Design document

Design document for U2.

Created by:

Petar Bakalov – 4634705

Contents

Versioning table

Version	Date	Description
1.0	10/04/2024	Initial architecture

Introduction

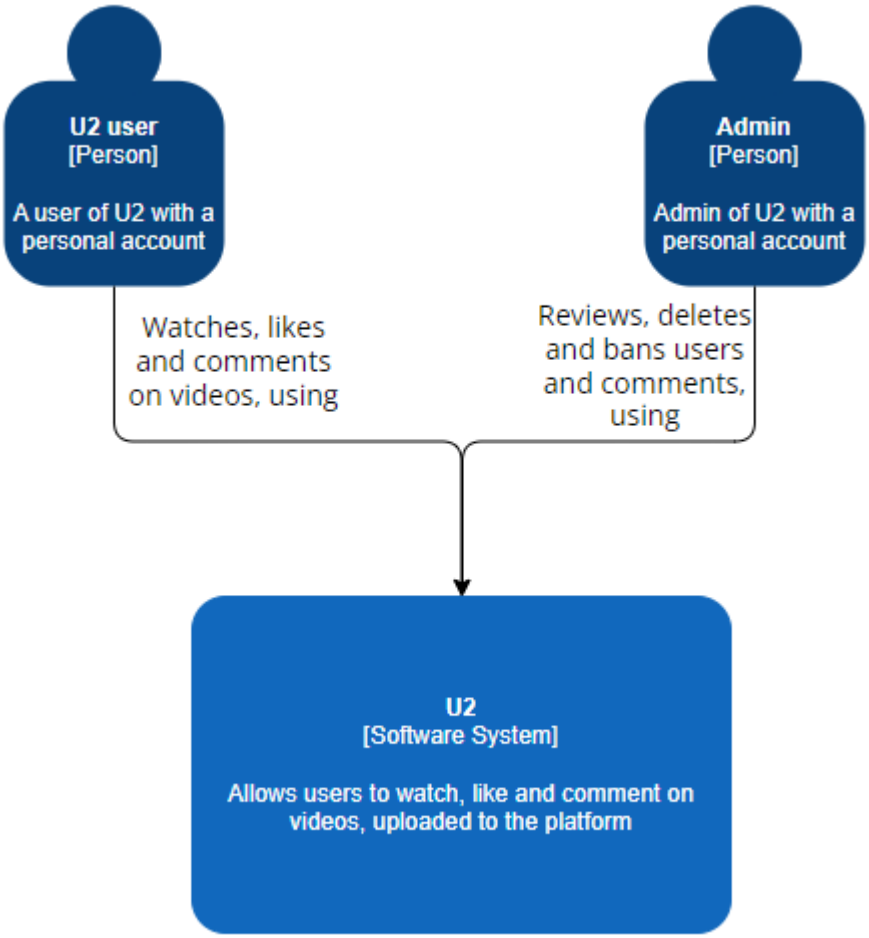
In this document I am going to present my C4 architecture of the backend of my application. The main given requirements are using microservice and data distribution architecture.

After brief research, the final decision is to use API gateway, which unifies and distributes requests to the appropriate microservice. Different services could be implemented using various technologies. Moreover, the front-end technology I chose is ReactJS, as it has component-based architecture, allowing to break your UI into small reusable components.

Architecture

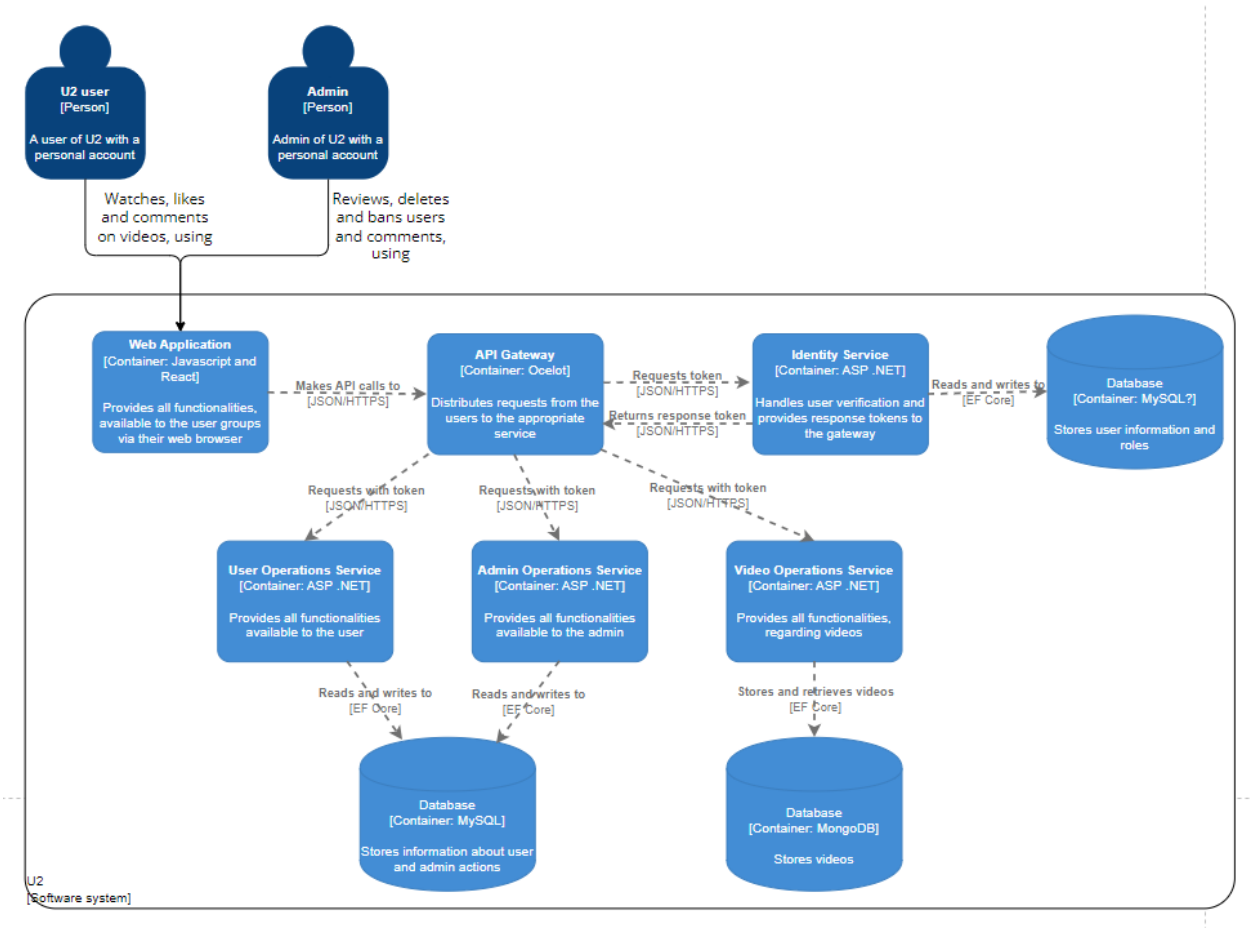
General microservice architecture with API gateway will be used for implementing U2. There could be more services added in the future, serving different purpose, which makes this structure scalable. However, since U2 is a YouTube mockup, it is going to be implemented as a single page application.

Level 1



There are two types of user groups that are using the application – users and admins. Users are able to create their profile and see videos that are uploaded on U2, performing a number of actions on a video. An upload functionality could be implemented in the future. Admins are able to see every video and all its comments. Moreover, they could delete a video, comment and ban a user.

Level 2



The backend solution includes multiservice and data distributive architecture with each service, responsible for single purpose actions. There is a service responsible for the operations of both user groups, as well as a service responsible only for operations, regarding videos. Moreover, the identity service provides response tokens, with which different operations could be requested by the users. For example, a normal user won't be able to delete a video, but an admin with access token would.

Justifying choices

React

Pros:

- Reusable components
- Performance Enhancement
- Supports many handy tools
- Easy to test

Cons:

- High pace of development

.Net API

Pros:

- Performance
- Allows easy connection with a database with EF Core
- Integration with Microsoft Services

Cons:

- Resource consumption
- Platform dependency (if using .NET Framework)

JavaScript

Pros:

- Speed
- Simplicity
- Interoperability
- Versatility

Cons:

- Client-side Security
- Lack of Debugging Facility
- Browser Support