Akademia Górniczo-Hutnicza w Krakowie

Wydział Fizyki i Informatyki Stosowanej

Python in the Enterprise

Security System

Andrzej Świętek
Marcin Knapczyk
Mateusz Wawrzyczek
Bartosz Biesaga

28.01.2025

Contents

1 Project description

The project is a comprehensive security and access management system designed to help enterprises efficiently control and monitor their facilities, users, and security assets. It integrates user authentication, role-based access control (RBAC), surveillance camera management, and facial recognition to ensure high-security standards and streamlined administration.
This system is particularly useful for companies managing multiple buildings, restricted zones, and large user bases, where security automation, controlled access, and real-time monitoring are critical. By leveraging modern authentication mechanisms, including OAuth2 with third-party integrations (Google, GitHub), and advanced AI-driven facial recognition, the platform ensures both security and convenience.
Our project focuses on developing an efficient AI-based security system designed specifically for office spaces. By leveraging facial and silhouette recognition, the system verifies employee identities at various access points throughout a facility, creating a secure and streamlined process for managing entry and monitoring activities. The primary aim is to establish a robust, real-time, automated verification system that safeguards access to restricted areas.

The source code of the project is located in the GitHub repository, available here.

In addition, a brief demo video presenting key functionalities: Video.

2 Core Functionalities

To fulfil project assumptions we decided to include the following:

- web application developed with django framework, which allowed us to quickly create modules needed for:
  - registration/login and profile management – allows user to upload and view uploaded photos,
  - user CRUD,
  - company CRUD, assigning users to companies and managing their position (firing, demoting/promoting),
  - building CRUD,
  - zone CRUD,
  - camera CRUD, camera feed view and upload
  - permission CRUD, listing users/zones with associated permissions, managing user's/zone's permissions.
- feature extraction from user's face and silhouette photos. Described in detail in this section.
- PostgreSQL database for managing listed entities with PGVector extension for storing face and silhouette features extracted from user uploaded images and then calculating distances between embeddings stored in database and those extracted from camera feed photos.

3 Project Management and Development Process

The development of the security management system was carried out using **Agile methodologies**, ensuring an iterative and flexible approach to project execution. Our workflow was structured to promote **collaboration, transparency, and continuous improvement**, with well-defined processes for **task management, version control, and code review**.

3.1 Agile Workflow and Task Management

To efficiently manage development tasks, we adopted the **Kanban methodology**, which allowed us to maintain a clear overview of progress and prioritize work effectively. Our process included:

- **Weekly Team Meetings** – We held internal **weekly meetings** to discuss development progress, address challenges, and refine our roadmap.
- **Weekly Review Sessions** – Additionally, structured project review meetings took place during class sessions, ensuring alignment with project goals.
- **Task Prioritization & Breakdown** – Tasks were split into smaller, well-defined objectives, following Agile best practices to maintain continuous delivery.
- **Kanban Board** – We used a **Kanban-based task management system**, where tasks were categorized into:
  - **Backlog** (planned but not started)
  - **In Progress** (actively being worked on)
  - **Code Review** (awaiting approval)
  - **Completed** (successfully merged and deployed)

This approach ensured that development was always **structured, efficient, and adaptable** to evolving project needs.

3.2 Version Control and Collaboration (GitHub)

The project was **hosted on GitHub**, where we employed **branching strategies and pull request workflows** to maintain high code quality and team collaboration. Our version control strategy was based on:

3.2.1 Branching Model

The repository followed a structured branching model:

- **main** – Stable, production-ready code
- **dev** – Active development branch
- **feature/<name>** – Dedicated branches for new features
- **bugfix/<name>** – Hotfix branches for resolving issues

3.2.2 Pull Request Workflow & Code Review

- Every feature or fix was implemented on a separate **feature branch**, which was then submitted as a **Pull Request (PR)**.
- PRs required **at least one approval and a thorough code review** before being merged. This ensured:
  - High-quality, maintainable code
  - Consistency in coding standards
  - Prevention of regression or conflicts

3.2.3 Continuous Integration & Merging

- Merging into the **develop** branch was only allowed after passing **code review and team approval**.
- The **main** branch was protected to prevent unapproved direct modifications.

3.3 Key Benefits of Our Development Approach

- **Clear task ownership** through structured Kanban workflow
- **Consistent progress tracking** via weekly meetings and Agile task management
- **Code integrity and maintainability** ensured through mandatory pull requests and reviews
- **Efficient team collaboration** using GitHub branches and structured merging

By adhering to Agile methodologies, maintaining **strict version control**, and emphasizing **team collaboration**, we ensured that our project development was **efficient, well-organized, and high-quality**.

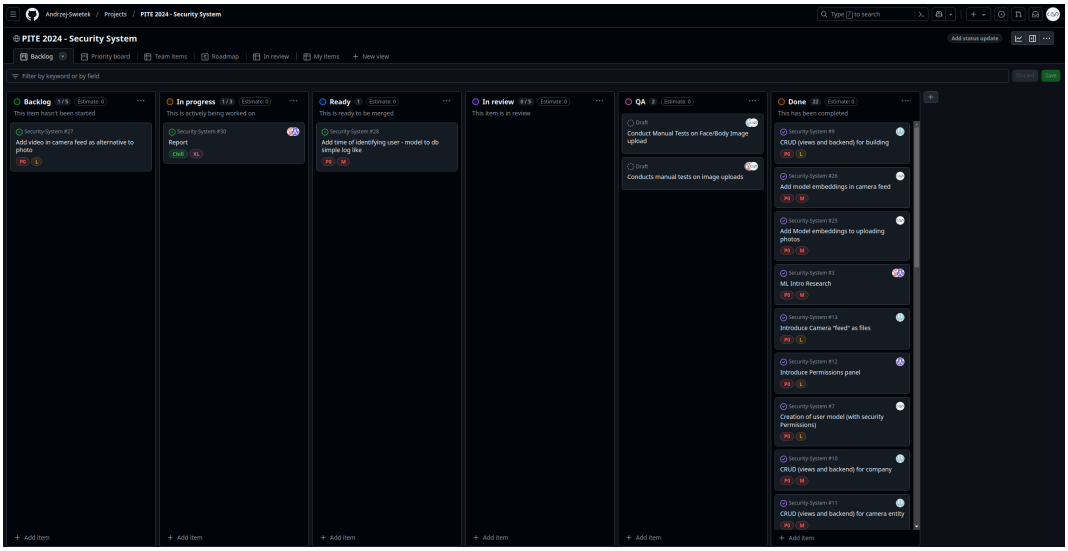

Figure 1: Kanban

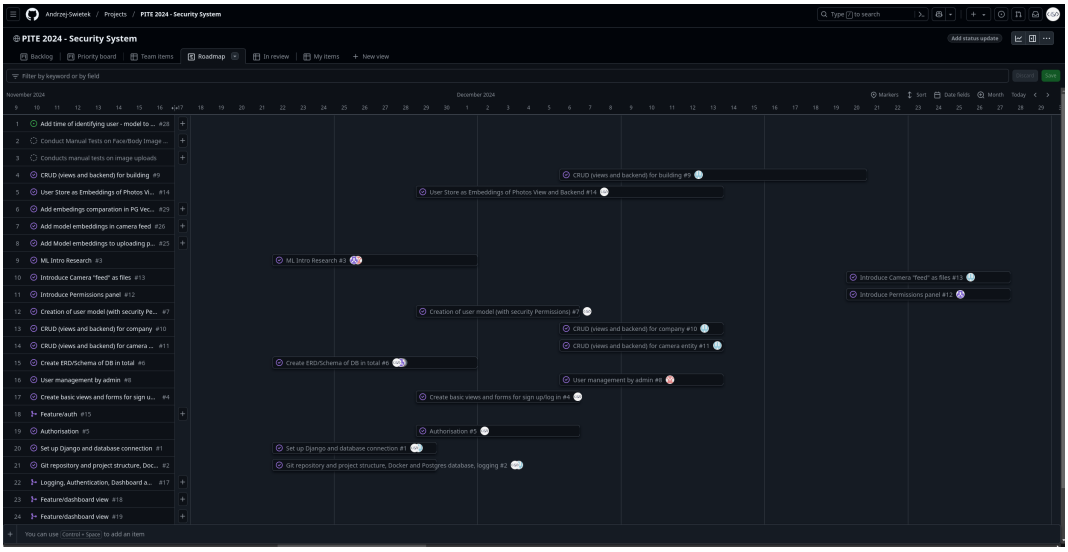

Figure 2: Road Map

4 Database structure - Entity Relations Diagram

The database schema is designed to efficiently store and manage security-related data while maintaining logical relationships between entities. The Entity Relationship Diagram (ERD) (Figure 3) visually represents the core entities, their attributes, and the relationships between them, ensuring data integrity and seamless system interactions.
The schema is structured around several key domains:

- Authentication & User Management
  - auth_user: Stores user credentials and authentication details.
  - account_emailaddress, account_emailconfirmation: Handle email verification and authentication-related operations.
  - authentication_userprofile, authentication_userimage: Extend user data with additional profile details and images.
- Building & Zone Management
  - buildings_building: Represents individual buildings under security surveillance.
  - buildings_company: Links buildings to organizations managing them.
  - buildings_zone: Defines security zones within buildings.
- Camera & Surveillance
  - cameras_camera: Stores information about security cameras, including their locations and configurations.
  - cameras_camerafeed: Tracks recorded footage and live feed metadata.
- Access Control & Permissions
  - permissions_permission: Defines security permissions.
  - permissions_permission_users, permissions_permission_zones: Manage access rights for users and security zones.

These relationships enable robust role-based access control (RBAC), real-time security monitoring, and audit trail logging.

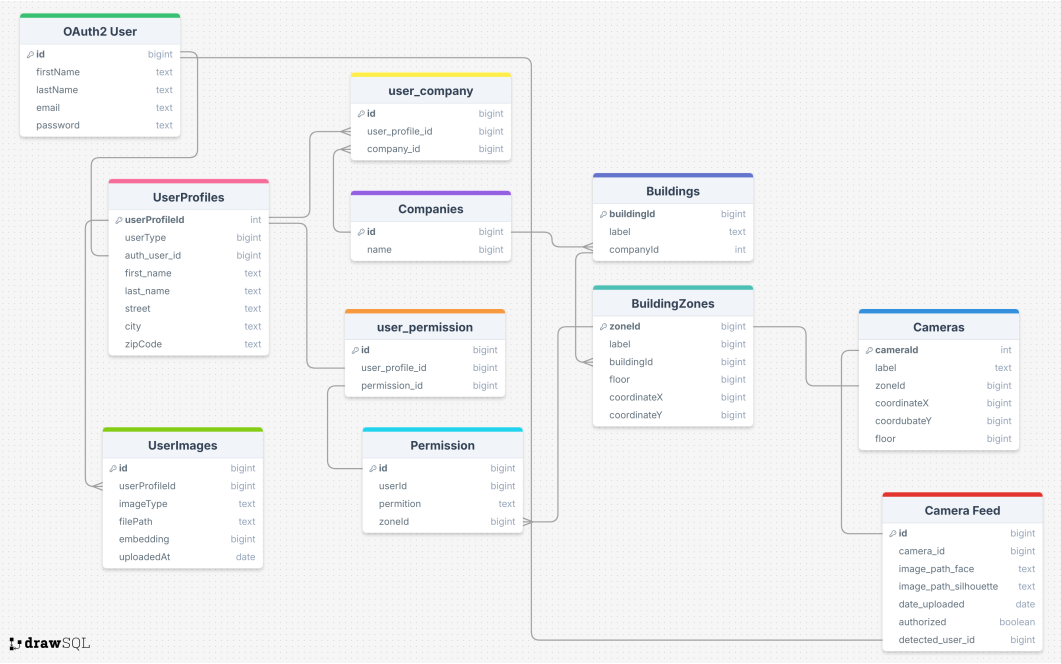

Figure 3: ERD

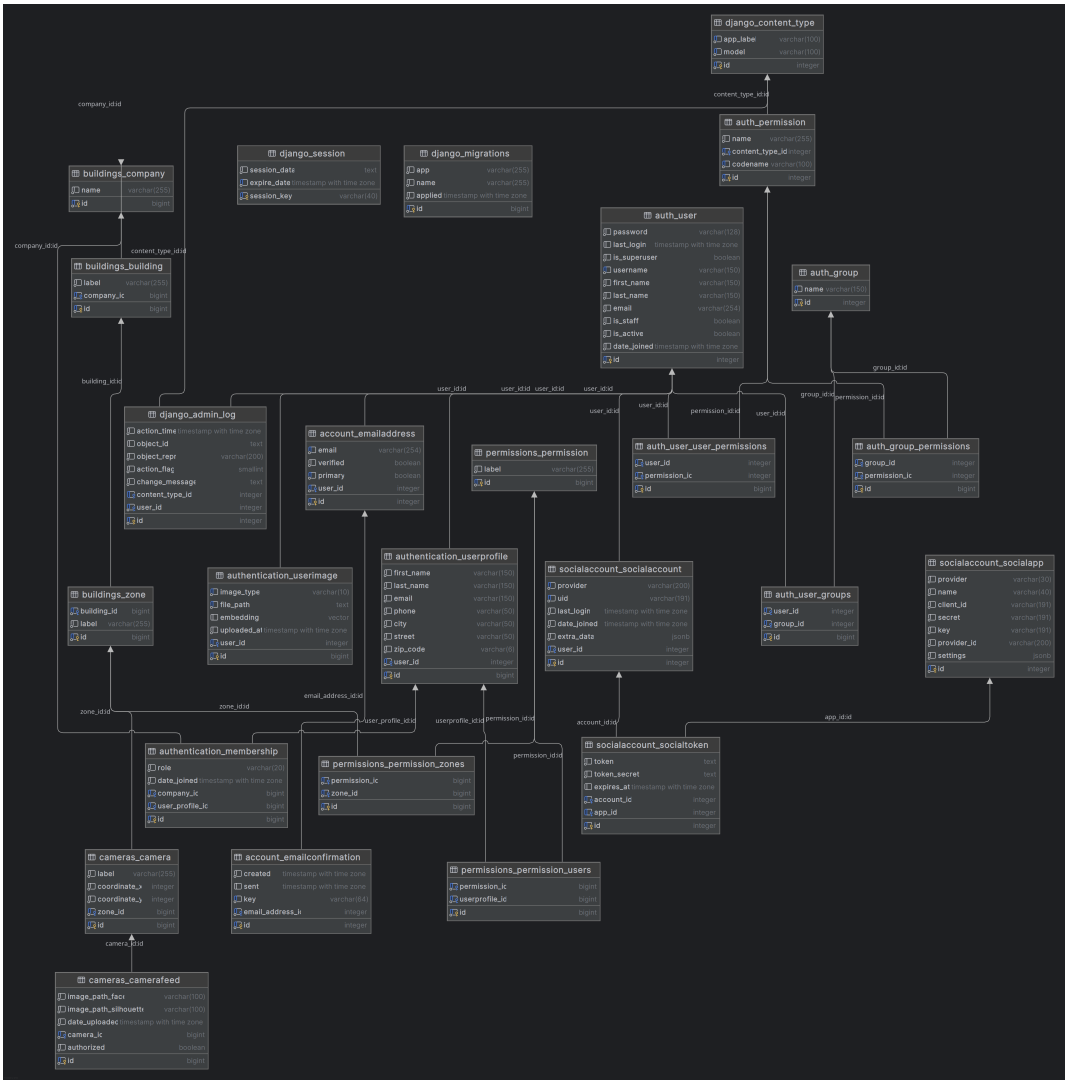

Figure 4: Full Database Entity-Relations Diagram

5 Containerization and modularization

To ensure scalability, portability, and maintainability, the system is containerized using Docker. Each module runs as an independent container, allowing for better resource management and isolation. The key modules include:

- Django Backend: Runs as a containerized service, handling authentication, security event processing, and API endpoints along wiht frontend views.
- Database Service: A PostgreSQL container is used for structured data storage, ensuring reliable and secure transactions. Moreover it enables developer teams to easly set up all database required extensions such us PG Vector.

A Docker Compose file orchestrates these services, simplifying deployment and scaling across different environments. The modular approach ensures that each component can be updated, tested, or replaced without affecting the entire system.
Additionally, the project's folder structure adheres to Django's best practices, following a modular monolith approach. Key Django applications (modules) include:

- authentication – Manages user authentication, profiles, and permissions.

- buildings – Handles building and security zone data.
- cameras – Manages security camera configurations and feeds.
- recognitions – (Potential) Module for face or object recognition within surveillance feeds.
- user_management – Provides administrative controls over system users.

Service	Port
Django	:8000
PG Vector (PostgreSQL)	:5432

Table 1: List of Services and Their Ports

This modular architecture allows for easier maintainability, scalability, and extensibility, ensuring the security system remains robust and adaptable.

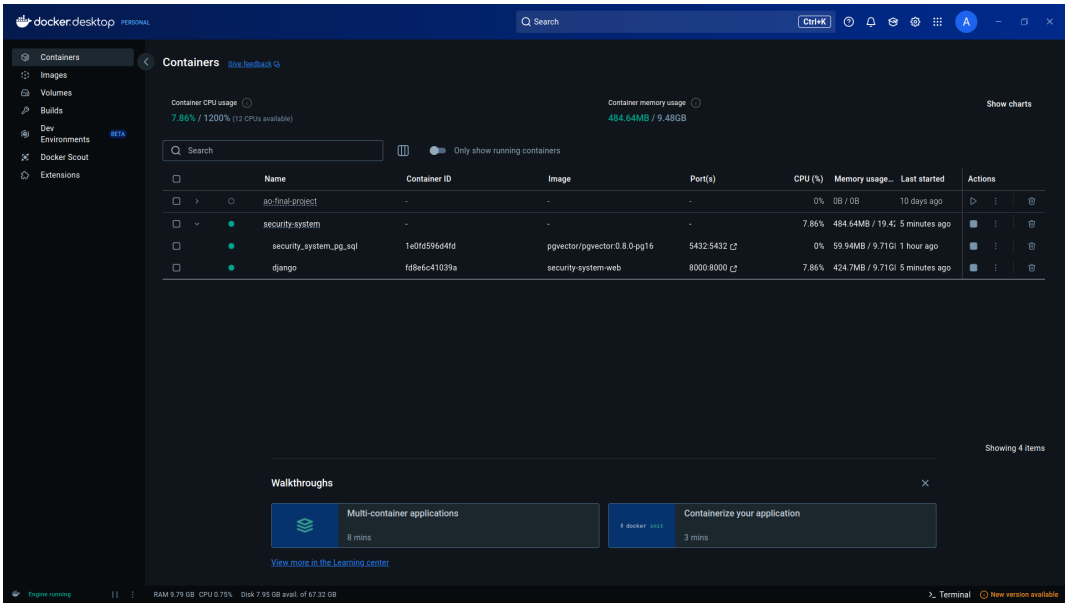

Figure 5: Docker Desktop view

6 Authentication and OAuth2 Integration

The security system implements **OAuth2-based authentication** to provide a secure and flexible login mechanism. Users can authenticate using two primary methods:

1. **Email and Password Authentication** – Traditional login and registration using credentials stored in the system.
2. **Third-Party Authentication** – OAuth2-based authentication via external identity providers, such as **Google** and **GitHub**.

6.1 OAuth2 Workflow

OAuth2 is an industry-standard protocol for secure authorization. It enables authentication without exposing user credentials to the application, enhancing security and user convenience. The authentication flow follows these key steps:

1. **User Initiates Login**
   - If logging in with email and password, credentials are submitted via the login form.
   - If using a third-party provider, the user is redirected to the provider's OAuth2 authorization page.
2. **Authorization and Token Exchange**
   - For third-party login, the provider (e.g., Google or GitHub) requests user consent to share profile information.
   - Upon approval, the provider issues an *authorization code* that our backend exchanges for an *access token*.
   - The system validates this token and retrieves user details.
3. **User Creation or Login**
   - If the user exists in the database, they are logged in.
   - If logging in for the first time with a third-party provider, a new account is created using the retrieved profile data.
4. **Session Management**
   - A secure session or JWT (JSON Web Token) is generated and returned to the frontend for subsequent authenticated requests.

6.2 Supported Authentication Methods

Authentication Method	**Description**
Email & Password	Users register and log in using their email and a secure password.
Google OAuth2	Users log in via their Google account, leveraging Google's authentication system.
GitHub OAuth2	Users log in using their GitHub account credentials.

Table 2: Supported Authentication Methods

6.3 Implementation Details

- **Django Allauth**: The authentication system is built using `django-allauth`, which provides seamless OAuth2 integration and email-based authentication.
- **OAuth2 Providers**: The system is configured to support both **Google** and **GitHub**, allowing users to authenticate via their preferred provider.
- **Secure Token Storage**: OAuth2 tokens are securely stored and managed, preventing unauthorized access.
- **Automatic Profile Linking**: If a user registers with an email and later logs in via Google or GitHub with the same email, the accounts are linked automatically.

This approach ensures a **secure, user-friendly, and scalable authentication system**, providing flexibility for users while maintaining high security standards.

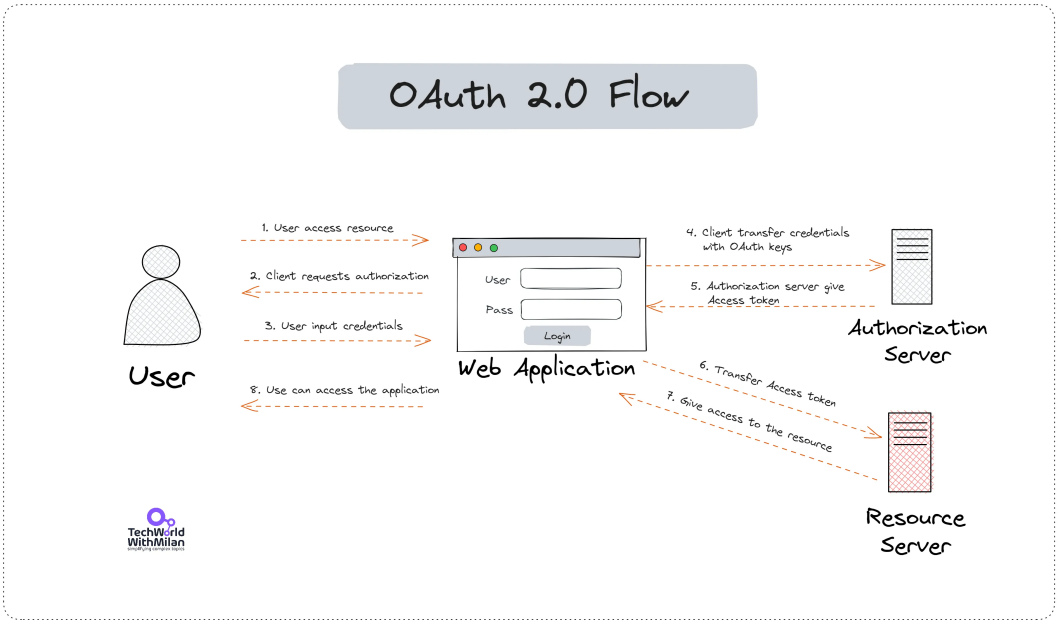

Figure 6: OAuth 2.0

7 User manual and data management

7.1 Registration/Login

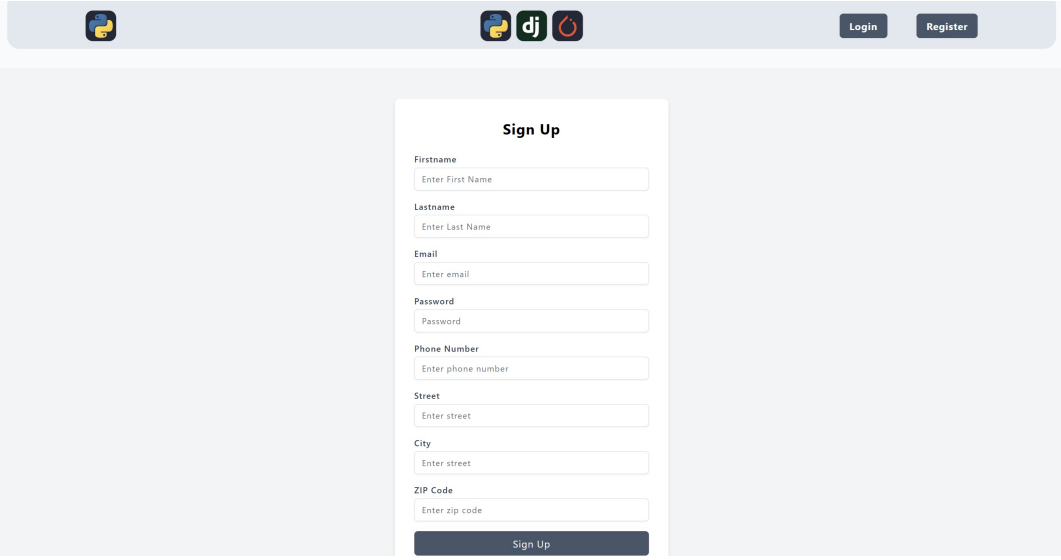

Figure 7: Registration screen

There are two ways to log in, by creating an account (Figure 1) and using a Github account (Figure 2).

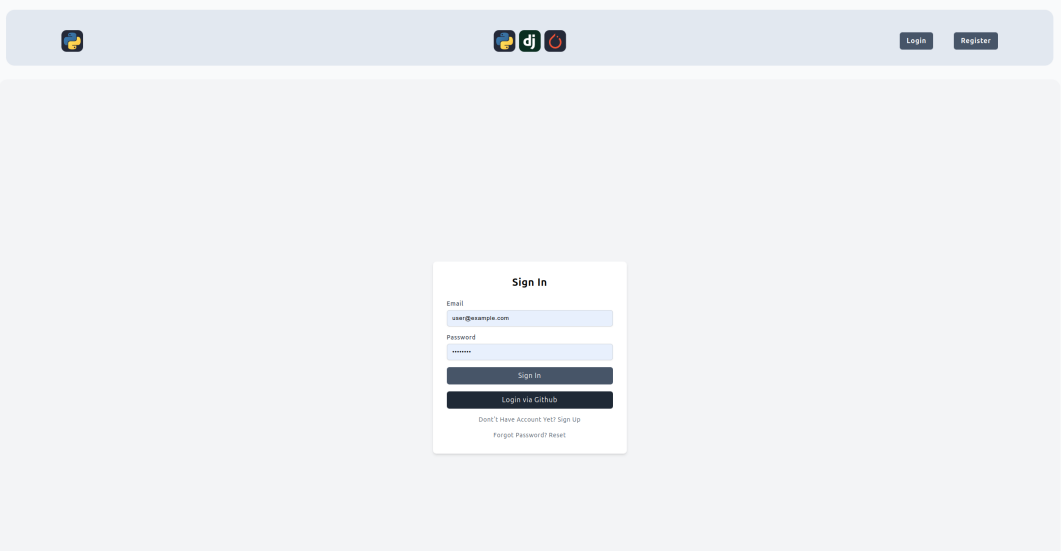

Figure 8: Login screen

Without logging-in, the user has access only to browsing information about buildings and permits. To perform actions such as delete, add, or edit, the user must be logged in, also must have permissions granted.

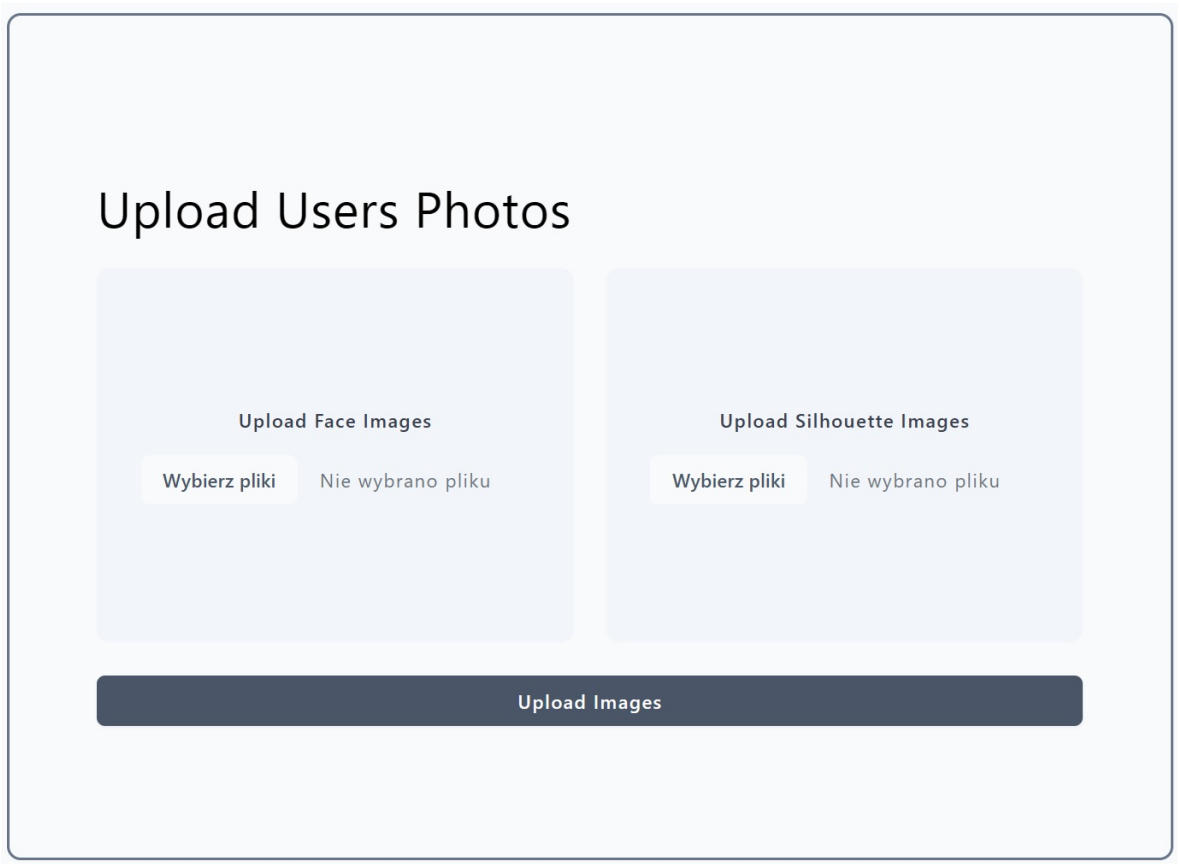

Figure 9: Uploading photo screen

A regular user, after logging in/registering, has access to their account and the permissions granted to them. He can upload photos, which will be used for access to buildings/zones/rooms.

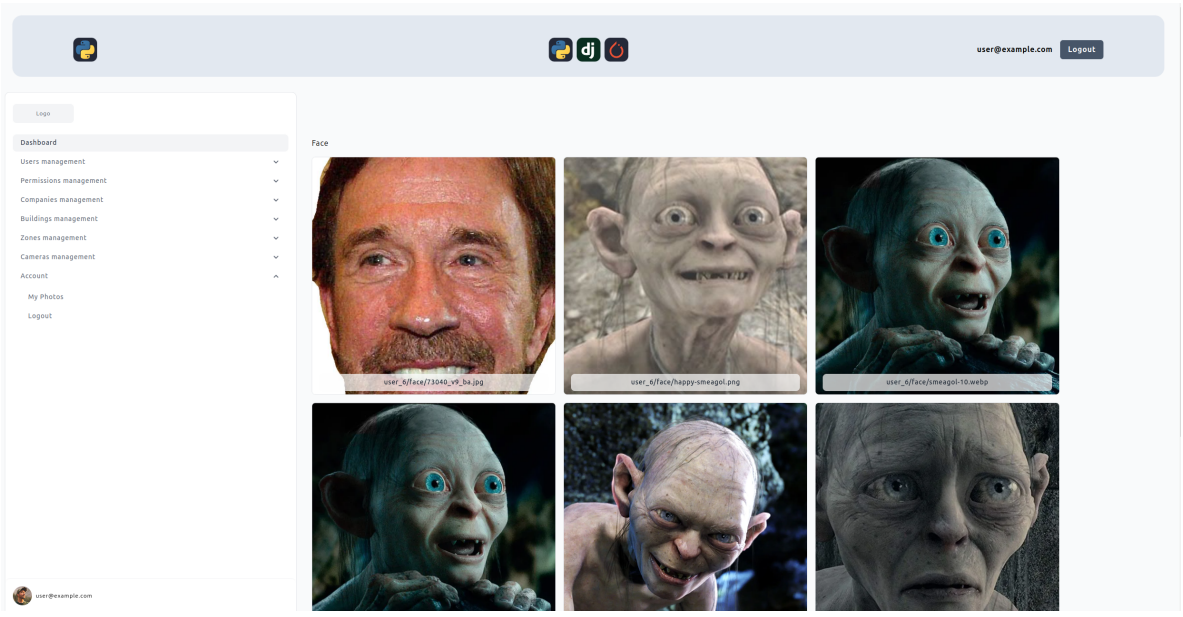

Figure 10: Uploaded Photos screen

He can also list entities but does not have permission to edit them. When the employee doesn't have

access to the function, he gets a special announcement. Every single user has access only to his own photos.

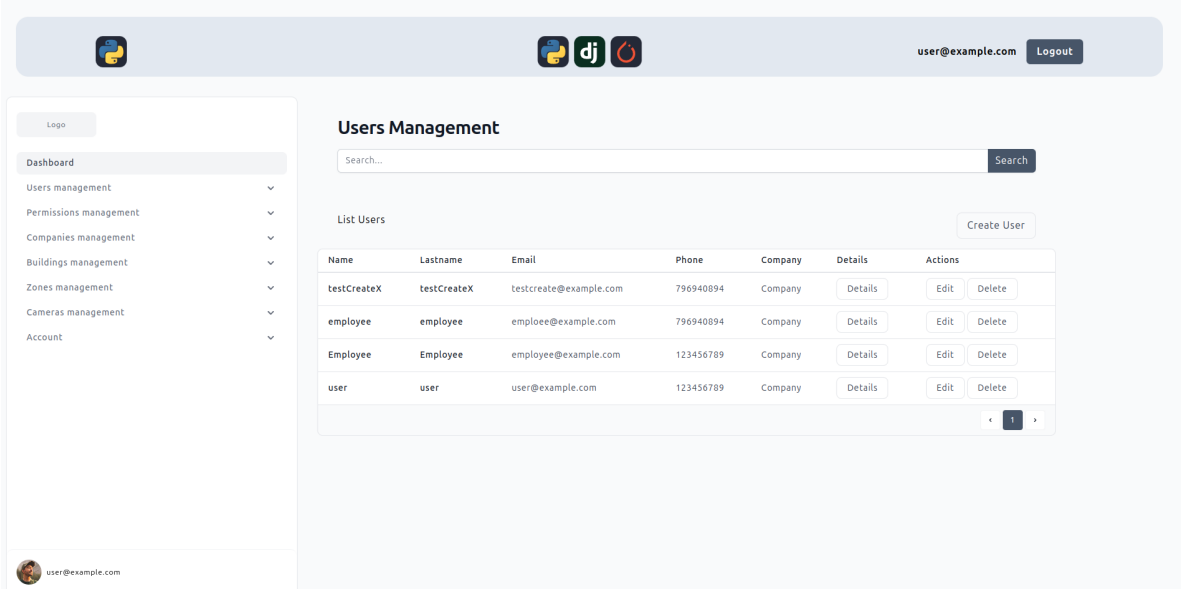

Figure 11: User list

All user information is listed in the table and also by clicking details.

7.2 Data in the app

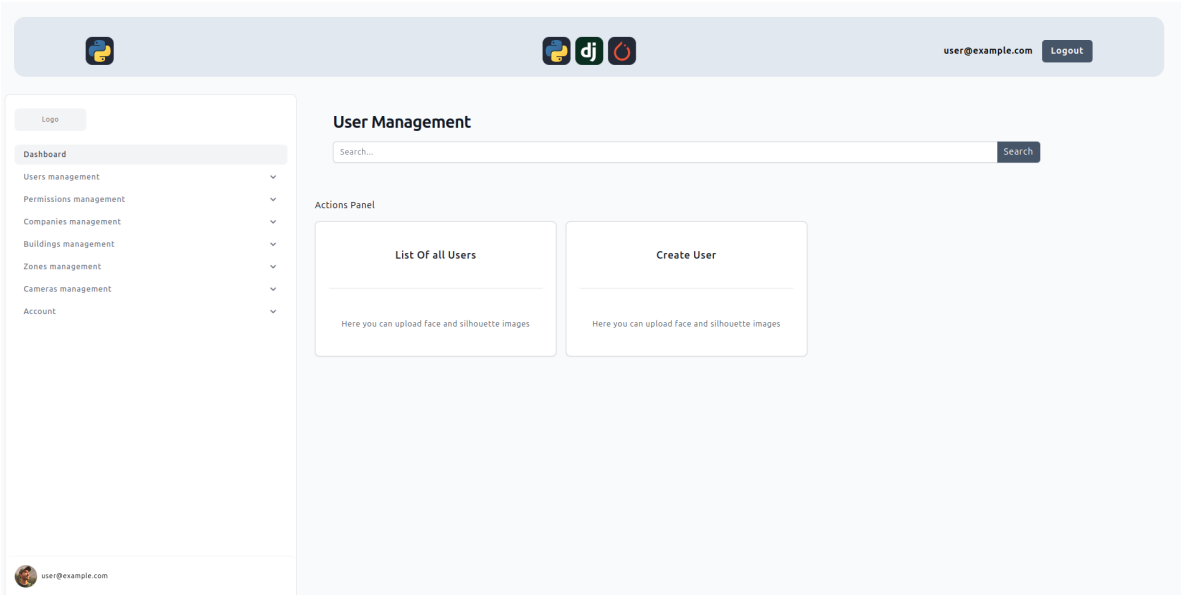

Figure 12: User management screen

Data editing within a company is the responsibility of the admins. One of their roles is user management. They can create users, view the user list, and delete individual users. They have access to all user data (first name, last name, phone number, email, address, postal code). Admins also are responsible for assigning users to company.

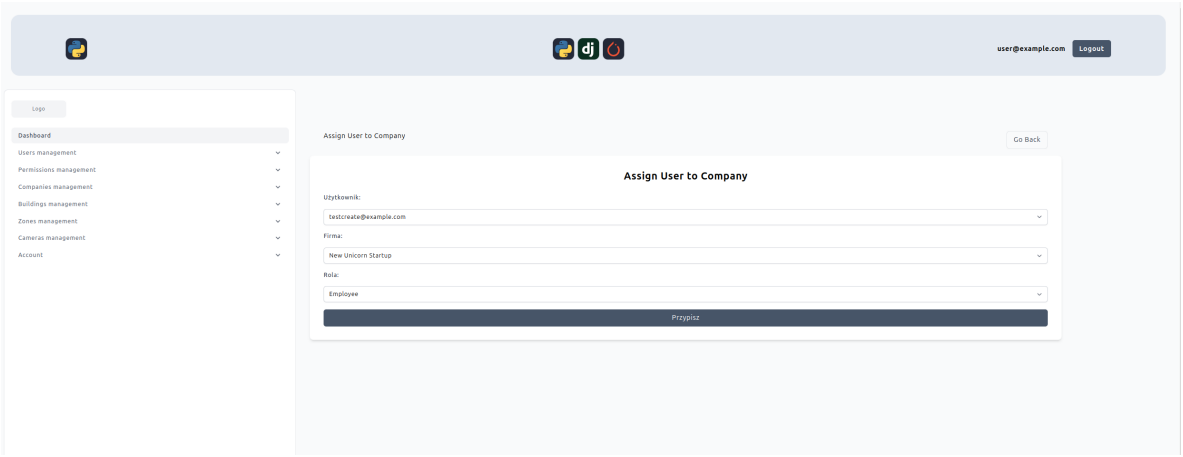

Figure 13: Assign User to Company

7.2.1 Buildings

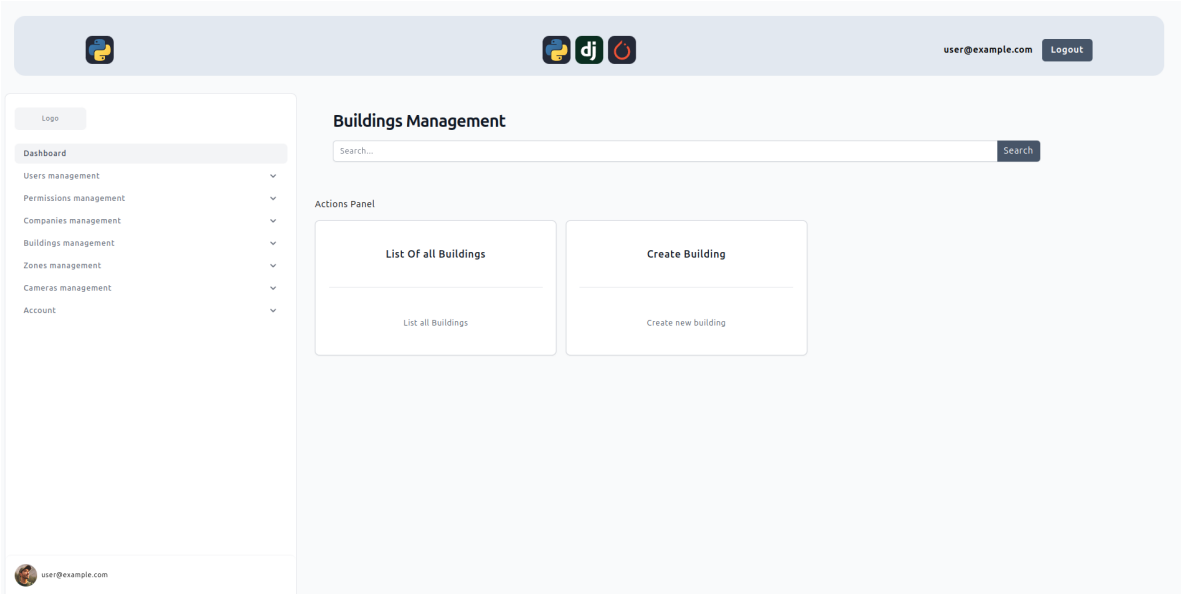

Figure 14: Buildings management screen

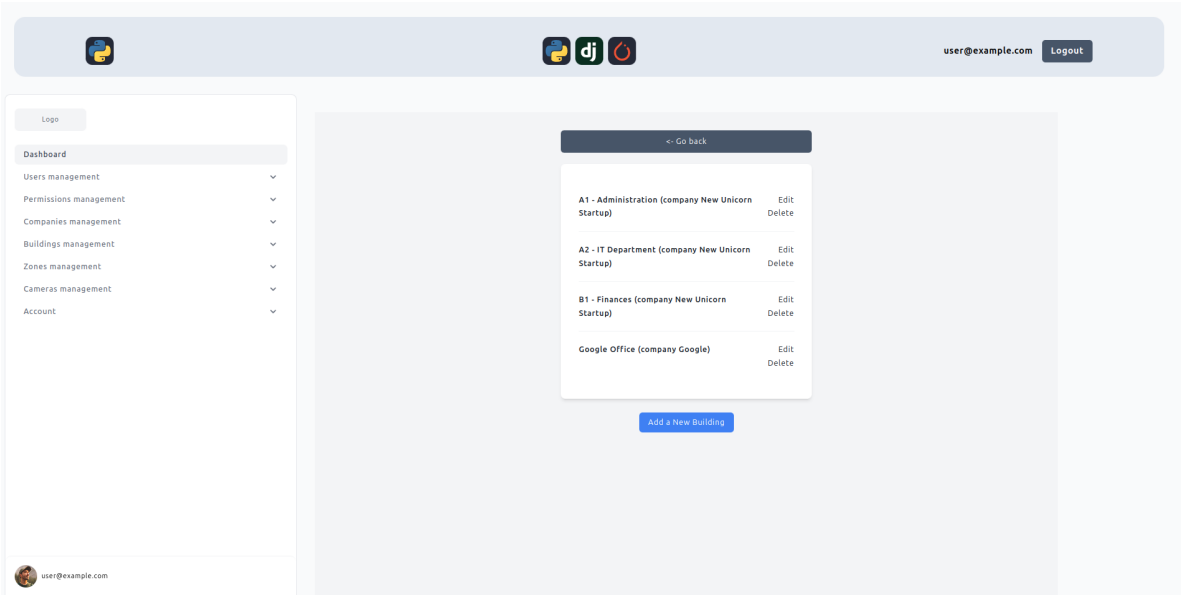

Figure 15: Buildings management screen

Similarly, as in the case of user management, admin is responsible for editing and adding data (Figure 15). Buildings are assigned to specific companies. In the buildings section, there is a basic view from which you can go to the list of buildings and also to the view for creating new buildings.

7.2.2 Zones

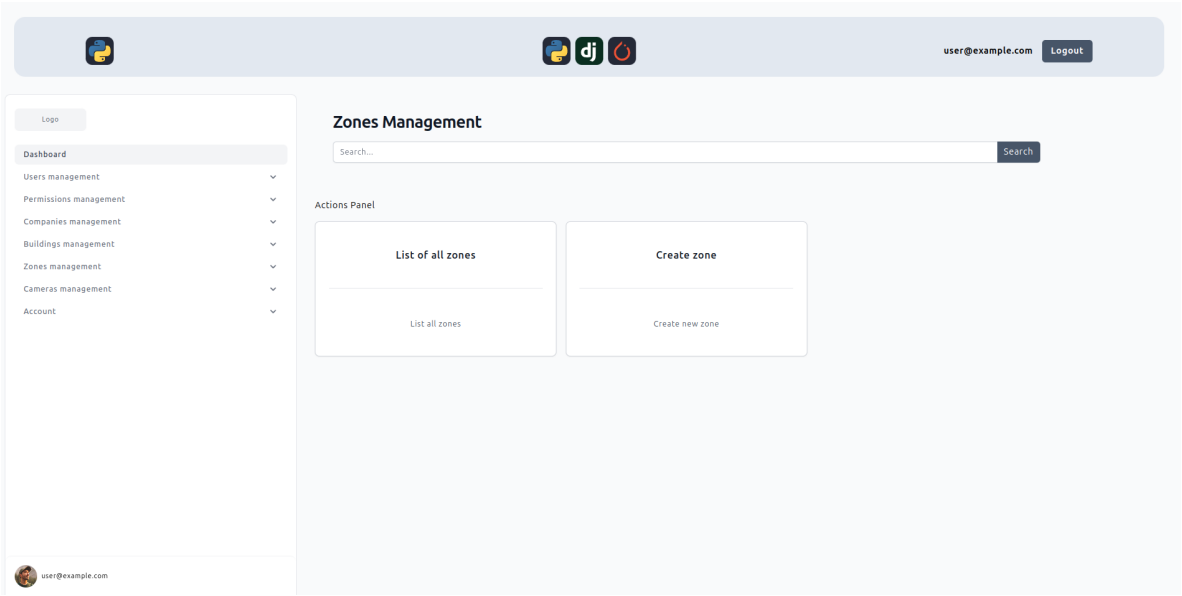

Figure 16: Zones management screen

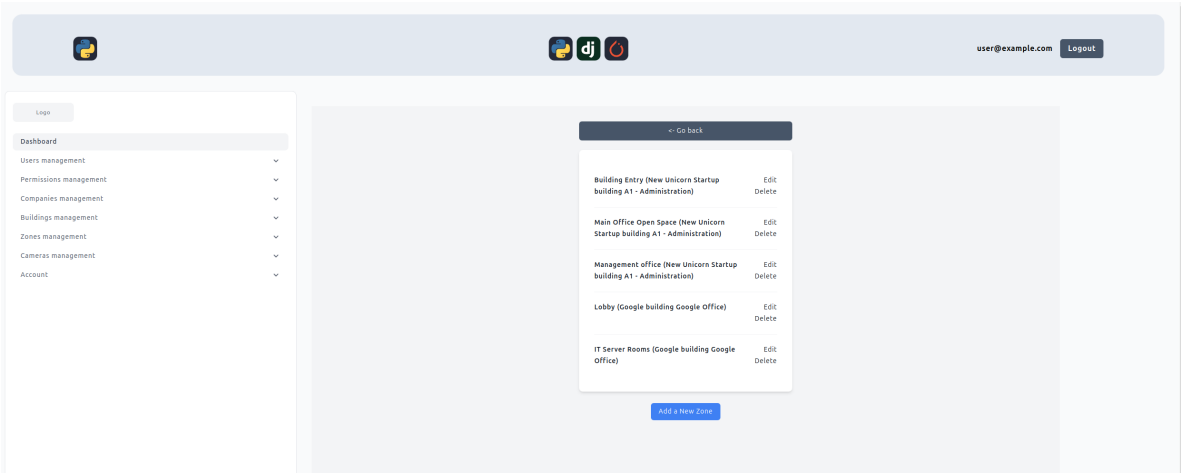

Figure 17: List Zones screen

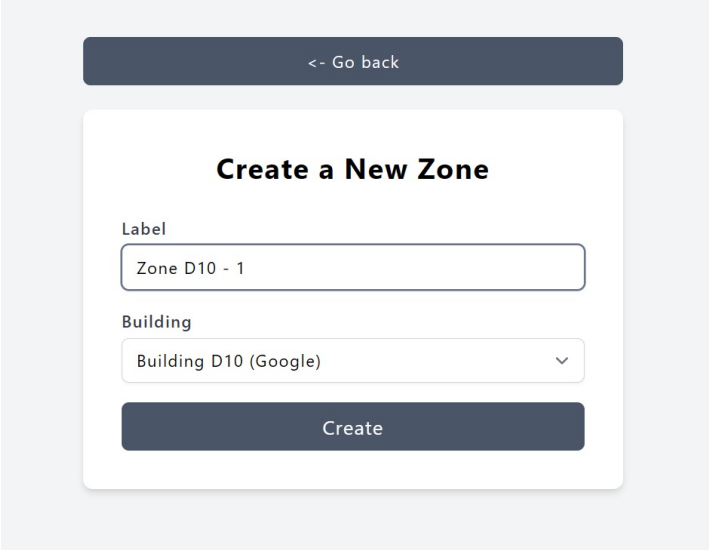

Figure 18: Create zone screen

Another section is zone management. There are views for editing, highlighting, and adding zones. Each user has access to specific zones within the company. With assigned permission, he can easily enter a building/room based on their profile/face. Here, we can also list all permissions.

7.2.3 Permission Management and Access System

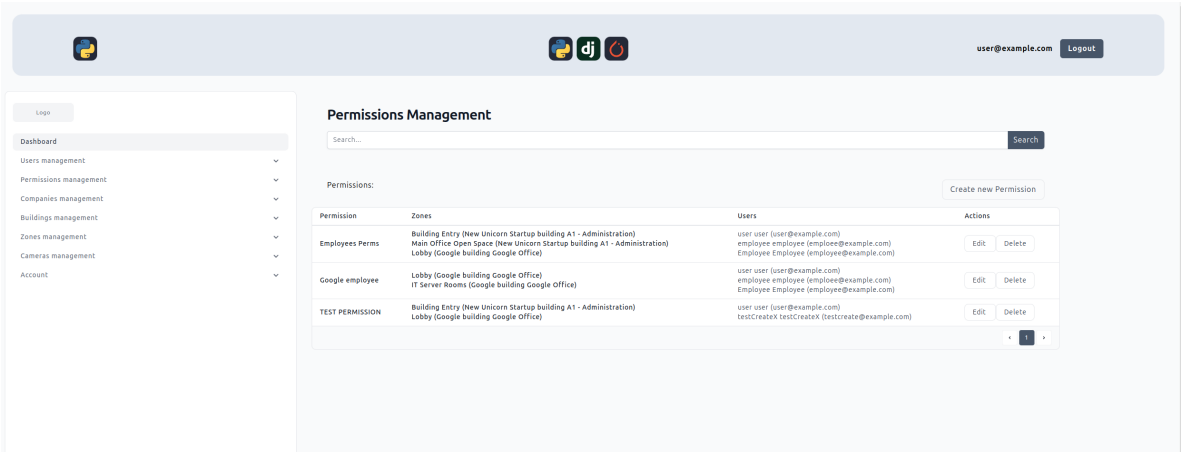

Figure 19: Permission management screen

All permissions are clearly shown. Admin can delete and edit permissions.

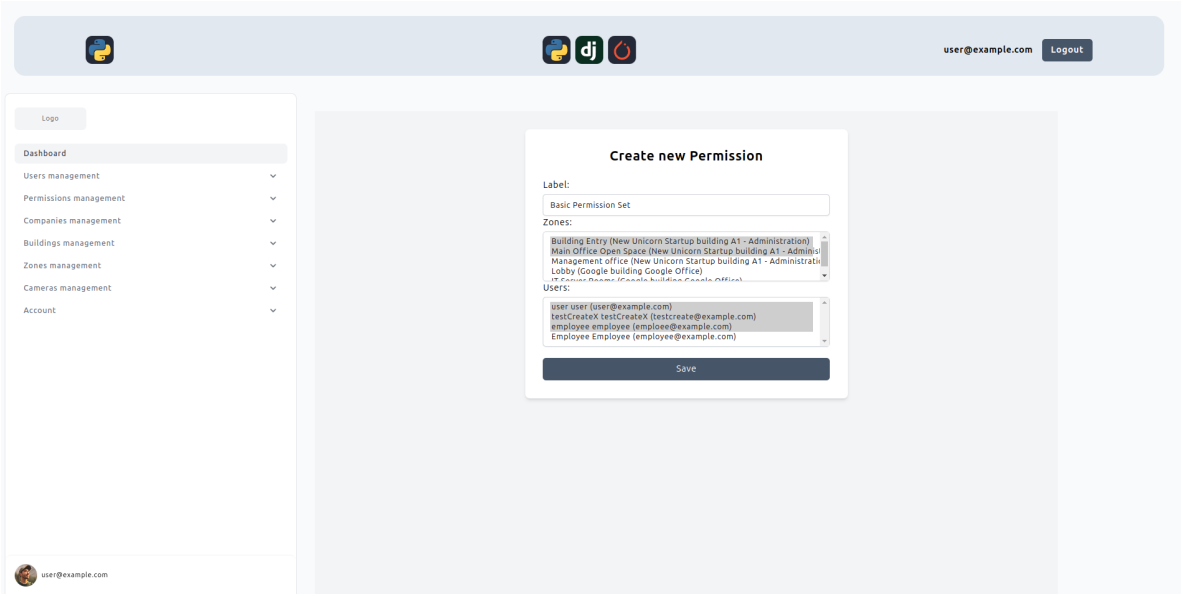

Figure 20: Create new Permission screen

Adding a new permission is pretty easy. There is a multiple choice list, which allows admin to assign access to zone, to person.

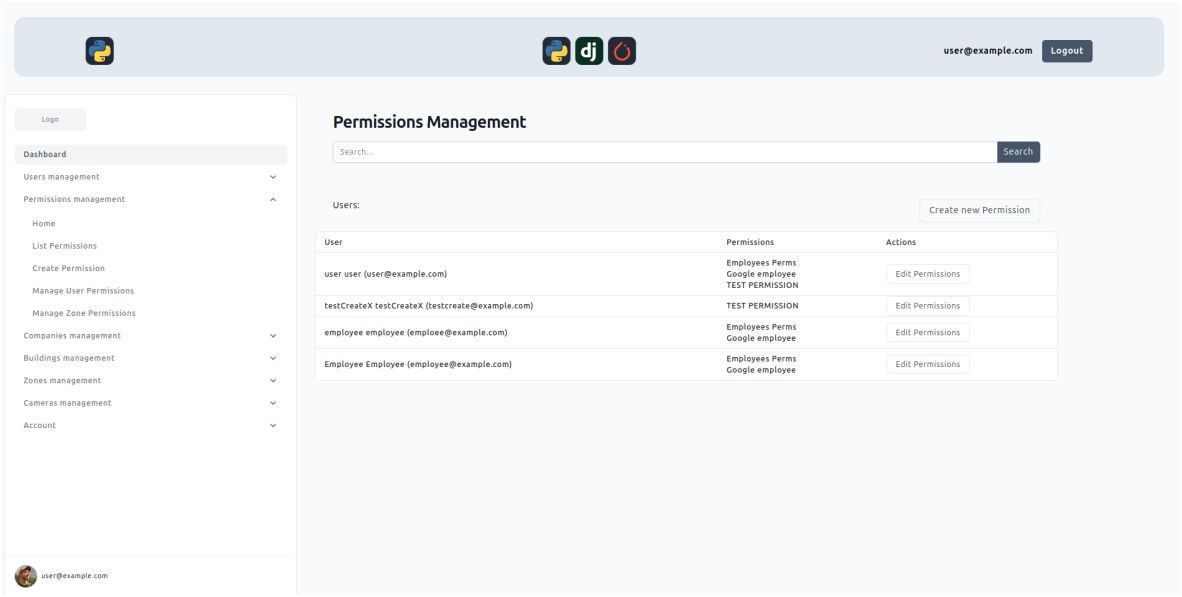

Figure 21: User Permission management screen

Editing and adding permissions looks and works similarly to other sections.

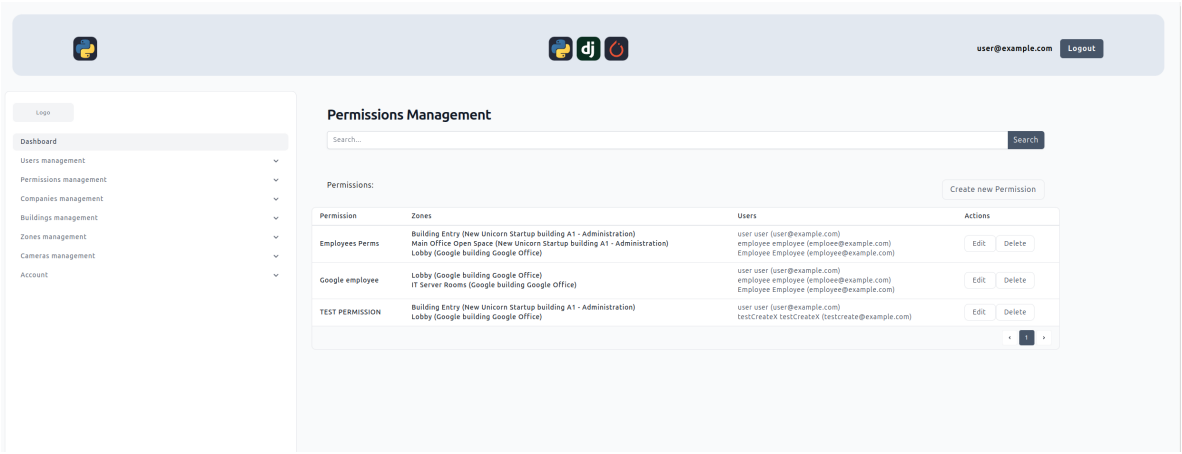

Figure 22: Permission management screen

7.2.4 Cameras Management

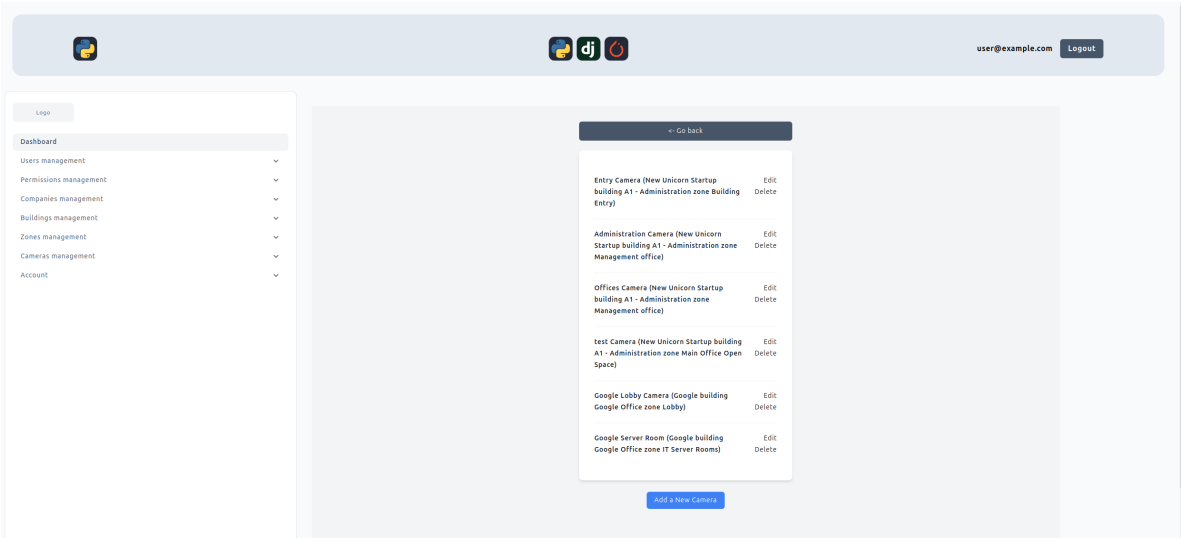

Figure 23: Camera management screen

Cameras are created in the same way as every other element in the app. The key issue is to select the appropriate zone to which the camera is to be assigned. To unlock access to a given zone, the user must have permission, which is granted by the admin.

7.2.5 Camera Feed Screen

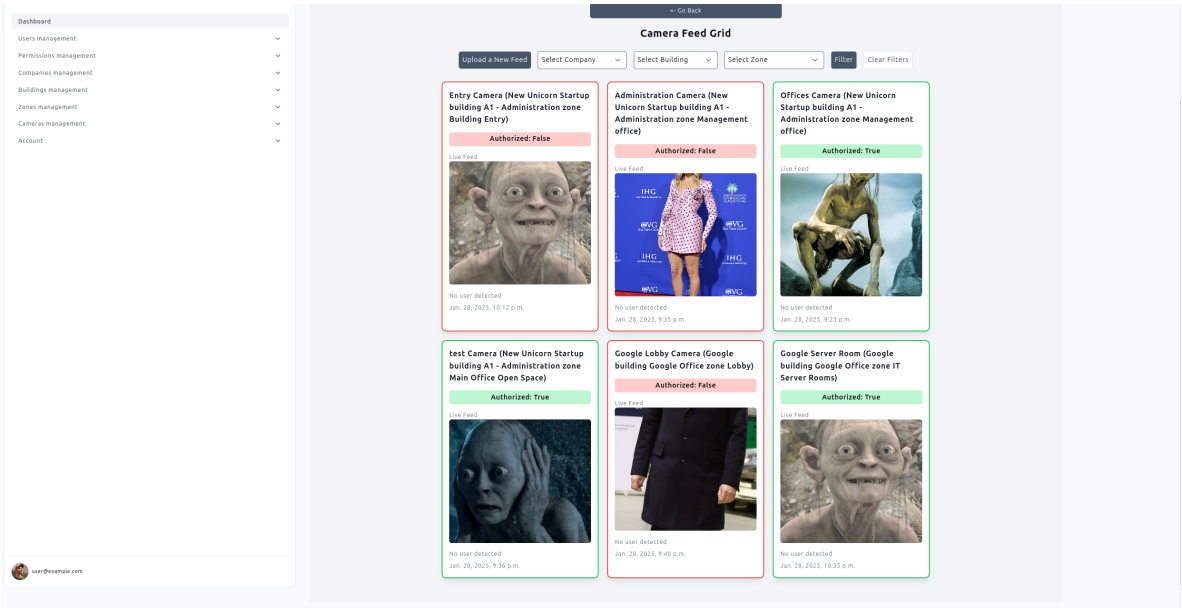

Figure 24: Camera Feed screen

All key information is shown in the camera feed description. When user is authorized, the the feed is green, otherwise user gets refusal. In addition, the system records the date and time of authorization.

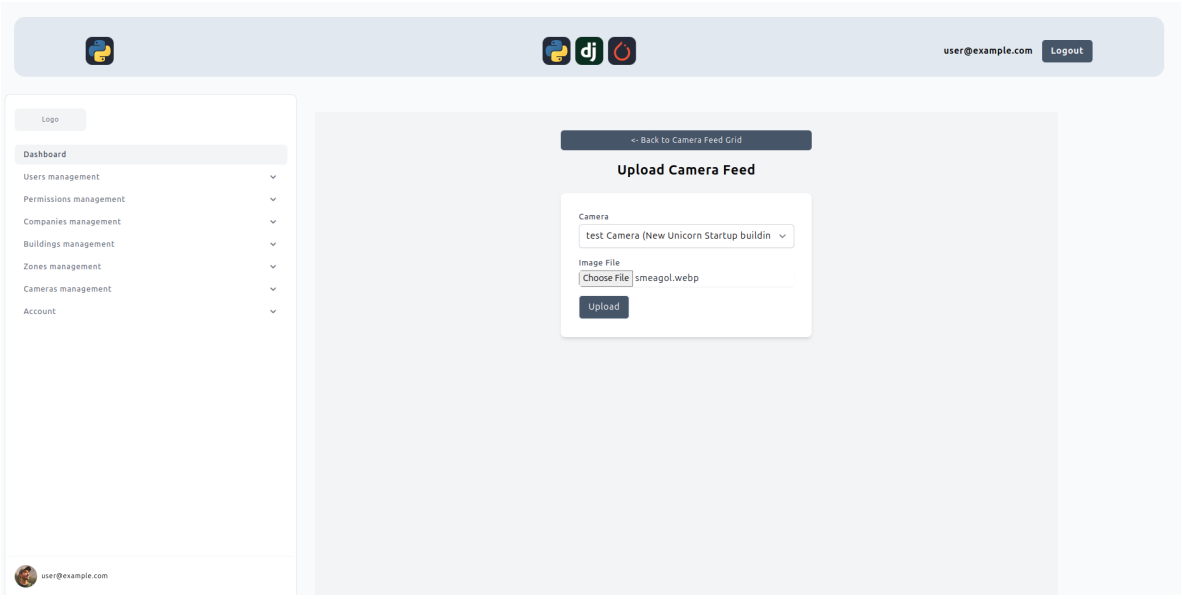

Figure 25: Adding new camera feed

To add new camera feed user has to choose photo from the computer.

7.2.6 Other features

This system is rich in amenities and utilities that make the navigation of the application much easier. Each setting or management option is accessible from multiple places in the application in order to improve UX by reducing the number of clicks that user needs to commit in order to access given entity or feature. One of the features that can be categorized as such is full text search by all important entities and their relations in database.

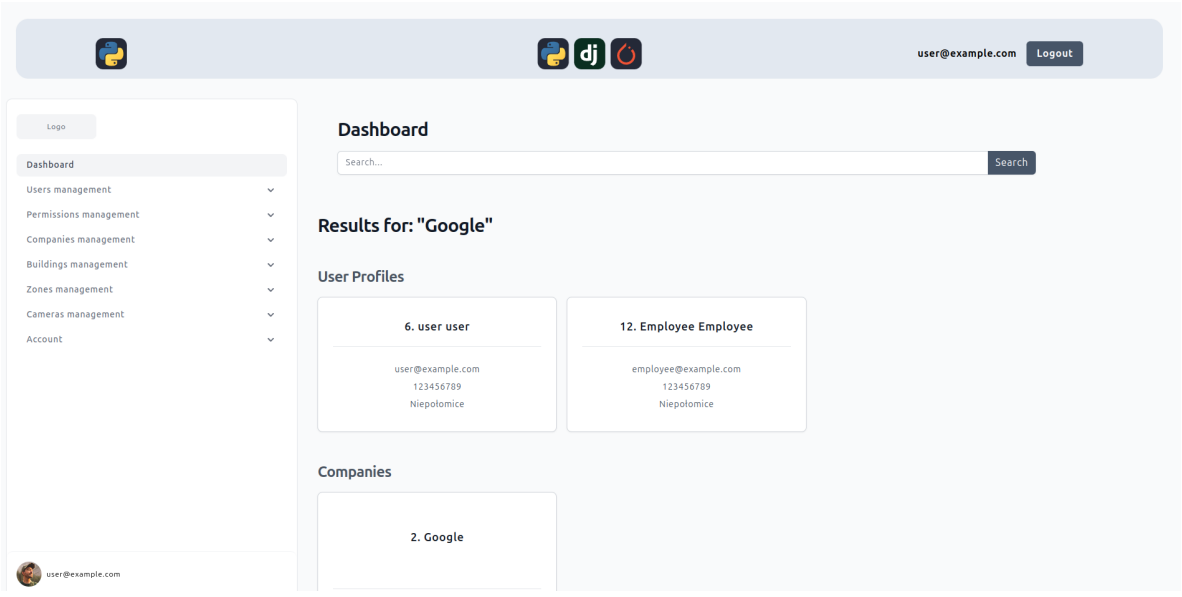

Figure 26: Full Text Search

7.3 Video

There is also a video on YouTube that shows how the system works. Video.

7.4 Access

Access to specific tabs and views is strictly determinated by users' role at ones company. For instance if user is just an employee one cannot access views dedicated for management that are responsible for creating, editing, and deleting given entities. We achieved this by employing decorator design pattern.

guards/custom_permission_required.py

```python
from functools import wraps

from django.http import HttpResponseForbidden
from django.shortcuts import redirect


def custom_permission_required(field_name: str, expected_value, redirect_url=None):
    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):

            if not request.user.is_authenticated:
                return redirect('account_login')

            user_profile = getattr(request.user, 'user_profile', None)
            if not user_profile
              or getattr(user_profile, field_name, None) != expected_value:
                if redirect_url:
                    return redirect(redirect_url)
                return HttpResponseForbidden(
                    "This user has no permission to this view in panel"
                )

            return view_func(request, *args, **kwargs)
        return _wrapped_view
    return decorator
```

Figure 27: Auth Guard Example

guards/user_company_required.py

```python
from functools import wraps

from django.http import HttpResponseForbidden
from django.shortcuts import redirect

from apps.buildings.models import Company


def user_company_required(company_lookup_field='id', param_name='company_id'):
    """
        Dekorator sprawdzający, czy użytkownik ma dostęp do firmy.

        :param company_lookup_field: Pole, według którego będzie wyszukiwane
            (np. 'id', 'name').
        :param param_name: Nazwa parametru przekazywanego do widoku
            (np. 'company_id', 'company_name').
    """
    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):

            company_value = kwargs.get(param_name)

            if not company_value:
                return HttpResponseForbidden("Missing company identifier")

            try:
                company = Company.objects.get(
                  **{company_lookup_field: company_value}
                )
            except Company.DoesNotExist:
                return HttpResponseForbidden("Company does not exist")

            user_profile = getattr(request.user, 'profile', None)
            if not user_profile or not
user_profile.companies.filter(pk=company.pk).exists():
                return HttpResponseForbidden("User has no access for this company")

            request.company = company

            return view_func(request, *args, **kwargs)
        return _wrapped_view
    return decorator
```

Figure 28: Auth Guard Example

guards/user_membership_role.py

```python
from django.http import HttpResponseForbidden
from django.shortcuts import render


def user_membership_role(roles: List[str]):
    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):

            user_profile = getattr(request.user, 'profile', None)
            if not user_profile:
                return render(
                    request,
                    '403.html',
                    context={
                      "message": "No access: You dont have required privileges"
                    },
                    status=403
                )

            memberships = user_profile.memberships.all()
            user_roles = [membership.role for membership in memberships]

            logging.info(f"User roles: {user_roles}")

            if not any(role in roles for role in user_roles):
                logging.warning(
                "No Required Role. Breach attempt: {} ".format(user_profile.email)
                )
                return render(
                    request,
                    '403.html',
                    context={
                    "message": "No access: You dont have required privileges"},
                    status=403
                )

            return view_func(request, *args, **kwargs)

        return _wrapped_view
    return decorator
```

Figure 29: Auth Guard Example

usage.py

```python
@login_required
@user_membership_role(roles=['MANAGEMENT', 'ADMIN'])
def company_create(request: HttpRequest) -> HttpResponse:
    if request.method == 'POST':
        name = request.POST['name']
        if name:
            company = Company(name=name)
            company.save()
        return redirect('company_list')
    else:
        return render(request, 'buildings/company/company_create.html')
```

Figure 30: Auth Guard usage Example

7.4.1 Role of super admin

The difference admin and the super admin is that the super admin has has access to company management. They can also assign individual people to a specific company. A person can belong to multiple companies at the same time, which means that he can have multiple roles. The super admin account is created directly in a database.

8 Face and silhouette feature extraction

8.1 Face feature extraction

8.1.1 What was done in the field

In one of the approaches key points (landmarks) of face such as center of the pupils, mouth corners, extreme points of the face etc. are located on the image. Then distance between chosen key points is calculated (e.g. distance between center of the pupils, corners of the mouth, edges of the face). In the end features as ratio between chosen distances are calculated and combined into a feature vector. Alternatively features could simply be distance or angles between landmarks.
Other approach would be using set of 'eigenfaces', which resemble blurred black and white human faces, but actually are components of human face that value different face patterns in its own unique way. Linear combination of such components can be used to recreate approximation of human face and other way around – human face can be decomposed to a linear combination of such components. Then weights from such linear combination can be used as a feature vector.
Last approach would be using deep learning model that was e.g. trained on big, labeled dataset to identify faces. Such model takes image of the face as input and then performs many matrix calculations in a sequence using weights of consecutive layers and output of previous layers creating more and more complex representation of human face features that are incomprehensible for humans. During training model's weights are modified in such a way that it minimizes value of loss function and at the same time improves how well model performs in a given task. After model is trained we can use output of its (usually) last hidden layer as a feature vector of face from the input.

8.1.2 What we used

We decided to choose last approach as it seems most interesting and according to these papers (1) (access to all links in this section on 2024.12.28), (2) it outperforms both facial landmark and eigenfaces approach.
We decided to use deepface library which provides many state of the art solutions to face recognition. From those we chose GhostFaceNet since according to this website it's currently the best solution.
Deepface library also takes care of face detection and alignment, which are very important steps in face recognition. It again provides many detection options to choose from and they recommend RetinaFace and MtCnn for best performance or their opencv and ssd solutions when speed is more important. Since we are creating security system performance is more important than speed, so we used MtCnn which gave good results significantly faster than RetinaFace.
Since provided face recognition models expect inputs of specified size resizing is required before inference. To avoid deformation caused by resizing, deepface library adds padding pixels after detection and alignment, so we don't have to do that on our own.

8.2 Silhouette feature extraction

8.2.1 What was done in the field

There was definitely less attempts in recognizing persons based on their silhouette which is understandable when recognizing based on face or iris may be used instead, e.g. because they are more characteristic and unique. Because of that there is not too much materials and approaches about feature extraction from silhouette. Therefore we used silhouette recognition only as a supplement to face recognition.
One of the approaches could be extracting person's silhouette from the image, marking points on its edge and then calculating Fourier descriptors of these points using discrete Fourier transform and storing given amount of them as feature vector, but this approach is very susceptible for any changes in person's silhouette like different pose, clothes, even hairstyle, so it would require very rigorous environment and even then probably wouldn't perform too well.
Other approach would be using pose estimation which predicts position of key points like elbows, hands, knees etc. It allows us to then create a feature vector containing e.g. distances between connected pairs of those special points or ratios between such distances. It may be problematic, because those distances change easily when our pose or camera perspective changes, but same applies to face features

and it doesn't seem to make face recognition completely unusable. We also expect people who want to access given zone to collaborate instead of lowering they chances of being recognized by lining up in an unusual way.
The only way we found that person's body was actually used for recognition was their gait (their way of walking), which isn't applicable to our solution where we use only images and not videos.

8.2.2 What we used

Since Fourier descriptors weren't really used in such tasks and intuitively we don't expect them to work well, we used pose estimation in a way mentioned above.
We decided to use mmpose since just like deepface library it provides us with many options to choose from. We have to choose model and dataset that model was trained on and we decided to choose RTMO model trained on crowdpose dataset. This model since according to paper where it was introduced it was both faster and better in pose estimation than other state of the art models. That dataset, because according to config file from mmpose library it gives us access to 14 key points that are: 2 shoulders, 2 elbows, 2 wrists, 2 hips, 2 knees, 2 ankles, top of the head and neck – none of which are connected to the face which we consider separately. After extracting those keypoints we create a feature vector containing distances between nearby joints (2x wrist-elbow, 2x elbow-shoulder, 2x shoulder-neck, 2x shoulder-hip, neck-head, 2x hip-knee, 2x knee-ankle, hip-hip, shoulder-shoulder) divided by mean distance between shoulder and corresponding hip as a way to standardize features no matter how far a person is on a photo. We choose distance between shoulder and hip, because we think that it's the most resistant to slight changes in pose like not facing camera perfectly, walking, holding something or gesticulating.

9 User recognition

We decided to solve user recognition in the following way:

1. extract face and silhouette features in previously described way from new camera feed image,
2. calculate cosine distances between extracted features and each corresponding embedding saved in database,
3. filter those face and silhouette images in database for which distance is lower than chosen threshold (0.75 for face and 0.04 for silhouette),
4. - if there is such face image in database then verify user from camera feed image as user with lowest face embeddings distance from database,
   - otherwise if there is such pose image in database then verify user from camera feed image as user with lowest pose embeddings distance from database,
   - otherwise we recognize user from camera feed image as unknown,
5. then we check if recognized user has permission to zone in which camera is located. If so, we grant him access, otherwise we do not.

In this approach we favor face embeddings over pose embeddings since they are way more characteristic and unique, but also utilize pose embeddings with rigorous threshold to help employee recognition in bad conditions when face recognition fails.

9.1 Testing our method

Initially we tried L2 distance, but sometimes it gave smaller distance for images of two different persons than for different images of same person, which wasn't the case for cosine distance. Also since cosine distance yields way smaller values it was more intuitive to work with and choose thresholds for.
To test our approach we decided to create a small database of 10 persons – 5 men and 5 women – for each we extracted face and pose features from 2 images. Then we created a test sample of 24 images – 2 for each person from database, but one with blurred face to test recognition using only pose features

in combined approach and 4 images of people that are not registered in database. We present our results in tables below.
First row of each table (without last column) contains a label of person identity which is their sex (m for man and w for woman) and digit from 1 to 5. Last column of first row tells us how someone was recognized (if using face embeddings then there is word 'face', if using pose embeddings there is word 'pose') and label of as who they were recognized. Leftmost column (without first row) contains label of person from test set, underscore at the start means that they don't appear in database and „_blur" at the end means that only their pose was used for recognition. Remaining cells contain information about distances between embeddings extracted from test image and embeddings saved in database divided by chosen threshold (0.75 for face embeddings and 0.04 for pose embeddings), each ratio is separated with semicolon. If there are 4 values in a cell first two are face embedding distance ratios. With such ratios, value above 1 means that distance was greater than threshold and below 1 means that it was smaller.

9.1.1 Using only face for recognition

Table 3: Results for only face recognition.

.	m1	m2	m3	m4	m5	w1	w2	w3	w4	w5	recognized
_m6	1.39;1.47;	1.41;1.46;	1.35;1.40;	1.18;1.02;	1.35;1.41;	1.35;1.42;	1.46;1.50;	1.49;1.47;	1.35;1.15;	1.53;1.43;	not in database
_m7	1.52;1.61;	1.28;1.39;	1.02;1.13;	1.53;1.61;	1.15;1.24;	1.58;1.42;	1.30;1.32;	1.28;1.43;	1.30;1.21;	1.35;1.31;	not in database
_w6	1.24;1.37;	1.31;1.43;	1.27;1.32;	1.33;1.39;	1.28;1.31;	1.08;1.05;	1.07;1.11;	0.94;1.17;	1.15;1.02;	1.16;1.16;	face: w3
_w7	1.35;1.31;	1.43;1.47;	1.32;1.42;	1.30;1.23;	1.58;1.56;	1.40;1.50;	1.32;1.17;	1.53;1.26;	1.47;1.53;	1.16;1.32;	not in database
m1	1.07;1.00;	1.21;1.44;	1.26;1.34;	1.32;1.26;	1.48;1.41;	1.39;1.32;	1.54;1.36;	1.53;1.48;	1.40;1.32;	1.31;1.47;	not in database
m2	1.42;1.30;	0.76;1.14;	1.11;1.14;	1.38;1.33;	1.17;1.19;	1.34;1.44;	1.53;1.31;	1.44;1.42;	1.37;1.41;	1.34;1.49;	face: m2
m3	1.34;1.18;	1.23;1.38;	0.58;0.69;	1.23;1.36;	1.22;1.26;	1.50;1.46;	1.23;1.32;	1.36;1.32;	1.35;1.28;	1.17;1.25;	face: m3
m4	1.28;1.21;	1.28;1.25;	1.42;1.45;	0.34;0.42;	1.25;1.32;	1.25;1.28;	1.36;1.24;	1.47;1.36;	1.39;1.17;	1.38;1.20;	face: m4
m5	1.40;1.48;	1.40;1.40;	1.04;1.11;	1.33;1.21;	0.41;0.57;	1.38;1.33;	1.25;1.21;	1.28;1.21;	1.38;1.23;	1.10;1.34;	face: m5
w1	1.34;1.53;	1.44;1.36;	1.34;1.46;	1.49;1.40;	1.11;1.24;	1.00;0.87;	1.18;1.07;	1.27;1.30;	1.21;1.51;	1.18;1.04;	face: w1
w2	1.22;1.27;	1.40;1.55;	1.23;1.33;	1.43;1.36;	1.12;1.23;	1.52;1.30;	0.76;0.84;	1.32;1.46;	1.36;1.36;	1.17;1.36;	face: w2
w3	1.25;1.32;	1.35;1.40;	1.37;1.35;	1.39;1.38;	1.42;1.22;	1.43;1.35;	1.20;1.05;	0.97;1.26;	1.08;1.32;	1.30;1.00;	face: w3
w4	1.27;1.47;	1.35;1.50;	1.25;1.24;	1.32;1.23;	1.40;1.31;	1.34;1.34;	1.40;1.37;	1.30;1.30;	0.78;0.37;	1.36;1.35;	face: w4
w5	1.49;1.44;	1.44;1.36;	1.32;1.38;	1.33;1.32;	1.30;1.27;	1.24;1.26;	1.32;1.28;	1.33;1.29;	1.52;1.48;	0.82;1.25;	face: w5

When we used only face for recognition and threshold equal to 0.75, we got quite good results with only one person from database barely not getting recognized (their best ratio was slightly above 1.0) and one person outside of database getting recognized as someone from database. All other people got recognized as themselves.

9.1.2 Using only pose for recognition

Table 4: Results for only pose recognition.

.	m1	m2	m3	m4	m5	w1	w2	w3	w4	w5	recognized
_m6	0.08;0.22;	0.02;0.11;	0.05;0.08;	0.30;0.19;	0.04;0.06;	0.22;0.16;	0.37;0.10;	0.32;0.21;	0.21;0.08;	0.37;0.28;	pose: m2
_m7	0.07;0.15;	0.13;0.28;	0.18;0.12;	0.13;0.07;	0.19;0.24;	0.07;0.06;	0.13;0.08;	0.15;0.10;	0.07;0.07;	0.13;0.09;	pose: w1
_w6	0.07;0.16;	0.11;0.22;	0.16;0.10;	0.19;0.13;	0.17;0.19;	0.12;0.08;	0.22;0.05;	0.24;0.15;	0.09;0.02;	0.18;0.13;	pose: w4
_w7	0.26;0.25;	0.30;0.43;	0.42;0.33;	0.09;0.07;	0.40;0.43;	0.08;0.08;	0.06;0.17;	0.09;0.03;	0.21;0.26;	0.23;0.19;	pose: w3
m1	0.07;0.14;	0.05;0.20;	0.10;0.08;	0.29;0.18;	0.08;0.11;	0.19;0.16;	0.29;0.12;	0.28;0.19;	0.10;0.12;	0.25;0.17;	pose: m2
m2	0.06;0.21;	0.11;0.28;	0.13;0.11;	0.21;0.13;	0.12;0.19;	0.14;0.17;	0.27;0.17;	0.24;0.18;	0.09;0.12;	0.23;0.16;	pose: m1
m3	0.07;0.21;	0.04;0.17;	0.03;0.06;	0.46;0.31;	0.02;0.04;	0.32;0.28;	0.52;0.23;	0.42;0.34;	0.20;0.16;	0.38;0.29;	pose: m5
m4	0.10;0.28;	0.19;0.39;	0.21;0.17;	0.08;0.05;	0.22;0.29;	0.07;0.10;	0.15;0.15;	0.16;0.12;	0.15;0.12;	0.21;0.17;	pose: m4
m5	0.08;0.25;	0.06;0.20;	0.12;0.14;	0.15;0.09;	0.11;0.16;	0.11;0.10;	0.21;0.06;	0.20;0.13;	0.15;0.09;	0.29;0.22;	pose: w2
w1	0.12;0.11;	0.20;0.23;	0.26;0.18;	0.25;0.19;	0.24;0.30;	0.08;0.12;	0.20;0.19;	0.08;0.09;	0.10;0.18;	0.13;0.10;	pose: w1
w2	0.17;0.27;	0.24;0.37;	0.30;0.25;	0.12;0.11;	0.29;0.37;	0.05;0.12;	0.14;0.21;	0.05;0.07;	0.17;0.22;	0.21;0.18;	pose: w1
w3	0.09;0.25;	0.26;0.37;	0.24;0.15;	0.24;0.20;	0.26;0.32;	0.16;0.19;	0.34;0.28;	0.27;0.22;	0.17;0.13;	0.21;0.15;	pose: m1
w4	0.08;0.22;	0.09;0.15;	0.10;0.08;	0.34;0.24;	0.12;0.13;	0.25;0.17;	0.42;0.11;	0.39;0.27;	0.20;0.02;	0.32;0.25;	pose: w4
w5	0.29;0.29;	0.51;0.67;	0.58;0.38;	0.15;0.18;	0.61;0.68;	0.12;0.16;	0.11;0.30;	0.18;0.17;	0.17;0.25;	0.08;0.10;	pose: w5

When we used just pose features results got catastrophically worse. Every single test subject from outside of database got recognized as someone which means that 4 unknown people could get unauthorized access. Also 6 samples of people that are in database got recognized as someone else from it. It leaves us with just 5 correct recognitions. We can easily conclude that recognition using only pose features is neither safe nor accurate. Also it's worth noting that distance ratios are below 1 in every single cell. It's caused by pose features not being neither characteristic nor unique.

9.1.3 Face and pose recognition

Since combined approach really is just a two stage recognition where we first try to recognize using face and when we don't find good enough face match then we try to recognize using pose, we end up with always recognizing person from given image as someone from database. It happens, because pose features are not very distinctive between different individuals. So in this approach we have some alright stuff since we use face embeddings first and when we find good enough match (below threshold) they work quite well just like it was shown in this section, but we also get a lot of bad stuff from recognition using only pose, which introduces a lot of error when it comes to people from outside of database and those from database but without face available.
Most of the good results in table below come from face recognition and most of the bad come from pose recognition. In face only recognition in this section we got 75% rejection rate for people outside of database and 90% proper recognition for people from database, in combined approach we have 0% rejection rate for former and 65% proper recognition for the latter group.

Table 5: Results for combined approach.

.	m1	m2	m3	m4	m5	w1	w2	w3	w4	w5	recognized
_m6	1.39;1.47;0.08;0.22;	1.41;1.46;0.02;0.11;	1.35;1.40;0.05;0.08;	1.18;1.02;0.30;0.19;	1.35;1.41;0.04;0.06;	1.35;1.42;0.22;0.16;	1.46;1.50;0.37;0.10;	1.49;1.47;0.32;0.21;	1.35;1.15;0.21;0.08;	1.53;1.43;0.37;0.28;	pose: m2
_m7	1.52;1.61;0.07;0.15;	1.28;1.39;0.13;0.28;	1.02;1.13;0.18;0.12;	1.53;1.61;0.13;0.07;	1.15;1.24;0.19;0.24;	1.58;1.42;0.07;0.06;	1.30;1.32;0.13;0.08;	1.28;1.43;0.15;0.10;	1.30;1.21;0.07;0.07;	1.35;1.31;0.13;0.09;	pose: w1
_w6	1.24;1.37;0.07;0.16;	1.31;1.43;0.11;0.22;	1.27;1.32;0.16;0.10;	1.33;1.39;0.19;0.13;	1.28;1.31;0.17;0.19;	1.08;1.05;0.12;0.08;	1.07;1.11;0.22;0.05;	0.94;1.17;0.24;0.15;	1.15;1.02;0.09;0.02;	1.16;1.16;0.18;0.13;	face: w3
_w7	1.35;1.31;0.26;0.25;	1.43;1.47;0.30;0.43;	1.32;1.42;0.42;0.33;	1.30;1.23;0.09;0.07;	1.58;1.56;0.40;0.43;	1.40;1.50;0.08;0.08;	1.32;1.17;0.06;0.17;	1.53;1.26;0.09;0.03;	1.47;1.53;0.21;0.26;	1.16;1.32;0.23;0.19;	pose: w3
m1	1.07;1.00;0.07;0.14;	1.21;1.44;0.05;0.20;	1.26;1.34;0.10;0.08;	1.32;1.26;0.29;0.18;	1.48;1.41;0.08;0.11;	1.39;1.32;0.19;0.16;	1.54;1.36;0.29;0.12;	1.53;1.48;0.28;0.19;	1.40;1.32;0.10;0.12;	1.31;1.47;0.25;0.17;	pose: m2
m1_blur	2.00;2.00;0.07;0.15;	2.00;2.00;0.07;0.24;	2.00;2.00;0.12;0.10;	2.00;2.00;0.26;0.16;	2.00;2.00;0.12;0.15;	2.00;2.00;0.17;0.15;	2.00;2.00;0.24;0.12;	2.00;2.00;0.26;0.18;	2.00;2.00;0.08;0.12;	2.00;2.00;0.22;0.15;	pose: m1
m2	1.42;1.30;0.06;0.21;	0.76;1.14;0.11;0.28;	1.11;1.14;0.13;0.11;	1.38;1.33;0.21;0.13;	1.17;1.19;0.12;0.19;	1.34;1.44;0.14;0.17;	1.53;1.31;0.27;0.17;	1.44;1.42;0.24;0.18;	1.37;1.41;0.09;0.12;	1.34;1.49;0.23;0.16;	face: m2
m2_blur	2.00;2.00;0.06;0.22;	2.00;2.00;0.12;0.29;	2.00;2.00;0.13;0.10;	2.00;2.00;0.22;0.14;	2.00;2.00;0.12;0.20;	2.00;2.00;0.15;0.19;	2.00;2.00;0.29;0.19;	2.00;2.00;0.25;0.20;	2.00;2.00;0.10;0.13;	2.00;2.00;0.24;0.16;	pose: m1
m3	1.34;1.18;0.07;0.21;	1.23;1.38;0.04;0.17;	0.58;0.69;0.03;0.06;	1.23;1.36;0.46;0.31;	1.22;1.26;0.02;0.04;	1.50;1.46;0.32;0.28;	1.23;1.32;0.52;0.23;	1.36;1.32;0.42;0.34;	1.35;1.28;0.20;0.16;	1.17;1.25;0.38;0.29;	face: m3
m3_blur	2.00;2.00;0.06;0.20;	2.00;2.00;0.03;0.17;	2.00;2.00;0.03;0.06;	2.00;2.00;0.45;0.30;	2.00;2.00;0.02;0.04;	2.00;2.00;0.30;0.27;	2.00;2.00;0.50;0.23;	2.00;2.00;0.41;0.32;	2.00;2.00;0.19;0.16;	2.00;2.00;0.37;0.27;	pose: m5
m4	1.28;1.21;0.10;0.28;	1.28;1.25;0.19;0.39;	1.42;1.45;0.21;0.17;	0.34;0.42;0.08;0.05;	1.25;1.32;0.22;0.29;	1.25;1.28;0.07;0.10;	1.36;1.24;0.15;0.15;	1.47;1.36;0.16;0.12;	1.39;1.17;0.15;0.12;	1.38;1.20;0.21;0.17;	face: m4
m4_blur	2.00;2.00;0.11;0.29;	2.00;2.00;0.20;0.41;	2.00;2.00;0.23;0.18;	2.00;2.00;0.07;0.05;	2.00;2.00;0.24;0.31;	2.00;2.00;0.07;0.10;	2.00;2.00;0.15;0.16;	2.00;2.00;0.16;0.12;	2.00;2.00;0.16;0.13;	2.00;2.00;0.21;0.17;	pose: m4
m5	1.40;1.48;0.08;0.25;	1.40;1.40;0.06;0.20;	1.04;1.11;0.12;0.14;	1.33;1.21;0.15;0.09;	0.41;0.57;0.11;0.16;	1.38;1.33;0.11;0.10;	1.25;1.21;0.21;0.06;	1.28;1.21;0.20;0.13;	1.38;1.23;0.15;0.09;	1.10;1.34;0.29;0.22;	face: m5
m5_blur	2.00;2.00;0.09;0.26;	2.00;2.00;0.10;0.24;	2.00;2.00;0.16;0.16;	2.00;2.00;0.11;0.07;	2.00;2.00;0.15;0.21;	2.00;2.00;0.09;0.08;	2.00;2.00;0.17;0.06;	2.00;2.00;0.18;0.11;	2.00;2.00;0.15;0.08;	2.00;2.00;0.27;0.21;	pose: w2
w1	1.34;1.53;0.12;0.11;	1.44;1.36;0.20;0.23;	1.34;1.46;0.26;0.18;	1.49;1.40;0.25;0.19;	1.11;1.24;0.24;0.30;	1.00;0.87;0.08;0.12;	1.18;1.07;0.20;0.19;	1.27;1.30;0.08;0.09;	1.21;1.51;0.10;0.18;	1.18;1.04;0.13;0.10;	face: w1
w1_blur	2.00;2.00;0.14;0.13;	2.00;2.00;0.22;0.27;	2.00;2.00;0.29;0.20;	2.00;2.00;0.24;0.18;	2.00;2.00;0.28;0.33;	2.00;2.00;0.07;0.12;	2.00;2.00;0.18;0.19;	2.00;2.00;0.07;0.08;	2.00;2.00;0.10;0.19;	2.00;2.00;0.11;0.10;	pose: w3
w2	1.22;1.27;0.17;0.27;	1.40;1.55;0.24;0.37;	1.23;1.33;0.30;0.25;	1.43;1.36;0.12;0.11;	1.12;1.23;0.29;0.37;	1.52;1.30;0.05;0.12;	0.76;0.84;0.14;0.21;	1.32;1.46;0.05;0.07;	1.36;1.36;0.17;0.22;	1.17;1.36;0.21;0.18;	face: w2
w2_blur	2.00;2.00;0.18;0.27;	2.00;2.00;0.24;0.37;	2.00;2.00;0.30;0.26;	2.00;2.00;0.12;0.10;	2.00;2.00;0.29;0.37;	2.00;2.00;0.05;0.12;	2.00;2.00;0.14;0.21;	2.00;2.00;0.05;0.07;	2.00;2.00;0.18;0.23;	2.00;2.00;0.21;0.18;	pose: w1
w3	1.25;1.32;0.09;0.25;	1.35;1.40;0.26;0.37;	1.37;1.35;0.24;0.15;	1.39;1.38;0.24;0.20;	1.42;1.22;0.26;0.32;	1.43;1.35;0.16;0.19;	1.20;1.05;0.34;0.28;	0.97;1.26;0.27;0.22;	1.08;1.32;0.17;0.13;	1.30;1.00;0.21;0.15;	face: w3
w3_blur	2.00;2.00;0.09;0.26;	2.00;2.00;0.26;0.37;	2.00;2.00;0.24;0.15;	2.00;2.00;0.24;0.20;	2.00;2.00;0.25;0.33;	2.00;2.00;0.16;0.19;	2.00;2.00;0.34;0.28;	2.00;2.00;0.27;0.22;	2.00;2.00;0.17;0.13;	2.00;2.00;0.21;0.16;	pose: m1
w4	1.27;1.47;0.08;0.22;	1.35;1.50;0.09;0.15;	1.25;1.24;0.10;0.08;	1.32;1.23;0.34;0.24;	1.40;1.31;0.12;0.13;	1.34;1.34;0.25;0.17;	1.40;1.37;0.42;0.11;	1.30;1.30;0.39;0.27;	0.78;0.37;0.20;0.02;	1.36;1.35;0.32;0.25;	face: w4
w4_blur	2.00;2.00;0.09;0.22;	2.00;2.00;0.09;0.15;	2.00;2.00;0.10;0.08;	2.00;2.00;0.33;0.24;	2.00;2.00;0.12;0.13;	2.00;2.00;0.24;0.17;	2.00;2.00;0.41;0.11;	2.00;2.00;0.38;0.26;	2.00;2.00;0.20;0.02;	2.00;2.00;0.32;0.25;	pose: w4
w5	1.49;1.44;0.29;0.29;	1.44;1.36;0.51;0.67;	1.32;1.38;0.58;0.38;	1.33;1.32;0.15;0.18;	1.30;1.27;0.61;0.68;	1.24;1.26;0.12;0.16;	1.32;1.28;0.11;0.30;	1.33;1.29;0.18;0.17;	1.52;1.48;0.17;0.25;	0.82;1.25;0.08;0.10;	face: w5
w5_blur	2.00;2.00;0.31;0.32;	2.00;2.00;0.53;0.71;	2.00;2.00;0.60;0.40;	2.00;2.00;0.14;0.18;	2.00;2.00;0.63;0.70;	2.00;2.00;0.13;0.17;	2.00;2.00;0.11;0.32;	2.00;2.00;0.19;0.18;	2.00;2.00;0.18;0.27;	2.00;2.00;0.10;0.11;	pose: w5

9.1.4 Conclusions

When it comes to multimodal biometric recognition, it looks like using person's pose in proposed way causes more harm than good. Another approach that we have considered – calculating weighted average of distances between embeddings of face and pose features – could turn out to be less harmful, but since used pose features are neither unique nor characteristic they would probably always negatively impact system's performance. Modifying how exactly pose features are used doesn't seem to have any accuracy advantage over changing biometric feature to something more distinctive between individuals like iris, voice or fingerprints. The only advantage that we can think of is that it doesn't require any additional equipment than camera that is needed for face recognition anyway and that it doesn't require any actions from the user – they just need to properly align for the photo.

10 Image Embedding Comparison Mechanism

The image recognition system relies on **embedding vectors** to perform identity verification and object matching. Each image is transformed into a high-dimensional vector representation, which allows efficient similarity comparisons between images. This mechanism is essential for **facial recognition** and **silhouette-based identification** in the enterprise security system.

10.1 Mathematical Foundations

The system employs vector similarity measures to determine the closeness of embeddings. Two primary distance metrics are used:

10.1.1 Cosine Distance

Cosine distance measures the angular difference between two vectors. It is defined as:

$$d_{\cos}(A, B) = 1 - \frac{A \cdot B}{\|A\|\|B\|} \tag{1}$$

where:

- A, B are embedding vectors.
- $A \cdot B$ represents the dot product of the vectors.
- $\|A\|$ and $\|B\|$ denote the Euclidean norms (magnitudes) of the vectors.

A cosine distance close to 0 indicates high similarity.

10.1.2 Euclidean (L2) Distance

The Euclidean distance, or L_2-norm, measures the straight-line distance between two vectors in the embedding space:

$$d_{L2}(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2} \tag{2}$$

where n is the dimension of the embedding vectors.

10.2 Database Query Execution for Similarity Search

The system uses **PostgreSQL with PGVector** to perform efficient similarity searches based on stored embedding vectors.

10.2.1 Filtering by Cosine Distance

To retrieve images whose embedding vectors are within a given cosine distance threshold, the following SQL query is executed:

```
SELECT id, user_id, image_type, file_path,
       embedding <=> ARRAY[...]
FROM user_image
WHERE image_type = 'face'
AND embedding <=> ARRAY[...] < 0.75
ORDER BY embedding <=> ARRAY[...]
LIMIT 10;
```

```
SELECT
    id,
    embedding,
    embedding <=>
        '[-0.47044072,-0.11218968, ... ,-0.86074734,2.0002904]' AS cos_distance
FROM
    public.authentication_userimage
WHERE image_type = 'face' AND embedding <=>
    '[-0.47044072,-0.11218968, ... ,-0.86074734,2.0002904]' < 1
ORDER BY
    cos_distance ASC
LIMIT 10;
```

Here, the operator `<=>` computes the cosine distance.

10.2.2 Retrieving the Closest Match

To find the single closest matching image:

```
SELECT id, user_id, image_type, file_path,
       embedding <=> ARRAY[...]
FROM user_image
ORDER BY embedding <=> ARRAY[...]
LIMIT 1;
```

10.3 Efficient Indexing of Embedding Vectors

Given that embedding vectors are high-dimensional, performing brute-force searches over large datasets is computationally expensive. To optimize similarity search performance, the system leverages specialized indexing methods:

10.3.1 HNSW (Hierarchical Navigable Small World)

HNSW is an **approximate nearest neighbor** (ANN) search algorithm that builds a multi-layer graph structure for fast lookups. It is particularly well-suited for high-dimensional data.
In PostgreSQL with PGVector, an HNSW index can be created as follows:

```
CREATE INDEX ON user_image
USING hnsw (embedding vector_cosine_ops);
```

This significantly accelerates similarity searches compared to sequential scans.

10.3.2 IVFFlat (Inverted File Index with Flat Search)

IVFFlat partitions the data into clusters and performs searches only within the most relevant clusters. While not as fast as HNSW, it provides a balance between speed and accuracy.
To create an IVFFlat index:

```
CREATE INDEX ON user_image
USING ivfflat (embedding vector_l2_ops)
WITH (lists = 100);
```

The parameter `lists = 100` defines the number of clusters, which should be tuned based on dataset size.

10.4 Implementation in Django

The following Django ORM query is used to retrieve images based on their similarity to a given embedding vector:

```
queryset.annotate(distance=CosineDistance(F('embedding'), embedding))
       .filter(distance__lt=threshold)
       .order_by('distance')
```

For retrieving the top k closest matches:

```
queryset.annotate(distance=L2Distance(F('embedding'), embedding))
       .order_by('distance')[:k]
```

10.5 Security and Performance Considerations

To ensure optimal performance and security:

- Embeddings are indexed using **Approximate Nearest Neighbors (ANN)** with HNSW.
- Distance calculations are executed efficiently within the database to minimize application-layer overhead.
- User images are stored securely, and access is restricted based on roles and permissions.

10.6 Conclusion

The purpose of the project was to create a system for managing access to buildings and zones in the company. It was possible to implement comprehensive user authorization mechanisms, including logging in, managing permissions, and handling identity verification cameras.
The system supports different levels of user authorization, and access to buildings is based on cameras with image analysis. In addition, an intuitive interface has been implemented to enable quick management of users and access zones.
The image recognition system integrates **deep learning-based embedding generation** with **efficient database querying** to enable real-time identification. The use of optimized distance metrics and indexing ensures high accuracy while maintaining scalability.
Ultimately, the feed camera should be connected to real cameras. Additionally, system could handle emergency situations (fire etc.) by showing some info on employees monitors.