Qanda

Project documentation

This is the documentation for the different parts of the system that is Qanda. The setup process for running the program can be found in the README.md in the repository. This document serves as a visual guide to the communication between the systems. It will display graphs for the different parts of the total project.

Structure

Frontend

Components

The project utilizes both components and views. Two of the most important functionalities, *creating quiz* and *displaying quiz* are structured with the same component principle. The quiz creator tool lets the user pick between three different question types. Each of these are separate components. This allows for reusability. This usability is shown in the landingpage of the website with the advertisement for using a flipcard. After a quiz is created, the quizreader also has three separate components that parse the information based on what type of question it is. This allowed for great flexibility for our user to mix questions as they please within the quizzes.

Store

The website utilizes Store for inputs, and allows the user to navigate around the site without losing their quizzes. This creates a seamless experience that always keeps the users quizzes as long as they wish for it to be there, or until an eventual expiry.

Backend

Composition

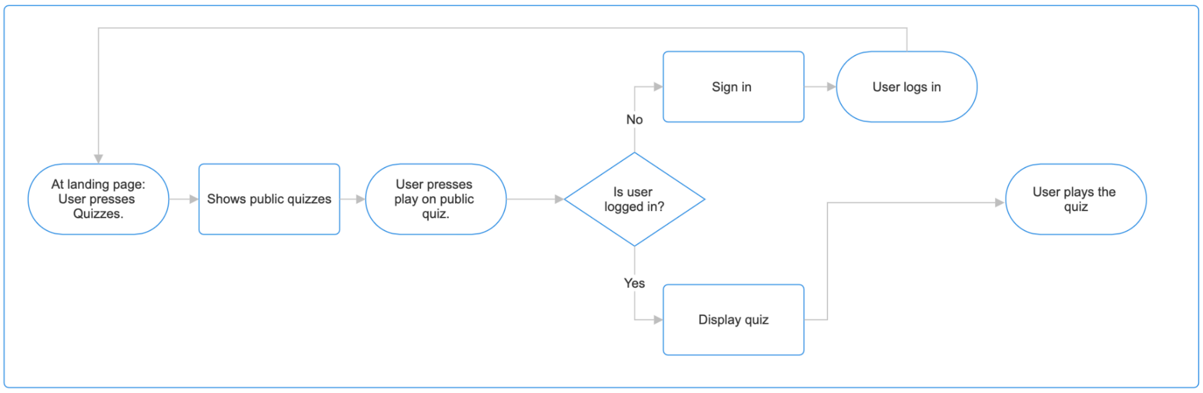
Our backend consists of config, controller, dto, exception, mapper, model, repository, security, service and startup. This separation of responsibilities creates a modular and adaptable backend with possibilities for many further implementations.

Endpoints

To view the available endpoints, it is recommended to look within the Swagger. Here you can see all the different endpoints, with detailed information about how they are used and how they are formatted. The endpoints provide a very effective way for the frontend to collect its needed information.

Flowcharts

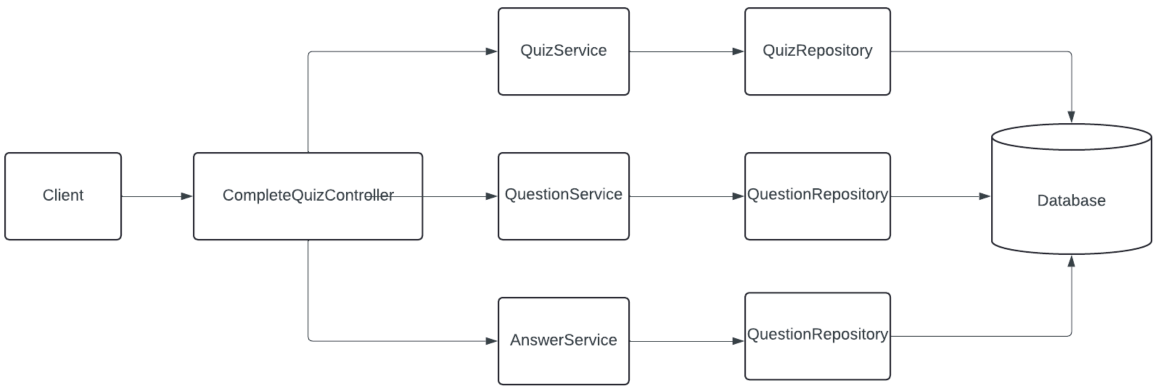
Frontend



The principle for the website is to let the available content so that they are intized with playing the quizzes, and then when they press play they are prompted to log in or create a user. This allows for an accessible experience while still being able to record a user's attempts.

Backend

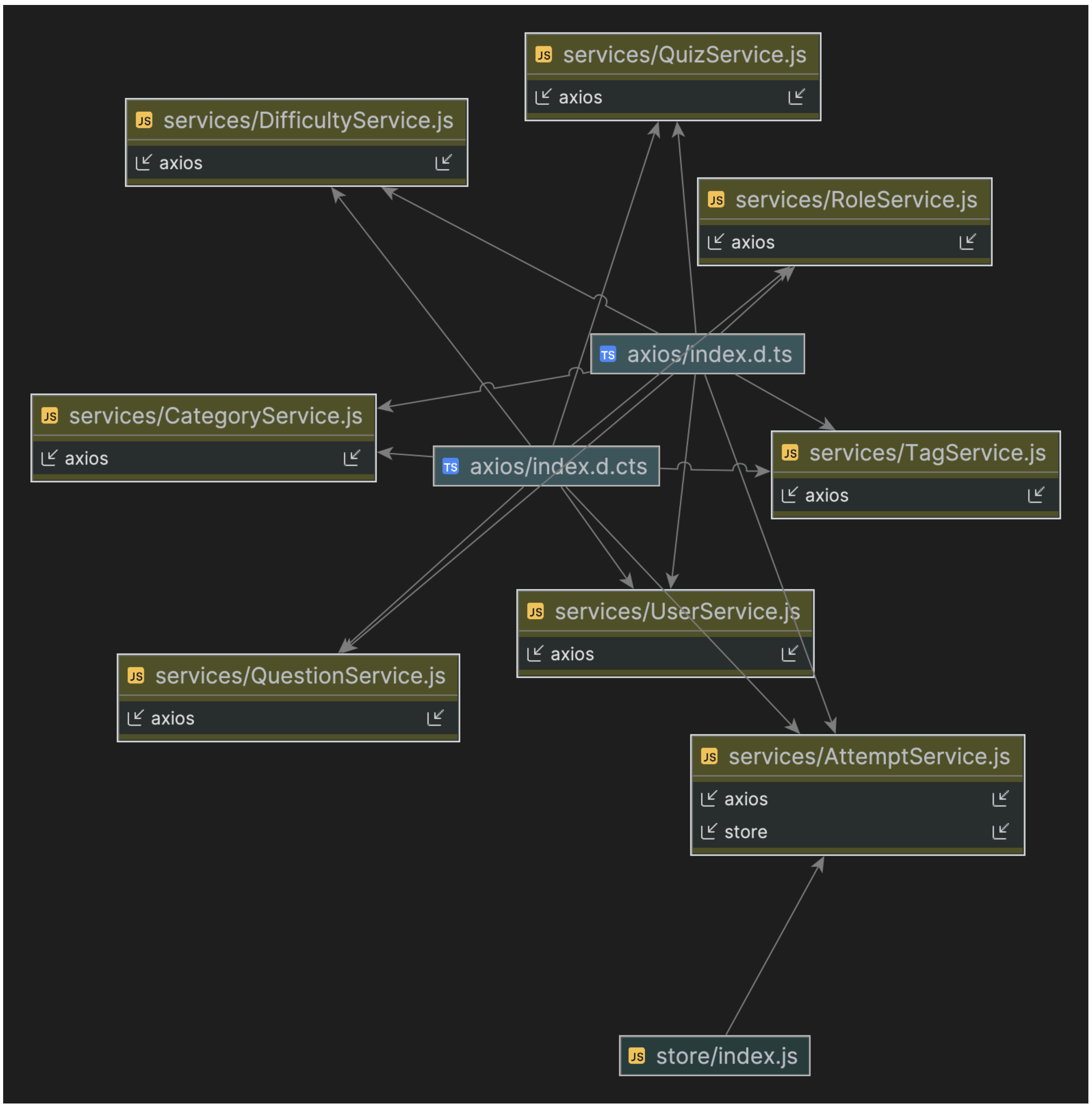
FlowChart for CompleteQuiz:



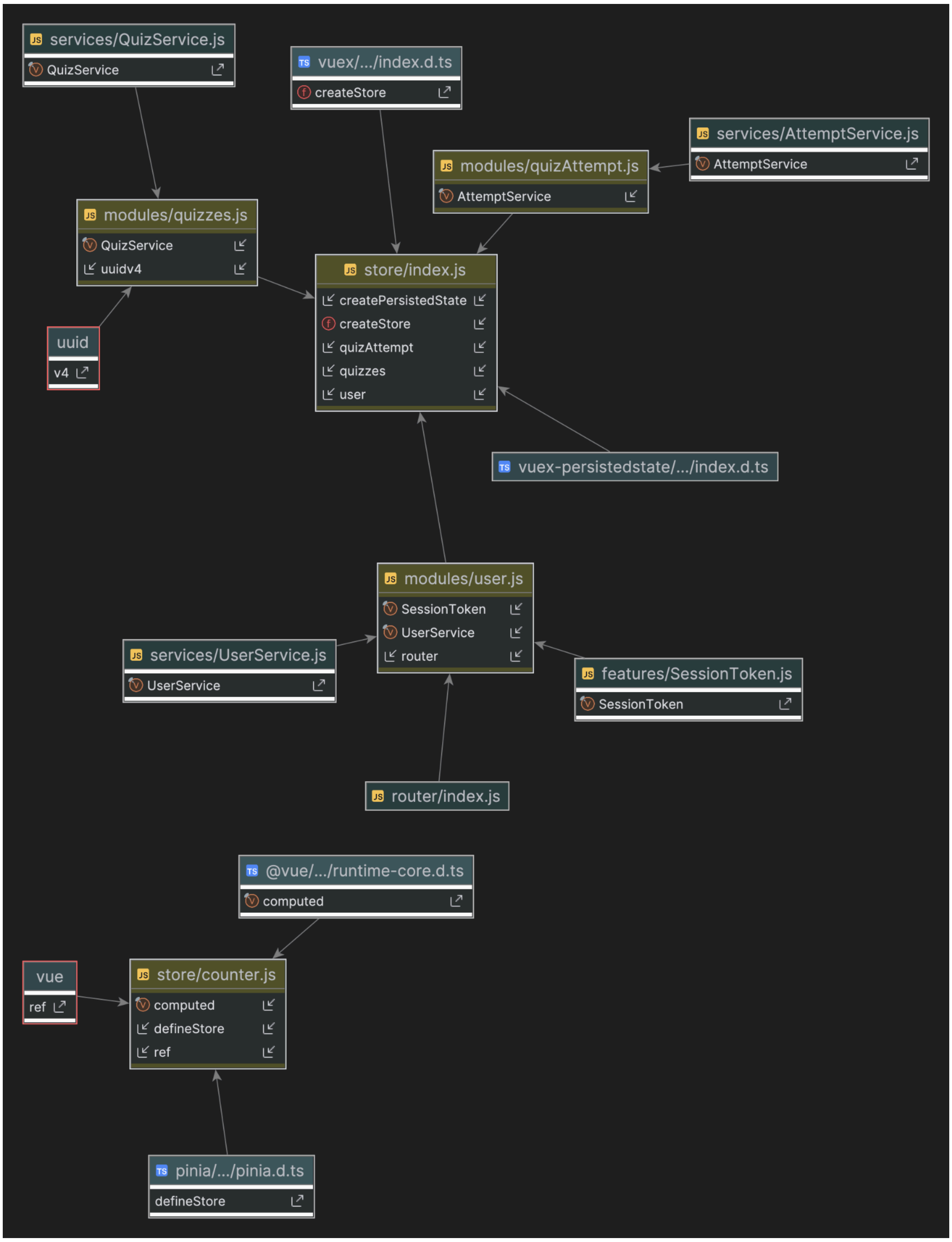
Diagrams

Frontend

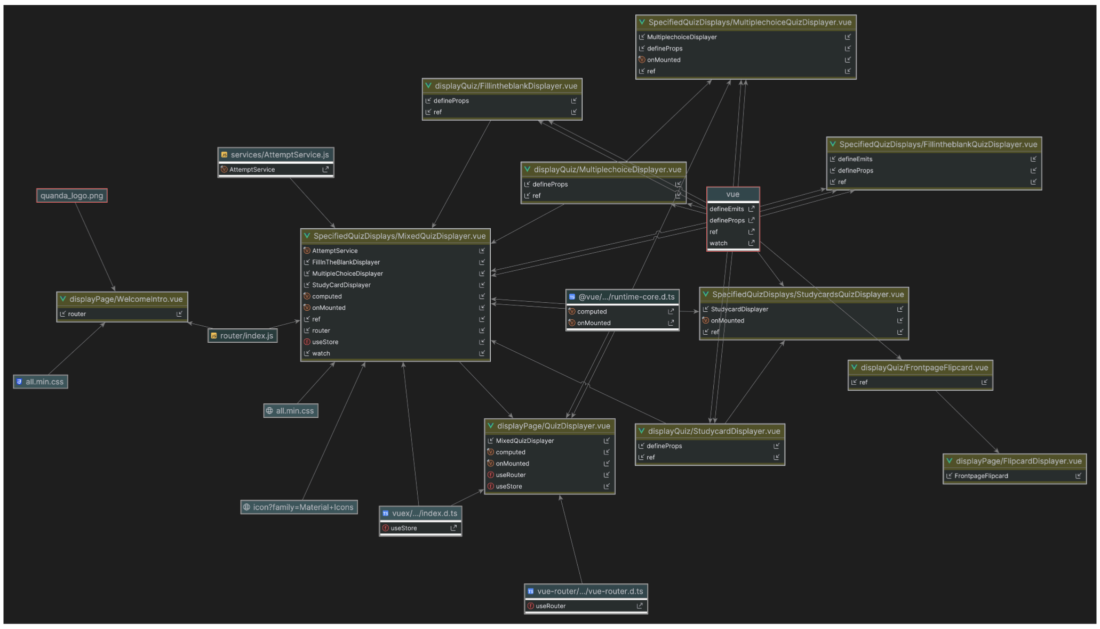
Services



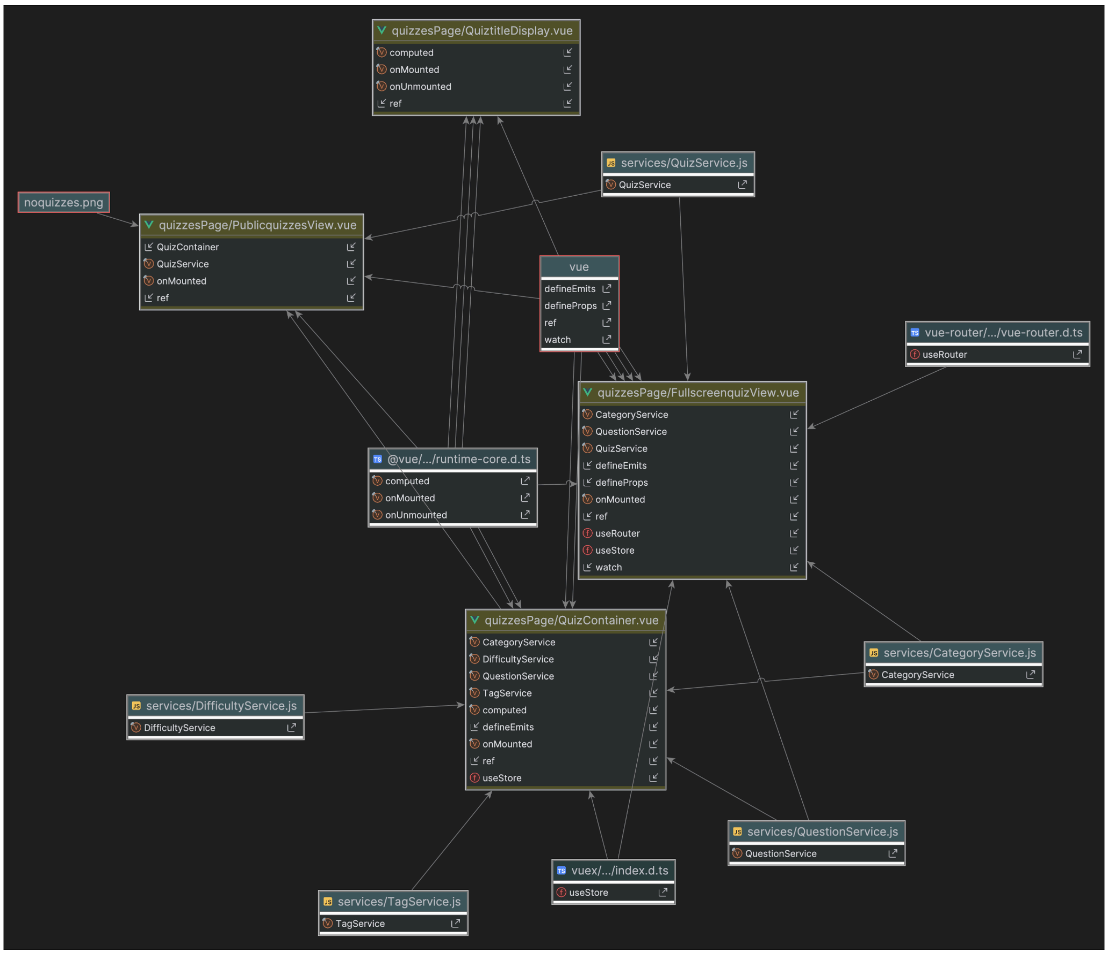
Store


services/QuizService.js
QuizService
vuex/.../index.d.ts
createStore
services/AttemptService.js
AttemptService
modules/quizAttempt.js
AttemptService
modules/quizzes.js
QuizService
uuidv4
store/index.js
createPersistedState
createStore
quizAttempt
quizzes
user
uuid
v4
vuex-persistedstate/.../index.d.ts
modules/user.js
SessionToken
UserService
router
services/UserService.js
UserService
features/SessionToken.js
SessionToken
router/index.js
@vue/.../runtime-core.d.ts
computed
vue
ref
store/counter.js
computed
defineStore
ref
pinia/.../pinia.d.ts
defineStore

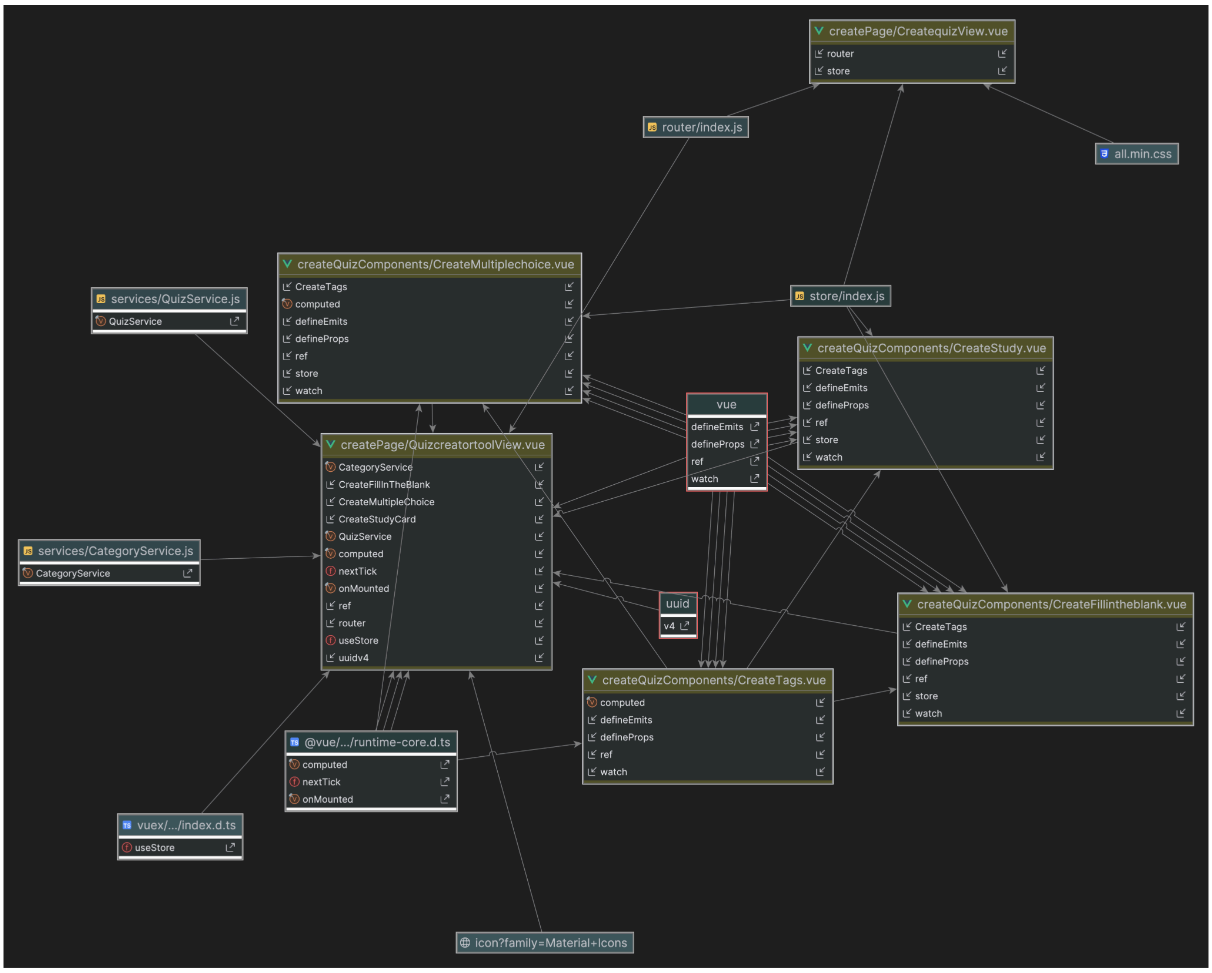
DisplayPage


SpecifiedQuizDisplays/MultiplechoiceQuizDisplayer.vue
MultiplechoiceDisplayer
defineProps
onMounted
ref
displayQuiz/FillintheblankDisplayer.vue
defineProps
ref
services/AttemptService.js
AttemptService
quanda_logo.png
displayQuiz/MultiplechoiceDisplayer.vue
defineProps
ref
SpecifiedQuizDisplays/FillintheblankQuizDisplayer.vue
defineEmits
defineProps
ref
vue
defineEmits
defineProps
ref
watch
SpecifiedQuizDisplays/MixedQuizDisplayer.vue
AttemptService
FillInTheBlankDisplayer
MultipleChoiceDisplayer
StudyCardDisplayer
computed
onMounted
ref
router
useStore
watch
displayPage/WelcomeIntro.vue
router
router/index.js
@vue/.../runtime-core.d.ts
computed
onMounted
SpecifiedQuizDisplays/StudycardsQuizDisplayer.vue
StudycardDisplayer
onMounted
ref
displayQuiz/FrontpageFlipcard.vue
ref
all.min.css
all.min.css
displayPage/QuizDisplayer.vue
MixedQuizDisplayer
computed
onMounted
useRouter
useStore
displayQuiz/StudycardDisplayer.vue
defineProps
ref
displayPage/FlipcardDisplayer.vue
FrontpageFlipcard
icon?family=Material+Icons
vuex/.../index.d.ts
useStore
vue-router/.../vue-router.d.ts
useRouter

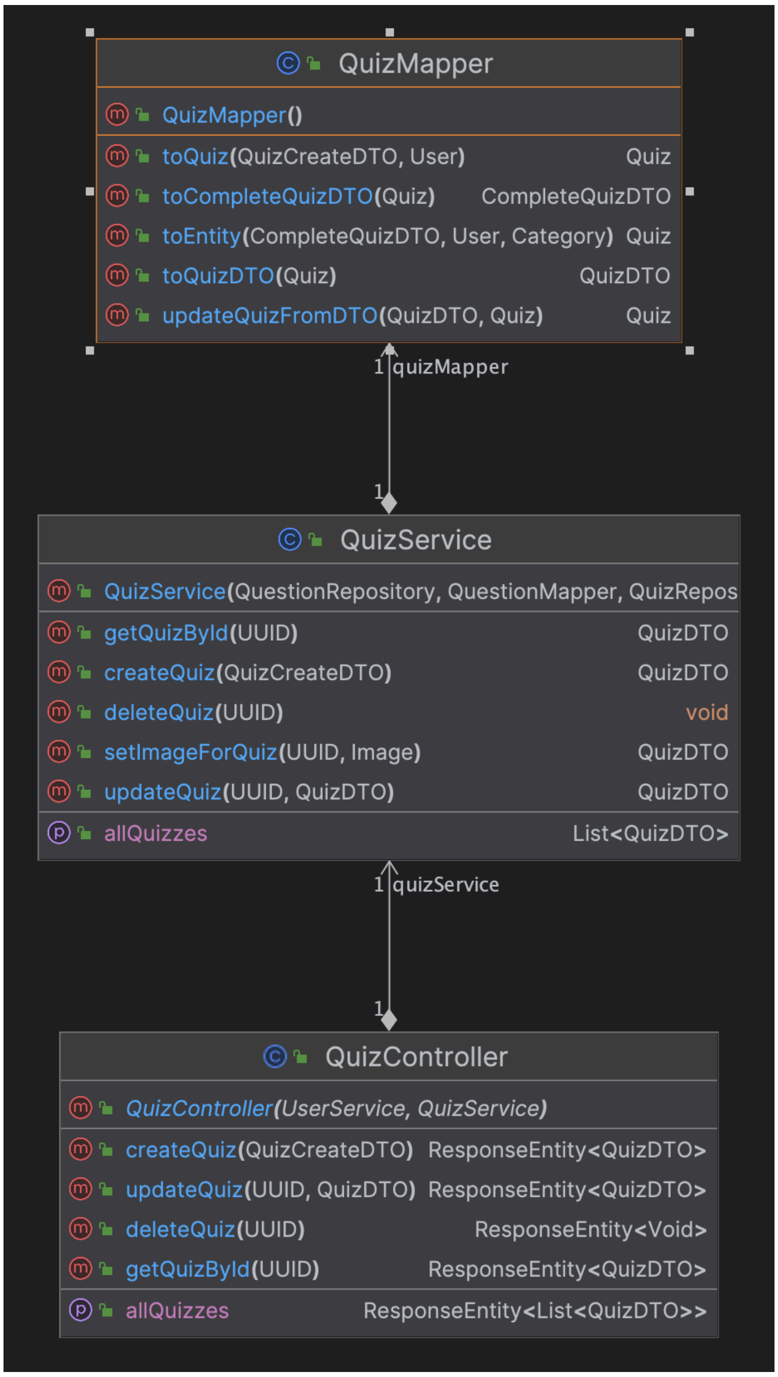
CreatePage

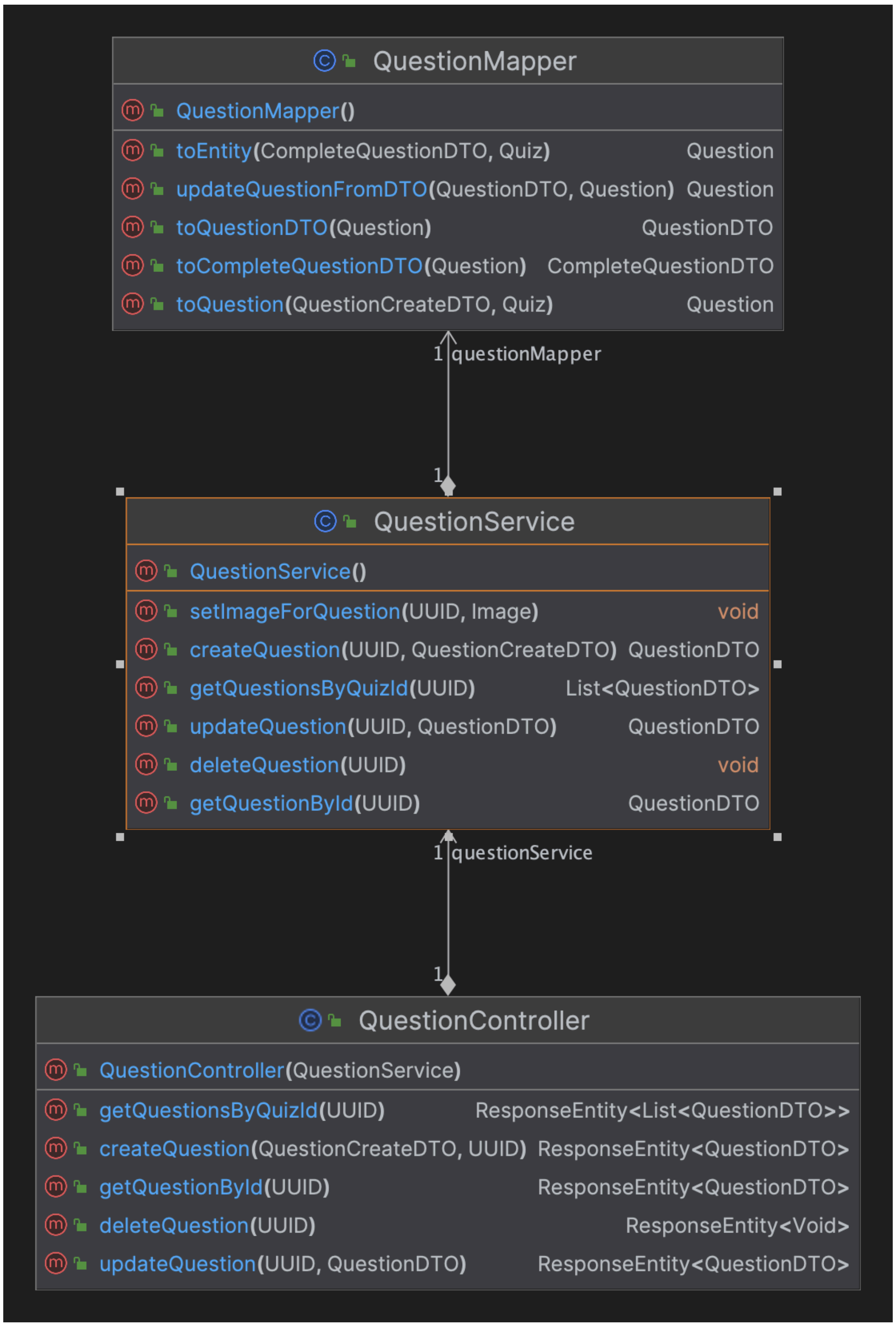


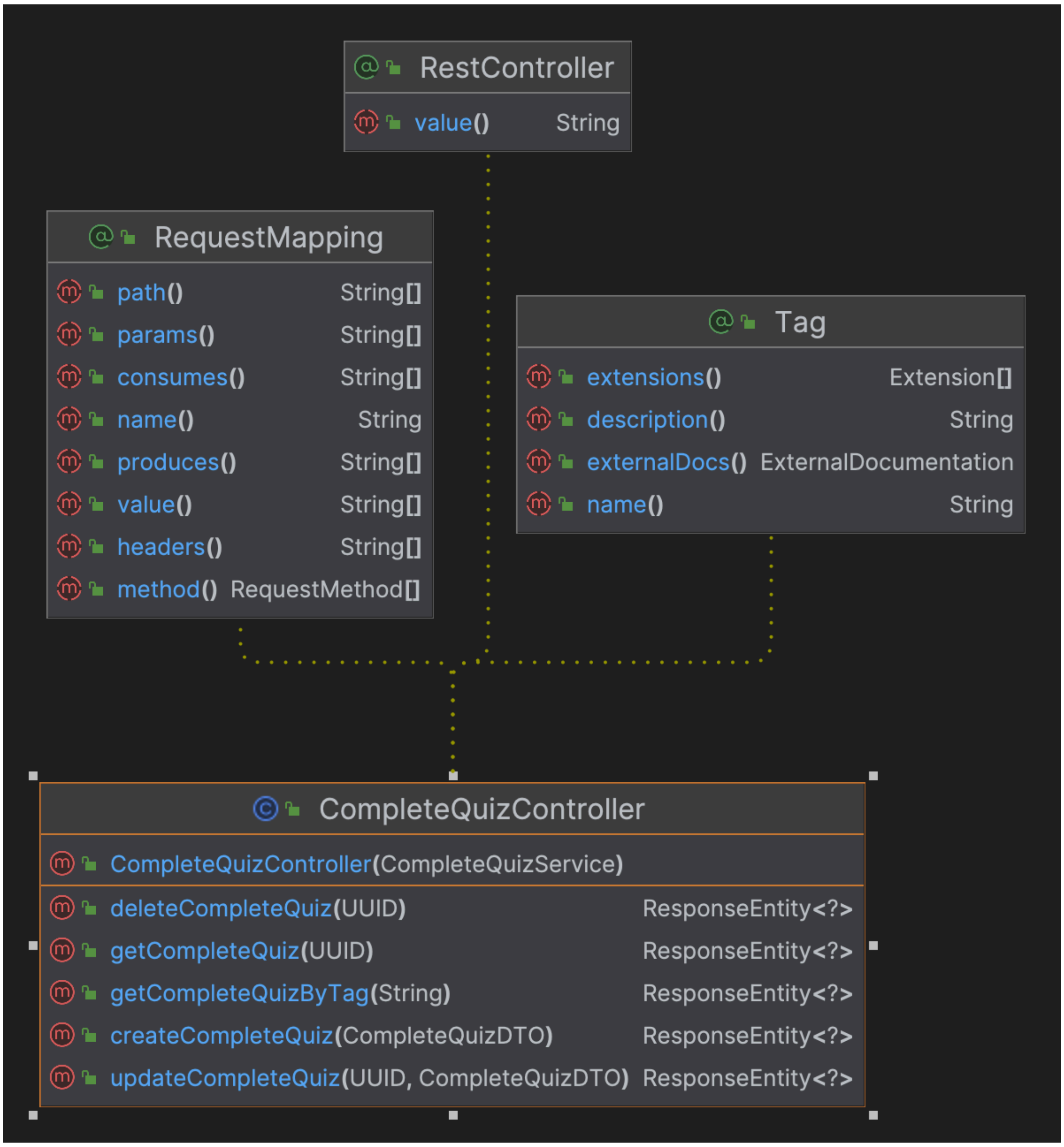
QuizzesPage



Backend


QuizMapper
QuizMapper()
toQuiz(QuizCreateDTO, User) Quiz
toCompleteQuizDTO(Quiz) CompleteQuizDTO
toEntity(CompleteQuizDTO, User, Category) Quiz
toQuizDTO(Quiz) QuizDTO
updateQuizFromDTO(QuizDTO, Quiz) Quiz
1 quizMapper
1
QuizService
QuizService(QuestionRepository, QuestionMapper, QuizRepos
getQuizById(UUID) QuizDTO
createQuiz(QuizCreateDTO) QuizDTO
deleteQuiz(UUID) void
setImageForQuiz(UUID, Image) QuizDTO
updateQuiz(UUID, QuizDTO) QuizDTO
allQuizzes List<QuizDTO>
1 quizService
1
QuizController
QuizController(UserService, QuizService)
createQuiz(QuizCreateDTO) ResponseEntity<QuizDTO>
updateQuiz(UUID, QuizDTO) ResponseEntity<QuizDTO>
deleteQuiz(UUID) ResponseEntity<Void>
getQuizById(UUID) ResponseEntity<QuizDTO>
allQuizzes ResponseEntity<List<QuizDTO>>

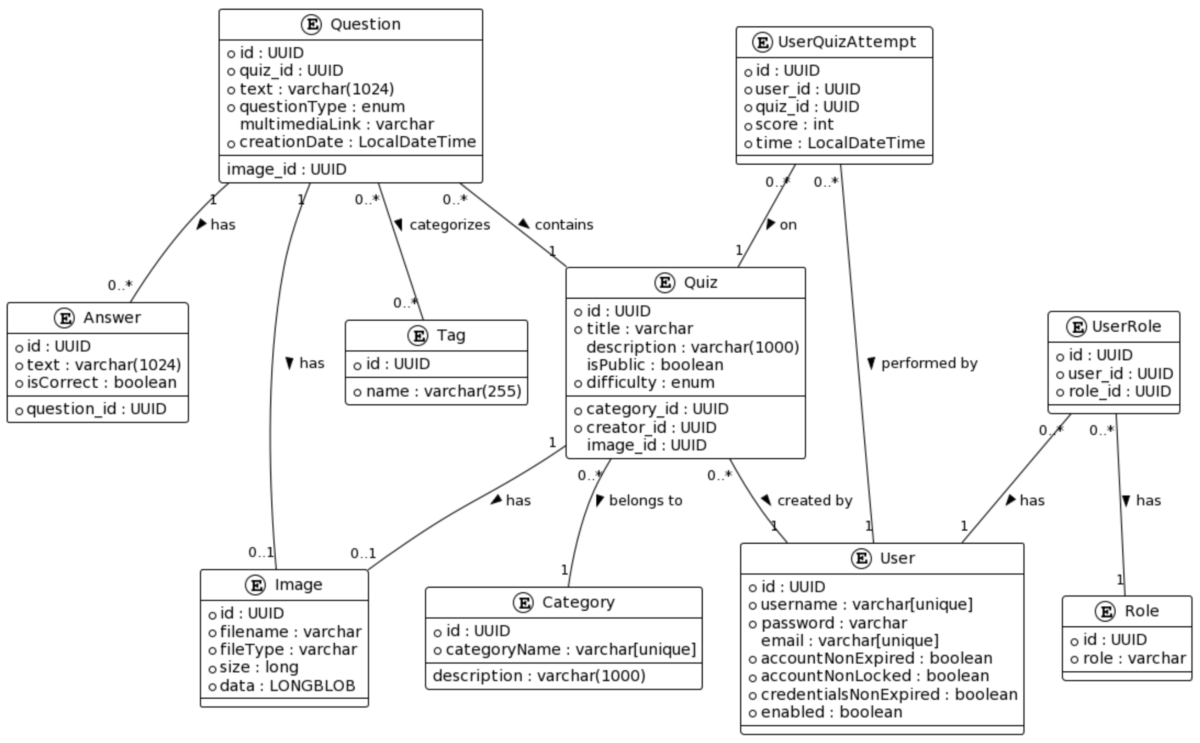

QuestionMapper
QuestionMapper()
toEntity(CompleteQuestionDTO, Quiz) Question
updateQuestionFromDTO(QuestionDTO, Question) Question
toQuestionDTO(Question) QuestionDTO
toCompleteQuestionDTO(Question) CompleteQuestionDTO
toQuestion(QuestionCreateDTO, Quiz) Question
1 questionMapper
1
QuestionService
QuestionService()
setImageForQuestion(UUID, Image) void
createQuestion(UUID, QuestionCreateDTO) QuestionDTO
getQuestionsByQuizId(UUID) List<QuestionDTO>
updateQuestion(UUID, QuestionDTO) QuestionDTO
deleteQuestion(UUID) void
getQuestionById(UUID) QuestionDTO
1 questionService
1
QuestionController
QuestionController(QuestionService)
getQuestionsByQuizId(UUID) ResponseEntity<List<QuestionDTO>>
createQuestion(QuestionCreateDTO, UUID) ResponseEntity<QuestionDTO>
getQuestionById(UUID) ResponseEntity<QuestionDTO>
deleteQuestion(UUID) ResponseEntity<Void>
updateQuestion(UUID, QuestionDTO) ResponseEntity<QuestionDTO>


RestController
value() String
RequestMapping
path() String[]
params() String[]
consumes() String[]
name() String
produces() String[]
value() String[]
headers() String[]
method() RequestMethod[]
Tag
extensions() Extension[]
description() String
externalDocs() ExternalDocumentation
name() String
CompleteQuizController
CompleteQuizController(CompleteQuizService)
deleteCompleteQuiz(UUID) ResponseEntity<?>
getCompleteQuiz(UUID) ResponseEntity<?>
getCompleteQuizByTag(String) ResponseEntity<?>
createCompleteQuiz(CompleteQuizDTO) ResponseEntity<?>
updateCompleteQuiz(UUID, CompleteQuizDTO) ResponseEntity<?>

Database

Model EER diagram

This diagram displays the communication between the different tables within the database.


Question
id : UUID
quiz_id : UUID
text : varchar(1024)
questionType : enum
multimediaLink : varchar
creationDate : LocalDateTime
image_id : UUID
UserQuizAttempt
id : UUID
user_id : UUID
quiz_id : UUID
score : int
time : LocalDateTime
Answer
id : UUID
text : varchar(1024)
isCorrect : boolean
question_id : UUID
Tag
id : UUID
name : varchar(255)
Quiz
id : UUID
title : varchar
description : varchar(1000)
isPublic : boolean
difficulty : enum
category_id : UUID
creator_id : UUID
image_id : UUID
UserRole
id : UUID
user_id : UUID
role_id : UUID
Image
id : UUID
filename : varchar
fileType : varchar
size : long
data : LONGBLOB
Category
id : UUID
categoryName : varchar[unique]
description : varchar(1000)
User
id : UUID
username : varchar[unique]
password : varchar
email : varchar[unique]
accountNonExpired : boolean
accountNonLocked : boolean
credentialsNonExpired : boolean
enabled : boolean
Role
id : UUID
role : varchar
has
has
categorizes
contains
on
performed by
has
belongs to
created by
has
has
1
0..*
1
0..1
0..*
0..*
0..*
1
0..*
1
0..*
1
1
0..1
0..*
1
0..*
1
0..*
1
0..*
1