

TECHNICAL DESIGN

PROJECT CLIENT ON BOARD

Date: 2022/11/24

Group: Brave Alligators

Authors:

Adrian Krantz – 524202

Darius Bejan - 516471

Georgs Jakubovskis - 517100

Hai Ha Pham - 493611

Tiffany Deng – 452428

Sadra Samadzadeh - 519407

Table of contents

Introduction	2
General Overview and approach	2
Design Consideration	3
System architecture	4
Information Architecture	5
Software Architecture Diagram	6
Database Design	7
User Interface Design.....	13
Customer:	13
Global Admin:.....	17
Company Admin:.....	27
Security Design	8

Introduction

As part of “Client on board” class the team of authors were tasked to undertake a project of creating an application requested by a client. The client “ProfitFlow” had requested for the authors to build a Web application that would simulate the kind of environment the client currently was using. This application had to be capable of retrieving the status of converters, as well as generating a ticket if the status did display an error. To solve this task the authors had to integrate API’s of converter companies. Each company had their own respective API that authors had to research and integrate to retrieve the converter status’s. This document describes how the authors planned to implement the proposed solution from the Functional Design. These decisions are argued and supported by the authors’ reasoning and the clients preferences.

General Overview and approach

To achieve the given task, the team authors had to work in a developer team, meaning the tasks needed to be defined and distributed amongst the team members. To achieve the needed result the team of authors used such work frameworks as SCRUM to provide an organized way of working on and solving problems.

The group also utilized GitLab as a shared work environment where group members can add and edit code on their local machine and eventually merge the code with the main code base of the group.

Project development was split in 4 sprints including Sprint 0, where every sprint is dedicated to a development period of the application.

Each sprint has its own set of backlog items, excluding Sprint 0, since during that period most of the initial setup for the start of the project is done.

Design Consideration

The client had given the team of authors a set of preferences describes in the case document. Client had stated the it would be appreciated if the end product that the teams delivers would consist of frontend and backend that are both written in TypeScript. TypeScript is a superset of JavaScript, meaning both languages are similar. Since authors had previous experience in web development with Java Script it would not require time to research Type Script as it uses most of the same concepts as Java Script.

A PostgreSQL database was to be created to store data retrieved. To retrieve the data gotten from the converters and users a database is needed to store the data in a well-structured manner. The team of authors had decided to host the database on a web server via Microsoft Azure for ease of access and taking account that the alternative being the virtualization application “Docker” is highly disliked by the development team.

A CRON job was also requested to be a part of the application. CRON is a job scheduler on Unix-like operating systems. Users who set up and maintain software environments use cron to schedule jobs, also known as CRON jobs, to run periodically at fixed times, dates, or intervals.

One of the clients’ requirements was for the author’s to use a CRON job. The authors chose to use a CRON job for retrieving the statuses of the converters, since CRON can be scheduled to execute scripts. The CRON job can be used to fetch the converter’s statuses daily or hourly, depending on the clients preferences.

System architecture

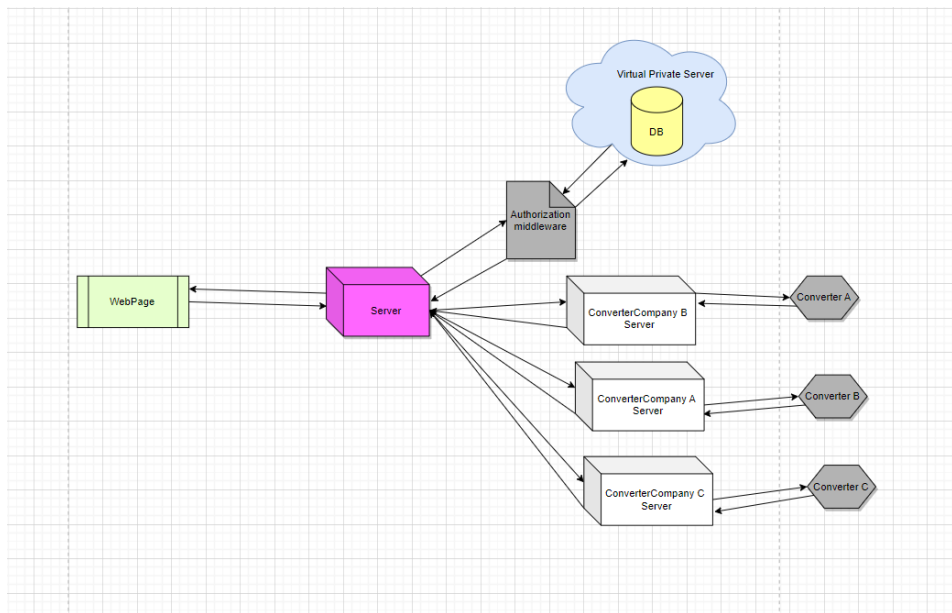


Figure 1 - Logical View (functional components)

Figure 1 displays the logical view of the authors had developed. This diagram displays the inner working of our application, and how its different parts communicate with one another.

A user who is not a direct administrator of the system always has to interact with the application via the Web Page, that acts as a gateway from the user to the server storing all the required data and processing algorithms.

When a user is communicating with the server, all CRUD actions that involve change of data requires authorization, in order to prevent unwanted access to data.

Authorization is also required to determine the rights that the certain user class has. Rights vary from Client to Global Admin.

After user request is authorized and checked, the input is then relayed to the server, which uses its programmed processing algorithms and the database to return a valid response

As described before getting the status of the converters is one of the primary required functions of the application. Figure 1 also demonstrates the communication of converters from different companies to the server, which is done by fetching the status of the converters with the help of each respective converters API.

The server hosts a CRON job that is responsible for checking in with the converters fetching their status from the respective company servers.

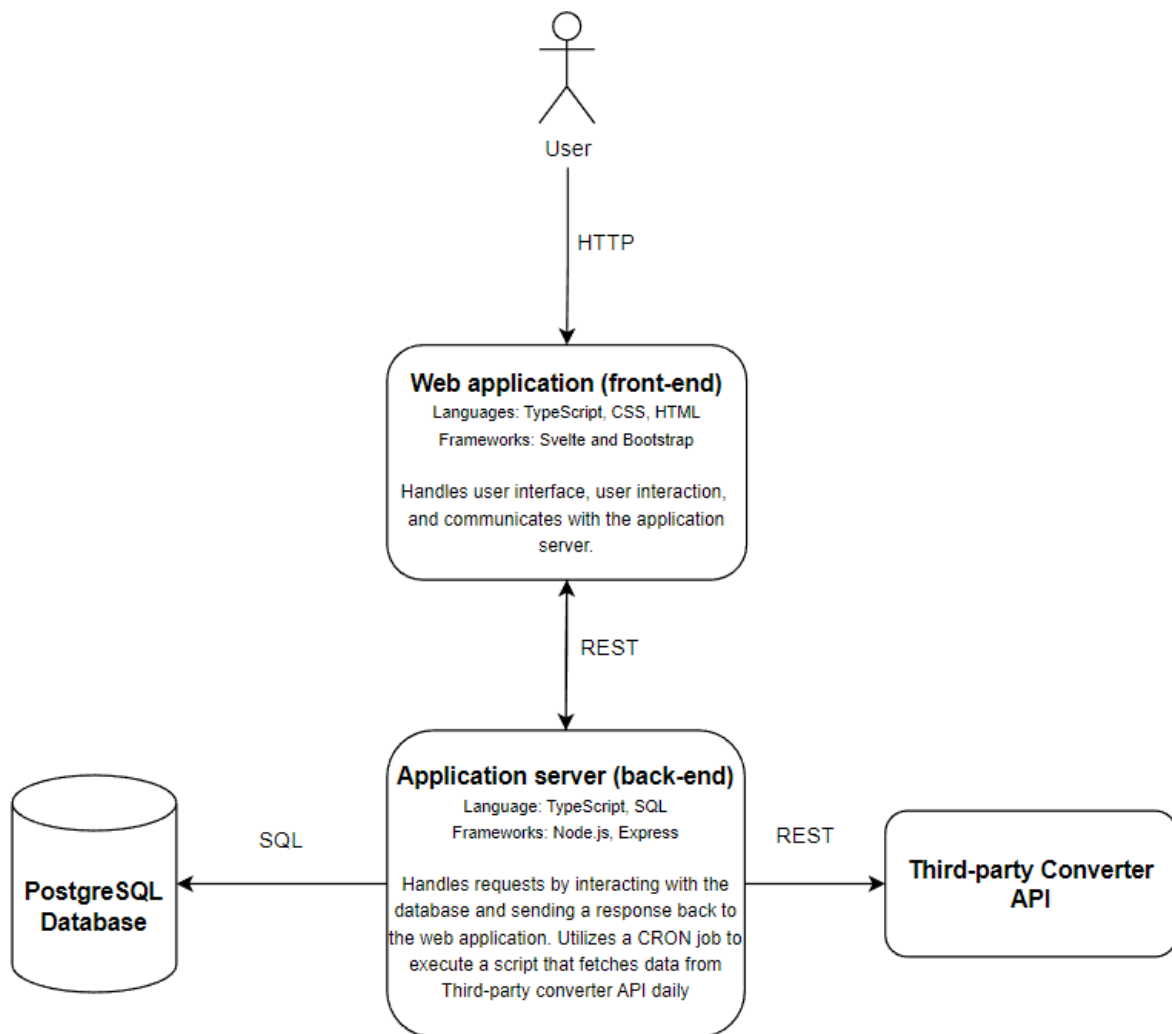
The Postgres database is hosted on a virtual private server that communicates with our server to exchange data.

Information Architecture

As can be seen in the Database Design section, the information that will be stored in the system including:

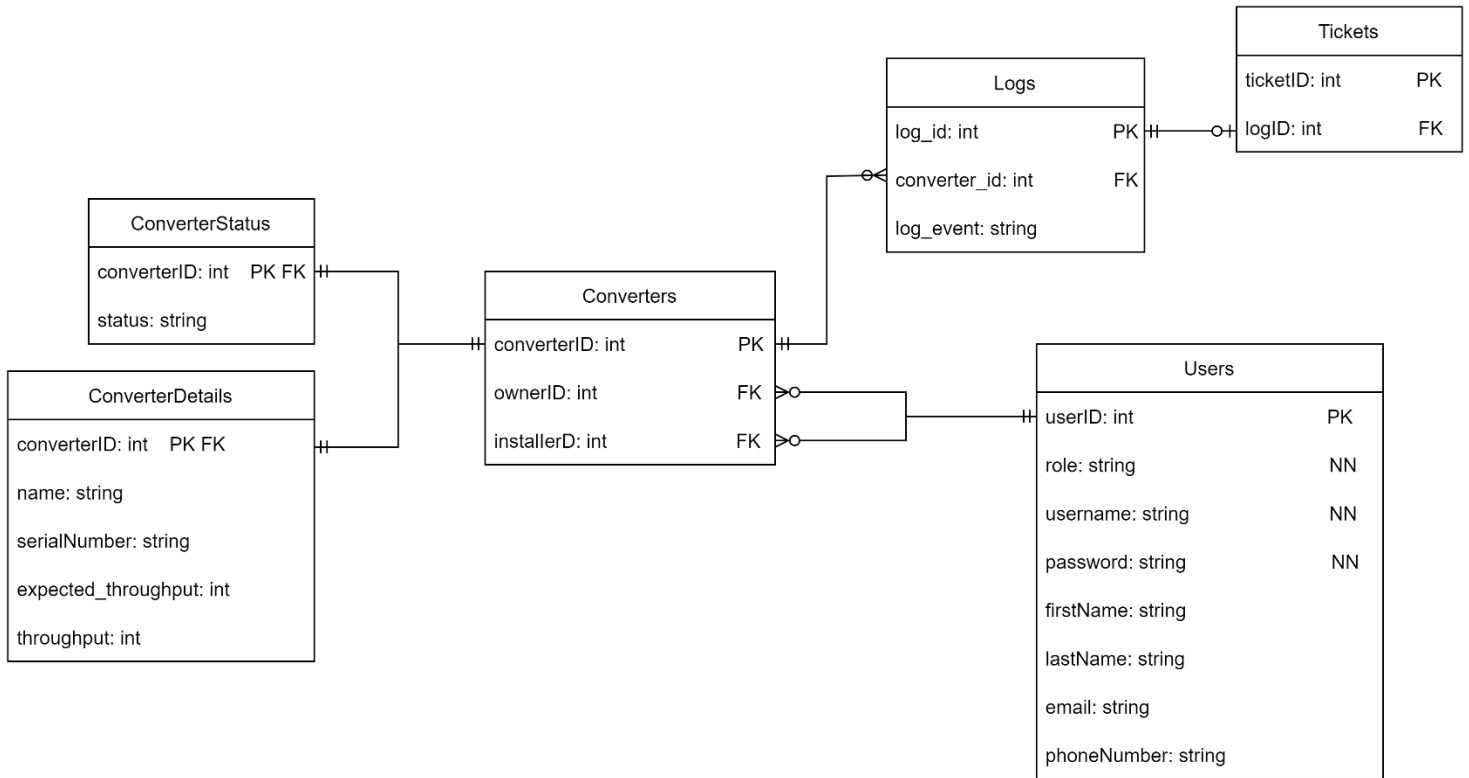
- Personal information (e.g., first names, last names, email addresses, phone numbers) and login credentials (e.g., usernames and passwords) of the users. These users include global administrators, company administrators and customers. Personal information of the users can be considered personally identifiable information (PII).
- Information about the converters: names, serial numbers, their owners, and the companies that installed them, as well as the expected throughputs, status, and the IP address of these converters.
- Event logs of the converters, with (possibly) tickets associated to these logs.

Software Architecture Diagram



The above software architecture diagram represents the structural design of our system and the relationships between each component in the system. It acts as a blueprint for the design and development of our system.

Database Design



The above figure displays the authors' current proposed database design for the application. It includes the overall structure of the database, including the organization of data into tables and the relationships between tables.

Table dictionary

Below we explain the composition of every table in our database. (### are used to indicate numbers and XXX are used to indicate letters)

users

Field name	Data Type	Format	Required	PK or FK
user_id	int	###	Yes	PK
role	varchar(13)	"Client", "CompanyAdmin", "GlobalAdmin"	Yes	
username	varchar(25)	XXX	Yes	
password	varchar(255)	XXX	Yes	
first_name	varchar(25)	XXX	No	
last_name	varchar(25)	XXX	No	
email	varchar(30)	%@%.%	No	
phone_number	varchar(15)	XXX	No	

companies

Field name	Data Type	Format	Required	PK or FK
company_id	int	###	Yes	PK
company_name	Varchar(25)	XXX	Yes	

converters

Field name	Data Type	Format	Required	PK or FK
converter_id	int	###	Yes	PK
owner_id	int	###	Yes	FK
installer_id	Int	###	Yes	FK

converter_status

Field name	Data Type	Format	Required	PK or FK
converter_id	int	###	Yes	PK & FK
status	varchar(15)	XXX	Yes	

converter_details

Field name	Data Type	Format	Required	PK or FK
converter_id	int	###	Yes	PK & FK
converter_name	varchar(15)	XXX	No	
serial_number	varchar(20)	XXX	No	
expected_throughput	numeric	###	No	
throughput	numeric	###	No	

logs

Field name	Data Type	Format	Required	PK or FK
log_id	int	###	Yes	PK
installer_id	int	###	No	FK
log_event	text	XXX	No	
created_at	timestamp	dd:mm:yyyy	Yes	

tickets

Field name	Data Type	Format	Required	PK or FK
ticket_id	Int	###	Yes	PK
log_id	Int	###	No	FK
created_at	timestamp	dd:mm:yyyy	Yes	

Security Architecture

Our security architecture is primarily based around using JWT tokens for the authorization of users and admins and encrypting the users' passwords in our database using bcrypt which hashes and salts passwords before storing them in our database.

1. JWT Token

JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

We use it to get some information regarding the user from our database and send it to our front end when the user has been identified and authorized.

When a user identifies themselves with the correct username and password, we create a "payload" which contains the users username, date of creation and their role. Then we sign the token with the payload and a secret.

Then with a middleware we check if the user has a valid token by getting the token from the authorization header and verifying it. If the token is valid the payload details are available to be used from the backend calls.

```
export function isLoggedIn (req: { get: (arg0: string) => any; token: string; user: (jwt.Jwt & jwt.JwtPayload & (string | jwt.JwtPayload)) | null; }, res: Response, next: () => void) {
  let token = req.get('Authorization');
  token = token.split(" ");

  if (token[1].length < 10) {
    return res.status(401).json({error: 'You are not logged in'})
  }

  // @ts-ignore
  jwt.verify(token[1], secret, {options: {algorithm: 'HS256'}}, (err) => {
    if (err) {
      return res.status(401).json({error: 'The token is not valid'})
    } else {
      // @ts-ignore
      req.user = jwt.decode(req.token, secret);
      return next();
    }
  })
}
```

2. Bcrypt

```
router.post('/', tokenBodyDetails, compareLoginDetails, (req, res) => {
  let payload = {
    username: req.body.username,
    date: new Date(),
    // @ts-ignore
    roles: req.role
  };

  router.post(path: '/', handlers: async(req: Request, res: Response<ResBody, Locals>) => {
    bcrypt.hash(req.body.password, saltOrRounds: 10, callback: function (err: Error | undefined, hash: string) {
      return res.status(500).json({body: {error: 'Something went wrong with the token'}})
    })

    pool.query(queryText: `INSERT INTO users (role,username,password,first_name,last_name,email,phone_number) VALUES ($1,$2,$3,$4,$5,$6,$7)`, (err, results) => {
      if (err){
        throw err
        return res.status(400).json({body: {error:"Server side issue (POST)"}})
      }
      return res.status(201).json(body);
    })
  })
})
```

Bcrypt is a password-hashing function where you input a password string (up to 72 bytes), a numeric cost, and a 16-byte (128-bit) salt value. The bcrypt function uses these inputs to compute a 24-byte (192-bit) hash.

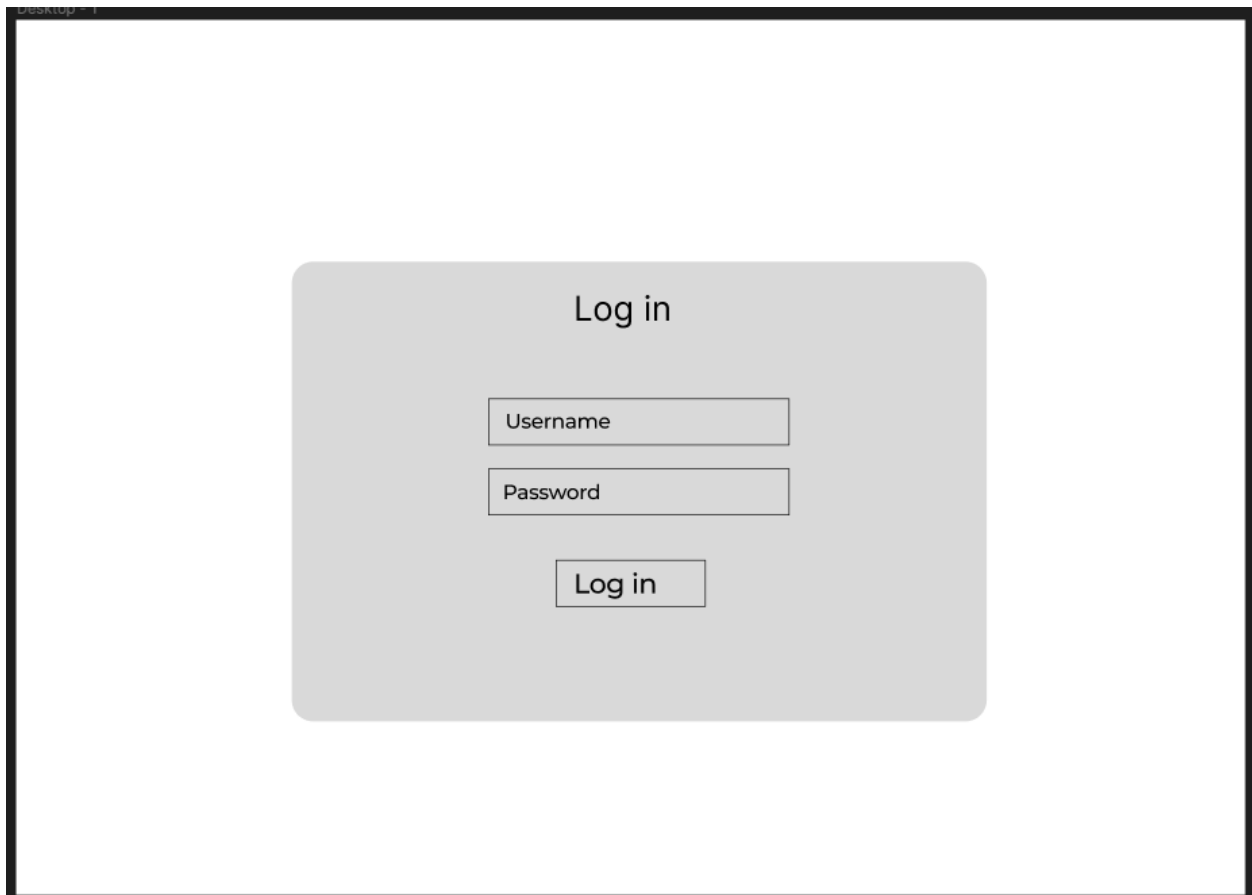
When a user registers we take the user's password and encrypt it before inserting the registration data to our database.

The bcrypt library is also used to compare the password when a user is trying to log in.

```
export function compareLoginDetails (req: any, res: { status: (arg0: number) => { (): any; new(): any; json: { (arg0: { error: string; status: number; }) => {  
  let isUsernameFound = false;  
  pool.query( queryTextOrConfig: `SELECT * FROM users`, {callback: (err:any,result: { rows:any;}) => {  
    if (err){  
      return res.json( param: {error:"Token issue"});  
    }  
    //Loops through all the users and checks if the username and the password match  
    result.rows.forEach((item: { role: string; password: string; username: string; }) => {  
      if (item.Username === req.body.Username) {  
        isUsernameFound = true;  
        bcrypt.compare(req.body.password, item.password, {callback: (err :Error|undefined , result :boolean ) => {  
          if (err) {  
            return res.status( arg0: 500).json( arg0: {error: 'An error occurred while comparing passwords'});  
          }  
          if (result) {  
            return next();  
          } else {  
            return res.status( arg0: 404).json( arg0: {error: 'The password does not match!'});  
          }  
        });  
        req.role = item.role;  
      }  
    });  
  });  
  //If the username has not been found it will return a 404 error  
  if (!isUsernameFound) {  
    return res.status( arg0: 404).json( arg0: {error: 'The username does not match!'});  
  }  
});
```

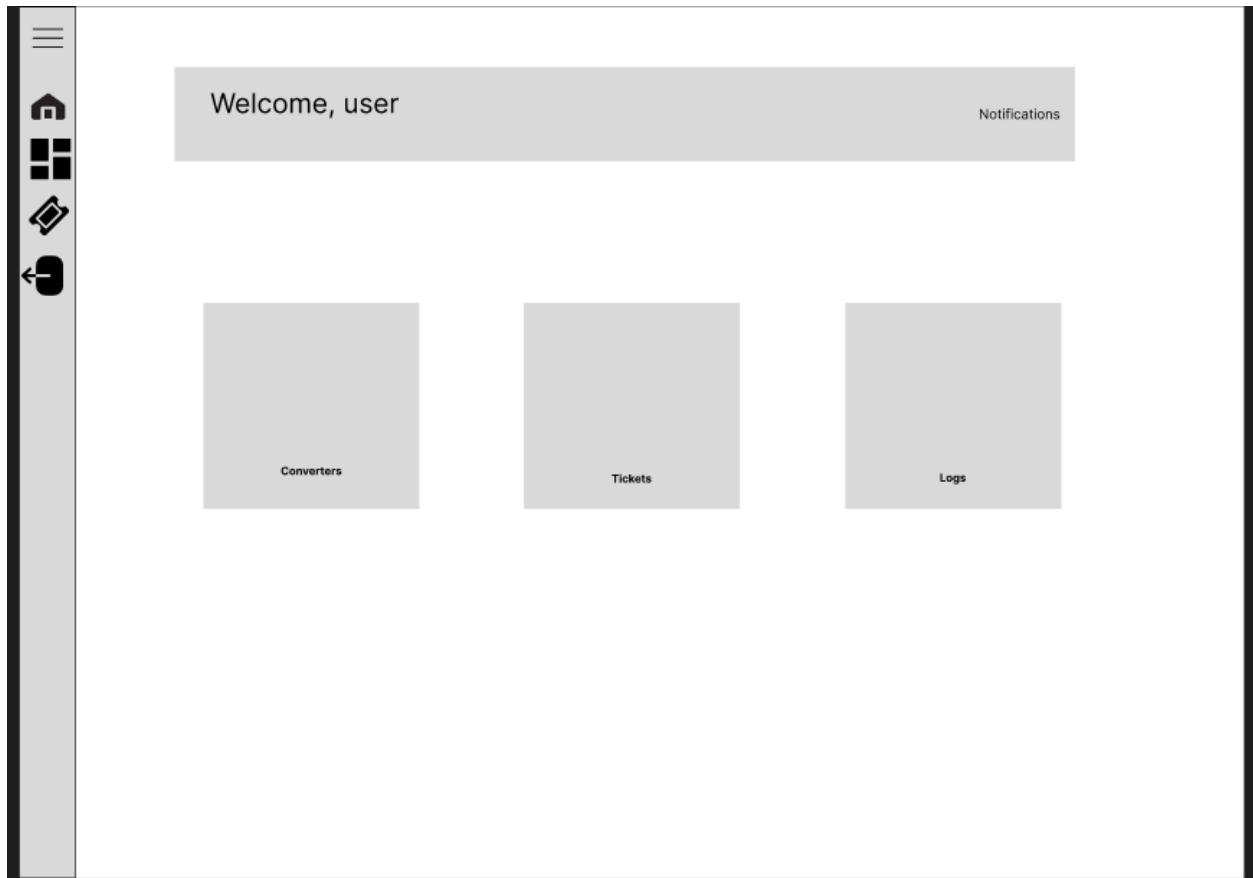
User Interface Design

Customer:



The image shows a wireframe of a login interface. It is contained within a black rectangular border. At the top left of the border, the text "Desktop - 1" is visible. In the center of the interface is a light gray rounded rectangle. Inside this gray box, the text "Log in" is centered at the top. Below it are two text input fields, one labeled "Username" and one labeled "Password", stacked vertically. At the bottom of the gray box is a button labeled "Log in".

First of all, the user needs to log in to access the system. This is possible by entering the credentials(username and password) and by clicking the button “Log in” the user will be retrieved immediately to the home page.

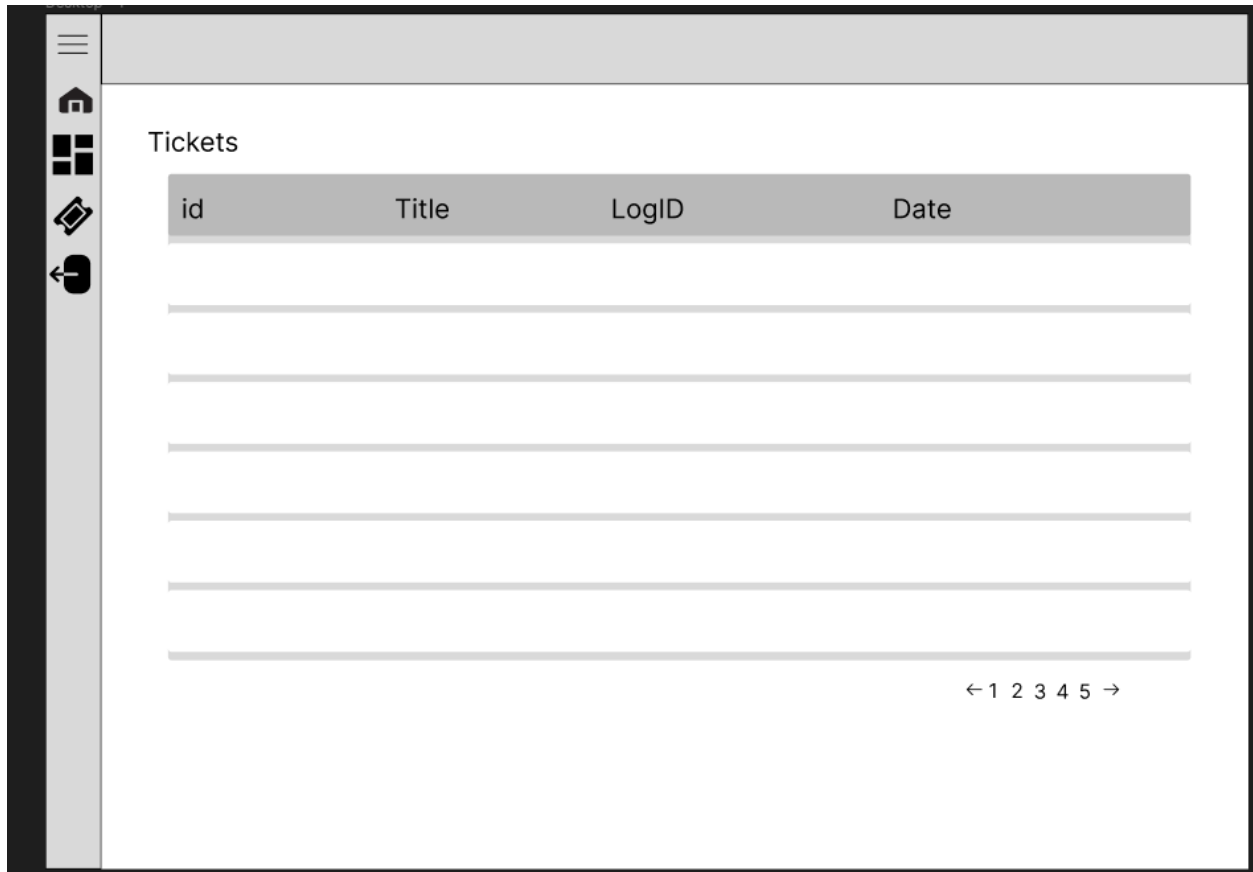


After the user arrives on the home page, they will be able to access their converters, logs and tickets. Furthermore, by clicking on the converter card, the user will be retrieved to the page where the user can check their tickets on this page r status.



Add converter

ID	Owner	Installer	Throughput	Actions

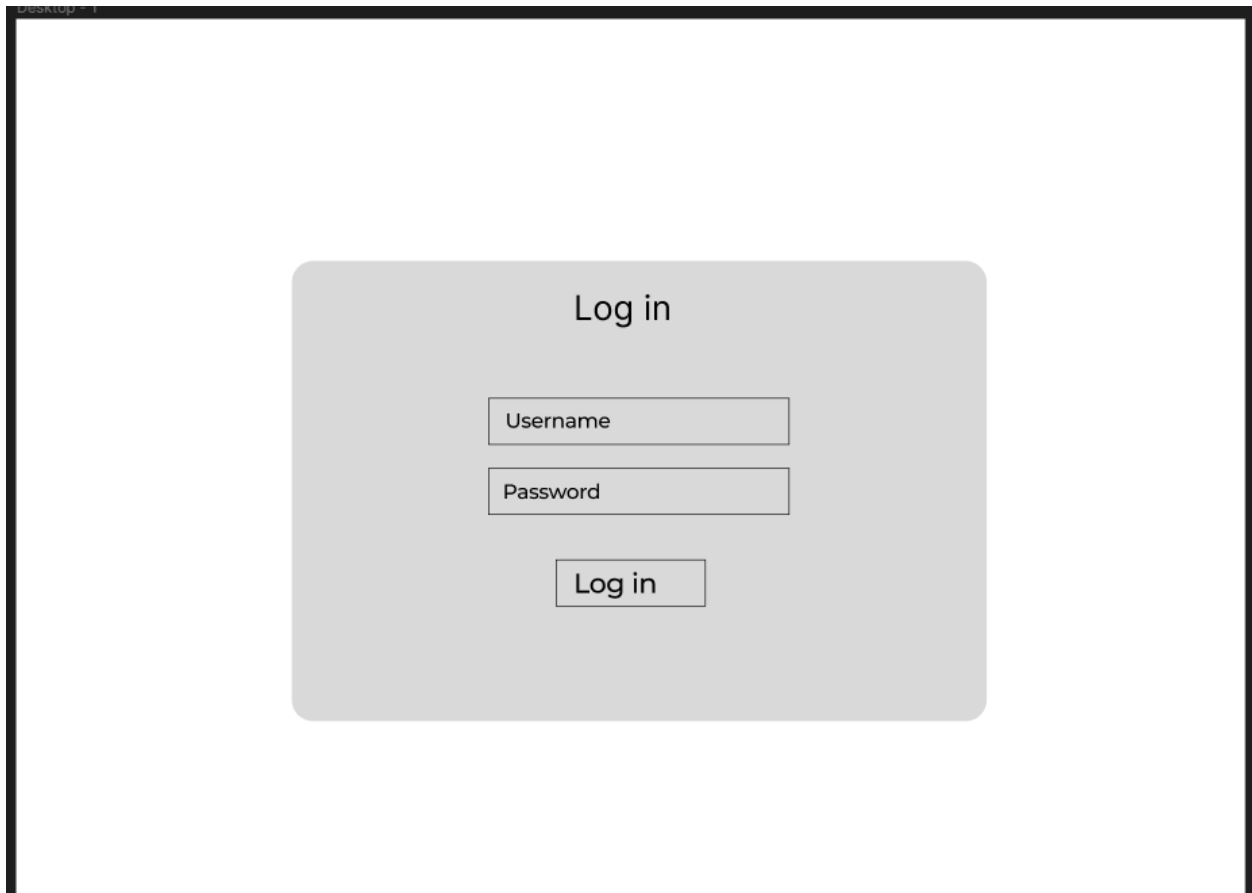


On this page, the user can check their tickets when one of their converters fails. A ticket it's automatically created by the system when something goes wrong and the user it's immediately alerted.

We've also created a navigation bar located on the screen's left side. Having a navigation bar allows the user to navigate on different pages. In the navigation bar, the last icon, right below the tickets icon, is the log-out option, allowing the user to log out from the system. After pressing that button the user will be retrieved to the log-in page, so if the user wants to access the system again, they can log in again by entering their credentials.

Global Admin:

The Global Admin it's the role which can do everything in this system, but they also need to log in first.



Desktop - 1

Log in

Username

Password

Log in

The Global Admin can access the system by entering their credentials, and after they log in they will be retrieved to the home page



On this page the Global Admin can access any page, by clicking on the customer card, the Global Admin will be retrieved on the customer page where the admin it's able to add new customers by clicking on the "Add new customer" button located above the table.

Add customer

ID	Firstname		il	Actions
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>

< 1 2 3 4 5 >

Add

username

password

repeat password

first name

last name

email

phone number

Close Add

In order to add a new customer the admin needs to fill in all the required fields. After a new customer has been added to the system the Admin it's also able to edit the customer details by clicking on the pencil located in between the delete button and the customer converters button.

Add customer

ID	Firstname		ail	Actions
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>
				<div><div></div><div></div><div></div></div>

← 1 2 3 4 5 →

Edit

username

password

repeat password

first name

last name

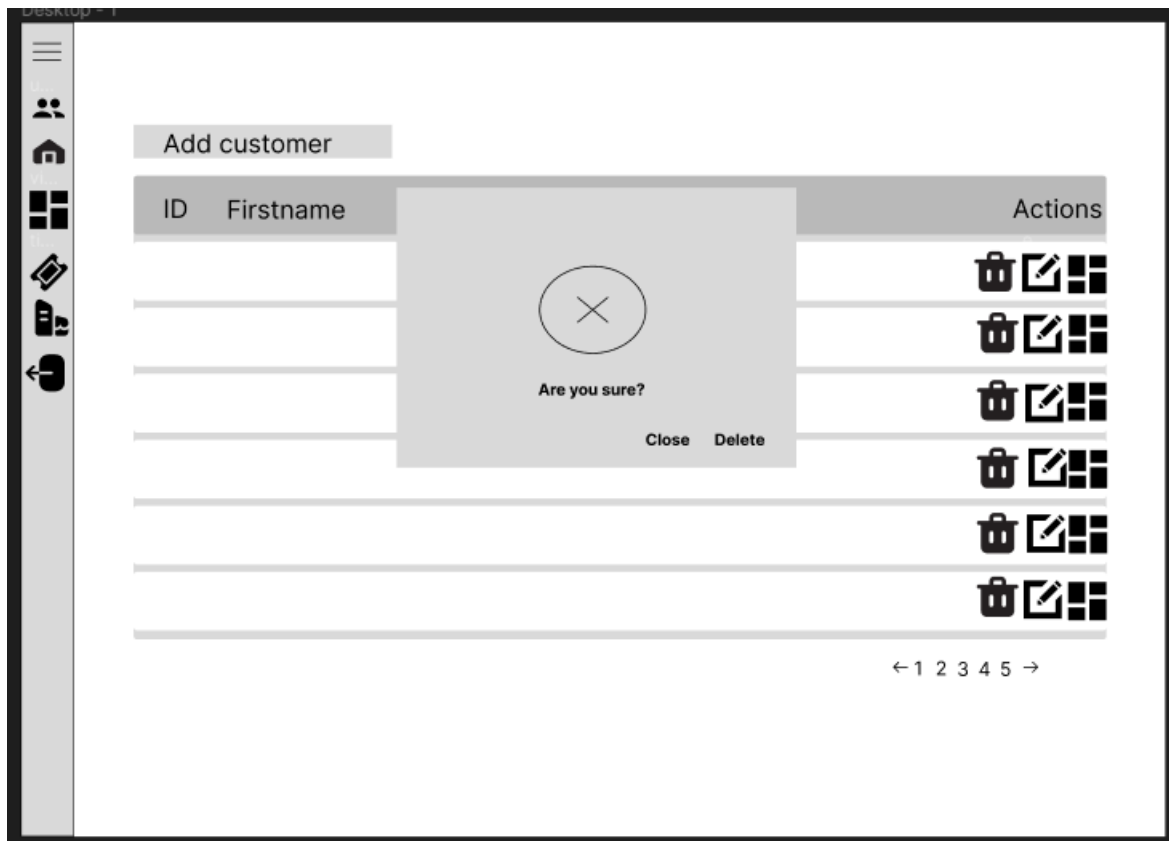
email

phone number

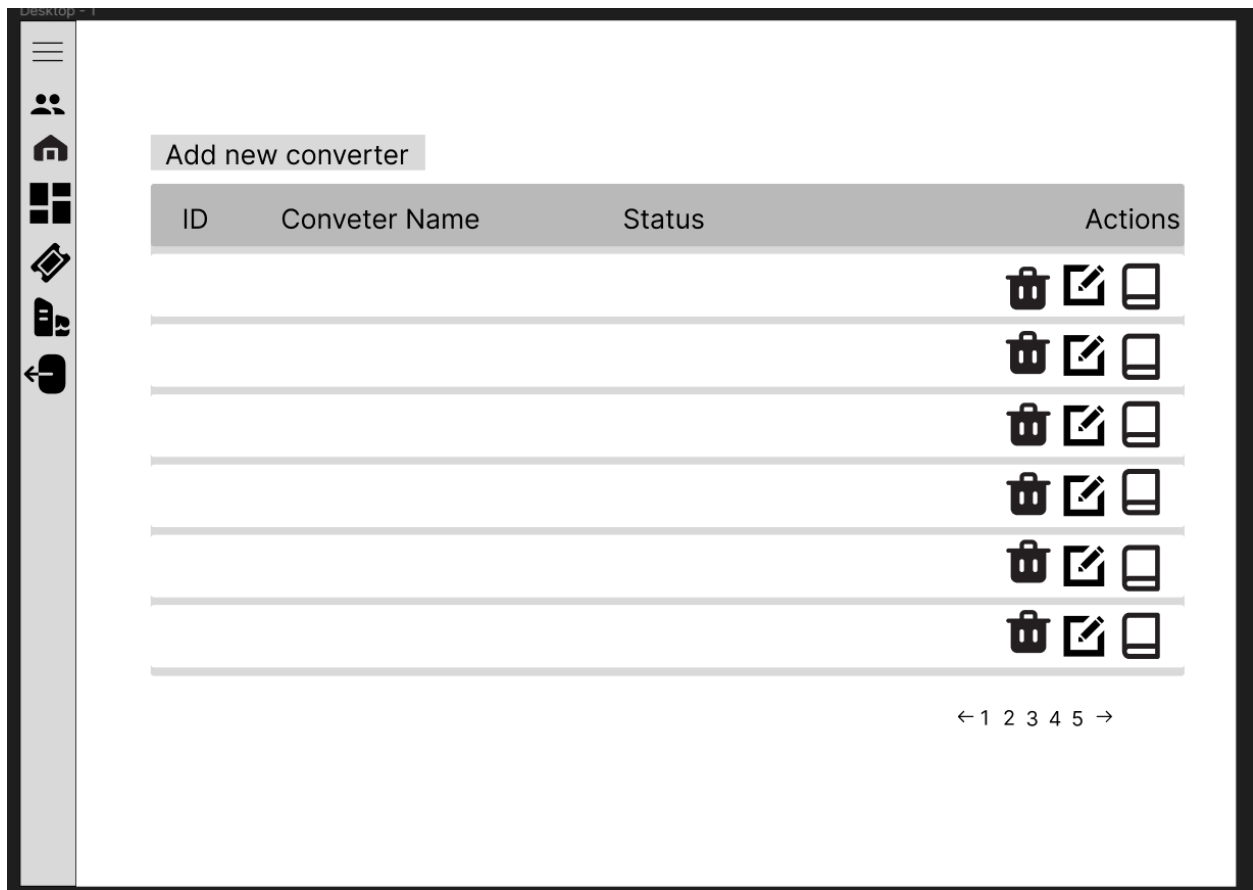
Close

Edit

If there is a customer that needs to be deleted this option it's available by clicking on the trash button, after the Admin clicks on that button a warning will appear which makes sure that you want to delete the customer on purpose.



The last option on this page is to see the converters of a customer, this feature is possible by clicking on the converter button located on the right side of the edit button.



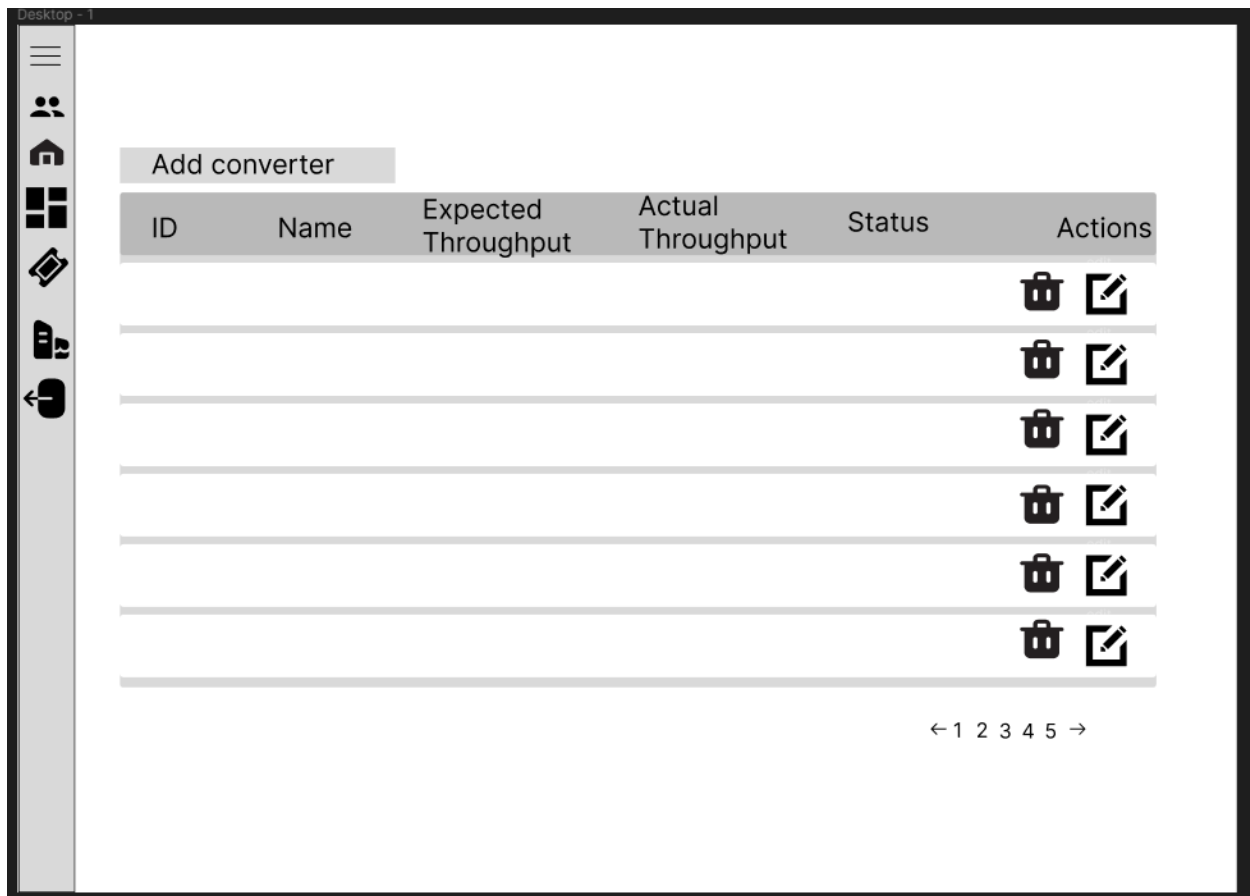
This page will show customer converters together with the id, name and status of a converter. In order to add, edit and delete, the admin must follow the same process as shown on the customer page. Furthermore, the last option on this page is to see the converter's logs. This option is available by clicking on the book icon located on the right side of the edit icon.



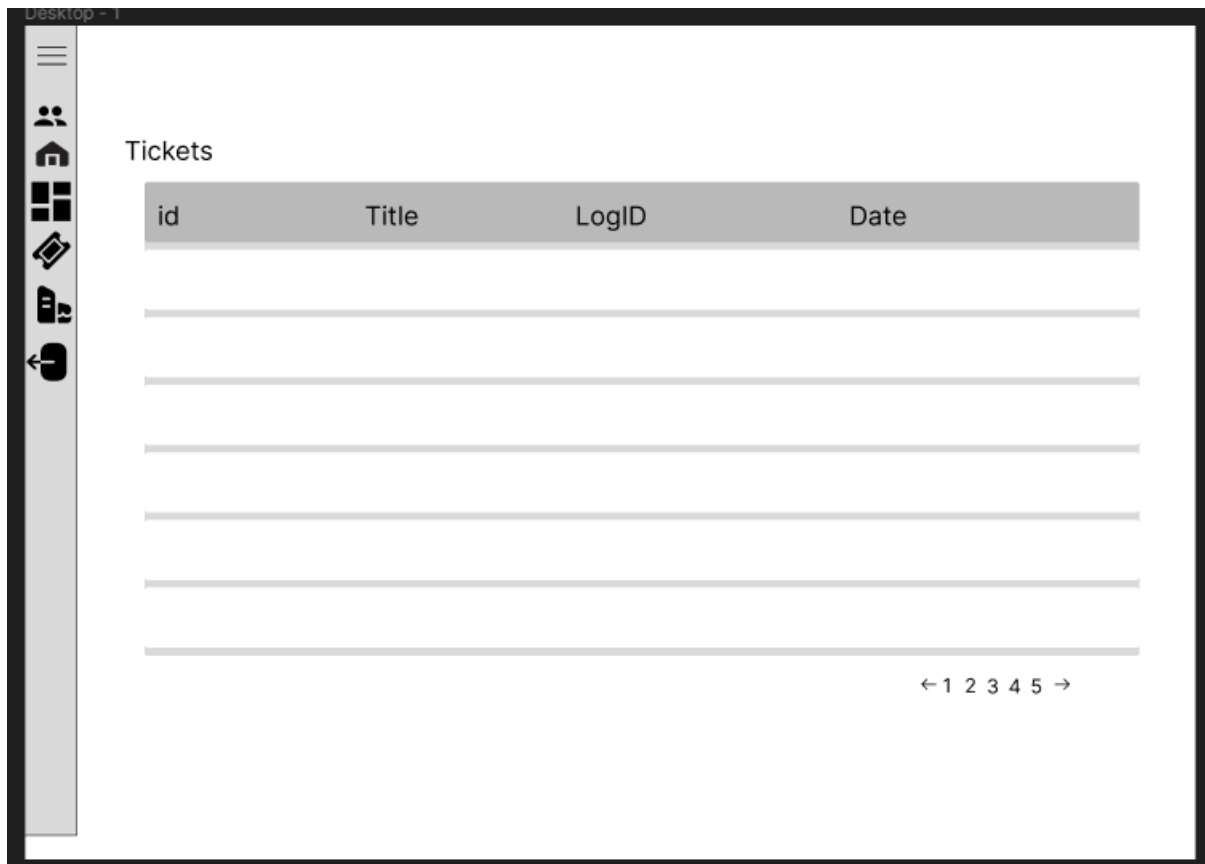
Converters logs

id	Event	Date

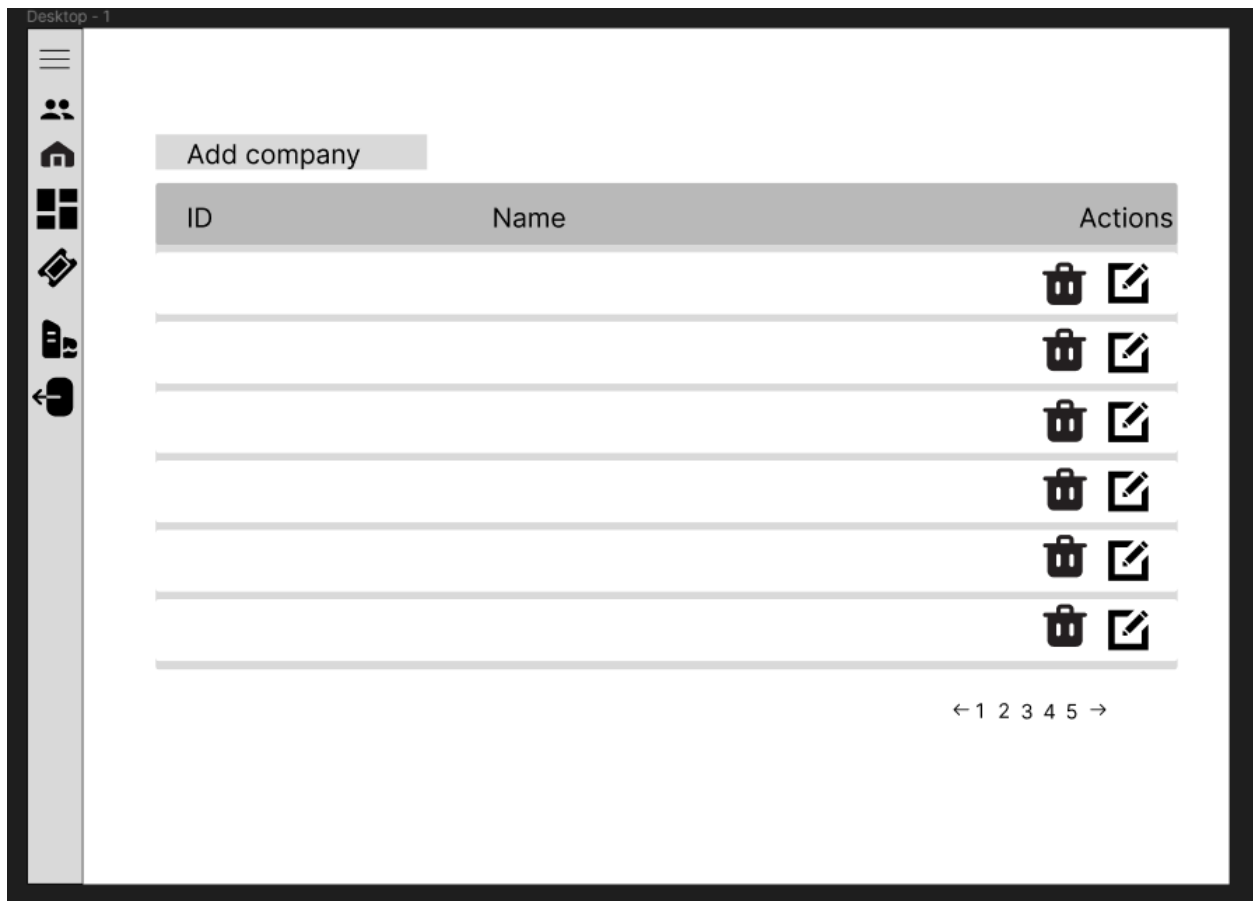
The converters page can be accessed by clicking on the button below the home button located on the navigation bar.



This page shows all existing converters, a new converter can be added by clicking on the “Add converter” button, located on the left top side. A converter can be edited by clicking on the pencil icon located on the right side of each row, but a converter can also be deleted by clicking on the trash icon located on the right side of each row. The forms are displayed in the same way as the customer page shows.



In order to access the tickets page, the admin needs to click on the ticket icon located in the navigation bar right below the converters icon. This page will display all tickets from all customer converters.



In order to access this page, the admin needs to click on the company icon located below the tickets icon. On this page will be displayed all the companies. In order to add a new company the admin needs to click on the “Add Company” button located above the table and a modal will pop up with a form as shown on the customer page, also to edit and delete a company it’s the same process as we’ve shown before. Nevertheless, the admin is able to switch pages if there is a lot of data displayed in the tables.

Company Admin:

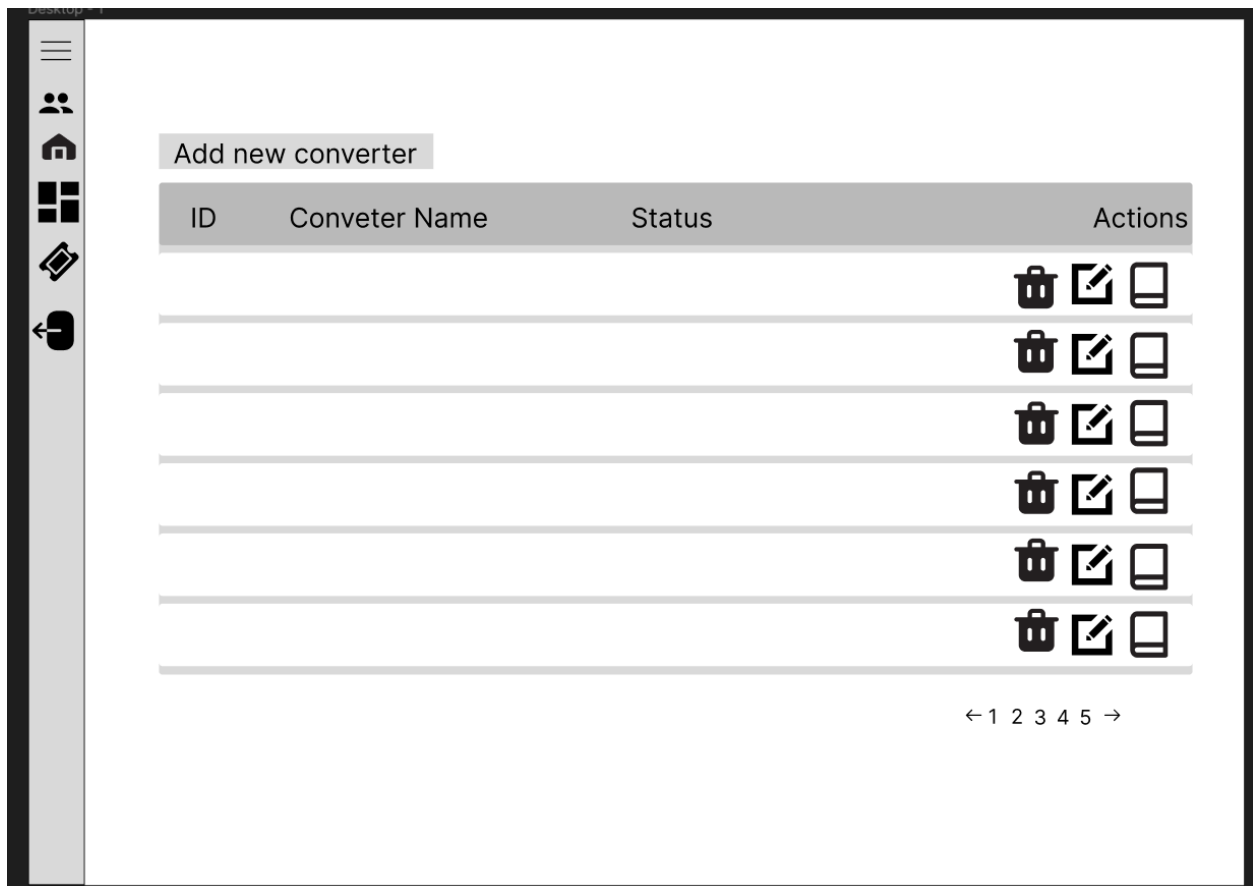


As a company admin, the user will have almost the same functionalities as a global admin. Therefore, in order to make use of this role the user can go to the “Customers” page where the company admin will see his customer, the company admin can add, edit and delete other customers by clicking on the add customer button, pencil icon button and trash icon button. The converters button located on the right side of the edit button will show the converters of a specific customer.

Add customer

ID	Firstname	Lastname	Email	Actions
				<div></div> <div></div> <div></div>
				<div></div> <div></div> <div></div>
				<div></div> <div></div> <div></div>
				<div></div> <div></div> <div></div>
				<div></div> <div></div> <div></div>
				<div></div> <div></div> <div></div>

← 1 2 3 4 5 →















On this page, the company admin can see the converters of a customer. The company admin is also allowed to add, edit and delete a converter, this process works the same we've shown in the Global admin chapter, customer page. Furthermore, by clicking on the book icon the company admin can see the logs of that converter.




Converters logs

id	Event	Date

Add converter					
ID	Name	Expected Throughput	Actual Throughput	Status	Actions
					 
					 
					 
					 
					 
					 
← 1 2 3 4 5 →					

This page can be accessed by clicking on the converter icon located below the home button in the navigation bar. On this page, the company admin is able to see all converters of all users, and the admin it's allowed to add, edit and delete a converter.

Desktop - 1



ID	Converter_id	Issue	LogId	Date

This page displays all the tickets created and can be accessed by the company admin by clicking on the ticket icon located below the ticket icon in the navigation bar.