Stage Strobbo 2022

Realisatiedocument

Victor Welters

3 APP

Academiejaar 2021-2022

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

LID VAN ASSOCIATIE KU LEUVEN

THOMAS MORE

INHOUDSTAFEL

DANKWOORD

Eerst en vooral zou ik heel graag Strobbo willen bedanken voor de opportuniteit voor een geweldige stage. Dit was natuurlijk niet mogelijk zonder Stefan en Jorn (stagementoren)! Elk moment heb ik genoten van de sfeer en de geweldige collega's.

Ik wil ook mijn stage collega Jana de letter bedanken voor de geweldige samenwerking tussen ons 2. Ik heb er enorm van genoten om met haar samen te werken.

Uiteraard kunnen we ook niet Dirk Mervis (stagebegeleider) vergeten. Hij heeft elk document tijdig met oog naar detail verbeterd en zorgde voor steeds plezierige maar informatieve terugkommomenten!

1 TERMINOLOGIE

TERM	UITLEG
TENANT	Een omgeving voor één of meerdere klanten.
COMPANY	Een BVBA/ onderneming
WORKSPACE	Vestiging/ een fysieke locatie
API-KEY	Een sleutel waarmee klanten de Strobbo applicatie kunnen aanspreken

2 DE OPDRACHT

2.1 Het bedrijf

De opdracht werd uitgevoerd op locatie in de kantoren van Strobbo. Strobbo is een bedrijf gebaseerd in Lommel dat zich specialiseert in personeelsplanning & tijdregistratie voor flexibele workforces. Zij hebben een core product dat de personeelsplanning & tijdsregistratie automatiseert met payroll integratie, automatische dimona aangifte en een digitale tik klok. Dit doen ze voor klanten zoals McDonalds, Burger King, Pizza Hut, Delhaize, ...

2.2 Het projectteam

Het project dat tijdens deze stage is gemaakt, is door 2 stagiaires gerealiseerd namelijk:

- Mijzelf (Victor Welters)
- Jana de Letter

Voor de begeleiding binnen Strobbo stond Jorn Snoeks in als mijn stagementor.

Mijn stagebegeleider bij Thomas More was Dirk Mervis.

2.3 Projectbeschrijving

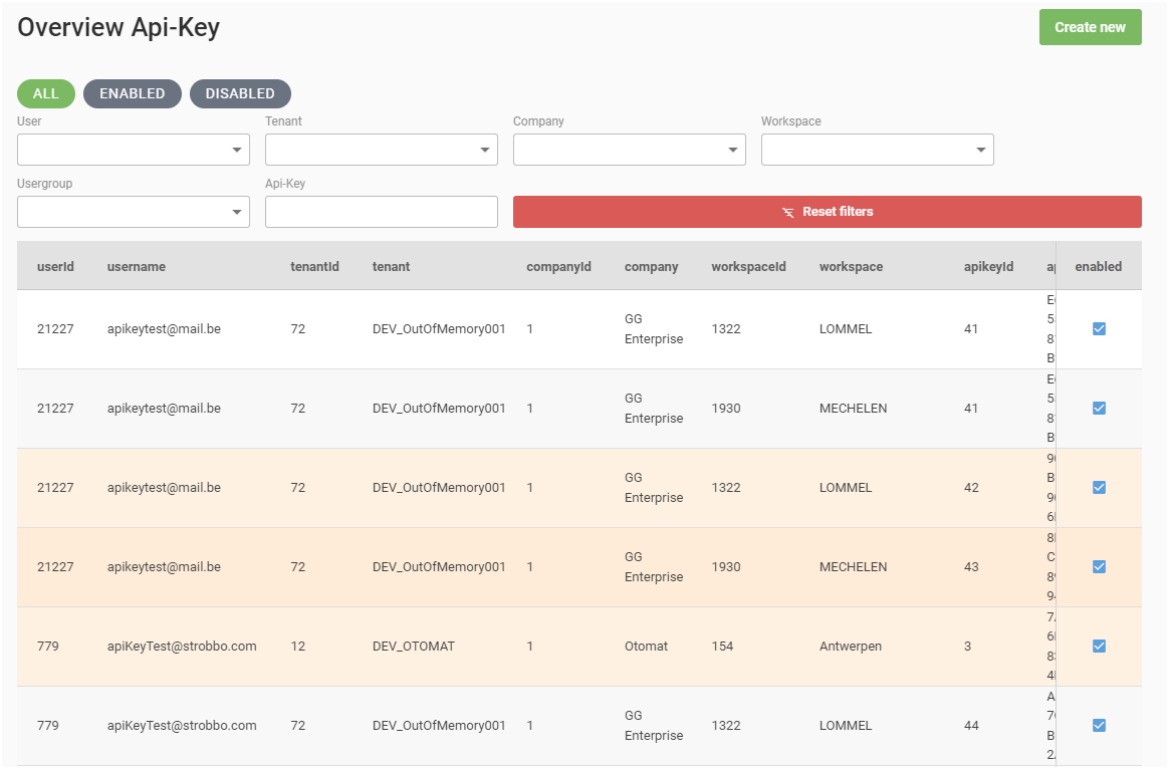
Strobbo gebruikte een SQL-script om API-keys aan te maken en een ander SQL-script om API-keys terug te zoeken, waarvan u hieronder een voorbeeld kan terugvinden. Het was de taak van mijzelf en mijn mede-stagiaire Jana de Letter om deze scripts om te vormen naar een web-interface in de interne support-portal.

Dit zou het ten eerste makkelijker en toegankelijker maken voor iedereen die API-keys zou willen zoeken/maken, maar zorgt ook dat het maken met minder fouten gebeurt want met id's kopieren en zoeken kunnen er al snel fouten voorvallen.

Dit was dus een taak waar er een back- en frontend moest geschreven worden. Ik werkte voornamelijk aan het backend luik terwijl Jana aan het frontend deel werkte. Er is momenteel al een front- en backend voor de support portal. De huidige frontend is geschreven in Vue.js (typescript) en de huidige backend is geschreven in Node.Js (ook typescript) en knex.js voor de database connection en SQL queries. Met beide frameworks, typescript en de SQL query builder had ik nog geen ervaring.

Als start heeft Jana de designs voor het overzicht van de Api-keys gemaakt.

Het idee was om voor het zoeken van API-keys natuurlijk filters te hebben op User, Tenant, Company, Workspace en Usergroup. Hiervoor moesten er dan zeker filters zijn op de API-keys getter, maar ook zeker getters voor elk van die filters zodat er bijvoorbeeld een user gemakkelijk kan gekozen worden op de frontend als filterparameter. Hiernaast werd er ook nog gevraagd dat een API-key enabled en disabled zou kunnen worden en natuurlijk het creëren van een nieuw API-key moest ook kunnen.



2.3.1 Voorbeeld SQL-script API-keys zoeken

```
USE [OwrMainDB];

SELECT  t.id,
    t.tenantname
    ,ws.CompanyName
    ,ws.WorkspaceName
    ,u.username
    ,ak.id AS 'apikeyid'
    ,ak.guid AS 'APIKey'
    ,ak.CreatedOn
    ,ws.*
FROM tenants t
INNER JOIN usertenants ut ON (t.id = ut.tenantid)
INNER JOIN apikeys ak ON (uT.id = ak.usertenantid)
INNER JOIN users u ON u.id = ut.userid
INNER JOIN workspaces ws ON (
        ws.OwrCompanyId = ak.CompanyId
        AND ws.OwrWorkspaceId = isnull(ak.workspaceid, ws.owrworkspaceid)
        AND ws.TenantId = ut.TenantId)
WHERE t.tenantname LIKE '%tenantnamehere%';
```

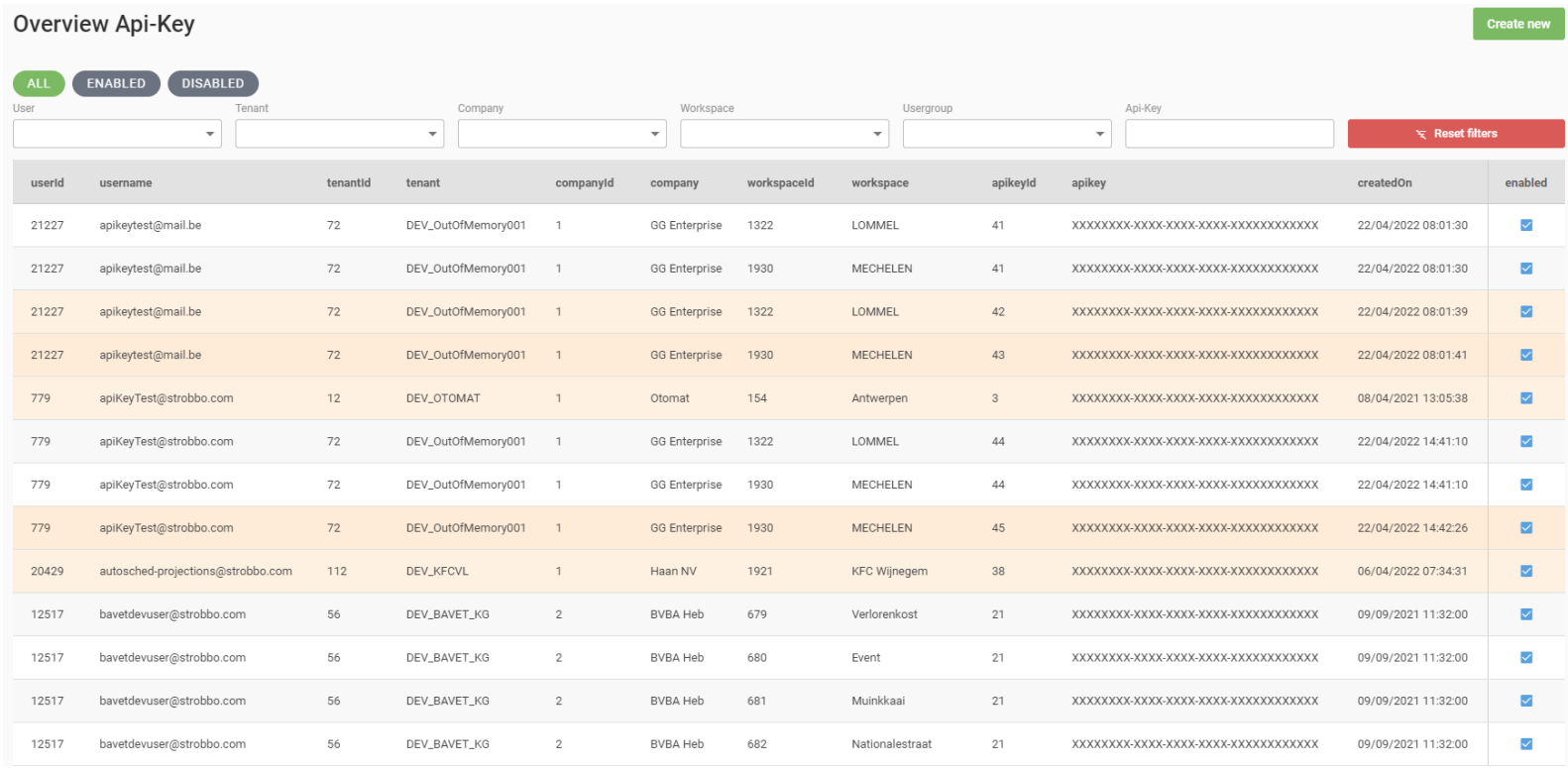
3 REALISATIE OPDRACHT

3.1 Backend

Eerst en vooral heb ik me moeten inwerken in de backend omgeving van Strobbo! Daar begonnen natuurlijk ook de eerste challenges. Ik heb mijzelf de principes van DDD (Domain Driven Design) aangeleerd en dat maakte de hele structuur van de backend al een heel stuk duidelijker. Vervolgens moest ik mijzelf nog Typescript en nodeJS aanleren. Dit heb ik gedaan met behulp van de andere modules die al in de support-portal zaten, samen met wat trial en error. Dit ging redelijk goed, de eerste week had ik zelfs al een werkende getter!

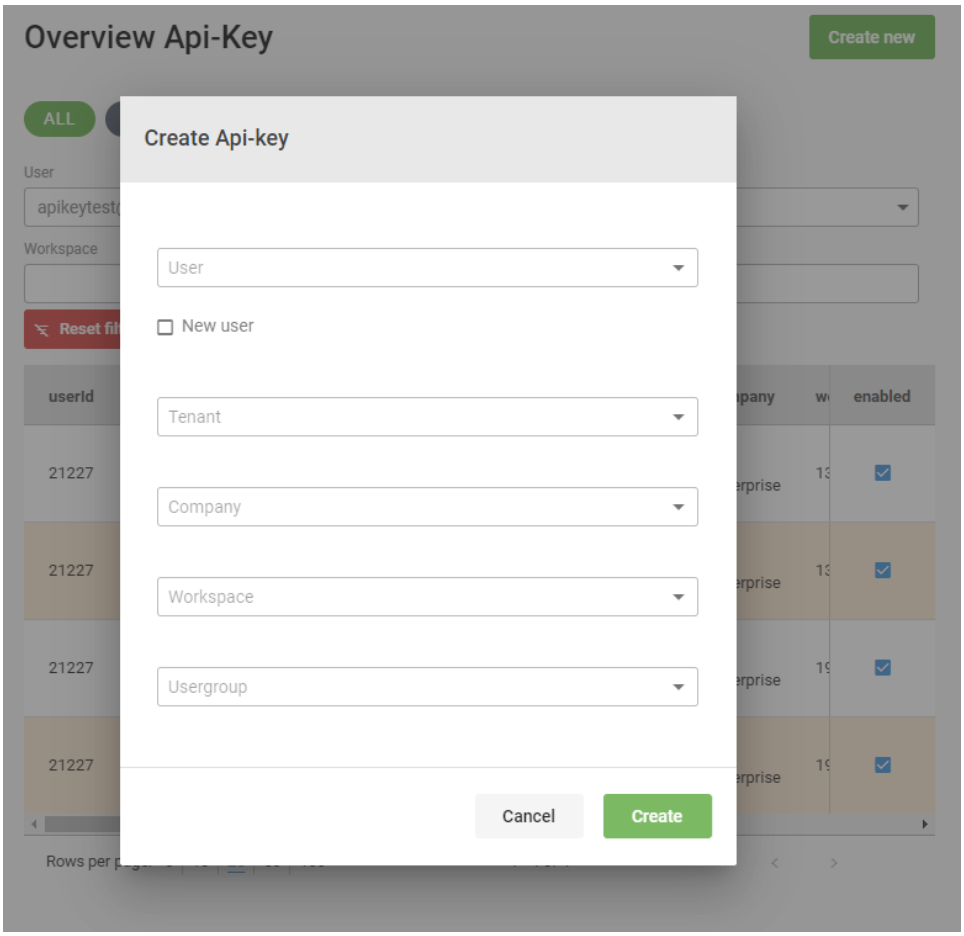
3.1.1 Backend realisatie

Kort gezegd heb ik veel verschillende CRUDs gemaakt! Deze CRUDs zijn allemaal voor de frontend. Hiervoor heb ik veel overleg gedaan met Jana van hoe de endpoints werken, en welke data ze terugsturen. Hieronder vindt u wat screenshots terug hoe de pagina er uit ziet om gemakkelijker een idee te krijgen welke endpoint bij welke functionaliteit hoort.



userId	username	tenantId	tenant	companyId	company	workspaceId	workspace	apikeyId	apikey	createdOn	enabled
21227	apikeytest@mail.be	72	DEV_OutOfMemory001	1	GG Enterprise	1322	LOMMEL	41	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	22/04/2022 08:01:30	☑
21227	apikeytest@mail.be	72	DEV_OutOfMemory001	1	GG Enterprise	1930	MECHELEN	41	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	22/04/2022 08:01:30	☑
21227	apikeytest@mail.be	72	DEV_OutOfMemory001	1	GG Enterprise	1322	LOMMEL	42	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	22/04/2022 08:01:39	☑
21227	apikeytest@mail.be	72	DEV_OutOfMemory001	1	GG Enterprise	1930	MECHELEN	43	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	22/04/2022 08:01:41	☑
779	apiKeyTest@strobbo.com	12	DEV_OTOMAT	1	Otomat	154	Antwerpen	3	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	08/04/2021 13:05:38	☑
779	apiKeyTest@strobbo.com	72	DEV_OutOfMemory001	1	GG Enterprise	1322	LOMMEL	44	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	22/04/2022 14:41:10	☑
779	apiKeyTest@strobbo.com	72	DEV_OutOfMemory001	1	GG Enterprise	1930	MECHELEN	44	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	22/04/2022 14:41:10	☑
779	apiKeyTest@strobbo.com	72	DEV_OutOfMemory001	1	GG Enterprise	1930	MECHELEN	45	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	22/04/2022 14:42:26	☑
20429	autosched-projections@strobbo.com	112	DEV_KFCVL	1	Haan NV	1921	KFC Wijnegem	38	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	06/04/2022 07:34:31	☑
12517	bavetdevuser@strobbo.com	56	DEV_BAVET_KG	2	BVBA Heb	679	Verlorenkost	21	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	09/09/2021 11:32:00	☑
12517	bavetdevuser@strobbo.com	56	DEV_BAVET_KG	2	BVBA Heb	680	Event	21	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	09/09/2021 11:32:00	☑
12517	bavetdevuser@strobbo.com	56	DEV_BAVET_KG	2	BVBA Heb	681	Muinkkaai	21	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	09/09/2021 11:32:00	☑
12517	bavetdevuser@strobbo.com	56	DEV_BAVET_KG	2	BVBA Heb	682	Nationalestraat	21	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX	09/09/2021 11:32:00	☑

Figuur 1 Overview API-keys



Figuur 2 Create API-key modal

3.1.2 GET Users connected to API-Keys

Deze endpoint wordt gebruikt voor op de frontend API-keys op users te filteren. Hierdoor kan er gemakkelijk een user worden gekozen uit een zoekbare lijst op de overzichtspagina en daarna kan er vanuit de frontend een request worden gestuurd naar de API-key endpoint om daar op de gekozen user te filteren.

Deze endpoint heeft nog een INNER JOIN op API-keys voor alleen de users terug te geven die verbonden zijn aan een API-key dit beperkt de lijst van Users al heel wat en zorgt dat er alleen nuttige users worden weergegeven.

3.1.2.1 Pagination

Deze endpoint heeft uiteraard pagination, dat betekent dat er maar een gekozen aantal (standaard 10) users tegelijkertijd worden getoond. Zo gebeurt de request sneller omdat er minder informatie moet meegestuurd worden en uiteraad is de lijst die de overzichtspagina toont kleiner en gemakkelijker te lezen. Deze grootte per page en uiteraard de pagenumber kan aangepast worden via parameters in de URL, hieronder vindt u een voorbeeld.

https://www.example.com/api/api-keys/users?page=2&size=15

3.1.2.2 Filters

Deze endpoint heeft ook 3 filters, dit betekend dat de selectie van de users kan beperkt worden. Elk van deze filters kunnen aangesproken worden via parameters in de URL.

- Filter tenantId
    - De filter naast users is de tenant filter, als daar een tenant wordt gekozen wordt deze filter aangevuld met de tenantId van de gekozen tenant. Hiermee worden alleen de users getoond die aan die tenant hangen. Dit beperkt de userkeuze naar alleen de nuttige users.

    - https://www.example.com/api/api-keys/users?filter[tenantId]=5

- Filter userName
    - De select boxes in de overzichtspagina zijn doorzoekbaar, dit betekent dat er gezocht moet kunnen worden op een specifieke user. Als er “hoorn” wordt getypt in de select box zou de select box de user “ehoorntje@info.be” moeten tonen, daarom dat deze filter ook met een LIKE operator werkt.

    - https://www.example.com/api/api-keys/users?filter[userName]=hoorn

- Filter userId
    - Als er iets in de select box geselecteerd is kan deze filter worden ingevuld en die geeft alleen de user met die ID terug. Dit zou ook moeten gebruikt worden op een search op ID maar die is uiteindelijk niet geïmplementeerd.

    - https://www.example.com/api/api-keys/users?filter[userId]=6

3.1.2.3 Example output


```
{
  "page": {
    "size": 10,
    "number": 2,
    "totalItems": 2,
    "totalPages": 2
  },
  "items": [
    {
      "id": 2,
      "userName": "example@example.com"
    }
  ]
}
```

3.1.3 GET Users

Deze is hetzelfde als GET Users connected to API-Keys, maar dan zonder de JOIN op API-keys. Deze geeft dus ALLE users weer.

Deze endpoint wordt gebruikt bij de create API-key modal, bij het aanmaken moet een userID worden meegestuurd in de POST van de API-key. Hier kan er makkelijk een user worden geselecteerd door behulp van deze endpoint.

Dezelfde filtering en pagination als hierboven wordt ook gebruikt.

https://www.example.com/api/users

3.1.4 POST (dummy) User

Deze POST wordt uiteraard gebruikt om users aan te maken. Specifieker bij de create API-key. Hier staat een checkbox "New User" waar deze POST gaat aangesproken worden.

Deze POST accepteert alleen een username (mailadres) en maakt met deze een dummy user aan om aan de API-key te koppelen. Deze stond eerst als een eigen endpoint maar is nu geïntegreerd in de POST van API-keys omdat die alleen dan word gebruikt.

3.1.4.1 Checks

Om iets aan te maken moeten er natuurlijk heel wat checks zijn om zeker te zijn dat alles goed verloopt!

- Check if username is mail
    - De gegeven username MOET een mailadres zijn. Dit is gemakkelijk te doen met de library class-validator. Met een simpele @IsMail() kan ik zien of de username een goede username is. Zo niet gooit deze een error.

- Check if username exists
    - Natuurlijk willen we nooit dubbele users hebben. Dit is een check voor zeker en vast GEEN dubbele users aan te maken. Deze gooit een custom error UserAlreadyExists.

3.1.5 GET Tenants connected to API-keys

Deze endpoint wordt gebruikt om aan clientzijde op API-keys op een tenant te filteren. Hierdoor kan er gemakkelijk een tenant worden gekozen uit een zoekbare lijst op de frontend en daarna kan er vanuit de overzichtspagina een request worden gestuurd naar de API-key endpoint om daar op de gekozen tenant te filteren.

Deze endpoint heeft nog een inner JOIN op API-keys voor alleen de tenants terug te geven die verbonden zijn aan een API-key. Dit beperkt de lijst van tenants al heel wat en zorgt dat er alleen nuttige tenants worden weergegeven.

3.1.5.1 Pagination

Deze endpoint heeft ook pagination. De grootte per page en uiteraard de pagenumber kan aangepast worden via parameters in de URL, hieronder vindt u een voorbeeld.

https://www.example.com/api/api-keys/tenants?page=2&size=15

3.1.5.2 Filters

Deze endpoint heeft ook 3 filters, dit betekend dat de selectie van de tenant kan beperkt worden. Elk van deze filters kunnen aangesproken worden via parameters in de URL.

- Filter userId
    - De filter naast tenant is de user filter, als daar een user wordt gekozen wordt deze filter aangevuld met de userId van de gekozen user. Hiermee worden alleen de tenants getoond die aan die user hangen. Dit beperkt de tenantkeuze naar alleen de nuttige tenants.

    - https://www.example.com/api/api-keys/tenants?filter[userId]=6

- Filter tenantName
    - De select boxes in de overzichtspagina zijn zoekbaar, dit betekent dat er gezocht moet kunnen worden op een specifieke tenant. Als er “shop” wordt getypt in de selectbox zou de selectbox de tenant “vapeShop” moeten tonen, daarom dat deze filter ook met een LIKE operator werkt.

    - https://www.example.com/api/api-keys/tenants?filter[tenantName]=shop

- Filter tenantId
    - Als er iets in de select box geselecteerd is kan deze filter worden ingevuld en die geeft alleen de tenant met die ID terug. Dit zou ook moeten gebruikt zijn op een search bij ID maar die is uiteindelijk niet geïmplementeerd.

    - https://www.example.com/api/api-keys/tenants?filter[tenantId]=6

3.1.5.3 Example output


```
{
  "page": {
    "size": 10,
    "number": 2,
    "totalItems": 2,
    "totalPages": 2
  },
  "items": [
    {
      "id": 14,
      "tenantName": "TenantNameHere"
    }
  ]
}
```

3.1.6 GET tenants

Deze is hetzelfde als GET Tenants connected to API-keys, maar dan zonder de JOIN op API-keys. Deze geeft dus ALLE tenants weer.

Deze endpoint wordt gebruikt bij de create API-key modal, bij het aanmaken moet een tenantId worden meegestuurd in de POST van API-key. Hier kan er makkelijk een tenant worden geselecteerd door behulp van deze endpoint.

Dezelfde filtering en pagination als hierboven wordt ook gebruikt.

https://www.example.com/api/tenants

3.1.7 GET Companies

Deze endpoint wordt gebruikt om op de overzichtspagina API-keys op companies te filteren. Hierdoor kan er gemakkelijk een company worden gekozen uit een zoekbare lijst aan de client zijde en daarna kan er vanuit de frontend een request worden gestuurd naar de API-key endpoint om daar op de gekozen company te filteren.

Voor aan de companies te geraken haal ik de companyName uit de Workspace tabel (Hier wordt geen Id meegestuurd). De companies van een tenant zitten in een andere database dus is deze omweg nodig.

Het kan zijn dat in verschillende tenants dezelfde companyName zou voorkomen. Dus is deze endpoint gemaakt met een DISTINCT-operator. Dit betekent dat deze alleen de unieke companies toont.

3.1.7.1 Pagination

Deze endpoint heeft ook pagination. Deze grootte per page en uiteraad de pagenumber kan aangepast worden via parameters in de URL, hieronder vindt u een voorbeeld.

https://www.example.com/api/companies?page=2&size=15

3.1.7.2 Filters

Deze endpoint heeft ook 4 filters, dit betekend dat de selectie van de companies kan beperkt worden. Elk van deze filters kunnen aangesproken worden via parameters in de URL.

- Filter workspaceName
    - De filter naast companies is de workspace filter (deze gebruikt alleen workspaceNames, geen Id's), als daar een workspace wordt gekozen wordt deze filter aangevuld met de workspaceName van de gekozen workspace. Hiermee worden alleen de companies getoond die aan die workspace hangen. Dit beperkt de companykeuze naar alleen de nuttige companies.

      Het idee was om deze filter ook te laten werken met halve workspaceNames, dus is deze filter gemaakt met een LIKE operator. Maar dit is uiteindelijk niet gebeurd.

    - https://www.example.com/api/companies?filter[workspaceName] =workspaceName

- Filter tenantId
    - De filter naast companies is de tenant filter, als daar een tenant wordt gekozen wordt deze filter aangevuld met de tenantId van de gekozen tenant. Hiermee worden alleen de companies getoond die aan die tenant hangen. Dit beperkt de company keuze naar alleen de nuttige companies.

    - https://www.example.com/api/companies?filter[tenantId]=5

- Filter companyName
    - De select boxes in de frontend zijn zoekbaar, dit betekent dat er gezocht moet kunnen worden op een specifieke company. Als er

"vape" wordt getypt in de selectbox zou de selectbox de company "vapeShop" moeten tonen, daarom dat deze filter ook met een LIKE operator werkt.

  - https://www.example.com/api/companies?filter[companyName]=vape

- Filter companyId
  - Deze filter was bedoelt voor een searchCompanyById maar deze is uiteindelijk nooit in de frontend geïntegreerd.

  - https://www.example.com/api/companies?filter[companyId]=3

3.1.7.3 Example output

```
{
    "page": {
      "size": 10,
      "number": 2,
      "totalItems": 2,
      "totalPages": 2
    },
    "items": [
      {
        "companyName": "CompanyNameHere"
      }
    ]
  }
```

3.1.8 GET companies by tenantId

Deze endpoint returned hetzelfde als GET Companies maar deze toont alleen de companies die aan een bepaalde tenant hangen. Dit wordt gebruikt bij de create API-key modal. Hier moet eerst een tenant geselecteerd worden en dan wordt de tenantId als parameter in de URL meegestuurd. Hier is geen DISTINCT operator op.

Hier wordt ook dezelfde pagination en filters gebruikt, buiten de tenantId filter natuurlijk.

https://www.example.com/api/tenants/{id}/companies

3.1.9 GET workspaces

Deze endpoint wordt gebruikt voor op de overzichtspagina API-keys op workspaces te filteren. Hierdoor kan er gemakkelijk een workspace worden gekozen uit een zoekbare lijst aan client zijde en daarna kan er vanuit de frontend een request worden gestuurd naar de API-key endpoint om daar op de gekozen workspace te filteren.

Deze endpoint wordt ook gebruikt in de create modal van API-keys om gemakkelijk daar een workspace te kiezen op de overzichtspagina voor een API-key aan te maken.

Het kan zijn dat in verschillende tenants dezelfde workspaceName zou voorkomen. Dus is deze endpoint gemaakt met een DISTINCT-operator. Dit betekent dat deze alleen de unieke workspaces toont.

3.1.9.1 Pagination

Deze endpoint heeft pagination. De grootte per page en uiteraad de pagenumber kan aangepast worden via parameters in de URL, hieronder vindt u een voorbeeld.

https://www.example.com/api/workspaces/?page=2&size=15

3.1.9.2 Filters

Deze endpoint heeft ook 4 filters, dit betekend dat de selectie van de workspaces kan beperkt worden. Elk van deze filters kunnen aangesproken worden via parameters in de URL.

- Filter tenantId
    - Een paar filters naast de workspaces is de tenant filter, als daar een tenant wordt gekozen wordt deze filter aangevuld met de tenantId van de gekozen tenant. Hiermee worden alleen de workspaces getoond die aan die workspace hangen. Dit beperkt de workspacekeuze naar alleen de nuttige workspaces.

    - https://www.example.com/api/workspaces?filter[tenantId]=5

- Filter companyName
    - De filter naast workspace is de company filter (deze gebruikt alleen companyNames, geen Id's), als daar een company wordt gekozen wordt deze filter aangevuld met de companyName van de gekozen company. Hiermee worden alleen de workspaces getoond die aan die company hangen. Dit beperkt de workspace keuze naar alleen de nuttige workspaces.

      Het idee was om deze filter ook te laten werken met halve companyNames, dus is deze filter gemaakt met een LIKE operator. Maar dit is uiteindelijk niet gebeurd.

    - https://www.example.com/api/workspaces?filter[companyName]=companyName

- Filter workspaceName
    - De select boxes in de frontend zijn zoekbaar, dit betekent dat er gezocht moet kunnen worden op een specifieke workspace. Als er

"vape" wordt getypt in de selectbox zou de selectbox de workspace "vapeShop" moeten tonen, daarom dat deze filter ook met een LIKE operator werkt.

- https://www.example.com/api/workspaces?filter[workspaceName]=vape

- Filter workspaceId
  - Deze filter was bedoeld voor een searchByWorkspaceId maar deze is uiteindelijk nooit in de frontend gekomen.

  - https://www.example.com/api/companies?filter[workspaceId]=3

3.1.9.3 Example output

```
{
    "page": {
      "size": 10,
      "number": 2,
      "totalItems": 2,
      "totalPages": 2
    },
    "items": [
      {
        "workspaceName": "WorkspaceNameHere"
      }
    ]
  }
```

3.1.10 GET Workspaces by tenantId

Deze endpoint returned hetzelfde als GET workspaces maar deze toont alleen de workspaces die aan een bepaalde tenant hangen. Dit wordt gebruikt bij de create API-key modal. Hier moet eerst een tenant geselecteerd worden en dan wordt de tenantId als parameter in de URL meegestuurd. Hier is geen DISTINCT-operator op.

Hier wordt ook dezelfde pagination en filters gebruikt, buiten de tenantId filter natuurlijk.

https://www.example.com/api/tenants/{id}/companies

3.1.11 GET userGroups

Deze endpoint wordt gebruikt om op de overzichtspagina API-keys op userGroups te filteren. Hierdoor kan er gemakkelijk een userGroup worden gekozen uit een zoekbare lijst aan client zijde en daarna kan er vanuit de frontend een request worden gestuurd naar de API-key endpoint om daar op de gekozen userGroup te filteren.

Deze endpoint wordt ook gebruikt in de create modal van API-keys om gemakkelijk daar een userGroup te kiezen op de overzichtspagina voor een API-key aan te maken.

Deze userGroups komen uit een colom van de apiKeys tabel. Hier staat nog een WHERE NOT NULL op en een WHERE NOT '' op. Dit zorgt dat de lege usergroups niet mee in het getoonde overzicht komen te staan.

3.1.11.1 Pagination

Deze endpoint heeft pagination. De grootte per page en uiteraard de pagenumber kan aangepast worden via parameters in de URL, hieronder vindt u een voorbeeld.

https://www.example.com/api/userGroups?page=2&size=15

3.1.11.2 Filters

Deze endpoint heeft maar 1 filter, dit betekend dat de selectie van de userGroups kan beperkt worden. Deze filter kan worden aangesproken via een parameter in de URL.

- Filter userGroup
    - De select boxes in de overzichtspagina zijn zoekbaar, dit betekent dat er gezocht moet kunnen worden op een specifieke userGroup. Als er "group" wordt getypt in de selectbox zou de selectbox de userGroup "userGroup" moeten tonen, daarom dat deze filter ook met een LIKE operator werkt.

    - https://www.example.com/api/user-groups?filter[userGroup]=group

3.1.11.3 Example output


```
{
    "page": {
      "size": 10,
      "number": 2,
      "totalItems": 2,
      "totalPages": 2
    },
    "items": [
      {
        "userGroup": "UserGroupHere"
      }
    ]
  }
```

3.1.12 GET API-keys

Deze endpoint wordt gebruikt om de lijst van API-keys te tonen op de overzichtspagina.

Deze endpoint geeft heel wat data terug zoals de userName, userId, tenantName, tenantId, companyName, API-key zelf en nog meer. Om dit te doen zijn er tussen de tabellen heel wat joins nodig. De ApiKeys tabel heeft inner joins op UserTenants (tussentabel user en tenants), een inner join op Users, een inner join op Tenants en een inner join op Workspaces.

In deze endpoint wordt ook nog een extra colom isCompanyKey meegestuurd. Dit is een berekening en zorgt ervoor dat het aan clientzijde duidelijk is wat company wide keys zijn en welke specifiek aan een workspace hangen.

3.1.12.1 Pagination

Deze endpoint heeft pagination. De grootte per page en uiteraad de pagenumber kan aangepast worden via parameters in de URL, hieronder vindt u een voorbeeld.

https://www.example.com/api/api-keys?page=2&size=15

3.1.12.2 SortBy

Op deze endpoint zit er ook een SortBy parameter. Dit zorgt ervoor dat de endpoint een andere volgorde kan aannemen, bijvoorbeeld gesorteerd op companyName of workspaceId in dalende en stijgende volgorde. Dit is toegevoegd zodat er in de header van de tabel een kolom kan worden gekozen om op te sorteren.

Deze filter is zelfs zo gemaakt dat er meerdere kolommen en richtingen kunnen gestuurd worden voor op meerdere parameters te kunnen sorteren.

De meegestuurde kolom en richting wordt eerst nagekeken door een regex functie die ervoor zorgt dat deze filter zeker geen errors kan maken door een niet-bestaande kolom of richting.

https://www.example.com/api/api-keys/?sort[]=companyName,asc&sort[]=id,desc

3.1.12.3 Filters

Deze endpoint heeft 8 filters, dit betekend dat de selectie van de API-keys kan beperkt worden. Deze filters kunnen worden aangesproken via parameters in de URL.

- Filter enabled
    - Zoals de naam zegt filtert deze de enabled en disabled API-keys. Op de frontend zijn er buttons om te kiezen of alle API-keys, enkel de Enabled of enkel de Disabled API-keys te laten zien. Deze filter werkt als een Boolean. Enabled=false of Enabled=true. Als deze filter niet aanwezig is displayed de endpoint enabled als disabled API-keys

    - https://www.example.com/api/api-keys?filter[enabled]=false

- Filter workspaceName
    - Boven de lijst van API-keys staat er op de overzichtspagina een workspace filter. Deze gaat uiteraard filteren op workspaces, als er daar een workspace is gekozen gaat deze filter worden ingevuld om de API-keys lijst te filteren op die workspace.

    - Deze filter werkt met een LIKE operator omdat het de bedoeling was om op halve workspace names te kunnen filteren, maar dat is uiteindelijk niet geïmplementeerd in de frontend.

    - https://www.example.com/api/api-keys?filter[workspaceName]=workspaceName

- Filter companyName
    - Boven de lijst van API-keys staat er op de overzichtspagina een company filter. Deze gaat uiteraard filteren op companies, als er daar een company is gekozen gaat deze filter worden ingevuld om de API-keys lijst te filteren op die company.

    - Deze filter werkt met een LIKE operator omdat het de bedoeling was om op halve company names te kunnen filteren, maar dat is uiteindelijk niet geïmplementeerd in de frontend.

    - https://www.example.com/api/api-keys?filter[companyName]=companyName

- Filter userGroup
    - Boven de lijst van API-keys staat er op de overzichtspagina een userGroup filter. Deze gaat uiteraard filteren op userGroups, als er daar een userGroup is gekozen gaat deze filter worden ingevuld om de API-keys lijst te filteren op die userGroup.

    - Deze filter werkt met een LIKE operator omdat het de bedoeling was om op halve userGroup names te kunnen filteren, maar dat is uiteindelijk niet geïmplementeerd in de frontend.

    - https://www.example.com/api/api-keys?filter[userGroup]=userGroup

- Filter tenantId
    - Boven de lijst van API-keys staat er op de zoekpagina een tenant filter. Deze gaat uiteraard filteren op tenants, als er daar een tenant is gekozen gaat deze filter worden ingevuld om de API-keys lijst te filteren op die tenantId.

    - https://www.example.com/api/api-keys?filter[tenantId]=1

- Filter userId
    - Boven de lijst van API-keys staat er op de overzichtspagina een user filter. Deze gaat uiteraard filteren op users, als er daar een user is gekozen gaat deze filter worden ingevuld om de API-keys lijst te filteren op die userId.

    - https://www.example.com/api/api-keys?filter[userId]=1

- Filter apiKeyId
    - Deze filter filtert de API-keys op de id. Dit is nooit geïmplementeerd in de frontend.

    - https://www.example.com/api/api-keys?filter[apiKeyId]=1

- Filter apiKey
    - Boven de lijst van API-keys staat er op de overzichtspagina een API-key filter. Deze gaat uiteraard filteren op API-keys, als er daar een API-key wordt ingegeven gaat deze filter worden ingevuld om de API-keys lijst te filteren op die API-key.

    - Deze filter verschilt van Filter apiKeyId. Deze filter filtert op de GUID (Globally unique identifier) wat uiteindelijk ook de API-key is.

    - https://www.example.com/api/api-keys?filter[apiKey]= XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

3.1.12.4 Example output


```
{
    "page": {
      "size": 10,
      "number": 2,
      "totalItems": 2,
      "totalPages": 2
    },
    "items": [
      {
        "id": 1,
        "apiKey": "apiKeyHere"
        "enabled": true,
        "createdOn": "2021-04-01 06:33:18.723",
        "modifiedOn": "2021-04-01 06:33:18.723",
        "userTenantId": 1,
        "userGroup": "UserGroupHere",
        "isCompanyKey": true,
        "userName": "UserNameHere",
        "userId": 1,
        "tenantName": "tenantNameHere",
        "tenantId": 1,
        "workspaceId": 1,
        "workspaceName": "WorkspaceNameHere",
        "companyName": "CompanyNameHere",
        "companyId": 1
      }
    ]
  }
```

3.1.13 PUT API-keys (enable by Id)

Deze PUT wordt gebruikt om API-keys te enabelen met behulp van de id. Aan clientzijde wordt dit gedaan door een checkbox "enabled" in de API-keys lijst.

Deze PUT gebruikt enkel de apiKeyId als parameter in de URL om een API-key te enabelen.

PUT https://www.example.com/api/api-keys/{id}/enable

3.1.13.1 Checks

Om iets aan te maken moeten er natuurlijk heel wat checks zijn om zeker te zijn dat alles goed verloopt!

- Check if apiKeyId exists
  - De gegeven apiKeyId moet natuurlijk wel bestaan, dus om errors te voorkomen wordt eerst gecheckt of deze wellicht bestaat. Zoniet, geeft deze een custom ApiKeyNotFoundError terug.

3.1.13.2 Example output

```
{
    "id": 1,
    "apiKey": "apiKeyHere"
    "enabled": true,
    "createdOn": "2021-04-01 06:33:18.723",
    "modifiedOn": "2021-04-01 06:33:18.723",
    "userTenantId": 1,
    "userGroup": "userGroupHere"
  }
```

3.1.14 PUT API-keys (disable by Id)

Deze PUT is hetzelfde als PUT API-keys (enable by Id) maar in de plaats van enable, disabled dit de API-key

Dit gebruikt uiteraard ook dezelfde Checks als hierboven.

PUT https://www.example.com/api/api-keys/{id}/disable

3.1.15 POST API-key

Deze POST wordt uiteraard gebruikt om een nieuwe API-key aan te maken. Specifieker op de frontend bij de create API-key. Deze is in frontend net niet geïmplementeerd kunnen worden. Normaal zou er met de info die meegegeven is in de selectboxes een API-key kunnen aangemaakt worden (userId, tenantId, companyId, workspaceId (optioneel) en userGroup (optioneel))

De workspaceId en userGroup zijn optioneel hier. Als de workspaceId null is of wordt leeggelaten wordt er een companywide key aangemaakt. Anders een workspace specific key. De userGroup wordt momenteel niet echt gebruikt maar is hier voor de toekomst.

Zoals uitgelegd in POST (dummy) User is het aanmaken van een dummy user in deze endpoint geïntegreerd. Om een bestaande user te koppelen aan de API-key moet een userId worden meegegeven, om een nieuwe dummy user te koppelen moet er een userName worden meegegeven. Als (geen) beide zouden aanwezig zijn wordt er een custom UserNotSpecifiedError gegeven.

Deze endpoint maakt ook userTenants aan als deze nog niet bestaan met de user en tenant combinatie. Hiernaast wordt de workspace aan de userTenant toegewezen en dan nadat de API-key is aangemaakt wordt er nog een role toegewezen aan de API-key.

3.1.15.1 Checks

Om iets aan te maken moeten er natuurlijk heel wat checks zijn om zeker te zijn dat alles goed verloopt!

- Check parameters
  - De gegeven id's en names MOETEN een mailadres zijn. Dit is gemakkelijk te doen met de library class-validator.
    - De userId is optioneel (omdat het kan zijn dat er een user moet aangemaakt worden en dan wordt alleen de userName meegegeven) en wordt nagekeken dat het een Nummer is.
    - Zoals gezegd in POST (dummy) User de userName wordt nagekeken of het een mailadres is en of het een string is. Deze is natuurlijk ook optioneel.
    - De tenantId is VERPLICHT en wordt nagekeken of het een nummer is.
    - De companyId is VERPLICHT en wordt nagekeken of het een nummer is.
    - De workspaceId is optioneel en wordt nagekeken of het een nummer is.
    - De userGroup is optioneel en wordt nagekeken of het een string is.

- Check userName and userId are specified
  - Zoals uitgelegd hierboven mogen beide niet aanwezig zijn. Als beide aanwezig zijn wordt er een custom UserNotSpecifiedError teruggestuurd.
  - Als er geen van beide aanwezig zijn, wordt dezelfde UserNotSpeciefiedError teruggestuurd.

- Check if userId exists
  - De meegegeven user moet weldegelijk bestaan. Als deze niet bestaat wordt er een custom UserNotFoundError teruggestuurd.

- Check if username exists
  - Lees POST (dummy) User Checks

- Check if tenantId exists
  - De meegegeven tenant moet weldegelijk bestaan. Als deze niet bestaat wordt er een custom TenantNotFoundError teruggestuurd.

- Check if workspace/company are synced
  - Dit is een check die gaat nakijken of de company en workspace weldegelijk bestaan binnen die tenant.

- Check if API-key already exists
  - Als er een API-key al is met de meegegeven parameters moet er natuurlijk geen nieuwe worden aangemaakt. Als dit gebeurt wordt er een custom ApiKeyAlreadyExistsError teruggestuurd.

3.1.15.2 Example request

```
{
    "userId": 1,
    "userName": "example@example.com",
    "tenantId": 1,
    "companyId": 1,
    "workspaceId": 1,
    "userGroup": null
  }
```

3.1.15.3 Example output

```
{
    "id": 1,
    "apiKey": "ApiKeyHere",
    "enabled": true,
    "createdOn": "2021-04-01 06:33:18.723",
    "modifiedOn": "2021-04-01 06:33:18.723",
    "userTenantId": 1,
    "userGroup": "UserGroupHere"
  }
```

3.2 Frontend

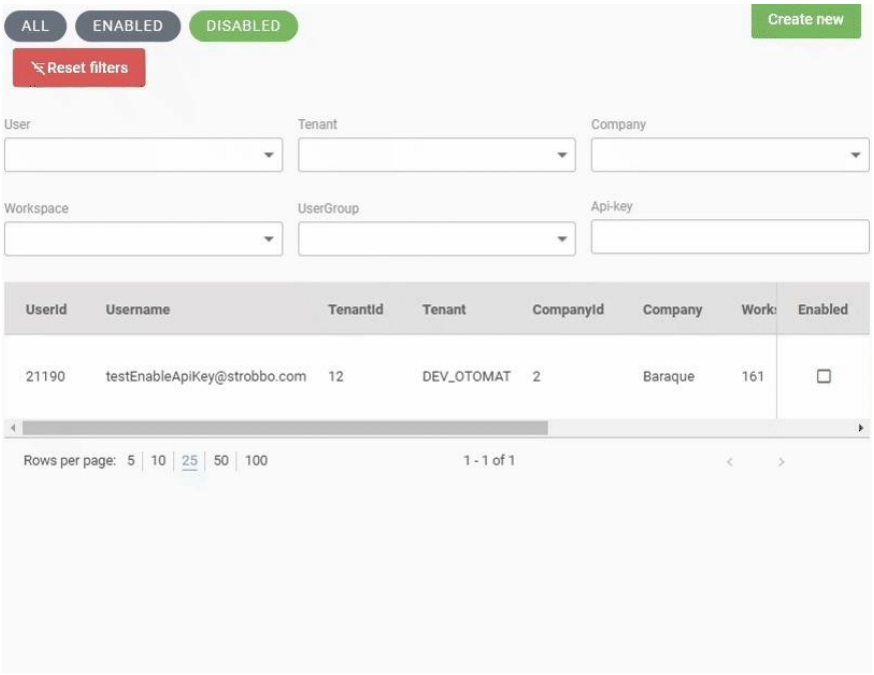
Toen ik me begon in te werken in de frontend hadden we nog maar een maand en half stage over. Ik heb me dus eerst en vooral moeten inwerken in Vue.js, de Typescript taal was al niet meer zo nieuw voor mij aangezien ik daarmee had gewerkt in het backend luik van deze stage. Al snel ben ik begonnen met een kleine issue om me te verdiepen in de frontend.

3.2.1 Frontend realisatie

Het meeste frontend werk is opgenomen door mijn mede-stagiaire Jana. Mijn issue's op de frontend waren meestal kleinere bugfixes of features.

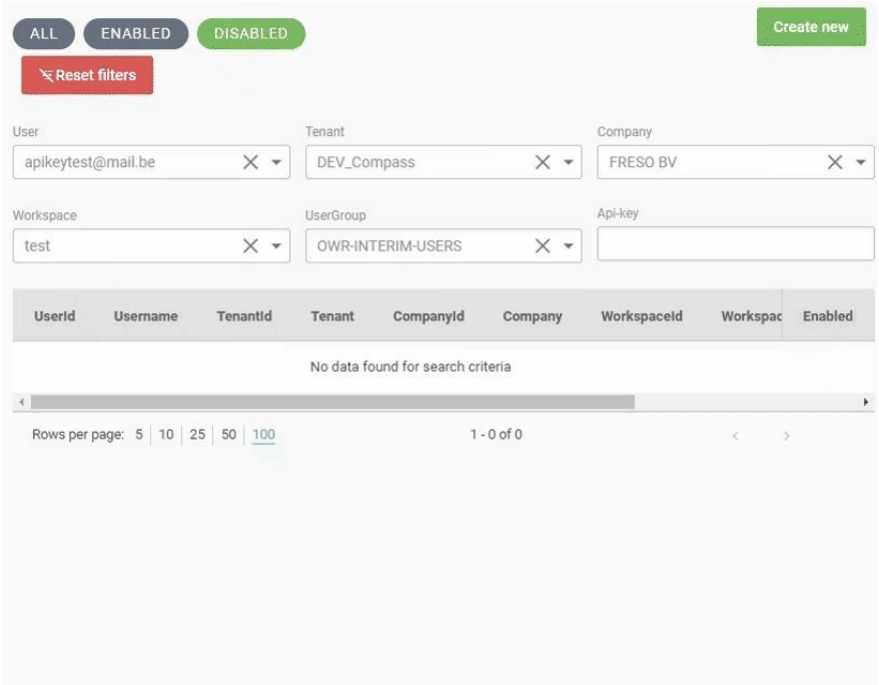
3.2.2 Disabled filter API-keys

Eerst stonden hier alleen een ALL en ENABLED-knop. Mijn opdracht hier was ook om een DISABLED-knop toe te voegen om ook de API-keys te kunnen filteren op disabled API-keys. Dit ging moeilijker als verwacht. Hierachter was eerst een simpele Boolean om ALL of ENABLED aan te duiden, dit heb ik moeten veranderen naar een ENUM (Enumerate) die dan uiteindelijk met een omvorming de juiste request moest sturen naar de backend.



3.2.3 Reset filters button

Natuurlijk is het niet gemakkelijk als elke filter een voor een moet weggeklikt worden. Hiervoor heb ik een reset filters button gemaakt. Deze gaat zoals het zegt, alle filters resetten.



3.2.4 User, Tenant filter new endpoint

Eerst stonden deze filters op de normale /users en /tenants endpoint waar ALLE users en tenants worden teruggestuurd. De opdracht was hier om die naar de toen nieuwgemaakte /api-keys/users en /api-keys/tenants endpoint te zetten waar alleen de users en tenants die aan een API-key vasthangen worden teruggestuurd.

3.2.5 Capitalize labels

Boven de filter selectboxes staan labels. Sommige van deze labels waren in lowercase en dan andere normaal met een hoofdletter. Hier heb ik dan de labels een hoofdletter gegeven waardoor ze consistent werden.

3.2.6 API-key filter error when empty

Toen Jana een nieuwe filter API-key had gemaakt voor op de GUIDs van een API-key te kunnen filteren werd er een probleem al vrij snel duidelijk. Wanneer de textbox terug leeg gemaakt werd deze rood en gaf een error dat de API-key niet valide is. Ik heb dit opgelost door te zorgen dat de lege textbox geen error kan geven.

3.2.7 Disable autocomplete on filters

Dit was een probleem gevonden tijdens dat iedereen tijdens een sprint de pagina aan het testen was. Blijkbaar stond autocomplete nooit uit dus kwamen hier suggesties van chrome of edge zelf in. Dit was gemakkelijk opgelost door een simpele `autocomplete="off"`

3.2.8 API-keys filter twice

Als er een filter werd toegevoegd werden er 2 requests uitgestuurd en refreshte de lijst ook 2 keer wat natuurlijk te veel is. Dit heb ik opgelost door de logica achter het selecteren te herwerken.

3.2.9 Create API-key mandatory fields

Bij de create modal van API-key zijn er heel wat verplichte velden zoals User, tenant en company. Als deze niet worden ingevuld moeten deze een error geven en duidelijk aanduiden waar de error zit. Hiervoor heb ik de zelfgemaakte component van Jana moeten aanpassen met een error klasse.