JWT Authentication with Multiple Services

David Hellinga
Semester 3 Lente 2022

Introduction

For our project this semester we will have multiple separate backend APIs/services as part of a single web application. Account registration and login are handled by an authentication server that provides a JWT to the client device. The question we seek to answer in this research is how to authenticate that JWT when accessing the other services. It is important to note that we make use of our own API for account handling and thus OAuth is not an option.

Initial Research Questions and Methods

What method of authentication should we use for a webapp with multiple APIs/services?

1) What methods are there for authenticating on multiple services? (Literature research)
2) What implications do the methods have for our application's architecture and are these changes viable? (IT architecture sketching, reflection on 1.)
3) Prototype viable methods (Prototyping)

Q1: What methods are there for authenticating on multiple services?

Method: Literature research

(Ayoub, 2018) (*Microservice Authentication Guide: Tools and Use Cases*, n.d.)

Option 1: Dedicated Authentication Service

The first option is to have a dedicated authentication service that other services use to verify users. This means that every time a service receives a request that requires authentication it sends a request to the authentication service to verify the user's JWT.
Advantages:

1) Only the authentication service needs to know how to verify the JWT and thus the used secret key.
2) Only the authentication service needs access to the database with the user's authentication related data (email, username, hashed password, etc.)

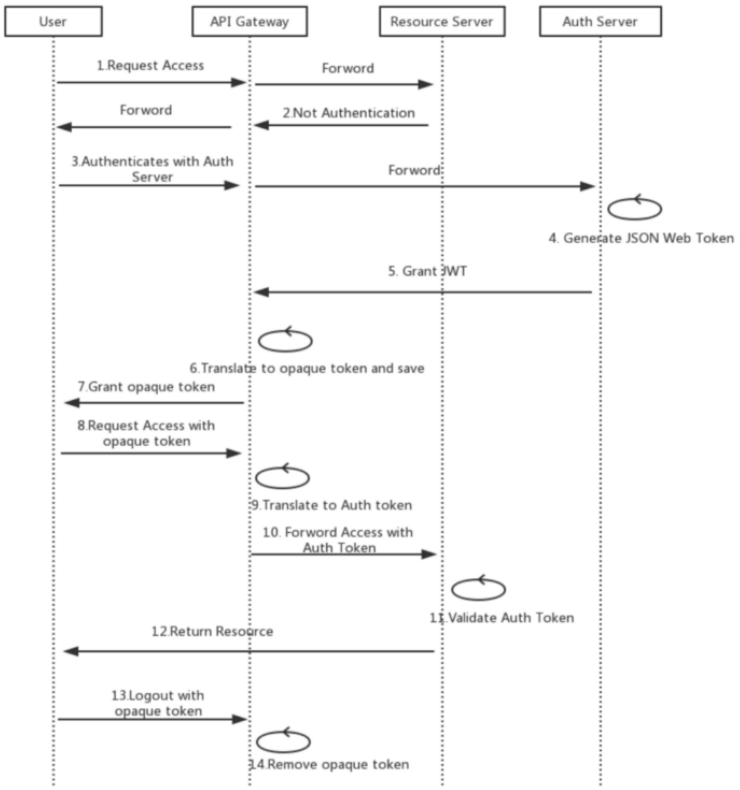
Disadvantages:

1) Generates a lot of trivial network traffic

2) The full verification process has to be done every time any service is used. If services are dependant on each other this might mean multiple authentications as part of a single end-user request
3) The authentication service is both a possible bottleneck and single point of failure

API Gateway

An API Gateway is a service that serves as a single point of access to the rest of the services that is only responsible for token translation and passing on requests to services. When a user signs in the gateway forwards the request to a separate auth server that generates a JWT that is only used upstream from the gateway. The gateway then translates this JWT to an opaque token - a token that only the gateway can read - which is given to the user. The user then uses the opaque token to authenticate requests. The gateway decodes the opaque token and forwards the request with the JWT to the requested service. That service can then use the JWT to validate the user.



API Gateway authentication (Ayoub, 2018)

Advantages:

1) Gateway encapsulates the internal structure of the application (Richardson, 2015) allowing the client to talk to a single access point, simplifying client code and allowing for client type specific APIs without changing the services.
2) Services are hidden from the user, improving security
3) Auth Token isn't exposed to the user. This means that data stored in the auth token is secure and stops possible decryption of the token on the user side.

Disadvantages:

1) Both a possible bottleneck and single point of failure
2) Extra work - every service requires manual exposing on the gateway

Shared Secret Key

Another possible solution to the issue is to share the secret key used by the Auth server between services. This way all services can decrypt and verify the JWT the user sends with requests.

Advantages:

1) Doesn't require an extra service
2) No single point of failure or bottleneck

Disadvantages:

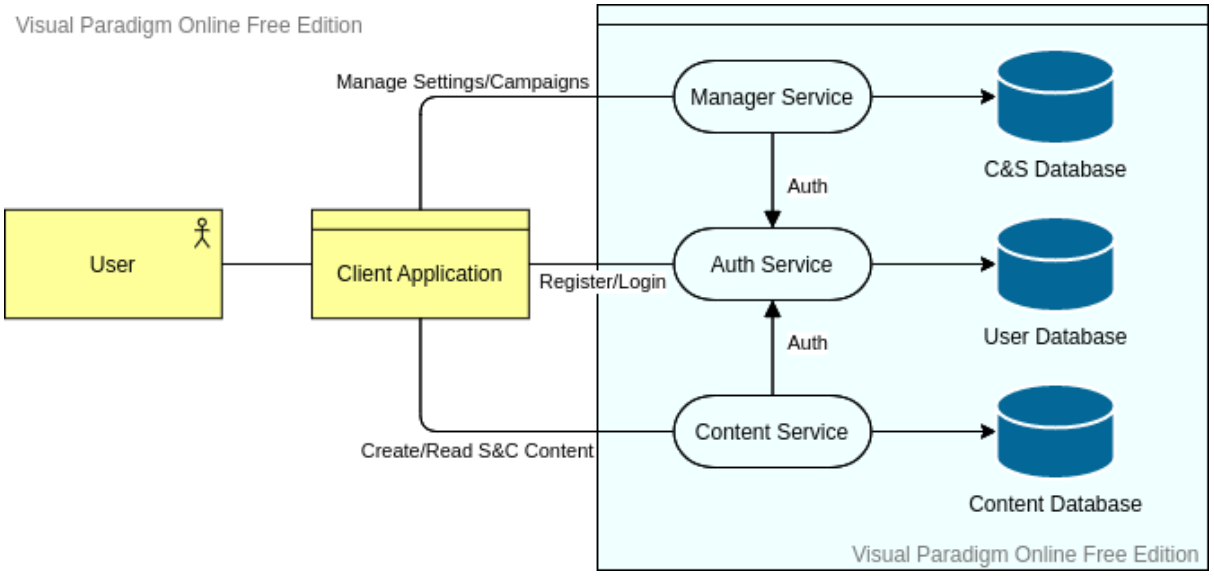
1) All services require functionality to decrypt the JWT
2) All services must have access to the secret key which requires some solution to share it.
3) Possible security vulnerability due to the secret key being available in so many places.

Q2: What implications do the methods have for our application's architecture?

Method: IT architecture sketching, reflection on Q1

Dedicated Authentication Service

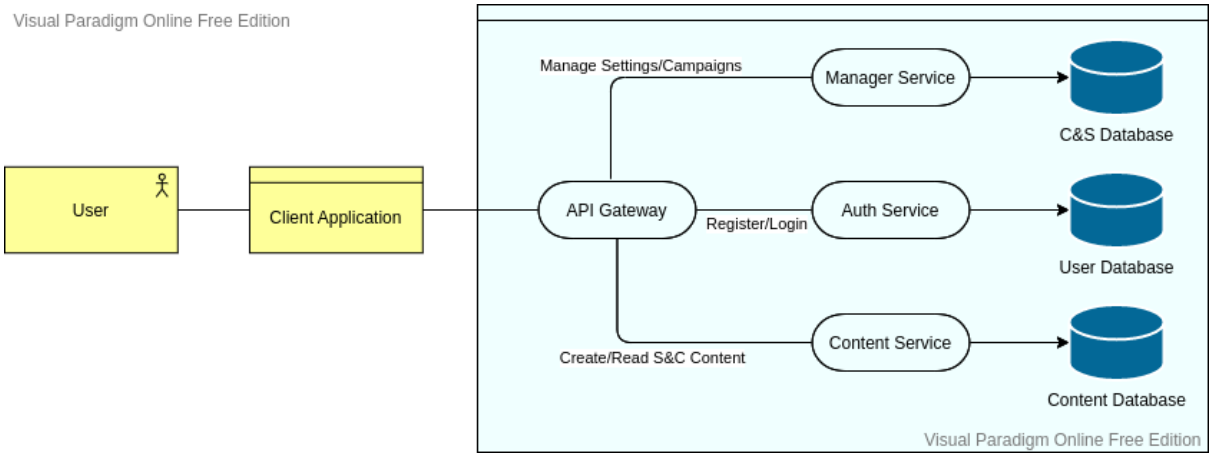
Within the context of the current project using a dedicated auth service would mean extending the current auth api to handle internal communication between services and resolve JWT verification requests. No other changes are needed to the current systems. See Mutual Authentication below about the security aspects of this.



Expanding the dedicated auth api is a viable solution for our project's scale. We have few services and the frequency with which users send requests is relatively low since we have no live functionality.

API Gateway

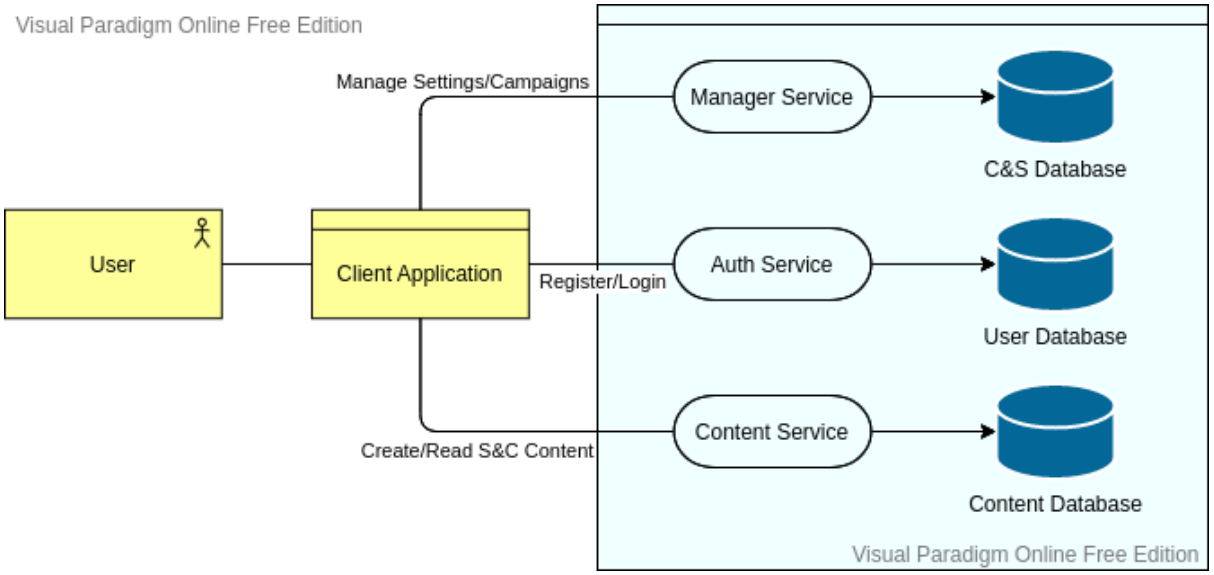
An API gateway would add a new layer in front of the existing bakend. This would require the implementation of the gateway as well as rewriting all existing endpoints to handle the new routing. See Mutual Authentication below about the security aspects of this.



Due to the scope of this project an API gateway is not viable. It adds a whole new service and requires changing multiple other ones. If we were planning to scale the project further this would most likely be the best solution but the extra work is not worth it with the little value it brings.

Shared Secret Key

No changes to architecture. JWT handers have to be added to services.



Sharing secret keys is viable from a technical perspective but we will not be choosing this method due to security concerns.

Sidenote: Mutual Authentication

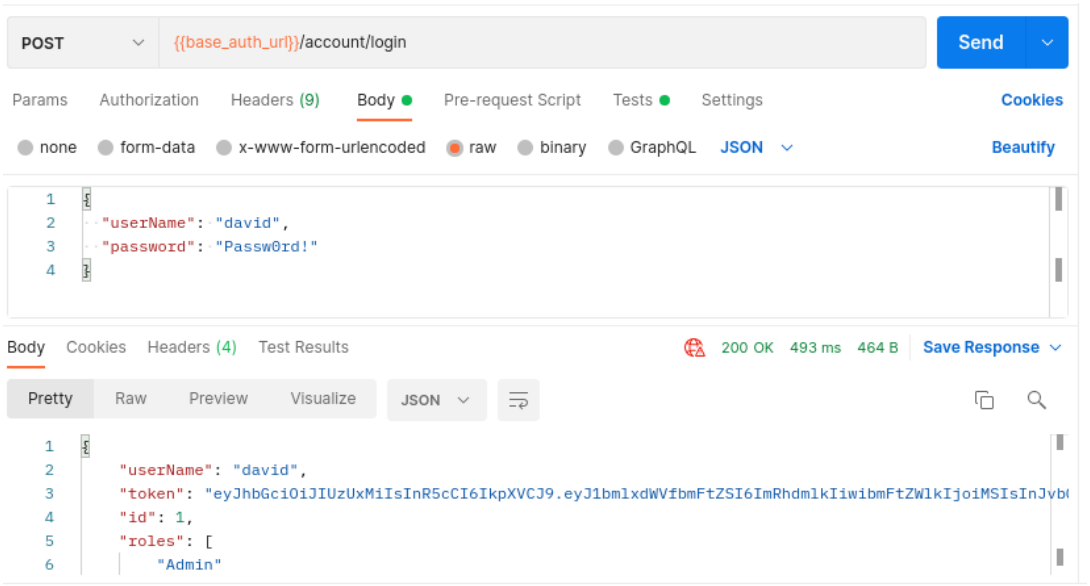
In an environment where services have to communicate with each other it is necessary for those services to be able to securely identify each other and communicate. An example option for this would be Mutual SSL with an internal certificate authority. Due to the complexity of this issue and the extra research questions it raises we will have to research this separately. For development purposes we can leave out mutual authentication for now as it does not impact functionality, only security. This will have to be corrected before release to production environments.

Product Backlog for resolving this issue here.

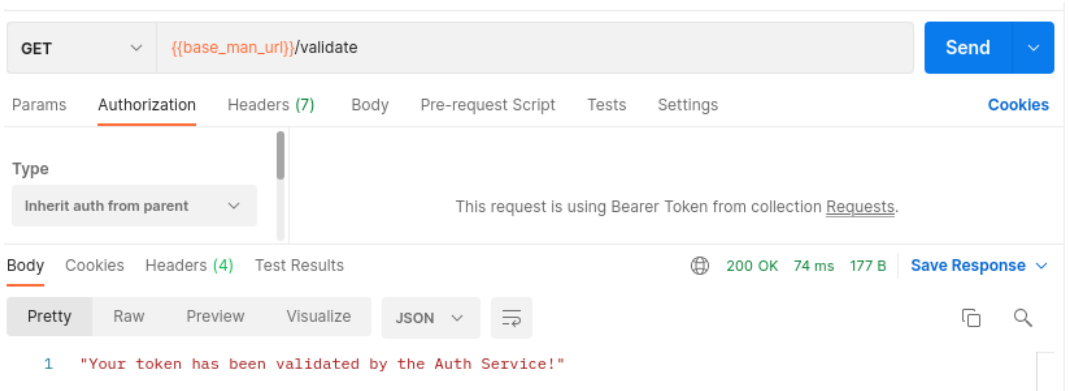
Q3: Prototype viable methods

To verify that an authentication service would work we created a prototype API (Python, FastAPI) that, when it receives an API call, takes the user's Bearer token and sends it to a new endpoint on a fork of the existing Auth Service. The AS then verifies the token and returns the user's ID if the token is valid or throws unauthorised if it isn't. See the Repo for the code.

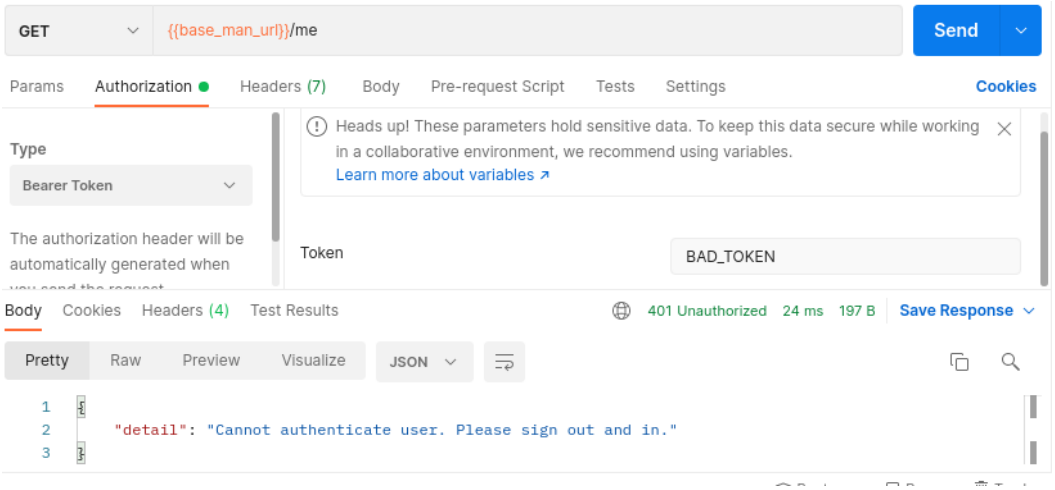
To test the prototype we first sign in with an account. This gives us a token to test.



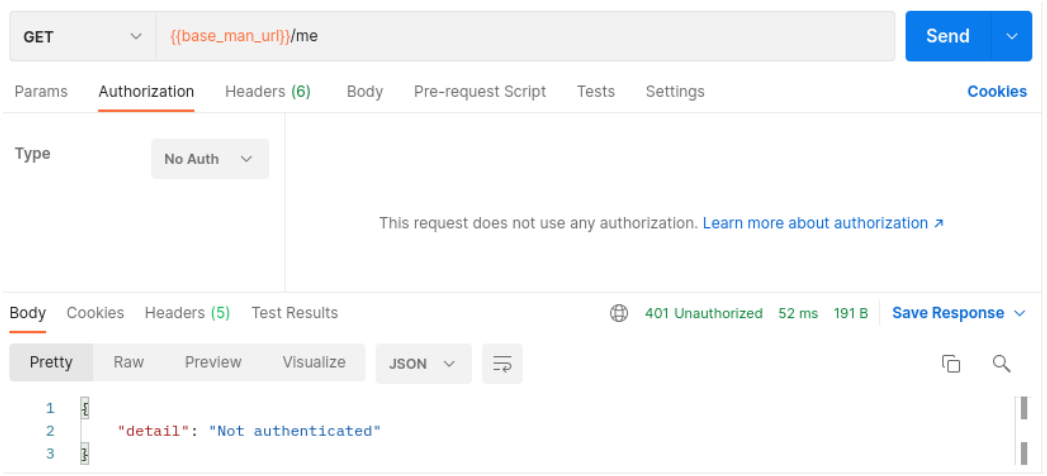
We then send a request to the prototype API to validate the token. This uses the previously acquired token in the header and the server responds that our token has been validated - the custom response we set in the prototype API, not just a forwarded message from the Auth Service.



If we then try to send a request with an incorrect Bearer token in the header we receive an error that we are unauthorised and to please relog.



Sending the request without a token gives us a simple Not authenticated - meaning that the prototype doesn't try to query the Auth Service since no token was found.



Conclusion

As demonstrated in Q3 a dedicated authentication service provides a working authentication system that is viable and suitable for FOTL. In practice we would have to use an internal certificate authority to secure communication between the Auth Service and other services such as the prototype, but due to time restrictions we won't be able to do that here.