



UNIVERSITÀ DI PISA

Progetto Finale
Large Scale and Multistructured Databases

AUTORI:

FEDERICO BRACCI, ALESSIO DI RICCO, MARIELLA MELECHI

ANNO ACCADEMICO:

2022/2023

Table of Contents

1	Description.....	3
2	Main Actor	4
3	Functional Requirements.....	4
4	Non-functional	5
5	Use case Diagram.....	6
6	UML class diagram	7
7	Dataset Modelling	8
	7.1 Dataset Description.....	8
	7.2 Scraping Data.....	8
	7.3 Databases and Motivation.....	9
8	Design of MongoDB	9
	8.1 Collection: User.....	9
	8.2 Collection: BusinessActivity	11
	8.3 Collection: Open Notification	13
9	Design of Neo4j	14
10	Implementation	17
	10.1 Frameworks.....	17
	10.2 Package Organization.....	17
	10.2.1 it.unipi.lsmsd.controller.....	17
	10.2.2 it.unipi.lsmsd.controller.impl.....	17
	10.2.3 it.unipi.lsmsd.DTO.....	18
	10.2.4 it.unipi.lsmsd.entity.....	18
	10.2.5 it.unipi.lsmsd.service.....	22
	10.2.6 it.unipi.lsmsd.util	23
	10.3 MongoDB relevant operations	24
	10.3.1 Update Reservation.....	24
	10.3.2 Follow User.....	27
	10.4 Analytics.....	30
	10.4.1 User side.....	30
	10.4.2 Business side.....	32
	10.5 Neo4j relevant Graph-Domain Queries.....	37

11 Indexes.....	43
12 Replica configuration.....	44
13 Sharding idea.....	45
14 Consistency	46
15 Usage Manual.....	50

1 DESCRIPTION

If you want to rent a vehicle, probably the first question you ask is:

How can I find a professional and trustworthy rental business? How and where can I book a rental at an advantageous price?

That's why you'll find RentaForm useful for this purpose.

Rentaform is an application that collects the services and the offers of different rental companies. Specifically, the application allows various rental companies to offer their vehicle rental services to users. The latter can browse the various business's services and decide according to their needs or based on the reviews of users who have already used the services. In fact, user can rate and give advice about their own experience with the rental company, so help other users in all these tasks. They can also view and follow other experienced user, to stay informed about their activities and, for instance, to view their favorites based on the highest reviews of the followed users.

These latter functions are specific for users who register on the platform, as well as the booking of the specific rental.

2 Main Actors

The application will interact only with users that are distinguished in three roles:

- **Unregistered User:** user with limited access to the application.
- **Registered User:** user with complete access to the functionalities.
- **Business User:** user who can link at the account his/her business activity and offers services.
- **Admin:** registered user who is in charge to check any violation of the code of conduct from all the other roles, moreover they offer aid.

3 Functional requirements

- **Unregistered User**
 - The system must allow unregistered user to browse the list of business user activities and see their services.
 - The system must allow unregistered user to browse the list of rental services and vehicles for a specific activity
 - View information about a specific vehicle.
 - Look for the price of a service proposed by a specific business activity.
 - Browse reviews about specific business activity.
 - The system must allow an unregistered user to signup/login as a registered user.
- **BusinessActivity** need also to sign up, but they must provide additional information, to prove that they are a real business activity owner (o who is in charge).
- **Admin** don't follow the "normal" registration process, they'll have to do an interview to obtain the system-supervisor credential access (in the first instance only the application developers will cover this role).
- **Registered User**
 - Has the same unregistered user functionalities
 - Can rent a vehicle for a certain amount of time
 - Can view history orders.
 - Can review a **BusinessActivity** with a comment and a grade from 0 to 5.

- Can search for a specific **Business User** via the search bar.
- Must book for the specific rent.
- Can view the **Business User** information.
- Can send an assistance ticket.
- Can get some information using analytics.
- Can follow another user.
- Can get information about his followers regarding their followers/followed list.

- **BusinessActivity**

- Providing information about the activity itself.
- Adding the offered rental services.
- Can Add new vehicles
- Making special offer on one or more services.
- Can report to the **Admin** if he/she thinks that a review is unfair (e.g., fake review).
- can see his/her activity's analytics.
- Can send an assistance report
- Can view history orders.
- Can view history review.

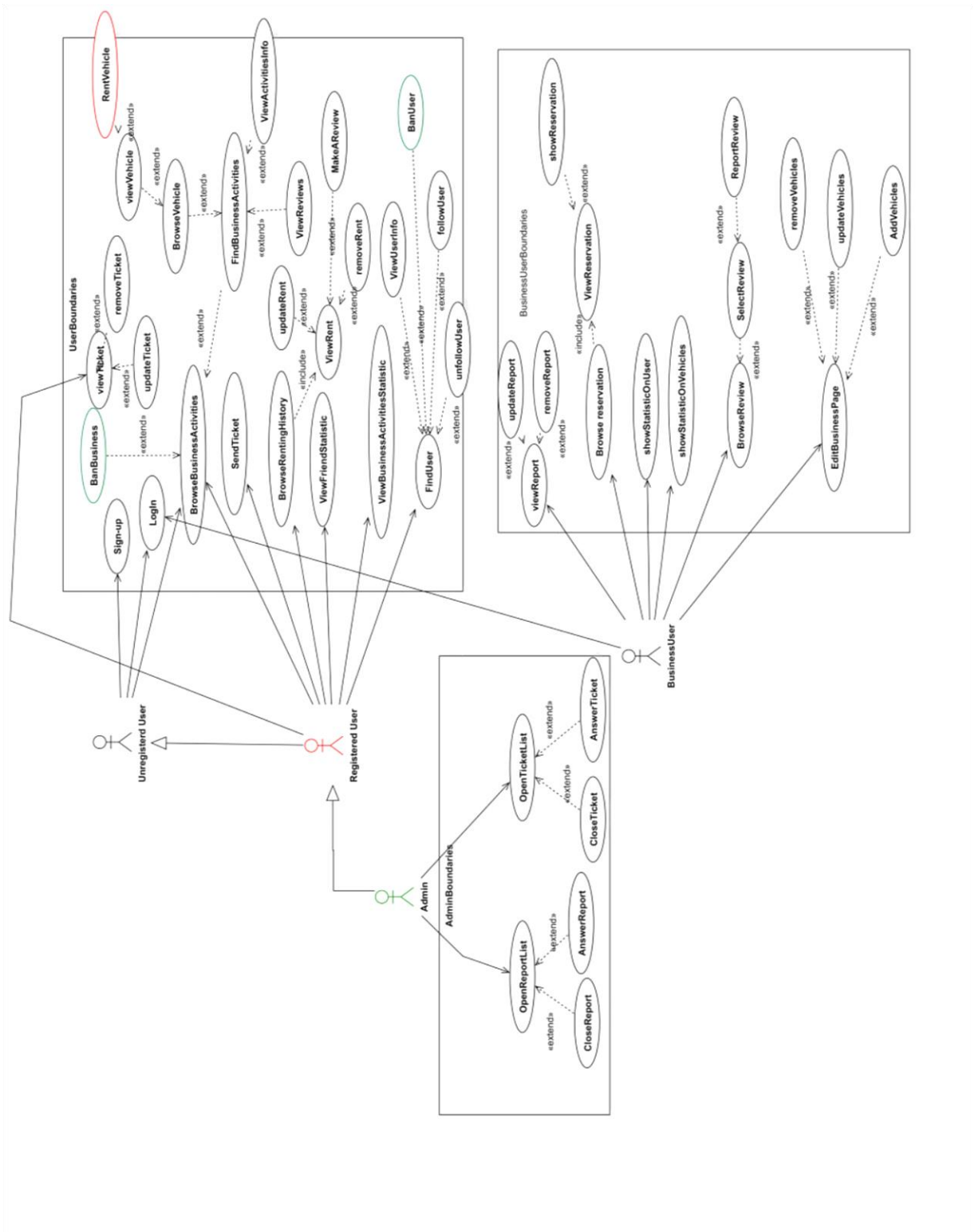
- **Admin**

- can see the report send from **BusinessActivity**
- can see the ticket send from **RegisteredUser**
- can ban a **User** or a **Business** from the Application
- can read assistance tickets/report requests and reply for give support.

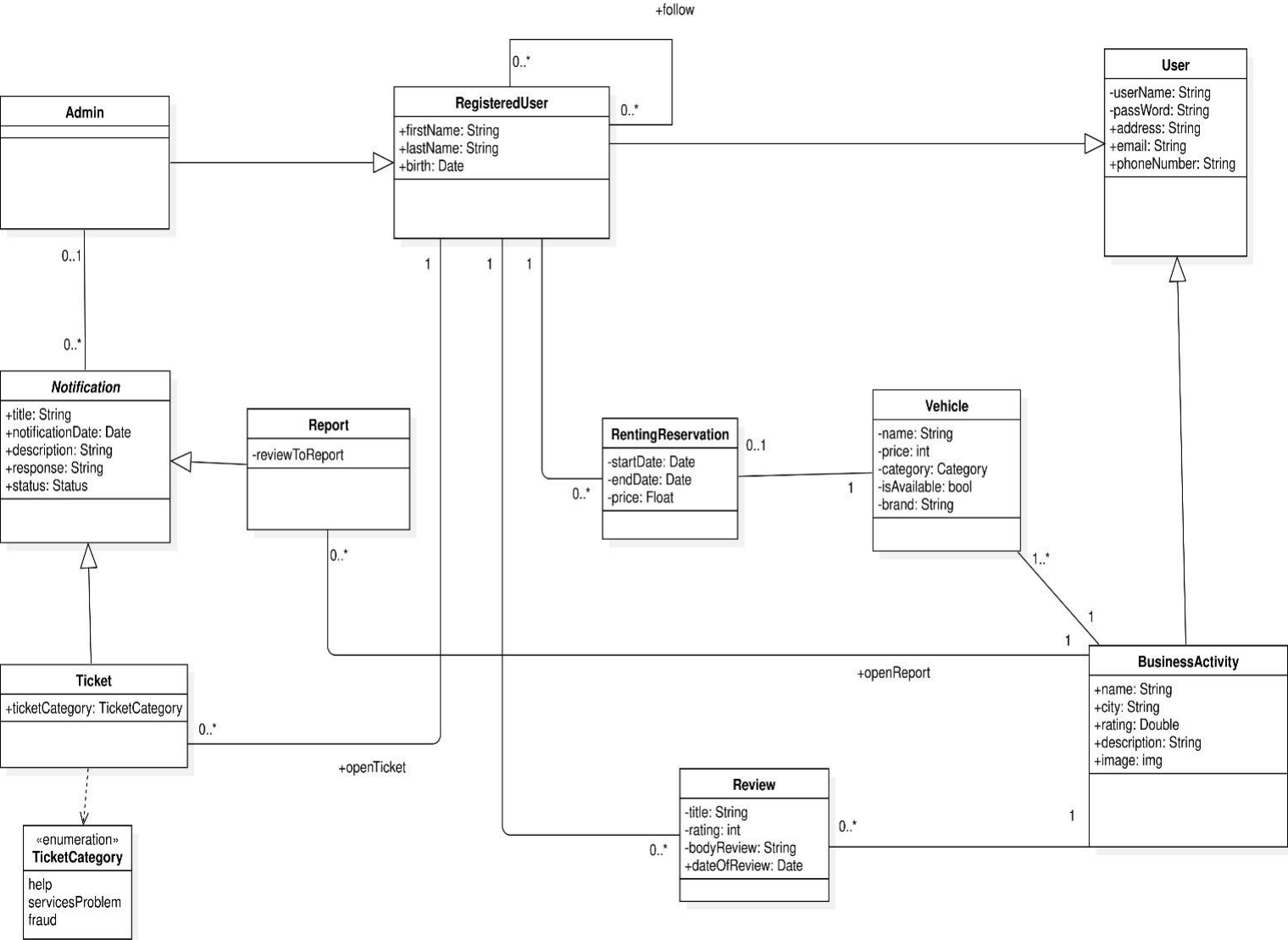
4 Non-Functional requirements

- **Performance** requirements:
 - **Response time:** System should response to the input user request in a order of 10-20 ms.
- **Dependability** requirements:
 - **Robustness:** System should continue to work even under erroneous user input
 - **Security:** As a web application it should provides a good level of security against malicious attack and preserve the user's personal data (e.g. password must be encrypted)
 - **Availability:** System should be operative 24/7
 - The systems must support hundreds of **concurrent accesses**.
 - The system must communicate with the via **HTTP**
 - Avoid single point of failure
- **Maintenance** requirements:
 - **Portability:** System should be installed only on the server machine, while a normal user needs only a pc with internet connection
- The code must be written in a OOP

5 Use-Case Diagram



6 UML Class Diagram



7 Data Modelling

7.1 Dataset Description

Dataset contains different vehicles such as car and motorbike with reviews in JSON format. These reviews will be assigned to a random generated user with a random pattern (0 to 10 reviews each user)

Volume: 63Mb

Variety: Data are scraped using different sources such as Edmund and rentCars

Velocity/Variability: Our dataset doesn't lose importance after time, so no refresh mechanism from the external world is taken in account. The dataset will be only modified from the interaction with the users through the application.

7.2 Scraping Data

The User collection was generated through the Random User API (<https://randomuser.me/>) and then we have associated both the vehicle reviews and the various bookings, obtained by scraping various booking sites such as Trustpilot, to the users, maintaining consistency between them.

The scraping process was made with python and Selenium, here down below a snippet of the scraping code.

```
1 for name,site in SITE_TO_SCRAPED.items():
2     for page in range(25):
3         sleep(5)
4         driver.get(f'{site}/?page={page+1}')
5
6         wait = WebDriverWait(driver, 20)
7         wait.until(EC.visibility_of_element_located((By.XPATH, "//img")))
8
9         elements = driver.find_elements(By.XPATH ,
10                                         "//*[contains(@class, 'styles_cardWrapper__LcCPA styles_show__HUXRb styles_reviewCard__9HxJJ')]")
11
12         for element in elements:
13             #rating = element.find_element_by_css_selector("img")
14             rating = element.find_element(By.CLASS_NAME,"styles_reviewHeader__iU9Px").get_attribute(
15                 "data-service-review-rating")
16             subject = element.find_element(By.TAG_NAME,"h2").text
17             review = element.find_element(By.TAG_NAME,"p").text
18             ratings.append(rating)
19             subjects.append(subject)
20             reviews.append(review)
21
22 data_frame_to_save = pd.DataFrame({"subject":subjects,"review":reviews,"rating":ratings})
23 data_frame_to_save.to_excel(f"dataset/{name}.xlsx",index=False)
```

7.3 Database and Motivation

We have chosen to adopt two different databases for our application:

- **MongoDB**: a document database to exploit the support that provides in terms of storage, indexing and complex queries elaboration (critical for the analytic part of the application), schema-less: improved application flexibility, lastly sharding replication in order to help the system availability.
- **Neo4j**: we have chosen a graph database to efficiently represent the relationships between different entities in the application (e.g., the “follow” relationship between users and the “rating” of a business Activity) and to analyze the paths created by these relationships, follow and rating are values easily representable via a graph.

8 Design of MongoDB

8.1 Collection: User

User’s collection stores all information about the registered user who uses the platform.

As we can see, there are many information that are saved during the registration process. The ‘isAdmin’ field is used during login to determine whether the user logged in is admin or not, while the ‘credential’ field is used to keep the password information saved in the signup phase, where a salt is generated and then it is added to the password entered by the user and then encrypted with SHA1.

```
_id: ObjectId('64610784b1dd804abd229b3e')
address: "979 Rue Paul-Duvivier"
▼ credential: Object
  salt: "40nqoAci"
  sha1: "05687f29699c09440af4a724319d410ddea34b3f"
dateOfBirth: 1993-08-15T00:00:00.000+00:00
email: "leon.francois@example.com"
firstName: "Léon"
isAdmin: false
lastName: "Francois"
occupation: "public employee"
phoneNumber: "077 997 02 33"
► rentingReservation: Array
► reviews: Array
username: "beautifulfish950"
```

We decided to embed the Reviews and the Reservations documents thinking about the fact that rarely we will want to see all reviews regarding each user, but frequently we will want to access one or more reviews/reservations depending on some parameters.

```
-----  
▼ rentingReservation: Array  
  ▼ 0: Object  
    businessActivity: "Refined Ride"  
    category: "car"  
    endDate: 2020-02-11T00:00:00.000+00:00  
    identifier: "6464a313627baf253343604d"  
    price: 310  
    startDate: 2020-02-01T00:00:00.000+00:00  
    vehicle: "Prius"
```

In the Renting Reservations document, we have decided to save some essential information to book a vehicle. Specifically, the booking start and end date and vehicle identifier are essential information as they are used to make the booked vehicles unavailable in case other users want to book it on the same day. Furthermore, when cancelling and modifying a reservation, these fields are used to enable/disable a scheduler which takes care of modifying the availability of the vehicle in question.

```
reviews: Array  
▼ 0: Object  
  businessActivity: "Refined Ride"  
  dateOfReview: 2020-02-26T00:00:00.000+00:00  
  rating: 5  
  review: "Prompt service, excellent customer service. Will surely use this servi..."  
  subject: "Great customer service"
```

In the same way, the review document presents among the fields the rating which is attributed by the user to the service offered by the business, and which is used to carry out a series of evaluations.

8.2 Collection: BusinessActivity

Business Activity's collection store all information about the business activity who offer their services.

```
_id: ObjectId('64611d3209b8388e0635eb06')
address: "9721 East Mulberry Street Sacramento, CA 95820"
city: "California"
▸ credential: Object
  description: "Maiami rent offers a wide range of vehicles for rent, including cars, ..."
  image: "Maiami_rent.png"
  name: "Maiami rent"
  phoneNumber: "+1 202-918-2132"
▸ rentingReservation: Array
▸ reportList: Array
▸ reviews: Array
  username: "Maiamirent@Business"
▸ vehicle: Array
```

As for the users, we have decided to embed both the reviews and the reservations but there is also the collection of vehicles that each business activity offers. they can be both cars and scooters.

```
▼ vehicle: Array
  ▼ 0: Object
    automaticTrasmission: true
    brand: "Ferrari"
    category: "car"
    image: "Ferrari.png"
    isAvailable: true
    name: "California"
    price: 955
    vehicleIdentifier: "64647518627baf253343527d"
    year: 2011
```

8.3 Collection: open notification

Open notification collection is in charge to store all the reports and ticket send by respectively BusinessActivity and User, waiting to be taken by an admin.

These documents respectively represent the reports that business activities can request for any problems with user reviews that do not comply with site policy, and the support tickets that users can send for different kind of support.

The document structure for tickets and reports inside the collection is the following:

```
_id: ObjectId('648f60eb9655ac40344e8712')
applicant: "beautifulfish950"
requestCategory: "TICKET"
▼ ticket: Object
  category: "HELP"
  description: "Hello i'd like to know if there are other ways to contact the platform..."
  ticketDate: 2023-06-18T19:54:19.000+00:00
  ticketId: "648f60eb9655ac40344e8712"
  title: "Contact Customer Service"

_id: ObjectId('648f613d9655ac40344e8715')
applicant: "Maïamirent@Business"
▼ report: Object
  dateOfReport: 2023-06-18T19:55:41.000+00:00
  description: "i have a problem with this review"
  reportId: "648f613d9655ac40344e8715"
  ▼ review: Object
    dateOfReview: 2020-09-21T00:00:00.000+00:00
    rating: 2
    review: "I never received confirmation by email. Luckily I had not closed the w..."
    subject: "I never received confirmation by email"
    username: "brownbird570"
    title: "problem with this review"
  requestCategory: "REPORT"
```

Tickets and reports document are also embedded inside the User and Business document as array of embedded object.

9 Design of Neo4j

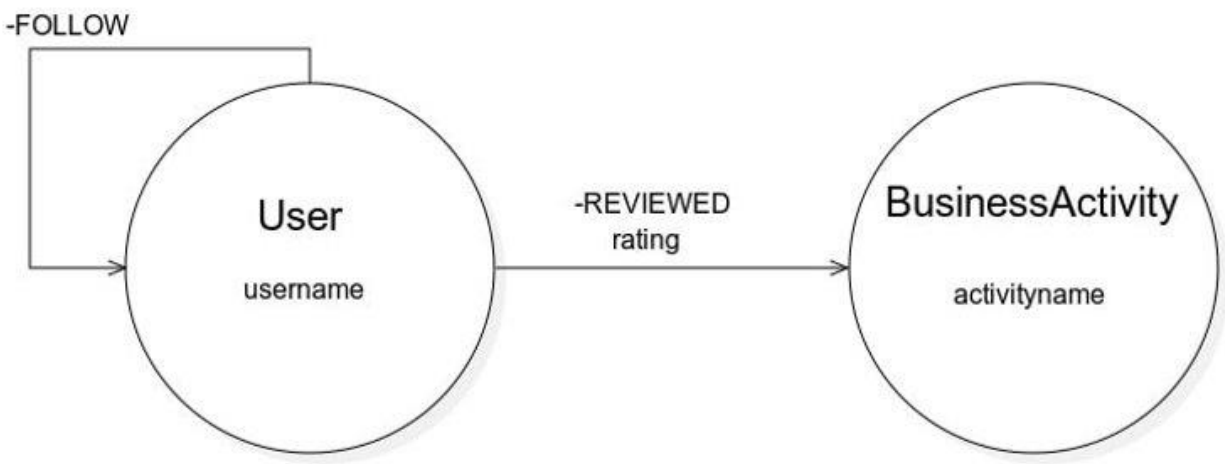
We decided to insert to the graph database two entity:

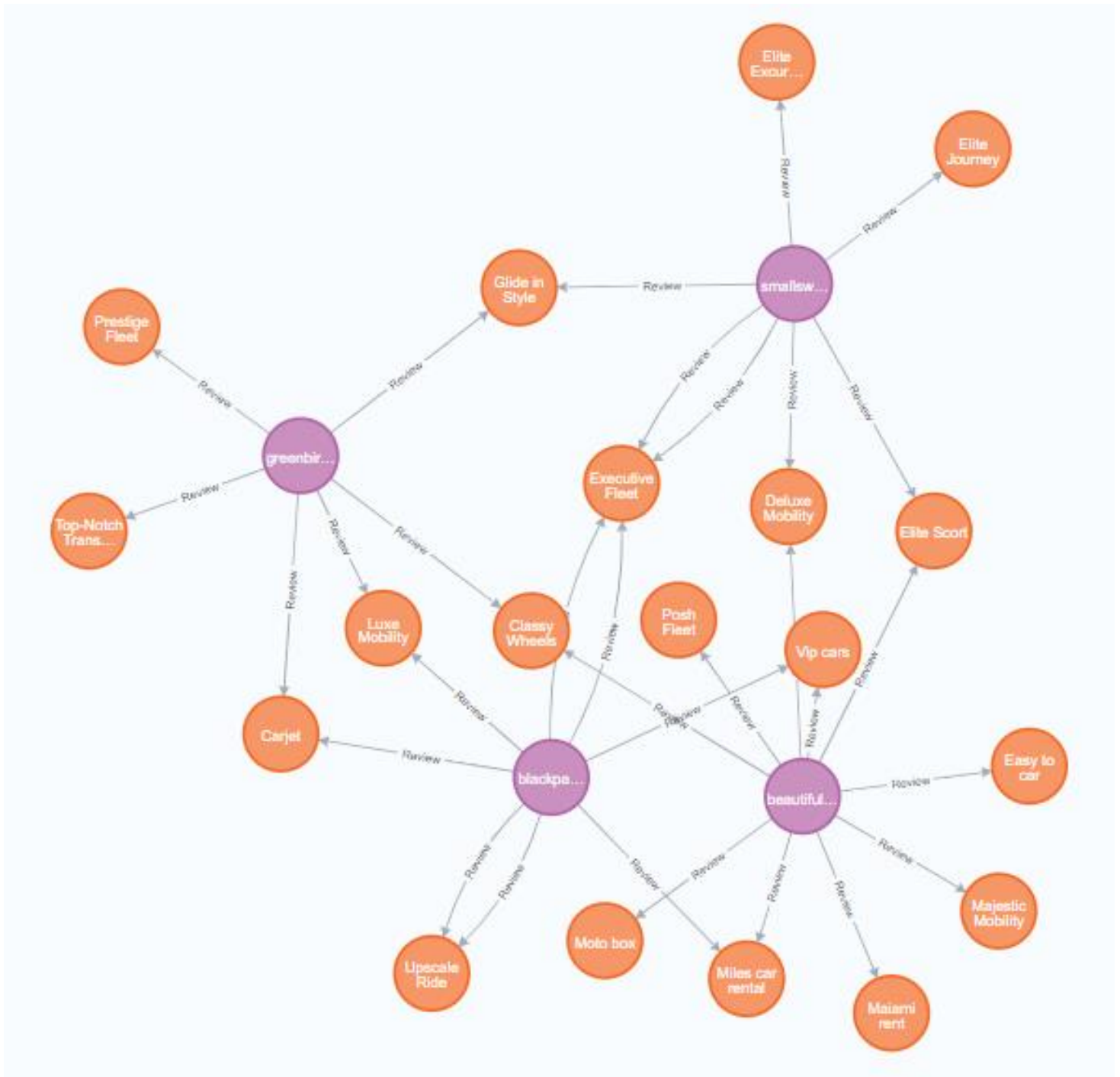
1. User
2. BusinessActivity

And two relationships:

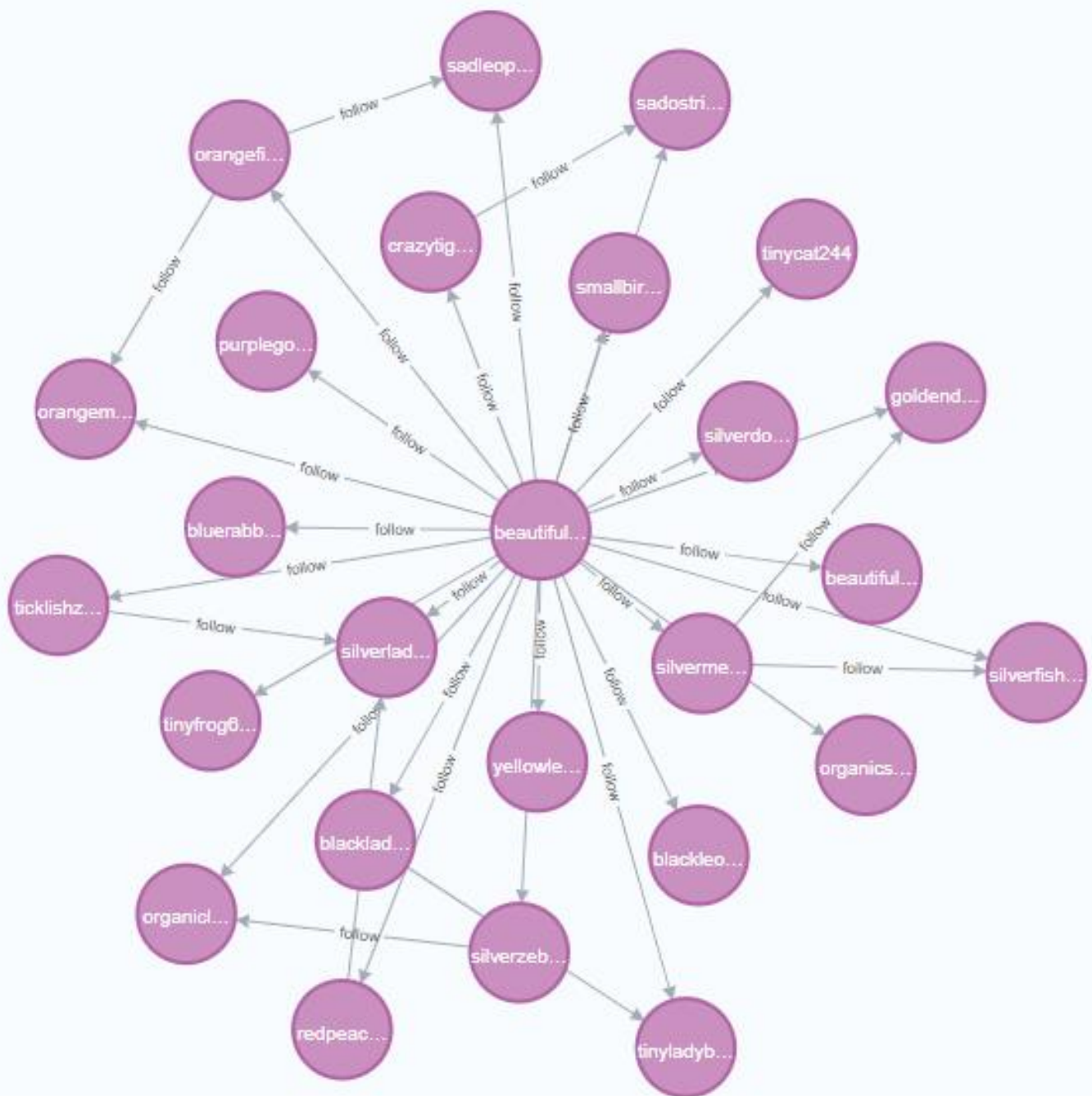
1. **FOLLOWS**
2. **RATING**

The user entity is identified by name. This entity can have 2 types of relationship: **FOLLOWS**, between user and user to implement the social part of the platform, and **RATING** between user and Business Activity to keep track of the rating of business reviews to then make rankings. In the same way, the BusinessActivity entity is identified by its name.





Example of RATING relationship



Example of FOLLOW relationship

10 IMPLEMENTATIONS

10.1 Frameworks

We have decided to use the Spring-boot Framework to handle the entire project:

- We used the Spring-Dependency Injection via the `@Autowired` annotation.
- We used the `@SessionAttributes` annotation to save session attributes during the project (like logged status or current game selected).
- We used `@Controller`, `@RestController`, `@Service`, `@GetMapping` in order to expose the application services to the front-end (via API).

10.2 Package Organization

We divided our project in the following packages:

10.2.1 `it.unipi.lsmsd.controller`

In this section we insert all interfaces relative to the implementation part which is described later.

Below we list all interfaces we have introduced:

- `AdminControllerInterface`
- `BusinessActivityControllerInterface`
- `UserControllerInterface`
- `GenericControllerInterface`

10.2.2 `it.unipi.lsmsd.controller.impl`

Here are grouped all class labeled with the annotation `@RestController`, which are used to return DTOs and answers to requests (Figure 11).

We decided to get a controller class for each entity plus a generic one to handle common request from all the entities.

Below we list all `@RestControllers` we have introduced:

- `AdminController`
- `BusinessActivityController`
- `UserController`

- GenericController

10.2.3 it.unipi.lsmsd.DTO

Inside this package we list all DTOs that are passed to the controller to help the front end to build pages. DTOs names are self-explanatory, we alight only some of them in the list below:

- AnalyticsRequest
- Business → Aggregation of business activity information such as rating, city, etc
- GenericPage → Aggregation of all the user information
- GenericResponse
- RentingReservation
- Review
- SpecialUserAddRequest → Used to handle new business or admin insertion into the application.
- SupportRequest
- UpdateNotificationRequest
- UpdateReportRequest
- UpdateReservation
- UpdateReview
- UpdateTicketRequest
- UserAccessRequest
- User → Aggregation of all the user information's used in signup stage.
- Vehicle → Aggregation of all the vehicle information.
- VehicleInsertRequest
- VehicleUpdateRequest
- VehicleUpdateResponse

10.2.4 it.unipi.lsmsd.entity

In this package we list all models that reflects the back end. Models are only accessible from Service and Component and Controller levels. We implemented a model for the main entities of our application: BusinessActivity (Figure X), RentingReservation (Figure X), Report, Review, Ticket (Figure X), Vehicle (Figure X) and User (Figure X).

```
@Document(collection = "BusinessActivity")
@JsonInclude(JsonInclude.Include.NON_NULL)
public class BusinessActivity {

    @BsonProperty(value = "name")
    private String name;
    @BsonProperty(value = "username")
    private String username;
    @BsonProperty(value = "address")
    private String address;
    @BsonProperty(value = "phoneNumber")
    private String phoneNumber;
    @BsonProperty(value = "email")
    private String email;
    @BsonProperty(value = "city")
    private String city;
    //@BsonProperty(value = "password")
    private Password password;
    @BsonProperty(value = "description")
    private String description;
    @BsonProperty(value = "rating")
    private Double rating;
    @BsonProperty(value = "reviews")
    private List<Review> reviews;
    @BsonProperty(value = "rentingReservation")
    private List<RentingReservation> rentingReservation;
    @BsonProperty(value = "vehicle")
    private List<Vehicle> vehicle;
    @BsonProperty(value = "reportList")
    public List<Report> reportList;
```

BusinessActivity entity

```

@JsonInclude(JsonInclude.Include.NON_NULL)
public class RentingReservation {

    @BsonProperty(value = "id")
    private Integer id;

    private String businessActivity;
    private String user;
    private String vehicle;

    @JsonFormat(pattern = "yyyy-MM-dd'T'HH:mm Z")
    private Date startDate;
    @JsonFormat(pattern = "yyyy-MM-dd'T'HH:mm Z")
    private Date endDate;
    private Float price;
    private String category;

    private String identifier;

```

RentingReservation entity

```

@JsonInclude(JsonInclude.Include.NON_NULL)
public class Report {

    private String businessActivityName;
    private String title;
    private String description;
    private SupportStatus status;
    private String admin;
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss.SSS", timezone = "UTC")
    private Date dateOfReport;
    private Review review;
    private String reportId;
    private String response;

```

Report entity

```

public class Vehicle {

    @BsonProperty(value = "id")
    private Integer id;
    @BsonProperty(value="automaticTrasmission")
    private Boolean automaticTrasmission;
    @BsonProperty(value="brand")
    private String brand;
    @BsonProperty(value="name")
    private String name;
    @BsonProperty(value="year")
    private Integer year;
    @BsonProperty(value="price")
    private Float price;
    @BsonProperty(value="category")
    private Costant.VehicleCategory category;
    @BsonProperty(value="isAvailable")
    private Boolean isAvailable;
    @BsonProperty(value="vehicleIdentifier")
    private String vehicleIdentifier;

```

Vehicle entity

```

@JsonInclude(JsonInclude.Include.NON_NULL)
public class Ticket {
    private String username;
    private String title;
    @JsonFormat(pattern = "yyyy-MM-dd' 'HH:mm:ss",timezone = "UTC")

    private Date ticketDate;
    private String admin;
    private String description;

    private TicketCategory category;

    private String response;

    private SupportStatus status;

    private String ticketId;

```

Ticket Entity

```

@JsonInclude(JsonInclude.Include.NON_NULL)
@Document(collection = "User")
public class User {

    private Object id_;
    private Boolean isAdmin;

    private String occupation;
    private String firstName;
    private String lastName;
    private String username;
    private Password password;
    private String address;
    private String email;
    private String phoneNumber;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private Date dateOfBirth;
    private List<Review> reviews ;
    private List<RentingReservation> rentingReservation;
    private List<Ticket> ticketList ;
    private List<Report> reportList;

```

User entity

```

@JsonInclude(JsonInclude.Include.NON_NULL)
public class Review {
    private String id_;
    private String subject;
    private String review;
    private Integer rating;
    @JsonFormat(pattern = "yyyy-MM-dd",timezone = "UTC")
    private Date dateOfReview;
    private String username;
    private String businessActivity;

```

Review Entity

10.2.5 it.unipi.lsmsd.service

In this package we insert classes that follows the singleton design pattern and provide service like the connection to the 2 DBs used in this project: Neo4j and MongoDB and the notification manager.

This package consists of the following classes:

- MongoDBDriver
- GraphDriver
- NotificationManager

MongoDriver and GraphDriver manage the database connection at the application start and exploit some useful methods like getters for retrieve collections and client session (Used for transactions for both Mongo and Neo4j).

NotificationManager is in charge to manage tickets and Reports that can be send respectively by Users and Business.

When a new ticket (or report) arrives, the Service, will save it in the document embedded list of the relative users inside the User collection and manipulate it in a new Entity that is also saved in a different collection called **open_collection**, so it will be available for all the admin.

Important functionalities are:

```
public GenericResponse addToNotification(SupportRequest supportRequest){}

public GenericResponse dispatchSupportRequest(SupportRequest
supportRequest){}

public List<SupportRequest> getAll(int page){};

public List<Document> getAllTicket(int page,String username,String
sortTarget,String direction,String searchText){}

public List<Document>getAllReport(int page, String username,String
sortTarget,String direction,String searchText) {}

public GenericResponse update(UpdateNotificationRequest updateRequest){}
```

Add to notification as written above, it allows user (or business) to send their ticket (or report).

Dispatch is used by the admin to take in charge a user's ticket (or Business Report), and it will implement all the necessary logic to remove the ticket from the open_notification collection, assign it to the applicant admin and also set the ticket status as "TAKEN".

The various getters allow user, business, and admin to have a look up on their ticket or report (for the admin it shows the ones he/she taken in charge or in the case of the “getAll”, to get everything that hasn't already been taken care of by any admin).

The update method is useful for user (and business) if they need to make a change to their ticket(or report) or if they want to delete it.

For admin on the other hand, it is useful for replying or for flag it with the status “CLOSED”.

10.2.6 it.unipi.lsmsd.util

Inside this package we put three classes:

- Inside class Constants we put RequestCategory, UserCategory, VehicleCategory that keeps the typology of a specific request or entity and SupportStatus to handle the status of a ticket.
- Inside class Password we handle login passwords, retrieve them from DB encryption and compare to verify correctness.
- Inside class TransactionUtil there are some methods used to handle transaction in both db separately and even between both.

10.3 MongoDB Relevant Operations

We put here implementation of both some common operations described before and Admin Side analysis functions.

10.3.1 Update Reservation

To update a reservation the application calls the following methods:

- `public ResponseEntity<String> updateReservation (UpdateReservation reservationUser, HttpSession httpsession)`
- `public boolean UpdateReservation (UpdateReservation reservationUser)`
- `public MongoCollection<BusinessActivity> getBusinessActivityCollection ()`

- `public String getReservation ()`

The first method is a `@RestController`, and its task is to handle the get request performed when a user try to modify a reservation. Once the request has been handled by the `@RestController`, the application calls the

`@Component` associated to the `RentingReservation` entity, In particular, the method `UpdateReservation ()`.

At this point the `@Component` uses the methods made available by `@Service` and `DTO` classes to retrieve the correct reservation.

Below are inserted some snapshot regarding the implementation of the first two functions.

```
@PostMapping("/updateReservation")
@Override
public ResponseEntity<String> updateReservation( UpdateReservation reservationUser) {
    RentingReservation rs = new RentingReservation(reservationUser.getReservation().getBusinessActivity(), reservationUser.getUser(), reservati
    boolean result;
    result=BService.reservationValidity(rs);
    if(!result) {
        return new ResponseEntity<>( body: "VEHICLE NOT AVAILABLE ON THESE DATES,CHOOSE OTHERS", HttpStatus.BAD_REQUEST);
    }
    result = BService.UpdateReservation(reservationUser);
    if (result)
        return new ResponseEntity<>( body: "Reservation modified", HttpStatus.OK);
    return new ResponseEntity<>( body: "Reservation not modified,vehicle not available in these date", HttpStatus.BAD_REQUEST);
}
```

After checking if the user is correctly logged in the session `UpdateReservation` function is called, depending on the result obtained an `Http` request is send to the frontend handler.

```

public boolean UpdateReservation(UpdateReservation reservationUser) {
    MongoCollection<User> userColl = mongoDriver.getUserCollection();
    MongoCollection<BusinessActivity> businessColl = mongoDriver.getBusinessActivityCollection();
    ClientSession session = mongoDriver.getSession();
    TransactionBody<Boolean> transaction = () -> {
        Field[] field = reservationUser.getClass().getDeclaredFields();
        ArrayList<Bson> updates = new ArrayList<>();
        for(Iterator<Field> iterator = Arrays.stream(field).iterator(); iterator.hasNext();){
            Field element = iterator.next();
            switch (element.getName()){
                case "StartDate":
                    if(reservationUser.getStartDate() != null){
                        Bson updatesDate = Updates.set("rentingReservation.$.startDate",reservationUser.getStartDate());
                        updates.add(updatesDate);
                    }
                    break;
                case "endDate":
                    if(reservationUser.getEndDate() != null){
                        Bson updateEdate = Updates.set("rentingReservation.$.endDate",reservationUser.getEndDate());
                        updates.add(updateEdate);
                    }
                    break;
                case "id":
                    if(reservationUser.getId() != null){
                        Bson updatefId = Updates.set("rentingReservation.$.id",reservationUser.getId());
                        updates.add(updatefId);
                    }
                    break;
            }
        }
    }
}

```

After selecting the correct reservation and depending on the field to update we modify the value on the DB with updates. Add ().

To maintain consistency when a user creates, deletes, or modifies a reservation, we have implemented a scheduler via **Quartz API** provided by Spring. Specifically, when the booking is created, the scheduler is started which, depending on the booking start and end date, performs an operation on the database, modifying the 'is Available' field of the vehicle involved to prevent other users from booking it. In the same way, if a user deletes or modify a reservation, the scheduler relating to the old dates is first stopped and a new one is created. Quartz has a modular architecture. It consists of several basic components that we can combine: **Job**, **Job Detail**, **Trigger** and **Scheduler**. The API provides a **Job** interface that has just one method, *execute*. It must be implemented by the class that contains the actual work to be done, i.e. the task. When a job's trigger fires, the scheduler invokes the *execute* method, passing it a *JobExecutionContext* object.

The *JobExecutionContext* provides the job instance with information about its runtime environment, including a handle to the scheduler, a handle to the trigger, and the job's *JobDetail* object. Each job and trigger created is identified in the application by vehicle identifier and reservation date. In the 'Is Available' method we start the scheduler which, using the Boolean 'v', will appropriately modify the database field.

```

import org.quartz.JobDataMap;
import org.quartz.JobExecutionContext;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.quartz.QuartzJobBean;
import org.springframework.stereotype.Component;

```

```

@Component
public class MyJob extends QuartzJobBean {

    @Autowired
    UserComponentLogic UserService;

    @Override
    protected void executeInternal(JobExecutionContext jec) {
        JobDataMap dataMap = jec.getJobDetail().getJobDataMap();
        String id = dataMap.getString( key: "id");
        String name=dataMap.getString( key: "name");
        boolean v=dataMap.getBoolean( key: "v");
        UserService.schedule_reservation(name,id,v);
    }
}

```

```

public void isAvailable(Date date,String id,String name, boolean v) {
    try {
        System.out.println(date);
        JobDetail job = JobBuilder.newJob(MyJob.class).withIdentity(id,date.toString()).build();
        job.getJobDataMap().put("name", name);
        job.getJobDataMap().put("id" ,id);
        job.getJobDataMap().put("v",v);
        Trigger trigger = TriggerBuilder.newTrigger().withIdentity(id,date.toString()).startAt(date).build();
        Scheduler scheduler = schedulerConfig.schedulerFactoryBean().getScheduler();
        scheduler.scheduleJob(job, trigger);
        scheduler.start();
    }catch (IOException | SchedulerException e) {
        e.printStackTrace();
    }
}
}

```

10.3.2 Follow User

To insert a new user into the DB the application calls the following methods:

- `public ResponseEntity<String> followUser (String username, HttpSession httpsession)`
- `public boolean FollowUser (String user, String friend)`
- `public Driver getGraphDriver ()`
- `public void addUserNeo4j (String username)`

The first method is a `@RestController`, and its task is to handle the get request performed when a user try to follow new Users. Once the request has been handled by the `@RestController`, the application calls the `@Component` functions associated to the user entity, in particular, the method `followUser()`. At this point the `@Component` uses the methods made available by `@Service` and DTO classes to retrieve the correct reservation.

Below are inserted some snapshot regarding the implementation of the component and neo4j functions.

```
@PostMapping("/followUser")
@Override
public ResponseEntity<String> followUser(String username, HttpSession httpsession) {
    boolean result = UserService.FollowUser(httpsession.getAttribute("userlog").toString(), username);
    if (result)
        return new ResponseEntity<>("NEW FOLLOWER ADDED", HttpStatus.OK);
    return new ResponseEntity<>(HttpStatus.NO_CONTENT);
}
```

```

public boolean FollowUser(String user, String friend) {

    boolean result = true;
    try {
        Session session = graphDriver.getGraphDriver().session();
        session.run( s: "MATCH (uA:User) WHERE uA.username=$user" +
            " MATCH (uB:User) WHERE uB.username=$friend" +
            " MERGE (uA)-[:follow]->(uB)",
            parameters( ...keysAndValues: "user", user, "friend", friend));
        session.close();
    } catch (Exception e) {
        e.printStackTrace();
        result = false;
    }
    return result;
}

```

```

public void addUserNeo4j(String username) {
    Session session = graphDriver.getGraphDriver().session();
    Transaction graphTransaction = session.beginTransaction();

    try {

        graphTransaction.run( s: "CREATE (u: user {name: $username})", parameters( ...keysAndValues: "username", username));

    } catch (Exception e) {
        graphTransaction.rollback();
        e.printStackTrace();
    }
    finally {
        graphTransaction.commit();
        session.close();
    }
}

```

10.4 Analytics

Both users and Business Activities can perform some analytics on data. Implementation and description are described in chapters below.

10.4.1 User side

-Top activities ranked by average rating for each city.

In Java we use **AggregateIterable<Document>** to scroll through all the documents in the collection and aggregate them in this specific case by name, city and average of reviews. After which, the list is sorted by highest average rating and the first 5 most quoted business activities are selected using the 'skip' operator.

```
[
  {
    $unwind: "$reviews",
  },
  {
    $group: {
      _id: {
        BusinessActivity: "$name",
        city: "$city",
      },
      avg_rating: {
        $avg: "$reviews.rating",
      },
    },
  },
  {
    $sort: {
      avg_rating: -1,
    },
  },
  {
    $limit: 5,
  },
]
```

-Activity ranking by last year earnings for historic vehicles

This query allows the User to keep track of the total gain in recent years considering historic vehicles

```
[
  {
    $unwind: "$rentingReservation",
  },
  {
    $match: {
      "rentingReservation.startDate": {
        $gte: ISODate("2020-01-01T00:00:00.000Z"),
        $lt: ISODate("2021-01-01T00:00:00.000Z"),
      },
    },
  },
  {
    $unwind: "$vehicle",
  },
  {
    $project: {
      businessActivity: "$name",
      rentingReservation: 1,
      vehicle: 1,
      compare: {
        $eq: [
          "$rentingReservation.vehicle",
          "$vehicle.name",
        ],
      },
    },
  },
]
```

```
    },
  },
  {
    $match: {
      compare: true,
      "vehicle.year": {
        $lt: 2010,
      },
    },
  },
  {
    $group: {
      _id: "$businessActivity",
      totalReservations: {
        $sum: "$rentingReservation.price",
      },
    },
  },
  {
    $sort: {
      totalReservations: -1,
    },
  },
]
```


-Most rented vehicle brand with customer's age greater/lower/equal than a parameter.

As we can see below, we adopt the same technique used for all the other analytics, grouping the documents by brand, category and city, counting the number of reservations made on that vehicle by users and choosing the brands of the most rented vehicles, provided that the calculation of the age made previously is greater than the specified parameter.

```
[
  {
    $match: {
      name: "Miami rent",
    },
  },
  {
    $unwind: "$rentingReservation",
  },
  {
    $lookup: {
      from: "User",
      localField: "rentingReservation.user",
      foreignField: "username",
      as: "UserInfo",
    },
  },
  {
    $unwind: "$UserInfo",
  },
  {
    $addFields: {
      age: {
        $subtract: [
          {

```

```

        {
            $year: "$$NOW",
        },
        {
            $year: "$UserInfo.dateOfBirth",
        },
    ],
},
},
},
{
    $match: {
        age: {
            $gt: 35,
        },
    },
},
{
    $unwind: "$vehicle",
},
{
    $addFields: {
        isMatch: {
            $cond: {
                if: {

```

```

            $eq: [
                "$vehicle.id",
                "$rentingReservation.id",
            ],
        },
        then: true,
        else: false,
    },
},
},
{
    $match: {
        isMatch: true,
    },
},
{
    $group: {
        _id: {
            Brand: "$vehicle.brand",
        },
        count: {
            $sum: 1,
        },
    },
},

```

```
    },  
  },  
  {  
    $sort: {  
      count: -1,  
    },  
  },  
  {  
    $limit: 5,  
  },  
}
```

10.4.1 BusinessActivity side

-Most used vehicle for user work typology

With these analytics we analyze the vehicles that have been rented the most considering the occupancy of the users who have rented.

Can be useful for a Business Activity to understand what kind of customer they have and what kind of vehicle they preferred to rent.

```
1 * {
2 * {
3 *   $match: {
4 *     name: "Miami rent",
5 *   },
6 * },
7 * {
8 *   $unwind: {
9 *     path: "$rentingReservation",
10 *   },
11 * },
12 * {
13 *   $lookup: {
14 *     from: "user",
15 *     localField: "rentingReservation.user",
16 *     foreignField: "username",
17 *     as: "join",
18 *   },
19 * },
20 * {
21 *   $replaceRoot: {
22 *     newRoot: {
23 *       $mergeObjects: [
24 *         {
25 *           $arrayElemAt: ["$join", 0],
26 *         },
27 *         "$$ROOT",
28 *       ],
29 *     },
30 *   },
31 * },
32 * {
33 *   $project: {
34 *     join: 0,
35 *   },
36 * },
37 * {
38 *   $group: {
39 *     _id: {
40 *       vehicle: "$rentingReservation.vehicle",
41 *       occupation: "$occupation",
42 *     },
43 *     count: {
44 *       $sum: 1,
45 *     },
46 *   },
47 * },
48 * {
49 *   $sort:
50 *   /**
51 *    * Provide any number of field/order pairs.
52 *    */
53 *   {
54 *     count: -1,
55 *   },
56 * },
57 * {
58 *   $group: {
59 *     _id: {
60 *       occupation: "$_id.occupation",
61 *     },
62 *     vehicle: {
63 *       $first: "$_id.vehicle",
64 *     },
65 *     count: {
66 *       $max: "$count",
67 *     },
68 *   },
69 * },
70 * {
71 *   $sort:
72 *   /**
73 *    * Provide any number of field/order pairs.
74 *    */
75 *   {
76 *     count: -1,
77 *   },
78 * },
79 }
```

-For each vehicle, count the total number of rentals in a period.

This query allows the business activity to keep track of what kind of vehicle is rented and its frequency across the years.

```
1  [
2  {
3    $match:
4    /**
5     * query: The query in MQL.
6     */
7    {
8      name: "Miami rent",
9    },
10  },
11  {
12    $unwind: {
13      path: "$rentingReservation",
14    },
15  },
16  {
17    $match: {
18      "rentingReservation.startDate": {
19        $lte: new Date("2024-01-02"),
20        $gte: new Date("2021-01-01"),
21      },
22    },
23  },
24  {
25    $group: {
26      _id: {
27        vehicle: "$rentingReservation.vehicle",
28      },
29      count: {
30        $sum: 1,
31      },
32    },
33  },
34  {
35    $sort: {
36      count: -1,
37    },
38  },
39  ]
```

10.5 Neo4j Relevant Graph-Domain Queries

-Show recommended rental activity based on the rating of your followers.

This query will navigate across the followed user and will return a list of business activity based on how they have been reviewed.

```
"MATCH (user:user {username:$username})-[:follow]->(following:user)" +  
  " MATCH (following)-[review:Review]->(activity:BusinessActivity)" +  
  " WITH activity, toFloat(avg(review.rating)) as avg_rating," +  
COUNT(DISTINCT following) as num_followers" +  
  " ORDER BY avg_rating %s" +  
  " SKIP $skip" +  
  " LIMIT 10" +  
  " RETURN activity.name as activity, avg_rating"
```

-Most active users among followed ones

This query will return a list of users based on their number of reviews.

```
MATCH (user_target: user)-[follow:follow]->(user: user)" +  
MATCH (user: user)-[Review:Review]->(BusinessActivity:BusinessActivity)" +  
WHERE user_target.username=$username" +  
RETURN DISTINCT " +  
user.username AS FOLLOWER, " +  
(count(DISTINCT BusinessActivity)) AS REVIEW_N" +  
ORDER BY REVIEW_N DESC" +  
LIMIT 5",
```

-Most followed users among followed ones

This query will return a list of users based on their number of followers.

```
"MATCH (user1:user)-[s:follow]->(friend:user)<-[r:follow]-
(friend_friend:user)" +
  " WHERE user1.username=$username" +
  " RETURN friend as FRIEND, count(r)+1 as COUNTER" +
  " ORDER BY COUNTER %s" +
  " SKIP $skip" +
  " LIMIT 10"
```

11 INDEXES

In this section we're going to make an analysis on the database architecture, and in particular on indexes. In our application some of the most used operations are searching for a User or a Business Activity using the username or displaying reviews, reservations, and vehicles sorted by some specific field. For this reason, the use of several indexes allows the user to quickly access to the database and to retrieve documents fast. The results shown below are taken in local, but we assume them to be the same if collected in remote.

11.1 MongoDB Indexes

To have fast readings for the user, which is searched by username, we decided to insert a regular index in the user collection using the username itself. MongoDB Compass shows that, before inserting the index we had to search between 3000 documents, spending 10 ms. After inserting the index, we obtain much better results in terms of documents inspected: 1 only document, and 0 ms spent in searching. We did the same for the Business Activity collection, where we have a regular index on the business name.

Filter ⓘ ⓘ {username : "redmeercat616"}

Query Performance Summary [Learn more](#) ⓘ

Documents Returned: 1

Index Keys Examined: 0

Documents Examined: 3000

Actual Query Execution Time (ms): 10

Sorted in Memory: no

⚠ No index available for this query.

Filter ⓘ ⓘ {username: "redleopard420"}

Reset

Query Performance Summary [Learn more](#) ⓘ

Documents Returned: 1

Index Keys Examined: 1

Documents Examined: 1

Actual Query Execution Time (ms): 0

Sorted in Memory: no

Query used the following index: USERNAME ↑

Filter ⓘ ⓘ {name: "Elite Excursion"}

Query Performance Summary [Learn more](#) ⓘ

Documents Returned: 1

Index Keys Examined: 0

Documents Examined: 40

Actual Query Execution Time (ms): 0

Sorted in Memory: no

⚠ No index available for this query.

Filter ⓘ ⓘ {name: "Miami rent"}

Query Performance Summary [Learn more](#) ⓘ

Documents Returned: 1

Index Keys Examined: 1

Documents Examined: 1

Actual Query Execution Time (ms): 0

Sorted in Memory: no

Query used the following index: NAME ↑

In addition, to sort the documents, various indices were considered to study their performance.

1. Indexes for sorting vehicles by brand: we add a compound index on both name of the Business Activity and the brand of the vehicle. These are the results:

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 63,  
  totalKeysExamined: 0,  
  totalDocsExamined: 40,
```

Results without index

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 35,  
  totalKeysExamined: 35,  
  totalDocsExamined: 1,
```

Results with index

2. Indexes for sorting reviews by rating : we add a index on both name of the Business Activity and the review's rating. In the same way we analyzed the case of sorting with the review's date and since it showed better performance we decided to add it.

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 6,  
  totalKeysExamined: 35,  
  totalDocsExamined: 1,
```

Results without index

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 2,  
  totalKeysExamined: 35,  
  totalDocsExamined: 1,
```

Results with index

3. Indexes for sorting reservation: we add an index on name of the Business Activity, start date, end date and brand.

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 4958,  
  totalKeysExamined: 5,  
  totalDocsExamined: 1,
```

Results without index (sorting by date)

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 4318,  
  totalKeysExamined: 5,  
  totalDocsExamined: 1,
```

Results with index (sorting by date)

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 4881,  
  totalKeysExamined: 5,  
  totalDocsExamined: 1,
```

Results without index (sorting by brand)

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 4282,  
  totalKeysExamined: 1,  
  totalDocsExamined: 1,
```

Results with index (sorting by brand)

11.2 Neo4j Indexes

Since every time we search for a user to display her/his relationships, we decided to add indexes on Neo4j too.

The first index is on the username for the User entity. From the image we can see that, without this index, searching a node by username would spend 240 ms, while after inserting the index we need only 27 ms.

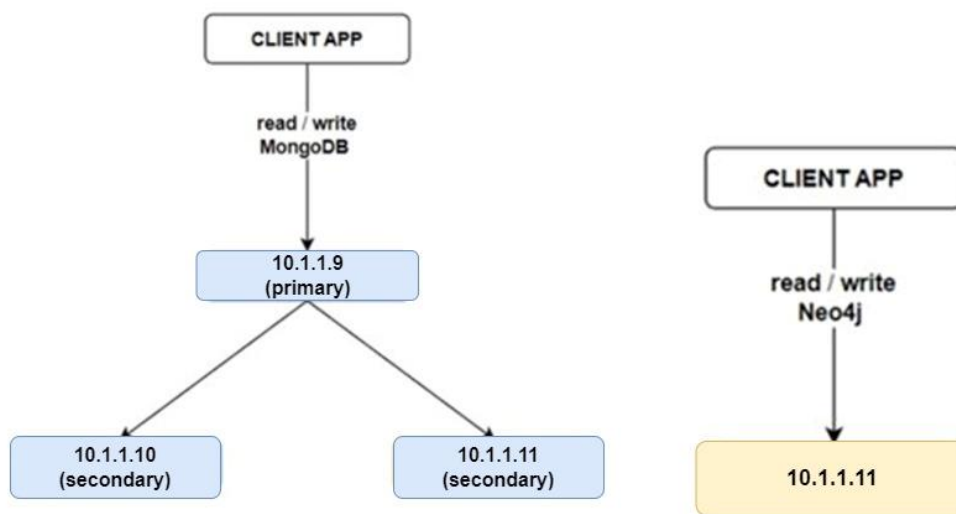
The screenshot shows the Neo4j Cypher query interface. The query is `rentaform$ MATCH (u:user {username:"beautifulfish950"}) RETURN u`. The results are displayed in a table view, showing a single record for a user with the following properties: `{ "identity": 1, "labels": ["user"], "properties": { "username": "beautifulfish950" }, "elementId": "1" }`. The status bar at the bottom indicates: "Started streaming 1 records after 1 ms and completed after 240 ms."

The screenshot shows the Neo4j Cypher query interface. The query is `rentaform$ MATCH (u:user {username:"beautifulfish950"})`. The results are displayed in a table view, showing a single record for a user with the following properties: `{ "identity": 1, "labels": ["user"], "properties": { "username": "beautifulfish950" }, "elementId": "1" }`. The status bar at the bottom indicates: "Started streaming 1 records after 20 ms and completed after 27 ms."

12 Replicas

One of the non-functional requirements is to ensure high-availability. To reach this goal we have designed the following replica set configuration:

- **MongoDB**, the replica set is composed of a primary replica, the latter takes client requests, and two secondaries which are the servers that keep copies of the primary's data.
- **Neo4j**, we have only one Neo4j instance .



The configuration is:

```
rsconf = {
  _id: "rs0",
  members: [
    { _id: 0, host: "10.1.1.9:27017" priority:5},
    { _id: 1, host: "10.1.1.10:27017" priority:2},
    { _id: 2, host: "10.1.1.11:27017" priority:1}}}
```

We put the lowest level of priority on 10.1.1.11 machine because it is the one containing the only instance of Neo4j.

Moreover, we decided to introduce both Write Concern and Read Preferences constraints. Various considerations have been made by reviewing the various collections to analyze the degree of

consistency required for the various operations. In general, having a greater number of writes before returning control to the application allows for low consistency in the event of a failure. In our application it was decided to put **w=1** for the Review operations. In this way the application regains control once the first copy is written, without waiting that the primary server propagates the write to the secondary ones. This situation could let to lose some data if the primary nodes crashes before propagating the writing and after acknowledging the write request to the application. We accept it because we want to advantage Availability over Consistency. About the write operation, we decide to set **retryWrites = true** in the connection options for MongoDB., so we allow MongoDB drivers to automatically retry write operations a single time if they encounter network errors, or if they cannot find a healthy primary in the replica sets.

While for the Reservation operations it was decided to have a greater number of writes because we cannot tolerate inconsistency since the reservations include important information such as the rental price.

Furthermore, we put the **readPreference = nearest** - We allow the MongoDB driver to read from any of the replica because of the non-functional requirements low latency and high availability. We accept the fact that data obtained from the MongoDB server could be not updated to the latest version.

13 SHARDING IDEA

To improve the availability of the system, we propose to adopt sharding concurrently to the data replication. We propose sharding for only MongoDB collections, because Neo4j didn't allow it.

- User Collection : we propose **Username** as sharding key because we retrieve the user document by username, that is always present. This sharding proposal doesn't affect the actual collection structure.
- Business Activity Collection : we propose **Name** as sharding key for the same reasons of above.
- Open_Notification Collection: we propose **Applicant** as sharding key that is the username of the user who open the notification, in this way we can be sure that all tickets and reports relating to a user are on the same server that owns the same user document.

13 CONSISTENCY

Database Consistency Management

Operations of creating review, deleting one of them, deleting user are operations that involve both databases. To ensure consistency between them, these are the operations performed.

```
public boolean removeReviewTransaction(ClientSession mongoSession, Session graphSession, Review reviewToDelete, String user){
    boolean response;
    mongoSession.startTransaction();
    Transaction graphTransaction = graphSession.beginTransaction();
    try{
        //Instant utcTime = reviewToDelete.getDateOfReview().toInstant().atOffset(ZoneOffset.UTC).toInstant();
        MongoCollection<User> userColl = mongoDriver.getUserCollection();
        MongoCollection<BusinessActivity> businessColl = mongoDriver.getBusinessActivityCollection();
        Bson businessFilter = eq( fieldName: "name", reviewToDelete.getBusinessActivity());
        Bson userFilter= eq( fieldName: "username",user);
        Bson businessDeleteReview = Updates.pull( fieldName: "reviews",new Document("username",user)/*.append("dateOfReview",utcTime)
        Bson userDeleteReview = Updates.pull( fieldName: "reviews",new Document("businessActivity",reviewToDelete.getBusinessActivity()
        businessColl.updateOne(mongoSession,businessFilter,businessDeleteReview);
        userColl.updateOne(mongoSession,userFilter,userDeleteReview);
        graphTransaction.run( s: "MATCH (uA:user) WHERE uA.username=$username" +
            " MATCH (uB:BusinessActivity) WHERE uB.name=$business" +
            " MATCH (uA)-[r:Review]->(uB) WHERE r.rating=$rating"+
            " DELETE r",
            parameters( ...keysAndValues: "username", user, "business", reviewToDelete.getBusinessActivity(),"rating",reviewToDelete
        mongoSession.commitTransaction();
        graphTransaction.commit();
        response = true;
    }catch(Exception e){
        mongoSession.abortTransaction();
        graphTransaction.rollback();
        return false;
    }

    return response;
}
```

```

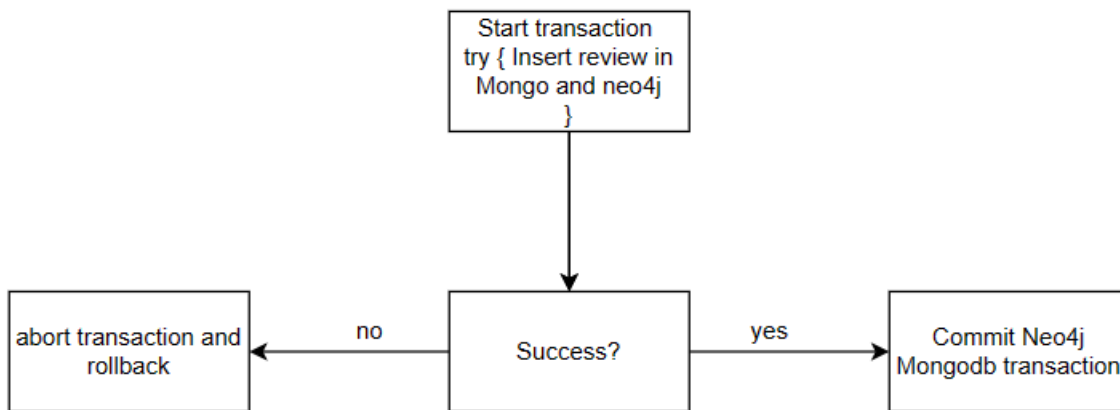
public Boolean addReviewTransaction(ClientSession mongoSession, Session graphSession, Review reviewToInsert, String user){
    mongoSession.startTransaction();
    boolean response;
    Transaction graphTransaction = graphSession.beginTransaction();
    try {
        MongoClient<User> userColl = mongoDriver.getUserCollection();
        MongoClient<BusinessActivity> businessColl = mongoDriver.getBusinessActivityCollection();

        UpdateResult insertReviewUser = userColl.updateOne(mongoSession, Filters.eq( fieldName: "username",user), Updates.push(
        String business = reviewToInsert.getBusinessActivity();
        reviewToInsert.setBusinessActivity(null);
        reviewToInsert.setUsername(user);
        UpdateResult insertReviewBusiness = businessColl.updateOne(mongoSession, Filters.eq( fieldName: "name",business),Updates

        ResultSummary graphResult = graphTransaction.run( s: "MERGE (uA:user { username:$username})"+
            "MERGE (uB: BusinessActivity { name:$business})"+
            "MERGE (uA)-[r:Review {rating:$rating}]->(uB)",
            parameters( ...keysAndValues: "username",user,"business",business,"rating",reviewToInsert.getRating()).consume();
        if(graphResult.counters().relationshipsCreated() == 1 && insertReviewBusiness.getModifiedCount() == 1 &&
            insertReviewUser.getMatchedCount() ==1 ){
            mongoSession.commitTransaction();
            graphTransaction.commit();
            response = true;
        }
        else{
            mongoSession.abortTransaction();
            graphTransaction.rollback();
            response = false;
        }
    }
    }catch(Exception e){
        mongoSession.abortTransaction();
        graphTransaction.rollback();
        response = false;
    }
    return response;
}

```


In both cases, the try-catch statement is used to perform the transaction. Transactions are initiated before the try-catch block. Inside it we define the operations to be performed in both databases. If the operations are successful, both transactions are committed, otherwise if one of the two fails, the abort and rollback are performed, in the same way if check for an error in the catch block, canceling any changes made.



14 Usage Manual

If a visitor wants to fully experience our application, the first thing he/her must do is to sign up. Here, he/she will be asked to insert some personal details, including First and Last Name, Country, etc. Whatever signup has been just done, or if it has been already done, you will be asked to login, inserting username and password provided during signup process.

Password

Name

Surname

Address

E-Mail

Phone Number

Date of birth

Submit

Signup form

Username

Password

Login

Login form

When logged in, homepage will be showed.

OUR ACTIVITIES

Look for the best service for your needs with RentaForm

Alphabetic ascending



Carjet

New Jersey

Carjet offers novelty vehicle rentals, such as convertibles, muscle cars, and classic cars. They provide a unique and fun rental experience for those looking to make a statement.

AVAILABLE VEHICLES»

Rating **4**



Chic Drive

Texas

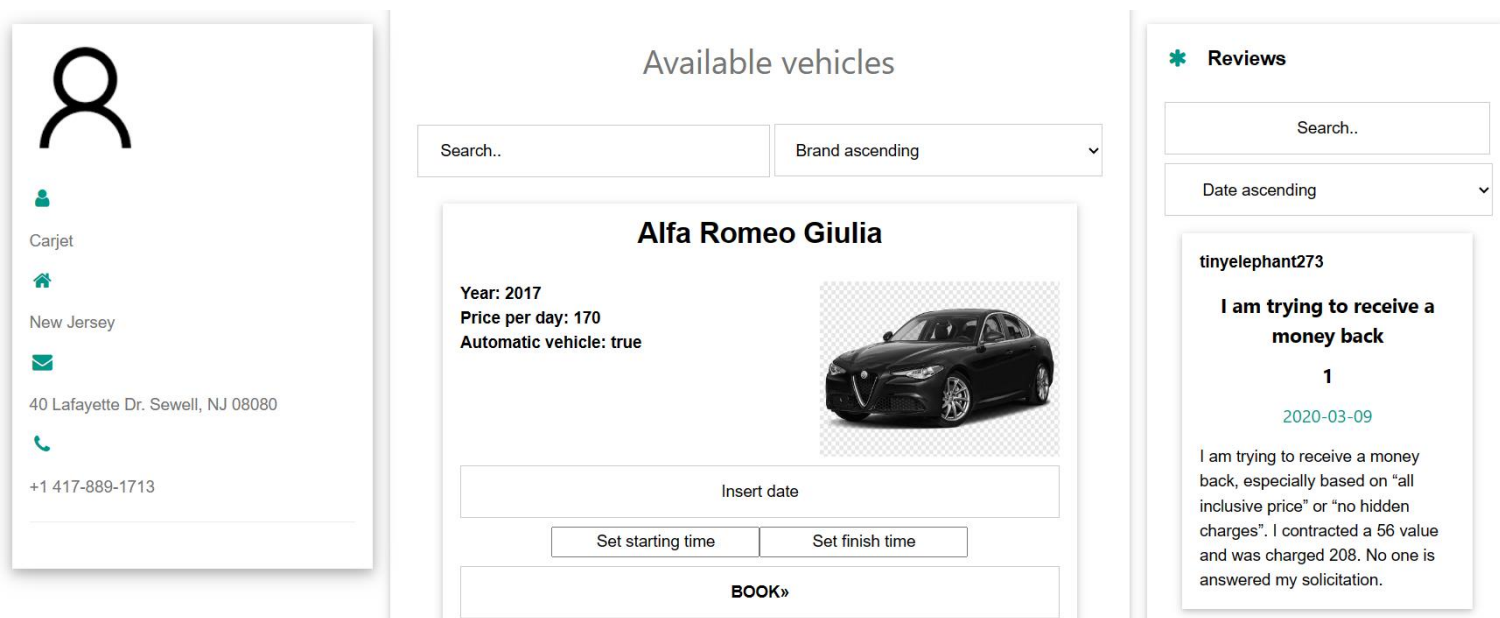
Get behind the wheel of one of our chic vehicles and enjoy a stylish and comfortable travel



Homepage

As we can see, on the homepage you can view the various businesses that offer the renting service. These activities have their own logo, the average rating calculated from the reviews of other users who

have already used their services and a brief description. Those assets can be searched by name or sorted by different parameters like name and rating.









BusinessActivity page

By clicking on available vehicles, you enter the business page where a more in-depth description is provided, such as telephone number, address, and city. Furthermore, the user has the possibility to view all the available vehicles, which can also be ordered in different ways, and by clicking on book and entering the vehicle booking dates it is possible to book it. If the vehicle is available on those dates, the booking will be successful otherwise it will be necessary to choose other dates. On the right instead we find all the reviews that have been carried out by users who have already booked these services, these too can be sorted by rating, date and description or searched directly.

An user can also navigate within his personal page, where the structure of the page is similar to those of the business ones, therefore you have the information that the user provided during registration, and it is possible to view both the reservations made by various companies. These can be eliminated, or the booking date can be modified if necessary. In the same way all the user's reviews can be modified or eliminated.

Moreover, the "friend" page shows all the users that the user follows and from which he is followed with the possibility of being able to unfollow them or to follow them in case he/she does not.




Léon

public employee

979RuePaulDuvivier

leon.francois@example.com

0779970233

ACTIVITIES

Search..

Recent First

Miami rent

Golf GTI

Total amount: 1280

2026-11-29 00:00 -- 2026-11-30 00:00

Write a review»

Modify»

Remove activity»

* Reviews

Search..

Oldest First

Miles Car Rental

5

2020-08-15

Miles Car Rental - USA is the Best!!!

Modify review»

Remove review»

User page

Date interval

2026-11-23

EndDate

2026-11-24

Starting Time

00:00

End Time

00:00

Submit

Modify Reservation form

Subject

Worst ever first experience

Review


For some reason the booking confirmation was 2 hours short of the time I booked. Given I have paid for full days (in fact wher

Rating

1

Submit

Modify review form



Gildo

freelance






3344RuaQuatro

gildo.ribeiro@example.com

(55)4824-0924

Followed Users

Search..Alfabetic ascending

	angryrabbit279
	angrytiger466
	beautifulfish950
	beautifulfrog492
	beautifulkoala187

Reviews

Search..Recent First

Good i like it

5

2020-12-28

Date of experience: January 26, 2023

Going round the houses to get refund of...






1

2020-12-24


Going round the houses to get refund of mis sold insurance. Still waiting.. Do better !!!!

Friend page

Followers

Search..	Alfabetic ascending	▼
	angrybutterfly383	UNFOLLOW»
	angrygorilla596	UNFOLLOW»
	beautifulostrich939	UNFOLLOW»
	blackgorilla250	UNFOLLOW»
	blackladybug956	UNFOLLOW»

Friend Page



Gildo






freelance

3344RuaQuatro

gildo.ribeiro@example.com

(55)4824-0924

Followed Users

Search..	Alfabetic ascending	▼
	angryrabbit279	
	angrytiger466	
	beautifulfish950	
	beautifulfrog492	
	beautifulkoala187	

Reviews

Search..

Recent First

▼

Good i like it

5

2020-12-28

Date of experience: January 26, 2023

Going round the houses to get refund of...

1

2020-12-24

Going round the houses to get refund of mis sold insurance. Still waiting.. Do better !!!!


Friend Page

Moreover, a user can view the various analytics previously mentioned by clicking on the appropriate button.

Finally, both the activities and admin can have access to their personal pages. Businesses can view their vehicles, edit them, and add new ones, view reviews, bookings and analytics, and they can also send reports to the admin or delete existing ones.

In the same way, the admin, in addition to all the user features, can ban users and businesses and replies to the reports/tickets of the latter.

REPORTS

Recent First 

2023-06-18

problem with this review

i have a problem with this review

OPEN

Modify report

Remove report

INSERT NEW VEHICLE

AutomaticTrasmission

Automatic



Brand

Enter Brand

Name

Enter Name

Year

Enter Year

Price

Look for the best service for your needs with RentaForm

Search..

Alphabetic ascending



Carjet

New Jersey

Carjet offers novelty vehicle rentals, such as convertibles, muscle cars, and classic cars. They provide a unique and fun rental experience for those looking to make a statement.

AVAILABLE VEHICLES»

Rating **4**



DELETE BUSINESS ACTIVITY: Carjet

Admin can delete Business Activity in Home Page

2023-06-18

GENERIC

Need Some Info

Hello i need some info about the Renting service and how i can reserve a vehicle


TAKEN

Answer Ticket


Close Ticket

In the notification section an admin can answer to a specific ticket or report or can close that ticket. If the ticket is closed or the admin has answered that ticket, Users can not modify anymore the ticket.


merchant



6145Ringstraße



carolin.brettschneider@example.com



0351-1977697

BAN USER:

FOLLOW

UNFOLLOW

When an admin goes into a User page besides that follow and unfollow that specific user he can even perform a ban operation which remove that user from the db. In that case even people who follow this user will delete this association.