17-06-2016

The making of StudLife

Cervisia Technologies Inc.

1 Voorwoord

Voor Research & Development hebben wij in het voorjaar van 2016 een groepsproject mogen doen. Naast kleine opdrachten zoals een usability onderzoek, was ons hoofddoel een goed functionerende, visueel aantrekkelijke Android app te maken. Dit is dan ook het doel van dit verslag. Wij hebben een app genaamd 'StudLife' gemaakt, en nemen u graag mee door het ontwikkelingsproces. StudLife is een spelletje wat doet denken aan Pou en Tamagotchi. Het doel van het spel is om het in-game-karakter zolang mogelijk in leven te houden, en daarbij zoveel mogelijk punten te verdienen. Het verslag is volgens de richtlijnen opgedeeld in drie delen: Een beschrijving, ontwerp en een reflectie. In de beschrijving zal te lezen zijn wat de naam eigenlijk al zegt, een beschrijving van hoe de app werkt, eruit ziet en wat de functionaliteit is. Onder het kopje productverantwoording zal te lezen zijn hoe wij tot het ontwerp zijn gekomen en zullen enkele keuzes daarin worden toegelicht. De reflectie zal een inzicht geven in onze werkwijze, zowel de zaken die goed gingen als de zaken waarbij winst te halen valt. Tot slot zullen wij een conclusie hierover geven.
Dit document is als volgt opgebouwd:

- Beschrijving
  - Inleiding
  - Productverantwoording
  - Specificaties
- Ontwerp
  - Globaal ontwerp
  - Detail ontwerp
  - Verantwoording
- Reflectie
  - Positieve punten
  - Negatieve punten
  - Conclusie

1.1 Auteurs

- Roeland Hoefsloot S4629388
- Dennis Kleverwal S4598164
- Richard van Ginkel S4599047

Contents

2 Beschrijving

2.1 Inleiding

Onze app StudLife bootst het leven van een student overdreven en komisch na. Naast het thuis kunnen zijn om te slapen en het kunnen studeren in college, is er natuurlijk ook de mogelijkheid om uit te gaan. Op elke locatie is er een keuze uit eten en drinken toegespitst op de locatie. Zo is er een mogelijkheid om thuis een pak melk te drinken en kan men tijdens het uitgaan een iconische Nijmeegse toeter eten. Dit kan doormiddel van drag and drop functionaliteit worden gevoerd aan de student. Wordt het eten boven de mond gehouden, dan zal de student zijn mond open doen. Laat men het eten of drinken daar los, dan zal de student dit door een paar keer te kauwen tot zich nemen. Er is een keuze uit gezonde versnaperingen en wat ongezondere voeding, alles voor een eigen geldbedrag. Uiteraard heeft een keuze hierin ook gevolgen voor de student.

Aan de hand van drie statusbalken worden drie gegevens van de student bijgehouden: energie, gezondheid en geluk. Deze termen spreken redelijk voor zich. Energie is de hoeveelheid energie de student nog heeft. Dit daalt met de tijd, maar ook door andere activiteiten die later nog aan bod komen. Dat dalen geldt ook voor gezondheid en geluk. Het is dus zaak om de student goed te onderhouden, anders raken de balkjes leeg. Een leeg balkje betekent dat de student overlijdt, en het spel voorbij is. Ook zijn er drie waardes die doormiddel van een getal worden weergegeven: studie, sociaal en geldbalans. Studie en sociaal beginnen beiden op nul bij een nieuw spel, en kunnen door bepaalde keuzes in het spel toe -of afnemen. Deze afweging moet de speler zelf maken, beiden hoog houden is bij voorbaat lastig gemaakt. Zowel studiepunten, als sociale punten, tellen positief mee voor de topscore die men kan halen. Het is dus noodzakelijk dat de speler nadenkt over de keuzes die hij maakt, en hoe deze zijn student, en dus zijn score, benvloeden.

Figure 1: The startscreen

2.2 Specificaties

Hieronder worden de drie locaties verder uitgewerkt:

2.2.1 Thuis

Thuis is er de mogelijkheid te slapen. Dit kan door een spinner te openen en een tijd aan te klikken voor hoelang er geslapen moet worden. Hier staat ook bij hoeveel energie en gezondheid

er bij komt door het slapen. Tijdens het slapen zal de student zijn ogen sluiten en niets meer kunnen
doen. Eten, drinken en naar een andere locatie gaan is niet meer mogelijk, totdat de student weer ontwaakt. Voor het eten het drinken zijn er de volgende keuzes: melk, wortels, appel, peer en brood. Dit zijn stuk voor stuk vrij gezonde producten en dus zal de gezondheid stijgen als dit gegeten of gedronken wordt. Ook de energie zal toenemen, aan de andere kant wordt de student wel steeds ongelukkiger als er iets gezonds genuttigd wordt.

Figure 2: Thuis locatie

2.2.2 College

In college is het vanzelfsprekend mogelijk te studeren. Dit levert studiepunten op. Hoeveel dit oplevert is weer in een spinner te zien. Hier staan ook de lengte van studeren, de afname van energie en de afname van geluk in. Tijdens het studeren is het hier wel mogelijk om te eten en te drinken, maar het is niet mogelijk naar een andere locatie te gaan. Qua eten en drinken kunnen we kiezen uit water, energydrink, cola, chips en brood. Energydrink, cola en chips slecht voor de gezondheid. Water en brood leveren vooral energie op.

Figure 3: College locatie

2.2.3 Uitgaan

Het uitgaan stelt de student in staat alcoholische versnaperingen te nuttigen. Ook de nodige fastfood is hier te koop. De Nijmeegse toeter, een hamburger, friet en een pizza behoren tot de mogelijkheden. Geen van deze dingen is goed voor de gezondheid, maar de student wordt er wel gelukkiger van. Er is nog een laatste button, in de vorm van twee dobbelstenen. Deze staan voor een te wagen gokje. Nadat de student vijf biertjes op heeft, is het mogelijk deze knop te gebruiken. Na het drukken erop, zal er een scherm tevoorschijn komen. Er zijn tien mogelijke uitkomsten voor dit scherm en welke er naar voren komt, is willekeurig. Onder deze situaties vallen onder andere het koppen van een muur, wat uiteraard slecht is voor de gezondheid, het geven van een rondje, wat extra geld kost, en het vinden van geld. Dit is de manier om de waarde

van sociaal te verhogen. Dit zorgt dus voor een hogere score, en voor de nodige hilariteit.

2.3 Productverantwoording

Tijdens het brainstormen voor onze app kwamen we al vrij snel tot de conclusie dat we iets wilden maken wat mensen zoals wij, andere studenten echt zou aanspreken. Het werd ook al snel duidelijk dat het een spelletje zou gaan worden. De vraag was echter nog wel: hoe combineer je dat tot een goed idee? Na een beetje rond kijken naar andere apps om inspiratie op te doen kwamen wij op het idee om een Tamagotchi te maken, maar dan voor studenten. Wat spreekt studenten meer aan dan college volgen en, nog veel meer, uitgaan? Daarom hebben wij ervoor gekozen het studentenleven na te bootsen in onze app. Een naam was snel verzonnen. Het ging voor het leven van een student, dus vandaar de afkorting StudLife, wat uiteraard staat voor Studentenleven. Zover bekend, bestaat er nog geen andere app die hier heel erg op lijkt. Wel zijn er apps als Pou, een app waarmee je een driehoekig wezentje in leven moet houden. Hoewel dit een uitstekende applicatie is, spreekt deze onze doelgroep niet langdurig aan. De vraag was dan ook hoe wij spelers wel voor langere tijd geboeid konden houden. Dit hebben wij gedaan door de app ook een zekere vorm van hilariteit te geven. Door de tien events die kunnen plaatsvinden na veel drinken trachten humor en verrassing toe te voegen zodat mensen geboeid blijven. Het is echter wel zo dat een aantal grappen die in de app zitten vooral voor eerstejaars informatica-studenten van de Radboud Universiteit bedoeld zijn, maar dit is dan ook tevens onze belangrijkste doelgroep.

3 Ontwerp

3.1 Globaal ontwerp

De ontwikkeling van de applicatie is strict volgens het MVC principe, waarbij er een aantoonbare scheiding wordt gemaakt tussen de model, view en controller. Dit voor overzichtelijke en onderhoudbare code. StudLife bestaat uit 18 klassen, waarvan de klassen die aan de XML zijn gekoppeld middels de onCreate() methode de views van ons MVC principe beschrijven. Verder vormen de listeners -en selectedsmethoden de controllers. De overigen klassen vormen het model, waarbij de ApplicationClass een uitzondering is. Deze wordt namelijk getiliseerd om slechts een enkel student object aan te maken, in plaats van allemaal verschillende. Dit in tegenstelling tot het gebruik van een implementatie van serializable, om hem vervolgens via putExtra() aan de intent mee te geven. De klasse Student is het centrale punt binnen het model en de applicatie. Hier worden alle gegevens verzameld, gecontroleerd en ook vanuit opgeslagen. Verder bestaat er nog een klasse Time, die op zijn eigen thread draait. Deze berekent constant aan de hand van de systeemtijd alle gegevens die afhankelijk zijn van de tijd, zo nodig worden deze doorgegeven aan de student. De keuze voor een aparte thread is hier gemaakt omdat onze implementatie van tijd en het gebruik continu is. Als dit niet op een aparte thread zou draaien zou de applicatie dusdanig langzaam worden dat deze niet meer bruikbaar is.
Verder is er een Highscore klasse. In deze klasse worden de highscores opgeslagen of uitgelezen. Deze klasse staat op zichzelf, omdat ze niet in de HighscoresActivity hoort. HighscoresActivity is een view en, volgens het MVC principe, hoort deze zich niet bezig te houden met andere dingen dan het beeld weergeven. De methodes binnen de Highscore klasse behoren ook niet tot de klasse Student, want highscores zijn geen onderdeel van een student, maar onafhankelijk hiervan. Elke Student heeft geen eigen highscores, maar de highscores hebben verschillende studenten.
De View spreekt voor zichzelf want deze klassen, die Activity implementeren, zorgen allemaal voor de communicatie met de XML en verzorgen dus de actief de weergave op het scherm samen met de XML. Al deze klassen bevatten ook de klasse Student om de, soms nodige acties, binnen het model uit te voeren. De controllers zijn in deze klassen toegewezen aan de Buttons en ImageViews.
Tenslotte zijn er nog de controller klassen. Deze zorgen ervoor dat de Views worden gepdate en/of gegevens in het model worden aangepast. We hebben twee OnItemSelectedListeners. n om van activity te veranderen en n voor het slapen of studeren. Verder hebben we nog een OnTouchListener, DragShadowBuilder en een DropListener voor de drag & drop functionaliteit.

3.2 Detail ontwerp

3.2.1 Model

Het model bestaat uit 3 klassen: Time, Highscores en de belangrijkst Student. Het model staat los van de Controllers en de Views, behalve op n punt. Dit is wanneer de methode setProgressBarsAndTextViewsValues wordt aangeroepen vanuit het model. Dit komt doordat er gekozen is om de thread van Time, die later besproken wordt, op te starten wanneer de applicatie geopend wordt. En niet af te sluiten wanneer de activity wordt afgesloten om deze vervolgens weer op te starten wanneer een andere activity wordt geopend. De thread beindigen en weer opstarten zou onnodig geheugengebruik met zich mee brengen. Door deze oplossing konden de Progressbars en TextFields niet meegegeven worden aan de klasse Time, want de gebruiker is in staat zich naar een andere omgeving te begeven(Thuis, College of Uitgaan)waar reeds andere Progressbars en TextFields aanwezig zijn. Zodoende is er een update methode in elke View aanwezig die gepdate kan worden wanneer er iets in het model veranderd. Deze wordt binnen het model aangeroepen in

een methode waar Time bij kan omdat Student wel meegegeven wordt bij het aanmaken van een Time object. Dit is in lijn met het MVC principe. Er is niet voor een Observable gekozen, omdat niet alle View klassen gepdate kunnen worden, omdat er slechts een enkele telkens open staat.

Student De klasse Student bevat methodes om de gegevens van de student, zoals health, happiness en energy op te slaan, op te halen en te wissen. Deze methodes zijn nodig om met het spel door te gaan nadat de applicatie is afgesloten. De methode om te wissen is nodig wanneer de student dood is, want dan moeten de gegevens weer teruggezet worden naar de beginwaarden(default).
Ook bevat de klasse een methode om te checken of de student dood is. Als dit het geval is dan wordt vanuit deze methode de huidige Activity afgesloten en die voor GameOver opgestart. Deze methode is essentieel, aangezien het doel van het spel het in leven houden van de student is. Hier moet dan natuurlijk wel op gecheckt worden. Verder bevat de klasse een methode die er voor zorgt dat de waarden van health, happiness en energy niet boven de 100 uit komen, want de progressbars lopen maar tot en met 100. Als deze 3 gegevens boven de 100 zouden komen geven de progressbars niet meer dan 100 aan, maar staan de waardes in het model wel boven de 100. Als vervolgens een waarde wordt afgewaardeerd van bijvoorbeeld 110 naar 106, dan valt er niks te zien op de progressbar.
Student heeft een methode die de progressbars bij het consumeren van eten of drinken aanpast, genaamd EatOrDrink. Er is tevens een vergelijkbare methode, maar dan voor events. Health wordt bijvoorbeeld berekend door 100 - healthNotFromTime - healthFromTime. De berekeningen voor het eten of drinken en de events worden dan met het NotFromTime gedaan. Het gedeelte FromTime wordt alleen aangepast door de klasse Time. De berekening op deze manier was noodzakelijk vanwege de manier waarop waardes uit tijd worden bepaald.
De laatste drie methodes zijn kleiner maar hebben ook een belangrijke taak. Ze dienen namelijk om de highscore te berekenen, te checken of er nog wel genoeg geld is voor je aankoop en om de thread aan te maken waar de waardeveranderingen per seconde op loopt.

Time Time heeft een methode updateTime die elke seconde wordt aangeroepen op een eigen thread. Deze methode haalt de systeemtijd op in milliseconde, wat hij als attribuut van deze klasse neerzet. Vervolgens roept hij de volgende methode binnen Time aan, makeProgressbarsValues. Wanneer de applicatie voor het eerst wordt opgestart wordt de systeemtijd van het tot-leven-komen opgeslagen. Met behulp van dit gegeven wordt berekend hoeveel milliseconde de student al in leven is. Door dit te delen door het aantal milliseconden dat 1 health erover moet doen om in mindering gebracht te worden krijg je het totale aantal health dat in mindering gebracht moet worden door de tijd. Zo gaat dit ook in zijn werk voor de andere waardes.
Tenslotte is er een check aanwezig die reguleert of er wel veranderingen in Student moeten worden doorgevoerd, want als er elke seconde de waardes gezet gaan worden, zou een aparte thread overbodig geweest zijn. Hiervoor is er een methode om te kijken of de nieuw berekende waarde anders is dan de vorige. Zo ja, dan pas worden de waardes in Student gewijzigd.

Highscores De klasse Highscores heeft eigenlijk maar n echt belangrijke methode en dat is addHighscore. Deze methode is nodig voor het toevoegen van een highscore wanneer de student is overleden. Als de student is overleden wordt er door deze methode gekeken of de score tot de top tien behoord. Als dit het geval is schuift hij alles wat onder de nieuwe highscore hoort te liggen een plekje naar onder, beginnend bij de een na onderste. Hierdoor komt er een plek vrij.
De constructor van de klasse Highscores zorgt er voor dat de highscores uit een eigen SharedPreferences gehaald worden. Zodat deze ook na het afsluiten van de applicatie bewaard blijven.

3.2.2 View

College-, Thuis-, UitgaanActivity Deze drie klassen lijken op elkaar en implementeren ook dezelfde klasse. Het enige verschillen zijn ander dat CollegeActivity een spinner bevat om te studeren, ThuisActivity een spinner bevat om te slapen, UitgaanActivity een ImageView met onClick heeft om events te kunnen laten plaatsvinden en dat elk van deze activity's ander eten en drinken bevat.
Wat de klassen wel gemeen hebben zijn de methoden uit de klasse die ze implementeren. De eerste is setProgressBarsAndTextViewsValues. Doormiddel van deze methode aan te roepen worden de progressbars en de textfield voor geld gepdate vanuit Student. De reden waarom dit hier gebeurd is al eerder behandeld in het stuk over Model. Ook zijn in deze klassen de onStop overrided, want bij het afsluiten van de applicatie moeten namelijk eerst de waardes uit Student worden opgeslagen.
Tenslotte hebben deze klassen een dropSwitch. Deze bepaald aan de hand van welke ImageView er is gedropt op zijn eindbestemming met welke waarden de eerder genoemde methode eatOrDrink moet worden uitgevoerd. Dit hebben we gedaan met een switch op id van de ImageView. Deze staan in de View klassen, omdat ze per klasse verschillend zijn door de verschillende etensware.

Event Event is een klasse met maar n essentile methode en dat is eventSwitch. Deze methode switcht aan de hand van een random nummer. Per case wordt er een bijpassende achtergrond gezet en een daar bij horende toast. Ook wordt er op de achtergrond doEvent uitgevoerd, wat een methode uit Student is. Hier worden per geval verschillende waarden aan mee gegeven, bijbehoren bij het event plus plaatje. Hierdoor worden de eerder genoemde NotFromTime gedeeltes in Student weer aangepast.

Game Over GameOver zet de achtergrond van het scherm een bijpassende tekst, afhankelijk van de behaalde studie- en socialepunten. Dit wordt gedaan in de methode setBackground. Als de punten 10 van elkaar afzitten komt er een bericht die behoord tot balans. Als beide punten onder de 20 zijn komt er een tekst waarin gezegd wordt dat er niet veel bereikt is in het leven de student. Verder is er nog een bericht voor als de sociale punten hoog zijn en ver van de studiepunten. Ook is er een tekst voor de omgekeerde variant. Dit voor een totaal van 3 mogelijke uitkomsten.
De button in deze klasse zorgt ervoor dat de naam uit de tekstfield wordt gehaald, de highscore wordt berekend doormiddel van de methode in Student en deze toevoegt aan de highscore via de eerde genoemde addHighscore in de Highscore klasse. Uiteindelijk wordt er voor gezorgd dat de gebruiker weer in het menu terecht komt.

3.2.3 Controller

CustomOnItemSelectedListener Deze itemselected zorgt ervoor dat de gebruiker naar de activity switcht die gekozen is. Dit wordt gedaan doormiddel van naar de positie die aangeklikt is te kijken. Zodoende wordt de nieuwe activity opgestart en de oude afgesloten.

CustomOnItemSelectedListenerStudySleep CustomOnItemSelectedListenerStudySleep is de tweede itemselected en wordt gebruikt voor het slapen en studeren. De spinner waar deze controller op aangeroepen wordt geeft door middel van een string de actie mee. Hierop wordt in de constructor van CustomOnItemSelectedListenerStudySleep op gecheckt en vervolgens de

bijbehorende methode uitgevoerd: de methode sleep of study. Hierna worden de spinner gedisabled met de methode disableSpinners. Hierin wordt ook de update aangeroepen van de student, omdat er in de methode study of sleep de waardes in Student aangepast worden. Als de spinner op 0 minuten staat dan wordt er gecheckt of de student nog hoort te slapen of studeren. Zo ja, dan worden alle spinners alsnog gedisabled. Zo wordt voorkomen dat de student opeens weer wakker is wanneer de applicatie wordt afgesloten en weer gestart wordt. De methode doNothing checkt of de tijd op dit tijdstip groter is dan de tijd dat de student weer wakker mag worden of mag stoppen met studeren. Als dit het geval is dan kunnen de spinners weer gebruikt worden.

Drag and drop De onTouch klasse is onderdeel van de drag en drop functionaliteit. Als de student niet slaapt, omdat er natuurlijk niet gegeten in de slaap mag worden, dan wordt er een DragShadow aangemaakt, wordt de clipdata gezet en vervolgens wordt de drag gestart.
De DragShadow klasse maakt een schaduw aan die onder je vinger blijft zitten totdat het scherm losgelaten wordt. Deze wordt aangemaakt wanneer de constructor van deze klasse wordt aangeroepen, wat in onTouch dus gebeurd.
De DropListener klasse bevat n methode met een switch. Deze switch bepaald aan de hand van welke actie is gebeurd, wat er moet gebeuren. Zo moet de student stoppen met bewegen en zijn mond open doen als er voedsel of drinken boven zijn mond gehouden wordt. Als het voedsel of drinken weer van zijn mond af gehaald wordt, moet de student weer naar zijn oude situatie toe. Dit doen we doormiddel van de animatie weer als background te kiezen en deze weer te starten. Als het voedsel of drinken wel op de mond gedropt wordt, dan gaat de student kauwen door als background een animatie van het kauwen te zetten en deze te starten. Ook wordt de dropSwitch uit de View klasse waar de student zich momenteel in bevind aangeroepen. Hierin worden zoals eerder genoemd de waardes mee veranderd. Nadat er iets in de mond gedropt moet er drie seconde later de originele animatie weer als achtergrond gezet worden en daarna gestart. Dit moet na drie seconden, omdat de animatie van het kauwen drie seconde duurt en deze moet eerst afgelopen zijn. Dit is gedaan met een postDelay op een handler die gedefinieerd staat in de View waar de gebruiker zich momenteel in bevind. Deze is meegegeven aan de drag en drop klassen.

3.3 Verantwoording

Ons ontwerp is een goed ontwerp, omdat we hebben gedacht in oplossingen in plaats van problemen. Ook hebben we op cruciale punten in ons ontwerp goed nagedacht over de verschillende mogelijkheden die we tot onze beschikking hadden. Hierbij volgen twee voorbeelden hiervan.

3.3.1 Keuze voor de tijdsrepresentatie

De klasse Time bestaat momenteel, zoals eerder genoemd, uit een methode updateTime die telkens wordt aangeroepen. Hierin wordt de systeemtijd in milliseconden opgehaald. We hadden er ook voor kunnen kiezen om de tijd op te halen als string doormiddel van een SimpleDateFormat. Hier hebben we niet voor gekozen, omdat je de string dan eerst zou moeten omzetten naar allemaal aparte integers. Vervolgens zou je heel omslachtig moeten gaan rekenen met de tijd als je bijvoorbeeld 1 gezondheid er af wilt halen om de 16 minuten. Nu we de tijd in n eenheid hadden en al gerepresentateerd als getallen konden we al gelijk delen door het aantal milliseconden gelijk aan 16 minuten. Echter had je dan een veel te groot getal, want je student leeft immers niet vanaf het begin van de systeemtijd. Vanwege dit fenomeen wordt er bijgehouden wanneer de student 'tot leven is gekomen'. Door de tijd waarop je student tot leven is gekomen van de huidige tijd af te halen, en dit door de tijd te delen hoelang bijvoorbeeld gezondheid er over doet

om eentje minder te worden, verkrijg je het getal wat er tot nu toe van gezondheid af moet zijn. Echter is tijd niet de enige die de progressbars kan aanpassen, maar ook de events. Ook het slapen bezorgt er juist weer gezondheid bij. Als we dit zouden verrekenen met het getal wat er via onze tijd uit zou komen, dan zou de gezondheid weer terug springen naar het getal dat alleen van de tijd komt. Dit komt door de manier van berekenen van onze waardes die afhankelijk zijn van tijd. Om deze reden hebben is de keus gemaakt om een extra variabele aan te maken. Hierdoor is er een variabele die het getal bevat wat er van het maximale door de tijd af moet worden gehaald en een getal dat er af of bij op moet worden geteld wat van alle andere dingen behalve tijd af hangt. Dit is gedaan voor alle waarden die afhankelijk zijn van de tijd.
We hadden ook voor een andere manier van berekenen kunnen gebruiken, namelijk door telkens een tijd neer te zetten waarop de waardes weer moesten worden gepdate. Toch leek ons de oplossing hoe we het nu hebben aangepakt het beste, omdat wanneer je de applicatie afsluit en een tijd later weer opnieuw opstart, je de waardes die er in de tussentijd af zouden moeten zijn gegaan helemaal terug moet berekenen. Op de manier hoe we het nu hebben gedaan is het maar n berekening voor elke waarde, terwijl je op de manier van terug rekenen telkens stapjes zou moeten nemen.

3.3.2 Gebruik van SharedPreferences boven andere opslagmethodes

Ook de keuze voor SharedPreferences boven alle andere opslagmethodes was een hele bewuste keuze. SharedPreferences slaat namelijk alle gegevens die het meekrijgt op in een xml bestand en voegt deze aan de applicatie toe. Dit kan echter maar met een beperkt aantal basis datatypen. We hadden ook kunnen kiezen voor een SQL database of voor interne opslag op de mobiele telefoon. De data die wij bij houden zijn alleen maar strings, integers, floats en booleans, dus was de keus al snel gemaakt. Een SQL database zou veel te veel werk voor niets zijn geweest, aangezien we maar een beperkt aantal waarden op hoeven te slaan. Ook het opslaan in het interne geheugen van de telefoon zou onnodig zijn, aangezien we niet zoveel op te slaan hebben en voor het interne geheugen van de telefoon permissies moeten worden aangevraagd. Dit hoeft daarentegen niet voor SharedPreferences.

4 Reflectie

4.1 Positieve punten

Om maar direct met de deur in huis te vallen: wij zijn erg tevreden over het resultaat wat we neergezet hebben. Dit is pas de tweede keer dat we een applicatie bouwen en dan meteen dit al neer kunnen zetten, daar zijn wij stiekem wel een beetje trots op. We hebben veel nieuwe methoden en opties ontdekt binnen Android Studio en hebben leren omgaan met de koppeling tussen java en XML. De layout van onze applicatie is er uiteindelijk beter uit komen te zien dan oorspronkelijk gedacht, hier zijn wij over te spreken. De applicatie lijkt goed te werken en bied ruimte voor toekomstige uitbreidingen. Natuurlijk zullen er op ten duur kleine foutjes in de applicatie gevonden worden, maar om die er helemaal uit te houden is een ontwikkel plus test periode van vier weken te kort. Tenslotte vonden we het een prettige samenwerking tussen de actieve groepsleden. Jammer genoeg waren dit slechts drie van de vier leden, waar we later bij de negatieve punten op terug komen.

4.2 Negatieve punten

Wij betreuren het dat de samenwerking slechts met 3 van de 4 deelnemers goed verliep. Tevens hadden we de spinners in de applicatie iets overzichtelijker en op een nettere manier willen weergeven ten opzichte van de achtergrond. Dit is ons helaas niet gelukt. Telkens als we een soort van oplossing hadden gevonden, was deze totaal niet uit karakter met de stijl en paste dus niet bij de layout van de applicatie. Ondanks dat het grafische gedeelte beter was dan we vooraf hadden verwacht, vallen er ook nog wel op grafisch gebied verbeteringen te behalen. We vinden het jammer dat we er niet meer features in hebben kunnen stoppen, maar dit was gezien de tijd en de lagere bezetting niet te realiseren.

4.3 Conclusie

Als wij ooit nog een keer een project als deze zouden moeten doen, dan zouden wij het weer op dezelfde manier aanpakken. Wij hadden een fijne en professionele samenwerking tussen de studenten die actief deelnamen. De applicatie is voltooid binnen de beoogde tijd, echter zouden wij graag nog extra mogelijkheden toegevoegd hebben. Dit is ons, mede door uitval van het 4e groepslid, helaas niet gelukt. In de toekomst zullen wij ogenschijnlijk terugkijken op dit project en ons afvragen waarom we sommigen dingen niet hebben gedaan. Dit is echter altijd achteraf, op een moment waarop we ongetwijfeld meer kennis bezitten. Al met al zijn wij tevreden met het product wat wij hebben neergezet maar ook dankbaar voor de werkvorm waarin dit heeft mogen plaatsvinden. Wij delen allen de mening dat dit een aangename afwisseling was op de 'normale' werkvormen. Bovendien heeft dit vak zich ook ingezet om een aantal soft-skills aan te scherpen, wat zij zeker niet onaardig vinden.