UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

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

Homework 1 – Server-side Design and Development

Submission date: 23 April 2021

Last Name	First Name	Badge Number
Deronjic	Denis	1231829
Ivancich	Stefano	1227846
Maino	Nicola	1239112
Moroldo	Luca	1234011
Pham	Francesco	1234004
Vaccaro	Fabio	1231830

Objectives

Nowadays, the number of internet-connected devices is estimated to be around 25 billion and by 2030, this figure is expected to jump to 125 billion. A growing number of IoT devices are being deployed to enable the collection of a vast amount of digital data.
We impersonate a company selling different IoT sensors. The main purpose of our web application is to provide an e-commerce platform to accommodate the ordering of our devices, and an online platform to collect the measurements from the sensors and visualize a clear illustration of the data in a dashboard.

Main functionalities

The core functionality of our web application is the IoT device monitoring through a dashboard. The dashboard is divided into several pages where the user can interact and monitor the owned devices. The sensors collect different kinds of data at a regular time interval: one may measure the wind speed and wind bearing, another sensor measures temperatures, humidity and pressure. Our design and development choices also consider that more sensors with different types of information provided may be added in the future.

In order to let the user monitor its sensors, he must own those devices. So, we also provide an ecommerce page where the user can order as many sensors as he wants. When the user completes a purchase, the sensors he bought are automatically added to the dashboard.

The user firstly needs to be registered on our website. He can do that immediately when he opens the site through the login/signup page, or he can register during the purchase phase. User's email address is used for the login.

In order to be maintainable in the long-term, the dashboard charges the users counting the total number of monthly API calls from his sensors. We provide a free plan of 1000 calls/month and a pro plan is charged with a subscription model where the number of API calls that he can make are up to 10k calls/month.

When the user receives the sensor, he has to add a token generated by our dashboard and insert it into the sensor. This allows the authentication of the sensor which is sending the data. Our platform for a security reason, updates automatically the token of the sensors. This helps to avoid a malicious user to use a stolen token to perform illegitimate actions.

Each sensor can be added to a group whose name is selected by the user. The groups help to better organize the sensors since more than one sensor can be added to a group.

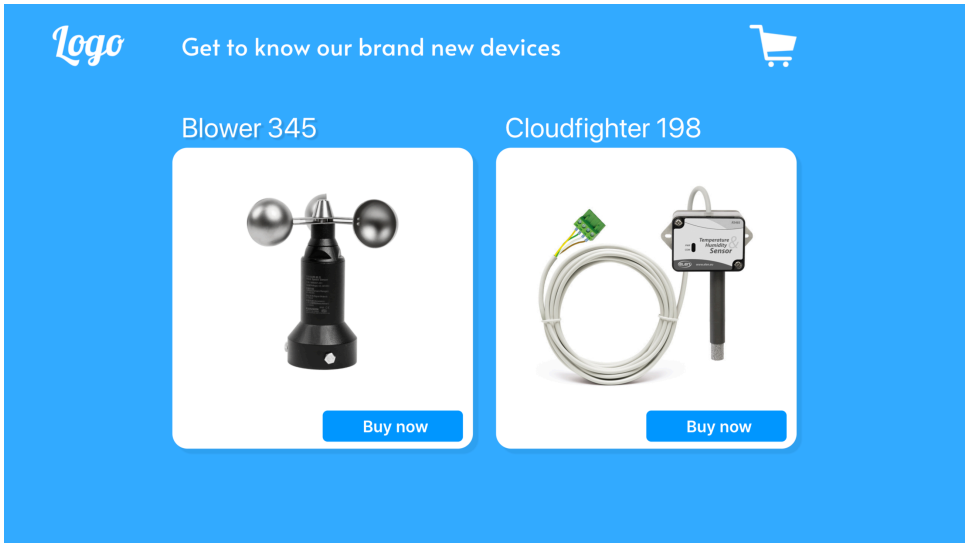
All the data taken from the devices are visualized in a dedicated page. Sensor's internal information is visualized at a glance in the dedicated sensors page which helps the user to monitor the battery status and other internal parameters. The state of a device is critical in an IoT application, so our platform alerts the user whether a certain amount of battery level or a data stream are under a designated threshold that the user could define.

Presentation Logic Layer

The project is thought to be divided into 2 sub-sites:

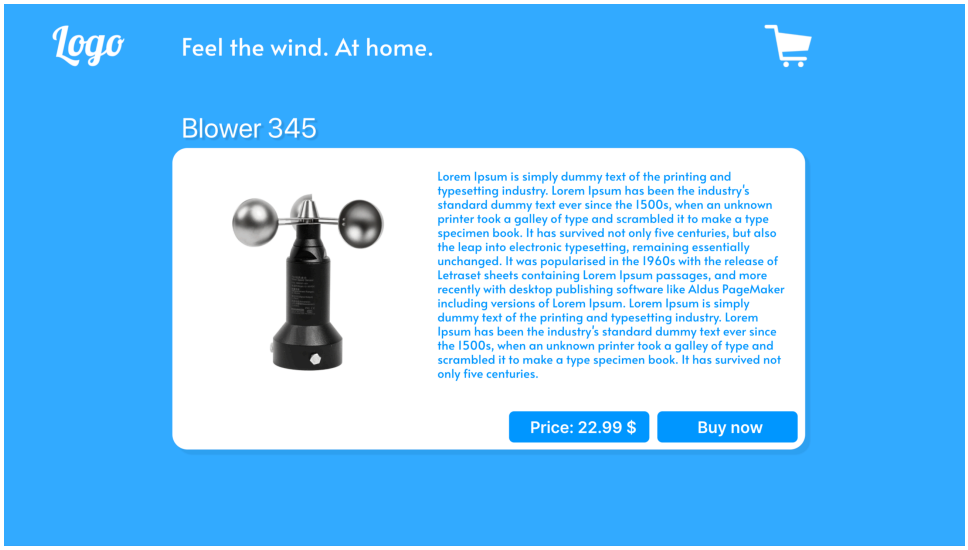
- Shop: where the customer has the possibility to order new devices. In opposition to the dashboard part, the shop has a more traditional design, as it should be more attractive. It is subdivided in the following pages:
    - Homepage: contains a brief description of the devices that are sold
    - Product page: contains a full description of the device with all the technical details
    - Shopping cart: contains all the products already put in the cart and the order details
    - Sign up/Sign in: page to allow the user to sign up or to sign in. It is common to the Dashboard section of the web app.
    - Orders list: contains the list of orders made by the customer
    - Order page: contains the list of products included in the order
    - Customer profile: allows the user to view and modify the setting of hers/his account
- Dashboard: It has a design similar to a control panel and it exploits the whole size of the screen to allow the user to see more data. All the pages of the dashboard part have a sidebar in which there is a menu to allow the user to surf the site. On the bottom a section with the customer's name, link to her/his profile page, the account type and the number of calls already done by the customer's devices. It is subdivided in the following pages:
    - Homepage: page where the user can inspect its own devices pieces of information on
    - Map: page where the user can visualize where its devices have been deployed
    - Groups list: page on which all the groups created by the current user are displayed.
    - Device: page where the details relevant to the single device are displayed.
    - Device configuration: page on which the device configuration can be customized.

Shop - Home Page (Interface Mockup)



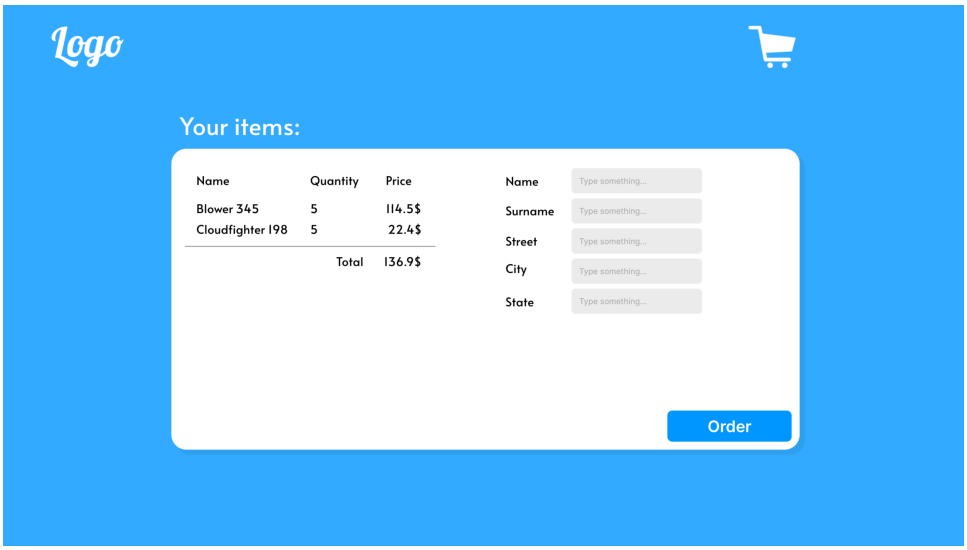
The shop homepage is split into two parts, each one corresponding to a device. Each part contains a brief description of the device to be sold, together with its picture and name. By clicking on the product image, it is possible to access the products' page where more details such as the specifications are listed. A link to the shopping cart is instead provided by clicking on the "buy now" button: this is to directly order one device. On the top right corner of the page, a shopping cart button allows the user to go straight to the shopping cart page where the order can be sent.
This should be considered the landing page for the products purchase so it is important to be appealing and well designed.

Shop - Product page (Interface Mockup)



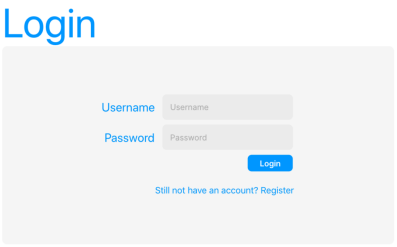
This page contains the product's description with all the details such as the device technical specifications and the price per single device. A "buy now" button redirects to the shopping cart page where it is possible to finalize the order process.

Shop - Shopping cart (Interface Mockup)



This page shows an overview of the products already added to the shopping cart together with their relative quantity and prices. The total price is also provided on the bottom. On the right, all the shipping details are listed to full-fill the order. The order button is present to finalize the order. This page is accessible from anywhere by clicking on the provided shopping cart icon. In particular, from through the redirection from the product page or the shop homepage thanks to the buy button. Once the order has been made successfully a pop-up notifies the user the order has successfully been submitted.

Sign up/Sign up (Interface Mockup)



The page allows the user to login into the system. The page is in common with both the shop and the dashboard. If the customer is not registered yet, she/he can reach the sign-up page from a link below the login form.

Shop - Orders list (Interface Mockup)

This page contains the list of all the orders made by the customer. Each single order has a link to the order page to have further details on the order itself and a list of products contained in the order. The partial and total price is also shown.

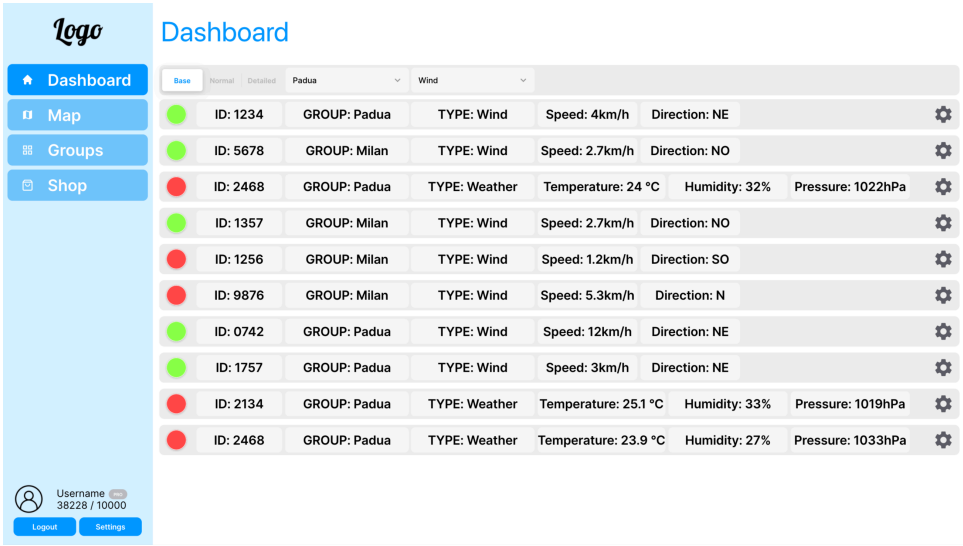
Shop - Order page (Interface Mockup)

This page contains the list of all the products included in the order, the quantities for each product, the partial price and the total count. The destination address is also included in the page.

Shop - Customer profile (Interface Mockup)

This page contains the profile information of the customer, including the account type (free, pro) and the number of API calls made by the customer's devices (the free plan has a monthly limit of calls). This page also allows to edit the settings, like the account type and to modify the password and the username.

Dashboard - Homepage (Interface Mockup)



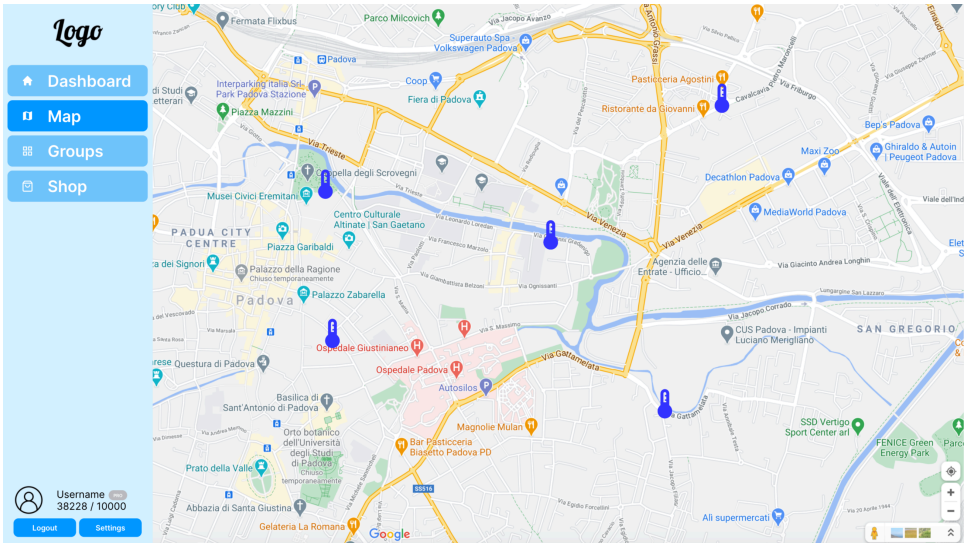
The dashboard is the first page that appears to the user after the log-in and it consists of a list containing the main pieces of information regarding the devices the user has already purchased and deployed.
By looking deeper, the list looks like it is split into cards, each one containing details covering a specific device. The details to be exposed depend also on the type of the device: a wind sensor will show data concerning the wind only, a weather one will show atmospheric statistics instead. On the top of the list a special card is shown, allowing the user to set multiple ways of visualizing the device details; the base feature allows the user to look the main pieces of information only; the normal and advanced ones give the user the possibility to go deeper and see many more attributes. This customization is useful in case a huge number of devices has been purchased and it lets the user decide whether a compact way of listing the device is more suitable to him/her. In addition, the user can select an even more clever way of listing its items by using the filters Groups

and Type. The groups feature allows to show a list of devices belonging to a specific group the user has previously defined, while the Type filter permits to visualize all the devices of a certain type (eg: list all the wind sensors instead of the weather ones).
Getting back to the devices card, depending both on the amount of details the user likes to visualize and the sensor type, the following parameters might be shown:

- A green or red semaphore: it allows to immediately see whether a device is enabled or not
- Group: the group the device belongs to
- Type: the type of the device
- Speed: the last wind velocity the sensor has registered
- Direction: the last wind direction the wind is blowing to the sensor has registered
- Temperature: the last temperature the sensor has registered.
- Humidity: the last humidity value the sensor has registered
- Pressure: the atmospheric pressure the sensor has registered.

Moreover, a gear wheel icon is shown for each device and by clicking on it the user can access the page from where device options can be changed.

Dashboard - Map (Interface Mockup)

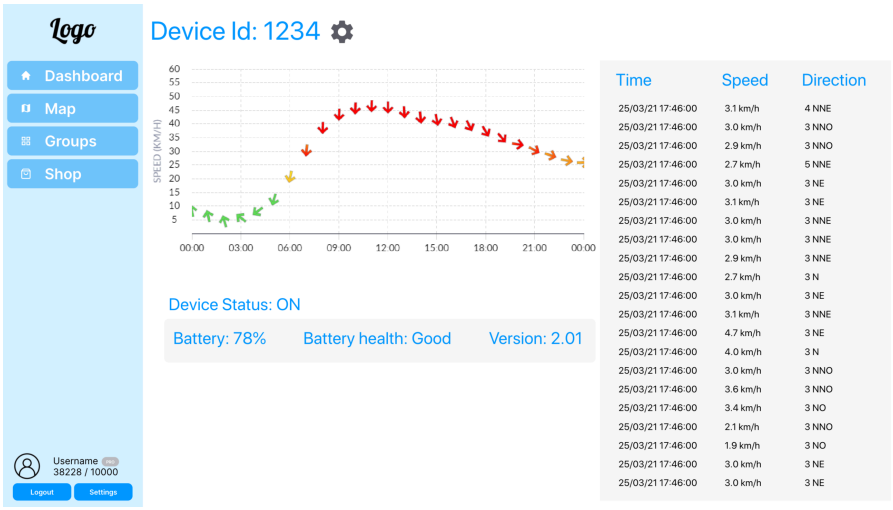


The map page contains a map on which the user can locate where its own devices have been deployed. The location of a device is based on the geographical coordinates the user has set in the configuration stage.
Each device is displayed on the map by an icon it can have different color depending on the current device status. This helps the customer to get an immediate look for devices which battery is low, or the ones which are deactivated. It is also possible to display only devices from a group.

Dashboard - Groups list (Interface Mockup)

The groups list is the page on which all the groups created by the current user are displayed.

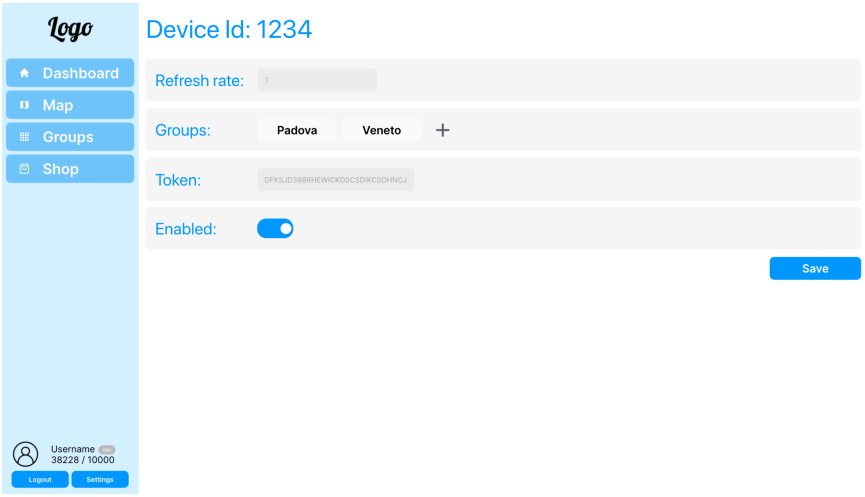
Dashboard - Device (Interface Mockup)



It is the page where the details relevant to the single device are displayed. The device to be shown is selected from the dashboard page and it can be easily noticed by its ID showed on the top; a gear wheel is also present, and it is a direct link to the device configuration page where the device parameters can be customized.
The page is dived into three main sections each one containing features:

- A chart showing the trend of the data collected by the device over time. Depending whether the sensor type is a wind or a weather one, the kind of information displayed is different
- A dataset showing all the records the sensor has registered over time. It looks like a table where the physical dimensions measured are listed on top and below it the actual records are shown
- A device status section to visualize the most important data regarding the current devise status, in particular its battery level and health and the version of the firmware the devise is running.

Dashboard - Device configuration (Interface Mockup)

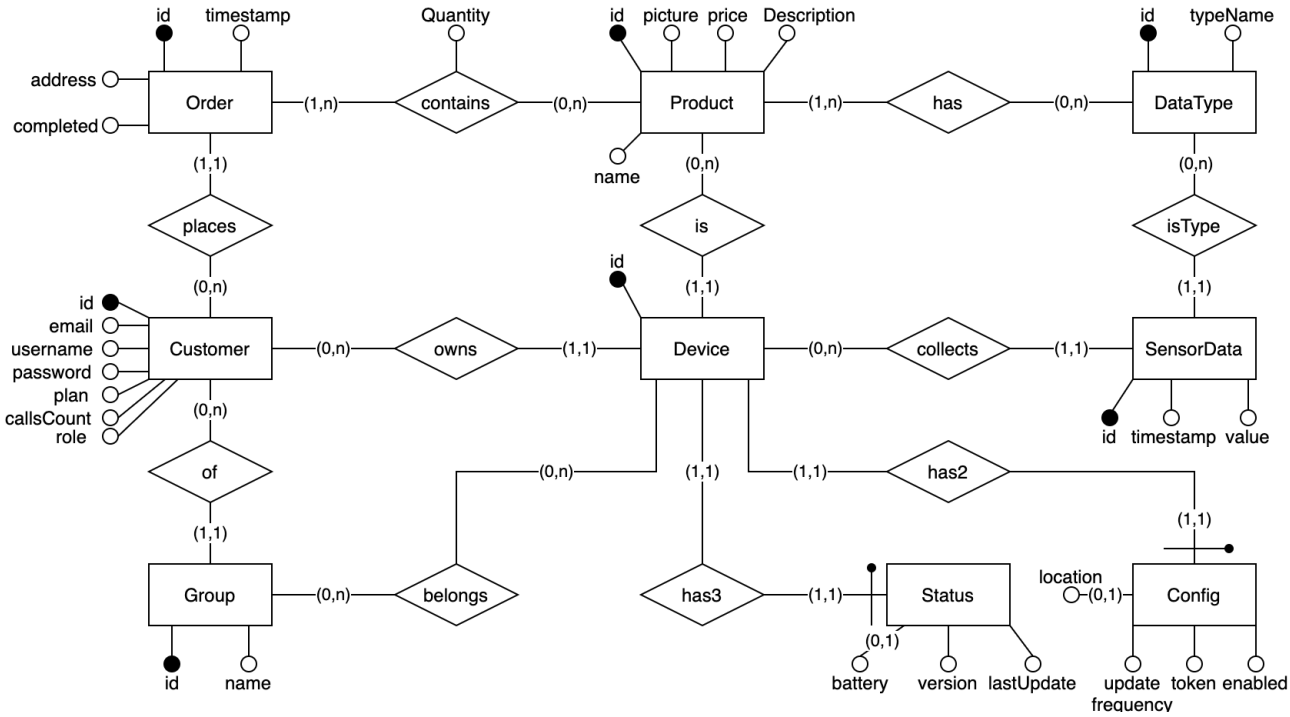


This is the page on which the device configuration can be customized. The overall design of the page consists of, as the dashboard does, cards containing the parameters the user intends to modify. The list of these features is presented below:

- Refresh rate: by changing this option the customer can adjust the frequency the device collects data and send them to the server. By lowering this value, the user can directly update the number of calls the sensor does as instance with the purpose of being below the premium threshold. On the other side a user interested in having a fine data collection might want to raise this parameter.
- Groups: this card lists all the groups the device belongs to. And add icon is present and it allow the user to add a new group for the current device.
- Token: it shows the token the device has. There is the possibility of generating a new token.
- Enabled: gives the user the possibility of deactivating the current device. Again, this might be useful to save on the calls number

Data Logic Layer

Entity-Relationship Schema

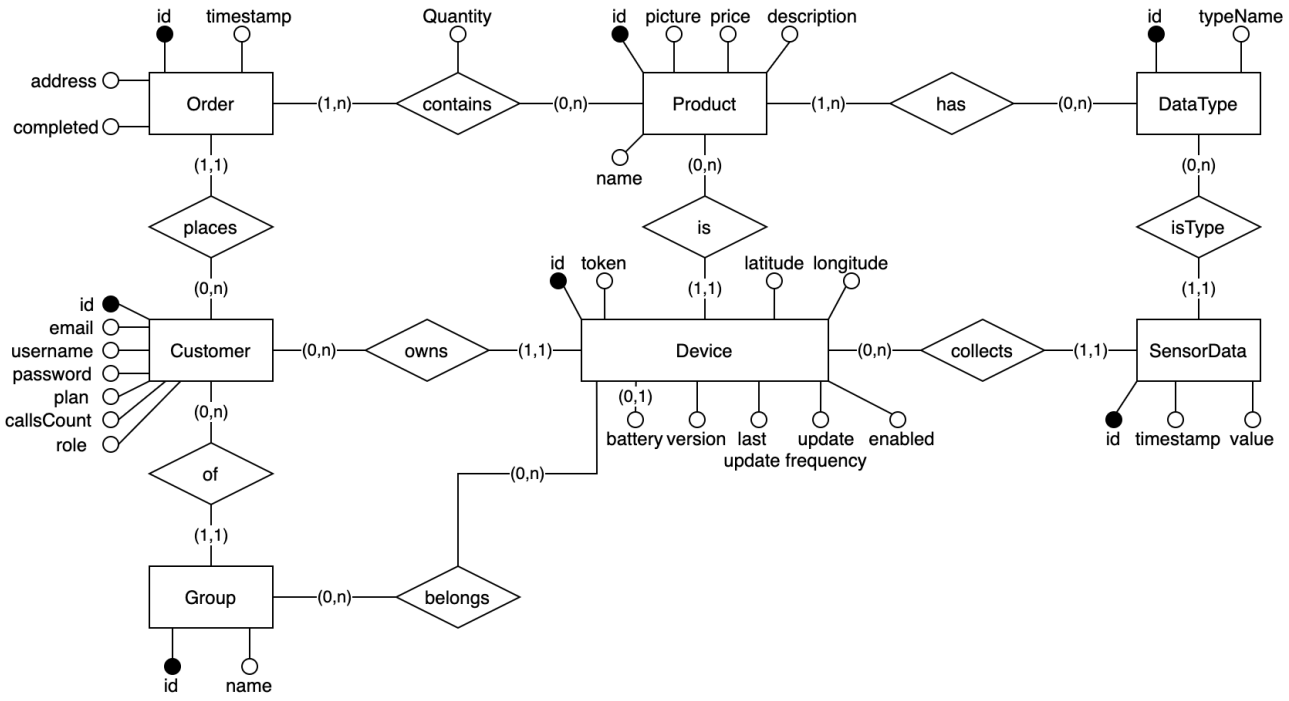


The entity-relationship schema contains 9 main entities:

- Customer: each customer has as primary key an id. Both email and username must be unique. We store the user's password hashed using the BCrypt strong hashing function. Customers with Role=admin are added manually, they can perform a range of special actions on the website, like adding and updating new products, datatypes, and so on. Each customer has a plan that can be either FREE or PREMIUM. We also store a counter of API calls which is incremented each time a device owned by the customer sends some data, this allows us to check if the user has exceeded its plan limit.
- Order: a customer can place zero ore more orders, which can be completed or not. For each customer there can only exists at most one non-completed order (the cart). Each order has as primary key an id, a shipping address and a timestamp as attributes. The customer can place one or more products in the order in a specified quantity. When the customer completes the payment, the non-completed order is transformed into completed and for each ordered product, the specified quantity of devices is created.
- Product: a product is a type of device that a customer can buy. It's identified by an id, its attributes consist of a name, a picture, a description and a price. Each product can register different kind of data (determining a relation N-N with DataType)
- Device: Its primary key is an integer incremented automatically upon insertion. Each device is an instance of a product (determining a relation N-1 with Product) and is owned by a customer (determining a relation N-1 with Customer). The customer who owns the device and its product are therefore memorized as external keys directly on the device entity instances.
- Status: weak entity which represents the status of a device. It contains the battery and version of the devices and the timestamp of their last collected data.
- Config: weak entity which represents the configuration of a device. It contains frequency of data collection, its token, its location and a Boolean specifying whether the device is enabled.

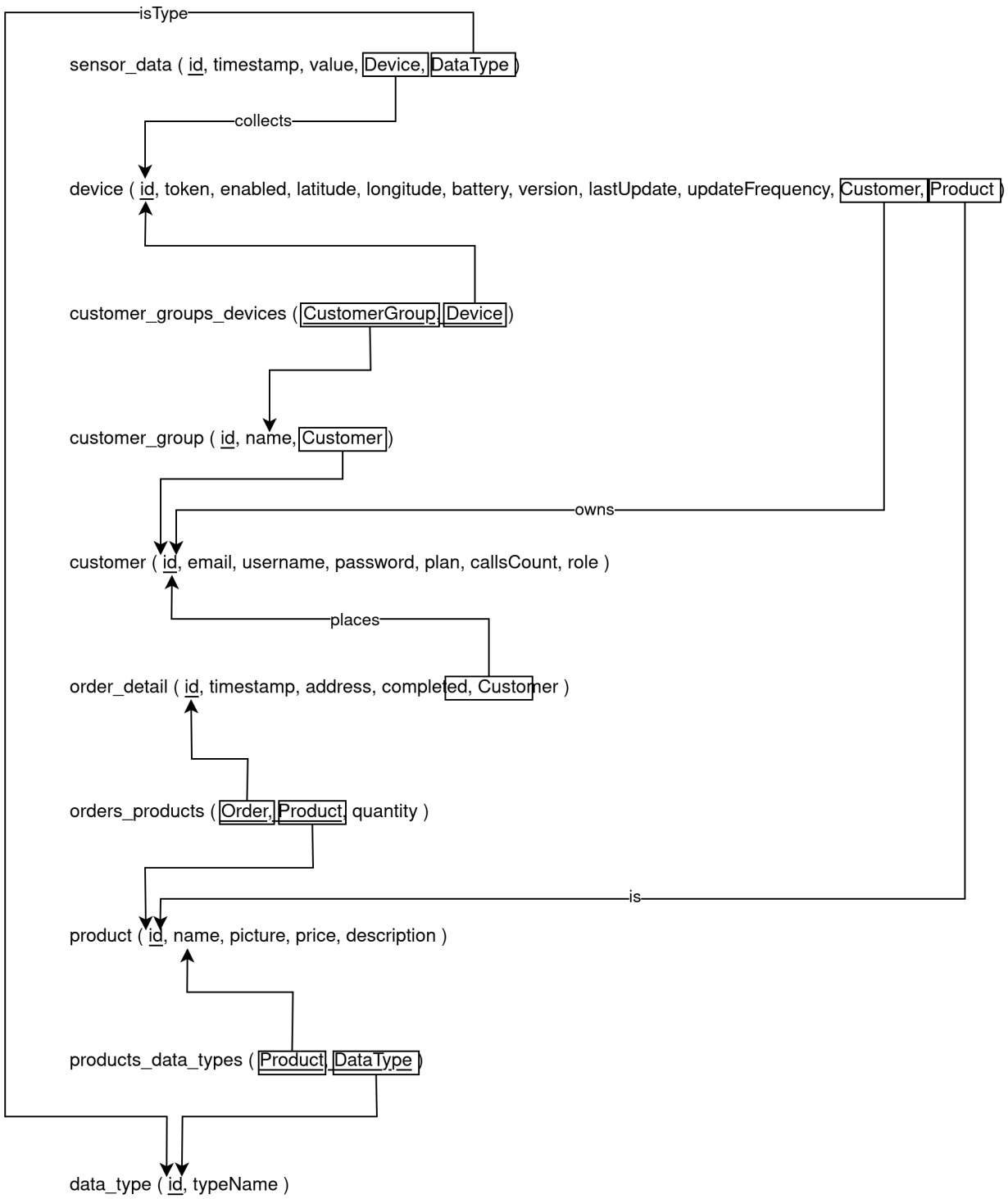
- SensorData: it is identified by an id. It represents a single measure taken by one of the sensors present in a device. It is collected by a specific device in a precise timestamp, and has one single float value, that can be interpreted by the relationship with DataType.
- DataType: a type of data that a device can register (e.g., temperature, humidity, wind speed…). It is identified by an id and has its name as an attribute. This design was chosen considering that more sensors providing different types of information may be added in the future. Moreover, we can have products that collect multiple datatypes, and some datatypes could be shared between products.
- Group: A customer can group its devices into groups, a group is identified by and id and has a name, a group owned and accessed by just one customer, it cannot be shared between different customers, a group can contain zero or more devices of its customer.

Entity-Relationship Schema Restructured



As you can see, the main difference of the Restructured ER schema is that we merged the entities Status and Config with the corresponding attributes into Device, since they have a one-to-one relationship with the entity Device.

Relational Schema



Other Information

The two entities that can exist without being connected to the others are: DataType and Customer (this can be seen better in the relational schema).
We distinguish the cart with a normal order by the label “completed”, there must exist at most one order not-completed per user.

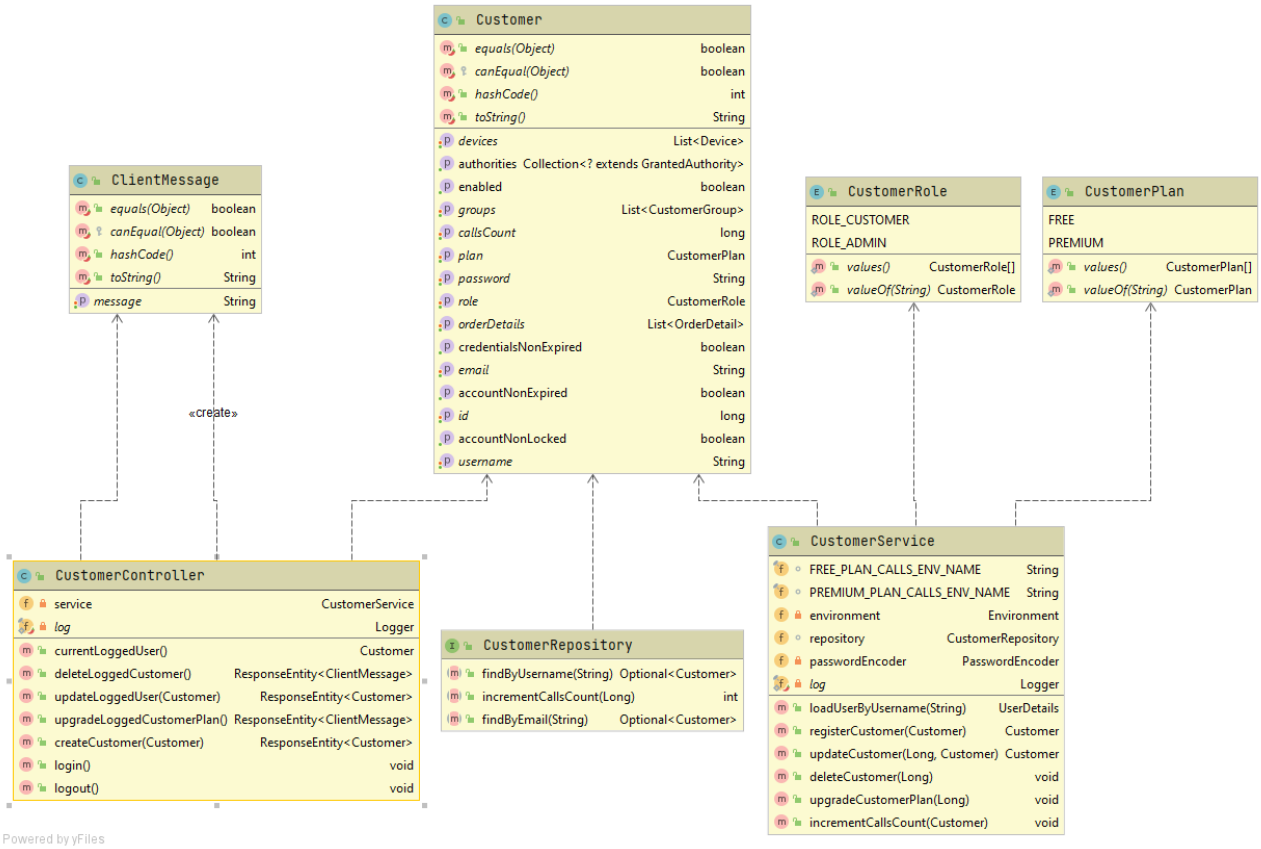
Business Logic Layer

We used spring boot, which uses the MVC pattern.

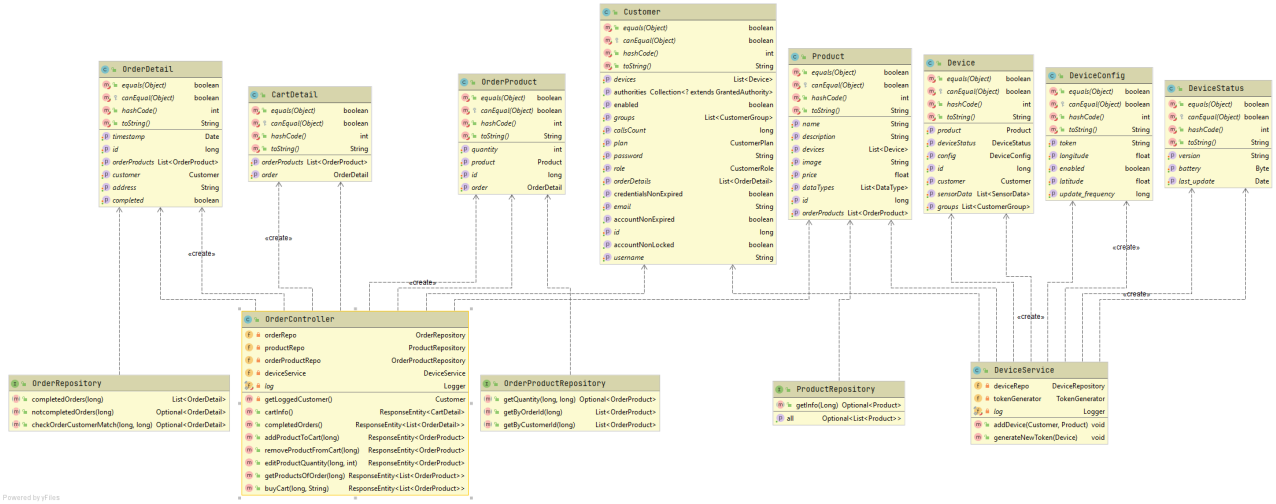
Class Diagram

We have split the class diagram on the basis of the controller's logic, otherwise the diagram would be too big. The project consists of **6 controllers**, one for each entity except for DataType: *CustomerController, OrderController, ProductController, DeviceController, SensorDataController, GroupController*. There is one repository interface for each entity that contains the SQL queries regarding the specific entity.
In the controller classes we have the declaration of the REST API endpoints. The controllers use models, repositories and services to elaborate the REST requests.

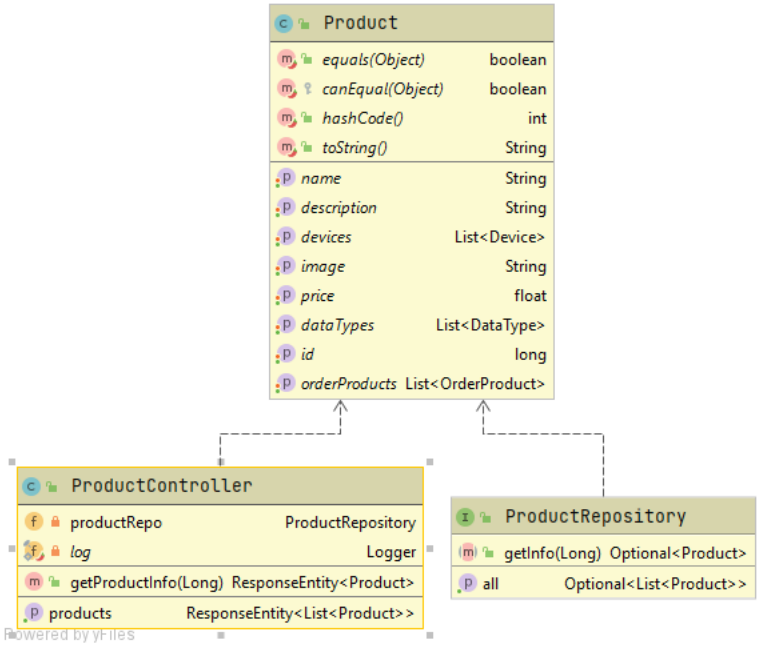
Customer



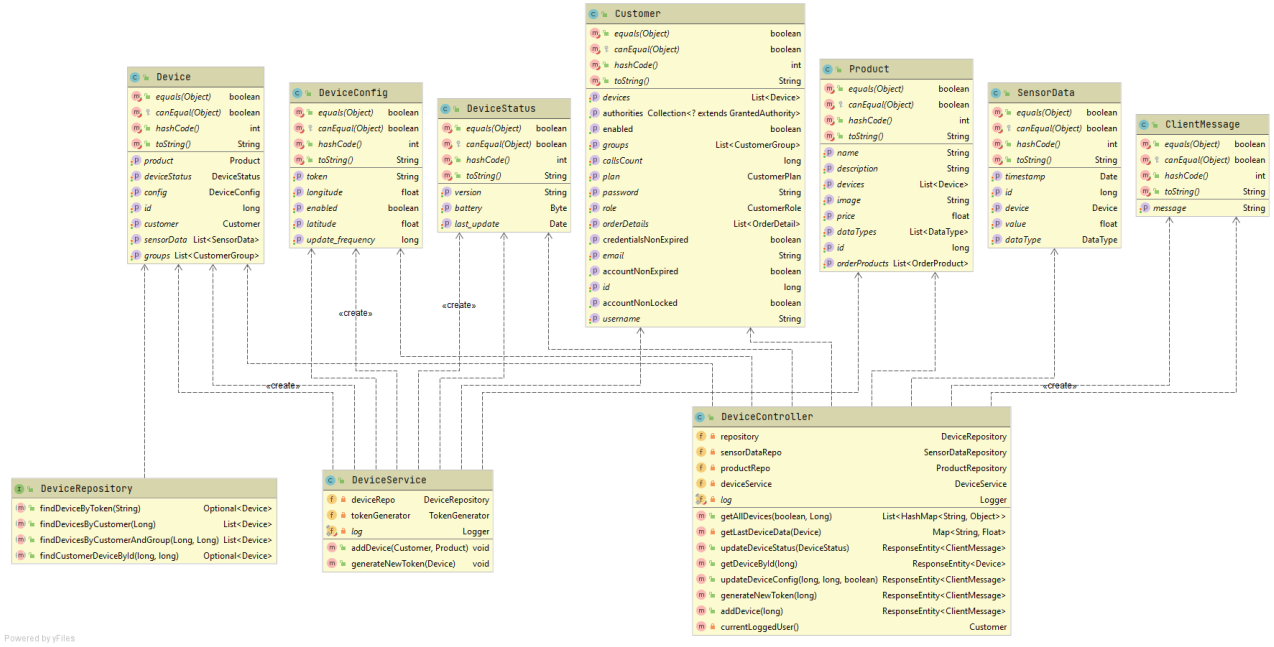
Order



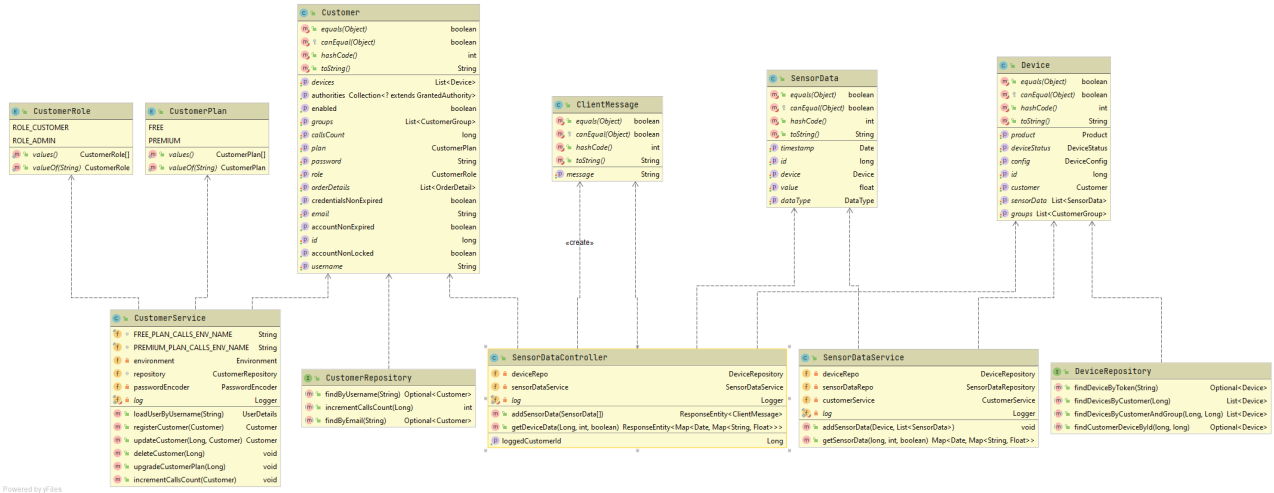
Product



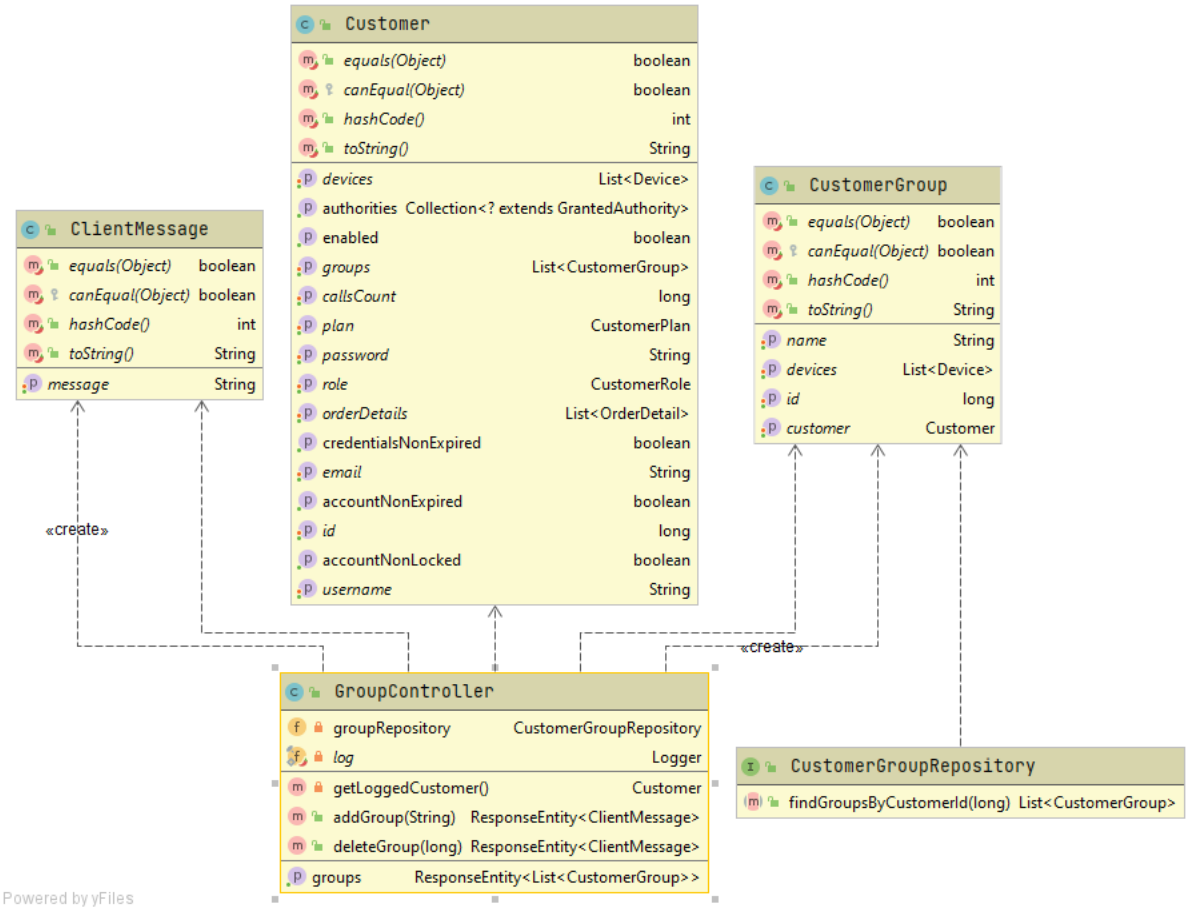
Device



SensorData



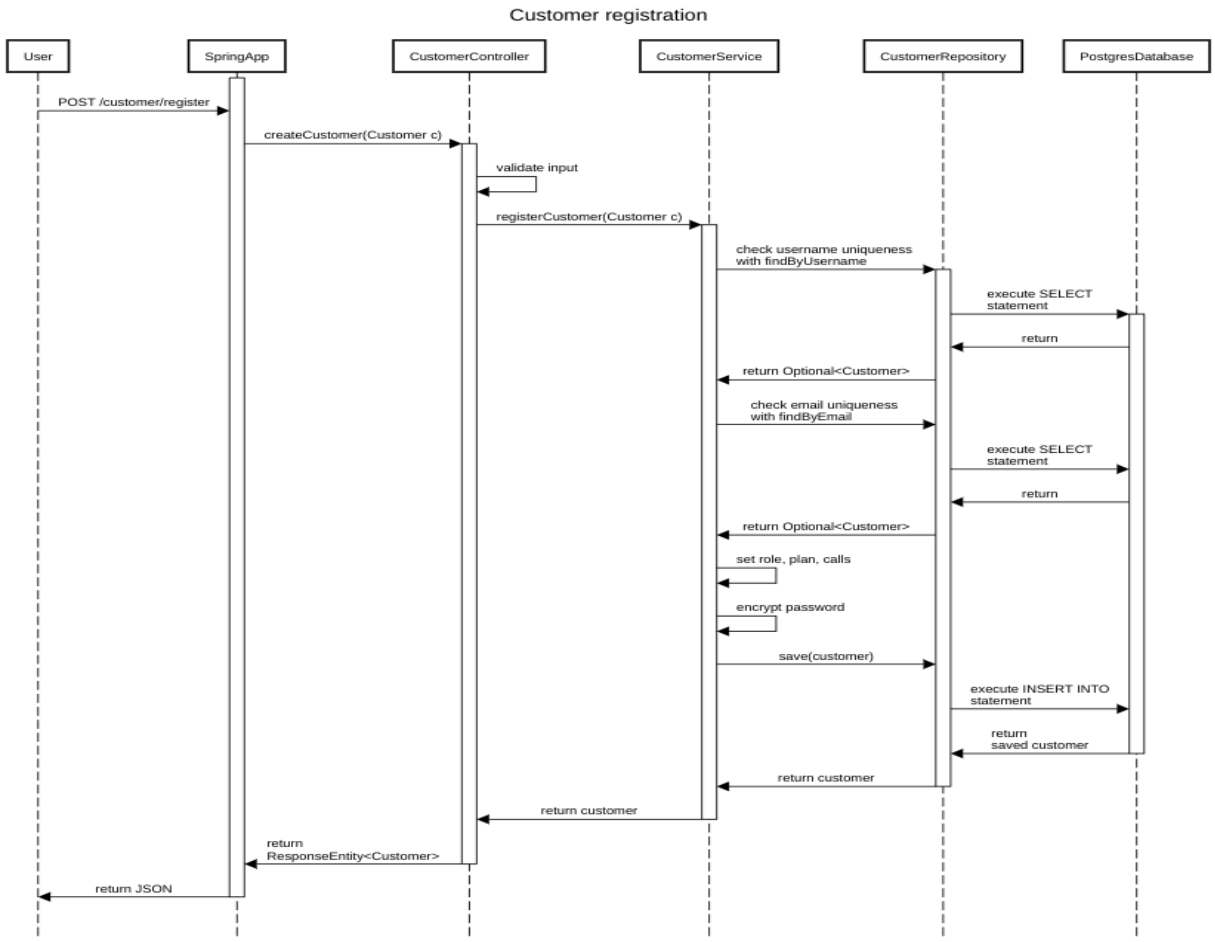
Group


Customer
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
devices List<Device>
authorities Collection<? extends GrantedAuthority>
enabled boolean
groups List<CustomerGroup>
callsCount long
plan CustomerPlan
password String
role CustomerRole
orderDetails List<OrderDetail>
credentialsNonExpired boolean
email String
accountNonExpired boolean
id long
accountNonLocked boolean
username String
ClientMessage
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
message String
CustomerGroup
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
name String
devices List<Device>
id long
customer Customer
«create»
«create»
GroupController
groupRepository CustomerGroupRepository
log Logger
getLoggedCustomer() Customer
addGroup(String) ResponseEntity<ClientMessage>
deleteGroup(long) ResponseEntity<ClientMessage>
groups ResponseEntity<List<CustomerGroup>>
CustomerGroupRepository
findGroupsByCustomerId(long) List<CustomerGroup>
Powered by yFiles

Sequence Diagram

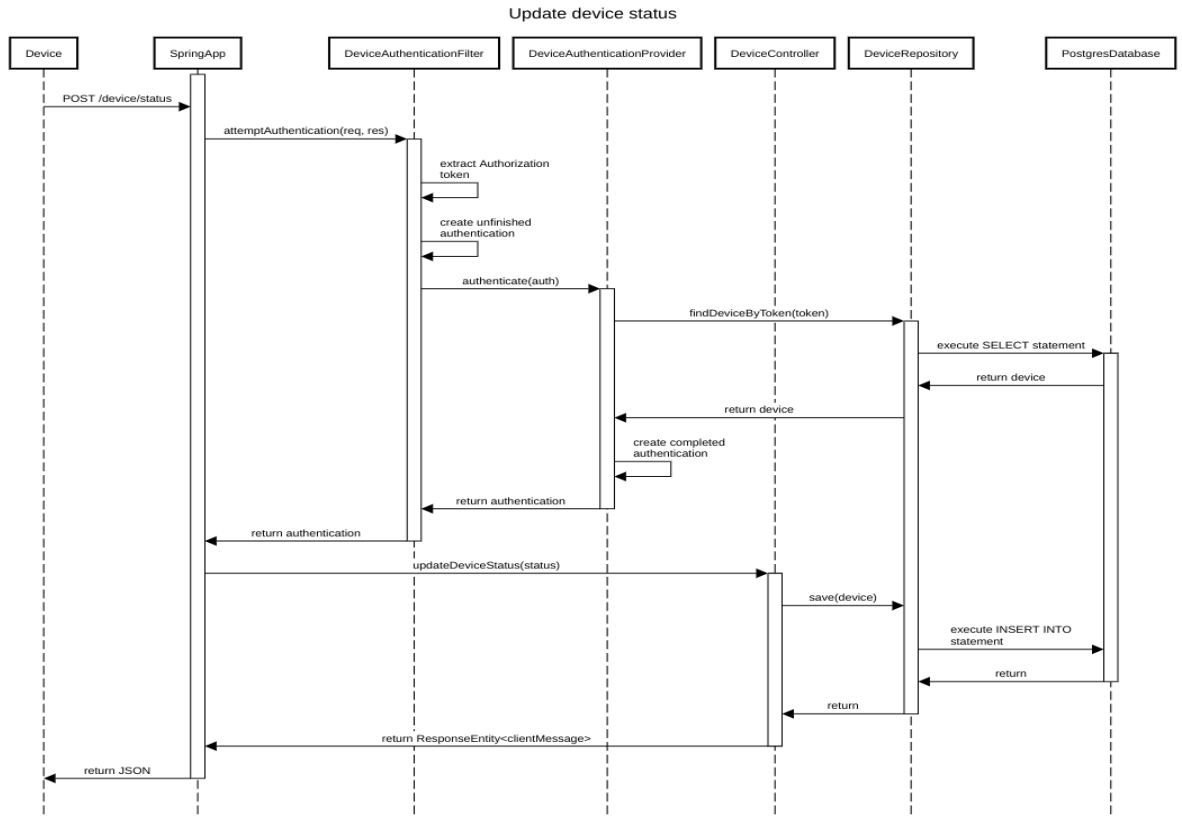
The following two examples are representative of how requests are handled for public endpoints and for endpoints requiring authentication and authorization.

- Customer registration example:



When a client performs a POST request to the "/customer/register" URI, the Spring Application receives the request and Spring Security checks if the URI is protected.
The registration operation is public and doesn't require any authentication, therefore the request is forwarded to the controller method associated with the URI "/customer/register". Spring also initializes the customer object which is parsed from the request body.
CustomerController validates the input (e.g., missing fields from customer) and calls CustomerService with a valid customer instance.
CustomerService, using CustomerRepository, checks that there are no customers with the given username and email. If so, it sets the default values of role, plan and number of calls (which are "ROLE_CUSTOMER", "FREE", and 0).
The customer is stored in the database by CustomerRepository.
The saved entity is returned up to the CustomerController, which sets the customer as the body of the ResponseEntity and the OK status code.
Finally, the Spring Application in collaboration with the Jackson library returns the JSON representation of the customer with the given HTTP status code.

- Device post data example:



When a client executes a POST request to the "/device/status" URI, the Spring Application receives the request and Spring Security checks if the URI is protected.
As the access is restricted to devices and associated with the DeviceAuthenticationFilter, the request is forwarded to this component.
The DeviceAuthenticationFilter extracts the device token from the Authorization header and delegates the authentication to the default Spring Security AuthenticationManager, which calls DeviceAuthenticationProvider (the only provider available to resolve this authentication) with the token extracted from the headers. Note that at this point the device is not yet authenticated.
DeviceAuthenticationProvider uses the token to find the associated device and creates a valid device authentication object with the role "ROLE_DEVICE".

Note that the authentication fails if the request doesn't contain the Authorization header or if the given token is not associated with a valid device. If the authentication fails, the request doesn't reach the controller.

After obtaining a valid authentication object, Spring forwards the request to the controller with the device status parsed from the request's body. Spring Security also checks that the authority of the authentication object is allowed to access the "/device/status" URI, which is permitted to authentication objects with role "ROLE_DEVICE" (I.e, an authenticated customer with role "ROLE_CUSTOMER" couldn't access this endpoint).
At this point, the request flow is similar to the previous example: DeviceController uses DeviceRepository to save the device with the updated status and returns a ResponseEntity object containing a client message, that is converted to JSON by the Spring Framework.

Other Information

Customer authentication is similar to device authentication: a filter extracts username and password from the request and delegates the authentication to the default Spring Security AuthenticationManager, which uses the function loadUserByUsername of CustomerService to retrieve a UserDetail object (I.e., a customer) and checks the given credentials.

The interactions between repositories and the PostgreSQL database are managed by the Spring framework that under the hood uses the javax.persistence package.

REST API Summary

Here we show the list of all REST API endpoints. For each endpoint, the first path segment of the URI tells the backend controller responsible for managing the request. The REST method is mapped to the corresponding CRUD operation in the backend.
D indicates that only devices can access to the endpoint, U indicates that only users (I.e., customers) can access the endpoint, A that only administrators can request the specified URI to the server.

URI	Method	Description	Role
/customer/login	POST	Performs a login operation, creating a session	U
/customer/logout	POST	Invalidates the session	U
/customer/register	POST	Creates a new customer	U
/customer/me	GET	Retrieves currently logged customer	U
/customer/me	PUT	Updates currently logged customer	U
/customer/me	DELETE	Deletes currently logged customer (including any related data)	U
/customer/me/upgrade	POST	Upgrades currently logged customer plan to PREMIUM	U
/device/sensordata	POST	Adds sensor data sent by the device	D
/devices/{id}/data	GET	Returns the sensor data of a particular device	U
/devices	GET	Retrieves all devices of a customer, optionally including last data and filtered by group	U
/device/status	POST	Update device status sent by device	D
/devices/{id}	GET	Get a single device with status and config	U
/devices/{id}/config	PUT	Update a device's configuration	U
/devices/{id}/generatetoken	PUT	Generate a new token for a device	U
/devices	POST	Add a new device	A
/groups	GET	Returns the list of groups owned by user	U

/groups/{name}	POST	Adds a new group with specified name	U
/groups/{id}/delete	DELETE	Deletes the group specified with the id	U
/products	GET	Retrieves a list of all the available products	U
/products/{id}	GET	Retrieves all the pieces of information concerning the device the id is given	U

REST API Error Codes

In the following table we list all the errors defined in the application. We named the error code following the pattern E + <controller acronym> + <progressive number>

Error Code	HTTP Status Code	Description
ESDA1	403 Forbidden	Device is disabled
ESDA2	404 Not Found	Device of current user not found
EDEV1	404 Not Found	Not found any device with specified id owned by logged customer
EDEV2	404 Not Found	Invalid product id
INTERNAL	500 Internal server Error	An internal error occurred
ECUS1	404 Not Found	Customer not found
ECUS2	409 Conflict	Customer with the given username already exists
ECUS3	409 Conflict	Customer with the given email already exists
ECUS4	403 Forbidden	Customer exceeded the available number of device API calls
ECUS5	400 Bad Request	Customer input doesn't contain email
ECUS6	400 Bad Request	Customer input doesn't contain username
ECUS7	400 Bad Request	Customer input doesn't contain password

ELOG1	401 Unauthorized	Invalid login operation (missing parameters or invalid credentials)
EAUT1	401 Unauthorized	Invalid device authentication (missing or invalid token)
EAUT2	401 Unauthorized	Invalid customer authentication (missing credentials)
EAUT3	401 Unauthorized	Access denied due to lack of permissions
ECGR1	404 Not Found	The Group with specified id not found or not belonging to the current customer
EPRD1	404 Not Found	The product specified does not exist
EPRD2	404 Not Found	The product specified does not belong in the cart
EPRD3	400 Bad Request	A wrong quantity specified for the selected product in the cart
EORD1	404 Not Found	A non-completed order is not found
EORD2	404 Not Found	The specified order does not exist (nor completed nor non-completed)
EORD3	404 Not Found	Customer does not own the specified order
EORD4	400 Bad Request	Customer input doesn’t contain order’s address
EORD5	400 Bad Request	Customer is trying to buy an already completed order
EORD6	400 Bad Request	Customer is trying to buy from an empty cart

REST API Details

We report here 3 different resource types that our web application handles during its functioning.

Add Sensor Data

It receives the data from the device and adds them into the device. Only devices with a valid token and enabled are allowed. In this call the number of used calls is also checked to ensure the number of used calls are not over the limit of the current user account type.

- URL: /device/sensordata
- Method: POST
- URL Params: No url params required
- Headers: Authorization: Bearer <token>
- Data Params:
  Required:
  - JSON containing the sensor data
    Content:

```
[
	{
		"value": 2.5,
		"dataType": {
			"id": 1
		}
	},
	{
		"value": 5.5,
		"dataType": {
			"id": 3
		}
	}
]
```

- Success Response:
  Code: 200
  Content:

```
{
	"message": "added {number} measurements"
}
```

- Error Response:
  Code: 403 Forbidden
  Content:

```
{
	"timestamp": "2021-04-15 03:25:18",
	"status": 403,
	"reason": "Forbidden",
	"description": "Device is disabled",
	"errorCode": "ESDA1"
}
```

  When: The device is not enabled

  Code: 403 Forbidden
  Content:

```
{
	"timestamp": "2021-04-15 03:28:37",
	"status": 403,
	"reason": "Forbidden",
	"description": "No more calls available for customer
username1",
	"errorCode": "ECUS4"
}
```

  When: The customer has used more calls than his account allows

Get devices

Retrieves all devices of a customer, optionally including last data and filtered by group

- **URL:** /devices
- **Method:** GET
- **URL Params:**
  Optional:
  - groupId = {integer}
    Filter devices by group id. Retrieves all user's devices if this is not specified.
  - includeLastData = {boolean}
    If this is true, last sensor data is included in the response. False if not specified.
- **Data Params:** None
- **Success Response:**
  Code: 200
  Content: List of devices with status and config and optionally last data.

  ```
  [{"data":{"temperature":34.8},"device":{"id":2,"deviceStatus":{…},"
  config":{…}}},
  {"data":{"temperature":44.8},"device":{"id":3,"deviceStatus":{…},"c
  onfig":{…}}}]
  ```

- **Error Response**:
  Code: 401 Unauthorized
  Content:
  ```
  {
        "status":401,
        "reason":"Bad request",
        "description":
        "Invalid customer authentication",
        "errorCode":"EAUT2"
  }
  ```

  When: The user is not authenticated

Update device status

Update device status with new battery and version sent by device

- **URL:** /device/status
- **Method:** POST
- **Data Params:**
  Required:
  JSON containing new device status information. e.g.
  ```
  {
        "battery": 70,
        "version": "1.2.3"
  }
  ```
- **Success Response:**
  Code: 200
  Content:
  ```
        {
              "message": "Device status updated"
        }
  ```
- **Error Response:**
  Code: 401 Unauthorized
  Content:

```
{
      "status": 401,
      "reason": "Bad request",
      "description": "Invalid device authentication",
      "errorCode": "EAUT1"
}
```

When: the device is not authenticated with the correct token