BÄRNHÄCKT

«Danger Ranger» «Nachhaltigkeit in Gemeinden (PLANVAL AG)»

Technische Informationen für die Jury

Technische Informationen für die Jury

Aktueller Stand des Sourcecodes

- Link zu Github Repository: https://github.com/magasser/bernhaeckt-2023

Ausgangslage

- Worauf habt ihr euch fokussiert?
  Wir haben uns darauf konzentriert, die Daten der Planval AG bestmöglich in eine Datenbank auszulagern. Dafür überlegten wir uns zuerst, wie wir die 50 verschiedenen Metriken, welche zum Teil sehr unterschiedlich waren, in ein passendes Schema bringen können.
  Schlussendlich entschieden wir uns 3 verschieden Module zu erstellen. Das erste Modul ist ein Admininterface, mit welchem man die Daten der Planval AG verwalten kann. Hierbei haben wir uns zuerst überlegt, wie wir die Datenerhebung automatisieren können. Die Sources können beim Erfassen neuer Daten ausgewählt werden, um automatisch die Daten auszulesen. Ausserdem besteht die Möglichkeit direkt Werte einzugeben, falls keine passende Automatische Source vorhanden ist. Dieses Vorgehen ermöglicht es für den gleichen Datenindikator pro Gemeinde unterschiedliche Quellen zu verwenden.
  In der Datenbank sind neben den Werten pro Indikator und Gemeinde auch Regeln hinterlegt. Mithilfe einer SQL-View werden die Regeln auf die gesetzten Daten angewendet. Hiermit kann eine Bewertung von 0-10 errechnet werden.
  Das Zweite Modul ist ein Dashboard, welches verschiedene Werte abbildet. Die generierten Charts können für eine Datenanalyse verwendet werden. Damit das Dashboard je nach Vorlieben angepasst werden kann, ist es komplett individualisierbar. Dafür wird ein JSON definiert, indem die Config des Dashboards festgelegt ist.
  Weiter haben wir uns überlegt ein Umfragemodul zu erstellen, mit welchem man den Gemeinden eine Plattform bieten könnte, ihre Gemeindedaten einzutragen. Dies war in diesem Zeitlichem Rahmen leider nicht möglich.
- Welche technischen Grundsatzentscheide habt ihr gefällt?
  Wir haben uns entschieden, das ganze Projekt mithilfe von Docker in Container auszulagern. Mithilfe dieses Ansatzes ermöglichen wir eine Plattform unabhängige Lösung.
  Als Projektmanagement wurde Kanban verwendet. Um die Umsetzung zu vereinfachen, verwenden wir einen Monolithen anstatt einer Microservice Architektur. Bevor wir mit der Umsetzung des Backend / Frontend gestartet haben, wurde das Datenbank Schema konzipiert und festgehalten. Damit war bereits vor der Umsetzung klar, welche Models erstellt werden müssen. Aus Sicht der Architektur haben wir uns für das MVC-Pattern entschieden, damit eine klare Trennung besteht.

Technischer Aufbau

- Welche Komponenten und Frameworks habt ihr verwendet?
  Folgender Tech-Stack wurde verwendet:
  Spring Boot, Spring Security, Java, JPA, Flyway, Swagger / OpenAPI, Git / GitHub, Angular, Material Design / SCSS, Postgres, Docker / Docker-Compose, ChartJs, PrimeNg, Lombok, Maven, NPM
- Wozu und wie werden diese eingesetzt?
  Spring Boot in Kombination mit Spring Security und Java wurde verwendet, um dem Frontend eine API zur Verfügung zu stellen. Mithilfe von JPA wurde eine Verbindung zwischen Backend und Datenbank hergestellt. Damit Daten gespeichert werden können wurden verschiedene Tabellen in einer Postgres Datenbank angelegt. Um allfällige Datenbank Migrationen durchzuführen, wurde Flyway eingesetzt. Unsere API wird mithilfe von Swagger dokumentiert und kann direkt getestet werden. Zur Verwaltung des Sourcecodes wurde Git in Kombination mit GitHub verwendet. Als Frontend Framework haben wir uns für Angular entschieden. Material Design zusammen mit SCSS wurde angewandt, um ein modernes und ästhetisches UI zu

definieren. Damit die Applikation in Containern laufen kann, haben wir Docker und Docker-Compose ausgewählt. Anschauliche Diagramme werden mit ChartJs und PrimeNg generiert. Lombok wird verwendet, um den Boilerplate code im Backend zu reduzieren.
NPM und Maven dienten als Dependency Management im Frontend / Backend.

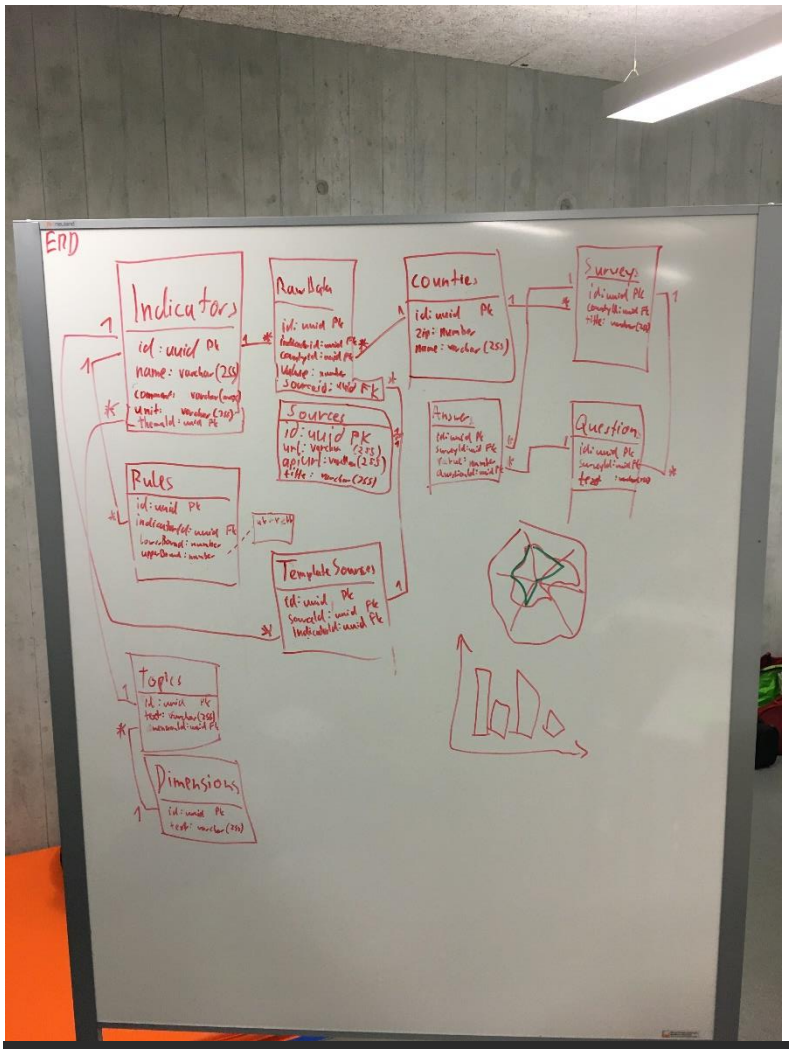
Implementation

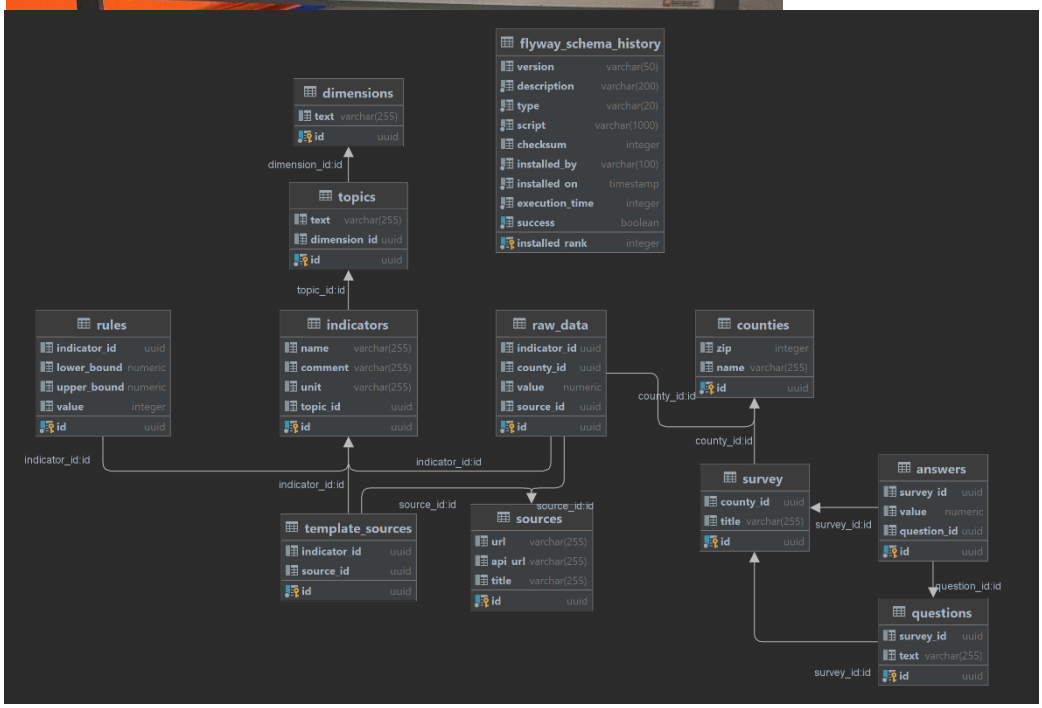
- Gibt es etwas Spezielles, was ihr zur Implementation erwähnen wollt?
  Wir haben grossen Wert auf ein modernes UI und auf eine klare Datenbank Struktur gelegt. Eine klare Trennung im Backend war uns sehr wichtig.
- Was ist aus technischer Sicht besonders cool an eurer Lösung?
  Viele unnötige Zeilen Code wurden eingespart. Eine Performance Verbesserung konnte durch die Verwendung von SQL-Views verwendet. Wir führen die Berechnung der Bewertungen in der Datenbank durch mithilfe einer Rules Tabelle.

Abgrenzung / Offene Punkte

- Welche Abgrenzungen habt ihr bewusst vorgenommen und damit nicht implementiert? Weshalb?
  Wir haben uns Entschieden nicht alle Daten in die DB zu übernehmen. Grund dafür war, dass dies zu zeitaufwändig ist. Stattdessen wurde sich auf eine Handvoll Testdaten beschränkt.
  Zeittechnisch hat es uns nicht gereicht alle Testdaten APIs anzubinden, deswegen wurden die Endpunkte gemockt.
  Das Modul zum Erstellen und Verwalten von Umfragen wurde aufgrund von Zeitmangel gestrichen.


TO DO
In Progress
Done
Chart Dar
Demo Data für Pitch
Tech Doc erstellen
Pitch vorbereiten
Share Config
Project Dir Config
Indicator anpassen
Swagger einbinden
Docker für Spring
Endpoints to create Countries
Ui zum Datenerstellen
Neuen Indicator erstellen
Erstellen Ui Mock für erstellen von Daten
Chart erstellen Spieler
Dashboard übersicht


END
Indicators
id: uuid PK
name: varchar(255)
RawData
Counties
id: uuid PK
zip: number
Survey
Sources
id: uuid PK
Answer
Question
Rules
TemplateSources
Topics
Dimensions


flyway_schema_history
version varchar(50)
description varchar(200)
type varchar(20)
script varchar(1000)
checksum integer
installed_by varchar(100)
installed_on timestamp
execution_time integer
success boolean
installed_rank integer
dimensions
text varchar(255)
id uuid
dimension_id:id
topics
text varchar(255)
dimension_id uuid
id uuid
topic_id:id
rules
indicator_id uuid
lower_bound numeric
upper_bound numeric
value integer
id uuid
indicators
name varchar(255)
comment varchar(255)
unit varchar(255)
topic_id uuid
id uuid
raw_data
indicator_id uuid
county_id uuid
value numeric
source_id uuid
id uuid
counties
zip integer
name varchar(255)
id uuid
county_id:id
indicator_id:id
source_id:id
survey
county_id uuid
title varchar(255)
id uuid
survey_id:id
answers
survey_id uuid
value numeric
question_id uuid
id uuid
question_id:id
template_sources
indicator_id uuid
source_id uuid
id uuid
sources
url varchar(255)
api_url varchar(255)
title varchar(255)
id uuid
questions
survey_id uuid
text varchar(255)
id uuid