

www.Info Support.com



Ontwerp en analyse document

Afstudeeropdracht: Paaspop de ultieme festival App

Merik Westerveld - 2737450

13 mei 2019

Concept

**Hoofdkantoor**

Kruisboog 42

3905 TG Veenendaal

Nederland

Tel. +31(0)318 - 55 20 20

Fax +31(0)318 - 55 23 55

**Kenniscentrum**

De Smalle Zijde 39

3903 LM Veenendaal

Tel. +31(0)318 - 50 11 19

Fax +31(0)318 - 51 83 59

info.nl@Info Support.com

www.Info Support.com

K.v.K 3013 5370

BTW NL8062.30.277B01

IBAN NL92 RABO 0305 9528 89

BIC RABONL2U

IBAN NL74 INGB 0004 7385 93

BIC INGBNL2A

Ontwerp en analyse document

Afstudeeropdracht: Paaspop de ultieme festival App

|  |  |
| --- | --- |
| Gegevens |  |
|  | Ontwerp en analyse document |
|  | Afstudeeropdracht: Paaspop de ultieme festival App |
|  | 1.3 |
|  | Concept |
|  | 13 mei 2019 |
|  | Ontwerp en analyste document – Merik Westerveld – Info Support |
|  | Info Support |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 1.0 | Concept | 15 Februari 2019 | Merik Westerveld | Opzet document |
| 1.1 | Concept | 22 Februari 2019 | Merik Westerveld | Feedback verwerkt op de functionele en niet functionele eisen en het ontwerp |
| 1.2 | Concept | 23 April 2019 | Merik Westerveld | Feedback van Gert Jan Timmerman verwerkt |
| 1.3 | Concept | 13 Mei 2019 | Merik Westerveld | Low level architectuur schetsen |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1.0 | Concept | 18 Februari 2019 | Hans Geurtsen |
| 1.1 | Concept | 10 April 2019 | Gert Jan Timmerman |

**© Info Support B.V., Veenendaal 2019**

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotokopie, microfilm of op welke andere wijze ook, zonder voorafgaande toestemming van **Info Support B.V.**

No part of this publication may be reproduced in any form by print, photo print, microfilm or any other means without written permission by **Info Support B.V.**

Prijsopgaven en leveringen geschieden volgens de Algemene Voorwaarden van **Info Support B.V.** gedeponeerd bij de K.v.K. te Utrecht onder nr. 30135370. Een exemplaar zenden wij u op uw verzoek per omgaande kosteloos toe.

# **Inleiding en leeswijzer**

Voor de opdracht “Paaspop special: De Ultieme Festival App” word een Proof Of Concept (PoC) ontwikkeld waarbij gebruikt wordt van een .NET API en een React-Native front-end om een cross platform mobiele applicatie te ontwikkelen. De app gaat gebruikt worden door festival bezoekers van het festival Paaspop.

In hoofdstuk 2 staan de eisen van de app beschreven. Deze functionele eisen vloeien voort uit de analyse die is gedaan onder potentiële festivalbezoekers door middel van een enquête. De niet functionele eisen zijn samen met de opdrachtgever besproken en uitgewerkt. Er is hier gebruik gemaakt van de MoSCoW prioriteringsmethode om duidelijk inzichtelijk te krijgen welke eisen het belangrijkst zijn en welke minder belangrijk zijn in de context van het PoC.

In hoofdstuk 3 zijn de user stories gedefinieerd welke een rol spelen in het gebruik van Kanban om de taken van de ontwikkelaar inzichtelijk te krijgen.

Hoofdstuk 4 laat het ontwerp zien dat gemaakt is voordat de app ontwikkeld is. Dit ontwerp geldt dan ook als basis voor het ontwikkelen van de app en is goedgekeurd door de opdrachtgever.

Hoofdstuk 5 bevat een functioneel en technisch erd (Entity Relationship Model) wat laat zien hoe de API in elkaar zit en hoe de gebruikte klassen relaties hebben met elkaar.

In hoofdstuk 6 staat beschreven hoe de overall architectuur van de API in elkaar zit en welke patronen er gebruikt staan. Ook kan er een korte toelichting van de gemaakte keuze gevonden worden betreffende de gebruikte architectuur en patronen.

Hoofdstuk 7 en 8 laten elk voor één functionaliteit van zowel de back-end als de front-end zien hoe dit geïmplementeerd is.

# **Inhoudsopgave**

[1. Inleiding en leeswijzer 3](#_Toc8637370)

[Inhoudsopgave 4](#_Toc8637371)

[2. Eisen 5](#_Toc8637372)

[2.1 Functionele eisen 5](#_Toc8637373)

[2.1.1 Mobiele applicatie 5](#_Toc8637374)

[2.1.2 Dashboard 6](#_Toc8637375)

[2.2 Niet-functionele eisen 6](#_Toc8637376)

[3. User stories 7](#_Toc8637377)

[4. Mockup mobiele app 9](#_Toc8637378)

[5. ERD (Entity-relationshipmodel) 11](#_Toc8637379)

[5.1 Functioneel ERD 11](#_Toc8637380)

[5.2 Technisch ERD 12](#_Toc8637381)

[6. Architectuur backend 13](#_Toc8637382)

[6.1 Hoog level architectuur schets 13](#_Toc8637383)

[6.2 Low level architectuur schets 14](#_Toc8637384)

[6.3 Clean architecture 15](#_Toc8637385)

[6.4 CQS 16](#_Toc8637386)

[6.5 Third party libraries 16](#_Toc8637387)

[6.5.1 Mediatr 16](#_Toc8637388)

[6.5.2 AutoMapper 17](#_Toc8637389)

[6.5.3 FluentValidation 17](#_Toc8637390)

[6.6 Backend code voorbeeld van een API call 19](#_Toc8637391)

[7. Front-end structuur 20](#_Toc8637392)

[7.1 Opzet 20](#_Toc8637393)

[7.2 Low level architectuur schets 21](#_Toc8637394)

[7.3 Gebruik van Redux 22](#_Toc8637395)

[8. Bibliografie 29](#_Toc8637396)

# Eisen

## Functionele eisen

### Mobiele applicatie

De mobiele applicatie is de technologische toepassing die ervoor moet zorgen dat festivalbezoekers door middel van een app op hun telefoon een betere ervaring krijgen op het festival.

|  |  |  |
| --- | --- | --- |
| ID | Omschrijving | MoSCoW |
| FEM1. | De gebruiker moet bij de eerste keer gebruik van de app aan kunnen geven of de festivalbezoeker een man / vrouw / gender neutraal is en de leeftijd om zo gebruik te kunnen maken van de app en ervoor te zorgen dat in de volgende edities de juiste artiesten worden gekozen voor de juiste doelgroepen. | M |
| FEM2. | De gebruiker moet in kunnen zien welke artiest waar en wanneer speelt. | M |
| FEM3. | De gebruiker moet kunnen zien welke bar en wc het meest geschikt is om naar toe te gaan wat wordt gebaseerd op afstand en drukte. | M |
| FEM4. | De gebruiker moet een plattegrond van het festival kunnen zien. | M |
| FEM5. | De gebruiker moet aan kunnen geven naar welke artiest hij / zij wilt. | M |
| FEM6. | De gebruiker moet de weersverwachting kunnen zien van het festival. | S |
| FEM7. | De gebruiker moet om een bepaalde tijd een melding krijgen dat die water moet drinken. | S |
| FEM8. | De gebruiker moet een meeting punt kunnen laten genereren voor de beste plek om zijn / haar vrienden te ontmoeten. | S |
| FEM9. | De gebruiker moet zijn / haar gegevens kunnen verwijderen, zowel lokaal als wat in de cloud staat opgeslagen. | S |
| FEM10. | De gebruiker moet het interesse level van een artiest in kunnen zien. | S |
| FEM11. | De gebruiker moet een push melding krijgen 10 minuten voor aanvang van een concert. | S |
| FEM12. | De gebruiker moet op basis van zijn / haar interesse aanraders krijgen. | C |
| FEM13. | De gebruiker moet zijn budget aan kunnen geven en bijhouden hoeveel muntjes / geld hij of zij nog over heeft. | C |
| FEM14. | De gebruiker kan een paklijst aanmaken en aanvinken wat hij / zij heeft. | W |
| FEM15. | De gebruiker kan zijn / haar tent op de camping neerzetten en deze locatie in de app opslaan zodat hij / zij de tent op de camping altijd terug kan vinden. | W |
| FEM16. | De gebruiker moet de app kunnen laten kiezen naar welke artiest hij / zij gaat. | W |

### Dashboard

Het dashboard is een webapplicatie die gebruikt kan worden op een desktop pc of laptop waar de beheerders van het festival de input van de app kunnen beheren.

|  |  |  |
| --- | --- | --- |
| ID | Omschrijving | MoSCoW |
| FED1. | De gebruiker moet aan kunnen geven welke artiest waar en wanneer speelt. | C |
| FED2. | De gebruiker moet in kunnen zien hoe populair elke artiest is en bij welke doelgroep. | C |
| FED3. | De gebruiker moet een push melding kunnen sturen naar alle gebruikers van de mobiele applicatie. | C |

## Niet-functionele eisen

Deze niet-functionele eisen zijn gebaseerd op ISO 9126.

|  |  |  |
| --- | --- | --- |
| ID | Omschrijving | MoSCoW |
| NFEM1. | De app moet zonder internet ook beschikbaar zijn (niet alle functionaliteiten) [[1]](#footnote-1). | M |
| NFEM2. | Ongeautoriseerde personen mogen niet in het dashboard komen. | M |
| NFEM3. | De app en het opslaan van data moet voldoen aan de privacy wetgeving. | M |
| NFEM4. | Voor beide de app en het dashboard (back-end) moet de software goed gedocumenteerd zijn door middel van API documentatie zoals swagger zodat op deze manier de kennis kan worden overdragen. | M |
| NFEM5. | Alle software moet op een juiste manier getest worden (unit testen, integratie testen in de unit testen en statische code kwaliteit testen door middel van SonarQube). | M |
| NFEM6. | De app moet voldoen aan de huisstijl van Paaspop. | M |
| NFEM7. | De API moet onderhoudbaar, schaalbaar en componenten moeten uitwisselbaar zijn. | M |
| NFEM8. | De mobiele app moet beschikbaar zijn op zowel IOS als Android. | M |
| NFEM9. | Beide de app en het dashboard mogen geen laadtijden hebben die langer dan 3 seconden duren. | S |
| NFEM10. | De app moet zonder locatietoegang ook werken (niet alle functionaliteiten) [[2]](#footnote-2). | S |
| NFEM11. | De app moet op basis van UX testen geschikt bevonden zijn. | C |
| NFEM12. | De app moet op beide platformen native uitzien (design). | C |

# User stories

**FEM1.:** Als gebruiker wil ik bij het eerste keer openen van de app mijn gegevens achter kunnen laten zodat ik gebruik kan maken van de applicatie.

**FEM2.:** Als gebruiker wil ik kunnen inzien welke artiest waar en op welk podium speelt zodat ik altijd weet waar ik wanneer naar toe moet om de artiesten te zien die ik wil.

**FEM3.:** Als gebruiker wil ik kunnen inzien welke bar en wc het meest geschikt is om naar toe te gaan op basis van de afstand en de drukte zodat ik zo min mogelijk tijd hoef te besteden daaraan en meer tijd van het festival kan genieten.

**FEM4.:** Als gebruiker wil ik een plattegrond van het festivalterrein kunnen zien zodat ik precies weet waar wat staat.

**FEM5.:** Als gebruiker wil ik aan kunnen geven naar welke artiest ik wil zodat ik een lijst van mijn favoriete bij kan houden en niks vergeet.

**FEM6.:** Als gebruiker wil ik een melding ontvangen van het weer zodat ik weet wanneer het gaat regenen of de zon sterk gaat schijnen en ik me daarop kan voorbereiden.

**FEM7.:** Als gebruiker wil ik een melding ontvangen die mij eraan herinnert om genoeg water te drinken zodat ik genoeg water blijf drinken door de andere consumpties heen waardoor ik geen uitdroog verschijnselen zoals hoofdpijn krijg.

**FEM8.:** Als gebruiker wil ik een meeting punt kunnen laten genereren voor de beste plek om mijn vrienden weer te ontmoeten zodat ik op een rustige plek snel mijn vrienden gevonden krijg.

**FEM9.:** Als gebruiker wil ik mijn gegevens kunnen verwijderen zodat mijn gegevens niet meer op een server opgeslagen staan als ik dat niet meer wil.

**FEM10.:** Als gebruiker wil ik het interesse level van een artiest in kunnen zien zodat ik weet welke artiest populair is en waar het druk wordt en niet.

**FEM11.:** Als gebruiker wil ik een push melding krijgen 10 minuten voor aanvang van een artiest die ik een favoriet heb gegeven zodat ik een show van mijn favoriete artiest niet mis.

**FEM12.:** Als gebruiker wil ik op basis van de artiesten die ik een favoriet heb gegeven aanraders krijgen zodat als ik niks te doen heb op het festival ik nieuwe artiesten op deze manier makkelijk kan ontdekken.

**FEM13.:** Als gebruiker wil ik mijn budget bij kunnen houden en aangeven hoeveel muntjes / geld ik nog over heb zodat ik niet teveel uit geef op het festival.

**FEM14.:** Als gebruiker wil ik een paklijst aan kunnen maken en aan kunnen vinken wat ik wel en niet heb gepakt zodat ik niks vergeet mee te nemen naar het festival toe.

**FEM15.:** Als gebruiker wil ik op een kaart aan kunnen geven waar ik mijn tent heb geplaatst zodat ik deze altijd terug kan vinden.

**FEM16.:** Als gebruiker wil ik de app kunnen laten kiezen naar welke artiest ik moet gaan als er overlap is tussen artiesten die ik leuk vindt zodat ik niet zelf een keuze hoef te maken.

**FED1.:** Als beheerder wil ik aan kunnen geven welke artiest waar en wanneer speelt zodat dit niet voorgeprogrammeerd in de app staat en ik het altijd kan aanpassen.

**FED2.:** Als beheerder wil ik in kunnen zien welke artiest populair is onder welke doelgroep zodat ik in de toekomst weet welke artiest wel of niet geboekt moet worden en een juiste dagindeling kan maken voor de juiste doelgroep.

**FED3.:** Als beheerder wil ik een push melding kunnen sturen naar alle gebruikers van de mobiele applicatie zodat ik deze mensen kan bereiken met een bericht zonder ik daar contactgegevens voor nodig heb.

# Mockup mobiele app

1: Scherm één is het ‘splash screen’, ook wel het eerste scherm wat de gebruiker te zien krijgt waar ondertussen de app laadt. Hier wordt het Paaspop logo op afgebeeld zodat de gebruiker gelijk kennis maakt met de huisstijl die verder in de app gebruikt wordt.

2: Dit tweede scherm komt de gebruiker van de app maar één keer in zijn gebruik tegen, tenzij de gebruiker er voor kiest om alle data te verwijderen (mogelijk in scherm 6) en de app daarna weer opnieuw te gebruiken. Op dit scherm kies de gebruiker de sekse waar de persoon zich mee identificeert. Ook bepalen ze de leeftijd. Na het invullen van deze data kan verder gegaan worden met het gebruiken van de app.

3: Het derde scherm is het scherm waar de gebruiker op terecht komt wanneer de gebruiker scherm twee al minimaal één keer doorlopen heeft. Op dit scherm kan de gebruiker de plattegrond zien en zijn / haar locatie. Daarnaast kan er op de knop ‘Meeting punt’ gedrukt worden waardoor de app een meeting punt kan genereren welke de gebruiker naar zijn / haar vrienden kan sturen.

4: Op scherm vier kan de gebruiker per stage zien welke artiest er wanneer staat. Hier kan die een artiest een ‘hartje’ geven waardoor die in de lijst komt te staan van artiesten die de gebruiker wilt zien op het festival. Ook kan er op een artiest geklikt worden waarna de gebruiker op scherm 7 komt.

5: Scherm 5 wordt gebruikt als ‘plekken’ scherm waar de gebruiker kan zien wat de beste wc, bar of eettent is op basis van de afstand van de gebruiker en de drukte van de locatie. Als de gebruiker op een plek klikt gaat deze door naar scherm 8.

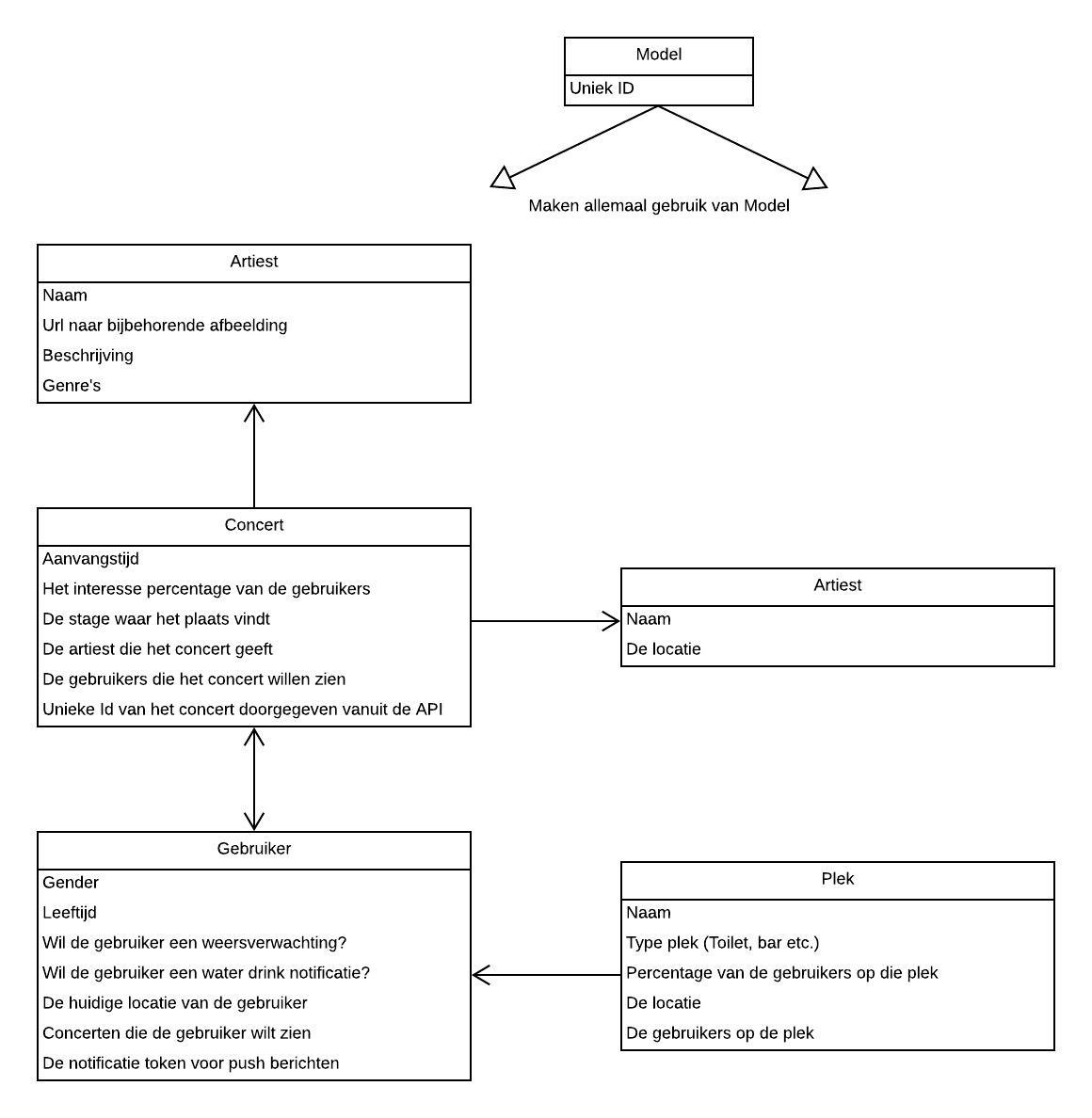
6: Het laatste scherm in de tab navigator is het ‘zelf’ scherm. Hier kan de gebruiker kiezen of hij / zij de weersverwachting of water drink melding wilt krijgen. Ook kan de gebruiker hier ervoor kiezen om zijn account te verwijderen en kan hij / zij zijn / haar rooster van favoriete artiesten inzien.

7: Via scherm 4 kan de gebruiker naar scherm 7 gaan waar hij / zij meer informatie krijgt te zien over de artiest en staat er een indicator hoeveel interesse er is voor deze artiest.

8: Via scherm 5 komt de gebruiker bij scherm 8 waar het locatie punt getoond wordt van de plek die is aangeklikt en de locatie van de gebruiker zodat deze er makkelijk naartoe kan.

# ERD (Entity-relationshipmodel)

## Functioneel ERD

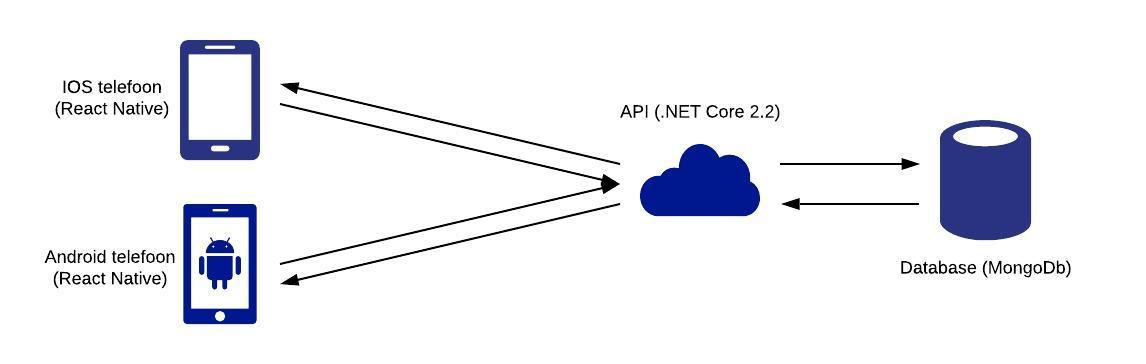


## Technisch ERD

# Architectuur backend

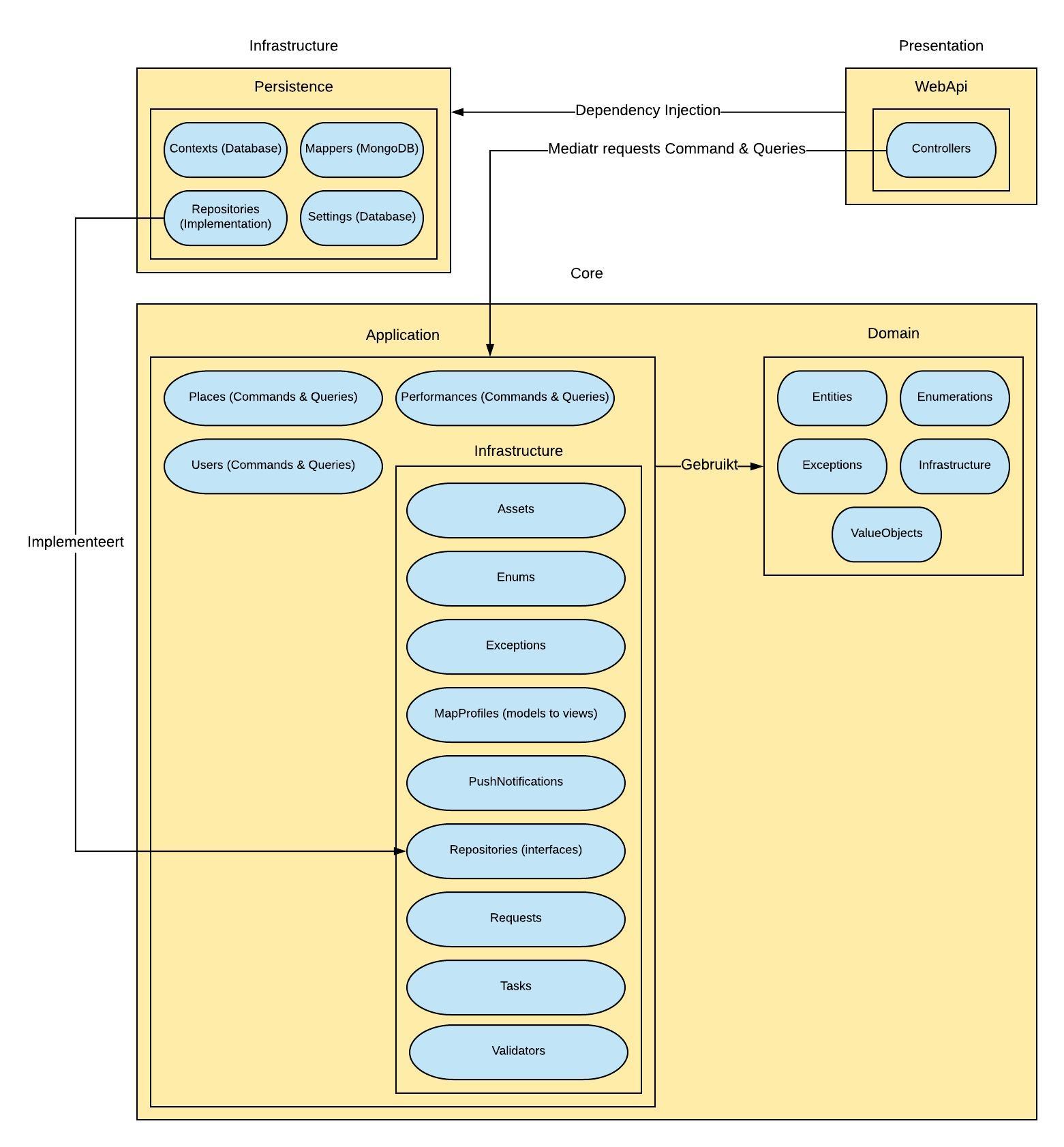
## Hoog level architectuur schets

Op een hoog niveau is er een architectuur schets gemaakt welke hier onder te zien is:



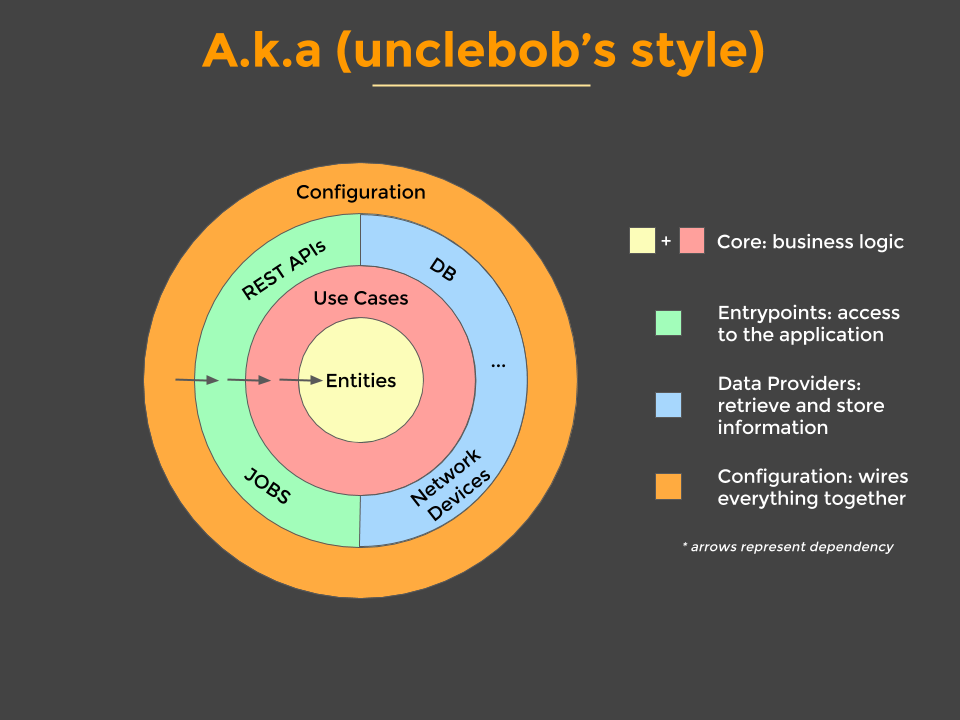
Via een client, in dit geval een IOS of Android telefoon, kan de gebruiker de react native paaspop app opstarten. Deze app staat in verbinding met de .NET core 2.2 API welke alle requests doorloopt en, waar kan, data verwerkt en terug / door stuurt.

## Low level architectuur schets



Zoals beschreven in hoofdstuk 6.3 over clean architectuur is de architectuur opgedeeld in verschillende componenten die, waar dit mogelijk is, losgekoppeld of veranderd kunnen worden. De presentation laag is momenteel een API waar gebruik gemaakt wordt van controllers welke door middel van het mediator patroon commands of queries uitvoeren in de application laag. Deze presentation laag kan hierdoor zonder dat andere componenten moeten veranderd verwisseld worden door bijvoorbeeld razor pages. Daarnaast bevat de infrastructure laag de persistentie door middel van een database implementatie. De werking van de repositories staat beschreven in de application laag en de implementatie in de presistence laag. Door deze constructie staat de persistence laag los van de application laag waardoor er elke willekeurige vorm van persistentie toegepast kan worden zolang de repositorie maar wordt gebruikt. De core laag is de daadwerkelijke werking van de applicatie bestaande uit het domein en de acties die de applicatie uitvoert.

## Clean architecture

De backend architectuur is gebaseerd op clean architecture, een ontwerp bedacht door Robert Cecil Martin ook wel bekend als uncle bob. Deze architectuur is gebaseerd op de volgende vijf elementen [[3]](#footnote-3):

1. Onafhankelijk van frameworks.

2. Alle business rules moeten kunnen worden getest zonder de UI, database en alle andere externe elementen.

3. Onafhankelijk van de UI.

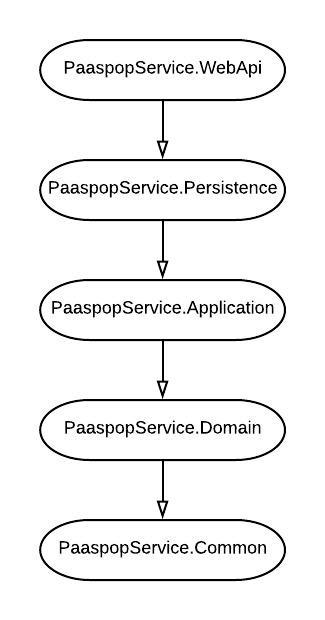
4. Onafhankelijk van de Database.

5. Onafhankelijk van enige andere externe partijen.

In de afbeelding hieronder staat de architectuur beschreven in een diagram.

Figuur: diagram van clean architecture [[4]](#footnote-4)

De code is opgedeeld in verschillende lagen welke allemaal hun eigen verantwoordelijkheid hebben. In de backend die voor Paaspop wordt gebruikt ziet het diagram er als volgt uit:



PaaspopService.WebApi: is de laag die verantwoordelijk is voor de UI / API. Di tis ook wel het entrypoint.

PaaspopService.Persistence: de laag die verbinding maakt met de desbetreffende database.

PaaspopService.Application: In deze laag worden de use cases beschreven, bijvoorbeeld een gebruiker vinden.

PaaspopService.Domain: de entiteiten laag (vormt samen

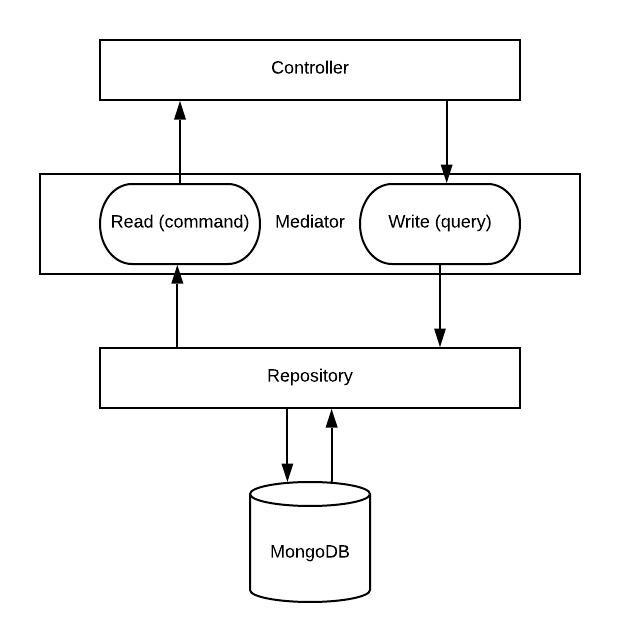
PaaspopService.Application de core van de applicatie).

PaaspopService.Common: elementen van de applicatie die ook zonder aanpassingen in andere applicaties gebruikt zou kunnen worden.

Er is voor deze architectuur gekozen omdat het onnodige koppelingen weg haalt in een applicatie waardoor de applicatie beter getest kan worden en uitwisselbaar, onderhoudbaar en schaalbaar is.

## CQS

CQS, ook wel Command and Query Segregation, is een patroon waarbij de verantwoordelijkheid tussen het schrijven (command) en het lezen (queries) van data is gescheiden.



Er is voor dit patroon gekozen om de modellen van het schrijven en lezen van elkaar te scheiden waardoor er wordt gegarandeerd dat enkel de juiste modellen de juiste acties uitvoeren. Daarnaast is het makkelijker schaalbaar (eventueel met het creëren van een aparte schrijf en aparte lees database waardoor het CQRS wordt), onderhoudbaar (duidelijke scheiding waar wat geïmplementeerd moet worden) en uitwisselbaar (mochten de lees queries een andere koppeling nodig hebben dan de schrijf commands).

## Third party libraries

### Mediatr

Mediatr is een third party library waarmee binnen de applicatie een pipeline opgezet kan worden waar een request doorheen gaat. Dit werkt goed samen met CQRS (5.2) en FluentValidation (5.3.3). Daarnaast worden de request objecten en handlers gescheiden van elkaar. Hieronder staat een beschrijving van een request:

public class GetArtistQuery : IRequest<ArtistViewModel>

{

public string Id { get; set; }

}

Vervolgens is er een handler, welke verbinding maakt met een database implementatie, die gebruik maakt van het hierboven beschreven request:

public class GetArtistsQueryHandler : GeneralRequestHandler<GetArtistQuery, ArtistViewModel>

{

private readonly IArtistsRepository \_artistsRepository;

public GetArtistsQueryHandler(IMapper mapper, IArtistsRepository artistsRepository)

: base(mapper)

{

\_artistsRepository = artistsRepository;

}

public override async Task<ArtistViewModel> Handle(GetArtistQuery request, CancellationToken cancellationToken)

{

Artist result = await \_artistsRepository.GetArtistById(request.Id, cancellationToken);

return Mapper.Map<ArtistViewModel>(result);

}

}

Deze handler wordt door de mediatr library en dependency injection via de controller uitgevoerd:

[Route("api/[controller]")]

[ApiController]

public class ArtistsController : BaseController

{

[HttpGet("{id}", Name = "Get")]

public async Task<ActionResult<ArtistViewModel>> Get(string id)

{

return Ok(await GetMediator().Send(new GetArtistQuery {Id = id}));

}

}

Al deze requests gaan door een pipeline wat elementen bevat zoals een logger. Deze logger is een IRequestPreProcessor wat een voorbeeld is van een pipeline element wat vóór een request dankzij het gebruik van mediatr wordt uitgevoerd[[5]](#footnote-5):

public class RequestLogger<TRequest> : IRequestPreProcessor<TRequest>

{

private readonly ILogger \_logger;

public RequestLogger(ILogger<TRequest> logger)

{

\_logger = logger;

}

public Task Process(TRequest request, CancellationToken cancellationToken)

{

var name = typeof(TRequest).Name;

\_logger.LogInformation("Paaspop service Request: {Name} {@Request}", name, request);

return Task.CompletedTask;

}

}

### AutoMapper

Automapper zorgt ervoor dat objecten automatisch naar elkaar gemapt kunnen worden zonder dat hier altijd aparte methodes voor geschreven hoeven te worden. Ter voorbeeld, als eerste wordt er een profile aangemaakt waarin automatisch gemappt kan worden, maar ook zelf regels geschreven kunnen worden:

public class ArtistProfile : Profile

{

public ArtistProfile()

{

CreateMap<Artist, ArtistViewModel>();

}

}

Door middel van dependency injection kan deze mapper gebruikt worden zoals in een request waarin het resultaat van een Artist automatisch gemapt wordt naar een ArtistViewModel:

return Mapper.Map<ArtistViewModel>(result);

### FluentValidation

Door middel van fluent validation kunnen requests automatisch worden gevalideerd voordat ze uitgevoerd worden. Er kunnen zelf eisen worden opgesteld waar een request (zie GetArtistQuery object in 5.3.1) aan moet voldoen. Als een request niet aan de juiste eisen voldoet kan er een exceptie opgegooid worden. Een voorbeeld van een validator:

public class ArtistViewModelValidator : AbstractValidator<GetArtistQuery>

{

public ArtistViewModelValidator()

{

RuleFor(req => req.Id).NotEmpty().Length(24);

}

}

En om er voor te zorgen dat elke request gevalideerd wordt, is het volgende stukje code geïmplementeerd in de mediatr pipeline [[6]](#footnote-6):

public class RequestValidationBehavior<TRequest, TResponse> : IPipelineBehavior<TRequest, TResponse>

where TRequest : IRequest<TResponse>

{

private readonly IEnumerable<IValidator<TRequest>> \_validators;

private readonly ILogger \_logger;

public RequestValidationBehavior(IEnumerable<IValidator<TRequest>> validators, ILogger<TRequest> logger)

{

\_validators = validators;

\_logger = logger;

}

public Task<TResponse> Handle(TRequest request, CancellationToken cancellationToken,

RequestHandlerDelegate<TResponse> next)

{

var failures = \_validators

.Select(v => v.Validate(request))

.SelectMany(result => result.Errors)

.Where(f => f != null)

.ToList();

if (failures.Count != 0) LogErrors(failures);

return next();

}

public void LogErrors(List<ValidationFailure> failures)

{

foreach (var failure in failures)

{

\_logger.LogError(failure.ErrorMessage);

throw new CustomValidationException(failure);

}

}

}

## Backend code voorbeeld van een API call

Het volgende is een voorbeeld van een API request dat via de backend loopt.

[Route("api/[controller]")]

[ApiController]

public class BaseController : Controller

{

protected IMediator Mediator;

protected JsonSerializerSettings JsonDictionaryAsArrayResolver;

public BaseController()

{

JsonDictionaryAsArrayResolver = new JsonSerializerSettings

{

ContractResolver = new DictionaryAsArrayResolver(),

};

}

protected IMediator GetMediator()

{

return Mediator ?? (Mediator = HttpContext.RequestServices.GetService<IMediator>());

}

}

[Route("api/[controller]")]

[ApiController]

public class PlacesController : BaseController

{

[HttpGet("best/{lat}/{lon}")]

public async Task<ActionResult<PerformanceViewModel>> GetBest(double lat, double lon)

{

var result = await GetMediator().Send(new GetBestPlacesQuery

{

UserLocationCoordinate = new LocationCoordinate(lat, lon)

});

var arrayDictResult = JsonConvert.SerializeObject(result, JsonDictionaryAsArrayResolver);

return Ok(arrayDictResult);

}

}

public class GetBestPlacesHandler : GeneralRequestHandler<GetBestPlacesQuery, BestPlacesViewModel>

{

private readonly IPlacesRepository \_placessRepository;

public GetBestPlacesHandler(IMapper mapper, IPlacesRepository placesRepository) : base(mapper)

{

\_placessRepository = placesRepository;

}

public override async Task<BestPlacesViewModel> Handle(GetBestPlacesQuery request, CancellationToken cancellationToken)

{

var places = await \_placessRepository.GetPlaces();

var bestPlacesDict = new Dictionary<string, List<BestPlace>>();

var maxPercentage = 0;

foreach (var place in places)

{

List<BestPlace> bestPlacesList;

if (bestPlacesDict.ContainsKey(place.Type.ToString()))

{

bestPlacesDict.TryGetValue(place.Type.ToString(), out bestPlacesList);

bestPlacesDict.Remove(place.Type.ToString());

}

else

{

bestPlacesList = new List<BestPlace>();

}

bestPlacesList?.Add(Mapper.Map<BestPlace>(place, opt => opt.Items["userlocation"] = request.UserLocationCoordinate));

bestPlacesDict.Add(place.Type.ToString(), bestPlacesList);

if (maxPercentage < place.CrowdPercentage.AbsolutePercentage)

maxPercentage = place.CrowdPercentage.AbsolutePercentage;

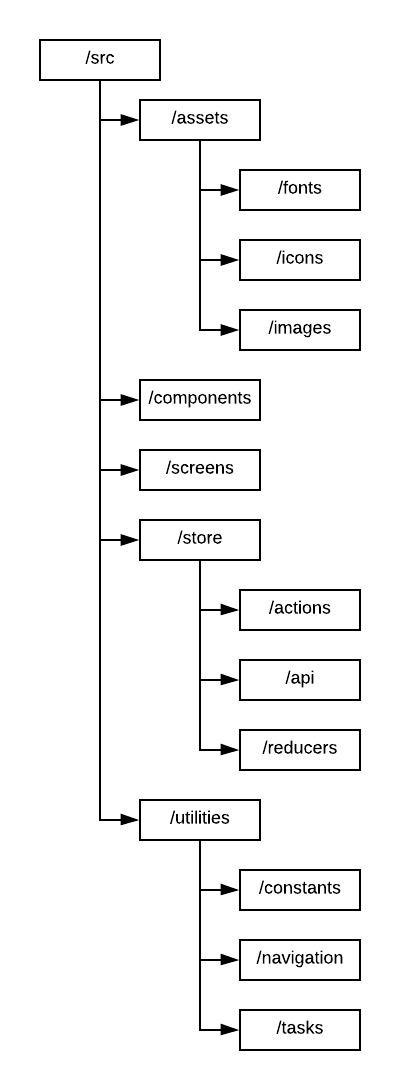
}

foreach (var entry in bestPlacesDict) entry.Value.Sort();

return Mapper.Map<BestPlacesViewModel>(bestPlacesDict, opt => opt.Items["maxpercentage"] = maxPercentage);

# Front-end structuur

## Opzet

Op de afbeelding links staat aangetoond hoe de basis structuur van de front-end is opgezet onder de hoofdmap genaamd src.

De eerste laag zijn de assets. De assets bevatten de externe middelen die de app nodig heeft om in zijn volledigheid getoond te kunnen worden naar aanleiding van het gemaakte ontwerp. Denk hierbij aan de juiste fonts, iconen en plaatjes.

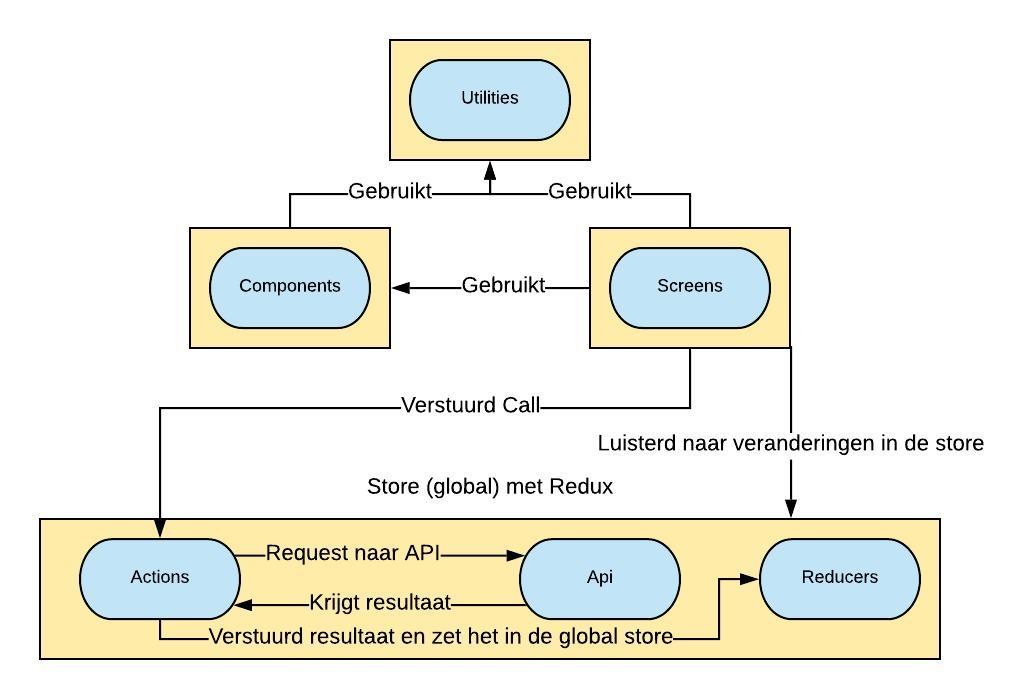
In de componenten folder bevinden zich alle (mogelijke) herbruikbare componenten waarbij geen gebruik gemaakt hoeft te worden van de globale state of store. Hierin is het de bedoeling om op een zo’n klein mogelijk niveau zo min mogelijk duplicatie in de front-end te bouwen.

De screens folder bevat de schermen die getoond worden in de app en waar ook naar genavigeerd kan worden. Deze kunnen ook de globale state of store gebruiken.

De store folder bevat de complete logica van het redux framework waarbij in deze app gebruik van wordt gemaakt. Hier zitten alle reducers, api logica en actions in om de globale state bij te houden en losgekoppeld van de view verbinding te maken met de backend.

Als laatste is er de utilities laag wat geen componenten, schermen of actie logica is. Hierbij moet worden gedacht aan een bestand die de constanten bijhoudt, de navigatie functionaliteiten en de taken die in de achtergrond moeten draaien.

## Low level architectuur schets



De front-end architectuur bevat net als de back-end architectuur verschillende componenten. Er zijn 2 verschillende UI componenten, 1 utility component en 1 store / logica component. Het utility component kan gebruikt worden door de twee verschillende UI componenten (Components en Screens), hierin staan een aantal overlappende elementen van de applicatie beschreven zoals constanten en de navigatie. De 2 UI componenten zijn opgedeeld waar het Components element verschillende store- en/of stateless visuele aspecten bevat zoals een knop of header en het Screens component bevat de schermen die de gebruiker daadwerkelijk ziet en verschillende componenten kan bundelen. Het store / logica component is erg belangrijk voor de data uitwisseling tussen de verschillende schermen en de API. Een scherm (screen) kan een actie (call) versturen naar het Actions element welke, waar nodig, een request naar de API stuurt. Wanneer de action een resultaat terug krijgt wordt dit doorgestuurd naar de reducer welke het bij behorende object update in de globale store. Wanneer een object in de globale store veranderd, kan een scherm die hier naar luistert een verandering aanbrengen. Op deze manier kan er asynchroon data ingeladen worden.

## Gebruik van Redux

Redux is een Javascript framework dat gebruikt wordt om de globale state van een applicatie te beheren. Naast Redux wordt ook Redux-Thunk gebruikt als middleware zodat niet alleen een resultaat van een actie als return type gebruikt kan worden maar ook een nieuwe actie verzonden kan worden. Onderstaand toont aan hoe de applicatie gebruik maakt van Redux en hoe de architectuur hier van in elkaar zit met als voorbeeld de acties voor een gebruiker.

/src/App.js  
/\* eslint react/prefer-stateless-function: 0 \*/

import React, { Component } from 'react';

import { Provider } from 'react-redux';

import InitialNavigator from './src/utilities/navigation/InitialNavigator';

import configureStore from './src/store/configureStore';

const store = configureStore();

class App extends Component {

render() {

return (

<Provider store={store}>

<InitialNavigator />

</Provider>

);

}

}

export default App;

/src/store/configureStore.js

import { createStore, combineReducers, applyMiddleware } from 'redux';

import thunk from 'redux-thunk';

import usersReducer from './reducers/users';

import performancesReducer from './reducers/performances';

import placesReducer from './reducers/places';

const rootReducer = combineReducers({

usersStore: usersReducer,

performancesStore: performancesReducer,

placesStore: placesReducer,

});

const configureStore = () => {

return createStore(rootReducer, applyMiddleware(thunk));

};

export default configureStore;

/src/store/redurcers/users.js

import {

ADD\_USER\_IS\_LOADING,

ADD\_USER\_SUCCESS,

ADD\_USER\_FAIL,

UPDATE\_USER\_IS\_LOADING,

UPDATE\_USER\_SUCCESS,

UPDATE\_USER\_FAIL,

REMOVE\_USER\_IS\_LOADING,

REMOVE\_USER\_SUCCESS,

REMOVE\_USER\_FAIL,

} from '../actions/actionTypes';

const initialState = {

addUserAction: {

newUser: {},

error: false,

loading: false,

succes: false,

},

updateUserAction: {

updateUser: {},

error: false,

loading: false,

succes: false,

},

removeUserAction: {

error: false,

loading: false,

succes: false,

},

};

const reducer = (state = initialState, action) => {

switch (action.type) {

case ADD\_USER\_IS\_LOADING:

return {

...state,

addUserAction: { newUser: null, loading: action.payload, error: false, succes: false },

};

case ADD\_USER\_SUCCESS:

return {

...state,

/src/store/redurcers/users.js

addUserAction: { newUser: action.payload, loading: false, error: false, succes: true },

};

case ADD\_USER\_FAIL:

return {

...state,

addUserAction: { newUser: null, loading: false, error: action.payload, succes: false },

};

case UPDATE\_USER\_IS\_LOADING:

return {

...state,

updateUserAction: {

updateUser: null,

loading: action.payload,

error: false,

succes: false,

},

};

case UPDATE\_USER\_SUCCESS:

return {

...state,

updateUserAction: {

updateUser: action.payload,

loading: false,

error: false,

succes: true,

},

};

case UPDATE\_USER\_FAIL:

return {

...state,

updateUserAction: {

updateUser: null,

loading: false,

error: action.payload,

succes: false,

},

};

case REMOVE\_USER\_IS\_LOADING:

return {

...state,

removeUserAction: {

error: false,

loading: true,

succes: false,

/src/store/redurcers/users.js

},

};

case REMOVE\_USER\_SUCCESS:

return {

...state,

removeUserAction: {

error: false,

loading: false,

succes: true,

},

};

case REMOVE\_USER\_FAIL:

return {

...state,

removeUserAction: {

error: action.payload,

loading: false,

succes: false,

},

};

default:

return state;

}

};

export default reducer;

/src/store/actions/users.js  
import { AsyncStorage } from 'react-native';

d

import { LocalStorageKeys } from '../../utilities/constants/constants';

import { Post, Update, Delete } from '../api/serverRequests';

import {

ADD\_USER\_IS\_LOADING,

ADD\_USER\_SUCCESS,

ADD\_USER\_FAIL,

UPDATE\_USER\_IS\_LOADING,

UPDATE\_USER\_SUCCESS,

UPDATE\_USER\_FAIL,

REMOVE\_USER\_IS\_LOADING,

REMOVE\_USER\_SUCCESS,

REMOVE\_USER\_FAIL,

} from './actionTypes';

/src/store/actions/users.js

export const addUser = user => {

d

return dispatch => {

dispatch(addUserIsLoading(true));

Post('users', user)

.then(async result => {

if (result.status !== 200) {

throw Error(result.statusText);

}

user = result.data;

user.currentLocation = {};

await AsyncStorage.setItem(LocalStorageKeys.User.Key, JSON.stringify(user)).then(() => {

return user;

});

})

.then(result => JSON.stringify(result))

.then(user => dispatch(addUserSuccess(user)))

.catch(() => dispatch(addUserFailure(true)));

};

};

export const addUserIsLoading = bool => {

return {

type: ADD\_USER\_IS\_LOADING,

payload: bool,

};

};

export const addUserSuccess = newUser => {

return {

type: ADD\_USER\_SUCCESS,

payload: newUser,

};

};

export const addUserFailure = error => {

return {

type: ADD\_USER\_FAIL,

payload: error,

};

};

export const updateUser = user => {

return dispatch => {

dispatch(updateUserIsLoading(true));

Update('users', user)

/src/store/actions/users.js

.then(async result => {

d

if (result.status !== 200) {

throw Error(result.statusText);

}

await AsyncStorage.mergeItem(LocalStorageKeys.User.Key, JSON.stringify(result.data)).then(

() => {

return result.data;

}

);

})

.then(result => JSON.stringify(result))

.then(user => dispatch(updateUserSuccess(user)))

.catch();

};

};

export const updateUserIsLoading = bool => {

return {

type: UPDATE\_USER\_IS\_LOADING,

payload: bool,

};

};

export const updateUserSuccess = newUser => {

return {

type: UPDATE\_USER\_SUCCESS,

payload: newUser,

};

};

export const updateUserFailure = error => {

return {

type: UPDATE\_USER\_FAIL,

payload: error,

};

};

export const removeUser = userId => {

return dispatch => {

dispatch(removeUserIsLoading(true));

Delete(`users/${userId}`)

.then(async result => {

if (result.status !== 200) {

throw Error(result.statusText);

}

/src/store/actions/users.js

d

await AsyncStorage.clear().then(() => dispatch(removeUserSuccess()));

})

.then(() => dispatch(removeUserSuccess()))

.catch(error => dispatch(removeUserFailure(error)));

};

};

export const removeUserIsLoading = bool => {

return {

type: REMOVE\_USER\_IS\_LOADING,

payload: bool,

};

};

export const removeUserSuccess = () => {

return {

type: REMOVE\_USER\_SUCCESS,

};

};

export const removeUserFailure = error => {

return {

type: REMOVE\_USER\_FAIL,

payload: error,

};

};

# Bibliografie

Atlassian. (sd). *Gitflow Workflow*. Opgehaald van Atlassian: https://es.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Battiston, M. (2016, April 23). *Clean Architecture example*. Opgehaald van Github: https://github.com/mattia-battiston/clean-architecture-example

Martin, R. C. (2012, August 13). *The Clean Architecture*. Opgehaald van The Clean Code Blog: http://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Microsoft. (2018, Augustus 30). *CQRS architecture style*. Opgehaald van Microsoft Azure: https://docs.microsoft.com/nl-nl/azure/architecture/guide/architecture-styles/cqrs

Ploeg, P. v. (2018, Mei 16). *Festivals zijn een wankele groeimarkt*. Opgehaald van nrc: https://www.nrc.nl/nieuws/2018/05/16/festivals-zijn-een-wankele-groeimarkt-a1603174

Rehkopf, M. (sd). *Kanban vs. Scrum* . Opgehaald van Atlassian: https://www.atlassian.com/agile/kanban/kanban-vs-scrum

Vogel, P. (2018, April 01). *Paaspop ontvangt recordaantal van 81.000 bezoekers*. Opgehaald van bd: https://www.bd.nl/schijndel/paaspop-ontvangt-recordaantal-van-81-000-bezoekers~a96429a6/

1. Functionaliteiten die wel moeten werken: rooster van concerten, eigen rooster en artiesten detail pagina. [↑](#footnote-ref-1)
2. Functionaliteiten die niet hoeven te werken: locatie op de kaart, plekken lijst. [↑](#footnote-ref-2)
3. (Martin, 2012) [↑](#footnote-ref-3)
4. (Battiston, 2016) [↑](#footnote-ref-4)
5. Deze requestlogger is niet zelfgeschreven maar is uit deels uit het volgende project overgenomen: https://github.com/JasonGT/NorthwindTraders [↑](#footnote-ref-5)
6. Deze validation pipeline is niet zelfgeschreven maar komt uit het volgende project: https://github.com/JasonGT/NorthwindTraders [↑](#footnote-ref-6)