
NOMAD ADVISOR APPLICATION

Task 2 – Large Scale and Multi-structured Databases



UNIVERSITÀ DI PISA

Eugenia Petrangeli, Daniela Comola, Carmelo Aparo, Leonardo Fontanelli

INDEX

INDEX	2
INTRODUCTION	3
<i>Description of the application</i>	3
<i>Functional requirements</i>	3
<i>Non-Functional requirements</i>	5
<i>Data pre-processing</i>	5
DESIGN.....	6
<i>UML diagram – Use Cases</i>	6
<i>UML diagram – Analysis Classes</i>	7
<i>Data model</i>	7
<i>Architecture of the platform and frameworks</i>	10
IMPLEMENTATION	10
<i>Authentication</i>	10
<i>Implementation Class Diagram</i>	12
TESTS	13
<i>Replica Tests</i>	13
<i>Speed Test</i>	16
USER MANUAL	18
<i>Login/Registration Interface</i>	18
<i>Customer Interface</i>	19
<i>Hotel Interface</i>	22
<i>Customer profile Interface</i>	25
<i>Employee Interface</i>	27
SHARDING PROPOSAL	36
CONCLUSIONS.....	37

Introduction

Description of the application

The system is an application that allows a customer to choose a city to visit and a hotel where he/she can stay overnight.

A customer can register and log in, after the log in he/she's able to consult a list of cities where he/she can see many details regarding the life in the city, monthly cost of living in dollars, nightlife and so on.

He/she can also filter the list of cities by selecting the preferred characteristic.

In addition, the application shows him/her a list of hotels where he/she can stay overnight. By selecting a hotel, it is possible to see the related reviews with rate and comments.

Moreover, he/she can add new reviews about hotels.

In this area he/she can also define some preferences about the characteristics of cities, and the system will use these data in order to show him/her the cities that fits with the selected characteristics.

The system has also an administration area where an employee can perform some management operations.

He/she can view all the registered customers and manage hotels and cities present in the system by selecting, adding, removing or updating them.

He can also see some statistics such as:

- A pie chart that shows the distribution of users' preferences about cities
- A pie chart that shows the distribution of characteristic of the cities that are stored in the system.

Thanks to these two statistics an Employee can understand the type of application user and check if the service offered reflects the request. Thanks to the latest chart an Employee can also see if there are few cities that offer certain services and possibly evaluate the expansion of the site in order to cover more characteristics and attract more users.

Functional requirements

Following are the requirements of the computer system.

The system provides a log-in/registration interface. In order to register to the system, a Customer must fill a form indicating Name, Surname, Password, Nickname and Email. These two last attributes must be unique in the system. It is assumed that only the Customer can register because the Employees are already registered.

In order to log-in to the system a Customer must fill a form indicating the email and the password chosen in the registration phase.

The system provides a customer interface that allows to search the cities. It's possible to search by specifying one or more of these filters: Temperature, Air Quality, Quality of Life, Friendly, Healthcare, Nightlife, Cost, Safety, Walkability, Free WiFi and English Speaking. Each of these values have a score from one to five except Temperature and Cost. The system searches the cities having these values greater than the values inserted by the Customer (except for Temperature and Cost).

Temperature has four possible ranges expressed in Celsius: lower than 0, from 0 to 15, from 15 to 25 and greater than 25. Cost has a value expressed in dollars and the system search the cities having a monthly cost lower than the value specified by the customer. The system shows the first 30 cities.

At the beginning the system retrieves from the logged customer the preferences he/she inserted in his/her profile and shows the best 30 cities that satisfy the Customer preferences. In particular, the preferences refer to the city's characteristics and the system searches the cities in which those features have a score greater than three except for Temperature and Cost. Search cities where the temperature is from 15 to 25 degrees Celsius and the monthly cost is less than 2000\$.

For each city the system shows in a table the Name, the State and the set of attributes listed in the filters before. A Customer can select the city that he/she prefers, and he/she can view the hotels related to it.

In relation to the last action, the system provides an interface in which the Customer can see a table with all the hotels with their information: Name, Address and Website. This last attribute is not always available data. By selecting one of them the system provides the first 30 related reviews from the most recent one with their attributes: Username, Rating, Comment and Date. Moreover, he/she can add a review by selecting a mark and writing a comment.

The system allows to go back to the interface with the cities information so that he/she can start a new research.

The system provides the possibility to create a customized profile.

The customer can visit his/her private area, containing his/her personal information, both from Cities and Hotels Interfaces.

In this area, a customer can set/update his/her city's preferences: Temperature, Air Quality, Quality of Life, Friendly, Healthcare, Nightlife, Cost, Safety, Walkability, Free Wi-Fi and English Speaking. For each of them the Customer can tick what he/she prefers.

The system allows to go back to the interface with the cities information so that he can start a new research.

The system provides also an Employee Interface where only him/her is able to access. Here, only he/she can see all the registered customers and the existing cities and hotels in the system. In particular, he/she can see a table showing the customers personal information: Name, Surname, Username and Email.

He/she can also see a table with cities information: Name, State and Characteristics as Temperature, Air Quality, Quality of Life, Friendly, Healthcare, Nightlife, Cost, Safety, Walkability, Free Wi-Fi and English Speaking.

The system provides a form in order to add new cities or update the existing ones. In this form, in order to add a new city, the Employee shall specify all the related fields: Name, State, Temperature, Air Quality, Quality of Life, Friendly, Healthcare, Nightlife, Cost, Safety, Walkability, Free Wi-Fi and English Speaking. We assume that the pair "Name-State" of the city is unique.

When a pair of these attributes refers to an already existing one, the inserted fields in the form will be used to update the previous ones.

Alternatively, he/she can also choose to only update a city in the system. In this case, the Employee shall specify Name and State of the interested city and fill only the fields of the Characteristics that he/she wants to change.

Choosing a city, he can view all the related hotels in a table showing their information: Name, Address and Website. These two last attributes are not always available data.

The system provides a form in order to add new hotels in the selected city. In this form the Employee shall specify all the related fields: Name, Address and Website. We assume that the name of the hotel for a certain city is unique. When a hotel's name refers to an already existing name, the inserted fields in the form will be used to update the previous one.

By selecting a row in the Cities or Hotels tables, he/she can delete one of them.

The system performs some statistics on the available data. In particular, allows to the Employees to view all the customer's preferences in percentage thanks to a Pie Chart. The system also allows to view by a Pie Chart the number of cities that have a specific characteristic in percentage, only the cities with an attribute's score greater than three are considered in this counting except for the Temperature and Cost. It considers cities where the temperature is from 15 to 25 degrees Celsius and the monthly cost is less than 2000\$.

All the described interfaces allow to log out and come back to the login/registration interface.

Non-Functional requirements

The system is designed in order to be user friendly; it has a graphical interface that is simple and intuitive and guide the user to make the right choice by showing also messages in case of errors.

The system manages the passwords in a secure way because it encrypts the passwords using SHA-1 when the user inserts them, so the passwords are sent to the DB already encrypted. This is done in order to prevent sniffing and to store them already encrypted.

In addition, the system uses a specific user to connect to the DB, in this way the application has limited privileges on the DB in particular it can only modify the specific DB dedicated for the application.

Since the system uses a replica set to guarantee high availability, the database is configured to use an internal authentication protocol among replica servers to have a secure communication among them.

We retrieved the city's data from the website <https://nomadlist.com/> by scraping. Before doing this the Term of Service of the site has been read in order to be sure that it was allowed.

Moreover the file <https://nomadlist.com/robots.txt> is empty, this imply that is allowed to do scraping/crawling.

Data pre-processing

The datasets used to produce the data for the application are:

<https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe>

<https://www.kaggle.com/datafiniti/hotel-reviews>

These are two datasets about hotel reviews. The first one contains 515k hotel reviews in the most important Europe cities, the second one contains 20k hotel reviews in different cities around the USA.

We realized a Python script to extract the data on which we were interested and to uniform the format of the two datasets.

So, we produced a JSON file for each collection that contains only the data useful for the application using the chosen format.

Then for each city in the dataset, we added information by scraping the website

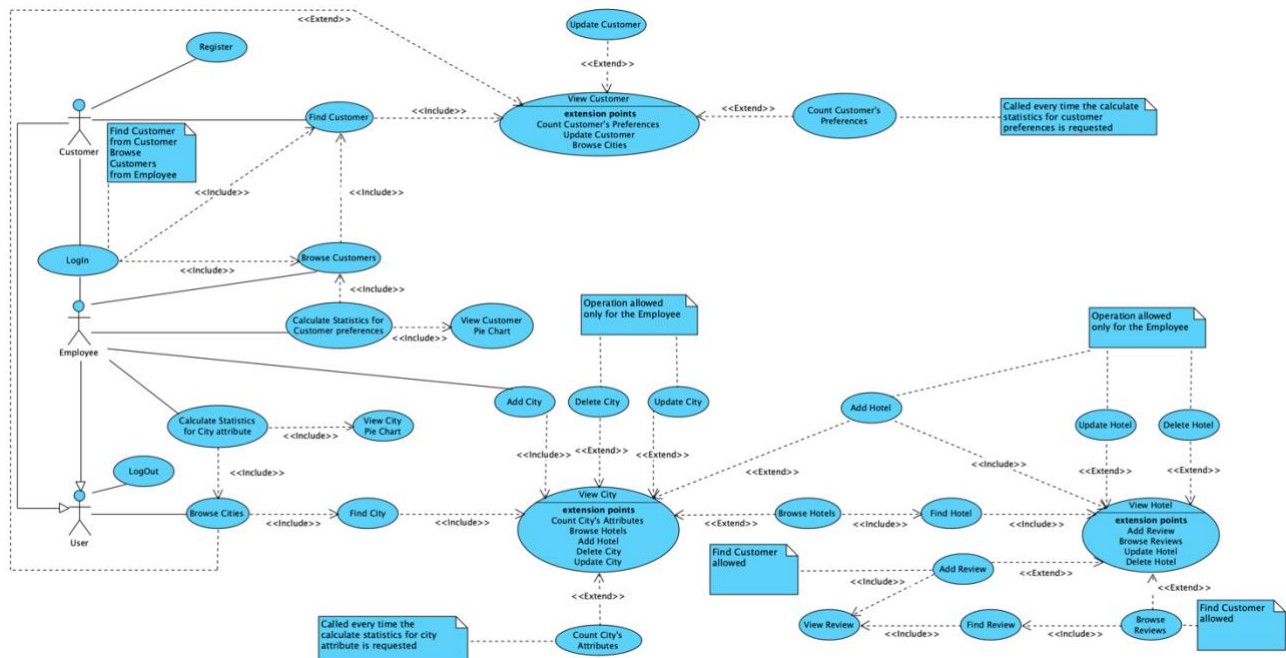
<https://nomadlist.com/> that provides a lot of information regarding the most important cities in the world.

The scraping activity has been carried out by implementing a Python script with BeautifulSoup framework that allowed to parse the HTML documents.

The downloaded page contains the data on which we were interested on a table, so we looked for the “<tr>” tag to retrieve them.

Design

UML diagram – Use Cases



The main actors of the system are the Employee and the Customer.

Both ones can:

- Log in to the system
- Log out
- Browse the cities
- Browse the hotels related to a City

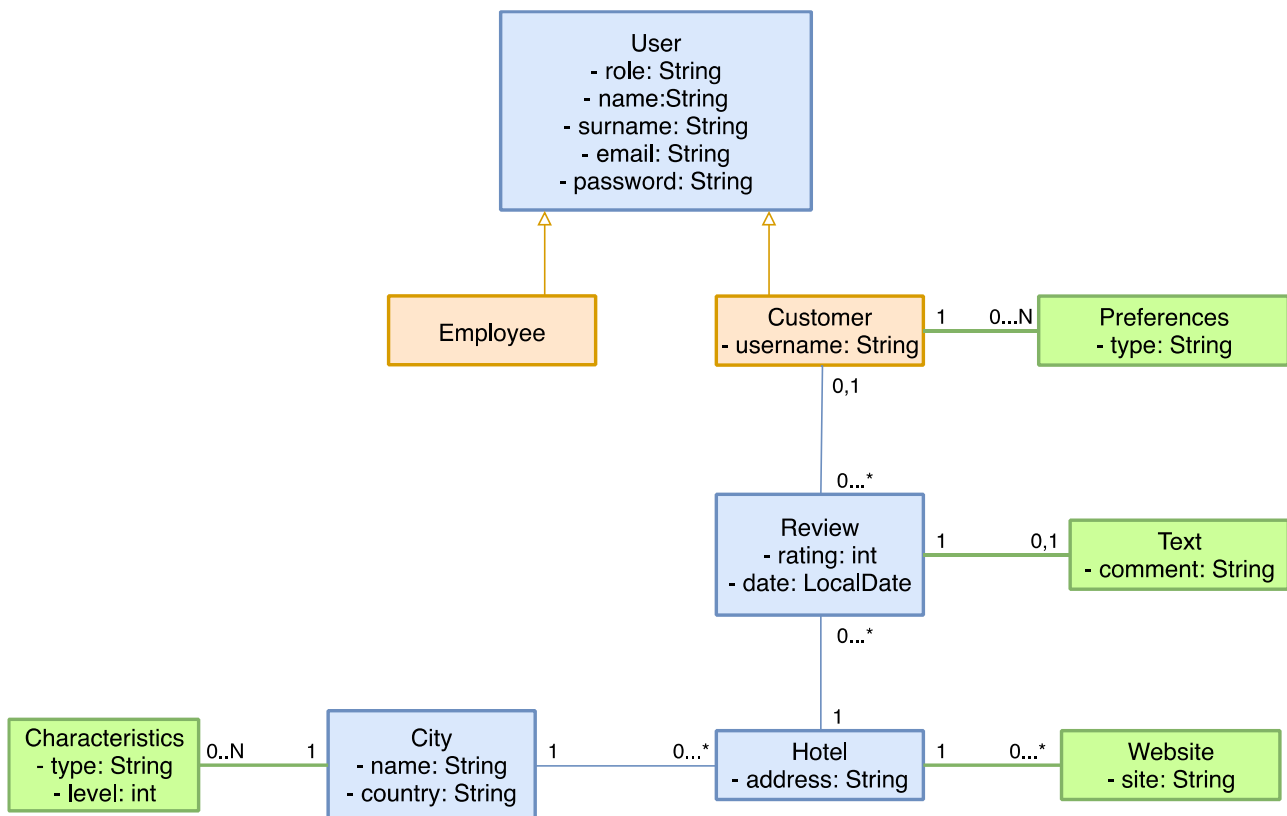
The Customer can:

- Register
- Find his/her personal data
- Browse the reviews
- Add a new review

The Employee can:

- Browse all the registered Customers
- View the Pie Chart related to Customers statistics and to Cities statistics
- Delete a Hotel
- Add/Update a Hotel
- Delete a City
- Add/Update a City

UML diagram – Analysis Classes



The User class has a one to many relationship with the Review class, because each user can write many reviews about his/her experience with an Hotel but each Review is associated with only one user.

The Hotel class has a one to many relationship with the Review class because each Hotel can have zero or many reviews, but each review is related only to one Hotel.

The City class has a one to many relationship with the Hotel class because each City can have zero or many hotels, but each Hotel belong only to one City.

Data model

In order to store the record related to the application we used MongoDB, that is a NoSQL Document database.

The followings are the models of the documents:

- City's document:

```
{
  "_id": {
    "city": "Amsterdam",
    "country": "Netherlands"
  }
  "cost": 5297,
  "temperature": 11,
  "airQuality": 4,
  "safety": 4,
```

```

    "qualityLife": 4,
    "walkability": 5,
    "healthcare": 4,
    "nightlife": 4,
    "wifi": 4,
    "foreignersFriendly": 4,
    "english": 4
  }

```

- Hotel's document

```

{
  "_id": {
    "name": "Hyatt House Seattle/Downtown",
    "city": "Seattle",
    "country": "United States"
  },
  "address": "201 5th Ave N",
  "websites": "http://seattledowntown.house.hyatt.com/en/hotel/home.html"
}

```

It is important to specify that the website field, in our dataset, is not available for every hotel, so it will be stored and consequently shown to the customers whenever is present in the hotel's information.

- Review's document:

```

{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "username": "GGTravels2016",
  "rating": 1,
  "text": "Good vacation!",
  "date": "2016-05-16",
  "hotelId": {
    "name": "21c Museum Hotel Lexington",
    "city": "Lexington",
    "country": "United States"
  }
}

```

The MongoDB driver automatically generates an ObjectId for the "_id" field.

In this case is important to specify that the username attribute is not present in every document of the collection. This is because, the European dataset from which we have taken the data to store in the database, did not have the username field for the reviews.

- User's document:

```

{
  "_id": ObjectId("507f1f77bcf86cd799439012"),
  "role": "customer",
  "name": "Mario",
  "surname": "Rossi",
}

```



```
"email": "tatsurok2018@gmail.com",  
"password": "tatsurok2018",  
"username": "tatsurok2018",  
"preferences": ["walkability"]  
}
```

The MongoDB driver automatically generates an ObjectId for the "_id" field.

The role can assume the values "customer" or "employee" based on the user's role.

In this case is important to specify that the "name" and "surname" attributes is not present in every document of the collection. This is because, the dataset from which we have taken the data to store in the database, did not have these fields. They will be added for new registration in the application.

If the user is a customer in the document is present also "username" field and could be present also "preferences" field.

In this last one will be stored the attributes of the city that have been chosen by the customer in his personal page, and this will be helpful in order to retrieve the statistic about the users' preferences.

An advantage in using the Document Database is the possibility to use nested document.

We could put all in the city's collection, but the dimension of the document could be too big. We could solve this problem by using two collections, one that contains city documents with nested hotel documents and the other one with the review documents. We didn't do it because once a customer selects the desired city, the system shows the list regarding the hotels that are present in that city and consequently, selecting a certain hotel, shows the reviews related to it, so our application doesn't need all the information about cities, hotels and reviews together.

The only case in which the application could gain the advantage in having the nested documents is when a delete of a city or a hotel occurs. For example, if the application has to delete a city, it deletes both the city and all the related hotels and reviews, so if the hotels and reviews would be nested in the city document this operation would be easier. Since this operation doesn't occur frequently, is not convenient.

MongoDB offers the possibility to build aggregation pipelines, that process data records and return computed results, grouping values from multiple documents together.

In our case 4 types of aggregation have been defined:

- Aggregation that allows to retrieve the number of cities that have a specific characteristic's score greater than three.
- Aggregation that allows to retrieve the number of cities that have a specific temperature's value from 15 to 25 degrees Celsius.
- Aggregation that allows to retrieve the number of cities that have a specific monthly cost's value less than 2000\$.
- Aggregation that allows to count customers that have chosen a certain characteristic.

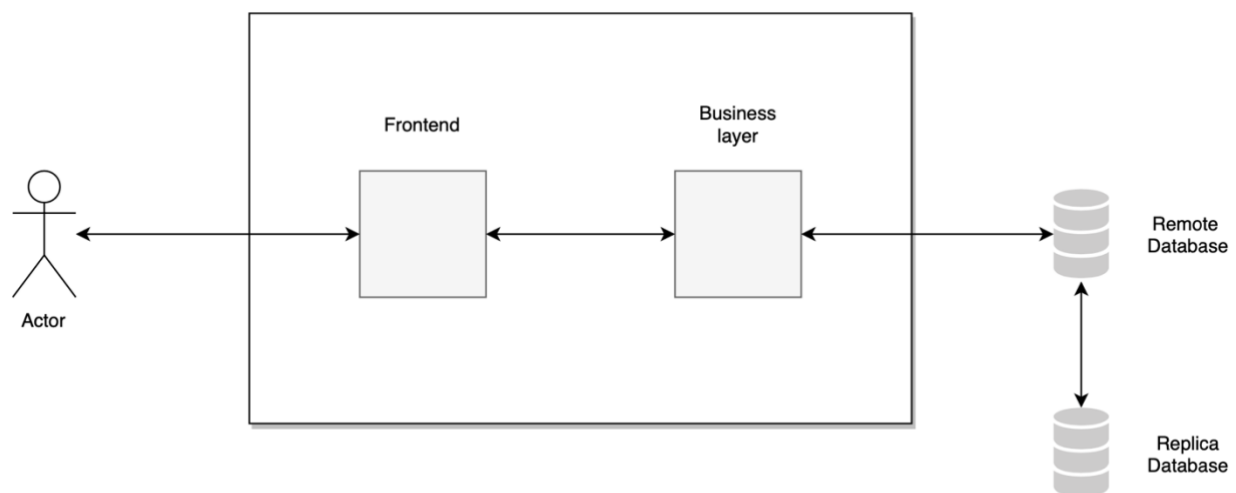
Another useful functionality is the index, that supports the efficient execution of queries in MongoDB. In our case we used 5 indexes in order to improve the performance of the queries.

- In the hotel's collection, we used the index on the city's name, in order to order the document by the city's attribute.
- In the Review's collection, we used the Compound index that refers to the name of the hotel and to the date and order the reviews by hotels and then by date.

- In the Review's collection, we used the TTL index in order to maintain only the documents of the last four years.
- On the Customer collection we used the unique index on email field in order to have only one occurrence of this attribute in the collection.
- On the Customer collection we used the unique index on username field in order to have only one occurrence of this attribute in the collection.

Architecture of the platform and frameworks

The system is composed by a client application which is divided into a front-end and a business layer.



The last one is connected to a remote NoSQL document database, MongoDB, where are stored all the records of the application.

The architecture of the client is implemented in JAVA.

The frontend is composed by the graphic interfaces for both Customers and Employees.

Those classes are responsible of taking the inputs of the users for all the operations permitted and they send the data to the business layer. When the customer inserts the credentials, the password is encrypted with SHA-1 algorithm.

The business layer is the one that satisfies the requests coming from the interface using MongoDB JAVA Driver to perform queries on the remote database. At the end of the operations, it returns the result to the graphic interface.

A Replica Database is defined in order to ensure high availability of the application with automatic failover and data redundancy.

Implementation

Authentication

As said before, a non-functional requirement for this system is to guarantee a secure communication between the application and the DBMS.

In addition, the database of the application uses replicas to guarantee high availability, so there is also an exchange of messages among replica servers. For this reason, we configured MongoDB to use an internal authentication protocol among them.

We generated a complex pseudo-random 1024 character string by using *openssl* libraries that is used as shared password among replica set.

In order to enable this authentication mechanism, each server has to store a file that contains the shared key and the *mongod* process of each server is started by adding the *-keyFile* option that specifies the path of the file.

In this way only the servers that authenticate themselves with this password can exchange information.

In order to enable an authentication protocol between the application and the DBMS, we created an administrator user to manage all the operations that require administration privileges and a user that has privileges only to read and write the application's collections.

The last user is the one used by the system in order to limit the privileges of the application on the database.

Implementation Class Diagram

```
classDiagram
    class NomadHandler {
        +readUser(user : User, msg : StringBuilder) : User
        +createCustomer(customer : Customer) : String
        +getCustomers() : List<Customer>
        +createCityAddedCity : City : String
        +updateCityUpdatingCity : City : String
        +deleteCitySelectedCity : City : String
        +deleteHotel(hotel : Hotel) : String
        +getCity(pref : HashMap<String, Integer>) : List<City>
        +getCity(cityName : String) : List<City>
        +addReview(review : Review) : boolean
        +getReviews(hotel : Hotel) : List<Review>
        +getHotels(city : City) : List<Hotel>
        +updatePreferences(customer : Customer, preferences : List<String>) : String
        +createHotel(name : String, city : String, country : String, address : String, website : String) : String
        +computePieChartsData() : List<HashMap<String, Integer>>
        +openConnection() : void
        +closeConnection() : void
    }
```

```
classDiagram
    class MongoDBHandle {
        -mongoClient : MongoClient
        -database : MongoDatabase
        -userCollection : MongoCollection<Document>
        -cityCollection : MongoCollection<Document>
        -hotelCollection : MongoCollection<Document>
        -reviewCollection : MongoCollection<Document>
        +openConnection() : void
        +finish() : void
        +readUser(user : User, msg : StringBuilder) : User
        +createCustomer(customer : Customer) : int
        +selectCities() : List<City>
        +selectCities(pref : HashMap<String, Integer>) : List<City>
        +selectCities(name : String) : List<City>
        ~buildCityId : Document : City
        +selectHotels(city : String, country : String) : List<Hotel>
        +selectReviews(hotelName : String, city : String, country : String) : List<Review>
        +createReview(review : Review) : boolean
        +updatePreferences(customer : Customer) : boolean
        +selectCustomers() : List<Customer>
        +deleteCity(city : City) : boolean
        +deleteHotel(hotelName : String, cityName : String, country : String) : boolean
        +createCityAdded : City : int
        +updateCity(city : City) : int
        +deleteHotel(hotel : Hotel) : boolean
        +createHotel(hotel : Hotel) : int
        +updateHotel(hotel : Hotel) : int
        +aggregateCustomersPreferences() : HashMap<String, Integer>
        +aggregateCitiesCharacteristics() : HashMap<String, Integer>
    }
```

```
classDiagram
    class CityNames {
        <<enumerations>>
        TEMPERATURE
        COST
        AIR_QUALITY
        SAFETY
        QUALITY_OF_LIFE
        WALKABILITY
        HEALTHCARE
        NIGHTLIFE
        WIFI
        FOREIGNERS
        ENGLISH
    }
```

```
classDiagram
    class Review {
        -username : SimpleStringProperty
        -rating : SimpleIntegerProperty
        -text : SimpleStringProperty
        -date : SimpleObjectProperty<LocalDate>
        -hotelName : SimpleStringProperty
        -cityName : SimpleStringProperty
        -countryName : SimpleStringProperty
        +Review()
        +Review(username : String, rating : int, text : String, date : LocalDate, hotelName : String, cityName : String, countryName : String)
        +getUsername() : String
        +setUsername(username : String) : void
        +getRating() : int
        +ratingProperty() : SimpleIntegerProperty
        +setRating(rating : int) : void
        +getText() : String
        +textProperty() : SimpleStringProperty
        +setText(text : String) : void
        +getDate() : LocalDate
        +dateProperty() : SimpleObjectProperty<LocalDate>
        +setDate(date : LocalDate) : void
        +getHotelName() : String
        +hotelNameProperty() : SimpleStringProperty
        +setHotelName(hotelName : String) : void
        +getCityName() : String
        +cityNameProperty() : SimpleStringProperty
        +setCityName(cityName : String) : void
        +getCountryName() : String
        +countryNameProperty() : SimpleStringProperty
        +setCountryName(countryName : String) : void
    }
```

```
classDiagram
    class DBConnection {
        +mongoClient : MongoClient
        <<Property>>
        -DBConnection()
    }
```

```
classDiagram
    class Utils {
        +cityAttributes : HashMap<CityNames, String> = new HashMap()
        +WRAPPING_CELL_FACTORY : Callback<TableColumn<City, String>, TableCell<City, String>>
        +id : String = "id"
        +CITY : String = "city"
        +COUNTRY : String = "country"
        +PREFERENCES : String = "preferences"
        +encryptPwd(pwd : String) : String
        +printUser(user : User) : void
    }
```

```
classDiagram
    class CityInterface {
        +getCity() : City
        +getCityId() : String
        +getCityName() : String
        +getCityCountry() : String
        +getCityAddress() : String
        +getCityWebsite() : String
        +getCityTemperature() : Integer
        +getCityCost() : Integer
        +getCityAirQuality() : Integer
        +getCitySafety() : Integer
        +getCityQualityOfLife() : Integer
        +getCityWalkability() : Integer
        +getCityHealthcare() : Integer
        +getCityNightLife() : Integer
        +getCityWifi() : Integer
        +getCityForeigners() : Integer
        +getCityEnglish() : Integer
    }
```

```
classDiagram
    class City {
        -cityName : SimpleStringProperty
        -cityId : SimpleIntegerProperty
        -cityCountry : SimpleStringProperty
        -cityAddress : SimpleStringProperty
        -cityWebsite : SimpleStringProperty
        -cityTemperature : SimpleIntegerProperty
        -cityCost : SimpleIntegerProperty
        -cityAirQuality : SimpleIntegerProperty
        -citySafety : SimpleIntegerProperty
        -cityQualityOfLife : SimpleIntegerProperty
        -cityWalkability : SimpleIntegerProperty
        -cityHealthcare : SimpleIntegerProperty
        -cityNightLife : SimpleIntegerProperty
        -cityWifi : SimpleIntegerProperty
        -cityForeigners : SimpleIntegerProperty
        -cityEnglish : SimpleIntegerProperty
    }
```

```
classDiagram
    class HotelInterface {
        +getHotel() : Hotel
        +getHotelId() : String
        +getHotelName() : String
        +getHotelCity() : String
        +getHotelCountry() : String
        +getHotelAddress() : String
        +getHotelWebsite() : String
        +getHotelTemperature() : Integer
        +getHotelCost() : Integer
        +getHotelAirQuality() : Integer
        +getHotelSafety() : Integer
        +getHotelQualityOfLife() : Integer
        +getHotelWalkability() : Integer
        +getHotelHealthcare() : Integer
        +getHotelNightLife() : Integer
        +getHotelWifi() : Integer
        +getHotelForeigners() : Integer
        +getHotelEnglish() : Integer
    }
```

```
classDiagram
    class Hotel {
        -hotelName : SimpleStringProperty
        -hotelId : SimpleIntegerProperty
        -hotelCity : SimpleStringProperty
        -hotelCountry : SimpleStringProperty
        -hotelAddress : SimpleStringProperty
        -hotelWebsite : SimpleStringProperty
        -hotelTemperature : SimpleIntegerProperty
        -hotelCost : SimpleIntegerProperty
        -hotelAirQuality : SimpleIntegerProperty
        -hotelSafety : SimpleIntegerProperty
        -hotelQualityOfLife : SimpleIntegerProperty
        -hotelWalkability : SimpleIntegerProperty
        -hotelHealthcare : SimpleIntegerProperty
        -hotelNightLife : SimpleIntegerProperty
        -hotelWifi : SimpleIntegerProperty
        -hotelForeigners : SimpleIntegerProperty
        -hotelEnglish : SimpleIntegerProperty
    }
```

```
classDiagram
    class ReviewInterface {
        +getReview() : Review
        +getReviewId() : String
        +getReviewUsername() : String
        +getReviewRating() : Integer
        +getReviewText() : String
        +getReviewDate() : LocalDate
        +getReviewHotelName() : String
        +getReviewCityName() : String
        +getReviewCountryName() : String
    }
```

```
classDiagram
    class Review {
        -reviewUsername : SimpleStringProperty
        -reviewRating : SimpleIntegerProperty
        -reviewText : SimpleStringProperty
        -reviewDate : SimpleObjectProperty<LocalDate>
        -reviewHotelName : SimpleStringProperty
        -reviewCityName : SimpleStringProperty
        -reviewCountryName : SimpleStringProperty
    }
```

```
classDiagram
    class CustomerInterface {
        +getCustomer() : Customer
        +getCustomerId() : String
        +getCustomerUsername() : String
        +getCustomerPreferences() : List<String>
    }
```

```
classDiagram
    class Customer {
        -customerUsername : SimpleStringProperty
        -customerId : SimpleIntegerProperty
        -customerPreferences : List<String>
    }
```

- **LoginInterface, CityInterface, HotelInterface, PersonalAreaInterface, EmployeeInterface, CityFormController, HotelFormInterface, StatisticsInterface:** These classes are the controllers for the related FXML files. So, they handle every event related to the interfaces.
- **City, Customer, Employee, Hotel, Review, User:** These classes are the entities related to the collections in the DB.
- **NomadAdvisor:** Contains the main method used to run the application (start() method) and also the method that implements the change of the scene which can be requested by the user (changeScene()).
- **NomadHandler, MongoDBHandle:** The purpose of these classes is to handle the operation to the DB. NomadHandler takes the requests from the interfaces and forwards them to MongoDBHandle, which is directly connected to the Database through MongoDB Java Driver.
- **DBConnection:** This is a singleton class which is responsible of creating the connection to the Database

Tests

Replica Tests

Our replica set is composed by a primary, the server that takes client requests, and two secondaries, the servers that keep copies of the primary's data. If the primary crashes, the secondaries can elect a new primary from amongst themselves. In this way the data are still accessible because we have one copy in each replica. In our application we have a three-node replica set on a single machine for practical reasons, usually it's better to allocate the replicas in dedicated hosts in order to avoid resource contention and provide isolation against server failure.

We have three separated data directories, one for each node and we have three separate mongod processes running on one machine.

Each member of the set is able to communicate with the other members and they have three different ports in which they listen: 27017, 27018 and 27019.

Secondaries may fall behind the primary (or lag) and not have the most current writes, so secondaries will refuse read requests by default to prevent applications from accidentally reading stale data.

The writes are replicated to all the majority of the replica set, but they are acknowledged by the primary server without waiting the propagation of the writes to the majority of the replica set, so we choose to don't use the write concern. The advantages of doing this is that the application will be faster because it won't have to wait the replication of the write operation to the majority of the members of the replica set.

It may happen in rare circumstance where the primary goes down and the newly elected primary did not replicate the very last writes to the former primary, those writes will be rolled back when the former primary comes back up. This can cause information loss but, in our application, there aren't critical writes, so having a small number of writes rolled back is not a problem.

The thing we have to take in mind is that we need a majority of members to elect a primary and it can stay primary as long as it can reach a majority, and a write is safe when it's been replicated to a majority. The majority is the "more than half of all members in the set". In our case we have three members in the set, so the majority is two.

Below we can see an example of tests on the functioning of replicas.

When the replica set is started, we can see which of these has been chosen as primary, in this case the one on the port 27019:

```
NomadAdvisor:PRIMARY> rs.isMaster()
{
  "hosts" : [
    "localhost:27017",
    "localhost:27018",
    "localhost:27019"
  ],
  "setName" : "NomadAdvisor",
  "setVersion" : 1,
  "ismaster" : true,
  "secondary" : false,
  "primary" : "localhost:27019",
  "me" : "localhost:27019",
  "electionId" : ObjectId("7fffffff000000000000000011"),
  "lastWrite" : {
    "opTime" : {
      "ts" : Timestamp(1577619738, 1),
      "t" : NumberLong(17)
    },
    "lastWriteDate" : ISODate("2019-12-29T11:42:18Z"),
    "majorityOpTime" : {
      "ts" : Timestamp(1577619738, 1),
      "t" : NumberLong(17)
    },
    "majorityWriteDate" : ISODate("2019-12-29T11:42:18Z")
  },
  "maxBsonObjectSize" : 16777216,
  "maxMessageSizeBytes" : 48000000,
  "maxWriteBatchSize" : 100000,
  "localTime" : ISODate("2019-12-29T11:42:23.372Z"),
  "logicalSessionTimeoutMinutes" : 30,
  "connectionId" : 41,
  "minWireVersion" : 0,
  "maxWireVersion" : 8,
  "readOnly" : false,
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1577619738, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  },
  "operationTime" : Timestamp(1577619738, 1)
}
```

If we shut down the replica on port 27019 a new primary is elected, in this case the one on the port 27017:

```

NomadAdvisor:PRIMARY> rs.isMaster()
{
  "hosts" : [
    "localhost:27017",
    "localhost:27018",
    "localhost:27019"
  ],
  "setName" : "NomadAdvisor",
  "setVersion" : 1,
  "ismaster" : true,
  "secondary" : false,
  "primary" : "localhost:27017",
  "me" : "localhost:27017",
  "electionId" : ObjectId("7fffffff000000000000000012"),
  "lastWrite" : {
    "opTime" : {
      "ts" : Timestamp(1577619856, 1),
      "t" : NumberLong(18)
    },
    "lastWriteDate" : ISODate("2019-12-29T11:44:16Z"),
    "majorityOpTime" : {
      "ts" : Timestamp(1577619856, 1),
      "t" : NumberLong(18)
    },
    "majorityWriteDate" : ISODate("2019-12-29T11:44:16Z")
  },
  "maxBsonObjectSize" : 16777216,
  "maxMessageSizeBytes" : 48000000,
  "maxWriteBatchSize" : 100000,
  "localTime" : ISODate("2019-12-29T11:44:17.205Z"),
  "logicalSessionTimeoutMinutes" : 30,
  "connectionId" : 29,
  "minWireVersion" : 0,
  "maxWireVersion" : 8,
  "readOnly" : false,
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1577619856, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  },
  "operationTime" : Timestamp(1577619856, 1)
}

```

Even in the application side we can see when the communication with the server on port 27019 is lost and a new connection with the one on port 27017 is established. In this way the application can continue to work:

```

INFORMAZIONI: Exception in monitor thread while connecting to server localhost:27019
com.mongodb.MongoSocketOpenException: Exception opening socket
    at com.mongodb.internal.connection.SocketStream.open(SocketStream.java:70)
    at com.mongodb.internal.connection.InternalStreamConnection.open(InternalStreamConnection.java:128)
    at com.mongodb.internal.connection.DefaultServerMonitor$ServerMonitorRunnable.run(DefaultServerMonitor.java:131)
    at java.lang.Thread.run(Thread.java:748)
Caused by: java.net.ConnectException: Connection refused (Connection refused)
    at java.net.PlainSocketImpl.socketConnect(Native Method)
    at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
    at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
    at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
    at java.net.Socket.connect(Socket.java:589)
    at com.mongodb.internal.connection.SocketStreamHelper.initialize(SocketStreamHelper.java:64)
    at com.mongodb.internal.connection.SocketStream.initializeSocket(SocketStream.java:79)
    at com.mongodb.internal.connection.SocketStream.open(SocketStream.java:65)
    ... 3 more

dic 29, 2019 5:25:44 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Discovered replica set primary localhost:27017
dic 29, 2019 5:25:44 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Rediscovering type of existing primary localhost:27019
dic 29, 2019 5:25:52 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Opened connection [connectionId{localValue:24, serverValue:13}] to localhost:27017

```


Speed Test

The system is read oriented, so since most of the operations are read ones, we decided to use indexes in order to speed up the application.

We performed some test in order to measure the speed improvement obtained by using indexes. The heaviest operations in the system are when the application has to find the hotels in a city and to find the reviews for a hotel.

In order to perform these tests, we used the *explain()* method of the cursor object of MongoDB which allows to execute a query and to show some statistics about the execution such as the execution milliseconds, the number of scanned keys and documents.

Find hotels in a city

The reference query on which we performed the first test is the following one:

```
db.hotel.find({$and: [{"_id.city": "Milan"}, {"_id.country": "Italy"}]}).limit(30).explain("executionStats")
```

on which the system finds all the hotels in Milan taking only 30 documents in order to limit the number of documents retrieved.

First of all, we executed the query without using indexes and the result was the following one:

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 30,
  "executionTimeMillis" : 5,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 1181
```

As we can see the execution time is 5 ms, the number of keys examined is 0 because there are no indexes that corresponds to the search filters and the number of scanned documents is 1181.

Since the number of scanned documents is high, we introduced an index that sorts the hotels with respect to the name of the city. In this way all the hotels that belongs to the same city are grouped together.

Executing the query using this index, we obtained the following result:

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 30,
  "executionTimeMillis" : 2,
  "totalKeysExamined" : 30,
  "totalDocsExamined" : 30,
```

Now the execution time is reduced to 2 ms, the number of keys examined is 30 because the hotels are searched in the database using the index in the city name and the number of scanned documents is reduced from 1181 to exactly 30 documents (the only ones that are retrieved).

Find reviews for a hotel

The reference query on which we performed the first test is the following one:


```
db.review.find({ $and: [{"hotelId.name": "Hotel Galileo"}, {"hotelId.city": "Milan"}, {"hotelId.country": "Italy"}]}).sort({date: -1}).limit(30).explain("executionStats")
```

on which the system finds the reviews for a hotel by using the name of the hotel, its city and country. As the application does during the execution, the results are sorted by date, in order to retrieve the most recent ones, and are limited to 30 reviews in order to not retrieve too much documents.

First of all, we executed the query without using indexes and the result was the following one:

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 30,
  "executionTimeMillis" : 2374,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 434021,
```

As we can see the execution time is 2374, the number of keys examined is 0, because there are no indexes that match the query filters, and the number of documents scanned is 434021 that is very high.

So, we inserted an index to sort the reviews by hotel name in order to have all the reviews of the same hotel grouped together. In addition, the index sorts the reviews of the same hotel by date, because the query asks for reviews ordered by date.

Executing the query using this index, we obtained the following result:


```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 30,
  "executionTimeMillis" : 1,
  "totalKeysExamined" : 30,
  "totalDocsExamined" : 30,
```

The execution time is much smaller than the previous case, the number of keys and scanned documents is 30 (the only ones that are retrieved), because the reviews are searched by using the index in the hotel name and the documents found are already sorted by date.

User Manual

LogIn/Registration Interface

A customer can register himself using the form on the right:




Log In	Registration
Email:	Name: <input type="text" value="Name"/>
<input type="text" value=""/>	Surname: <input type="text" value="Surname"/>
Password	Username: <input type="text" value="Username"/>
<input type="text" value="Password"/>	Email: <input type="text" value="Email"/>
<input type="button" value="Log In"/>	Password: <input type="text" value="Password"/>
	<input type="button" value="Register"/>

Name, surname, fiscal code, username, email and password are required. After entering the previous fields, simply click on the "Register" button.

The user cannot enter a username or email that has already been used by someone else.

A user can login using the form on the left:



Log In

Email:

Password

Log In

Registration

Name:

Surname:

Username:

Email:

Password:

Register

Email and password inserted during registration are required. After entering the previous fields, simply click on the "Log In" button.

After logging in, the Customer Interface or Employer Interface will be displayed depending on the role of the user.

Customer Interface

This interface allows the customer to view all the cities available with their characteristics and to search them by name or based on specific characteristics as shown in the picture below. At the beginning is shown the first thirty cities or the cities that meet the user's preferences, entered on his/her profile page.

[My Profile](#)
[Log Out](#)



City

2

Choose the preferences:

Free WiFi

Temperature (°C)

Quality Of Life

English Speaking

Friendly for foreigners

Nigthlife

Cost (\$)

Healthcare

Safety

Walkability

Air Quality

1

Choose the City:


Country	City	Characteristics
Spain	Barcelona	walkability=5 airQuality=4 wifi=3 cost=3409 qualityLife=4 safety=4 nightlife=5 temperature=16 english=3 foreignersFriendly=3 healthcare=3
Italy	Milan	walkability=5 airQuality=2 wifi=4 cost=4582 qualityLife=4 safety=5 nightlife=4 temperature=8 english=2

In the section number one the Customer can search the city by name and find it, whenever is available in the system, simply clicking on the "Search" button.

In the section number two the Customer can set up many characteristics he/she wants and views them clicking on the "Search" button.

If no characteristics are selected the first thirty cities are shown, the same happens if no name is specified in section one.

It is possible to select a city shown in the table and view the hotels by clicking on the "View Hotels" button, then the relative interface will be opened.



My Profile

Log Out

Choose the preferences:

Free WiFi

Temperature (°C)

Quality Of Life

English Speaking

Friendly for foreigners

Nighthlife

Cost (\$)

Healthcare

Safety

Walkability

Air Quality

Search

Choose the City:


City Name

Search

Country	City	Characteristics
		safety=4 nightlife=5 temperature=16 english=3 foreignersFriendly=3 healthcare=3
Italy	Milan	walkability=5 airQuality=2 wifi=4 cost=4582 qualityLife=4 safety=5 nightlife=4 temperature=8 english=2 foreignersFriendly=4 healthcare=4
United States	Lexington	walkability=5 airQuality=4

View Hotels

The customer can log out and return to the log in/registration page by clicking on the “Log Out” button or he/she can access his/her profile page by clicking on the “My Profile” button.



My Profile

Log Out

Choose the preferences:

Free WiFi

Temperature (°C)

Quality Of Life

English Speaking

Friendly for foreigners

Nighthlife

Cost (\$)

Healthcare

Safety

Walkability

Air Quality

Search

Choose the City:

City Name

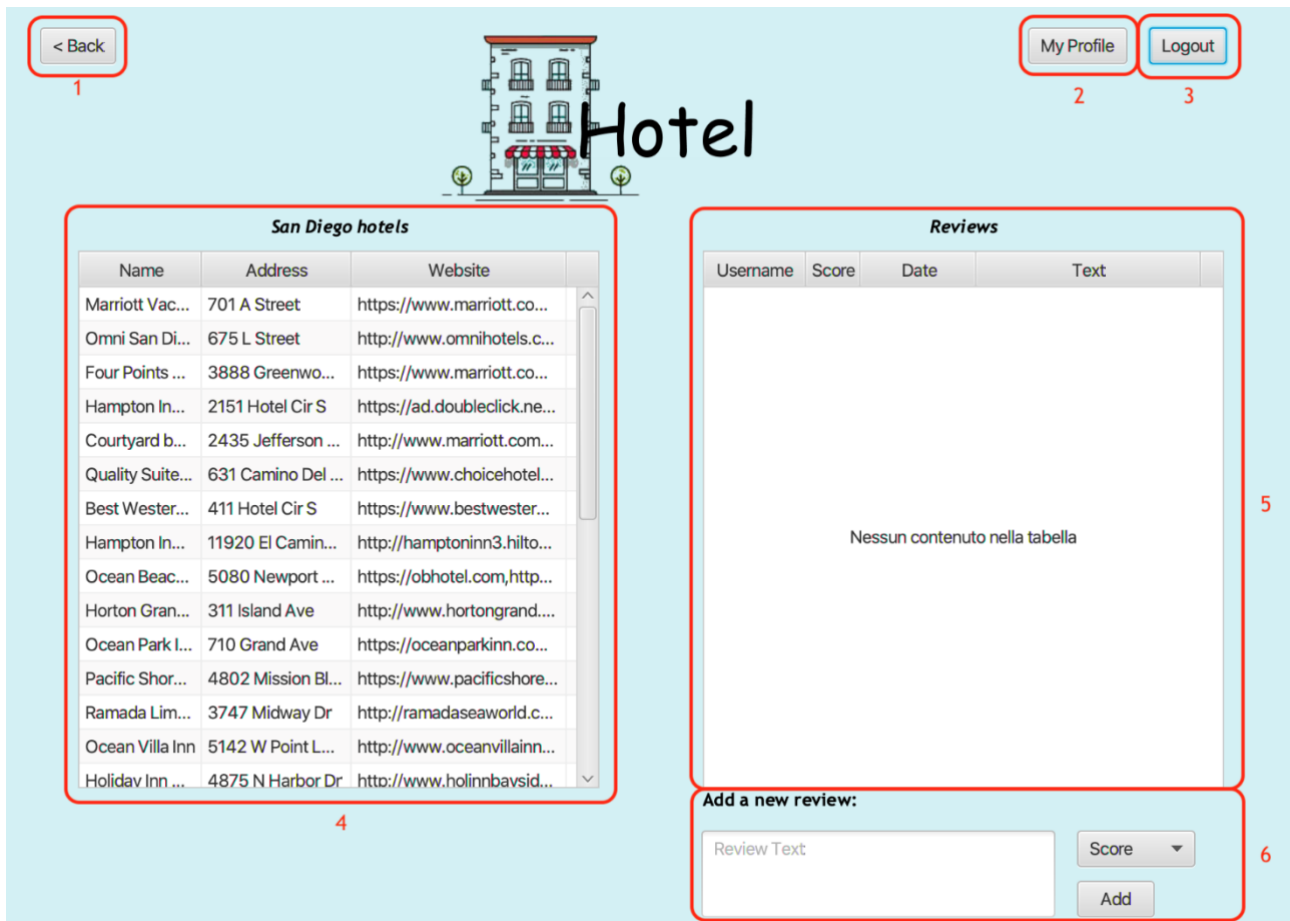
Search

Country	City	Characteristics
Spain	Barcelona	walkability=5 airQuality=4 wifi=3 cost=3409 qualityLife=4 safety=4 nightlife=5 temperature=16 english=3 foreignersFriendly=3 healthcare=3
Italy	Milan	walkability=5 airQuality=2 wifi=4 cost=4582 qualityLife=4 safety=5 nightlife=4 temperature=8 english=2

View Hotels

Hotel Interface

Once a City has been selected in the previous interface, the application shows the Hotel Interface, which is the following:




By clicking on the “Back” button on the top left (section 1 of the above image), it’s possible to go back to the city interface, in order to select another city.

Clicking the “My Profile” button on the top right (section 2), the customer can access its personal interface.

With the “Logout” (section 3) button on the top right the user can go back to the Login Interface. The section 4 shows the hotels’ list for the selected city. For every hotel, the application shows the Hotel’s name, the address and the website (whenever is present in the database, otherwise the field will be blank). By selecting a row (a hotel), in the Reviews’ table (section 5) will be shown the reviews related to that one.

Supposing that the customer has selected the second hotel in the list, the interface will be in the following situation:

< Back



Hotel

My Profile
Logout

San Diego hotels

Name	Address	Website
Marriott Vac...	701 A Street	https://www.marriott.co...
Omni San Di...	675 L Street	http://www.omnihotels.c...
Four Points ...	3888 Greenwo...	https://www.marriott.co...
Hampton In...	2151 Hotel Cir S	https://ad.doubleclick.ne...
Courtyard b...	2435 Jefferson ...	http://www.marriott.com...
Quality Suite...	631 Camino Del ...	https://www.choicehotel...
Best Wester...	411 Hotel Cir S	https://www.bestwester...
Hampton In...	11920 El Camin...	http://hamptoninn3.hilto...
Ocean Beac...	5080 Newport ...	https://obhotel.com/http...
Horton Gran...	311 Island Ave	http://www.hortongrand...
Ocean Park I...	710 Grand Ave	https://oceanparkinn.co...
Pacific Shor...	4802 Mission Bl...	https://www.pacificshcre...
Ramada Lim...	3747 Midway Dr	http://ramadaseaworld.c...
Ocean Villa Inn	5142 W Point L...	http://www.oceanvillainn...
Holiday Inn ...	4875 N Harbor Dr	http://www.holinnbaysid...

Reviews

Username	Score	Date	Text:
Vacation...	5	2018-05-07	The hotel is really nice the restaurants have a great food and the staff is excellent. The location makes it easy to get to anywhere been in downtown San Diego and the area is located with several nice restaurants to choose from . The staff...More
Dean L	5	2018-05-07	Stayed here for 4 nights in Nov and absolutely loved it! Great rooms, great staff, great location (right in Gas Lamp area), great bar and restaurant. Will definitely stay here again when I'm back in San Diego (soon, hopefully). A lovely place to come back...MoreDear Dean L. Thank you very much for

Add a new review for Omni San Diego Hotel:

Score

▼

Add


The Customer is now able to consult the reviews of the selected Hotel and to add a new one by filling the form located in the section 6 of the previous image.

When the customer has selected a hotel, in order to add a new review, he/she must select a score in the combo-box (from 1 up to 5) and optionally write a textual review.

Add a new review for Omni San Diego Hotel:


5

Add



After clicking the Add button, the review will be added in the list:

[< Back](#)



Hotel

[My Profile](#)
[Logout](#)

San Diego hotels

Name	Address	Website
Marriott Vac...	701 A Street	https://www.marriott.co...
Omni San Di...	675 L Street	http://www.omnihotels.c...
Four Points ...	3888 Greenwo...	https://www.marriott.co...
Hampton In...	2151 Hotel Cir S	https://ad.doubleclick.ne...
Courtyard b...	2435 Jefferson ...	http://www.marriott.com...
Quality Suite...	631 Camino Del ...	https://www.choicehotel...
Best Wester...	411 Hotel Cir S	https://www.bestwester...
Hampton In...	11920 El Camin...	http://hamptoninn3.hilto...
Ocean Beac...	5080 Newport ...	https://obhotel.com,http...
Horton Gran...	311 Island Ave	http://www.hortongrand....
Ocean Park I...	710 Grand Ave	https://oceanparkinn.co...
Pacific Shor...	4802 Mission Bl...	https://www.pacificshcre...
Ramada Lim...	3747 Midway Dr	http://ramadaseaworld.c...
Ocean Villa Inn	5142 W Point L...	http://www.oceanvillainn...
Holiday Inn ...	4875 N Harbor Dr	http://www.holinnbaysid...

Review successfully inserted

Reviews

Username	Score	Date	Text
Fonta	5	2020-01-02	Great vacation, all perfect!
Vacation...	5	2018-05-07	The hotel is really nice the restaurants have a great food and the staff is excellent. The location makes it easy to get to anywhere been in downtown San Diego and the area is located with several nice restaurants to choose from . The staff...More
Dean L	5	2018-05-07	Stayed here for 4 nights in Nov and absolutely loved it! Great rooms, great staff, great location (right in Gas Lamp area), great bar and restaurant. Will definitely stay here again when I'm back in San Diego (soon, hopefully). A lovely place to


Add a new review for Omni San Diego Hotel:

Score ▼

Add

Customer profile Interface

As already mentioned, by clicking on the “My Profile” button, in the previous interfaces, a Customer can access to his/her Personal Area.

[Back](#)anto65[Logout](#)

Personal Area

Name: antonio

Surname: bianchi


Email: anto@gmail.com

Select your preferences:

<input type="checkbox"/> Temperature	<input type="checkbox"/> Friendly for foreigners	<input type="checkbox"/> Cost
<input type="checkbox"/> Air Quality	<input type="checkbox"/> Healthcare	<input type="checkbox"/> Safety
<input type="checkbox"/> Quality of life	<input type="checkbox"/> Nightlife	<input type="checkbox"/> Walkability
<input type="checkbox"/> Free wi-fi	<input type="checkbox"/> English speaking	

[Save](#)

In this area, he/she can find his/her personal data as Name, Surname and Email.
On the top-right side the Customer's username is shown.

[Back](#)anto65[Logout](#)

Personal Area

Name: antonio

Surname: bianchi

Email: anto@gmail.com

In the section below he/she can select his/her preferences, those things that are important to find in a city for him/her.

Select your preferences:

<input type="checkbox"/> Temperature	<input type="checkbox"/> Friendly for foreigners	<input type="checkbox"/> Cost
<input type="checkbox"/> Air Quality	<input type="checkbox"/> Healthcare	<input type="checkbox"/> Safety
<input type="checkbox"/> Quality of life	<input type="checkbox"/> Nightlife	<input type="checkbox"/> Walkability
<input type="checkbox"/> Free wi-fi	<input type="checkbox"/> English speaking	

[Save](#)

Once he/she has chosen his/her preferences, he/she can save them by clicking on the Save button. In this way these data are stored. The customer can modify them repeating these operations or he/she can exit the Personal Area and choose to come back to the City's interface by clicking on the Back button or to the Login/Registration Interface using the Logout Button.

Back

anto65

Logout

Personal Area

Name: antonio Surname: bianchi Email: anto@gmail.com

Success!!!

Select your preferences:

<input checked="" type="checkbox"/> Temperature	<input type="checkbox"/> Friendly for foreigners	<input checked="" type="checkbox"/> Cost
<input checked="" type="checkbox"/> Air Quality	<input checked="" type="checkbox"/> Healthcare	<input type="checkbox"/> Safety
<input type="checkbox"/> Quality of life	<input type="checkbox"/> Nightlife	<input type="checkbox"/> Walkability
<input type="checkbox"/> Free wi-fi	<input type="checkbox"/> English speaking	


Save

Employee Interface

This interface allows to an Employee to have a view of the whole system by seeing all the registered customers and all the inserted cities with their hotel.

Customer View

Once an Employee has logged in, the interface is opened showing at first all the customers data in a table.



Administration Area

LOGOUT


Customer View

Show Statistics

Customers List

Name	Surname	Username	Email
Leonardo	Fontanelli	fonta	fonta@gmail.com
Eugenia	Petrangeli	geggi	geggi@gmail.com
Daniela	Comola	dani	dani@gmail.com
Carmelo	Aparo	aparo94	carmelo@gmail.com
bianca	pratesi	blank45	bianca@gmail.com
antonio	bianchi	anto65	anto@gmail.com
andrea	sbrilli	andresss78	andre@gmail.com
Achille	rossi	ach096	achillerossi@gmail.com
		STEPHEN N	STEPHEN_N@gmail.com
		tatsurok2018	tatsurok2018@gmail.com
		Wilfredo M	Wilfredo_M@gmail.com
		Luc D	Luc_D@gmail.com
		GGTravels2016	GGTravels2016@gmail.com

He/she can visualize the desired data easily by selecting the proper value from the menu:



Administration Area

LOGOUT

Customer View

Show Statistics


Customers List

Name	Surname	Username	Email
Leonardo	Fontanelli	fonta	fonta@gmail.com
Eugenia	Petrangeli	geggi	geggi@gmail.com

City View

By selecting "City View" from the menu the Employee can manage all the cities and related hotels in the system.

[LOGOUT](#)



Administration Area

City view ▼

[Show Statistics](#)

Cities List

Name	Country	Characteristics
Amsterdam	Netherlands	walkability=5 airQuality=4 wifi=4 cost=5297 qualityLife=4 safety=4 nightlife=4 temperature=11 english=4 foreignersFriendly=4 healthcare=4
London	United Kingdom	walkability=5 airQuality=4 wifi=4 cost=4469

[SEARCH](#)

[NEW](#)
[DELETE](#)

City's Hotels List

Name	Address	Websites
Nessun contenuto nella tabella		

[NEW](#)
[DELETE](#)

In the table on the left all the cities in the system are shown. When one of them is selected, the related hotels are visualized in the table on the right. At the beginning, there's not any city selected.

A new city can be added to the system by clicking the NEW button. A form with all the related fields will appear. In order to add a new city, all of them must be filled.

[SEARCH](#)

[NEW](#)
[DELETE](#)

Add a new City:

Name: Country:

Characteristics:

Temperature (°C):	<input type="text" value="22"/>	Cost (\$):	<input type="text" value="1500"/>
Free WiFi:	<input type="text" value="3"/>	Healthcare:	<input type="text" value="4"/>
Quality of Life:	<input type="text" value="1"/>	Safety:	<input type="text" value="2"/>
English Speaking:	<input type="text" value="2"/>	Walkability:	<input type="text" value="2"/>
Nightlife:	<input type="text" value="3"/>	Air Quality:	<input type="text" value="3"/>
Friendly for Foreigners:	<input type="text" value="4"/>		

Once all the fields are completed, the Employee can click on the ADD CITY button and the data are stored.

If he/she would update an existing city, he/she must insert the Name and Country of the required city and only the interested fields to change. Then the city is updated by clicking the UPDATE button.

Add a new City:

Success!

Name: Country:

Characteristics:

Temperature (°C): Cost (\$):

Free WiFi: Healthcare:

Quality of Life: Safety:

English Speaking: Walkability:

Nightlife: Air Quality:


Friendly for Foreigners:

The Employee can look for a certain city specifying its name in the research field and clicking the Search Button.

If no name is specified, all the cities are shown.

Once the interested city is found, the Employee can select it and all the related hotels will appear in the right table.

[LOGOUT](#)



Administration Area

City view ▼

[Show Statistics](#)

Cities List

Name	Country	Characteristics
Amsterdam	Netherlands	walkability=5 airQuality=4 wifi=4 cost=5297 qualityLife=4 safety=4 nightlife=4 temperature=11 english=4 foreignersFriendly=4 healthcare=4
London	United Kingdom	walkability=5 airQuality=4 wifi=4 cost=4469

SEARCH

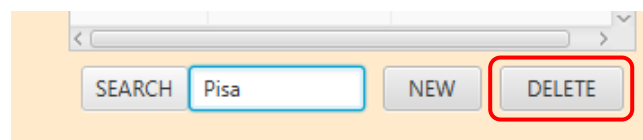
NEW
DELETE

Amsterdam's Hotels List

Name	Address	Websites
Hotel Arena	s Gravesandestra...	
Morgan Mees	2e Hugo de Groo...	
Corendon Vitalit...	Aletta Jacobslaan ...	
Hampshire Hotel...	Amstelstraat 17 A...	
Sir Albert Hotel	Albert Cuypstraat...	
Pillows Anna van...	Anna van den Vo...	
Best Western De...	Apollolaan 101 1...	
Apollo Hotel Am...	Apollolaan 2 Oud...	
Hilton Amsterdam	Apollolaan 138 O...	
Element Amster...	Arend Janszoon E...	
Urban Lodge Ho...	Arlandaweg 10 W...	


NEW
DELETE

At this point, he/she can delete it by clicking on the DELETE button and all its hotels, with all their reviews, will be deleted as well.



Otherwise, he/she can examine all the retrieved hotels and selecting one of them he/she can delete the chosen hotel with all he related reviews.

Hotel view



Administration Area

City view

Show Statistics

Cities List

Name	Country	Characteristics
Amsterdam	Netherlands	walkability=5 airQuality=4 wifi=4 cost=5297 qualityLife=4 safety=4 nightlife=4 temperature=11 english=4 foreignersFriendly=4 healthcare=4
London	United Kingdom	walkability=5 airQuality=4 wifi=4 cost=4469

SEARCH

city Name

NEW

DELETE

Amsterdam's Hotels List

Name	Address	Websites
Hotel Arena	s Gravesandestra...	
Morgan Mees	2e Hugo de Groo...	
Corendon Vitalit...	Aletta Jacobslaan ...	
Hampshire Hotel...	Amstelstraat 17 A...	
Sir Albert Hotel	Albert Cuypstraat...	
Pillows Anna van...	Anna van den Vo...	
Best Western De...	Apollolaan 101 1...	
Apollo Hotel Am...	Apollolaan 2 Oud...	
Hilton Amsterdam	Apollolaan 138 O...	
Element Amster...	Arend Janszoon E...	
Urban Lodge Ho...	Arlandaweg 10 W...	

NEW

DELETE

In addition, the Employee can add a new hotel for the selected city. By clicking the NEW button a form with the related fields will appear. It is mandatory to fill the Name and Address ones, than by clicking the ADD HOTEL button data are stored.

airQuality=4
wifi=4
cost=4469

SEARCH

city Name

NEW

DELETE

Element Amster... Arend Janszoon E...
Urban Lodge Ho... Arlandaweg 10 W...

NEW

DELETE

City view

Cities List

Name	Country	Charac
Amsterdam	Netherlands	walkability= airQuality= wifi=4 cost=5297 qualityLife= safety=4 nightlife=4 temperatur english=4 foreignersF healthcare=
London	United Kingdom	walkability= airQuality= wifi=4 cost=4469

SEARCH city Name NEW

Add a new hotel:

City: Amsterdam

Country: Netherlands

Name: Hotel Royal

Address: Amsterlat 27 Avenue

Website:

Operation successfully completed

ADD HOTEL

View Statistics

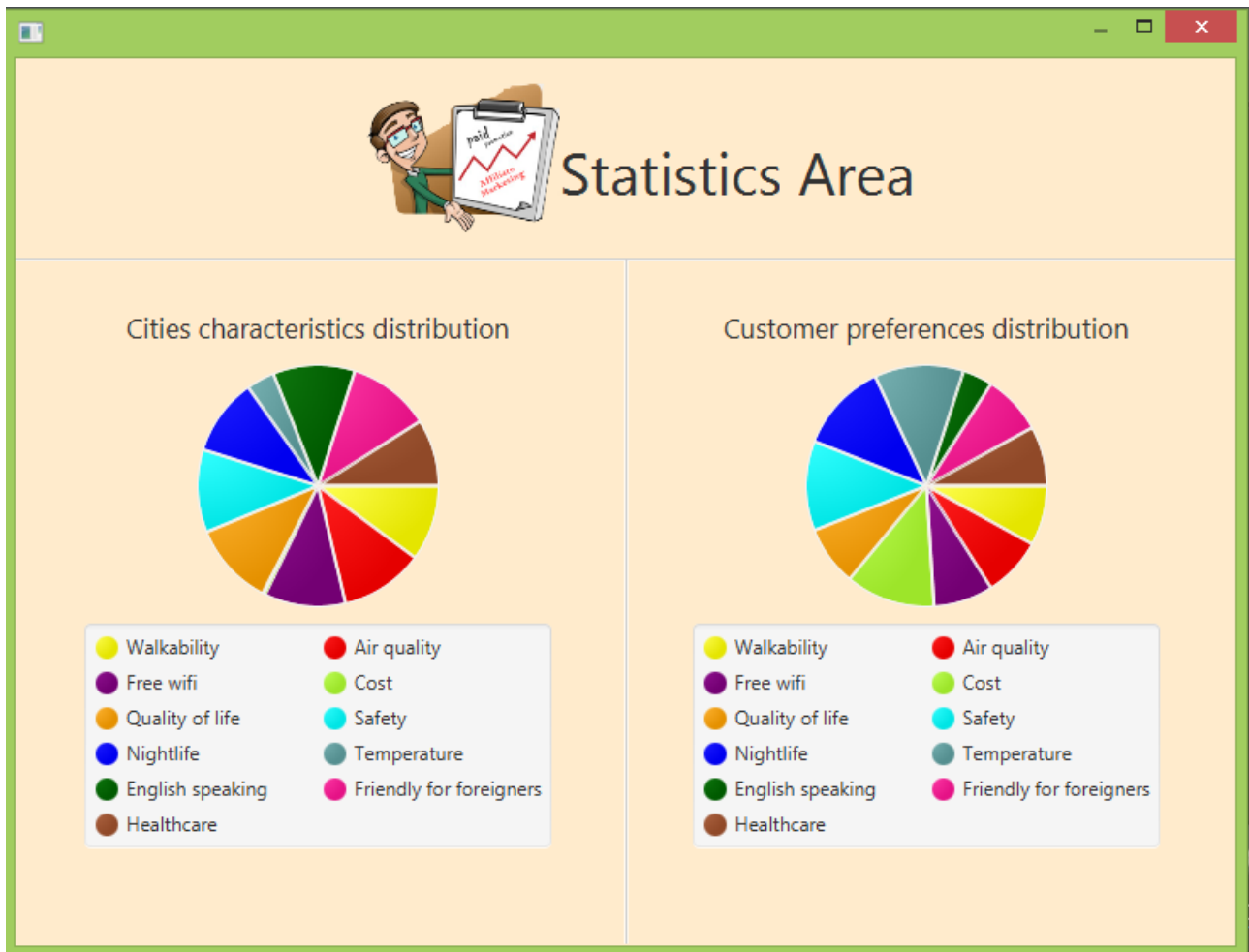
From both Customer View and City View the Employee can access to a page dedicated to statistics, by clicking the Show Statistics button.

City view

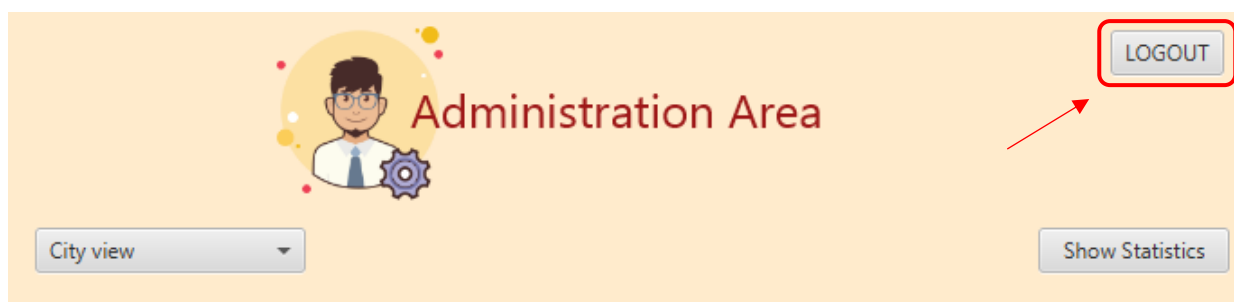
Administration Area

LOGOUT

Show Statistics

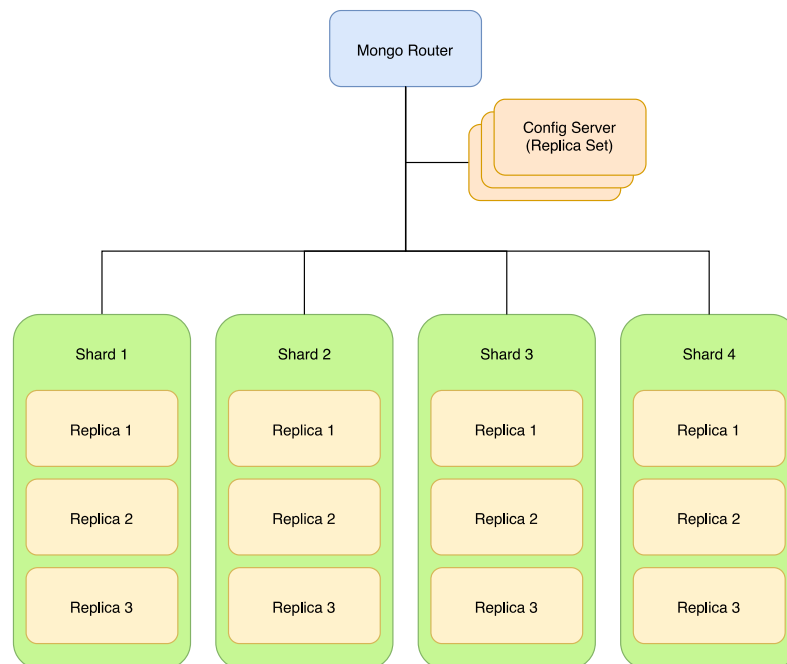


By closing this window, the Employee come back to the main Interface. By clicking on the LOGOUT button, he/she come back to the Login/Registration Interface.



Sharding proposal

In order to improve the scalability of the system we could implement a sharding mechanism based on this structure:



The main components of the structure are:

- A **mongo router**, that is a specific node that acts as interface between the client application and the shards.
It receives the requests from the application and divides them among the shards. If the number of read requests increases, there is the possibility to increment also the number of mongo routers in order to balance the workload.
- A **config server** stores the information about the structure of the cluster.
It is an important component because the mongo router forwards the requests depending on these data. So, in order to avoid having a single point of failure, it should be better to have a replica set. We could have 3 replica servers in order to guarantee a sufficient availability.
- Four **shards** to store different chunks of the dataset. They could be built starting from the replica set we already have. As shown in the picture we could use 3 replicas for each shard in order to prevent failures.

Given that we have 4 collections, we should define 4 different shard keys:

- For the user collection we could use the **email field** because it is used for most of the queries.
- For the city collection we could use the **city field** in order to evenly distribute the documents.
- For the hotel collection we could use the **city field** because it is used for all queries.
- For the review collection we could use the **city field** because it is used for all queries.

We could use a **ranged sharding strategy**, because the shard keys for all the collections are non-monotonically changing and have a large cardinality, so it is the strategy more appropriated in order to balance the workload.

Conclusions

Below are the features we exploited using MongoDB:

- Documents do not have a fixed structure, so in one document we may find a value that is not present in another document as it is for users. These may and may not have a list of preferences which may be different for each user. Another example is for hotel documents, some of which may have one or more links on their site while others none.
- Thanks to the use of indexes, many readings have been speeded up since it was possible to obtain the information that interested us without having to scroll through all the documents
- Thanks to the fact that they support polymorphic schemes, it was easy to insert in a single collection both the customer and the employees. They have information in common such as name, surname and email but other fields that are not in common.
- Simplicity in querying the database to take the information necessary for the operation of the application, also going to filter one or more attributes without difficulty.
- The document database is also convenient for managing large-scale data and not being forced to define a data structure right away, but it is easy to change it over time.
- Another feature of MongoDB is the management of the replicas, that ensured high availability for our application.