MongoDb

Librairies

- gson
- Mongo-java-driver

Sérialisation/dé-sérialisation JSON

Gson va permettre de sérialiser et désérialiser pour passer d'objet java à JSON et inversement :

```
public class DBOAdapter {
    private static Gson gson;

    private static Gson getGson() {
        if (gson == null) {
            GsonBuilder gsonBuilder = new GsonBuilder();
            gsonBuilder.registerTypeAdapter(ObjectId.class, new ObjectIdAdapter());
            gsonBuilder.excludeFieldsWithoutExposeAnnotation().setDateFormat("MMMM
dd, yyyy HH:mm:ss");
            gson = gsonBuilder.create();
        }
        return gson;
    }

    public static <T extends Model> T dboToModel(DBObject dbObject, Class<T> clazz)
{
        gson = getGson();
        String json = gson.toJson(dbObject);
        return gson.fromJson(json, clazz);
    }

    public static BasicDBObject objectToDBObject(Object object) {
        BasicDBObject obj = (BasicDBObject) JSON.parse(getGson().toJson(object));
        return obj;
    }

    public static DBObject[] objectToDBObjectArray(Object object) {
        BasicDBObject[] objects = new BasicDBObject[] { objectToDBObject(object) };
        return objects;
    }
}
```

Gestion de l'objectId mongoDB

```
public class ObjectIdAdapter
        implements JsonSerializer<ObjectId>, JsonDeserializer<ObjectId> {
```

```
    @Override
    public JsonElement serialize(ObjectId id, Type typeOfT,
JsonSerializationContext context) {
        JsonObject jo = new JsonObject();
        jo.addProperty("$oid", id.toHexString());
        return jo;
    }

    @Override
    public ObjectId deserialize(JsonElement json, Type typeOfT,
JsonDeserializationContext context) throws JsonParseException {
        try {
            return new ObjectId(json.getAsJsonObject().get("$oid").getAsString());
        } catch (Exception e) {
            return null;
        }
    }

}
```

Opérations CRUD

Classe à intégrer, à documenter et à tester (Tests unitaires)

```
public class MyMongo {
    protected DB db;
    protected MongoClient mongoClient;
    private DBCollection collection;

    public DB getDb() {
        return db;
    }

    public boolean connect(String dbname) throws UnknownHostException {
        return this.connect(dbname, "127.0.0.1", 27017);
    }

    public boolean connect(String dbname, String server, int port) throws
UnknownHostException {
        mongoClient = new MongoClient(server, port);
        List<String> dbnames = mongoClient.getDatabaseNames();
        db = mongoClient.getDB(dbname);
        return dbnames.contains(dbname);
    }

    public DBCollection getCollection(String name) {
        return db.getCollection(name);
    }

    public WriteResult insert(String collectionName, Object object) {
        setCollection(collectionName);
        return insert(object);
    }
```

```
    public WriteResult insert(Object object) {
        return collection.insert(DBOAdapter.objectToDBObject(object));
    }

    public DBObject findOne() {
        return collection.findOne();
    }

    public DBObject findOne(String collectionName) {
        setCollection(collectionName);
        return findOne();
    }

    public DBObject findOne(BasicDBObject query) {
        return collection.findOne(query);
    }

    public DBObject findOne(String collectionName, BasicDBObject query) {
        setCollection(collectionName);
        return findOne(query);
    }

    /**
     * Retourne tous les documents de la collection
     *
     * @param collectionName
     * @return
     */
    public Cursor find(String collectionName) {
        setCollection(collectionName);
        return find();
    }

    public Cursor find() {
        return collection.find();
    }

    public WriteResult insert(String collectionName, List<? extends Model> objects)
{
        setCollection(collectionName);
        BasicDBObject[] dbList = new BasicDBObject[objects.size()];
        int i = 0;
        for (Model m : objects) {
            dbList[i++] = DBOAdapter.objectToDBObject(m);
        }
        return collection.insert(dbList);
    }

    public void save(String collectionName, List<? extends Model> objects) {
        setCollection(collectionName);
        for (Model m : objects) {
            collection.save(DBOAdapter.objectToDBObject(m));
        }
    }

    public Cursor find(String collectionName, BasicDBObject query) {
```

```
        setCollection(collectionName);
        return find(query);
    }

    public <T extends Model> List<T> load(Cursor cursor, Class<T> clazz) {
        setCollection(clazz.getSimpleName());
        List<T> result = new ArrayList<>();
        while (cursor.hasNext()) {
            result.add(DBOAdapter.dboToModel(cursor.next(), clazz));
        }
        return result;
    }

    public <T extends Model> List<T> load(BasicDBObject query, Class<T> clazz) {
        return load(find(query), clazz);
    }

    public <T extends Model> List<T> load(Class<T> clazz) {
        return load(find(clazz.getSimpleName()), clazz);
    }

    public Cursor find(BasicDBObject query) {
        return collection.find(query);
    }

    public void setCollection(String name) {
        collection = db.getCollection(name);
    }

    public List<String> getDbNames() {
        return mongoClient.getDatabaseNames();
    }

    public void close() {
        mongoClient.close();
    }

    public void dropCollection(String name) {
        this.setCollection(name);
        collection.drop();
    }

    public void dropCollection(Class<? extends Model> clazz) {
        this.dropCollection(clazz.getSimpleName());
    }

    @SuppressWarnings({ "rawtypes", "unchecked" })
    public void dropCollections(Class... classes) {
        for (Class clazz : classes) {
            dropCollection(clazz);
        }
    }

}
```