Explainable Similarity Detector

Explainable Similarity Detector

AMOS WS 2021/2022

Project team 6

09.02.2022

Table of contents

1 Runtime components **2**

2 Code components **4**

3 Technology stack **6**

4 Textual explanation **7**

4.1 Frontend 7

4.2 Backend 7

List of figures

1 Runtime components - Backend 2

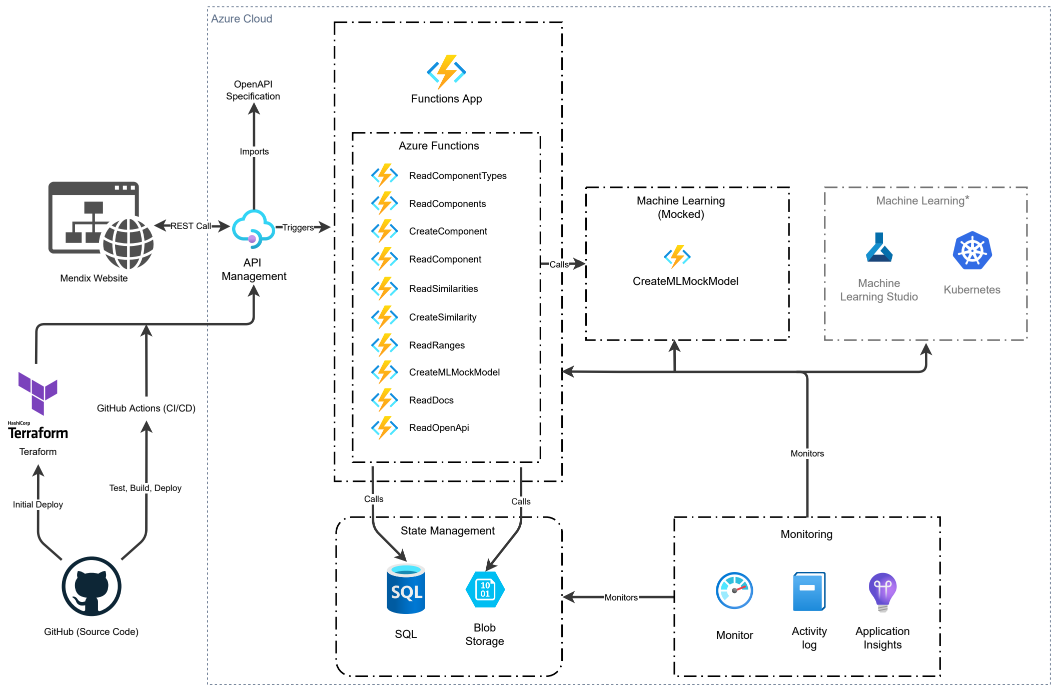
2 Runtime components - Frontend 3

3 Code components - Part 1 4

4 Code components - Part 2 5

1 Runtime components

Overview of the runtime components



*For our final release for Siemens we planned to use the Azure Machine Learning Studio (API).
In the current state the Azure Machine Learning Studio (response) is mocked by an Azure Function (CreateMLMockModel).

Abbildung 1: Runtime components - Backend

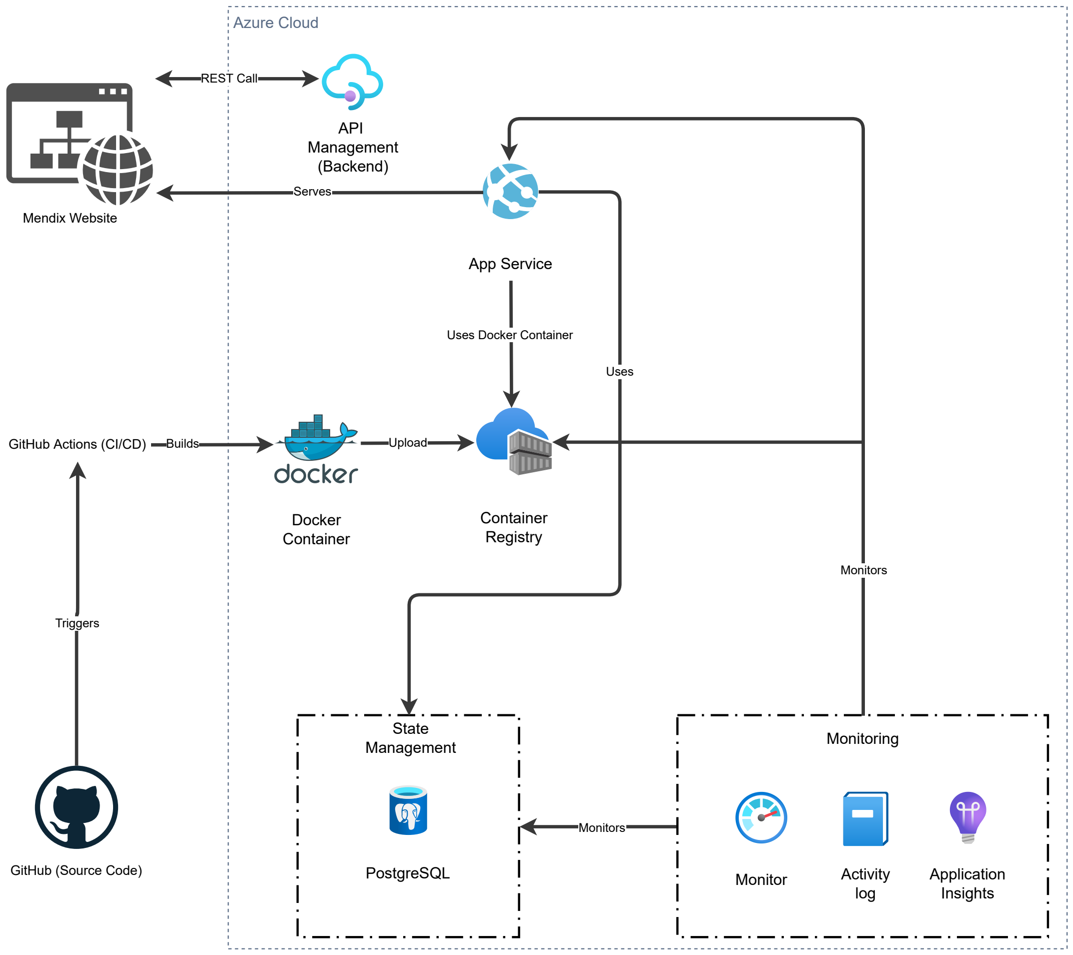


Abbildung 2: Runtime components - Frontend

2 Code components

Overview of the code components

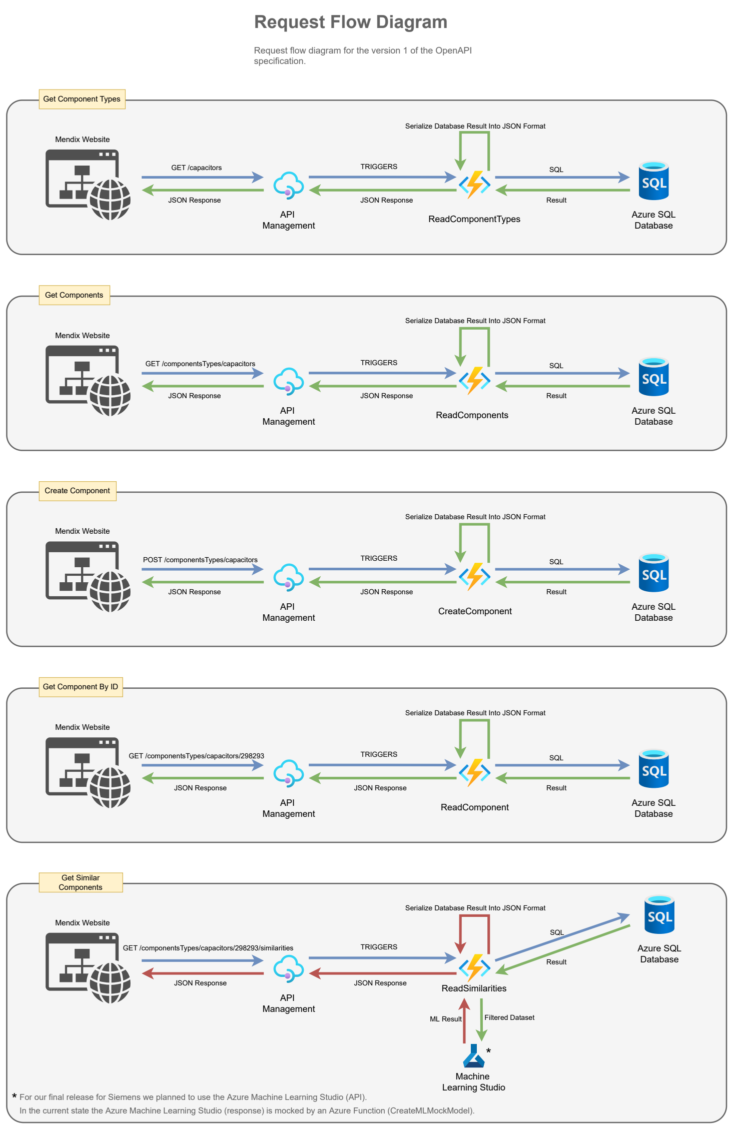


Abbildung 3: Code components - Part 1

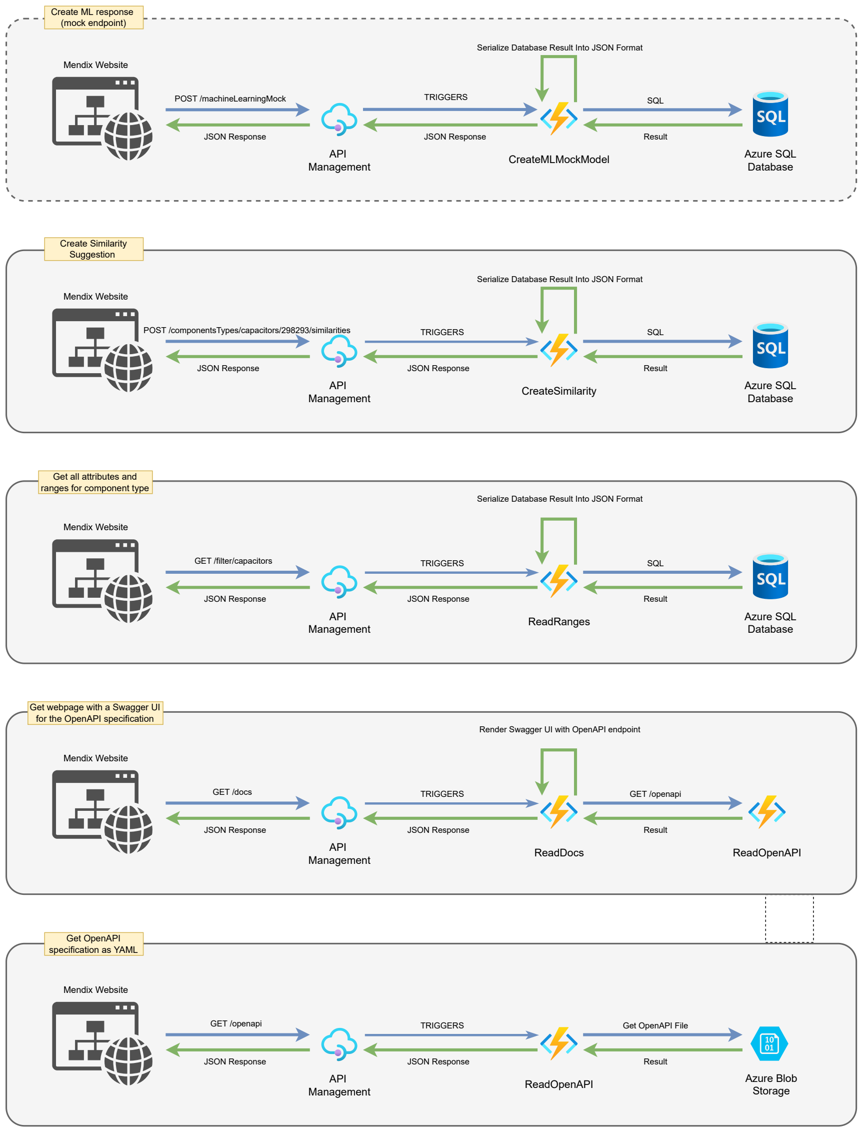


Abbildung 4: Code components - Part 2

3 Technology stack

Overview of the underlying technology stack

- **Frontend**
  - Mendix
  - Pluggable Web Widgets (React)
  - Azure App Service
  - Azure Container Registry
  - Azure PostgreSQL
  - Azure Activity Logs
  - Docker
  - GitHub
  - GitHub Actions (CI/CD)
- **Backend**
  - Azure API Management
  - Azure Functions App
  - Azure Functions
  - Azure SQL
  - Azure Blob Storage
  - Azure Machine Learning Studio
  - Azure Activity Logs
  - Azure CLI
  - Terraform
  - Docker
  - SQLAlchemy
  - pydantic
  - OpenAPI
  - act (Simulate GitHub Actions locally)
  - black (Python code formatter)
  - Azure VSCode Extensions
  - OpenAPI / Swagger VSCode Extensions
  - GitHub
  - GitHub Actions (CI/CD)

4 Textual explanation

Textual explanation of the diagrams and choices

4.1 Frontend

Our industry partner (Siemens) has desired to use the low-code software platform called Mendix for the frontend. In addition, we use what Mendix calls Pluggable Web Widgets to extend the widgets provided by Mendix using React. There requests can be made to the backend via REST request. Since Mendix provides a backend in addition to a frontend, we use Docker to deploy Mendix. This allows us to deploy easily. To make this possible we use the Azure Container Registry to upload the built image of our Mendix website. Using an Azure App Service we created, we host a container of the uploaded image of Mendix. For the Mendix custom database, we use an Azure PostgreSQL database. In it, components are cached that need to be verified by an administrator beforehand.

4.2 Backend

For the backend, we use Microsoft's Azure cloud computing platform. All REST requests are received by the Azure API Management (see 1). The API Management imports the OpenAPI specification defined by us. The Azure Functions stored there (which are grouped by the Functions App) are assigned to corresponding REST endpoints. These Azure Functions parse the request and then typically access the database or Machine Learning Studio. They then send the received data back to the frontend in a format specified in OpenAPI.

In Azure Machine Learning Studio, the machine learning model can be found, which is able to find similar electrical components.

The required data is stored in an Azure SQL database. For support and debugging/tracing, various Azure services are used, such as the Activity Log.

To ensure a high code quality we used black as Python code formatter. We also used the linters/tools flake8, bandit, pyupgrade, pydocstyle, pyright, pylint and autoflake. Furthermore we used unit and integration tests to ensure that the code works like expected. The unit tests are executed with our CI on every commit to make sure that the commit doesn't has errors. The integration tests are executed from our CI on every new tag. We also use a pull request template to have a consistent review schedule.