Technical Design Documentation

Overview

This technical design documentation provides information on the architectural design decisions of the product. It will illustrate the components of the system and their relation with each other. Users of the software will gain an understanding of how the product works as well as understand the front- and backend architecture decisions.

First, the architecture and data model will be discussed and afterwards it will go more into detail about the implementation of certain components and which technology was used.

Terminology

Here you can find some terms that are used in this documentation.

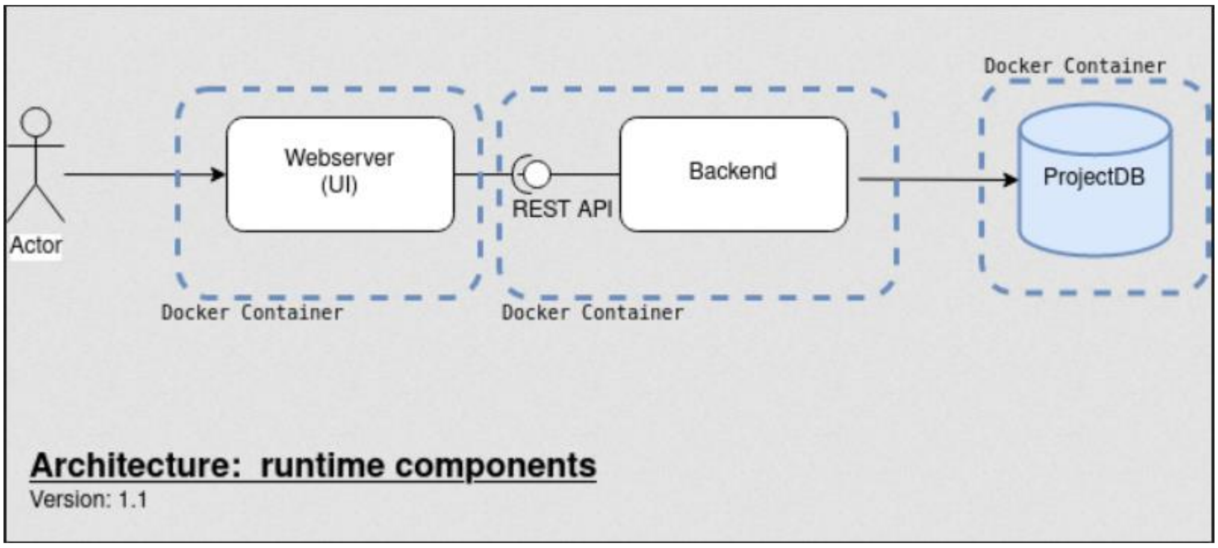
Term	Definition
Admin	The admin can create/delete users and assign roles
Consultant	The consultant is the main user of the software, which can conduct analysis, enter ratings and print charts.
Evaluation view	An Epic describes several issues which are grouped as one topic
Komplexitätskriterium/-treiber	Is a class of questions with the same topic
Marge	The margin whichs displays the y-axis in the in the pie chart
Piechart (äußerer Ring)	The customer complexity, for each Produkt three fields for the percentage of hoch, mittel & gering complex customer is given.
Piechart (innerer Ring)	The distribution of Bewertungen of Produktvarianten for one Produkt.
Piechart (Größe)	The volume of the piechart displays the criteria "credit volume" in the economical evaluation
Produktbereiche	Every product area has to sub areas (private and corporate) & and there are 1 to n products for each product area
Produkt	Every product has one to n product variants, which are divided in product areas.
Produktvariante	One or many Produktvarianten are forming the Produkt; The Produkt is a Produktvariante of itself; Exp: Sofortdispo, Dispo Fix & Dispo Variabel
Rating	Hoch, Mittel & gering (with colours: red, yellow & green); is set by a consultant
Result view	The webpage with the charts
Projekt	A Projekt has one or many Produkte; Each Projekt has 1-n Produktbereiche
User	A User is ther overall class for Consultant, Admin and Project Manager

Term	Definition
Gesamteinschätzung wirtschaftliche Bewertung	This value is used to determine the economical complexity of Produkt or a Produktvariante; This value is displayed in the inner piechart ring

Architecture & Design Principles

Software Architecture Diagrams

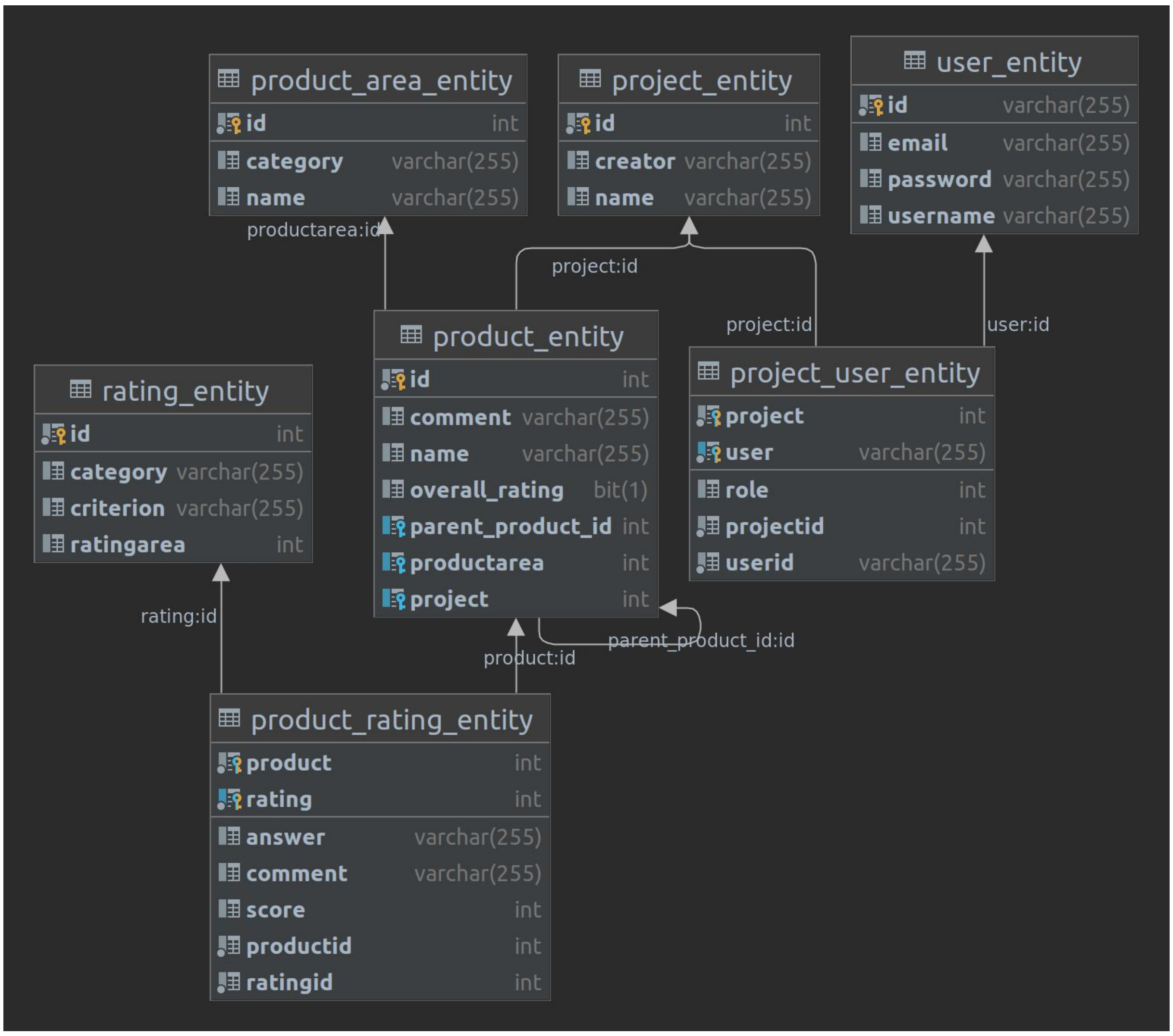
Runtime Components



The runtime components of the software consists of a frontend, backend and the database service. These three components are fully containerized with docker. The communication between front- and backend takes place over a REST API, which definition can be found below. In the current version of the software the backend communicates directly with database via a spring libary. The backend is written in java and has and in REST API is integrated in this service through spring boot. The fronted is based on Javascript and React.

The micro-service architecture and fully containerization of each service improves the reliability and maintainability. So it es possible to restart and malfunctioning service, change the database on the fly or even deploy more units of service the deal with high demand of a service. Further, the REST API which relies on well know and widely used interfaces and protocols is an ideal foundation of this web service.

Data Model



In this chapter we will focus on the data model of the database. It's easy to see that the Entity classes and the structure of the database are alike since these classes are used to fill data into the database.

The main point of the database and this software in general are projects or for the database **project_entity**(s). Their **id** (the project id) is used as a foreign key for products (**product_entity**) or to assign users to a project (**project_user_entity**). The **project_entity** table starts empty is filled by the users of this program, probably consultants or admins.

As already mentioned before the **project_user_entity** table links entities from the **user_entity** table to a project. Therefore, this table is dynamic and grows with each project. In current state of this program the list of users is fixed and preloaded into the **user_entity** table, which also applies for the **product_area_entity table**

In this software each project has products, which is represented as the **project_entity** and **product_entity** relationship. A product can have Produktvarianten. A Produktvarianten behaves like a product with the exception that a Produktvariante can't have another Produktvariante. This relationship is represented over the foreign key **parent_product_id** in a product_entity entity which is a foreign key to already existing product id.

The **product_rating_entity** table links answers, comments and scores to rating from **ratings_entity** table and links both to a product. So changes on the answers would not affect a rating entity itself but product_rating entity.

Backend

The Backend is built in a Microservice Architecture style with Spring Boot, which are listed as follows:

Service	Description
ProductArea Service	Handles the product areas. You can get listed all the product areas or create new product areas here
ProductRating Service	Handles the ratings for products. You can create product ratings, update product ratings or find the rating for a specific product here.
Product Service	Handles the products. You can for example create & delete products or update & find products by the ID here.
Project Service	Handles the projects. You can create projects, find & update projects or get shown all projects here.
Rating Service	Handles the ratings. You can get all ratings here or get a rating for a specific rating area.
User Service	Handles the user management, like create or delete a user, finding a user or update the users email address and password

DTO's (Data Transfer Objects) are used to send Data between Backend and Frontend.

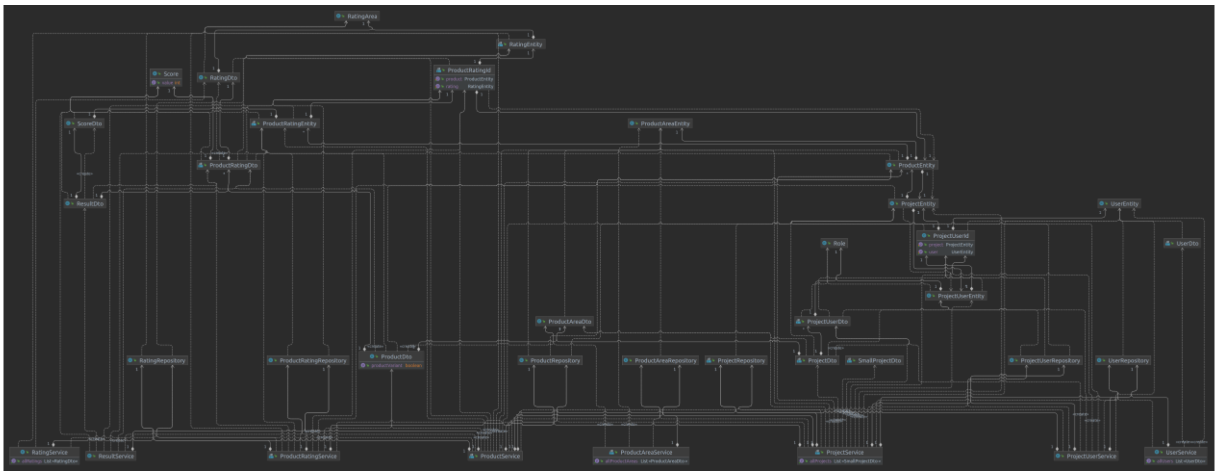
DTO	Attributes	Description
UserDto	UUID userID , String userEmail, String userName, String password	contains Data about User
ProjectDto	int projectID, UUID creatorID, String projectName, List members, List productAreas	contains Data about Project
ProjectUserDto	public UUID userID, String userName, String userEmail, Role role	contaisn Data of Project User (see ProjectUserDto in ProjectDto)
ProductDto	int productID, String productName, ProductAreaDto, int projectID, int parentID, float progressComplexity, float progressEconomic, List ratings, List productVariations, String comment, List resources	contains data about a product or a product variation
ProductAreaDto	int id, String name, String category	describes Area of Product (see ProductDto)
ProductRatingDto	int ratingID, RatingDto rating	contains rating of a Product (see ProductDto)
RatingDto	int id, String criterion, String category, RatingArea ratingArea	explicit Rating of a Product (see ProductRatingDto)

DTO	Attributes	Description
ResultDto	int productID, String productName, List ratings, ScoreDto[] scores, int[] counts	contains "analyze" results
ScoreDto	Score score, int count	contains explicit score data (see ResultDto)

Entity's are used to save Data in Database. DTO's and Entity's are quit similar in Attributes and Description.

Own Classes:

Class	Attributes	Description
Score	int value	Enum can be "HOCH, "Mittel" "GERING". Used for Rating. (see Rating)
ProductRatingId	ProductEntity product, RatingEntity rating	Serializable class to identify Rating with a hash Code (see Rating)
Role	CLIENT, PROJECT_MANAGER, ADMIN	Enum can be "CLIENT", "PROJECT_MANAGER", "ADMIN". Defines role of a User. (see User)



Frontend

React

The frontend is built with the React framework, a javascript framework for building web applications. The root location for the frontend source code is `/frontend/src`. This directory has the following contents:

Subdirectory	Description
`App.jsx`	App.jsx is the root Component of our React application. It mainly acts as a container for all other pages of our application.
`index.js`	index.js is the entrypoint to our application. It renderes the App.jsx component.
`/components`	Contains reusable React components we implemented to be used across the project (e.g. the menu bar).

Subdirectory	Description
`/pages`	Contains all components specific to a single page. For example, all components specific to the result page are contained in `/src/resultPage`
`/styles`	Contains JavaScript files determining CSS styles for reusable components. The style definitions extend the standard ChakraUI themes.
`/utils`	Contains three JavaScript files: • `apiClient.js` -- The api client used to commicate with the backend • `const.js` -- Constants used by the application • `notifications.js` -- Function to show toast message to the user
`store.js`	Contains JavaScript file responsible for state management based on the easy-peasy library.

API client

To communicate with the backend Wretch is used. Wretch is a wrapper for fetch and provides simplified handling of API communication e.g. processing a request body or managing request errors. The Wretch documentation is provided here.

State Management

For state management the Easy Peasy library was used. Easy Peasy is an abstraction of Redux, an open-source JavaScript library for managing and centralizing application state. Easy Peasy's core concepts include hooks and methods such as store, state or actions whereas the letter is used to update the application store. Further information about Easy Peasy can be found on their webpage.

UI Framework

We used Chakra UI to implement the above mentioned components in frontend. Chakra UI is a component library which provides React components using Chakra UI's default CSS styles. It offers basic building blocks such as Buttons, TextAreas and Flex boxes. Further information about Chakra UI can be found here.

Technology Stack

In the following list you can see the projects underlying technology stack. It is a combination of the technologies, which we have used to build and run the application. To be more accurate it displays our used programming languages, front- and backend frameworks, software applications and our database decision.

1. **Programming Languages**

- *Java (Backend)*: Java is an object-oriented programming language which consists of a development tool for creating code and a runtime environment to run code.
- *JavaScript (Frontend)*: JavaScript is a scripting & just-in-time compiled programming language for web pages. It supports object orientation and is imperativ.

- *HTML (Frontend)*: HTML is a markup language for documents which can be displayed on web pages. It is used for structuring web pages and the content on the web page.
- *CSS (Frontend)*: CSS is a style language which describes the presentation of a web site written in HTML.

2. **Frameworks**

- *React (Frontend)*: Is an open-source frontend framework that is based on JavaScript.
- *Spring Boot (Backend)*: Spring Boot is an open-source java framework used for a microservice architecture.

3. **Version Control Software**

- *Git*: Git is a distributed version control system with the aim to help our group to develop the software together.
- *GitHub*: GitHub is the place where our repository lays. It is a provider for hosting software development and version control using Git.

4. **Database System**

- *MySQL*: Is a relational database system.

5. **CI/CD-Tools**

- *GitHub Actions*: Is a continuous integration and continuous delivery platform, which automates building, testing and the development.

6. **Other development tools**

- *Docker Engine*: Docker is for isolation of an application using containers.
- *Docker CLI*: It is the Command Line Interace using Docker.