Stock Management System

Evidence for realization

Applied Computer Science

David Silva Troya

THOMAS MORE

Table of Contents

INTRODUCTION

The internship at CERCUITS provided an enriching experience for acquiring practical knowledge and skills in the field. In this document I present a comprehensive overview of the completed assignment at CERcuits, which involved the analysis and development of a Stock Management System.

The assignment was undertaken during an internship period spanning from February 25th to May 26th, 2023. Saving all the documentation of the assignment in the Confluence page of CERcuits, the space for all the IT information with a page called "Stock Management System".

In the next pages are covered a short overview of CERcuits and various aspects form the assignment such as the plan of actions, project objectives, roadmap planning, and the utilization of Agile methodology.

Additionally, it delves into the initiation phase, encompassing the selection of application software, the design of entity relationship diagrams, class diagrams, setup of a virtual machine.

Continuing with the evidence of the realization phase, which explores the structure of files, web application security, database management, table creation constraints, and data managing from new tables.

Finally, showing screenshots of the demonstration sessions, specific the first and the last demo, to have a better idea of how the software improved not only in the logic but also in the User Interface, based int the feedback from every demonstration meeting.

1 INTERNSHIP AT CERCUITS

1.1 CERcuits

CERcuits is a company that specializes in ceramic PCB and substrates for prototypes & small series. (CERcuits, 2023)

CERcuits specializes in manufacturing ceramic PCBs with superior temperature resistance, dimensional stability, and resistance to corrosion and chemicals. They cater specifically to low-volume and high-mix projects, utilizing rapid prototyping technologies and eliminating tooling costs to reduce lead times and production expenses. CERcuits aims to simplify and accelerate the design, testing, and validation of ceramic PCB materials, offering convenient online ordering and providing design and production support when required. (CERcuits, 2023)

1.2 Assigment

The company already has an inventory system in place but is looking to have better control over the inventory with personalized features that meet the company's specific needs.

Here are three important points given by the company about the project, related with what is expected from the software:

- ✓ An Automated Stock Management system to ensure greater control and accuracy in inventory management.
- ✓ The student is tasked with conducting a comprehensive analysis using the "should, could, nice to have" framework.
- ✓ Additionally, an easy-to-use system is needed for scanning materials on Android smartphones and computers to promptly update the database when a product is retrieved using QR codes.

It is important to mention that the requirements could be improve with or after the analysis, and here are the most important requirements for these project:

- ✓ Create a software in python for the Stock Management System.
- ✓ Software shall run in a virtual machine(VM) with Linux OS of the local server.
- ✓ Database for the software shall run in the VM of the local server.
- ✓ Software shall run in Desktop and Smartphones.
- ✓ Documentation of the software and related information.

To have access to the documentation of the company and also to can make the documentation, the company gave me an email, with this email I was able to use all the necessary Atlassian software.

2   INTERNSHIP ASSIGNMENT

2.1   Plan of actions

With the information that the company provided and what was expected, it was necessary to create a plan of actions with the overview of the project. This plan was helpful to create objectives, goals and tasks to be done, taking into account the review and approval from my mentor.

The plan of actions had 5 steps, starting with the analysis and finishing with the execution of task to complete the project:

1. Analysis of the requirements for the internship assignment or project.
2. Creation of the objectives to set a project planning and the time each objective will be completed.
3. Review, corrections and approval of the objectives and project plan.
4. Creation of task to complete each objective in the time proposed.
5. Execution of tasks to complete all the objectives.

2.2   Project Objectives

To create a Project plan was necessary to first make a overview of the project requirements, and then with this create a number of specific objectives with a general objective.

General Objective:

- Develop a Stock Management System with automatic notifications for a better control of the company warehouse

Specific objectives:

- Select the frameworks to create the Stock Management System using Python as Program Language
- Select the Relational Database to be used in the Ubuntu Virtual Machine on the Server.
- Design the Relational Database for the Stock Management System.
- Design the Class Diagrams for the Stock Management System.
- Develop a web-browser version of the Stock Management System.
- Develop a mobile version of the Stock Management System.
- Complete the provided requirements from the company to the System.
- Provide Documentation of the Developed Stock Management System.

2.3   Roadmap Planning

With all the objectives defined, it was possible to make goals to be achieved and set them in a specific period of time to be completed, these goals were also split into task for Jira.

With this roadmap was also possible to inform important events coming, like the meetings at the University.

Roadmap Planning
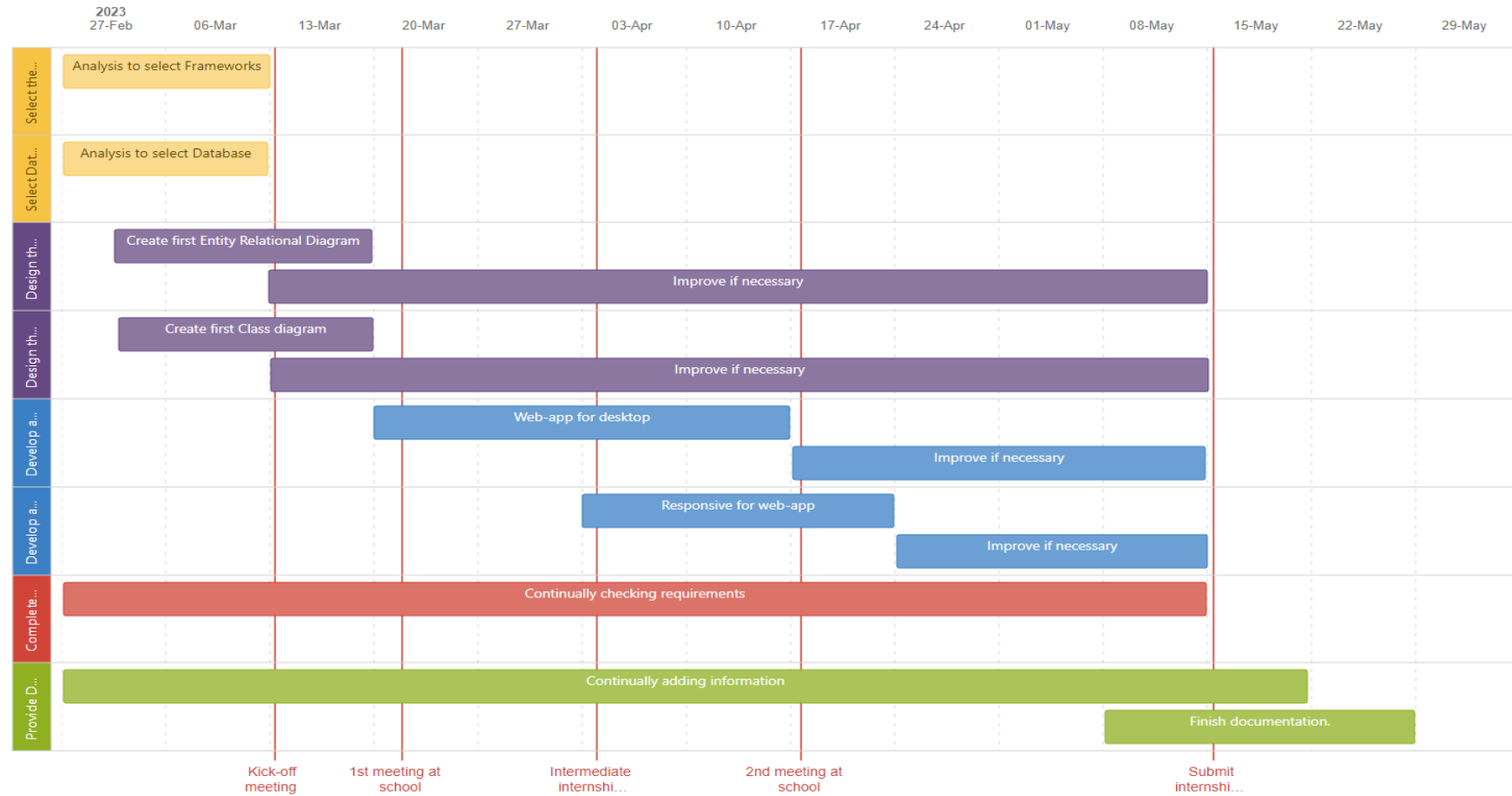
2.4   Agile Method

With the goals set in specific periods, it was possible to create small task to reach these goals. The Agile Method was using in Jira, with only 3 columns to know what tasks, logs or issues where still need to be done, which one was in progress and which one was already completed.

Because Atlassian has Jira, Confluence and Bitbucket, it was possible to connect the Jira dashboard with the page were all the documentation was saved and also in Bitbucket repository with the project.

In the next image is possible to see an example of how the dashboard was looking between the weeks 6 and 9. STMA is just the abbreviation of the first 2 letters in the words Stock and Management, created by the Project Manager.
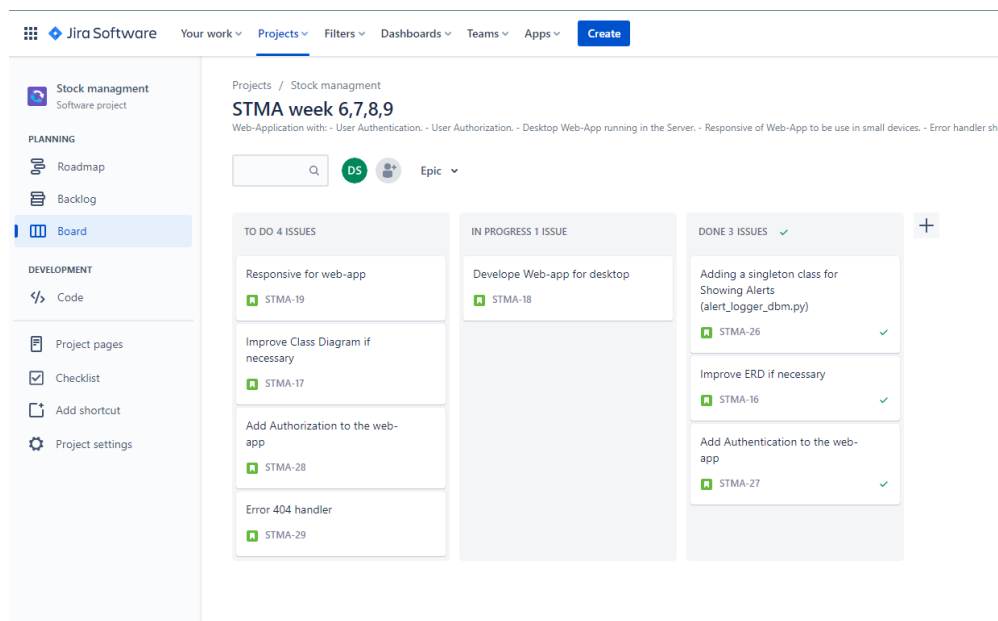
2.5   Project Status

The project is been testing by the company and I am still adding all the information for the documentation about all the last changes.

The testing of the methods are not completed, but mostly the web-application works without errors. However the users can always make things that were out of the programmer mind and new bugs could be find.

The web-application is working with flask which is a micro-framework that allows to have freedom in the way to make a project structure, but it could be a good idea to check the possibility to move to a more structured framework as it is Django. Since both are based in python, it should not be a big problem to make the change.

3   INITIATION PHASE

This phase was during the first 3 weeks of the internship, this phase was to know better the company, the team, the assignment, the place where I would be working and the tools from the company I could use.

With all the information of the project, gave by the company, I started with an analysis to provide a solution that fits with the requirements and what was expected. In total 3 meeting were done, the first one to get all the information, the second one give me analysis and the third one for the adjusts based in feedback from the second meeting and to have the approval of the objectives and planning.

3.1   Application Software selected

The Web-Application was the best option to can cover the use of smartphones and computers. The web application works like a website but with a strong program language in the backend, in this case python. Between all the option for a framework of python to create a web application, Flask was the option selected because it allows to start small and slowly add more features, being possible to have an API if it is necessary in the future. (Pallets Projects, 2023)

Still, all the information of what software and the different possibilities, together with the reasons of the selection, was saved in Confluence.

In the next picture is possible to see a part of the documentation of the "why" a web-application is the best option, based on the requirements. But it is also possible to see, in the left menu, all the other tabs with more information of frameworks and the project itself.

3.2   Entity Relationship Diagram designed

An Entity Relationship Diagram or ERD, is very useful to know how the data will be store and allowing a faster creation and maintenance. As it was shown in the roadmap planning, this diagram was always being improved based on new features or improving the tables as the web application was growing. (Visual Paradigm, 2023)

In the next picture is possible to see the documentation about the ERD for the Web Application, this diagram shows the "main tables", called like that since these tables are the one with the information to create new tables, using dynamic classes in python and Declarative Mapping Styles of SQLAlchemy. (python, 2023) (SQLAlchemy, 2023)
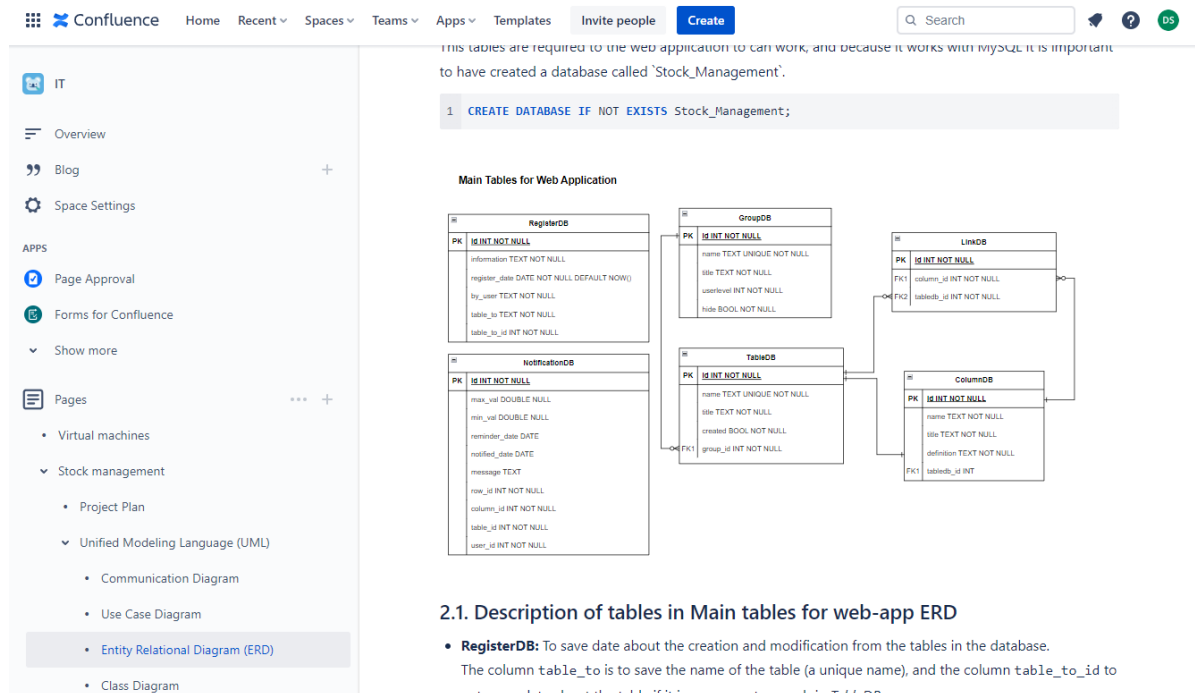
For a better understanding of the ERD in the picture, it was called the "Main Tables" ERD, this tables only save data to create new tables in a dynamic way. There is table to register what user made a creation or modification, another table to save information related to notifications for the users, and all the other tables are related to the information to create the new tables, being possible to make groups of tables, set a name for the database an a name that the user understand, the columns and if there is a connection or link between tables.

Another ERDs were created, being "Users" ERD made to save information about users like username, name, password, address, etc. And another ERD called "Base Substrates" for the logic related to store that information, but this one being created in a dynamic way, storing the data about the creation in the main tables of the web application.

In fact, the web application has the feature of creating multiple tables, having groups that users can see or not depending of the role. This taking in count the necessity of avoiding to repeat names in the database, for tables and columns within the table.

The web application also allows modification of the tables like the groups where they belong, the table name, columns name, number of columns, type of columns and the connections between tables, or in other words a column with a foreign key pointing to another table.

3.3   Class Diagram

A class diagram is necessary to have a better understanding of the logic behind the program, to can create a better abstraction and keep a cleaner code. Allowing a faster maintenance of the software since another developer will understand better the web application without necessity of going to the code. (IBM, 2023)

The first diagram allows me to develop the code in a faster way, at the beginning is easy to remember what class does what, but once that more features are added or removed, it is difficult to remember what class has the necessary method. More when there is heritance and relations in the classes.
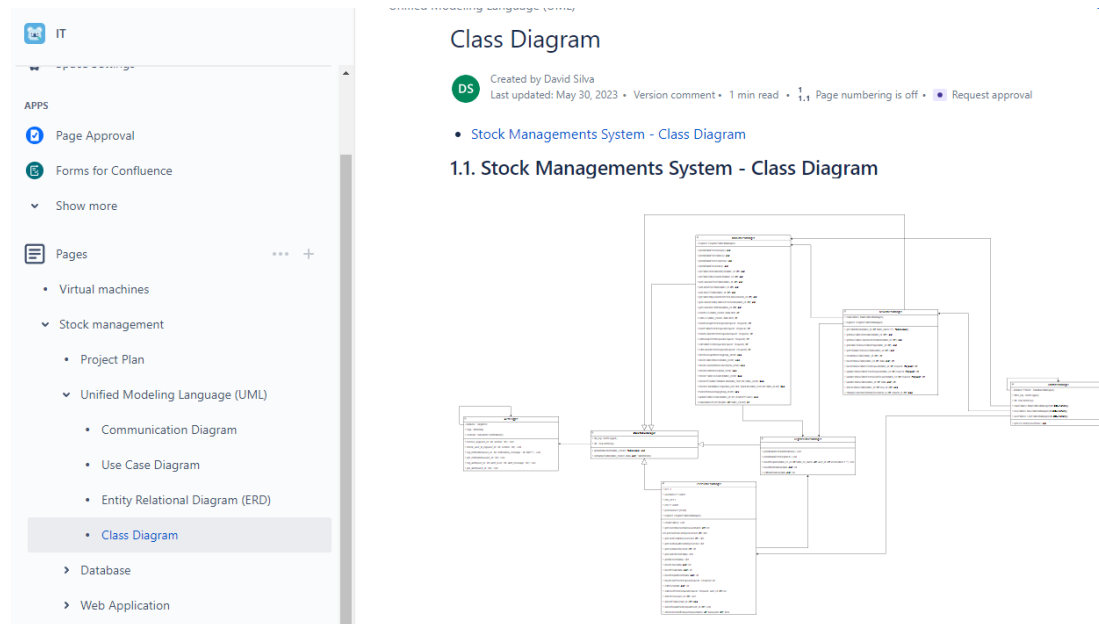
It is important to mention that in the previous picture is shown the last version of the class diagram, this diagram changed along the time with the project, to be adapted based in the libraries used. The biggest change happened when the library Flask_msqldb was changed for Flask-SQLAlchemy.

At the beginning Flask_msqldb was perfect for the creation of tables in a dynamic way, but when the project started to grow and more features were need, Flask-SQLAlchemy was a better option, allowing to use less code than with the previous library. (Ferland, 2023) (Pallets Projects, 2023)

3.4   Virtual Machine setup

The Virtual Machine works with Ubuntu, this virtual machine is working in the local server of the company. This means that the only way to use the web application is if the user is being connected to the local network of the company.

I will not going into details of all the configuration in the local server to allows the use of the virtual machine in an specific IP port, since this is private information of the company. But I can show some information about the configuration of the web application since these only works the first time that the libraries are installed.

But part of the evidence of what was done can be seeing in the next picture, where are located the headers of the titles from the page of "ubuntu configuration". The virtual environment works with Conda, Gunicorn is to deploy the flask application or web

application and the System Service for the web application is a background processes in Ubuntu to run the web application every time the Virtual machine starts. (Conda, 2023) (gunicorn, 2023) (Rendek, 2023)
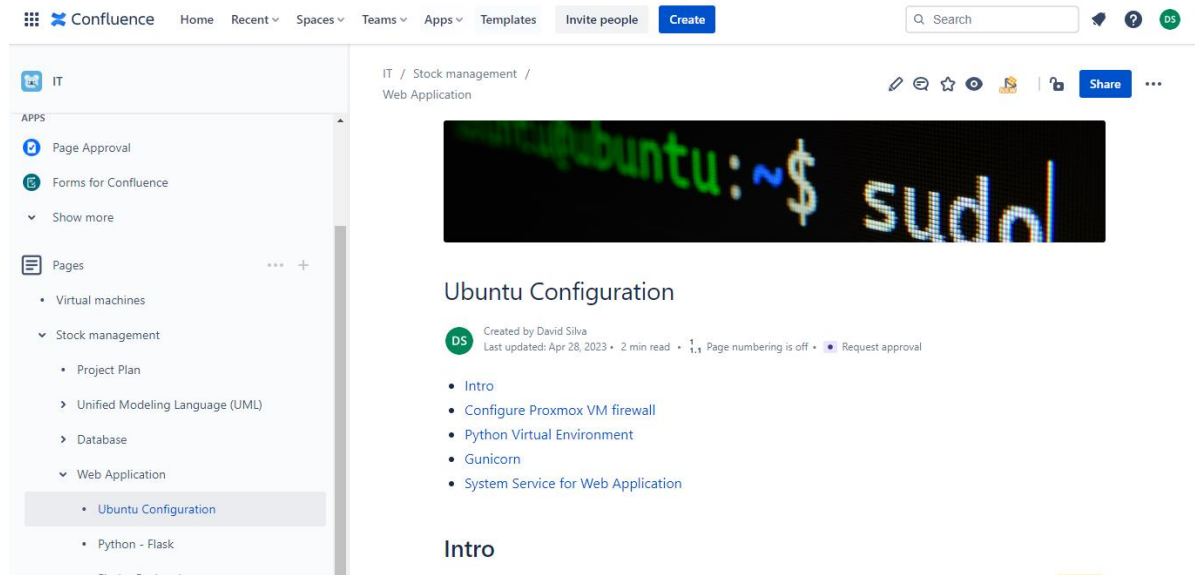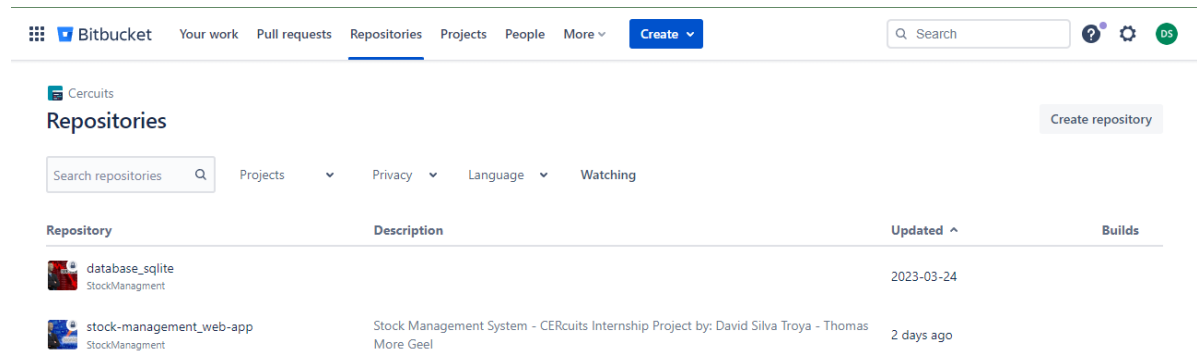
About the relational database, MySQL, has an easy way to be installed in ubuntu when it is known how to use the terminal. And all the information of the configuration was written in Confluence. In the next picture is possible to see documentation about the installation and configuration of MySQL in Ubuntu. (Ubuntu, 2023)
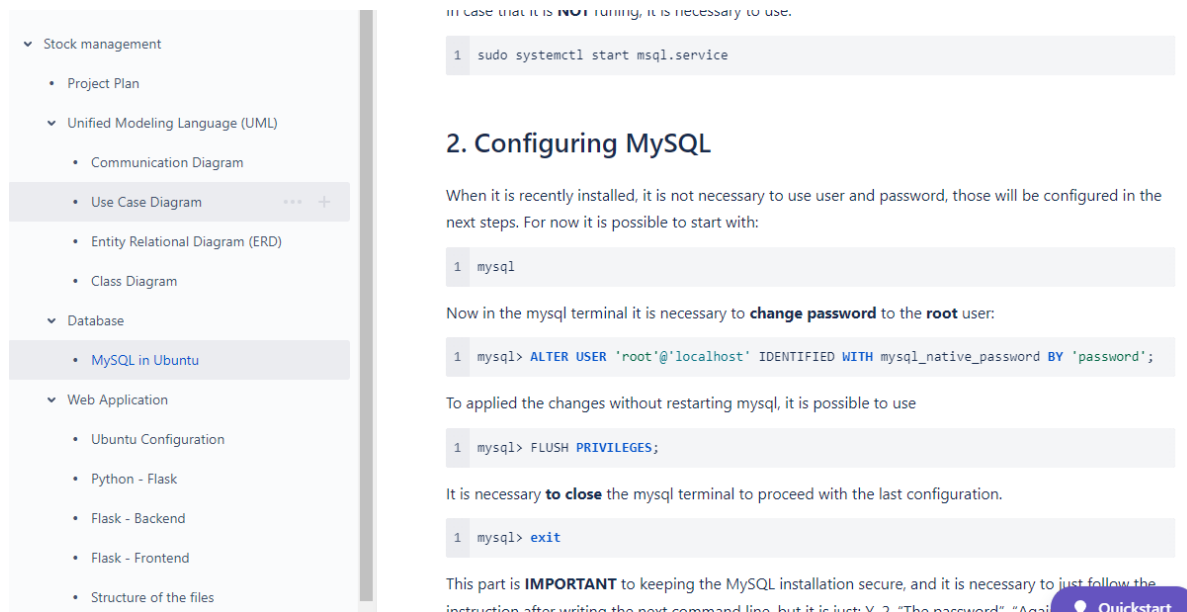
3.5   Atlassian software

As it has being mentioned before, the company has the documentation saved in Confluence, for the tracking of tasks is used Jira, and for the code storage is used Bitbucket. (ATLASSIAN, 2023)

Because I already cover how the use of Jira and Confluence is used within the project, now I will be concentrated in Bitbucket. For developing the software, the web application, is necessary to use a software of version control, and that where Git is used, together with Bitbucket, being this last one the one which have repositories of the software code. (git, 2023) (ATLASSIAN, 2023)
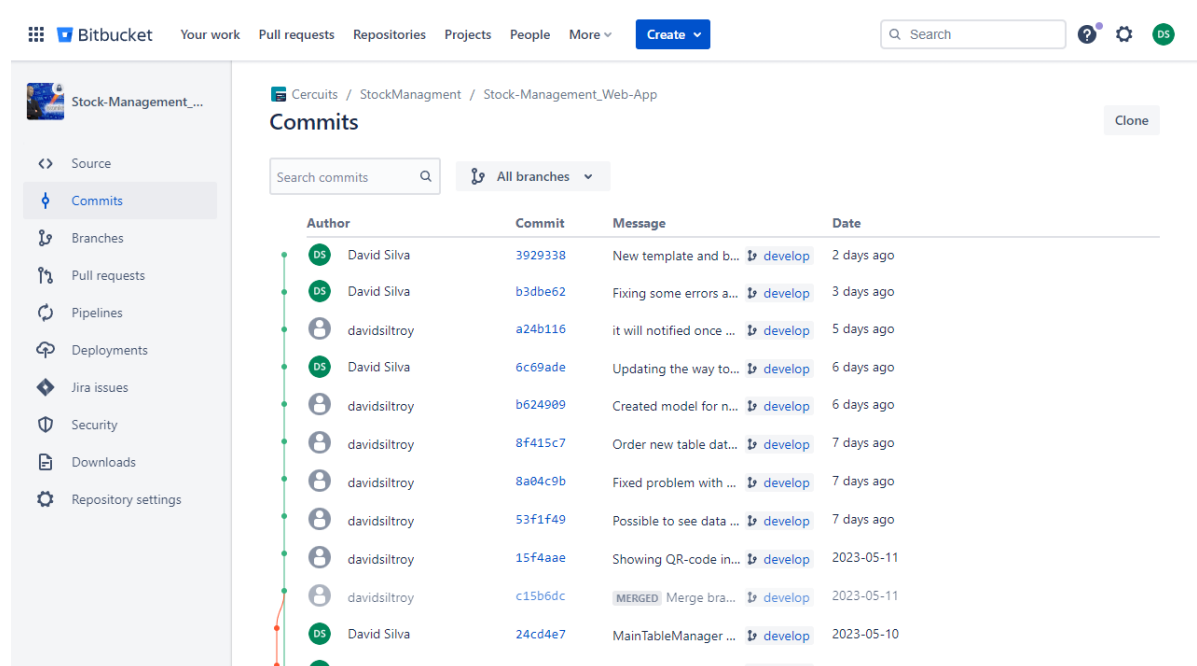
Something important to mention about the initiation phase, because it was also to make quick tests of how a framework, library or database would work, not only in the PC for developing but also in the Virtual Machine of the Server, more than one repository was created in Bitbucket.

In this case 2 repositories related with the web application, in the original requirements, SQLite was mentioned to be use as relational database. But when it was tested and together with the analysis, it results that SQLite is not a good option for the kind of application as the Stock Management System. (SQLite, 2023)

4 REALIZATION PHASE

The realization phase started fast and well, making me thing I will finish the project before the expected time, I was wrong. When the first demo was created, it was already possible to see that some important features were missing, some important options to show in the user interface or sometimes too much information.

The documentation, related with the frameworks and diagrams, was modified multiple times, ERD and class diagram were improve, code was improve and even a complete library was changed because it ended up being a better option than the current library.

Multiple demos were made after every 2 weeks to show the results, what was done, what was changed, the reason of the changes, problems discovered, feedback to improve the application, new features to be add, etc.

4.1 Structure of files

The file structure was changing over the time, starting with the simple structure that every small project of Flask has, to then start adding for blueprints, controller and model. Mostly following structures of large files that the Flask documentation provides and a tutorial from Digital Ocean (Pallets Projects, 2023) (Digital Ocrean, 2023).

The end result of the structure of files from the project is also in the documentation of CERcuits, this is an example of an overview of all the folders and some requirements of libraries and configurations script.

```
1   ...
2   .
3   └── web-app
4       ├── requirements.txt
5       ├── config.py
6       ├── stock_management_system
7       │   ├── blueprints
8       │   ├── controller
9       │   ├── models
10      │   ├── static
11      │   ├── templates
12      │   ├── automation.py
13      │   ├── decorators.py
14      │   ├── extensions.py
15      │   └── __init__.py
16      └── tests
17  ...
```

In this way it is easier to understand where a file is, if we compare to how all the folders and files look in VScode. The web application became a large project, so it is very useful to have this folders structure in the documentation.
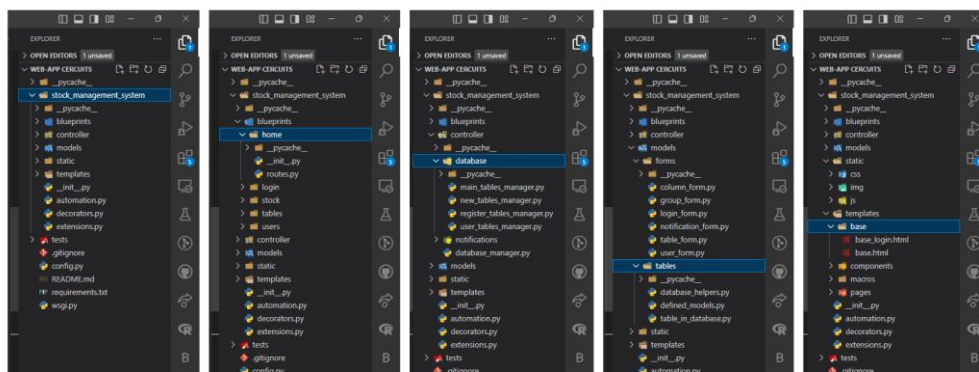
4.2   Security for the Web Application

In the Operation System is possible to set variables, all the important information in those variable to avoid share them when the code is being saved in the cloud, mostly if the code will be save in a public repository.

```python
config.py > ...
    ...
1   import os
2
3   basedir = os.path.abspath(os.path.dirname(__file__))
4
5
    ...
6   class Config:
7
8       ''' This values should be set in the enviorament to can get them and not have them in the code'''
9       SECRET_KEY = os.environ.get('SECRET_KEY')
10      MYSQL_HOST = os.environ.get('MYSQL_HOST')
11      MYSQL_USER = os.environ.get('MYSQL_USER')
12      MYSQL_PASSWORD = os.environ.get('MYSQL_PASSWORD')
13      MYSQL_DB = os.environ.get('MYSQL_DB')
14
15      ...
16      To set the variables in the system go to the comand line and:
17
18      setx SECRET_KEY "this is an example"
19      setx MYSQL_HOST "localhostmaybe"
20      setx MYSQL_USER "superuserher"
21      setx MYSQL_PASSWORD "this is not the password 123"
22      setx MYSQL_DB "name_of_the_datbase"
23
24      ...
25
```

Then to set the configuration is real simple and now safety, in the script where the application is initialized it is possible to set the configuration, extensions, blueprints, global functions, etc.

```python
stock_management_system > __init__.py > create_app
    You, 1 second ago | 4 authors (David Silva Troya and others)
1   from flask import Flask, redirect, url_for
2   import builtins
3   from stock_management_system.automation import StockAutomation
4   from stock_management_system.controller.database_manager import DatabaseManager
5   from config import Config
6   import threading
7
8   #run the app once the script is run
9   def create_app():
10
11      #----> configuring web-app with ist name and folders
12      app = Flask(
13      "Stock Management" ,
14      static_folder='stock_management_system/static',
15      template_folder='stock_management_system/templates'
16      )
17
18      #----> setting web-app configuration
19      app.config.from_object(Config)        David Silva, 2 months ago • Setting a new folder structure …
20      app.jinja_env.globals.update(list=builtins.list)
21      # app.wsgi_app = ProxyFix(app.wsgi_app, x_proto=1)
22
23      #----> Initialize extensions
24      from stock_management_system.extensions import db
25      db.init_app(app)
26      app.config["MYSQL"] = db
27 >    with app.app_context(): ...
30
31      from stock_management_system.extensions import qrcode
32      qrcode.init_app(app)
33
34      from stock_management_system.extensions import login_manager
```

Because for CERcuits it is important to keep private their information, the web application needs to have a user login with users role in order to allow or not the user

to create, read, update or delete. This is also known as CRUD. (Mozilla Developer Network, 2023)

The first feature related with the user access, was the authentication, this was done by using the library *flask_login*, same that allows to add decorators in the routes which allows access only to users with an account. But also allows to add more information to the user itself, in this case information related with the role. (Frazier, 2023)

This library was used in almost all the routes, located in the folder "blueprints" and using the decorators, with the decorator `@login_required` is possible to send the user to the login page in case they don't have a authenticated account.

```
13
14    @home_views.route('')
15    @home_views.route('/')
16    @home_views.route('/index')
17    @home_views.route('/home')
18    @login_required
19    def home():
20        '''
```

And as I mentioned before, it is also possible to add new attributes and with this attributes was possible to create new decorators. In this case, to be sure the user has a role, which it is necessary to be able of making changes, the decorator `role_required` was created.

```
stock_management_system > decorators.py > ...
       You, 1 second ago | 2 authors (David Silva Troya and others)
1      from functools import wraps
2      from flask import redirect, url_for
3      from flask_login import current_user
4
5      #custom decorators        You, 1 second ago • Uncommitted changes
6
7      def role_required(role):
8          def decorator(func):
9              @wraps(func)
10             def wrapper(*args, **kwargs):
11
12                 if not current_user.id or not int(current_user.role_id) < role:
13                     return redirect(url_for('home_views.home'))
14                 return func(*args, **kwargs)
15             return wrapper
16         return decorator
```

So now there was a new security layer to prevent data is modified when the user has not the necessary role. The role ID is saved in the program, with more information about the role, like the CRUD permission, and a name that the user can understand.

```
22
23
24     ''' Home in Database Management '''
25     #Home of database
26     @tables_views.route('', methods=['GET'])
27     @role_required(2)
28     @login_required
29     def home():
30         '''
```

One advantage of the library `flask_login` is the possibility of use it in the render of the html, since Flask use Jinja to render pages, the way to code is quite similar to python. In this case allowing to add another level of security in the web application, avoiding the rendering of options that the user has not access. (Pallets Projects, 2023)

```
166
167    <section id="gtc_data_option_macro">
168        <!--
169            Macro for Group, Tables or Column Option in Navbar
170        -->
171        {% macro groups_data_navbar() %}
172            </ul>
173                <h6 class="sidebar-heading d-flex justify-content-between align-items-center px-3 mt-4 mb-2 text-muted">
174                    <span class="text-white">All Groups of Tables</span>
175                    {% if current_user.role_id|int < 2 %}        David Silva Troya, 4 weeks ago • Adding notifications and alets
176                        <a
177                            class="link-secondary"
178                            href="{{ url_for('tables_views.create', option='group' ) }}"
179                            aria-label="Add a new report"
180                            data-bs-container="body"
181                            data-bs-toggle="popover"
182                            data-bs-content="Create a new Group of tables"
183                            data-bs-trigger="hover focus"
184                            data-bs-placement="Right"
185                        >
186                            {{webapp_icon('add-in-circle',24)}}
187                        </a>
188                    {% endif %}
189                </h6>
190
191                <div class="accordion accordion-flush bg-transparent" id="accordion-tables-group-data">
192                    {% for g_id in cercuitsDB["groups"] %}
193                        {% set group = {'id': g_id} %}
194                        {% set _ = group.update(cercuitsDB["groups"][g_id]) %}
195                        {% if current_user.role_id is defined %}
```

And because more attributes could be set in the user class once the user is authenticated, it was also possible to get the username at the moment of the rendering, without extra code in the route.

```
<a class="nav-link" data-bs-toggle="collapse" href="#collapseUserOptions" role="button" a
    <div class="d-flex justify-content-center flex-nowrap  align-items-center my-1">
        <h5 class=" d-none d-md-block mh-100 h-100 text-truncate text-start ms-1">
            {{
                webapp_icon('person-circle',26)
            }}
            {% if current_user.username is defined  %}
                {{current_user.username}}
            {% else %}
                Username...
            {% endif %}
        </h5>
        <h5 class=" d-md-none mh-100 h-100 text-truncate text-start ms-1">
            {{
                webapp_icon('person-circle',26)
            }}
        </h5>
```

When a user was created in the database, the password was encrypted with a library of python called `Werkzeug`, so even with the access to the database is not possible to see what password each user has. (Pallters Projects, 2023)

```python
def insertUser(self,data:dict):
    '''
    Save a new user in the database, encrypting the password.
    '''
    try:
        new_user = self.setDataInTable(User(), data)
        new_user.password = generate_password_hash(new_user.password)
        self.db.session.add(new_user)
        self.db.session.commit()
        #return id from group created in database
        return new_user.id

    except Exception as e:
        error_msg = f'Error when creating new User, {e}'
        self.db_log.log_alert(
            user_id = current_user.id,
            alert_level = 3,
            alert_message= error_msg
        )
        print(error_msg)
        return 0
```

Important information like the user password was not saved in any variable in the route, even the class only was returning True or False if the password provided by the user was right or not. At the same time the password is saved in the database using a hash encoding.

```python
def checkUserAndPassword(self, username:str, password:str):
    user = User.query.filter_by(username=username).first()
    if not user or not check_password_hash(user.password, password):
        return False
    return True
```

The login page, like other parts of the web application where a form is used, has a model of the form to be used with the library of `flask_wtf`, this library allows a flexible render of the web forms. (Jacob, 2023)

```
18
19          {% if form is defined %}
20              <form method="POST" action="{{ url_for('login_views.home') }}">
21              {{ form.csrf_token }}
22              <p>Please login to your account</p>
23
24              <div class="form-outline mb-1">
25                  {{ form.username(class_="form-control", size=32) }}
26                  <label class="form-label" for="username">Username</label>
27              </div>
28
29              <div class="form-outline mb-3">
30                  {{ form.password(class="form-control", size=32) }}
31                  <label class="form-label" for="password">Password</label>
32              </div>
33
34              <div class="text-center pt-1 mb-5 pb-1">
```

The model, a python class inheriting a `FlaskForm` class, define the inputs of the form as the attributes of the class, and in this attributes is possible to user more classes of `flask_wtf` in order to defined the type of input, if data is required, max length, etc.

```
20    class UserLoginDB_form(FlaskForm):
21        username      = StringField('Table Name',validators= [ Regexp(r'^[a-zA-Z0-9_ -]*$', message="Only number
22        password = PasswordField('New Password', validators= [DataRequired()] )
      David Silva, 2 months ago | 1 author (David Silva)
23
```

The route for the login is the one that take the necessary information, when the condition of the method post is not True, we can see that the way to implement the form in the UI is very easy.

```
14    @user_login.route('', methods=['GET', 'POST'])
15    def home():
16        #to alert the user about an event
17        alerts = []
18        #when data is sent by the form
19        if request.method == 'POST':
20            #connecting to the database
21            cdb = DatabaseManager(current_app.config["MYSQL"])
22            # #getting username in form
23            username = request.form['username']
24            # #getting user password in FORM
25            password = request.form['password']        David Silva Troya, last month • C
26            # #getting that username from the database
27            if cdb.usersTables.checkUserAndPassword(username,password):
28                user = cdb.usersTables
29                user_id = cdb.usersTables.getUserIdByUsername(username)
30                user.id =  user_id
31                user.username = cdb.usersTables.getUsernameByID(int(user_id))
32                role_info = user.getUserRoleInfoByUserID(int(user_id))
33                user.role_id = role_info['id']
34                user.role = role_info['name']
35                user.permission = role_info['permission']
36                login_user(user)
37                return redirect(url_for('home_views.home'))
38            #when user does not exists in the database
39            else:
40                alerts.append((f'Error with you User or Password', 3 ,""))
41        #creating form to show in login
42        form = UserLoginDB_form()
43        return render_template("pages/login/login.html",form=form,alerts=alerts)
44
```

In the route of the user login, once the user send the information in the form, the method post is True, then the program will check the password returning an alert if it is wrong or setting the user information and sending the user to the home page if the password and user information is correct.
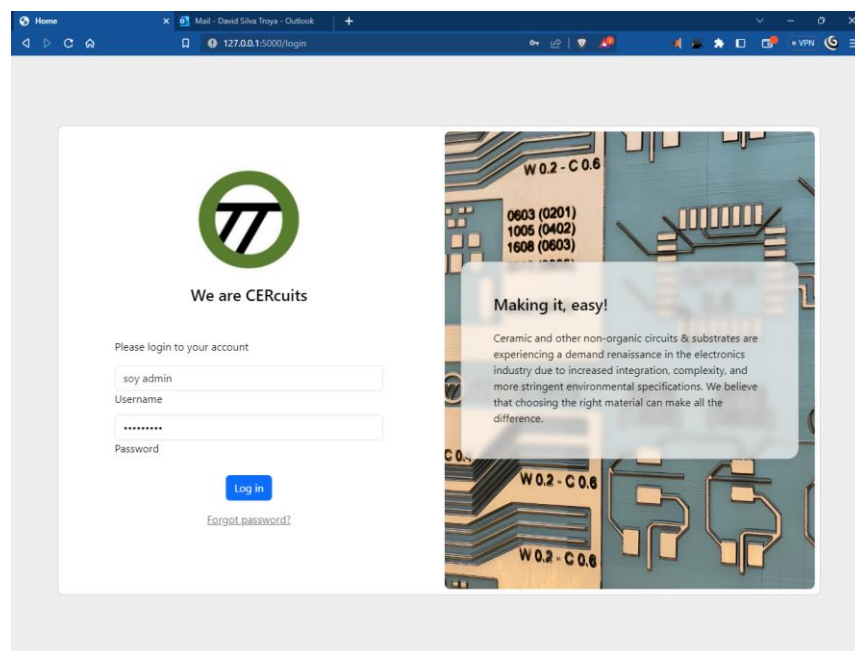
4.3   Web Application database

The Stock Management System allows the users, with a specific role, to create new tables. Because SQLAlchemy is used to manage the database, the tables need to be created with a model or class. (SQLAlchemy, 2023)

Python provides the option to create classes in a dynamic way, meaning that is possible to create models in a dynamic way to be used with SQLAlchemy. So now in order to create this tables it is necessary to save the information to create the new tables, for that are created the main models in the web application. (python, 2023)

The main models, known as the "main tables" in the web application, are the models to create the tables that will save information in the database to then with this information create dynamic classes and with this new tables in the database.

The models of the web application were created in based of the ERD created in the Initiation phase, being this a big help to modify the code in a easier way when it was necessary to add or remove specific features after every demo presentation.

```
stock_management_system > models > tables > 🐍 defined_models.py > ...
        David Silva Troya, 2 weeks ago | 2 authors (David Silva Troya and others)
   1    from stock_management_system.extensions import db        David Silva Troya, last month • Still changing meth
   2    from datetime import datetime
   3
   4
   5    '''
   6     For the information to create new Groups, Tables, Columns and Links between tables.
   7    '''
   8
        David Silva Troya, last month | 1 author (David Silva Troya)
   9    class GroupTable(db.Model):
  10        __tablename__ = 'CERCUITS_groupsDB'
  11        id = db.Column(db.Integer, primary_key=True)
  12        name = db.Column(db.String(80), nullable=False)
  13        title = db.Column(db.String(200), nullable=False)
  14        user_role = db.Column(db.Integer, nullable=False)
  15        hide = db.Column(db.Boolean, default=False , nullable=True)
  16        tables = db.relationship('TablesTable', backref='groupsDB', lazy=True)
  17
        You, 4 weeks ago | 2 authors (David Silva Troya and others)
  18    class TablesTable(db.Model):
  19        __tablename__ = 'CERCUITS_tablesDB'
  20        id = db.Column(db.Integer, primary_key=True)
  21        name = db.Column(db.String(80), nullable=False)
  22        title = db.Column(db.String(200), nullable=False)
  23        created = db.Column(db.Boolean,default=False, nullable=True)
  24        group_id = db.Column(db.Integer, db.ForeignKey('CERCUITS_groupsDB.id'))
  25        columns = db.relationship('ColumnsTable', backref='CERCUITS_tablesDB', lazy=True)
  26        links = db.relationship('LinksTable', backref='CERCUITS_tablesDB', lazy=True)
  27        notifications = db.relationship('NotificationsTable', backref='CERCUITS_tablesDB', lazy=True)
  28
        David Silva Troya, 3 weeks ago | 1 author (David Silva Troya)
  29    class ColumnsTable(db.Model):
  30        __tablename__ = 'CERCUITS_columnsDB'
  31        id = db.Column(db.Integer, primary_key=True)
```

A class called `MainTablesManager` had the methods to can use the main models of the web application, this class was created and improved in base to the Class Diagram but also in the other way around, the Class Diagram had some modifications once the methods were created and used.

The documentation is not only written in Confluence but also in the code of the classes, with comments and specials way to set information so the class and methods could provide information while the program was being developed.

```
14    '''
15
16    Control of the main database for the web application
17
18    '''
19
      David Silva Troya, 2 weeks ago | 2 authors (David Silva Troya and others)
20    class MainTablesManager(BaseTableManager):
21        '''
22        This class is created to manage all the the main tables in the database,
23        the tables are: Groups, tables, columns and links.
24        All edition, insertion and deletion is added to the register table with the user ID.
25        '''
26
27        def __init__(self, mysql: SQLAlchemy):
28            self.db = mysql
29            self.register = RegisterTablesManager(mysql)
30            super(MainTablesManager,self).__init__(mysql=self.db)
31
32        '''All GETs in Main Tables with registering'''
33        #checked
34        def getAllDataFromGroups(self):
35            '''
36            Return a dictionary with all the data from groups in the database
37            '''
38            try:
39                groups = self.getAllDataInDict(GroupTable)
40            except Exception as e:
41                groups = {}
42                alert_msg = f'Error getting all the data from the groups of tables, more information about this error: {e}'
43                self.db_log.log_alert(current_user.id,3,alert_msg)
44            return groups
45
46        #checked
47        def
```

With this way to write comments in the classes and methods, every programmer that is going to use the code in the future, can see important information about it, just by hovering the mouse over the class and method or while writing the code.

```
stock_management_system > controller > 🐍 database_manager.py > ...
      You, 4 hours ago | 3 authors (David Silva Troya and others)         (class) MainTablesManager
1     from flask import url_for
2     from flask_login import current_user                                This class is created to manage all the the main tables in the database,
3     from flask_sqlalchemy import SQLAlchemy                              the tables are: Groups, tables, columns and links. All edition, insertion and deletion is added to the register table with the user ID.
4     from stock_management_system.controller.database.main_tables_manager import MainTablesManager
5     from stock_management_system.controller.database.new_tables_manager import NewTablesManager
6     from stock_management_system.controller.database.user_tables_manager import UserTablesManager
7     from stock_management_system.controller.notifications.alert_logger_dbm import AlertLogger
8
9
```

```
52              user_role_id = self.usersTables.role_id
53          except:
54              user_info = self.usersTables.getCurrentUserData()
55              user_role_id = user_i    (method) def getAllDataFromGroups() -> dict
56
57          #getting all groups infor   Return a dictionary with all the data from groups in the database
58          groups = self.mainTables.getAllDataFromGroups()
59
```

```
28          self.db = mysql
29          self.mainTables = MainTablesManager(mysql)
30          self.register = RegisterTablesManager  (method) def getTableInformationByID(table_id: int) -> (Any | Literal[''])
31          super(NewTablesManager,self).__init_
32                                                 This will search for the table name with the ID of the table. Returns the table information in a dictionary.
33      #checked
34      def getTableModel(self, table_id:int, tab    Dictionary example:
35          '''                                     {
36          Function to get the model of the tabl
37          Needs the table id in order to get al      'id': int,
38          the table name is in case this model       'name': str,
39          '''                                         'title': str,
40          try:                                        'created': bool,
41              #get table name
42              if table_name == '':                    'group_id': int
43                  table_name = self.mainTables.getTableInformationByID(table_id)['name']
44              #get columns name and definition for table
45              columns = self.mainTables.getColumnsForModel(table_id)
46              if len(columns) <= 0:
47                  if current_user and current_user.id:
```

One of the functions that took more time to implement, because of the complexity of the dynamic creation, is the function to get the model of the new tables. This functions

has exceptions to inform the user what the error is but also to handle the error avoiding that the program breaks.

```python
def getTableModel(self, table_id:int, table_name =''):
    '''
    Function to get the model of the table to be use with flask_sqlAlchemy.
    Needs the table id in order to get all the data related to the table and
    the table name is in case this model will be to set a migration table with other name.          David Silva Troya, 2 weeks ago • Possible to
    '''
    try:
        #get table name
        if table_name == '':
            table_name = self.mainTables.getTableInformationByID(table_id)['name']
        #get columns name and definition for table
        columns = self.mainTables.getColumnsForModel(table_id)
        if len(columns) <= 0:
            if current_user and current_user.id:
                self.register.db_log.log_notification( int(current_user.id),'Not columns found for this table.')
            raise Exception('Columns are necessary to create table')

        # clear existing metadata
        try:
            self.db.metadata.clear()
            Base = self.db.Model
        except:
            self.db.bind.metadata = MetaData()
            Base = declarative_base()

        table_class = type(
            f'{table_name}_model',
            (Base,),
            {
                '__tablename__': table_name,
                'id': Column(Integer, primary_key=True)
            }
        )

        for column in columns:
            setattr(table_class, column.name, column)
        return table_class
    except Exception as e:
        print(f'\n\n---->\033[91m Error creating model of new table, {e} at line {sys.exc_info()[-1].tb_lineno} \033[0m')
        return None
```

The method `getTableModel` has the option to use flask_SQLAlchemy and SQLAlchemy. The main difference between the 2 libraries is that one is made to work with the flask context, meaning that any other script out of the context that one user has, is not going to work. This gives a problem for the automation, so SQLAlchemy is required then, but because the class is already create to use the dynamic models, it is better to add the option of use both libraries than create a new class for each library. (Pallets Projects, 2023) (SQLAlchemy, 2023)

4.4   Constraints creating new tables

Because at the moment of the creation of a new table, the possibility of creating a table pointing to another table, or better say, a table with a column with foreign keys, this columns only had information about the value of the ID from the other table, information difficult to read by the user and sadly a feature of SQLAlchemy that I could not implement with the dynamic classes.

In fact, the columns with foreign keys were columns with the INT type, to save integer values from the IDs of the rows in the other tables. This results in the importance to create some constraints to avoid things like the table pointing to itself.

Another constraints were related with the way of getting the data from the database, since the foreign key values had to be replace for the values of the rows in the other table, it was important to avoid that any column in that table was pointing back. Or in some cases this could create an infinite loop with the recursion of the method.
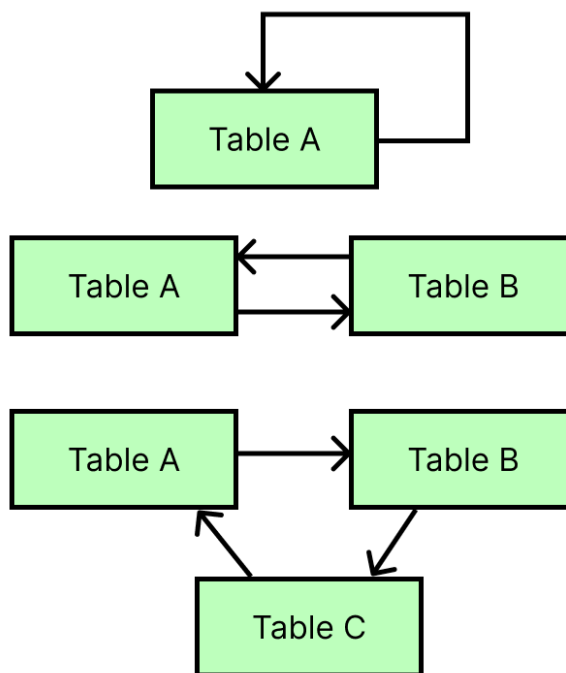
Checking that the tables created followed the logic, avoiding the constraints was also a difficult part, it was necessary to JavaScript to prevent that the user has the option to point to the table in creation and also to handle the error in case this was not enough to limit the user. And checking that the tables are not creating a loop, taking in mine that it could be a connection with hundreds of tables.

```python
790    def checkIfLinksMakeLoop(self,links_info:dict,stack:list,table_from,table_to):
791        ''' Returns True or False.
792            Recursive function, use:
793            - link_info: To check tables connections.
794            - stack: To save IDs of tables already checked.
795            - table_from: To know in what table is checking.
796            - table_to: To know to what table the current table(table_from) connects.
797        '''
798        #saving the current table ID in the stack
799        stack.append(table_from)
800        #checking if the next table was already visited
801        if table_to in stack:
802            return True
803        #looping to check in all the
804        for l_id in links_info:
805            new_table_from = int(links_info[l_id]['table_from'] )
806            new_table_to= int(links_info[l_id]['table_to'])
807            #when the new table is the same as the previous table pointed
808            if new_table_from == table_to:
809                #returning the result of the recursivity
810                return self.checkIfLinksMakeLoop(links_info,stack,new_table_from,new_table_to)
811    return False        You, last month • Conection between table is constrained, not possi…
812
```

This was fixed by using recursion in the method to check that a loop is not created with the columns that have foreign keys, checking every connection and returning a True once a node or table was being checked twice.

4.5 Getting data from the new tables

This process requires to first get all the data related to the creation of the table, the name and columns information to create the model and after that not only get the information of the table but also get the information of the foreign keys if there is any, here is the reason why the constraints were so important in order to avoid infinite loops.

```python
119    def getDataToShowInTablePage(self,table_id: int):
120        '''
121        Returns dictionary with structure:\n
122        { 1: #row_id\n
123            {\n
124                column_name #Name of one column\n
125                {\n
126                    'value' : value, # this is the real value of the column\n
127                    'value_fk' : value, # when the column if FK then this has the value of the row from the table pointed\n
128                    'definition' : definition,\n
129                    'title' : definition,\n
130                    'col_id' : column id,\n
131                },\n
132                ...\n
133            },\n
134            ...\n
135        }\n
136        '''
137        newTableData = {}          David Silva Troya, 3 weeks ago • Fixing some errors and way to show notifications
138
139        #get table data
140        try:
141            newTable = self.getTableModel(table_id)
142            newTableData = self.getAllDataInDict(newTable)
143            for row_id in newTableData:
144                newTableData[row_id].pop('id')
145        except Exception as e:
146            print(f"Error in the function getDataToShowInTablePage() part-1 from class NewTablesManager. More about the information error: {e}.
147            return {}
148
149        #get columns information
150        columns_info = self.getNewTableColumnsInformation(table_id)
```

The method `getDataToShowInTablePage` only needs the ID of the table in the database, to then get the model and with the model the data in a dictionary. In the second part of the method, the dictionary is updated with more data that Jinja is going to use in the rendering, the recursion works when the column is a foreign key type, to take the data from the table pointed and show to the user this data in a way that can be understood, not only a row ID.

```python
149        #get columns information
150        columns_info = self.getNewTableColumnsInformation(table_id)
151        try:
152            for col_id in columns_info:
153                col_name = columns_info[col_id]["name"]
154                col_definition = columns_info[col_id]["definition"]
155                col_title = columns_info[col_id]["title"]
156                for row_id in newTableData:
157                    for c_name in newTableData[row_id]:
158                        if col_name == c_name:
159                            cell_value = newTableData[row_id][c_name]
160                            newTableData[row_id][c_name] = {}
161                            data = {
162                                "value" : cell_value,
163                                "value_fk" : cell_value,
164                                "definition" : col_definition,
165                                "title" : col_title,
166                                "col_id" : col_id
167                            }
168                            print(f'\nSO now HERE:  data = {data} \n')
169                            if col_definition == 'FK' and not cell_value is None:
170                                try:
171                                    fk_table_id = self.mainTables.getTableIDbyColumnIDFromLinks(int(col_id))
172                                    fk_table_data = self.getDataToShowInTablePage(int(fk_table_id))          David Silva Troya, 2 weeks ago • Fixed
173                                    fk_data_in_row = fk_table_data[int(cell_value)]
174                                    fk_data_to_show = ''
175                                    for fk_col_name in fk_data_in_row:
176                                        fk_data_to_show += f' {fk_data_in_row[fk_col_name]["value_fk"]} {self.separator}'
177                                    data['value_fk'] = fk_data_to_show[:-(len(self.separator))]
178                                except:
179                                    data['value_fk'] = None
180
181                            newTableData[row_id][c_name] = data
182        except Exception as e:
183            print(f"Error in the function getDataToShowInTablePage() part-2 from class NewTablesManager. More about the information error: {e}.
184
185        return newTableData
```

The data goes from the route to the template of the new tables. Rendering these tables was also complicated, but the result is a table that allows the user to edit more than one cell, even if this is a value of a foreign table, displaying a select input to change the ID, meanwhile the user see all the option from the other table.

```
169                  </td>
170              {% for c_name in row_data %}
171                  <td
172                      {% if current_user.role_id|int < 2 %}
173                          class="new-table-cell position-relative ps-4"
174                      {% else %}
175                          class="position-relative ps-4"
176                      {% endif %}
177                      data-row="{{row_id}}"
178                      data-col="{{c_name}}"
179                      data-col-definition="{{row_data[c_name]['definition']}}"
180                      id="{{c_name}}-{{row_id}}"
181                  >
182                      {% if row_data[c_name]["definition"] == "FK"  %}
183                          {% set fk_col, col_id, cell_value = cercuitsDB["fk_data_to_select"],row_data[c_name]["col_id"], row_data[c_name]["value"]%}
184                          {% set vars = {'data': 'no data..'} %}
185
186                          <select
187                              class="form-select new-table-cell-select d-none"
188                              data-row="{{row_id}}"
189                              data-col="{{c_name}}"
190                              id="select--{{c_name}}-{{row_id}}"
191                              name="select--{{row_data[c_name]['name']}}-{{row_id}}">       You, 4 weeks ago • Possible to see data in chart
192                              {% for fk_col_id in fk_col[col_id]  %}
193                                  {% if cell_value == fk_col_id %}
194                                      {% set _ = vars.update({'data': fk_col[col_id][fk_col_id] }) %}
195                                      <option value="{{fk_col_id}}" selected>{{fk_col[col_id][fk_col_id]}}</option>
196                                  {% else %}
197                                      <option value="{{fk_col_id}}">{{fk_col[col_id][fk_col_id]}}</option>
198                                  {% endif %}
199                              {% endfor %}
200                          </select>
201
202                          <span class="span-form-select">
203                              {{row_data[c_name]["value_fk"]}}
204                          </span>
205
206                      {% else %}
207                          <span class="span-cell-value">
208                              {{row_data[c_name]["value"]}}
209                          </span>
210                      {% endif %}
211                  </td>
212              {% endfor %}
213          </tr>
```

To show how it works in the user interface, I have created two tables, one called Table A and Table B. Table A will have a text column and a integer column, and the Table B will have a text column and a Foreign keys column that is pointing to the Table A.

Table A has now 4 rows with text and integer data, meanwhile Table B has text and a integer data that is not shown but instead the data from the Table A based in the integer. The string "||" is added between values of the columns from the pointed table.

4.6   Inserting data to the new tables

A form is created with the information received from the route, this use a for loop in Jinja to iterate the creation of all the inputs for all the columns.

```html
279        <form method="POST" id="insert-data-in-new-table-form" action="{{ url_for('stock_views.add_data', table_id=cercuitsDB['tabl
280          <div class="row justify-content-center">
281            {% for col_id in col_info  %}
282              <div class="col-12 col-md-6 col-lg-4 col-xl-3">
283                <label for="{{col_info[col_id]['name']}}" class="form-label">
284                  {{col_info[col_id]['title']}}
285                </label>
286                {% if col_info[col_id]['definition'] == "FK" %}
287                  <select
288                    class="form-select"
289                    id="{{col_info[col_id]['name']}}"
290                    name="{{col_info[col_id]['name']}}">
291                    {% for fk_col_id in fk_col[col_id]  %}
292                      <option value="{{fk_col_id}}">{{fk_col[col_id][fk_col_id]}}</option>
293                    {% endfor %}
294                  </select>
295                {% else %}
296                  <div
297                    {% if col_info[col_id]['definition'] == 'BOOLEAN'%}
298                     class="form-check form-switch my-1"
299                    {% endif %}
300                  >
301                    <input
302                      {{input_type(col_info[col_id]['definition'])}}
303                      id="{{col_info[col_id]['name']}}"
304                      name="{{col_info[col_id]['name']}}"
305                      aria-describedby="{{col_info[col_id]['name']}}-Help">
306                  </div>
307                {% endif %}

309                <div id="{{col_info[col_id]['name']}}-Help"
310                  class="form-text pb-4">
311                  Data for column called: {{col_info[col_id]['title']}} <br>
312                  With definition: {{col_info[col_id]['definition']}}
313                </div>
314              </div>
315            {% endfor %}
```

The method `insertInNewTableFromRequest` gets the table ID and the form with all the data, setting all the data in a dictionary to then use the method `insertInNewTable` that is going to add the data in the database.

```python
261    def insertInNewTable(self, table_id: int, data:dict):
262        try:
263            newTableModel = self.getTableModel(table_id)
264            newTable = newTableModel()
265            newTable = self.setDataInTable(newTable,data)
266            self.db.session.add(newTable)
267            self.db.session.commit()
268            return newTable.id
269        except:
270            error_msg = f'Error inserting data in this new table'
271            self.db_log.log_alert(
272                user_id = current_user.id,
273                alert_level = 3,
274                alert_message= error_msg
275            )
276            print(error_msg)
277            row_id = 0
278        return row_id
279    #
280    def insertInNewTableFromRequest(self, table_id:int, request : Request):
281        try:
282            #getting the table to insert data
283            data = {}
284            data.update(request.form)
285            row_id = self.insertInNewTable(table_id,data)
286            return row_id

288        except Exception as e:
289            error_msg = f'Error getting the data in the form to insert data in new table, {e}'
290            self.db_log.log_alert(
291                user_id = current_user.id,
292                alert_level = 3,
293                alert_message= error_msg
294            )
295            print(error_msg)
296            return 0
```

The form is rendered in a modal, a component of bootstrap 5.3, giving the option to cancel the insertion if it is necessary. With this the user does not have to be moved to a new page dedicated only for the creation of a new row. (Bootstrap team, 2023)
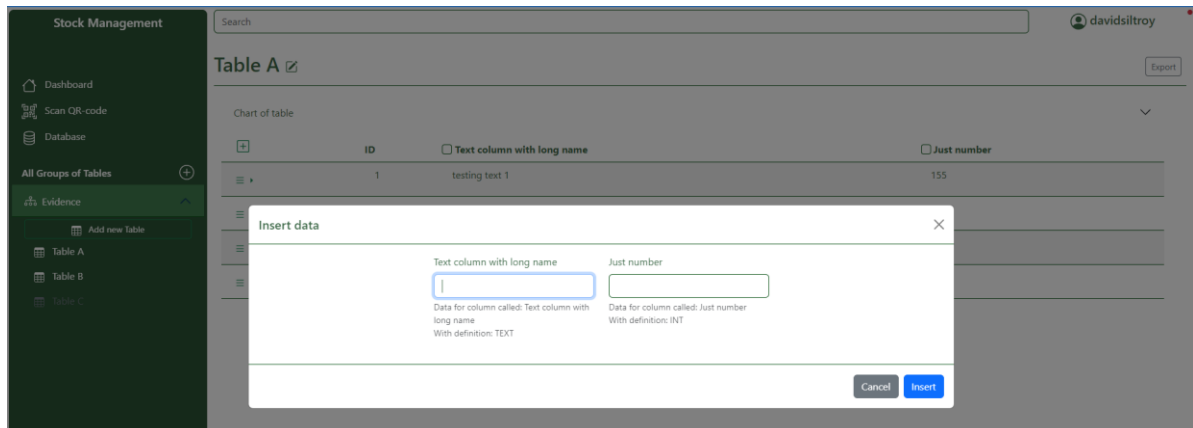

When the table has a column that save foreign keys, then a select input is displayed, giving the different options and saving in the form the ID of the row related to the option selected.
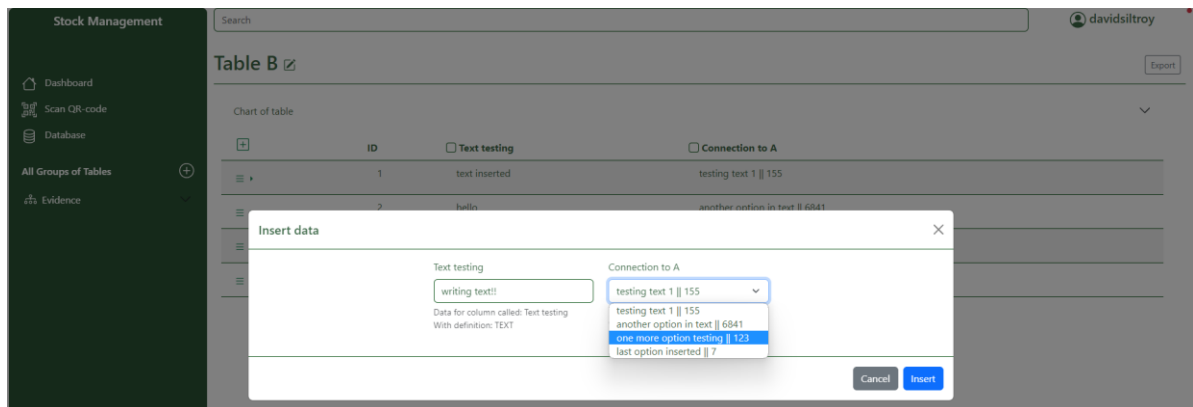
4.7   Editing data from the new tables

There are two options for the edition of the data, one is by clicking the menu in every row, located in the first column of the table. And the other option is by clicking the cell itself to then clicking in the "Save Data" button that is always located at the bottom of the UI aligned with the table.
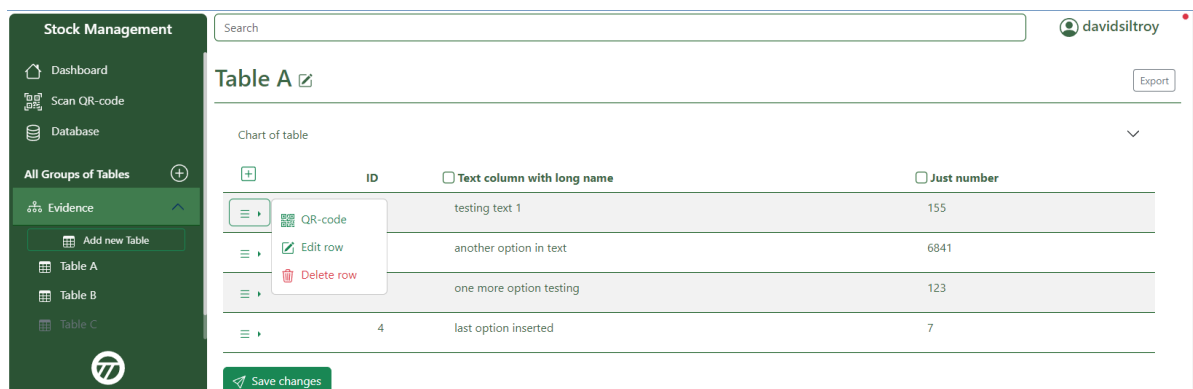
The first option, by clicking in the "Edit row" button, will send the user to another page where it is possible to make changes but also to add extra information, information related to notifications, but only if the column is a number type.

And the second option to edit values in the table is with AJAX, or Asynchronous JavaScript And XML. A special route was created for this option, this route returns true or false depending on the result of the modifications to then inform the user. (Mozilla Developer Network, 2023)

The JavaScript code is too long to be shared and explained in this document, but what it does is to set a Listener in the cells, saving the current value and only giving the option to save changes when that value changes. Saving all the values and data related to that cell, like the table ID, row ID and column name.

Once the "Save Changes" button is clicked, all the modified data will be send to the route made to receive AJAX requests. And, once the server finish the process, this is going to send an answer that JavaScript will show to the user as an Alert, with the color green when all is ok and red when not.

And all the data saved from the edition will be deleted, removing the classes in the cells to show the table like it would look like once the user goes to that table. But in this case the user never was moved to another page.

The tables with the foreign key will show a select input in the columns where the data of the foreign key is saved. Also working with AJAX so the user does not have to go row by row when a change is necessary in more than one field of the table.

4.8 Deleting data from the new tables

The deletion process is easier than everything else, but it is only possible to be done by a user with the higher role. This is to avoid losing valuable information, so only specific users can see the option to delete.
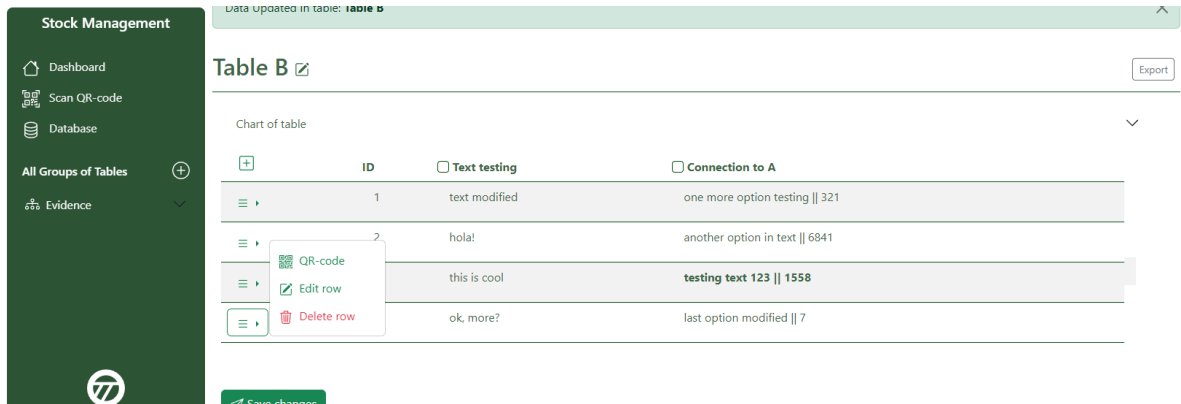
Just by clicking the options in the left of the rows, then the option called "Delete row", the row will be deleted from the database.
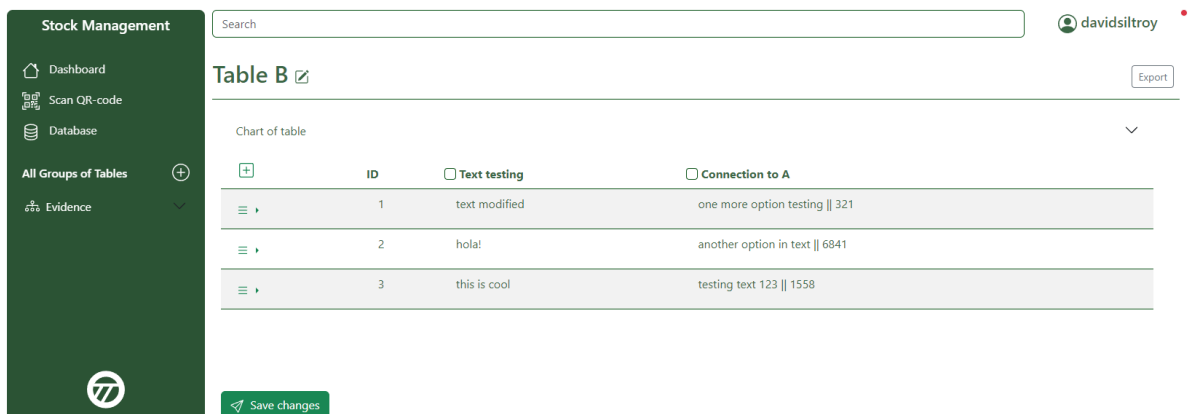
And the method will return a True or False depending on the result of the process of deleting the specific row.

```python
367    def deleteInNewTable(self, table_id : int, row_id: int):
368        all_deleted = False
369        try:
370            newTableModel = self.getTableModel(table_id)
371            newTable = newTableModel.query.get(row_id)
372            self.db.session.delete(newTable)
373            self.db.session.commit()
374            all_deleted = True
375        except:
376            error_msg = f'Error deleting data in this table'
377            self.db_log.log_alert(
378                user_id = current_user.id,
379                alert_level = 3,
380                alert_message= error_msg
381            )
382            print(error_msg)
383
384        return all_deleted
385    #
```

4.9   QR-code generation

The web application has the option to generate and read QR-codes, creating a URL referring to the row ID of the row selected to generate the code. The user can generate the code in the first column of the table, at the level of the row that wants to get a QR-code.
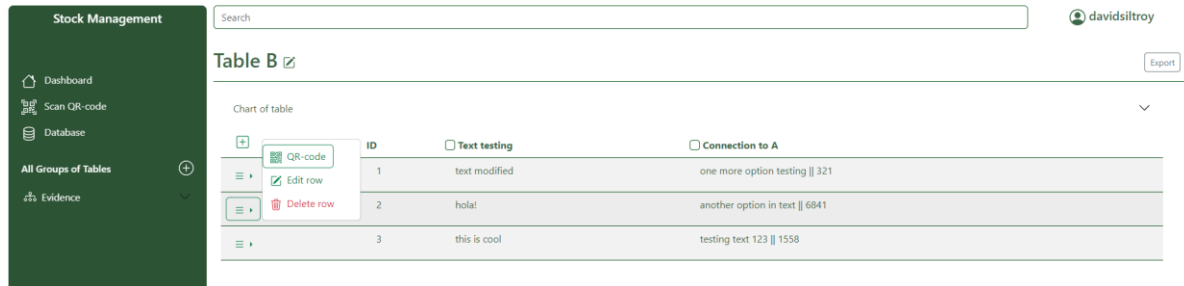
The QR-code is generated with a python library called `flask_qrcode`, using AJAX to request the QR-code from an specific route, created to return a QR-code image code in base64. (Agner, 2023) (Mozilla Developer Network., 2023)

```python
209    @stock_views.route('/getQRcodeOfRow/<table_id>/<row_id>',methods=['GET'])
210    @login_required
211    def js_generate_qrcode(table_id,row_id):
212        cdb,_ = refreshValuesFromDB()
213        base_url = request.host_url
214        tables = cdb.mainTables.getAllDataFromTables()
215        table_name = tables[int(table_id)]['name']
216        url = f'{base_url}{url_for("stock_views.add_data", table_id=table_id,table_name=table_name,row_id=row_id)}'
217        qr = qrcode(url, version=2, error_correction='H', box_size=10, border=4)
218        return jsonify({'qr': qr})
219
```

The JavaScript code send data to generate a QR-code and starts the creation of a image to be inserted in a modal, once the QR-code is received. The modal is showed using the bootstraps class for JavaScript.

```javascript
195
196        /*
197            To create and show QR-code in a modal
198        */
199        const qrGeneratorBtns = document.querySelectorAll('.qr-generator-link');
200        if(qrGeneratorBtns){
201            qrGeneratorBtns.forEach(link => {
202            link.addEventListener('click', e => {
203                e.preventDefault();
204                const row_id = link.getAttribute('data-row-id');
205                const table_id = link.getAttribute('data-table-id');
206                fetch(`/stock/getQRcodeOfRow/${table_id}/${row_id}`)          You, 4 weeks ago • Possible to se
207                .then(response => response.json())
208                .then(data => {
209                    const img = new Image();
210                    img.src = data.qr;
211                    img.style.width = '100%';
212                    const qrModal = document.getElementById('qr-code-new-table-modal');
213                    if (qrModal) {
214                    var myModal = new bootstrap.Modal(qrModal,{keyboard: false})
215                    const qrSection = qrModal.querySelector('div.modal-body');
216                    qrSection.innerHTML = '';
217                    qrSection.appendChild(img);
218                    myModal.show();
219                    }
220                });
221            });
222            });
223        }
224
```

The user will be able to see the image in a modal, with the option to save it like any other picture in internet, using the right click.

The web application has a QR-code reader, but it is not possible to use the camera since the web browser does not trust in a web page with not SSL certificate. Still there is an option to update an image with a QR-code.
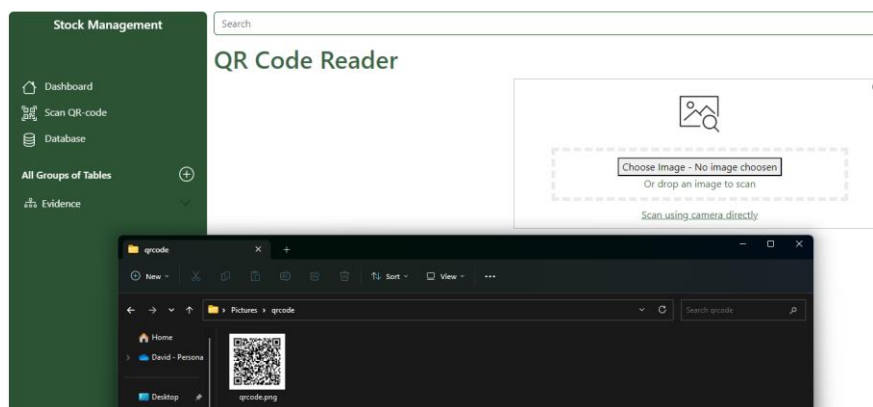
The result is that the web application will show the table with the information of the QR-code, setting in the first position the row where the QR-code was generated and also changing the background of this row, so the user can be sure that is the right row.
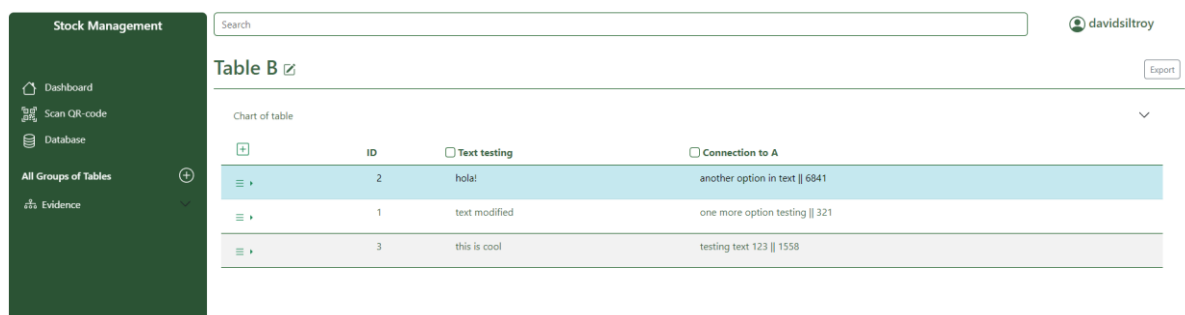
4.10   Notifications

In the part of the Edition of data from the new tables, was mentioned the possibility of creating notifications, when a row has columns with number type. Here we are going to see more into details the options provided for the notifications, which is the option in blue, located next to the name of the column with the number type.

It is possible to add extra information to the rows, this information is related to a maximum or minimum value of the that column in the specific edited row. But also a reminder to get a notification in an specific date, with not necessity of a value. And lastly, the option to save a message to provide more information about the notification.

The extra information is saved in the database and later is used by the Automation part of the web application, which is a background process that is checking all this values.

And when the value of the row is changed, then is possible to see in the notifications option, under the username, that we have one notification.

The notification gives the option to delete it or also to click the text to go to the table, showing the row in the first place, in the same way thar it works when a QR-code is scanned.

4.11   Automation

The automation part is a class running with an infinite loop, and in parallel with the main class of the web application. Here is used `SQLAlchemy` and not `flask_SQLAlchemy` since the parallel process is outside of the context of the user.

```python
11   class StockAutomation:
12       #email configuration
13       email_sender = ''
14       email_password = ''
15       email_server = 'smtp.office365.com'
16       email_port = 587
17
18       # __init__ method or constructor
19       def __init__(self, config={}):
20           self.engine = create_engine(f'mysql://{config["user"]}:{config["password"]}@{config["host"]}/{config["database"]}')
21
22       def chekRowsMaxMinValues(self):
23           Session = sessionmaker(bind=self.engine)
24           while True:
25               session = Session()
26               newTables = NewTablesManager(session)
27               notifications = newTables.register.getAllDataFromNotifications()
28               tables = newTables.mainTables.getAllDataFromTables()
29               current_date = datetime.now()
30   >           for n_id in notifications: ...
107
108               time.sleep(2)
109
```

This class may have multiple methods, but in this case I am using only one to check the extra values added to a numeric cell in the row of a new table. After getting that information is going to check the values of that specific cell to be compared with the extra information or can be the case of only checking the date for a reminder.

```python
30                   for n_id in notifications:
31                       data = {'id': n_id}
32                       data.update(notifications[n_id])
33
34                       user_id = int(data['user_id'])
35                       table_id = int(data['table_id'])
36                       row_id = int(data['row_id'])
37                       column_id = int(data['column_id'])
38                       max_value = data['max_value']
39                       min_value = data['min_value']
40                       reminder_date = data['reminder_date']
41                       notified_date = data['notified_date']
42                       message = data['message']
43                       message = '' if message is None else message
44
45                       newTableData = newTables.getDataToShowInTablePage(table_id)
46                       row = newTableData[row_id]
47
48                       table_name = tables[int(table_id)]['name']
49                       table_title = tables[int(table_id)]['title']
50                       for row_name in row:
```

The initial function of this class was to send an email to the specified user once the specific value or reminder date was reached, but there was a problem using the company email.

Since the company email works with Microsoft, seems the year 2020 an update is not allowing users to use the SMTP option. Different solutions were tried and not success, so it was decided to only show the notification in the User Interface, for now.

```python
for row_name in row:
    link = f'/stock/{table_id}/{table_name}?row_id={row_id}'
    msg = ''
    if row[row_name]['col_id'] == column_id:
        col_value = row[row_name]['value']
        col_title = row[row_name]['title']

        col_value = float(col_value)
        if not min_value is None and col_value < min_value:
            msg += f'Value in column "{col_title}" of table "{table_title}" is bellow MIN'

        elif not max_value is None and col_value > max_value:
            msg += f'Value in column "{col_title}" of table "{table_title}" is above MAX'
        else:
            pass

        if not reminder_date is None and len(str(reminder_date))>0:
            if reminder_date<= current_date:
                msg += f'REMINDER, check column {col_title} in table "{table_title}"'

        if len(msg)>0:
            if len(message)>0:
                msg+=f'.({message})'

            #to show notifications in the User Interface
            newTables.db_log.ui_log_notification(user_id , msg, link)

            #To show notification dinamicly in as toast and send email to the user
            if notified_date <= current_date - timedelta( hours=8): #timedelta(hours=3, minutes=3, seconds=3):
                data['notified_date'] = current_date
                newTables.mainTables.register.editNotification(data)
                newTables.db_log.log_notification(user_id , msg, link)
                session.commit()
```

Using the singleton class `AlertLogger` it is possible to set a notification that the web application will display to the user based in the user ID. Because it is save in the database when the user was notified, this allows to set a specific time between notifications, to not get the same notification all the time.

5 DEMOS

5.1 First demo

The first demo was to give an idea of how the web application will look like and what was possible to be done with it. To have a fasted development in the User Interface, was used as template, a dashboard example of Bootstrap 5. (Bootstrap, 2023)
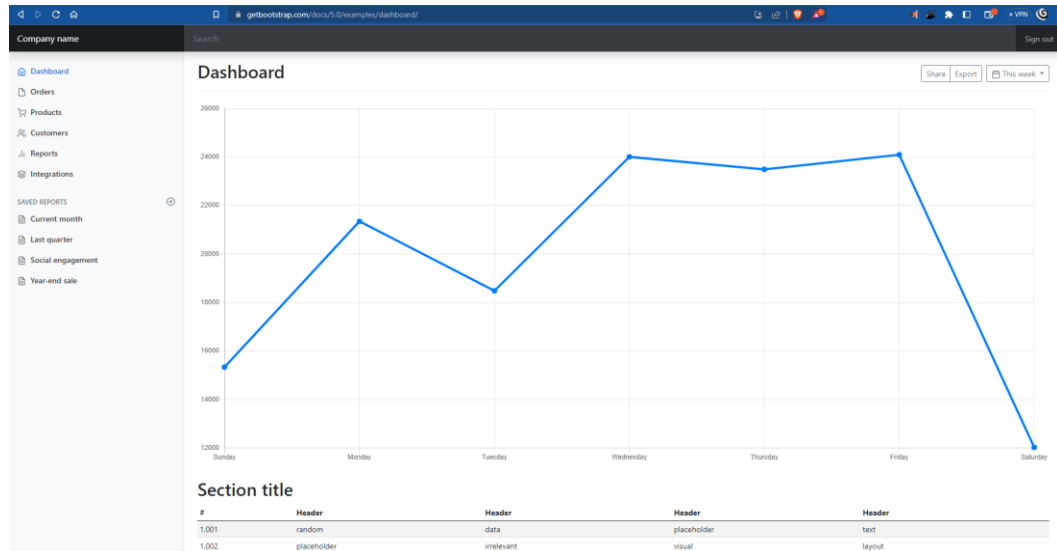

The demos was always starting with the web application without data, to show the step by step how to manage the tables to then manage the data in them. At the same time all the demonstrations had the goal of get all the feedback of the user interface and user experience.
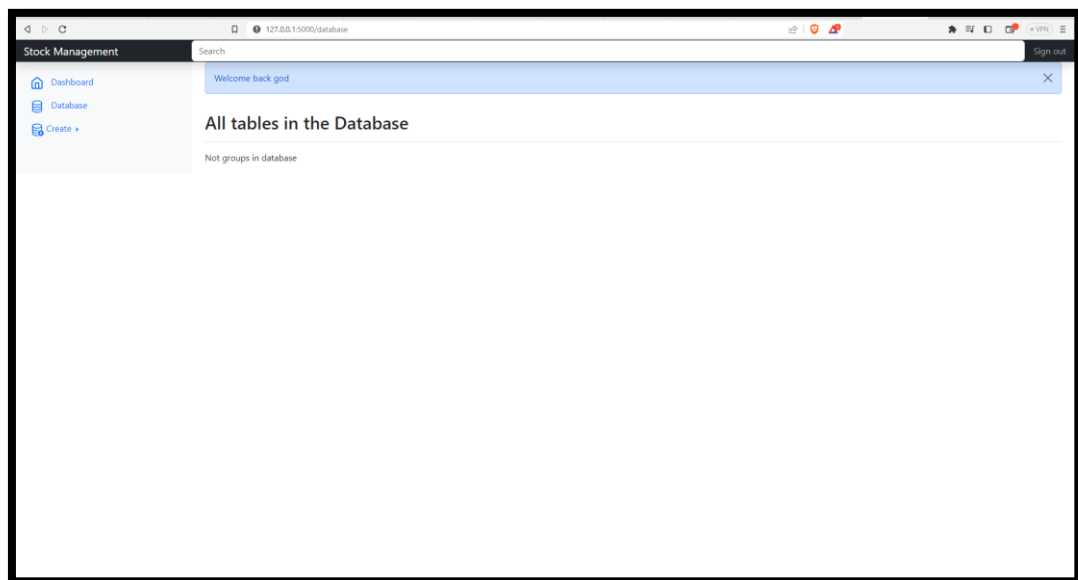
More option in the left side menu started to be visible when the some data was set, for example the option of Edit was there only when there was information to be edited like a group, a table or a column.
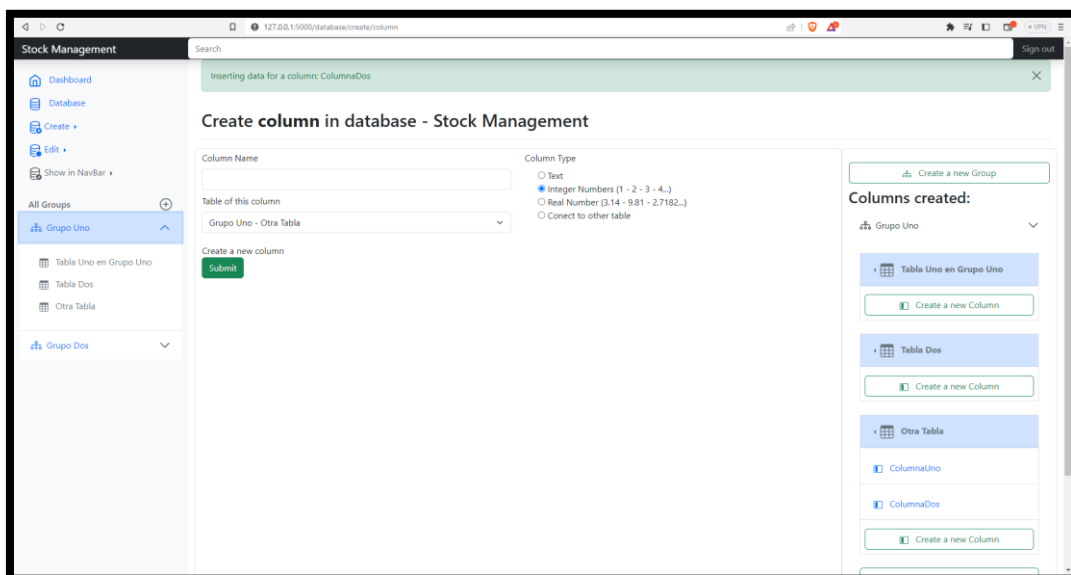
Once the user was in the Edition part, was able also to delete the information stored about that element, which could be a group of table, a table or a column from a table and for easy access also was possible see a list of other elements related to the one that is been edited.
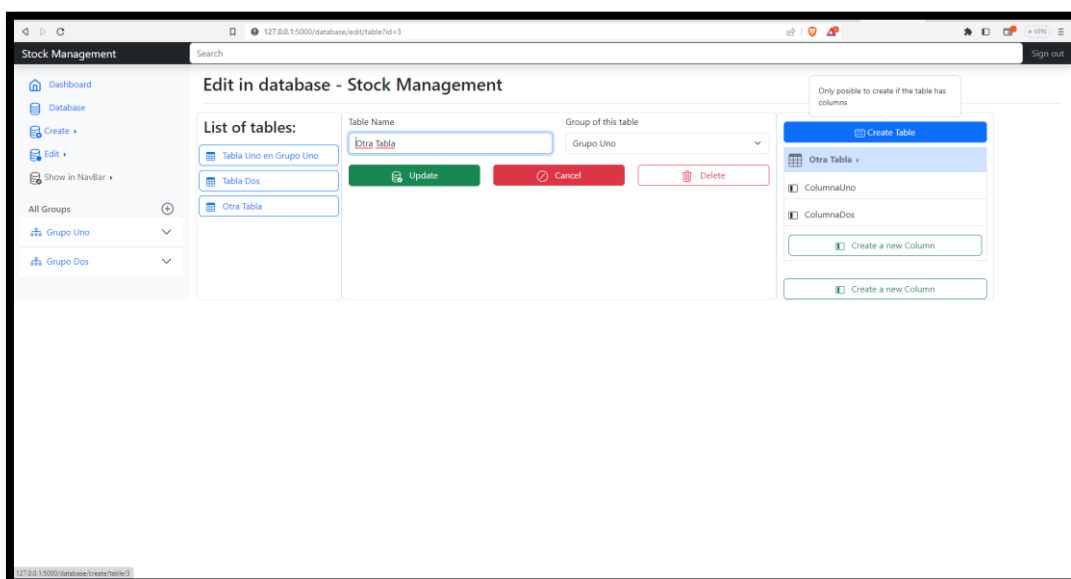
And once all the necessary information to create a table was set in the database, it was possible to create it, now to store data in that table. But at this point is still not possible to edit that data, set a notification once a maximum or minimum value has been reached or even to generate a QR-code.

This is an example of how would look a new table created in a dynamic way with the information saved in the database for it.
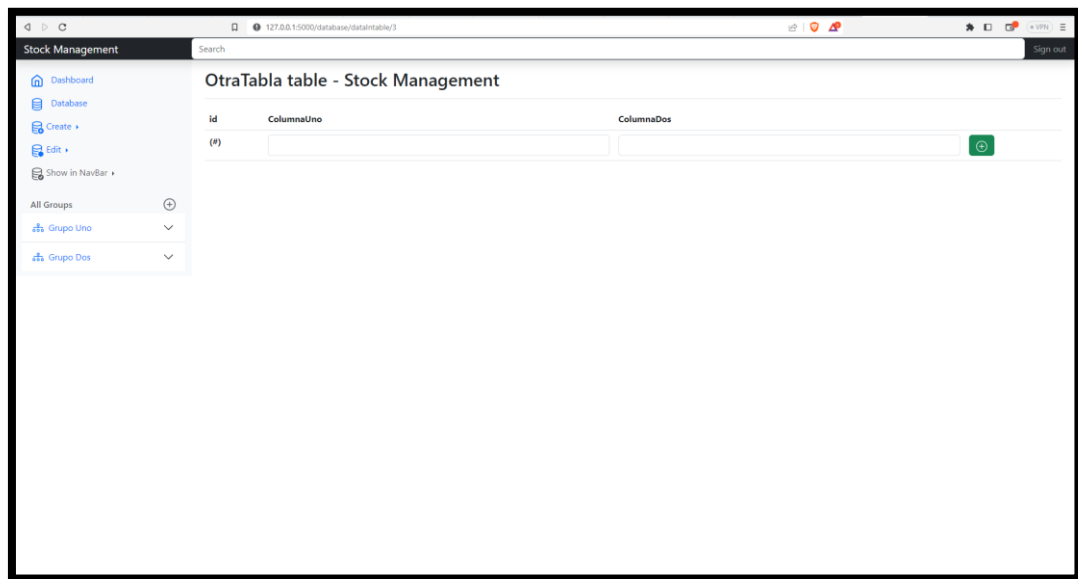
5.2   Last demo

The last demonstration was the result of many others demonstrations and all the feedback gathered from the meetings or just with specific features testing.

The user interface is an important part of the web application because the final user has to be able to understand how everything works, I modified the template of the web application in based to a design made by Hector Silva.
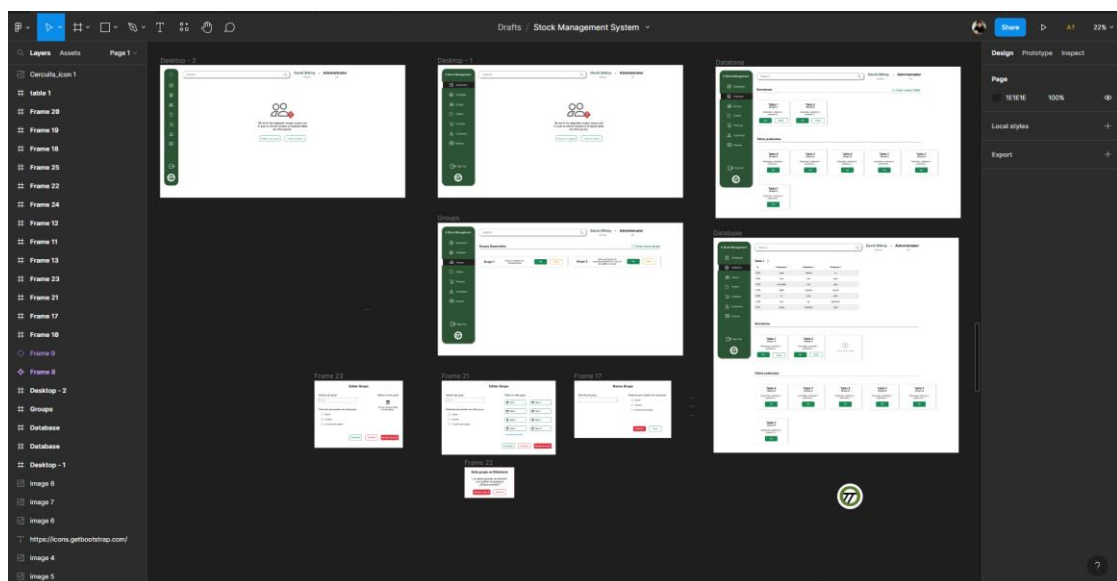

One of the main things for the last demo is the new Template, using better the spaces of the screen. And some new options like the QR-code scan and the option in the database to edit tables details for the creation or users that are allowed to use the web application.
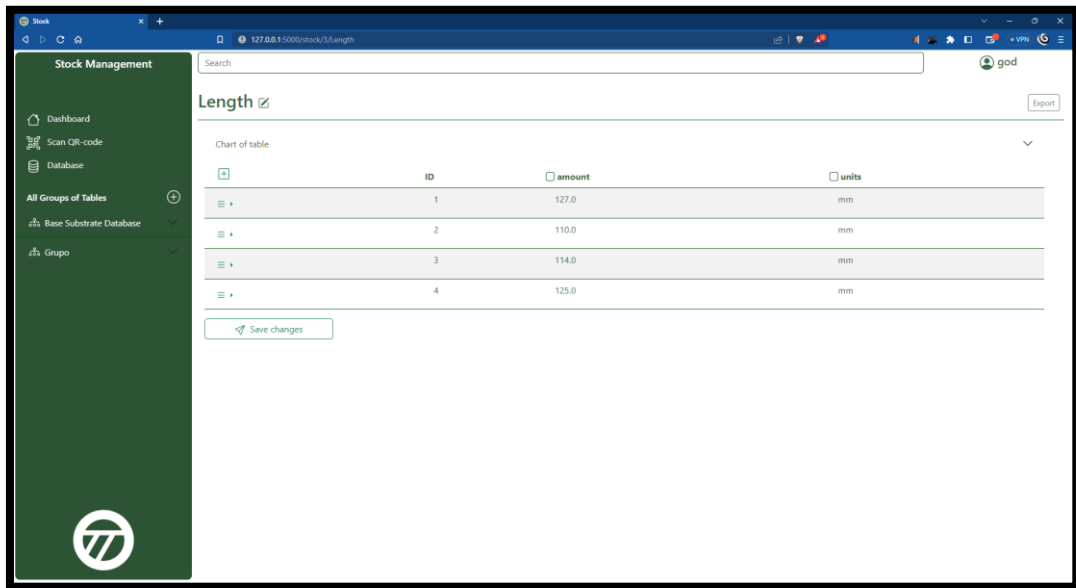
The Dashboard or home page was mean to be displaying last data edited, some news or a general graph from specific data taken of the new tables. But currently this just send you to a random new table created, or just display a message that there are no tables created yet.

Now it is possible to see notifications in the bottom right corner of the web application and the group, table and column creation can be done with a modal.
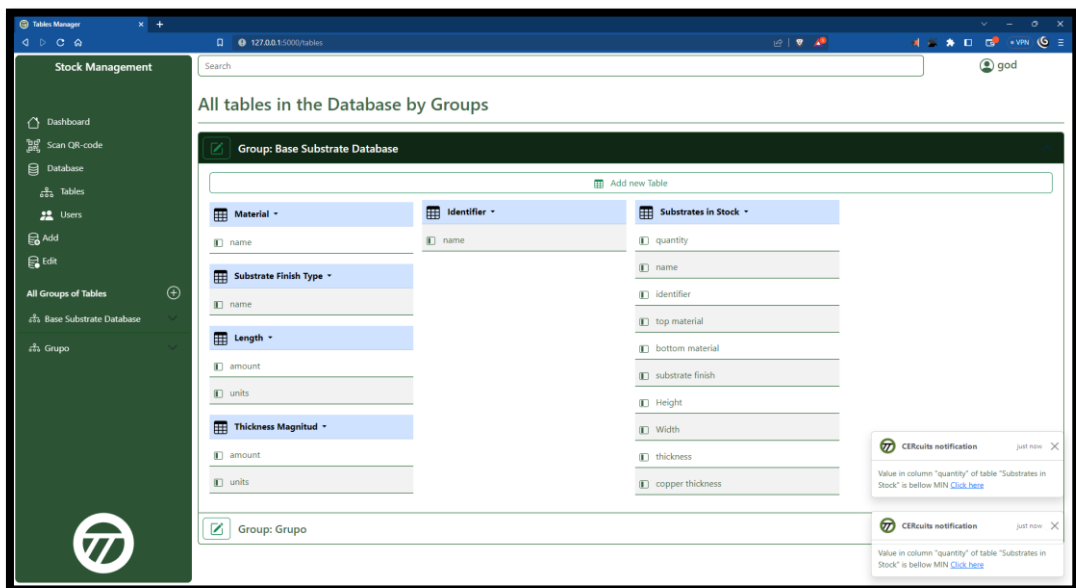
For the edition of the data, now the list of options are displayed in a better way, but still can be too much information for a beginner user.
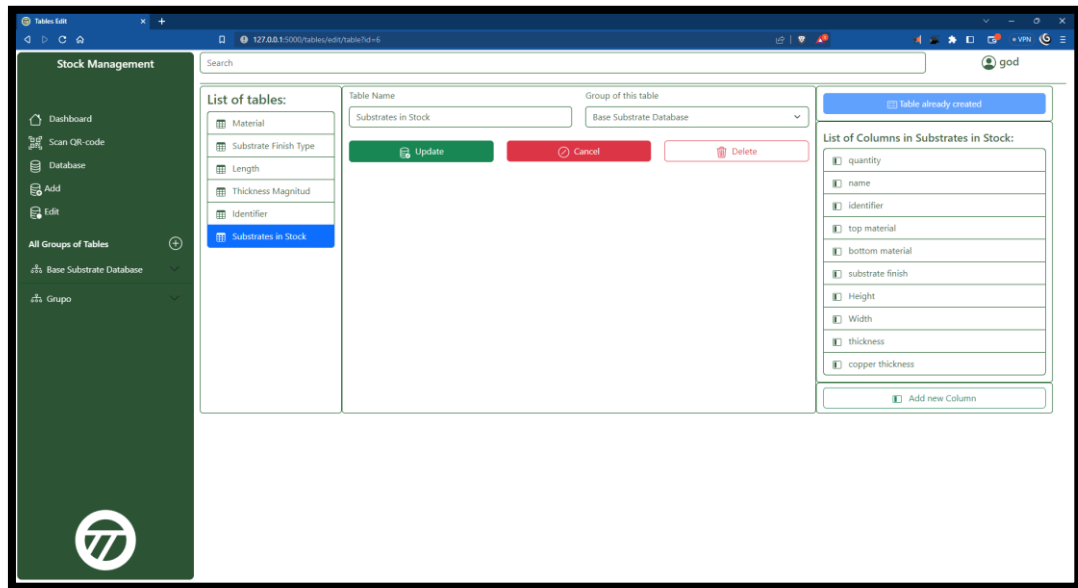
The information displayed in the new tables are even information from others table if a column has a connection or foreign key to another table in one column, when this data want to be edited a 'input select' appears with the options that are all the rows of the pointed table in that column.
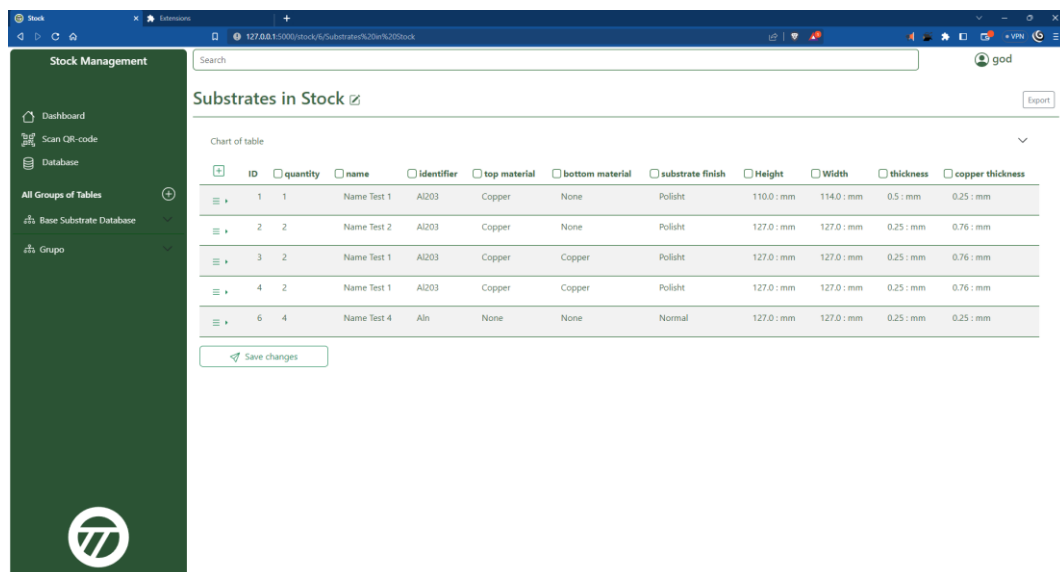

One other feature, is the option to see a chart with the data of 2 columns selected. This is done with a JavaScript library called chart.js and still need to be improved. It does not allows to select more than 2 columns and it won't display a chart when the 2 columns selected are just names.

The chart is in a accordion component of bootstrap, so it can be hide and show clicking in the title "Chart of table".

Also it is possible to edit multiple values in different rows, with not necessity of editing row by row. This is done using AJAX and a special function or route for it.

Once all the changer are done in all the different cells of the table, it is necessary to click the Save changes button so all the changes are save in the database.
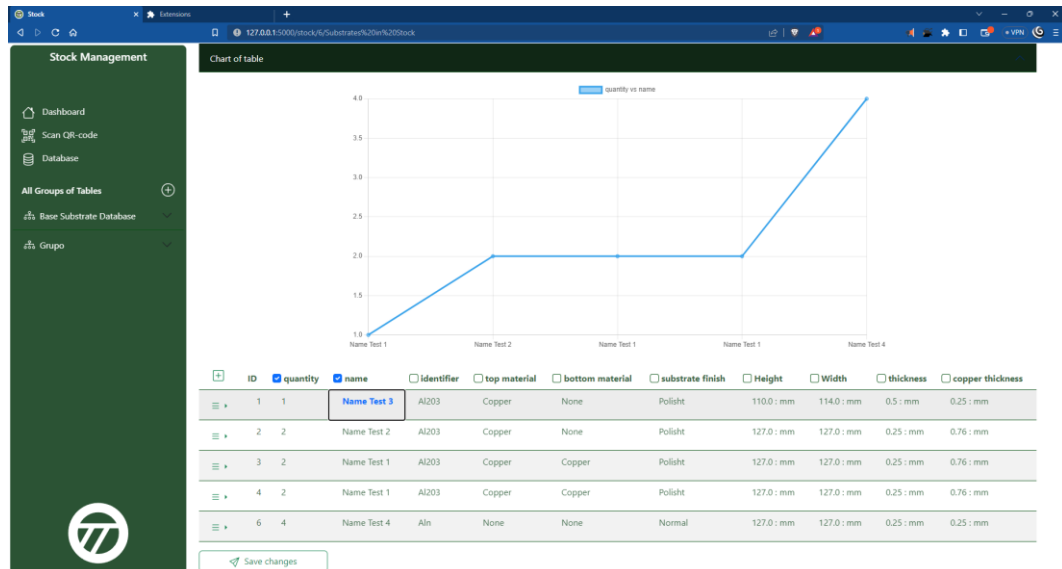
Every row can also be edited in an individual way, in the first column of the table is a button with options for that row.

The option for the row can be edit the data and there also set an alert to get a notification once a value is in a specific limit or a reminder after some time. Another option is to delete the complete row and the first option is to create a QR-code.
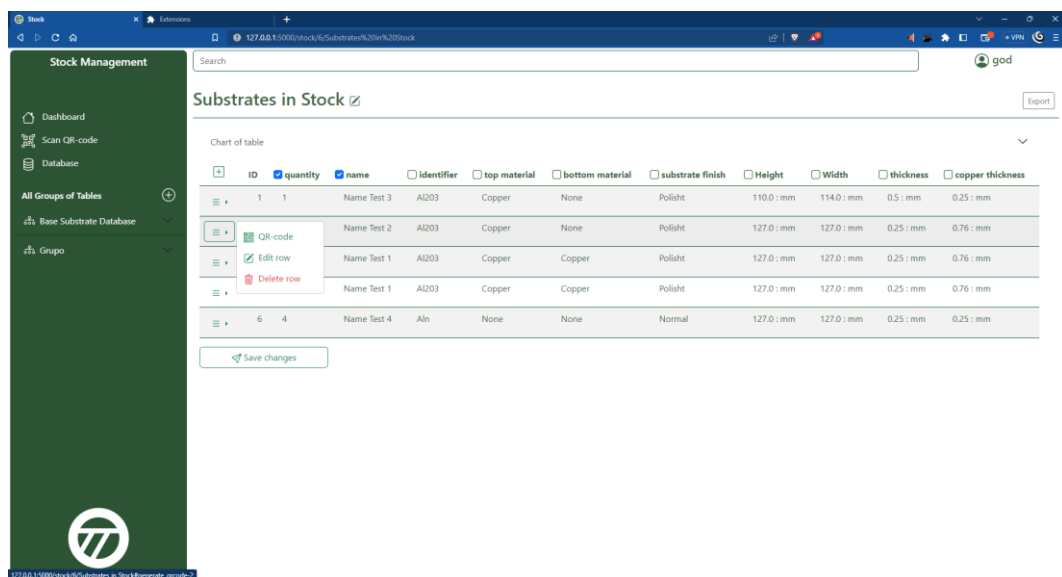
The QR-code is generated by every row, this QR-code contains a link that will take the user to the table with that row, and the row will be shown in the first position of the table.

This QR-code is displayed in a modal component of bootstrap, and is a image that can be downloaded. So the image can be printed and pasted where the stock, item or material is being saved.
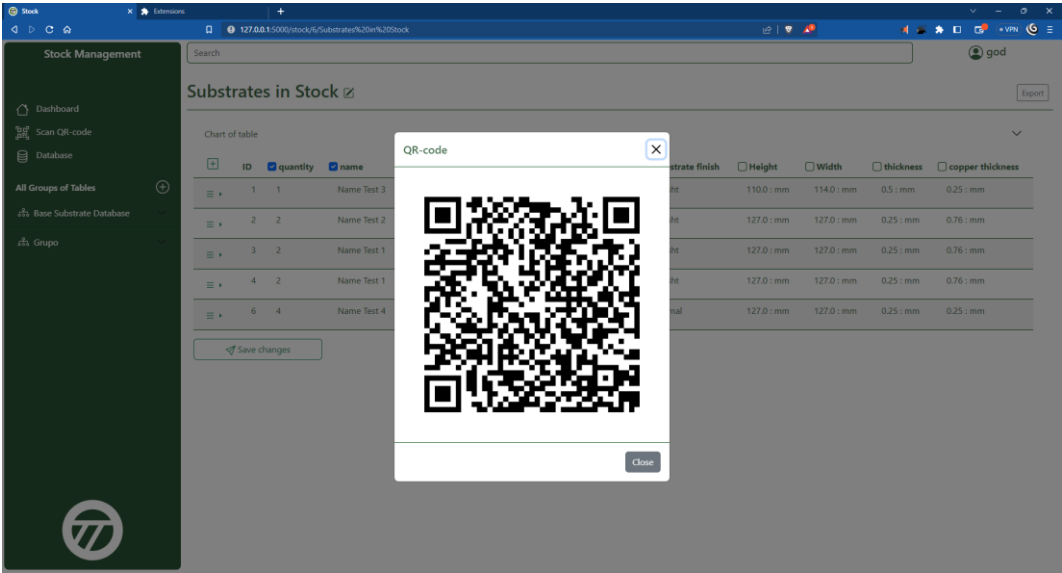
6   CONCLUSIONS

In conclusion, the project underwent significant growth over time, necessitating the restructuring of folders to enhance code comprehension and maintenance. Despite this challenge, we successfully completed the project, achieving all proposed objectives and surpassing expectations.

While meeting the company's requirements, it became evident that there are numerous additional features to be incorporated. These new functions and characteristics have been documented for future iterations of the project.

Throughout the development process, the web application has been undergoing testing and refinement within the company. Each progress demonstration allowed for the collection of new data, leading to the implementation of additional features. We are diligently adding data and thoroughly reviewing the application's behavior to ensure it aligns with the desired expectations.

Undoubtedly, my limited experience in full-stack development initially led to inaccurately estimating the time required to complete certain functions and features. However, I am grateful for the unwavering support and valuable advice provided by my mentor.

The meticulous analysis, well-structured action plan, and clear diagrams played pivotal roles in guiding the development process. These tools ensured that even with the complexity of multiple folders and files, it was always evident which aspects of the web application required development.

Overall, this project served as a valuable learning experience, highlighting the importance of adaptability, mentorship, and careful planning in achieving successful outcomes.

7 REFERENCES

Agner, M. (2023, June 6). *Welcome to Flask-QRcode*. Retrieved from Github pages: https://marcoagner.github.io/Flask-QRcode/

ATLASSIAN. (2023, June 5). *Bitbucket | Git solution for teams using Jira* . Retrieved from ATLASSIAN Bitbucket: https://bitbucket.org/

ATLASSIAN. (2023, June 4). *Collaboration software for software, IT and business teams* . Retrieved from ATLASSIAN: https://www.atlassian.com/

Bootstrap. (2023, June 5). *Dashboard*. Retrieved from Bootstrap: https://getbootstrap.com/docs/5.0/examples/dashboard/

Bootstrap team. (2023, June 6). *Get started with Bootstrap · Bootstrap v5.2*. Retrieved from Bootstrap: https://getbootstrap.com/docs/5.2/getting-started/introduction/

CERcuits. (2023, June 4). *Ceramic PCB and substrate products - Cercuits*. Retrieved from CERcuits: https://cercuits.com/ceramic-pcb-substrates-products/

CERcuits. (2023, June 4). *Ceramic PCB and substrates made easy - CERcuits*. Retrieved from CERcuits: https://cercuits.com/

Conda. (2023, June 5). *Conda documentation*. Retrieved from Conda: https://docs.conda.io/en/latest/

Digital Ocrean. (2023, June 1). *How To Structure a Large Flask Application with Flask Blueprints and Flask-SQLAlchemy*. Retrieved from DigitalOcean: https://www.digitalocean.com/community/tutorials/how-to-structure-a-large-flask-application-with-flask-blueprints-and-flask-sqlalchemy

Ferland, A. (2023, June 5). *Welcome to Flask-MySQLdb's documentation!* Retrieved from Flask-MySQLdb: https://flask-mysqldb.readthedocs.io/en/latest/

Frazier, M. (2023, June 5). *Flask-Login — Flask-Login 0.7.0 documentation*. Retrieved from Read the Docs: https://flask-login.readthedocs.io/en/latest/

git. (2023, June 5). *Git*. Retrieved from git: https://git-scm.com/

gunicorn. (2023, June 5). *Gunicorn - Python WSGI HTTP Server for UNIX*. Retrieved from gunicorn: https://gunicorn.org/

IBM. (2023, June 5). *Class diagram*. Retrieved from IBM: https://www.ibm.com/docs/en/rsm/7.5.0?topic=structure-class-diagrams

Jacob, D. (2023, June 6). *Flask-WTF — Flask-WTF Documentation (1.0.x)*. Retrieved from Read the Docs: https://flask-wtf.readthedocs.io/en/1.0.x/

Mozilla Developer Network. (2023, June 6). *Ajax - Developer guides | MDN*. Retrieved from Mozilla Developer Network Web Docs: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

Mozilla Developer Network. (2023, June 6). *CRUD - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. Retrieved from Mozilla Developer Network Web Docs: https://developer.mozilla.org/en-US/docs/Glossary/CRUD

Mozilla Developer Network. (2023, June 6). *Base64 - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. Retrieved from MDN Web Docs Glossary: https://developer.mozilla.org/en-US/docs/Glossary/Base64

Pallets Projects. (2023, June 5). *Flask Application Context — Flask-SQLAlchemy Documentation (3.0.x)*. Retrieved from Pallets Projects: https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/contexts/

Pallets Projects. (2023, June 5). *Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (3.0.x)*. Retrieved from Pallets Projects: https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/

Pallets Projects. (2023, June 6). *Jinja*. Retrieved from The pallets Projects: https://palletsprojects.com/p/jinja/

Pallets Projects. (2023, June 2). *Project Layout — Flask Documentation (2.3.x)*. Retrieved from Pallets Projects: https://flask.palletsprojects.com/en/2.3.x/tutorial/layout/

Pallets Projects. (2023, June 2). *Welcome to Flask*. Retrieved from Pallets Projects: https://flask.palletsprojects.com/en/2.3.x/

Pallters Projects. (2023, June 5). *Werkzeug — Werkzeug Documentation (2.3.x)*. Retrieved from Pallters Projects: https://werkzeug.palletsprojects.com/en/2.3.x/

python. (2023, June 3). *types — Dynamic type creation and names for built-in types¶*. Retrieved from python: https://docs.python.org/3/library/types.html

Rendek, L. (2023, June 5). *Ubuntu 20.04 list services - Linux Tutorials - Learn Linux Configuration*. Retrieved from LINUXCONFIG.ORG: https://linuxconfig.org/ubuntu-20-04-list-services

SQLAlchemy. (2023, June 5). *Declarative Mapping Styles¶*. Retrieved from SQLAlchemy: https://docs.sqlalchemy.org/en/14/orm/declarative_styles.html#using-a-generated-base-class

SQLite. (2023, June 5). *SQLite Home page*. Retrieved from SQLite: https://www.sqlite.org/index.html

Ubuntu. (2023, June 5). *Install and configure a MySQL server*. Retrieved from Ubuntu: https://ubuntu.com/server/docs/databases-mysql

Visual Paradigm. (2023, June 5). *What is Entity Relationship Diagram* . Retrieved from Visual Paradigm: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/