Dokumentacja

None

None

None

Table of contents

1. 🏫 System Zarządzania Przedszkolem (MarchewkaDjango) 3
   1.1 🌟 Główne Funkcjonalności 3
   1.2 🏗️ Architektura Systemu 3
   1.3 📊 Schemat Powiązań Bazodanowych 3
   1.4
   4
   1.5 🚀 Jak zacząć? 4
2. 🛠️ Wdrożenie i Start 5
   2.1 Szybki Start (Dev) 5
   2.2 Główny Przepływ (Flow) 6
3. 🧩 Moduły Aplikacji 7
   3.1 🔐 Accounts (Konta) 7
   3.2 🎓 Director (Dyrektor) 10
   3.3 👩‍🏫 Teacher (Kadry) 13
   3.4 👨‍👩‍👧 Parent (Rodzice) 0
   3.5 👶 Children (Dzieci) 0
   3.6 👥 Groups (Grupy) 0
   3.7 📅 Calendar (Kalendarz) 0
   3.8 🥗 Meals (Żywienie) 0
   3.9 💳 Payments (Finanse) 0
   3.10 📝 Blog (Wydarzenia) 0

```
erDiagram
    USER ||--|| DIRECTOR : "posiada profil"
    USER ||--|| TEACHER : "posiada profil"
    USER ||--|| PARENT : "posiada profil"

    DIRECTOR ||--o{ GROUP : "zarządza"
    DIRECTOR ||--o{ KID : "nadzoruje"
    DIRECTOR ||--o{ MEAL : "definiuje"
    DIRECTOR ||--o{ PAYMENT_PLAN : "ustala"

    GROUP ||--o{ KID : "zrzesza"
    GROUP ||--o{ TEACHER : "ma przypisanych"

    PARENT ||--o{ KID : "opiekuje się"

    KID ||--o{ PRESENCE : "posiada wpisy"
    KID }o--|| MEAL : "korzysta z"
    KID }o--|| PAYMENT_PLAN : "jest rozliczane"

    TEACHER ||--o{ POST : "publikuje"
    DIRECTOR ||--o{ POST : "publikuje"
    POST }o--|| GROUP : "jest adresowany do"
```

1. 🏫 System Zarządzania Przedszkolem (MarchewkaDjango)

Witaj w oficjalnej dokumentacji projektu **KindergartenDjangoDev**. Jest to zintegrowana platforma klasy ERP stworzona w Django, służąca do kompleksowego zarządzania placówką przedszkolną.

1.1 🌟 Główne Funkcjonalności

System został podzielony na dedykowane moduły (Aplikacje Django), które realizują konkretne cele biznesowe:

1.1.1 👤 Profile i Uprawnienia

- **Accounts**: Zaawansowane zarządzanie użytkownikami i system logowania.
- **Director, Teacher, Parent**: Trzy niezależne panele sterowania dostosowane do ról w przedszkolu.

1.1.2 🧒 Zarządzanie Podopiecznymi

- **Children**: Centralny rejestr dzieci, informacje o zdrowiu i opiekunach.
- **Groups**: Organizacja dzieci w grupy wiekowe i przypisywanie wychowawców.

1.1.3 🍎 Życie Przedszkolne

- **Meals**: Planowanie jadłospisów, monitorowanie wydawanych posiłków i diet.
- **Calendar App**: Interaktywny harmonogram zajęć, świąt i wydarzeń przedszkolnych.

1.2 🏗️ Architektura Systemu

Projekt wykorzystuje nowoczesny stos technologiczny, zapewniający stabilność i skalowalność:

Komponent	Technologia	Opis
Backend	Django 4.x	Stabilna logika biznesowa i ORM.
Tasks	Celery & Redis	Obsługa powiadomień i zadań w tle.
UI Framework	Bootstrap 5	Responsywny interfejs użytkownika przez `crispy_bootstrap5`.
Frontend	Crispy Forms	Eleganckie i walidowane formularze systemowe.

1.3 📊 Schemat Powiązań Bazodanowych

Poniższy diagram przedstawia architekturę relacji, gdzie Dyrektor stanowi centralny punkt izolacji danych (każda placówka posiada własny graf powiązań).

1.4


```
erDiagram
    USER ||--|| DIRECTOR : "posiada profil"
    USER ||--|| TEACHER : "posiada profil"
    USER ||--|| PARENT : "posiada profil"

    DIRECTOR ||--o{ GROUP : "zarządza"
    DIRECTOR ||--o{ KID : "nadzoruje"
    DIRECTOR ||--o{ MEAL : "definiuje"
    DIRECTOR ||--o{ PAYMENT_PLAN : "ustala"

    GROUP ||--o{ KID : "zrzesza"
    GROUP ||--o{ TEACHER : "ma przypisanych"

    PARENT ||--o{ KID : "opiekuje się"

    KID ||--o{ PRESENCE : "posiada wpisy"
    KID }o--|| MEAL : "korzysta z"
    KID }o--|| PAYMENT_PLAN : "jest rozliczane"

    TEACHER ||--o{ POST : "publikuje"
    DIRECTOR ||--o{ POST : "publikuje"
    POST }o--|| GROUP : "jest adresowany do"
```

1.5 🚀 Jak zacząć?

Zapoznaj się z sekcją 🛠 **Wdrożenie i Start** w menu bocznym, aby uruchomić projekt lokalnie i zrozumieć główny przepływ danych.

2. 🛠 Wdrożenie i Start

2.1 Szybki Start (Dev)

2.1.1 🛠 Szybki Start dla Programisty

Ten przewodnik pomoże Ci błyskawicznie skonfigurować lokalne środowisko deweloperskie dla projektu **Kindergarten Management System**.

1. Wymagania systemowe

Zanim zaczniesz, upewnij się, że masz zainstalowane następujące technologie: * **Python 3.10** (zgodnie ze strukturą `.venv`). * **PostgreSQL** (jako główny silnik bazy danych). * **Redis** (wymagany jako broker wiadomości dla Celery).

2. Konfiguracja środowiska

Wykonaj poniższe polecenia w terminalu, aby przygotować projekt:

```
# 1. Sklonuj repozytorium
git clone https://github.com/DawidSzoka1/KindergartenDjangoDev
cd KindergartenDjangoDev
# 2. Utwórz i aktywuj środowisko wirtualne
python -m venv .venv
source .venv/bin/activate  # Windows: .venv\Scripts\activate

# 3. Zainstaluj wymagane biblioteki
pip install -r requirements.txt
```

3. Konfiguracja bazy dancyh (PostgreSQL)

System do poprawnego działania wymaga pliku z lokalnymi ustawieniami. Utwórz plik MarchewkaDjango/local_settings.py i uzupełnij go swoimi danymi:

```
# MarchewkaDjango/local_settings.py

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'name',
        'USER': 'usser',
        'PASSWORD': 'pass',
        'HOST': 'localhost',
        'PORT': 5432,
    }
}
secret = 'secret' # Twoj unikalny klucz SECRET_KEY
```

4. Inicjalizacja bazy danych i plików

Po skonfigurowaniu połączenia z PostgreSQL, przygotuj strukturę bazy danych oraz zasoby statyczne:

```
# Wygeneruj i wykonaj migracje
python manage.py makemigrations
python manage.py migrate

# Zbierz pliki statyczne (wymagane dla CSS/JS)
python manage.py collectstatic

# Opcjonalnie: Utwórz administratora panelu Django
python manage.py createsuperuser

# Uruchomienie serwera
python manage.py runserver
```

Adres do pracy: System będzie dostępny pod adresem, który wyświetli serwer WWW (zazwyczaj http://127.0.0.1:8000/).

2.2 Główny Przepływ (Flow)

2.2.1 🔄 Główny Scenariusz Operacyjny (End-to-End Flow)

Poniższy opis przedstawia modelową ścieżkę życia danych w systemie — od momentu założenia placówki po miesięczne rozliczenie. Zrozumienie tego przepływu jes>

1. Inicjacja Systemu (Accounts & Director)

- **Rejestracja Dyrektora**: Proces zaczyna się od rejestracji nowego Dyrektora. System automatycznie tworzy unikalną i odizolowaną instancję placówki przed>
- **Konfiguracja Mediów**: Dyrektor wgrywa ikony dla grup i posiłków (automatycznie skalowane do 300x300px), co jest wymagane do dalszej pracy w systemie.

2. Przygotowanie Fundamentów (Groups, Meals, Payments)

- **Tworzenie Struktur**: Dyrektor definiuje grupy (np. "Biedronki"), plany żywieniowe (np. "Dieta Standardowa") oraz stałe plany płatnicze (czesne).
- **Blokada Walidacyjna**: System uniemożliwia dodanie dziecka, dopóki te trzy elementy nie zostaną poprawnie skonfigurowane dla danej placówki.

3. Zarządzanie Kadrami i Społecznością (Teacher & Parent)

- **Zatrudnienie Nauczycieli**: Dyrektor zaprasza kadrę pedagogiczną. Nauczyciele są przypisywani do konkretnych grup, co daje im wgląd w dane dzieci z tych>
- **Onboarding Rodziców**: Dyrektor wysyła zaproszenia mailowe do rodziców. Po zalogowaniu rodzic widzi wyłącznie profile dzieci, które zostały mu formalnie>

4. Życie Codzienne (Children, Calendar, Blog)

- **Monitorowanie Frekwencji**: Nauczyciele każdego ranka sprawdzają listę obecności w interaktywnym kalendarzu.
- **Proaktywność Rodziców**: Rodzice zgłaszają planowane nieobecności (np. wizyta u lekarza) z wyprzedzeniem.
- **Komunikacja**: Nauczyciele i dyrekcja publikują ogłoszenia, filtrując odbiorców według konkretnych grup przedszkolnych.

3. 🧩 Moduły Aplikacji

3.1 🔐 Accounts (Konta)

3.1.1 🔐 Moduł Kont i Autoryzacji (Accounts)

Moduł odpowiada za fundament bezpieczeństwa systemu, zarządzanie tożsamością użytkowników oraz definiowanie hierarchii dostępu w ramach placówek przedszkoln>

Logika Biznesowa i Hierarchia

- **Model "Dyrektor-Pierwszy"**: Proces rejestracji (`Register`) jest przeznaczony wyłącznie dla nowych Dyrektorów. Każda rejestracja automatycznie tworzy n>
- **System Zaproszeń**: Pozostali użytkownicy (Nauczyciele, Rodzice) nie rejestrują się samodzielnie. Są oni dodawani do konkretnej placówki przez Dyrektora>
- **Custom User Model**: System rezygnuje z tradycyjnych nazw użytkownika (username) na rzecz unikalnych adresów e-mail jako głównego identyfikatora logowan>

🔄 Workflow / Przykład użycia:

🎓 DLA NOWEGO DYREKTORA (WŁAŚCICIELA)

- **Zadanie**: Założenie nowej placówki przedszkolnej w systemie.
- **Działanie**: Dyrektor wypełnia formularz rejestracyjny. System tworzy obiekt `User` , nadaje mu uprawnienia `is_director` i automatycznie inicjuje powiąz>

👥 DLA PRACOWNIKÓW I RODZICÓW

- **Zadanie**: Uzyskanie dostępu do panelu swojej placówki.
- **Działanie**: Użytkownik otrzymuje wiadomość e-mail z danymi dostępowymi. Przy pierwszym logowaniu system wymusza lub sugeruje zmianę tymczasowego hasła >

Kluczowe Funkcjonalności Techniczne

- **Zarządzanie uprawnieniami**: Wykorzystuje system grup i pozwoleń Django do rygorystycznego separowania ról: Dyrektor (zarządzanie), Nauczyciel (edukacja>
- **Bezpieczeństwo sesji**: Zastosowano `update_session_auth_hash` . Dzięki temu po zmianie hasła przez użytkownika jego aktualna sesja nie zostaje przerwana>

Dokumentacja Techniczna

3.1.2 `accounts.models.User`

Bases: `AbstractBaseUser` , `PermissionsMixin`

Source code in `accounts/models.py`

```
38  class User(AbstractBaseUser, PermissionsMixin):
39      email = models.EmailField(db_index=True, unique=True, max_length=254)
40
41      is_staff = models.BooleanField(default=False)
42      is_active = models.BooleanField(default=True)
43      is_superuser = models.BooleanField(default=False)
44
45      objects = CustomUserManager()
46
47      USERNAME_FIELD = 'email'
48      REQUIRED_FIELDS = []
49
50      class Meta:
51          verbose_name = 'User'
52          verbose_name_plural = 'Users'
```

3.1.3 `accounts.views.Register`

Bases: `View`

Source code in `accounts/views.py`

```
11  class Register(View):
12      def get(self, request):
13          if request.user.is_active:
14              messages.info(request, 'Masz juz konto')
15              return redirect('home_page')
16          form = UserRegisterForm()
17          return render(request, "register.html", {'form': form})
18
19      def post(self, request):
20          if request.user.is_active:
21              messages.info(request, 'Masz juz konto')
22              return redirect('home_page')
23          form = UserRegisterForm(request.POST)
24          if form.is_valid():
25              email = form.cleaned_data.get('email')
26              if User.objects.filter(email=email).exists():
27                  form.add_error('email', 'Ten email jest już zajęty.')
28              else:
29                  form.save()
30                  messages.success(request, 'Konto zostało utworzone. Możesz się teraz zalogować.')
31                  return redirect('login')
32          else:
33              messages.error(request, "W formularzu wystąpiły błędy. Sprawdź pola.")
34          return render(request, "register.html", {'form': form})
```

3.1.4 `accounts.views.ProfilePasswordUpdate`

Bases: `LoginRequiredMixin`, `View`

Source code in `accounts/views.py`

```
37  class ProfilePasswordUpdate(LoginRequiredMixin, View):
38      def get(self, request):
39
40          password_form = PasswordChangeForm(request.user)
41
42          context = {
43              'password_form': password_form,
44          }
45          return render(request, 'change_password.html', context)
46
47      def post(self, request):
48          password_form = PasswordChangeForm(request.user, request.POST)
49          if password_form.is_valid():
50              user = password_form.save()
51              update_session_auth_hash(request, user)
52
53              return redirect('home_page')
54          messages.error(request, 'wypelnij poprawnie formularz')
55          return redirect('password_change')
```

```
erDiagram
    %% CORE TABLES: Accounts
```

```
    ACCOUNTS_USER {
        bigint id PK
        varchar password
        varchar email UK "LOGIN"
        boolean is_staff
        boolean is_active
        boolean is_superuser
    }

    %% RELATIONS: Permissions & Groups (Standard Django Auth)
    AUTH_GROUP {
        integer id PK
        varchar name UK
    }
    AUTH_PERMISSION {
        integer id PK
        varchar codename UK
        integer content_type_id FK
    }

    %% JUNCTION TABLES (ManyToMany)
    ACCOUNTS_USER_GROUPS {
        bigint id PK
        bigint user_id FK
        integer group_id FK
    }
    ACCOUNTS_USER_USER_PERMISSIONS {
        bigint id PK
        bigint user_id FK
        integer permission_id FK
    }

    %% KEY LINKS
    ACCOUNTS_USER ||--o{ ACCOUNTS_USER_GROUPS : "ManyToMany - Membership"
    AUTH_GROUP ||--o{ ACCOUNTS_USER_GROUPS : "ManyToMany - Group"

    ACCOUNTS_USER ||--o{ ACCOUNTS_USER_USER_PERMISSIONS : "ManyToMany - User Permissions"
    AUTH_PERMISSION ||--o{ ACCOUNTS_USER_USER_PERMISSIONS : "ManyToMany - Permission"
```

3.2 🎓 Director (Dyrektor)

3.2.1 🎓 Moduł Dyrektora (Director)

Moduł ten służy jako **Centrum Dowodzenia i Zarządzania Placówką**. Dyrektor pełni rolę głównego administratora, kontrolując wszystkie dane, zasoby i konfig>

Logika Biznesowa (Izolacja Danych)

- **Zarządzanie Własną Placówką**: Wszystkie zasoby – od dzieci (`Kid`), przez posiłki (`Meals`), po zdjęcia (`GroupPhotos`) – są ściśle powiązane z obiekte>
- **Personalizacja Systemu**: Dyrektor jako jedyny definiuje kluczowe elementy operacyjne, takie jak:
- **Standardy Graficzne**: Zarządzanie biblioteką zdjęć i ikon do identyfikacji grup i posiłków.
- **Dane Kontaktowe**: Ustalanie oficjalnych danych teleadresowych placówki (`ContactModel`) z walidacją formatów.

🔄 Workflow / Przykład użycia:

🎓 DLA DYREKTORA (ZARZĄDZANIE ZASOBAMI)

- **Zadanie**: Dyrektor chce zoptymalizować przestrzeń dyskową, jednocześnie zachowując wysoką jakość wizualną.
- **Działanie**: Przy wgrywaniu zdjęcia grupy (przez `PhotosAddView`), system automatycznie skaluje obraz do maksymalnych wymiarów **300x300px**, co poprawi>
- **Zadanie**: Dyrektor chce delegować część obowiązków wicedyrektorowi.
- **Działanie**: Dyrektor używa widoku `GiveDirectorPermissions` , aby nadać uprawnienia `is_director` innemu pracownikowi (`Employee`), który od tego moment>

Dokumentacja Techniczna

3.2.2 `director.models.Director`

Bases: `Model`

Source code in `director/models.py`

```python
class Director(models.Model):
    gender_choices = ((1, 'Mezczyzna'), (2, 'Kobieta'))
    first_name = models.CharField(max_length=128, null=True)
    last_name = models.CharField(max_length=128, null=True)
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    gender = models.IntegerField(choices=gender_choices, null=True)

    class Meta:
        permissions = [
            ("is_director", "Is the director of kindergarten")
        ]

    def __str__(self):
        return f"{self.user.email}"
```

3.2.3 `director.models.GroupPhotos`

Bases: `Model`

Source code in `director/models.py`

```
34  class GroupPhotos(models.Model):
35      group_photos = models.ImageField(null=True, upload_to=user_group_path, unique=True)
36      principal = models.ForeignKey(Director, on_delete=models.CASCADE, null=True)
37      name = models.CharField(max_length=64, null=True)
38      is_active = models.BooleanField(default=True)
39      date_created = models.DateTimeField(auto_now_add=True, null=True)
40
41      def save(self, *args, **kwargs):
42          super().save(*args, **kwargs)
43          if self.group_photos:
44              img = Image.open(self.group_photos.path)
45
46              if img.height > 300 or img.width > 300:
47                  output_size = (300, 300)
48                  img.thumbnail(output_size)
49                  img.save(self.group_photos.path)
50
51      def __str__(self):
52          return f'{self.group_photos.url}'
```

3.2.4 `director.models.ContactModel`

Bases: `Model`

Source code in `director/models.py`

```
85  class ContactModel(models.Model):
86      director = models.OneToOneField(Director, on_delete=models.CASCADE)
87      email_address = models.EmailField(null=True)
88      phone_regex = RegexValidator(regex=r'^\+?1?\d{9,15}$',
89                                   message="Numer telefonu w formacie: '+99 999 999 999'")
90      phone = models.CharField(null=True, unique=True, max_length=17, validators=[phone_regex])
91      localization = models.TextField(max_length=128, null=True)
```

3.2.5 `director.views.PhotosAddView`

Bases: `PermissionRequiredMixin`, `View`

Source code in `director/views.py`

```
40  class PhotosAddView(PermissionRequiredMixin, View):
41      permission_required = "director.is_director"
42
43      def get(self, request):
44          return render(request, 'photo-add.html')
45
46      def post(self, request):
47          photo_type = request.POST.get('type')
48          file = request.FILES.get('file')
49          name = request.POST.get('name')
50          if not photo_type:
51              messages.error(request, 'Wybierz typ iconki')
52              return redirect('photo_add')
53          elif not file:
54              messages.error(request, 'Wybierz jakis plik')
55              return redirect('photo_add')
56          elif not name:
57              messages.error(request, 'Nazwa jest wymagana')
58              return redirect('photo_add')
59          if photo_type == 'group':
60              GroupPhotos.objects.create(group_photos=file, principal=request.user.director, name=name)
61          elif photo_type == 'meal':
62              MealPhotos.objects.create(meal_photos=file, principal=request.user.director, name=name)
63          messages.success(request, 'Poprawnie dodano ikonkę')
64          return redirect('photos_list')
```

3.2.6 `director.views.GiveDirectorPermissions`

Bases: `PermissionRequiredMixin`, `LoginRequiredMixin`, `View`

Source code in `director/views.py`

```
170  class GiveDirectorPermissions(PermissionRequiredMixin, LoginRequiredMixin, View):
171      permission_required = 'director.is_director'
172
173      def get(self, request):
174          employees = Employee.objects.filter(principal=Director.objects.get(user=request.user)).order_by('-id')
175          paginator = Paginator(employees, 10)
176          page = request.GET.get('page')
177          page_obj = paginator.get_page(page)
178          return render(request, 'give-director-permission.html', context={'page_obj': page_obj})
179
180      def post(self, request):
181          search = request.POST.get('search')
182          if search:
183              employees = Employee.objects.filter(principal=Director.objects.get(user=request.user)).filter(
184                  user__email__icontains=search).order_by('-id')
185              paginator = Paginator(employees, 10)
186              page = request.GET.get('page')
187              page_obj = paginator.get_page(page)
188              return render(request, 'give-director-permission.html', context={'page_obj': page_obj})
189          content_type = ContentType.objects.get_for_model(Director)
190          permission = Permission.objects.get(content_type=content_type, codename='is_director')
191          pk = list(map(int, request.POST.getlist('pk')))
192          employees = Employee.objects.filter(id__in=pk)
193          for employee in employees:
194              employee.user.user_permissions.add(permission)
195
196          return redirect('give-permissions')
```

KLUCZOWE ZABEZPIECZENIA:

- **Weryfikacja Własności**: Każdy widok w tym module rygorystycznie sprawdza, czy edytowany obiekt (np. zdjęcie lub dane kontaktowe) należy do zalogowanego>

```
erDiagram
    %% CORE TABLE: Director Profile
    ACCOUNTS_USER ||--|| DIRECTOR_DIRECTOR : "OneToOne, Właściciel"

    DIRECTOR_DIRECTOR {
        bigint id PK
        bigint user_id FK UK
        varchar first_name
        varchar last_name
        integer gender
    }

    %% RELATIONSHIPS: Core Director Resources
    DIRECTOR_DIRECTOR ||--|| DIRECTOR_CONTACTMODEL : "OneToOne, Dane Placówki"
    DIRECTOR_DIRECTOR ||--o{ DIRECTOR_GROUPPHOTOS : "OneToMany, Ikony Grup"
    DIRECTOR_DIRECTOR ||--o{ DIRECTOR_MEALPHOTOS : "OneToMany, Ikony Posiłków"
    DIRECTOR_DIRECTOR ||--o{ DIRECTOR_FREEDAYSMODEL : "OneToMany, Dni Wolne"
    TEACHER_EMPLOYEE }o--|| DIRECTOR_DIRECTOR : "ManyToMany, Przełożony"
    PARENT_PARENTA }o--|| DIRECTOR_DIRECTOR : "ManyToMany, Nadzór Rodzica"

    %% DETAIL TABLES
    DIRECTOR_CONTACTMODEL {
        bigint id PK
        bigint director_id FK UK
        varchar phone UK "Walidacja REGEX"
        varchar email_address
        text localization
    }
    DIRECTOR_GROUPPHOTOS {
        bigint id PK
        bigint principal_id FK
        varchar group_photos UK "Plik"
        varchar name
    }
    DIRECTOR_FREEDAYSMODEL {
        bigint id PK
        bigint principal_id FK
        varchar title
        timestamp start_time
        timestamp end_time
    }
```

3.3 👩‍🏫 Teacher (Kadry)

3.3.1 👩‍🏫 Moduł Nauczyciela i Kadr (Teacher)

Moduł ten odpowiada za kompleksowe zarządzanie personelem placówki, definiowanie ról zawodowych oraz nadzór nad strukturą zatrudnienia i wynagrodzeniami.

Logika Biznesowa (Zarządzanie Kadrami i Widoczność)

- **Zróżnicowanie Ról**: System pozwala na definiowanie ról takich jak nauczyciel, pomoc nauczyciela czy inne stanowiska administracyjne, przypisując im odp>
- **Pełna Izolacja Płacowa**: Dane o wynagrodzeniach (`salary`) są ściśle chronione i widoczne wyłącznie dla Dyrektora placówki. Nauczyciele ani rodzice nie>
- **Inteligentna Widoczność Profilu**: Profil nauczyciela jest udostępniany rodzicom w sposób dynamiczny – rodzic widzi dane kontaktowe wychowawcy tylko wte>

🔄 **Workflow / Przykład użycia:**

🎓 SCENARIUSZ: ZATRUDNIENIE NOWEGO CZŁONKA PERSONELU

- **Akcja**: Dyrektor rejestruje pracownika za pomocą widoku `EmployeeAddView`, podając jego e-mail i rolę.
- **Automatyzacja**: System w ramach jednej transakcji tworzy konto użytkownika, nadaje mu uprawnienia techniczne `is_teacher`, generuje losowe hasło i wysy>
- **Zaleta**: Dyrektor oszczędza czas na ręcznej konfiguracji kont, a system gwarantuje, że nowy pracownik od razu trafia pod nadzór właściwego administrato>

👩‍🏫 SCENARIUSZ: AKTUALIZACJA DANYCH I ZARZĄDZANIE GRUPĄ

- **Akcja**: Nauczyciel uzupełnia swój profil o numer telefonu (walidowany przez Regex) w celu ułatwienia kontaktu z rodzicami.
- **Działanie**: Dyrektor może w dowolnym momencie zmienić przydział nauczyciela do grupy za pomocą `ChangeEmployeeGroupView`, co automatycznie aktualizuje >

Dokumentacja Techniczna

3.3.2 `teacher.models.Employee`

Bases: `Model`

Source code in `teacher/models.py`

```
class Employee(models.Model):
    gender_choices = ((1, 'Mezczyzna'), (2, 'Kobieta'))
    first_name = models.CharField(max_length=128, null=True)
    last_name = models.CharField(max_length=128, null=True)
    role = models.IntegerField(choices=roles, default=3)
    salary = models.DecimalField(max_digits=10, decimal_places=2, null=True)
    group = models.ForeignKey(Groups, on_delete=models.CASCADE, null=True)
    phone_regex = RegexValidator(regex=r'^(?<!\w)(\(?(\+|00)?48\)?)?[ -]?\d{3}[ -]?\d{3}[ -]?\d{3}(?!\w)$',
                                 message="Numer telefony musi byc w formie: '+48 999 999 999' albo '(+48 999 999 999)' albo '999 999 999'.",
                                 code='invalid_numer_telefonu')
    phone = models.CharField(null=True, unique=True, max_length=17, validators=[phone_regex])
    city = models.CharField(max_length=64, null=True)
    address = models.CharField(max_length=128, null=True)
    zip_code = models.CharField(null=True, max_length=6, validators=[RegexValidator(
        regex=r'^([0-9]{2}-[0-9]{3})$',
        message=(u'Kod pocztowy musi byc w formacie 00-000'),
        code="invalid_zip_code",
    )])
    gender = models.IntegerField(choices=gender_choices, null=True)
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    principal = models.ManyToManyField(Director)
    is_active = models.BooleanField(default=True)

    class Meta:
        permissions = [
            ("is_teacher", "Is the teacher of some group")
        ]

    def __str__(self):
        return f"{self.user.email}"
```

3.3.3 `teacher.views.EmployeesListView`

Bases: `PermissionRequiredMixin`, `LoginRequiredMixin`, `View`

Source code in `teacher/views.py`

```
class EmployeesListView(PermissionRequiredMixin, LoginRequiredMixin, View):
    permission_required = "director.is_director"

    def get(self, request):
        # 1. Pobieranie danych i filtry
        user_director = Director.objects.get(user=request.user.id)

        # Pobieranie parametru wyszukiwania z GET
        search_query = request.GET.get('search', '')

        # Używamy startowego QuerySetu
        teachers = Employee.objects.filter(principal=user_director)

        # 2. Logika filtrowania (jeśli jest zapytanie)
        if search_query:
            # Wyszukujemy po emailu (z icontains)
            teachers = teachers.filter(
                user__email__icontains=search_query
            )

        # Sortowanie
        teachers = teachers.order_by('-id')

        # 3. Paginacja
        paginator = Paginator(teachers, 10)
        page_number = request.GET.get("page")
        page_obj = paginator.get_page(page_number)

        # Dodajemy 'search_query' do kontekstu, aby móc go użyć w szablonie (np. do zachowania wartości w polu wyszukiwania)
        context = {
            'page_obj': page_obj,
            'search_query': search_query,
        }

        return render(request, 'employees-list.html', context)

    def post(self, request):
        # W metodzie POST tylko pobieramy wartość z formularza i przekierowujemy do GET
        search = request.POST.get('search')

        base_url = reverse('list_teachers')

        if search:
            # 2. Dodaj parametr zapytania (?search=...) do podstawowego URL
            full_url = f'{base_url}?search={search}'
            return redirect(full_url)

        # Jeśli pole było puste, nadal wracamy do czystej listy
        return redirect('list_teachers')
```

3.3.4 `teacher.views.EmployeeProfileView`

Bases: `LoginRequiredMixin`, `View`

Source code in `teacher/views.py`

```
class EmployeeProfileView(LoginRequiredMixin, View):

    def get(self, request, pk):
        employee = get_object_or_404(Employee, id=int(pk))
        user = self.request.user
        if user.get_user_permissions() == {'director.is_director'}:
            if employee.principal.first() == user.director:
                groups = Groups.objects.filter(principal=user.director, is_active=True )
                return render(request, 'employee-profile.html',
                              {'employee': employee, 'groups_list': groups})
            messages.error(request, f"Nie ma takiego nauczyciela")
            return redirect('list_teachers')
        elif user.get_user_permissions() == {'teacher.is_teacher'}:
            if user.employee == employee:
                return render(request, 'employee-profile.html',
                              {'employee': employee})
        elif user.get_user_permissions() == {'parent.is_parent'}:
            group = employee.group
            kids = group.kid_set.filter(is_active=True)
            parent = ParentA.objects.get(user=user.id)
            allow = False
            for kid in kids:
                if kid in parent.kids.filter(is_active=True):
                    allow = True
                    break
            if allow:
                return render(request, 'employee-profile.html',
                              {'employee': employee})
        raise PermissionDenied
```

3.3.5 `teacher.views.EmployeeAddView`

Bases: `PermissionRequiredMixin`, `View`

Source code in `teacher/views.py`