdelijke map waar de pdf moet worden opgeslagen. Dit wordt als parameter meegegeven omdat dit een relatief pad is.

In de constructor van CreatePdf wordt ook nog een string met 160 puntjes gemaakt die zal dienen om invullijntjes te maken in de pdf tussen elke vraag.

|  |
| --- |
| **namespace** BLL;  **class** CreatePdf {  **protected $fId**;  **protected $questionString**;  **protected $pdfData**;  **protected $preparedData**;  **protected $Pdf**;  **protected $name**;  **protected $fpdf**;  **protected $context**;  **protected $tempFolder**;  **protected $line**;   **public function** \_\_construct($\_context, $\_fpdf, $\_tempFolder)  {  $this->**context** = $\_context;  $this->**fpdf** = $\_fpdf;  $this->**tempFolder** = $\_tempFolder;  $this->**line** = *str\_repeat*(**'.'**, 160);  }  …  } |

Figuur 33: CreatePdf.php deel

De volgorde waarin de methods verder worden aangeroepen is reeds besproken in de DataController.

#### parseData()

Deze methode zal de array van ID’s van vragen omvormen tot een kommagescheiden reeks zodat de stored procedure meerdere rijen kan opvragen met 1 parameter, deze string.

|  |
| --- |
| **public function** parseData($array) {  **try** {  **foreach** ($array **as** $val) {  $valint = *intval*($val);  **if** ($valint != 0) {  **if** (*strlen*($this->**questionString**) != 0) {  $this->**questionString** .= **",**$valint**"**;  } **else** {  $this->**questionString** .= **"**$valint**"**;  }  }  }  } **catch** (Exception $e) {  **echo** $e;  } } |

Figuur 34: parseData uit CreatePdf

#### getData()

Hier wordt de stored procedure aangesproken met de reeks van ID’s.

|  |
| --- |
| **public function** getData() {  $idList = *preg\_replace*(**"/[^0-9,]/"**, **""**, $this->**questionString**);  $this->**pdfData** = $this->**context**->selectReportData($idList); } |

Figuur 35: getData uit CreatePdf

#### prepareData()

Deze methode ordent de data in een array van arrays met als key de naam van de competentie. Zo kan in de volgende methode met een lus in een lus de pdf gemakkelijk gemaakt worden. Deze methode wordt vanuit de buildPdf methode aangroepen.

Om fouten te vermijden wordt eerst gecontroleerd of er minstens één competentie is om weer te geven. Daarna wordt gecontroleerd of bij de huidige competentie al een array zit, zo neen, wordt een nieuwe array aangemaakt. De opgevraagde array krijgt dan de vraag mee.

|  |
| --- |
| **private function** prepareData() {  **if** ($this->**pdfData**[0][**'CompNaam'**] == **null**) {**return**;  }   $comp = **""**;  $data = **array**();  **foreach** ($this->**pdfData as** $datarow) {  **if** ($datarow[**'CompNaam'**] != $comp) {  *// if new competence, create a new array* $data[$datarow[**'CompNaam'**]] = **array**();  $comp = $datarow[**'CompNaam'**];  }  *array\_push*($data[$datarow[**'CompNaam'**]], $datarow[**'Vraag'**]);  }   $this->**preparedData** = $data; } |

Figuur 36: prepareData uit CreatePdf

#### buildPdf()

Hier wordt de eigenlijke pdf opgebouwd. De klasse variabele $preparedData wordt gebruikt. Er wordt een lus gemaakt over deze array en aangezien elk van de elementen ook nog een array nog eens een lus om zo alle elementen te doorlopen. De key voor elk element in $preparedData is de competentie en wordt telkens in het blauw geplaatst als titel. Daarna wordt elke vraag geplaatst, gevolgd door 3 lijnen met puntjes. Als laatste stap wordt de naam van de pdf bepaald, met de huidige tijd in verwerkt om er voor te zorgen dat er een uniek bestand is op de server. Het gebruik van een unieke ID zou beter zijn, maar dit werd voor dit project niet verder uitgewerkt.

|  |
| --- |
| **public function** buildPdf() { $this->**fpdf**->AliasNbPages();  $this->**fpdf**->AddPage();  $this->**fpdf**->SetFont(**"Arial"**, **""**, 12);   $this->prepareData();   **foreach** ($this->**preparedData as** $key => $value) {  $this->**fpdf**->SetTextColor(68, 110, 155);  $this->**fpdf**->Cell(0, 10, $key, 0, 1);  $this->**fpdf**->SetTextColor(0, 0, 0);  $this->**fpdf**->Ln(0);  **foreach** ($value **as** $vraag) {  $this->**fpdf**->SetTextColor(0, 0, 0);  $this->**fpdf**->MultiCell(0, 5, $vraag, 0, 1);  $this->**fpdf**->Ln(0);  $this->addTripleLine();  $this->**fpdf**->Ln(5);  }  $this->**fpdf**->Ln(0);  }  $t = *time*();  $this->**name** = **'Interview\_'** . *date*(**'Y-m-d\_H-i-s'**, $t) . **'.pdf'**; } |

Figuur 37: buildPdf uit CreatePdf

#### addTripleLine()

Deze methode maakt drie lijnen van puntjes.

|  |
| --- |
| **private function** addTripleLine() { **for** ($i = 0; $i < 3; $i++) {  $this->**fpdf**->Cell(0, 10, $this->**line**, 0, 1);  $this->**fpdf**->Ln(-2);  } } |

Figuur 38: addTripleLine uit CreatePdf

#### outputToDownloadLater()

Deze methode schrijft het bestand weg naar de tijdelijke map.

|  |
| --- |
| **public function** outputToDownloadLater() {  $this->**fpdf**->Output($this->**tempFolder** . $this->**name**, **'F'**); } |

Figuur 39: outputToDownloadLater uit CreatePdf

#### outputDirect()

Deze methode schrijft het pdf bestand rechtstreeks naar de browser. Dit werkt niet in deze applicatie, maar wordt wel gebruikt in de CreadePdfTest.php pagina.

|  |
| --- |
| **public function** outputDirect() {  $this->**fpdf**->Output($this->**name**, **'I'**); } |

Figuur 40: outputDirect uit CreatePdf

### CreatePdfTest.php

Deze pagina wordt gebruikt als test voor het CreatePdf klasse. Er wordt een array met test data gemaakt die manueel wordt doorgegeven aan de parseData methode. De data wordt verwerkt en een pdf wordt aangemaakt. Meestal zal dit rechtstreeks gebeuren, maar er kan ook getest worden of het bestand op de server kan aangemaakt worden. Op het einde worden ook tijdelijke bestanden opgekuist.

|  |
| --- |
| **include\_once**(**'../vendor/autoload.php'**);  $pdf = **new** \BLL\CreatePdf(**new** \DAL\InterviewContext(), **new** \BLL\PDF(), **"../temp/"**); $array = [1,2,31,4,11,50,15,18,40,7,9]; *// testdata*  $pdf->parseData($array); $pdf->getData(); $pdf->buildPdf();  *// use this to see the result in the browser directly* $pdf->OutputDirect(); *// or use this to create the file on the server (does not download automatically) //$pdf->OutputToDownloadLater();  // do cleanup manually in the test* $datacontroller = **new** \BLL\DataController(); $datacontroller->deleteOldTempFiles(); |

Figuur 41: CreatePdfTest.php

## 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"**;} } |

Figuur 42: DAL.php

### 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**> |

Figuur 43: ConnectionTest.php

### 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;  } |

Figuur 44: InterviewContext.php

### 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**;  } } |

Figuur 45: InterviewContext.php

### AddQuestion.php

Deze pagina wordt door Ajax aangeroepen wanneer de gebruiker een vraag wilt toevoegen.

#### sendQuestion()

Het form element form-add-question wordt gecodeerd zodat het volledig naar de AddQuestion pagina kan meegeven worden. Dit zal alleen gebeuren als er een competentie geselecteerd is en als het tekstvak iets van inhoud bevat. Het resultaat van de Ajax functie zal in een div element worden weergegeven. Hierbij wordt ook de volledige JobFunctions opnieuw aangemaakt en dus wordt jobFunctionsSelectChanged() gebruikt om ook de CompetencesView opnieuw aan te maken. Dit zorgt er voor dat de nieuwe vraag meteen voor de gebruiker beschikbaar is. Aangezien de views gemakkelijk apart kunnen vernieuwd worden, is het niet nodig de hele pagina te vernieuwen. Als de gebruiker al vragen geselecteerd heeft, die niet de standaard selectie is bij een functie, gaat deze wel verloren.

|  |
| --- |
| **function** *sendQuestion*() {  **if**(!$.trim($(**'form#form-add-question option:selected'**).val())){   $(**'#divCompetence'**).addClass(**'has-error'**).focus();   **return**; } **if** (!$.trim($(**'form#form-add-question textarea'**).val())){   $(**'#divQuestion'**).addClass(**'has-error'**).focus();   **return**; }  **var** formdata = $(**'form#form-add-question'**).serialize();  **var** funcId = $(**'#jobfunction-select option:selected'**).val();   $.ajax({  **type**: **"POST"**,  **url**: **"../BLL/AddQuestion.php"**,  **data**: {**functionId**: funcId ,**formdata**: formdata},  success: **function** (msg) {  $(**"#interview-form"**).empty();  *HideButtons*();  $(**"#interview-form"**).html(msg);  *jobFunctionSelectChanged*();  $(**"#add-question-modal"**).modal(**'hide'**); *//hide modal* },  error: **function** () {  alert(**"Something went wrong, please try again."**);  }  }); } |

Figuur 46: sendQuestion() uit scripts.js

#### AddQuestion.php pagina

Deze pagina krijgt het gecodeerde formulier door. Dit wordt gebruikt om een stored procedure op te roepen die deze gegevens gebuikt om een nieuwe vraag toe te voegen. Het resultaat van deze pagina, hangt af van het resultaat van die stored procedure. Als de vraag werd toegevoegd, krijgt de gebruiker een groen kadertje te zien en worden de Views vernieuwd. Als de gebruiker een fout heeft gemaakt, wordt een foutboodschap weergegeven en worden de views niet vernieuwd.

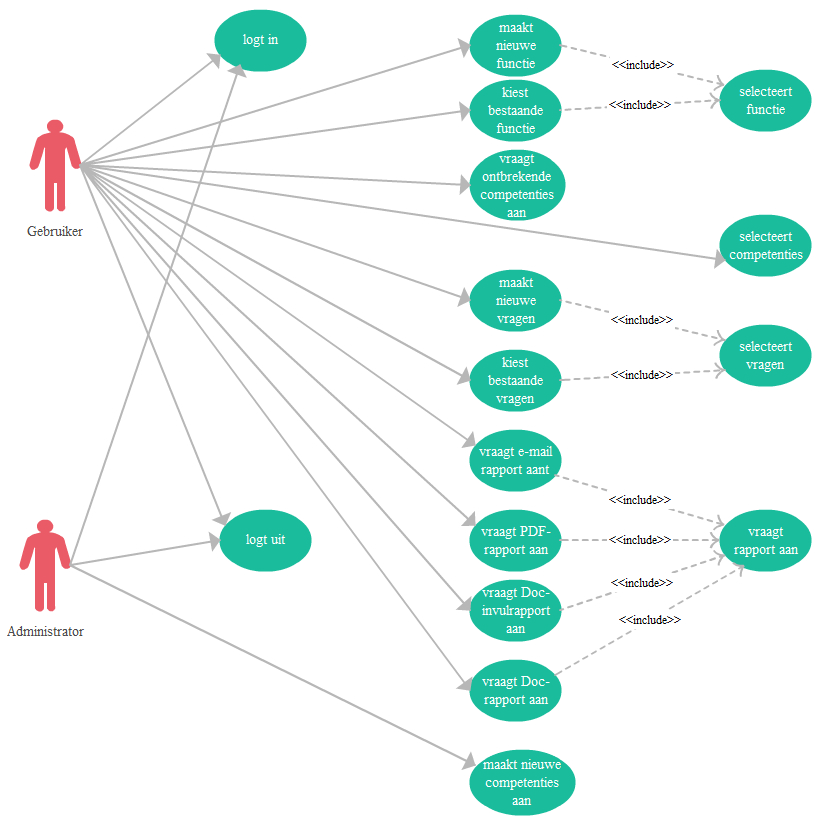
|  |
| --- |
| **include**(**'../vendor/autoload.php'**);  *header*(**'Content-type: text/html; charset=UTF-8'**) ;  **if** (**isset**($\_POST[**'functionId'**])) {  $controller = **new** \Controller\Controller();  $functionId = *strip\_tags*($\_POST[**'functionId'**]);  *parse\_str*(*strip\_tags*($\_POST[**'formdata'**]), $formData);  $competence = $formData[**'competence-select'**];  $question = $formData[**'question-text'**];  $result = $controller->insertNewQuestion($competence, $question);  $competenceFull = $controller->selectCompetenceById( *intval*($competence))->getName();   **if** ($result == **true**) {  $message = **<<<DOC  <br><div class="alert alert-success alert-dismissible" role="alert">  <button type="button" class="close" data-dismiss="alert"><span aria-hidden="true">&times;</span></button>  <strong>Success!</strong> De nieuwe vraag is toegevoegd aan de databank.<br>  <stong>Competentie: </strong>**$competenceFull**<br>  <stong>Vraag: </strong>**$question**<br>  </div> DOC**;  **echo** $message;  **require '../app/views/JobFunctionsView.php'**;  } **else** {  $message = **<<<DOC  <br><div class="alert alert-danger alert-dismissible" role="alert">  <button type="button" class="close" data-dismiss="alert"><span aria-hidden="true">&times;</span></button>  <strong>Failed!</strong> Data not entered in database: <br>**$result**<br>  </div> DOC**;  **echo** $message;  } } **else** {  $message = **<<<DOC  <br><div class="alert alert-danger alert-dismissible" role="alert">  <button type="button" class="close" data-dismiss="alert"><span aria-hidden="true">&times;</span></button>  <strong>Failed!</strong> Nothing in post<br>  </div> DOC**;  **echo** $message; } |

Figuur 47: AddQuestion.php

# Uitbreiding

Er zijn veel mogelijkheden die hier zijn uitgewerkt vanwege een gebrek aan tijd. Dat was de enige reden aangezien verwacht wordt dat veel ideeën niet echt moeilijke code zal vereisen maar eerder een uitbreiding zal zijn op bestaande code of een bestaand concept.

## Uitgebreid Use case diagram



Figuur 48: Uitgebreid Use case diagram

### Vragen en competenties

Op dit moment kan een gebruiker alleen nieuwe vragen aanmaken. Het zou ook mogelijk moeten zijn om vragen te verwijderen, aan te passen, nieuwe functies aan te maken en nog veel meer. De stored procedures hiervoor zijn al aangemaakt in de database. Waarschijnlijk moeten hiervoor een aantal views aangepast worden en modals bijgemaakt worden maar dit is allemaal vrij eenvoudig.

### Groepen gebruikers

Op dit moment is er geen administrator. De gebruiker is in dit geval een publieke gebruiker en heeft evenveel rechten als elke andere gebruiker. Sommige zaken moeten door een administrator beheerd worden, zoals competenties en vragen verwijderen. Hiervoor moet dan ook meteen een Login systeem toegevoegd worden. Dit werd reeds in de les besproken en dat gebruikte systeem is voldoende voor dit project.

### Exporteren

In dit project is er enkel de mogelijkheid een pdf te maken en deze te downloaden. Bij het vastleggen van het concept van dit project zijn ook andere mogelijkheden aangehaald om te exporteren. Zo kan op dezelfde manier ook een Word document gemaakt worden, waar de gebruiker rechtstreeks antwoorden kan in ingeven. Of kan het document, pdf of doc, doorgemaild worden zodat het ook na een download beschikbaar blijft vanuit de mail van de gebruiker.

## Categorieën competenties

Bij het opstellen van de ERD is ook de mogelijkheid aan bod gekomen om competenties een categorie te geven. Zo zijn er verschillende competenties die kunnen gegroepeerd worden onder meerdere soorten vaardigheden. Hierdoor komt er dus een entiteit bij en zal er waarschijnlijk ook een extra View nodig zijn. Er werd beslist om het project te implementeren zonder deze categorieën.

## Antwoorden invullen

Een laatste suggestie, die wel heel ingrijpend is, is de mogelijkheid om de antwoorden van de kandidaat meteen in te vullen in tekstvakken en deze mee in de database op te slaan. Hiervoor moeten meerdere entiteiten worden bijgemaakt en zou onder andere local storage kunnen gebruikt worden om de inhoud van deze tekstvakken goed bij te houden in het geval van een netwerkprobleem bij het doorsturen.