Department of Computer Science

CS 411 – Software Architecture Design

Project 1 Report

Abdul Razak Daher Khatib – 21801340

Efe Şaman – 21903248

Çağla Ataoğlu – 21902820

Alperen CAN - 21601740

Instructor: Haluk Altunel

Teaching Assistant: Cihan Erkan

09/10/22

CONTENTS

CONTENTS **1**

Chosen project **2**
- Reason 2

Current Architectural Style **2**
- Problem Definition and Domain 2
- Technical Problems and Solutions 3
  - Groups' Memberships Mismatch with LDAP 3
  - Sessions Management 3
  - Single-tenancy Performance Issues with Large Numbers of Realms 5
  - Bulk Updates to the REST APIs in Keycloak 6
  - Readability Complications of Deployment and Property Configurations 6
- Stakeholders 7
  - Developers 7
  - Contributors 7
  - Users of Keycloak 7
  - End Users of Keycloak 7
- Quality requirements 8
  - Performance 8
  - Security 8
  - Reliability 8
  - Usability 8
  - Customization 8
- UML Diagrams 9
  - Use Case Diagram 9
  - Activity Diagram 10
  - Component Diagram 11
- Rationale 12
- Architectural patterns 14
  - Broker Pattern 14
  - Client-server pattern 15
  - Policy Enforcement Point Pattern 16

Proposed Extension **17**
- Evolution Area 17
- Architectural Change 17
- Rationale 18

References **20**

Chosen project

Name of project: Keycloak

Reason

Keycloak is an interesting open source project since it deals with a sensitive and critical process; user authentication and accounts management. The process of managing users and authenticating them requires a mix of quality requirements, like security, privacy, performance, reliability, and usability. Additionally, the project is written in Java, where design and architectural patterns are especially emphasized. Finally, the project is well-documented and the official pages are resourceful, making it easier and faster to understand.

Current Architectural Style

Problem Definition and Domain

Keycloak is an open source project that provides authorization, authentication and user storage services. It allows single sign on, which means that users don't need to sign in to individual applications, they can sign in to them at once using Keycloak. The project promises high security and strong, reliable authentication. Its aim is to provide robust authorization for applications. Keycloak uses a variety of methods to manage users and their authentication. One way is creating a Keycloak account and choosing a password, another method is to use Social Identity Providers to manage the login process, finally, another common way that Keycloak uses is One Time Passwords. In order to store the accounts and their passwords and information Keycloak uses either LDAP/AD integration, SSSD and FreeIPA, or Custom Providers. The most common way to manage accounts is through LDAP. Users have attributes, like email, username, etc. But more attributes can be mapped using LDAP. Common examples of these additional attributes are roles and groups.

Technical Problems and Solutions

Groups' Memberships Mismatch with LDAP

LDAP stand for "The Lightweight Directory Access Protocol (LDAP /ˈɛldæp/) is used a mechanism to interact with directory servers with specific standards. LDAP can be used as a directory to keep the users and their passwords [1]. KeyCloak can be integrated with it and use LDAP to validate users.

The problem is, when a group is updated and users added in LDAP are not automatically updated and periodic synchronization is not available, resulting in a mismatch between LDAP and KeyCloak groups [2].

Per KeyCloak documentation LDAP mappers manage groups and their memberships. Additionally, if the "import" option is enabled synchronization for users will happen automatically. However, the only time LDAP mappers are imported are when the users sign in or when the user is queried by admin [3]. When this happens the LDAP provider will import the users and their info into KeyCloak, and then authenticate the user. While an admin can sync users on-demand or periodically, there is no such option for groups.

While this decision architecturally is logical since it will not force admins to import a large number of users at once, it has its downfalls; if groups of existing users are updated KeyCloak will not see the update immediately. As a consequence, if some users depend heavily on groups this issue would be a major problem that can prevent them from using the system altogether.

The solution proposed is to enable an optional automatic or an on-demand synchronization for LDAP mappers that map the users and their groups. This solution is suggested since it will not cause the original problem that this mechanism tried to avoid but at the same time solve the problem discussed above, especially that groups' changes are not as common as users changes.

Sessions Management

As expected a sophisticated system that manages users' logins will have to make difficult decisions when it comes to managing sessions. Two problems related to session management and tokens will be discussed here.

When the user first is authenticated three tokens are obtained: *identity*, *access* and *refresh*. Access tokens expire within minutes while refresh tokens obtain a new access token every time

it expires. If the refresh token is revoked the session will end as soon as the last access token expires[4]. The first problem arises when a user has more than one tab opened of the same account [5]. When a user opens the first tab and logs in, a browser session starts and another session for the tab also starts. Afterwards, every new tab starts a tab session only that is seemingly “independent”. However, when the user signs out from one tab both the tab and the browser sessions get terminated, revoking the refresh tokens of the other tabs and consequently logging them out. While this behavior can be justified some users found it confusing. Another issue that users face is that when the user logs in a new tab after the first tab, they will have to complete the whole log in process before they are told that they are already logged in and then redirect them to the application [6].

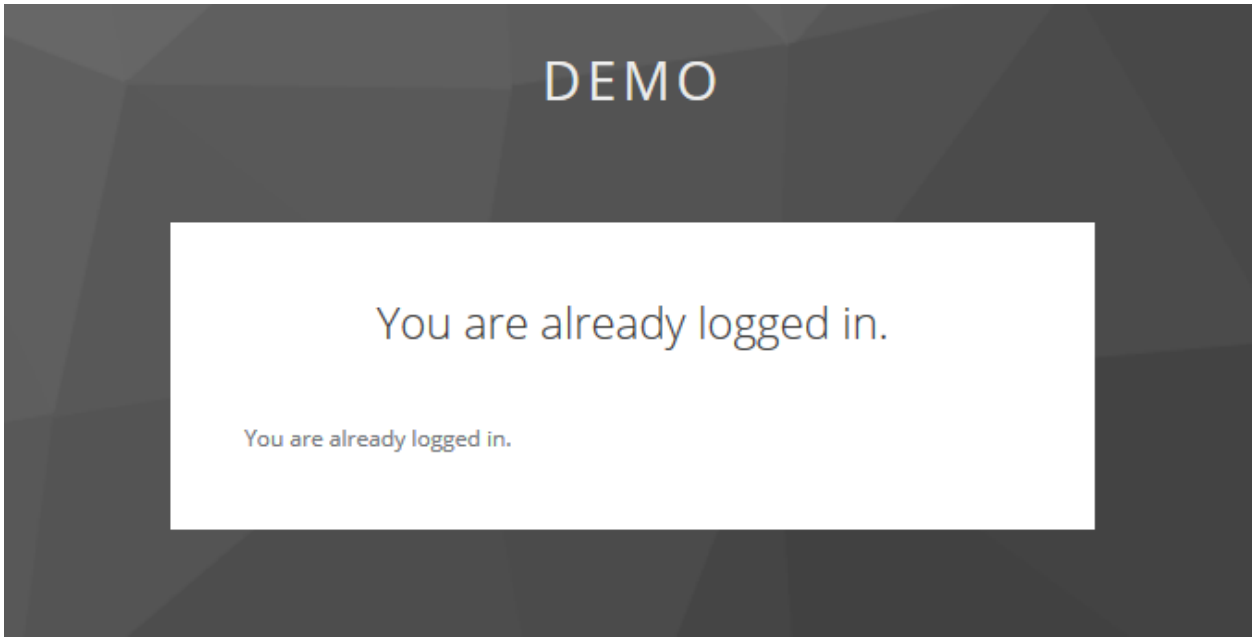

Figure 1: “You already logged in”

The problem is caused by the fact that the browser session’s token gets deleted shortly after being obtained, so the browser does not remember the fact that the user logged in already until they get a new browser’s session token [7]. While the current behavior saves memory by not keeping the token it causes a confusing behavior for users and forces them to go through unnecessary login flow.

The solution we are proposing is regarding the problematic rowser’s token. First, the token should not be deleted immediately after it is obtained, this will prevent the unnecessary login in flow. Second, the token should not be deleted with the first tab’s session termination but rather

with the last one. This will prevent the refresh token from being revoked, as a consequence it will create real “independent” tabs’ sessions.

Single-tenancy Performance Issues with Large Numbers of Realms

Keycloak realms do not support multi-tenancy. How Keycloak uses multi-tenancy is by having multiple ‘realms’. Different domains for applications can be implemented through realms. Realms, as the name suggests, act as a ‘realm’ for the domain. These realms have users, groups, and roles implemented in them, and they can have multiple applications/clients configured in them. For example, a customer realm can have some application configured that customers can access.This is presented in the diagram below. Relations between the users and applications in a realm are explained as the following by hcl tech: “Each user from a particular realm talks to a client application configured as a client within a realm for authentication” [8].

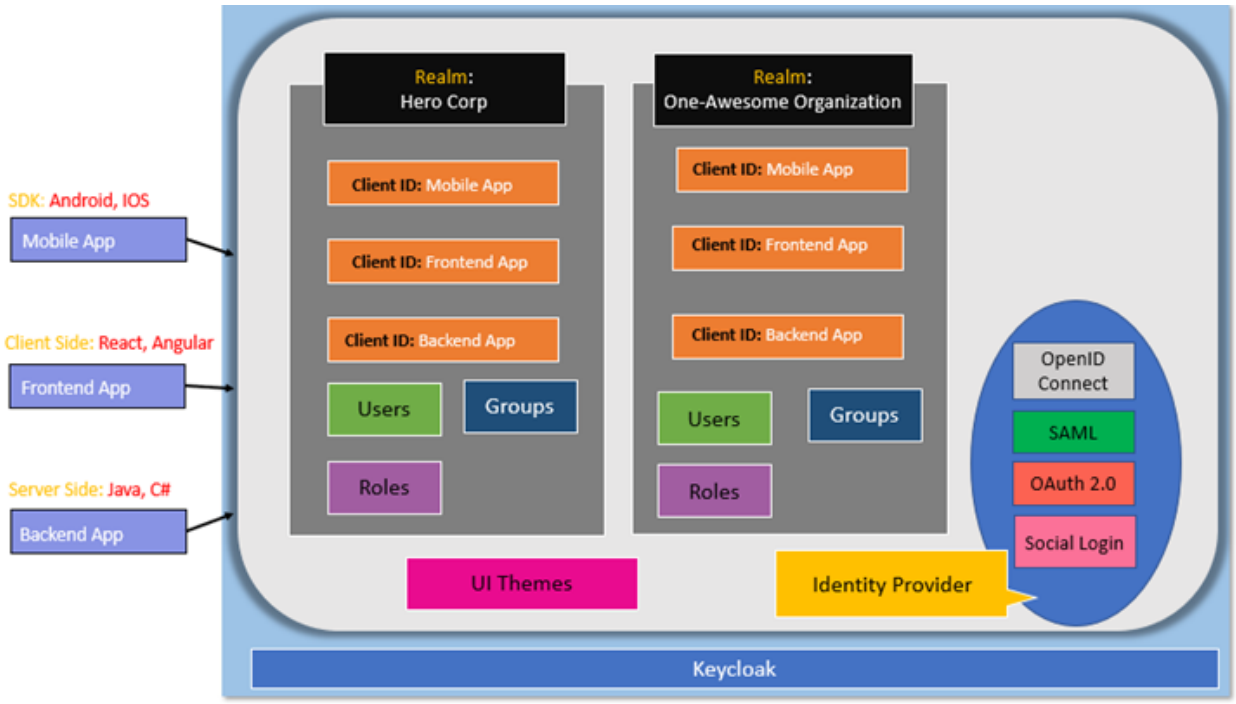

Figure 2: KeyCloak Realms Diagram [8]

As every realm is implemented with a different configuration, every realm would introduce their own applications as clients. For small numbers of realms, this doesn not cause performance issues, however for greater amounts of realms (such as more than 400), start causing

performance issues to the system. [9] discusses performance issues caused by the high numbers of realms for certain applications. Having separate application implementations in every realm is single-tenancy, which is having different application instances between the user and database for every different use case. However, this can be implemented in other ways, such as multi-tenancy.

Bulk Updates to the REST APIs in Keycloak

Updates to users and realms cannot be done in bulk. This is a problem because for many users, doing updates as a batch would be much faster and easier. This is pointed out in the github issues section of the project: https://github.com/keycloak/keycloak/issues/9316. Keycloak uses RESTful APIs to update their databases. These updates are made one by one to every row, rather than in bulk. A solution to this is to add a new PATCH HTTP method in their API so that the update can be made in bulk. The PATCH method allows the user to propagate a list of values in a RESTful way.

Readability Complications of Deployment and Property Configurations

Property configuration is not separate from deployment. This negatively affects code manageability. Moreover, the mixing of configuration and deployment methods decreases readability. In KeycloakDeployment.java (https://github.com/keycloak/keycloak/blob/main/operator/src/main/java/org/keycloak/operator/controllers/KeycloakDeployment.java) containing Keycloak's deployment functionality, configuration related methods are included as well. It would be much better for readability's sake if configuration behavior was separate from deployment. Users of Keycloak have recognized (https://github.com/keycloak/keycloak/issues/14739) this readability/manageability issue as well.

The solution to this problem would be that config methods should be in a separate file from deployment methods. This would not only make the deployment class file much shorter, but it would mean that as more deployment and config methods are added, the class does not become overwhelmingly large. This would benefit developers/contributors who write code and users who read code. Additionally, according to the single responsibility principle, it's good practice to have classes that "do only one thing", either deployment or configuration, not both [10].

Stakeholders

Developers

The developers are the main stakeholders in this case. They are responsible for most of the contributions to the project's code. Developers fix bugs after users report them, add new requested features, re-structure the project whenever it is necessary to keep up with new technologies. Developers are of an even greater importance for open source projects because of the low or non-existing funding.

Contributors

Since this is an open source project, there are many other contributors who send pull requests to the project's repository. For these external contributors, readability is very important because they're trying to understand and edit other people's code. Moreover, they expect a well organized architecture so that changes can be made much more smoothly. Contributors can be other developers or users that want to work on a specific feature or fix a bug that faces them.

Users of Keycloak

Users of Keycloak are admin of other applications looking for a safe, and easy way to manage their users and authenticate their logins and registrations. Users mostly care about how well the project works and how fast and secure it is. They expect the features to satisfy their application's requirements. These users look for an easy to implement, configure and deploy system to integrate with their applications, as well as, an easy-to-use, fail-proof, and organized system to satisfy the needs of their end users.

End Users of Keycloak

The users of these aforementioned applications also have expectations from Keycloak. Since the applications store their authentication data, Keycloak needs to be safe and secure. They want the authentication process to work in a fast, private, secure and user-friendly way. These users vary greatly in their technical knowledge since all kinds of applications can use Keycloak, so user-friendliness is of great importance to them.

Quality requirements

Performance

End users would expect the sign on to be simple and quick. For this purpose, the system needs to be fast and perform well. Requests should be handled as quickly as possible and critical user journeys should be kept simple and easy to use.

Security

Authentication involves handling sensitive data, most importantly passwords. It's crucial for a service like Keycloak to be extremely cautious on how they transmit and use this data. Architectural designs that prioritize security should be preferred. This way, leaks are prevented and user trust is strengthened.

Reliability

Authentication process is a sensitive and crucial part of any application with registered users. Thus, such systems need to be fail-proof and meet the users needs.

Usability

The end users of authentication systems are users of any application that uses them, in other words all demographics. Because of this, usability is crucial. The end user should not be expected to have any technical knowledge to be able to use this service. The process must be simple and smooth.

Customization

Applications that use Keycloak come in a variety of types. It's not realistic to expect vastly different architectures to use Keycloak the exact same way. Because of this, it's important to allow users to customize the services to their application's needs.

UML Diagrams

Use Case Diagram

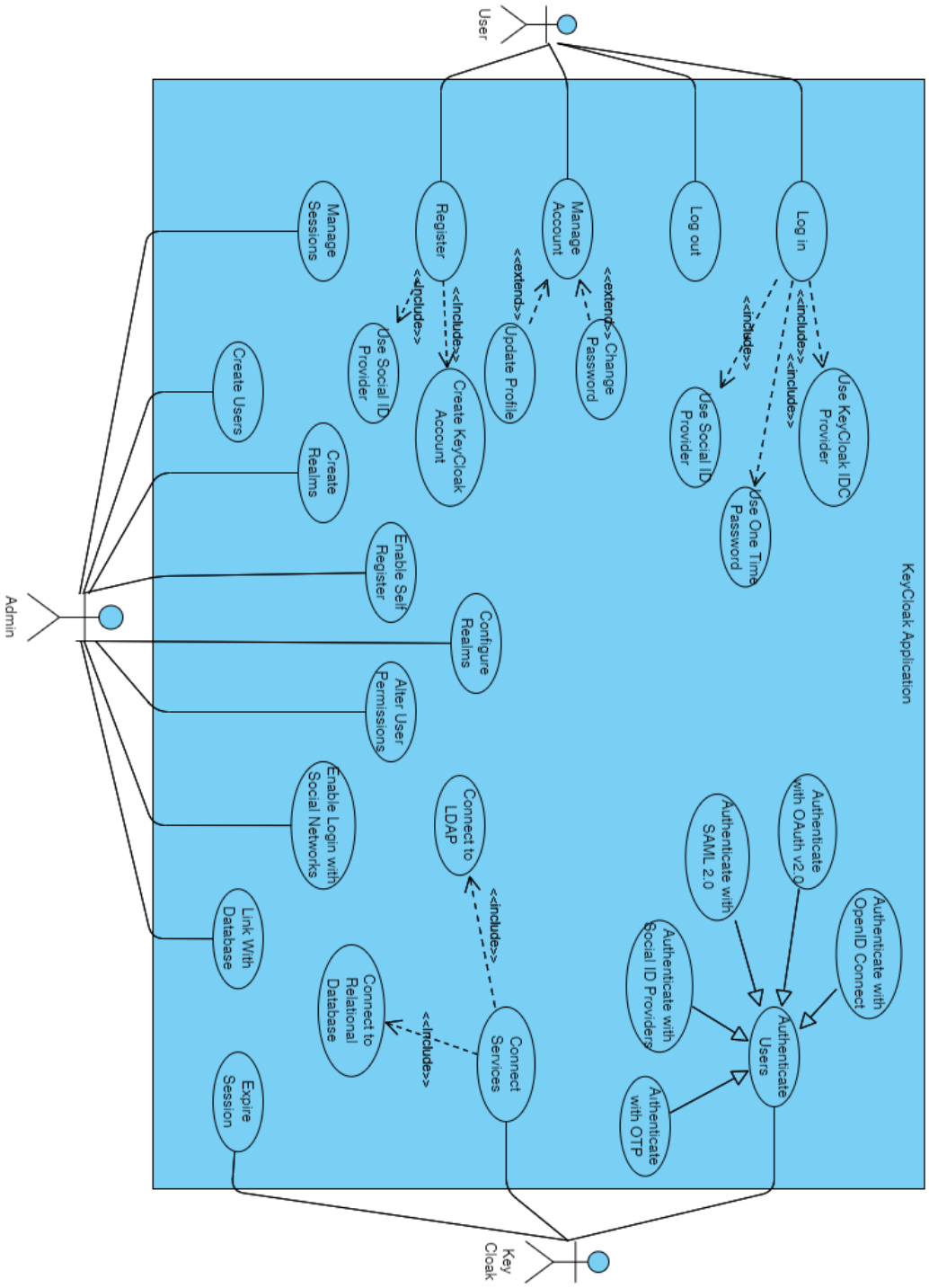

Figure 3: Use Case Diagram

Activity Diagram

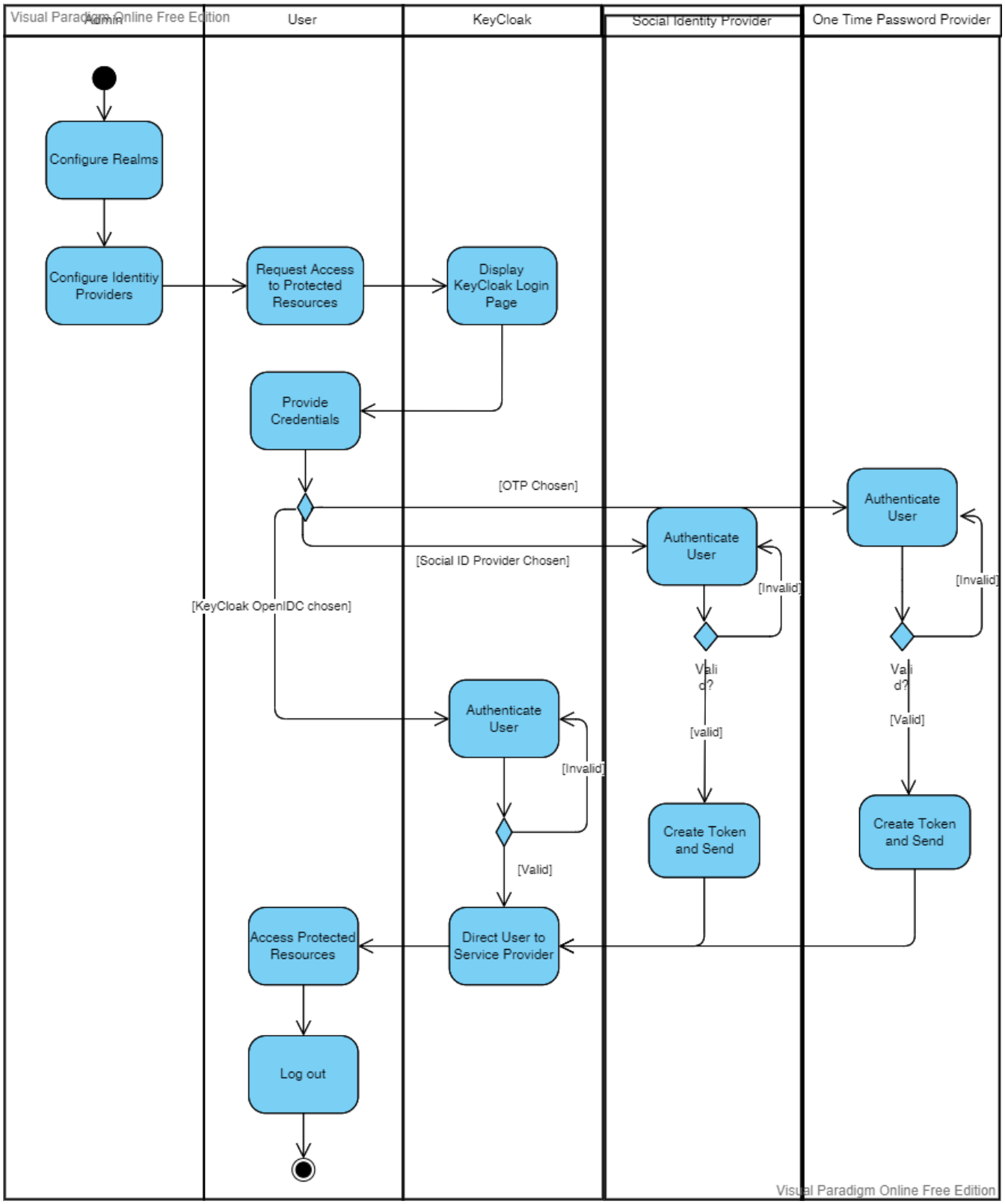

Figure 4: Activity Diagram

Component Diagram

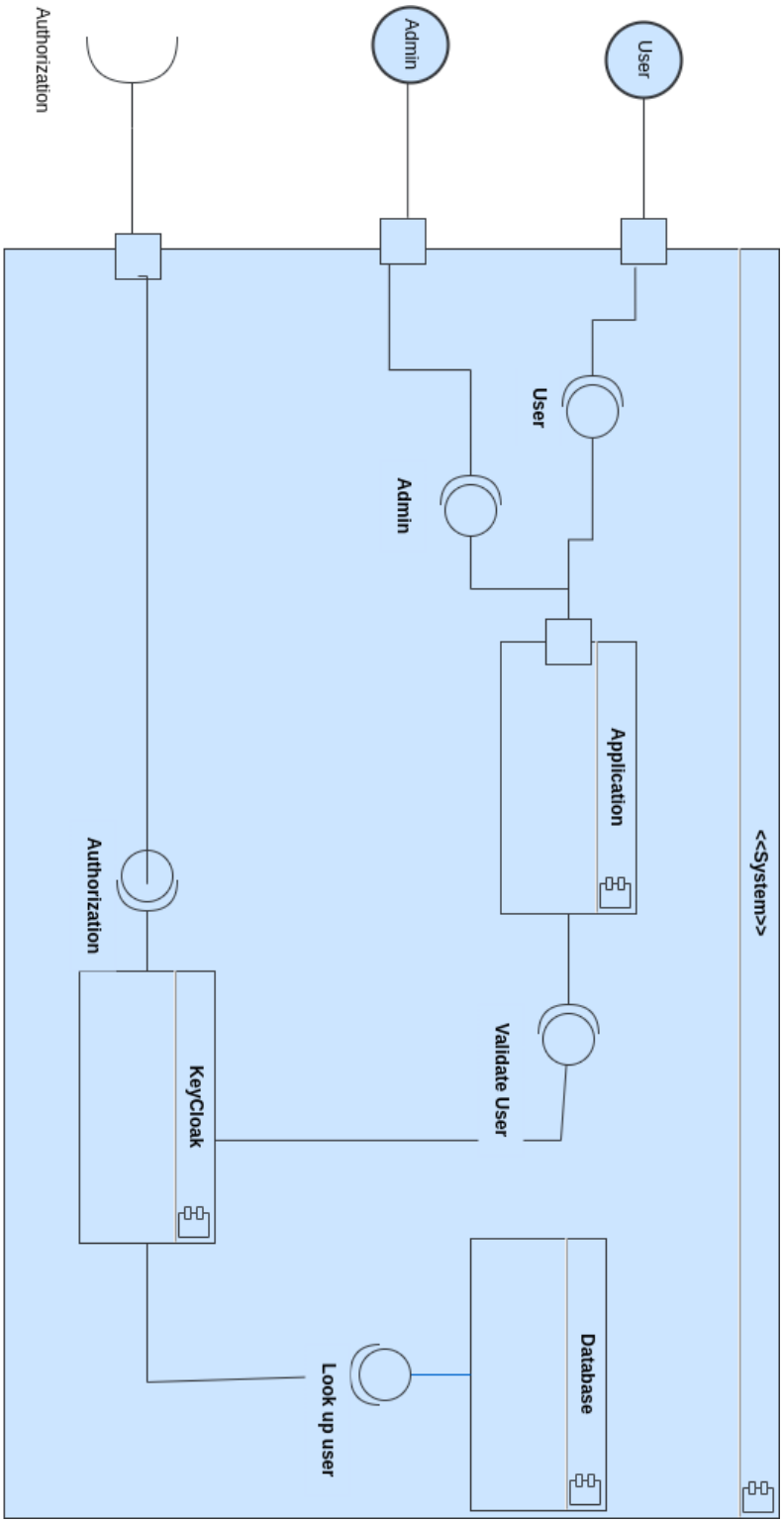

Figure 5: Component Diagram

Rationale

The rationale behind the architecture of KeyCloak is based on providing privacy, performance, reliability and security.

For decisions related to privacy see Figure 1 below, which shows the architecture for authorization services. The applied layered design style in this architecture benefits in terms of access rights, data privacy, and also separation of concerns. Due to the limited access to the lower tiers, the storage has been kept in the lower tier for privacy purposes.

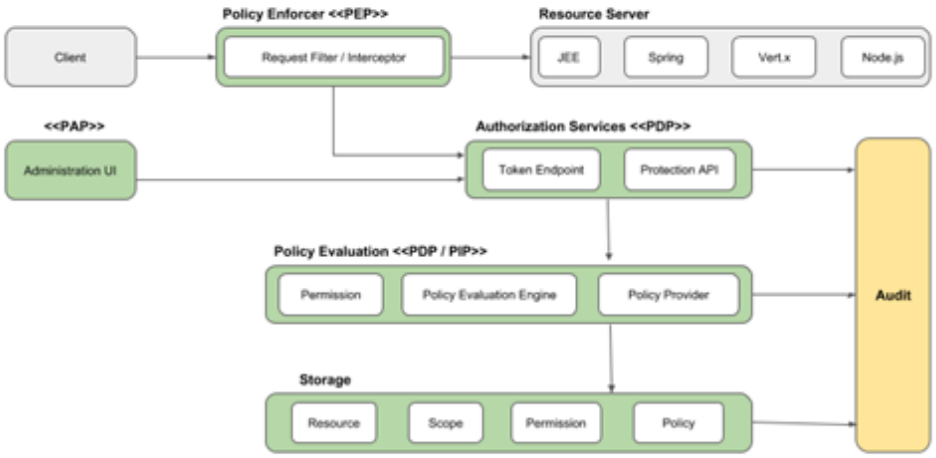

Figure 6: Overall Architecture of Authorization Services of KeyCloak [11]

For decisions regarding performance and reliability see Figure 2 below, which is the example architecture diagram for cross-site replication mode. Cross-site replication allows backing up data from one cluster to other clusters, mostly settled in different geographical locations. A database is used to persist permanent data like user information. On the other hand, infinispan cache is used to cache short-lived and changing data which is much faster than database, but it is not expected to be permanent when cluster is restarted. At this point, KeyCloak offers two

options: reliability which is used in Active/Active mode, and performance which is used in Active/Passive Mode.

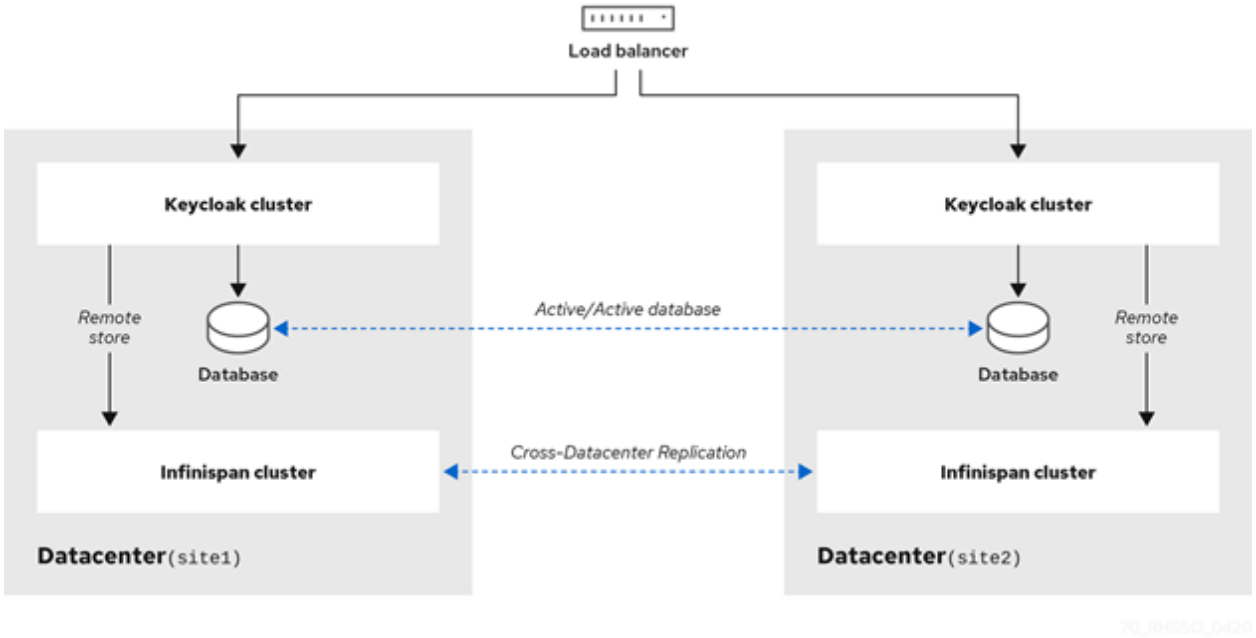

Figure 7: Architecture Diagram of Cross-Site Replication Mode of KeyCloak [12]

For decisions related with security see figure 3 below, which describes the realms and applications in KeyCloak. The aim of this design is to isolate realms from one another, and enable them to only manage and authenticate the users that they control. Owing to this security model, user accounts are permitted access to only necessary powers. Moreover, accidental changes are prevented thanks to the isolation of realms.

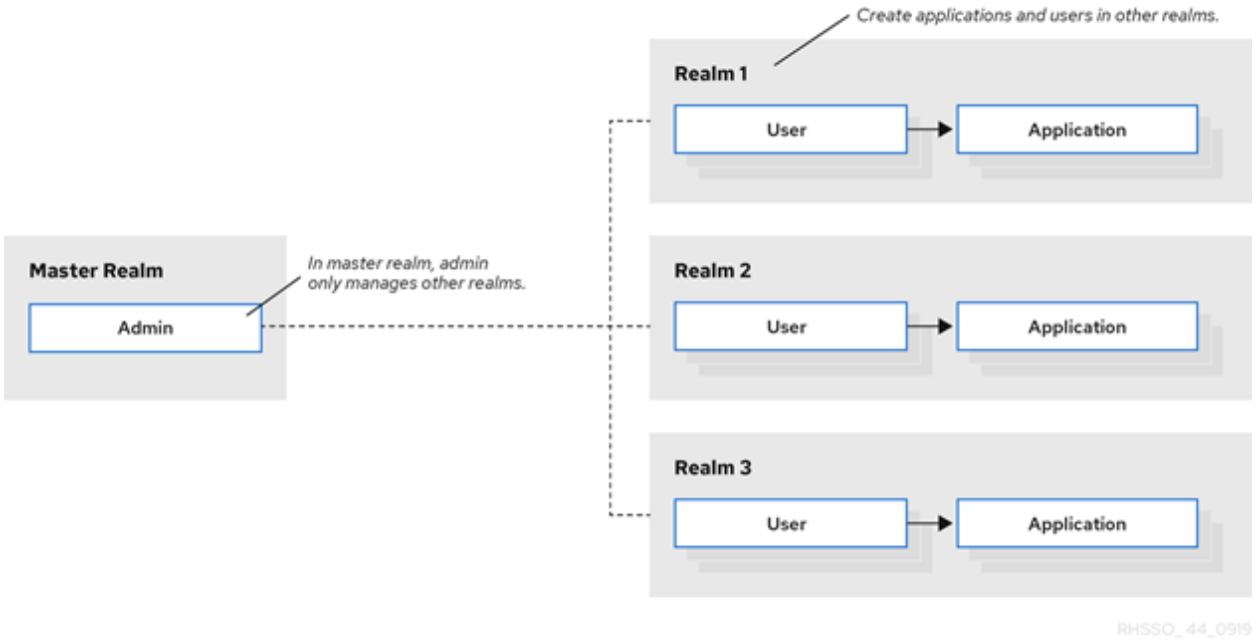

Figure 8: Realms and Applications [13]

Architectural patterns

Broker Pattern

In the Broker architectural pattern, a broker component is responsible for distributing requests across service providers and then managing the received responses [14]. One usage of the broker pattern is a service called identity brokers. Identity broker services offer their clients multiple types of identity services, such as different authentication methods. Then they can distribute authentication requests across these identity service providers. Keycloak can be used as an identity broker as well. For example, while performing a sign on through Keycloak, multiple identity service providers are available, for instance social identity provider (SIP) or an OpenID provider. Keycloak then distributes incoming requests among these different identity providers and then sends the response from the selected provider back to the user.

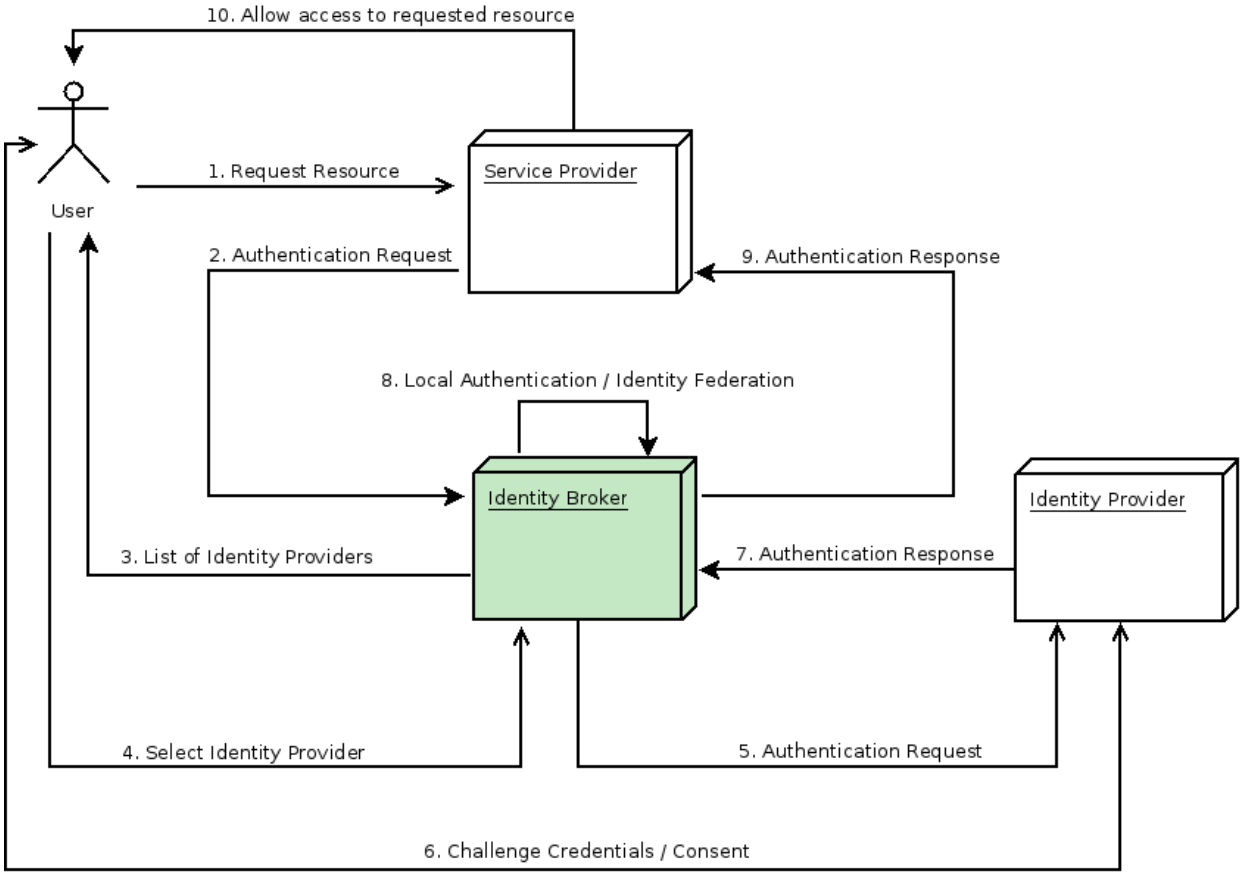

Figure 9: Identity Broker architecture used in Keycloak [15]

Client-server pattern

Client-server is an extremely common and useful architectural pattern. This pattern separates the applications into two categories: client and server. Servers are the part that contains the actual application logic [16]. The clients can request the server to perform the application logic for them, referred to as a 'service'. This is why servers are called service providers or producers, and clients are called consumers. In Keycloak's architecture, clients use servers in different ways. The most essential example is that a client application can simply want Keycloak to handle single sign on (SSO). Keycloak users can initialize a server first, then create and configure clients as they wish [17].

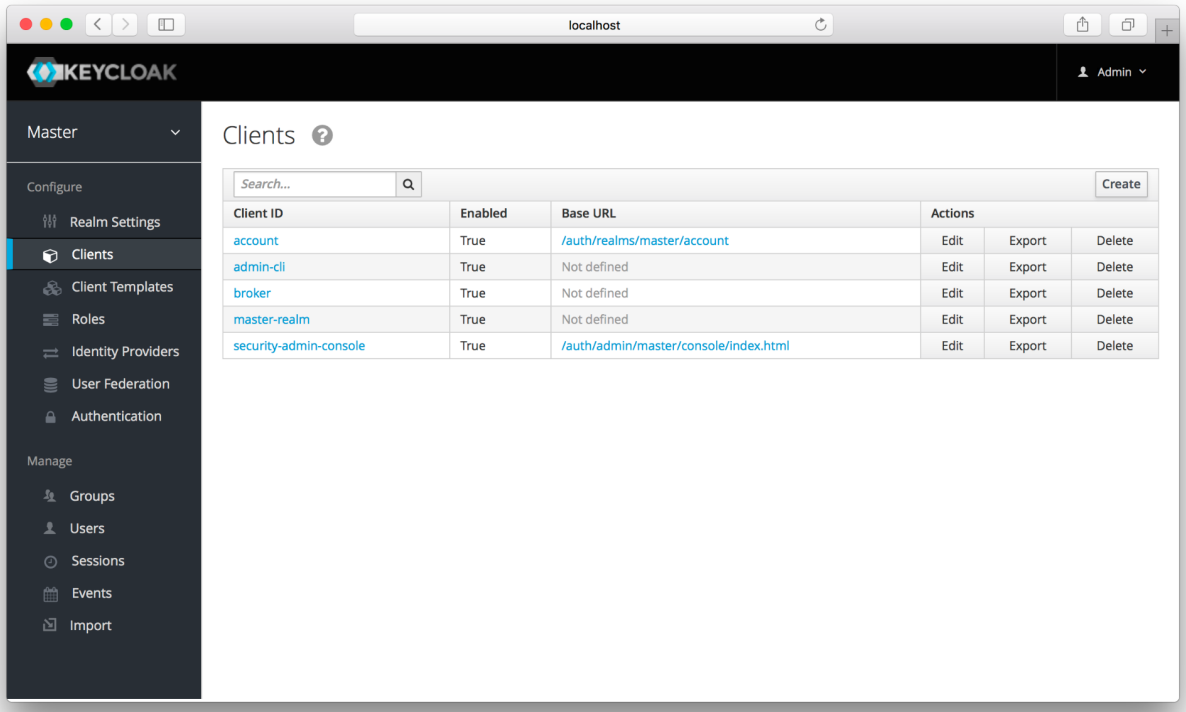

Figure 10: Client configuration in Keycloak [18]

Policy Enforcement Point Pattern

A policy enforcement point (PEP) is used for access control. PEPs are placed at points where authorization logic is required and security should be enforced. Then, authorization requests are received by the PEP. This way, the API is not exposed and authorization happens in a more secure way [19]. Keycloak provides the necessary tooling for users who would like to implement PEPs in their applications.

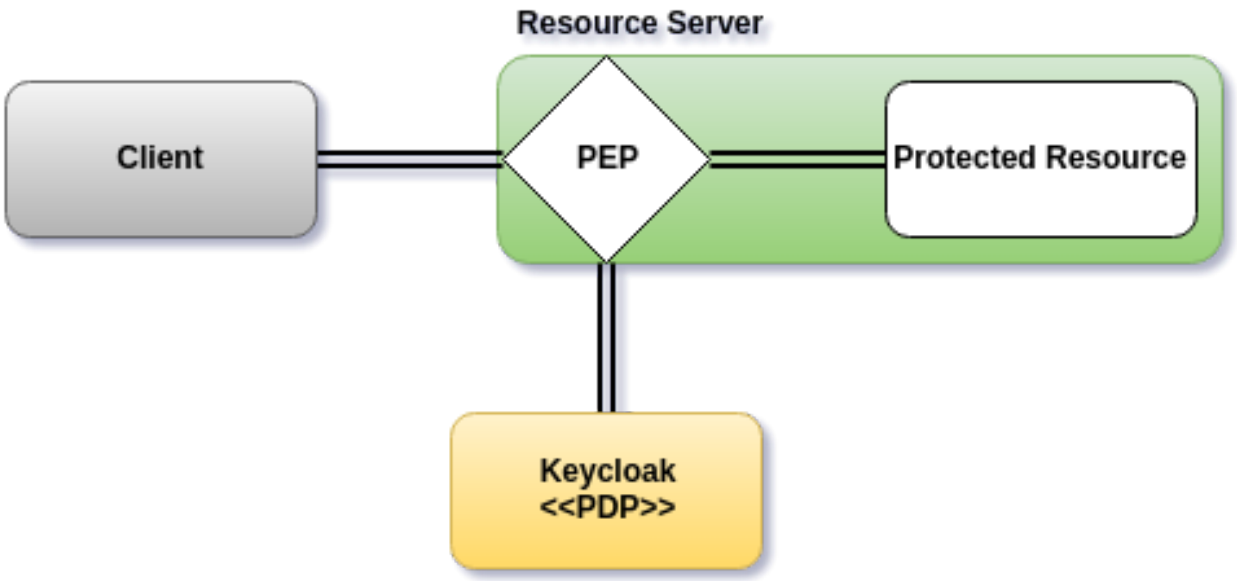

Figure 11: Policy enforcement point architecture in Keycloak [20]

Proposed Extension

Evolution Area

The problems mentioned above revolve around the performance aspect of the project. They also have great correlation to project's scalability/maintainability and usability. So, the proposed extension will mainly focus on these. Our goal is to propose architectural improvements that deliver readable, clean code while also delivering Keycloak's features with enhanced performance. Another dimension of the evolution area is improving the functionality of the architecture. The functionality of the system is the most important aspect of the architecture, hence we will focus on improving/enhancing the architecture so that the system functions as planned.

Architectural Change

One architectural change to the project will be in the architecture of the realms. As mentioned in the 'Single-tenancy performance issues with large numbers of realms' section above, the single-tenant architecture of the realms cause performance issues for large amounts of ram. Even though performance is an issue here, it also has pros like strong security and customizability. Single application/server for a single use, is very secure as only a number of

people can interact with it. Also, because of the same reason, developers can customize these realms according to the needs of the applications. However, as the number of realms increase, the cost and maintenance become cons to the architecture. So, for projects that will introduce a high number of realms, keycloak should allow developers to use multi-tenancy. This will reduce the cost significantly as the number of servers will decrease and help increase the performance of the system for the same reason. A comparison of the two can be seen in the diagram below.

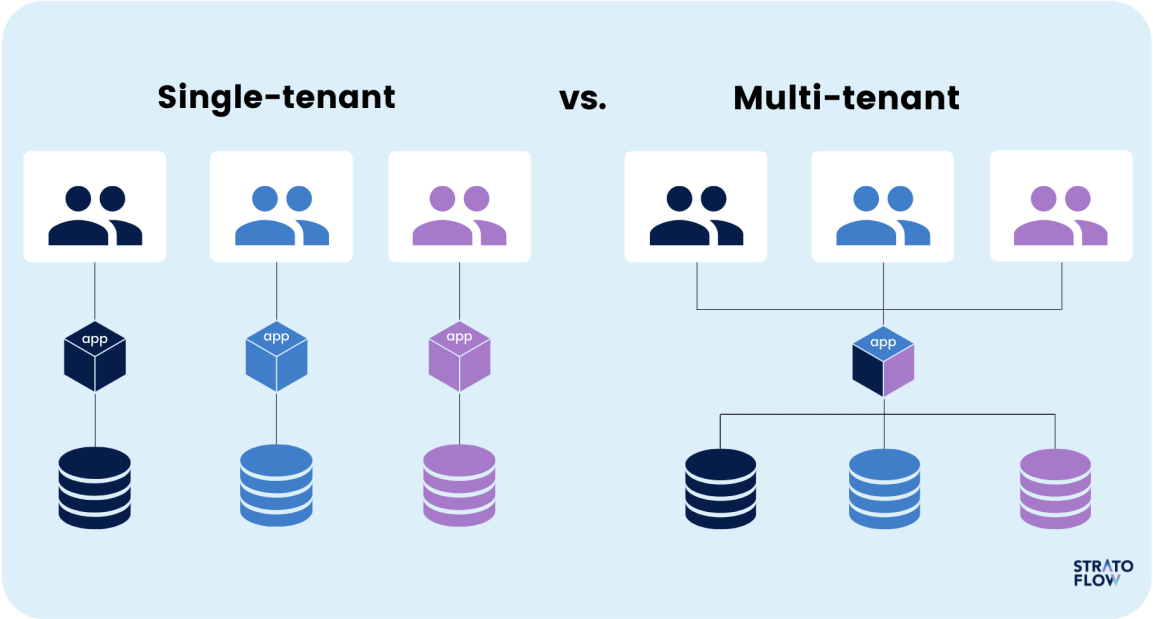

Figure 12: Single-tenancy versus multi-tenancy comparison [21]

Another architectural change will be made as an enhancement to the already mentioned broker pattern. As mentioned as a problem/solution (see 'Groups' Memberships Mismatch with LDAP' under 'Technical problems and solutions'), currently there are some synchronization problems that cause functional faulties. The changes in request management in the architecture will allow properly propagating syncs to update user values.

Rationale

Our main rationale behind making the change of adding multi-tenancy support is that multi-tenancy is extremely beneficial in terms of performance. Since multiple applications/users are using multiple services, it makes sense to 'group' clients so there's no need to create new

instances of a server. This way, the cost of starting and maintaining all those servers would be cut.

References

[1] "OpenLDAP Software 2.4 Administrator's Guide: Introduction to OpenLDAP Directory Services", *Openldap.org*, 2022. [Online]. Available: https://www.openldap.org/doc/admin24/intro.html. [Accessed: 09- Oct- 2022].

[2] mposolda Keycloak, "Automatic update of LDAP groups and LDAP GROUP MEMBERSHIPS DURING PERIODIC SYNC · issue #9609 · Keycloak/Keycloak," *GitHub*, 18-Jan-2022. [Online]. Available: https://github.com/keycloak/keycloak/issues/9609. [Accessed: 10-Oct-2022].

[3] "LDAP/AD Integration | keycloak-documentation", *Wjw465150.gitbooks.io*, 2022. [Online]. Available: https://wjw465150.gitbooks.io/keycloak-documentation/content/server_admin/topics/user-federation/ldap.html. [Accessed: 09- Oct- 2022].

[4] "Server Administration Guide," *Server administration guide*. [Online]. Available: https://www.keycloak.org/docs/9.0/server_admin/. [Accessed: 10-Oct-2022].

[5] A. K. Kristensen, "[keycloak-user] Two browser tabs result in two," *[Keycloak-user] two browser tabs result in two*. [Online]. Available: https://lists.jboss.org/pipermail/keycloak-user/2017-October/011975.html. [Accessed: 10-Oct-2022].

[6] acolombier, "Remove 'you are already logged-in' during authentication · issue #12406 · Keycloak/Keycloak," *GitHub*, 08-Jun-2022. [Online]. Available: https://github.com/keycloak/keycloak/issues/12406. [Accessed: 10-Oct-2022].

[7] V. Ramik, "[keycloak-dev] 'You are already logged-in' issue," *[Keycloak-dev] "you are already logged-in" issue*. [Online]. Available: https://lists.jboss.org/pipermail/keycloak-dev/2019-June/012146.html. [Accessed: 10-Oct-2022].

[8]"Keycloak Multitenancy", *Hcltech.com*, 2022. [Online]. Available: https://www.hcltech.com/blogs/keycloak-multitenancy. [Accessed: 10- Oct- 2022].

[9] "Maximum Limit of Realms", *Keycloak*, 2022. [Online]. Available: https://keycloak.discourse.group/t/maximum-limit-of-realms/8189/7. [Accessed: 10- Oct- 2022].

[10] "Single-responsibility principle", *Wikipedia.org*, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Single-responsibility_principle. [Accessed: 10- Oct- 2022].

[11] "Authorization Services Guide", *Keycloak.org*, 2022. [Online]. Available: https://www.keycloak.org/docs/latest/authorization_services/index.html#_overview_architecture. [Accessed: 10- Oct- 2022].

[12] "Server Installation and Configuration Guide", *Keycloak.org*, 2022. [Online]. Available: https://www.keycloak.org/docs/latest/server_installation/index.html#crossdc-mode. [Accessed: 10- Oct- 2022].

[13] "Server Administration Guide", *Keycloak.org*, 2022. [Online]. Available: https://www.keycloak.org/docs/latest/server_admin/index.html#the-master-realm. [Accessed: 10- Oct- 2022].

[14]L. Alosaimi, “Broker pattern,” Medium, 25-Jul-2021. [Online]. Available: https://medium.com/@lalosaimi/broker-pattern-297ac3cff6c5. [Accessed: 10-Oct-2022].

[15] "Brokering Overview | keycloak-documentation", *Wjw465150.gitbooks.io*, 2022. [Online]. Available: https://wjw465150.gitbooks.io/keycloak-documentation/content/server_admin/topics/identity-broker/overview.html. [Accessed: 10- Oct- 2022].

[16] K. Gayantha, “Software architectural patterns,” Medium, 09-Feb-2020. [Online]. Available: https://medium.com/@96kavindugayantha/software-architectural-patterns-17b05bfa3aef. [Accessed: 10-Oct-2022].

[17]“Keycloak-documentation,” Managing Clients | keycloak-documentation. [Online]. Available: https://wjw465150.gitbooks.io/keycloak-documentation/content/server_admin/topics/clients.html. [Accessed: 10-Oct-2022].

[18] "OIDC Clients | keycloak-documentation", *Wjw465150.gitbooks.io*, 2022. [Online]. Available: https://wjw465150.gitbooks.io/keycloak-documentation/content/server_admin/topics/clients/client-oidc.html. [Accessed: 10- Oct- 2022].

[19] “SQL Server 2000 operations guide: Microsoft prescriptive guidance,” Amazon, 2002. [Online]. Available:

https://docs.aws.amazon.com/prescriptive-guidance/latest/saas-multitenant-api-access-authorization/pep.html. [Accessed: 10-Oct-2022].

[20] "Policy Enforcers | keycloak-documentation", *Wjw465150.gitbooks.io*, 2022. [Online]. Available: https://wjw465150.gitbooks.io/keycloak-documentation/content/authorization_services/topics/enforcer/overview.html. [Accessed: 10- Oct- 2022].

[21] "What Is Multitenancy: Definition, Importance, and Applications", *Simplilearn*, 2022. [Online]. Available: https://www.simplilearn.com/what-is-multitenancy-article. [Accessed: 10- Oct- 2022].