Raport Project Databases: Maart

1 Status

1.1 Taakverdeling

1. ERM Schema: samen opgesteld. Digitalisatie: Mathias[1]
2. ERM Model omzetten in SQL tables: Alexander, Timo, Elias[2]
3. Parser schrijven: Bruno, Mathias[3]
4. Basis datastructuren: Alexander, Elias
5. *Teams* tabel vullen (Bruno)
6. *Coaches* tabel vullen (Bruno)
7. *Player* tabel moet extra informatie meekrijgen zoals lengte, gewicht, positie, geboortedatum, ... Bruno
8. *Date* attribuut van tabellen in orde brengen (Elias)
9. Feitelijk login script, met sessions, registraties, login pagina, ... (Alexander)
10. Verdere implementatie van *get* queries voor de klassen. (Elias)
11. Pagina's en algemeen ontwerp van de site (Timo en Mathias)
12. *Analyse van de data* Timo
13. *Herstructureren van de klassen* Elias
14. *Crawler die automatisch nieuwe wedstrijden binnenhaalt, en de scores update van gespeelde wedstrijden.* Bruno
15. *Informatie includen op pagina's* Mathias
16. *RSS feed ophalen* Mathias

[1] De verdere digitalisatie en verdere aanpassingen werden door Alexander gedaan.

[2] De verfijning hiervan (wat het meeste tijd in beslag neemt) werd vooral door Timo gedaan.

[3] Bruno heeft zich vooral toegelegd op de parser, en heeft er dan ook veruit het meeste voor gedaan.

17. *Configuratie scherm gebruiker* Alexander
18. *Pagina's stylen, nadat ze geïmplementeerd zijn* Alexander[4]
19. *Update functies/queries voor goals en wedstrijden* Elias
20. *Functies voor de 'bets'* Alexander
21. *Geavanceerde view, met statistieken e.d.* Mathias
22. *Parser afmaken* Bruno
23. *Prognose* Timo
24. *Constraints* Timo
25. *Bets uitbreiden* Alexander
26. *Unit tests database* Elias
27. *Laatste details aan user interface* Mathias

2 Design

2.1 UML diagram

2.2 MVC

Om geen spaghetti te maken van de implementatie van de pagina's, hebben we geopteerd voor het *Model View Pattern*. We routeren onze urls via *GluePHP* en *mod_rewrite*. Apache geeft alle urls door aan *GluePHP*, welke a.h.v. gedefiniëerde reguliere expressies de url zal mappen op een bepaalde klasse. Het framework zal dan de GET of POST functie (afhangende van het type request) uitvoeren op de nieuwe instantie van de juiste klasse. (In onze implementatie hebben we er echter voor gezorgd dat het framework dat object ook terugkeert, zodat we het verder kunnen gebruiken. Verder hebben we ook een catch-all controller toegevoegd aan het framework om error pagina's te kunnen genereren.)
Zo laten we *GluePHP* een instantie van een *Controller* aanmaken, die we bij het aanmaken de nodige datastructuren en informatie zal inladen. Op elke controller kunnen we dan een *template* functie oproepen, die het overeenkomstige thema gedeelte zal insluiten op de pagina.

[4] Alexander heeft vooral de gebruikers- en wed pagina's van een design voorzien. Het algemeen design werd vooral door Mathias gedaan, gebruik makende van Bootstrap

```
$urls = array(
        'error' => 'Controller\Error', //Catch all
        INSTALL_DIR . 'player/(\d+)' => 'Controller\
            Player',
        INSTALL_DIR . 'competition/(\d+)' => '
            Controller\Competition',
        INSTALL_DIR . 'match/(\d+)' => 'Controller\
            Match',
        INSTALL_DIR . 'news' => 'Controller\News',
);

$controller = glue::stick($urls);
```

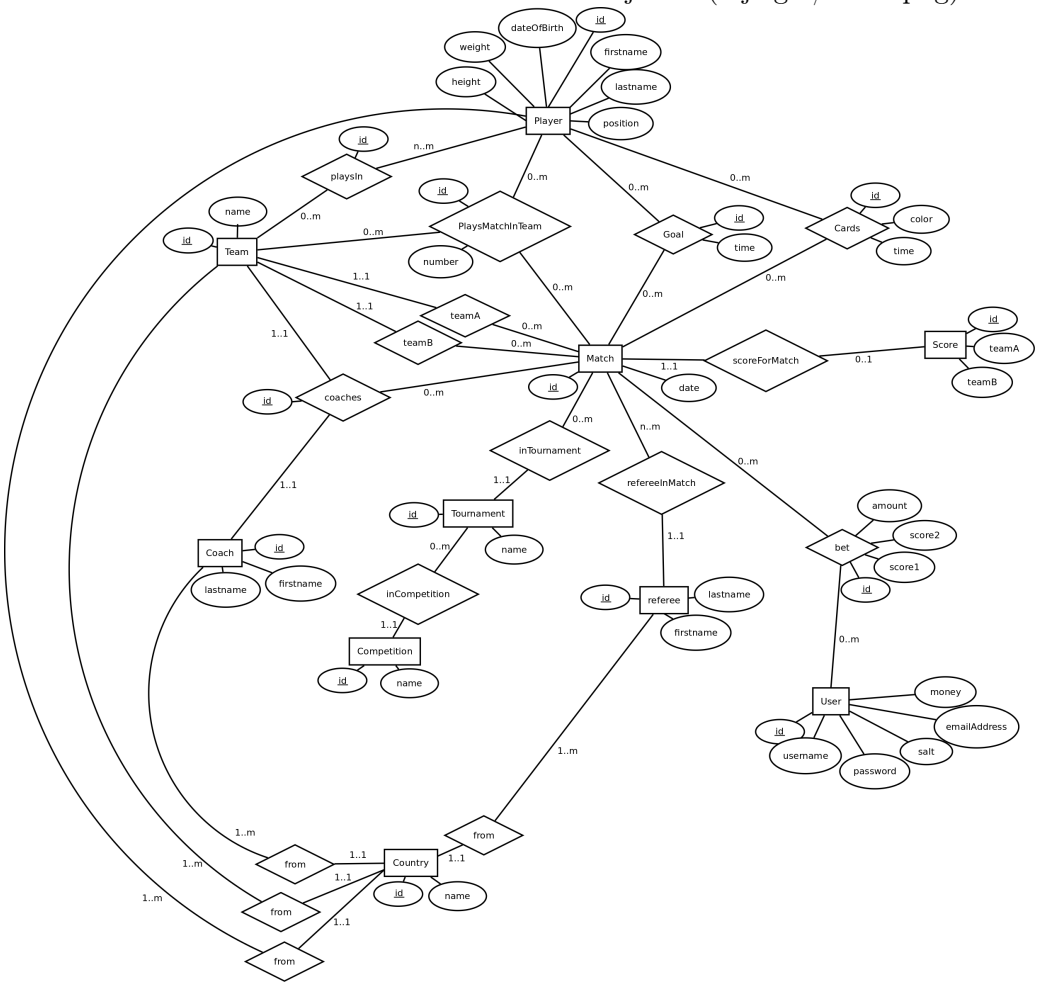
```
//Include the header template
include(dirname(__FILE__) . '/theme/header.php');


//Include the theme part
$controller->template();


//Include the footer template
include(dirname(__FILE__) . '/theme/footer.php');
```

3 Database

3.1 Schema (ER-diagramma)

Bij het aanvatten van een project zo omvangrijk in database structuur als dit, is het erg belangrijk om van in het begin een solide databasestructuur af te spreken. Hiervoor hebben we dan ook meteen onze eerste samenkomst opgeëist. We hebben goed nagedacht over welke zaken en hoe we deze met elkaar in verband gingen moeten brengen (wat niet zo eenvoudig was gezien niemand van onze groep een voetbalkenner is).

Het heeft alleszins zijn vruchten afgeworpen gezien onze huidige (geïmplementeerde) databasestructuur erg weinig afwijkt van de eerste kribbel die we destijds op papier gezet hadden (met uitzondering van het nog niet geïmplementeerde deel).
In de ERM zijn we vertrokken vanuit de entiteiten Match, Player en Team. Hierop hebben we de rest van het 'voetbal' gedeelte van onze ERM gebouwd. Ten slotte hebben we nog een 'gebruikers' gedeelte toegevoegd.
Gezien de omvang van de ERM kan het zijn dat u deze apart moet openen om deze voldoende leesbaar te kunnen bekijken. (bijlages/ERM.png)



3.2 Constraints

4 User Interface

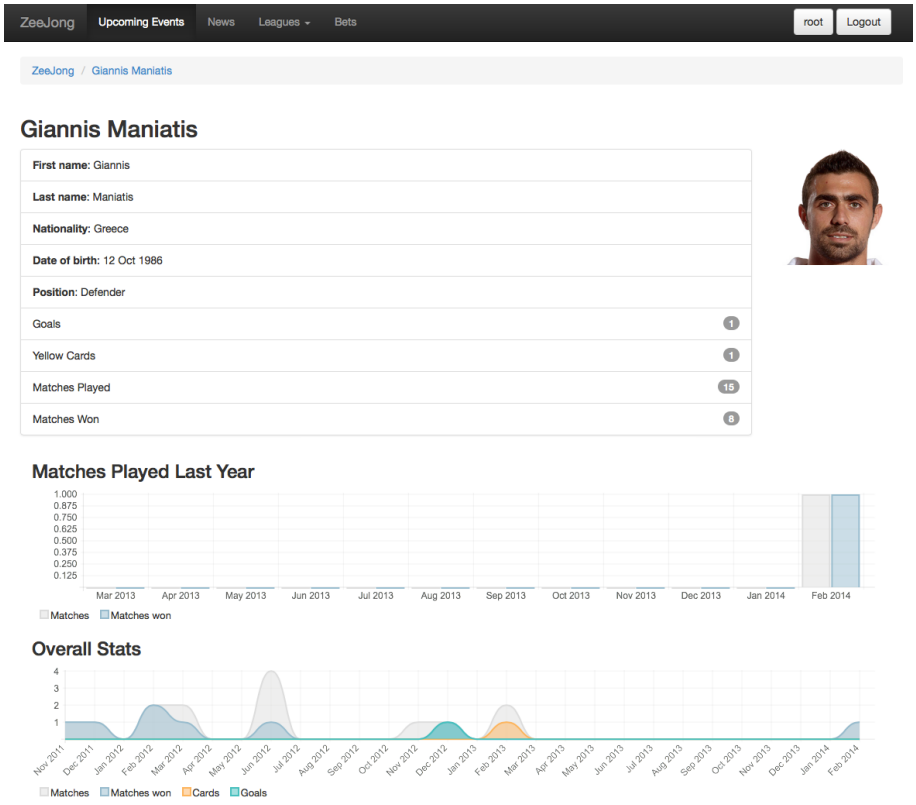
De user interface concentreert zich vooral op matchen (met de voorspellingen en weddenschappen) en de spelers. Gebruikers navigeren vanaf de homepage door de competities, seizoenen (binnen zo'n competitie) en krijgen dan een overzichtspagina van een bepaalde wedstrijd. Op deze match pagina staat dan wat algemenen informatie zoals de score, scheidsrechters, ..., geflankeerd door de teams met hun spelers (voor die bepaalde match) opgelijst.
Gebruikers kunnen te allen tijde doorklikken op spelers, teams, coaches, competities, om zo telkens weer op een overzichtelijke pagina te komen waarop wat meer informatie staat.
Bij de spelers voorzien we de bezoeker van de nodige statistische data over een speler. Zo staan er het aantal gespeelde wedstrijden, gewonnen wedstrijden, kaarten en goals. Daarnaast staat er ook wat metadata over de spelers, zoals diens nationaliteit, positie op het veld, lengte, gewicht, etc. Onderaan de pagina kan de bezoeker een grafiek zien met de wedstrijden van het laatste jaar. Daaronder staat een overzichts grafiek die de tijdlijn van de speler voorstelt. Met daarop alle matchen van zijn carrière.
(Zie figuur 1 p. 6 voor een voorbeeld van een speler pagina.)

Om het design tot een goed einde te brengen hebben we gebruik gemaakt van het *Bootstrap* framework. Hierdoor hadden we een stevige basis om op te steunen bij het stylen van de pagina's. (Voor de installatie pagina hebben we echter de minimalistische framework *Pure CSS* gebruikt.)
De grafieken worden in javascript gerenderd op de html5 canvas d.m.v. de javascript module *ChartJS*.
Intern hebben we slechts één framework gebruikt, namelijk *GluePHP*. Dit framework zet urls om in klasse objecten op basis van reguliere expressies. Zo konden we met Apache's *mod_rewrite* onze urls afhandelen.

5 Extra Functionaliteit

5.1 Selector class

```
$sel = new \Selector('Coaches');
$sel->filter([['teamId', '=', $teamId]]);
```



Figuur 1: Schermafbeelding van de speler pagina

```
$sel->filter([['matchId', '=', $matchId]]);
$sel->join('Coach', 'coachId', 'id');
$sel->select(['Coach.*']);

$result = $this->select($sel);
$coaches = $this->resultToCoaches($result);
return $coaches[0];
```

5.2 Unit Test Framework

Om ons ervan te vergewissen dat database de functie volledig werken, hebben we unit tests voorzien. Voor vele talen vind je dan zeer snel een zeer eenvou-

dig framework. Dit was echter niet meteen snel te vinden voor PHP. Daarom heeft Mathias zelf een test framework geschreven, waarmee het schrijven van unit tests veel gemakkelijk moet gaan.

Een test *case* wordt gedeclareerd door simpelweg een PHP klasse te declareren. Een test *section* binnen zulk een *case* is dan gewoon een functie met daarin enkele uitvoerbare tests. Voor het uitvoeren van de tests zijn er enkele *REQUIRE* functies, zoals *require_true, require_false, require_equal, require_notequal* en *require_exception.*
Een voorbeeld van een test bestand ziet er dan als volgt uit:

```
class Example extends UnitTest {

        public function AnotherTest() {

                $this->REQUIRE_EQUAL($var, $this->
                    testVarA);
                $this->REQUIRE_NOTEQUAL('one', 'two'
                    );
                $this->REQUIRE_TRUE('one' == 'one');
                $this->REQUIRE_FALSE('one' == 'two')
                    ;

        }

        public function ExceptionTest() {

                $this->BEGIN_REQUIRE_EXCEPTION();

                //A statement between BEGIN_REQUIRE
                    and
                //END_REQUIRE should throw an
                    exception
                //in order to make the performed
                    test 'passed'.

                divide(2, 0);
```

```
                $this->END_REQUIRE_EXCEPTION();

        }

}
```

De gebruiker van het test framework plaatst alle te uitvoeren tests in de *test* map, en het framework handelt de rest af. Je krijgt uiteraard een overzichtelijke html output bij het runnen van de tests (Zie figuur 2 p. 8).

Figuur 2: HTML uitvoer van unit tests

5.3 RSS nieuwsfeed

Om de gebruikers meer interessante inhoud op de website te bieden, hebben we onze site voorzien van een *News* pagina, waarop we met RSS de laatste voetbal nieuwtjes binnehalen. Omdat zelf een RSS parser schrijven geen

gemakkelijke klus is, gebruiken we *SimplePie RSS*, een nieuwer alternatief voor *Magpie RSS*.

6 Planning

Voor de volgende deadline verwachten we om het wedden op matchen volledig geïmplementeerd te hebben. Daarnaast zou er ook een administratie pagina moeten komen voor dit geheel, waar de admins hun betting site in de gaten kunnen houden, en handmatig wedstrijden toevoegen of wijzigen.

Zoals we reeds deden, zullen we ook nu weer elke donderdag afspreken om afspraken te maken en issues te bespreken.

7 Appendix

7.1 Volledige Queries

```
SELECT veldna(a)m(en)
 FROM Tabelnaam
  [WHERE conditie]
  [GROUP BY veldnaam [, veldnaam ...]]
  [HAVING conditie2]
  [ORDER BY veldnaam [ASC | DESC] [, veldnaam [ASC |
     DESC] ...]];
```