UNIVERSITÀ DEGLI STUDI DI FIRENZE

SWAM EXAM

Development of a Backend based on a GraphDBMS for an App

Candidate: Lorenzo Macchiarini

Academic Year 2021/2022

Topics covered:

- How to represent data for a social app
- Approaches of interfacing with a GraphDB
- How architecture changes wrt a GraphDB

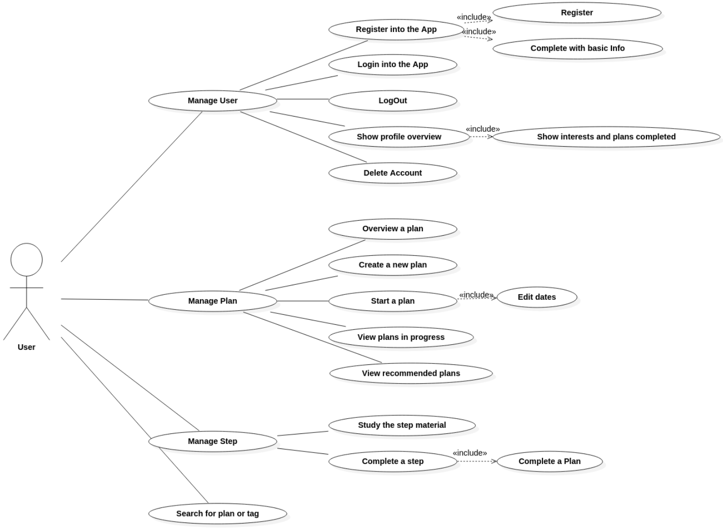
Why Litto

- Help user to reach their goals in a organized and timed way and to discover new interests
- Main idea similar to Coursera or Udemy but Litto allows users to create their goals

Litto
Learn It Together

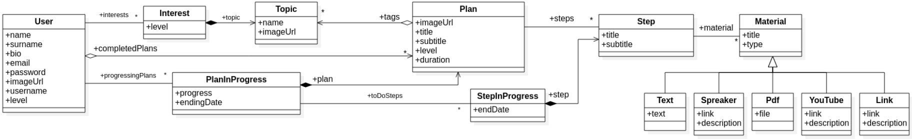
Use Case definition

Made interviews to know which are the main use cases of the app



Domain Model definition

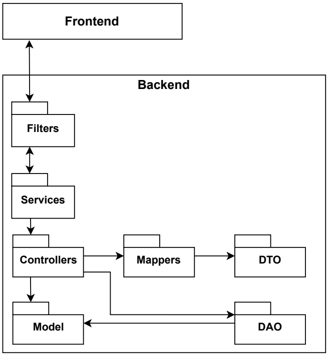
- Domain Model based on use cases
- Composition preferred over inheritance
- Anemic Domain Model looking forward to the real implementation



Main Architecture Structure

Based on REST paradigm and MVC

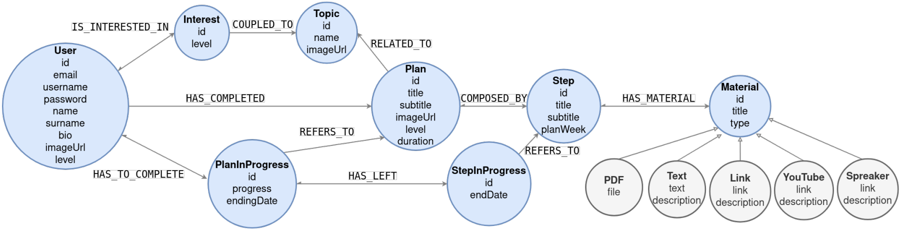
- frontend exposes the UI and handles the user inputs
- backend manages the Use Cases and the Model, using a GraphDBMS
- stateless requests



Graph Data Base Neo4j

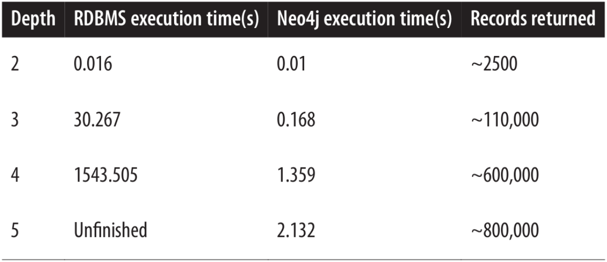
Based on the **Labeled Property Graph** concept

- represents Nodes and Relationship
- Nodes contain key-value properties
- Relationship have a name, a direction, a starting and ending Node



Graph Data Base Neo4j vs RDBMS

- **Native Graph Storage**: Data stored as graph structure
- **Index Free Adjacency**: Each Node has its own index that links all the Nodes in relationship with it
- **Traversing** the graph is faster than executing Joins of a RDBMS

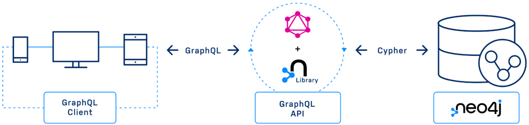


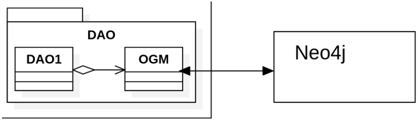
Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

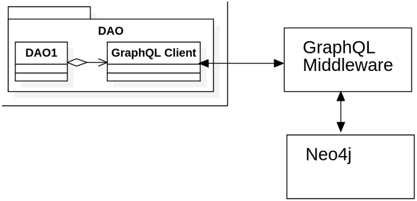
Test with graph with 1 mln users, each with 50 friends.
Retrieving all the friends at a depth from 2 to 5

Neo4j OGM vs GraphQL

- **Object Graph Mapper** similar to the ORM: persists Java Objects to the DB into Nodes and Relationships
- **GraphQL** specification of query language based on Graph Structure. Generalized using a middleware.

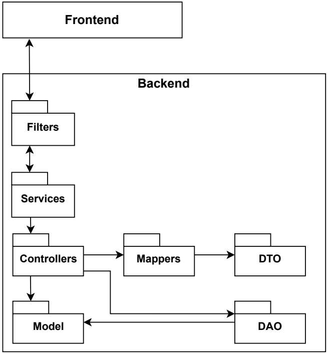






Overview

- JEE integrated in a WildFly server using Jersey.
- Different projects for the OGM and GraphQL implementation: mainly different DAO, Controllers and Model
- CDI Beans used when needed, always ApplicationScoped



Filters

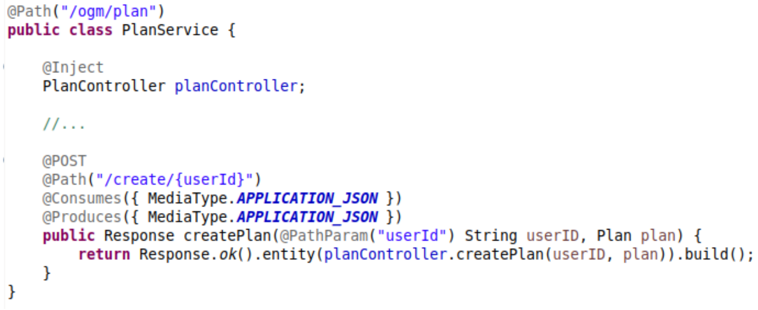
Handle the server’s requests and responses before they arrive to Services and when they leave them

- **ServiceRequestFilter**: recieves the Request from client and executes the authentication using a JWT Token
- **ServiceResponseFilter**: sends the Response to the client and adds the correct headers to handle CORS

Services

Exposing Services of the server using Jax-RS

- One service per Use Case
- Uses the Jax-RS annotations: @Path, @GET, @Consumes, ecc
- Uses Jackson to convert the Requests body to Java Objects
- Uses Controllers as Beans to invke the correct methods



```
@Path("/ogm/plan")
public class PlanService {

    @Inject
    PlanController planController;

    //...

    @POST
    @Path("/create/{userId}")
    @Consumes({ MediaType.APPLICATION_JSON })
    @Produces({ MediaType.APPLICATION_JSON })
    public Response createPlan(@PathParam("userId") String userID, Plan plan) {
        return Response.ok().entity(planController.createPlan(userID, plan)).build();
    }
}
```

Model

Implemented the Anemic Class Diagram shown

- Controllers will implement the Business Logic, no login in Model

Different implementations depending on the technology

GraphQL:

- plain Java classes
- ID autogenerated by the Middleware, in Model only for User and Plan

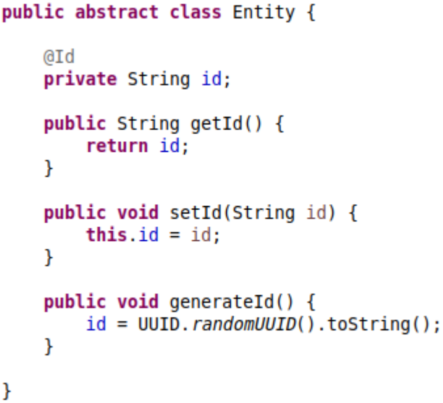
Neo4j OGM:

- plain Java classes with OGM annotations to map Entities into DB
- ID generated by the Java Server and required by the OGM

Relationships as *@OneToMany* in JPA but no need for Joins
Directions IN are like *@MappedBy* in JPA

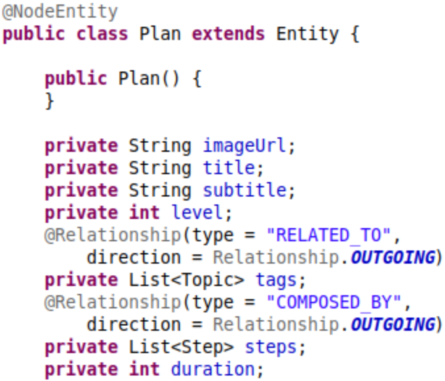
Model

How ID is integrated in Model: inheriting from Entity



```java
public abstract class Entity {

    @Id
    private String id;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public void generateId() {
        id = UUID.randomUUID().toString();
    }

}
```

Example of using NodeEntity and Relationships in OGM Model



```java
@NodeEntity
public class Plan extends Entity {

    public Plan() {
    }

    private String imageUrl;
    private String title;
    private String subtitle;
    private int level;
    @Relationship(type = "RELATED_TO",
        direction = Relationship.OUTGOING)
    private List<Topic> tags;
    @Relationship(type = "COMPOSED_BY",
        direction = Relationship.OUTGOING)
    private List<Step> steps;
    private int duration;
```

GraphQL DAO

- Created a Middleware with Neo4jGraphQL JS library and ApolloServer
- Defined the GraphQL Schema related to Neo4j DB

```
const typeDefs = ‘
  type User {
    id: ID @id
    name: String
    surname: String
    bio: String
    email: String @unique
    password: String
    imageUrl: String
    username: String
    level: Int
    interests: [Interest] @relationship(type: "IS_INTERESTED_IN",
        direction: OUT)
    completedPlans: [Plan] @relationship(type: "HAS_COMPLETED",
        direction: OUT)
    progressingPlans: [PlanInProgress] @relationship(type:
        "HAS_TO_COMPLETE", direction: OUT)
  }
‘;
```

GraphQL DAO, intefacing with Middleware

- Creating queries and mutations from the schema
- Defining precise structure for queries and mutations (entry point, arguments and selection set)
- Middleware translates queries in Cypher, execute against Neo4j
- JSON as response body

GraphQL DAO, Choosing Client

Nodes by American Express

- Annotating POJOs of the Domain Model (@GraphQLArgument, ecc)
- Creating queries setting the arguments step by step

Nodes query

```
query {
  plans(id:"0f8d16ba-1d65-4419-81a0-a9ecf7415d59") {
    title
    subtitle
    imageUrl
    level
  }
}
```

Correct query

```
query {
  plans(where:{id:"0f8d16ba-1d65-4419-81a0-a9ecf7415d59"}) {
    title
    subtitle
    imageUrl
    level
  }
}
```

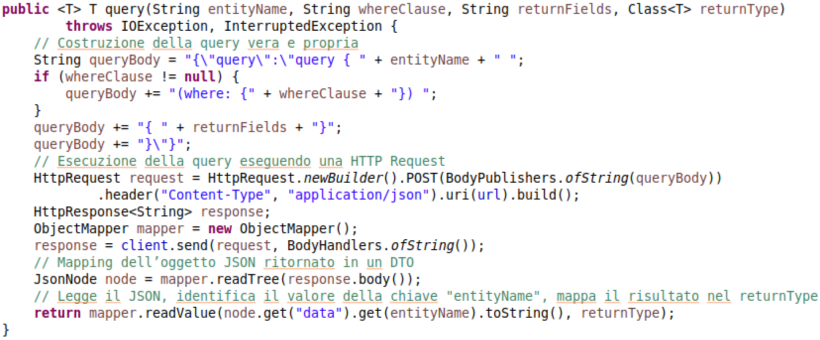
```
const resolvers = {
  Query: {
    plan(parent, args, context, info) {
      return plan.find(plan => plan.id === plan.id);
    }
  }
}
```

GraphQL DAO, Choosing Client

Created a custom **GraphQL Client**

- Using HTTP requests with the query string as body
- Jackson to map JSON responses into DTOs
- Methods available: **query**, **create**, **update**, **customQuery**

Query creation based on string concatenation over defined structure



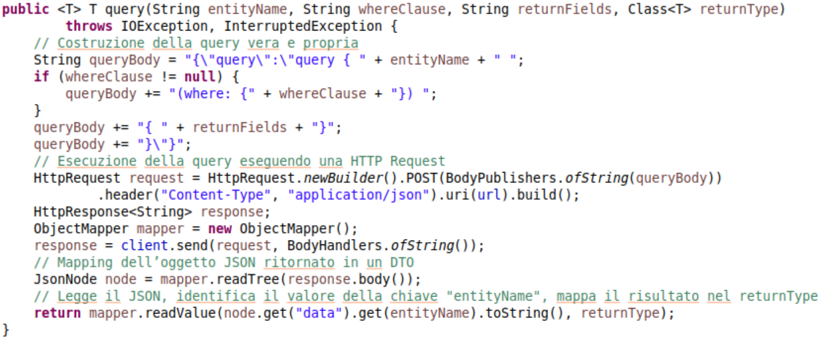
```
public <T> T query(String entityName, String whereClause, String returnFields, Class<T> returnType)
        throws IOException, InterruptedException {
    // Costruzione della query vera e propria
    String queryBody = "{\"query\":\"query { " + entityName + " ";
    if (whereClause != null) {
        queryBody += "(where: {" + whereClause + "}) ";
    }
    queryBody += "{ " + returnFields + "}";
    queryBody += "}\"}";
    // Esecuzione della query eseguendo una HTTP Request
    HttpRequest request = HttpRequest.newBuilder().POST(BodyPublishers.ofString(queryBody))
            .header("Content-Type", "application/json").uri(url).build();
    HttpResponse<String> response;
    ObjectMapper mapper = new ObjectMapper();
    response = client.send(request, BodyHandlers.ofString());
    // Mapping dell'oggetto JSON ritornato in un DTO
    JsonNode node = mapper.readTree(response.body());
    // Legge il JSON, identifica il valore della chiave "entityName", mappa il risultato nel returnType
    return mapper.readValue(node.get("data").get(entityName).toString(), returnType);
}
```

GraphQL DAO, Choosing Client

Created a custom **GraphQL Client**

- Using HTTP requests with the query string as body
- Jackson to map JSON responses into DTOs
- Methods available: **query**, **create**, **update**, **customQuery**

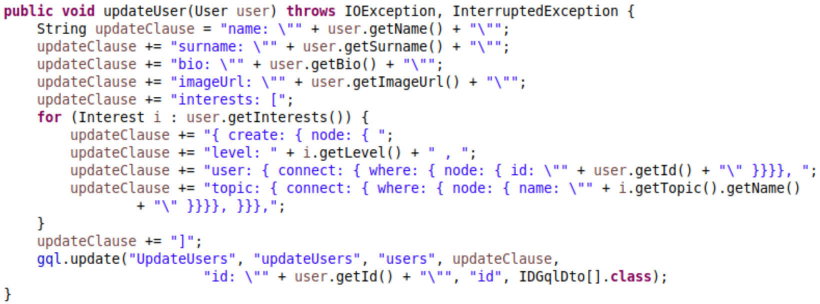
Query creation based on string concatenation over defined structure



```java
public <T> T query(String entityName, String whereClause, String returnFields, Class<T> returnType)
        throws IOException, InterruptedException {
    // Costruzione della query vera e propria
    String queryBody = "{\"query\":\"query { " + entityName + " ";
    if (whereClause != null) {
        queryBody += "(where: {" + whereClause + "}) ";
    }
    queryBody += "{ " + returnFields + "}";
    queryBody += "}\"}";
    // Esecuzione della query eseguendo una HTTP Request
    HttpRequest request = HttpRequest.newBuilder().POST(BodyPublishers.ofString(queryBody))
            .header("Content-Type", "application/json").uri(url).build();
    HttpResponse<String> response;
    ObjectMapper mapper = new ObjectMapper();
    response = client.send(request, BodyHandlers.ofString());
    // Mapping dell'oggetto JSON ritornato in un DTO
    JsonNode node = mapper.readTree(response.body());
    // Legge il JSON, identifica il valore della chiave "entityName", mappa il risultato nel returnType
    return mapper.readValue(node.get("data").get(entityName).toString(), returnType);
}
```

GraphQL DAO

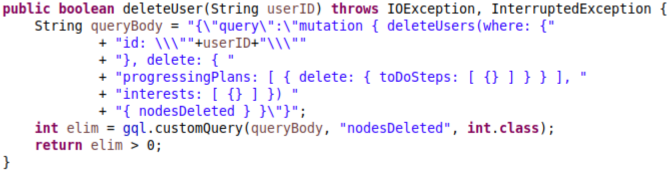
- One DAO for each class to be persisted autonomously
- GraphQL Client as a Bean in each DAO
- Manual mapping to create the query string



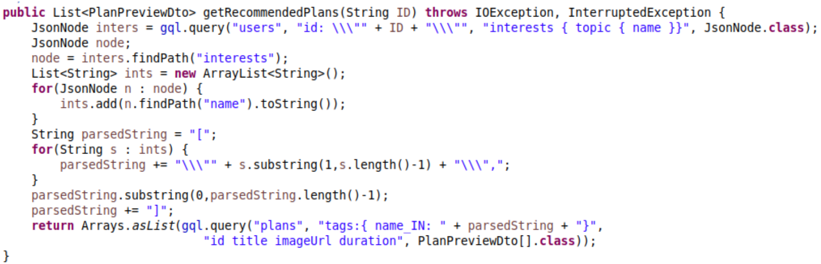
```java
public void updateUser(User user) throws IOException, InterruptedException {
    String updateClause = "name: \"" + user.getName() + "\"";
    updateClause += "surname: \"" + user.getSurname() + "\"";
    updateClause += "bio: \"" + user.getBio() + "\"";
    updateClause += "imageUrl: \"" + user.getImageUrl() + "\"";
    updateClause += "interests: [";
    for (Interest i : user.getInterests()) {
        updateClause += "{ create: { node: { ";
        updateClause += "level: " + i.getLevel() + " , ";
        updateClause += "user: { connect: { where: { node: { id: \"" + user.getId() + "\" }}}}, ";
        updateClause += "topic: { connect: { where: { node: { name: \"" + i.getTopic().getName()
                + "\" }}}}, }}},";
    }
    updateClause += "]";
    gql.update("UpdateUsers", "updateUsers", "users", updateClause,
                        "id: \"" + user.getId() + "\"", "id", IDGqlDto[].class);
}
```

GraphQL DAO

- Creating customQueries means no structure from GraphQL Client
- Allows delete that have no cascading
- Custom parsing of object with JsonNode



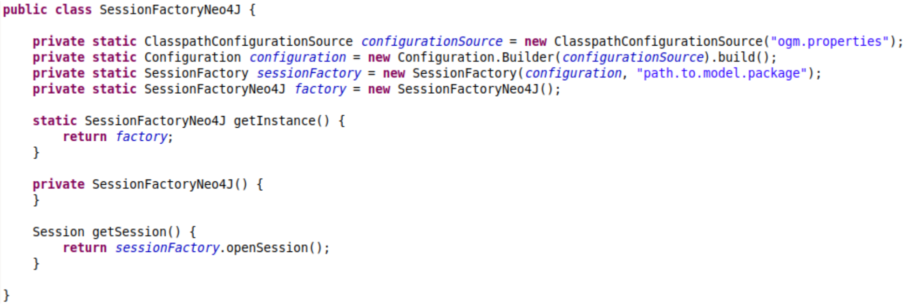
```java
public boolean deleteUser(String userID) throws IOException, InterruptedException {
    String queryBody = "{\"query\":\"mutation { deleteUsers(where: {"
            + "id: \\\""+userID+"\\\""
            + "}, delete: { "
            + "progressingPlans: [ { delete: { toDoSteps: [ {} ] } } ], "
            + "interests: [ {} ] }) "
            + "{ nodesDeleted } }\"}";
    int elim = gql.customQuery(queryBody, "nodesDeleted", int.class);
    return elim > 0;
}
```



```java
public List<PlanPreviewDto> getRecommendedPlans(String ID) throws IOException, InterruptedException {
    JsonNode inters = gql.query("users", "id: \\\"" + ID + "\\\"", "interests { topic { name }}", JsonNode.class);
    JsonNode node;
    node = inters.findPath("interests");
    List<String> ints = new ArrayList<String>();
    for(JsonNode n : node) {
        ints.add(n.findPath("name").toString());
    }
    String parsedString = "[";
    for(String s : ints) {
        parsedString += "\\\"" + s.substring(1,s.length()-1) + "\\\",";
    }
    parsedString.substring(0,parsedString.length()-1);
    parsedString += "]";
    return Arrays.asList(gql.query("plans", "tags:{ name_IN: " + parsedString + "}",
                        "id title imageUrl duration", PlanPreviewDto[].class));
}
```

Neo4j OGM DAO

- Uses **Session** to run queries to DB
- Sessions generated by a SessionFactory as a Singleton
- SessionFactory configured once to reach DB



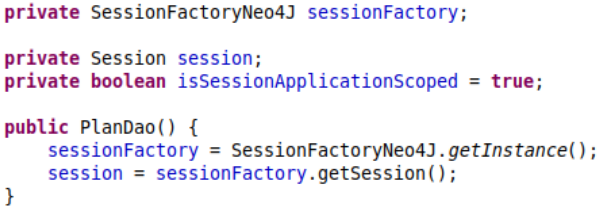
```java
public class SessionFactoryNeo4J {

    private static ClasspathConfigurationSource configurationSource = new ClasspathConfigurationSource("ogm.properties");
    private static Configuration configuration = new Configuration.Builder(configurationSource).build();
    private static SessionFactory sessionFactory = new SessionFactory(configuration, "path.to.model.package");
    private static SessionFactoryNeo4J factory = new SessionFactoryNeo4J();

    static SessionFactoryNeo4J getInstance() {
        return factory;
    }

    private SessionFactoryNeo4J() {
    }

    Session getSession() {
        return sessionFactory.openSession();
    }

}
```

Neo4j OGM DAO

Session keeps a cache with entities persisted by her

- Can be fully used if Session scope sufficently wide
- If too wide the cache will be big and data not coherent with DB
- If too small there is no cache

Using *isSessionApplicationScoped* to distinguish between the two scopes



```
private SessionFactoryNeo4J sessionFactory;

private Session session;
private boolean isSessionApplicationScoped = true;

public PlanDao() {
    sessionFactory = SessionFactoryNeo4J.getInstance();
    session = sessionFactory.getSession();
}
```

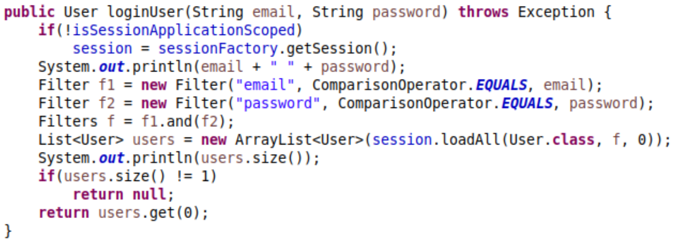
Queries in Neo4j OGM DAO

Session can execute queries (CRUD operations) on DB and persist with:

- **save**, **load**, **loadAll**, **delete**

Using *depth* to retrive Nodes in relationship with a given Node within the given depth

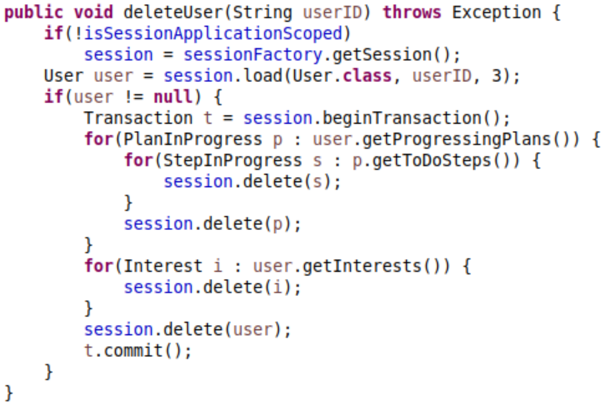
- Obtaining part of a Node, like DTO
- Saving objects allows to save its relationships (not in delete)
- Using *Traversing* in controllers



```
public User loginUser(String email, String password) throws Exception {
    if(!isSessionApplicationScoped)
        session = sessionFactory.getSession();
    System.out.println(email + " " + password);
    Filter f1 = new Filter("email", ComparisonOperator.EQUALS, email);
    Filter f2 = new Filter("password", ComparisonOperator.EQUALS, password);
    Filters f = f1.and(f2);
    List<User> users = new ArrayList<User>(session.loadAll(User.class, f, 0));
    System.out.println(users.size());
    if(users.size() != 1)
        return null;
    return users.get(0);
}
```

Transactions in Neo4j OGM DAO

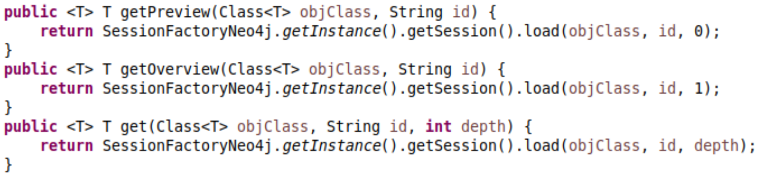
- Transactions are autocommitted on Session operations
- Sometimes need to explicitate it



```java
public void deleteUser(String userID) throws Exception {
    if(!isSessionApplicationScoped)
        session = sessionFactory.getSession();
    User user = session.load(User.class, userID, 3);
    if(user != null) {
        Transaction t = session.beginTransaction();
        for(PlanInProgress p : user.getProgressingPlans()) {
            for(StepInProgress s : p.getToDoSteps()) {
                session.delete(s);
            }
            session.delete(p);
        }
        for(Interest i : user.getInterests()) {
            session.delete(i);
        }
        session.delete(user);
        t.commit();
    }
}
```

Generic DAO

- Simple queries could be generalized in a Generic DAO
- DAO exposing simple CRUD function at different level of depth
- Generalizes the operation using Generics
- Predefined depth levels



```
public <T> T getPreview(Class<T> objClass, String id) {
    return SessionFactoryNeo4j.getInstance().getSession().load(objClass, id, 0);
}
public <T> T getOverview(Class<T> objClass, String id) {
    return SessionFactoryNeo4j.getInstance().getSession().load(objClass, id, 1);
}
public <T> T get(Class<T> objClass, String id, int depth) {
    return SessionFactoryNeo4j.getInstance().getSession().load(objClass, id, depth);
}
```

- Controllers have more responsibility
- Too much uniformity means no clear responsibility partition

Controllers

Different controllers for different DAO:

GraphQL

- No mapping operations since DAOs can retrive needed DTOs
- Less responsibility

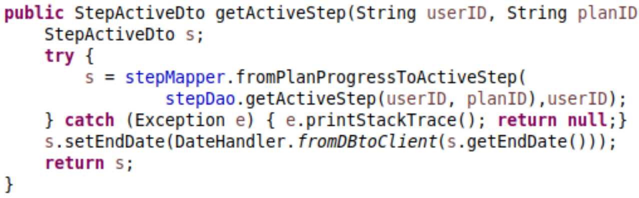
Neo4j OGM

- ID generation for each persisted object
- Mapping responsibility
- Has to do *Traversing* to obtain the needed object
- More responsibility

Both have to:

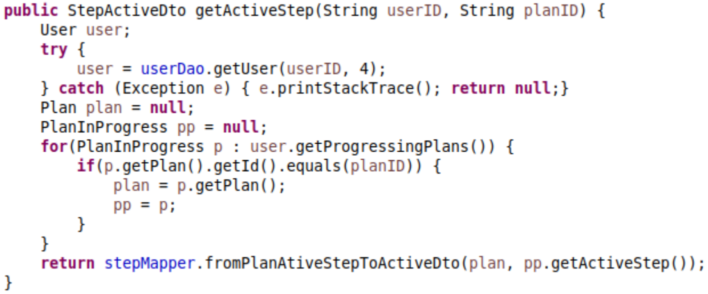
- Handle JWT Token creation
- Handle exceptions thrown by DAOs

Controllers



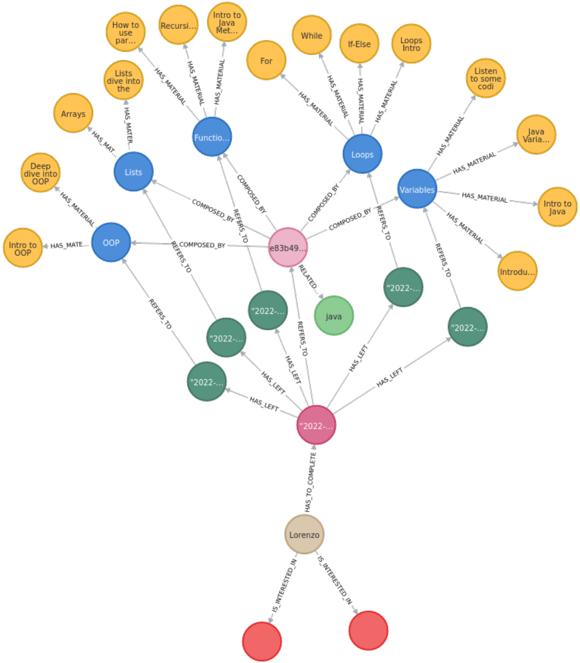
```
public StepActiveDto getActiveStep(String userID, String planID) {
    StepActiveDto s;
    try {
        s = stepMapper.fromPlanProgressToActiveStep(
                stepDao.getActiveStep(userID, planID),userID);
    } catch (Exception e) { e.printStackTrace(); return null;}
    s.setEndDate(DateHandler.fromDBtoClient(s.getEndDate()));
    return s;
}
```

GraphQL



```
public StepActiveDto getActiveStep(String userID, String planID) {
    User user;
    try {
        user = userDao.getUser(userID, 4);
    } catch (Exception e) { e.printStackTrace(); return null;}
    Plan plan = null;
    PlanInProgress pp = null;
    for(PlanInProgress p : user.getProgressingPlans()) {
        if(p.getPlan().getId().equals(planID)) {
            plan = p.getPlan();
            pp = p;
        }
    }
    return stepMapper.fromPlanAtiveStepToActiveDto(plan, pp.getActiveStep());
}
```

Neo4j OGM


How to use par...
Recursi...
Intro to Java Met...
For
While
If-Else
Loops Intro
Listen to some codi
Lists dive into the
Arrays
Functio...
Loops
Java Varia...
Deep dive into OOP
Lists
Variables
Intro to Java
Intro to OOP
OOP
e83b49...
Introdu...
"2022-...
"2022-...
java
"2022-...
"2022-...
"2022-...
"2022-...
Lorenzo
HAS_MATERIAL
HAS_MATERIAL
HAS_MATERIAL
HAS_MATERIAL
HAS_MATERIAL
HAS_MATERIAL
HAS_MATERIAL
HAS_MATER...
HAS_MAT...
HAS_MATERIAL
HAS_MATERIAL
HAS_MATERIAL
HAS_MATERIAL
HAS_MATERIAL
HAS_MATE...
COMPOSED_BY
COMPOSED_BY
COMPOSED_BY
COMPOSED_BY
COMPOSED_BY
REFERS_TO
REFERS_TO
REFERS_TO
REFERS_TO
REFERS_TO
REFERS_TO
RELATED...
HAS_LEFT
HAS_LEFT
HAS_LEFT
HAS_LEFT
HAS_LEFT
HAS_TO_COMPLETE
IS_INTERESTED_IN
IS_INTERESTED_IN

Comparison Neo4j OGM - GraphQL

Neo4j OGM

- + easy and plain Java queries
- + direct interface with DB
- + caching
- - possible data incoherence
- - retrieved unnecessary data

GraphQL

- + retrieved only necessary data
- + queries unaware of used DB
- - middleware slows times
- - no Java client
- - no caching

Expectations:

- GraphQL responses lighter than Neo4j OGM ones
- Neo4j OGM responses faster than GraphQL ones, faster using cache

Comparison Neo4j OGM - GraphQL

Query Execution Times (ms)

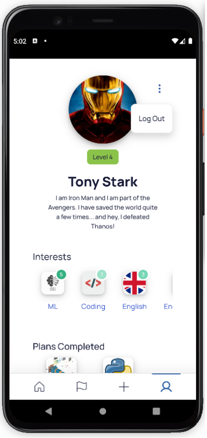
HTTP Request	**OGM Session req.**	**OGM Session app.**	**GraphQL**
GET /user/id	106	108	201
GET /plan/id	100	109	191

Query Response Bytes

HTTP Request	**OGM**	**GraphQL**
GET /user/id	787	424
GET /plan/id	1216	696

Frontend Implementation

- Angular + Taiga UI + Ionic
- State in frontend: localStorage keeps userID and Token
- Services to send requests to server (has an Interceptor to handle CORS)
- Components to handle the View and Controller responsibilities
- Angular Model as DTOs

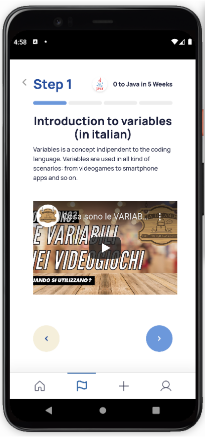


Tony Stark

I am Iron Man and I am part of the Avengers. I have saved the world quite a few times... and hey, I defeated Thanos!

Interests

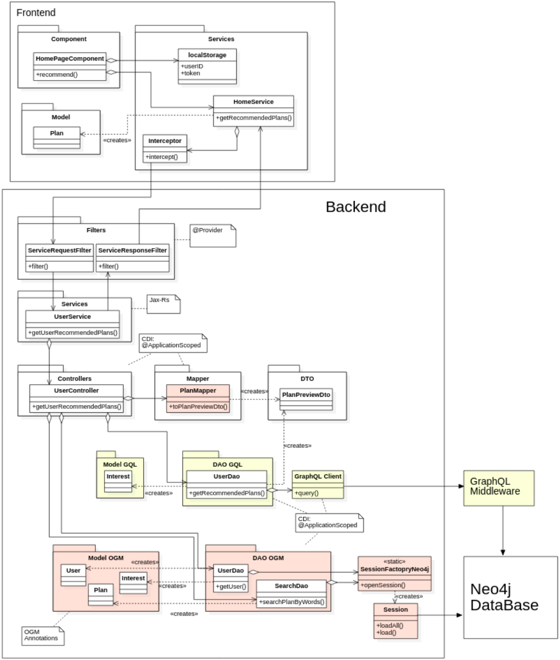
Plans Completed



Introduction to variables (in italian)

Variables is a concept indipendent to the coding language. Variables are used in all kind of scenarios: from videogames to smartphone apps and so on.

Complete Architecture



Conclusions

UseCases fully implemented by the backend instead of the frontend

Future Works:

- Native GraphQL backend implemented in middleware
- Cypher queries in OGM to retrieve less data
- More Use Cases of the app

Thank you for your attention

Litto

Learn It Together