Bewijsstukken

Beschrijving

Bachelor in de Toegepaste Informatica
keuzerichting Application Development

Brent Broeckx

Academiejaar 2021-2022

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

LID VAN ASSOCIATIE KU LEUVEN

THOMAS MORE

Inhoud

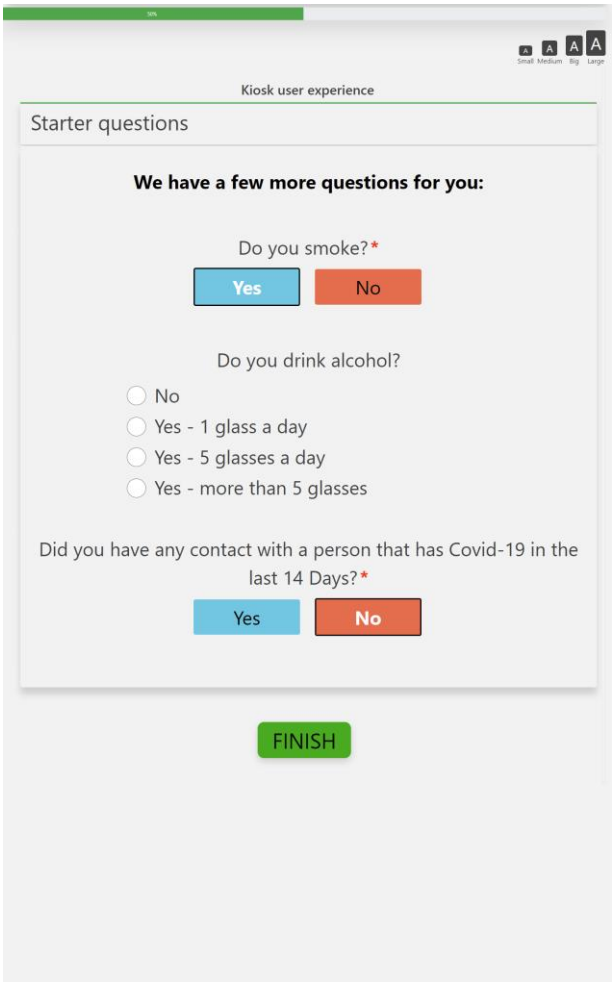
1 KIOSK PROJECT 4

1.1 Design 4
1.2 Gebruiksvriendelijkheid 4
1.3 Eigen datepicker 4
1.4 SiteKiosk 5

2 IFTTT PROJECT 6

2.1 Figma 6
2.2 Configuratie maken (Angular & .NET) 6
2.3 Gebruik van Regex (.NET) 7
2.4 Eigen datatabel (Angular) 8
2.5 Server side zoek functionaliteit (Angular & .NET) 9
2.6 Server side pagination (Angular & .NET) 9
2.7 Dynamic controls (Angular) 10
2.8 Gebruik van decorator (Angular) 10
2.9 Importeren van templates in configuratie (Angular & .NET) 11
2.10 Eigen Authenticate Attribute (.NET) 12

1 KIOSK PROJECT



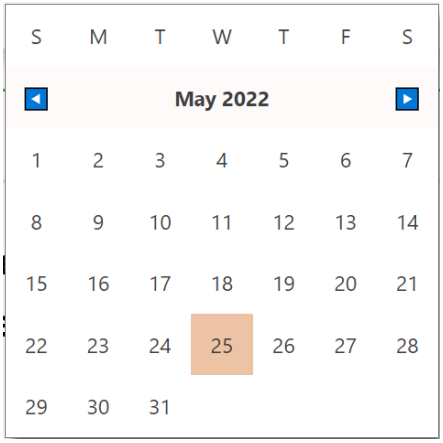
1.1 Design

Het was mijn taak om het design van de assessments aan te passen zodat deze responsief is en er goed uitziet maar nog wel met de stijl dat E.Care gebruikt. Ik ben op zoek gegaan op het internet naar wat de "best practice" is voor kiosk designs. Na grondig onderzoek ben ik aan de slag gegaan en heb ik het design aangepast. Dit heeft ervoor gezorgd dat ik extra inzicht heb gekregen in het ontwerpen van websites.

1.2 Gebruiksvriendelijkheid

Iedereen moet deze applicatie kunnen bedienen waardoor het zeer belangrijk was dat de applicatie duidelijk is voor zowel jong als oud. De applicatie moet ook bedienbaar zijn door mensen met een beperking. Hiervoor heb ik nog een onderzoek gedaan over "best practices" voor gebruiksvriendelijkheid. Na het onderzoek heb ik een lettertype zoom functie gemaakt die het lettertype zal vergroten of verkleinen (rechtsboven in de foto) en is er een progressiebalk voorzien zodat iedereen weet hoe ver hij of zij in het proces zit.

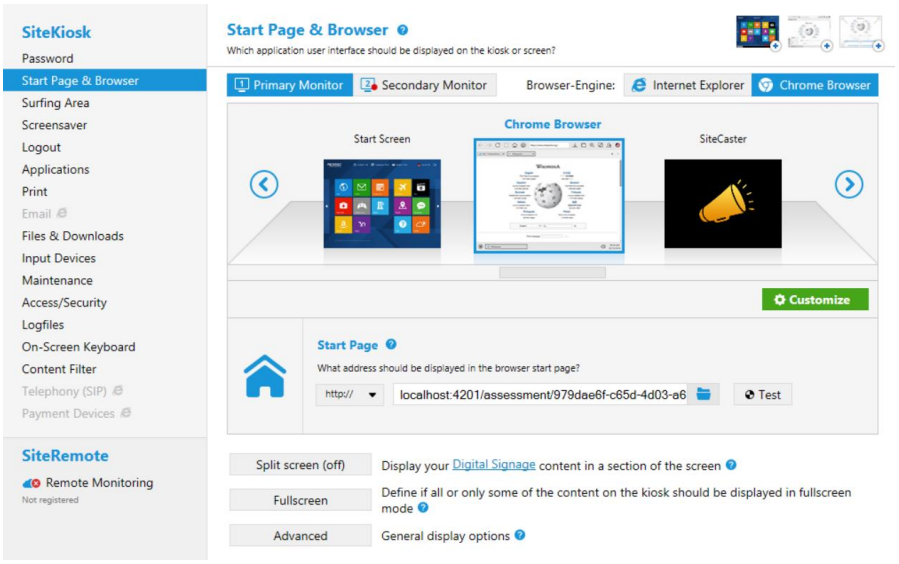
1.3 Eigen datepicker



In de assessment module (die gebruikt wordt in dit project om vragen te tonen) gebruikten we een gewone datepicker van html maar deze was niet goed genoeg voor wat wij nodig hadden. Omdat we onze lettertypes specifiek willen vergroten en verschillende taal mogelijkheden gebruiken heb ik voorgesteld om een eigen datepicker te maken die ze ook later zelf nog kunnen aanpassen aan het project waar ze deze in gebruiken. Een eigen datepicker heeft als voordeel dat we ermee kunnen doen wat we zelf willen en we niet tegen gehouden worden door de voor gedefinieerde functionaliteiten.

Bij het maken van deze datepicker heb ik enkele functionaliteiten leren kennen die je kan importeren vanuit Angular. Dit waren functionaliteiten zoals coercion angular package, en de "locale functionaliteiten" van angular common package. Deze werden gebruikt om de juiste datums/afkortingen en tijden op te halen.

1.4 SiteKiosk



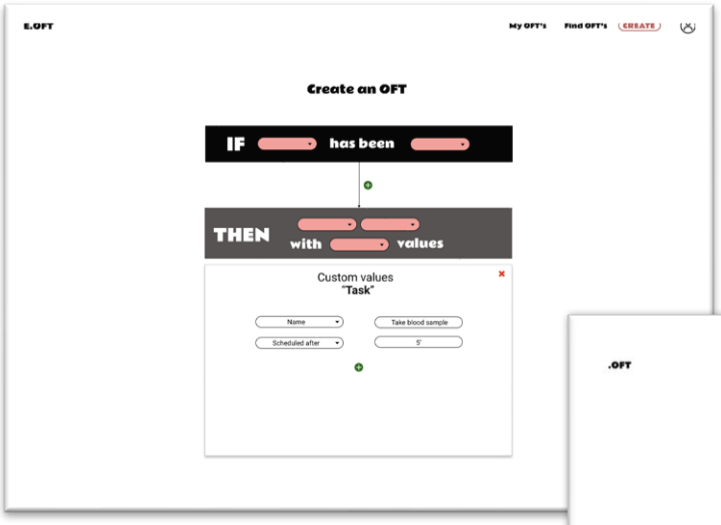
SiteKiosk heb ik leren gebruiken omdat we dit nodig hadden om kiosk environment na te maken. Hiermee kunnen we onze lokale applicatie gebruiken binnen een beveiligde omgeving zodat je kan testen hoe dit zou werken in een kiosk. De bedoeling van SiteKiosk is het afzonderen van functionaliteiten zoals handmatig navigeren, extra opties uit te zetten, …

1.5 Communicatie met partner

In het kiosk project was ik niet enkel verantwoordelijk voor het design maar had ik ook taak gekregen om steeds te communiceren met onze mogelijke partner. De communicatie bevatte steeds informatie over het plannen van meetings, maken van mockups of aanpassingen die moeten gebeuren aan de mockup. De communicatie verliep steeds in het Engels. Omdat ik in het Engels moest praten is mijn vaardigheid in het communiceren sterk verbeterd omdat ik meer zelfvertrouwen heb gekregen in het communiceren in het Engels.

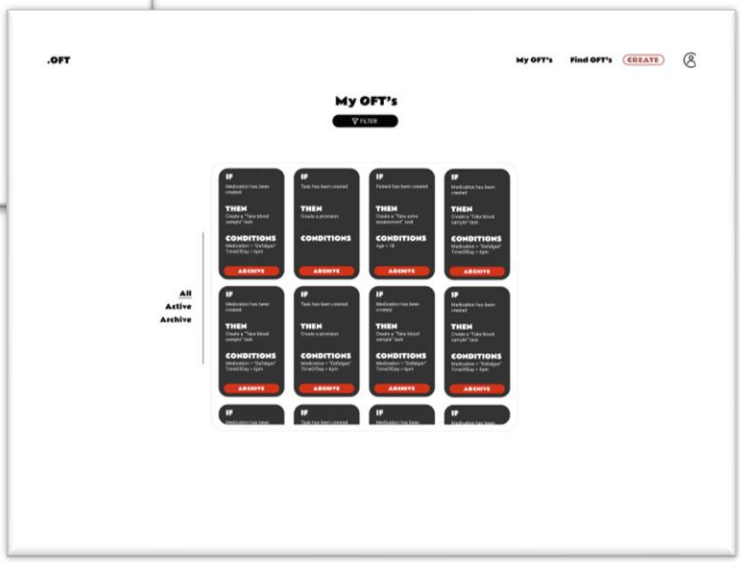
In dit project heb ik ook demo’s moeten geven aan het management team wat mij meer zelfvertrouwen heeft gegeven in het presenteren omdat ik dit volledig zelf voorbereid had.

2 IFTTT PROJECT



2.1 Figma

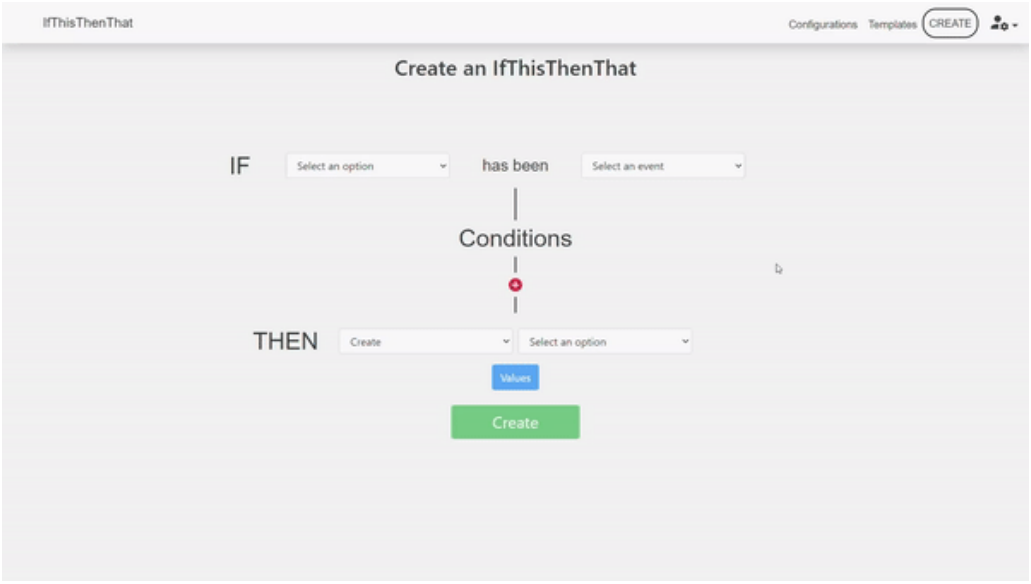
Voor het IFTTT project heb ik Figma gebruikt om een design te maken. Het voordeel hier van is dat ik bij het uitwerken meteen weet hoe de layout gemaakt moet worden.



In een vorig project had ik al een keer Figma gebruikt en wou ik er graag meer over leren. Daarom heb ik in dit project een echt prototype in Figma gemaakt waarbij er animaties zijn en een werkende navigatie. Hiervoor heb ik eerst onderzoek gedaan naar hoe je dit maakt en ben ik dit gaan gebruiken.

2.2 **Configuratie maken** (Angular & .NET)

Eén van de belangrijkste aspecten in dit project is het maken van een configuratie. Ik heb een pagina gemaakt waar IFTTT (If This, Then That) volledig in getoond word. Het maken van een configuratie moet duidelijk zijn waardoor ik deze heb verdeeld in 3 blokken. Het design is ook veranderd omdat deze meer een bootstrap stijl moest zijn zodat dit er hetzelfde uitziet als andere webapplicaties van E.Care.



Op deze pagina kan de gebruiker enkele functionaliteiten uitvoeren zoals

- Kiezen van IF item en bijhorende event (IF item is de trigger die moet gebeuren alvorens het uitvoeren van het THEN item)
- Condities toevoegen aan het IF item die voldaan moeten zijn voordat het THEN item uitgevoerd zal worden. (optioneel).
- Als volgende stap kan je een THEN item kiezen. (THEN item is de actie die uitgevoerd zal worden als vorige condities voldoen aan de voorwaarden)
    - De actie zal altijd een record creëren in de database. Hierdoor is er een functionaliteit om eigen gegevens toe te kennen aan het item dat gemaakt wordt. Er worden maar enkele velden toegelaten om aan te passen van dat item. Hiervoor gebruiken we een decorator (2.8 Gebruik van Decorator)

2.3 Gebruik van Regex (.NET)

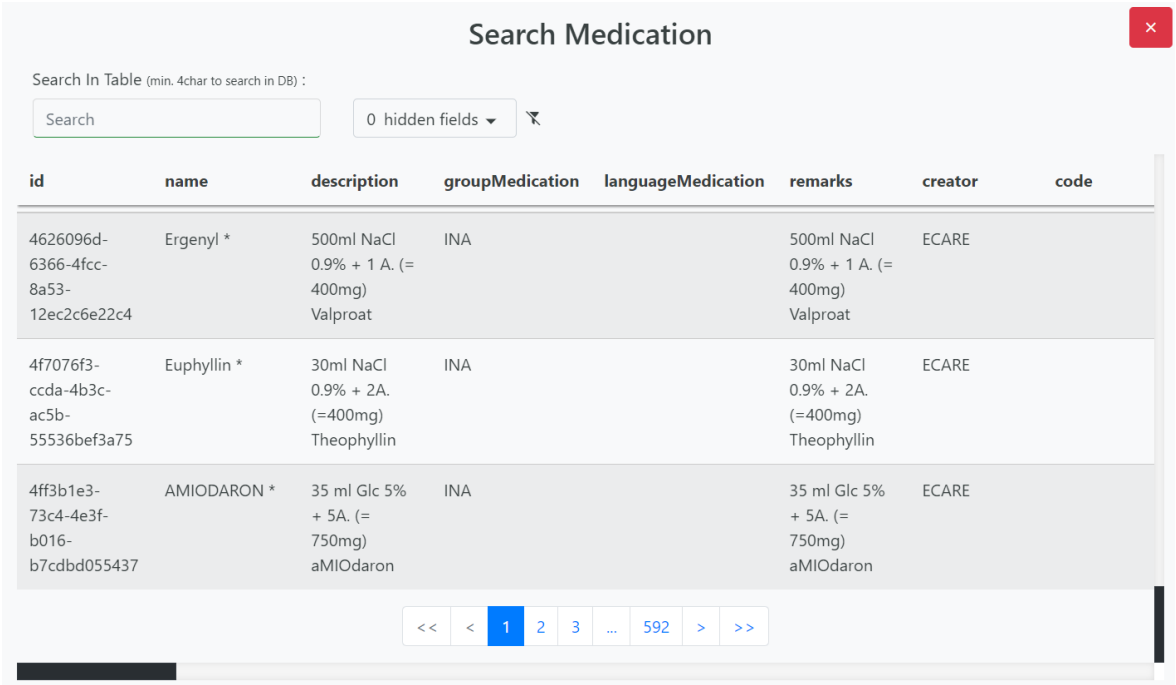
De condities die geconfigureerd worden, worden opgeslagen in een string. Om deze string terug naar condities om te zetten heb ik gebruik gemaakt van Regex. Deze zal de verschillende delen uit de string kunnen halen en weer een lijst van condities terug kunnen geven.

```
const string regex = @"(?<Target>([a-zA-Z0-9_]*)) (?<Operator>([!=><])*) \'(?<Value>([a-zA-Z0-9_* ]*))\' ?(?<AndOrOperator>(AND|OR))*";
```

Dit is één van de eerste keren dat ik een eigen Regex heb gemaakt en heb moeten gebruiken. Eerst heb ik onderzocht hoe een Regex juist werkt en wat je er allemaal mee kan doen. Vervolgens ben ik via websites die je hierbij helpen een Regex samen gaan stellen en ben ik zo tot een werkend resultaat gekomen.

In de Regex zie je op sommige plaatsten “?<…>()” voorbij komen. Dit creëert groepen in een Regex met de aangegeven naam. Als je later deze Regex gaat controleren dan zal het een lijst met matches teruggeven met de aangegeven namen zodat je makkelijker onderscheid kunt maken tussen de verschillende delen en ze ook apart kunt gebruiken indien nodig.

2.4 **Eigen datatabel** (Angular)



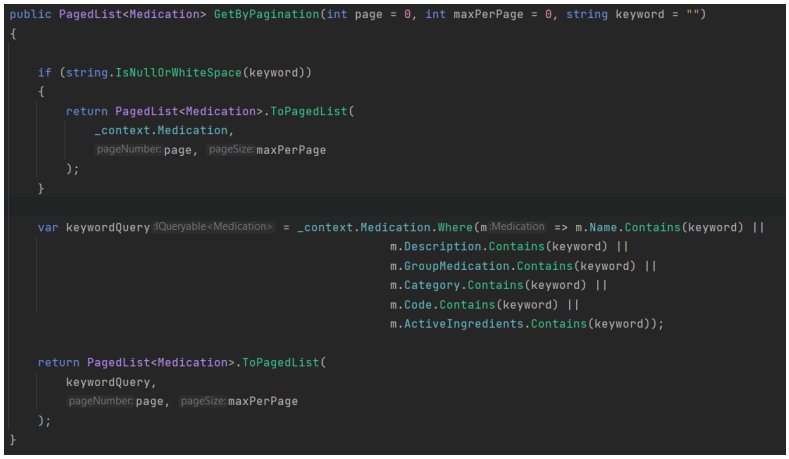
Omdat er met veel verschillende data gewerkt word, maar wel overal dezelfde tabel getoond zal worden heb ik een eigen datatabel component gemaakt. De tabel zal zichzelf dynamisch aanpassen aan de data dat hij binnen krijgt (velden & kolommen) en de functionaliteit hiervan. Omdat de tabel zeer dynamisch kan werken moeten andere componenten enkel het nodige type van data meegeven en gebeurd de rest automatisch.

Verder beschikt deze tabel over het verbergen of tonen van de kolommen en een zoek functie. Deze zoek functie gebeurde eerst via de Client maar omdat er zoveel data opgehaald moet worden hebben we dit later verplaatst naar een server side zoekfunctie.

Alle data word getoond aan de hand van paginering. Ook deze functionaliteit werd eerst via de Client gedaan maar omdat er met veel data gewerkt wordt is dit later verplaats naar server side paginering.

De veranderingen naar server side werden gedaan om alles zo vlot mogelijk te laten verlopen. De gebruiker moet nu geen data inladen die hij of zij ook niet ziet.

2.5 **Server side zoek functionaliteit** (Angular & .NET)



```
public PagedList<Medication> GetByPagination(int page = 0, int maxPerPage = 0, string keyword = "")
{

    if (string.IsNullOrWhiteSpace(keyword))
    {
        return PagedList<Medication>.ToPagedList(
            _context.Medication,
            pageNumber: page, pageSize: maxPerPage
        );
    }

    var keywordQuery :IQueryable<Medication> = _context.Medication.Where(m :Medication => m.Name.Contains(keyword) ||
                                                m.Description.Contains(keyword) ||
                                                m.GroupMedication.Contains(keyword) ||
                                                m.Category.Contains(keyword) ||
                                                m.Code.Contains(keyword) ||
                                                m.ActiveIngredients.Contains(keyword));

    return PagedList<Medication>.ToPagedList(
        keywordQuery,
        pageNumber: page, pageSize: maxPerPage
    );
}
```

E.Care heet altijd zijn zoek functionaliteit via de client gedaan. Aangezien ik al ervaring had met het maken van zoek functionaliteiten via de server hebben we voorgesteld om deze functionaliteit niet via de Client te laten gebeuren maar via de server.

De client zal het ingegeven woord (of letters) meegeven aan de server waar de server vervolgens data gaat ophalen waar bepaalde velden voldoen aan een conditie. Bij een zoekfunctionaliteit zal de conditie een .Contains() functie zijn. Deze functie gaat kijken of het bepaalde woord in het veld voorkomt waar je de functie op gebruikt.

2.6 **Server side pagination** (Angular & .NET)

E.Care heeft zijn pagination functionaliteit altijd al via de client gedaan. Net zoals de zoekfunctionaliteit had ik ook al ervaring met server side paginering. Omdat ik er al ervaring in had, heb ik uitgelegd wat je hiervoor kan gebruiken en heb ik de opdracht gekregen om dit te implementeren.

Om de paginering werkend te krijgen heb ik een nieuwe class aangemaakt genaamd “PagedList”. De class moet dezelfde functies bevatten als een gewone lijst maar met extra’s zoals het voorzien van het totaal aantal items in de database, de grootte van de pagina, huidige pagina, de totale pagina en of er een volgende of vorige pagina is.

```
public int CurrentPage { get; private set; }
public int TotalPages { get; private set; }
public int PageSize { get; private set; }
public int TotalCount { get; private set; }
public bool HasPrevious => CurrentPage > 1;
public bool HasNext => CurrentPage < TotalPages;
```

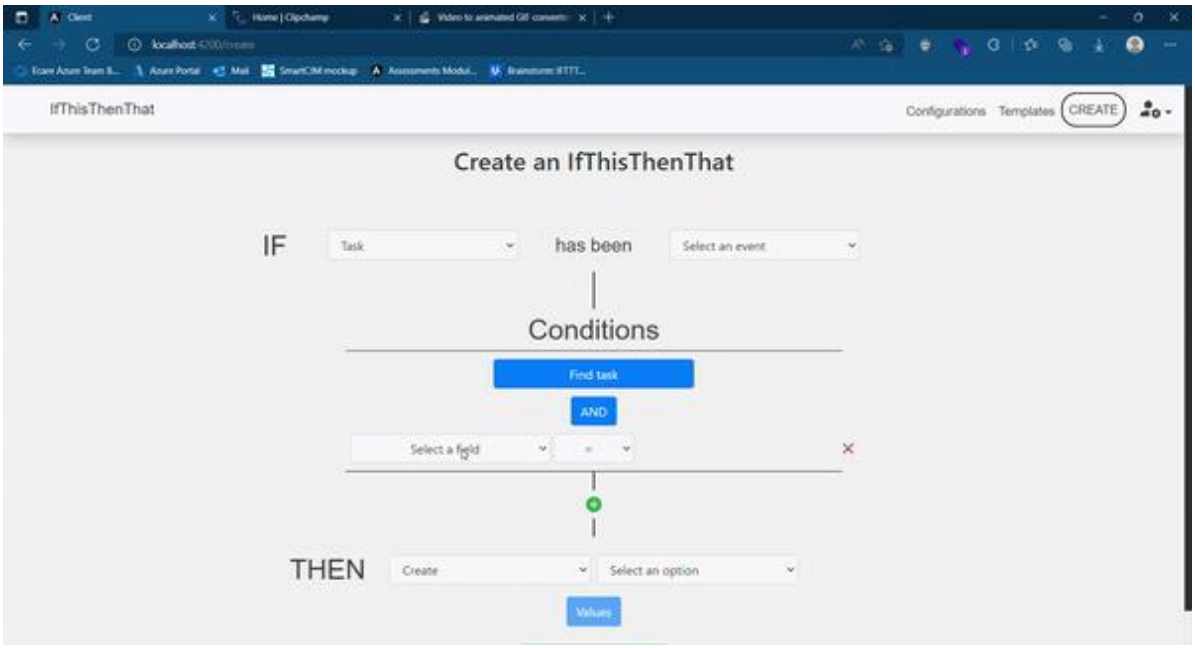
De class bevatte ook een functie om deze lijst te maken waar je een query aan mee kon geven. De functie zal vervolgens de query gebruiken om de functies .Skip() en .Take() op de gebruiken. .Skip() zal een het aantal ingevoerde items overslaan en .Take() zal het aantal ingevoerde items nemen van wat de query gevonden heeft.

```
16 usages   brentb
public static PagedList<T> ToPagedList(IQueryable<T> source, int pageNumber, int pageSize)
{
    var count :int = source.Count();
    var items :List<T> = source.Skip((pageNumber - 1) * pageSize).Take(pageSize).ToList();
    return new PagedList<T>(items, count, pageNumber, pageSize);
}
```

Deze class kan ik nu gebruiken om de nodige gegevens door de sturen. Deze gegevens zijn zowel de data die de client opvraagt maar ook de extra gegevens die meegegeven zullen worden in de header zodat de client hier gebruik van kan maken.

2.7 **Dynamic controls** (Angular)

Om tegen te gaan dat je zomaar alles in velden kunt invullen heb ik "dynamic controls" gemaakt. Met dynamic controls zorg ik ervoor dat bij de input velden eerst gekeken word naar het gekozen veld van het model. Hier neem ik het type van het veld en kan ik op basis van dit type een gepast input veld teruggeven.



Ik heb voor deze functionaliteit enkel dingen moeten onderzoeken. Eerst had ik een idee om een functie te maken die een object terugstuurt met lege waardes waarvan ik het type kan nemen. Na het uitwerken en voorstellen aan mijn stagementor vond mijn stagementor dit een goed initiatief maar niet heel makkelijk. Hierbij stelde mijn stagementor voor om te kijken of je in Angular/Typescript iets kunt gebruiken zoals Attributen in .NET. Ik ben meteen na ons gesprek gaan opzoeken en ben het gebruik van Decorators tegenkomen. Hier heb ik over geleerd en heb ik een decorator gebruikt om het type op te halen en te gebruiken. (Zie 2.8 Gebruik van Decorator)

2.8 **Gebruik van decorator** (Angular)

Voor enkele functionaliteiten heb ik gebruik gemaakt van een decorator.

Een decorator is een functie waarmee we metadata kunnen toewijzen aan een class, functie, property, parameters, … . Samen met deze functionaliteit heb ik de reflect-metadata library gebruikt. Reflect-metadata heeft functies om van een onbekend object alles te weten te komen tijdens run-time. Dit heb ik nodig om de types van de properties op te halen omdat we tijdens run-time werken.

In onderstaande foto kunt u een voorbeeld zien van een functie die ik gemaakt heb. Hierbij staat een volledige beschrijving van wat dit doet zodat dit zeer duidelijk is naar mijn collega's toe in de toekomst. Reflect-metadata beschikt dus over functies zoals getMetaData en defineMetaData. Hiermee kunnen we het typen opvragen van een property en kunnen we onze eigen metadata aanmaken zodat we dit later weer kunnen ophalen. Deze functie word tevens ook gebruikt om aan te duiden welke velden wel getoond mogen worden in de webapplicatie. Als het veld in de metadata zit (@viewable voor de property) dan mag deze getoond worden.

Voor extra gedetailleerde informatie kunt u kijken naar de groene commentaar lijnen in de foto. Hierin word ook met voorbeelden gewerkt.

Enkel ID zal niet getoond worden in de webapplicatie in dit voorbeeld.

```
export class Medication {
    id: string | null;
    @viewable name: string | null;
    @viewable description: string | null;
    @viewable groupMedication: string | null;
    @viewable remarks: string | null;
    @viewable activeIngredients: string | null;
    @viewable dose: string | null;
}       You, 3 weeks ago • #514 Frontend models …
```

```
/*

Explanation:    1.  Reflect-metadata allows us do to runtime reflections on types.
                    This is needed because Typescript compiles models and removes the types until a value is assigned
                    this means we can't get the type without instantiating with a value.
                    reflect-metadata allows us to read types before its compiled.

                2.  We create our own metadata with the modelname and values => (object with keys and types)
                    This way we can access this data within its instantiated model (metadata is saved in instantiated model)
                    {
                        name: String,
                        createdOn: Object (Typeof Date will be of type Object in Typescript when getting the type)
                        isPhysician: Number
                    }

                3.  We can instantiate a model and use that to get our own metadata from within that model (see example)
                    var medicationline = new MedicationLine(); (instantiate model)
                    var metaDataValues = Reflect.getMetadata(MedicationLine.name, medicationline)
                    => (Get metadata that we stored by passing the model name and the instatiated model)
                    => the metadata is stored in the instantiated model, because of this we need to pass the instantiated model as "target"

Source: https://www.npmjs.com/package/reflect-metadata

*/

// When placed on a field in a model it will pass a target / key
// target = instantiated model it is in
// key = field that you placed this function on
export function viewable(target : any, key : string) {
    // Gets the Type of your field (design:type) (stores type in <variable>.name)
    // Needs model and field you want the type of
    var t = Reflect.getMetadata("design:type", target, key);

    // Get our own metadata or [] if it doesn't exist yet
    let variables = Reflect.getMetadata(target.constructor.name, target) || [];
    variables[key] = t.name;

    /*
        - Create our own metadata => way to store a value that can be used.
        - Pass a name (for later access), data and the instantiated model it is in (target);
        This way we create a metadata for every model (that has at least one instance of this function on a field)
        with the model name and insert an object wich contains the key with the type
    */
    Reflect.defineMetadata(target.constructor.name, variables, target);
}       You, 6 days ago • #619 Decorator to allow fields to be viewed / cha…
```

2.9 Importeren van templates in configuratie (Angular & .NET)

```
[
 {
   "Label" : "Medication => Task",
   "Description" : null,
   "IdThis" : null,
   "ThisName" : "Medication ABC",
   "ThisType" : "Medication",
   "ThisSource" : "ecare",
   "IdThat" : null,
   "ThatName" : "Task ABC",
   "ThatType" : "Task",
   "ThatSource" : "ecare",
   "Conditions" : "Id = 123456",
   "Event" : "OnMedicationLineAdded",
   "Active" : true,
   "templateActive": true
 },
```

Een feature dat gevraagd werd door mijn stagementor is om templates ter beschikking te stellen zodat de gebruiker deze kan importeren in zijn configuraties en kan aanpassen waar nodig. Omdat de templates verschillend kunnen zijn per klant kan dit niet op de client “hard coded” geplaatst worden. Om dit op te lossen heb ik een .json bestand in de API gemaakt waar wij als developer standaard templates in kunnen plaatsen. Omdat we gebruik maken van het .json bestand kunnen we dus ook makkelijk per klant in de API het bestand aanpassen wanneer nodig.

2.10 **Eigen Authenticate Attribute** (.NET)

Het IFTTT project mag enkel toegankelijk zijn voor administrators. E.Care heeft zijn eigen Authenticate attribuut gemaakt maar hier hebben ze niet de mogelijkheid gemaakt om ook bepaalde rollen mee te kunnen geven die enkel toegankelijk mogen zijn wanneer deze ingevuld zijn.

Ik ben gaan uitzoeken hoe ik de authenticatie functie kon gaan aanpassen zodat deze functionaliteit wel beschikbaar is. Eerst heb ik opgezocht hoe ik ervoor kon zorgen dat je rollen mee kan geven aan een attribuut. Na het vinden van een oplossing heb ik dit geïmplementeerd en heb ik de authenticatie functie aangepast zodat enkel de meegegeven rol toegang heeft tot wat opgevraagd wordt.

```
public class AuthenticateRole : Attribute
{
    1 usage  brentb
    public AuthenticateRole(string roles = "")
    {
        this.Roles = roles;
    }

    3 usages
    public virtual string Roles { get; }
}
```

```
[ApiController]
[Route( template: "/api/[controller]s")]
[AuthenticateRole( roles: "Administrator, FinancialControl")]
brentb *
public class MedicationController: Controller<Medication>
{
    private MedicationManager _manager;

    brentb
    public MedicationController(MedicationManager manager) : base(manager)
    {
        Options.AllowGetAll = true;
        _manager = manager;
    }
```