

# **CSC 154 PROJECT**

## **Group 3**

### **Members**

Trang Tran

Hamzah Alazzeh

Kevin Esquivel

Illya Gordyy

Rodrigo Guzman

## OBJECTIVE

The goal of this project is to create a Role-Based Access Control (RBAC) system on an operating system with three roles: Manager, Engineer, and Human Resources. The system will have specific rules, such as allowing only one Manager and one Human Resources account, but there can be as many Engineer accounts as needed. Users can view, edit, and delete their own files, and Engineers can view files created by other Engineers. Files created by Human Resources will be private, but they can allow the Manager to see certain files. Files created by the Manager will also be private, but it has an option to allow specific users to access them. The RBAC system will ensure that file access is secure and follows these rules. Moreover, we also implemented Task 2 – Two Way Authentication System that manages to store user credentials in a file and send authentication code to users' phone number.

## MEMBER ROLES

Roles available: Project manager, research analyst, technical specialist and document specialist

- Trang Tran – Document Specialist
- Hamzah Alazzeh –Project Manager
- Kevin Esquivel – Document Specialist
- Illya Gordyy – Technical Specialist, Research Analyst
- Rodrigo Guzman – Document Specialist, Reasearch Analyst

**Illya Gordyy:** Implemented a Role-Based Access Control model using the Python Flask framework, along with a SQLite3 database for managing users' data and their files. Moreover, I used HTML, CSS, JavaScript with some Jinja2 features. Python backend manages access based on user roles, ensuring that only authorized users can view or modify specific files according to their assigned roles, such as Manager, Human Resources, and Engineer. Added an authentication that sends a randomly generated 5-digit code to a phone number. Moreover, I designed and developed a website that interacts with the Python server, allowing users to create, edit, and manage files based on their permissions.

**Trang Tran:** Set up and structured the document to include all necessary sections, such as the cover page with all team members' names, the objective, implementation design, project outputs, and limitations and challenges. I reviewed and corrected any grammatical errors in the text to ensure clarity and professionalism. I also applied consistent formatting throughout the document, ensuring uniformity in headings, font styles, and bullet points.

**Hamzah Alazzeh:** Brought the group together, making sure that everyone understood the project goals. I scheduled regular meetings to keep the team on track and to make sure everyone had something to contribute. I also helped with the documentation a little.

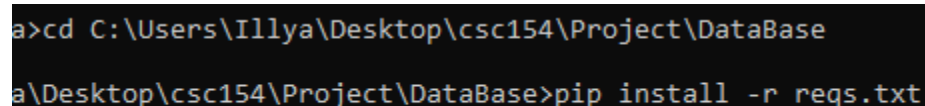
**Rodrigo Guzman:** I followed up on what Trang did and restructured the “Implementation / Design of the Project” portion of the paper. I reorganized this section to improve the flow and clarity of the topics we were going over. This made covering the technical aspects of our project much easier. I also included the relevant images of our code so that our explanations of the steps were seen visually and to add more context to the steps that we were discussing.

**Kevin Esquivel:** Assisted with proofreading the document for minute mistakes and looked over formatting of certain things, as well as providing small feedback on items around the document. Helped organize, format and edit the document and describe some project technicalities. Lastly, helped Rodrigo to structure the implementation and challenges section, and aligned all the screenshots according to their description.

## IMPLEMENTATION / DESIGN OF THE PROJECT

### 1. System Requirements

This project requires the use of Windows since Linux does not give the necessary access that is required for this project. Before executing this application by running the following files, ‘python main.py’ or ‘python3 main.py’, the user must first install all the necessary dependencies, and this is done by running the following command in the terminal: ‘pip install -r reqs.txt’. The ‘reqs.txt’ file is in the project’s directory. The following IP address, ‘127.0.0.1:5000’, is used to locally access the website/database.



```
a>cd C:\Users\Illya\Desktop\csc154\Project\DataBase
a\Desktop\csc154\Project\DataBase>pip install -r reqs.txt
```

### 2. Backend Development

Python flask was used to create the infrastructure for the RBAC system. The backend is used for managing database operations such as storing, updating, and deleting user and/or file data. Below is a rundown of how the database is set up.

Database Setup:

- User database: Stores the necessary user information including ‘user\_id’ (PK), username, password, phone number and ‘role\_id’.

- File database: Manages the files metadata which includes file ownership and access permissions.

#### Core Backend Functions:

- The Flask application handles HTTP requests from the frontend and interacts with the SQLite database. In Turn, it returns the appropriate responses based on the user's role or requests.

```
cur.execute("CREATE TABLE IF NOT EXISTS users ("
            "key INTEGER PRIMARY KEY AUTOINCREMENT, "
            "role_id INTEGER, "
            "name TEXT UNIQUE NOT NULL, "
            "password TEXT NOT NULL,"
            "phone TEXT NOT NULL,"
            "phoneKey TEXT NOT NULL"
            ")")

cur.execute("CREATE TABLE IF NOT EXISTS files ("
            "filenum INTEGER PRIMARY KEY AUTOINCREMENT, "
            "filename TEXT UNIQUE NOT NULL, "
            "owner_id INTEGER NOT NULL,"
            "owner_role_id INTEGER NOT NULL,"
            "view_by_manager BOOLEAN NOT NULL DEFAULT FALSE, "
            "view_by_user TEXT DEFAULT '', "
            "FOREIGN KEY (owner_id) REFERENCES users(key), "
            "FOREIGN KEY (view_by_user) REFERENCES users(key))")

def get_db_connection():
    conn = sqlite3.connect('rbac.db')
    conn.row_factory = sqlite3.Row
    return conn

def key_db():
    conn = sqlite3.connect('key.db')
    conn.row_factory = sqlite3.Row
    return conn

id_dir = {"manager": 1, "human resources": 2, "engineer": 3}

def id_search(cur, role_name):
    cur.execute("SELECT key FROM users WHERE role_id = ?", (id_dir[role_name.lower()],))
    return cur.fetchone()

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
```

### 3. Frontend Integration

Once the backend was finished, the next necessary step was to develop a website interface and link it to the backend.

#### Index Page:

- Is responsible for displaying all existing users and files in the database.
- The front end was designed by using HTML, CSS, and JavaScript, with Flask's Jinja2 templating engine which allows for dynamic content rendering.

```

@app.route('/')
def index():
    conn = get_db_connection()
    cur = conn.cursor()
    table_creation(cur)
    cur.execute("SELECT * FROM users")
    users = cur.fetchall()
    cur.execute("SELECT * FROM files")
    files = cur.fetchall()
    conn.close()
    return render_template('index.html', users=users, files=files)

```

### User Creation:

- A user creation form was also used, allowing users to input their name, password, and phone number. They are also able to select their role e.g. Manager, Human Resources, Engineer.

**Create User**

Name:

Password:

Phone:

Role:

Manager
▼

Create User

[Back](#)

- After a successful user is created, user credentials are stored, and the user is redirected to the index page. If a user already exists or there are other issues when creating a user, then an error message will pop up indicating the problem.

```

def create_user():
    if request.method == 'POST':
        name = request.form['name']
        password = hash_password(request.form['password'])
        phone, phoneKey = encrypt(request.form['phone'])
        role_name = request.form['role_name']
        conn = get_db_connection()
        cur = conn.cursor()

        print(decrypt(phoneKey.decode(), phone.decode()))

        cur.execute("SELECT * FROM users WHERE name = ?", (name,))
        exists = cur.fetchone()
        if exists:
            return "User already exists"

        if role_name.lower() in ["manager", "human resources"]:
            if id_search(cur, role_name):
                return "There can only be a single role: {}".format(role_name)
            else:
                cur.execute("INSERT INTO users (role_id, name, password, phone, phoneKey) VALUES (?, ?, ?, ?, ?)",
                    (id_dir[role_name.lower()], name, password, phone.decode(), phoneKey.decode()))
        elif role_name.lower() == "engineer":
            cur.execute("INSERT INTO users (role_id, name, password, phone, phoneKey) VALUES (?, ?, ?, ?, ?)",
                (id_dir[role_name.lower()], name, password, phone.decode(), phoneKey.decode()))
        else:
            return "Invalid Role: {}".format(role_name)

        # If exists
        if not os.path.exists(USER_DIR):
            os.makedirs(USER_DIR)

        with open(os.path.join(USER_DIR, name + ".txt"), 'w') as file:
            file.write("name: %s\n" % (name))
            file.write("password: %s\n" % (password))
            file.write("phone number: %s\n" % phone)
            file.close

        conn.commit()
        conn.close()
        return redirect(url_for('index', no_match = "", no_password= ""))
    return render_template("create_user.html")

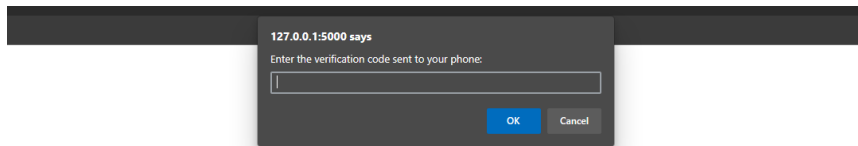
```

## 4. Authentication Process

A big challenge that was overcome was the implementation of the user authentication mechanism.

Login Process:

- Basic login fields such as, username and password
- Two-factor authentication was added to further improve security. This system requires the user to input a 5-digit code that is sent to their phone number. This was done by using the Text Belt SMS API service.
- Challenges faced include managing subscription limitations, handling various login errors and making sure communication between the backend and Text Belt was functioning as intended.
- If successful, the website redirects us to our user page



```
@app.route('/user', methods=['GET', 'POST'])
def login():
    if request.method == "POST":
        username = request.form["username"]
        password = hash_password(request.form["password"])

        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute("SELECT key, password, phone, phoneKey FROM users WHERE name = ?", (username,))
        user = cur.fetchone()

        keyCon = key_db()
        cr = keyCon.cursor()
        cr.execute("SELECT token, key FROM tokenTable")
        encr = cr.fetchone()

        if user:
            if user["password"] == password:
                code = generate_code()
                resp = requests.post('https://textbelt.com/text', {
                    'phone': decrypt(user["phoneKey"], user["phone"]),
                    'message': code,
                    'key': decrypt(incr["key"], encr["token"]),
                })
                print(code)
                session['verification_code'] = code
                session['user_id'] = user["key"]

                return render_template('index.html', show_verification=True)

            return "Password is incorrect"  # TO DO: MAKE THE TEXT BETTER
        else:
            return "No user found"
    return redirect(url_for('index'))
```

```
@app.route('/user/<int:user_id>')
def user_page(user_id):
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT * FROM users WHERE key = ?", (user_id,))
    user = cur.fetchone()
    files_viewable = None

    if user["role_id"] == id_dir["engineer"]:
        cur.execute("SELECT * FROM files WHERE (owner_role_id = ? OR (' || view_by_user || ',' LIKE ?)) AND owner_id != ?", ((id_dir["engineer"], f'{user_id},{user_id}'))
        files_viewable = cur.fetchall()
        cur.execute("SELECT * FROM files WHERE owner_id = ?", (user_id,))
    elif user["role_id"] == id_dir["manager"]:
        cur.execute("SELECT * FROM files WHERE view_by_manager = ?", (True,))
        files_viewable = cur.fetchall()
        cur.execute("SELECT * FROM files WHERE owner_id = ?", (user_id,))
    elif user["role_id"] == id_dir["human resources"]:
        cur.execute("SELECT * FROM files WHERE (' || view_by_user || ',' LIKE ?) AND owner_id != ?", (f'{user_id},{user_id}'))
        files_viewable = cur.fetchall()
        cur.execute("SELECT * FROM files WHERE owner_id = ?", (user_id,))

    files = cur.fetchall()
    conn.close()
    return render_template('user.html', username=user["name"], files=files, files_viewable = files_viewable, key=user["key"], role_id=user["role_id"])
```

## 5. File Management

Different users have different roles and as a result they have different permissions depending on their given role. Depending on the given role they can create files, view files and delete files.

File Creation and Access:

- Managers: Can grant or revoke access to files and certain users.

- Human Resources: Can share files with Managers while maintaining privacy from other roles.
- Engineers: Can view files created by their peers within the Engineer group unless the manager restricts their access.

```
@app.route('/create_file', methods=['GET', 'POST'])
def create_file():
    if request.method == 'POST':
        filename = request.form['filename']
        owner_id = int(request.form['owner_id'])
        owner_role_id = int(request.form['owner_role_id'])
        view_by_user = None
        view_by_manager = False

        conn = get_db_connection()
        cur = conn.cursor()

        if owner_role_id == 2: # Human Resources
            view_by_manager = 'view_by_manager' in request.form
        elif owner_role_id == 1: # Manager
            view_by_user_input = request.form.get('view_by_user')
            cur.execute("SELECT key FROM users WHERE key != ?", (owner_id,))
            users_exist = [str(row['key']) for row in cur.fetchall()] # Convert to strings for comparison
            if view_by_user_input:
                ids = re.split(r'[ ,]+', view_by_user_input.strip())
                valid_ids = [user_id for user_id in ids if user_id in users_exist]
                if valid_ids:
                    if view_by_user:
                        view_by_user += ',' + ','.join(valid_ids)
                    else:
                        view_by_user = ','.join(valid_ids)

        cur.execute("SELECT * FROM files WHERE filename = ?", (filename,))
        exists = cur.fetchone()
        if exists:
            return "File already exists"

        cur.execute("INSERT INTO files (filename, owner_id, owner_role_id, view_by_manager, view_by_user) VALUES (?, ?, ?, ?, ?)",
                    (filename, owner_id, owner_role_id, int(view_by_manager), view_by_user))

        # If exists
        if not os.path.exists(FILE_DIR):
            os.makedirs(FILE_DIR)

        with open(os.path.join(FILE_DIR, filename + ".txt"), 'w') as file:
            file.write("")
            file.close()

        conn.commit()
        conn.close()
        return redirect(url_for('user_page', user_id=owner_id))

owner_id = int(request.args.get('owner_id'))
owner_role_id = int(request.args.get('owner_role_id'))
return render_template('create_file.html', owner_id=owner_id, owner_role_id=owner_role_id)
```

[View our file page](#)



```

@app.route('/files/<filename>/<int:user_id>')
def view_file(filename, user_id):
    try:
        # Read the file content
        with open(os.path.join(FILE_DIR, filename + ".txt"), 'r') as file:
            content = file.read()

        conn = get_db_connection()
        cur = conn.cursor()

        # Get the user's role_id
        cur.execute("SELECT * FROM users WHERE key = ?", (user_id,))
        user = cur.fetchone()
        role_id = user['role_id']
        print(f"Viewing file {filename} by user {user['name']} with role_id {role_id}") # Debugging output

        # Get the file data, including view_by_manager and view_by_user
        cur.execute("SELECT * FROM files WHERE filename = ?", (filename,))
        file_data = cur.fetchone()

        # Initialize available_list and user_names
        available_list = file_data["view_by_user"]
        user_names = []

        if available_list:
            available_list = available_list.split(",")
            for i in available_list:
                cur.execute("SELECT name FROM users WHERE key = ?", (int(i),))
                user_name = cur.fetchone()
                if user_name:
                    user_names.append(user_name["name"])
            else:
                available_list = []

        conn.close()

        return render_template('view_file.html', filename=filename, content=content, user_id=user_id, role_id=role_id,
                               file=file_data, owned_by=file_data["owner_id"], available_list=available_list, user_names=user_names)

    except FileNotFoundError:
        return "File not found", 404

```

## Save our File

```

@app.route('/save_edit', methods=['POST'])
def save_edit():
    filename = request.form['filename']
    user_id = request.form['user_id']
    content = request.form['text']
    role_id = int(request.form['role_id'])

    conn = get_db_connection()
    cur = conn.cursor()

    if role_id == 2: # For human resources
        view_by_manager = 'view_by_manager' in request.form
        cur.execute("UPDATE files SET view_by_manager = ? WHERE filename = ?", (int(view_by_manager), filename))
    # Save the file content
    try:
        with open(os.path.join(FILE_DIR, filename + ".txt"), 'w') as file:
            file.write(content)
        conn.commit()
        conn.close()
        return redirect(url_for('user_page', user_id=user_id))
    except FileNotFoundError:
        return "File not found", 404

```

## File Deletion:

- Users can delete their files from their account page. This action is restricted to files **only** owned by the user that created them.

```
#Delete a file |
@app.route('/delete_file/<filename>', methods=['GET', 'POST'])
def delete_file(filename):
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("DELETE FROM files WHERE filename = ?", (filename,))
    conn.commit()
    conn.close()

    file_path = os.path.join(FILE_DIR, filename + ".txt")
    if os.path.exists(file_path):
        os.remove(file_path)

    return redirect(request.referrer)
```

### Manager Access Revoking:

- The manager can choose to add or revoke access to his files from the account page. This action is restricted to files **only** owned by the manager that created them.

### Giving access to a user

```
@app.route('/add_user', methods = ['POST'])
def add_user():
    filename = request.form['filename']
    user_id = request.form['user_id']

    conn = get_db_connection()
    cur = conn.cursor()

    cur.execute("SELECT key FROM users WHERE key != ?", (user_id,))
    users_exist = [str(row['key']) for row in cur.fetchall()] # Convert to strings for comparison

    cur.execute("SELECT view_by_user FROM files WHERE filename = ?", (filename,))
    result = cur.fetchone()
    users_available = result["view_by_user"] if result and result["view_by_user"] else ""
    view_by_user_input = request.form.get('view_by_user')

    if view_by_user_input:
        ids = re.split(r'[ ,]+', view_by_user_input.strip())
        valid_ids = [user_id for user_id in ids if user_id in users_exist]
        exist = users_available.split(',') if users_available else []
        unique_ids = list(set(valid_ids) - set(exist))
        if unique_ids:
            if users_available:
                users_available += ' ' + ' '.join(unique_ids)
            else:
                users_available = ' '.join(unique_ids)
    cur.execute("UPDATE files SET view_by_user = ? WHERE filename = ?", (users_available, filename))
    print(type(users_available))
    conn.commit()
    conn.close()
    return redirect(request.referrer)
```

## Revoking the access

```
#Delete user from accessing a file
@app.route('/delete_user', methods=['POST'])
def delete_user():
    filename = request.form['filename']
    user_to_delete = request.form['user_to_delete']

    conn = get_db_connection()
    cur = conn.cursor()

    cur.execute("SELECT * FROM files WHERE filename = ?", (filename,))
    file_data = cur.fetchone()

    if file_data:
        owner_id = file_data["owner_id"]
        users = file_data["view_by_user"]

        if users:
            users_list = users.split(",")
            if str(user_to_delete) in users_list:
                users_list.remove(str(user_to_delete))

            updated_users = ",".join(users_list)
            cur.execute("UPDATE files SET view_by_user = ? WHERE filename = ?", (updated_users, filename))

            conn.commit()

    conn.close()

    return redirect(request.referrer)
```

## 6. Challenges and Solutions

Some key issues that had to addressed were:

- Storing User Permissions: Managing user permissions in SQLite3 was difficult due to the lack of native support. This was addressed by storing IDs as comma-separated strings and parsing them during access checks.

```
cur.execute("SELECT view_by_user FROM files WHERE filename = ?", (filename,))
result = cur.fetchone()
users_available = result["view_by_user"] if result and result["view_by_user"] else ""
view_by_user_input = request.form.get('view_by_user')

if view_by_user_input:
    ids = re.split(r'[ ,]+', view_by_user_input.strip())
    valid_ids = [user_id for user_id in ids if user_id in users_exist]
    exist = users_available.split(',') if users_available else []
    unique_ids = list(set(valid_ids) - set(exist))
    if unique_ids:
        if users_available:
            users_available += ',' + ','.join(unique_ids)
        else:
            users_available = ','.join(unique_ids)
```

- Phone Number Encryption: Encrypting and securely storing phone numbers for 2FA required translating data to base64 encoding for reliable database storage.

```

#Encryption
def encrypt(phone): #Encrypts and returns a key along with the encrypted phone, also translates it to base64 for save SQL storing
    key = Fernet.generate_key()
    fernet = Fernet(key)
    encryptedPhone = fernet.encrypt(phone.encode())
    return base64.urlsafe_b64encode(encryptedPhone), base64.urlsafe_b64encode(key)

#Decryption using key and base64 for save SQL storing
def decrypt(key, phone):
    key = base64.urlsafe_b64decode(key)
    phone = base64.urlsafe_b64decode(phone)
    fernet = Fernet(key)
    return fernet.decrypt(phone).decode()

```

- SMS Authentication: Selecting an appropriate SMS service was very important for a reliable way of implementing 2FA. Text Belt was used for this issue due to the simplicity and API integration regardless of the initial issues setting it up.

```

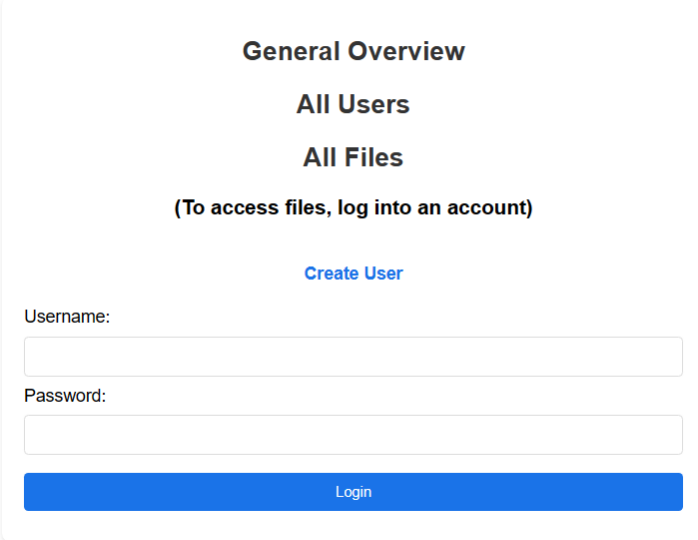
resp = requests.post('https://textbelt.com/text', {
    'phone': decrypt(user['phoneKey'], user["phone"]),
    'message': code,
    'key': decrypt(encr["key"], encr["token"]),
})

```

- Personal Challenges: Due to everyone's availability being different a big challenge we faced was setting up meeting times, since everyone's schedules were all over the place. We found that the best time to work on the project together was after this class's meeting time.

## 7. PROJECT OUTPUT

- Main page, also functions as the logging page



The image shows a web form titled "General Overview". Below the title are three lines of text: "All Users", "All Files", and "(To access files, log into an account)". There is a blue link "Create User" below the text. The form has two input fields: "Username:" and "Password:". Below the password field is a blue "Login" button.

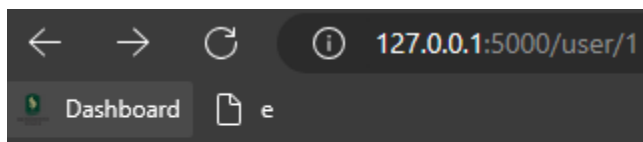
- In case if we enter credentials of a non-existing user

No user found

- In case if the user exists, but the password is incorrect

Password is incorrect

- If a client tries to access a user page with a link. For example, /user/1



Unauthorized

- Account Creating page

### Create User

Name:

Password:

Phone:

Role:

Manager

Create User

Back

- Our main page after creating a user

### General Overview

#### All Users

ID: 1. Illya (Role ID: 1)

#### All Files

(To access files, log into an account)

Create User

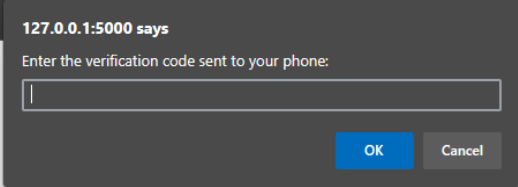
Username:

Password:

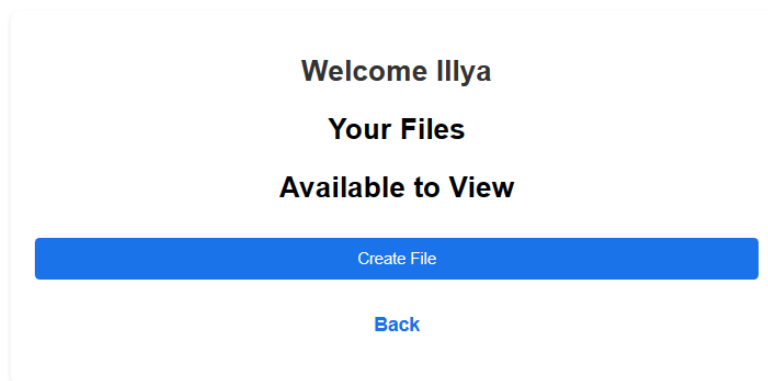
Login

- When logging in it prompts us to enter a secret 5-digit code send to our real phone and to the python terminal for debugging purposes.

```
127.0.0.1 - - [09/Aug/2024 19:57:54] "GET /static/styles.css HTTP/1.1" 34180
127.0.0.1 - - [09/Aug/2024 19:58:33] "POST /user HTTP/1.1" 200 -
```



- User main page



- File creation page

### Create File

Filename:

Viewable by User ID:

Create File

[Back](#)

- User page after creating a file

### Welcome Illya

### Your Files

× File (Owner ID: 1)

### Available to View

Create File

[Back](#)



- Main page when a second user is created

General Overview

All Users

ID: 1. Illya (Role ID: 1)

ID: 2. HR (Role ID: 2)

All Files

(To access files, log into an account)

File (Owner's ID: 1)

Create User

Username:

Illya

Password:

...

Login

- After going back to user Illya and entering a page for a file named File we can a user to view by entering his ID

File: File

He11d

Save Changes

Viewable by User ID:

2

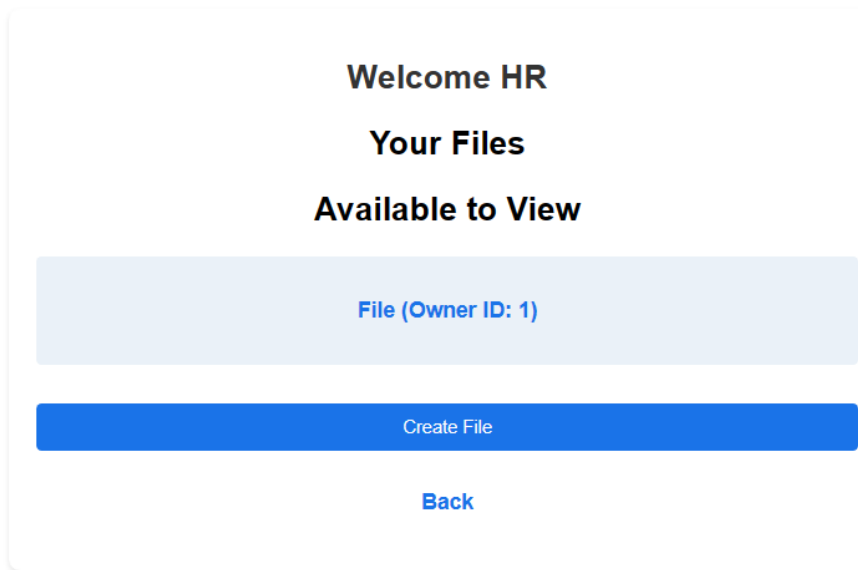
Add User

Back

Already have access to this file

ID: 2 User: HR

- User HR can only view the file



- Now for example the HR gives view access to manager

**File: FileByHR**

Hello Manager

Viewable by Manager:

☒

Save Changes

Back

- Manager's view

**Welcome Illya**

**Your Files**

**Available to View**

FileByHR (Owner ID: 1)

Create File

Back

- For example, an engineer creates a file

**Welcome Eng**  
**Your Files**

✖ File\_by\_Eng (Owner ID: 5)

**Available to View**

Create File

Back

- Another engineer can also view this file

**Welcome Engineer**  
**Your Files**  
**Available to View**

File\_by\_Eng (Owner ID: 5)

Create File

Back