THOMAS MORE

Pension Architects Export Grids

Plan van aanpak

Sebastiaan Daniëls
Student Bachelor in de Toegepaste Informatica – Applicatieontwikkeling

2024 - 2025

Inhoudsopgave

1. INLEIDING 4

2. HET PENSION ARCHITECTS ADMINPANEEL 5

3. PROBLEEMSTELLING 6

4. HUIDIGE WERKING 7

5. NIEUWE WERKING 8

6. VOORGESTELDE FLOW VAN HET PROJECT 10

6.1. Adminpanel toont een grid 10

6.2. Gebruiker drukt op de downloadknop 12

6.3. Adminpanel scheduled een job genaamd ‘export-grids’ en geeft alle argumenten van het grid mee. 13

6.4. De job-scheduler start export-grids. 15

6.5. Export-grids haalt alle argumenten op uit de job-scheduler database. 16

6.6. Export-grids maakt een excel file op basis van de argumenten die gegeven zijn. 16

6.7. De dispatcher zorgt ervoor dat de file bij de user terecht komt. 16

7. CONCRETE TIJDLIJN 17

8. BESLUIT 18

LITERATUURLIJST 19

1. Inleiding

"Pension Architects is gespecialiseerd in het administratief en actuarieel beheer van pensioenplannen in het kader van groepsverzekeringen en pensioenfondsen. Vanuit onze vier kernwaarden – innovatie door technologie, continuïteit, betrouwbaarheid en ethisch handelen – zijn we erin geslaagd een beheersysteem te ontwikkelen dat de grootste pensioenplannen moeiteloos kan beheren. Met name grote plannen kunnen hun (financieel) voordeel halen uit onze innovatieve oplossingen.

Door onze jarenlange ervaring begrijpen we de processen ten gronde, en met behulp van doorgedreven automatisatie en digitalisering, hebben we een modulair geïntegreerd platform ontwikkeld. Elke dag opnieuw slagen we erin om de processen bij onze klanten doelgericht en op een kostenefficiënte wijze te organiseren." (Pension Architects, 2025)

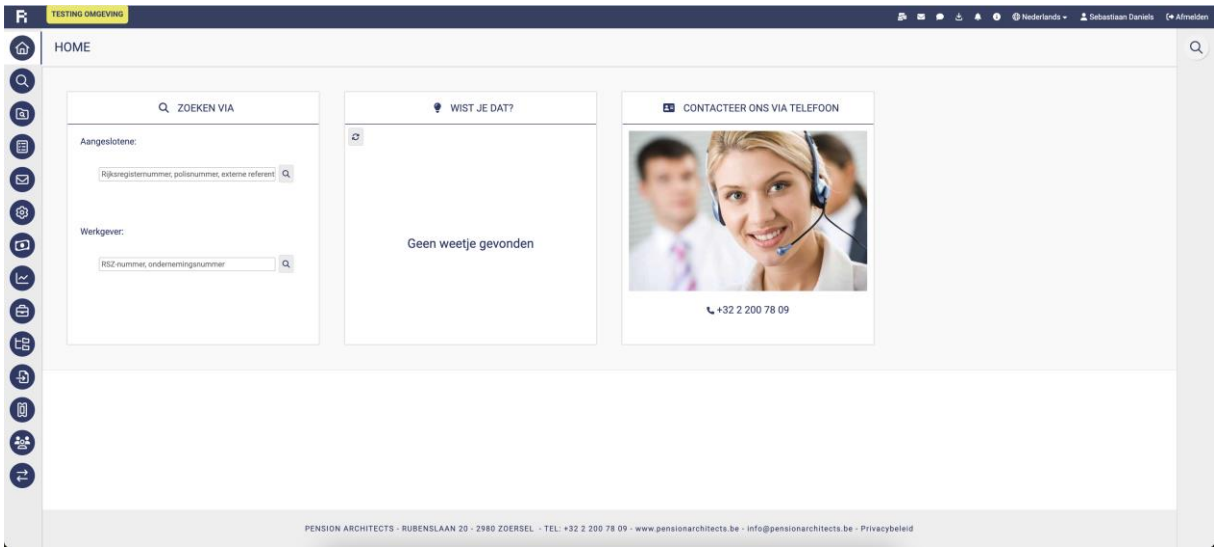
Ik doe mijn stage in 2025 bij Pension Architects. Dit is een bedrijf dat gespecialiseerd is in het beheer van pensioenen. Zij hebben ook een admin paneel ontwikkeld waar klanten de pensioenen van hun werknemers kunnen beheren. Ze beheren de pensioenen van verschillende sectoren zoals de metaalsector, bediendesector, chemiesector, … Ook werken ze nauw samen met het Vlaams Pensioen Fonds.

Er werken op heden ongeveer een 15-tal full time developers aan het ontwikkelen en onderhouden van het adminpaneel en achterliggende infrastructuur. Ik zal hen de komende 13 weken het team ondersteunen door een project te ontwikkelen binnenin het adminpaneel.

2. Het Pension Architects adminpaneel

Het PA (Pension Architects) adminpaneel is een paneel waar klanten pensioenplannen kunnen beheren. Het is geschreven in Java met behulp van het Tapestry framework. Het wordt gehost op AWS. Het adminpaneel werkt met verschillende zogenaamde “componenten”.

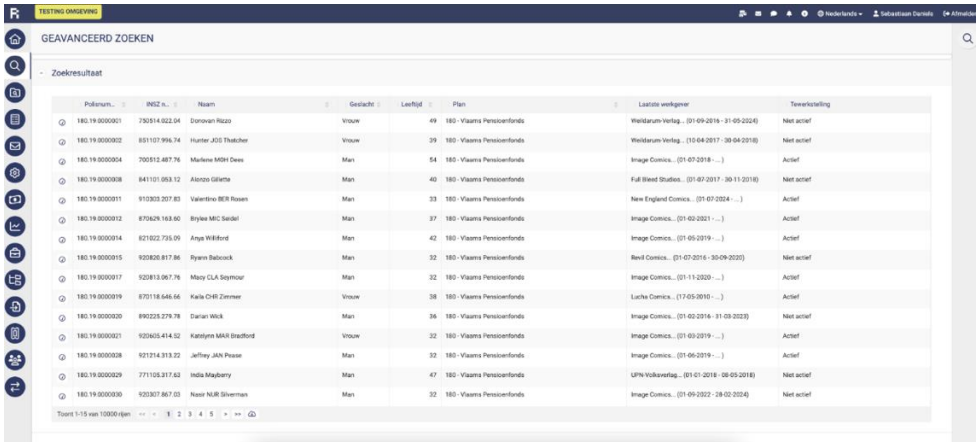
Elk component is verantwoordelijk voor een deel van de applicatie. Bv; *adminpanel (*de front-end die de gebruikers zien*), admindb* (De database), *admindb-service-layer* (de laag die communicatie tussen front- en backend mogelijk maakt), *components* (componenten die gebruikt worden in de front-end; gelijkaardig aan bv Angular of React componenten), *export-grids (*een tool om grids te exporteren, dit is tevens ook de stageopdracht*).*



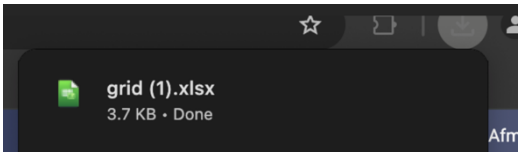
Op dit dashboard kunnen klanten zo goed als alles doen wat met pensioenen te maken heeft. Het is enorm groot, complex, en de meeste componenten zijn buiten de scope van deze stageopdracht. Je kan bv. O.a.: Brieven sturen, uitbetalingen doen, pensioenplannen maken en beheren, etc...

3. Probleemstelling

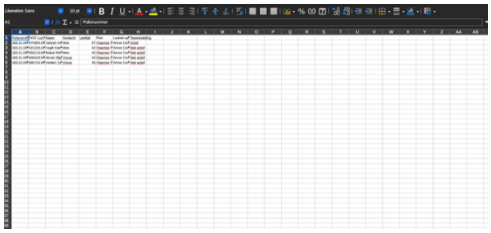
In het adminpaneel zijn er verschillende “grids” die met behulp van verschillende functies, filters, … kunnen opgevraagd worden. Deze grids kunnen ook op verschillende pagina's staan.



Onder dit grid vinden we een pagination en een download-knop. Wanneer men op de download knop drukt, wordt de grid precies zoals weergegeven gedownload in een Excel bestand.



Echter, bij grote grids met meer dan een paar duizend rijen crasht het adminpaneel (En dat voor iedereen). Dit omdat de *webserver* dit bestand probeert te genereren, maar niet genoeg kracht heeft. Dit is uiteraard een worst-case scenario aangezien een eindgebruiker best makkelijk de hele toepassing voor iedereen neer kan halen.

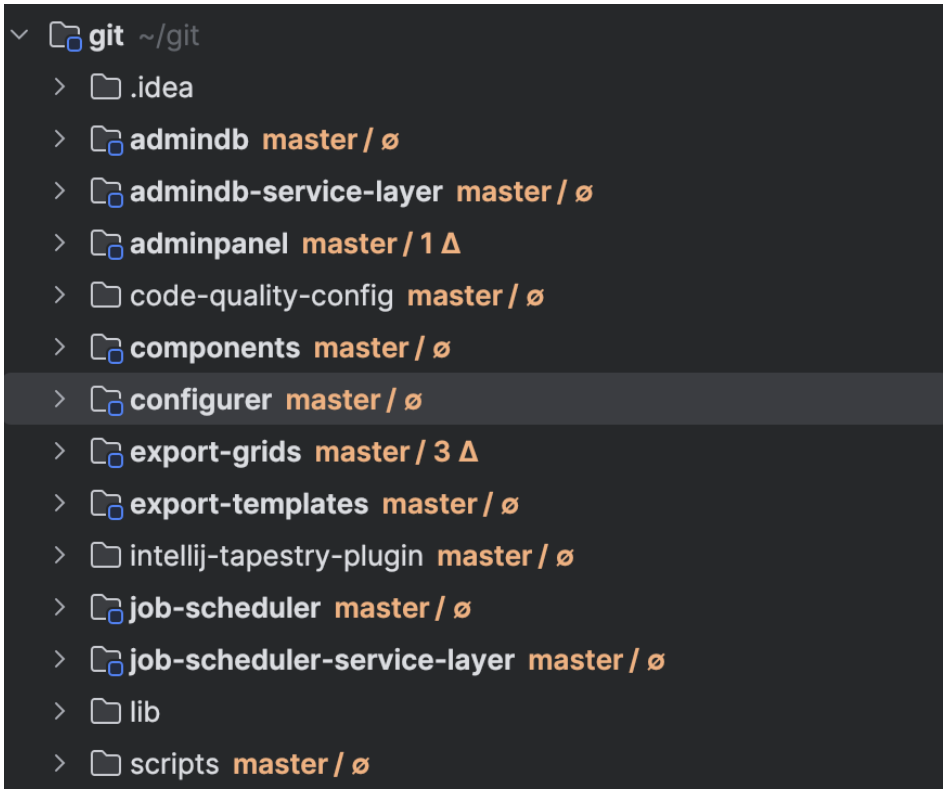


De bedoeling van mijn stage is om een manier te vinden om een nieuwe download functie uit te werken die niet via de *webserver* werkt, maar via een job scheduler die dit dan asynchroon. Hierdoor wordt de webserver niet belast. Deze scheduled dan een job op een aparte *applicatie* server, Wanneer deze de export gedaan heeft, krijgt de eindgebruiker een melding in de “meldingen” pagina om deze te downloaden.

4. Huidige werking

Op dit moment kunnen gebruikers grids bekijken op een pagina. Wanneer een gebruiker deze wil bekijken, gebeurt alles in de front-end in een module genaamd *adminpanel*.

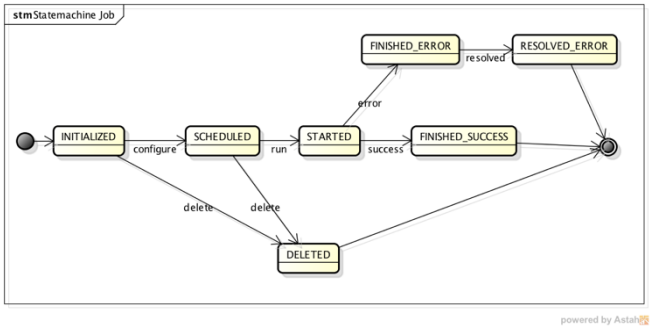
Om deze data op te halen, maakt de adminpanel een paginated *Grid* object. De front-end spreekt daarna een *admindb-service-layer* aan. Deze haalt de eerste 15 resultaten op uit de database en geeft deze terug aan *adminpanel*. Via pagination zal de servicelayer nieuwe data opvragen. Dit om de webserver niet te belastten. Wanneer een eindgebruiker op de download-knop drukt. Zal de *adminpanel* de service layer aanspreken en **alle** data opvragen. Wanneer de webserver de data heeft ontvangen converteert deze de ontvangen dto's naar een Excel die dan door de eindgebruiker gedownload wordt. Echter als het grid te groot is (>10k rijen) kan de webserver de data niet meer aan en gaat hij down. Grote klanten kunnen makkelijk grids te zien krijgen met meer dan 300k rijen. Dit moet dus opgelost worden zodat de eindgebruikers niet de hele server kunnen laten crashen. Momenteel is de download knop uitgezet wanneer er meer dan 5k rijen in het grid zijn.



5. Nieuwe werking

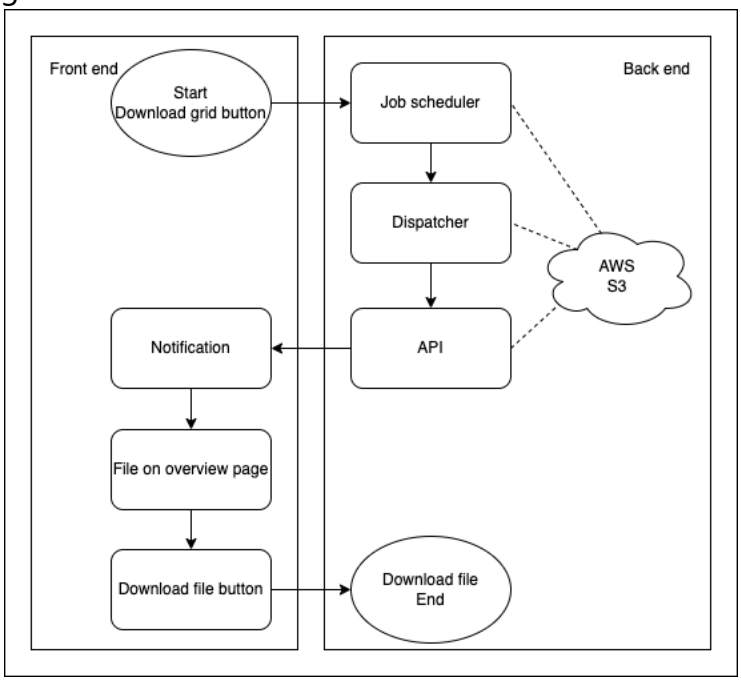
In het nieuwe systeem zal een nieuw component genaamd *export-grids* de effectieve export als backend overnemen. Er heeft al een vorige stagiair een deeltje van de nieuwe werking gemaakt.

PA werkt voor grote backend transacties met een *job-scheduler*. Deze zal jobs asynchroon van de frontend uitvoeren. Het *adminpanel* kan een job aanmaken dat een programma opstart (dit zal in ons geval het programma *export-grids* zijn) met bepaalde parameters. (Bv het id van de grid, username, en filters).

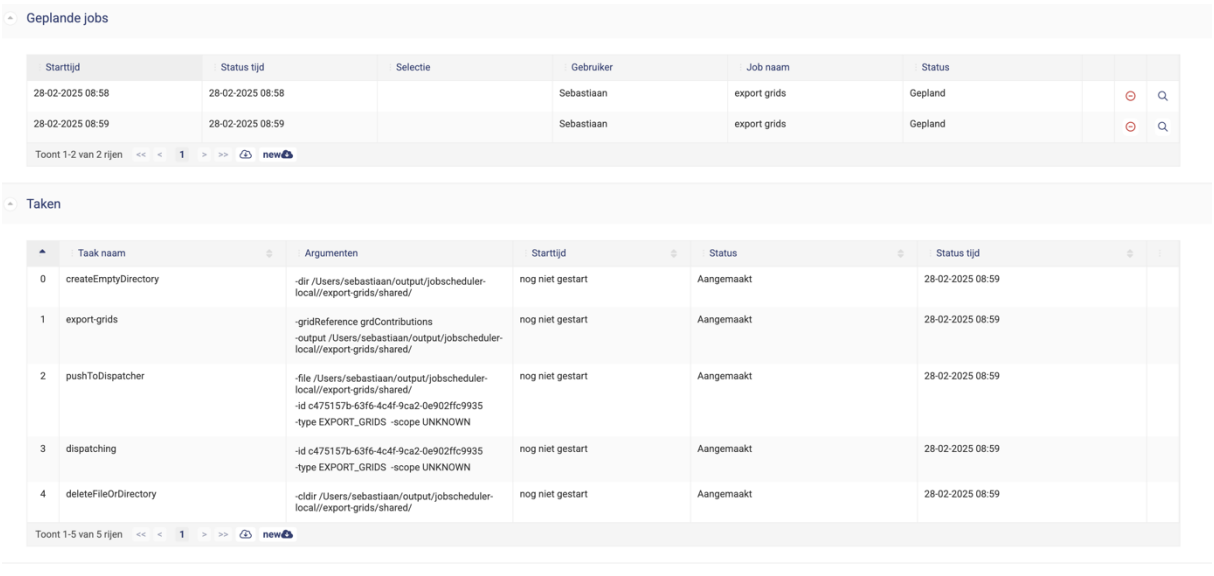


Voor het exporteren van een grid drukt de eindgebruiker op de download knop. De frontend (*adminpanel*) spreekt nu de *job-scheduler* aan met verschillende argumenten:

- GridReference (ID van het grid dat geëxporteerd moet worden)
- Output (folder PATH waar de excel moet opgeslagen worden)
- Andere argumenten



Wanneer de job wordt opgestart zal export grids een grid moeten maken op basis van de argumenten die gegeven zijn.
Wanneer de excel is gemaakt, zal een *dispatcher* het bestand opslaan op de juiste plaats en de eindgebruiker op de hoogte stellen dat zijn/haar download beschikbaar is d.m.v een notificatie.
De implementatie van het schedulen van een job, en het dispatchen van het resultaat zijn deels gedaan door de vorige stagiair. Het is nu aan mij om de juiste argumenten mee te geven, de grids te bouwen in de applicatie server in plaats van de webserver, en te exporteren. Dit moet uiteraard zeer goed secured zijn zodat enkel geauthorizeerde users een grid kunnen exporteren (Deze grids kunnen *zéér* gevoelige data bevatten).



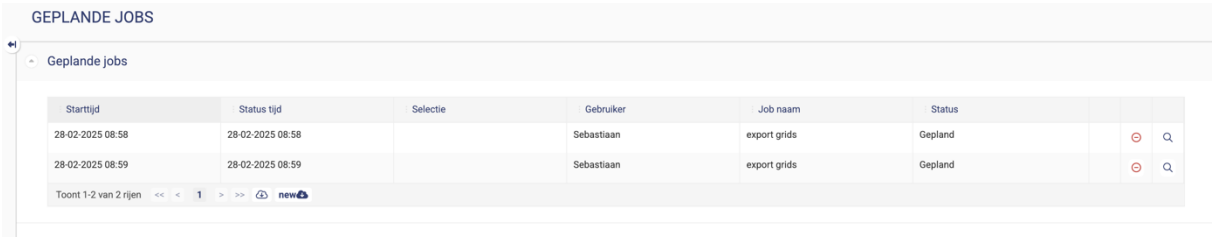
6. Voorgestelde flow van het project

Dit is het voorstel dat ik maak voor de nieuwe implementatie van export grids. Ik zal per deel ook toelichten hoe het nu werkt of zal geïmplementeerd worden.

1. *Adminpanel* toont een grid
2. Gebruiker drukt op de downloadknop
3. *Adminpanel* scheduled een job genaamd '*export-grids*' en geeft alle argumenten van het grid mee.
4. De *job-scheduler* start *export-grids*.
5. *Export-grids* haalt alle argumenten op uit de job-scheduler database.
6. *Export-grids* maakt een excel file op basis van de argumenten die gegeven zijn.
7. *Export-grids* slaat de file op.
8. De *dispatcher* zorgt ervoor dat de file bij de user terecht komt

6.1. Adminpanel toont een grid

Wanneer een gebruiker een Grid wil bekijken, gebeurt er van alles in de achtergrond. Bv bij het grid "*jobscheduleroverview*"



Dit grid wordt gebouwd met volgende code:

In `jobs.tml` staat de html code om het grid op te bouwen met alle colomnamen, in de achterliggende file `jobs.java` staat de code die het mogelijk maakt om de grids te tonen.

Hierin wordt de code aangeroepen om een grid te bouwen;

In `jobs.tml:`

```
 1. <t:pac.datagrid t:id="grdJobs"
 2.     t:gridModel="jobsGridModel" t:rowsPerPage="10"
 3.     t:exclude="prop:excludeColumns"
 4.     t:noResultsMessage="grdJobs.noresults" t:object="jobLoop"
 5.
t:colNonSortable="remove,details,reschedule,jobName,scheduleTime,selection,state,stateTime,title,
userName"
 6.     t:colNonHidable="remove,details,reschedule"
 7.     t:colWidths="remove:40, details:40, reschedule:40"
 8.     t:colHozAligns="remove:center, details:center, reschedule:center"
 9.     t:value="jobGridValue" t:autoRefresh="30000">
10.
```

`t:gridModel="jobsGridModel"` is het argument dat aangeeft welk *gridModel* te gebruiken.

in `jobs.java`

```
1. public GridModel<JobDto> getJobsGridModel() {
2.     JobSearchFilter jobFilter = new JobSearchFilter();
3.     jobFilter.addState(JobState.FINISHED_ERROR);
4.     jobFilter.addState(JobState.STARTED);
5.     jobFilter.addState(JobState.SCHEDULED);
6.     jobFilter.addState(JobState.ENQUEUED);
7.     return gridModelSource.createJobsModel(jobFilter, false, resources, "grdJobs.");
8. }
9.
```

De *JobSearchFilter* is de filter waarin wordt aangegeven waarop moet gefilterd worden.

De *GridModelSource* is een interface (`JobSchedulerGridModelSource.java`) die geïmplementeert wordt door `JobSchedulerGridModelSourceImpl.java`.

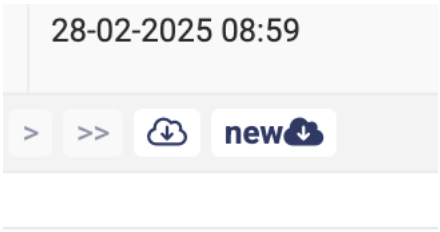
Hierin zit de geïmplementeerde functie *CreateJobsModel:*

```
 1. public GridModel<JobDto> createJobsModel(
 2.         JobSearchFilter filter, boolean sortDescending, ComponentResources resources, String
labelPrefix) {
 3.     QueryPagedGridModelImpl<JobDto> model = new QueryPagedGridModelImpl<>(
 4.           JobDto.class,
 5.           resources.getMessages(),
 6.           labelPrefix,
 7.           dataTypeAnalyzer,
 8.           beanModelSource,
 9.           propertyConduitSource,
10.           jobService.find(filter));
11.     model.addProperty(JOB_SCHEDULE_TIME, JOB_SCHEDULE_TIME);
12.     // Nog vanalle andere .addProperty's
13.     model.addSortProperty(JOB_SCHEDULE_TIME, JobDto.SortableFields.SCHEDULE_TIME);
14.     model.includeProperties(
15.           JOB_SCHEDULE_TIME, "jobName", JOB_SELECTION, JOB_USER, STATE, STATE_TIME,
RESCHEDULE, REMOVE, DETAILS);
16.     model.reorderProperties(JOB_SCHEDULE_TIME, STATE_TIME, JOB_SELECTION, JOB_USER);
17.     model.setDefaultSortProperty(JOB_SCHEDULE_TIME);
18.     return model;
19. }
20.
```

Deze functie maakt de echte grid en bouwt deze op uit de service layer. Deze wordt dan recursief teruggegeven aan de vorige pagina, en zo gaat dit door tot deze terug in de `jobs.tml` file terecht komt. Deze functies staan *allemaal* in het project **adminpanel.** Dit betekent dus ook dat alles in de front-end op de webserver wordt berekend. Enkel de functie `jobService.find()` is een functie uit de *admindb-service-layer*. Hier wordt de effectieve data opgehaald uit de database.

6.2. Gebruiker drukt op de downloadknop

Op het grid staan twee knoppen om het grid te exporteren. De ‘downloadknop’, en de ‘Nieuwe downloadknop’. De oude knop exporteert het grid in de browser. Het is de bedoeling om het grid via de nieuwe knop asynchroon te exporteren in de app server via de *job-scheduler*.



Wanneer de nieuwe downloadknop wordt ingedrukt, wordt de exportfunctie aangesproken.

6.3. Adminpanel scheduled een job genaamd ‘export-grids’ en geeft alle argumenten van het grid mee.

Eerst hebben we in `jobs.tml` een *pac.datagrid*:

```
 1. <t:pac.datagrid t:id="grdJobs"
 2.     t:gridModel="jobsGridModel" t:rowsPerPage="10"
 3.     t:exclude="prop:excludeColumns"
 4.     t:noResultsMessage="grdJobs.noresults" t:object="jobLoop"
 5.
t:colNonSortable="remove,details,reschedule,jobName,scheduleTime,selection,state,stateTime,title,
userName"
 6.     t:colNonHidable="remove,details,reschedule"
 7.     t:colWidths="remove:40, details:40, reschedule:40"
 8.     t:colHozAligns="remove:center, details:center, reschedule:center"
 9.     t:value="jobGridValue" t:autoRefresh="30000">
10.
```

In de achterliggende code van dat grid hebben we een export en ExportAsync functie:

In `DataGrid.java`:

```
 1. @OnEvent(EVENT_EXPORT)
 2. StreamResponse exportGrid(EventContext eventContext) {
 3.     processEventContext(eventContext, true);
 4.     return export(processedSortName, processedSortOrder);
 5. }
 6. @OnEvent(EVENT_EXPORT_ASYNC)
 7. void exportGridAsync(EventContext eventContext) {
 8.     processEventContext(eventContext, false);
 9.     String gridReference = processedClientId;
10.     GridAsyncExporter asyncExporter = gridAsyncExporterSource.getExporter()
11.           .orElseThrow(() -> new IllegalStateException("No async grid exporter found"));
12.     asyncExporter.scheduleExport(gridReference);
13.     rvlJobConfirmation.show("rvlJobConfirmation");
14. }
15.
```

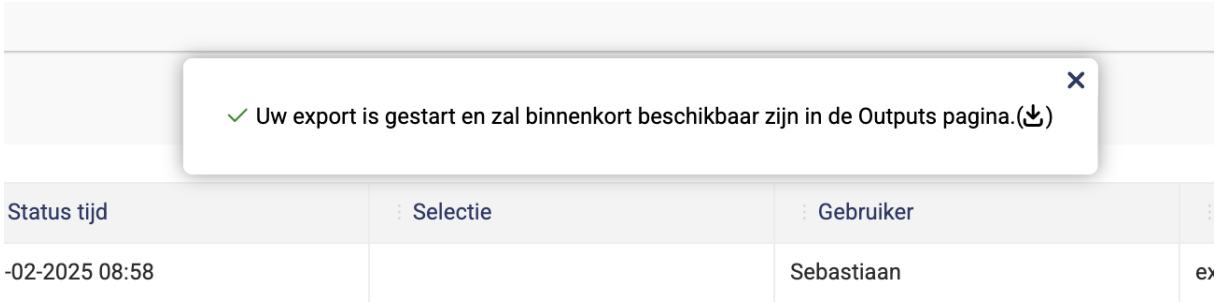
De functie *exportGrid* gaat door naar een andere klasse die het grid in de browser probeert te exporteren. Dit is de oude implementatie en deze code wordt uitgedoofd.

De functie exportGridAsync zal de nieuwe implementatie worden om grids asynchroon te exporteren.

Lijn 9: De *gridReference* is eigenlijk het ID van de grid. Hiermee kunnen we in *export-grids* zien welk Grid we moeten ophalen

Lijn 10 - 12: Dit maakt een object aan van het klasse *GridAsyncExporter* en scheduled een export samen met het referentie ID

Lijn 13: Dit toont een pop-up aan de eindgebruiker om hun te vertellen dat de job is gestart.



De *GridAsyncExporter* is een interface die geïmplementeerd wordt door `AdminpanelBaseModule.java`. Hierin staat de volgende functie dit de interface gebruikt:

```
1. @Contribute(GridAsyncExporterSource.class)
2. public static void contributeGridAsyncExporterSource(
3.         Configuration<GridAsyncExporter> config,
4.         JobSchedulingFrontService jobSchedulingService) {
5.     // This is disabled until we find time to finish the async grid exports project started by
a student
6.     config.add(jobSchedulingService::createGridExportJob);
7. }
8.
```

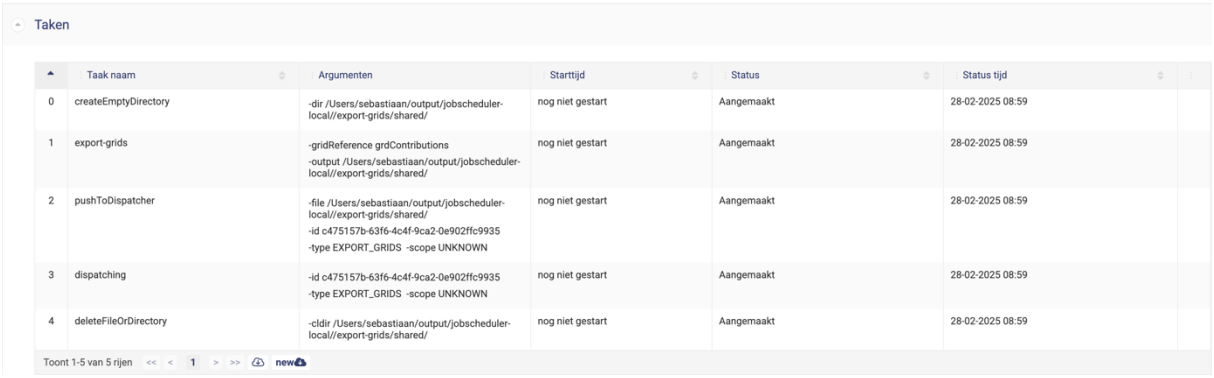
In deze functie wordt de configuratie van de job meegegeven aan *createGridExportJob*. Deze service zal de job effectief schedulen.

In `JobSchedulingServiceImpl.java`:

```
 1. public int createGridExportJob(String gridReference) {
 2.     String outputDir = getOutputFolder(TaskNames.EXPORT_GRIDS.getName(), FOLDER_SHARED);
 3.     String batchId = UUID.randomUUID().toString();
 4.     JobBuilder jobBuilder = new JobBuilder(JobNames.EXPORT_GRIDS)
 5.           .withOutputDirectory(outputDir)
 6.           .withTask(createExportGridTask(gridReference, outputDir))
 7.           .withTask(createPushToDispatcherTask(outputDir, EXPORT_GRIDS, SELECTION_UNKNOWN,
batchId))
 8.           .withTask(createDispatchingTask(batchId, EXPORT_GRIDS, SELECTION_UNKNOWN));
 9.     Job job = jobRepository.save(jobBuilder.build());
10.     jobSessionManager.commit();
11.     return job.getId();
12. }
13.
```

De functie `createGridExportJob` zal nu de job aanmaken. Hij geeft de verschillende taken weer in de *JobBuilder* en voegt de argumenten toe. Hij slaat de job op en "commit" deze.

Het resultaat kunnen we terugvinden in de *jobs* pagina op het admin dashboard.



De job is nu aangemaakt en zal meegenomen worden in de job queue.
Merk op dat we op dit moment enkel de output folder en een gridReference meegeven. Dit is niet genoeg informatie en de extra data die zal moeten meegegeven worden zal ik moeten uitwerken in het project. Ik zal een manier moeten vinden waarop ik dezelfde argumenten die worden gebruikt om een grid op te bouwen kan doorgeven aan de job scheduler. Bv. als json:

```
-gridReference grdContributions
-output /Users/sebastiaan/output/jobscheduler-local//export-grids/shared/
-gridArgs {"arg1":"value","arg2":"value"} // op een manier alle filters naar json serializen
```

6.4. De job-scheduler start export-grids.

Dit gebeurt automatisch wanneer de job-scheduler al zijn vorige taken afgewerkt heeft. Op dit moment voert deze synchroon alle taken één voor één uit.

Het is de bedoeling dat de job-scheduler binnenkort asynchroon taken kan uitvoeren zodat de eindgebruikers niet op elkaar moeten wachten bij meerdere gelijklopende jobs. Dit is echter buiten de scope van mijn stageopdracht.

6.5. Export-grids haalt alle argumenten op uit de job-scheduler database.

Het programma *export-grids* zal het effectieve programma worden dat de grids maakt op de applicatie server. Allereerst moeten we weten *welk* grid we moeten opbouwen. Daarvoor zullen we de argumenten moeten ophalen die waren meegegeven aan de *job-scheduler*. Deze data zal op een specifieke manier encoded zijn, dus die moeten we eerst decoderen naar de juiste parameters, filters; etc.

6.6. Export-grids maakt een excel file op basis van de argumenten die gegeven zijn.

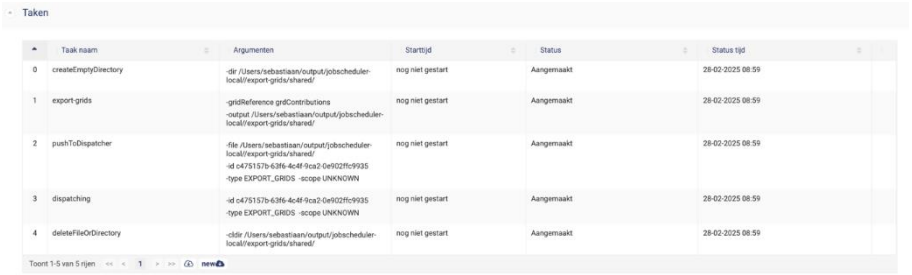
Wanneer we de juiste parameters hebben, kunnen we beginnen met de excel op te bouwen. Dit zal het grootste en moeilijkste deel worden. Dit omdat alle functies en modules om grids op te bouwen enkel bestaan in het *adminpanel* component, en dat willen we juist vermijden. Daarom zal ik proberen de functionaliteiten van het opbouwen van grids proberen te abstraheren naar *export-grids*.

Eenmaal we hetzelfde model hebben opgebouwd als in de frontend getoond werd, kunnen we het proberen weg te schrijven naar een excel file.

Export-grids slaat de file op. Wanneer de excel file klaar is moeten we ervoor zorgen dat deze bij de eindgebruiker terecht komt. Momenteel werken we nog op de applicatieserver. Als we even terugblikken op de *job-scheduler* merken we dat we een output folder meegeven. We slaan de excel file eerst lokaal op de applicatieserver op.

6.7. De dispatcher zorgt ervoor dat de file bij de user terecht komt.

Nadat de file op de *applicatieserver* is opgeslagen willen we deze bij de eindgebruiker brengen. Aangezien deze niet aan de appserver kan, zullen we via een *file dispatcher* het bestand op de *webserver* zetten zodat de eindgebruiker hier aan kan. Deze dispatcher zal automatisch ook een melding naar de gebruiker sturen op het dashboard dat het bestand klaar staat om te downloaden. Op het einde verwijderen we nog de tijdelijke folder op de appserver. Het grid is nu succesvol geëxporteerd.



7. Concrete tijdlijn

WEEK	DOEL
WEEK 1: 24/02 - 28/02	Introductie bij Pension Architects, codebase opzetten
WEEK 2: 03/03 - 07/03	Plan van aanpak maken
WEEK 3: 10/03 - 14/03	PvA voorstellen & aanpassen + beginnen met experimenteren
WEEK 4: 17/03 - 21/03	Encoding systeem maken voor grid options - Job scheduler + meeting op school
WEEK 5: 24/03 - 28/03	Decoding systeem en experimenteren in service-layer
WEEK 6: 31/03 - 04/04	Intermediate evaluatie + grid opbouwen uit options (1 grid)
WEEK 7: 07/04 - 11/04	Grid opbouwen uit options (1 grid)
WEEK 8: 14/04 - 18/04	2de meeting op school + alle grids kunnen opbouwen
WEEK 9: 21/04 - 25/04	Grids exporteren naar Excel file + dispatchen
WEEK 10: 28/04 - 02/05	Bugfixes
WEEK 11: 05/05 - 09/05	“Nice to haves” implementeren/ bufferweek
WEEK 12: 12/05 - 16/05	Realisatiedocument, samenvatting, reflectie stage maken & uploaden
WEEK 13: 19/05 - 23/05	ev. bugfixes/ andere zaken oplossen, project afronden

8. Besluit

In dit document hebben we het plan van aanpak besproken voor het export-grids project bij Pension Architects. Het project kan opgedeeld worden in drie grote subcategorieën.

Allereerst zullen we de argumenten die belangrijk zijn voor het opbouwen van een grid moeten kunnen doorgeven aan de scheduler. Dit zou in principe niet heel moeilijk moet zijn. De grootste zorg is hoe we deze data juist kunnen *encoden* en *decoden*, aangezien we enkel strings mogen meegeven aan de job scheduler.

Als volgt komen we bij het programma *"export-grids"*. Dit zal het grootste deel van de tijd in beslag nemen. Het exporteren van een model naar een Excel bestand is in principe niet zo moeilijk en kan redelijk makkelijk gedaan worden. In deze stap zal het moeilijkste zijn dat het momenteel niet mogelijk is om een grid op te bouwen in de backend. Dit kan nu enkel in de front-end omdat de code in *"adminpanel"* staat. Het zal enorm veel tijd en werk in beslag nemen om deze te abstraheren naar een algemeen grid dat van overal kan opgebouwd worden. Een ander probleem zal de security zijn. In de front-end is er een systeem dat bijhoudt welke user is ingelogd, en welke rechten deze heeft. Op de applicatieserver is dit niet het geval. Hier bestaat enkel een admin user die jobs kan uitvoeren.

De laatste stap zal zijn om het geëxporteerde grid bij de eindgebruiker te krijgen. Dit zou niet zo moeilijk moeten zijn aangezien een *dispatcher* dit voor ons kan doen. Ook heeft de vorige stagiair hier al een deel van gemaakt.

Literatuurlijst

Pension Architects. (2025, Maart 05). *Over Ons*. Opgehaald van Pension Architects: https://www.pensionarchitects.be/over-ons/