4.(Before analytics) Basic CRUD queries implementation

In this chapter are described as the CRUD operation, described in chapter 3.2.1, are implemented in our project.

* Insert a **User** into the system at the registration time

When we insert a new user into the system we have to store it both in MongoDb and in Neo4j as the cose of SignUp.java shows up:

// Get the Date

LocalDate localDate = birthdayDP.getValue();

Instant instant = Instant.*from*(localDate.atStartOfDay(ZoneId.*systemDefault*()));

Date date = Date.*from*(instant);

// Insert the User’s values using the inputs inserted in the TextField

User user = new User(

false,

surnameTF.getText(),

nameTF.getText(),

usernameTF.getText(),

passwordTF.getText(),

emailTF.getText(),

date,

country.getValue().toString());

// Create a connection to MongoDB and insert the user

UserManager userManager = UserManagerFactory.*buildManager*();

if(userManager.register(user)) {

// Change the label for the result in order to say to the client that the new User has been correctly created

resultLabel = new InvalidFormEntryLabel("Sign up successfully done", 800, 600, true);

resultLabel.setStyle("-fx-background-color: green;");

// ADD IT ALSO IN NEO4J

UserNetworkManager userNetworkManager = UserNetworkManagerFactory.*buildManager*();

try{

userNetworkManager.addUser(user);

} catch(DuplicateUserException due){

System.*out*.println("Duplicate of user");

}

}

Let’s see how the database methods used in this piece of code are implemented. The *userManager.register(user)* is actually implemented by *UserManagerOnMongoDb.java*:

public boolean register(User toRegister) {

Logger.*vvlog*("Trying to register " + toRegister.getUsername());

return insert(toRegister);

}

public boolean insert(Object toInsert) {

if(!(toInsert instanceof User))

return false;

MongoCollection<Document> collection = getCollection(collectionName); //also opens connection

User userToInsert = (User)toInsert;

userToInsert.setPassword(PasswordEncryptor.*encryptPassword*(userToInsert.getPassword()));

Document doc = UserToDocument(userToInsert);

Logger.*vvlog*("ADDED " + doc.toJson());

collection.insertOne(doc);

closeConnection();

return true;

}

The userNetworkManager.addUser(user) is implemented by UserNetworkManagerOnNeo4j.java:

public boolean addUser(User u) throws DuplicateUserException{

if(userAlreadyExists(u))

throw new DuplicateUserException();

String query = "MERGE (u:User { username: $username, country: $country})";

return insert(query, *parameters*("username", u.getUsername(), "country", u.getCountry()));

}

* Create a new **Pokémon** (*admin* only)

For creating a new **Pokémon** a similar procedure of the one described above is made, in the AdminAddRemovePane.java:

…

if (pokemonManager.addPokemon(

new Pokemon(

nameTF.getText(),

types,

Integer.*parseInt*(idTF.getText()),

catchrate,

Integer.*parseInt*(heightTF.getText()),

Integer.*parseInt*(weightTF.getText()),

biologyTF.getText(),

portraitTF.getText(),

spriteTF.getText()

)

) ) {

TeamManager teamManager = TeamManagerFactory.*buildManager*();

try{

teamManager.addPokemon(new Pokemon(

nameTF.getText(),

types,

Integer.*parseInt*(idTF.getText()),

Double.*parseDouble*(catchRateTF.getText()),

Integer.*parseInt*(heightTF.getText()),

Integer.*parseInt*(weightTF.getText()),

biologyTF.getText(),

portraitTF.getText(),

spriteTF.getText()

));

…

Let’s see how the database methods used in this piece of code are implemented. The *pokemonManager.addPokemon(…)* is actually implemented by *PokemonManagerOnMongoDb.java*:

public boolean addPokemon(Pokemon toAdd) {

return insert(toAdd);

}

public boolean insert(Object toInsert) {

if(!(toInsert instanceof Pokemon))

return false;

MongoCollection<Document> collection = getCollection(collectionName); //also opens connection

Document doc = PokemonToDocument((Pokemon)toInsert);

Logger.*vvlog*("ADDED " + doc.toJson());

collection.insertOne(doc);

closeConnection();

return true;

}

The *teamManager.addPokemon(…)* implemented in *TeamManagerOnNeo4j.java*:

public boolean addPokemon(Pokemon p) throws DuplicatePokemonException{

if(pokemonAlreadyExist(p))

throw new DuplicatePokemonException();

String query = "MERGE (b:Pokemon { name: $name, type: [" + (p.getTypesSingleStringForCipher()) +

"],sprite: $sprite, capture\_rate: $capture\_rate})";

return insert(query, *parameters*("name", p.getName(), "sprite", p.getSprite(), "capture\_rate", p.getCapture\_rate()));

}

* Retrieve **User** information at login time

This is done querying MongoDb, in *UserManagerOnMongoDb.java*:

public User login(User toLog) {

return login(toLog.getUsername(), toLog.getPassword());

}

public User login(String username, String password) {

Bson query = *and*(*eq*("username", username), *eq*("password", PasswordEncryptor.*encryptPassword*(password)));

ArrayList<Object> matched = getWithFilter(query);

if(matched.size()!=1)

return null;

User logger = (User)matched.get(0);

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", Locale.*US*);

Date now = new Date();

// Compare dates and update pokeball in case

Date oldLastLogin = logger.getLastLogin();

Calendar oldCal = Calendar.*getInstance*();

oldCal.setTime(oldLastLogin);

oldCal.add(Calendar.*DATE*, 1);

oldCal.set(Calendar.*HOUR\_OF\_DAY*, 0);

oldCal.set(Calendar.*MINUTE*, 0);

oldCal.set(Calendar.*MILLISECOND*, 0);

Calendar newCal = Calendar.*getInstance*();

newCal.setTime(now);

if(newCal.after(oldCal)) {

updateNumberOfPokeballTo10(logger);

logger.resetDailyPokeball();

}

// End

String dateString = sdf.format(now);

Logger.*vvlog*("Updated last login for " + username + ": " + dateString);

update(query, *set*("lastLogin", dateString));

return logger;

}

public ArrayList<Object> getWithFilter(Object filter) {

if(!(filter instanceof Bson))

return null;

List<Document> docs= getCollection(collectionName).find((Bson)filter).into(new ArrayList<>());

ArrayList<Object> users = new ArrayList<>();

for(Document d:docs){

users.add(DocumentToUser(d));

}

closeConnection();

return users;

}

* Retrieve **User** by *username*

This is done using the *getWithFilter(Object filter)* function, passing a Bson in which we ask to retrieve the user information related to a specific username (*eq(“username”, username)*, where username is the String username we want to use).

* Retrieve **Pokémon** information using several filters

This is done querying MongoDb, in *PokemonManagerOnMongoDb*:

public ArrayList<Pokemon> searchWithFilter(Map<Filter, String> parameters) {

ArrayList<Filter> keys = new ArrayList<>(parameters.keySet());

Bson query = queryBuilder(parameters);

Logger.*vvlog*("SEARCH WITH FILTER query: " + query);

ArrayList<Pokemon> result = new ArrayList<>();

ArrayList<Object> matched = getWithFilter(query);

for(Object o:matched)

result.add((Pokemon)o);

return result;

}

private Bson queryBuilder(Map<Filter, String> params){

Document toReturn=new Document();

if(params.get(*NAME*)!=null)

toReturn.append("name", params.get(*NAME*));

if(params.get(*POKEDEX\_ID*)!=null)

toReturn.append("id", Integer.*parseInt*(params.get(*POKEDEX\_ID*)));

if(params.get(*MIN\_WEIGHT*)!=null && params.get(*MAX\_WEIGHT*)!=null)

toReturn.append("weight", new Document("$gte", Integer.*parseInt*(params.get(*MIN\_WEIGHT*))).append("$lte", Integer.*parseInt*(params.get(*MAX\_WEIGHT*))));

else if(params.get(*MIN\_WEIGHT*)!=null)

toReturn.append("weight", new Document("$gte", Integer.*parseInt*(params.get(*MIN\_WEIGHT*))));

else if(params.get(*MAX\_WEIGHT*)!=null)

toReturn.append("weight", new Document("$lte", Integer.*parseInt*(params.get(*MAX\_WEIGHT*))));

if(params.get(*MIN\_HEIGHT*)!=null && params.get(*MAX\_HEIGHT*)!=null)

toReturn.append("height", new Document("$gte", Integer.*parseInt*(params.get(*MIN\_HEIGHT*))).append("$lte", Integer.*parseInt*(params.get(*MAX\_HEIGHT*))));

else if(params.get(*MIN\_HEIGHT*)!=null)

toReturn.append("height", new Document("$gte", Integer.*parseInt*(params.get(*MIN\_HEIGHT*))));

else if(params.get(*MAX\_HEIGHT*)!=null)

toReturn.append("height", new Document("$lte", Integer.*parseInt*(params.get(*MAX\_HEIGHT*))));

if(params.get(*TYPE1*)!=null)

toReturn.append("types", params.get(*TYPE1*));

if(params.get(*TYPE2*)!=null)

toReturn.append("types", params.get(*TYPE2*));

if(params.get(*MIN\_CATCH\_RATE*)!=null && params.get(*MAX\_CATCH\_RATE*)!=null)

toReturn.append("capture\_rate", new Document("$gte", Integer.*parseInt*(params.get(*MIN\_CATCH\_RATE*))).append("$lte", Integer.*parseInt*(params.get(*MAX\_CATCH\_RATE*))));

else if(params.get(*MIN\_CATCH\_RATE*)!=null)

toReturn.append("capture\_rate", new Document("$gte", Integer.*parseInt*(params.get(*MIN\_CATCH\_RATE*))));

else if(params.get(*MAX\_CATCH\_RATE*)!=null)

toReturn.append("capture\_rate", new Document("$lte", Integer.*parseInt*(params.get(*MAX\_CATCH\_RATE*))));

Logger.*vlog*("Query built: " + toReturn.toJson());

return toReturn;

}

* Retrieve a **Pokémon** by *name* when trying to catch it

This is done using s*earchWithFilter(hashMap)*, where hashMap is initialized as:

HashMap<Filter, String> hashMap = new HashMap<>();

hashMap.put(Filter.NAME, pokemonName);

* Modify **User** settings (email, password, country)

In MondoDb:

@Override

public boolean changeEmail(User target, String newEmail) {

Logger.*vvlog*("Trying to update " + target.getUsername() + " email: " + newEmail);

return update(target, *set*("email", newEmail));

}

@Override

public boolean changePassword(User target, String newPassword) {

Logger.*vvlog*("Trying to update " + target.getUsername() + " password");

return update(target, *set*("password", PasswordEncryptor.*encryptPassword*(newPassword)));

}

@Override

public boolean changeCountry(User target, String newCountry) {

Logger.*vvlog*("Trying to update " + target.getUsername() + " country: " + newCountry);

return update(target, *set*("country", newCountry));

}

In Neo4j we change only the country (*UserNetworkManagerOnNeo4j.java*):

public boolean updateCountry(User target, String newCountry) {

String query = "MATCH (n:User) WHERE n.username = $username " +

"SET n.country = $country";

return update(query, *parameters*("username", target.getUsername(), "country", newCountry));

}

public boolean update(Object target, Object newValue) {

if(!(newValue instanceof Bson))

return false;

MongoCollection<Document> collection = getCollection(collectionName);

UpdateResult ur;

if (target instanceof User){

ur = collection.updateOne(*eq*("username", ((User) target).getUsername()),(Bson)newValue);

}

else if(target instanceof Bson){

ur = collection.updateMany((Bson)target, (Bson)newValue);

}

else {

closeConnection();

return false;

}

closeConnection();

Logger.*vvlog*("UPDATED " + ur.getModifiedCount() + " user");

return ur.getModifiedCount()>0;

}

* Update **Team**’s *name*

Done in UserManagerOnMongoDb.java:

public boolean changeTeamName(User involved, String newName) {

Logger.*vvlog*("Trying to change " + involved.getUsername() + "'s team name: " + newName);

return update(involved, *set*("teamName", newName));

}

* Update **Team**’s *points*

Done in UserManagerOnMongoDb.java:

public boolean updatePoints(User target, double points) {

Logger.*vvlog*("Trying to update points of " + target.getUsername() + ": " + points);

return update(target, *set*("points", points));

}

* Update **Pokémon**’s *catch rates*

Done in the *PokeMongoAnalyticServer, in the TeamManagerOnNeo4j.java:*

public void updateCatchRateOfPokemon(List<PokemonAndCatchRate> catch\_rates){

String start = "[";

for(int i=0; i<catch\_rates.size(); i++){

start += "{name:\"" + catch\_rates.get(i).name + "\"," + "catchRate:" + catch\_rates.get(i).catchRate + "}";

if(i < catch\_rates.size() - 1)

start += ",";

}

start += "]";

String query = "UNWIND " + start + "as catch " +

"MATCH (p:Pokemon{name: catch.name}) SET p.capture\_rate = catch.catchRate";

update(query, *parameters*());

}

* Remove a **User** (*admin* only)

This operation is done both in MongoDb and in Neo4j.

For MongoDb we use a method in *UserManagerOnMongoDb.java*:

@Override

public boolean removeUser(User target) {

Logger.*vvlog*("Trying to remove " + target.getUsername());

if(target.isAdmin())

return false;

return remove(target);

}

@Override

public boolean removeUser(String username) {

Logger.*vvlog*("Trying to remove " + username);

Bson query = *eq*("username", username);

ArrayList<Object> target = getWithFilter(query);

if(target.size()!=1 || ((User)(target.get(0))).isAdmin())

return false;

return remove(query);

}

public boolean remove(Object o) {

MongoCollection<Document> collection = getCollection(collectionName);

DeleteResult dr;

if (o instanceof User){

dr = collection.deleteOne(*eq*("username", ((User) o).getUsername()));

}

else if(o instanceof Bson){

dr = collection.deleteMany((Bson)o);

}

else {

closeConnection();

return false;

}

closeConnection();

Logger.*vvlog*("DELETED " + dr.getDeletedCount() + " pokemon");

return dr.getDeletedCount()>0;

}

For Neo4j we use a method in *PokemonManagerOnMongoDb.java*:

@Override

public boolean deleteUser(User u){

return deleteUser(u.getUsername());

}

@Override

public boolean deleteUser(String username){

String query = "MATCH (u:User) WHERE u.username = $username OPTIONAL MATCH (u)-[:CREATED]->(p:Post) " +

"OPTIONAL MATCH (p)<-[:TOPIC]-(p1:Post)" +

"DETACH DELETE u, p, p1";

return remove(query, *parameters*("username", username));

}

Where the *remove* method, in this case, is implemented in Neo4jDbDatabase.java:

public boolean remove(Object query){

if(!(query instanceof String))

return false;

getConnection();

try (Session session = driver.session()) {

session.writeTransaction((TransactionWork<Void>) tx -> {

tx.run((String)query);

return null;

});

}

closeConnection();

return true;

}

* Remove a **Pokémon** (*admin* only)

This operation is done both in MongoDb and in Neo4j.

For MongoDb we use a method in *PokemonManagerOnMongoDb.java*:

public boolean remove(Object o) {

MongoCollection<Document> collection = getCollection(collectionName);

DeleteResult dr;

if (o instanceof Pokemon){

dr = collection.deleteOne(*eq*("id", ((Pokemon) o).getId()));

}

else if(o instanceof Bson){

dr = collection.deleteMany((Bson)o);

}

else {

closeConnection();

return false;

}

closeConnection();

Logger.*vvlog*("DELETED " + dr.getDeletedCount() + " pokemon");

return dr.getDeletedCount()>0;

}

public boolean removePokemon(Pokemon toRemove) {

return remove(toRemove);

}

public boolean removePokemon(String name) {

return remove(*eq*("name", name));

}

For Neo4j we use a method in *TeamManagerOnNeo4j*:

@Override

public boolean deletePokemon(String pokemonName){

String query = "MATCH (p:Pokemon) WHERE p.name = $name " +

"OPTIONAL MATCH (p:Pokemon)<-[:TOPIC]-(pp:Post)<-[:TOPIC]-(pp1:Post)" +

"DETACH DELETE p,pp,pp1";

return remove(query, *parameters*("name", pokemonName));

}

Where the *remove* method is the one shown above in the *Neo4jDbDatabase.java* file.