TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Karl Viik 185043IAIB

SERVER-SIDE APPLICATION OF KNOWLEDGE SHARING SYSTEM IN QUESTION AND ANSWER FORMAT

Bachelor's thesis

Supervisor: Ago Luberg
MSc

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl Viik 185043IAIB

KÜSIMUSTE JA VASTUSTE VORMINGUS TEADMISTE JAGAMISE SÜSTEEMI SERVERIPOOLNE RAKENDUS

Bakalaureusetöö

Juhendaja: Ago Luberg
MSc

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Karl Viik

11.05.2021

Abstract

The goals of the given thesis is to implement a back-end application of a knowledge sharing system. This system is in the questions and answers format, where users can submit questions onto the platform on topics or details they would like answered, and other users can then submit answers to the questions. These questions are stored in a searchable format to reduce the need to ask questions that have already been asked and answered.

In the analysis, functional and non-functional requirements are set that the system must fulfil. Different systems created in the application for fulfilling these requirements are separately covered, including the system for grouping users and permissions together and the permission system itself.

Result of the thesis is a Spring Boot application which the front-end application can interact with through 68 different API endpoints to ask and answer questions, group content and users, and manage user access to the content.

This thesis is written in English and is 49 pages long, including 7 chapters and 10 figures.

Annotatsioon

Küsimuste ja vastuste vormingus teadmiste jagamise süsteemi serveripoolne rakendus

Antud töö eesmärk on luua serveripoolne rakendus, mida kasutatakse teadmiste jagamiseks kollaboratiivses keskkonnas. Formaat, mida kasutakse süsteemis teadmiste jagamiseks, on küsimused ja vastused, kus kasutajad saavad esitada küsimusi eri teemade ja detailide kohta, ning teised kasutajad saavad antud küsimustele vastata. Küsitud küsimused salvestatakse süsteemi otsitaval kujul, et vähendada vajadust juba küsitud küsimuste esitamiseks läbi otsimise võimaluse.

Analüüsis täpsustatakse funktsionaalsed ja mitte-funktsionaalsed nõuded, mida implementeeritav süsteem peab täitma. Arenduse peatükis kaetakse loodava aplikatsiooni alam-süsteeme, mille eesmärk on varem defineeritud nõudeid täita, nagu näiteks kasutajate ja õiguste grupeerimise süsteem või rollide süsteem.

Töö tulemuseks on Spring Bootil põhinev serverirakendus, millega süsteemi visualiseeriv veebisait saab suhelda läbi 68 erineva sissetulevat infot valideeriva *end-point*i, et luua küsimusi, anda vastuseid neile küsimustele, lisada kommentaare nii küsimustele ja vastustele, grupeerida kasutajate poolt loodud sisu kasutades kategooriaid ja *shard*e, ning kontrollida nendele ligipääsu läbi gruppide, õigustega rollide, ning eelmainitud *shard*idega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 49 leheküljel, 7 peatükki, 10 joonist.

List of abbreviations and terms

Content	User-created questions in the system
DAO	Database Access Object
DTO	Data Transfer Object
FR	Functional Requirement
JWT	JSON Web Token
NFR	Non-Functional Requirement
Nibble	Any user-created submission in the system
POJO	Plain Old Java Object
SaaS	Software as a Service
Shard	Grouping of content with permissions to limit their access
URI	Universal Resource Identifier
URL	Uniform Resource Locator

Table of contents

1 Introduction 10

2 Existing solutions 11

3 Analysis 12

3.1 Functional requirements 12

3.2 Non-functional requirements 13

3.3 Technological choices 13

3.3.1 Java 13

3.3.2 Spring Boot 13

3.3.3 Gradle 13

3.3.4 PostgreSQL 14

3.3.5 Liquibase 14

4 Development 15

4.1 Architecture 15

4.1.1 Filters 15

4.1.2 Controllers 16

4.1.3 Services 16

4.1.4 Repositories 16

4.1.5 Database Access Objects 16

4.1.6 Database 17

4.2 Questions and answers system 17

4.3 Shard system 18

4.4 Endpoints 19

4.4.1 Request objects 19

4.4.2 Data transfer objects 20

4.4.3 Endpoint documentation 20

4.5 Input validation 21

4.5.1 First layer 21

4.5.2 Second layer 21

4.6 Authentication and authorization 22

4.6.1 Username and password based authentication 22
4.6.2 OpenID Connect based authentication 22
4.6.3 Authorization 24
4.7 Permission system 24
4.7.1 User's account status 25
4.7.2 User's role in a shard 25
4.7.3 User's role in all parent shards 26
4.8 Shared Identifiers 26
4.8.1 Parties 27
4.8.2 Nibbles 27
4.8.3 Content 27
4.9 Exception handling 28
4.10 Database access 28
4.10.1 DAOs 28
4.10.2 Repositories 29
4.11 Testing 29
4.11.1 Unit testing 29
4.11.2 Integration testing 29
5 Deployment 31
5.1 Docker 31
5.2 GitLab CD 31
6 Future plans 32
7 Summary 33
References 34
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis 36
Appendix 2 – Database schema of the system 37
Appendix 3 – Table of endpoints with short description of the action and the required permissions 38
Appendix 4 – Table of existing solutions with short describing summaries 46

List of figures

Figure 1. Architectural layers of the system. 15
Figure 2. Relationship between question, answer, and comment entities. 17
Figure 3. Example of an organization's structure represented with shards. 18
Figure 4. Section of user interface for interacting with the API endpoints. 20
Figure 5. Request object's field validation and OpenAPI properties. 21
Figure 6. OpenID Connect authorization flow implementation in the application. 23
Figure 7. Relationship diagram for describing relationship between shards, users, and permissions. 26
Figure 8. Visualization of entities using the shared party identifier. 27
Figure 9. Visualization of entities using the shared nibble and content identifiers. 27
Figure 10. Error message from a search request with a disallowed page size. 28

1 Introduction

Sharing of knowledge is a natural component of a collaborative environment, be it a commercial project, a university course, or a whole company spanning multiple projects. One way of sharing knowledge is by posing a question to ask the others. This question can then be asked in for example the environment specific messaging solution or face-to-face. Problems with such approaches are that the shared knowledge may not spread as easily to everyone who could benefit from it and they don’t leave an easily searchable trace for when someone else has that question in the future. These are the problems that knowledge sharing systems aim to solve.

One format for a knowledge sharing system is a questions-answers format, where individuals ask questions and answer those questions collaboratively. The value of the system comes from storing those questions and answers in a searchable format and making them available to others.

The objective of this thesis is to implement such knowledge sharing system that is aimed for a closed collaborative environment, where access is limited to a preapproved list of individuals, for example employees of a company or students of an university, and where the system allows organizing the questions in a way that matches organization structure while also having ways to limit access and permissions.

2 Existing solutions

Due to the sharing of knowledge being an important part of many collaborative environments, many question and answer format knowledge sharing systems have been created. All existing solutions that were gone over as part of this thesis are listed in Appendix 4 along with a short summary for each of them.

Many of the solutions are not actively maintained which makes them a bad choice if looking for a minimal hassle solution. When it comes to being tailored for organizational usage, then most of the solutions fall short due to not having features that can be used to effectively organize content. An example of an existing solution that has such feature is SabaiDiscuss, but it in turn is limited by being a WordPress plugin.

TalkYard is an open-source solution that is aimed towards many use cases, including usage in companies and schools. It however lacks a mechanism to efficiently represent different separate parts of an organization.

In summary, when looking for solutions to use in an organization and not a single team, the choices are largely limited to expensive paid solutions such as AllAnswered and Confluence Questions. There seems to be room for a free or affordable solution that is tailored specifically for usage in organizations.

3 Analysis

This section describes the higher-level requirements for the system as well as the technological choices that are used for implementing the system.

3.1 Functional requirements

Functional requirements specify the actions the system performs to accomplish the tasks that users are trying to perform with the system [1]. The following list contains the functional requirements of the system, numbered with the prefix FR (Functional Requirement):

- FR01 Create shards that group content, such as questions, with permissions.
- FR02 Create categories in shards for grouping content without permissions
- FR03 Create roles in shards for grouping permissions
- FR04 Create groups in shards for grouping users
- FR05 Assign users and groups to roles
- FR06 Submit questions
- FR07 Submit answers to questions
- FR08 Submit comments to questions and answers
- FR09 Search for questions
- FR10 Vote on questions, answers and comments
- FR11 Edit questions, answers or comments
- FR12 Store previous versions of questions and answers
- FR13 Set an answer as the accepted answer of a question
- FR14 Follow shards, questions, answers or comments to receive notifications on notable events.
- FR15 Create notifications on certain actions to relevant users
- FR16 Manually create accounts
- FR17 Option of incorporating an OpenID Connect identity provider

3.2 Non-functional requirements

Non-functional requirements refer to the general qualities of a system [1]. The following list specifies the requirements for the system, numbered with the prefix NFR (Non-Functional Requirement):

- NFR01 Users may only perform actions and access resources to which they have the required permissions
- NFR02 Simple to deploy system
- NFR03 Sufficiently documented external interfaces
- NFR04 Validate incoming data
- NFR05 Support complex organizational structures

3.3 Technological choices

This section describes the technologies chosen for implementing the application.

3.3.1 Java

Java is an object-oriented programming language first released in 1995 by Sun Microsystems [2]. As of May 2021, it is the third most popular programming language according to TIOBE index [3].

Choice of using Java is based on author's familiarity with and preference of the language. Java version 11 is used due to it being the latest Long-Term-Support version as of start of 2021 [4].

3.3.2 Spring Boot

Spring is a framework for building Java applications with out of the box features to speed up the development, and Spring Boot simplifies development further by removing the need of some of the configuration that must be present with plain Spring and by having embedded web server [5].

3.3.3 Gradle

Gradle [6] is an open source tool used for building software. It was chosen over Maven, another popular build tool for Java-based applications, due to its better performance on building after small changes [7].

3.3.4 PostgreSQL

PostgreSQL [8] is an open source relational database system. It is used due to author's familiarity with it.

3.3.5 Liquibase

Liquibase [9] is a database schema management library. It is used to simplify upgrading database schema. Liquibase was chosen over Flyway, another popular library, due to the rollback support being in the free version. This feature can act as a backup in a production environment in case a database schema upgrade goes wrong.

4 Development

This section describes the architecture and different parts of the whole application along with choices made in implementation of these parts in detail.

4.1 Architecture

The system's architecture follows a layered approach, where each layer fulfils a separate task. The layers are brought out in Figure 1. The processes to fulfil a request travel through the specified layers.

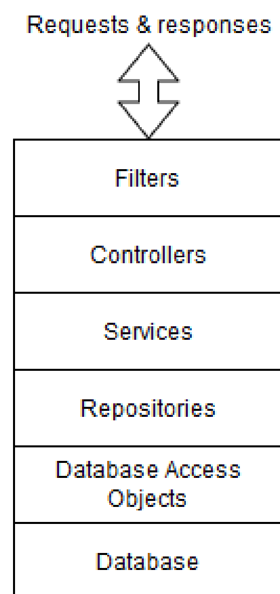


Figure 1. Architectural layers of the system.

4.1.1 Filters

The filter layer is the first layer incoming requests travel through. In practice, the filters implement the authentication and authorization of users. Authentication filters also act as controllers to take advantage of Spring's existing security mechanisms and interfaces to simplify the authentication implementation. Authorization filters trigger on all non-authentication requests to verify the user is authenticated and give the user's identifier that is making the request to the controllers to be used as a context for the request.

4.1.2 Controllers

The controllers define the available non-security related endpoints and what kind of information they require. Controllers perform initial validation of the data, described in paragraph 4.5.1.

If the incoming requests pass the initial validation at the controller level, a service method is invoked to perform the processing and obtain the data required for the response.

4.1.3 Services

The services perform further validation of the data, check if the user has permission for the requested action and perform other processes required for fulfilling the request.

Data validation at service layer involves checking validity of the data in relation to existing data in the system, described in detail in paragraph 4.5.2.

Validating that the user has the required permissions to perform the action is done at the service layer due to there not being enough information to do it at the filter layer. Details of the permission system and how they are calculated are described in paragraph 4.7.

Other processes to fulfil the request can involve obtaining different pieces of data and constructing a response from them, or creating notifications, or invalidating permission cache on actions that affect permissions.

4.1.4 Repositories

The repositories are the only way services interact with the persistent data storage. Repositories abstract away the schema of the database from the services. This is required for certain entities whose representation in the database span multiple tables. Examples and reasons for such entities are described in chapter 4.8.

4.1.5 Database Access Objects

DAOs (Database Access Objects) implement the SQL required to create, read, update, and delete information in the database. This layer is described in more detail in chapter 4.10.1.

4.1.6 Database

The database layer is the persistent storage of the system. The database schema is kept simple and does not include anything more complex than foreign keys and simple constraints such as disallowing nulls or requiring unique values or value pairs.

4.2 Questions and answers system

The questions and answers system forms the core of the system that the other systems enhance through the additional features they bring to the whole system. The entities of the system are the questions, answers, and comments, with their relationship illustrated in Figure 2, showing that a question can have many answers, both questions and answers can have many comments, and that each comment can have many comments.

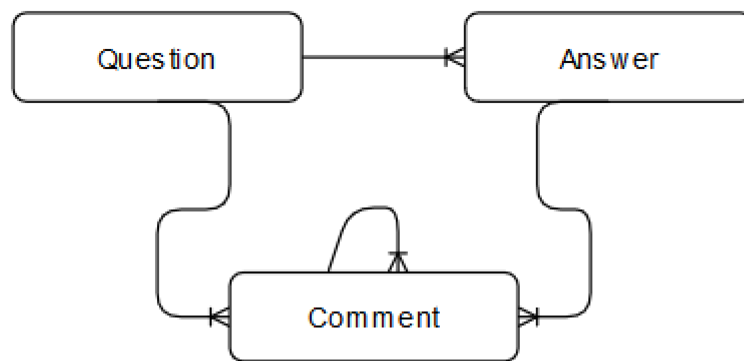


Figure 2. Relationship between question, answer, and comment entities.

A question is regarded to as a single piece of content. Due to future plans described in chapter 6, content is used as a more general concept and a question is one realization of that concept. This is reflected in the system through the usage of Java interfaces and in the database schema, described in more detail in chapter 4.8.3.

With each update to a question or answer, a new version of the item is created instead of overwriting the existing entry. This allows to store a version history for questions and answers for transparency, because different users can edit one question or answer if they have the required permissions.

A voting system is used to allow users to easily interact with submissions from other users, be it a question, an answer, or a comment, by showing their general like or dislike towards a submission by up-voting or down-voting it. Action of voting for any type of

submission is implemented through just a single endpoint by using a shared identifier for all named types, described in detail in chapter 4.8.2.

4.3 Shard system

Shard system is implementation of FR01. Each shard has its separate set of roles for permissions, own groups for grouping other groups and users, and own categories. In essence, each shard is a separate fully functional sub-system of the whole system.

The shard system is used even when no shards are created by the user as on initial start-up of the system, a special shard is created that acts as the root shard.

In order to also have support for complex organizational structures as required by NFR05, it is possible to make shards recursively, this means that a sub-shard of a shard can in turn have its own sub-shards. This however introduces additional challenges with the permission system, described in detail in chapter 4.7.

An example of a fictional organization's structure represented via usage of shards is shown in Figure 3. In short, any organizational structure that can be represented as a tree can be one-to-one implemented as shards in the system.

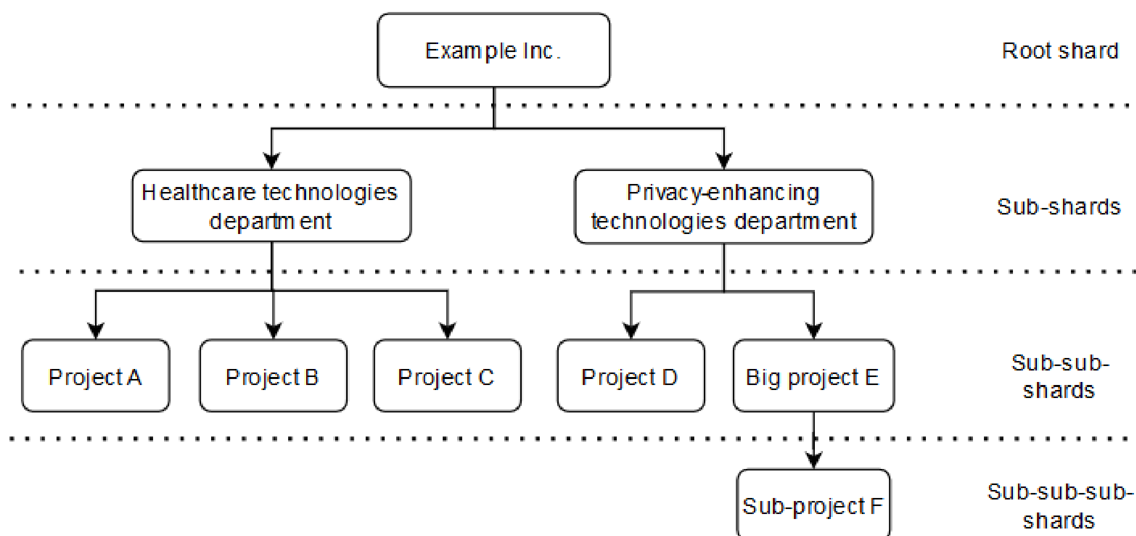


Figure 3. Example of an organization's structure represented with shards.

4.4 Endpoints

The application's resources are accessed via HTTP endpoints, where the action type is defined by the HTTP verb and the required data for the action is provided via query parameters or request body, depending on the HTTP verb used. This approach is commonly called REST [10].

POJOs (Plain Old Java Objects) are used to represent the required fields and structure of incoming data as well as fields and structure of outgoing data. POJOs of incoming data are called request objects, and POJOs of outgoing data are called DTOs (Data Transfer Objects).

There currently are 68 endpoint and HTTP verb combinations in total, described in detail in Appendix 3, and summarized in the following list:

- 2 for username and password authentication
- 2 for OpenID connect
- 5 for shards
- 3 for shard members
- 8 for shard groups
- 8 for shard roles
- 5 for shard categories
- 5 for user's followed content and shards
- 3 for user's favourite shards preference
- 5 for user management
- 2 for content and user or group searching
- 3 for content voting
- 13 for questions, answers, and comments
- 4 for user notifications

4.4.1 Request objects

Requests objects are exclusively used to represent incoming data at API endpoints. This approach increases code repetition by in essence being subsets of DTOs, but in return the request objects act as a documentation for the required data and its structure at each endpoint and also reduces risk of causing accidental changes in the API interfaces that

may cause front-end clients to break in some way. They also act as first layer of input validation, reducing the amount of validations at the service layer. This input validation is described in chapter 4.5.1.

4.4.2 Data transfer objects

DTOs are used for representing outgoing data. In general, each entity has just one representation as a DTO, this means that there is one DTO each for shards, roles, questions, and other entities. This approach keeps the structure of responses same, regardless of if the vote entity is attached to a comment or a question, which simplifies development of a front-end application by allowing to reuse components without modifications to their input data, if the front-end is developed in a modern framework such as Vue or React.

4.4.3 Endpoint documentation

Endpoints are automatically documented with OpenAPI specification [11] by using SpringFox [12]. The generated documentation is used for generating an interactive user interface for front-end developers to assist in development. Figure 4 contains a section of the page containing all endpoints of a single controller. Different annotations are used in controllers and on request objects to provide more info to SpringFox to use in the generation of the schema, such as short description of the action performed with the endpoint, as seen on the figure.

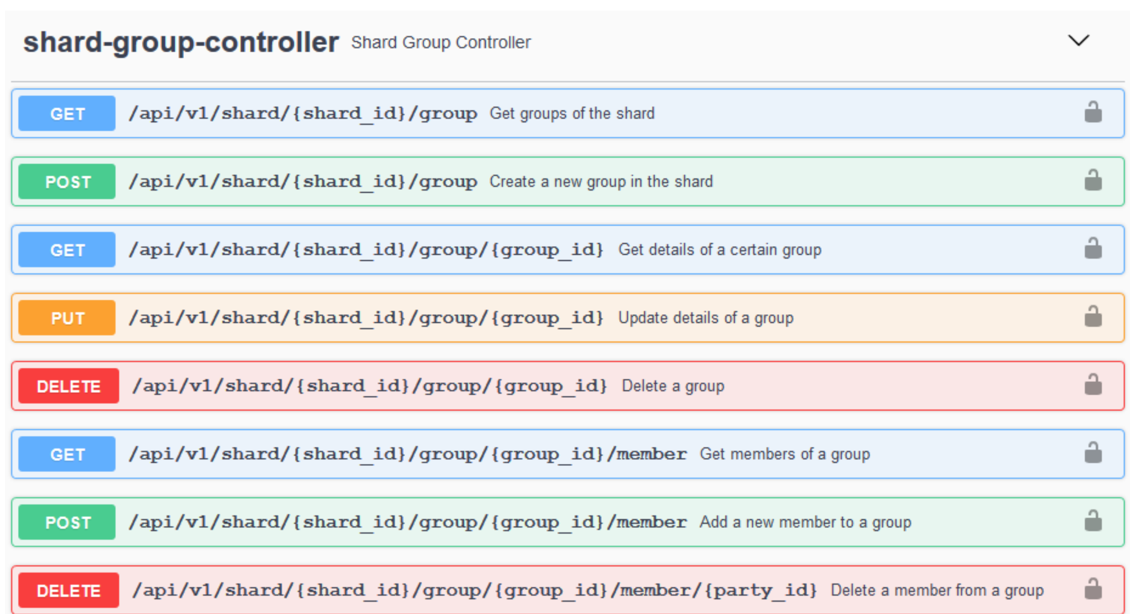


Figure 4. Section of user interface for interacting with the API endpoints.

4.5 Input validation

Validation of data submitted with the request is done in two layers, first layer being at the controller layer, and second layer being at the service layer.

4.5.1 First layer

The validation at the controller layer performs validation on the structure and individual values of the provided input.

Spring automatically checks the validity of the provided JSON where the endpoint requires input in form of request body. Incoming JSON data is automatically mapped to specified request objects. These request objects have per-field validation rules which are checked before executing code in the controller methods. The validation rules validate, for example, that a field is present, or that a numeric field's value is in a certain range. They also contain some additional information used by SpringFox, described in chapter 4.4.3.

Figure 5 contains an example of the validation rules on a field of a request object along with properties for SpringFox. The message in validation rules is returned in the request's response, mechanism for this is described in chapter 4.9.

```
@ApiModelProperty(
		notes = "Name of the group",
		required = true
)
@NotNull(message = "Name must be present")
@Length(
		min = 2,
		max = 50,
		message = "Name must be between {min} and {max}
characters"
)
private String name;
```

Figure 5. Request object's field validation and OpenAPI properties.

4.5.2 Second layer

Data validation at service layer involves checking validity of the data in relation to existing data in the system, such as validating that the category the user wants to set to a question exists in the shard where the question is, or that the role's permissions that the

user is trying to change are actually modifiable. If any of these validations fail, an appropriate exception is thrown.

4.6 Authentication and authorization

This section describes the authentication and authorization mechanisms used to limit access to API endpoints.

4.6.1 Username and password based authentication

Username and password authentication is the default method of obtaining access to the system's resources, meaning no additional configuration is required to enable it. Users can log in via manually administration-made accounts by providing a username and a password. It is checked if a user with such name exists and if their password matches the salted and hashed password of that account that is stored in the database. Bcrypt password-hashing function is used for protecting the passwords.

4.6.2 OpenID Connect based authentication

OpenID Connect 1.0 is an identity layer built on top of OAuth 2.0 [13]. An example of the specification's implementation is Microsoft's Azure Active Directory B2C [14], which plays the identity provider role.

The OpenID Connect 1.0 specification has three types of authentication flows: authorization code flow, implicit flow, and hybrid flow. The authorization flow is used to require the front-end client to do a second request to the system to complete the authentication. This allows to cleanly wrap the authentication result in a way that makes the exact authentication method irrelevant to the authorization. The exact steps to perform an authentication are described in Figure 6. Checking if an user of identity provider exists in the system is done using the subject identifier field of the identity token as that value is unique for every identity [15].

Implementation of the feature is done using Spring's OpenID Connect library, but in order to have better and simpler control over the library's behaviour, the security filters, which handle the authentication process, are manually added to the application's security configuration. This is done to control on what endpoints the flow is initiated and how the

authentication result is handled, as the default configuration initiates authentication on any endpoint and is more complex to configure in a way so that the result can be wrapped.

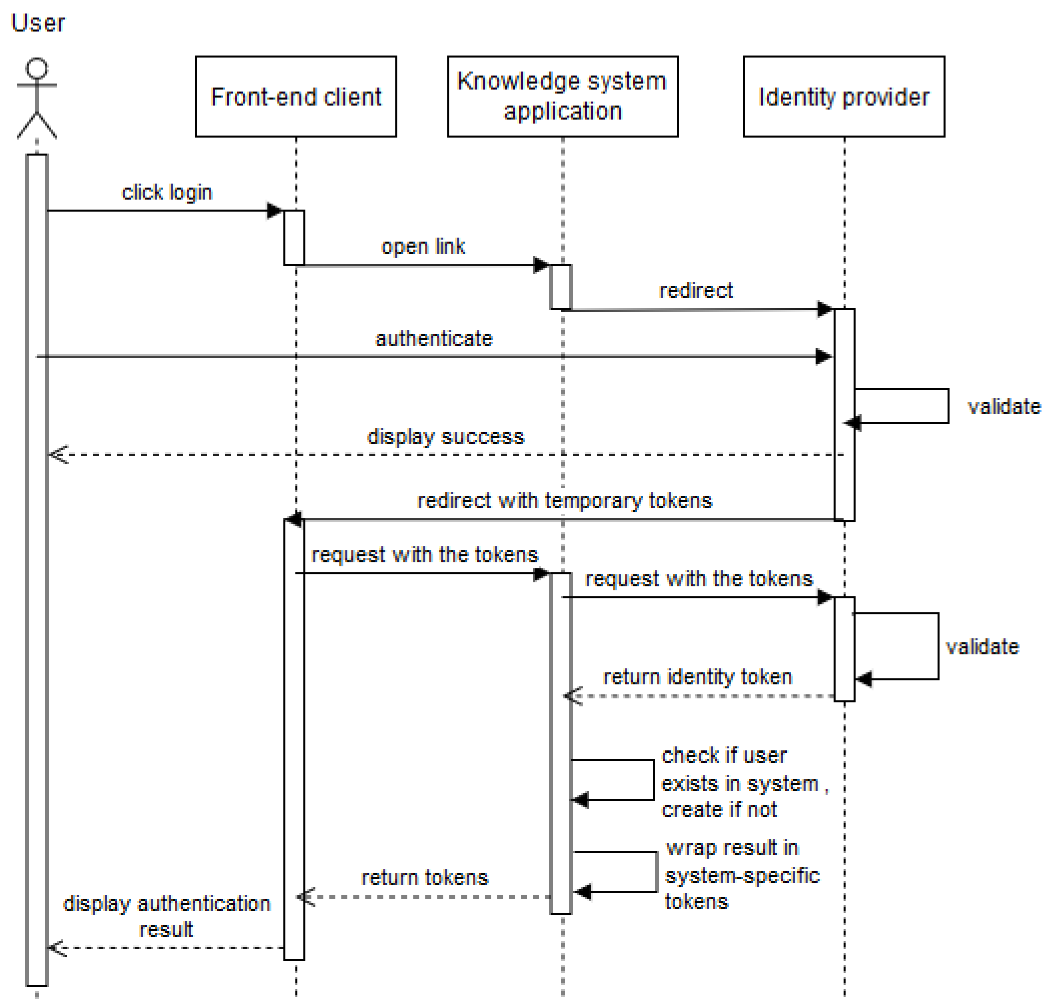


Figure 6. OpenID Connect authorization flow implementation in the application.

The configuration required to use OpenID Connect is very minimal but requires the identity provider to support the optional discovery mechanism that exposes the data required for interacting with it to relying parties in a standard way. The required configuration values are the following:

- Client ID: an identifier for the application, obtained from identity provider upon registration of the application as a new relying party.
- Client secret: a secret for the application, obtained during registration at identity provider
- Issuer URI (Universal Resource Identifier): URI for the issuer that the system uses to discover the discovery endpoint to obtain rest of the information required for interacting with the identity provider.

- Redirect URL (Uniform Resource Locator): URL for the front-end application of the system that is given to the identity provider on redirect, this must be one of the configured redirect URLs for the application in identity provider's configuration.

4.6.3 Authorization

On successful authentication, an access and refresh JWTs (JSON Web Tokens) are returned. These tokens contain the user's identifier which is used as a context in processing of the requests. An access token is checked for with every request to protected resources in the Authorization header of the HTTP request, and if present, the system checks if the token is valid and non-expired. Refresh token has a longer expiration time than access token and is used to obtain a new access token.

4.7 Permission system

Permission system encompasses roles with permissions along with shard structure of the application and is independent of the previously covered authentication and authorization using JWT tokens.

Permissions are grouped into roles. Each shard holds 3 default roles:

- Non-members for everyone not a member of the shard
- Member for everyone that is member of the shard
- Admin with all available permissions

The exception is the root shard which does not have a non-member role as all users of the platform are automatically members of the root shard. It is possible to create custom roles or to modify the permissions of all but the predefined admin role.

Permissions define what resources user has access to and what actions they can perform with the resources. On each request, the presence of required permissions is checked at the service level, and on absence of the permission an appropriate error is thrown. Appendix 3 contains the permissions that are required for each API endpoint.

The permissions of a user in a specific shard are determined by the following:

- User’s account status
- User’s role in the shard
- User’s roles in all parent shards

The permission checking is done at service layer due to decisions to flatten the API structure. This means that the shard’s identifier is not apparent from the request’s URI path. This API structure simplification however reduces the amount of validation checks that are required, reducing the risk of mistakes in validation logic.

Due to the calculation of permissions of a user in a shard also requires calculating permissions in all parent shards, a caching system is used to reduce load on the database and to speed up the requests.

4.7.1 User’s account status

User’s account can be in 3 different states: enabled, disabled, or in a state requiring a password change. When user’s state is not enabled, then the user has no permissions. The password change state is used to prompt the user to change their password after an administration has created the account with an administrator-set password.

4.7.2 User’s role in a shard

User may not directly be a holder of a role, but instead may hold the role indirectly by being a member of a group or member of a group’s group that holds the role. If they hold the predefined admin role, then they have all permissions in the shard as well as in all its recursive sub-shards.

Figure 7 illustrates the relationship between users, shards, and permissions. Of note is that each user is related to at least one group. This is due to shard membership being implemented as a group in order to not require another table for representing that relation. A detail that is omitted from the figure is that the groups used in these relations can only be groups belonging to the shard where the roles are in or belonging to that shard’s recursive parents.

Another detail of note is that each permission is related to at least one role. This is caused by the existence of the admin role in the root shard, which holds all permissions defined in the system, while admin roles of other shards hold a subset of these permissions.

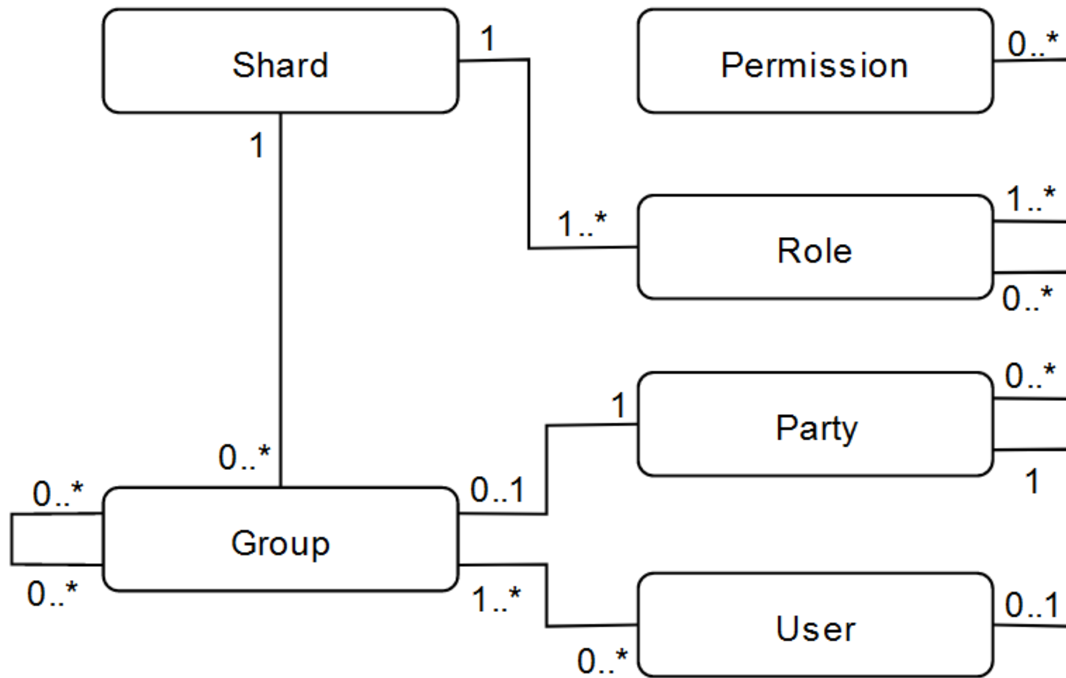


Figure 7. Relationship diagram for describing relationship between shards, users, and permissions.

4.7.3 User's role in all parent shards

There can be cases where user is a member of one shard but is not a member of a parent shard as it is difficult and error-prone to prohibit it due to the recursive nature of shards and groups. In such scenarios, these users are limited to the subset of the permissions they have in the parent shards to prevent these users from having permissions they should not have.

Another case where role in parent shard matters is the admin role, as every admin is also an admin in all sub-shards.

4.8 Shared Identifiers

Throughout the API, numeric identifiers are used to reference items such as specific shards, groups, or users. In order to reduce the number of endpoints and therefore the complexity of the API, some items share an identifier. This means that for an item that is part of the shared identifier mechanism, it is guaranteed that it is the only item that holds that identifier among the item types that share the identifier.

4.8.1 Parties

In order to simplify group and user management related endpoints, a shared identifier is used between groups and users. This removes the need to have separate endpoints for actions such as adding parties to a shard, or adding parties to a group. Visualization of user's and group's relationship to the shared party identifier is shown in Figure 8.

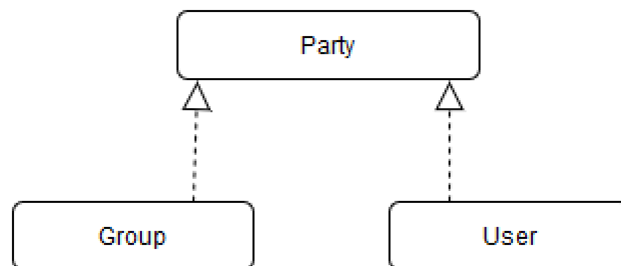


Figure 8. Visualization of entities using the shared party identifier.

4.8.2 Nibbles

A nibble is a term that covers all user-created submissions under all content types. Nibble covers questions, answers, and comments. Having these entities under a shared identifier allows to implement voting as a single simple endpoint. It also allows to simplify following of items for receiving notifications. Figure 9 is a visualization of entities using that shared identifier.

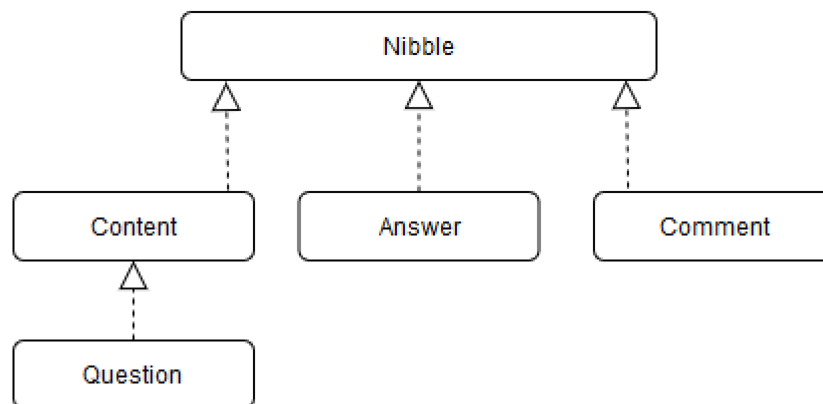


Figure 9. Visualization of entities using the shared nibble and content identifiers.

4.8.3 Content

Content shared identifier is used for grouping together different content types. Currently question is the only content type that is supported, but this approach simplifies

introducing other content types, such as polls or articles. Figure 9 holds a visualization of how the content shared identifier is related to the nibble shared identifier.

4.9 Exception handling

During processing, different exceptions can happen that are either the result of an user error, for example trying to access a resource for which they don't have the required permissions, or a system error, for example a database access fails. To give the API consumers a standard format, a global exception handling system is used. This handler maps business logic exceptions to appropriate HTTP error codes and includes the exception's message in the response, and on system exceptions returns a generic HTTP 500 response to avoid leaking system information outside.

Due to the Spring's provided method of implementing a central place for exception handling being limited by only working on exceptions thrown from controllers, an additional handler is introduced in order to catch exceptions that happen before the request arriving at controllers, such as security exceptions that happen at security filters. This handler rethrows the exception from a controller context so that they can be handled by the global exception handler.

An example of an error message returned as a result of having a disallowed page size in request object is shown in Figure 10.

```
{
    "error": "Bad Request",
    "message": "Page size has to be between 1 and 200",
    "status": 400
}
```

Figure 10. Error message from a search request with a disallowed page size.

4.10 Database access

Database access is done using two layers, repositories and DAOs.

4.10.1 DAOs

DAOs contain the SQL used for performing queries against the database. This approach is more error prone, but in turn allows to construct complex queries that allow to dome of the processing required for the request that is being fulfilled at the database layer, such as

using a recursive SQL query to find all users that are members of a specific group. Another reason for using plain SQL are the shared identifiers that cause the entities to be spread across multiple database tables.

SQL injection is prevented by using prepared statements to provide input to be used in the queries.

4.10.2 Repositories

Repositories are used in order to abstract away database schema implementation details from services. The database schema implementation may be more complex from entity classes due to shared identifier model. The schema used in database is illustrated in Appendix 2.

4.11 Testing

Testing is used to verify that the system behaves as expected when put under pre-defined scenarios.

4.11.1 Unit testing

Unit testing involves validating the behaviour of login in a small piece of code without inclusion of other layers of the system. In case of Spring, unit tests are in essence tests that do not initialize the Spring context. They are seldom used in this system due to there not being many things that can be easily tested without inclusion of other components.

4.11.2 Integration testing

In context of Spring, integration tests are tests that use the Spring context. These types of tests are extensively used for tests of the application's layers.

Due to the system requiring PostgreSQL-specific features for the database schema, an in-memory PostgreSQL is used. This embedded database is configured to clean its contents after every test class.

The tests largely test from three layers:

- Filter layer: check if username and password based authentication and JWT based authorization function as expected by sending HTTP requests to the servlet.

- DAO layer: validate if the written SQL in the classes functions as expected by using the embedded PostgreSQL database as the target for the tests.
- Controller layer: check that the endpoint methods only allow access for users with required permissions, and that the underlying layers behave as expected.

To avoid setting up the initial data with every integration test, some tests use an ordering system where after initial setup, multiple tests are ran in the specified order. An example is a commonly used pattern where first test tests if insertion works as intended, and second test tests if deletion behaves as expected by deleting the record inserted with the first test.

5 Deployment

This chapter covers the different aspects of deploying the system.

5.1 Docker

Application is made deployable as a Docker image. This allows for platform independent deployment without the need to install the correct dependencies on the hosting server [16]. Using Docker is part of fulfilling NFR02.

The two parts of using docker is the Dockerfile and the docker-compose file. Dockerfile defines how the application's Docker image should be built and ran, while docker-compose file defines other required images for running the application and is used to simplify the deployment of the system. In this case, the only other image required for running the application is the database image.

5.2 GitLab CD

During the development process, the updated application was automatically deployed to a server in order to make it available to the front-end developer using GitLab CI/CD. The automatic deployment consists of multiple steps:

- Build the application, verifying that there are no dependency or invalid code issues
- Test the application by running all unit and integration tests
- Create a docker image
- Deploy the docker image

These tests are all automatically ran consecutively on the same machine, but they can be configured to run on different machines, for example to deploy the image in a separate server.

6 Future plans

The future plans for the system is to develop it further to improve existing systems and introduce new systems to better fit the needs of organizations.

Some improvements that are planned is a more extensive OpenID Connect support along with adding SAML support.

It is also planned to add numerous other content types, such as articles and polls. Along with that a dynamic content morphing system would be explored, where questions and answers can be combined into an article for example, while still retaining the interactive nature of the question content type.

In the long term there is a plan to offer the system in a SaaS (Software as a Service) format. This can require fundamental changes to be able to securely, efficiently, and reliably host many instances of the platform with the least resources possible.

7 Summary

The goal of this thesis was to create a knowledge sharing system that is aimed at usage inside organizations, such as private companies with multiple active projects or universities. The created system supports representing the organizational structure with usage of the shard system, which allows to group user-created content with permissions to limit access to that content, should such separation between different parts of the organization be required. There are also the group and role systems to assist in management of users. It is also possible for users to contribute to the sharing of knowledge by creating questions and collaboratively answering them.

The created Spring Boot application fulfils the defined functional and non-functional requirements and the author plans to develop it further to increase the usefulness of the application in the organizational collaborative environment.

References

[1] AltexSoft, "Functional and Nonfunctional Requirements: Specification and Types," AltexSoft, 29 May 2018. [Online]. Available: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/. [Accessed 9 May 2021].

[2] Oracle Corporation, "What is Java and why do I need it?," Oracle Corporation, [Online]. Available: https://java.com/en/download/help/whatis_java.html. [Accessed 9 May 2021].

[3] TIOBE Software, "TIOBE Index," TIOBE Software, 2 May 2021. [Online]. Available: https://www.tiobe.com/tiobe-index/. [Accessed 9 May 2021].

[4] Oracle Corporation, "Oracle Java SE Support Roadmap," Oracle Corporation, 2 April 2021. [Online]. Available: https://www.oracle.com/java/technologies/java-se-support-roadmap.html. [Accessed 9 May 2021].

[5] Baeldung, "A Comparison Between Spring and Spring Boot," Baeldung, 24 March 2021. [Online]. Available: https://www.baeldung.com/spring-vs-spring-boot. [Accessed 9 May 2021].

[6] Gradle Inc., "Gradle Build Tool," Gradle Inc., 2021. [Online]. Available: https://gradle.org/. [Accessed 9 May 2021].

[7] Gradle Inc., "Gradle vs Maven Comparison," Gradle Inc., [Online]. Available: https://gradle.org/maven-vs-gradle/. [Accessed 9 May 2021].

[8] The PostgreSQL Global Development Group, "PostgreSQL," The PostgreSQL Global Development Group, [Online]. Available: https://www.postgresql.org/. [Accessed 9 May 2021].

[9] https://www.liquibase.org/, "Liquibase," https://www.liquibase.org/, [Online]. Available: https://www.liquibase.org/. [Accessed 9 May 2021].

[10] Codecademy, "What is REST?," Codecademy, [Online]. Available: https://www.codecademy.com/articles/what-is-rest. [Accessed 9 May 2021].

[11] "What Is OpenAPI?," [Online]. Available: https://swagger.io/docs/specification/about/. [Accessed 9 May 2021].

[12] "SpringFox," [Online]. Available: http://springfox.github.io/springfox/. [Accessed 9 May 2021].

[13] "OpenID Connect," [Online]. Available: https://openid.net/connect/. [Accessed 9 May 2021].

[14] "What is Azure Active Directory B2C?," 19 September 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/active-directory-b2c/overview. [Accessed 9 May 2021].

[15] N. Sakimura, NRI, J. Bradley, P. Identity, M. Jones, Microsoft, B. d. Medeiros, Google, C. Mortimore and Salesforce, "OpenID Connect Core 1.0 incorporating errata set 1," 8 November 2014. [Online]. Available:

https://openid.net/specs/openid-connect-core-1_0.html#IDToken. [Accessed 9 May 2021].

[16] “Why Docker?,” [Online]. Available: https://www.docker.com/why-docker. [Accessed 10 May 2021].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

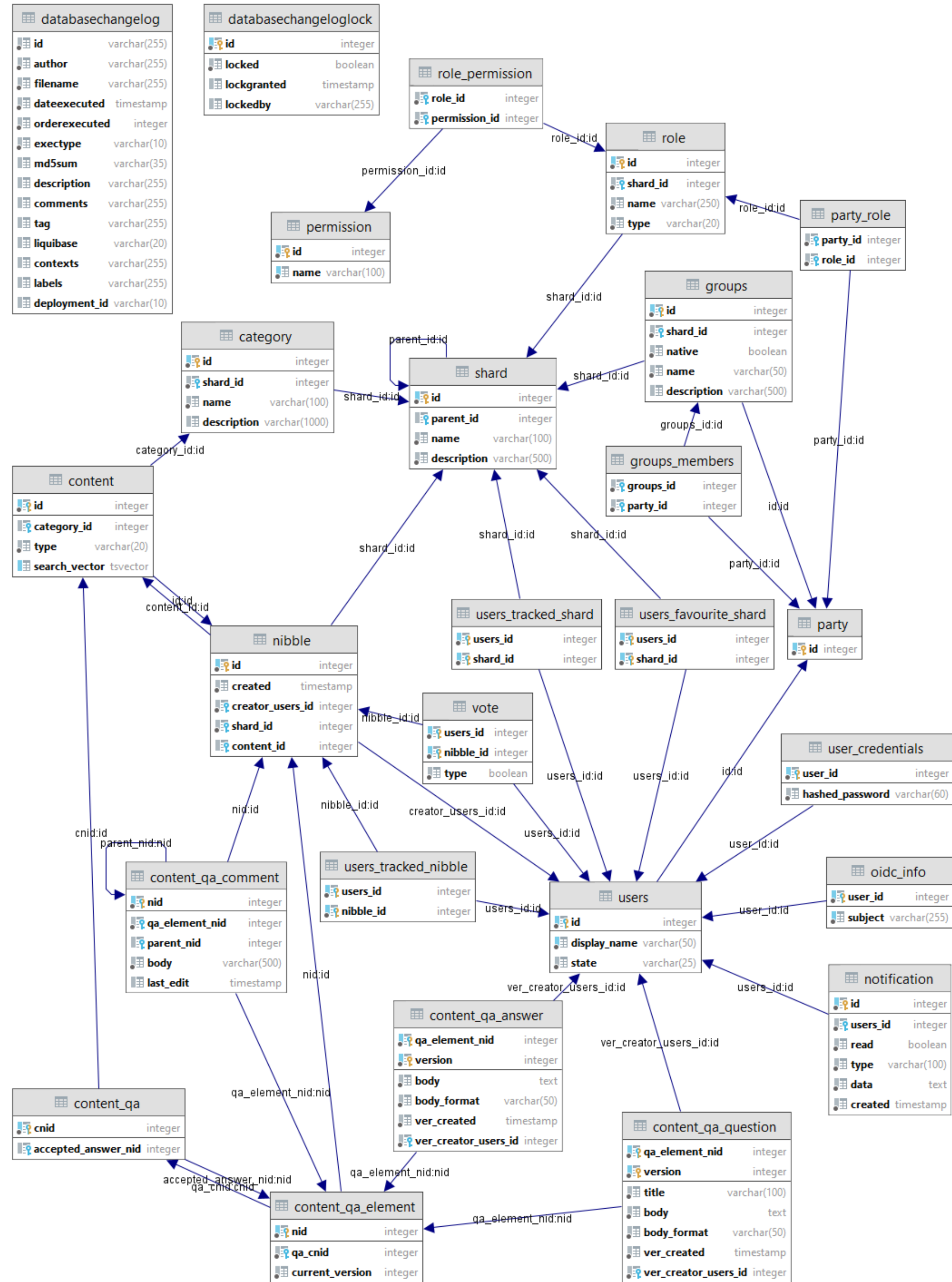
I Karl Viik

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Server-Side Application of Knowledge Sharing System in Question and Answer Format", supervised by Ago Luberg
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

11.05.2021

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Database schema of the system



Appendix 3 – Table of endpoints with short description of the action and the required permissions

HTTP verb	Endpoint path	Description of endpoint	Required permissions
POST	/api/auth/login	Authenticate the user with username and password combination.	N/A
GET	/api/auth/token	Retrieve a fresh access token with the provided refresh token.	N/A
GET	/api/oauth2/authorization/oidc	Redirect to identity provider's site for authentication.	N/A
GET	/api/oauth2/login/oidc	Exchange identity provider's token for access and refresh token.	N/A
GET	/api/v1/follow/shard	Retrieve list of the shards the requesting user is following.	Only shards where user has the READ permission are returned.
POST	/api/v1/follow/shard	Start following a new shard.	Can only start following shards where user has the READ permission.
DELETE	/api/v1/follow/shard/{id}	Stop following a shard.	No permissions are required.

POST	/api/v1/follow/item	Start following a nibble.	Can only start following nibbles that are from a shard where user has the READ permission.
DELETE	/api/v1/follow/item/{id}	Stop following a nibble.	No permissions are required.
GET	/api/v1/favourite/shard	Get list of shards that the user has marked as favourite.	Only shards where user has the READ permission are returned.
POST	/api/v1/favourite/shard	Mark a new shard as a favourite.	Can only mark shards where user has READ permission as favourite.
DELETE	/api/v1/favourite/shard/{id}	Unmark a shard as a favourite.	No permissions are required.
GET	/api/v1/category	Retrieve the list of categories of a category specified as a query parameter.	Requires READ permission of the shard.
POST	/api/v1/category	Create a new category in a shard.	Requires the CATEGORIES permission in the shard where user is attempting to add a new category.
GET	/api/v1/category/{id}	Retrieve details of a specific category.	Requires READ permission of the shard where the category is in.
DELETE	/api/v1/category/{id}	Delete a specific category.	Requires CATEGORIES permission in the shard where the category is in.
PUT	/api/v1/category/{id}	Update a category.	Requires the CATEGORIES permission in the shard where the category is in.
POST	/api/v1/user	Create a new user.	Requires the USER_ADMINISTRATION permission in the root shard.

GET	/api/v1/user/self	Retrieve details of currently authenticated user, including permissions in all shards.	No permissions are required.
GET	/api/v1/user/{id}	Retrieve details of a specific user.	No permissions are required.
PUT	/api/v1/user/{id}	Update details of a specific user.	Either requires the USER_ADMINISTRATION permission or the updating user to match the updated user.
POST	/api/v1/user/{id}/password	Change password of a specific user.	Either requires the USER_ADMINISTRATION permission or the updating user to match the updated user.
POST	/api/v1/search	Search for content in specified shards, with option of including content from sub-shards of the specified shards.	Only results from shards where user has the READ permission are returned.
GET	/api/v1/content/vote/item/{id}	Get the vote data of a certain nibble.	Requires READ permission in the shard where the nibble is.
POST	/api/v1/content/vote/item/{id}	Vote on a certain nibble.	Requires the CONTRIBUTE_CONTENT permission in the shard where the nibble is in.
GET	/api/v1/content/vote/content/{id}	Get the vote data of all nibbles under a certain content identifier.	Requires READ permission for the shard where the content is in.
POST	/api/v1/content/qa	Create a new question type content in the specified shard.	Requires the CONTRIBUTE_CONTENT permission in the shard.

DELETE	/api/v1/content/qa/{id}	Delete a specified question type content item.	Requires the DELETE_CONTENT permission in the shard where the content is in.
GET	/api/v1/content/qa/{id}	Retrieve data about a certain question type content.	Requires READ permission in the shard where the content is in.
GET	/api/v1/content/qa/{id}/history	Retrieve version history of the specified question.	Requires READ permission in the shard where the question is in.
PUT	/api/v1/content/qa/{id}	Update a question.	Either requires the EDIT_CONTENT permission or the updating user to be the creator of the question whilst having the CONTRIBUTE_CONTENT permission.
POST	/api/v1/content/qa/{id}/accept	Mark a specified answer as the accepted answer of a question.	Either requires the EDIT_CONTENT permission or the marking user to be the creator of the question whilst having the CONTRIBUTE_CONTENT permission.
DELETE	/api/v1/content/qa/{id}/accept	Remove the accepted answer.	Either requires the EDIT_CONTENT permission or the unmarking user to be the creator of the question whilst having the CONTRIBUTE_CONTENT permission.
POST	/api/v1/content/qa/{id}/answer	Create a new answer for the specified question.	Requires the CONTRIBUTE_CONTENT permission in the shard where the question is in.
GET	/api/v1/content/qa/{id}/answer/{id}	Retrieve details of the specified answer.	Requires the READ permission in the shard where the question is in.
GET	/api/v1/content/qa/{id}/answer/{id}/history	Retrieve version history of the specified answer.	Requires the READ permission in the shard where the question is in.

PUT	/api/v1/content/qa/{id}/answer/{id}	Update the specified answer.	Requires either the EDIT_CONTENT permission or the updater to be the creator of the answer whilst having CONTRIBUTE_CONTENT permission.
POST	/api/v1/content/qa/{id}/comment	Create a comment under the specified nibble under the specified question.	Requires the CONTRIBUTE_CONTENT permission in the shard where the question is in.
PUT	/api/v1/content/qa/{id}/comment/{id}	Update the specified comment.	Requires the updater to be the creator of the comment and hold the CONTRIBUTE_CONTENT permission.
GET	/api/v1/shard	Get list of all shards.	Only shards where user has the READ permission are returned, and list of sub-shards only includes those sub-shards where user also has the READ permission.
POST	/api/v1/shard	Create a new shard as a sub-shard of the specified shard.	Requires the SHARD_CHANGE permission in the parent shard of the newly created shard.
GET	/api/v1/shard/{id}	Retrieve details of the specified shard.	Requires the READ permission in the shard. Only these sub-shards are detailed in the response where user also has a READ permission.
DELETE	/api/v1/shard/{id}	Delete a shard if it does not have any sub-shards.	Requires the SHARD_DELETE permission for the shard.
PUT	/api/v1/shard/{id}	Update the details of a shard.	Requires the SHARD_CHANGE permission in the shard. Additionally, requires the SHARD_MOVE permission if user is attempting to change the parent of the shard.
GET	/api/v1/shard/{id}/member	Retrieve list of all members of the shard.	Requires the READ permission in the shard.

POST	/api/v1/shard/{id}/member	Add a new party to the list of members of the shard.	Requires the SHARD_MEMBERS permission in the shard.
DELETE	/api/v1/shard/{id}/member/{id}	Remove a party from the members of the shard.	Requires the SHARD_MEMBERS permission in the shard.
GET	/api/v1/shard/{id}/group	Retrieve list of all groups defined in the shard.	Requires the READ permission in the shard.
POST	/api/v1/shard/{id}/group	Create a new group in the shard.	Requires the SHARD_GROUPS permission in the shard.
GET	/api/v1/shard/{id}/group/{id}	Get details of a specific group in the shard.	Requires the READ permission in the shard.
PUT	/api/v1/shard/{id}/group/{id}	Update details of the specified group.	Requires the SHARD_GROUPS permission in the shard.
DELETE	/api/v1/shard/{id}/group/{id}	Delete the specified group.	Requires the SHARD_GROUPS permission in the shard.
GET	/api/v1/shard/{id}/group/{id}/member	Get the members of the specified group.	Requires the READ permission in the shard.
POST	/api/v1/shard/{id}/group/{id}/member	Add a new party to the list of members of the specified group.	Requires the SHARD_GROUPS permission in the shard.
DELETE	/api/v1/shard/{id}/group/{id}/member/{id}	Remove a party from the list of members of the group.	Requires the SHARD_GROUPS permission in the shard.
GET	/api/v1/shard/{id}/role	Retrieve list of roles defined in the shard.	Requires the READ permission in the shard.

POST	/api/v1/shard/{id}/role	Create a new custom role in the shard.	Requires the SHARD_ROLES permission in the shard.
GET	/api/v1/shard/{id}/role/{id}	Get details of the specified role, including list of permissions.	Requires the READ permission in the shard.
PUT	/api/v1/shard/{id}/role/{id}	Update the specified role.	Requires the SHARD_ROLES permission in the shard.
DELETE	/api/v1/shard/{id}/role/{id}	Delete the specified role if it is a custom role.	Requires the SHARD_ROLES permission in the shard.
GET	/api/v1/shard/{id}/role/{id}/member	Retrieve the list of parties that hold the specified role.	Requires the READ permission in the shard.
POST	/api/v1/shard/{id}/role/{id}/member	Add a new party to the list of the role's holders.	Requires the SHARD_ROLES permission in the shard.
DELETE	/api/v1/shard/{id}/role/{id}/member/{id}	Revoke the specified role from the specified party.	Requires the SHARD_ROLES permission in the shard.
GET	/api/v1/notification	Retrieve list of notifications, ordered by freshness. These notifications contain data the front-end can use to construct a link to the resource the notification is about.	No permissions are required.
GET	/api/v1/notification/count	Retrieve the amount of notifications the user has.	No permissions are required.
DELETE	/api/v1/notification/manage/{id}	Delete a certain notification.	No permissions are required.

PUT	/api/v1/notification/manage/{id}	Mark a certain notification as read.	No permissions are required.
POST	/api/api/v1/partysearch	Search for parties, used to simplify adding parties to roles, groups, and shards.	Requires the READ permission in the shard from which the search is being made, specified in the search request.

Appendix 4 – Table of existing solutions with short describing summaries

Name	Link	Summary
Wpmudev QA	https://github.com/wpmudev/qa	Open-source questions and answers system PHP plugin for Wordpress, allowing to set permissions for each role used on the site, not maintained since 2016.
PHPancake	https://sourceforge.net/projects/phpancake/	An open-source questions and answers system made in PHP, not maintained, last active with a beta release in 2009.
Cahoots!	https://sourceforge.net/projects/cahoots/	An open-source questions and answers platform made in PHP, aimed towards communities. Not maintained, last active in 2010.
CNPROG	https://github.com/chagel/CNPROG	An open-source questions and answers system made in Python. Features tags, votes and revision history for questions and answers. Not maintained, last active in 2010.
Shapado	https://github.com/ricodigo/shapado	An open-source questions and answers system for hosting different communities implemented in Ruby. Not maintained, last active in 2012.
LampCMS	https://github.com/snytkine/LampCMS	An open-source questions and answers program implemented in PHP using MongoDB, aimed towards being highly scalable. Not maintained, last active in 2015.

OSQA	https://github.com/OSQA/osqa https://github.com/dzone/osqa	An open-source questions and answer system implemented in Python, fork of the CNPROG project. The open-source project has been abandoned by the creators in favour of the paid version, AnswerHub. Last active in 2015.
Smartr	https://github.com/dkd/smartr/	An archived open-source questions and answers system implemented in Ruby with support for common features such as votes and tags. Not maintained, last active in 2014.
Qaror	https://github.com/mateuszdw/qaror	A simple open-source questions and answers platform developed in Ruby. Not maintained, last active in 2016.
Kunjika / Memoir	https://github.com/shivshankardayal/Kunjika https://github.com/shivshankardayal/memoir	A simple open-source questions and answers system built out of creator's dissatisfaction with other existing open-source options for interacting with readers of their books. Built in Python, not maintained, last active in 2017.
Qwench	https://github.com/anantgarg/Qwench	An open code self-proclaimed StackOverflow clone built using PHP in around 2010, not maintained, last active in 2018.
Django-knowledge	https://github.com/zapier/django-knowledge	A simple open-source questions and answers style knowledge base built using Python, aimed towards using as a help desk with features such as anonymous questions or limiting visibility to the question asker and the configured staff. Not maintained, last active in 2017.
Haydle	https://haydle.com/	A closed source questions and answers system aimed towards enterprises, currently not being made available to new customers as of May 2021.
SabaiDiscuss	https://sabaidiscuss.com/	A paid questions and answers plugin for WordPress sites. Has features such as hierarchical categories and limiting access based on roles. Actively supported by the creator.

Coordino	https://github.com/Datawalke/Coordino	An open-source knowledge software built with PHP, more aimed towards wider communities with features such as user rewards and spam filtering. Not maintained, last active in 2015.
Quora	https://www.quora.com/	A popular closed source and free to use question and answer community platform.
Question2Answer	https://www.question2answer.org/	A widely used open-source questions and answers system built in PHP, aimed towards being used on public sites for interaction with and between visitors. Still maintained with latest commit in repository being from November of 2020.
Allanswered	https://www.allanswered.com/	A closed source and paid knowledge management system encompassing questions and answers format interaction as well as wiki pages. Contains many integrations with other tools such as Slack and Microsoft Teams. Has feature of creating communities for grouping different things together.
Askbot	https://github.com/ASKBOT/askbot-devel	A standard open-source Q&A forum system built with Python. Somewhat maintained with last activity in December of 2020.
Talkyard	https://github.com/debiki/talkyard	An actively maintained and open-source community discussion platform developed in Scala. Is developed with many use cases in mind, such as using as customer support or obtaining feedback.
Biostar-central	https://github.com/ialbert/biostar-central	An open-source questions and answers system built with Python, with common features such as votes and tags. Actively maintained.
Scoold	https://github.com/Erudika/scoold	An open-source questions and answers platform built with Java featuring common features of a Q&A system. Aimed towards usage in teams. Actively maintained.

Quandora	https://www.quandora.com/	A closed source and paid solution with feature allowing to create separate bases for grouping questions, in essence categories.
Qpixel	https://github.com/codidact/qpixel	An open-source questions and answers system built in Ruby with common features, used in a network similar to the Stack Exchange communities. Actively maintained.
Mamute	https://github.com/caelum/mamute	An open-source questions and answers platform developed in Java, boasting common features of a Q&A platform. Not maintained, last activity in 2019.
Tribe	https://tribe.so/	A closed source and paid solution for a community platform. Supports many content types, such as questions and articles. Built to be public and used as a method of driving user engagement with a brand.
Vanilla	https://vanillaforums.com/en/	A closed source and paid solution aimed towards using it as a customer engagement platform with features such as gamification through points system, forum, private messaging and support-focused knowledge base.
Questions for Confluence	https://www.atlassian.com/software/confluence/questions	A closed source and paid addition to the already paid Confluence solution aimed towards usage in companies who are already using Confluence.
AnswerHub	https://devada.com/answerhub/	A closed source and paid solution aimed at enterprises with analytics and different content types. Originated from OSQA.
StackOverflow for Teams	https://stackoverflow.com/teams	A closed source and paid solution aimed at teams and enterprises with features such as articles, integrations with tools such as GitHub and Slack, and analytics.