Orbit

Toelichting

Proof of concept toelichtingen

Naar	Luuk de Weijer
	Scrumble
Betreft	Proof of concepts
Auteur(s)	Servi Huijbregts
Email	servi@scrumble.nl

Inleiding en doelstelling

Inleiding

De validatiefase is eigenlijk een verlengstuk van de analysefase. Tijdens deze validatiefase controleer ik, door middel van proof of concepts, of wat ik van plan ben te realiseren op basis van de requirements ook haalbaar is. Hieruit komt een advies richting mij als developer waarin aangegeven staat welke van de initiële requirements haalbaar zijn en welke niet. Bij degene die haalbaar zijn, koppel ik terug naar bewijslast waarom deze haalbaar zijn. Bij degene die niet haalbaar zijn, koppel ik terug naar bewijslast waaruit blijkt dat deze niet haalbaar zijn.

Doelstelling

Ik wil om verschillende requirements te valideren proof of concepts ontwikkelen. De volgende proof of concepts moeten ontwikkeld worden om requirements te valideren:

- Testapplicatie rauwe Redis queries.
- Redis in basisproject Scrumble (Supercharge) implementeren.
- Retry jobs from the front-end.
- Get Redis keys & group by tag.
- Package development.

Testapplicatie rauwe redis queries

Context Om te ontdekken hoe ik met Redis zou kunnen werken in een programmeeromgeving heb ik een proof of concept gemaakt waarbij ik ging controleren of ik items kon opslaan, ophalen en met behulp van parameters op kan halen uit/naar een Redis database. Ik heb de keuze gemaakt om deze proof of concept te ontwikkelen in een taal waar ik al iets meer thuis in was dus het is een Node.js applicatie geworden. Op de video die je in de tijdlijn kunt vinden zie je uitgewerkte code om bovenstaande functionaliteiten te bereiken. In deze video zie je dus de uitwerken van functionaliteiten waarbij je kunt zien dat mijn Node.js applicatie een connectie maakt met de Redis database en gegevens ophaalt.

Resultaten https://drive.google.com/file/d/1lDsi2KFbvRDauwm7Ws_8LjNG4le1jk4W/view?usp=sharing

Conclusie Door het maken van dit proof of concept heb ik bewezen dat ik kan werken met een Redis connectie. Ik heb tevens ontdekt dat je een externe package nodig hebt om een verbinding te leggen met Redis databases.

Redis in baseproject implementeren

Context Om te onderzoeken of ik het Redis kan gebruiken binnen de huidige stack van Scrumble ga ik een proof of concept maken waarbij ik een

aantal Redis functionaliteiten binnen het basisproject van Scrumble (Supercharge) ga implementeren. In de video zie je dat ik een tabel heb met hierin een redisconnectie (Dat is in deze context de prefix van de jobs in de redis database). Vervolgens zie je dat ik RedisInsights open, dit is een soort database monitor voor Redis databases, ik open deze om aan te tonen dat er daadwerkelijk jobs in de database staan. Ik open vervolgens 1 van de redis connecties om daarmee alle jobs die beginnen met deze prefix op te halen door middel van een Redis call die ik in de backend geschreven heb. In de video laat ik zien dat je vanuit de front end een job kan zoeken binnen de tabel met gebruik van front end filtering. Tevens laat ik in deze video zien dat ik door middel van een zelfgeschreven functie de gegevens van een specifieke job kan ophalen en omvormen naar leesbare frontend. Ook zie je bovenin het scherm 3 kaarten met daarop wat statistieken, deze haal ik tevens op met een zelfgeschreven Redis call in de backend. Verderop in de video laat ik zien dat deze kaarten ook elke 10 seconden gerefreshed worden door mijn queue worker aan te zetten die jobs gaat afhandelen.

Resultaten https://drive.google.com/file/d/1kBFiyskmBeQnVlcC43m5jk_qou9zxbVy/view?usp=sharing

Conclusie Door het maken van deze proof of concept heb ik gebruikgemaakt van de Predis package om backend calls naar de Redis database te maken. Ook heb ik geleerd om een queue worker aan te zetten. Verder heb ik geleerd hoe ik door middel van Axios deze backend calls kan maken vanuit de frontend en hoe ik door middel van React useEffect een call op een interval kan zetten. Daarnaast heb ik leren werken met de Scrumble quick table (Dit is een react bootstrap table met betere typing checks en filtering opties)

Retry jobs from front-end

Context Ik heb een proof of concept gemaakt waarbij ik door middel van het gebruik van bestaande Laravel Horizon functionaliteiten jobs opnieuw kan proberen vanuit de front-end wanneer deze gefaald zijn. Dit was een requirement omdat dit in de huidige Laravel Horizon frontend ook mogelijk was, echter wilde de developers niet steeds van frontend wisselen én kunnen ze op de manier zoals getoond in mijn proof of concept dus filteren door middel van de tabel en vervolgens een specifieke gevonden job opnieuw uitvoeren.

Resultaten https://drive.google.com/file/d/1QQUgbDOZqfocxqqzPuNCR0zGlEVzDu-t/view?usp=sharing

Conclusie Ik ben erachter gekomen hoe ik bestaande Laravel Horizon functionaliteiten kan hergebruiken in mijn frontend om zo aan de requirements vanuit Scrumble te voldoen en de user experience beter te maken. Ik kan nu door middel van zelfgeschreven backend jobs ophalen, deze filteren in de frontend en een betreffende job opnieuw starten. Ik kan ook alle jobs met een bepaalde prefix opnieuw starten met een knop die je in de video aan de bovenkant ziet.

Get redis keys & group by tag.

Context Om gebruiksvriendelijker te kunnen filteren door de jobs in de front-end werd het idee geopperd om redis keys op te halen op een nieuwe manier en vervolgens te groeperen op tag. Dit idee komt voort

uit een nieuwe implementatie van redis keys binnen een ander project van Scrumble waarbij de queue naam in de key meegegeven wordt. Mijn opdracht was nu aantonen dat deze key met hoge performance te splitsen was en vervolgens te groeperen. In de video zie je een dataset van 400000 redis keys die binnen 0.7 seconden gegroepeerd naar de frontend gestuurd worden. Ook toon ik in de video een optie om alles binnen deze groep te tonen en een mogelijkheid om op zowel groep als key te filteren in de frontend.

Resultaten https://drive.google.com/file/d/1nAWWTgJHXcKj5kKfEjlzs_stcdMSwORb/view?usp=sharing

Conclusie Het uiteindelijke doel was voornamelijk het sneller maken van de functionaliteit bij grotere datasets. Bij de voorheen uitgevoerde proof of concepts ging de snelheid van het ophalen van Redis keys erg achteruit naarmate er meer keys waren. Vanaf 100000 keys werd het zelfs onmogelijk omdat er dan een timeout optrad. De functie die ik voor dit proof of concept heb geschreven lost dit probleem dus zeker op. De conclusie hier is dus zorgen dat alle applicaties die gebruik gaan maken van Orbit voorzien worden van een abstracte job class zodat ik Redis keys zelf kan opbouwen in plaats van dat het een gewone UUID is, waardoor het ophalen sneller kan en groeperen zeer makkelijk wordt.

Package development

Context

Uit gesprekken met mijn opdrachtgever en resultaten van voorgaande proof of concepts hebben we besloten Orbit als package te realiseren in plaats van gehele applicatie. Hierdoor veranderen er wat zaken, ik moet laten zien dat er een php package gerealiseerd kan worden waarbij gegevens uit de applicatie zelf gehaald worden. Ik moet aantonen dat er React frontend in de package meegestuurd kan worden die ingeladen wordt in de hoofd applicatie. Ook moet ik bestaande Laravel Horizon classes kunnen overschrijven met eigen logica binnen mijn package. In de video zie je een voorbeeld van een in het base project geïmplementeerde package variant van Orbit. De front-end van de package is te bereiken via een admin route, de backend zit ook achter een admin middleware.

Resultaten

https://drive.google.com/file/d/1syoP-aEUH7X4BjF7GcpWdC928uOYxSes/view?usp=sharing

Conclusie

Ik heb ontdekt hoe ik een PHP package kan maken en hoe ik front-end ontwikkel en deze in de moederapplicatie kan gebruiken zonder grote aanpassingen in de applicatie. Ook heb ik ontdekt hoe ik gebruik kan maken van Laravel Horizon functionaliteiten en hoe ik deze kan overschrijven met eigen logica.