Ontwerpen

Binnen de competentie “Ontwerpen” heb ik verschillende ontwerpkeuzes gemaakt die essentieel zijn voor het succes van de webapplicatie. Deze keuzes zijn gebaseerd op de wensen van de klant, de technische mogelijkheden en de doelstellingen van het project. Hieronder beschrijf ik de belangrijkste ontwerpkeuzes die ik heb gemaakt.

- **Client-server database architectuur:** De client is de frontend applicatie dat direct wordt gebruikt door de eindgebruiker. De applicatie dient als interface waarmee gebruikers kunnen communiceren met de server. Verder is de client verantwoordelijk voor het presenteren van de gegevens aan de gebruiker.

De server is verantwoordelijk voor het ontvangen van de verzoeken van de client, het verwerken van de verzoeken en het terugsturen van de gegevens uit de database naar de client. De server is zorgt voor de consistentie, integriteit en beveiliging van de database.

Het client-server database model maakt communicatie mogelijk tussen de client en de database. De client kan verzoeken indienen bij de server, die vervolgens de gegevens uit de database haalt en terugstuurt naar de client. De client kan vervolgens de gegevens presenteren aan de gebruiker. Bovendien kunnen clients met verschillende technologieën communiceren met de server, zolang ze de ondersteunde protocollen gebruiken.

In mijn project maak ik gebruik van het client-server database model, waarbij een Neo4j server wordt beheerd door de server. De frontend applicatie dient als de client en communiceert met de server via een API. De server zorgt voor het beheer van de grafendatabase en haalt de gegevens op uit de database. Tenslotte stuurt de server de gegevens terug naar de client, die de gegevens presenteert aan de gebruiker.

- **Eerste ontwerp:** In het oorspronkelijke ontwerp van de applicatie was er geen server aanwezig. In plaats daarvan was er een directe connectie gelegd tussen de frontend applicatie en de grafendatabase. Dit betekende dat de frontend rechtstreeks communicatie had met de database, zonder tussenkomst van een server.

Het ontbreken van een server had enkele nadelen. Allereerst was er geen aparte laag voor gegevensverwerking. De frontend moest rechtstreeks queries uitvoeren op de database en de resultaten verwerken. Dit resulteerde in een zwaardere belasting van de frontend applicatie, aangezien deze zowel de gegevens moest verwerken als deze presenteren aan de gebruiker.

Dit resulteerde ook in moeilijkere schaalbaarheid, aangezien de frontend applicatie verantwoordelijk was voor het verwerken van de gegevens. Als de applicatie meer gebruikers zou krijgen, zou de frontend applicatie meer verzoeken moeten verwerken. Dit zou kunnen leiden tot een slechte gebruikerservaring en een slechte prestatie van de applicatie.

Na een grondige evaluatie met mijn partner en de klant, hebben we besloten om een server toe te voegen aan de applicatie. Het maakte een betere scheiding van verantwoordelijkheden mogelijk, waarbij de frontend enkel verantwoordelijk was voor het presenteren van de gegevens en de server zich kon concentreren op het verwerken en beheren van de gegevens.

Ook gaf een serverlaag tussen de frontend applicatie en database de mogelijkheid om meer functionaliteit toe te voegen, zoals gegevensvalidatie, caching en load balancing. Dit zou de prestaties van de applicatie verbeteren en de gebruikerservaring verbeteren.

- **Neo4j-OGM:** Een Object Graph Mapper (OGM) is een technologie die een connectie legt tussen een objectgeoriënteerde programmeertaal, zoals Java of in mijn geval PHP, en een grafendatabase, zoals Neo4j. Het doel van een OGM is om objecten eenvoudig te kunnen koppelen aan nodes en relaties in de database.

```php
$node = new Node();
$node
  ->setType("TWDIS")
  ->setExternalId("1234567890");
```

Voorbeeld 1: Voorbeeld van een Node object in PHP. Deze code maakt een nieuw Node object aan en stelt de type en externalId attributen in.