# UML system documentation

**Team: Digital Life 5**

**Team members:**

Bugra Karaaslan
Lawrance Bahem
Lucas Melis
Tico Vermeer
Moustafa Fadil

# Introduction

- Enterprise Web Application is developed using Spring Boot(backend) with Angular(front-end) framework.
- Project is focused mainly on handling document based projects.
- “User” can choose activities by filtering them on a specific category
- “User” can change the colors of the website by giving a command to the speech recognition
- Admin can manage the activities/categories/users like adding, deleting and editing them
- This web application is secure and robust. We have implemented JWT based Authentication technique in our application. The user will be logged out after an hour and directed to the login page.
- Using Postgres and Spring Data JPA for database management.

# Product vision

An Enterprise Web Application that aims at helping the users (elderly people) with visual impairments to do all kinds of activities that they'd like to do by finding an activity that suits their interests.

Absolutely there is a possibility to choose interests when creating a new account, which will help the user to see the appropriate activities based on the selected interests. The activities are divided into categories, to make it easier for the user to find certain activities of a certain category. By using the filter, it is possible to find the needed category/activity.

During the development process, we had to test the application with real users with visual impairments to improve the application based on the user's feedback. We tested the application with several color combinations to find out which color is the best to choose. But that was difficult since color blindness affects people in different manners and that makes it difficult to choose a safe color that applies to all colorblind users. To have this problem solved, we've built a speech recognition to help the user to apply the color as preferred. This function distinguishes our application from other applications since the possibility of changing the color at other websites is not easy to find. By clicking on the mic the user can give a command to change the color of the whole website and it's also possible to navigate through the app by using this speech recognition. We have made progress also by making it possible to change the color by clicking on a drop-down from the menu bar and choosing the preferred color. These features have helped us to feel confident since these features are rarely implemented in other applications (competitors) like Uitjes.nl, which has limited features for the user to use. That will undoubtedly help us to attract more users to enjoy our services that we've provided in our application.

# Important epic stories

- Speech recognition
- Application is manageable as an admin
- Users can make an account and edit it later
- All activities overview with filters and search bar
- All categories have their own page with every activity of that category
- The website gives a few suggestions for the user of activities that they may like according to their interests

## Speech recognition

Our website is designed for people with visual impairments. So that means that they can’t see everything on the website very well. For those users, we have implemented a speech recognition function which the user can use to change the color of the website and navigate throughout the website. The usability had to be very straightforward, so when the user enters the site for the first time a voice says that there is a button at the top of the website where the user can talk to. The speech recognition function uses a few keywords, which is very handy because the user can speak in full sentences and the website will know what to do because of the used keywords. When a user has navigated to an activity there is another function that reads out the description of the activity.

## Application is manageable as an admin

The website is very manageable for the admins, an admin gets an extra button in the navbar at the top of the website. With that button, the admin will be redirected to a page with all the admin functions. The admin functions include 3 tabs, one for user-management, one for activity-management and one for category-management. In those tabs, the admin can change all the details of the activities and categories and they can delete them as well. The admin can also block users and make other users admin.

## Users are able to make an account and edit it later

On our website, users can make their own accounts. Based on their account the website will give the user a few activities that they may like. If a user is logged in they can leave reactions on activities as well. If a user would like to change their information they can. On their profile page, the user can change all their details, if the user does that the suggested activities will change as well.

## All activities overview with filters and search bar

The website consists of a page where all activities are shown together where users can sign up for. The activities are divided into different categories and users are able to filter based on the category. Because of this users can see only the activities which they like. Besides this, users also have the option to type a certain activity in the search bar found on top of the overview page. So if the user has a specific activity in mind which they find interesting, they can type it in the bar and see if it is available.

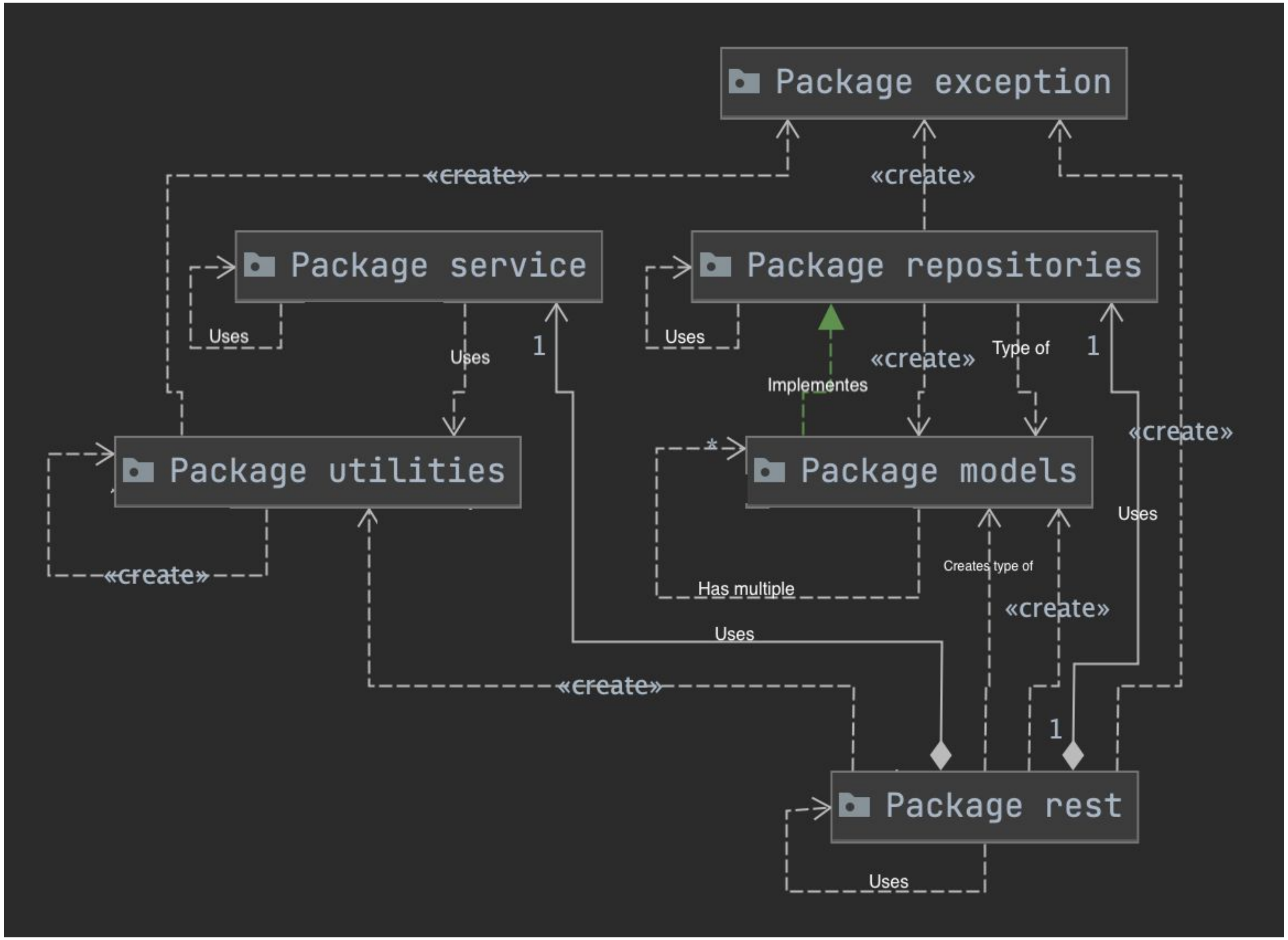
## All categories have their own page with every activity of that category

On the home-page of the website, all categories are displayed as cards. If a user clicks on one of those cards, the user will be redirected to the page of that category. On that page, all activities which are related to that category are displayed. The user can search through those activities with a search bar that looks through the activities and displays only the activities that have any attribute which matches the search input.

## The website gives a few suggestions for the user of activities that they may like according to their interests

When a user has made an account on the website. The interests that they have selected are compared to the categories linked to all of the activities. This is what the matching functionality is used for. By comparing the interests to the categories of the activities we are able to show only the activities where the categories accord to their own interests. This means the user will only see activities where they are interested in. A user can always change their interests later if they prefer to see more activities that they may like or if they simply forgot to add one to their profile.

# Layered Architecture Package Diagram



Exception package: has multiple exception classes that are used in other controllers, repositories and utilities like in JWttoken class which will throw an exception if the token invalid/wrong. Also, they are used in repositories for a certain situation like logging in with a wrong password/email.

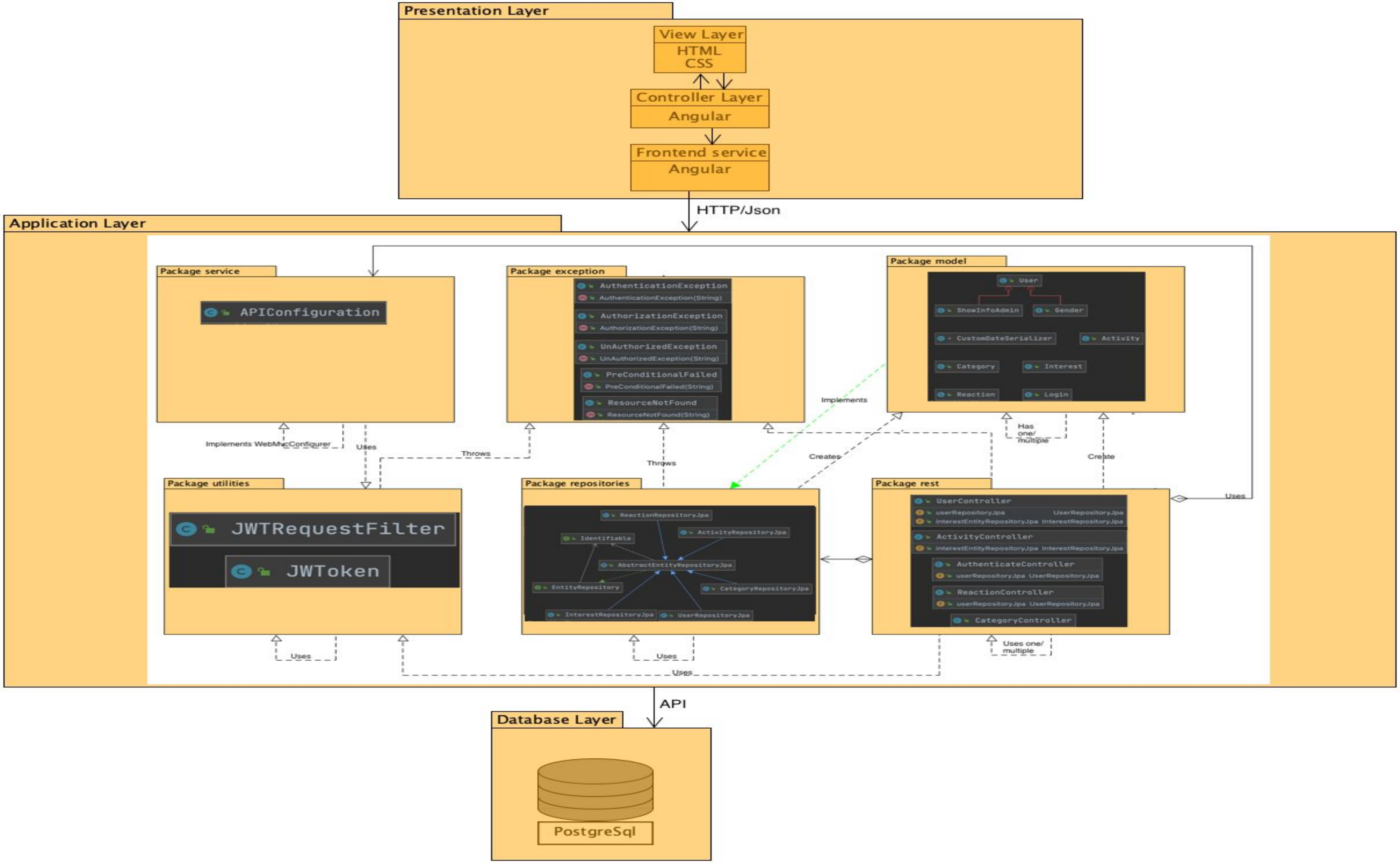
Package service: has ApiConfiguration that implements WebMvcConfigurer and initializes JWToken of the package utility

Package models: Some models are used in other models like User has one/many interests, also models are used in controllers, for example,userController would get a request to get the user information on a certain endpoint and the response will be a user which is a model

Package rest: Some controllers use the service ApiConfiguration to decode the token that is given in the header of the request that is done also by using the utility JWToken that has the encode and decode methods. Some controllers will throw an exception on a certain situation as mentioned previously, which means they create an exception of the exception package

Package repositories: Some repositories depend on another repository, like userRepository is depending on the InterestRepository because if we need to get for a certain user his/her interests we need to use the InterestRepository. Each repository is of a specific type of model.

Package utilities: This package contains the JWToken utilities, like JWToken class which has the functionality to encode and decode the sent/received token. In addition, it takes into account the expiration time of each token by encoding the expiration time of each token and decoding to check if the token is still valid when the client makes a request.


Presentation Layer
View Layer
HTML
CSS
Controller Layer
Angular
Frontend service
Angular
HTTP/Json
Application Layer
Package service
APIConfiguration
Package exception
AuthenticationException
AuthenticationException(String)
AuthorizationException
AuthorizationException(String)
UnAuthorizedException
UnAuthorizedException(String)
PreConditionalFailed
PreConditionalFailed(String)
ResourceNotFound
ResourceNotFound(String)
Package model
User
ShowInfoAdmin
Gender
CustomDateSerializer
Activity
Category
Interest
Reaction
Login
Implements
Has one/ multiple
Implements WebMvcConfigurer
Uses
Throws
Throws
Creates
Create
Uses
Package utilities
JWTRequestFilter
JWToken
Package repositories
ReactionRepositoryJpa
ActivityRepositoryJpa
Identifiable
AbstractEntityRepositoryJpa
EntityRepository
CategoryRepositoryJpa
InterestRepositoryJpa
UserRepositoryJpa
Package rest
UserController
userRepositoryJpa UserRepositoryJpa
interestEntityRepositoryJpa InterestRepositoryJpa
ActivityController
interestEntityRepositoryJpa InterestRepositoryJpa
AuthenticateController
userRepositoryJpa UserRepositoryJpa
ReactionController
userRepositoryJpa UserRepositoryJpa
CategoryController
Uses
Uses
Uses
Uses one/ multiple
API
Database Layer
PostgreSql

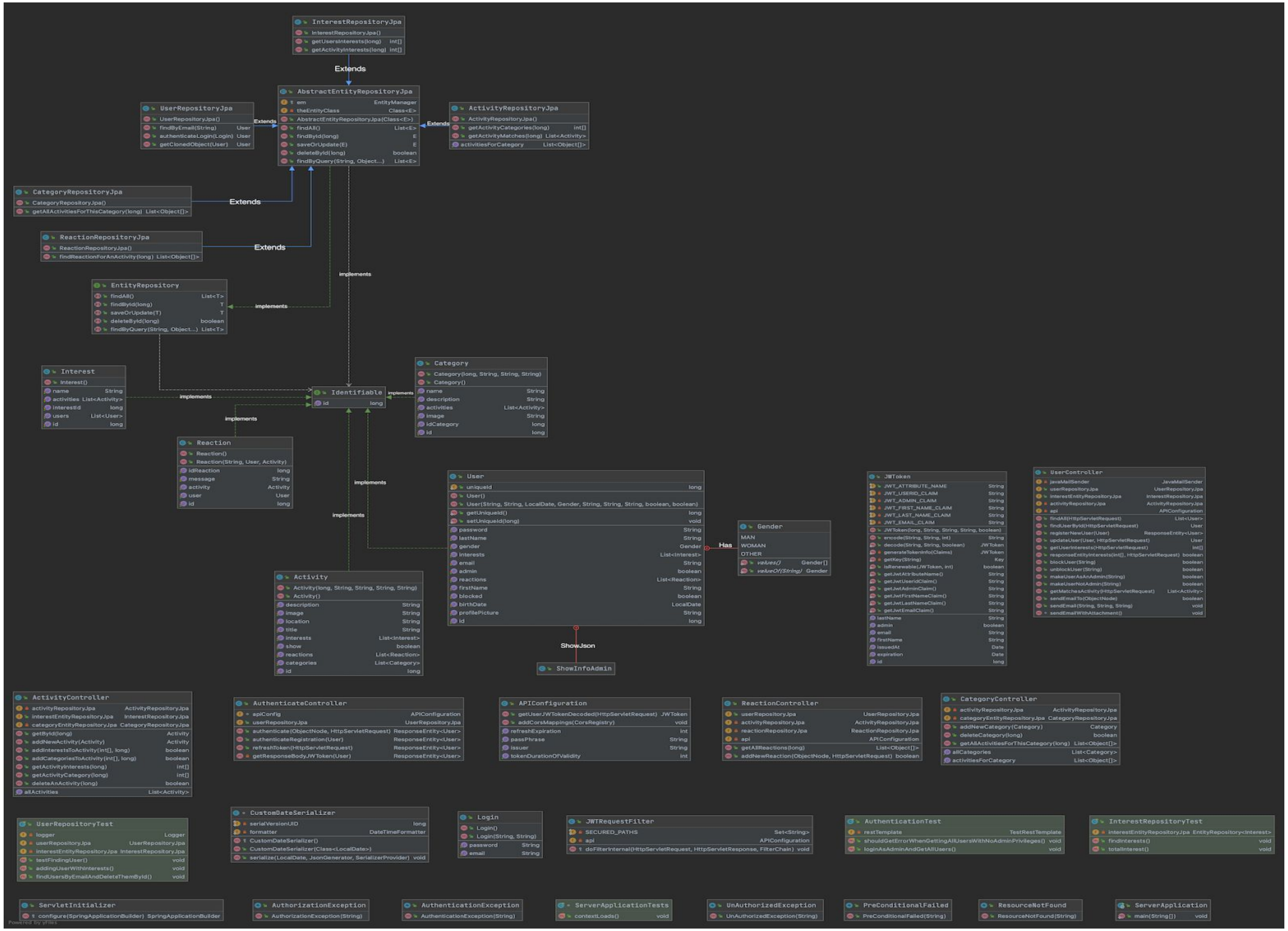
**Presentation Layer:** The frontend of the application. Contains HTML/CSS and web framework Angular. This layer displays the data it retrieves and does as little "thinking" as possible.

**Application Layer:** This is the middleware of our application (as shown in the previous picture of the packages). It has been implemented in Java using the spring-boot framework. This layer is the "engine" of our app and it connects the presentation layer to the data layer. We do not want any Views or UI in this layer, as this is the "great thinker" of the squad.

**Data Layer:** Our database/storage system. It is  PostgreSQL. The data would be accessed by the application layer via API calls (the API call itself is triggered by the presentation layer, but the presentation layer doesn't know what the application layer will do with that call, it's a blackbox between the two).

# Navigable Class Diagram

This is the whole class diagram of our application, the relation between each repository and controller is clearly given in each controller class
Below is an explanation of the class diagram:


InterestRepositoryJpa
InterestRepositoryJpa()
getUsersInterests(long) int[]
getActivityInterests(long) int[]
Extends
AbstractEntityRepositoryJpa
em EntityManager
theEntityClass Class<E>
AbstractEntityRepositoryJpa(Class<E>)
findAll() List<E>
findById(long) E
saveOrUpdate(E) E
deleteById(long) boolean
findByQuery(String, Object...) List<E>
UserRepositoryJpa
UserRepositoryJpa()
findByEmail(String) User
authenticateLogin(Login) User
getClonedObject(User) User
Extends
ActivityRepositoryJpa
ActivityRepositoryJpa()
getActivityCategories(long) int[]
getActivityMatches(long) List<Activity>
activitiesForCategory List<Object[]>
Extends
CategoryRepositoryJpa
CategoryRepositoryJpa()
getAllActivitiesForThisCategory(long) List<Object[]>
Extends
ReactionRepositoryJpa
ReactionRepositoryJpa()
findReactionForAnActivity(long) List<Object[]>
Extends
implements
EntityRepository
findAll() List<T>
findById(long) T
saveOrUpdate(T) T
deleteById(long) boolean
findByQuery(String, Object...) List<T>
implements
Interest
Interest()
name String
activities List<Activity>
interestId long
users List<User>
id long
implements
Identifiable
id long
implements
Category
Category(long, String, String, String)
Category()
name String
description String
activities List<Activity>
image String
idCategory long
id long
implements
Reaction
Reaction()
Reaction(String, User, Activity)
idReaction long
message String
activity Activity
user User
id long
implements
implements
User
uniqueId long
User()
User(String, String, LocalDate, Gender, String, String, String, boolean, boolean)
getUniqueId() long
setUniqueId(long) void
password String
lastName String
gender Gender
interests List<Interest>
email String
admin boolean
reactions List<Reaction>
firstName String
blocked boolean
birthDate LocalDate
profilePicture String
id long
Has
Gender
MAN
WOMAN
OTHER
values() Gender[]
valueOf(String) Gender
ShowJson
ShowInfoAdmin
Activity
Activity(long, String, String, String, String)
Activity()
description String
image String
location String
title String
interests List<Interest>
show boolean
reactions List<Reaction>
categories List<Category>
id long
JWToken
JWT_ATTRIBUTE_NAME String
JWT_USERID_CLAIM String
JWT_ADMIN_CLAIM String
JWT_FIRST_NAME_CLAIM String
JWT_LAST_NAME_CLAIM String
JWT_EMAIL_CLAIM String
JWToken(long, String, String, String, boolean)
encode(String, String, int) String
decode(String, String, boolean) JWToken
generateTokenInfo(Claims) JWToken
getKey(String) Key
isRenewable(JWToken, int) boolean
getJwtAttributeName() String
getJwtUseridClaim() String
getJwtAdminClaim() String
getJwtFirstNameClaim() String
getJwtLastNameClaim() String
getJwtEmailClaim() String
lastName String
admin boolean
email String
firstName String
issuedAt Date
expiration Date
id long
UserController
javaMailSender JavaMailSender
userRepositoryJpa UserRepositoryJpa
interestEntityRepositoryJpa InterestRepositoryJpa
activityRepositoryJpa ActivityRepositoryJpa
api APIConfiguration
findAll(HttpServletRequest) List<User>
findUserById(HttpServletRequest) User
registerNewUser(User) ResponseEntity<User>
updateUser(User, HttpServletRequest) User
getUserInterests(HttpServletRequest) int[]
responseEntityInterests(int[], HttpServletRequest) boolean
blockUser(String) boolean
unblockUser(String) boolean
makeUserAsAnAdmin(String) boolean
makeUserNotAdmin(String) boolean
getMatchesActivity(HttpServletRequest) List<Activity>
sendEmailTo(ObjectNode) boolean
sendEmail(String, String, String) void
sendEmailWithAttachment() void
ActivityController
activityRepositoryJpa ActivityRepositoryJpa
interestEntityRepositoryJpa InterestRepositoryJpa
categoryEntityRepositoryJpa CategoryRepositoryJpa
getById(long) Activity
addNewActivity(Activity) Activity
addInterestsToActivity(int[], long) boolean
addCategoriesToActivity(int[], long) boolean
getActivityInterests(long) int[]
getActivityCategory(long) int[]
deleteAnActivity(long) boolean
allActivities List<Activity>
AuthenticateController
apiConfig APIConfiguration
userRepositoryJpa UserRepositoryJpa
authenticate(ObjectNode, HttpServletRequest) ResponseEntity<User>
authenticateRegistration(User) ResponseEntity<User>
refreshToken(HttpServletRequest) ResponseEntity<User>
getResponseBodyJWToken(User) ResponseEntity<User>
APIConfiguration
getUserJWTokenDecoded(HttpServletRequest) JWToken
addCorsMappings(CorsRegistry) void
refreshExpiration int
passPhrase String
issuer String
tokenDurationOfValidity int
ReactionController
userRepositoryJpa UserRepositoryJpa
activityRepositoryJpa ActivityRepositoryJpa
reactionRepositoryJpa ReactionRepositoryJpa
api APIConfiguration
getAllReactions(long) List<Object[]>
addNewReaction(ObjectNode, HttpServletRequest) boolean
CategoryController
activityRepositoryJpa ActivityRepositoryJpa
categoryEntityRepositoryJpa CategoryRepositoryJpa
addNewCategory(Category) Category
deleteCategory(long) boolean
getAllActivitiesForThisCategory(long) List<Object[]>
allCategories List<Category>
activitiesForCategory List<Object[]>
UserRepositoryTest
logger Logger
userRepositoryJpa UserRepositoryJpa
interestEntityRepositoryJpa InterestRepositoryJpa
testFindingUser() void
addingUserWithInterests() void
findUsersByEmailAndDeleteThemById() void
CustomDateSerializer
serialVersionUID long
formatter DateTimeFormatter
CustomDateSerializer()
CustomDateSerializer(Class<LocalDate>)
serialize(LocalDate, JsonGenerator, SerializerProvider) void
Login
Login()
Login(String, String)
password String
email String
JWTRequestFilter
SECURED_PATHS Set<String>
api APIConfiguration
doFilterInternal(HttpServletRequest, HttpServletResponse, FilterChain) void
AuthenticationTest
restTemplate TestRestTemplate
shouldGetErrorWhenGettingAllUsersWithNoAdminPrivileges() void
loginAsAdminAndGetAllUsers() void
InterestRepositoryTest
interestEntityRepositoryJpa EntityRepository<Interest>
findInterests() void
totalInterest() void
ServletInitializer
configure(SpringApplicationBuilder) SpringApplicationBuilder
AuthorizationException
AuthorizationException(String)
AuthenticationException
AuthenticationException(String)
ServerApplicationTests
contextLoads() void
UnAuthorizedException
UnAuthorizedException(String)
PreConditionalFailed
PreConditionalFailed(String)
ResourceNotFound
ResourceNotFound(String)
ServerApplication
main(String[]) void
Powered by yFiles

**Models**
Each category has multiple activities and vice versa.
Each user has multiple interests and vice versa, and each activity has multiple interests and vice versa.
Each activity has zero/multiple reactions and each user has zero/multiple reactions.

**Repository**
Each modal has its own repository. The repositories are used to get the data from the database, like deleting, editing and saving. Also, repositories have their own functionalities like findByEmail in UserRepository which finds a user based on the given email. Not every Repository has the same functionalities. General functionalities for each Repository are implemented from the EntityRepository like findAll,findById, saveOrUpdate and deleteById.

**Controllers**
We need controllers so clients can make requests and get responses. all these things go through the controller. If a client wants to log in, they go to the login page. This is an example of a request that is made in the controller. Another example is if a client wants to see all the activities. To see all the activities the client has to go to the actual page. This is also a request.

**Service**
In the service package, there is only one file. This file is the APIConfiguration file. This file is used for security. In this file is specified which origins can communicate with the backend of our application. In this file, the token given to a logged-in user is decoded en checked if it is a correct token

**Exception**
Each exception will be thrown at a specific situation for example if the user has entered a wrong password/email

**Utilities**
The JWToken is to be used when encoding/decoding the token of the user, which will be stored in the sessionStorage.
JWTRequestFilter to check(The toke )and filter the requests of the client

## Challenges and alternative

The biggest challenges had to do with the visual impairment of our target audience. Because we didn't have any experience in this regard it was hard for us to figure out the best way to design our application. After discussing and experimenting with different things we came to the conclusion that we wanted to implement some features to help our users out. Features like speech recognition were a great solution for this. Because of the speech recognition, users are able to change the colors of the application, they can navigate through the entire application as well and different forms of text can also be read out loud if the users prefer that.

After some sprint reviews, we received feedback that it would be great if we could find another way to assist the users with their visual impairment next to the speech recognition. This is where we came up with the idea to make two buttons in the navigation bar, a plus button and a minus button. With these buttons, people can increase and decrease the font sizes of all text in the application. By doing this our users have the option to adjust the font size accordingly to their liking. This way they are able to read the text which is shown in the application, even when they have a visual impairment.

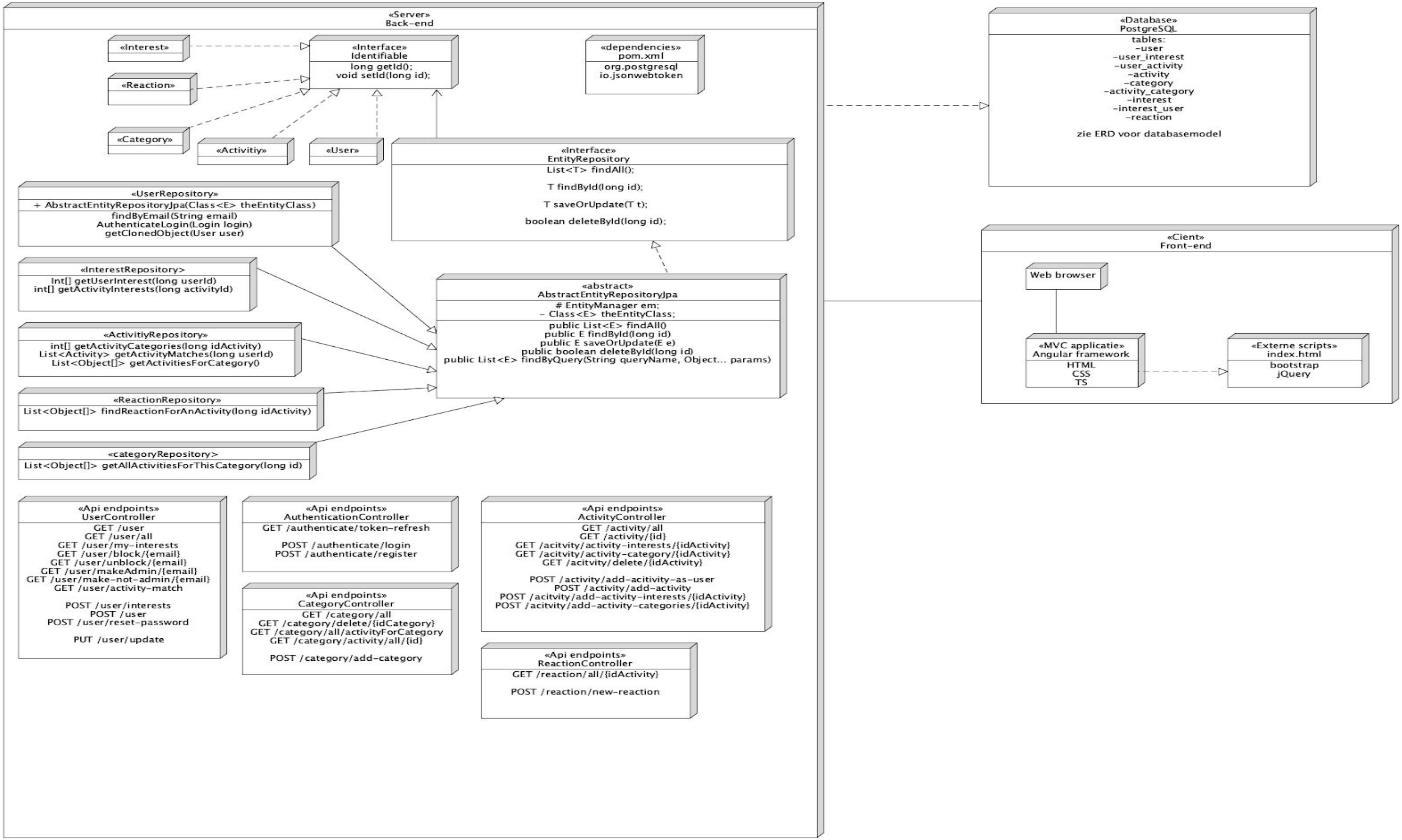
An alternative solution to the unreadable text for some users could be that we entirely changed the font sizes all over the website accordingly so that everyone would be able to read the text. This would likely have been a simple solution but we didn’t choose to solve the problem like this because there are still going to be people who are still able and prefer to read the smaller font sizes. Also, the admin users would most likely prefer that the font sizes aren’t too big. The two buttons that can adjust font sizes are the perfect solution for this, everyone can adjust the font sizes to whatever size they like.

# Deployment Diagram

**Back-end(server-side):**
Here we have the controllers and repositories that are necessary for our product to make API calls to the Database to retrieve the needed data for a certain request, that is made by the user. For security purposes, we’ve implemented the JWToken to track the user’s behavior during visiting our app and making sure that the user will be logged out after an hour of activeness on our app.

**Front-end (client-side):**
Contains HTML/CSS and web framework Angular. This layer shows the data that has been retrieved and does as little "interacting" as possible. We have used extern scripts like bootstrap and jquery for design purposes.

**Database:**
As shown in the figure below, we used PostgreSQL database to store/retrieve data

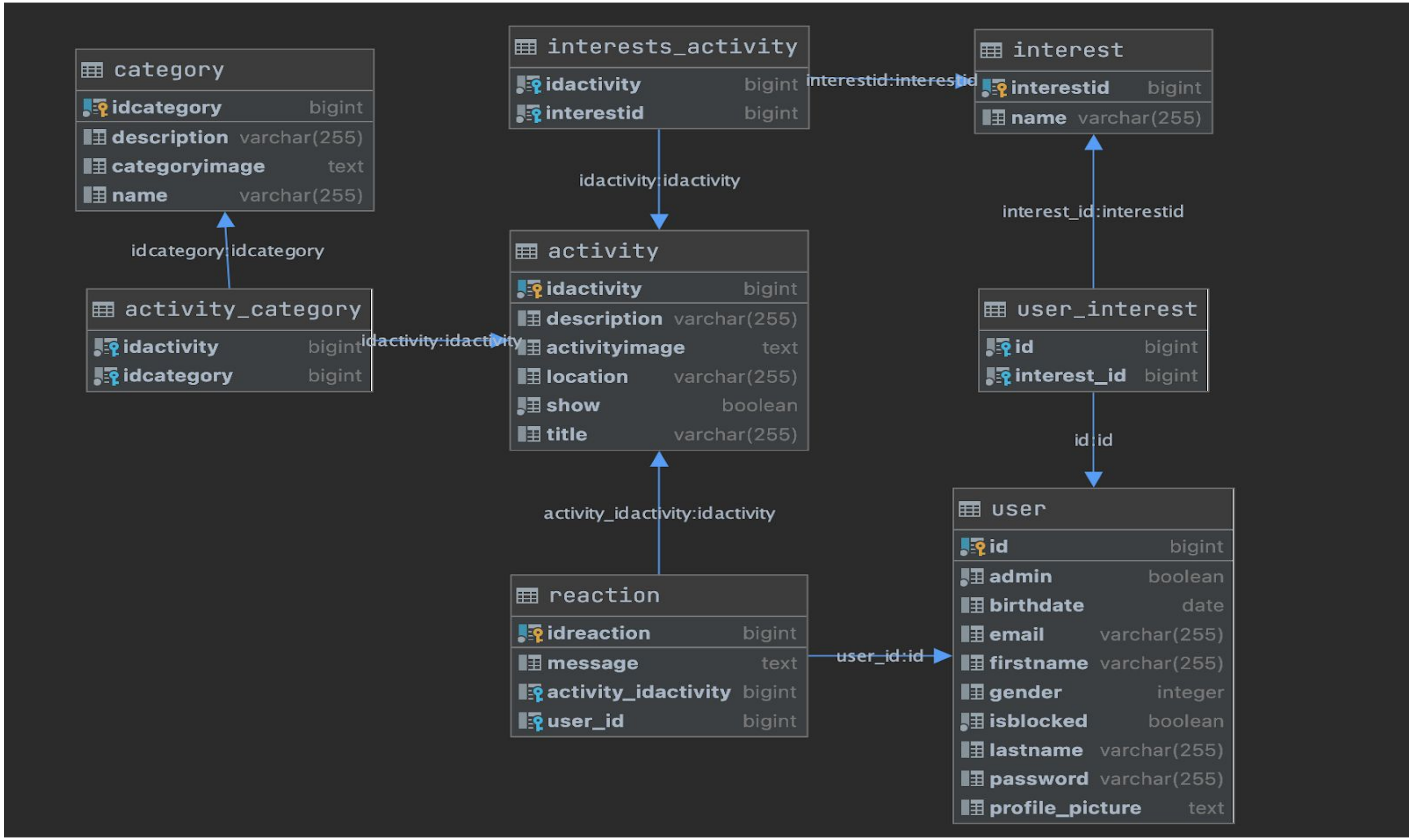

«Server»
Back-end
«Interest»
«Reaction»
«Category»
«Activitiy»
«User»
«Interface»
Identifiable
long getId();
void setId(long id);
«dependencies»
pom.xml
org.postgresql
io.jsonwebtoken
«Interface»
EntityRepository
List<T> findAll();
T findById(long id);
T saveOrUpdate(T t);
boolean deleteById(long id);
«UserRepository»
+ AbstractEntityRepositoryJpa(Class<E> theEntityClass)
findByEmail(String email)
AuthenticateLogin(Login login)
getClonedObject(User user)
«InterestRepository>
Int[] getUserInterest(long userId)
int[] getActivityInterests(long activityId)
«ActivitiyRepository»
int[] getActivityCategories(long idActivity)
List<Activity> getActivityMatches(long userId)
List<Object[]> getActivitiesForCategory()
«ReactionRepository»
List<Object[]> findReactionForAnActivity(long idActivity)
«categoryRepository>
List<Object[]> getAllActivitiesForThisCategory(long id)
«abstract»
AbstractEntityRepositoryJpa
# EntityManager em;
- Class<E> theEntityClass;
public List<E> findAll()
public E findById(long id)
public E saveOrUpdate(E e)
public boolean deleteById(long id)
public List<E> findByQuery(String queryName, Object... params)
«Api endpoints»
UserController
GET /user
GET /user/all
GET /user/my-interests
GET /user/block/{email}
GET /user/unblock/{email}
GET /user/makeAdmin/{email}
GET /user/make-not-admin/{email}
GET /user/activity-match
POST /user/interests
POST /user
POST /user/reset-password
PUT /user/update
«Api endpoints»
AuthenticationController
GET /authenticate/token-refresh
POST /authenticate/login
POST /authenticate/register
«Api endpoints»
CategoryController
GET /category/all
GET /category/delete/{idCategory}
GET /category/all/activityForCategory
GET /category/activity/all/{id}
POST /category/add-category
«Api endpoints»
ActivityController
GET /activity/all
GET /activity/{id}
GET /acitvity/activity-interests/{idActivity}
GET /acitvity/activity-category/{idActivity}
GET /acitvity/delete/{idActivity}
POST /activity/add-acitivity-as-user
POST /activity/add-activity
POST /acitvity/add-activity-interests/{idActivity}
POST /acitvity/add-activity-categories/{idActivity}
«Api endpoints»
ReactionController
GET /reaction/all/{idActivity}
POST /reaction/new-reaction
«Database»
PostgreSQL
tables:
-user
-user_interest
-user_activity
-activity
-category
-activity_category
-interest
-interest_user
-reaction
zie ERD voor databasemodel
«Cient»
Front-end
Web browser
«MVC applicatie»
Angular framework
HTML
CSS
TS
«Externe scripts»
index.html
bootstrap
jQuery

# Postgresql EERD

In our EERD we have eight entities. The most important entities are the User, Activity, Interest, Reaction and category. The relation between the activity and category entity is many to many because an activity can belong to one or more categories and each category can have one or more activities that belong to it.

In this case, we created the table ‘activity_category’. Another table that exists due to the many to many relationship is the “user_interest”. A user can have one or more interests and also an interest can belong to one or more users. The third between table is the “interest_activity”. An activity can belong to an interest, but also an interest can have one or more activities that belong to it.

The entity without a many to many relation is the reaction entity. This entity has a many to one relation with the entity activity, a reaction belongs only to one activity and an activity has one or many reactions. The relation between reaction and user entity is a many to one relation. A user can have one or more reactions, but a reaction belongs to one user.



| category | |
|---|---|
| idcategory | bigint |
| description | varchar(255) |
| categoryimage | text |
| name | varchar(255) |

| interests_activity | |
|---|---|
| idactivity | bigint |
| interestid | bigint |

| interest | |
|---|---|
| interestid | bigint |
| name | varchar(255) |

| activity_category | |
|---|---|
| idactivity | bigint |
| idcategory | bigint |

| activity | |
|---|---|
| idactivity | bigint |
| description | varchar(255) |
| activityimage | text |
| location | varchar(255) |
| show | boolean |
| title | varchar(255) |

| user_interest | |
|---|---|
| id | bigint |
| interest_id | bigint |

| reaction | |
|---|---|
| idreaction | bigint |
| message | text |
| activity_idactivity | bigint |
| user_id | bigint |

| user | |
|---|---|
| id | bigint |
| admin | boolean |
| birthdate | date |
| email | varchar(255) |
| firstname | varchar(255) |
| gender | integer |
| isblocked | boolean |
| lastname | varchar(255) |
| password | varchar(255) |
| profile_picture | text |

# Analytical reflection

Due to the problems that our target group (elderly people with visual impairments) have, we had to be careful with choosing the right color and contrast design. Before we started with designing we researched partially-sighted people. We made a few colored sketches to test them with real users that are partially-sighted, it was impossible to determine which color is the safest to use in our design because each user has seen the color in a different manner than reality.

We had to use colors with high contrast to make sure that the colors aren't annoying for our target group. The problem was way bigger than we thought, that’s why we had to implement speech recognition to make it possible for our target group to change the color of the website as preferred. However, that was not user friendly for the majority of our elderly target group, so we needed to make a drop-down menu item with standard colors, by just clicking on one of these colors the whole app will be colored with the selected color.

Also, we’ve taken into account that most of our users are not familiar with modern technology, due to that we extended our speech recognition by adding an extra feature to navigate through the page by giving a command to the speech recognition and that makes it more user friendly.

Therefore, we tested our app again to get feedback on what we’ve added to our app, the users were enthusiastic about the features we added. There was negative feedback about the multiple filters on the home page, the user had to fill all filters to find an appropriate activity/category. We decided to remove the unnecessary filters and place just one search filter instead.

Also, the menu bar wasn’t that clear for some users, because we used icons to refer to a certain page, we’ve been advised to add text below each icon to make it more distinct about what the icon means.

The registration page was also redesigned because, in the beginning, we had checkboxes for the interests, many users didn’t even see the interests while registering. In other words, they’ve skipped choosing an interest. Therefore, we’ve converted the checkboxes into reasonably large pictures.

Font-size was a bit annoying for the most partially-sighted users, they had difficulties reading the text properly because the font-size wasn't readable. We’ve tried to use a global font-size that would be readable for all users, again that was in vain so we had to add a feature to make it possible to change the font-size of the whole app by just clicking on the minus/plus button of the menu bar.