NOMAD ADVISOR 2.0

Task 3 – Large Scale and Multi-Structured Databases

UNIVERSITÀ DI PISA

Eugenia Petrangeli, Daniela Comola, Carmelo Aparo, Leonardo Fontanelli

Index

Introduction

Description of the application

In this application we would like to extend our previous NomadAdvisor Application by adding some functionalities both for Customer and Employee.

The Customer maintain all the already existing functionalities (searching info about cities and hotels with their reviews and the possibility to maintain a personal area with his/her personal data and preferences).

In addition, the system allows him/her to view all the other customers that share the same preferences with him/her, showing their email, username and common preferences.
In this way a Customer can directly contact some of them in order to ask for some suggestions about places to visit.

The administration area also remains the same, with the previous functionalities.
In addition, in the statistics area, an Employee can also see, the average age of customers that expressed a certain preference, for each preference.
In this way he/she will be able to evaluate if the offered service is suitable only for old or young people, for example, and eventually extend the application in order to attract more users.

Functional requirements

For the already existing application the requirements are not modified except for a little difference in the log-in/registration interface: in the registration form an "Age" field is added, so, in order to register to the system, a Customer shall now insert all the fields *Name, Surname, **Age**, Password, Nickname and Email*.
A Customer can update his/her age from the Personal Area Interface. Here, a Customer can also update his/her preferences.
The system retrieves from the logged user his/her preferences and shows him/her all the other customers, that had chosen the same preferences, in a table containing their *Username, Email* and the shared *Preferences.* The customers are sorted in descendent order based on the number of common preferences.

For the Administration Area, instead, the system performs an additional statistic, calculating the average age of the customers for each preference and showing the result in a bar chart.
The Employee can view this information in the statistics section, with the existing ones.

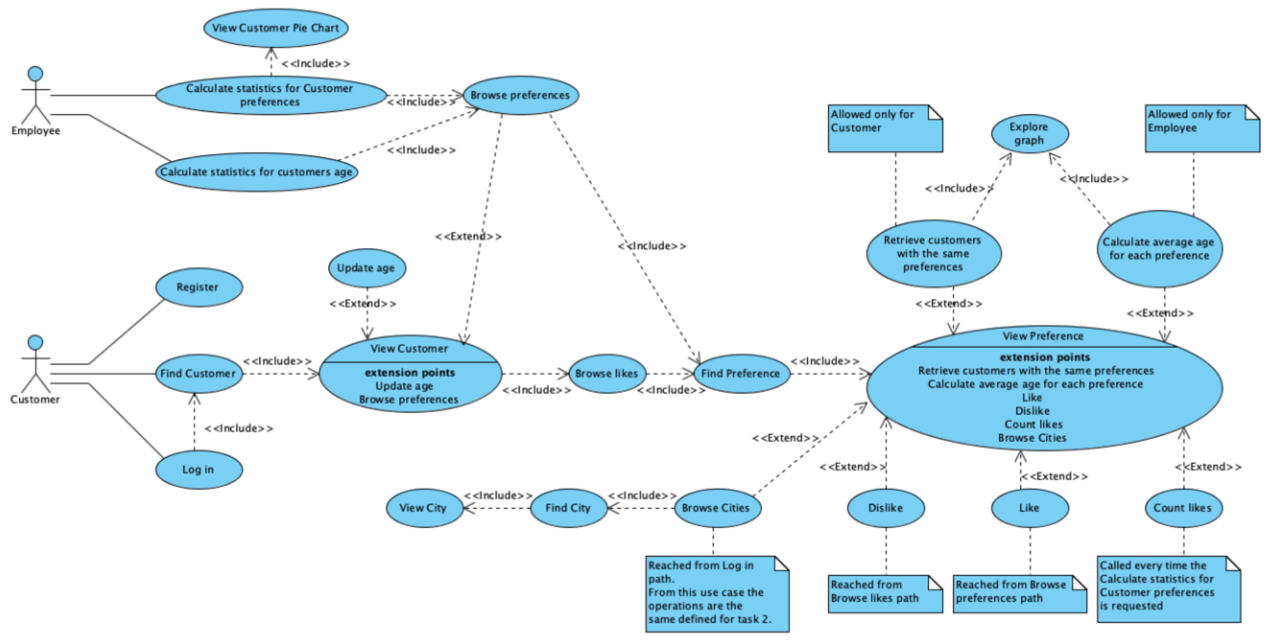
Non-functional requirements

The functional requirements of the Task 2 are not modified.
In addition, the system uses a specific user to connect to Neo4j with "*publisher*" role, in this way the application has limited privileges on the DB in particular it can only modify the specific DB dedicated for the application.

Design

UML diagram – Use Case

In the following diagram there are only the use cases regarding the graph database. The other use cases are the same of the Task 2.


View Customer Pie Chart
<<Include>>
Calculate statistics for Customer preferences
<<Include>>
Browse preferences
Employee
<<Include>>
Calculate statistics for customers age
<<Extend>>
<<Include>>
Allowed only for Customer
Explore graph
Allowed only for Employee
<<Include>>
<<Include>>
Retrieve customers with the same preferences
Calculate average age for each preference
Update age
<<Extend>>
<<Extend>>
<<Extend>>
Register
View Customer
extension points
Update age
Browse preferences
View Preference
extension points
Retrieve customers with the same preferences
Calculate average age for each preference
Like
Dislike
Count likes
Browse Cities
Customer
Find Customer
<<Include>>
Browse likes
Find Preference
<<Include>>
<<Include>>
<<Include>>
<<Include>>
<<Extend>>
Log in
<<Extend>>
<<Extend>>
<<Extend>>
View City
<<Include>>
Find City
<<Include>>
Browse Cities
Dislike
Like
Count likes
Reached from Log in path.
From this use case the operations are the same defined for task 2.
Reached from Browse likes path
Reached from Browse preferences path
Called every time the Calculate statistics for Customer preferences is requested

UML diagram – Analysis Classes

The Analysis Classes of the Task 2 is the same for this Task, with the exception of the "Age" attribute for the Customer's class which has been added now.

Data Model

In order to maintain the old functionalities and provide the new ones, the system integrates the already existing Document Database with a Graph Database.
In particular, the structure of the City's, Hotel's and Review's Documents are unchanged while the User's Document is now built in this way:

- User's document:
```
{
  "_id": ObjectId("507f1f77bcf86cd799439012"),
  "role": "customer",
  "name": "Mario",
  "surname": "Rossi",
  "age": 25,
  "email": "tatsurok2018@gmail.com",
  "password": "tatsurok2018",
  "username": "tatsurok2018"
}
```

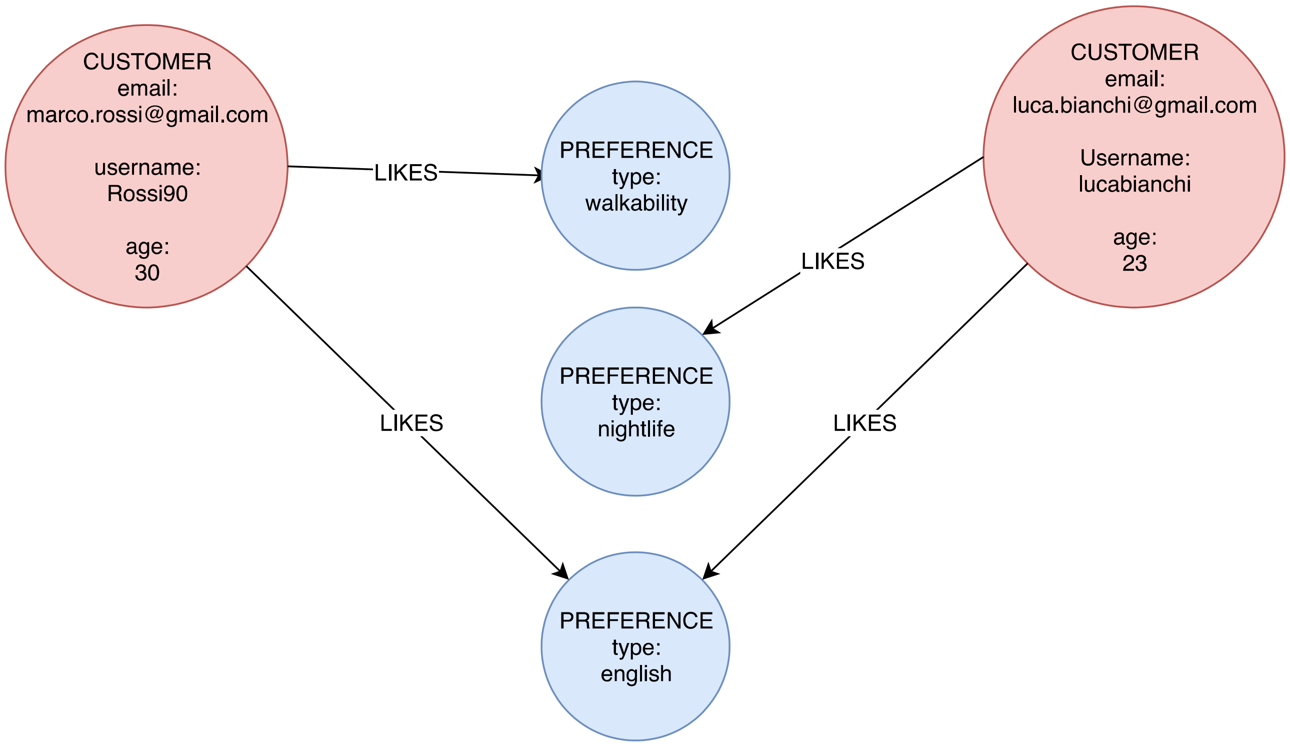
As we can see, there's no more the "preferences" field, since that this information is now maintained thanks to the graph database.
In this last database, in fact, there is one node for each possible preference and a node for each Customer in the system, containing only his/her information useful for the new functionalities, so *Email*, *Username* and *Age*.
Each Customer is linked with the preferences he/she has chosen.

We decide to duplicate Customer data so that we can continue in taking advantage of the speed of the document database in finding a user, during the login phase.
The graph database is exploited for all the other operations that involve these data.

We use Neo4j constraints to ensure that property values are unique for all nodes with a specific label. In particular:

- “email” property for the CUSTOMER nodes
- “username” property for the CUSTOMER nodes
- “type” property for the PREFEERENCE nodes

Snapshot of the graph



Queries

On-graph queries

Domain-specific query	Graphic-centric query
How many customers has selected a certain preference?	How many incoming edges are incident to a certain Preference vertex?
What is the average age of the customer that has selected a certain preference?	What is the average value of the attribute “age” of the vertices that are connected with a Preference vertex?
Given a certain customer, who has selected common preferences?	Given a certain Customer vertex, what are the Customer vertices that are connected to the same preferences vertices?

CRUD operations

- **CREATE:**
  - The system creates a vertex for every customer that register to the service.
  - If a customer selects a preference for a characteristic in his personal area, the system creates an edge between the customer's vertex and the one related to the selected preference.
- **READ:**
  - When the system needs to know the preferences related to a customer, it reads the preference vertices connected to the customer one.
- **UPDATE:** When a customer updates the age value from its personal area, the value is updated in the correspondent attribute in the customer's vertex as well.
- **DELETE:** If a customer deletes a preference from its personal area, the edge between the customer's vertex and the preferences' vertex is deletes as well.

Implementation

Implementation Class Diagram

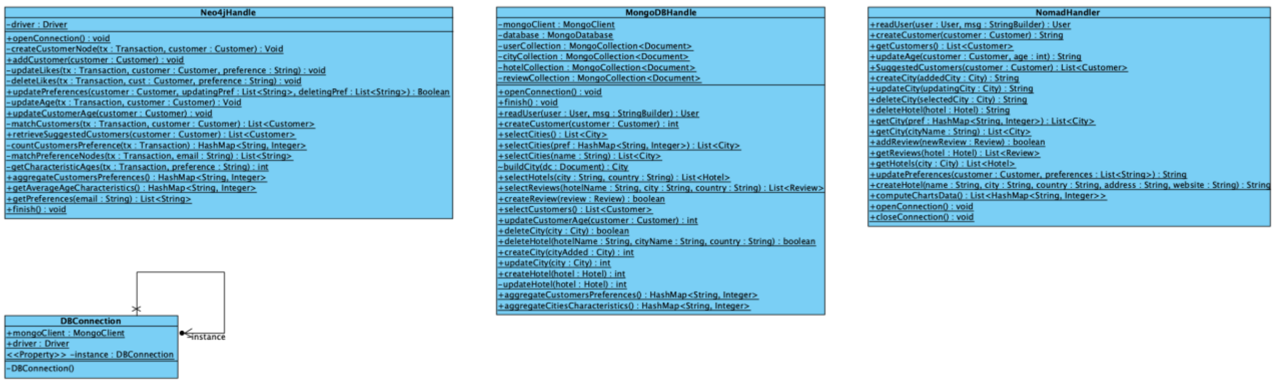
With respect to the implementation class diagram of the previous Task, the classes that have been added are:

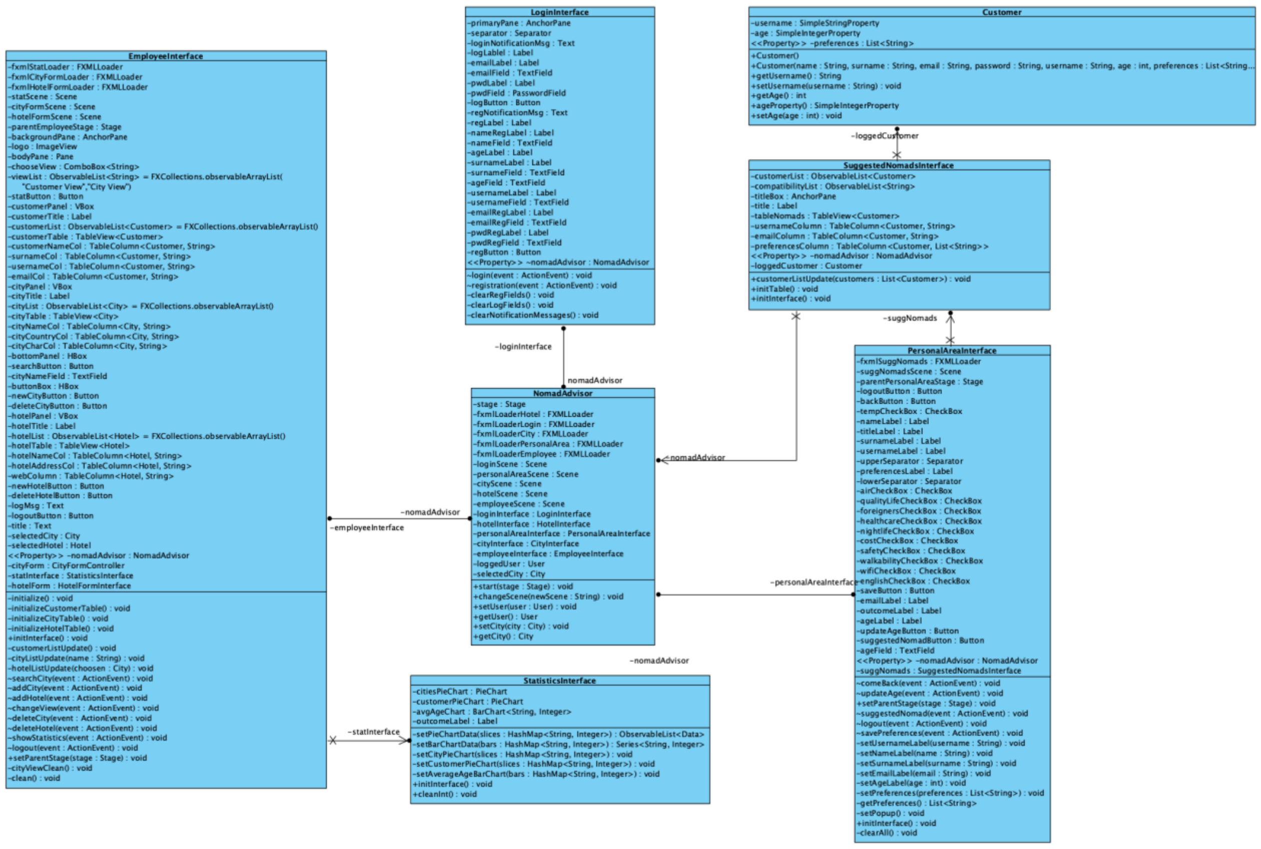
- **SuggestedNomadsInterface:** it's the controller for the related FXML file. It handles every event related to the interface.
- **Neo4jHandle:** This is the class that handles every operation on the Graph Database using Neo4j JAVA Driver

The following are the classes that have been modified:

- **NomadHandler:** it has been modified in order to manage Neo4j operation by calling Neo4jHandle's methods and it has been added a method in order to update the customer's age.
- **StatisticsInterface:** It has been added a bar chart in order to show the average age of customers that chose a specific preference.
- **LoginInterface:** It has been added the age field in the registration form.
- **PersonalAreaInterface:** It has been added a functionality in order to update the age value of the customer and the possibility to open the SuggestedNomadsInterface.

The following diagram shows only the changed classes with respect to the previous Task.


Neo4jHandle
-driver : Driver
+openConnection() : void
-createCustomerNode(tx : Transaction, customer : Customer) : Void
+addCustomer(customer : Customer) : void
-updateLikes(tx : Transaction, customer : Customer, preference : String) : void
-deleteLikes(tx : Transaction, cust : Customer, preference : String) : void
+updatePreferences(customer : Customer, updatingPref : List<String>, deletingPref : List<String>) : Boolean
-updateAge(tx : Transaction, customer : Customer) : Void
+updateCustomerAge(customer : Customer) : void
-matchCustomers(tx : Transaction, customer : Customer) : List<Customer>
+retrieveSuggestedCustomers(customer : Customer) : List<Customer>
-countCustomersPreference(tx : Transaction) : HashMap<String, Integer>
-matchPreferenceNodes(tx : Transaction, email : String) : List<String>
-getCharacteristicAges(tx : Transaction, preference : String) : int
+aggregateCustomersPreferences() : HashMap<String, Integer>
+getAverageAgeCharacteristics() : HashMap<String, Integer>
+getPreferences(email : String) : List<String>
+finish() : void
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>
-buildCity(dc : Document) : City
+selectHotels(city : String, country : String) : List<Hotel>
+selectReviews(hotelName : String, city : String, country : String) : List<Review>
+createReview(review : Review) : boolean
+selectCustomers() : List<Customer>
+updateCustomerAge(customer : Customer) : int
+deleteCity(city : City) : boolean
+deleteHotel(hotelName : String, cityName : String, country : String) : boolean
+createCity(cityAdded : City) : int
+updateCity(city : City) : int
+createHotel(hotel : Hotel) : int
-updateHotel(hotel : Hotel) : int
+aggregateCustomersPreferences() : HashMap<String, Integer>
+aggregateCitiesCharacteristics() : HashMap<String, Integer>
NomadHandler
+readUser(user : User, msg : StringBuilder) : User
+createCustomer(customer : Customer) : String
+getCustomers() : List<Customer>
+updateAge(customer : Customer, age : int) : String
+SuggestedCustomers(customer : Customer) : List<Customer>
+createCity(addedCity : City) : String
+updateCity(updatingCity : City) : String
+deleteCity(selectedCity : City) : String
+deleteHotel(hotel : Hotel) : String
+getCity(pref : HashMap<String, Integer>) : List<City>
+getCity(cityName : String) : List<City>
+addReview(newReview : 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
+computeChartsData() : List<HashMap<String, Integer>>
+openConnection() : void
+closeConnection() : void
DBConnection
+mongoClient : MongoClient
+driver : Driver
<<Property>> -instance : DBConnection
-DBConnection()
-instance



```
EmployeeInterface
-fxmlStatLoader : FXMLLoader
-fxmlCityFormLoader : FXMLLoader
-fxmlHotelFormLoader : FXMLLoader
-statScene : Scene
-cityFormScene : Scene
-hotelFormScene : Scene
-parentEmployeeStage : Stage
-backgroundPane : AnchorPane
-logo : ImageView
-bodyPane : Pane
-chooseView : ComboBox<String>
-viewList : ObservableList<String> = FXCollections.observableArrayList(
    "Customer View","City View")
-statButton : Button
-customerPanel : VBox
-customerTitle : Label
-customerList : ObservableList<Customer> = FXCollections.observableArrayList()
-customerTable : TableView<Customer>
-customerNameCol : TableColumn<Customer, String>
-surnameCol : TableColumn<Customer, String>
-usernameCol : TableColumn<Customer, String>
-emailCol : TableColumn<Customer, String>
-cityPanel : VBox
-cityTitle : Label
-cityList : ObservableList<City> = FXCollections.observableArrayList()
-cityTable : TableView<City>
-cityNameCol : TableColumn<City, String>
-cityCountryCol : TableColumn<City, String>
-cityCharCol : TableColumn<City, String>
-bottomPanel : HBox
-searchButton : Button
-cityNameField : TextField
-buttonBox : HBox
-newCityButton : Button
-deleteCityButton : Button
-hotelPanel : VBox
-hotelTitle : Label
-hotelList : ObservableList<Hotel> = FXCollections.observableArrayList()
-hotelTable : TableView<Hotel>
-hotelNameCol : TableColumn<Hotel, String>
-hotelAddressCol : TableColumn<Hotel, String>
-webColumn : TableColumn<Hotel, String>
-newHotelButton : Button
-deleteHotelButton : Button
-logMsg : Text
-logoutButton : Button
-title : Text
-selectedCity : City
-selectedHotel : Hotel
<<Property>> -nomadAdvisor : NomadAdvisor
-cityForm : CityFormController
-statInterface : StatisticsInterface
-hotelForm : HotelFormInterface
-initialize() : void
-initializeCustomerTable() : void
-initializeCityTable() : void
-initializeHotelTable() : void
+initInterface() : void
-customerListUpdate() : void
-cityListUpdate(name : String) : void
-hotelListUpdate(choosen : City) : void
~searchCity(event : ActionEvent) : void
~addCity(event : ActionEvent) : void
~addHotel(event : ActionEvent) : void
~changeView(event : ActionEvent) : void
~deleteCity(event : ActionEvent) : void
~deleteHotel(event : ActionEvent) : void
~showStatistics(event : ActionEvent) : void
-logout(event : ActionEvent) : void
+setParentStage(stage : Stage) : void
-cityViewClean() : void
-clean() : void
```

```
LoginInterface
-primaryPane : AnchorPane
-separator : Separator
-loginNotificationMsg : Text
-logLablel : Label
-emailLabel : Label
-emailField : TextField
-pwdLabel : Label
-pwdField : PasswordField
-logButton : Button
-regNotificationMsg : Text
-regLabel : Label
-nameRegLabel : Label
-nameField : TextField
-ageLabel : Label
-surnameLabel : Label
-surnameField : TextField
-ageField : TextField
-usernameLabel : Label
-usernameField : TextField
-emailRegLabel : Label
-emailRegField : TextField
-pwdRegLabel : Label
-pwdRegField : TextField
-regButton : Button
<<Property>> ~nomadAdvisor : NomadAdvisor
~login(event : ActionEvent) : void
~registration(event : ActionEvent) : void
-clearRegFields() : void
-clearLogFields() : void
-clearNotificationMessages() : void
```

```
NomadAdvisor
-stage : Stage
-fxmlLoaderHotel : FXMLLoader
-fxmlLoaderLogin : FXMLLoader
-fxmlLoaderCity : FXMLLoader
-fxmlLoaderPersonalArea : FXMLLoader
-fxmlLoaderEmployee : FXMLLoader
-loginScene : Scene
-personalAreaScene : Scene
-cityScene : Scene
-hotelScene : Scene
-employeeScene : Scene
-loginInterface : LoginInterface
-hotelInterface : HotelInterface
-personalAreaInterface : PersonalAreaInterface
-cityInterface : CityInterface
-employeeInterface : EmployeeInterface
-loggedUser : User
-selectedCity : City
+start(stage : Stage) : void
+changeScene(newScene : String) : void
+setUser(user : User) : void
+getUser() : User
+setCity(city : City) : void
+getCity() : City
```

```
StatisticsInterface
-citiesPieChart : PieChart
-customerPieChart : PieChart
-avgAgeChart : BarChart<String, Integer>
-outcomeLabel : Label
-setPieChartData(slices : HashMap<String, Integer>) : ObservableList<Data>
-setBarChartData(bars : HashMap<String, Integer>) : Series<String, Integer>
-setCityPieChart(slices : HashMap<String, Integer>) : void
-setCustomerPieChart(slices : HashMap<String, Integer>) : void
-setAverageAgeBarChart(bars : HashMap<String, Integer>) : void
+initInterface() : void
+cleanInt() : void
```

```
Customer
-username : SimpleStringProperty
-age : SimpleIntegerProperty
<<Property>> -preferences : List<String>
+Customer()
+Customer(name : String, surname : String, email : String, password : String, username : String, age : int, preferences : List<String...
+getUsername() : String
+setUsername(username : String) : void
+getAge() : int
+ageProperty() : SimpleIntegerProperty
+setAge(age : int) : void
```

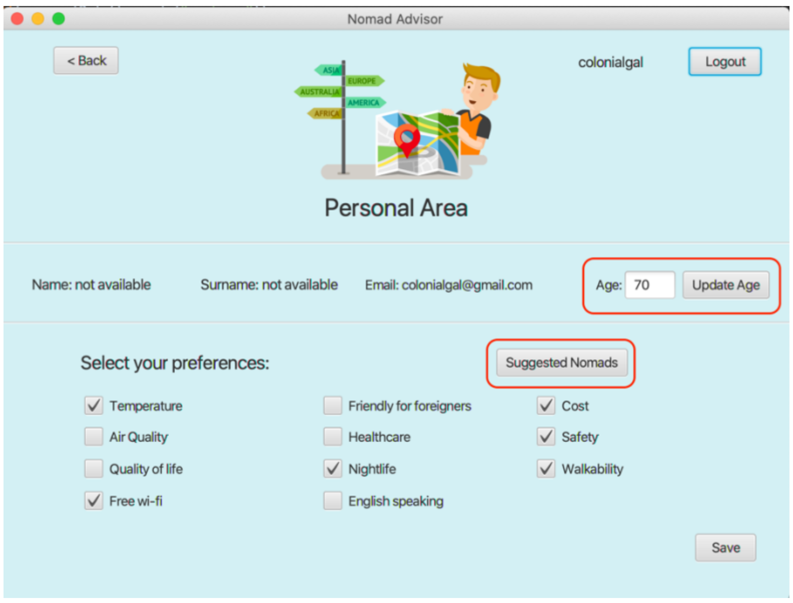
```
SuggestedNomadsInterface
-customerList : ObservableList<Customer>
-compatibilityList : ObservableList<String>
-titleBox : AnchorPane
-title : Label
-tableNomads : TableView<Customer>
-usernameColumn : TableColumn<Customer, String>
-emailColumn : TableColumn<Customer, String>
-preferencesColumn : TableColumn<Customer, List<String>>
<<Property>> -nomadAdvisor : NomadAdvisor
-loggedCustomer : Customer
+customerListUpdate(customers : List<Customer>) : void
+initTable() : void
+initInterface() : void
```

```
PersonalAreaInterface
-fxmlSuggNomads : FXMLLoader
-suggNomadsScene : Scene
-parentPersonalAreaStage : Stage
-logoutButton : Button
-backButton : Button
-tempCheckBox : CheckBox
-nameLabel : Label
-titleLabel : Label
-surnameLabel : Label
-usernameLabel : Label
-upperSeparator : Separator
-preferencesLabel : Label
-lowerSeparator : Separator
-airCheckBox : CheckBox
-qualityLifeCheckBox : CheckBox
-foreignersCheckBox : CheckBox
-healthcareCheckBox : CheckBox
-nightlifeCheckBox : CheckBox
-costCheckBox : CheckBox
-safetyCheckBox : CheckBox
-walkabilityCheckBox : CheckBox
-wifiCheckBox : CheckBox
-englishCheckBox : CheckBox
-saveButton : Button
-emailLabel : Label
-outcomeLabel : Label
-ageLabel : Label
-updateAgeButton : Button
-suggestedNomadButton : Button
-ageField : TextField
<<Property>> -nomadAdvisor : NomadAdvisor
-suggNomads : SuggestedNomadsInterface
~comeBack(event : ActionEvent) : void
~updateAge(event : ActionEvent) : void
+setParentStage(stage : Stage) : void
~suggestedNomad(event : ActionEvent) : void
~logout(event : ActionEvent) : void
~savePreferences(event : ActionEvent) : void
-setUsernameLabel(username : String) : void
-setNameLabel(name : String) : void
-setSurnameLabel(surname : String) : void
-setEmailLabel(email : String) : void
-setAgeLabel(age : int) : void
-setPreferences(preferences : List<String>) : void
-getPreferences() : List<String>
-setPopup() : void
+initInterface() : void
-clearAll() : void
```

User Manual

In this section are described the functionalities of the application that are different from the previous Task.

Personal Area Interface



Name: not available

Surname: not available

Email: colonialgal@gmail.com

Age: 70

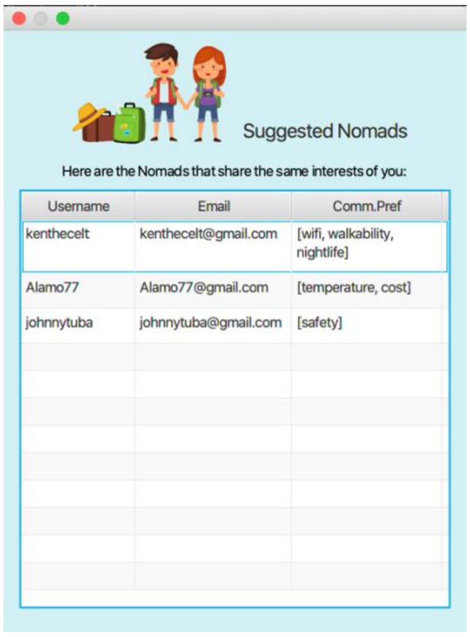
Select your preferences:

- [✓] Temperature
- [ ] Air Quality
- [ ] Quality of life
- [✓] Free wi-fi
- [ ] Friendly for foreigners
- [ ] Healthcare
- [✓] Nightlife
- [ ] English speaking
- [✓] Cost
- [✓] Safety
- [✓] Walkability

As showed in the above image, in this interface has been added the age form and the “Suggested Nomads” button.

When the interface is opened, in the age field will be showed the age inserted by the customer in the registration phase. Here it has the possibility to update this field by inserting the new value in the form and by clicking on the “Update Age” button.
The “Suggested Nomads” button allows the customer to open the Suggested Nomads Interface

Suggested Nomads Interface

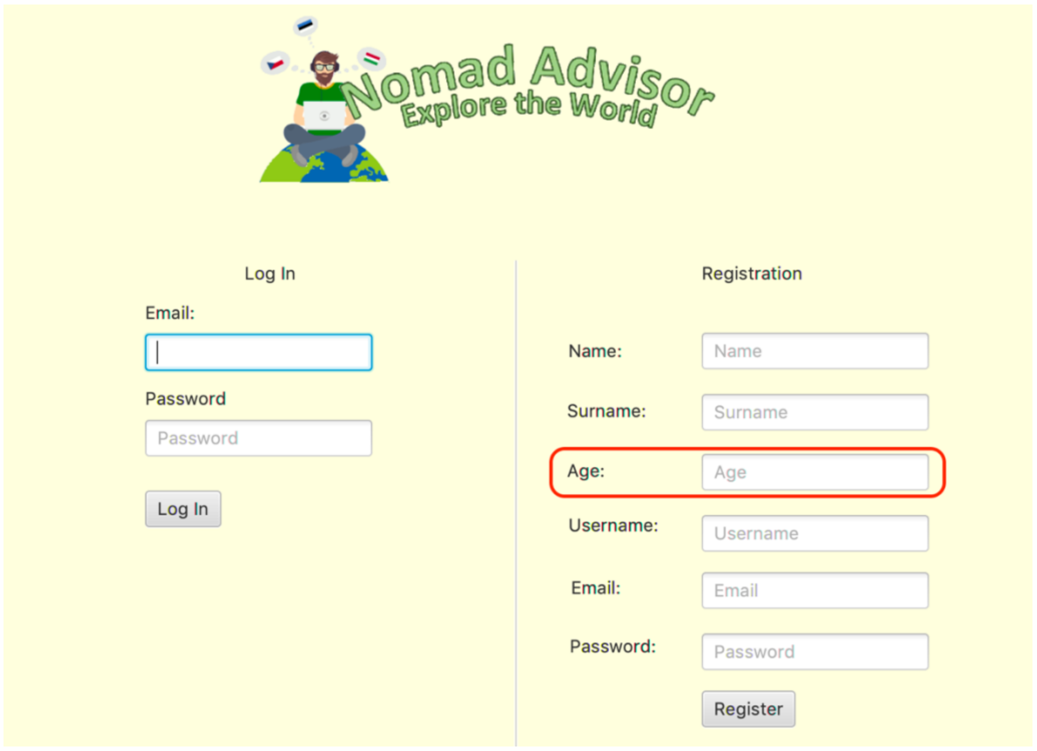


Here are the Nomads that share the same interests of you:

Username	Email	Comm.Pref
kenthecelt	kenthecelt@gmail.com	[wifi, walkability, nightlife]
Alamo77	Alamo77@gmail.com	[temperature, cost]
johnnytuba	johnnytuba@gmail.com	[safety]

In this Interface the customer is able to consult the list of customers that have selected common preference in their Personal Area. The table is composed by 3 columns: Username, Email e Common Preferences. Every row of the table represents a customer, the Common Preferences column shows the preferences that are in common with the logged customer.
The customers in the table are ordered by the number of preferences that are in common by descending order.

Login Interface



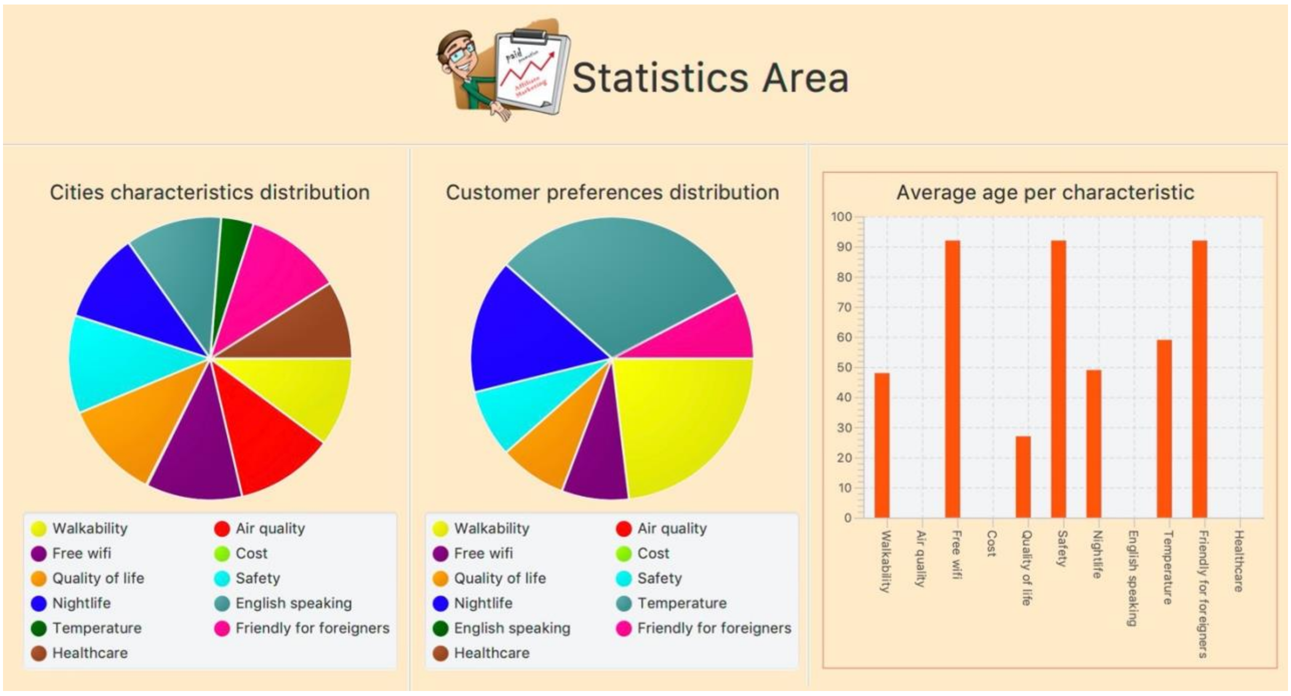
Log In	Registration

Email:	
Password	Password

Name:	Name
Surname:	Surname
Age:	Age
Username:	Username
Email:	Email
Password:	Password

As showed in the above image, the only change is the "Age" field in the Registration form. The customer must specify his age in order to complete the registration.

Statistics Interface



In this interface, has been added a bar chart in order to show to the employee the average age of customers that chose a specific preference. The above image shows the new configuration of the interface.