

Raport 1 - Project Databases

Bruno De Deken (s0080968)

Raphael Assa (s0102981)

Fouad Kichauat ()

Armin Halilovic (s0122210)

March 16, 2015

1 Status

Taken	Afgewerkt	Verantwoordelijke
Ontwerp database (ER-diagram)	X	Volledige groep
SQL queries	X	Bruno + Raphael
Algemeen ontwerp	X	Bruno
Grafisch design	X	Bruno
User login	X	Armin
Authorization	X	Armin
Database	X	Raphael
Python Simulatie	X	Bruno + Fouad
Series	X	Raphael
Exercises	X	Bruno
Design database verbeteren	X	Raphael
Friends + Groups	X	
Answers doorgeven	X	Raphael
Answers controleren		
Statistieken opvragen	X	Raphael
Grafieken	X	Bruno
Vertalingen	X	Fouad
Messages	X	Armin
Filteren		
Sorteren (populariteit)		
Grafieken uitbreiden		
Overzicht exercises (verbetering)		
Nuttige foutboodschappen		
(basis) Aanbevelingen maken		

2 Design

2.1 Keuzes

2.1.1 Programmeertaal

We zijn begonnen met het PHP framework *Laravel (5.0)*. Aangezien niemand van ons enige ervaring had met webdevelopment of webdesign was het een enorme hulp om een soort template te hebben waar we op konden voortgaan. Een bijkomende reden om Laravel te gebruiken is dat het een

zeer uitgebreide en actieve community bevat, evenals veel en duidelijke tutorials. Laravel gebruikt MVC design, wat we dan ook gevolgd hebben. Voor alle grote klassen werd een Controller aangemaakt. Deze Controller bevat de voornaamste php code. De resultaten die in de Controller behaald worden, worden dan gebruikt om Views te creëren die de content geformatteerd renderen (als html code).

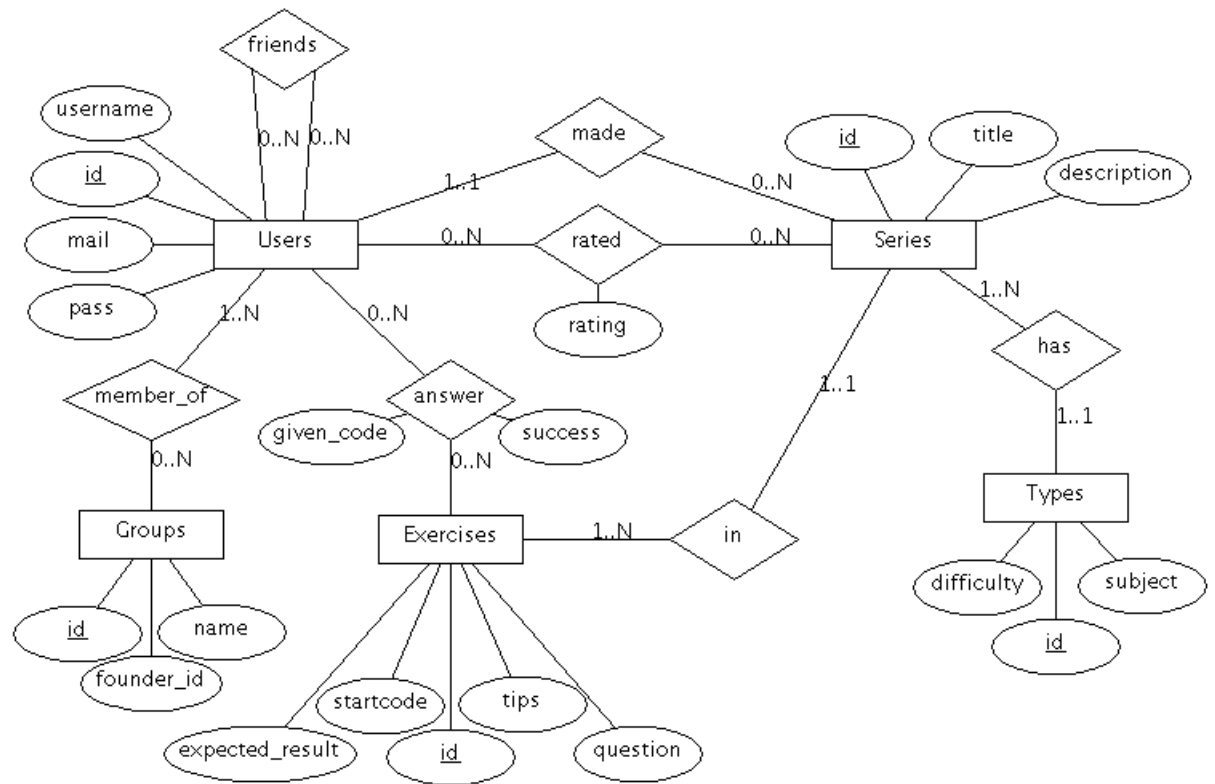
2.1.2 Database communicatie

Om met objecten te kunnen werken worden zogenaamde Models aangemaakt. Dit zijn php klassen die gebruikt worden als container voor de data die via SQL queries worden opgevraagd. Bovendien worden deze gebruikt om een zekere mate van beveiliging toe te voegen doordat in deze models gespecificeerd kan worden welke data wel of niet kan aangepast worden. SQL queries worden verzameld in een enkele file, en worden doorheen het programma gebruikt om te communiceren met de database.

2.1.3 Python interpreter

Voor de python interpreter hebben we gekozen voor *Skulpt*. Skulpt is een 'in-browser' implementatie van python. De voornaamste voordelen hiervan zijn dat beveiliging veel minder belangrijk is, omdat het kwetsbare systeem voornamelijk de eigen computer van de gebruiker is. Een tweede voordeel is dat de interpretatie van python code geen rekenkracht vergt van de servers. Dit maakt dat het systeem voorbereid is voor gelijktijdig gebruik door een groot aantal gebruikers. Een nadeel van dit systeem is dan weer dat Skulpt een implementatie is van Python, en dus niet hetzelfde is als de werkelijke python interpreter. Hierdoor is skulpt niet hetzelfde als de officiële python interpreter. Na afweging van voor- en nadelen hebben we besloten dat de voordelen zwaarder doorwegen. Voornamelijk omdat de applicatie dient om te leren programmeren. De nadruk ligt dus op relatief eenvoudige problemen, waardoor de meer gevorderde zaken die eventueel zouden ontbreken niet van toepassing zijn.

2.2 Database schema



2.3 UML Schema

3 Product

3.1 User Interface

3.1.1 Home page

Zoals zeer veel van de interface staat de home page nog niet helemaal op punt. De bedoeling is om in een oogopslag te weten hoe de website georganiseerd

is. Op de home page moet zeer duidelijk zijn waarvoor de website dient, waar je je kan inloggen/aanmelden en naar wat je kan navigeren.

3.1.2 Navigatie

Om navigatie te vereenvoudigen is er een permanente balk bovenaan geplaatst om snel naar de voornaamste delen te navigeren. Eenmaal daar kunnen meer specifieke zaken opgevraagd worden op de webpage zelf. Zaken die niet van toepassing zijn op een gegeven moment worden verborgen. Een niet-ingelogde bezoeker krijgt bijvoorbeeld geen optie te zien om een oefeningenreeks te maken.

3.1.3 Vertalingen

Om op ieder gewenst ogenblik te kunnen wisselen van taal, is een dropdown-menu voorzien die steeds zichtbaar is. Indien een gebruiker per ongeluk een taal aanduidt die hem compleet vreemd is, kan hij nog steeds op een vlag klikken om een courantere taal aan te duiden. Dit is relevant indien een gebruiker bijvoorbeeld op 'chinese' klikt ipv 'dutch'. Het spreekt voor zich dat de meeste gebruikers die 'dutch' willen aanduiden, niets kunnen met chinese tekens.

3.2 Extra functionaliteit

Aangezien "Skulpt" de mogelijkheid biedt om op een makkelijke manier "turtle graphics" te ondersteunen, zullen we deze uitbreiding alvast ter beschikking stellen. Daarnaast werd ook gekozen om een "messaging" systeem te implementeren zodat gebruikers onderling met elkaar communiceren. Deze keuze gaf dan aanleiding tot het volgende idee om een "chatbox" maken voor elke groep. Hoewel het idee i.v.m. de "chatbox" nog niet officieel werd afgesproken, bestaat er een grote kans dat deze functionaliteit eveneens gimplementeerd zal worden. Verder zijn er nog geen concrete keuzes gemaakt.

4 Planning

5 Appendix

In this section you find all the sql queries that are used throughout the project. If the query is used as part of a function, only the sql query is shown. This is done as not to clutter the appendix, since php code is not relevant for this section.

5.1 Table

Below is a listing of all queries that were used to create the database. It lists the tables of the database and the specifications of each column.

```
/* In case a database is still loaded, remove it */
DROP DATABASE if exists learn2program_db;
```

```
/* Load a new database */
CREATE DATABASE learn2program_db;
USE learn2program_db;
```

```
CREATE TABLE users (
    id int AUTO_INCREMENT,
    pass varchar(255) NOT NULL,
    username varchar(50) NOT NULL UNIQUE,
    mail varchar(50) NOT NULL,
    PRIMARY KEY(id)
);
```

```
CREATE TABLE friends (
    id1 int REFERENCES users(id),
    id2 int REFERENCES users(id),
    PRIMARY KEY(id1, id2)
);
```

```
CREATE TABLE conversations (
    id int AUTO_INCREMENT,
    userA int REFERENCES users(id),
    userB int REFERENCES users(id),
```

```

        PRIMARY KEY(id)
    );

CREATE TABLE messages (
    conversationId int REFERENCES conversations(id),
    message varchar(512) NOT NULL,
    author varchar(50) NOT NULL,          //author stored as username
        for easier displaying
    date timestamp                       /* 'YYYY-MM-DD HH:MM:SS' format */
);

CREATE TABLE groups (
    id int AUTO_INCREMENT,
    name varchar(30) NOT NULL UNIQUE,
    founderId int NOT NULL,
    FOREIGN KEY (founderId) REFERENCES users(id),
    PRIMARY KEY(id)
);

CREATE TABLE members_of_groups (
    memberId int REFERENCES users(id),
    groupId int REFERENCES Groups(id),
    PRIMARY KEY (memberId, groupId)
);

/* Only allow 4 values for difficulty? => easy, intermediate,
    hard, insane */
CREATE TABLE types (
    id int AUTO_INCREMENT,
    subject varchar(50) NOT NULL,
    difficulty ENUM('easy', 'intermediate', 'hard', 'insane') NOT
        NULL,
    PRIMARY KEY(id),
    UNIQUE(subject, difficulty)
);

/* Must make sure that at least 1 exercise belongs to a serie
    => design the site so that this is always the case? */
CREATE TABLE series (
    id int AUTO_INCREMENT,

```

```

    title varchar(50) NOT NULL,
    description varchar(500),
    makerId int NOT NULL,
    FOREIGN KEY (makerId) REFERENCES users(id),
    tId int NOT NULL,
    FOREIGN KEY (tId) REFERENCES types(id),
    PRIMARY KEY(id),
    UNIQUE(title, tId)
);

CREATE TABLE series_ratings (
    userId int REFERENCES users(id),
    serieId int REFERENCES series(id),
    rating ENUM('0', '1', '2', '3', '4', '5') NOT NULL,
    PRIMARY KEY (userId, serieId)
);

CREATE TABLE exercises (
    id int AUTO_INCREMENT,
    question varchar(767) NOT NULL,
    tips varchar(500),
    start_code text NOT NULL,
    expected_result text NOT NULL,
    serieId int NOT NULL,
    FOREIGN KEY (serieId) REFERENCES series(id),
    PRIMARY KEY(id)
);

CREATE TABLE exercises_answers (
    id int AUTO_INCREMENT,
    given_code text NOT NULL,
    success bool NOT NULL,
    uId int NOT NULL REFERENCES users(id),
    eId int NOT NULL REFERENCES exercises(id),
    PRIMARY KEY(id)
);

```

5.2 Triggers

Below is a list of all the triggers used to guarantee proper use of the database. Most triggers are strictly speaking not necessary as they were also implemented in the application itself. The triggers were added nonetheless since they are part of the database design and add an extra level of protection.

```
/* Make sure password, username and mail are not empty. */
```

```
delimiter //
```

```
CREATE TRIGGER check_users
```

```
BEFORE INSERT ON users
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    IF NEW.pass = "" THEN
```

```
        SET NEW.pass = Null;
```

```
    END IF;
```

```
    IF NEW.username = "" THEN
```

```
        SET NEW.username = Null;
```

```
    END IF;
```

```
    IF NEW.mail = "" THEN
```

```
        SET NEW.pass = Null;
```

```
    END IF;
```

```
END; //
```

```
delimiter ;
```

```
/* Make sure the answer is not empty. */
```

```
delimiter //
```

```
CREATE TRIGGER check_answer
```

```
BEFORE INSERT ON exercises_answers
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    IF NEW.given_code = "" THEN
```

```
        SET NEW.given_code = Null;
```

```
    END IF;
```

```
END; //
```

```
delimiter ;
```

```
/* Make sure the title from serie is not empty. */
```

```

delimiter //
CREATE TRIGGER check_title
BEFORE INSERT ON series
FOR EACH ROW
BEGIN
    IF NEW.title = "" THEN
        SET NEW.title = Null;
    END IF;
END;

delimiter ;

/* Make sure the group name is not empty. */
delimiter //
CREATE TRIGGER check_group
BEFORE INSERT ON groups
FOR EACH ROW
BEGIN
    IF NEW.name = "" THEN
        SET NEW.name = Null;
    END IF;
END;

delimiter ;

/* Make sure the subject and difficulty are not empty. */
delimiter //
CREATE TRIGGER check_type
BEFORE INSERT ON types
FOR EACH ROW
BEGIN
    IF NEW.subject = "" THEN
        SET NEW.subject = Null;
    END IF;
    IF NEW.difficulty = "" THEN
        SET NEW.difficulty = Null;
    END IF;
END;

delimiter ;

```

```

/* Make sure question, start_code and expected_result are not
   empty. */
delimiter //
CREATE TRIGGER check_exercise
BEFORE INSERT ON exercises
FOR EACH ROW
BEGIN
    IF NEW.question = "" THEN
        SET NEW.question = Null;
    END IF;
    IF NEW.start_code = "" THEN
        SET NEW.start_code = Null;
    END IF;
    IF NEW.expected_result = "" THEN
        SET NEW.expected_result = Null;
    END IF;
END;

/* Make sure question, start_code and expected_result are not
   empty. */
delimiter //
CREATE TRIGGER check_friends
BEFORE INSERT ON friends
FOR EACH ROW
BEGIN
    IF NEW.id1 = NEW.id2 THEN
        SET NEW.id1 = Null;
    END IF;
    IF EXISTS (SELECT * FROM friends WHERE (id1 = NEW.id1 and id2 =
        NEW.id2) or (id1 = NEW.id2 and id2 = NEW.id1)) THEN
        SET NEW.id1 = Null;
    END IF;
END;

```

5.3 Helpers

This section lists all queries that were used throughout the program. It contains queries such as 'inserts', 'updates', 'selects', etc. Above each query

is the function name in which it was used. Obviously some functions also contain php code, but this was omitted as stated at the top of this page.

```
/* GENERAL HELPERS NEEDED FOR THE SYSTEM */
/* loadusers() */
select * from users;

/* loadUser('name') */
select * from users where username = ? or id = ?, ['name', 'name'];

/* loadName('id') */
select username from users where id = ?, ['id'];

/* loadId('name') */
select id from users where username = ?, ['name'];

/* storeUser('user') */
insert into users (pass, username, mail VALUES (?, ?, ?),
    ['user'->pass, 'user'->username, 'user'->mail]);

/* findFriends('a', 'b') */
'idA' = loadUser('a')[0]->id;
'idB' = loadUser('b')[0]->id;
select * from friends where id1 = ? and id2 = ?, [min('idA',
    'idB'), max('idA', 'idB')];

/* updateUser('id', 'data') */
update users SET username = ?, mail = ?, pass = ? where id = ? or
    username = ?, ['data'->username, 'data'->mail, 'data'->pass,
    'id', 'id'];

/* function storeSerie('serie')*/
insert into series (title, description, makerId, tId) VALUES (?,
    ?, ?, ?), ['serie'->title, 'serie'->description,
    'serie'->makerId, 'serie'->tId];

/* function loadSerieWithId('id')*/
select * from series where id = ?, ['id'];

/* function loadSerieWithIdOrTitle('id')*/
```

```

select * from series where id = ? or title = ?, ['id', 'id'];

/* function loadSerie('title', 'tId')*/
select * from series where title = ? and tId = ?, ['title', 'tId'];

/* function loadAllSeries()*/
select * from series;

/* function loadAllDistinctSeries()*/
select * from series group by title;

/* function updateSerie('id', 'serie', 'typeId')*/
update series SET title = ?, description = ?, tId = ? where id =
    ?,['serie'->title, 'serie'->description, 'typeId', 'id'];

/* function isMakerOfSeries('sId', 'mId')*/
select * from series where (id = ? or title = ?) and makerId =
    ?,['sId', 'sId', 'mId'];

/* function SerieContainsExercises('sId')*/
select * from exercises where serieId = ?,['sId'];

/* function loadExercisesFromSerie('sId')*/
select * from exercises where serieId = ?,['sId'];

/* function storeExercise('exercise')*/
insert into exercises (question, tips, start_code,
    expected_result, serieId) VALUES (?, ?, ?, ?, ?),
    ['exercise'->question, 'exercise'->tips,
    'exercise'->start_code, 'exercise'->expected_result,
    'exercise'->serieId];

/* function loadAllExercises()*/
select * from exercises;

/* function loadExercise('id')*/
select * from exercises where id = ?, ['id'];

/* function removeUnusedTypes()*/
delete from types where id NOT IN (select distinct(tId) from

```

```

Series);

/* Subject and Difficulty => this combination is unique */
/* function loadTypeId('type')*/
select id from types where subject = ? and difficulty = ?,
       ['type' -> subject, 'type' -> difficulty];

/* function loadType1('type')*/
select * from types where subject = ? and difficulty = ?,
       ['type' -> subject, 'type' -> difficulty];

/* function loadType2('id')*/
select * from types where id = ?, ['id'];

/* function loadAllTypes()*/
select * from types;

/* function storeType('type')*/
insert into types (subject, difficulty) VALUES (?, ?),
       ['type' -> subject, 'type' -> difficulty];

/* function storeGroup('group')*/
insert into groups (name, founderId) VALUES (?, ?),
       ['group' -> name, 'group' -> founderId];

/* function loadAllGroups()*/
select * from groups ;

/* function loadGroup('group')*/
select * from groups where id = ? or name = ?, ['group', 'group'];

/* function isFounderOfGroup('groupId', 'founderId')*/
select * from groups where (id = ? or name = ?) and founderId =
       ?, ['groupId', 'groupId', 'founderId'];

/* function addMember2Group('uId', 'gId')*/
'group' = loadGroup('gId');
insert into members_of_groups (memberId, groupId) VALUES (?, ?),
       ['uId', 'group'[0] -> id];

```

```

/* function deleteMemberFromGroup('uId', 'gId')*/
'group' = loadGroup('gId');
delete from members_of_groups where memberId = ? and groupId = ?,
    ['uId', 'group'[0]->id];

/* function noMemberYet('uId', 'gId')*/
'group' = loadGroup('gId');
select * from members_of_groups where memberId = ? and groupId =
    ?,['uId', 'group'[0]->id];

/* function updateGroup('id', 'groupname')*/
'group' = loadGroup('id');
update groups SET name = ? where id = ?, ['groupname',
    'group'[0]->id];

/* function storeAnswer('ans')*/
insert into exercises_answers (given_code, success, uId, eId)
    values (?, ?, ?, ?), ['ans'->given_code, 'ans'->success,
    'ans'->uId, 'ans'->eId];

/* function notRatedYet('uId', 'sId')*/
select * from series_ratings where userId = ? and serieId = ?,
    ['uId', 'sId'];

/* function addRating('nr')*/
insert into series_ratings (rating, userId, serieId) VALUES (?, ?,
    ?), ['nr'->rating, 'nr'->userId, 'nr'->serieId];

/* function unratedSeries('sId')*/
select * from series_ratings where serieId = ?, ['sId'];

/* function storeConversation('id') */
'userId' = loadUser('id')[0]->id;
INSERT INTO conversation (userA, userB) VALUE (?, ?),
    [min(\Auth::id(), 'userId'), max(\Auth::id(), 'userId')];

/* function loadConversation('id') */
'id' = loadUser('id')[0]->id;
SELECT * FROM conversations WHERE userA = ? AND userB = ?,
    [min(\Auth::id(), 'id'), max(\Auth::id(), 'id')];

```

```

/* function loadLatestConversation() */
SELECT C.userA, C.userB
FROM conversations C
    JOIN messages M ON C.id = M.conversationId
WHERE C.userA = ? OR C.userB = ?
ORDER BY date DESC
LIMIT 1,
[\Auth::id(), \Auth::id()];

/* function storeMessage('cId', 'message') */
INSERT INTO messages (conversationId, author, message) VALUE (?,
    ?, ?), ['cId', \Auth::id(), 'message'];

/* function loadAllMessagesInDB() */
SELECT userA, userB, message, date
FROM conversations
    JOIN messages ON conversations.id = messages.conversationId
ORDER BY date;

/* function loadAllMessages('id') */
'id2' = loadUser('id')[0]->id;
SELECT U.username, M.message, M.date
FROM conversations C
    JOIN messages M ON C.id = M.conversationId
    JOIN users U ON U.id = M.author
WHERE C.userA = ? AND C.userB = ?,
[\min(\Auth::id(), 'id2'), \max(\Auth::id(), 'id2')];

/* Get all conversations for the logged in user, then only select
the latest message for each of them*/
/* function loadConversationsWithMessage() */
select userA, userB, message, date
from (select C.id, C.userA, C.userB, M.message, M.date
    from conversations C
        join messages M on C.id = M.conversationId
        join users U on U.id = M.author
    where C.userA = ? or C.userB = ?
    order by date desc) as X

```



```
group by id  
order by date desc,  
[\Auth::id(), \Auth::id()];
```
