Spring ApplicationEvents

Spring ApplicationEvents zijn mechanismen die ervoor zorgen dat componenten niet rechtstreeks van elkaar afhankelijk zijn. Hierdoor worden ze niet rechtstreeks aan elkaar gekoppeld, maar kunnen ze wel met elkaar communiceren. Dit maakt het eenvoudiger om losse componenten te ontwikkelen die samen kunnen worden gebracht, waardoor de code makkelijker aanpasbaar en dus onderhoudbaar is.

Binnen dit mechanisme zijn er drie hoofconcepten. Allereerst het event: dit is een klasse die de actie beschrijft. Dit kan bijvoorbeeld een order zijn die wordt geplaatst of een boot die binnenvaart. Vervolgens is er het concept van de event publisher, die dient om het event te publiceren. Ten slotte hebben we het concept van de event listener; de listener luistert naar een bepaald event en voert vervolgens verdere logica uit. Op deze manier zijn de klasse die de publisher aanroept en de klasse die de listener gebruikt in zijn logica verbonden, maar blijven ze flexibel.

Voordelen

Listeners zijn zeer flexibel; zo kun je meerdere listeners voor hetzelfde event hebben. Hierdoor wordt je code leesbaarder omdat je de verschillende afhandelingen kunt opsplitsen. Ze zijn ook herbruikbaar voor nieuwe events. Dit zorgt uiteraard ook voor een lagere koppeling tussen componenten.

Nadelen

Debuggen kan soms lastiger zijn omdat de afhandeling van events op verschillende plaatsen in de code kan plaatsvinden. Ook kan de verwerkingssnelheid dalen naarmate er meer listeners aan hetzelfde event hangen. Wanneer er veel listeners aan hetzelfde event zijn gekoppeld, is het moeilijk om de volgorde van de afhandeling te bepalen; hier moet dus goed over nagedacht worden.

Toepassen op KdG

Een voorbeeld is het genereren van de commissies op een vertrokke PO nu staat dit mee in processEndOfPurchaseOrderPickup in de PurchaseOrderPickUpService dit heeft daar eigenlijk geen plaats omdat dit qua logica niet veel heeft te maken met het einde van de pickup. Een betere manier zou zijn het einde publishen en die messeging publisher te laten luisteren naar het endOfPurchaseOrderPickupEvent maar ook een listener die het creeren van de commissies afhandelt. Zo zijn deze twee verschillende dingen gescheiden en worden ze beide toch uitgevoerd. Dit zorgt voor een mooiere single responsibility en leesbaardere code. Ook bij het maken van de invoices zou het handiger kunnen zijn dat er een invoiceCreatedEvent gepublished wordt waarbij er dan een listener instaat voor het schrijven naar de databank en een andere die de pdf genereerd.

Samenvattend kunnen we zeggen dat Spring ApplicationEvents de architectuur aanzienlijk kan versterken, vooral wanneer losse koppeling duidelijke voordelen biedt en de principes van single responsibility en modulariteit hiermee kunnen worden verbeterd.