UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

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	WASCOOT	
Last Name	**First Name**	**Badge Number**
Crivellari	Alberto	2061934
Gharehzad	Shiva	2041385
Huang	Borwoei	2044019
Kuijpers	Nick	2096202
Mada	Sreeshalini	2049543
Niknamhesar	Sara	2085443
Tahan	Paria	2043889

1 Objectives

The objective of this project is to develop a web application that assists managers in the scooter business with their daily operations. The scope of the project was defined for the Master in computer engineering program and involved 10 students at the beginning, with 7 students ultimately completing the project.

The web applicationwill managers analyze application allows, product, and staff data to make informed business decisions. The application will enable basic modifications such as adding or deleting data, and it is critical that the owner can perform these changes when necessary. The system will provide a comprehensive overview of all necessary information, allowing for long-term decision-making based on data analysis.

The main goal of the project is to create a platform that streamlines business processes, ultimately increasing efficiency and profitability. This application will be designed with user-friendly interfaces and interactive features, making it accessible to a wide range of users. The project will leverage modern web technologies to ensure the application is scalable and flexible enough to accommodate future business growth.

In summary, this project aims to create a web application that provides managers with the necessary tools to analyze and manage their business, ultimately improving their overall performance.

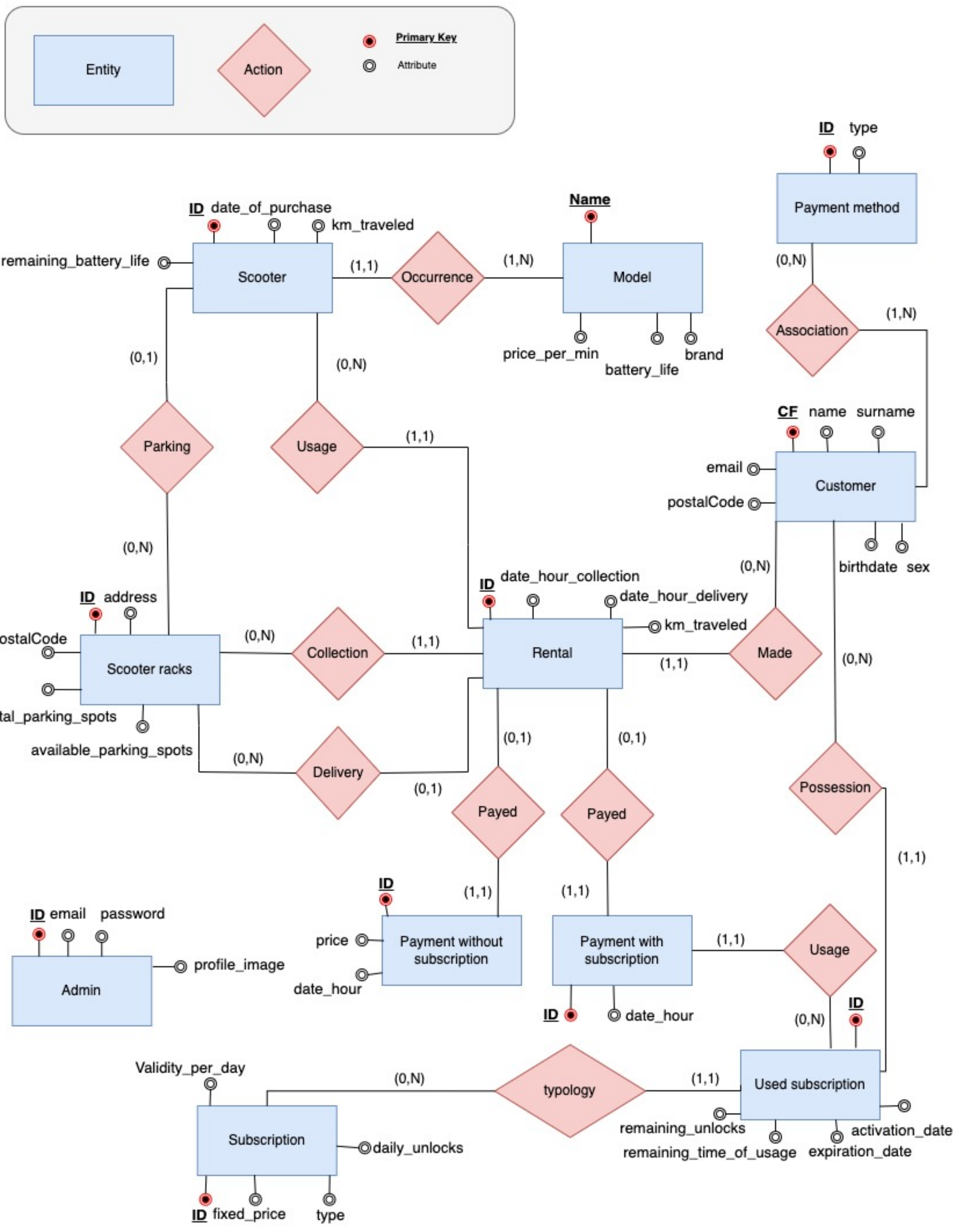
2 Main Functionalities

The web application is used to allow the administrator to interact with the database of WASCOOT, a web application regarding electric scooters to rent. Furthermore, another important functionality of the web application is to show the administrators some analytics regarding the rented scooters and customers, with the objective of giving some hints to improve the service. The website is divided into three main areas :

- Dashboard area: here we have our analytics and the most useful information displayed, such as the scooter racks positions;
- Login area: here administrators can log in and access the web application services;
- Manage pages area: this is the core of the web application, here the administrators can interact with the database, we have one of this pages for every core table in the database (for example scooters' manage page) and from there the administrator can do a few operations :
  - LIST, list all the rows of the table, e.g., list all the scooters in the database;
  - CREATE, create a new instance/row of that entity, e.g., create a new scooter;
  - EDIT, edit an existent row of the table, e.g., change something of a scooter.

3 Data Logic Layer

3.1 Entity-Relationship Schema



3.2 Data Dictionary: Entity Table

Entity	**Description**	**Attributes**	**P.Key**
Model	Entity representing the various models of a scooter.	• **Name**; specific type or version of the scooter; • **Price per min**; The amount of money for renting scooter for a minute; • **Brand**: model's manufacturer/company; • **Battery life**: model's battery life.	Name
Scooter	This entity represents the specific scooter. A number of scooters are provided for renters.	• **ID**: scooter identifier; • **Date of purchase**: when scooter has been bought; • **Km traveled**: distance covered by the scooter; • **Remaining battery life**: how much of the original battery life is left.	ID
Scooter racks	Structures designed to store and secure scooters when not in use.	• **ID**: scooter rack identifier; • **Address**: address of the parking spot; • **Postal code**: postal code of the rack's area; • **Total parking spots**: parking spot in this scooter rack; • **Available parking spots**: unoccupied parking spots in this scooter rack.	ID
Rental	This entity represents the process of renting a scooter.	• **ID**: ID of a specific scooter rental transaction; • **Date and hour collection**: date and hour in which a scooter is rented; • **Date and hour delivery**: date and hour in which a scooter is returned and delivered; • **Km traveled**: km traveled by the customer with this rental scooter.	ID

Customer	People registered in the application that can rent a scooter.	• **CF**: fiscal code of the customer; • **Email**; Email address of customer; • **Name**; The name of customer; • **Surname**; The surname of customer; • **Postal code**; The address of postal of customer; • **Sex**: customer's gender; • **Birthdate**: customer's birthdate.	CF
Admin	Entity representing the information regarding application's admins.	• **ID**; Admin identifier; • **Profile image**: admin's profile picture; • **Email**; Email address of the administrator • **Password**. Secret access to the administrator;	ID
Payment Method	Entity that lists the various types of payment methods.	• **ID**; Payment method's identifier; • **Type**: name of the payment method.	ID
Payment without Subscription	Entity representing the one-time payments.	• **ID**; The one-time payment identifier; • **Price**: price paid at the checkout after using the scooter; • **Date and hour**: when the payment has been received.	ID
Payment with Subscription	Entity representing payments	• **ID**; Payment identifier; • **Date and hour**: when the subscription's payment has been received.	ID

Used Subscription	This entity represents the information about the subscription process that the user has selected and paid for	• **ID**; Usage identifier; • **Remaining time of usage**: The amount of time left until the subscription ends; • **Activation date**: The date when the subscription was purchased; • **Expiration date**: The date when the subscription ends; • **Remaining unlocks**: how many other times can the customer unlock a scooter today.	ID
Subscription	This entity represents the information about the subscription process that allows the user to rent the scooters for a certain time and price	• **ID**; Subscription identifier; • **Fixed Price**: price of the subscription; • **Type**: subscriptions' types differ based on how many days it covers (examples: 1-day, 30-days, 6-months, . . .) • **Daily Unlocks**: how many times can the customer unlock scooters in a day with this subscription. • **Validity per Day**: how many hours can the customer use scooters with this subscription.	ID

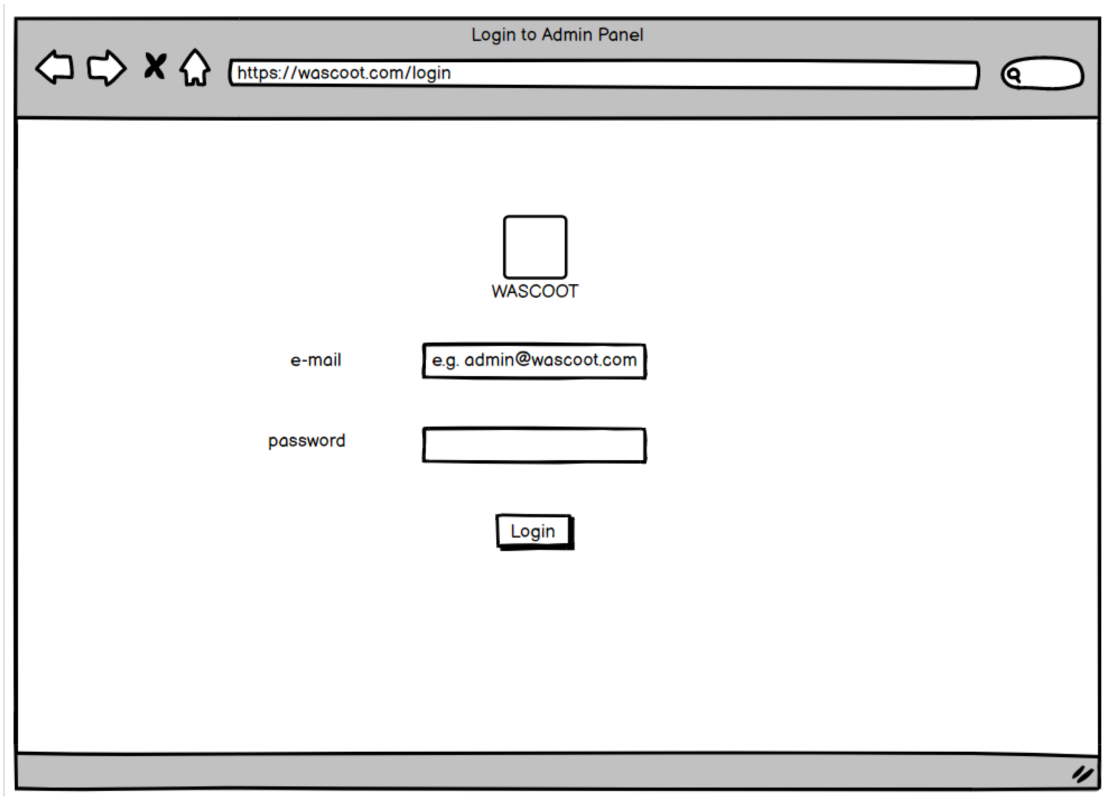
4 Presentation Logic Layer

The presentation logic layer of our web application consists of several pages that allow users to interact with the system. These pages are developed using JSP, servlets and REST technologies. We will describe the main pages.

4.1 Homepage

The homepage is the main entry point for the web application. It provides users with several options to interact with the system:

- **Login**: Users can log in to the system by clicking on the "Login" button, which will direct them to the home page.
- **Create Administrator**: Users can create a new administrator account by clicking on the "Create Administrator" button. This will direct them to a form where they can enter the administrator's details, including an optional profile picture.

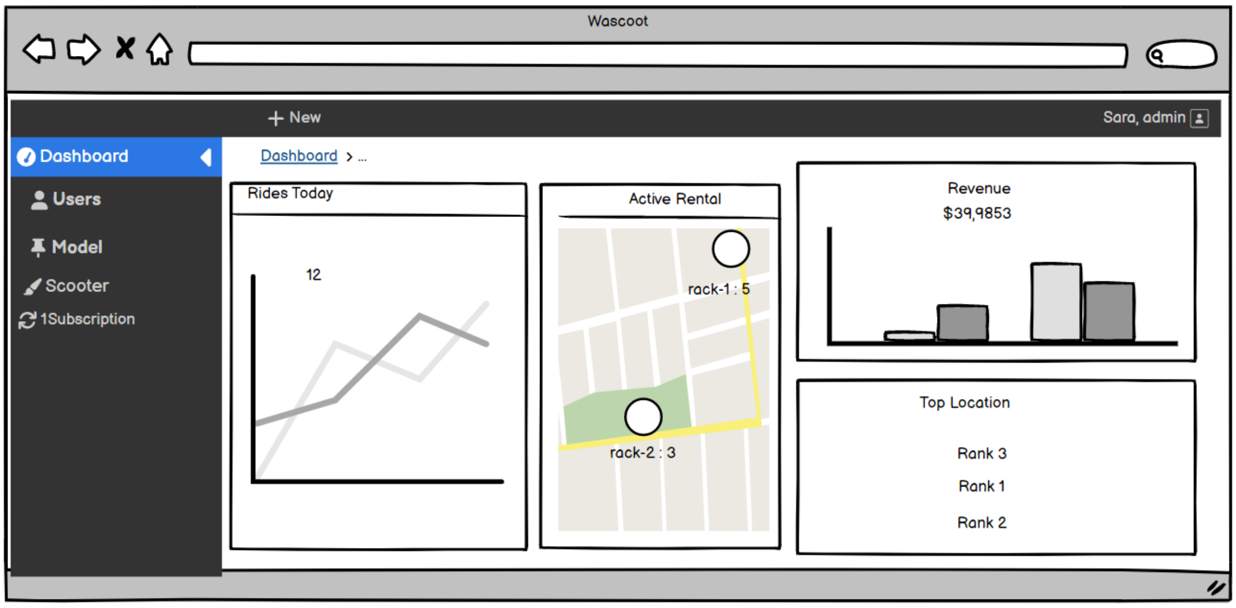


4.2 Dashboard

The dashboard is the main page of the web application. It provides users with an overview of their business data and allows them to visualize this data in various ways. The dashboard is developed using JSP and REST technologies.

On the dashboard, users can view the following information:

- **Scooter rack Locations**: Users can view the locations of their scooter racks on a map, along with information about each rack, such as its capacity and availability.
- **Revenue**: Users can view their business revenue over a specified period.
- **Top Locations**: Users can view a list of their top-performing locations based on their utilization.
- **Customer Information**: Users can view information about their customers, including age, rental history, and gender.



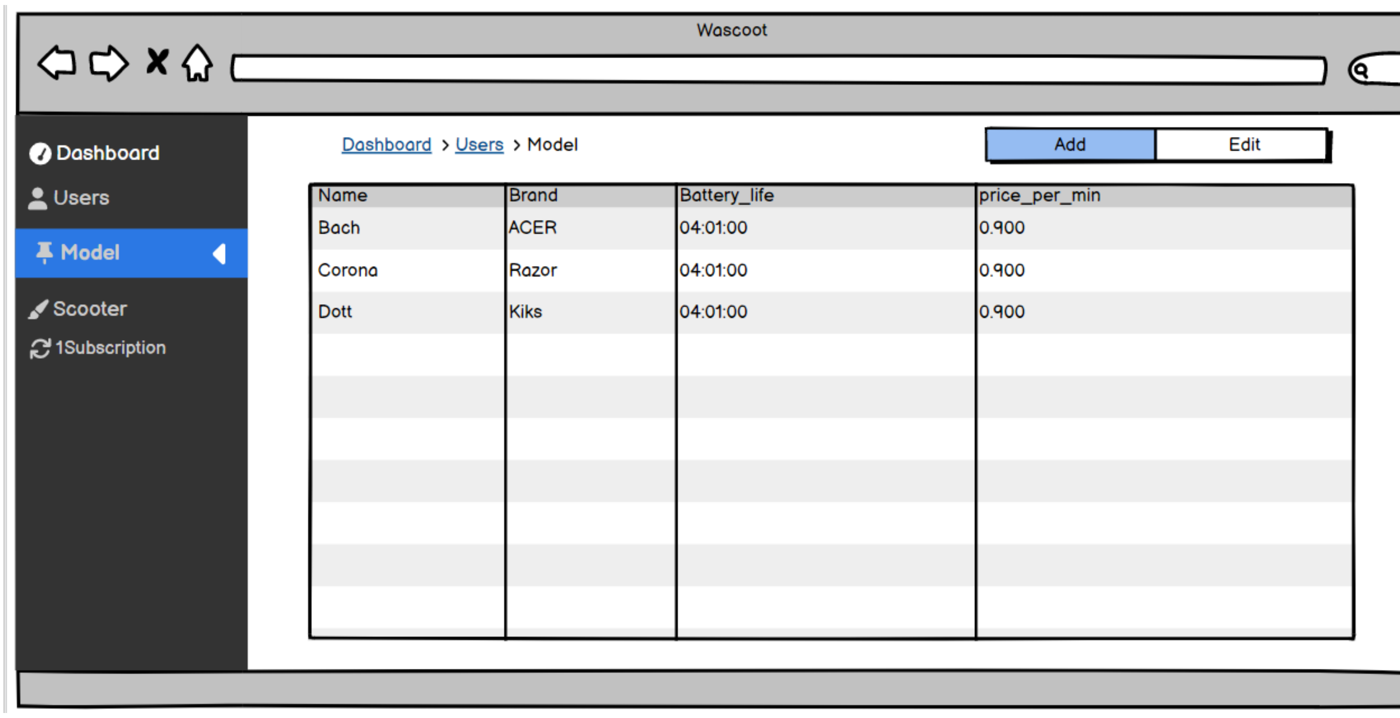
4.3 Manage Pages

In addition to the homepage and dashboard, our web application includes several manage pages that allow users to modify and create data. These pages are developed using JSP and REST technologies.

4.3.1 Model Page

The model page allows users to view information about scooter models and to modify or create new models through a form. The following pages are available:

- **Show Models**: Users can view a list of all scooter models in the system, along with information such as the model name, brand, and specifications.
- **Modify Model**: Users can modify an existing scooter model by selecting it from the list and editing its information in a form.
- **Create Model**: Users can create a new scooter model by filling out a form with the model's details.

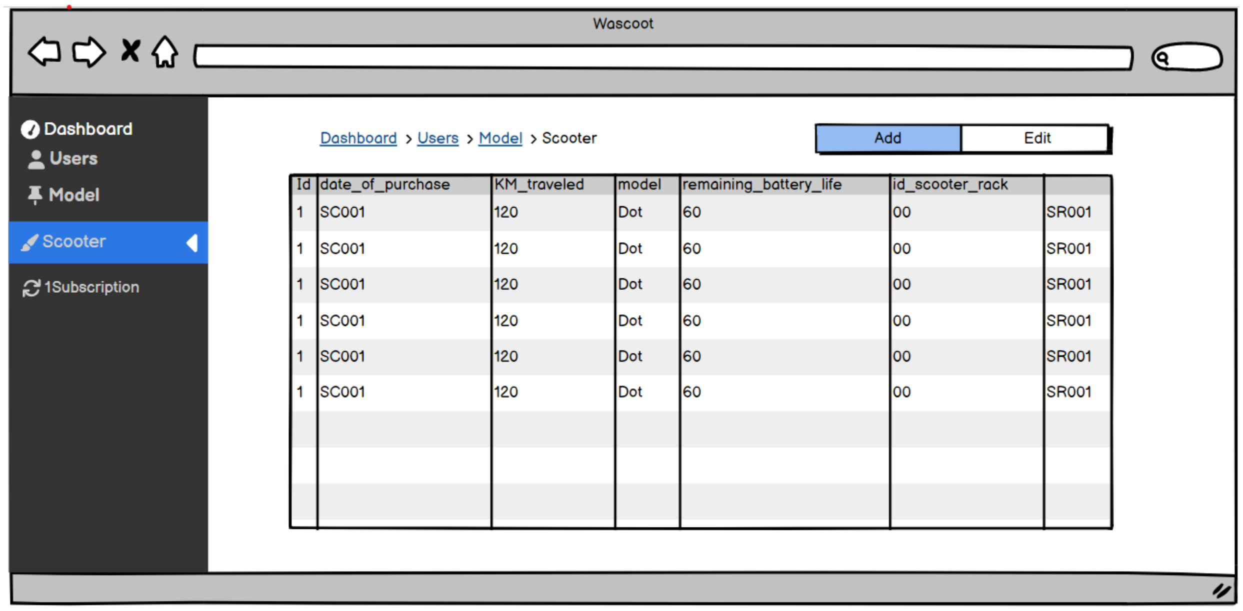


Name	Brand	Battery_life	price_per_min
Bach	ACER	04:01:00	0.900
Corona	Razor	04:01:00	0.900
Dott	Kiks	04:01:00	0.900

4.3.2 Scooter Page

The scooter page allows users to view information about scooters and to modify or create new scooters through a form. The following pages are available:

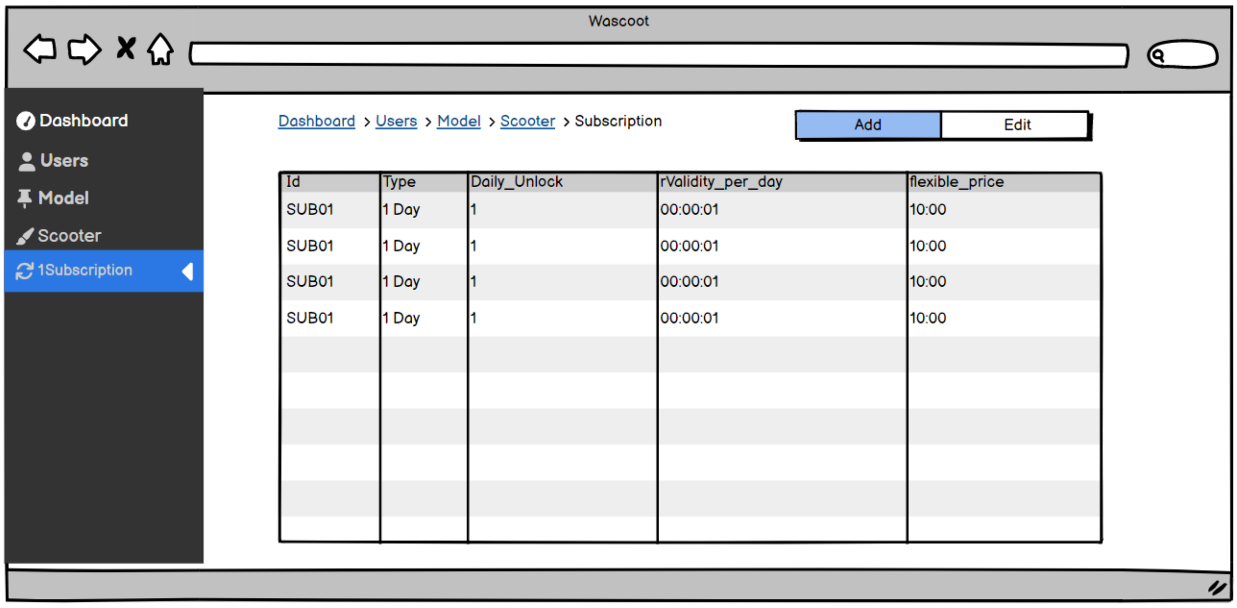
- **Show Scooters**: Users can view a list of all scooters in the system, along with useful information.
- **Modify Scooter**: Users can modify an existing scooter by selecting it from the list and editing its information in a form.
- **Create Scooter**: Users can create a new scooter by filling out a form with the scooter's details.



Id	date_of_purchase	KM_traveled	model	remaining_battery_life	id_scooter_rack	
1	SC001	120	Dot	60	00	SR001
1	SC001	120	Dot	60	00	SR001
1	SC001	120	Dot	60	00	SR001
1	SC001	120	Dot	60	00	SR001
1	SC001	120	Dot	60	00	SR001
1	SC001	120	Dot	60	00	SR001

4.4 Subscription Page

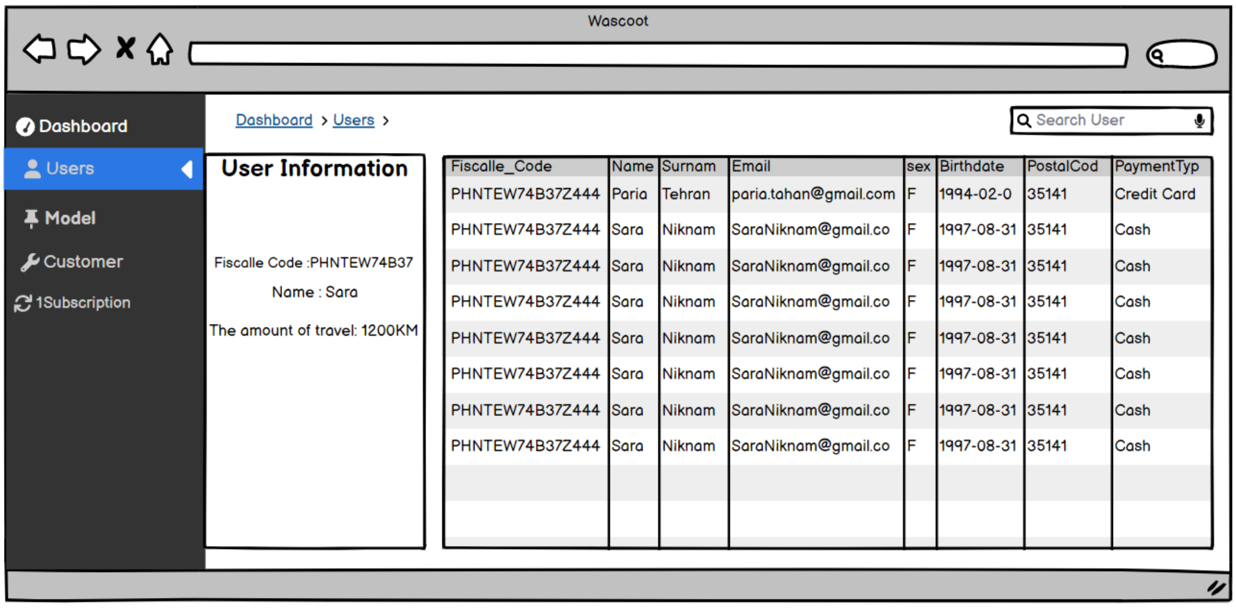
Similarly to the scooter page, we have a page to manage subscriptions.



Id	Type	Daily_Unlock	rValidity_per_day	flexible_price
SUB01	1 Day	1	00:00:01	10:00
SUB01	1 Day	1	00:00:01	10:00
SUB01	1 Day	1	00:00:01	10:00
SUB01	1 Day	1	00:00:01	10:00

4.5 Customer Page

Similarly to scooter and subscription pages, we have a page to manage customers.


Wascoot

Dashboard
Users
Model
Customer
1Subscription

Dashboard > Users >

Search User

User Information

Fiscalle Code :PHNTEW74B37

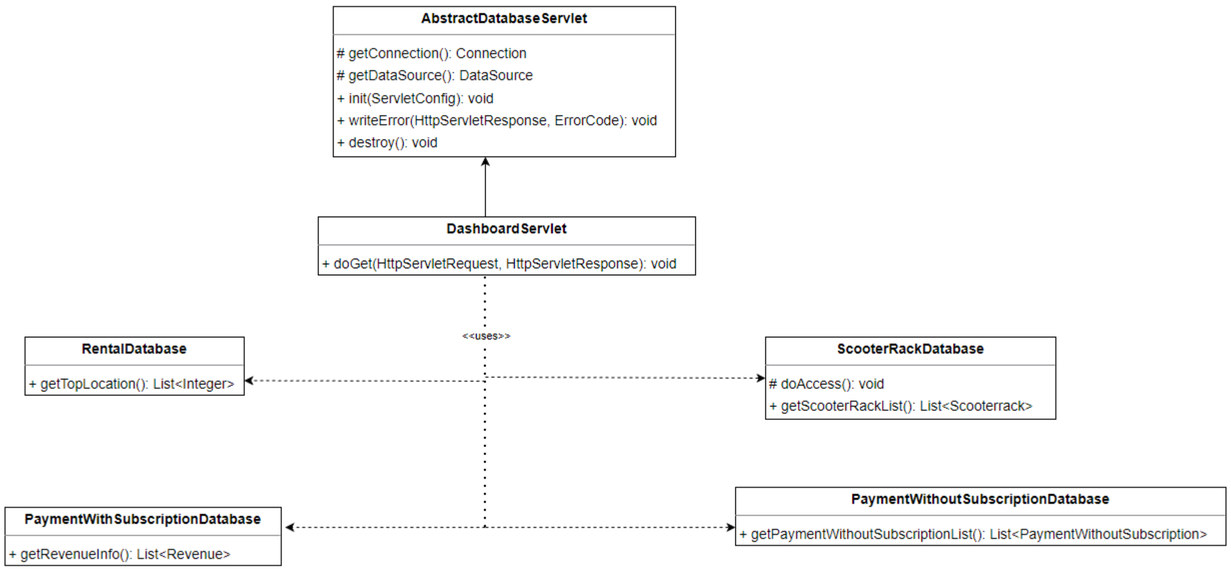
Name : Sara

The amount of travel: 1200KM

Fiscalle_Code	Name	Surnam	Email	sex	Birthdate	PostalCod	PaymentTyp
PHNTEW74B37Z444	Paria	Tehran	paria.tahan@gmail.com	F	1994-02-0	35141	Credit Card
PHNTEW74B37Z444	Sara	Niknam	SaraNiknam@gmail.co	F	1997-08-31	35141	Cash
PHNTEW74B37Z444	Sara	Niknam	SaraNiknam@gmail.co	F	1997-08-31	35141	Cash
PHNTEW74B37Z444	Sara	Niknam	SaraNiknam@gmail.co	F	1997-08-31	35141	Cash
PHNTEW74B37Z444	Sara	Niknam	SaraNiknam@gmail.co	F	1997-08-31	35141	Cash
PHNTEW74B37Z444	Sara	Niknam	SaraNiknam@gmail.co	F	1997-08-31	35141	Cash
PHNTEW74B37Z444	Sara	Niknam	SaraNiknam@gmail.co	F	1997-08-31	35141	Cash
PHNTEW74B37Z444	Sara	Niknam	SaraNiknam@gmail.co	F	1997-08-31	35141	Cash

5 Business Logic Layer

5.1 Class Diagrams


AbstractDatabaseServlet
# getConnection(): Connection
# getDataSource(): DataSource
+ init(ServletConfig): void
+ writeError(HttpServletResponse, ErrorCode): void
+ destroy(): void
DashboardServlet
+ doGet(HttpServletRequest, HttpServletResponse): void
<<uses>>
RentalDatabase
+ getTopLocation(): List<Integer>
ScooterRackDatabase
# doAccess(): void
+ getScooterRackList(): List<Scooterrack>
PaymentWithSubscriptionDatabase
+ getRevenueInfo(): List<Revenue>
PaymentWithoutSubscriptionDatabase
+ getPaymentWithoutSubscriptionList(): List<PaymentWithoutSubscription>

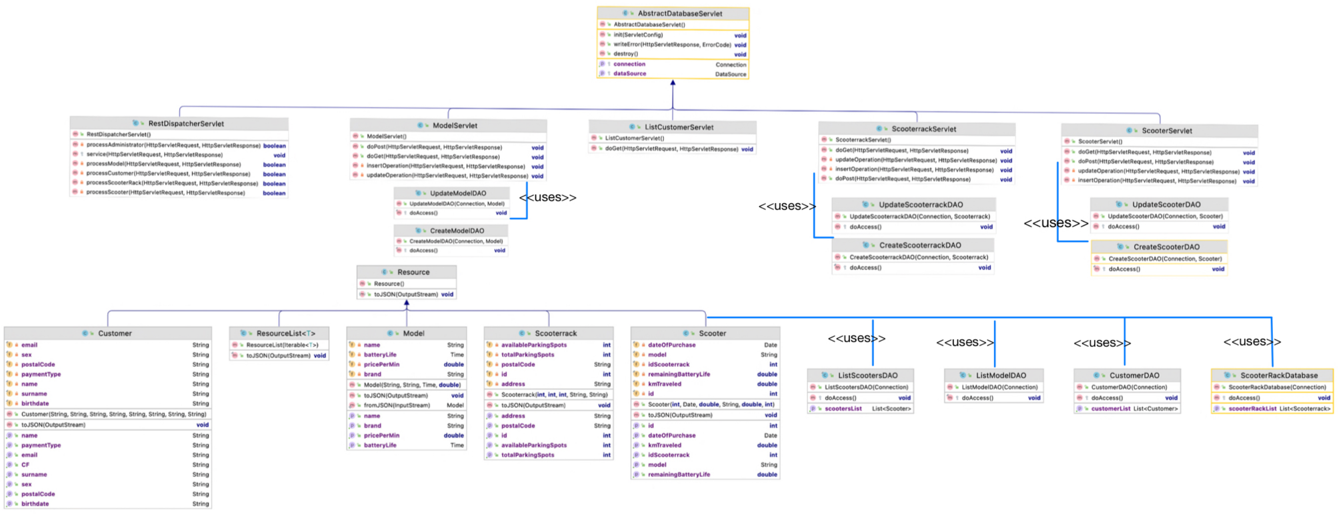
This servlet uses the classes shown to access the database. This class diagram represents the classes involved in the dashboard servlet, which manages the dashboard page, it contains (some of) the classes used to handle three types of resources:scooter-Rack-List, top location, and revenue info. We used the proper queries to retrieve the necessary lists for each of these needed sections. For this, we used JDBC to establish a connection to the PostgreSQL database and ran the necessary queries from there. The desired tables were extracted, and the extracted queries were then turned into Java objects, and we used the proper queries to return the necessary lists for each of these desired sections. To achieve this, we used JDBC to establish a connection to the PostgreSQL database and ran the necessary queries. To present the collected data to the user, we had to extract from the relevant tables, turn the extracted queries into Java objects, and use JSP and HttpServlet technologies.

As a result, we developed a servlet called dashboard servlet, and called the resources we already had, turned them into the appropriate Java models, then sent the desired characteristics to the page using the HttpServlet's APIs. We transferred the Jsp there and showed it, along with the following descriptions of the aforementioned sections.

1. **scooterRackList**: shows a list of all locations where scooter rentals are available.

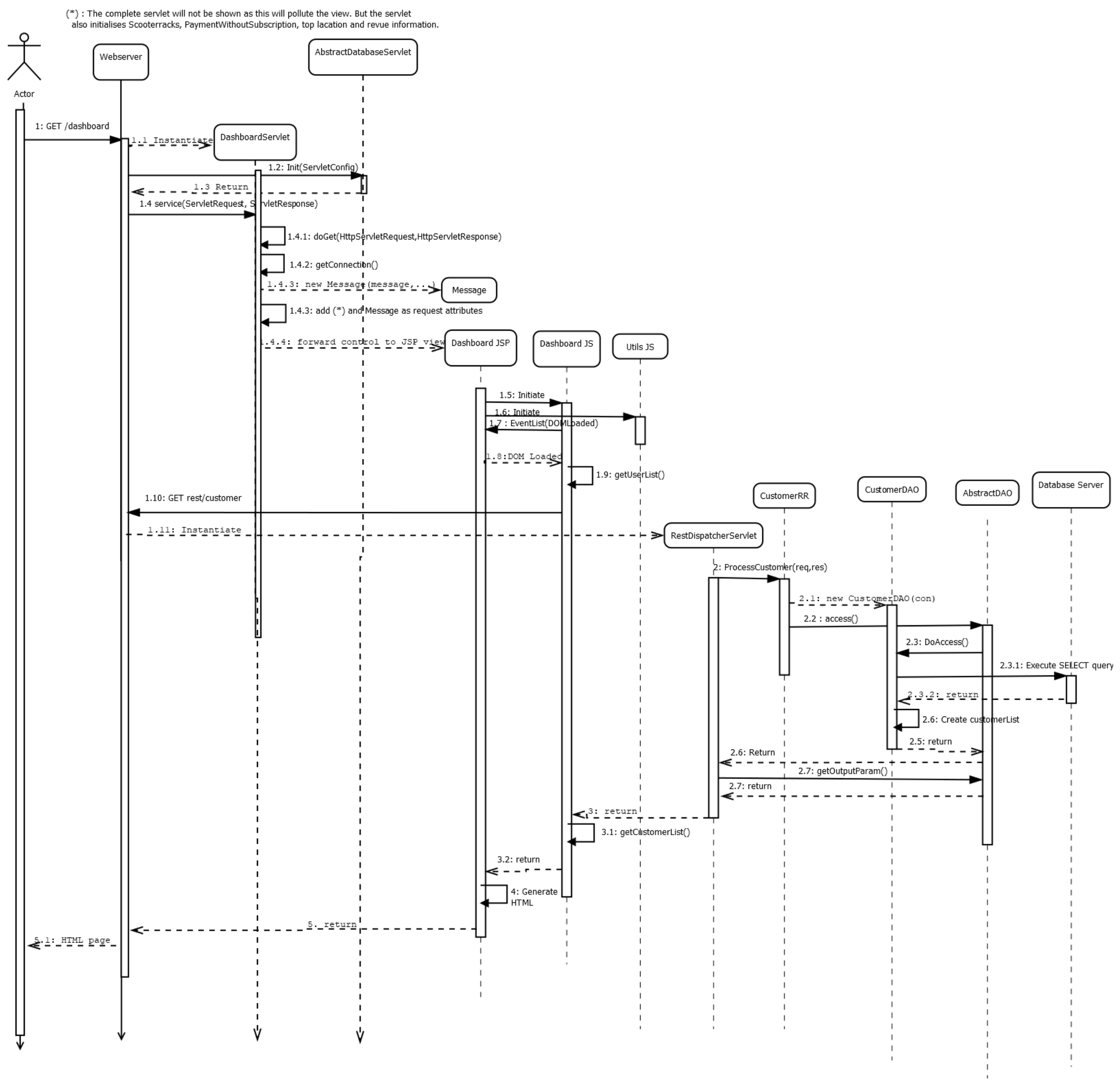
2. **revenueInfo**: this shows the overall amount of money we made from renting scooters and our monthly income from scooter rentals.

3. **top location**: using this reasoning, we can identify the city's busiest scooter rental location.



The second diagram represent the manage pages servlet and rest. The manage pages are model, scooter, scooter rack, rental, payment methods, customer and subscription. On these pages, we can interact with the respective tables of the database, for example: to view the list and add or edit that entity. In this case, we show the scooter, scooterracks, model and customer entities and have a servlet for viewing the list of scooters, one for adding a new scooter and one for editing an existing scooter. Then we also provide the REST API for getting the list of these entites. Both the rest and the servlets use some classes to access the database, as shown in the diagram.

5.2 Sequence Diagram



(*) : The complete servlet will not be shown as this will pollute the view. But the servlet also initialises Scooterracks, PaymentWithoutSubscription, top lacation and revue information.

This sequence diagram depicts the communication flow for a REST API call from a JSP page (Dashboard) in our web application. The objective of this diagram is to illustrate the specific steps involved in the communication flow between the different components involved in this process.

At the start of the sequence, an actor sends a request to the WebServer for the Dashboard page. The WebServer forwards this request to the DashboardServlet, which renders the DashBoardJSP page. The DashBoardJSP page loads the necessary DashBoardJS script, which makes an HTTP GET request to retrieve customer data from the REST API (/rest/customer).

The RestDispatcherServlet, which is responsible for handling all REST API requests in the web application, delegates the handling of the request to the CustomerRR class, which interacts with the CustomerDAO class to retrieve the customer data from the database. The customer data is returned to the CustomerRR object, which returns it to the RestDispatcherServlet. The RestDispatcherServlet then sends the response back to the dashboard javascript call, which in turn passes it to the last function (getCustomerList) which updates the HTML page.

Overall, this sequence diagram shows the complex interaction between different components involved in a REST API call in our web application, and how each component plays an important role in this process. It was challenging to comprehend this communication flow, and this diagram helps to provide a visual representation of how the different components work together to retrieve and display data from the REST API.

5.3 REST API Summary

The list of the rest API for this homweork are still "in progress" because we have to decide according to the frontend where it is better to use a REST call or not. There are some resources we used for rest in the next table.

URI	**Method**	**Description**
rest/scooter	GET	Returns the list of scooters available in the database
rest/scooterrack	GET	Returns the list of scooterracks available in the database
rest/model	GET	returns the list of models available in the database
rest/customer	GET	returns the list of customers available in the database
rest/administrator/id/{id}	GET	Returns the ID of the administrator from the database
rest/administrator/email/{email}	GET	Returns the Email of the administrator from the database

5.4 REST Error Codes

Here is the list of errors defined in the application.

Error Code	**HTTP Status Code**	**Description**
-103	BAD_REQUEST	missing Email
-104	BAD_REQUEST	missing Password
-105	BAD_REQUEST	Submitted credentials are wrong
-200	BAD_REQUEST	Operation unknown
-999	INTERNAL_SERVER _ERROR	Internal Error
-E100		cannot search for the rentals in the database
-E200		cannot search for the rentals: unexpected error while accessing the database
-E300		cannot create the administrator: administrator already exists
-E400		unsupported multimedia type for an employee photo. expected image/png or image/jpg
-E4A1	BAD_REQUEST	Accept request header missing
-E4A2	NOT_ACCEPTABLE	unsupported output media type. resources are represented only in application/json
-E4A3	BAD_REQUEST	Input media type not specified. content-type request header issing

-E4A4	BAD_REQUEST	unsupported input media type. resources are represented only in application/json.
-E4A5	METHOS_NOT _ALLOWED	unsupported operation
-E4A6	NOT_FOUND	unknown resource requested
-E4A7	BAD_REQUEST	wrong format for URL, e.g.: /administrator/email/email: no email specified
-E4A8	BAD_REQUEST	cannot create the resource, e.g.: no administrator JSON object found in the request
-E5A1	INTERNAL_SERVER _ERROR	Cannot create the resource: unexpected error.
-E5A2	CONFLICT	cannot create the resource. it already exists
-E5A3	NOT_FOUND	resource Not found. cannot delete it.
-E5A4	CONFLICT	cannot delete resource: other resources depend on it.
-E10A1	INTERNAL_SERVER _ERROR	cannot list resource: unexpected error

Table 4: REST API

5.5 REST API Details

We reported three different types of resources that our web applications handle.

Available scooter's list in the database

The following endpoint allows to a list getting all scooters which are available in the database.

- URL: `rest/scooters`
- Method: `GET`
- URL Parameters: No parameters are required in the URL
- Data Parameters: No parameters are required
- Success Response: Upon success, the servlet returns a JSON object with the required information.
- Error Response:
  Code: INTERNAL_SERVER_ERROR
  Content: "code": -E5A1, {"message" :{'Cannot list scooter(s): unexpected database error.'}}
  Code: INTERNAL_SERVER_ERROR
  Content: "code": -E5A1, {"message" :{'Cannot list scooter(s): unexpected error.'}}

Available scooter rack's list in the database

The following endpoint allows to a list getting all scooter racks which are available in the database.

- URL: `rest/scooterrack`
- Method: `GET`
- URL Parameters: No parameters are required in the URL

- Data Parameters: No data parameters required.
- Success Response: Upon success, the servlet returns a JSON object with the required information.
- Error Response: Code: INTERNAL_SERVER_ERROR
  Content: "code": -E5A1, {"message" :{'Cannot list employee(s): unexpected database error.'}}
  Code: INTERNAL_SERVER_ERROR
  Content: "code": -E5A1, {"message" :{'Cannot list employee(s): unexpected error.'}}

Scooter model's list in the database

The following endpoint allows to a list getting all scooter models in the database.

- URL: `rest/model`
- Method: `GET`
- URL Parameters: No parameters are required in the URL
- Data Parameters: No data parameters required.
- Success Response: Upon success, the servlet returns a JSON object with the required information.
- Error Response:

  Code: INTERNAL_SERVER_ERROR
  Content: "code": -E6A1, {"message" :{'Cannot list models: unexpected error'}}
  Code: INTERNAL_SERVER_ERROR
  Content: "code": -E6A2, {"message" :{'Cannot search models: unexpected error.'}}

Customer's list in the database

- URL: `rest/customer`
- Method: `GET`
- URL Parameters: No parameters are required in the URL
- Data Parameters: No data parameters required.
- Success Response: Upon success, the servlet returns a JSON object with the required information.
- Error Response:

  Code: INTERNAL_SERVER_ERROR
  Content: "code": -E10A1, {"message" :{'Cannot list customer(s): unexpected database error.'}}
  Code: INTERNAL_SERVER_ERROR
  Content: "code": -E10A1, {"message" :{'Cannot list customer(s): unexpected error.'}}

Administrator- ID

the following endpoint allows the access of the inf

- URL: `rest/administrator/id/{id}`
- Method: POST
- URL Parameters: Required: administrator ID: the identifier of the administrator.

- Data Parameters:
  - Id:{int}
- Success Response: Upon success, the servlet returns a JSON object with the required information.
  - Id:{int}
  - email:{string}
  - password:{string}

  Content:"administrator":{"id":1,"email":"admin@wascoot.com","password":"123321"}}
- Error Response:

  Code: INTERNAL_SERVER_ERROR
  Content:"code": -E5A1, {"message" :{'Cannot search administrator(s): unexpected database error.'}}
  Code: BAD_REQUEST
  Content:"code": -E4A7, {"message" :{'Cannot search administrator(s): wrong format for URI /administrator/id/id.'}}
  Code: INTERNAL_SERVER_ERROR
  Content:"code": -E5A1, {"message" :{'Cannot search administrator(s): unexpected error.'}}

Administrator- Email

- URL: `rest/administrator/email/{email}`
- Method: POST
- URL Parameters: Required: administrator email: the email of the administrator.
- Data Parameters:
  - email:{string}
- Success Response: Upon success, the servlet returns a JSON object with the required information.
  - Id:{int}
  - email:{string}
  - password:{string}

  "administrator":{"id":1,"email":"admin@wascoot.com","password":"123321"}
- Error Response: Code: INTERNAL_SERVER_ERROR
  Content:"code": -E5A1, {"message" :{'Cannot search administrator(s): unexpected error.'}}
  Code: BAD_REQUEST
  Content:"code": -E4A7, {"message" : 'Cannot search administrator(s): wrong format for URL/administrator/email/email.
  Code: INTERNAL_SERVER_ERROR
  Content:"code": -E5A1, {"message" :{'Cannot search administrator(s): unexpected database error'}}

6 Group Members Contribution

Alberto Crivellari: implemented rest of scooter, scooterrack and model, implemented servlets for scooter, scooterrack and rental for the manage page section, wrote the section regarding class diagrams, data dictionary:entity table and main functionalities, and helped with the rest API section.

Nick Kuijpers: Worked on the Dashboard with Sara. Did the Scooterrack, paymentwithoutsubscriptions servlets and the customer rest API. Made mockups about the scooterrack page and model page. Made the sequence diagram of the customer rest API. Wrote in the report the objectives chapter and the presentation logic layer.

Shiva Gharehzad: Worked on the database design, ER schema, SQL for database , inserting data, data layer description, rest API of the model and scooters, searching model by name in rest part, servlet for scooter, class diagram, wrote parts of the report.

Borwoei Huang: Did the login page for authorized administrators to log in to our web application. Filter is built to block the unauthorized users. To manage the administrator account, creation ad searching of the administrator functions were made. Administrator creation servlet manages multipart. Worked on the administrator rest api with AJAX administrator searching form reading the output of Id or Email admin-searching. Wrote parts of the report.

Sreeshalini Mada: Made administrator rest api for searching the admin table in the database. Wrote parts of the report.

Sara Niknamhesar: Working on the dashboard section with Nick, doing the topLocation, scooterRackList, and Revenue sections of the servlet, designing general Mockups about the display process on the dashboard and other panel pages, designing class diagrams and writing them in the Business Logic Layer section.

Paria Tahan: Completed the design of the database also created the ER-Schema. Built SQL for database schema and data insertion. Worked on the Model (Create, Update, List), Customer (List), Subscription (Create, Update, List) and Payment method (Update, List) from manage pages. Created the servlet file for each of the resources except Admin. Generated the manage pages class diagram. Helped with the Balsamiq mockups and writing the report.