UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Web Applications, A.Y. 2020/2021
Master Degree in Computer Engineering
Master Degree in ICT for Internet and Multimedia

Homework 1 – Server-side Design and Development

Submission date: 23 April 2021

Last Name	First Name	Badge Number
Giurisato	Francesco	2027577
Basso	Marco	2005796
Piva	Matteo	2020352
Esposito	Vittorio	2005795
Del Fiume	Gabriele	2016821
Benetti	Alessandro	1210974

Objectives

The main goal of this web application is to offer people a way to experience what Italy has to offer through special ads created by small local businesses such as hotels, restaurants, theaters, museums and many more. Each experience gives the opportunity to earn a different number of points, in order to take advantage of the discounts. Furthermore, each user can leave a review of the activity purchased, allowing the company to continuously improve the service offered.

Main functionalities

The platform allows users to register with two different roles. If the user represents a business, he can register as a company profile so he will be able to post advertisements which will be booked by other users having the role of client. The client (or tourist) can search for experiences filtering them by type of advertisement, location and date. Each ad is distinguished by a score (defined by the company), which will be assigned to each customer once they have used the service. In addition to this, there is another score (linked to reviews) whose purpose is to help the customer book the offer that best suits his needs.
A user can therefore leave a vote on the advertisement together with a short description.

To achieve these functionalities we defined 3 areas:

1. Public area: available to everyone. It allows unregistered users to navigate through the homepage and search for advertisements and consult them.
2. Tourist/Clients area: this area is accessible by the users with the tourist role. With this role it's possible to:
    a. search and get information about an ad
    b. make a booking
    c. view and manage the reservations made
    d. leave feedback and comments about ads
    e. manage it's own profile
3. Companies area: from here it's possible to
    a. add ad with special offers (and score)
    b. manage existing advertisements
    c. view booking of an ad
    d. view feedback about ads

In addition there are the registration and login pages. Once the user is registered, he will receive a confirmation email.

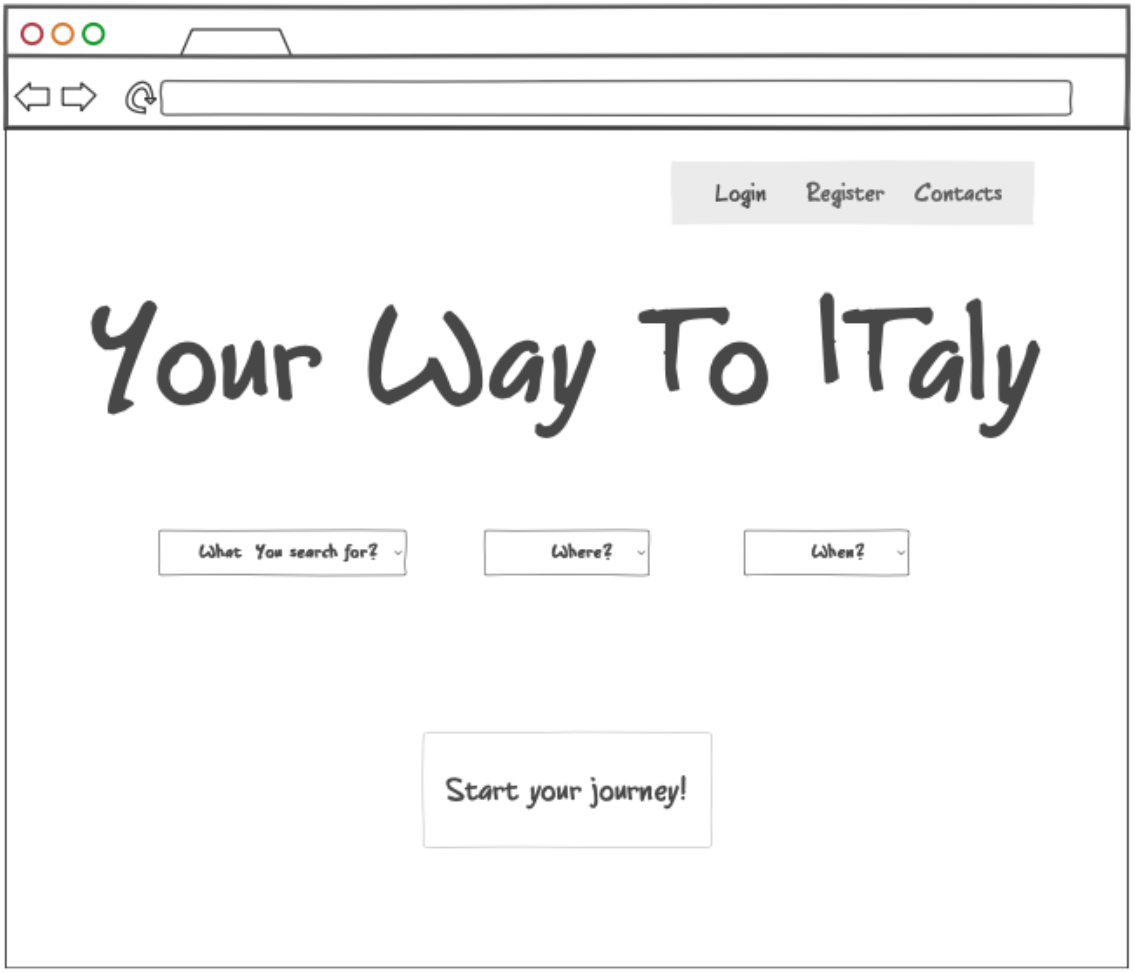
Presentation Logic Layer

The platform presentation is organised with mainly jsp pages with the following roles:

- Homepage [jsp]: contains the possibility to search for ads and visualize them, or to navigate to the Login/Register page. If the user is already logged in, he can jump to the User page.
- Login page [jsp]: allows the users login to the web application.
- Registration page [jsp]: allows the user to register to the web application.
- Show Advertisement [jsp]: allows you to see ad details, book it and leave a feedback.
- Show Profile [jsp]: allows to see information and the list of advertisements booked (Tourist role) or offered (Company role).
- Edit Profile page [jsp]: allows the user to edit It's own account information.
- Create Advertisement page [jsp]: allows a Company to add an advertisement.
- Upload Image page [jsp]: allows the company owner to upload some images about an ad.
- Edit Advertisement page [jsp]: allows a Company to edit an advertisement.
- Message page [jsp]: support page to display platform messages. This page is incorporated by the others.
- Contacts [html]: shows information about the developers.

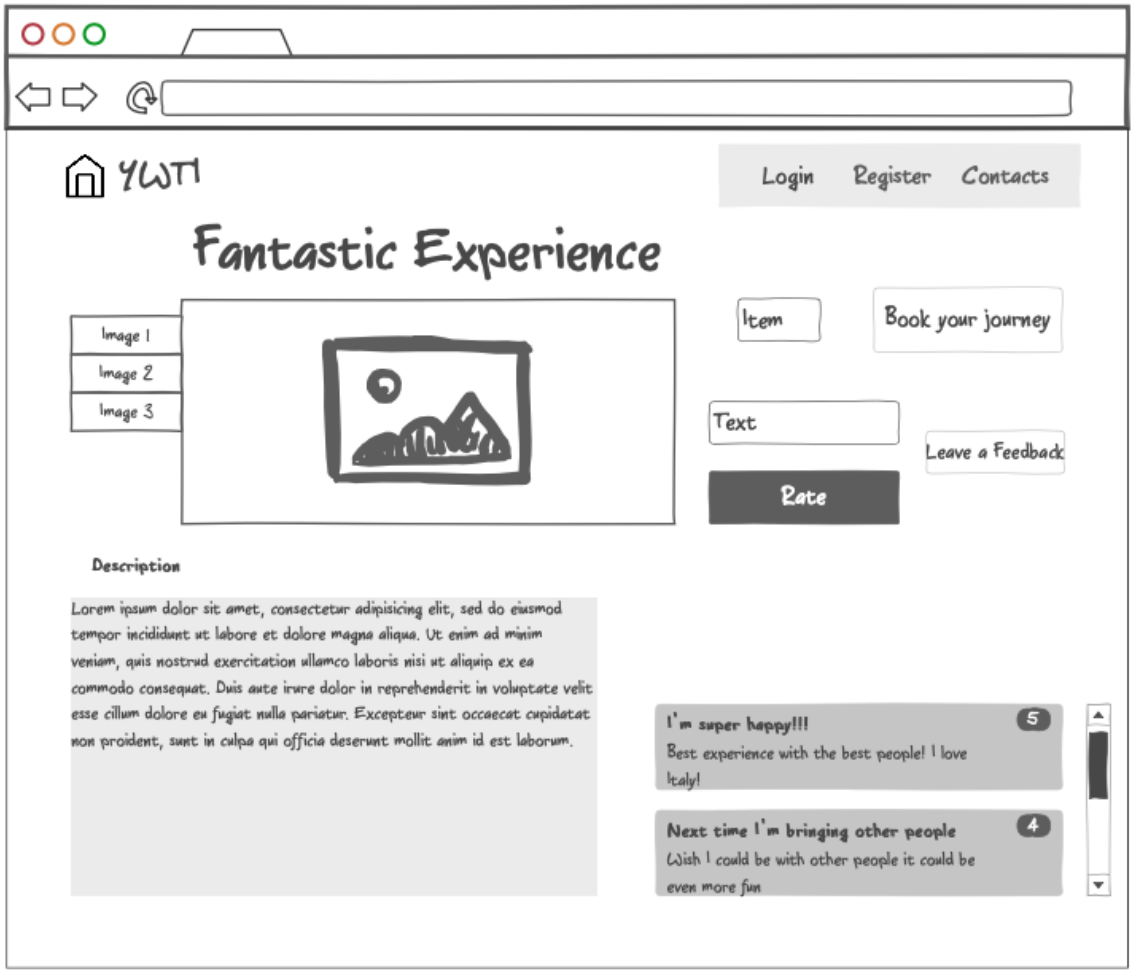
Home Page (Interface Mockup)

The home page allows the user, logged in or not, to start a search after selecting some search parameters. It is possible to access both the login or registration pages and the contacts page. In case the user is already logged in he can both access the user profile page to see all his information and logout. The homepage is organised with few elements to make it easy to use and clean.
It should be noted that in almost all the pages of the web application there is a button that the user can use to return to the “Homepage”.
It is important to remember that in the current implementation (mockup), to search for an ad based on its type and its city it is necessary to specify their associated IDs. In the final version, however, the client will have to choose between various types of ads and various cities through a drop-down menu that will show the respective textual values.



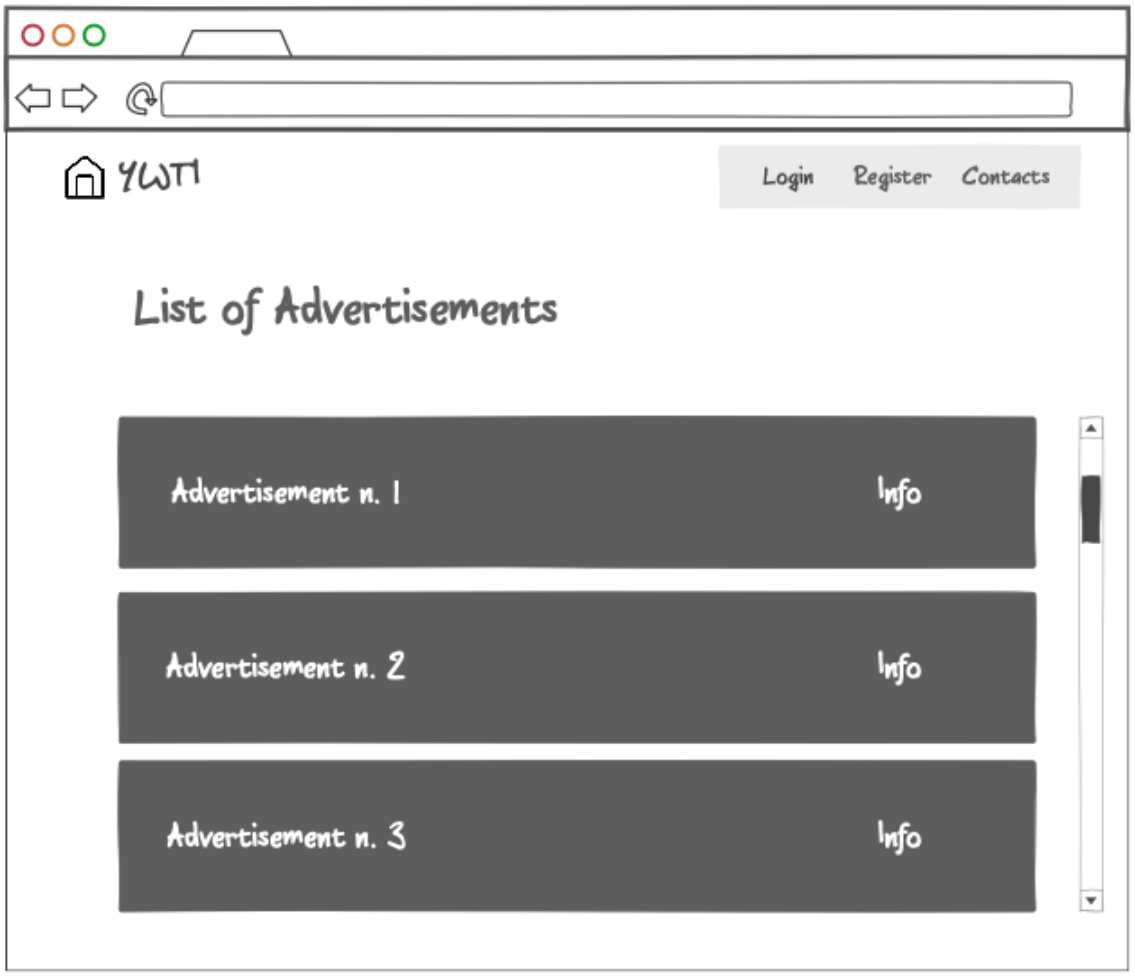
Advertisement Page (Interface Mockup)

Once the user has obtained the list of ads, he can select them one by one to get more information. Through the “Advertisement Page”, the user can view all the information of a specific ad. For instance he can see the description of the ad with the associated images. Moreover he can read the feedback posted by other registered tourists with the relative scores.
After entering the number of seats and the date for which he wants to book, the user can make the reservation using the "Book your journey" button. The tourist will be given a certain amount of credits, and a discount can be given based on some pseudo random criteria. The user has the possibility to cancel the reservation, but he will lose the points associated with the advertisement. Another feature, present if the tourist participated to the event, is the possibility to leave feedback, consisting of a text and a rate.



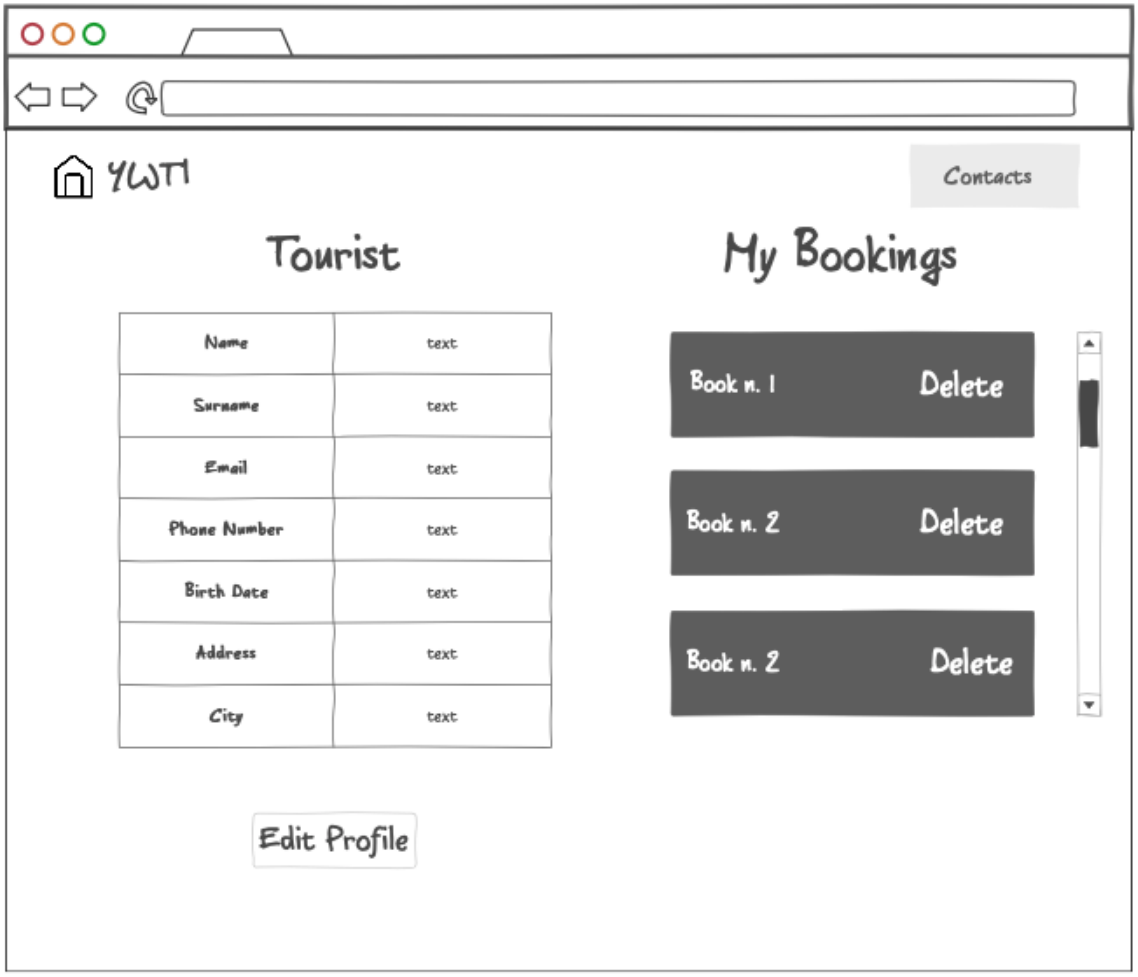
Search Results Page (Interface Mockup)

After completing the search form, the user has the opportunity to scroll through a list of ads that meet these requirements. Each ad on this list is characterized by its name and an "Info" button that allows the user to navigate to the “Advertisement Page”.
From this page, the user can also return to the “Homepage”. In case the user wants to register or log in, it is possible to use the buttons located at the top of the page. The user will then be redirected back to this page.

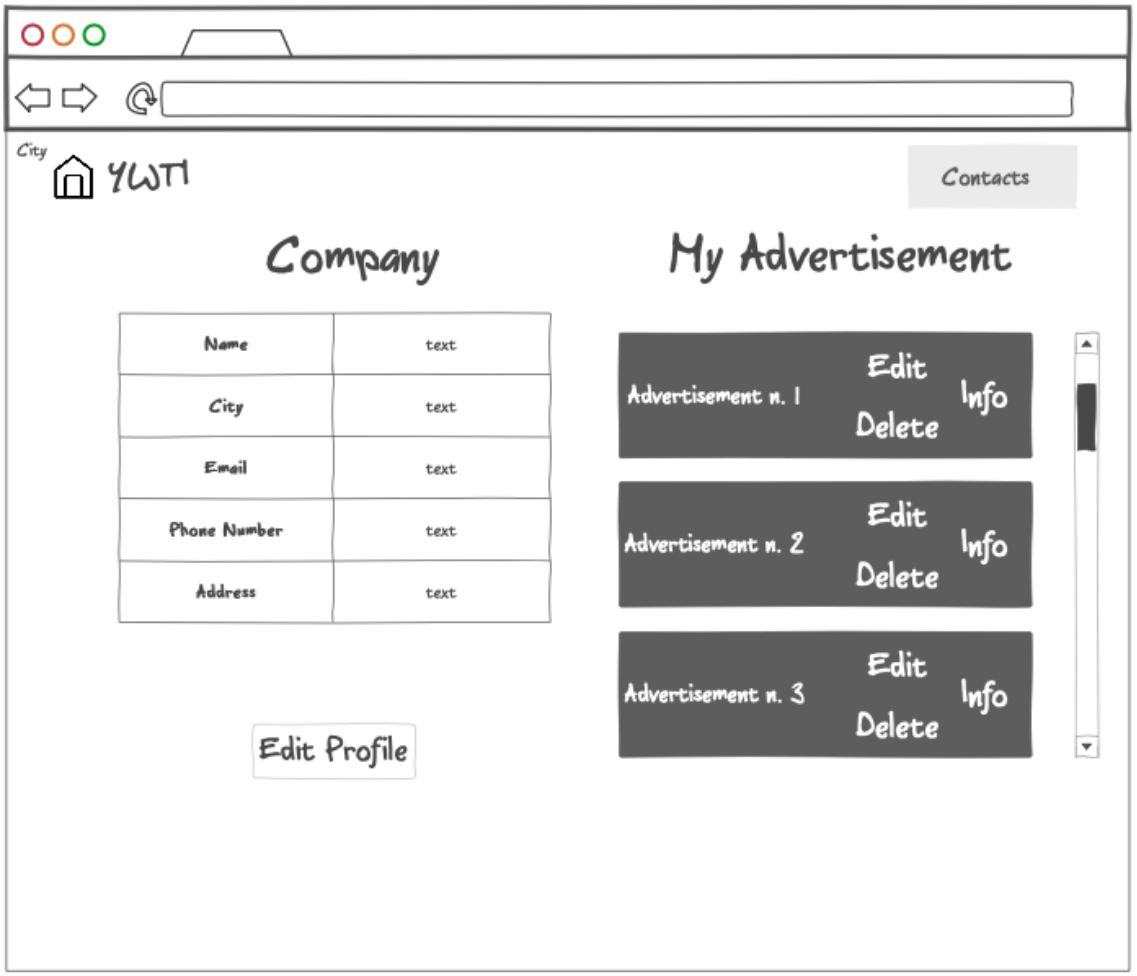


Tourist Profile Page (Interface Mockup)

The 2 profile pages (tourist and company) are listed below. For both types of users, their respective information will be shown, with the possibility of modifying the account. If a tourist is logged in, this page shows all his reservations and the total score. At this point the tourist can cancel the reservation relating to a particular advertisement. It is important to note that the cancellation of a reservation is only allowed if the event has not yet started.
On the company profile page, in addition to the account information, all the advertisements that have been published will be shown. In this case the company can modify or delete one of those present in the list or use the "Info" button to view all the information concerning it.

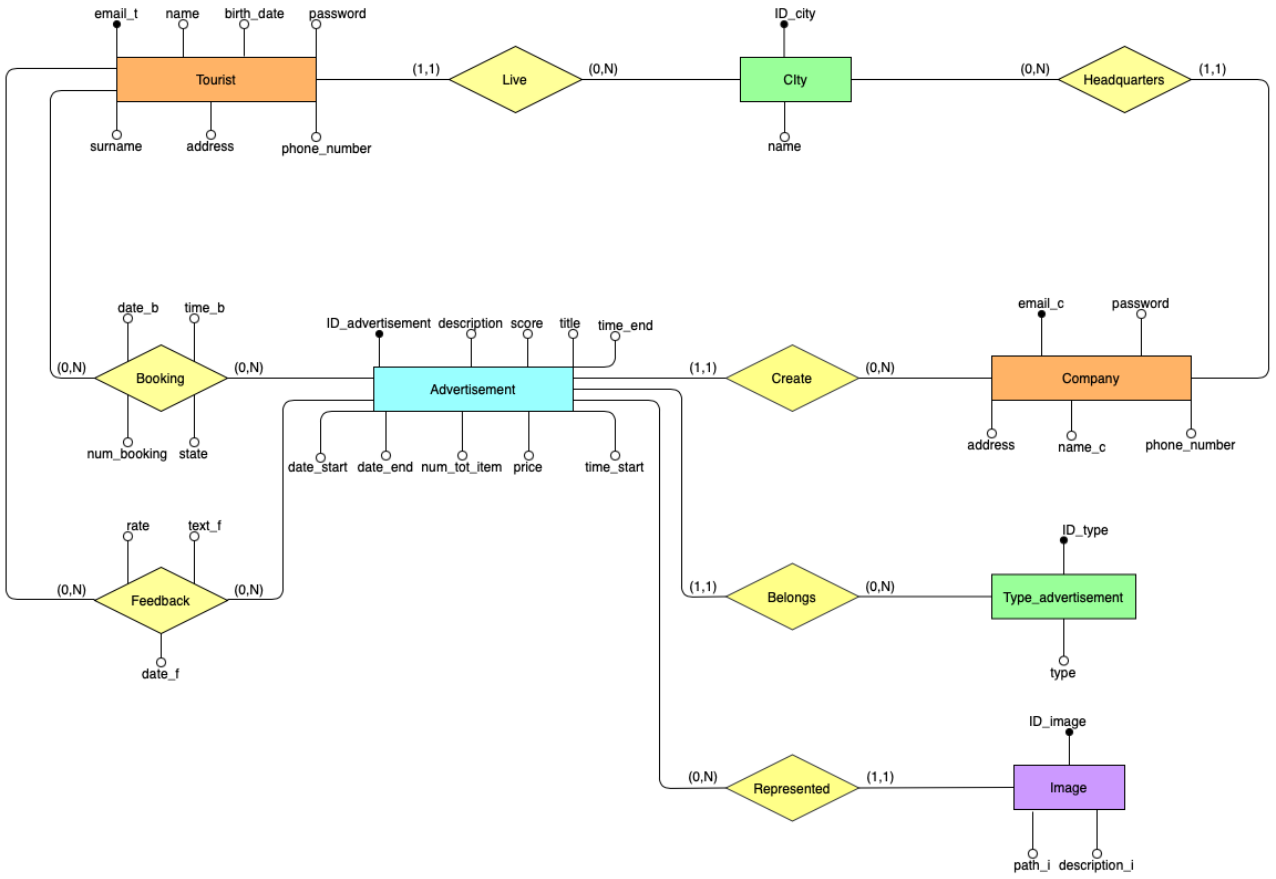


Company Profile Page (Interface Mockup)



Data Logic Layer

Entity-Relationship Schema



The entity-relationship schema contains 6 main entities:

- Advertisement: it describes the advertisements offered by the companies. Its primary key, “ID_advertisement”, is an auto increment integer and its two external keys are “ID_type” and “email_c”. The first is an integer value while the second is a varchar, because every advertisement belongs to a specific type (it is in relation 1-N with Type_advertisement entity) and it is created by a specific company (1-N relation). There are two (varchar) attributes, “title” and “description”, which give some general information about the ad. Then there are two (integer) attributes, “price” and “score”, used to indicate the price and the points that the booking gives to the tourist for a single item booked. The (integer) attribute “num_tot_item” is the amount of resources remaining that a tourist can book (i.e. the number of rooms in a hotel, or the number of free seats at the cinema). There are four more attributes, “date_start”, “date_end”, “time_start” and “time_end”, which contain respectively the starting and ending dates (date type) and times (time type).
- Tourist: each tourist has, as primary key, their “email”, which is of type varchar. For each tourist we also record their “name”, “surname” (of type varchar), “birth_date” (of type date), “address”, “phone_number” (of type varchar) and the “password”. Note that the password is hashed through MD5 before storing it in the database. Furthermore, the Tourist has an external key, “ID_city”, which refers to the italian city where the tourist resides.

- Company: as for the tourist, the company also has its “email” as primary key (of type varchar). For each company we also store their “name”, “address”, “phone_number” (of type varchar) and “password”. As before the password is hashed through MD5. Moreover It also has an external key, “ID_city”, which refers to the italian city where the company is located.
- City: it contains all the 110 italian cities. The primary key, “ID_city”, is an auto increment integer. It also has an attribute “name”, which refers to the name of the city. Each company is located in a specific city and also the tourist resides in one of them. These two relationships determine the two N-1 relations with company and tourist.
- Type_advertisement: it contains the 15 types of possible advertisements that a company can insert. The primary key, “ID_type”, is an auto increment integer and it also has the attribute “type”, which is a varchar that refers to the name of the type (i.e. restaurant, cinema, hotel). Each advertisement belongs to a specific type and for this reason it is in 1-N relation with advertisement.
- Image: it is used to store information about an image that a company decides to upload in his advertisement. The primary key, “ID_image”, is an auto increment integer. The attribute “path” is a varchar that contains the path to retrieve the image. There is also an attribute “description” (varchar) used in case a company wants to store a short description of the image. It also has an external key, “ID_advertisement”, that is the integer that identifies the corresponding advertisement (this is the meaning of the N-1 relation).

It also contains 2 N:N relationships which become two more entities in the logical implementation:

- Booking: a tourist can book more than one advertisement and the same advertisement can be booked by more than one tourist. This is the reason for the need to store information about bookings in the schema. The booking is identified by the pair “email_t” and “ID_advertisement”. For the booking we also store “date_b” and “time_b” (respectively of date and time types), which gives us more information about when a booking is created. We also set a “state” (of type varchar) which makes it possible for the user to cancel the reservation. The attribute “num_booking” (of type integer) is the amount of resources that the tourist wants to book (number of rooms in a hotel or number of seats at the cinema for example).
- Feedback: a tourist can also give feedback to the advertisements in which he participated. With the same reasoning of ‘booking’, a tourist can leave a feedback to more than one advertisement and the same advertisement can receive feedback from more than one tourist. As before, the feedback is identified by the pair “email_t” and “ID_advertisement”. The attribute “rate” is an integer number in the range 1-5 and “text” (of type varchar) is a review that allows the user to justify the rate inserted. We keep track also of the date in which the feedback is released with the attribute “date_f”.

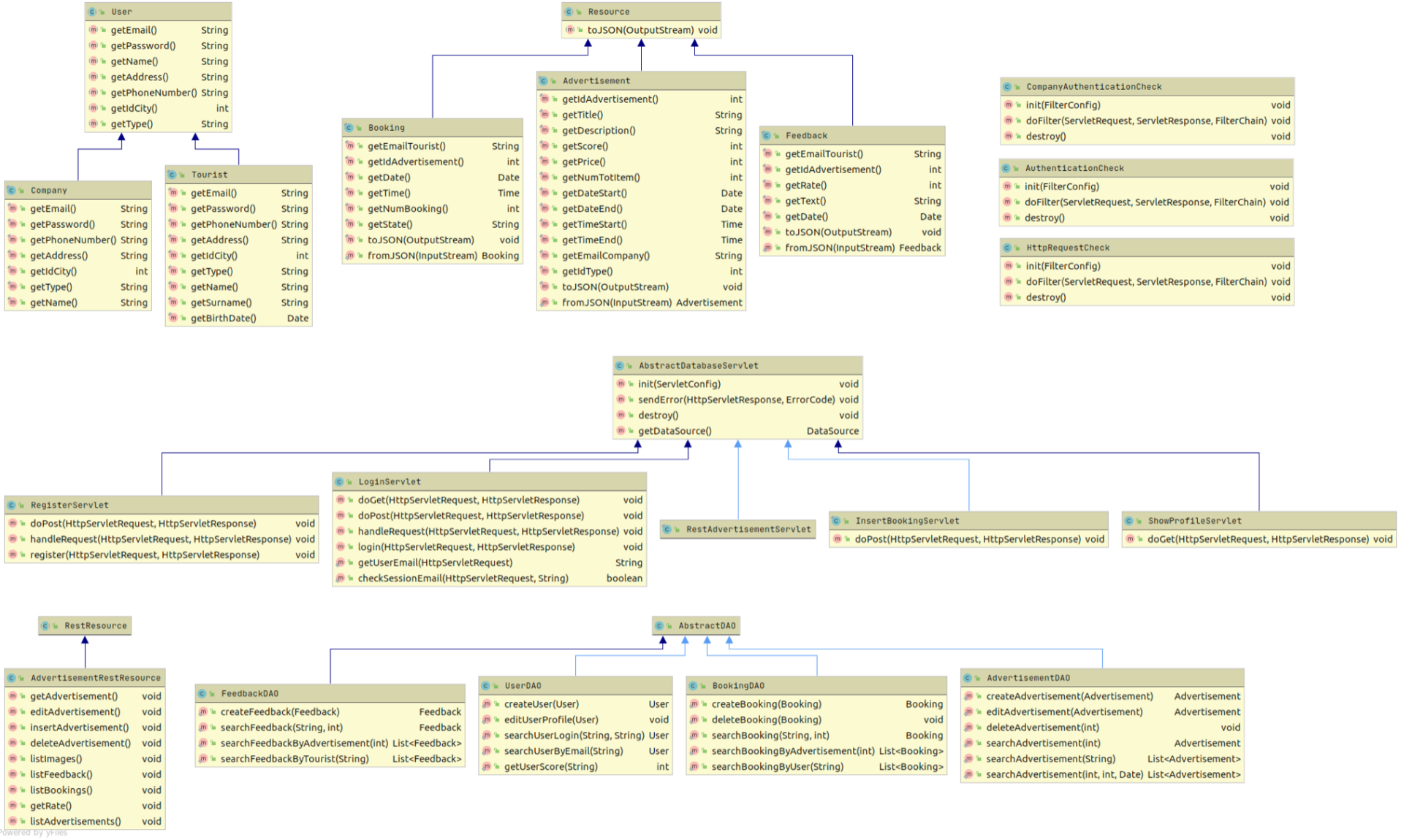
Other Information

It is possible to interact with most of the entities (insertion, deletion and update) directly via interface.
The only exception is for the City and Type_advertisement entities, which are pre-filled through specific files that respectively contain a list of all the italian cities and all the possible types of advertisement that we can host on the application (i.e. hotel, restaurant, cinema, etc).

Business Logic Layer

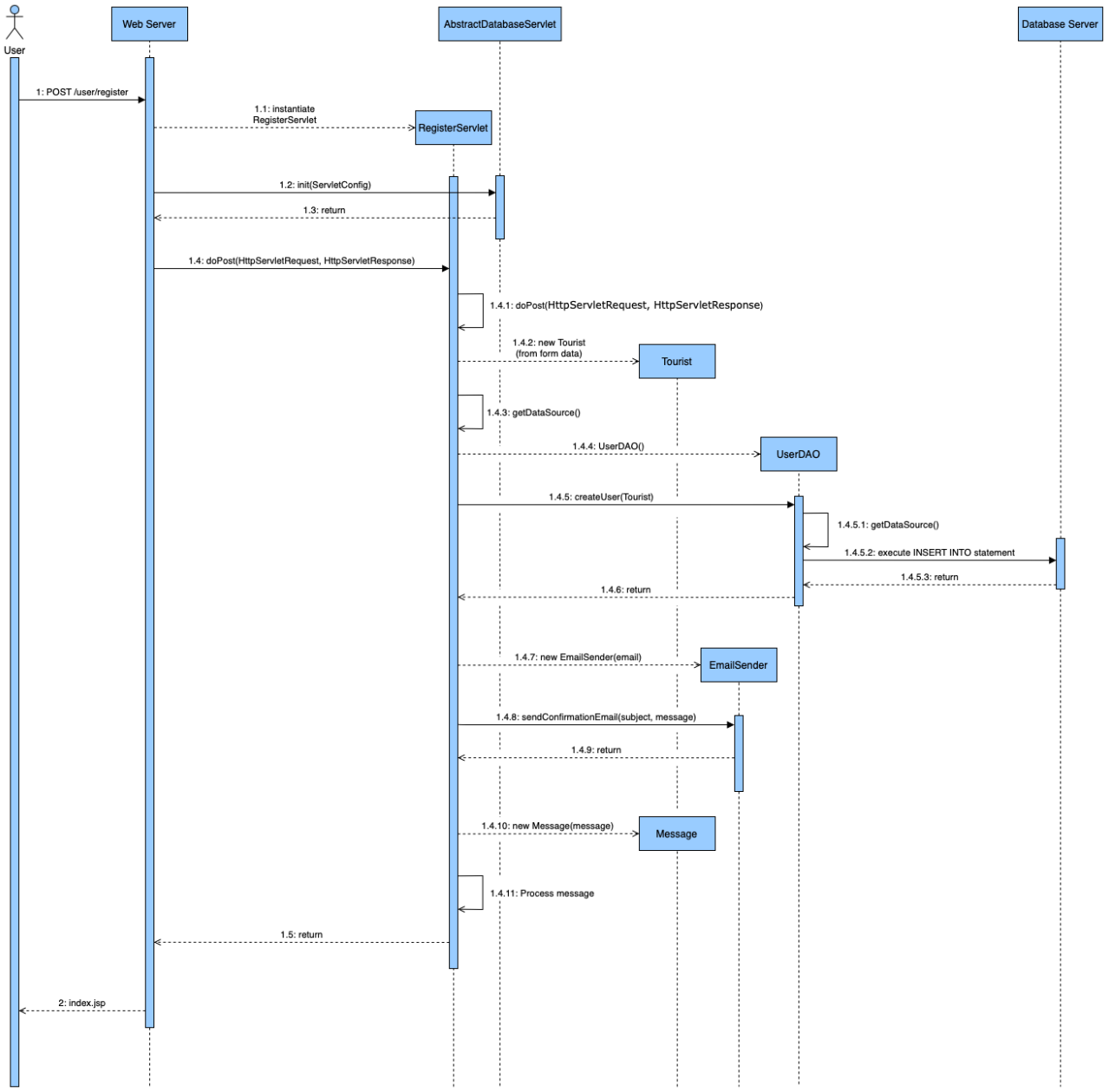
Class Diagram

The following image shows some of the main classes of the project. Some of these have been omitted for reasons of limited space. The upper part shows the classes belonging to the package resource, such as Tourist and Company. Both classes are derived from the User superclass. The reason for this implementation was that the two types of users have different attributes. In fact, the Tourist class has additional attributes such as "Surname" and "Birth date". In addition to these there are the Feedback, Advertisement and Booking classes. All three were used to complete the REST-based part of the project. Therefore, inside them, in addition to implementing the get methods (to obtain information about that particular object) there are the toJSON and fromJSON methods. The latter allow you to convert an object (of the type specified above) into a JSON object. This function is used when the client (for example a tourist) requests a page of the "show-advertisement" type. Since there are multiple resources on this page, the client will make multiple requests to obtain them. The server, to satisfy them, will create, starting from the requested objects, JSON type objects which will be sent as responses. Note how these 2 functions have as input parameters the streams from which / on which to receive / send messages.
The lower part shows the DAO (Data Access Object) classes. These classes are used by servlets to access and / or modify information contained within the database. In fact, various functions have been implemented that allow, for example, the search for information given the keys, create a new object and so on. Within these functions, queries have been used, that is, statements with which it is possible to query the database. Therefore, for the creation of a new user this statement will contain the term "INSERT", or to obtain some information "SELECT" will be used. Since these functions are sensitive, as they are able to alter the information in the database, authentication checks have been added to verify that the user is registered to the service and holds the permission.
These checks were implemented using the CompanyAuthenticationCheck class. As previously mentioned, 2 types of checks are carried out: the first is to verify that the user is logged in, while the second is to verify the type of user as there are different functions between Company and Tourist. In fact, assuming that the user is inside a page where the info of a certain advertisement is shown, if he is logged in as a Company he will not be able to leave feedback. This is because only tourists can leave comments on the ads.
Another type of check is carried out by the AuthenticationCheck class. The latter has the task of verifying that the request received by the server is an HTTP type.
Finally, in the central part there are the servlets. Of particular importance is the RestAdvertisementServlet class. This servlet is used to implement the REST paradigm. Inside, in addition to the header checks, the request is analyzed (both the URI and the method). For example, when the Company wants to create a new advertisement, a request characterized by the "POST" method will be received. If, on the other hand, the latter wishes to modify the data relating to its advertisement, it will make a "PUT" type request. It should be remembered that once the type of request has been identified, this class will invoke the AdvertisementRestResource class.
Regarding the REST paradigm, all requests and responses have been implemented in such a way to support JSON type objects.
Another important servlet is the one relating to the register function. In addition to the presence of the functions that manage POST requests, there is the possibility for the user to receive a confirmation email once registration has been successfully completed. The other classes present in the schema, once the type of request has been recognized, will use the classes described above (Resource and DAO) to provide the correct answer.


User
getEmail() String
getPassword() String
getName() String
getAddress() String
getPhoneNumber() String
getIdCity() int
getType() String
Company
getEmail() String
getPassword() String
getPhoneNumber() String
getAddress() String
getIdCity() int
getType() String
getName() String
Tourist
getEmail() String
getPassword() String
getPhoneNumber() String
getAddress() String
getIdCity() int
getType() String
getName() String
getSurname() String
getBirthDate() Date
Resource
toJSON(OutputStream) void
Booking
getEmailTourist() String
getIdAdvertisement() int
getDate() Date
getTime() Time
getNumBooking() int
getState() String
toJSON(OutputStream) void
fromJSON(InputStream) Booking
Advertisement
getIdAdvertisement() int
getTitle() String
getDescription() String
getScore() int
getPrice() int
getNumTotItem() int
getDateStart() Date
getDateEnd() Date
getTimeStart() Time
getTimeEnd() Time
getEmailCompany() String
getIdType() int
toJSON(OutputStream) void
fromJSON(InputStream) Advertisement
Feedback
getEmailTourist() String
getIdAdvertisement() int
getRate() int
getText() String
getDate() Date
toJSON(OutputStream) void
fromJSON(InputStream) Feedback
CompanyAuthenticationCheck
init(FilterConfig) void
doFilter(ServletRequest, ServletResponse, FilterChain) void
destroy() void
AuthenticationCheck
init(FilterConfig) void
doFilter(ServletRequest, ServletResponse, FilterChain) void
destroy() void
HttpRequestCheck
init(FilterConfig) void
doFilter(ServletRequest, ServletResponse, FilterChain) void
destroy() void
AbstractDatabaseServlet
init(ServletConfig) void
sendError(HttpServletResponse, ErrorCode) void
destroy() void
getDataSource() DataSource
RegisterServlet
doPost(HttpServletRequest, HttpServletResponse) void
handleRequest(HttpServletRequest, HttpServletResponse) void
register(HttpServletRequest, HttpServletResponse) void
LoginServlet
doGet(HttpServletRequest, HttpServletResponse) void
doPost(HttpServletRequest, HttpServletResponse) void
handleRequest(HttpServletRequest, HttpServletResponse) void
login(HttpServletRequest, HttpServletResponse) void
getUserEmail(HttpServletRequest) String
checkSessionEmail(HttpServletRequest, String) boolean
RestAdvertisementServlet
InsertBookingServlet
doPost(HttpServletRequest, HttpServletResponse) void
ShowProfileServlet
doGet(HttpServletRequest, HttpServletResponse) void
RestResource
AdvertisementRestResource
getAdvertisement() void
editAdvertisement() void
insertAdvertisement() void
deleteAdvertisement() void
listImages() void
listFeedback() void
listBookings() void
getRate() void
listAdvertisements() void
AbstractDAO
FeedbackDAO
createFeedback(Feedback) Feedback
searchFeedback(String, int) Feedback
searchFeedbackByAdvertisement(int) List<Feedback>
searchFeedbackByTourist(String) List<Feedback>
UserDAO
createUser(User) User
editUserProfile(User) void
searchUserLogin(String, String) User
searchUserByEmail(String) User
getUserScore(String) int
BookingDAO
createBooking(Booking) Booking
deleteBooking(Booking) void
searchBooking(String, int) Booking
searchBookingByAdvertisement(int) List<Booking>
searchBookingByUser(String) List<Booking>
AdvertisementDAO
createAdvertisement(Advertisement) Advertisement
editAdvertisement(Advertisement) Advertisement
deleteAdvertisement(int) void
searchAdvertisement(int) Advertisement
searchAdvertisement(String) List<Advertisement>
searchAdvertisement(int, int, Date) List<Advertisement>
Powered by yFiles

Sequence Diagram

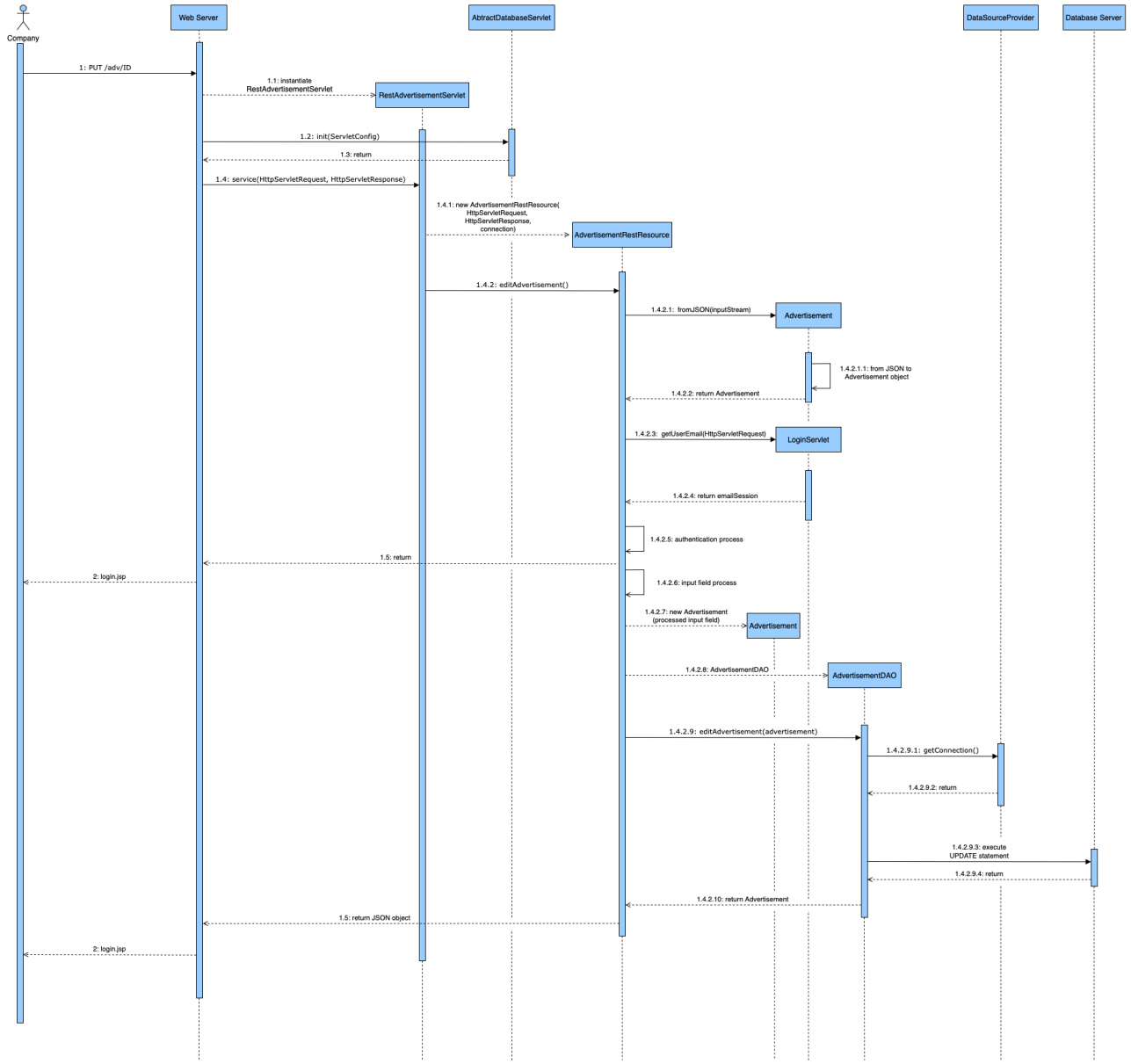
Register Class Diagram



The figure above shows the sequence diagram for registering a tourist. It is important to specify how the diagram shows the correct registration process. If invalid data is entered, the user receives one of the specific error messages. An example of error handling is shown in the second sequence diagram. A possible improvement that can be made to this function would consist in sending all these messages to make the user aware of all the fields that he has filled in erroneously.
The flow is as follows. The user makes a POST request to the web server, specifying the URI user/register.
The web server creates a RegisterServlet instance and calls its init method to get the data source from the JNDI context. At this point, through the mapping servlet present in the web.xml, the RegisterServlet is invoked.

It then calls the doPost methods passing the HttpServletRequest and the HttpServlet response. At this point the RegisterServlet analyzes the request and recognizes that it is a tourist registration operation and so the register method is called.
Before proceeding with the registration, several checks are carried out to verify that the data entered is correct. It is also checked whether the user is already registered using the email entered by him.
After these checks, the Tourist object is created with the supplied data passed through the Http request object. The Tourist Database object is created with the createUser() method of the UserDAO class passing as argument the newly created Tourist object. This method requires another connection, thanks to which the DAO User contacts the Database server and executes an INSERT INTO statement to create the tourist. Then the newly created tourist is returned to the RegisterServlet which checks if the process was successful. If not, an error will be generated and sent to the user in the form of a JSON object.
A new EmailSender object is therefore created getting the email settings from the JNDI context and passing the tourist's email as the argument. If the registration is successful, a confirmation email is sent to the tourist via the sendEmailConfirmation method, passing a couple of string objects as arguments. Furthermore, the RegisterServlet creates a new Message object setting as an argument "Successful registration!" which is processed and returned to the user on the index.jsp page. It is important to reiterate that in the event that an error occurs during the entire process, a message object is created containing the description of the error.

Rest Sequence Diagram



In this second diagram is reported the sequence diagram in which the company tries to edit an advertisement. Remember that the user will be authenticated through the CompanyAuthenticationCheck class before reaching the page to edit an advertisement. If the user is not logged in (and therefore has no session associated with him), he is redirected to the login page. If the user is already logged in, a check will also be made on his email. This check (carried out through the database) will verify the type of user. In case a tourist tries to modify the advertisement, he will receive an error message. Once the CompanyAuthenticationCheck class operations have been performed, the RestAdvertisementServlet is invoked.
The company executes a “PUT” request to the web server, specifying the URI advertisement/<ID>, where <ID> is the id of the advertisement that the company wants to edit. This parameter is obtained from the server, inspecting the JSON object (created from a javascript function by the client). On the server side there is a function (fromJSON) which takes care of receiving this object and converting it into a new object of type Advertisement. This way the server can then proceed with the actual change.

To provide an additional level of authentication (in addition to the actual existence of the account), when the company sends the data to modify the advertisement, it will be checked that it belongs to that company. This procedure was implemented to prevent a company from modifying the advertisements of other competing companies. For debugging purposes, these filters have been disabled, to allow you to test their actual functionality using the curl commands. When the project is completed (HW2), the filters will be restored and put into operation.
The web server instantiates the RestAdvertisementServlet class and calls the init method of the AbstractDatabaseServlet class to obtain the data source from the JNDI context. Then the service method of the RestAdvertisementServlet class is executed, passing to it the HttpServletRequest and the HttpServletResponse objects. At this point the RestAdvertisementServlet parses the request, verifying first that the syntax of the Header is correct (in particular, the "Content-Type" and "Accept" fields are analyzed) and then that it contains the proper method.
In this case, since the request method is of type “PUT”, RestAdvertisementServlet recognizes that the company is trying to edit an existing advertisement and initiates a new AdvertisementRestResource passing to it as parameters the HttpServletRequest, the HttpServletResponse, and the connection obtained. After creating this object, the editAdvertisement method is called.
As mentioned previously, there will be an additional authentication process at this point.
In case the user is correctly logged in and the session is valid, the AdvertisementRestResource initiates a new Advertisement object, passing to it the form data that has been obtained from the JSON object. The company can modify directly most of the fields, except the score. In fact, it is important to say that the score is correlated with the price. This field is modified according to the following formula: score = price / 3.14.
After checking that the data entered by the company are correct, the editAdvertisement method, of the AdvertisementDAO class, is invoked, to which the newly created object is passed. To reach the database and to perform correctly the query it is necessary to obtain a connection from the DataSourceProvider class. Obviously, the query that is used, is composed by the “UPDATE” statement. After executing the statement, the Advertisement object just inserted in the database is returned. Finally, this object is converted into a JSON object and sent to the company. Subsequent client-side implementations (using javascript) will allow the correct reception of this object.

Curl commands

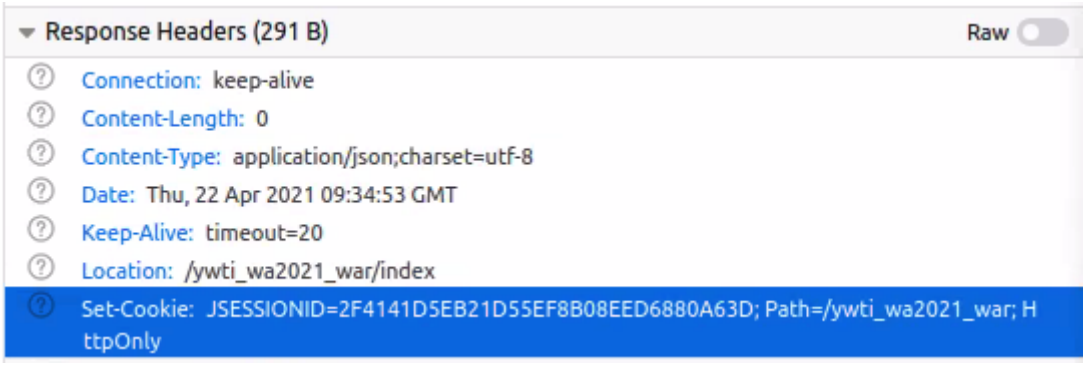
As mentioned earlier, mock JSP pages were used to test the features. To check the final version of the project developed in REST architecture, the curl commands were used. These commands are present in the test.pdf file and can be used to test the functioning of the REST paradigm. Examples are shown below. Inside the test.pdf file there is a complete list of all commands with their respective details.

curl -v -X GET -H "Accept: application/json" http://localhost:8080/ywti_wa2021_war/adv/1

curl -v -X GET -H "Accept: application/json" http://localhost:8080/ywti_wa2021_war/adv/1/rate

curl -v -X GET -H "Accept: application/json" http://localhost:8080/ywti_wa2021_war/adv/1/image

It is also possible, knowing the cookie ID of the current session, to access protected resources. In order to do this, you must therefore be logged in correctly.
Once logged in it is possible to obtain the session ID, using the developer tools present in the browser. Below is an example of the procedure to follow.



Below is the command used to modify the announcement:

curl -v --request PUT -H "Cookie:JSESSIONID=2F4141D5EB21D55EF8B08EED6880A63D" -H "Accept: application/json" -H "Content-Type: application/json" -d "{\"advertisement\":{\"idAdvertisement\":\"1\",\"title\":\"Da Pino Delicious dinner in the Dolomites, da Pino\",\"description\":\"Sette giorni di relax in mezzo alla neve delle dolomiti in Trentino alto adige a capodanno\",\"score\":\"100\",\"price\":\"800\",\"numTotItem\":\"29\",\"dateStart\":\"2021-12-28\",\"dateEnd\":\"2021-12-28\",\"timeStart\":\"14:14:14\",\"timeEnd\":\"14:14:15\",\"emailCompany\":\"hotelcentrale@gmail.com\",\"idType\":\"6\"}}" http://localhost:8080/ywti_wa2021_war/adv/1

REST API Summary

The project structure has been developed to implement the REST paradigm to both edit and show the Advertisement resources. Endpoints associated with advertisements are built in such a way that all the information needed to satisfy the request are directly included in the URI. The following illustrates the logical operation of a request from the user, to see the information relating to a certain advertisement. In this scenario, the user can be either on his own profile page (valid only for companies), or on the page containing the results of the search just carried out. Once the "Info" button is pressed, the user is redirected to a dynamic page. In fact, through the use of javascript (implementation left to the second part of the homework), the client (user) is able to make multiple requests for various resources. The latter are for example the list of feedbacks, the average rating of the service relating to the selected ad, the list of images made available by the company and so on. It is important to note that all these requests are satisfied from the backend through the generation of JSON objects.
The table below shows the different URIs that will be used to obtain the multiple resources. Unlike the Web 1.0 architecture, in which, for example, the JSP pages are structured and completed entirely on the server side, the Web 2.0 architecture allows the pages displayed by the user to be completed dynamically by making multiple requests to different resources.
For our project we have therefore decided to adopt a mixed implementation. In fact, as previously mentioned, REST has been implemented to create, view, modify and delete an advertisement. For the other parts of the project, however, we applied a Web 1.0 approach. An example of these is the page for viewing the profile. In this, therefore, there will not be multiple requests to obtain the resources, but there will be a JSP page completely created on the server side. The exchange of resources involving servlets and JSP pages is handled using the setAttribute and getAttribute methods of the HttpServletRequest class.
In order to test the web application and to perform the debug, we have written some curl commands to simulate the client side. Through javascript, these objects will be displayed within the page. Furthermore, by using the curl commands, we have simulated the different requests that the client can make in such a way as to test the functionality of the various URIs shown in the table.

To allow a correct workflow of the web application, authentication filters have been introduced. In this way, access to different parts of the site is limited to authorized users only. The project therefore includes 3 filters. The first 2 are used to verify and authenticate the user. The third instead has the task of verifying the type of requests that are made to the server. In fact, if the requests received by the server are not of the HTTP type, they will not be processed causing an exception to be thrown by the HttpRequestCheck class.

URI	Method	Description
/adv-create	POST	Allows the company to create a new Advertisement, passing data through a JSON object.
/adv/ID	GET	Server sends back to the client a JSON object containing the requested advertisement.
/adv/ID	PUT	Allows the company to edit an advertisement with a certain ID. Once the edit is done by the DAO, the server sends back to the client a JSON object containing the edited advertisement
/adv/ID	DELETE	Allows the company to delete an advertisement. Tourists will not be allowed any longer to view this ad.
/adv	GET	Allows the user to view a list of advertisements whose parameters correspond with the fields entered.
/adv/ID/rate	GET	Returns the average rating, entered by tourists, of the specific ad .
/adv/ID/image	GET	Returns all the images entities in JSON format belonging to the specific ID.
/adv/ID/feedback	GET	Returns all the feedback that have been posted by the tourist.
/adv/ID/booking	GET	Allows the company to see all the bookings of its ads.

REST API Error Codes

In the table below we can find the list of errors defined in the application. Application specific errors follow a progressive numeration starting from -1. METHOD NOT ALLOWED errors are identified with the error code -40. And finally we have internal errors with the error code -100 (crashes, servlet exceptions and problems with the input/output streams). An example of an error code that is used when a user tries to fill in a field by entering incorrect information is the "Wrong Format" (Error Code: -1). An example of an error that may occur regarding the REST architecture, is when a company tries to create or modify an advertisement by mistake. In this case the company will receive as a response a JSON object containing the parameters of the error message. If an error is caused by the throwing of any exception, the message object will contain the description of that exception just for debug purposes.

Error Code	HTTP Status Code	Description
-1	BAD_REQUEST	Wrong format.
-2	NOT_FOUND	Advertisement not found.
-3	BAD_REQUEST	One or more input fields are empty.
-4	BAD_REQUEST	The email is missing.
-5	BAD_REQUEST	The password is missing.
-6	CONFLICT	Different password inserted.
-7	CONFLICT	Email already used.
-8	NOT_FOUND	No results found.
-9	NOT_FOUND	User not found.
-10	BAD_REQUEST	The input json is in the wrong format.
-11	UNAUTHORIZED	User not allowed.
-12	CONFLICT	Booking already done.
-13	CONFLICT	Feedback already done.
-20	BAD_REQUEST	Operation unknown.
-40	METHOD_NOT_ALLOWED	The method is not allowed.
-100	INTERNAL_SERVER_ERROR	There was an internal error that the server was not able to manage.

REST API Details

We report here 3 different operations that our web application handles during its functioning. It is necessary to remember after the creation of a new advertisement it is possible to upload one or more images referring to it. Once the advertisement has been successfully created, the client (company) will receive a JSON object containing the message "Images successfully uploaded!". In HW2 it will therefore be necessary to implement a script to manage the receipt of the message.

Advertisement

The following endpoint allows inserting a new advertisement into the database.

- URL
  /adv-create

- Method
  POST

- URL Params
  No url params required

- Data Params
  Required:
  - `title = {string}`
    The title of the ad.
  - `description = {string}`
    The description of the ad.
  - `score = {integer}`
    The score that the ad gives.

- `price = {integer}`
- `numTotItem = {integer}`
  Total available items bookable.
- `dateStart = {date}`
- `dateEnd = {date}`
- `timeStart = {time}`
- `timeEnd = {time}`
- `emailCompany = {string}`
  The email of the company owner of the ad. It should belong to companies already inserted in the dataset.
- `idType = {integer}`
  The identifier of the type of the advertisement. It should belong to types already inserted in the dataset.

Optional:
- `none`

- Success Response

  Code: 200
  Content: The server sends back to the client a JSON object containing the advertisement.

- Error Response

  Code: 400 BAD_REQUEST
  Content: `{"error": {"code": -10, "message" : "The input json is in the wrong format."}}`
  When: the user has passed in the data payload a json ill-formatted or without all the required fields.

  Code: 400 BAD_REQUEST
  Content: `{"error": {"code": -1, "message" : "Wrong format."}}`
  When: the user has inserted a value not valid in one of the required fields.

  Code: 500 INTERNAL_SERVER_ERROR
  Content: `{"error": {"code": -100, "message" : "Internal Error."}}`
  When: there is a SQLException or a NamingException, this error is returned.

Advertisement

The following endpoint allows the user to view the advertisement identified by the ID passed in the URL.

- URL
  /adv/ID

- Method
  GET

- URL Params
  Required:
  - `idAdvertisement = {integer}`
    The identifier of the advertisement.

- Data Params
  No data is passed when requiring this method.

- Success Response

  Code: 200
  Content: `{"advertisement":`

```
{
  "idAdvertisement":"5",
  "title":"Delicious dinner in the Dolomites, da Pino",
  "description":"This is a mock advertisement. Requests to
  the REST web server will be made in order to show the
  desired advertisement.",
  "score":"6",
  "price":"19",
  "numTotItem":"13",
  "dateStart":"2021-04-19",
  "dateEnd":"2021-07-01",
  "timeStart":"18:30:00",
  "timeEnd":"23:30:00",
  "emailCompany":"hotelcentrale@gmail.com",
  "idType":"4"}
}
```

- Error Response

  Code: 400 BAD_REQUEST
  Content: `{“error”: {“code”: -1, “message” : “Wrong format.”}}`
  When: the user has inserted a value not valid in one of the required fields.

  Code: 404 NOT_FOUND
  Content: `{“error”: {“code”: -2, “message” : “Advertisement not found.”}}`
  When: the ID passed in the URL is not valid.

  Code: 500 INTERNAL_SERVER_ERROR
  Content: `{“error”: {“code”: -100, “message” : “Internal error.”}}`
  When: if there is a SQLException or a NamingException, this error is returned.

Advertisement

The following endpoint allows the user to delete the advertisement identified by the ID passed in the URL. It should be remembered that deleting an advertisement consists in setting the number of bookable seats equal to 0.

- URL
  /adv/ID

- Method
  DELETE

- URL Params
  Required:
  - `idAdvertisement = {integer}`
    The identifier of the advertisement.

- Data Params
  No data is passed when requiring this method.

- Success Response

  Code: 200
  Content: JSON object containing the message “Advertisement successfully deleted”.

- Error Response

  Code: 404 NOT_FOUND
  Content: `{"error": {"code": -2, "message" : "Advertisement not found."}}`
  When: the ID passed in the URL is not valid.

  Code: 401 UNAUTHORIZED
  Content: `{"error": {"code": -11, "message" : "User not allowed."}}`
  When: the email of the user is different from the one of the owner of the advertisement.

  Code: 500 INTERNAL_SERVER_ERROR
  Content: `{"error": {"code": -100, "message" : "Internal error."}}`
  When: if there is a SQLException or a NamingException, this error is returned.