# Oef 1 ( 8 punten)

De begintoestand van deze oefening en de sql-code voor de databank wordt je bezorgd via filzilla:

***ModelException***

Gegeven de klasse ModelException in het bestand src/model/ModelException.php. Let op de namespace:

**<?php namespace** model;  
  
**class** ModelException **extends** \Exception{  
}

Je mag dit bestand ongewijzigd laten:

***Message.php***

Maak de klasse Message in het bestand src/model/entities/Message.php. Voorzie de correcte namespace voor dit bestand.

Message heeft een private velden id en contents.

Voorzie een private constructor. Deze constructor heeft argumenten id en contents. Je mag ervan uitgaan dat id altijd een integer is en contents een string (dit hoeft niet de uitwerking van de methode gecontroleerd te worden, type-hinting is ook niet nodig). Indien id kleiner dan of gelijk aan 0 is of indien contents een lege string is , wordt een ModelException opgeworpen.

Voorzie de methode make. Deze methode heeft argumenten id en contents. De methode make maakt een Message-object aan en geeft dit terug. Het gebruik van de methode wordt hieronder geïllustreerd :

$message = Message::make(1, “Sic transit Gloria mundi”);

Voorzie getters voor id en contents.

***Sender.php***

Maak de klasse Sender in het bestand src/model/entities/Sender.php. Voorzie de correcte namespace voor dit bestand.

Sender heeft een private velden id, name en messages.

Voorzie een private constructor. Deze constructor heeft argumenten id en name. Je mag ervan uitgaan dat id altijd een integer is en name een. Indien id kleiner dan of gelijk aan 0 is of indien contents een lege string is , wordt een ModelException opgeworpen. In de constructor wordt ook het veld messages aangemaakt als zijnde een lege array.

Voorzie de methode make. Deze methode heeft argumenten id en name. De methode make maakt een Sender-object aan en geeft dit terug. Het gebruik van de methode wordt hieronder geïllustreerd :

$sender = Sender::make(1, “tim”);

Via de methode addMessage wordt een Message-object toegevoegd aan de array messages. Maak in deze methode gebruik van type-hinting om te controlerenof dit wel degelijk zo is.

$sender ->addMessage($message);

Maak getters voor id en name.

Voorzie de methode countNumberOfMessages. Deze methode geeft een int terug met daarin de lengte van de array messages.

$messageCount = $sender ->countNumberOfMessages();

Voorzie de methode getMessageByIndex. Deze methode geeft de message op in de ingegeven index terug (er hoeven geen controles te gebeuren op de correctheid van de index).

$message = $sender->getMessageByIndex(0);

***SenderRepository.php***

Maak de klasse SenderRepository in het bestand src/model/repository/ SenderRepository.php. Voorzie de correcte namespace voor dit bestand.

Binnen SenderRepository gebeurt de interactie met de databank.

SenderRepository heeft pdo als private eigenschap.

Binnen de public constructor wordt de waarde voor pdo voorzien. Pdo moet een object van de klasse PDO zijn, gebruik hiervoor type-hinting.

Maak de public methode getSenderById. Deze methode heeft een id als argument , je mag evan uitgaan dat id een int is.

De sender met het opgegeven id wordt samen met al zijn messages bevraagd uit de databank. Er wordt vervolgens aan de hand van de opgevraagde gegevens een sender object aangemaakt en teruggegeven.

Je mag hierbij gebruik maken van de onderstaande sql-query

SELECT sender.name, message.id, message.contents FROM sender,message WHERE sender.id = message.sender\_id and sende.id=1

Wanneer er geen sender met het opgegeven id in de databank staat wordt de waarde null teruggegeven.

Waneer er een PDO-exception opgeworpen wordt bij de interactie met de databank, dan wordt deze PDO-exception opgevangen en wordt er een ModelException opgeworpen.

***Ingave.html***

Ongewijzigd laten

<!doctype html>  
<html lang="en">  
<head>  
 <meta charset="utf-8">  
 <title>Ingave</title>  
</head>  
<body>  
 <form action="verwerk.php" method="get">  
 <input type="number" name="senderid">  
 <button type="submit" class="btn btn-primary">Submit</button>  
 </form>  
</body>  
</html>

***Verwerk.php***

Maak een autoloader voor de bovenstaande PHP-klassen. Gebruik de klassen via de autoloader in verwerk.php. van de ingegeven senderid wordt in verwerk.php de onderstaande weergave gemaakt.

Sende r met name tim heeft 3 berichten:

Ok  
😊  
ok

Via de gespecifieerde id van de sender wordt de name, het aantal berichten en de contents van elk bericht afgedrukt. Als de opgegegeven id niet bestaat wordt dit gemeld:

111 niet gevonden

Je kan vertrekken van de volgende code:

$user = 'root';  
$password = 'root';  
$database = 'examenwa2019';  
$server = 'localhost';  
$pdo = **null**;  
**try** {  
 $pdo = **new** PDO("mysql:host=$server;dbname=$database", $user, $password);  
 $pdo->setAttribute(  
 PDO::*ATTR\_ERRMODE*,  
 PDO::*ERRMODE\_EXCEPTION* );  
 $senderRepository=**new** SenderRepository($pdo);

# Oef2 (4 punten)

***Questionnaire:json***

{  
 "title": "Some relevant questions",  
 "questions": ["What is the purpose of life?", "Are you happy?"]  
}

***Index.html***

<!DOCTYPE html>  
<html>  
<head>  
 <title>oef2</title>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
</head>  
<body>  
 <button id="button\_get\_questions">Get questions</button><br/><br/>  
 <div id="div\_output"></div>  
 <script src="assets/js/questions.js"></script>  
</body>  
</html>

***Questions.js***

Pas het bestand assets/js/questions.js aan zodanig dat wanneer op de knop met opschrift ‘get questions’ geduwd wordt een fetch request vertuurd wordt naar het bestand questionnaire.json op de server. De responsewordt ontleed en de onderstaande weergave wordt in de browser getoond.

* titel staat in h1-element.
* het volgnummer van de vraag end e vraag zelf staan in een textnode.
* de html code voor het tekstvelden voor de vraag met volgnummer 1 is van de vorm:  
  <input type=”tekst” id=”q1”/>
* De verticale lijnen zijn hr-elementen

Door op de met opschrift send te klikken worden alle ingegeven antwoorden onderaan getoond

Een deel van de uitwerking en aantal hints kan je terugvinden in questions.js

# Oef3 (3punten)

***Course.js***

Maak de Javascript klasse Course in het bestand src/school/Course.js

Course heeft eigenschappen \_id, \_grade en \_completed.

De constructor heeft als argument een waarde voor de eigenschap \_id. \_grade wordt gelijk gesteld aan 0 en \_completed wordt gelijk gesteld aan false.

Voorzie getters en setter voor alle eigenschappen.

Je mag er vanuit gaan dat \_grade altijd een geheel getal bevat tussen 0 en 20.

***Student.js***

Maak de javascript klasse Student in het bestand src/school/Student.js en Student heeft eigenschappen \_id en \_courses.

De constructor heeft als argument een waarde voor de eigenschap id. \_courses wordt gelijk gesteld aan een lege array.

Voorzie de functionaddCourse. Deze functie verwacht een course-object als argument. Als het argument geen Course is wordt een error opgeworpen ( je kan dit controleren via instanceof) indien het argument wel een course is, wordt de course toegevoegd aan de array\_courses (een element toevoegen aan een array kan via push).

Maak de functie calculateGrade. In deze functie worden alle elemeneten uit\_courses doorlopen. Per completed course wordt de waarde van de grade opgeteld bij de som. De totale eindscore (grade) die teruggekeerd wordt, is de som gedeeld door het aantal cursussen die completed zijn. Waneer het aantal completed courses echter gelijk is aan 0, wordt er een error geworpen.

Via index.html en app.js kan je de toepassing testen.

Student.test.js

Voorzie 2 (jest) tests in het bestand test/js/school/Student.test.js.

1. Defunctie calculateGrade moet een error opwerpen wanneer hij aangeroepen wordt voor een student\_object die nog geen completed course heeft.
2. De functie calculateGrade moet 12 als terugkeerwaarde hebben voor een student die één(completed) course heeft met grade 12.