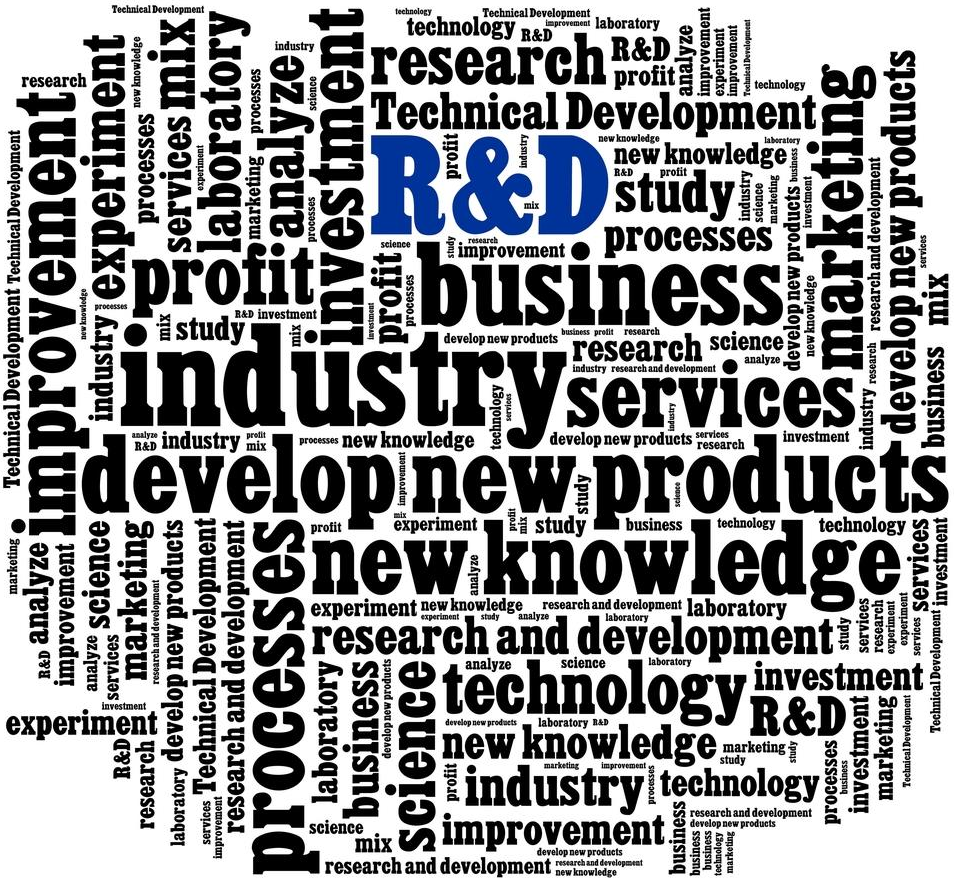
```
public class EindVerslag{
   private String group = "CampGemini";
   private String[] authors = new String[]{
       "Niels", "Nick", "Robin", "Sebastiaan", "Dennis"
   };
}
```



Dennis Fontein	*s4423445*
Sebastiaan Beke	*s4605101*
Robin Elbers	*s4225678*
Niels van Drueten	*s4496604*
Nick Nibbeling	*s4616146*

Voorwoord

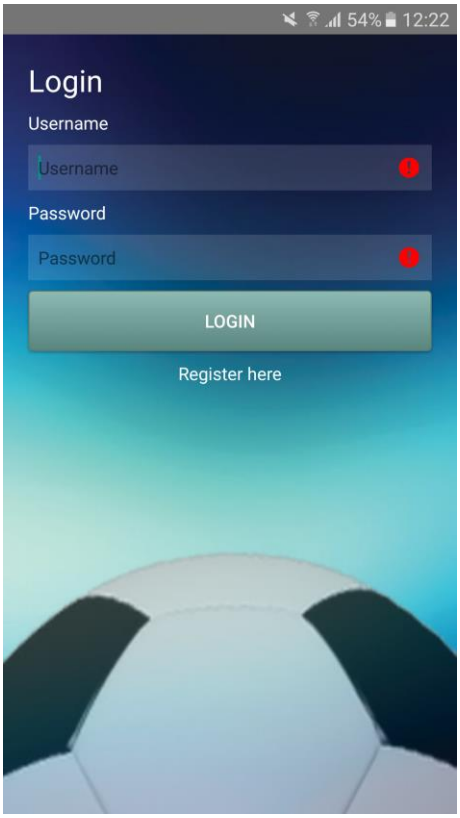
In dit verslag zullen we een beeld schetsen van de applicatie. We laten zien hoe het product tot stand is gekomen en welke beslissingen wij daarbij genomen hebben. Dit document zal beginnen met een beschrijving van de applicatie, gevolgd door het ontwerp. Daarnaast zullen wij reflecteren op het proces dat wij dit kwartaal doorlopen hebben. Dit verslag wordt afgerond door een conclusie.

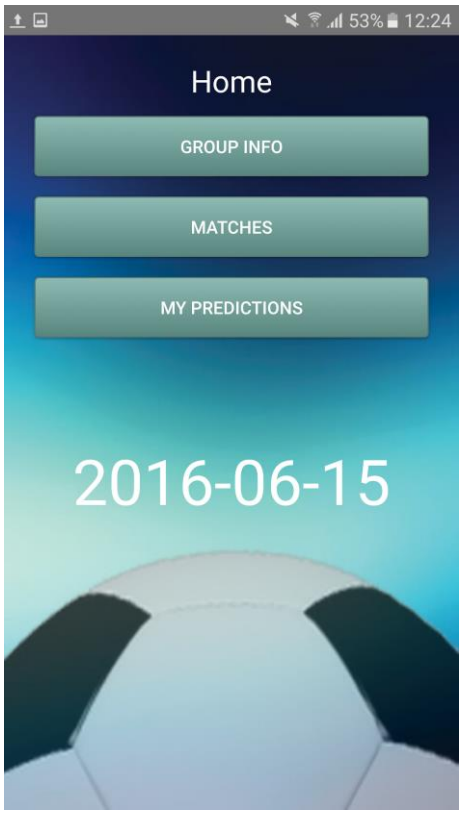
Inhoudsopgave

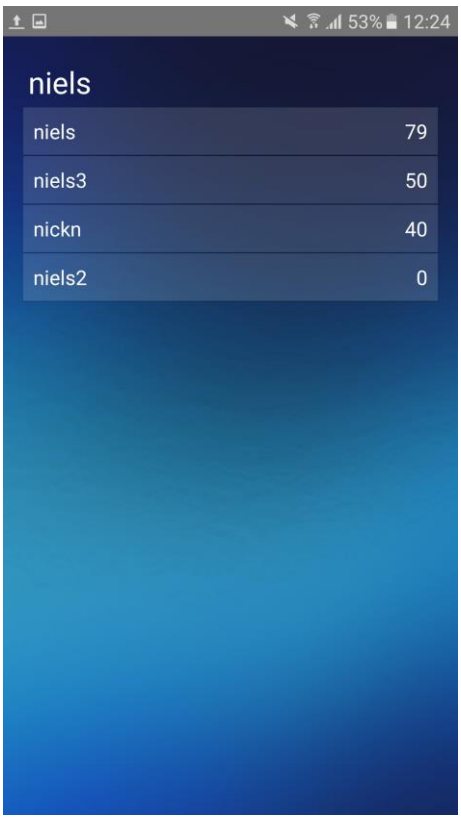
Beschrijving

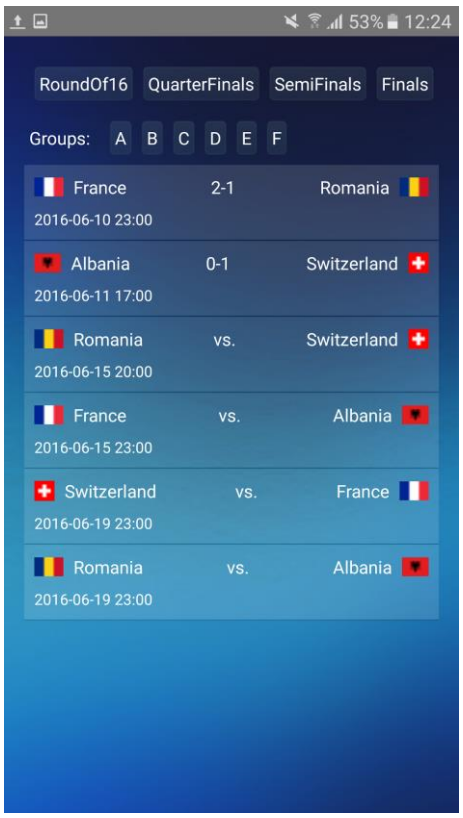
Inleiding

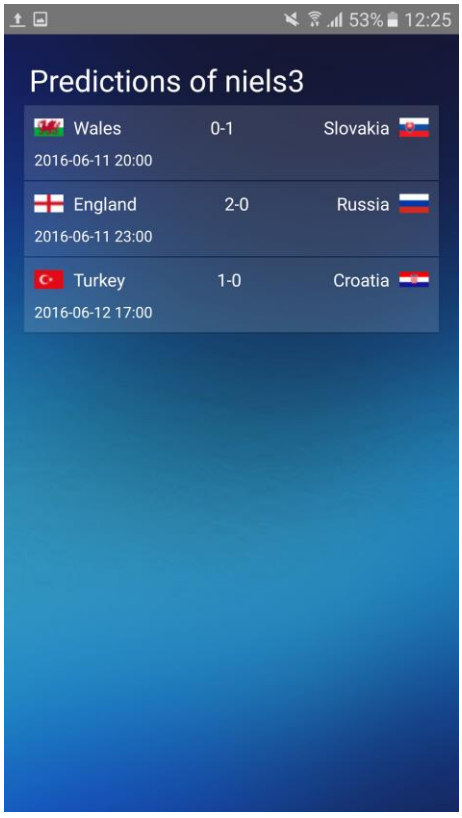
Het product is een applicatie waarmee gewed kan worden op wedstrijden van het Europees Kampioenschap Voetbal 2016. Om dit te doen maakt de gebruiker een groep aan of hij sluit zich aan bij een bestaande groep. In deze groep heeft de gebruiker een puntenaantal. Als een gebruiker een uitslag goed heeft ingevuld wordt er automatisch een aantal punten bij zijn score opgeteld. Ook voor het voorspellen van de winnaar of een goed doelsaldo van één van de teams krijgt de gebruiker al punten. De andere gebruikers die in dezelfde groep zitten kunnen in een lijst de hele groep met ieders punten zien. Hier volgen een paar screenshots van de UI van onder andere de login, homescreen, groupinfo, matches en predictions van de app.






53% 12:24
niels
niels 79
niels3 50
nickn 40
niels2 0


53% 12:24
RoundOf16
QuarterFinals
SemiFinals
Finals
Groups: A B C D E F
France 2-1 Romania
2016-06-10 23:00
Albania 0-1 Switzerland
2016-06-11 17:00
Romania vs. Switzerland
2016-06-15 20:00
France vs. Albania
2016-06-15 23:00
Switzerland vs. France
2016-06-19 23:00
Romania vs. Albania
2016-06-19 23:00


53% 12:25
Predictions of niels3
Wales 0-1 Slovakia
2016-06-11 20:00
England 2-0 Russia
2016-06-11 23:00
Turkey 1-0 Croatia
2016-06-12 17:00

Productverantwoording

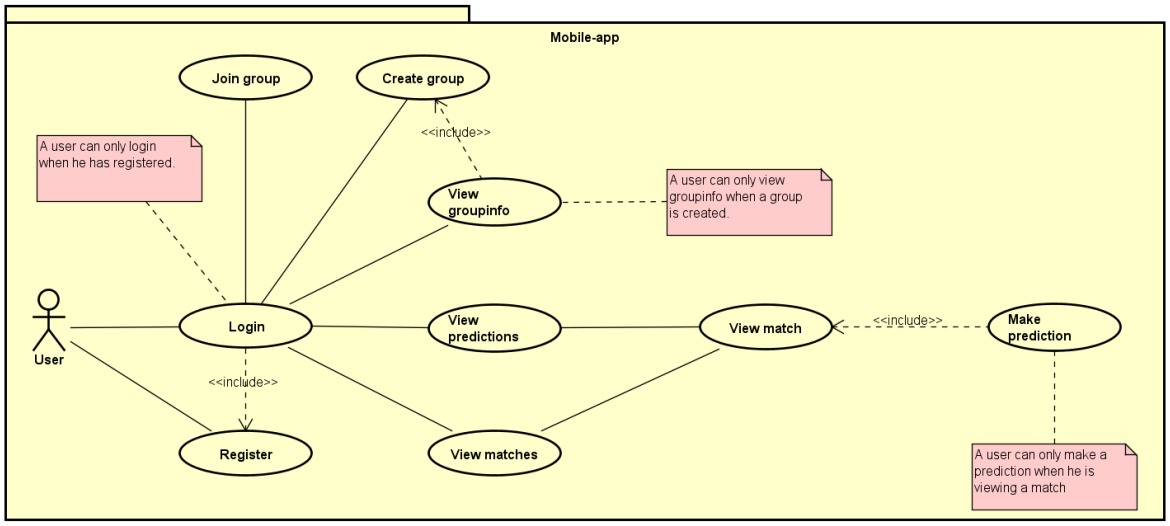
Toen wij eenmaal ons plan voor de applicatie hadden geïntroduceerd, kregen wij te horen dat dergelijke applicaties al zouden bestaan. Echter waren wij de eerste die een applicatie voor het EK van 2016 zouden maken. Terwijl we bezig waren met de applicatie verschenen er al applicaties met dezelfde functie op de Google Play Store. Natuurlijk zijn er door de jaren heen verschillende poule applicaties uitgebracht voor het EK en WK. Wat ons echter opviel is dat je bij deze applicaties, vaak in een enorme groep van mensen terecht kwam. Zo had je niet het overzicht om te zien wat je vrienden deden. Wij besloten dit probleem aan te pakken en hebben een handige functie geïmplementeerd. Gebruikers kunnen hun eigen groepen aan maken. Handig als je bijvoorbeeld een privé-groep voor alleen familie of vrienden wilt samenstellen. Het enige wat je daarvoor moet doen is jouw groepsnaam doorgeven.

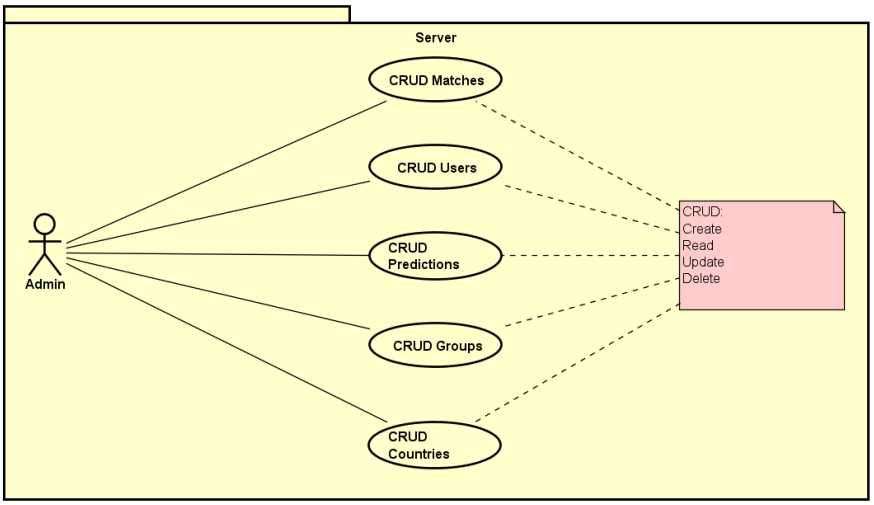
Specificaties

Functionele eigenschappen

Om de gewenste functionaliteit van de app duidelijk te maken hebben we een use-case diagram gemaakt. Dit is het eerste wat wij gemaakt hebben. De use-cases werden verdeeld over de verschillende groepsleden. De verschillende use-cases werden later samengevoegd.

De gebruiker zal eerst moeten registreren voordat hij de app kan gebruiken. Dan kan hij inloggen met zijn gebruikersnaam en wachtwoord. Vervolgens kan hij óf een groep starten óf een zich toevoegen aan een bestaande groep. De gebruiker kan nu informatie zien van de groep waar hij in zit, hij kan zijn eigen voorspellingen inzien en hij kan de wedstrijden inzien. De hoofdfunctionaliteit is het maken van een voorspelling. De gebruiker kan dit doen als hij een wedstrijd inziet. Bij de lijst met eigen voorspellingen kan de gebruiker ook de wedstrijd inzien waarvoor hij een voorspelling heeft gemaakt.





Niet-functionele eigenschappen

We hebben de niet-functionele eigenschappen bepaald aan de hand van het FURPS-model. Het eerste onderdeel, functionality is hierboven al in het use-casediagram uitgewerkt. Voor de niet-functionele eigenschappen volgt hier een opsomming:

- Usability:
  Knoppen zijn duidelijk in beeld. Het is voor de gebruiker duidelijk welke acties hij kan ondernemen op elk scherm. Als een handeling faalt zal de gebruiker hiervan op de hoogte gesteld worden.
- Reliability:
  De server heeft een uptime van minstens 99%. De app crasht niet.
- Performance:
  Voor elke handeling van de gebruiker duurt het maximaal 2 seconden voordat de app reageert. De applicatie verbruikt maximaal 2mb per dag.
- Supportabillity:
  Alle gebruikers met android smartphones met minimaal versie KitKat 4.4 kunnen onze applicatie gebruiken.

Ontwerp

Globale ontwerp

Zie ook het onderstaande diagram voor de globale architectuur van de applicatie.

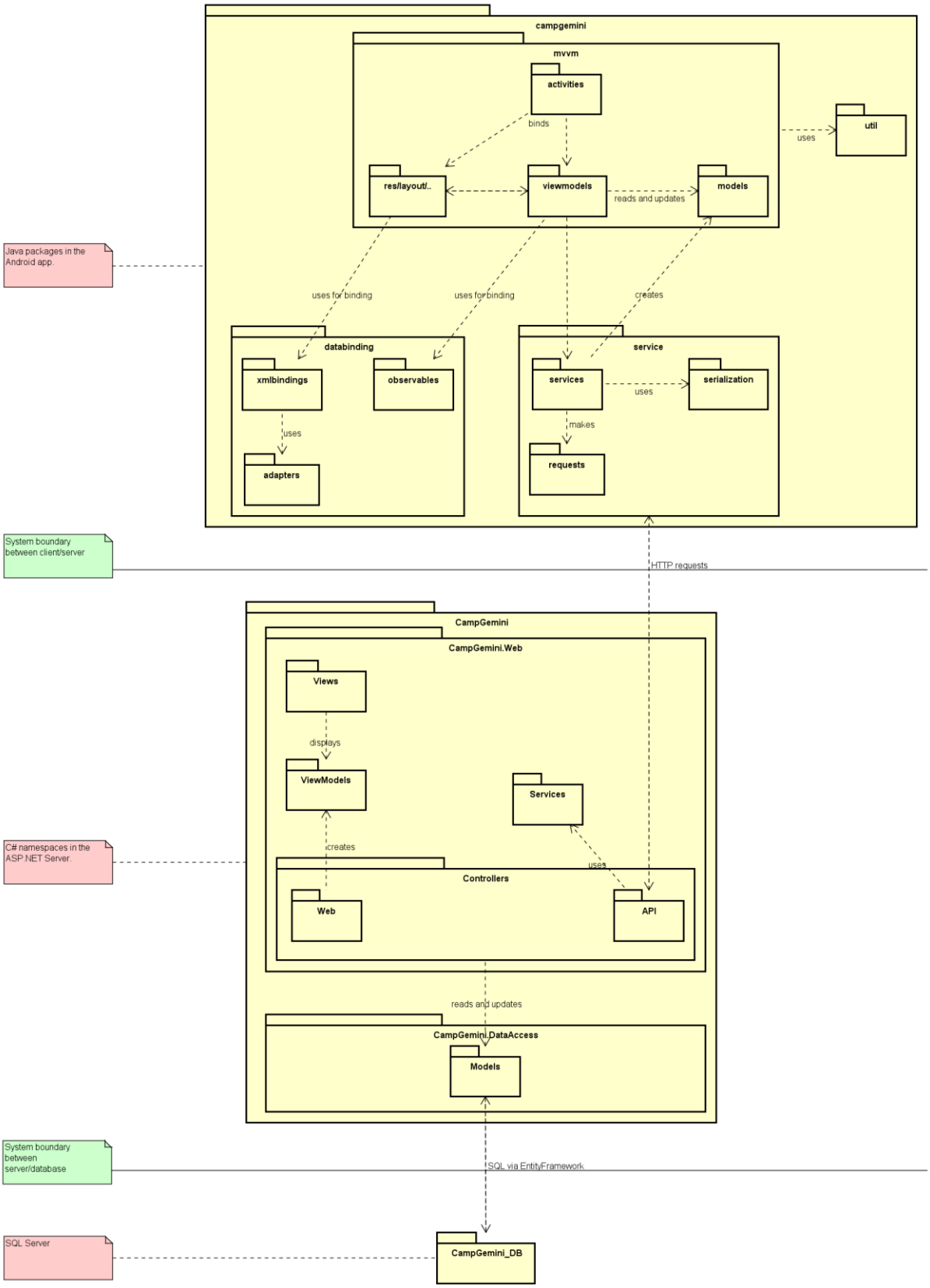
De gehele applicatie bestaat uit drie delen: een android app, een ASP.NET server en een SQL Server database. De server heeft een REST API. De android app kan communiceren met de server door HTTP-requests te sturen. Op de server zorgt het EntityFramework (EF) voor de communicatie met de database. EF is een ORM, hiermee kun je op een objectgeoriënteerde manier met de database praten. Functieaanroepen op DbSet<Model> objecten worden vertaald naar sql queries. DbSet implementeerd IEnumerable.

De android app maakt gebruik van het MVVM pattern (Model-View-ViewModel). In de eerste weken hebben we elk afzonderlijk een use-case gemaakt. Toen alles samengevoegd werd, kwam naar voren dat een aantal ButtonListeners veel Views nodig had. Dit was voor ons een code-smell. De Activities en Listeners waren te sterk gekoppeld aan de Views. De code hebben we toen omgezet naar het MVVM pattern. De activities zorgen nu alleen voor de binding tussen de View en Viewmodel en voor de navigatie tussen verschillende schermen. Voor de binding zijn een aantal speciale klasses en hulpfuncties nodig. Deze staan in de “campgemini.ecpoule.databinding”-package. De Views krijgen een instantie van een ViewModel. In de .xml files kun je direct refereren naar velden van de ViewModel en kun je direct void functies aanroepen. Als een property in een ViewModel veranderd, worden de Listeners (in dit geval dus alleen de View) hiervan genotificeerd. Een ViewModel is niet direct gekoppeld aan een View. Een bijkomend voordeel is ook dat je in de code niet meer naar views hoeft te zoeken door middel van R/id/<…>. In de views zul je ook zelden een id property vinden.

De ViewModels vragen data op van de server door middel van de services. De basis Service-klasse heeft synchrone methodes. Echter kun je op android geen networking doen op de mainthread. Dus is er een subtype AsyncService die asynchrone methods aanbiedt. De asynchrone methodes geven een Future<T> terug. In deze Future<T> zit in de toekomst ooit een waarde. Als je de .get() methode aanroept blocked de main thread tot deze waarde terug is van de server. Het is niet zo mooi dat hij blocked maar het blijkt geen probleem te zijn, omdat dit meestal niet lang duurt[1]. Een alternatief waaraan gedacht is, is om een continuation mee te geven. Dit een functie die wordt aangeroepen in dezelfde thread als de service-call klaar is. Deze continuation krijgt dan als parameter het resultaat van de service-call. Het probleem is alleen dat je in android geen lambda-expressies mag gebruiken. Het zou dus moeten met een anonieme inner class en daar wordt de code niet mooier van. Verder heeft het gebruik van continuations geen bijzonder voordeel boven de huidige oplossing.

De service krijgt een HTTP-response terug die JSON bevat. De JSON wordt door de service gedeserialiseerd naar het juiste object.

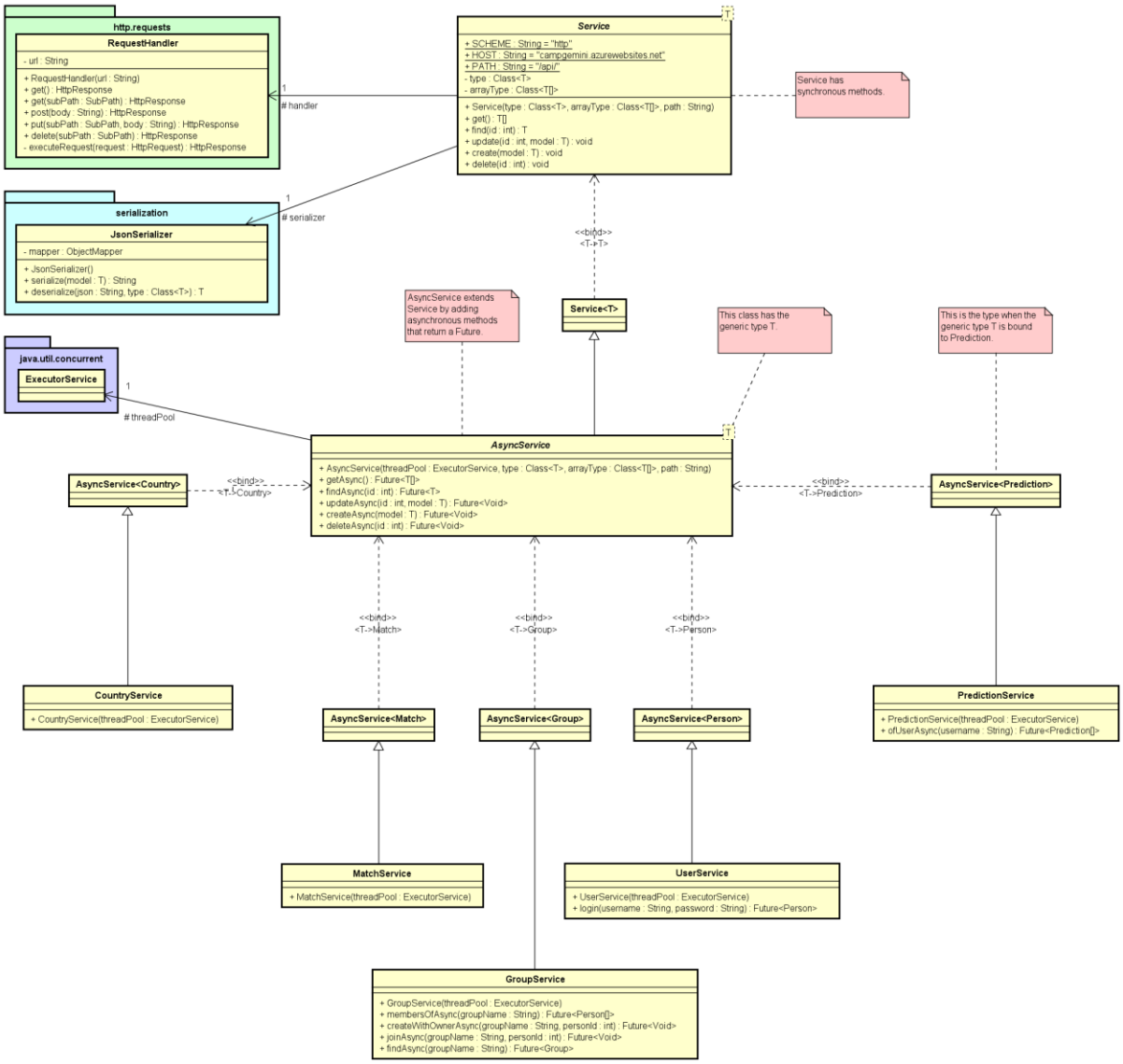
[1] De eerste keer dat de server aangeroepen wordt, reageert de server traag. Daarna is de server een stuk sneller, omdat de server het een en ander heeft kunnen cachen en omdat de JIT-compiler aan het werk is geweest.


campgemini
mvvm
activities
binds
util
uses
res/layout/..
viewmodels
reads and updates
models
Java packages in the Android app.
uses for binding
uses for binding
creates
databinding
xmlbindings
observables
uses
adapters
service
services
uses
serialization
makes
requests
System boundary between client/server
HTTP requests
CampGemini
CampGemini.Web
Views
displays
ViewModels
Services
creates
uses
Controllers
Web
API
C# namespaces in the ASP.NET Server.
reads and updates
CampGemini_DataAccess
Models
System boundary between server/database
SQL via EntityFramework
SQL Server
CampGemini_DB

Detailontwerp

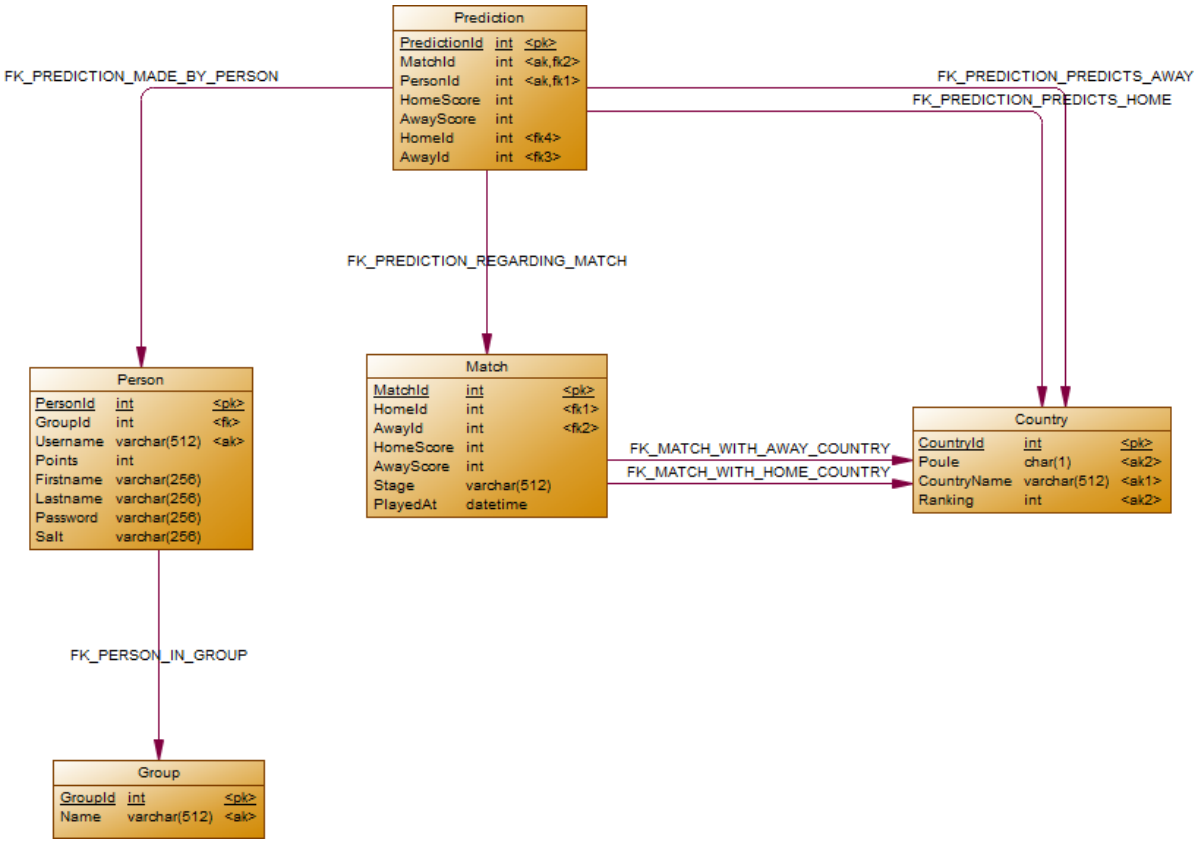
Class diagram

Klasse diagram van de services in de android app.



Entity Relation Diagram

Fysiek model van de database.



Ontwerpverantwoording

We hebben twee functionele beslissingen en drie technische beslissingen uitgewerkt.

Functionele beslissingen

1. Wij hebben ervoor gekozen om het maken van eigen groepen te implementeren in de applicatie. Dit vonden wij nodig om de poule persoonlijk te maken voor de gebruiker. De gebruiker kan nu samen met vrienden of familie een poule opzetten. We hadden er ook voor kunnen kiezen om één grote gezamenlijke poule te creëren. Dit zou een stuk makkelijker voor ons zijn geweest. Maar dat maakt de applicatie waarschijnlijk een stuk minder origineel ten opzichte van zijn concurrenten.
2. Het puntensysteem zorgt voor een eerlijke verdeling van de punten. We hebben het zo gemaakt dat iedereen punten weet te halen. Het is natuurlijk niet leuk dat iemand bij de kwartfinales nog steeds 0 punten heeft. Daarom hebben wij ervoor gekozen voor de volgende puntenverdeling:

Voorwaarde	Punten
Uit score goed	4
Thuis score goed	4
Bonus voor uit en thuis score goed	8
Winnaar van een wedstrijd in de groepsfase goed	2
Winnaar van een wedstrijd in de round of 16 goed	4
Winnaar van een wedstrijd in de kwartfinales goed	8
Winnaar van een wedstrijd in de halve finales goed	16
Winnaar van een wedstrijd in de finale goed	32
Deelname van een land in de round of 16 goed	2
Deelname van een land in de kwartfinales goed	4
Deelname van een land in de halve finales goed	8
Deelname van een land in de finale goed	16

   Je hoeft in onze applicatie niet alle landen te voorspellen. Wil je kans maken op de punten die je krijgt bij het goed voorspellen van de deelnemers aan bijvoorbeeld de halve finale, dan moet je die natuurlijk wel van tevoren hebben voorspeld. We hebben er voor gekozen om in het ontwerp de goede uitslag groen te laten kleuren en de foute uitslag rood. We hadden ervoor kunnen kiezen om de puntenverdeling klein te houden, maar hebben er toch voor gekozen om het complexer te maken. Zo kan iemand die bijna alles fout heeft in de groepsfase, toch nog veel punten behalen in de volgende fases.

Technische beslissingen

Waarom ASP.NET als server?

In de weken van het project hebben we geprobeerd de server in Java te schrijven. Het probleem hiermee was echter dat je veel libraries moet toevoegen om het werkend te krijgen en deze moeten allemaal goed samenwerken. Er moest een library toegevoegd worden voor de REST service, voor JSF, voor hibernate, voor postgresql, voor serialisatie en sommige van deze waren onderling weer afhankelijk. Het was te veel werk om alles te configureren zodat het goed samenwerkt. Voor het ASP.NET project hebben wij geen extra libraries nodig gehad. Het is standaard goed geconfigureerd en voldeed aan al onze eisen. Daar kwam het voordeel bij dat het makkelijk te deployen is op een virtuele Azure server. Met Java zouden we zelf een Glassfish of Tomcat server moeten hosten op een linux server.

Waarom SQL Server als database?

De eerste keuze voor database was bij Java PostgreSQL, maar toen de keuze voor ASP.NET was gemaakt was de keuze voor SQL Server ook makkelijk gemaakt. SQL Server kon namelijk samen met de webserver gedeployed worden op Azure. Daarnaast zou het niet voor de hand liggen om een NoSQL database te kiezen, omdat onze data relationeel is. Daarnaast is het voornaamste voordeel van NoSQL niet van toepassing hier, namelijk dat het makkelijker is om te schalen. Het is onwaarschijnlijk dat de hoeveelheid data die naar verwachting in de database komt, voor performance problemen zal zorgen. Als performance wel een probleem zou worden, dan zullen we eerst kijken of er indices bij moeten in de database.

Waarom hebben alle entiteiten in de database een Id?

Het is zo dat de meeste entiteiten in de database een natuurlijke sleutel hebben. Username voor Person bijvoorbeeld of CountryName bij Country. Er is voor gekozen om voor elke entiteit een surrogaatsleutel te introduceren om de REST API simpeler te maken. De client hoeft dan alleen een Id mee te geven om naar de juiste entiteit te gaan. Daarnaast hebben sommige entiteiten een complexe natuurlijke sleutel. Een Match zou geïdentificeerd worden door <HomeCountry, AwayCountry, Stage>. Dit zou het gebruik van de API moeilijker maken dan alleen een getal te gebruiken. Op de natuurlijke sleutels liggen wel uniqueness constraints om de integriteit van de data te bewaken.

Reflectie

Waar zijn wij tevreden over?

We zijn erg tevreden over ons eindproduct. De meeste dingen die wij in onze applicatie wilden hebben zijn gelukt. We hadden nog meer ideeën maar die hebben we niet af kunnen krijgen binnen de tijd. We wilden graag dat de cliënt een notificatie kreeg na een gespeelde wedstrijd. Ook wilden we een newsfeed maken waarin de belangrijkste gebeurtenissen worden weergeven van jouw groep. Een voorbeeld hiervan is als een speler op een dag veel wedstrijden goed voorspeld heeft. De groep krijgt deze gebeurtenis dan te zien in de newsfeed. We zijn hier niet aan toe gekomen, maar we denken dat we de belangrijkste functionaliteit van de app geïmplementeerd hebben.

Waar wij het meest trots op zijn is het aanmaken en joinen van groepen. In het begin wisten wij niet of dit zou gaan lukken. We twijfelde nog of we gewoon één grote groep zouden samenstellen, waar alle gebruikers worden ingestopt. Dit maakt het natuurlijk minder leuk voor gebruikers die graag alleen willen weten wat hun vrienden hebben gedaan.

Verder zijn wij tevreden over de server. De server werkt naar behoren. De cliënt kan groepen aanmaken, joinen, voorspellen en de wedstrijdinformatie ophalen. De server zorgt dat dit allemaal soepel verloopt.

Een ander belangrijk onderdeel van de applicatie is het voorspellen van wedstrijden. Als dat niet zou werken zou je niks aan deze applicatie hebben. Gelukkig hebben we dit ook weten te bereiken.

Waar zijn wij niet tevreden over?

Wij hebben in het begin besloten om ieder een use-case te laten uitwerken en dit later samen te voegen tot een geheel. Dat is natuurlijk niet erg handig omdat iedereen code op een andere manier gebruikt. Gelukkig is het bij ons uiteindelijk wel gelukt om alles samen te voegen, iets dat we in het vervolg anders zouden aanpakken. Achteraf gezien zou het beter zijn als we git hadden gebruikt om samen te werken.

Van iedere activity was bij ons de lay-out totaal verschillend dan die van een andere activity. Ook dat hebben we moeten rechttrekken. Op een gegeven moment kregen wij in de gaten dat één iemand zich bezig moest gaan houden met de lay-out. Toen hebben we ook styles kunnen toevoegen in de views om een app te krijgen die er uniform uit ziet. De volgende keer zou het goed zijn om eerst styles af te spreken. Dit scheelt werk bij het samenvoegen.

Analyse

De samenwerking heeft bij ons geen problemen ondervonden en dat is dan ook de belangrijkste factor voor onze tevreden kijk op het project. Twee leden van onze groep zijn halverwege gestopt met de opleiden, maar zij hebben toch daarna nog tijd gesteekt in het project. Als iemand ergens op vastliep, kon altijd iemand anders uit het groepje diegene wel helpen. Dat is waarschijnlijk ook de rede dat we uiteindelijk de aparte activities konden samenvoegen. Want als je elkaar helpt, kun je ook weer dingen van elkaar over nemen. Een tweede factor die erg belangrijk was, is het onderzoek gedeelte. Aangezien we zelf moesten uitzoeken hoe we een applicatie moesten maken, heeft iedereen zich erin verdiept om er verstand van te krijgen. De een had er natuurlijk meer verstand van dan de ander, maar dat mocht de pret niet drukken.

Conclusie

Al met al is de applicatie tot een goed einde gekomen. In de toekomst zouden wij het wel anders aanpakken. We zouden dan veel eerder beginnen want we kwamen er al snel achter dat de deadlines dicht achter elkaar zaten. Vooral met de opdrachten van andere vakken erbij. Zo hadden wij net een idee voor onze applicatie die we een of twee dagen later al moesten presenteren. Dat is natuurlijk wel erg kort dag. Over de samenwerking hebben wij niets te klagen. Die is vlekkeloos verlopen. Verder zouden wij in het vervolg ook niet meer ieder voor zich aan een use-case gaan werken. We hebben gemerkt dat het uiteindelijk erg onhandig is om alles later samen te voegen. De volgende keer maken we eerst een gezamenlijk project die goed geconfigureerd is (alle dependencies zijn toegevoegd, styles zijn toegevoegd enz.). Deze zetten we dan op git. Daarna kunnen we afzonderlijk verder aan de use cases. Als we goede afspraken maken zodat we niet in dezelfde files werken, kan git de code voor ons samenvoegen. Al met al het wij het werken aan dit project als prettig ervaren.