UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Web Applications A.Y. 2021-2022

Homework 1 – Server-side Design and Development

Master Degree in Computer Engineering
Master Degree in Cybersecurity
Master Degree in ICT for Internet and Multimedia

Deadline: 22 April, 2022

Group WAD-Team	**Electromechanics shop** A web application to manage a store 	
Last Name	**First Name**	**Student Number**
Bastasin	Simone	2053080
Bortolin	Simone	2038512
Ceccon	Gioele	2055040
Colussi	Davide	2056096
Czaczkes	Gil	2062036
Lando	Matteo	2055717
Nicoletti	Gianpietro	2053042
Pastore	Alessandra	2053082
Pozzer	Matteo	2053863

1 Objectives

The objective of the project, in continuation with what developed during the Foundation of Databases course by the e-team group, is to develop a web interface to manage an electromechanics shop. The web application indeed manages the products sold by the shop, the orders placed and the tickets opened by the customers.

2 Main Functionalities

The web application can be used by both registered and unregistered customers. While the first ones can buy products, check their invoices and ask for assistance through the opening of a ticket, the unregistered customers can only navigate its homepage and inspect products features. In addition to registered customers, each employee also has an account to log in.
Below we specify better the roles:

- **Customers**: they can register to our web application using a specific form. Only once registered, they can buy products, inspect the online invoices related to their orders, request for assistance tickets, modify their account changing the information provided during the registration.
- **Employee**: their account is provided by the administrator of the web application, and they can do different things according to their role.
  - **Sellers**: they manage products, adding new ones and modifying their details (e.g. name, brand, price, discount...). Furthermore, they manage the order placed by customers.
  - **Technicians**: they manage the incoming requests for assistance opened by customers (Assistance Tickets). Thus, they update the status of the tickets.
  - **Accountants**: they manage the payments done by customers and generate the invoices.
  - **Administrators**: they are to be considered the *superuser* of the web application. They own all the permissions of the other users and, in addition, they can create a new account for an employee user.

  In practice, to simplify the development of the project, we implemented only a generic employee (except for the administrator), but we kept into account the role subdivision of the employees in order to maintain a logical division between the functionalities. With "manage" we intend the main operations: Insert, Update, Delete, Select. Only for the products we decided to not implement the delete operation in order to maintain the history of the other information related to it (i.e. orders and tickets). In fact, also if it is possible to "delete" a product from the product management page, this will just set to 0 the quantity in stock of the product.

3 Data Logic Layer

3.1 Entity-Relationship Schema

We decided to develop our project using as a stepping stone the database designed by the e-team group during the FTB course. The original database's ER schema can be seen in Figure 1, while the relational schema is presented in Figure 2.

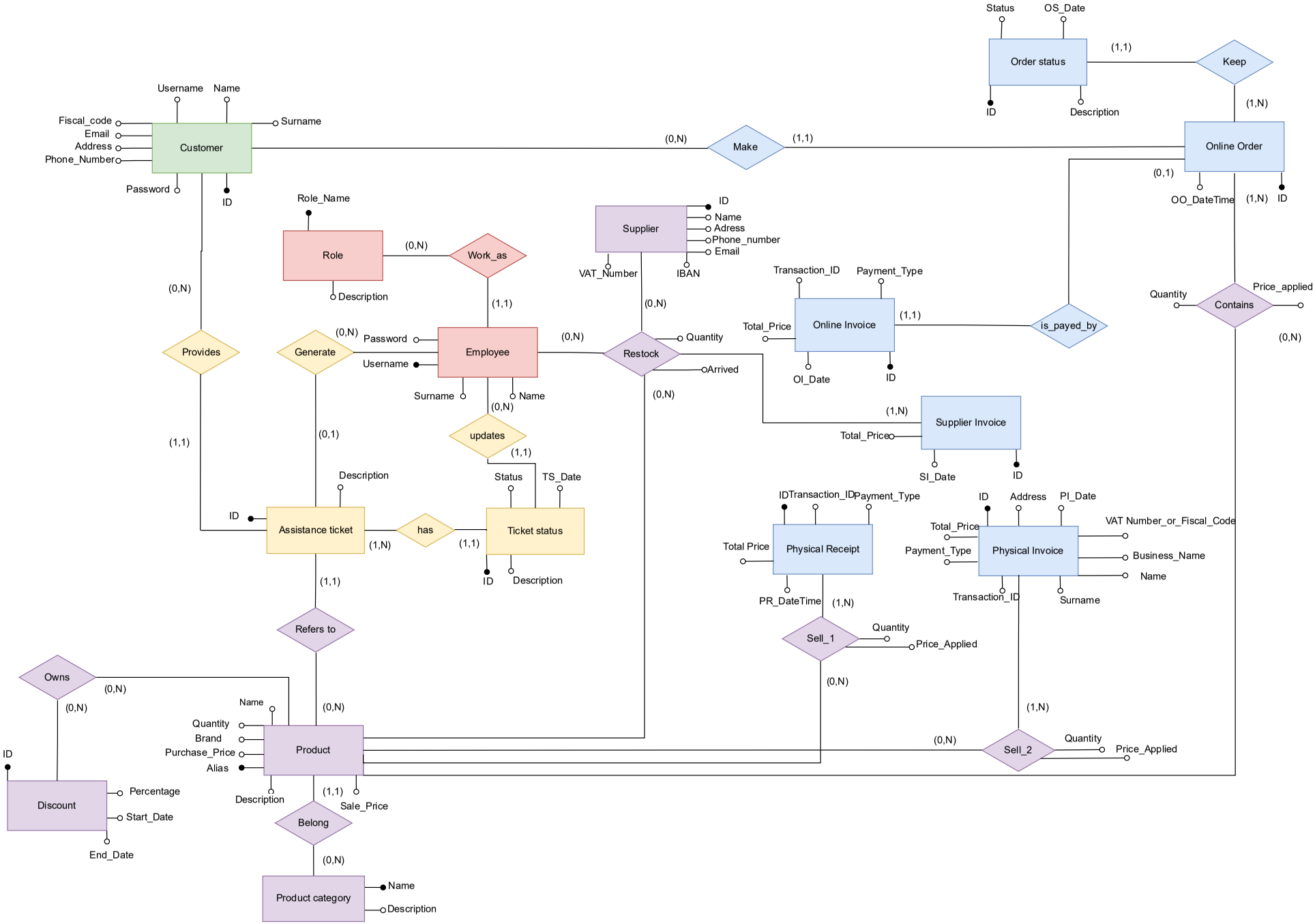

Figure 1: Original ER schema designed by e-team

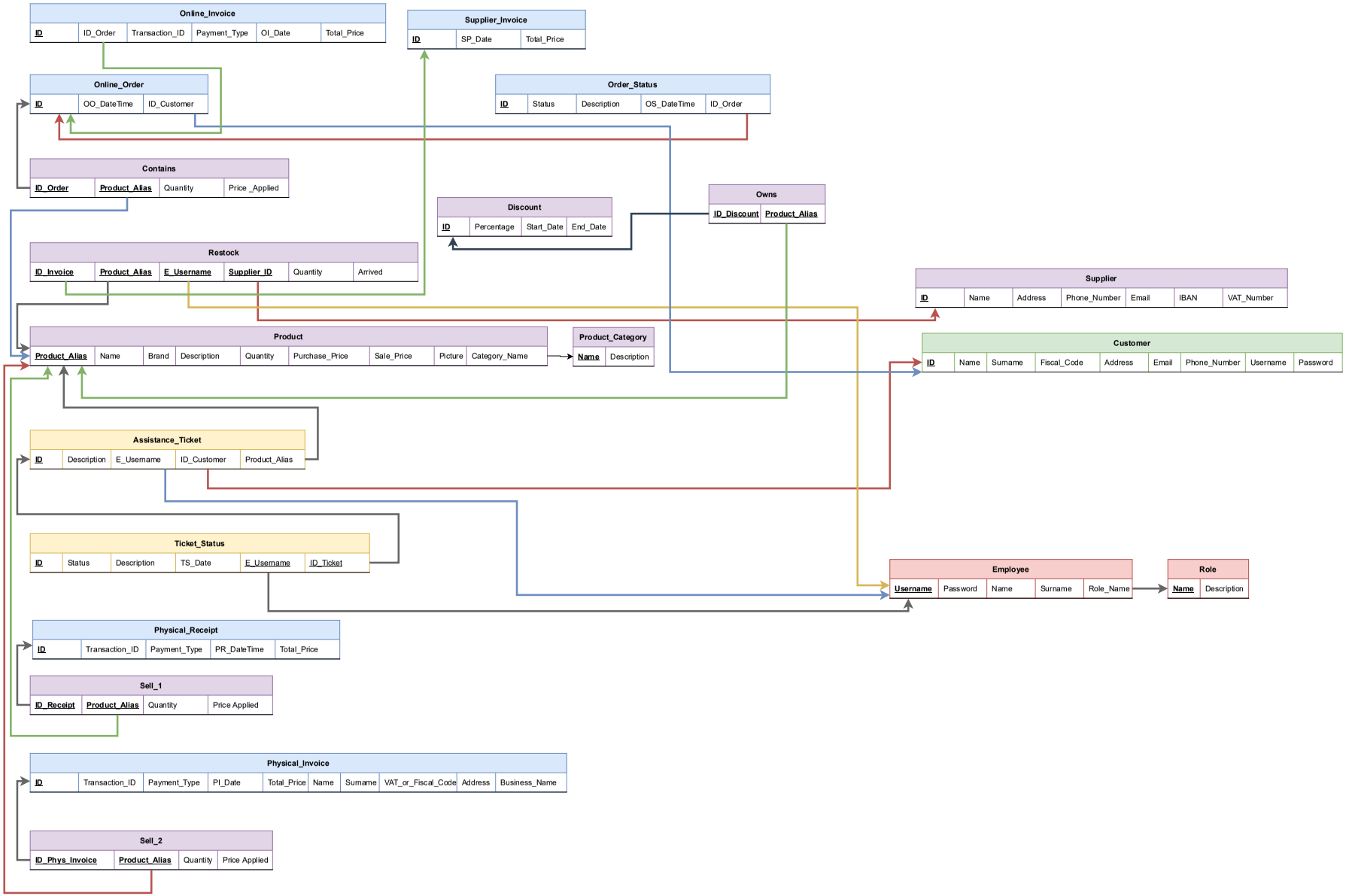

Figure 2: Original Relational schema designed by e-team

In order to maintain the project manageable, we decided to remove some relations. In particular, we decided to remove all the "offline services" and to build our web application only to manage the online side of the shop. We also removed the management of the suppliers, dedicating the development of our project only to the services seen by the client of the shop (excluding the management of the employees that is needed in our web application since we needed them to set up the permissions). In Figure 3 and 4 are reported the final version of the design for the data layer (ER schema and relational schema respectively).

3.2 Other Information

The database has been modified, compared to the one created by the e-team, improving the management of images, transforming it from hard-coded within the records of the product, to separated with an N:N association, this allowed to realize a more efficient and better management on the web app side.

The ticket part has been modified to make it 1:N (from 1:1) so it can contain all the information about the ticket history.

For the discount visualization, we thought about implementing a view, but this could have caused problems with the storage of the list in cache and it could have slowed down the uploading phase, so in the end we preferred a nested query.

For the management of orders, the relation has become 1:1 (from 1:N), because it was not considered useful to save all the history.

Finally, nullable values, defaults, and small writing errors discovered during the implementation of the application were corrected.

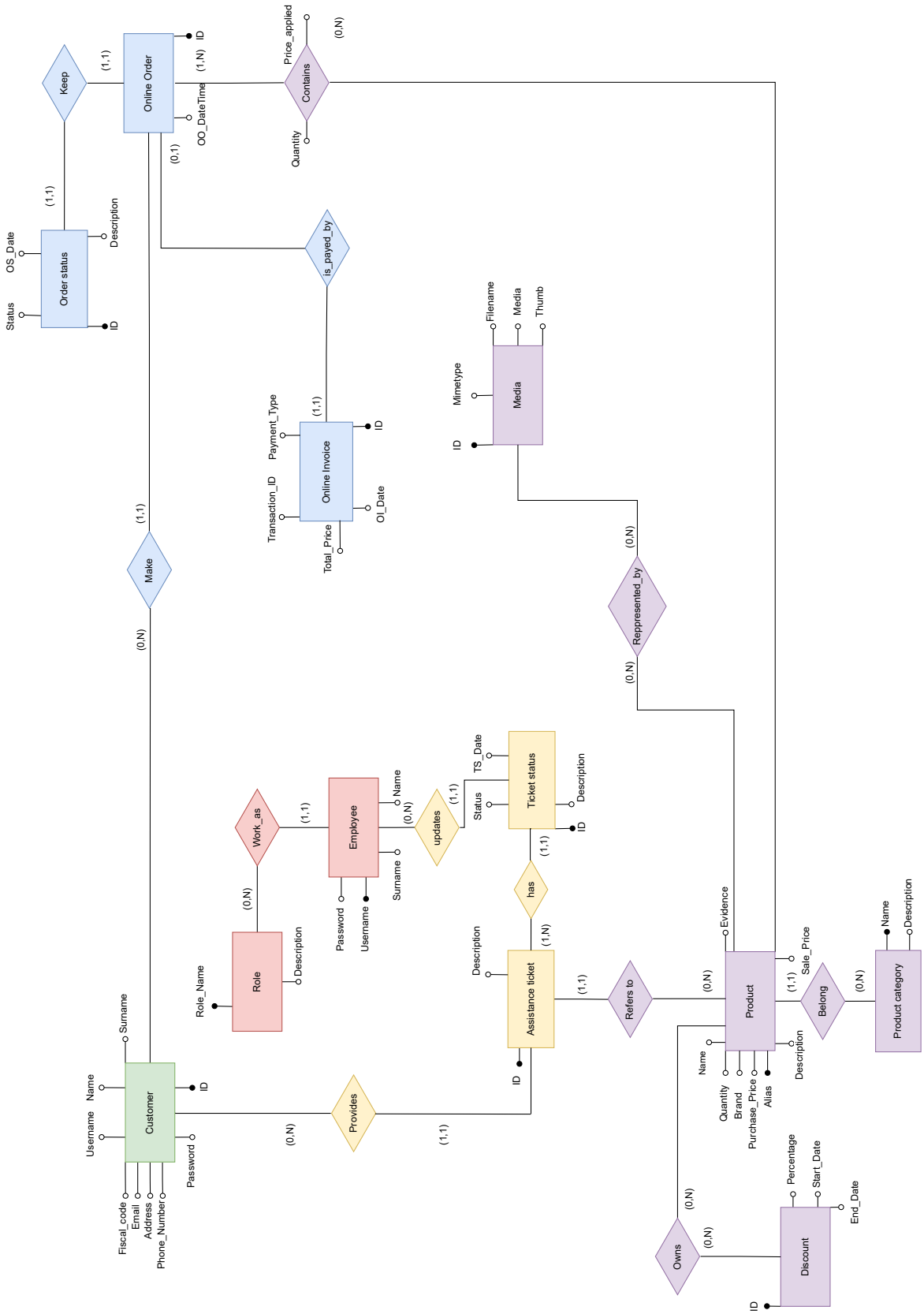

Figure 3: ER schema modified

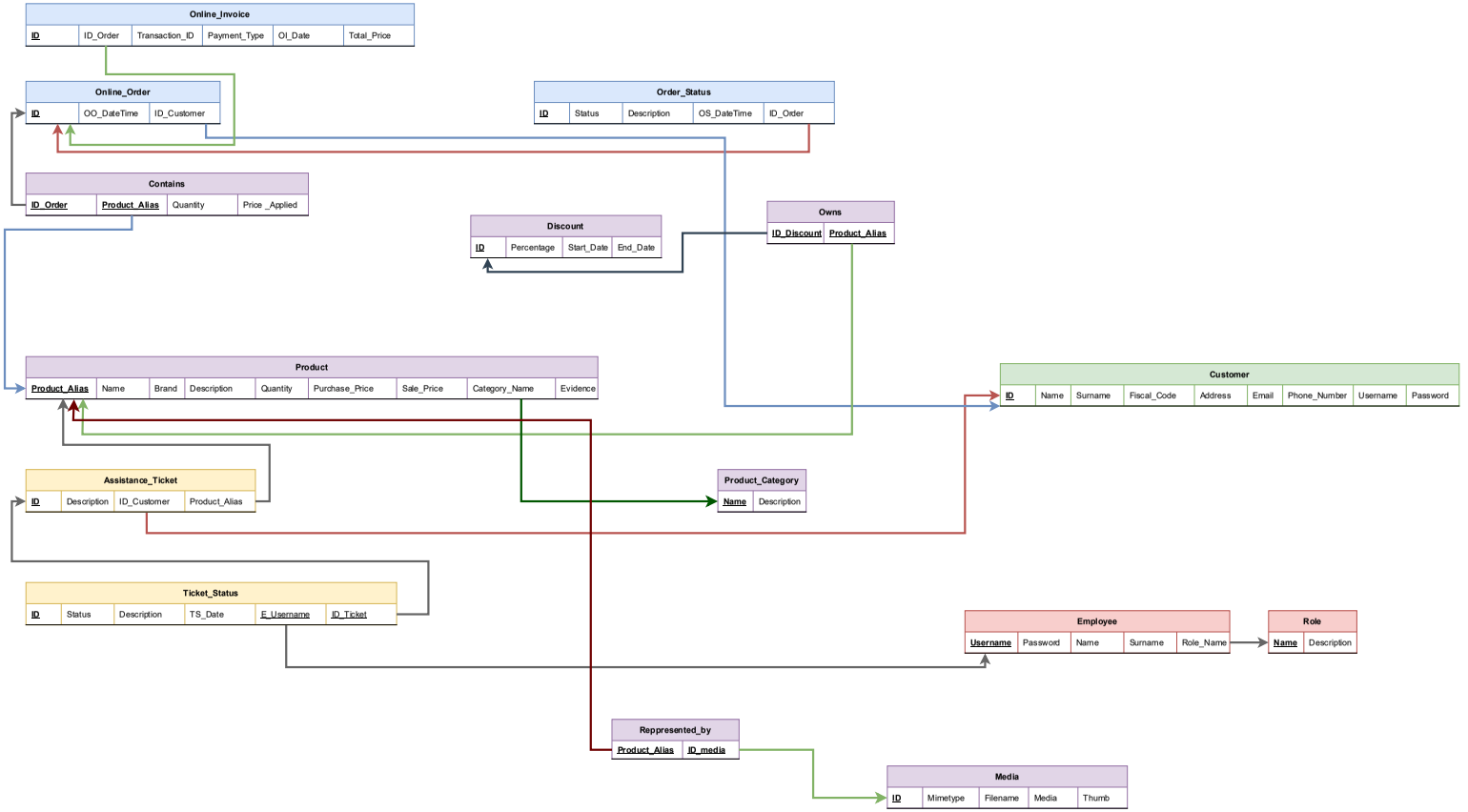

Figure 4: Relational schema modified

4 Presentation Logic Layer

The website is divided into the following pages:

- **Homepage**: contains the showcase of the featured products and the available products.
- **Product page**: allows the customer to inspect all the information about a given product. To buy the product the customer has to be logged in.
- **Login page**: allows both customers and employees to login.
- **Order list and Order details page**: allow each customer to look at their order history, to open a ticket, cancel the order or get the invoice.
- **Tickets page**: allows each customer to look at their ticket history. There exists an employee version of the page to manage each ticket.
- **Product management pages**: both simple employees and administrators are able to look at all the products and create/edit/delete them. Through a second page (Discount management) it is also possible to add some discounts to the products.
- **Orders management page**: both simple employees and administrators are able to look at all the orders and edit/cancel them. It's also possible to look at the existing invoices for each product.
- **User edit page**: allows both customers and employees to modify their personal data.
- **Administration page**: allows the administrators to edit the name, surname, role of all the users and even delete their account.

4.1 Homepage

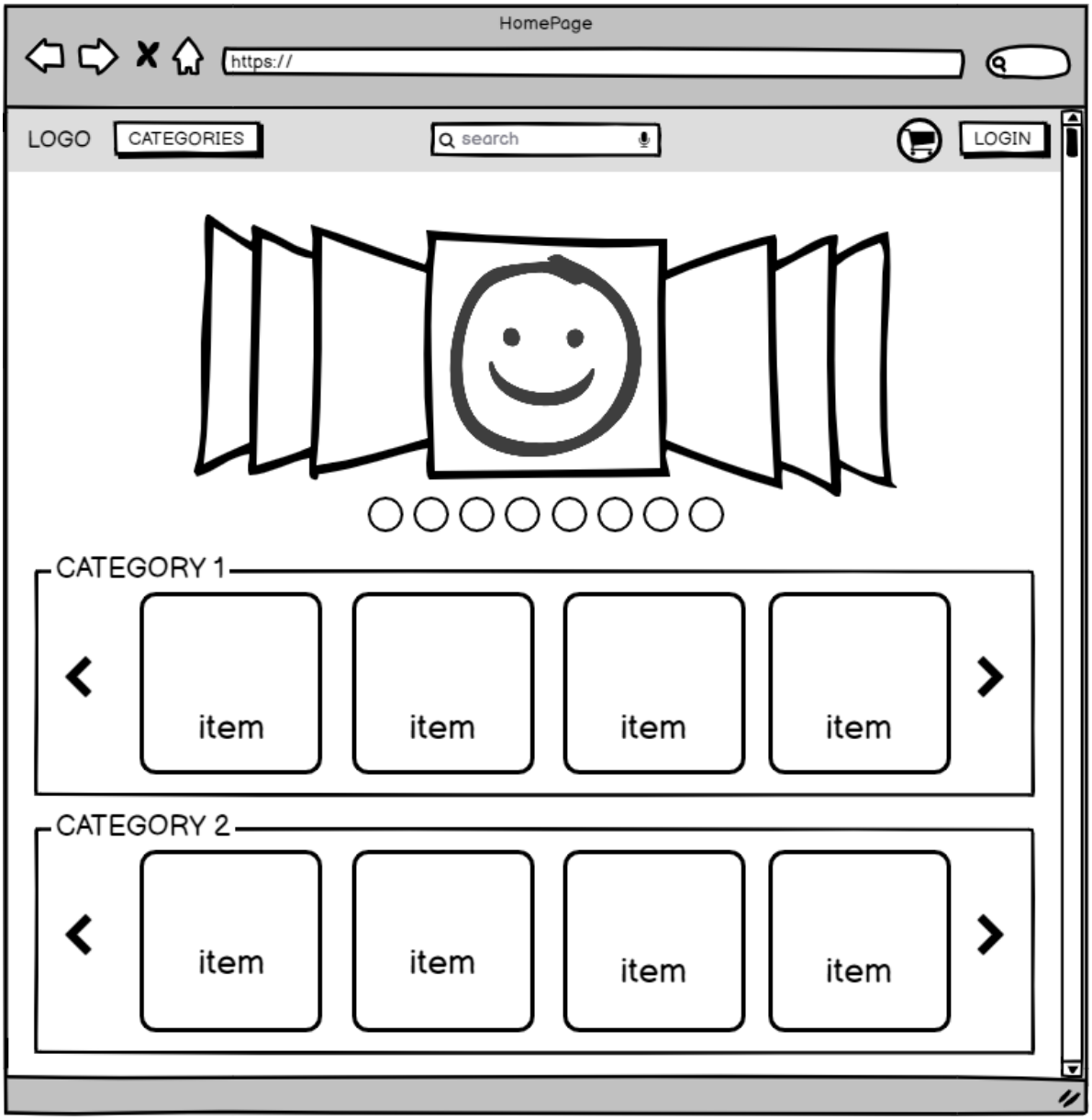

Figure 5: Homepage: main page of the web application

The homepage (as shown in figure 5) contains the main information regarding the various products available. The products are divided into subsets according to their category. This subdivision will be done by using a scrollable list. Moreover, the homepage contains a search bar that allows a user to search a product by its name. The homepage contains a status bar that allows the user to register, log in or log out once logged in. Finally, this status bar allows customers to show the history of the products purchased.

4.2 Product page

Figure 6: Product Page: page regarding the details of a product

The product page (figure 6) shows in detail the information regarding a specific product. This page shows all the pictures available for the product, the description and its price. In addition, it also shows any discount applied to the product at the moment. If the customer is logged in, it will be possible for him to purchase the product within the maximum quantity available. It is possible to enter these particular pages by clicking on a product from the homepage, from the specific category page or by searching it by name using the search bar.

4.3 OrderList and OrderDetails

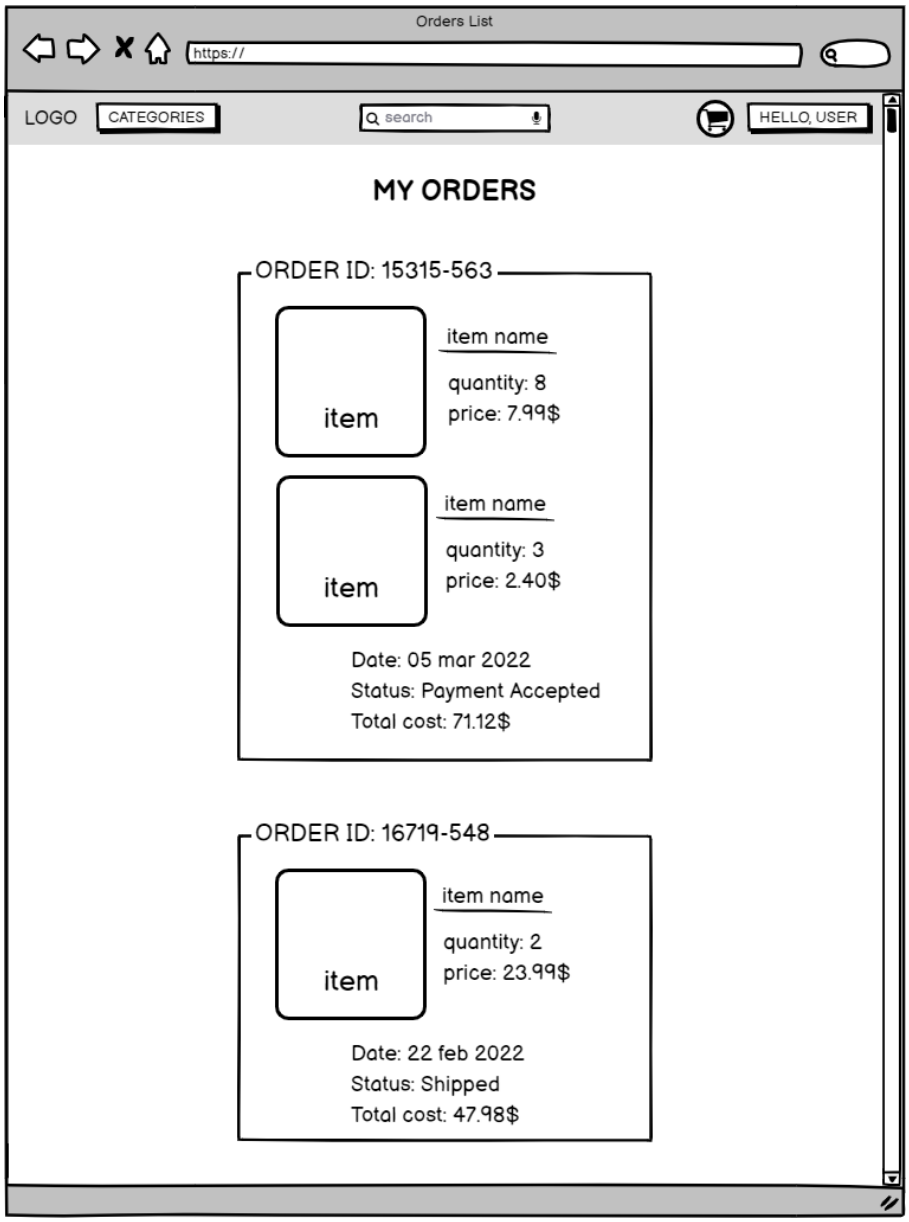

Figure 7: Order list page: page regarding all the orders of the logged customer

This page (fig. 7) shows in detail all the orders placed by the customer which is currently logged in. It shows the history of all orders placed since the user's registration till now. For each order, it shows the date, the total price and the status. Furthermore, for each order, there is a list of all the products purchased, their quantity and their unit price at the time of purchase.

4.4 Ticket list management

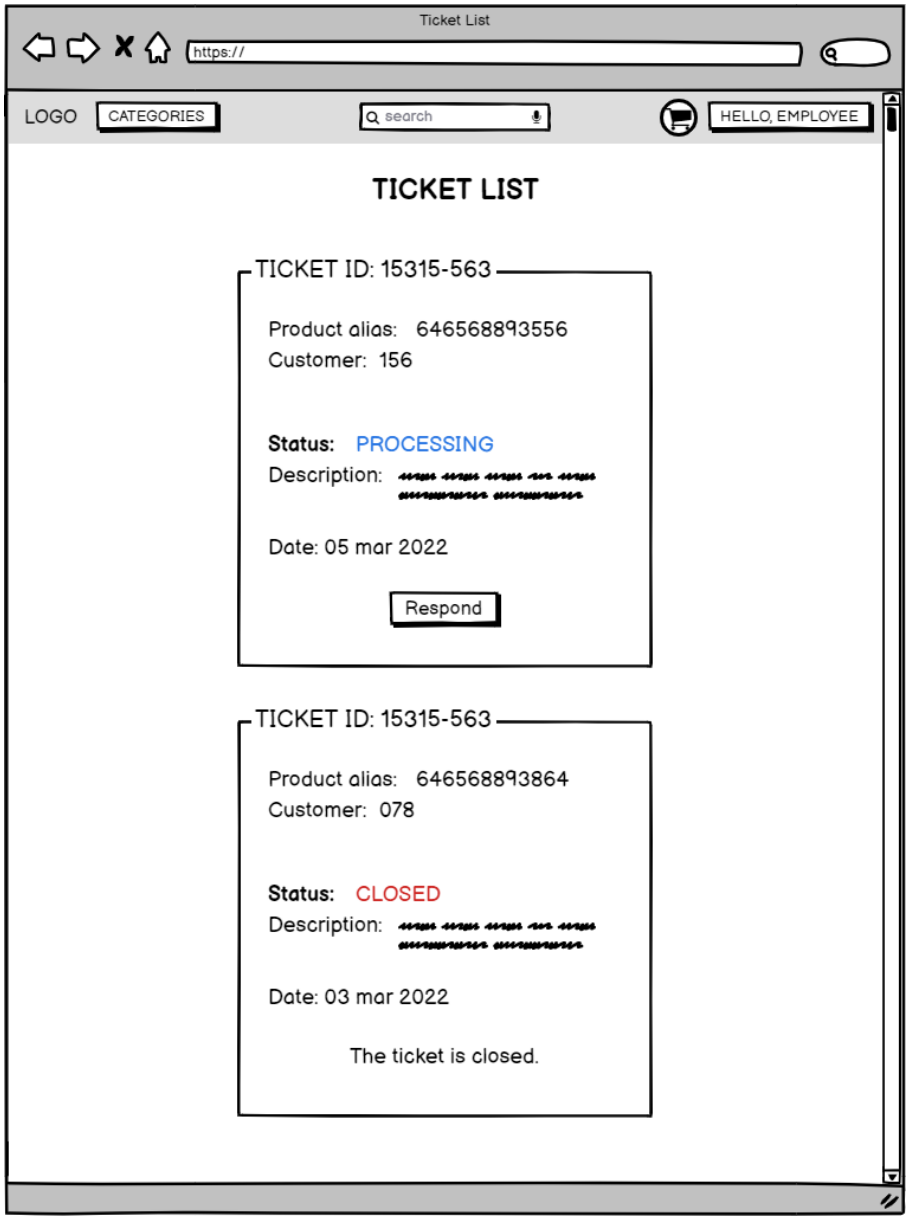

Figure 8: Ticket list management page: page regarding all the tickets opened

The ticket list page (fig. 8) can be accessed by both customers and employees. In the customer version, the page shows the list of the ticket opened by the current user. Each ticket displays its ID, its status and the product involved. In the employee version, each ticket shows also the ID of the customer. Furthermore, this page allows the employee to respond to the ticket and update its status.

4.5 UserEdit page

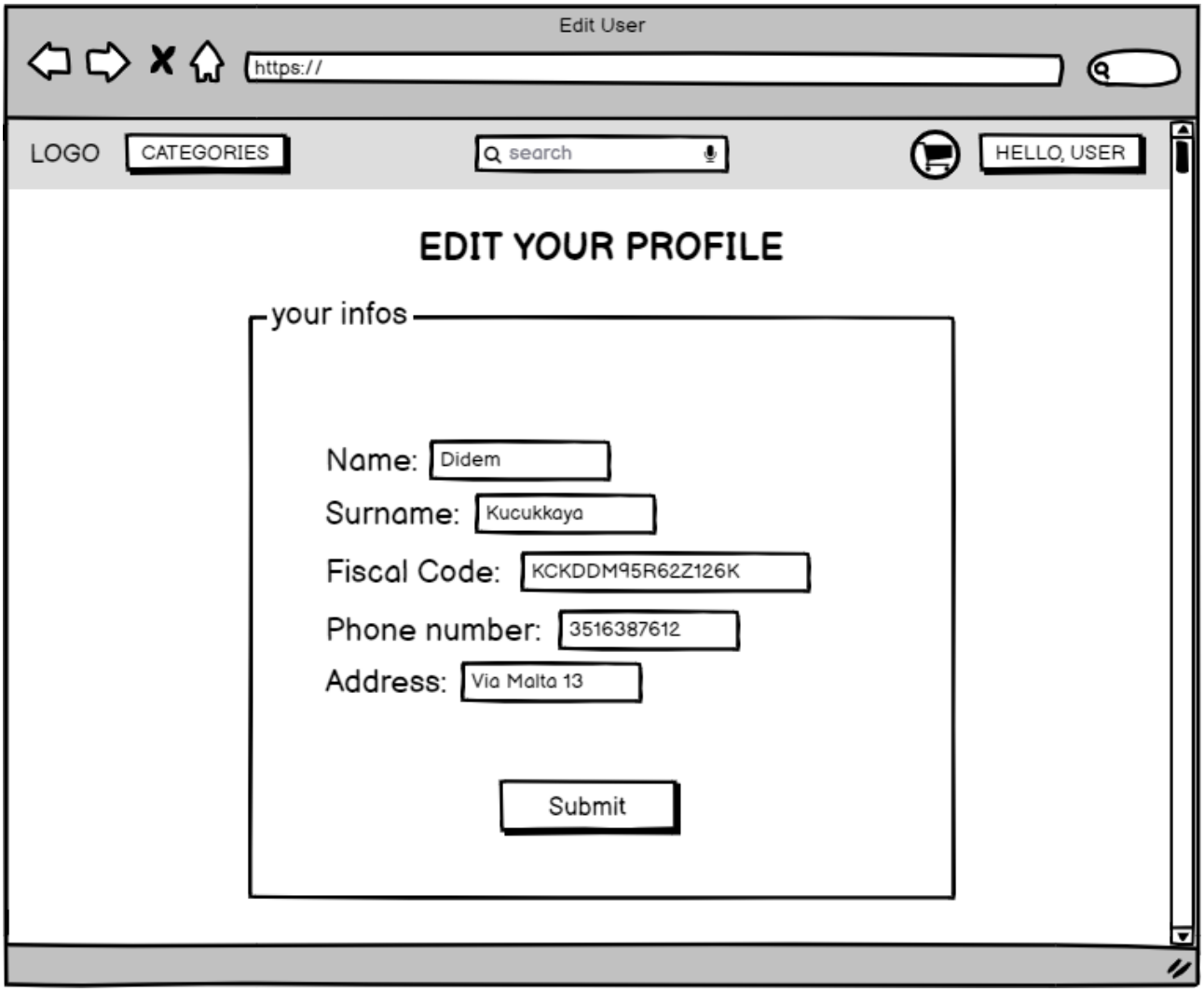

Figure 9: User edit page: page that allows the current user to edit their informations

The user edit page (as shown in fig. 9) allows the customer type user to change its personal data. The page consist of text boxes with default values corresponding to the values previously entered by the user. The page allows you to change data such as telephone number, name, surname but not unique / primary key, because this particular data is used for the login. Obviously, this page will be available only if the user is logged in.

4.6 Admin: product management

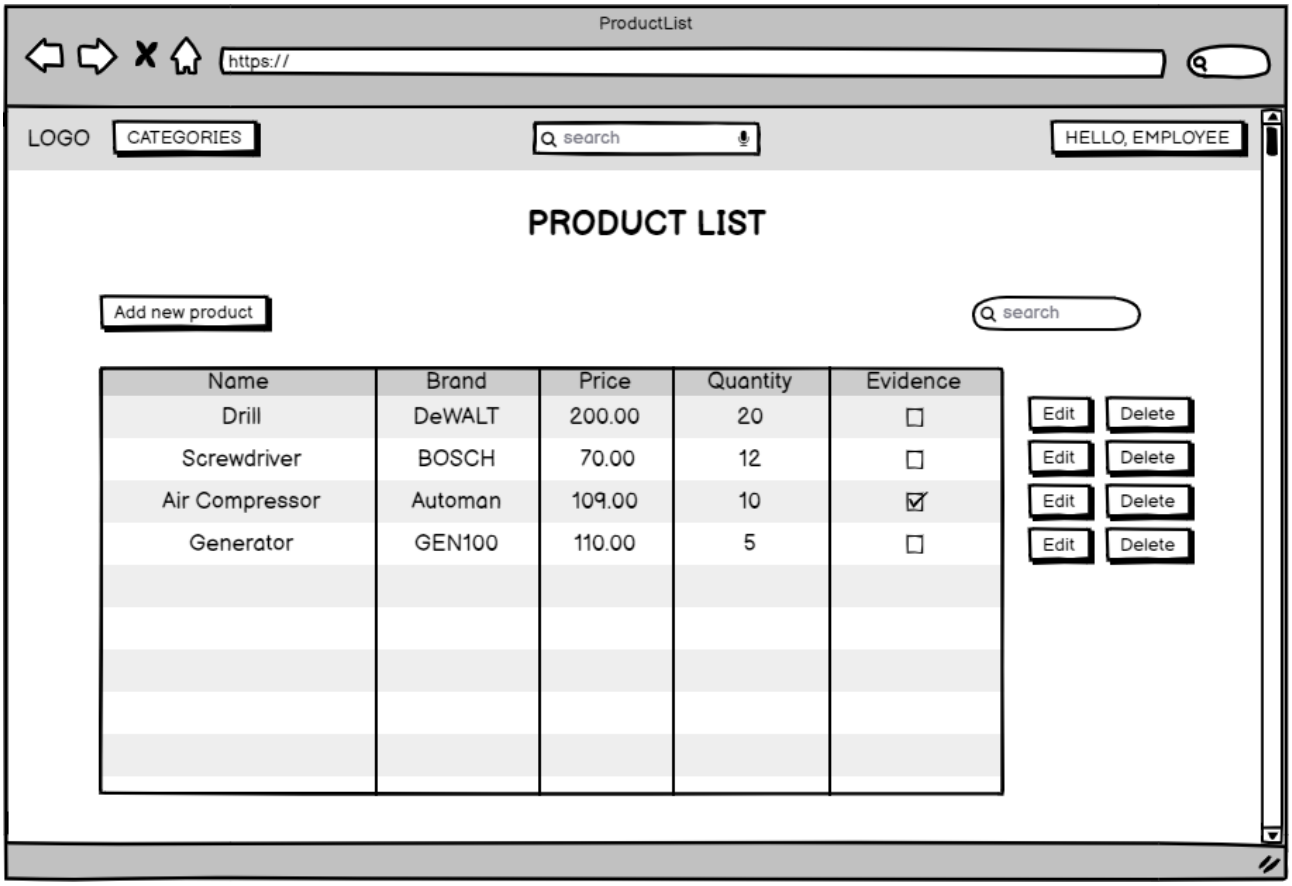

Figure 10: Product management page: page that shows all the products of the shop

Only the employees have access to the product management pages (fig.10) as well as administrators. The page, which refers to the management of products, allows the employees to view a table, which lists all the products of the e-shop, with their information, such as name, the brand, the selling price and the quantity. In addition, the page gives the possibility to edit the products. However, let's reiterate that products can't be deleted in any case: the only way to "logically delete" a product is by setting its quantity to zero. In this manner, products with a quantity equal to zero are unseen, therefore unavailable for the purchase. Finally there's an "add a new product" button for the creation of a new product to the list (see fig.11). Since every product belongs to a specific category, it is also possible to create a new category when adding a new product, or just use an existing one.

Thanks to this page, the owner of the e-shop will be able to upgrade his selling activity and offer more products in his catalogue.

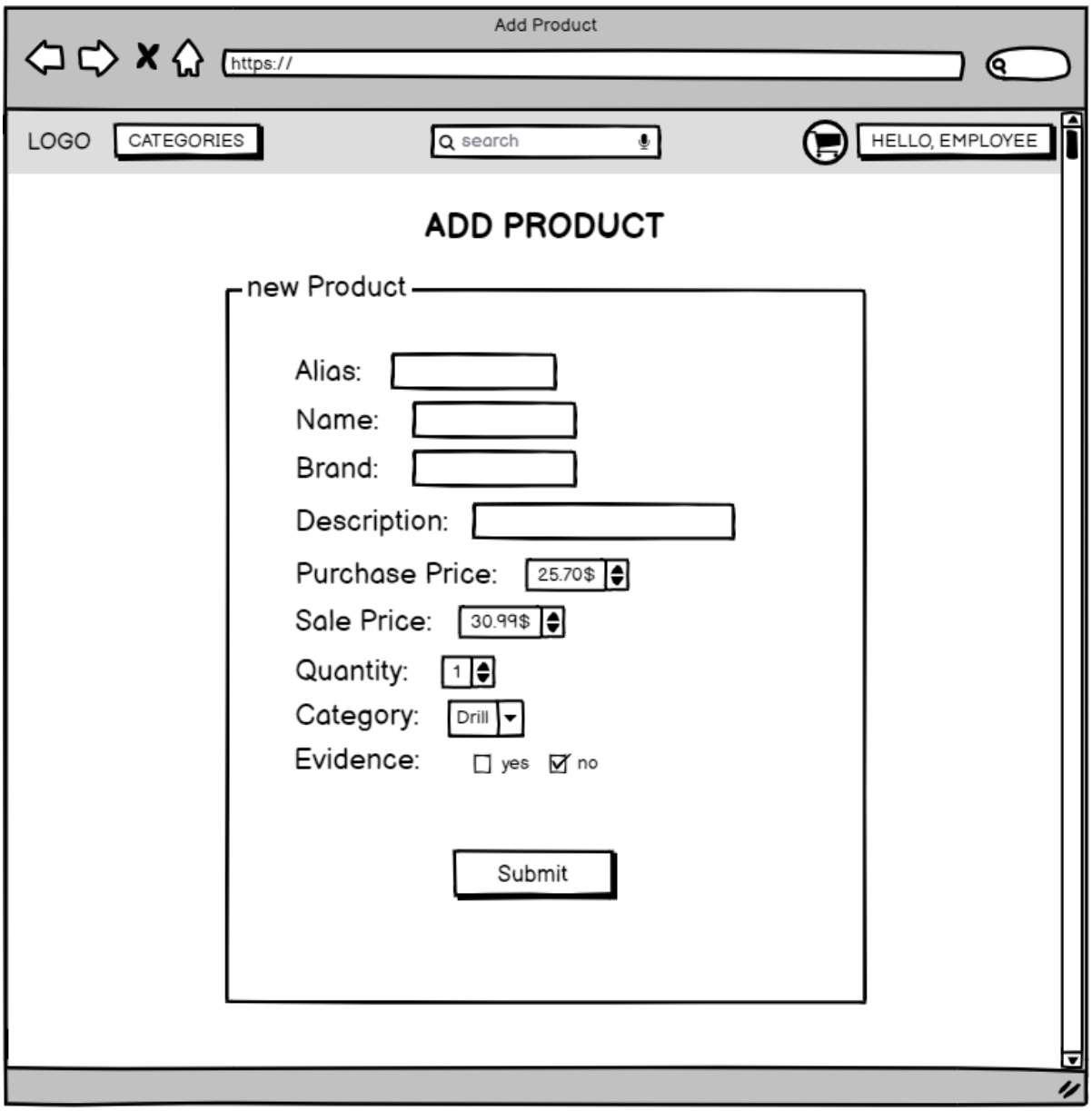

Figure 11: Add product page: page that allows to add a new product

4.7 Admin: user management

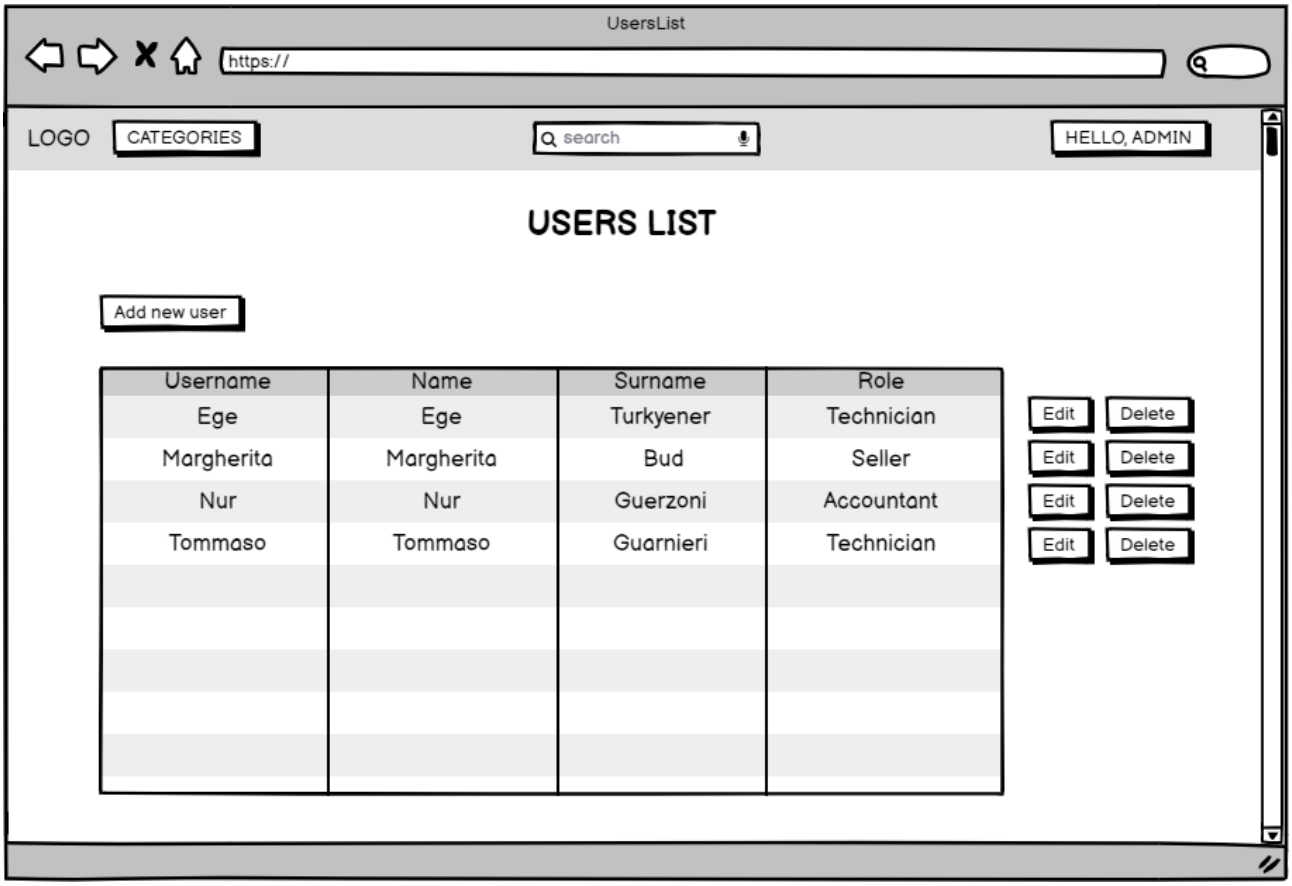

Figure 12: User management page: page that shows all users

The user management page (fig. 12) can be reached only by administrators. The page displays the username, name, surname and role of any employee registered in the website. It is also possible to edit, add, or delete a user. There exist two types of pages about "user management": one only for the employees and another for all the customers registered to the website.

5 Business Logic Layer

5.1 Class Diagram

The architecture of our webapp is based on four principal items:

- **Resource**: also called POJO (Plain Old Java Object) classes, are the classes used for the Java object representation of each tables in the database. They are implemented using a tree hierarchy.
- **DAO** (Data Access Object): are the classes that communicate with the database, taking care of the queries and saving their results in the resource classes.
- **Servlet**: are the classes that manage the interaction with the rest APIs and hypertext.
- **Filter**: are the classes that protect the servlet access allowing only those who really are entitled to it to use their functionalities.

Please notice that the following diagrams are in vectorial format (not raster) so that one is able to zoom in to see and read the details

Resource

In the figure 13 it is possible to observe the architecture of the resources, all implementing the Resource interface and containing the `toJSON()` method. The main reason for which it has been utilized an interface and not an abstract class is because of the greater possibilities that an interface offers in Java. An example could be that an enumeration can implement an interface and that it cannot extend an abstract class: this allows us to manage the enumerations as resources.

It has been used the `org.json` package (also used by the tutor), because the use is more immediate and requires less code overhead than the `fasterxml.jackson` package.

DAO

Since DAOs have no inheritance (there is no basic DAO), we do not include the DAO schema. Regardless, they are already divided into subpackages for the table they operate on, although they are an important part of the project.

Servlet

In the figure 14 it is possible to observe the architecture of the servlets. We defined this structure in order to maintain a functional separation between the common web pages and the ones dedicated to the login or the database. In particular:

- `AbstractServlet` extends `HttpServlet` and it contains the functions to print in output the web pages using JSON and JSP. `AbstractServlet` contains an abstract method that references the database connection as it was needed to properly implement the header without breaking the Model View Controller pattern;
- `AbstractServlet` also contains functions such as `writeResource`, `writeJsp`, `writeError`, `writeMessageOrRedirect`, `writeJSON`, `writeBlob`, `readJSON` and `readInputParameters` that allow to effectively and centrally manage all the output and marginally also the input. In detail the functions mentioned above are responsible for:
  - `writeResource` sends the request to a JSON or hypertext (HTML) output based on whether or not a REST request is made. Automatically provides the JSP with the session information in the variable `user`. The first two parameters passed to the function are the HTTP request and response, then we pass the JSP url followed by a boolean value to indicate if we want to pass a list (false) or a single object (true) as last parameter. The last parameter you pass is in fact a class that implements the Resource class via varargs and that supports arrays of single or multiple elements.

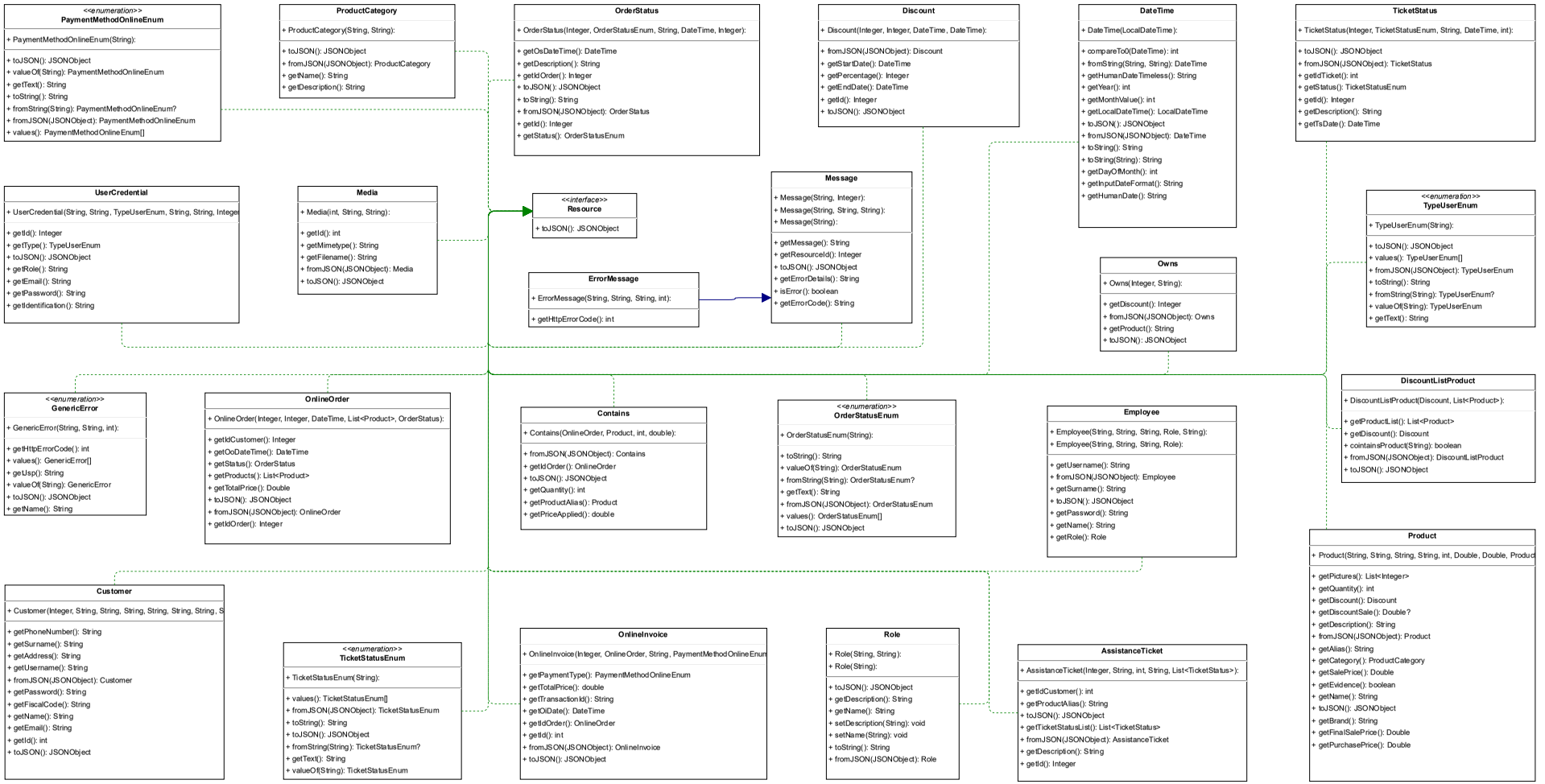

Figure 13: Resource class diagram

  - `writeJsp` is used to print a JSP page without any resource. It does not print a JSON in case of REST request as there is nothing to be printed.
  - `writeError` is used to print an error within an JSON or hypertext (HTML) output based on whether or not a REST request is made.
  - `writeMessageOrRedirect` is used when it's necessary to realize some redirects and since this is not possible using the rest API, it has been created this kind of function that is able to make the redirect only if it is hypertext. In the case of a JSON output it just prints a message with the useful data to reach the new page.
  - `writeJSON` is the function that prints the formatted JSON.
  - `writeBlob` is the function that prints a blob file with header and everything that the browser needs to read it.
  - `readJSON` is the function devoted to reading the informations from the header POST in JSON format.
  - `readInputParameters` is the function devoted to reading the information from the header POST returning an array of key-value pairs.

- `AbstractLoggerServlet` extends the `AbstractServlet`. It initialises Log4j so that each class has its own logger already initialized.
- `AbstractDatabaseServlet` extends `AbstractLoggerServlet`. It initialises the database. All servlets extend `AbstractDatabaseServlet`.

Filter

The filter package contains an abstract class that takes care of initializing the common filter information. The 4 filter classes that extend the abstract one, take care of filtering the data in the session.

In the figure 15 you can see the architecture of the filters. In the next pages we will use the first letter of the filter class name to indicate the filter. In particular **L** for `LoginFilter`, **A** for `AdminFilter`, **E** for `EmployeeFilter` and **C** for `CustomerFilter`.

A brief description of what the filters do:

- `LoginFilter` checks if the user is logged in
- `CustomerFilter` checks if a customer is logged in
- `EmployeeFilter` checks if an employee is logged in
- `AdminFilter` checks if an administrator is logged in

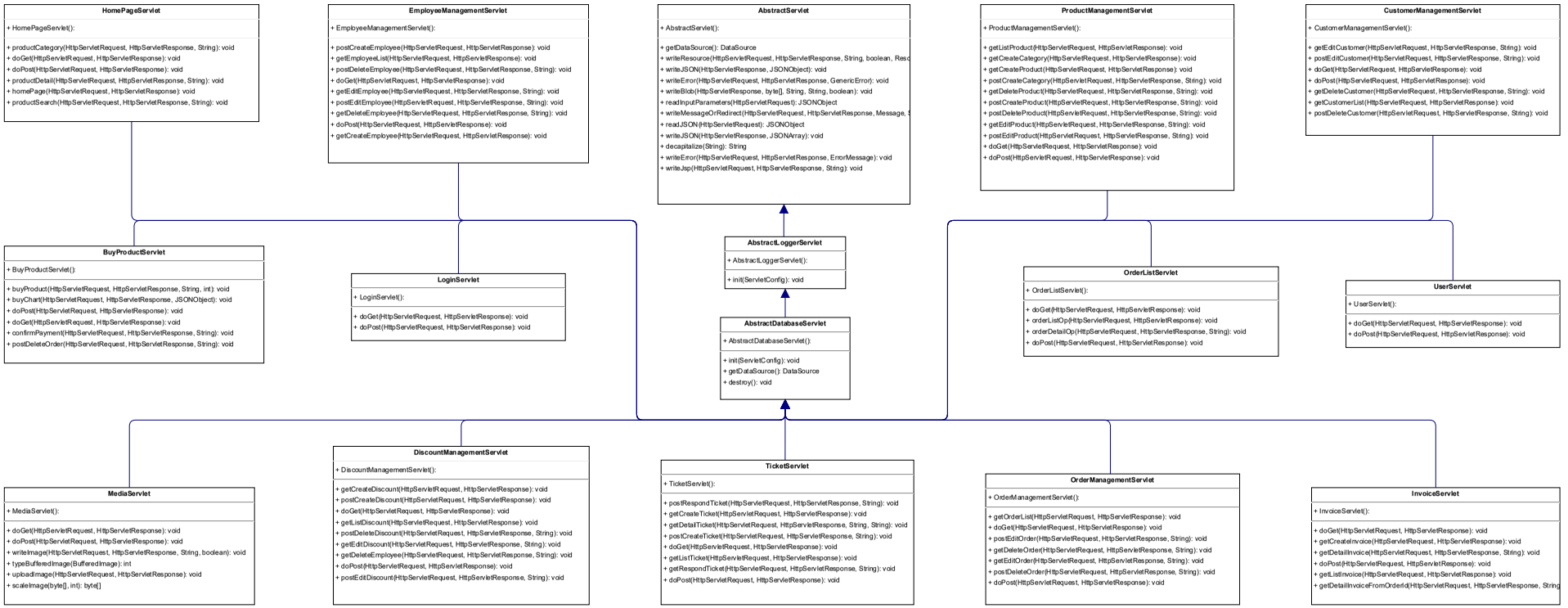

Figure 14: Servlet class diagram

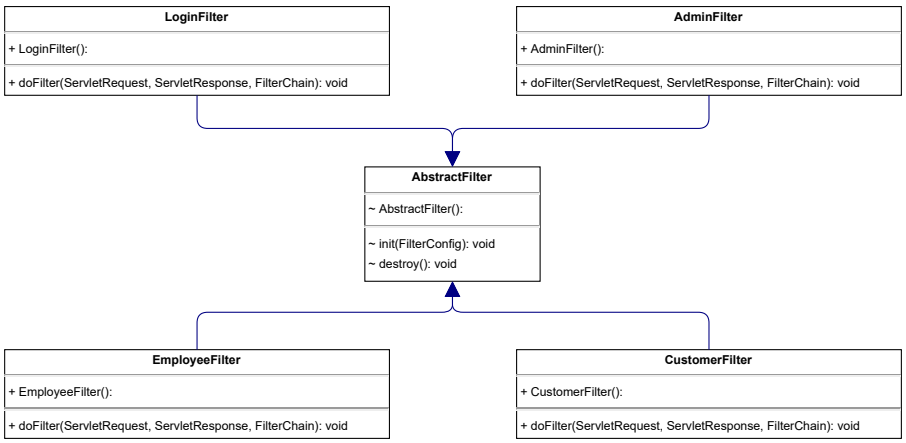

Figure 15: Filter class diagram

5.2 Sequence Diagram

In the next figures are illustrated some sequence diagrams of the methods `GET` (figures 16, 17, 18, 18, 21, 23) and `POST` (figures 20, 22, 24) of the main servlets.

In all the sequence diagrams it is common the presence of the operation `writeResource` and `writeJsp` that take care of handling uniformly the output.

Please notice that the following diagrams are in vectorial format (not raster) so that one is able to zoom in to see and read the details

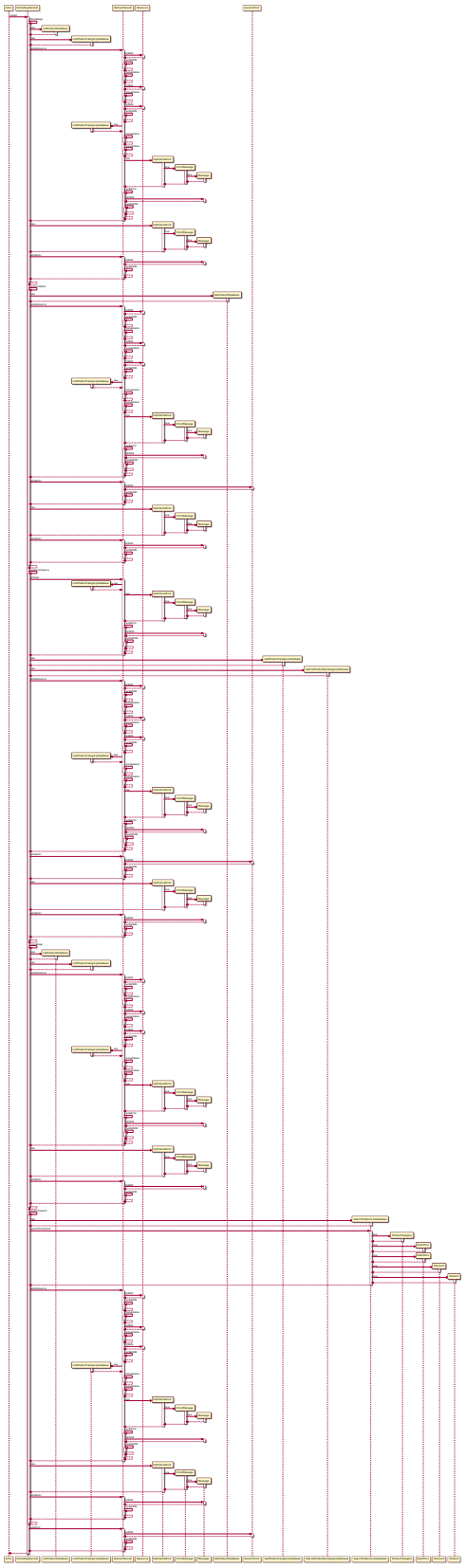

Figure 16: Sequence diagram of the `GET` method of HomePageServlet

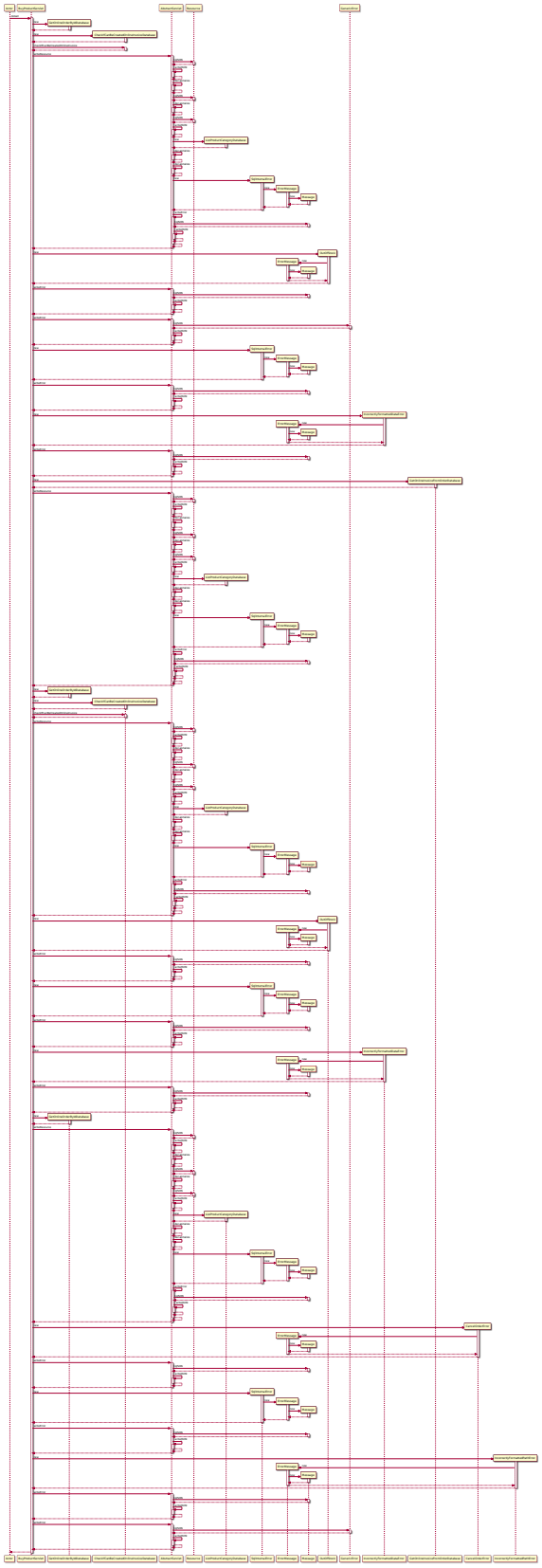

Figure 17: Sequence diagram of the `GET` method of BuyProductServlet

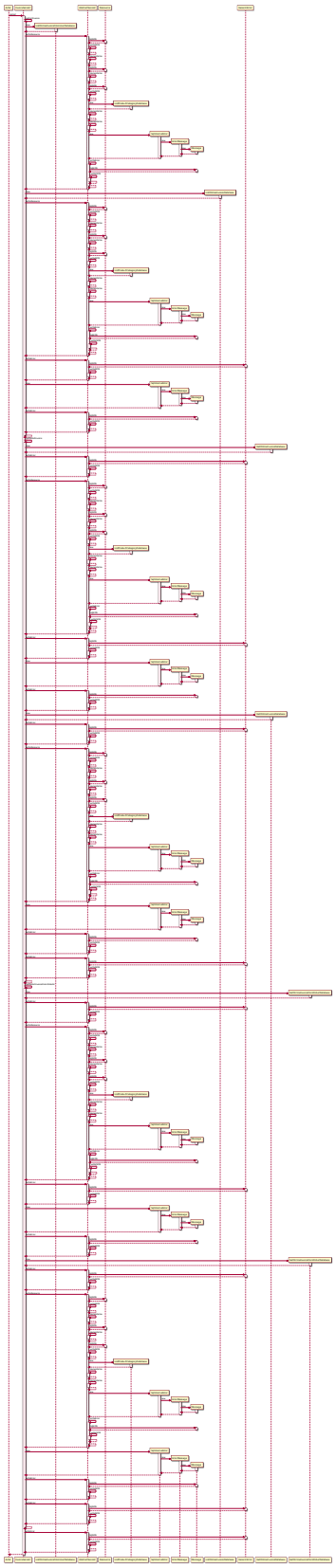

Figure 18: Sequence diagram of the `GET` method of InvoiceServlet

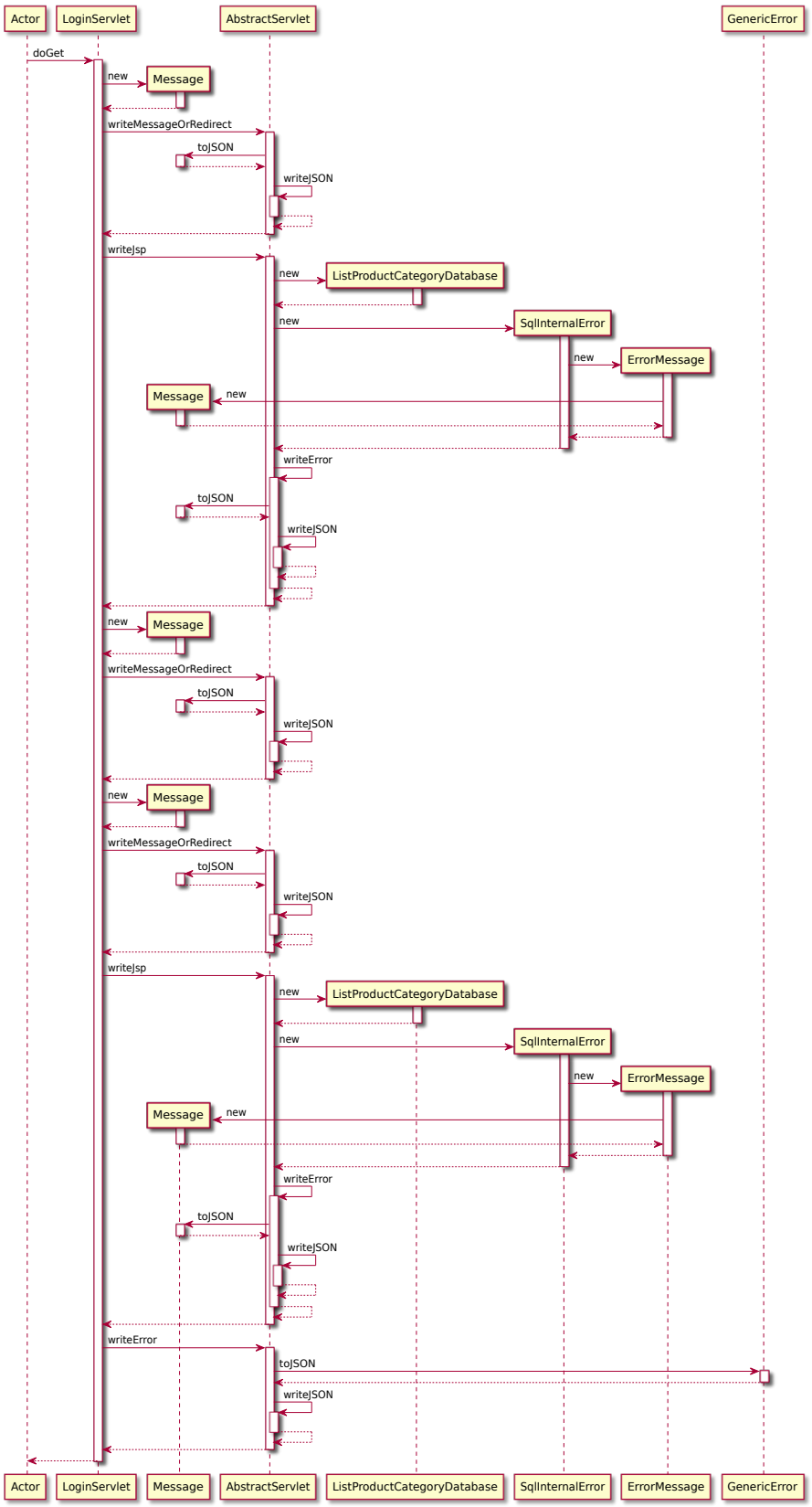

Figure 19: Sequence diagram of the `GET` method of LoginServlet

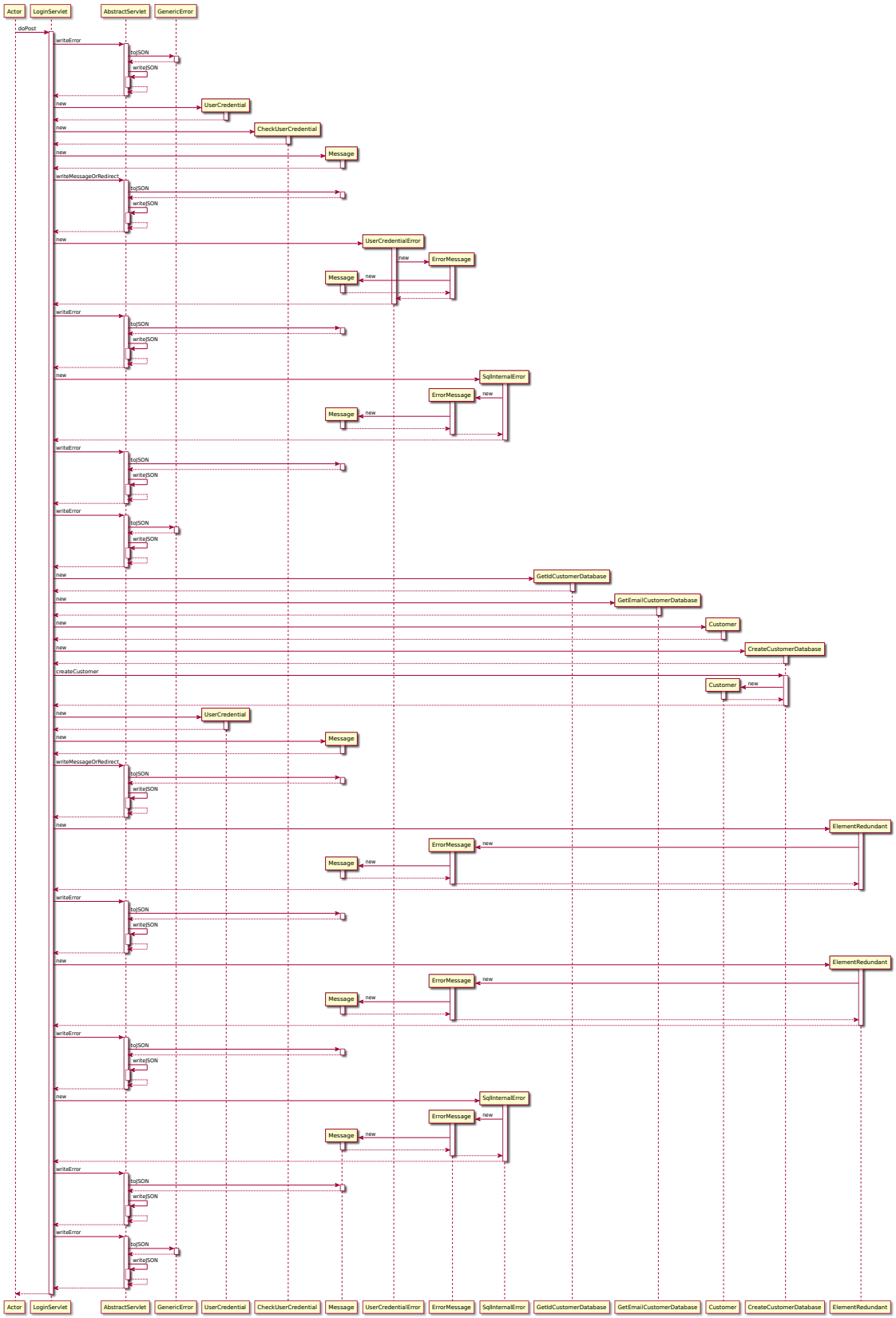

Figure 20: Sequence diagram of the `POST` method of LoginServlet

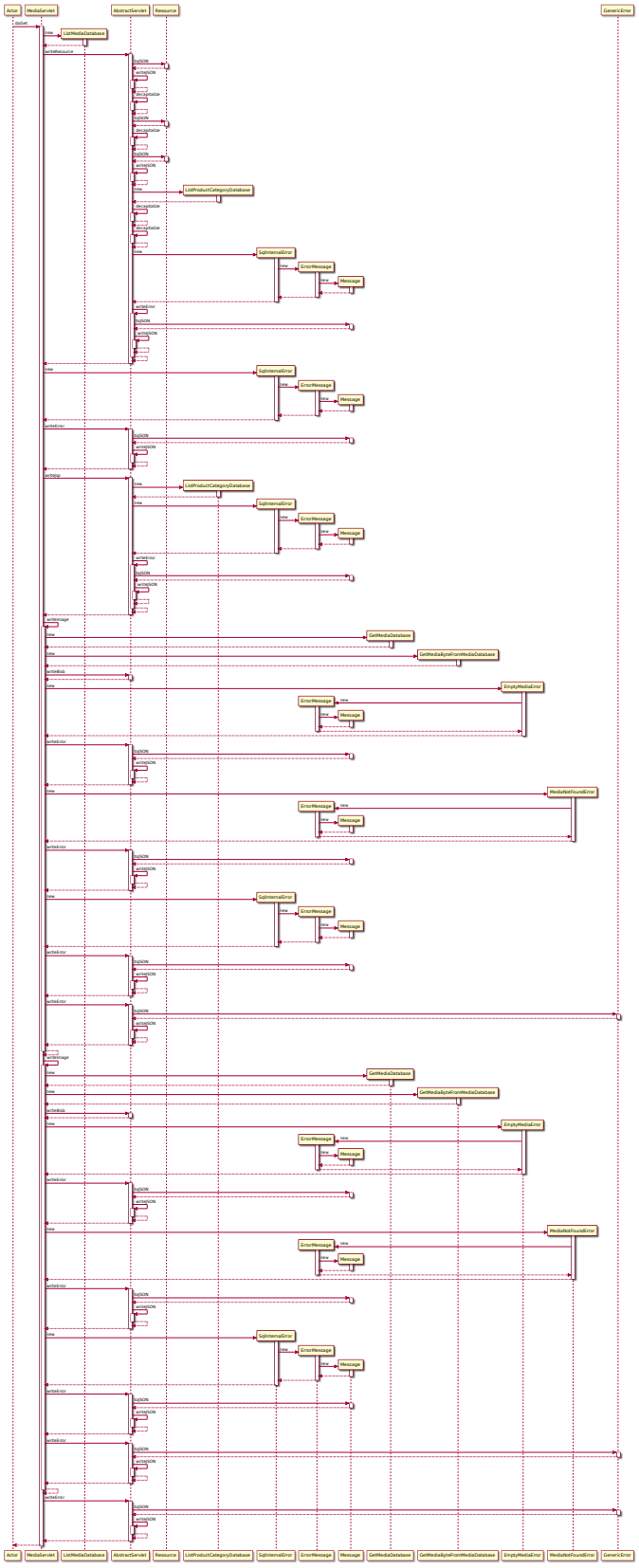

Figure 21: Sequence diagram of the `GET` method of MediaServlet

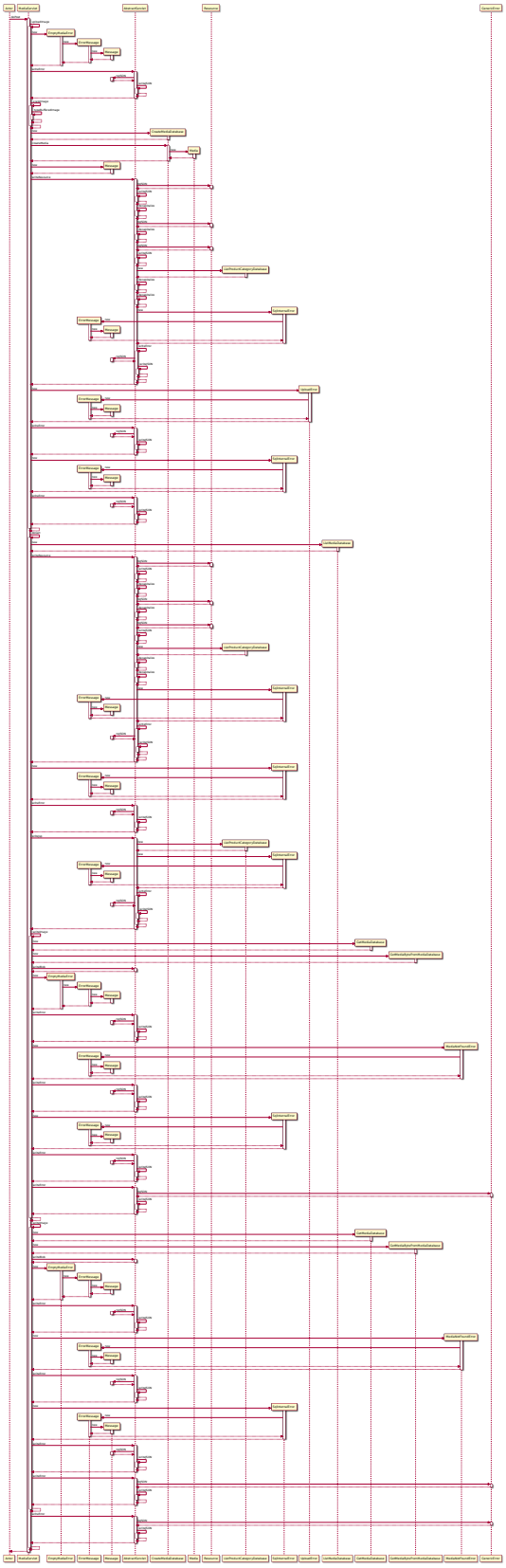

Figure 22: Sequence diagram of the `POST` method of MediaServlet

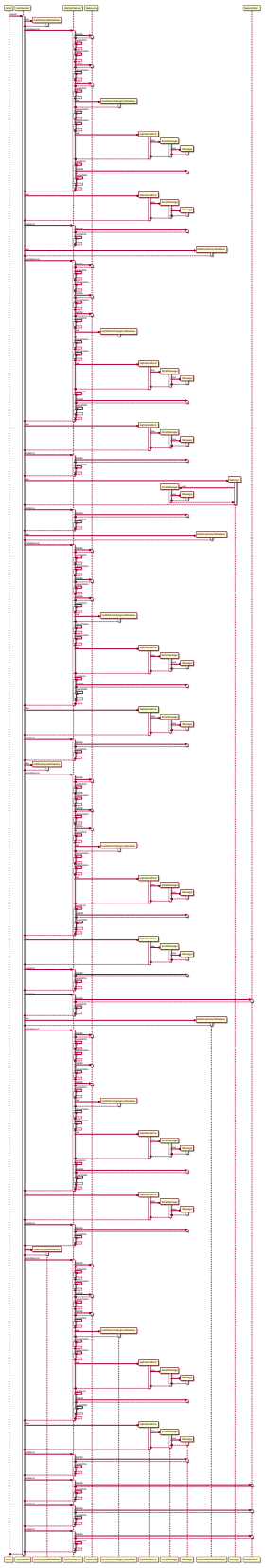

Figure 23: Sequence diagram of the `GET` method of UserServlet

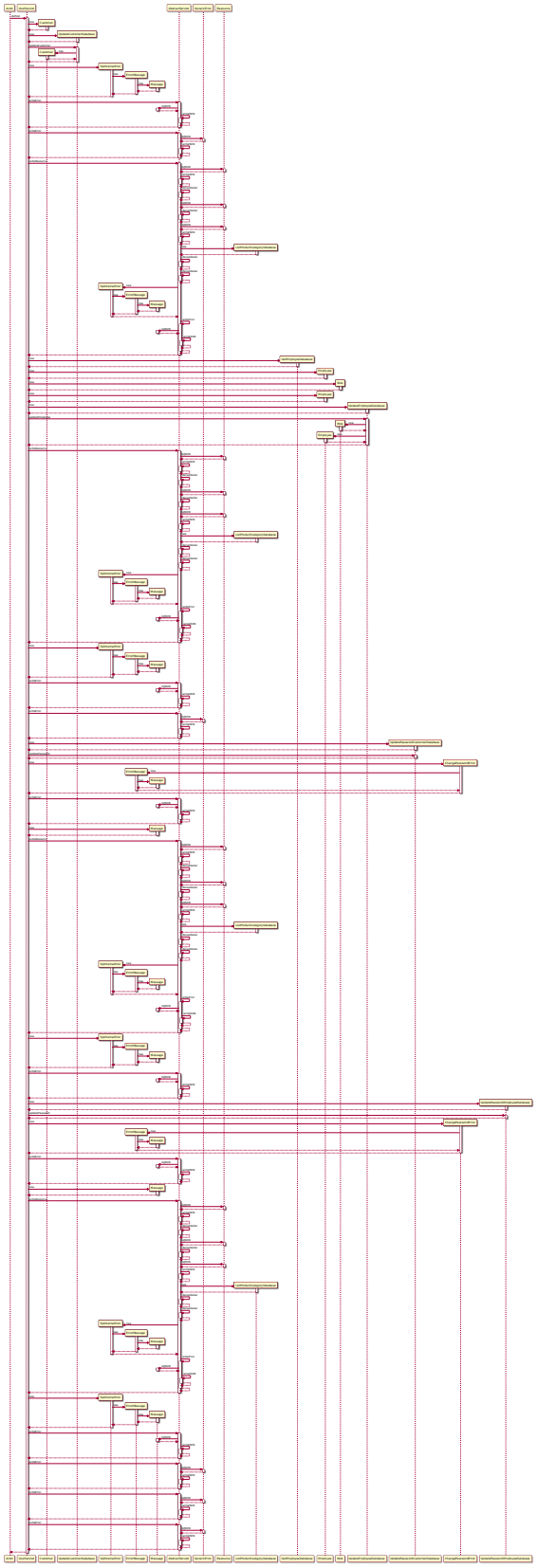

Figure 24: Sequence diagram of the `POST` method of UserServlet

5.3 REST API Summary

In table 2 you can see the list of the APIs present in the web application. Notice that the prefix `/rest` can be omitted if you pass the header `Accept` with value `application/json`.

In particular **L** for `LoginFilter`, **A** for `AdminFilter`, **E** for `EmployeeFilter` and **C** for `CustomerFilter`.

URI	Method	Description	Filter
`/rest/`	GET & POST	return a list of categories and a list of products	
`/rest/products/details/`*product alias*	GET & POST	return product details	
`/rest/products/category/`*category name*	GET & POST	returns a list of products belonging to a category	
`/rest/products/category/`	GET & POST	returns a list of all category	
`/rest/products/search/`	GET & POST	returns a list of products matching your search text	
`/rest/buy/product/`*product alias* ?	GET & POST	create an order with single product	C
`/rest/buy/cart/` ?	POST	creates an order with a JSON of product	C
`/rest/buy/pay/`*order ID* ?	POST	pays an order	C
`/rest/buy/cancel/`*order ID* ?	POST	deletes an order	C
`/rest/session/login/`	POST	login via a session	
`/rest/session/logout/`	GET	logout via a session	
`/rest/session/register/`	POST	register customer amd login via session	
`/rest/media/list/`	GET	returns a list containing ID, mimetype and name of uploaded media	E
`/rest/media/upload/`	POST	api for the upload of an image	E
`/rest/media/view/`*image ID*	GET & POST	view media	
`/rest/media/thumb/`*image ID*	GET & POST	view thumb of media	
`/rest/order/list/`	GET & POST	view order list (if customer can only see their own order, if employee all of them)	L
`/rest/order/detail/`*order ID*	GET & POST	view order detail (if customer can only see their own order, if employee all of them)	L
`/rest/invoice/list/`	GET & POST	view invoice list (if customer can only see their own invoices, if employee all of them)	L
`/rest/invoice/detail/`*invoice ID*	GET & POST	view invoice detail from invoice ID (if customer can only see their own invoices, if employee all of them)	L
`/rest/invoice/order/`*order ID*	GET & POST	view invoice detail from order ID (if customer can only see their own invoices, if employee all of them)	L
`/rest/ticket/create/`	POST	create a ticket	C
`/rest/ticket/list/`	GET & POST	list of ticket (if customer can only see their own invoices, if employee all of them)	L

`/rest/ticket/detail/`*ticket ID*	GET & POST	detail of a ticket from ticket ID (if customer can only see their own invoices, if employee all of them)	L
`/rest/ticket/respond/`*ticket ID*	GET & POST	respond a ticket: create ticket status	E
`/rest/user/info`	GET	view personal data	L
`/rest/user/modify`	POST	update personal data	L
`/rest/user/password`	POST	edit password	L
`/rest/management/orderManagement`	GET	show order list	E
`/rest/management/orderManagement/editOrder/`	POST	Edit a order	E
`/rest/management/orderManagement/deleteOrder/`	POST	Delete a order	E
`/rest/management/productManagement`	GET	show products list	E
`/rest/management/productManagement/createProduct`	POST	Create a new product	E
`/rest/management/productManagement/editProduct/`	POST	Edit a product	E
`/rest/management/productManagement/deleteProduct/`	POST	Delete a product	E
`/rest/management/discountManagement`	GET	show discounts list	E
`/rest/management/discountManagement/createDiscount`	POST	Create a new discount	E
`/rest/management/discountManagement/updateDiscount/`	POST	Edit a discount	E
`/rest/management/discountManagement/deleteDiscount/`	POST	Delete a discount	E
`/rest/management/customerManagement`	GET	show customers list	E
`/rest/management/customerManagement/editCustomer/`	POST	Edit a customer	E
`/rest/management/customerManagement/deleteCustomer/`	POST	Delete a customer	E
`/rest/management/employeeManagement`	GET	show discounts list	A
`/rest/management/employeeManagement/createEmployee/`	POST	Create a new employee	A
`/rest/management/employeeManagement/editEmployee/`	POST	Edit an employee	A
`/rest/management/employeeManagement/deleteEmployee/`	POST	Delete an employee	A

Table 2: API for the REST interface of the Electromechanics Shop application back-end

5.4 REST Error Codes

Here is reported the list of errors defined in the application. Application specific errors follows a progressive numeration starting from 100. Employees specific management errors are identified starting from 200. Internal errors, which correspond to crashes, servlet exceptions, or problems with the input/output streams are identified with the Error Code 999 and 998 (specific for SQL errors).

Error Code	HTTP Status Code	Description
100	UNAUTHORIZED	Login credentials are not correct
101	BAD_REQUEST	Unable to change password
102	UNAUTHORIZED	Login required to see this page
103	CONFLICT	User already logged in
104	BAD_REQUEST	Path incorrectly formatted
105	BAD_REQUEST	Data incorrectly formatted
106	NOT_ACCEPTABLE	Order error
107	NOT_ACCEPTABLE	Product are out of stock
108	NOT_ACCEPTABLE	Unable to delete order
109	CONFLICT	Data element already present
200	NO_CONTENT	Media cannot be empty
201	NOT_FOUND	Media not found
202	NOT_FOUND	Product not found
203	NOT_FOUND	Order not found
204	NOT_ACCEPTABLE	Upload media error
205	NOT_ACCEPTABLE	Product list cannot be empty
206	NOT_ACCEPTABLE	Start date cannot be later than end date
207	CONFLICT	Employee username already present
208	CONFLICT	Product alias already present
209	CONFLICT	Category name already present
210	CONFLICT	Unable to create discount
211	NOT_ACCEPTABLE	Unable to delete employee
212	NOT_ACCEPTABLE	Unable to delete customer
213	NOT_ACCEPTABLE	Unable to delete product
214	NOT_ACCEPTABLE	Unable to delete discount
998	INTERNAL_SERVER_ERROR	SQL Internal error
999	INTERNAL_SERVER_ERROR	Internal error

Table 3: Error codes for the REST interface of the Electromechanics Shop application back-end

5.5 REST API Details

Buy a product

This API returns a new order with a single product

- URL: `/buy/product/`
- Method: `POST`
- URL Parameters: *product alias*
- Data Parameters: *quantity*: the quantity of the product
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a JSON Object with the order ID.
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Buy a list product

This API returns a list of products and a list of categories

- URL: `/rest/cart/`
- Method: `POST`
- URL Parameters: Nothing
- Data Parameters: a JSON Object that contains a JSON Array on the `cart` key. The JSON Array contains some instances of a JSON Object containing the alias of the product and the quantity. Like: `"cart":["quantity":2,"alias":"6465661284410","quantity":3,"alias":"6465661284414"]`
- Data Parameters encoding: `application/json`
- Success Response: a JSON Object with the order ID.
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Pay an order

This API returns a list of products and a list of categories

- URL: `/rest/pay/`
- Method: `POST`
- URL Parameters: *order ID*
- Data Parameters: `payment` with a value among `CREDIT_CARD`, `GOOGLE_PAY`, `APPLE_PAY`.
- Success Response: a JSON message whit transaction ID
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Cancel an order

This API cancels an order

- URL: `/rest/cancel/`
- Method: `POST`
- URL Parameters: *order ID*
- Data Parameters: NOTHING
- Success Response: a JSON message
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

List of products and categories

This API returns a list of products and a list of categories

- URL: `/rest/`
- Method: `GET & POST`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Object with a productCategory list and a product list.
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Product detail

This API returns the product detail

- URL: `/rest/products/details/`
- Method: `GET & POST`
- URL Parameters: *product alias*
- Data Parameters: Nothing
- Success Response: a JSON Object with the product
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

List of categories

This API returns a list of categories

- URL: `/rest/products/category/`
- Method: `GET & POST`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Object with a productCategory list
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Products of a category

This API returns a list of product belonging to a category

- URL: `/rest/products/category/`
- Method: `GET & POST`
- URL Parameters: *category name*
- Data Parameters: Nothing
- Success Response: a JSON Array with the product list belonging to the category
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Search product

This API returns a list of products that match the search keyword

- URL: `/rest/products/search/`
- Method: `GET & POST`
- URL Parameters: Nothing
- Data Parameters: `q`: a query keyword
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a JSON Array with the product list that match the search keyword
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Media list

This API returns a list of media

- URL: `/rest/media/list/`
- Method: `GET`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Array with the media list
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Media list

This API returns a list of media

- URL: `/rest/media/list/`
- Method: `GET`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Array with the media list
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Upload media

This API is used to load a media in the database

- URL: `/rest/media/upload/`
- Method: `POST`
- URL Parameters: Nothing
- Data Parameters: `file`: the file to be upload
- Data Parameters encoding: `multipart/form-data`
- Success Response: a JSON Object with the media ID
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Media view

This API returns a media

- URL: `/rest/media/view/`
- Method: `GET & POST`
- URL Parameters: *media ID*
- Data Parameters: Nothing
- Success Response: a blob with media
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Media thumb view

This API returns a thumb of a media

- URL: `/rest/media/thumb/`
- Method: `GET & POST`
- URL Parameters: *media ID*
- Data Parameters: Nothing
- Success Response: a blob with thumb of the media
- Error Response: a JSON Objec: with a key: `isError` and value `true`, an error code, a message and details of the error.

Order list

This API returns a list of the orders. If the logged-in user is a customer, it is only possible to see their own orders, if employee all of them.

- URL: `/rest/order/list/`
- Method: `GET`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Array that contains the order list
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Order detail

This API returns the details of an order. If the logged-in user is a customer, it is only possible to see their own orders, if employee all of them.

- URL: `/rest/order/detail/`
- Method: `GET`
- URL Parameters: *order ID*
- Data Parameters: Nothing
- Success Response: a JSON Object that contains the order detail
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Invoice list

This API returns a list of the invoices. If the logged-in user is a customer, it is only possible to see their own invoices, if employee all of them.

- URL: `/rest/invoice/list/`
- Method: `GET`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Array that contains the invoice list
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Invoice detail

This API returns the details of an invoice. If the logged-in user is a customer, it is only possible to see their own invoices, if employee all of them.

- URL: if you want to access it with the invoice ID `/rest/invoice/detail/` or if you want to access it with the order ID `/rest/invoice/order/`
- Method: `GET`
- URL Parameters: if you want to access it with the invoice ID *invoice ID* or if you want to access it with the order ID *order ID*
- Data Parameters: Nothing
- Success Response: a JSON Object that contains the invoice detail
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Create ticket

This API creates a ticket.

- URL: `/rest/ticket/create/`
- Method: `POST`
- URL Parameters: *product alias*
- Data Parameters: `description`: the description of ticket
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a JSON Object that contains the newly created ticket
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Ticket list

This API returns a list of the tickets. If the logged-in user is a customer, it is only possible to see their own tickets, if employee all of them.

- URL: `/rest/ticket/list/`
- Method: `GET`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Array that contains the ticket list
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Ticket detail

This API returns the details of a ticket. If the logged-in user is a customer, it is only possible to see their own tickets, if employee all of them.

- URL: `/rest/ticket/detail/`
- Method: `GET`
- URL Parameters: *invoice ID*
- Data Parameters: Nothing
- Success Response: a JSON Object that contains the ticket detail
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Respond ticket

This API lets respond to ticket with a new ticket status.

- URL: `/rest/ticket/respond/`
- Method: `POST`
- URL Parameters: *ticket ID*
- Data Parameters:
  - `status`: a value among `OPEN`, `PROCESSING`, `CLOSED`. `RETURN`
  - `description`: the description of ticket
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a JSON Object that contains the new ticket response
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

User info

This API returns the info of current logged user.

- URL: `/rest/user/info`
- Method: `GET`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Object that contains the user info
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

User modify

This API allows to modify a generic user informations.

- URL: `/rest/user/modify`
- Method: `POST`
- URL Parameters: Nothing
- Data Parameters:
  - `name`: the name
  - `surname`: the surname
  - `fiscalCode`: the fiscal code (only customer)
  - `address`: the address (only customer)
  - `phoneNumber`: the phone number (only customer)
  - `role`: the role (only employee, to be discontinued)
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a JSON Object that contains the new user info
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

User modify password

This API allows to modify the password of the logged user.

- URL: `/rest/user/password`
- Method: `POST`
- URL Parameters: Nothing
- Data Parameters:
  - `oldPassword`: the old password
  - `newPassword`: the new password
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a JSON Object that contains password changed
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Order list

This API returns the list of order.

- URL: `/rest/management/orderManagement`
- Method: `GET`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Array that contains order list
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Edit order status

This API modify the status of a order.

- URL: `/rest/management/orderManagement/editOrder/`
- Method: `POST`
- URL Parameters: *order ID*
- Data Parameters:
  - `status`: with a value among `OPEN`, `PAYMENT_ACCEPTED`, `SHIPPED`, `DELIVERED`, `CANCELLED`.
  - `description`: a description
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a message stating that the status has been changed
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Delete order

This API set the status to `CANCELLATED` for a order.

- URL: `/rest/management/orderManagement/deleteOrder/`
- Method: `POST`
- URL Parameters: *order ID*
- Data Parameters: Nothing
- Success Response: a message stating that the status has been changed
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Product list

This API returns the list of products.

- URL: `/rest/management/productManagement`
- Method: `GET`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Array that contains product list
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Create product

This API returns the category and the list of media used to create a new product.

- URL: `/rest/management/productManagement/createProduct`
- Method: `POST`
- URL Parameters: Nothing
- Data Parameters:
  - `alias`: set the alias
  - `name`: set the name
  - `brand`: set the brand of the product
  - `description`: set the description of the product
  - `purchase`: set the purchase sale
  - `sale`: set the sale price
  - `quantity`: set the initial quantity
  - `category`: set the category of the product
  - `evidence`: set if the product should be in evidence
  - `media`: list of media that must be associated with the product
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a message stating that the product has been created
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Edit product

This API returns an Object Product with the new attributes and its list of all media associated to it.

- URL: `/rest/management/productManagement/editProduct`
- Method: `POST`
- URL Parameters: `productAlias`
- Data Parameters:
  - `alias`: set the alias
  - `name`: set the name
  - `brand`: set the brand of the product
  - `description`: set the description of the product
  - `purchase`: set the purchase sale
  - `sale`: set the sale price
  - `quantity`: set the initial quantity
  - `category`: set the category of the product
  - `evidence`: set if the product should be in evidence
  - `media`: list of media that must be associated with the product
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a message stating that the product has been modified
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Delete product

This API returns an Object Product.

- URL: `/rest/management/productManagement/deleteProduct`
- Method: `POST`
- URL Parameters: `productAlias`
- Data Parameters: Nothing
- Success Response: a message stating that the product has been deleted
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Discounts list

This API returns a list of Object Discount.

- URL: `/rest/management/discountManagement`
- Method: `POST`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Array that contains discounts and for each discount we have a vector of associated products
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Create discount

This API returns a list of Object Products used for creating new discount.

- URL: `/rest/management/discountManagement/createDiscount`
- Method: `POST`
- URL Parameters: Nothing
- Data Parameters:
  - `percentage`: percentage of new discount
  - `start`: discount start date
  - `end`: discount end date
  - `productList`: list of the products to discount
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a message stating that the discount has been created
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Edit discount

This API returns an Object Discount and a list of all products.

- URL: `/rest/management/discountManagement/updateDiscount`
- Method: `POST`
- URL Parameters: `idDiscount`
- Data Parameters:
  - `percentage`: percentage of new discount
  - `start`: discount start date
  - `end`: discount end date
  - `productList`: list of the products to discount

- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a message stating that the discount has been modified
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Delete discount

This API returns an Object Discount

- URL: `/rest/management/discountManagement/deleteDiscount`
- Method: `POST`
- URL Parameters: `idDiscount`
- Data Parameters: Nothing
- Success Response: a message stating that the discount has been deleted
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Customer list

This API returns an Array of customer

- URL: `/rest/management/customerManagement`
- Method: `GET`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Array of customer.
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Edit customer

This API returns the customer

- URL: `/rest/management/customerManagement/editCustomer`
- Method: `POST`
- URL Parameters: `idCustomer`
- Data Parameters:
  - `name` the name of customer,
  - `surname` the surname of customer,
  - `fiscalCode` the fiscal code of customer,
  - `address` the address of customer,
  - `email` the email of customer

  - `phoneNumber` the phone number of customer
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a message stating that the discount has been modified
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Delete customer

This API returns the Object Customer to delete

- URL: `/rest/management/customerManagement/deleteCustomer`
- Method: `POST`
- URL Parameters: `idCustomer`
- Data Parameters: Nothing.
- Success Response: a message stating that the discount has been deleted
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Employee list

This API returns the list of all employees.

- URL: `/rest/management/employeeManagement`
- Method: `GET`
- URL Parameters: Nothing
- Data Parameters: Nothing
- Success Response: a JSON Array of employee.
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Create employee

This API return the list of all roles.

- URL: `/rest/management/employeeManagement/createEmployee`
- Method: `POST`
- URL Parameters: Nothing
- Data Parameters:
  - `username` the username of employee
  - `name` the name of employee
  - `surname` the suername of employee
  - `role` the role of employee.

  - `password` the password of employee.
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a message stating that the discount has been created
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Edit employee

This API returns an Object Employee and a list of roles.

- URL: `/rest/management/employeeManagement/editEmployee`
- Method: `POST`
- URL Parameters: `username`
- Data Parameters:
  - `name` the name of employee
  - `surname` the suername of employee
  - `role` the role of employee.
- Data Parameters encoding: `x-www-form-urlencoded`
- Success Response: a message stating that the discount has been modified
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

Delete employee

This API returns the Object Employee that has deleted

- URL: `/rest/management/employeeManagement/deleteEmployee`
- Method: `POST`
- URL Parameters: `username`
- Data Parameters: Nothing.
- Success Response: a message stating that the employee has been deleted
- Error Response: a JSON Object with a key: `isError` and value `true`, an error code, a message and details of the error.

6 Group Members Contribution

At first, we discussed together the major functionalities of the web application we wanted to implement and scheduled how to carry on with the project. To do so, we split into three major groups composed of three people each, in order to develop the web application more efficiently.

Group 1: Composed by Giole Ceccon, Alessandra Pastore and Matteo Pozzer was in charge of developing

- the homepage
- the management of the orders
- the management of the purchases of a product by a customer
- the possible edit of customers and employees accounts
- the various filters to manage the session permits of each type of user

Group 2: Composed by Simone Bortolin, Gil Czaczkes and Gianpietro Nicoletti was in charge of developing

- the ticket part
- the login management
- the management of the invoices
- the upload and management of the media

Group 3: Composed by Simone Bastasin, Davide Colussi and Matteo Lando was in charge of the management side by employees and administrators, consisting of the following functions

- product management
- discount management
- order status management
- employee management
- customer management

For each task, each group coded all the corresponding DAOs, resources (POJO), servlets and JSP pages. This division is not to be considered strict since everyone collaborated with each other in order to obtain a better final work and to allow everyone to know and learn all kind of possible tasks. In particular this approach was useful to maintain a general idea of the project allowing everyone to contribute to the final project as a whole.