

**Web Applications A.Y. 2022-2023**  
**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: 28 April, 2023



Group Acronym	BITSEI	
Last Name	First Name	Badge Number
Mirco	Cazzaro	2076745
Nicola	Boscolo	2074285
Farzad	Shami	2090160
Fabio	Zanini	2088628
Marco	Martinelli	2087646
Christian	Marchiori	2078343
Andrea	Costa	2061900

# 1 Objectives

Our project (Business InTegrated System for Electronic Invoicing) aims to develop a web application utilizable by individual or juridical companies that let them keep track of daily selling to customers, being them physical goods or provided services, and from the data collected print out PDF warnings (used to ask for payments), courtesy PDF invoices and, most important, XML invoices compliant with Italian regulations. Also, data listing, insight plots, and notifications are part of the application.

# 2 Main Functionalities

Our web application allows users to manage potentially more than one company/economic activity, as long as they can have more than one. Users that are logged inside the webapp can modify their companies data, and also bank accounts linked to them. Furthermore, users can perform CRUD operations on their own products/services by their own catalogues, and also of their customers. Users can also extract lists of their products and their customers in a PDF file. Coming to the main part, users can decide to "open" a new invoice to a specific customer: once the user decide to bill, the invoice become closed and no longer editable; a PDF warning is produced and sent to the customer: when payment is performed, by a button the actual invoice in both formats (PDF as courtesy for the customer, XML for the italian system "SISTEMA DI INTERSCAMBIO") is printed out. Concluding, notifications are sent through telegram and / or email, as specified by the user in his preferences, and insight charts are available to him providing many information, as for example which customers revenue to the company in a specified time slot.

In particular, our web application is divided in four main parts:

- User Management: users do not have the possibility to register on the web application: this is due to the fact that the application is designed to be deployed on a specific container (taking advantage of virtualization technologies) for each customer: in this scenario, a single owner user is setted up manually. Moreover, the user is not allowed to create companies, as long for each company managed a license is required. Users can edit their data, data of their companies and of their bank accounts. Users can switch the company currently used in the application by selecting it from a dropdown menu always accessible in all pages of the web application.
- Business Management (the core part of the application): users can perform CRUD operations on customers and products linked to each of their companies, and getting lists of them both through the web application and in PDF format. They can create new invoice entities: an invoice in the early stages of creation has to be considered like an "open debt" to the customer to which the invoice is related: at the very first stage, in fact, the invoice is empty but from that moment, users can start to add rows to the invoice, that correspond to specific sellings, and that will appear on the final invoice as a detail row. Whenever the user decide to bill the invoice, that will become immutable, a document in PDF format will be provided to the user and sent to the customer that will able to perform the payment. Once that the payment has been performed by the customer, the user can emit the actual invoice in XML format that complies with italian regulations. This implementation also complies with the italian legislation, as long a company has 14 working days to submit the invoice to the SDI (Interchange System) before encountering sanctions, and so the invoice can be actually printed out after the payment.
- Notification Management: notifications are directed both to the user and to customers: for what concerns the user, he has the opportunity to get them both via mail and via Telegram. He can set up these preferences on his reserved area, as long as for telegram is also needed to specify his Chat ID. notifications for customers are only via mail. /\*when are performed?\*/
- Insight Printout: the user has the possibility to see, for each company, different typologies of insight plots, as for example histograms on invoices in a certain amount of time, sales in a certain amount of time, invoices filtered by customers, sales filtered by customers, total discount for a certain amount of time.

## 2.1 Deployment - CI/CD

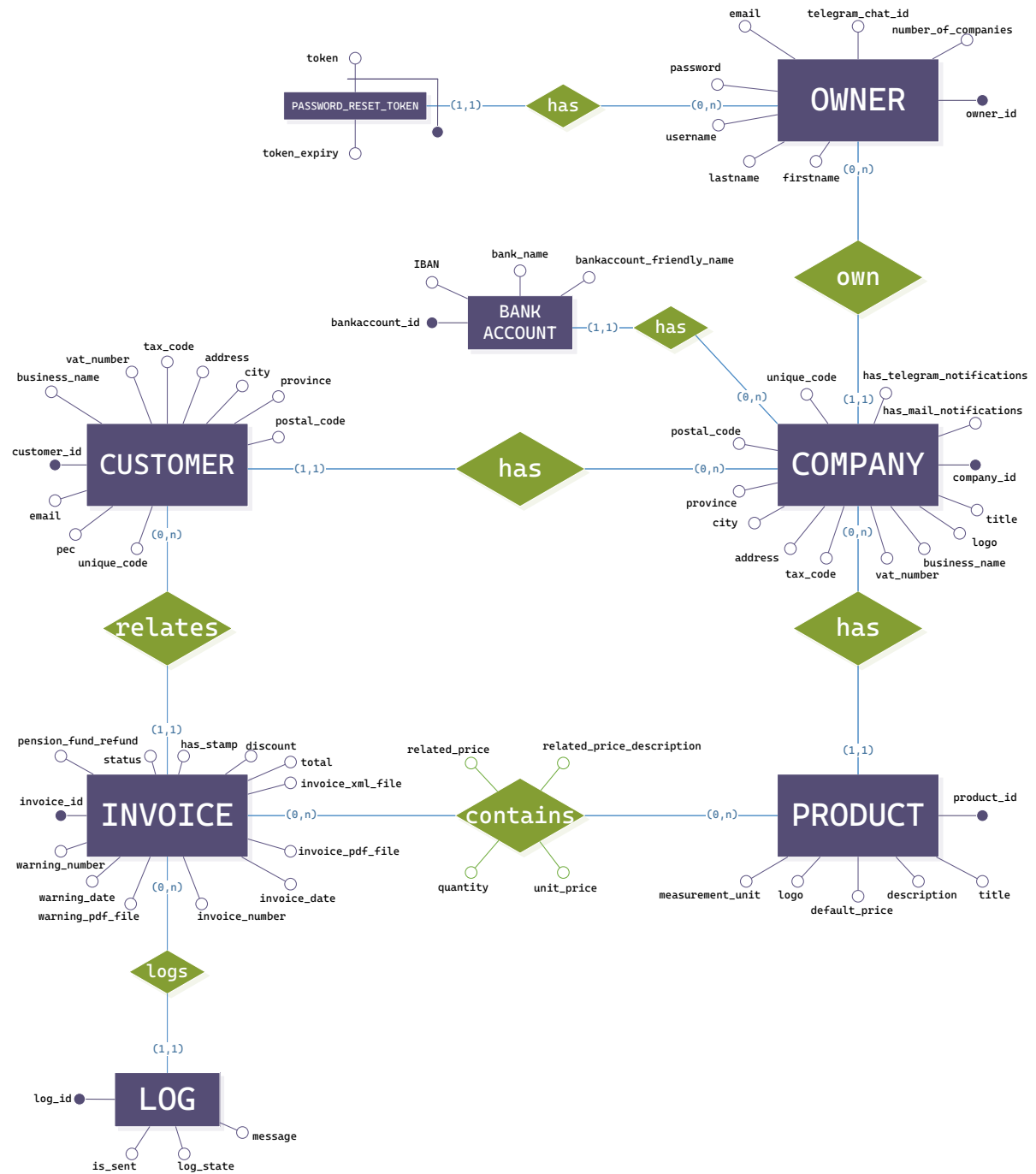
In Bitbucket, Pipelines is a powerful feature that automates the build, test, and deployment process of your code. It's a continuous integration and delivery (CI/CD) tool that helps you streamline your development workflow and ensure that your code is always up-to-date and error-free. Our pipeline consists of two essential steps. The first step is the build phase, which is responsible for compiling the code. This step helps us identify any errors or issues in our code early on, preventing unwanted errors from occurring in the future. If the build phase passes successfully, the next step is the deployment phase. This step deploys the latest version of our code to our real virtual private cloud server, ensuring that our application is always up-to-date and running smoothly.

The screenshot displays the Bitbucket Pipelines interface. On the left, a sidebar shows the pipeline status: a green checkmark indicates success, with a 'Rerun' button. The pipeline is labeled '#32' and shows a commit '0591bdd style(folder): Change structure of project' by 'master'. The pipeline steps are listed as 'Build and test' (25s) and 'Deploy to Artifactory' (35s). The main panel shows the 'Build' tab with a terminal view of the build process. The terminal output includes the command 'cd bitsei-webapp && mvn -s .ci/settings.xml clean verify' and subsequent Maven build logs, including 'Scanning for projects...', 'Building Business Integrated System for Electronic Invoicing 1.0', and 'clean:3.2.0:clean (default-clean) @ bitsei'.

```
Build Artifacts
Build setup 17s
cd bitsei-webapp && mvn -s .ci/settings.xml clean verify 5s
1 + cd bitsei-webapp && mvn -s .ci/settings.xml clean verify
2 [INFO] Scanning for projects...
3 [INFO]
4 [INFO] -----< it.unipd.dei.bitsei:bitsei >-----
5 [INFO] Building Business Integrated System for Electronic Invoicing 1.0
6 [INFO] from pom.xml
7 [INFO] -----[ war ]-----
8 [INFO]
9 [INFO] --- clean:3.2.0:clean (default-clean) @ bitsei ---
10 [INFO]
11 [INFO] --- resources:3.3.0:resources (default-resources) @ bitsei ---
12 [INFO] Copying 1 resource
13 [INFO]
14 [INFO] --- compiler:3.11.0:compile (default-compile) @ bitsei ---
15 [INFO] Changes detected - recompiling the module! :source
16 [INFO] Compiling 122 source files with javac [debug target 17] to target/classes
17 [INFO] /opt/atlassian/pipelines/agent/build/bitsei-webapp/src/main/java/it/unipd/dei/bitsei/servlet/Filter
18 [INFO] /opt/atlassian/pipelines/agent/build/bitsei-webapp/src/main/java/it/unipd/dei/bitsei/servlet/Filter
```

### 3 Data Logic Layer

#### Entity Relationship Diagram



The Entity-Relationship schema is composed of the following entities:

- Owner: contains the pieces of information about the owner of one or more companies that are registered in the system. The primary key owner\_id is an auto-increment (serial) integer ID. The other attributes are firstname (char), lastname (char), username (char), password (char), email (char), and telegram\_chat\_id (char).
- Company: contains the information about a company that is registered in the system. The primary key company\_id is an auto-increment (serial) integer ID. The other attributes are title (char), logo (char), business\_name (char), vat\_number (char), tax\_code (char), address (char), city (char), province (char), postal\_code (char), unique\_code (char), has\_telegram\_notifications (boolean), has\_mail\_notifications (boolean). Then, owner\_id (integer) is an external key pointing to the primary key of the entity Owner, indicating the owner of the company.
- BankAccount: contains the information about the bank account of a company. The primary key bankaccount\_id is an auto-increment (serial) integer ID. The other attributes are IBAN (char), bank\_name (char), and bankaccount\_friendly\_name (char). Then, company\_id (integer) is an external key pointing to the primary key of the entity Company, indicating the company owning to the bank account.
- Customer: contains the information about a customer/client of a company. The primary key customer\_id is an auto-increment (serial) integer ID. The other attributes are business\_name (char), vat\_number (char), tax\_code (char), address (char), city (char), province (char), postal\_code (char), email (char), pec (char), unique\_code (char). Then, company\_id (integer) is an external key pointing to the primary key of the entity Company, indicating the company that has the customer as a client.
- Product: contains the information about a product or a service sold by a company. The primary key product\_id is an auto-increment (serial) integer ID. The other attributes are title (char), default\_price (integer), logo (char), measurement\_unit (char), and description (char). Then, company\_id (integer) is an external key pointing to the primary key of the entity Company, indicating the company that sells the product.
- Invoice: contains the information about an invoice that is being generated and modified or has been already emitted. The primary key invoice\_id is an auto-increment (serial) integer ID. The other attributes are status (integer), warning\_number (char), warning\_date (date), warning\_pdf\_file (char), invoice\_number (char), invoice\_date (date), invoice\_pdf\_file (char), invoice\_xml\_file (char), total (double), discount (double), pension\_fund\_refund (double), has\_stamp (boolean). Then, customer\_id (integer) is an external key pointing to the primary key of the entity Customer, indicating the customer to which the invoice will be billed and sent.
- Log: contains the logs of an invoice and of the process of creating and modifying the invoice. The primary key log\_id is an auto-increment (serial) integer ID. The other attributes are is\_send (boolean), log\_state (boolean), and message (char). Then, invoice\_id (integer) is an external key pointing to the primary key of the entity Invoice, indicating the invoice to which the log is related.

Each of these entities corresponds to a table in the *SQL* schema.

The relation "contains" defines the products listed in an invoice and has the following attributes: quantity (integer), unit\_price (double), related\_price (double), related\_price\_description (char) and purchase\_date (date). The relation, in the *SQL* schema, is translated into the table Invoice\_Product that has as primary key the union of invoice\_id, an external key that points to the invoice of which the invoiceProduct is part, and product\_id, an external key that points to the product which the invoiceProduct relates to.

## 4 Presentation Logic Layer

Here are listed the main pages that will be available on our web application:

- Login Page: here the user can log in filling the form with his username and password;
- Reset password Page: here the user can reset his password;
- Home Page: the home page of the application: the home page will be reached immediately after the login part. Here some of the insight plots will be displayed. on the left side of the page, a nav menu will be shown, with the drop-down menu that will allow to choose the current company used, and all other following sections will be displayed as links;
- Settings Page: here the user can access his reserved area, in which will able to modify some settings such as notification settings, company settings, and bank account settings through forms;
- Product Creation Page: here the user will have a form to insert a new product in the database;
- Product Update Page: here the user will have first the possibility to choose one of his products through a radio button: once one is chosen, he will be redirected to a pre-filled form of the product, in which he will be able to modify it;
- Product Delete Page: here the user will have the possibility to delete a product, through a radio button;
- Product List Page: here a list of all products will be listed: a button will allow the user to download in *PDF* format a list of all the products;
- Customer Creation Page: here the user will have a form to insert a new customer in the database;
- Customer Update Page: here the user will have first the possibility to choose one of his customers through a radio button: once one is chosen, he will be redirected to a pre-filled form of the customer, in which he will be able to modify it;
- Customer Delete Page: here the user will have the possibility to delete a customer, through a radio button;
- Customer List Page: here a list of all customers will be listed: a button will allow the user to download in *PDF* format a list of all the customers;
- Invoice Creation Page: here a new invoice entity will be created, setting up some general data through a form, for example, to which customer the invoice will be related;
- Invoice List Page: here the user can look at the list of his invoices, which will display some general data for each invoice, and through buttons he will be able to update the invoice, edit the rows, and delete the invoice. Furthermore, by clicking on the actual row of the displayed table, a detailed page of the invoice will be accessible.
- Invoice Detail Page: here the user can look at the detail of the selected invoice; he will be also able to close the invoice if it is waiting to be billed (invoice warning PDF will be available for download), and to bill the invoice (actual invoice in both PDF and XML will be available for download) if it is closed. The status will be immediately available. Also from here, buttons for editing rows, elimination, and updating will be available. A section for downloads will be available.
- Invoice Update Page: here the user can edit the data entered in the creation part;
- Invoice Row List Page: by clicking on an invoice a list of all invoice rows of a specific invoice will be available;

- Invoice Row Insertion Page: by clicking on a specific button on open invoice row list, a form to insert a new row on the invoice will be available;
- Detailed Insights Page: a page where the user can see all the insights available.

## 4.1 Login page - Reset password page - New password page

- **Login page:** Through the log-in page, users can access the restricted area in which they can manage their companies' electronic invoicing. To access, it is necessary to enter e-mail and password in the appropriate fields and click on the "Log-in" button. In case the user has forgotten his password, he can click on the "Reset here" button to set a new one.
- **Reset password page:** On this page, you can make a request to reset the password. Once the e-mail is entered and "Send e-mail" is clicked, the user will be emailed a link to enter the new password. Clicking "Cancel" will return the user to the Login page.
- **New password page:** The user must enter the new password twice in the appropriate fields and click on "Confirm reset" to set the new password. Clicking on "Cancel" will return to the Login page.

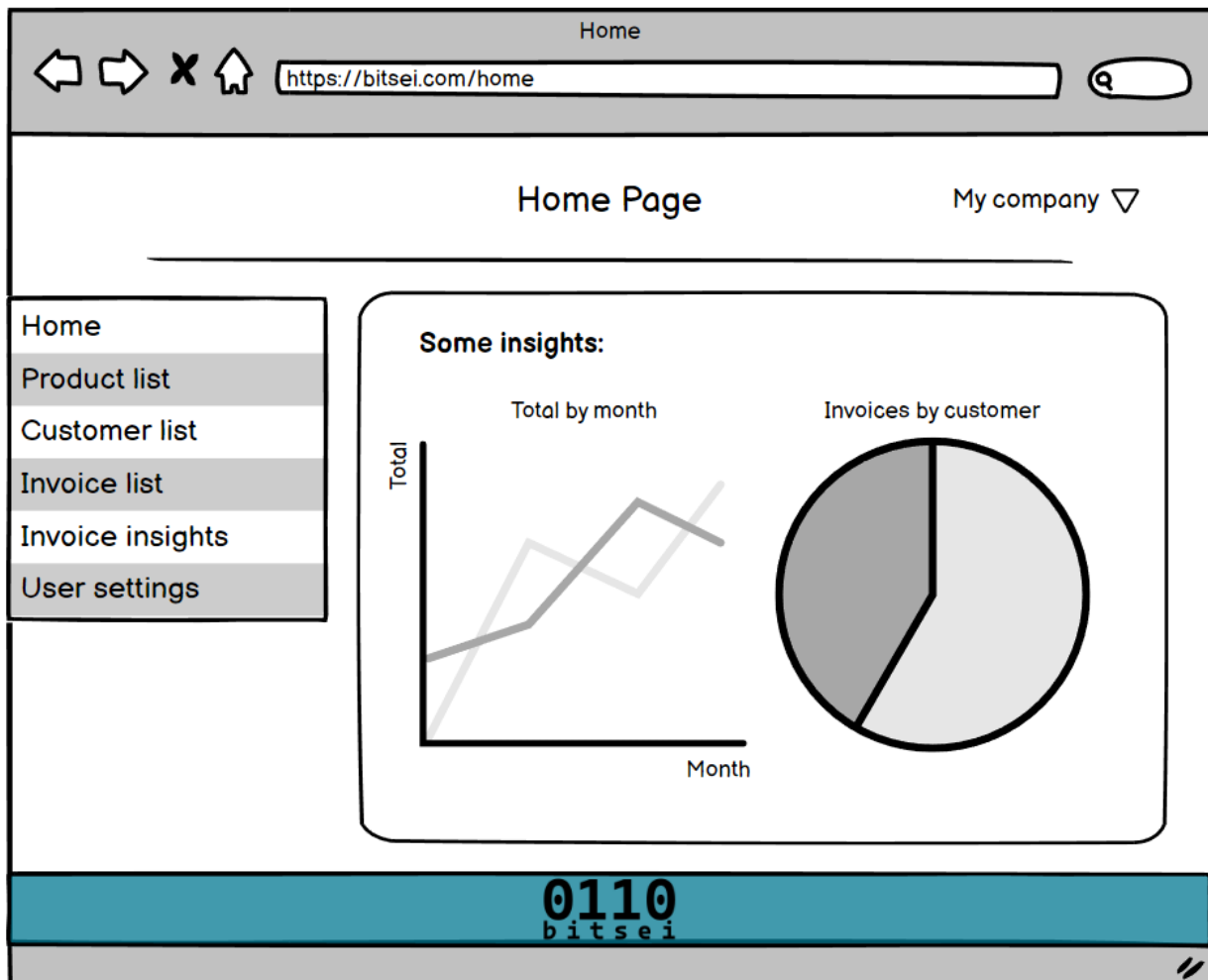
The image displays three wireframe mockups of a web application's authentication pages, arranged in a 2x2 grid with the bottom-right cell empty. Each mockup is contained within a browser window frame.

- Top Left: Login Page**
  - URL: `https://bitsei.com/login`
  - Title: **LOG IN**
  - Illustration: A hand holding a blue card with a person icon.
  - Form Fields: "Email:" with input "Insert your email...", "Password:" with input "Insert the password..." and a toggle icon.
  - Text: "Want to reset your password? [Reset here](#)"
  - Button: "Log-in"
  - Footer: "To register, send us an e-mail" and the **0110 bitsei** logo.
- Top Right: Reset password Page**
  - URL: `https://bitsei.com/reset_psw`
  - Title: **Reset your password**
  - Form Fields: "Email:" with input "Insert your email..."
  - Text: "You will receive a link to reset your password via email shortly."
  - Buttons: "Cancel" and "Send e-mail"
  - Footer: The **0110 bitsei** logo.
- Bottom Center: New password Page**
  - URL: `https://bitsei.com/new_psw`
  - Title: **New password**
  - Form Fields: "New password:" with input "Insert your new password..." and a toggle icon; "Confirm new password:" with input "Insert your new password..." and a toggle icon.
  - Buttons: "Cancel" and "Confirm reset"
  - Footer: The **0110 bitsei** logo.



## 4.2 Home page

The home page will be reached immediately after the login part. Here some of the insight plots will be displayed. On the left side of the page, a nav menu will be shown. In the upper right corner, there is a drop-down menu to choose the desired company.



### 4.3 Invoice list page

This page shows all invoices related to your company.

On the right side, you can set some filters for searching. You can filter invoices by total, discount, date, warning date, pension fund refund, customer, and product. To activate a filter you need to click on the switch next to it. In the list, each invoice is displayed with an identifier and other information to make searching easier. Clicking above the invoice will take you to the "Invoice details" page that allows you to view information regarding the invoice. Above each item in the list are 3 buttons: edit, edit rows, and delete. The first two lead to the respective dedicated pages. the last one is used to delete an invoice (when clicked, a pop-up is shown to confirm the deletion). These buttons are clickable only if the invoice has not reached the "Closed" status. Finally, in the list, there is a button to create a new invoice that leads to the "Create invoice" page.

In the upper right is an input field to search for the desired invoice.

The bottom left shows the number of invoices resulting from the search. In the lower center corner, you can choose how many invoices to display (by default all are displayed). Finally, at the top center, via a menu, you can choose in what order to display the invoices.

The screenshot shows the 'Invoices' page in a web application. The page has a navigation bar with links: Home, Invoices, Products, Customers, Invoice insights, Settings, and My company. The main content area is titled 'Invoices' and features a table of invoices. The table has columns for 'Invoice One', 'Invoice Two', 'Invoice Three', and 'Invoice Four', each with 'Edit', 'Edit rows', and 'Delete' buttons. A 'Create' button is at the bottom right of the table. Below the table, it says '26 invoices found' and 'Show the first 4 items'. A dropdown menu for 'Order by' is set to 'Relevance', with options for 'Date' and 'Money'. A search bar 'search invoice' is at the top right. A 'Filters' section on the right lists various filters: 'By total', 'By discount', 'By invoice date', 'By warning date', 'By Pfr', 'Customer', and 'Product'. Each filter has a toggle switch and input fields. A callout box titled 'All filters:' provides a detailed view of these filters, showing their current states and values.

**All filters:**

- By total:** Toggle: ☐ From: 200€ To: 900€
- By discount:** Toggle: ☐ From: 200€ To: 900€
- By invoice date:** Toggle: ☒ Start: 01/04/2023 End: 28/04/2023
- By warning date:** Toggle: ☐ Start: 01/04/2023 End: 28/04/2023
- By Pfr:** Toggle: ☐ From: From... To: To...
- Customer:** Search: Gianni, Tony. Selected: Gianni X
- Product:** TNT, Vaseline. Selected: TNT X

## 4.4 Create invoice page - Edit invoice page - Invoice rows list page

- **Create invoice page:** On this page, you can create a new invoice. At the top are fields to be filled in. To add rows to the invoice, you must first complete its creation. At the bottom are two buttons, one to cancel and return to the invoice list and one to confirm the creation of the invoice.
- **Edit invoice page:** This page is similar to the invoice creation page only it refers to an invoice that has already been created previously. In this case, the fields are already filled with the previous values. At the bottom, there are two buttons, one to cancel and return to the "Invoice list" page and one to save the changes.
- **Invoice rows page:** A table containing all rows concerning the invoice is displayed. Each cell in the table can be edited by double-clicking on it. Using checkboxes, various invoices can be selected and then deleted using the "Delete" button. There is a button in the table to add a new row (Initially it will be filled with default values). Finally, there is a button to return to the previous page.

The first screenshot shows the 'Create an Invoice' page. It has a navigation bar with 'Home', 'Invoices', 'Products', 'Customers', 'Invoice insights', and 'Settings'. Below the navigation bar, there is a form titled 'Create the invoice' with fields for 'Customer:', 'Discount:', and 'Pension found refund:'. At the bottom, there are 'Cancel' and 'Create' buttons.

The second screenshot shows the 'Edit the Invoice' page. It has a similar navigation bar. Below the navigation bar, there is a form titled 'Edit the invoice' with fields for 'Customer:', 'Discount:', and 'Pension found refund:'. Below these fields, there is a table titled 'Rows:' with columns 'Product', 'Quantity', 'Unit price', 'Related price', and 'Purchase date'. The table contains three rows of data. At the bottom, there are 'Cancel' and 'Save changes' buttons.

The third screenshot shows the 'Invoice rows' page. It has a navigation bar with 'Home', 'Invoices', 'Products', 'Customers', 'Invoice insights', and 'Settings'. Below the navigation bar, there is a table titled 'Invoice rows' with columns 'Product', 'Quantity', 'Unit price', 'Related price', 'Purchase date', and 'Select'. The table contains three rows of data. At the bottom, there are 'Delete' and 'Turn back' buttons.

Product	Quantity	Unit price	Related price	Purchase date
Product 1	40	30	1200	1/1/2020
Product 2	38	40	1520	2/2/2022
Product 3	41	50	2050	3/3/2023

Product	Quantity	Unit price	Related price	Purchase date	Select
Product 1	40	30	1200	1/1/2020	<input checked="" type="checkbox"/>
Product 2	38	40	1520	2/2/2022	<input checked="" type="checkbox"/>
Product 3	41	50	2050	3/3/2023	<input type="checkbox"/>

## 4.5 Invoice details page

On this page, you can view all the details of an invoice. The fields cannot be edited. At the bottom is a label indicating the status of the invoice (can be Created, Closed, or Completed). Then there are buttons to edit rows, edit the invoice, and delete the invoice. These buttons are only available if the status is "Created". By clicking on "Delete" a pop-up is shown to confirm the deletion. Finally, there is a button to return to the previous page. At the top right is a button for downloading the invoice document. This button is disabled in the "Created" state, in the "Closed" state only the invoice warning will be downloaded. In the "Completed" state you will be able to choose (through a pop-up) whether to download the invoice warning or the invoice in either *PDF* or *XML* format. In the pop-up if "warning" is selected then only *PDF* is selectable.



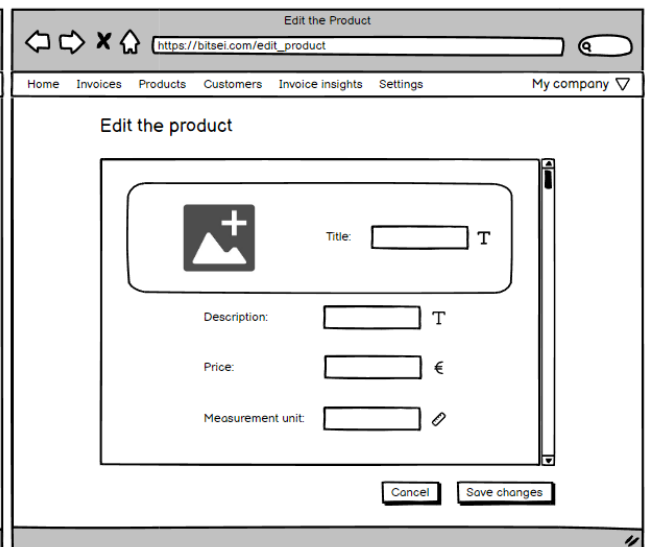
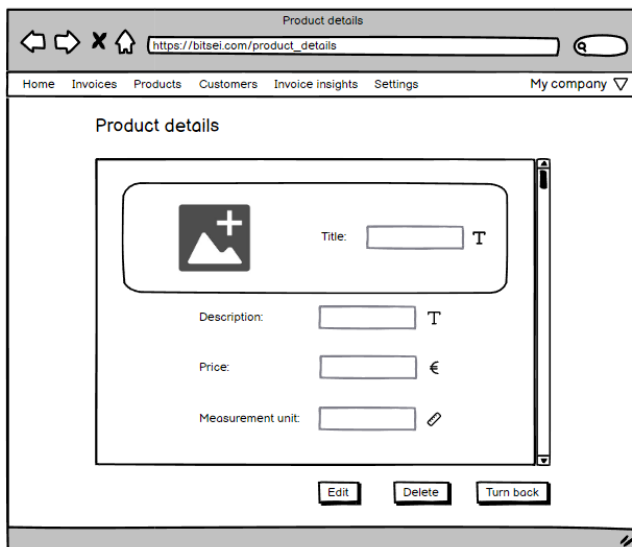
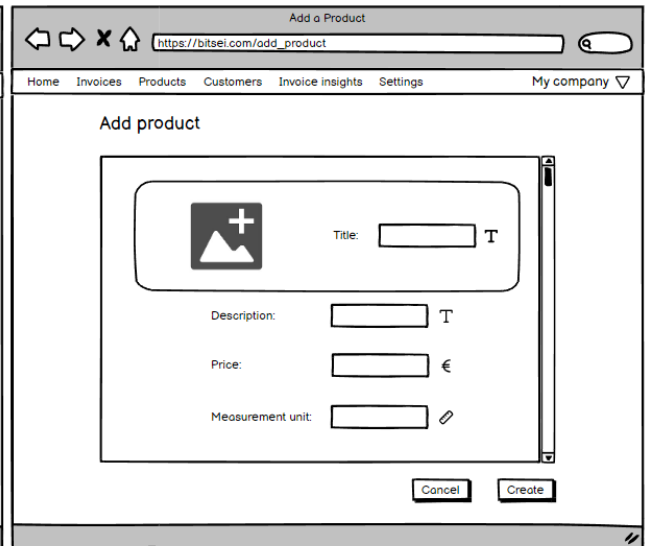
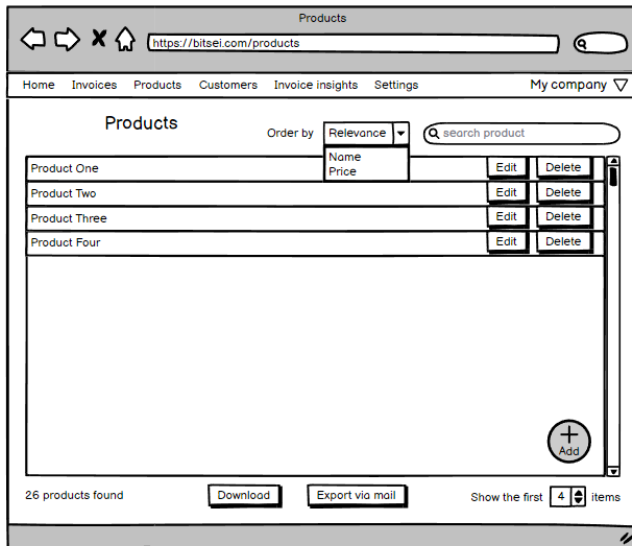
## 4.6 Invoice insights page

On this page, you can view some charts related to your company. The charts available are invoices by date, total by date, discount by date, invoices by customer, and total by customer. It is possible to change charts by selecting the desired one in the menu tab. Dates in the x-axis can be grouped by month, day, or year. Filters can be applied on the right so that only certain elements are taken into account. At the top right are two buttons for downloading and exporting the chart by e-mail.



## 4.7 Product list page - Create product page - Product details page - Edit product page

- **Product list page:** This page shows all products related to your company. In the list, each product is displayed with his name. Clicking above the product will take you to the "Product details" page which allows you to view information regarding the product. Above each item in the list are 2 buttons: edit, and delete. The first lead to the dedicated page. The second one is used to delete a product (when clicked, a pop-up is shown to confirm the deletion). In the list, there is a button to add a new product that leads to the "Create product" page. In the upper right is an input field to search for the desired product. The bottom left shows the number of products resulting from the search. In the lower right corner, you can choose how many products to display (by default all are displayed). Finally, at the top center, via a menu, you can choose in what order to display the products. At the bottom center are two buttons, one for downloading the product list summary and the other for exporting it via email.
- **Create product page:** On this page, you can add a new product. There are fields to be filled in. Still further down are two buttons, one to cancel and return to the product list and one to confirm the addition of the product.
- **Product details page:** On this page you can view all the details of a product. The fields cannot be edited. At the bottom, there are buttons to: edit the product, delete the product. By clicking on "Delete" a pop-up is shown to confirm the deletion. Finally, there is a button to return to the previous page.
- **Edit product page:** This page is similar to the product creation page only it refers to a product that has already been added previously. In this case, the fields are already filled with the previous values. At the bottom, there are two buttons, one to cancel and return to the "Product list" page and one to save the changes.



## 4.8 Customer list page - Create customer page - Customer details page - Edit customer page

- **Customer list page:** This page shows all customers related to your company. In the list, each customer is displayed with his name. Clicking above the customer will take you to the "Customer details" page that allows you to view information regarding the customer. Above each item in the list are 2 buttons: edit, and delete. The first lead to the dedicated page. The second one is used to delete a customer (when clicked, a pop-up is shown to confirm the deletion). In the list, there is a button to add a new customer that leads to the "Create customer" page. In the upper right is an input field to search for the desired customer. The bottom left shows the number of customers resulting from the search. In the lower right corner, you can choose how many customers to display (by default all are displayed). Finally, at the top center, via a menu, you can choose in what order to display the customers. At the bottom center are two buttons, one for downloading the customer list summary and the other for exporting it via email.
- **Create customer page:** On this page, you can add a new customer. There are fields to be filled in. At the bottom are two buttons, one to cancel and return to the customer list and one to confirm the addition of the customer.
- **Customer details page:** On this page, you can view all the details of a customer. The fields cannot be edited. At the bottom, there are buttons to: edit the customer, delete the customer. By clicking on "Delete" a pop-up is shown to confirm the deletion. Finally, there is a button to return to the previous page.
- **Edit customer page:** This page is similar to the customer creation page only it refers to a customer that has already been added previously. In this case, the fields are already filled with the previous values. At the bottom, there are two buttons, one to cancel and return to the "Customer list" page and one to save the changes.



Customers
https://bitsei.com/customers

Home Invoices Products Customers Invoice insights Settings My company

Customers

Order by Relevance

search customer

Customer One	Name	Edit	Delete
Customer Two	City	Edit	Delete
Customer Three	Province	Edit	Delete
Customer Four		Edit	Delete

+ Add

26 customers found
Download
Export via mail
Show the first 4 items

Add a Customer
https://bitsei.com/add\_customer

Home Invoices Products Customers Invoice insights Settings My company

Add customer

Business name:

Vat Numb: Tax Code:

Address: Province:

City: CAP:

E-mail: PEC:

Cancel Create

Customer details
https://bitsei.com/customer\_details

Home Invoices Products Customers Invoice insights Settings My company

Customer details

Business name:

Vat Numb: Tax Code:

Address: Province:

City: CAP:

E-mail: PEC:

Edit Delete Turn back

Edit the Customer
https://bitsei.com/edit\_customer

Home Invoices Products Customers Invoice insights Settings My company

Edit the customer

Business name:

Vat Numb: Tax Code:

Address: Province:

City: CAP:

E-mail: PEC:

Cancel Save changes

## 4.9 Settings page


This page is divided into two sections: Company settings and Bank Account settings. In the first one, you can view or edit all the information regarding your company. In the second one, you can manage your bank accounts. In the company section the fields are already filled with the previous values, in the bank account settings, the accounts are represented in a table. If you want to change something at the end you need to click on "Save changes", otherwise you can click on "Cancel" to return back.

Settings

https://bitsei.com/settings

Home Invoices Products Customers Invoice insights Settings My company ▾

### Company settings



Name:

Business name:

Vat Numb:

Tax Code:

Address:

Province:

City:

CAP:

Email notifications ☒

Telegram notifications ☐

Cancel

Save changes

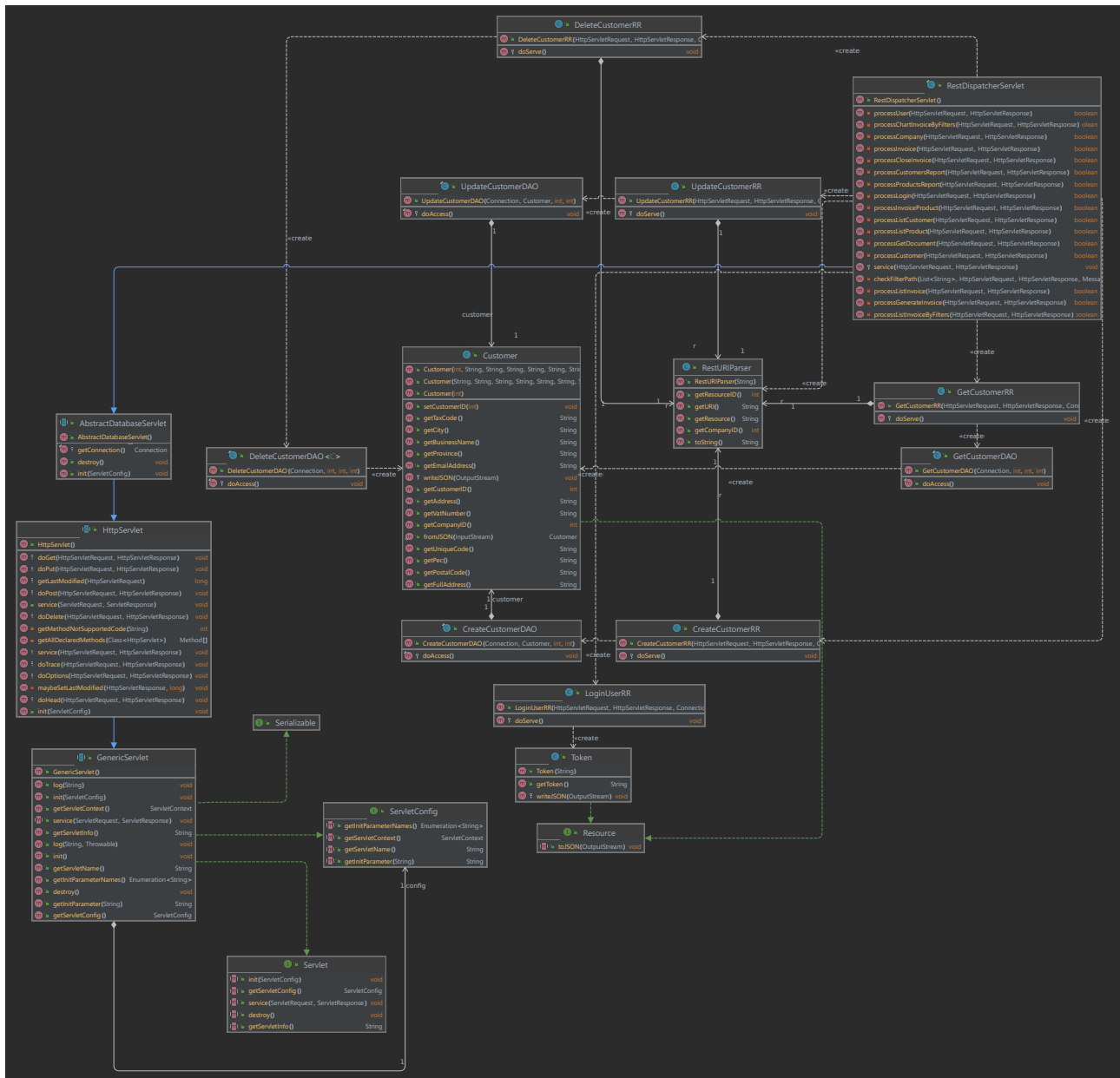
### Bank account settings

Account nam	Bank name	IBAN	Select
Account 1	Intesa San Paol	NL20RABO9372718300	<input checked="" type="checkbox"/>
Account 2	Banca sella	IT53J030020328088856149936	<input checked="" type="checkbox"/>
Account 3	Swiss bank	FR7730003000704714984719F84	<input type="checkbox"/>
			<div>+ Add</div>

Delete

## 5 Business Logic Layer

### 5.1 Class Diagram (sample on customer management)



Full class diagram available as vector image [here](#).

In the class diagram above, we can see the classes used to handle customer creation, deletion, update, and loading. We can see that there is a resource named *Customer* which implements the constructors and the get methods for the parameters (which corresponds to the parameters in the ER schema) and set method for the parameter that corresponds to the primary key in the ER schema. This *Customer* resource is implemented using rest.

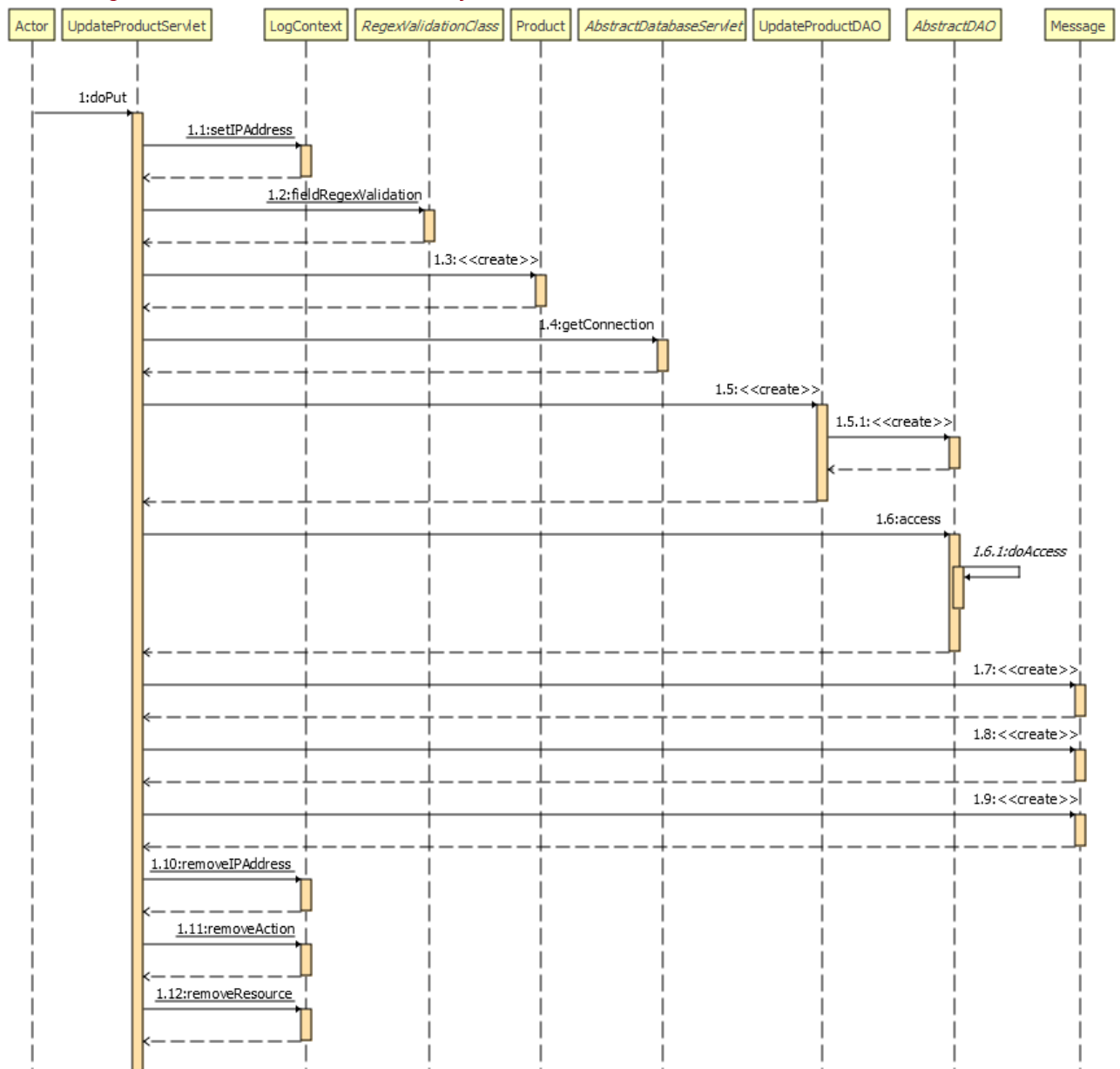
There is a *RestDispatcherServlet* servlet that, using the *RestURIParser* class, parses the *URI* and understands the resource and the method required. Then, the *RestDispatcherServlet* calls the corresponding rest resource (for example, if the request is a get for *Customer*, the *RestDispatcherServlet* calls *GetCustomerRR*). Each one of these resources retrieves the parameters passed in *JSON* format, if any, and check that they are as expected (checks for image file extensions, date consistency, and so on). After doing this, the rest resource calls the *DAO* for the requested method. The *DAO* checks that the user is authorized (check ownership statements) to make the actions required: if the user, for example, is not the owner of a company, he cannot change things about that company; he can only modify things about his companies. Then, if all is good, the *DAO* executes the *SQL* statement and gets (/stores) the data from (/in) the database. Then the *DAO* returns the values requested.

The *DAOs* for the creation of resources implement a *POST* request; the *DAOs* for deletion of resources implement a *DELETE* request; those for update implement a *PUT* request; finally, the *DAOs* for loading a resource, implement a *GET* request.

All the resources, apart from *Product*, are developed using *REST*. The *Product* resource, is implemented using servlets+*DAOs*. In this case, everything works similarly, just instead of having rest resources we have a servlet for each method (creation, deletion, loading, update). Each one of these servlets makes the checks on the parameters and calls the corresponding *DAOs*, which implements the *POST/DELETE/GET/PUT* method and executes the corresponding *SQL* statement (after checking ownership and possible data access violations).

## 5.2 Sequence Diagram

Sequence diagram for traditional product entity update.



In the schema above is shown the sequence diagram for the update of a product (servlet + DAO).

The user sends a *PUT* request to the web server. The web server calls the *UpdateProductServlet* and then sets the *IP* address of *LogContext* and also checks, using the class *regex validation*, that the parameters passed as inputs are valid.

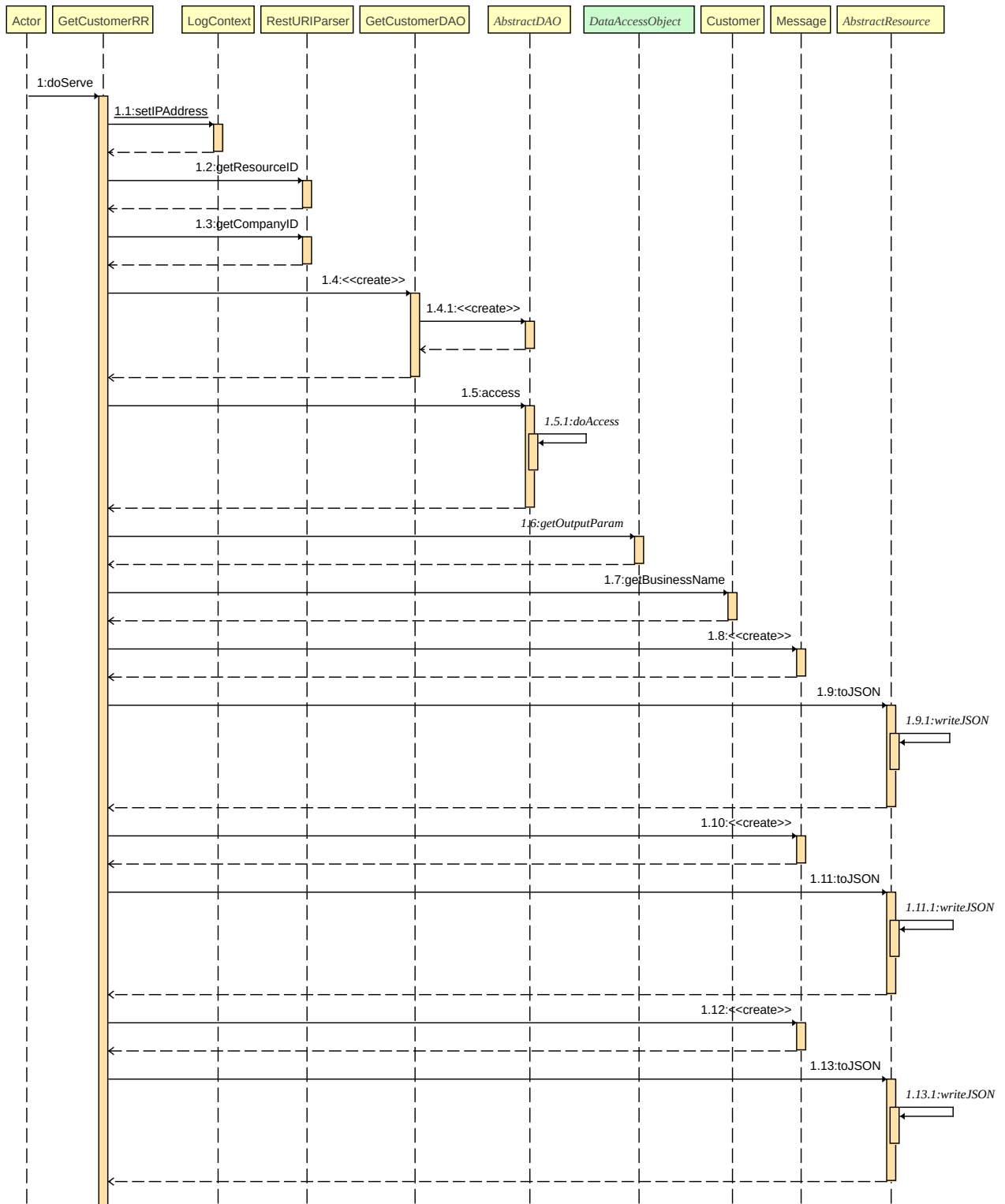
Then a new object of class *Product* is created, with the parameters passed in input. After this, the *UpdateProductServlet* calls the *getConnection()* method of the *AbstractDatabaseServlet*. After doing this, the servlet instantiate a new object of the class *UpdateProductDAO*, which extends *AbstractDAO*.

Then, with the *doAccess()* method of the class *AbstractDAO*, the *DAO* connects to the database.

After connecting to the database and executing the *SQL* statement, three messages for the servlet are created.

Finally, the resources are removed for the clean-up.

## Sequence diagram for the REST resource “get customer”.



In the schema above is shown the sequence diagram for the get of a customer (rest resource + DAO).

The user sends a *GET* request to the web server, which analyzes the request and calls the *GetCustomerRR*. This

class calls the *setIPAddress()* of the *LogContext* and then retrieves the *URI* parameters with the *getResourceID()* and *getCompanyID()* of the *RestURIParser* class.

After having the parameters stored, an instance of the class *GetCustomerDAO* is created by the *REST* resource, and also an instance of *AbstractDAO* is created since the *GetCustomerDAO* extends *AbstractDAO*. The *DAO* accesses the database with the *doAccess()* method and retrieves the customer.

The servlet calls the *getOutputParam()* method of the *DataAccessObject* class and gets the output parameters asked by the user. An instance of the *Customer* class is retrieved and, with the *getBusinessName()* method, the servlet gets the business name of the customer retrieved.

After doing all of this, three messages are created and, with the *toJSON()* method of the *AbstractResource* class, a *JSON* with the data of these messages is written (with *writeJSON()*). The content of this message is based on the results of the GET request.

### 5.3 REST API Summary

Most of the functionalities in our web app are available via *REST API*. The table below gathers all the *REST* resources implemented.

URI	Method	Description
rest/user	GET	Get the data of the user currently logged into the system
rest/user/reset-password	POST	Send an email to the user with a link to reset the password
rest/user/change-password	POST	Change the user password with the one passed in the request body
rest/login	POST	Authenticate the user using the <i>JSON</i> passed in the request body
rest/company	GET	Get the list of all the companies associated with the current user
rest/company	POST	Create a new company and associate it to the current user
rest/company/image/{id}	GET	Get the company image
rest/company/{id}	GET	Get the company associated with the <i>companyId</i> passed through the <i>URL</i>
rest/company/{id}	PUT	Update the company associated with the <i>companyId</i> passed through the <i>URL</i> with the parameters passed in the request body
rest/company/{id}	DELETE	Delete the company associated with the <i>companyId</i> passed through the <i>URL</i>
rest/customer	POST	Create a new customer using the <i>JSON</i> passed in the request body
rest/customer/{id}	GET	Get the customer associated to the <i>customerId</i> passed through the <i>URL</i>
rest/customer/{id}	DELETE	Delete the customer associated with the <i>customerId</i> passed through the <i>URL</i>
rest/customer/{id}	PUT	Update the customer associated to the <i>customerId</i> passed through the <i>URL</i> with the parameters passed in the request body
rest/customerreport/	GET	Generate a list of all customers of the current company in a PDF file
rest/productreport/	GET	Generate a list of all products of the current company in a PDF file
rest/invoice	POST	Create a new invoice using the <i>JSON</i> passed in the request body
rest/invoice/{id}	GET	Get the invoice associated with the <i>invoicId</i> passed through the <i>URL</i>
rest/invoice/{id}	DELETE	Delete the invoice associated with the <i>invoicId</i> passed through the <i>URL</i>



rest/invoice/{id}	PUT	Update the invoice associated to the <i>invoiceld</i> passed through the <i>URL</i> with the parameters passed in the request body
rest/closeinvoice/{id}	PUT	Update the invoice status to 1: this will block the possibility of updating the invoice and of editing its rows. A document in <i>PDF</i> format will be created and will be available for download and for being sent via E-mail and/or Telegram.
rest/generateinvoice/{id}	PUT	Update the invoice status to 2; this will block the possibility of updating the invoice and of editing its rows. Two documents in <i>PDF</i> and <i>XML</i> format will be created and will be available for download and for being sent via E-mail and/or Telegram.
rest/invoiceproduct/{inv_id}/{prod_id}	POST	Create a new invoice product related to the invoice with <i>{inv_id}</i> and the product with <i>{prod_id}</i> using the <i>JSON</i> passed in the request body
rest/invoiceproduct/{inv_id}/{prod_id}	GET	Get the invoice product associated with the <i>invoiceld</i> and the <i>productld</i> passed through the <i>URL</i>
rest/invoiceproduct/{inv_id}/{prod_id}	DELETE	Delete the invoice product associated with the <i>invoiceld</i> and the <i>productld</i> passed through the <i>URL</i>
rest/invoiceproduct/{inv_id}/{prod_id}	PUT	Update the invoice product associated with the <i>invoiceld</i> and the <i>productld</i> passed through the <i>URL</i> with the parameters passed in the request body
rest/list-invoice/{id}	GET	Get the list of all the invoices associated with the <i>companyld</i> passed through the <i>URL</i>
rest/list-customer/{id}	GET	Get the list of all the customers associated with the <i>companyld</i> passed through the <i>URL</i>
rest/list-product/{id}	GET	Get the list of all the products associated with the <i>companyld</i> passed through the <i>URL</i>
rest/getdocument/{type}/invoice/{id}	GET	Get the document specified by type: (0: <i>PDF</i> Warning; 1: <i>PDF</i> Invoice; 2: <i>XML</i> Invoice)
rest/filter-invoices	POST	Get the list of all the invoices filtered with the parameters passed in the request body in <i>JSON</i> format

rest/charts	POST	Plot the filtered chart according to the parameters passed in the request body in <i>JSON</i> format
-------------	------	--

Table 2: *REST* resources implemented by the *BITSEI* Web App

## 5.4 REST Error Codes

Error Code	HTTP Status Code	Description
200	OK	Standard response if there isn't any error
201	CREATED	Resource created
400	BAD REQUEST	Bad formatted request
401	UNAUTHORIZED	Authentication failed
404	NOT FOUND	Resource not found
405	METHOD NOT ALLOWED	HTTP method not allowed
406	NOT ACCEPTABLE	MIME not correct
500	INTERNAL SERVER ERROR	Standard response for errors of the server, but especially database-related ones

Table 3: Error codes generated by the *BITSEI* Web App

## 5.5 REST API Details

At the end of every *URI* used in the Rest calls below, it will be automatically added the following path:

company/{id}

which specifies the id of the company currently logged into the system.

Regarding the authentication aspect, when the user logs into the system a session token containing the owner id and the owner email is set.

If this token is not set, every call will be blocked until the user authenticates himself.

This token is particularly useful because it allows checking at every call to a *REST* Resource that the user is authorized for the operation he's trying to do.

So, the final flow will be:

- The user logs into the system using his E-mail and password.
- After authentication, the *JWT* token containing the owner identifier number and the owner email address is set.
- After the login phase, the user company is set by default. If a user has more companies, he can select which one to use.  
The company identifier is set as a session attribute.
- From now on, all the pages will show only the data associated with this company.

## Get the data of the user

The following endpoint allows to get the data of the user currently logged into the system.

- URL:
  - `/rest/user`
- Method:
  - GET
- URL Parameters:
  - No *URL* parameters required.
- Data Parameters:
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:

```
{
  "user": {
    "firstname": "Burr",
    "lastname": "Myciah",
    "username": "mmcclosh0",
    "email": "mchaudret0@dailymail.co.uk",
    "telegram_chat_id": null
  }
}
```
- Error Response:

Code	When
500	If any fatal error occurs in users listing
500	If any database-related error occurs

## Reset the password of a user registered in the system

The following endpoint allows sending a link to a user via email to reset his password.

- URL:
  - `/rest/user/reset-password`
- Method:
  - `POST`
- URL Parameters:
  - No *URL* parameters required.
- Data Parameters:
  - `ownerMail = {String}`  
The mail inserted by the user in which to receive the link to reset the password.
- Success Response:
  - Code: 200
  - Content:
    - {  
*message: Reset password token send to your email, check your inbox.*  
}
- Error Response:

Code	When
500	If any fatal error occurs in users listing
500	If any database-related error occurs

## Change the password of a user registered in the system

The following endpoint allows changing a user's password from inside the system.

- URL:
  - `/rest/user/change-password`
- Method:
  - `POST`
- URL Parameters:
  - No *URL* parameters required.
- Data Parameters:
  - `ownerId = {int}, ownerMail = {String}`  
The owner identifier and the owner's mail contained in the authentication token.
  - `newPassword = {String}`  
The new password to be set for the current user.
  - `resetToken = {String}`  
The reset token received via mail.
  -
- Success Response:
  - Code: 200
  - Content:
    - {  
*message: "Successfully done, now you can log in with your new password"*  
}
- Error Response:

Code	When
404	If the user who is trying to access this resource is not registered in the system
500	If any database-related error occurs

## Authenticate an user

The following endpoint allows authenticating a user into the system.

- URL:
  - /rest/login
- Method:
  - POST
- URL Parameters:
  - No *URL* parameters required.
- Data Parameters:
  - The *JSON* representation of the user email and password.
- Success Response:
  - Code: 200
  - Content:

```
{  
    "token": "Bearer eyJraWQiOiJrMSIsImFsZyI6IjEzMjU2In0.  
            eyJpc3MiOiJiaXRzZWlfd2ViYXBwliwiYXVkljoidXNlcilslmV4cCI6MTY4MjY4NDM1Mywia  
            .FrVxA0wcw5nBcWmUl67RPA3lxMxcfij3Ri0ny8SJElINQNEdymF-fuTD2NQSWph3LwDvUnw  
            -VkL5FkEeBOo21zmai2lUKw4V1GXF0eaoQtLwTorgDJvAkUM0EfA6tXLx74hg_CCtxpd6llu0  
            -ws3iAxmBUcjw"
```
- Error Response:

Code	When
401	If the email and/or password inserted by the user are wrong
500	If any error occurs in sending the authentication response
500	If any database-related error occurs

## List the companies associated to an user

The following endpoint allows to list all the companies associated to an user.

- URL:
  - /rest/company
- Method:
  - GET
- URL Parameters:
  - No *URL* parameters required.
- Data Parameters:
  - ownerId = {int}  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:

```
{ "resource-list":  
  [{ "company_id":1, "title":" Jakarta", "logo":" http://localhost:8080/  
    bitsei-1.0/rest/company/image/1", "business_name":" Jakarta inc.", "  
    vat_number":" 68856-067", "tax_code":" 43289-020", "address":" Via Roma  
    45", "province":" MI", "city":" Milano", "postal_code":null, "  
    unique_code":" 1", "has_mail_notifications":false, "  
    has_telegram_notifications":false }  
  ] }
```
- Error Response:

Code	When
500	If any fatal error occurs in companies listing
500	If any database-related error occurs

### Create a new company associated with a user

The following endpoint allows to create a new company and associate it to a user already registered into the system.

- URL:
  - /rest/company
- Method:
  - POST
- URL Parameters:
  - No *URL* parameters required.
- Data Parameters:
  - ownerId = {int}  
The identifier of the owner to which associate the new company being created.
  - The *JSON* representation of the company to create.
- Success Response:
  - Code: 200
  - Content:

```
{ "company_id":1, "title":" Jakarta", "logo":" http://localhost:8080/bitsei-1.0/rest/company/image/1", "business_name":" Jakarta inc.", "vat_number":" 68856-067", "tax_code":" 43289-020", "address":" Via Roma 45", "province":" MI", "city":" Milano", "postal_code":null, "unique_code":" 1", "has_mail_notifications":false, "has_telegram_notifications":false }
```
- Error Response:

Code	When
500	If the user has not bought a license for the new company he wants to create
500	If any database-related error occurs



### Get the image associated with a company

The following endpoint allows to get the company image of the company specified in the URL.

- URL:
  - `/rest/company/image/{id}`
- Method:
  - `POST`
- URL Parameters:
  - `company = {int}`  
The identifier of the company for which to retrieve the image.
- Data Parameters:
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:  
*The .png image associated with the specified company*
- Error Response:

Code	When
500	If the user is not allowed to access this company
500	If any fatal error occurs in the retrieval of the image
500	If any database-related error occurs

## Create a new customer associated with a company

The following endpoint allows the creation of a new customer and associates it with the company currently logged into the system.

- URL:
  - `/rest/customer/`
- Method:
  - `POST`
- URL Parameters:
  - `companyId = {int}`  
The identifier of the company to which associate the new customer.
- Data Parameters:
  - The *JSON* representation of the customer to create
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:

```
{ "customerID": "1", "businessName": "AppleLike Inc.", "vatNumber": "56479-182", "taxCode": "22342-305", "address": "Via Venezia 79", "city": "Padova", "province": "PD", "postalCode": null, "emailAddress": "applelike@google.com", "pec": "applelike@pec-mac.com", "uniqueCode": "1", "companyID": "1" }
```
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any database-related error occurs

## Get a customer associated with a company

The following endpoint allows to get a customer associated with the current company logged into the system.

- URL:
  - `/rest/customer/{id}`
- Method:
  - GET
- URL Parameters:
  - `customerId = {int}`  
The identifier of the customer to retrieve.
  - `company = {int}`  
The identifier of the company currently logged into the system.
- Data Parameters:
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:

```
{ "customerId": "1", "businessName": "AppleLike Inc.", "vatNumber": "56479-182", "taxCode": "22342-305", "address": "Via Venezia 79", "city": "Padova", "province": "PD", "postalCode": null, "emailAddress": "applelike@google.com", "pec": "applelike@pec-mac.com", "uniqueCode": "1", "companyId": "1" }
```
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any database-related error occurs

## Delete a customer associated with a company

The following endpoint allows to delete a customer associated with the current company logged into the system.

- URL:
  - `/rest/customer/{id}`
- Method:
  - DELETE
- URL Parameters:
  - `customerId = {int}`  
The identifier of the customer to delete.
  - `company = {int}`  
The identifier of the company currently logged into the system.
- Data Parameters:
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:

```
{ "customerId": "1", "businessName": "AppleLike Inc.", "vatNumber": "56479-182", "taxCode": "22342-305", "address": "Via Venezia 79", "city": "Padova", "province": "PD", "postalCode": null, "emailAddress": "applelike@google.com", "pec": "applelike@pec-mac.com", "uniqueCode": "1", "companyId": "1" }
```
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any database-related error occurs

## Update a customer associated with a company

The following endpoint allows to update a customer associated with the current company logged into the system.

- URL:
  - `/rest/customer/{id}`
- Method:
  - PUT
- URL Parameters:
  - `customerId = {int}`  
The identifier of the customer to delete.
  - `company = {int}`  
The identifier of the company currently logged into the system.
- Data Parameters:
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:

```
{ "customerId": "1", "businessName": "AppleLike Inc.", "vatNumber": "56479-182", "taxCode": "22342-305", "address": "Via Venezia 79", "city": "Padova", "province": "PD", "postalCode": null, "emailAddress": "applelike@google.com", "pec": "applelike@pec-mac.com", "uniqueCode": "1", "companyID": "1" }
```
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any database-related error occurs

## List the invoices associated with a company

The following endpoint allows to list all the invoices associated with the current company logged into the system.

- URL:
  - /rest/list-invoice
- Method:
  - GET
- URL Parameters:
  - company = {int}  
The identifier of the company currently logged into the system.
- Data Parameters:
  - ownerId = {int}  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:

```
{ "resource-list":  
  [{ "invoice": { "invoice_id": 1, "customer_id": 1, "status": 0, "warning_number": 1, "warning_date": "2022-01-05", "warning_pdf_file": "warning_pdf_file1.pdf", "invoice_number": "1", "invoice_date": "2022-02-06", "invoice_pdf_file": "invoice_pdf_file1.pdf", "invoice_xml_file": "invoice_xml_file1.xml", "total": 168.3, "discount": 15.0, "pension_fund_refund": 4.1, "has_stamp": false } },  
  { "invoice": { "invoice_id": 29, "customer_id": 29, "status": 0, "warning_number": 29, "warning_date": "2022-08-29", "warning_pdf_file": "warning_pdf_file29.pdf", "invoice_number": "29", "invoice_date": "2022-09-14", "invoice_pdf_file": "invoice_pdf_file29.pdf", "invoice_xml_file": "invoice_xml_file29.xml", "total": 846.8, "discount": 75.4, "pension_fund_refund": 1.2, "has_stamp": false } }  
]}
```
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any database-related error occurs

### List the customers associated with a company in a PDF file

The following endpoint allows to list all the customers associated with the current company logged into the system in a *PDF* file.

- URL:
  - `/rest/customerreport`
- Method:
  - GET
- URL Parameters:
  - `company = {int}`  
The identifier of the company currently logged into the system.
- Data Parameters:
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:
    - {  
  *message: "PDF file successfully generated!"*  
}
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any error occurs in generating the <i>PDF</i> file
500	If any database-related error occurs

### List the products associated to a company in a PDF file

The following endpoint allows to list all the products associated with the current company logged into the system in a *PDF* file.

- URL:
  - /rest/productreport
- Method:
  - GET
- URL Parameters:
  - company = {int}  
The identifier of the company currently logged into the system.
- Data Parameters:
  - ownerId = {int}  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content: *message: "PDF file successfully generated!"*
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any error occurs in generating the <i>PDF</i> file
500	If any database-related error occurs



## Close an invoice

The following endpoint allows closing the invoice having the identifier passed in the URL, blocking the possibility of updating and modifying it.

When the invoice is successfully closed, a document in *PDF* format will be available for download and will be sent to the company owner via E-mail and/or Telegram.

- URL:
  - `/rest/closeinvoice/{invoiceId}`
- Method:
  - PUT
- URL Parameters:
  - `invoiceId = {int}`  
The identifier of the invoice to close.
- Data Parameters:
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content: *message: "Invoice successfully closed and exported!"*
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any error occurs in generating the <i>PDF</i> file
500	If any error occurs in sending the <i>PDF</i> file via E-mail
500	If any database-related error occurs

## Generate an invoice

The following endpoint allows generating the invoice having the identifier passed in the URL, blocking the possibility of updating and modifying it.

When the invoice is successfully generated, two documents in *PDF* and *XML* formats will be available for download and will be sent to the company owner via E-mail and/or Telegram.

- URL:
  - `/rest/generateinvoice/{invoiceId}`
- Method:
  - PUT
- URL Parameters:
  - `invoiceId = {int}`  
The identifier of the invoice to generate.
- Data Parameters:
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content: *message: "Invoice successfully generated and exported!"*
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any error occurs in generating the <i>PDF</i> file
500	If any error occurs in sending the <i>PDF</i> file via E-mail
500	If any database-related error occurs

## List the customers associated with a company

The following endpoint allows to list all the customers associated with the current company logged into the system.

- URL:
  - /rest/list-customer
- Method:
  - GET
- URL Parameters:
  - company = {int}  
The identifier of the company currently logged into the system.
- Data Parameters:
  - ownerId = {int}  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:

```
{ "resource-list":  
  [{ "customer": { "customerID": "1", "businessName": "AppleLike Inc.", "vatNumber": "56479-182", "taxCode": "22342-305", "address": "Via Venezia 79", "city": "Padova", "province": "PD", "postalCode": null, "emailAddress": "applelike@google.com", "pec": "applelike@pec-mac.com", "uniqueCode": "1", "companyID": "1" } },  
  { "customer": { "customerID": "2", "businessName": "Reinger Group", "vatNumber": "49631-182", "taxCode": "14783-305", "address": "4 Mockingbird Junction", "city": "Rovigo", "province": "RO", "postalCode": "45100", "emailAddress": "rfrankum1@google.nl", "pec": "plyster1@pec-mac.com", "uniqueCode": "2", "companyID": "2" } }  
]}
```
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any database-related error occurs

## List the products associated with a company

The following endpoint allows to list all the products associated with the current company logged into the system.

- URL:
  - /rest/list-product
- Method:
  - GET
- URL Parameters:
  - company = {int}  
The identifier of the company currently logged into the system.
- Data Parameters:
  - ownerId = {int}  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:

```
{ "resource-list":  
  [ { "product": { "product_id": 1, "company_id": 1, "title": "ETHANOL", "default_price": 29, "logo": "http://dummyimage.com/222x224.png/5fa2dd/ffffff", "measurement_unit": "Kg", "description": "Pre-emptive upward-trending analyzer" } },  
    { "product": { "product_id": 2, "company_id": 2, "title": "Antimoiium crud, Benzoic ac, Ledum, Nux vom, Quercus, Rhododendron, Silicea", "default_price": 24, "logo": "http://dummyimage.com/225x208.png/5fa2dd/ffffff", "measurement_unit": "Kg", "description": "Ameliorated mission-critical adapter" } }  
  ]  
}
```
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any database-related error occurs

## Retrieve a document

The following endpoint allows the retrieval of the document associated with the invoice having the identifier passed in the *URL*, in the format specified in the *URL*.

The available formats are: *PDF Warning*, *PDF Invoice*, and *XML Invoice*.

- URL:
  - `/rest/getdocument/{type}/invoice/{invoiceId}`
- Method:
  - GET
- URL Parameters:
  - `type = {int}`  
The type of document associated with the invoice to retrieve:
    - \* 0 → *PDF Warning file*,
    - \* 1 → *PDF Invoice file*,
    - \* 2 → *XML Invoice file*.
  - `invoiceId = {int}`  
The identifier of the invoice to retrieve.
- Data Parameters:
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content: *message: "Document successfully retrieved!"*
- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any error occurs in generating the <i>PDF</i> file
500	If any error occurs in sending the <i>PDF</i> file via E-mail
500	If any database-related error occurs

## List the filtered invoices associated with a company

The following endpoint allows to filter and list the invoices associated with the current company logged into the system.

- URL:
  - /rest/filter-invoices
- Method:
  - POST
- URL Parameters:
  - company = {int}  
The identifier of the company currently logged into the system.
- Data Parameters:
  - ownerId = {int}  
The identifier of the owner contained in the authentication token.
  - the user can set one, many, or all the filters defined below:
    - \* fromTotal = {Double} - the lower bound for the *Total*,
    - \* toTotal = {Double} - the upper bound for the *Total*,
    - \* fromDiscount = {Double} - the lower bound for the *Discount*,
    - \* toDiscount = {Double} - the upper bound for the *Discount*,
    - \* fromPfr = {Double} - the lower bound for the *Pension Fund Refund*,
    - \* toPfr = {Double} - the upper bound for the *Pension Fund Refund*,
    - \* fromInvoiceDate = {Date} - the lower bound for the *Invoice Date*,
    - \* toInvoiceDate = {Date} - the upper bound for the *Invoice Date*,
    - \* fromWarningDate = {Date} - the lower bound for the *Warning Date*,
    - \* toWarningDate = {Date} - the upper bound for the *Warning Date*,
    - \* fromBusinessName = {String(1)---String(2)---...---String(n)} - the list of the *Customer Names* to filter the invoices,
    - \* fromProductTitle = {String(1)---String(2)---...---String(n)} - the list of the *Product Titles* to filter the invoices,
- Success Response:
  - Code: 200
  - Content:

```
{ "resource-list":  
  [{ "invoice": { "invoice_id": 1, "customer_id": 1, "status": 0, "warning_number": 1, "warning_date": "2022-01-05", "warning_pdf_file": "warning_pdf_file1.pdf", "invoice_number": "1", "invoice_date": "2022-02-06", "invoice_pdf_file": "invoice_pdf_file1.pdf", "invoice_xml_file": "invoice_xml_file1.xml", "total": 168.3, "discount": 15.0, "pension_fund_refund": 4.1, "has_stamp": false } } ] }
```

```
{
  "invoice": {
    "invoice_id": 20,
    "customer_id": 20,
    "status": 0,
    "warning_number": 20,
    "warning_date": "2022-09-17",
    "warning_pdf_file": "warning_pdf_file20.pdf",
    "invoice_number": "20",
    "invoice_date": "2022-09-20",
    "invoice_pdf_file": "invoice_pdf_file20.pdf",
    "invoice_xml_file": "invoice_xml_file20.xml",
    "total": 71.2,
    "discount": 46.8,
    "pension_fund_refund": 3.2,
    "has_stamp": false
  }
}
```

- Error Response:

Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any error occurs in parsing the filters
500	If any database-related error occurs

## Plot the filtered charts associated with a company

The following endpoint allows to filter and list the invoices associated to the current company logged into the system and plot a chart.

- URL:
  - /rest/charts
- Method:
  - POST
- URL Parameters:
  - company = {int}  
The identifier of the company currently logged into the system.
- Data Parameters:
  - ownerId = {int}  
The identifier of the owner contained in the authentication token.
  - the user can set one, many, or all the filters defined below:
    - \* fromTotal = {Double} - the lower bound for the *Total*,
    - \* toTotal = {Double} - the upper bound for the *Total*,
    - \* fromDiscount = {Double} - the lower bound for the *Discount*,
    - \* toDiscount = {Double} - the upper bound for the *Discount*,
    - \* fromPfr = {Double} - the lower bound for the *Pension Fund Refund*,
    - \* toPfr = {Double} - the upper bound for the *Pension Fund Refund*,
    - \* fromInvoiceDate = {Date} - the lower bound for the *Invoice Date*,
    - \* toInvoiceDate = {Date} - the upper bound for the *Invoice Date*,
    - \* fromWarningDate = {Date} - the lower bound for the *Warning Date*,
    - \* toWarningDate = {Date} - the upper bound for the *Warning Date*,
    - \* fromBusinessName = {String(1)---String(2)---...---String(n)} - the list of the *Customer Names* to filter the invoices,
    - \* fromProductTitle = {String(1)---String(2)---...---String(n)} - the list of the *Product Titles* to filter the invoices,
    - \* chart\_type = {int} - the chart type: 1 for Invoice by date, 2 for Total by date, 3 for Discount by date, 4 for Invoice by customer, 5 for Total by customer,
    - \* chart\_period = {int} - the chart period we want to group dates in the x-axis: 1 for Months, 2 for Years, 3 for Days
- Success Response:
  - Code: 200
  - Content:

```
{ "chart": {  
    "labels": ["Dicembre 2023"],  
    "data": ["1"],  
    "type": 1,  
    "period": 1}}
```
- Error Response:



Code	When
400	If any error occurs in parsing the <i>ownerId</i>
400	If the user is not allowed to access this company
500	If any error occurs in parsing the filters
500	If any database-related error occurs

### Get the company's bank account details

The following endpoint allows to get all the details about the bank account associated with the company currently logged into the system.

- URL: `/rest/bankaccount/{bankaccountId}/company/{companyId}`
- Method: GET
- URL Parameters:
  - `bankaccountId = {int}`  
The identifier of the bank account to retrieve
  - `companyId = {int}`  
The identifier of the company currently logged into the system.
- Data Parameters:
  - `ownerId = {int}`  
The identifier of the owner contained in the authentication token.
- Success Response:
  - Code: 200
  - Content:
 

```
{
  "bankaccount_id":1,"IBAN": "IT25674893201252722926","bank_name":"ISP",
  "bankaccount_friendly_name":"Intesa San Paolo","company_id":1}
```
- Error Response:

Code	When
500	If the required bank account is not found in the database
500	If any database-related error occurs

## 5.6 CURL examples

- For logging in:
 

```
curl --location 'localhost:8080/bitsei-1.0/rest/login/' \
--header 'Content-Type: application/json' \
--data-raw '{
  "email": "your_email",
  "password": "your_password"
}'
```

- For creating companies:

```
curl --location 'localhost:8080/bitsei-1.0/rest/company' \
--header 'Authorization: YOUR TOKEN' \
--form 'title="Youfeed"' \
--form 'business_name="Leannon and Sons"' \
--form 'vat_number="1111111111"' \
--form 'tax_code="59767-008"' \
--form 'address="88 Mandrak Road"' \
--form 'province="VR"' \
--form 'city="Helong"' \
--form 'unique_code="3"' \
--form 'has_mail_notifications="false"' \
--form 'has_telegram_notifications="false"' \
--form 'logo=@"IMAGE PATH"'
```

- For Getting the list of companies:

```
curl --location 'localhost:8080/bitsei-1.0/rest/company' \
--header 'Authorization: YOUR TOKEN'
```

## 6 Group Members Contribution

**Mirco Cazzaro** Initial contribution to the database design; implementation of Customer *CRUD* operations; implementation of the logic behind invoice status (invoice closure and generation); implementation of *JasperReport* library for *PDF* generation; implementation of *XML* electronic invoices that follow Italian regulations using *Dom4j* library; Implementation of *REST URI* parsing utility; Implementation of *JasperReport* export file utility;

**Marco Martinelli** Implementation of database reading section to fetch, as a list, Invoices, Products, and Customers. Implementation of the invoices listing by filters *fromValue*, *toValue*, *fromValue-toValue* using as parameters: total, discount, pension fund refund, invoice date, warning date, customer name, product name. Implementation of notification utils for both emails and telegram, used for sending plain text messages and messages with an attachment.

**Farzad Shami** Initial contribution on the project structure setup; Implementation of the authentication logic part using *JWT* tokens; Implementation of *JWTAuthentication Filter*; Implementation of company management part; Implementation of Reset Password logic; *CI/CD Pipeline* for automatic deployment on cloud server.

**Nicola Boscolo Cegion** Implementation of Bank account *CRUD* operations, implementation of login, implementation of *JWT* tokens.

**Christian Marchiori** Implementation of insight part by generating different typologies of charts (scatter plots, histograms, and pie charts) based on different filters. Implemented initial setup for displaying charts through *JavaScript* and the *Chart.js* library. Drawn detailed mock-ups of almost every page of the application.

**Fabio Zanini** Implementation of the logic behind *Invoices*, *Products* and *Invoice\_Products* *CRUD* operations in a restful way; Implementation of relative resources classes (*Product*, *Invoice*, *Invoice\_Product*). Creation of *SQL* inserts for toy data for the database.

**Andrea Costa** Implementation of *JSP* pages for products.