Technical design document

Author: Denny Cox
Class: S6-RB10
Semester: 6

Table of contents

Technical choices

Microservices

The Kwetter application is enterprise software and will be built using a microservices architecture. With this, the functionality of the application must be separated into different services. In the analysis document the project has been worked out with event storming. The different aggregations that came out of this can be used as microservices.

Frontend framework

For the frontend React was chosen because it is most familiar frontend framework for me and with microservices it doesn't really matter what frontend technology you use.

Backend framework

For the backend ASP.NET Core was chosen because it is most familiar backend framework to me and with the microservices that will be made for Kwetter it can be used for of them, since Identity and Entity Framework can easily be implemented.

API gateway

Since the application is broken up into smaller services there has to be a way to communicate from the frontend. For this an API gateway has been added. The advantage is that the endpoint of the different APIs in the system will not be exposed and protected from different kinds of attacks. Because every microservice will communicate via a single point of entry with the API gateway, things like authorization using API tokens will be easier to implement. This decreases complexity.
Disadvantages are that the single point of entry could also act as a single point of failure. The routing to the different microservices must also be managed during deployment to ensure the routing works correctly.
A technology must be chosen to create an API gateway. Important for the gateway is that it is well suited for a microservices architecture that need unified points of entry into their system. It should be fast, scalable, cross-platform and provides routing and authentication among many other features. In the end the choice was made to use Nginx.

	Weight	Ocelot	Kong	Azure API management	Nginx
Integration	0.13				
Docker support	0.10				
Documentation	0.20				
Community	0.15				
Open-source	0.12				
Monitoring	0,15				
Compatibility	0,15				
Total	1.00	0.60	0.85	0.88	1.00

Event bus

To keep the data in all the services up to date if data changes, the system uses an event bus. This architecture is called an event-driven microservice architecture. A technology must be chosen to create an event bus. The Following messaging technologies have been investigated: RabbitMQ, Kafka, ActiveMQ and NATS. It was important to have a good and popular choice for an open-source message broker that is easy to deploy for distributed systems. In the end the choice was made to use RabbitMQ.

	Weight	RabbitMQ	Kafka	ActiveMQ	NATS
Integration	0.13				
Docker support	0.10				
Documentation	0.20				
Community	0.15				
Open-source	0.12				
Monitoring	0,15				
Compatibility	0,15				
Total	1.00	1.00	0.85	0.65	1.00

Figure [1] shows the global architecture of the project. As shown, the different components will be able to run in separate Docker containers.

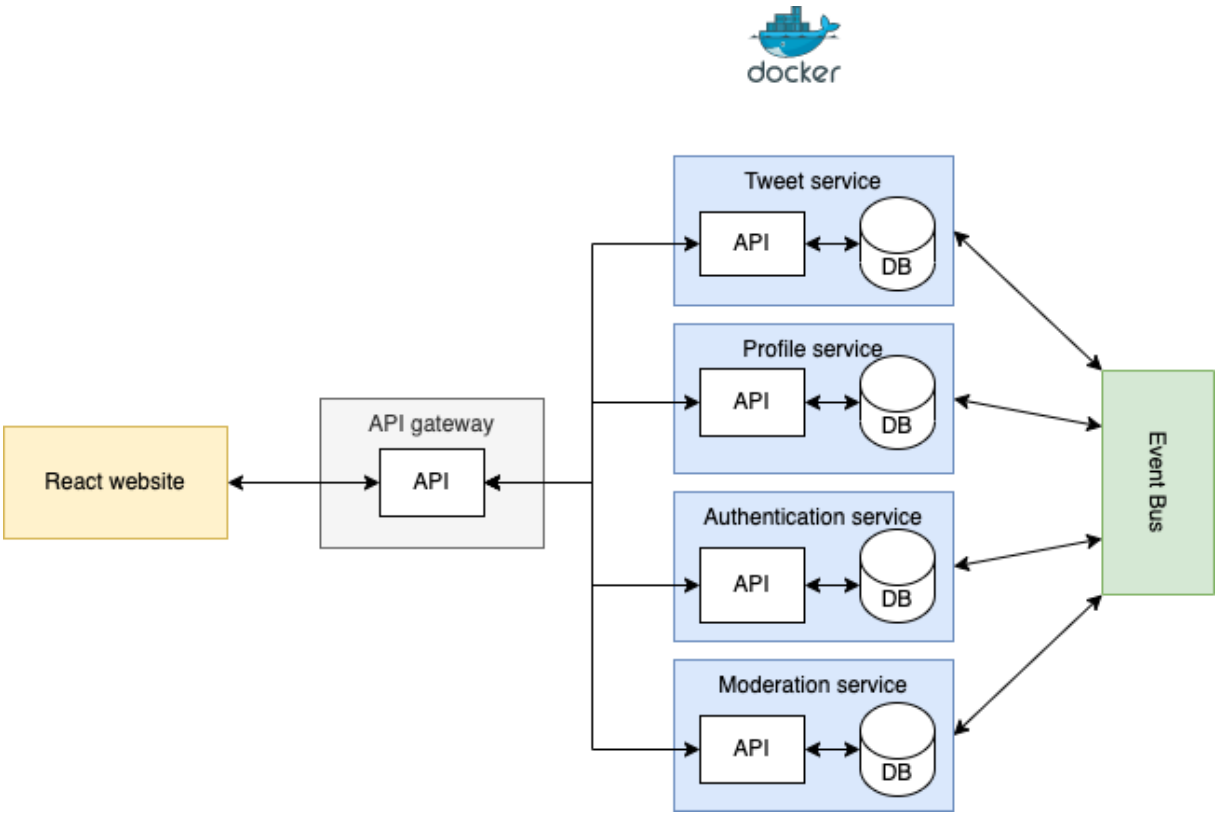

Figure 1 Global architecture

Security by design

Security has to be implemented right from the start. To make sure the security is implemented well only proven and well documented technologies will be used. Because the use of microservices, a microservice can be down without the other services being affected. With the development, the testing and the building of the software will also be automated. The OWASP security design principles will also be followed to make sure the most well-known vulnerability will be covered. A security risk- and impact analysis will also be performed to map out the possible security risks.

C4 models

Context diagram

Figure [2] shows the context diagram.

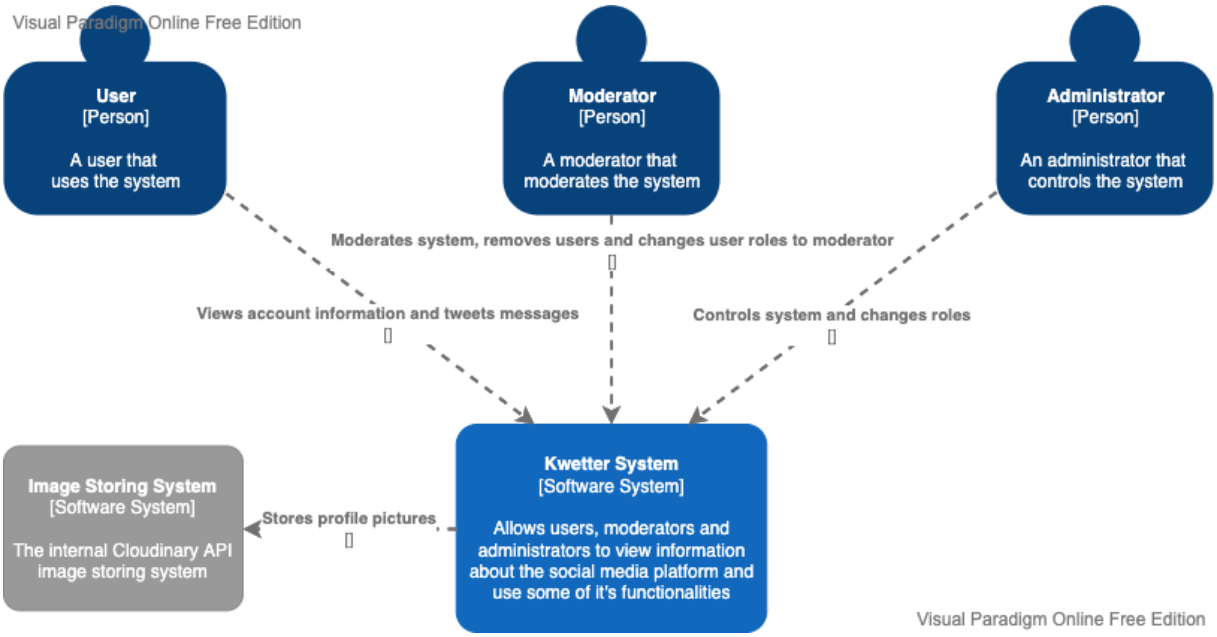

Figure 2 Context diagram

Container diagram

Figure [3] shows the container diagram.

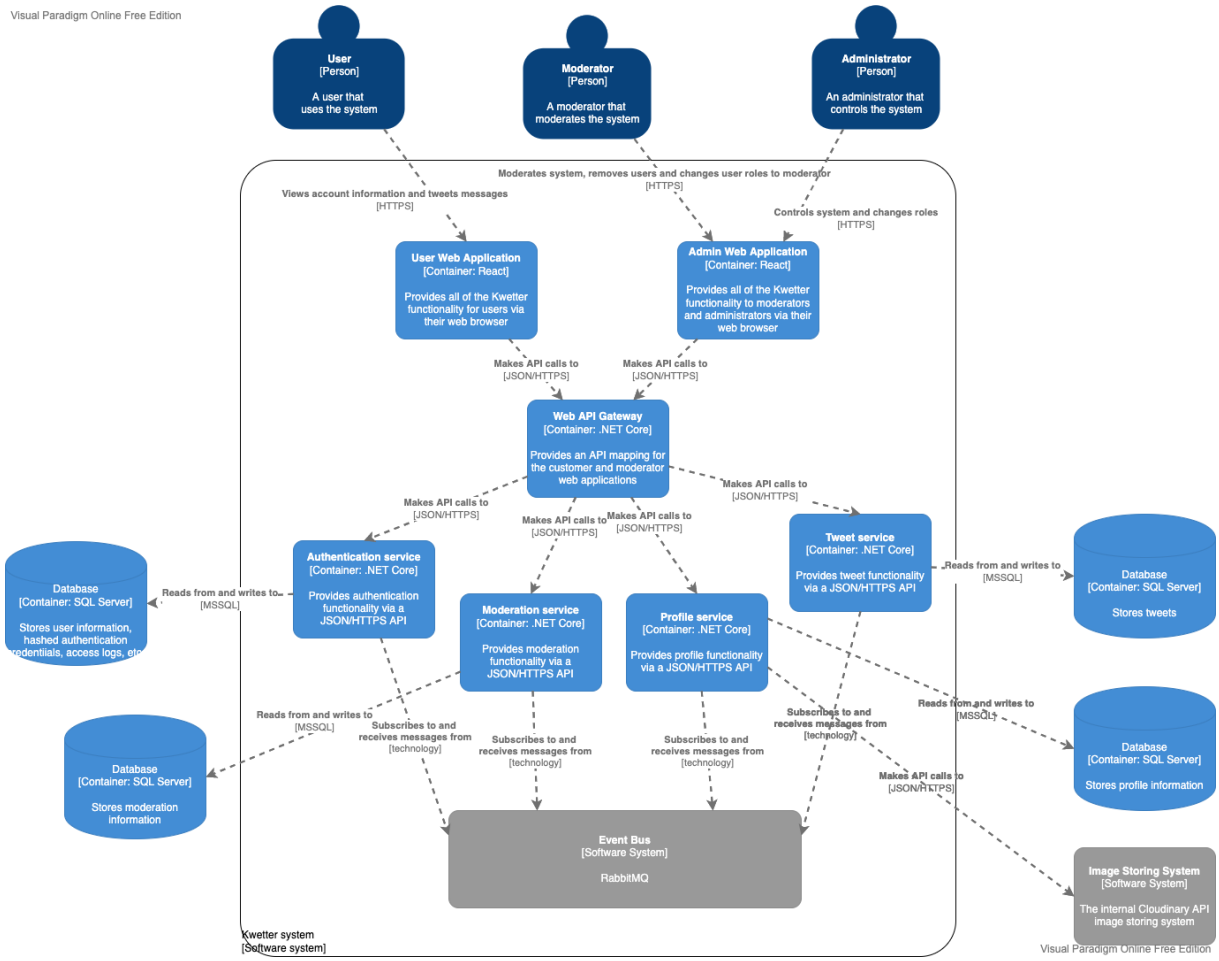

Figure 3 Container diagram

Component diagram

Figure [4] shows the component diagram of the authentication service.

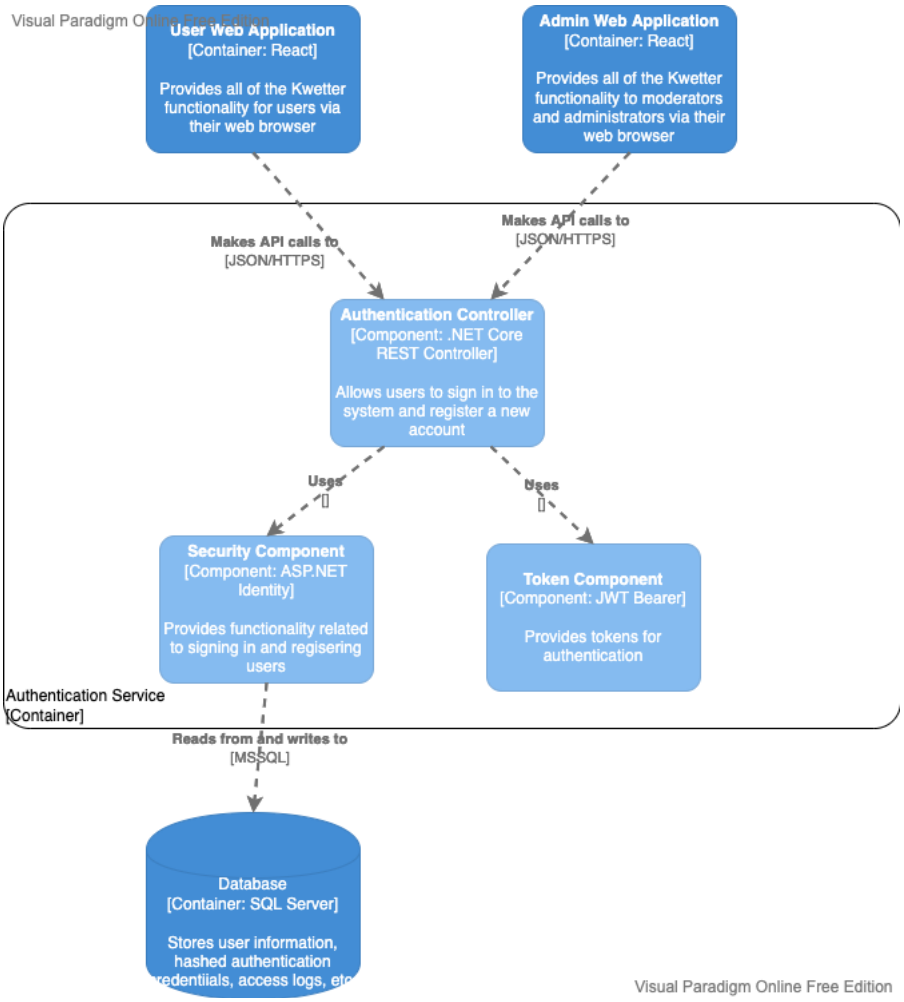

Figure 4 Component diagram

Deployment diagram

Figure [5] shows the deployment diagram of the Kwetter system.

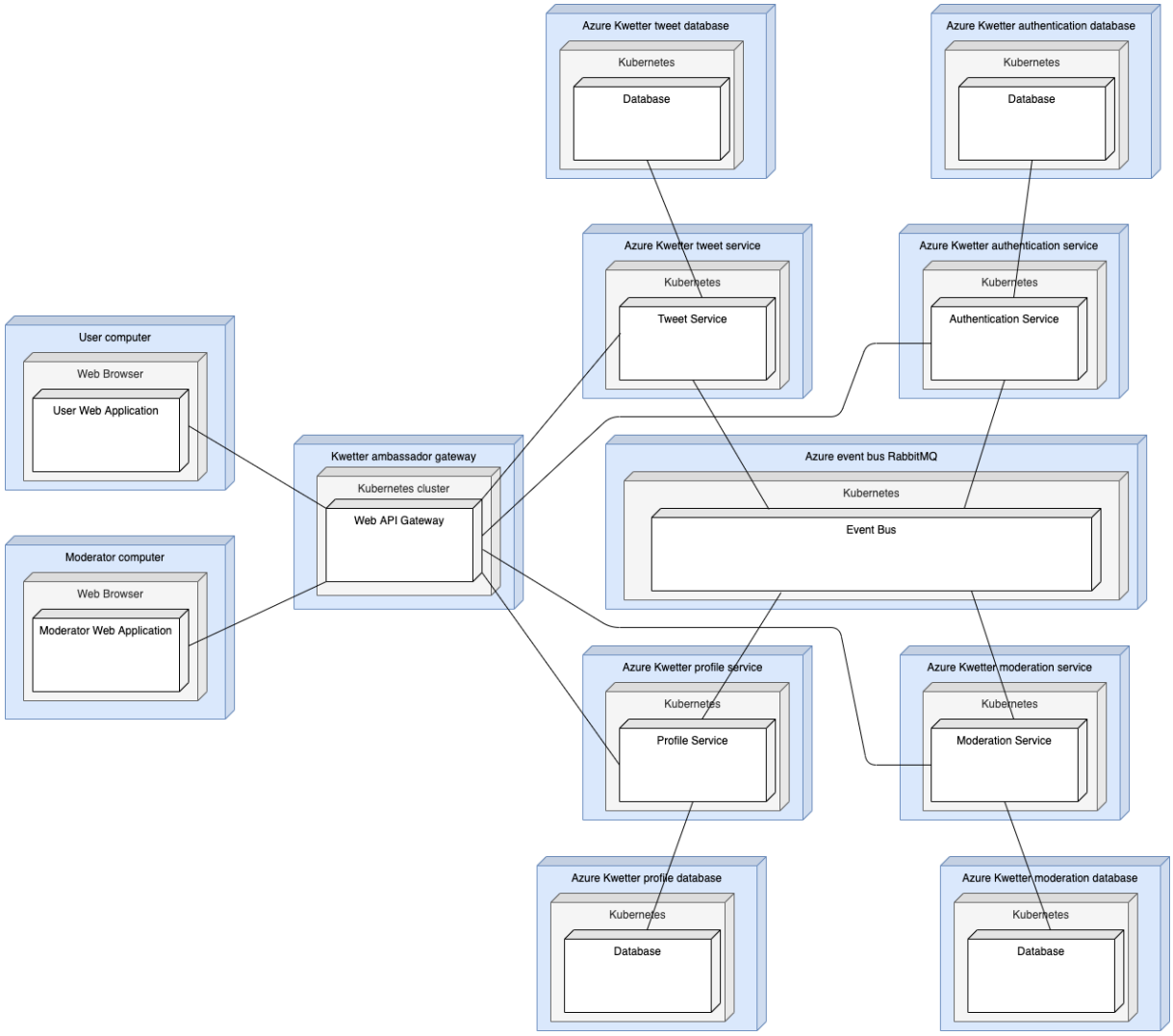

Figure 5 Deployment diagram

Class diagram

Figure [6] shows the class diagram.

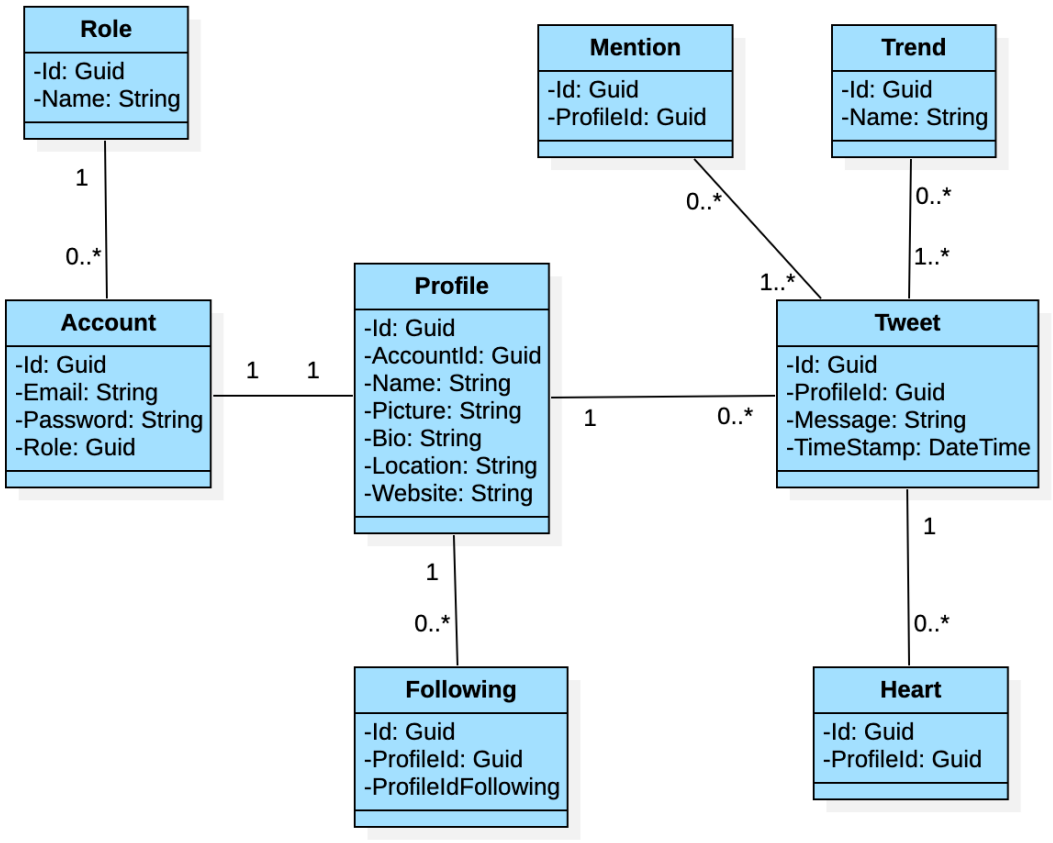

Figure 6 Class diagram

Database diagram

Figure [7] shows the database diagram.

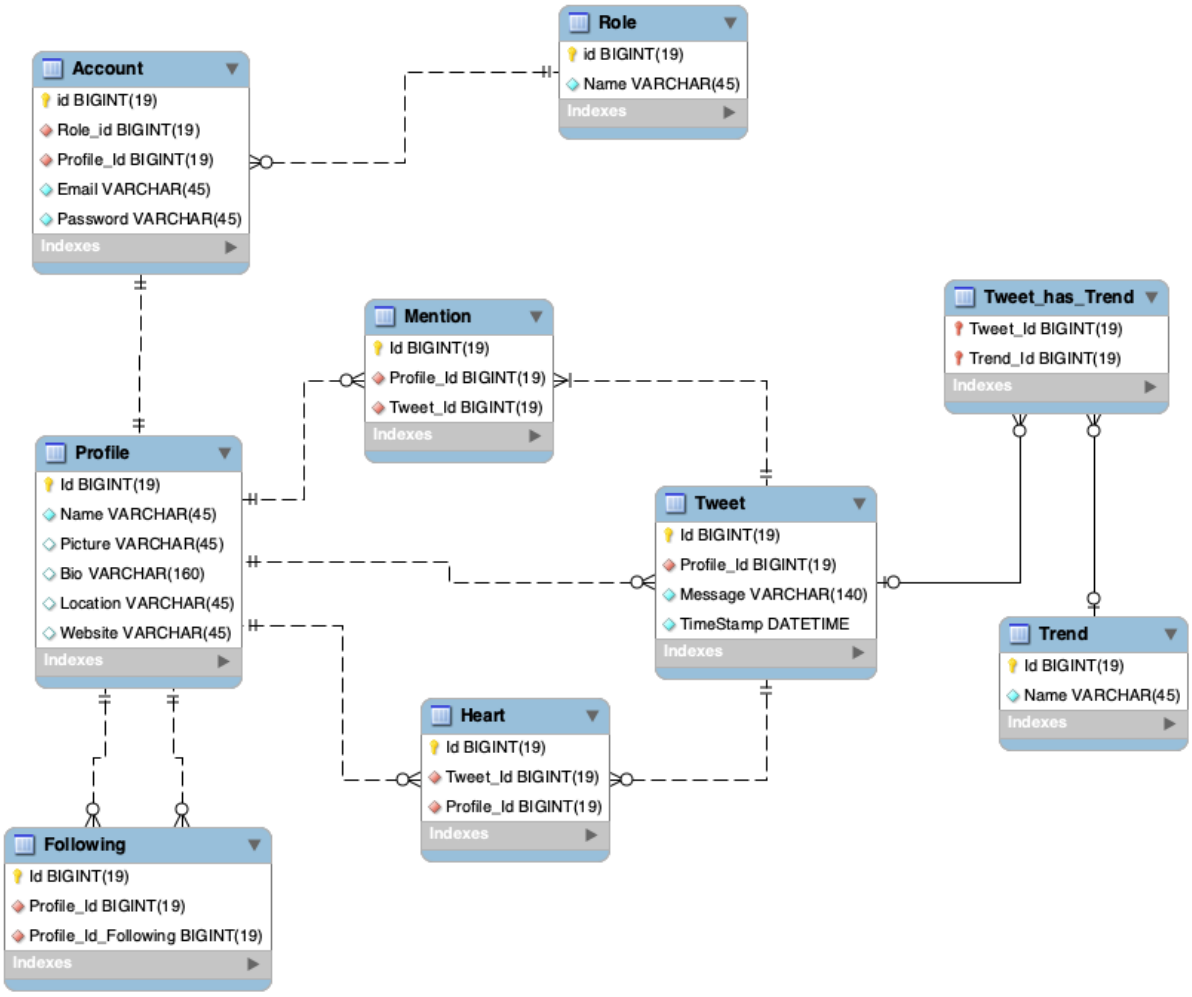

Figure 7 Database diagram