Architecture Document

Abra Cadabra

Philips

Date: 05-01-2021
Version: V0.5
State: Ready for review
Authors: Lars van den Brandt, Kevin Bevers, Kristian Lachev, Robbe Bryssinck, Denny Cox, Tijmen Coenders, Ivan Banov, Mureşeanu Gabriel

Version History

Version	Date	Author(s)	Changes	State
0.1	18-09-2020	Everyone	Wrote down components and technologies we will use for them	Ready for review
0.2	05-10-2020	Everyone	Changed the components to reflect our change of plan	Ready for review
0.3	01-12-2020	Everyone	Added an admin panel component with a proper diagram	Ready for review
0.4	04-01-2020	Everyone	Added the entity relationship diagram for the database	Ready for review
0.5	05-01-2020	Everyone	Improved the introduction, added stakeholders and constraints, high-level overview, improved components and improved database diagram	Ready for review

Contents

Introduction

The goal of this document is to convince the stakeholders of the technical choices in the project and how we have the project in mind. So, in this document you will find our technical choices elaborated on and explained in detail as well as a high-level overview of our components. Furthermore, this document contains a component diagram and an Entity relationship diagram for the data models we have in the project.

Diagrams

While making these diagrams we decided to use the C4 model. This is a well-known model to visualize software architectures. In this model there are 4 so called levels of detail. Level 1 is the system context, which is the highest-level diagram. It shows the scope of the software system. Level 2 is container diagram. It shows the high-level technical building blocks of the software system. Level 3 is the component diagram. It shows the container with the components inside of it. Level 4 is a code diagram. This can be a UML class diagram. It shows how a specific component is implemented.

Stakeholders and constraints

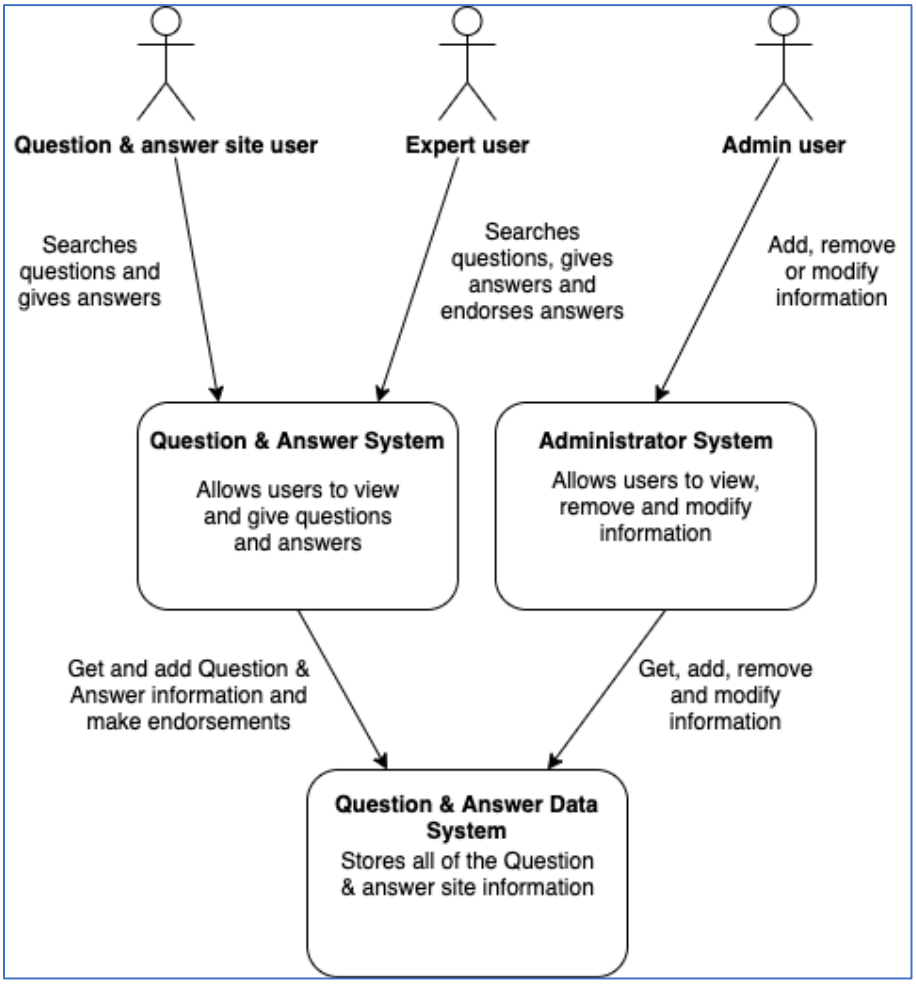
For our project we have different stakeholders. Liesbeth and Alexander from Philips, the users of the website, the experts and the administrators. They are stakeholders because they will all have a stake in the web application. The stakeholders and their goals are described in the table below.

Stakeholder	**Goal**
Liesbeth and Alexander (Philips)	The web application is safe to use
User	The web application is easy to use
Expert	The web application is easy to use
Admin	The admin environment is easy to use

The constraints that we have are the following: the first is financial, because we don't have a budget with the project. The second is time, because we have 18 weeks to finish the project. The third is ethical, because the web application should be ethical.

System context

To show the scope of the application we will describe it in a high-level overview. The different users that will use the application, of which there are three, are shown in the diagram below. The Question & Answer site user and the expert user use a separate system from the admin user. These two separate systems use the data system to access all the stored data.



Components

The application will be distributed. The front ends and the back end will be separated. This will allow for a modular design where there the front-end or the back end could easily be switched out, without having to rework the other component. The back end will be set up as a REST API. This architecture is very modular, making it easy to switch out or add more components. It is also a very standardized way of providing an interface to the data supplied by the back end. This makes it easy to understand for new developers coming into the project. It will also result in a quick MVP, since it requires minimal setup.

Web apps

The application will contain two web apps (the main web client and the admin client) and an API. They will both interact with the same API, and therefore the same database. This way, the application can still be separated while maintaining easy sharing of data and functionality between components through the

API. Both web apps will use largely the same technologies (as described below), so that the maintainers of the web client could also easily understand and possibly maintain the admin client, and vice versa. The web client and the admin client will be hosted on separate domains.

The front end is going to be run using ReactJS. React is a good fit for this project, since it makes heavy use of reusable components. Our application must render many simple things many times, like lists of answers, questions, subjects and so forth. React excels in this. React is also a very modern approach to web development, is being actively developed, and has a lot of extensions and library support.

For interacting with the back end, the front end will use Axios to make HTTP calls to the REST API. Axios is easy to use and makes use of modern functions to make the application as fast as possible by using asynchronous functions for example.

API

The API will be RESTful. This is a standardized architecture, performant, easy to understand and easy to work with. The main technology being used here is ASP.NET Core. We specifically chose Core over Framework because that seems to be the direction Microsoft is heading in to with their ASP.NET platform. It is also more modular since it is fully cross platform. ASP.NET Core uses C# as its programming language. The team already has a lot of familiarity with this language, and the language is very relevant in the industry.

Recommendation service

Based on performance we will build the recommendation algorithm as a separate service within the API. This service will be built with the “BackgroundService” class. It will use the Entity framework ORM to access the same database that the API uses. It will use this data to calculate trending questions.

Database

The database used will be MSSQL. MSSQL is very integrated in the ASP.NET ecosystem. For example, “localdb” can be used to easily debug the application and throw away the database afterwards. This database runs on MSSQL. The Entity framework ORM will be used to access the database from the API. This tool also works very well with ASP.NET, contains enough features for our use case and is

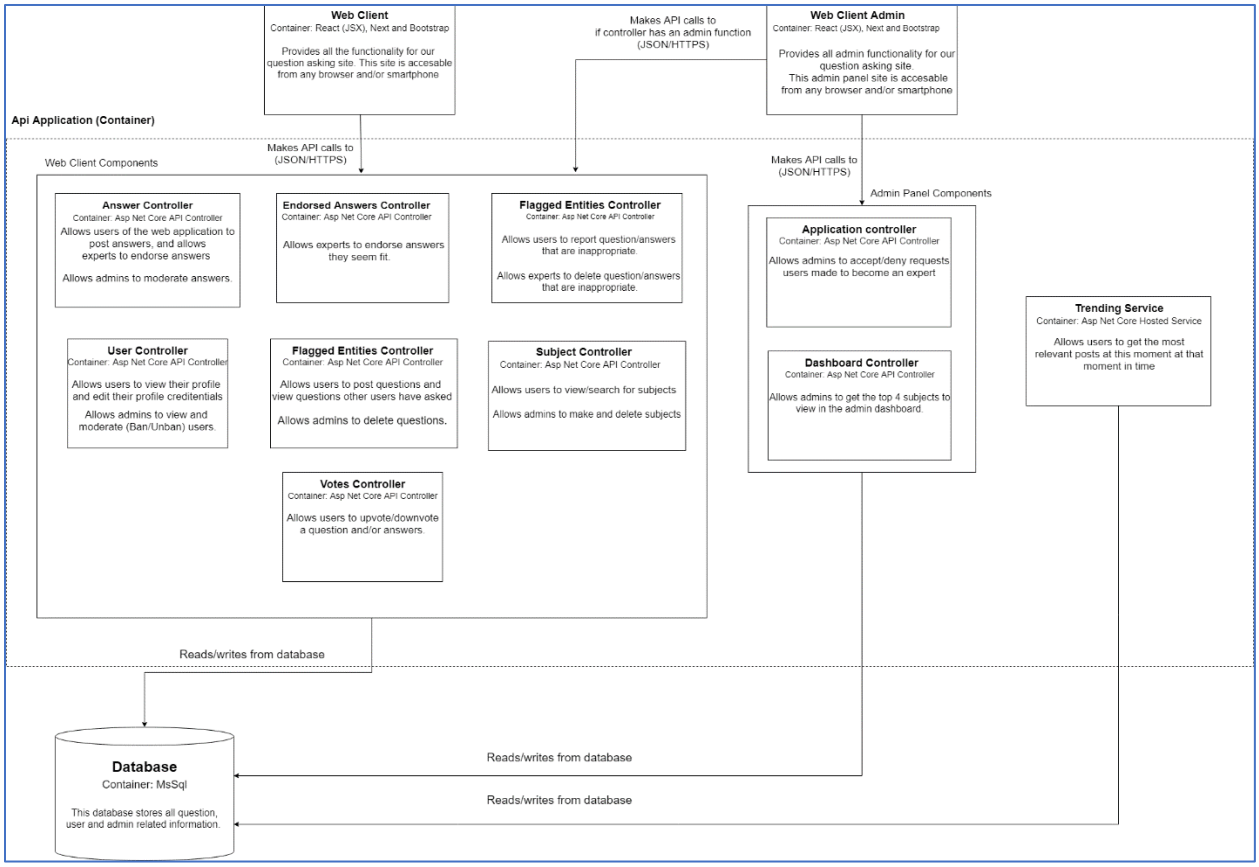


Figure 1: component diagram

Database Diagram

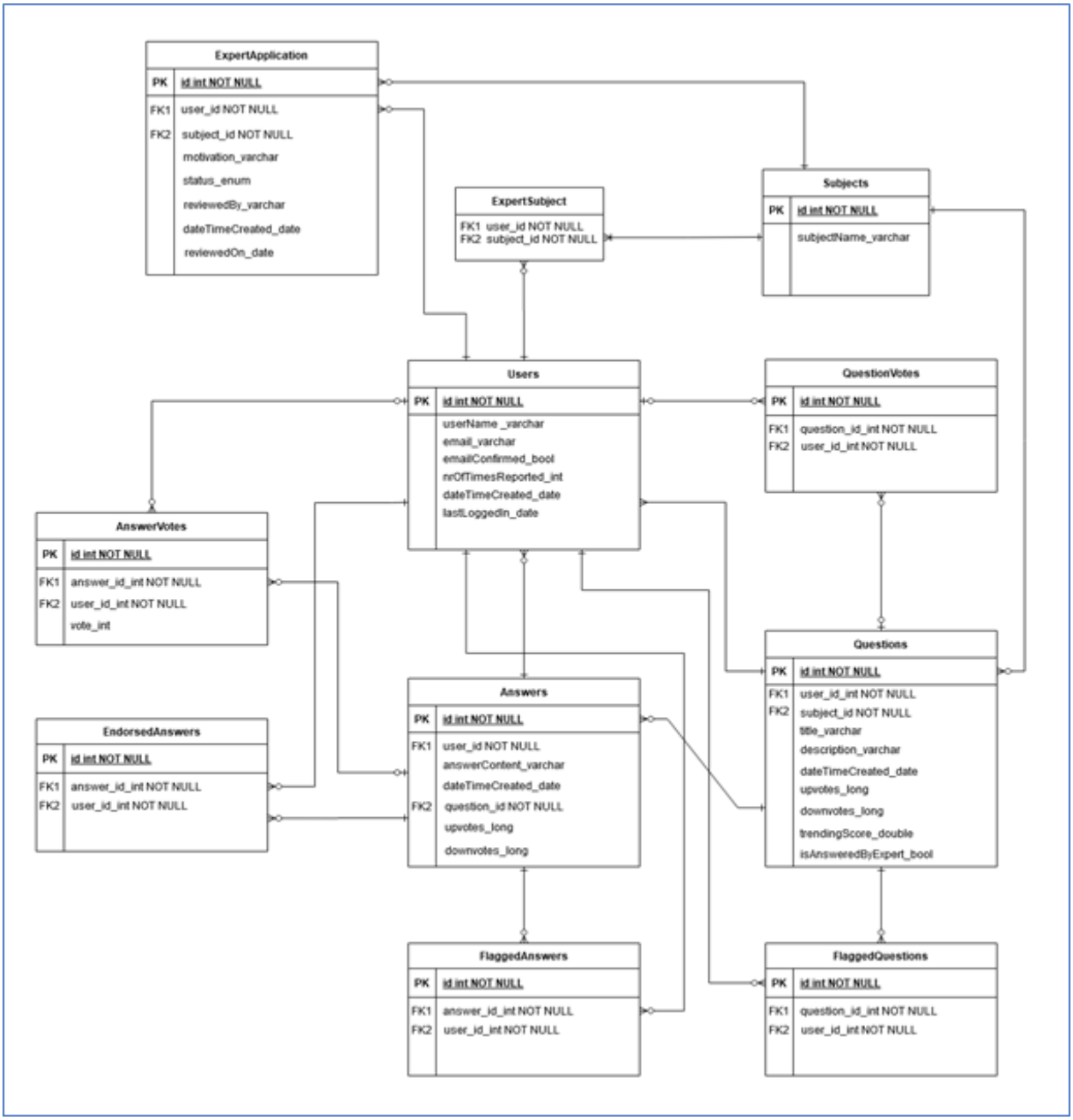


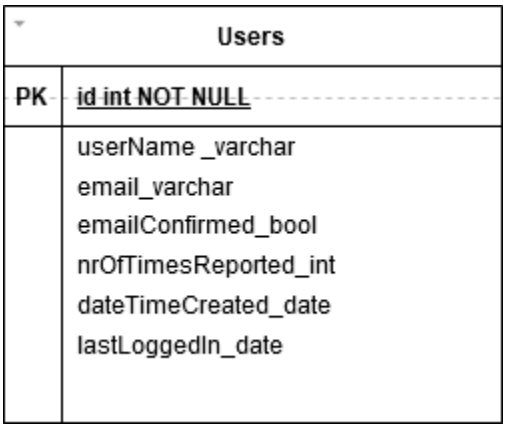
Figure 2: Database Diagram

Database Diagram Description

Users – Our application is revolved around the User and as such it is connected to almost every functionality that exists inside our application.

For the user we use a simple username and email. The **"nrOfTimesReported"** field is used to keep track of how many of the User's questions and/or answers have been reported. This enables us to separate

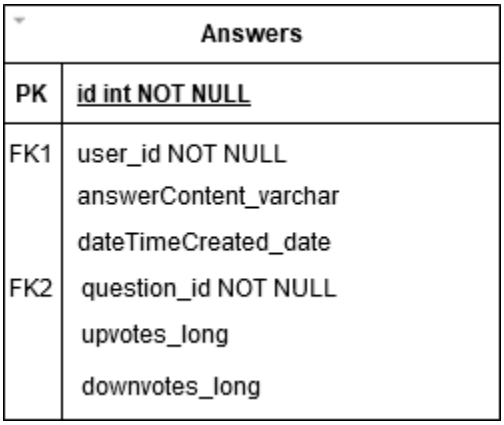
the normal users from the bad ones. We know that this field cannot be abused, as this is only counted if a flagged entity is created and the flagged entities are always identifiable and unique.

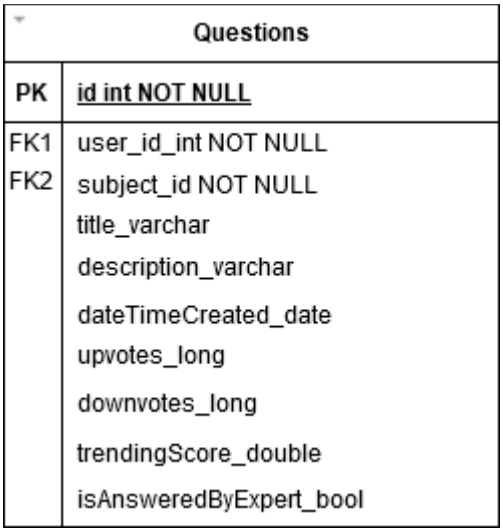


From the Database Diagram we can conclude that the user can **post** Questions and Answers. A user may post **many** Questions and/or Answers. A User Id is connected to both entities in order to identify the user that has posted them.

The interesting field here is the **"trendingScore"**. The trending score is calculated by the Trending algorithm which takes all the downvotes and upvotes and the time of the question's creation and calculates its score based on these factors. The trending score then enables us to filter the most interesting and liked question and display them.

The **"subject_id"** foreign key is used to determine which is the subject that this question is part of. Having the relation between subjects and question gives us the ability to categorize the questions based on the subject. The subject id can never be null as a question must always have a subject associated with it, otherwise it would not be valid.





A user can also **report/flag** Questions and/or Answers. A user may flag **many** Questions and/or Answers. We couldn't just use a simple flag counter as that would enable the User to flag a question and/or answer for an infinite number of times as there is no way to check if the user has already flagged a

question and/or answer. Using these junction tables makes it easier for us to keep track of which user flagged what, thus making it impossible for the user to flag an entity multiple times.

FlaggedAnswers	
PK	id int NOT NULL
FK1	answer_id_int NOT NULL
FK2	user_id_int NOT NULL

FlaggedQuestions	
PK	id int NOT NULL
FK1	question_id_int NOT NULL
FK2	user_id_int NOT NULL

A user can vote on Answers. A user may vote on **many** Answers once. The reason we have a junction table for this is the same as flagging questions and answers, we must have a way to identify the user which downvotes or upvotes the entity otherwise we would not be able to prevent multiple votes on the same entity by the same user.

The **"vote_int"** field used to determine the total vote score of downvotes and upvotes. The number is between +1 and –1 and every time a user votes it is saved in this table and whenever we need to use an Answer entity we just search for the id of the answer and we calculate the total votes. This makes everything more dynamic and easier to work with.

AnswerVotes	
PK	id int NOT NULL
FK1	answer_id_int NOT NULL
FK2	user_id_int NOT NULL
	vote_int

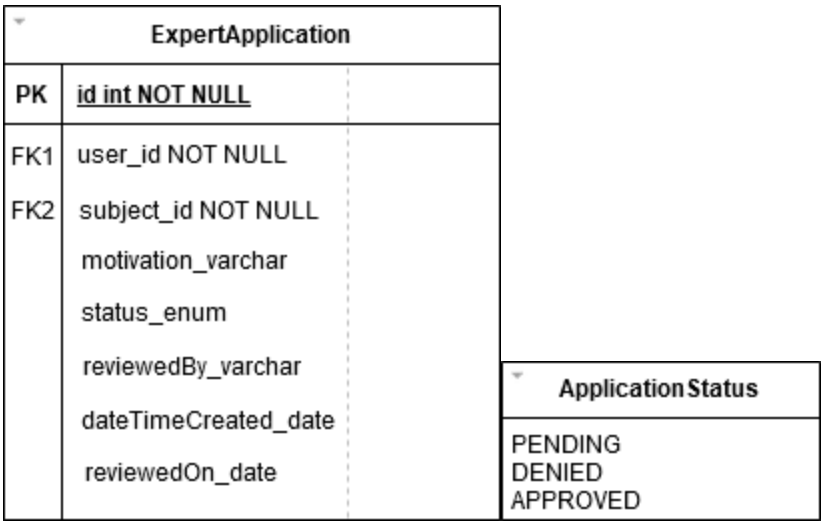
"ExpertApplication" table is used to keep track of all the User's applications for the Expert role, which gives the user the ability to **Endorse Answers**. When an Expert Application is posted it is displayed in the admin dashboard, where the admins can determine if the User is fit for the role after looking at his/her motivation. The ExpertApplication is always associated with the subject id as we need that to determine where the user want to become an expert on.

The "**status_enum"** is the way admins can communicate with the user about their verdict this enum can be either

APPROVED – which make the User an expert.

DENIED – The user is denied, and the process ends. A timer is placed so the user may not apply for the expert role in the specific subject for an X amount of time.

PENDING – This means that the application is still being reviewed and the process may continue.



In the database we do have one **many to many** relationship table called **"ExpertSubject"** and it serves as a connection between the User and the Subject he/she is an expert in. The reason for this is that it makes everything much more dynamic. This makes it possible to add users with multiple expert roles and it ensures the stakeholders that whenever a new subject theme is added the system will still work as intended.