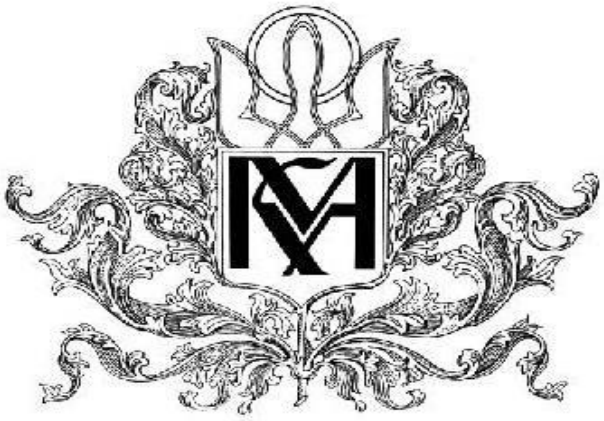
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики



Розробка CRM-системи брокерської компанії

Текстова частина до курсової роботи за спеціальністю
«Комп’ютерні науки» 122

Керівник курсової роботи
Сініцина Р. Б.

_____

(підпис)

“____” _____ 2020 р.

Виконала студентка
Харченко М. В.

“____” _____ 2020 р.

Київ 2020

Індивідуальне завдання

на курсову роботу

Студентці Харченко Марині Вадимівні факультету інформатики 3 курсу

ТЕМА: Розробка CRM-системи брокерської компанії

Вихідні дані:

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Частина 1: Огляд існуючих рішень

Частина 2: Проектування системи

Частина 3: Огляд технологій

Частина 4: Програмна реалізація системи

Висновки

Список використаної літератури

Додатки

Тема: Розробка CRM-системи

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	07.10.2019	
2.	Пошук тематичної літератури	20.10.2019	
3.	Ознайомлення з літературою	25.10.2019	
4.	Вивчення предметної області	05.11.2019	
5.	Огляд засобів реалізації	08.11.2019	
6.	Вивчення технології для розробки системи	20.01.2020	
7.	Побудова технічного завдання	05.02.2020	
8.	Написання практичної частини	10.02.2020	
9.	Написання першого розділу теоретично ї частини	04.03.2020	
10.	Написання другого розділу теоретично ї частини	15.04.2020	
11.	Написання третього розділу теоретичної розділу частини	24.04.2020	
12.	Написання четвертого розділу теоретично ї частини	30.04.2020	
13.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	08.05.2020	
14.	Створення презентації	15.05.2020	
15.	Захист роботи	20.05.2020	

Студент Харченко М. В.

Керівник Сініцина Р. Б.

“____” _____

Зміст

Анотація

Під час виконання даної курсової роботи були проаналізовані існуючі рішення та була створена власна CRM-система за заданою предметною областю. Було проаналізовано процес розробки та можливості, що надають використані технології.

Вступ

Наразі, існує велика кількість компаній, які потребують полегшення відслідковування своїх бізнес процесів, а якщо можливо, то і збільшення їх автоматизованості. Для того, щоб частково задовільнити ці потреби, компанії створюють або замовляють CRM-системи. CRM-система (англ. Customer relationship management) – система управління відносинами з клієнтами. Така система створюється лише для підвищення ефективності роботи співробітників компанії, тобто CRM-системи створюються лише для використання всередині компанії.

Об’єкт дослідження: процес роботи систем управління відносинами з клієнтами.
Мета роботи: створити власну систему, що для поглиблення розуміння об’єкту дослідження та вивчення нових технологій розробки.

Розділ 1. Огляд існуючих рішень

Наразі, на ринку представлена велика кількість CRM-систем, які можна адаптувати під потреби кожної конкретної компанії.

1.1 Microsoft Dynamics 365

Microsoft Dynamics 365 є відомою CRM-системою, з великою користувацькою базою та великим рівнем довіри з її сторони.

1.2 Terrasoft Creatio Studio

Terrasoft Creatio Studio надає великий набір інструментів, для ведення бізнесу, в тому числі власну CRM-систему. Перевагою даної платформи є її зручний для користування графічний інтерфейс. Для роботи з системою потрібно оформити підписку щонайменше на місяць. Платформа надає доступ до такого функціоналу:

- Управління бізнес-процесами
- Управління кейсами
- Інтелектуальні технології
- Дизайнер системи
- Інтеграційні рішення
- Мобільний застосунок
- Безпека та адміністрування
- Розробка застосунків

Дані CRM-системи як і подібні до них є дуже гнучкими та мають достатній функціонал, щоб задовільнити всі можливі потреби, що можуть виникнути у компанії. Проте, саме їх універсальність, в той же час, є їх недоліком. Ці системи потребують великий рівень ознайомленості та досвіду з їх архітектурою та засобами розробки. Тому часто лише розробники, що спеціалізуються на конкретній CRM-системі можуть адаптувати функціонал системи до потреб компанії-клієнта та підтримувати створене ними програмне забезпечення. Цей фактор додатково підвищує вартість створення системи на основі вже існуючого рішення, не кажучи про вартість підписки існуючої системи. Таким чином, компанія, що потребує CRM-систему, може вважати за краще звернутися до інструментів, що використовуються більш широко та створити повністю свою систему.

Розділ 2. Проектування системи

CRM-система розробляється для підвищення ефективності брокерської компанії, полегшення роботи для її співробітників та перегляд статистичних даних. Діяльність компанії – допомога в укладанні угод по купівлі та продажу цінних паперів. Компанія працює лише з юридичними особами зареєстрованими в Україні. Клієнтами компанії можуть бути як і інвестори, так і емітенти.

2.1 Формулювання предметної області

Співробітниками, які грають основну роль в бізнес-процесах компанії є брокери. Брокери ведуть переговори з одним чи декількома представниками компанії-клієнта. Компанія може неодноразово працювати зі своїми клієнтами. При висловленні компанією-клієнтом бажання щодо купівлі чи продажу цінних паперів, брокер створює нову справу. Справа може бути закрита успішно чи неуспішно, в залежності від того, чи була укладена угода. Компанія працює лише з акціями, обіг яких дозволений законодавством держави України, відповідно, їх вартість оцінюється в державній валюті.

2.2 Формальний опис бізнес-процесів

Терміни:

Брокер	Співробітник даної компанії, що належить до групи користувачів «Брокер». Є учасником більшості бізнес-процесів компанії
Співробітник	Людина, що працює в даній компанії та необов’язково є брокером. Співробітник може стати одним з суперкористувачів, чи повинен мати доступ до системи для перегляду статистичних даних.

Компанія	Компанія-клієнт – юридична особа, що є клієнтом даної компанії.
Контактна особа	Людина, що представляє інтереси компанії-клієнта, основна людина, з якою проводяться переговори щодо справ.
Лід	Компанія-клієнт, що висловила бажання отримати послуги, проте про яку дана компанія не має достатньої інформації. Лід потім може бути конвертований у компанію.
Справа	Інформаційний об’єкт, що містить дані про поточний стан підготовлюваної угоди та історію її змін.

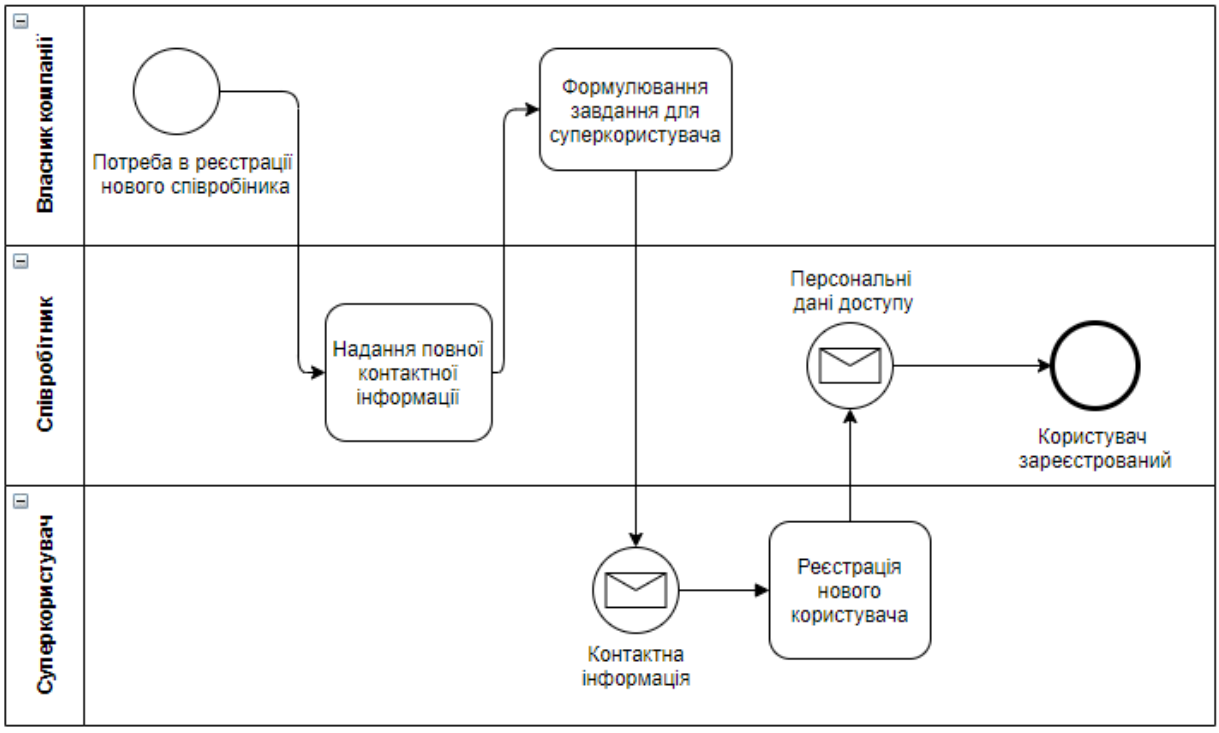


Рис. 2.2.1 Реєстрація в системі нового користувача

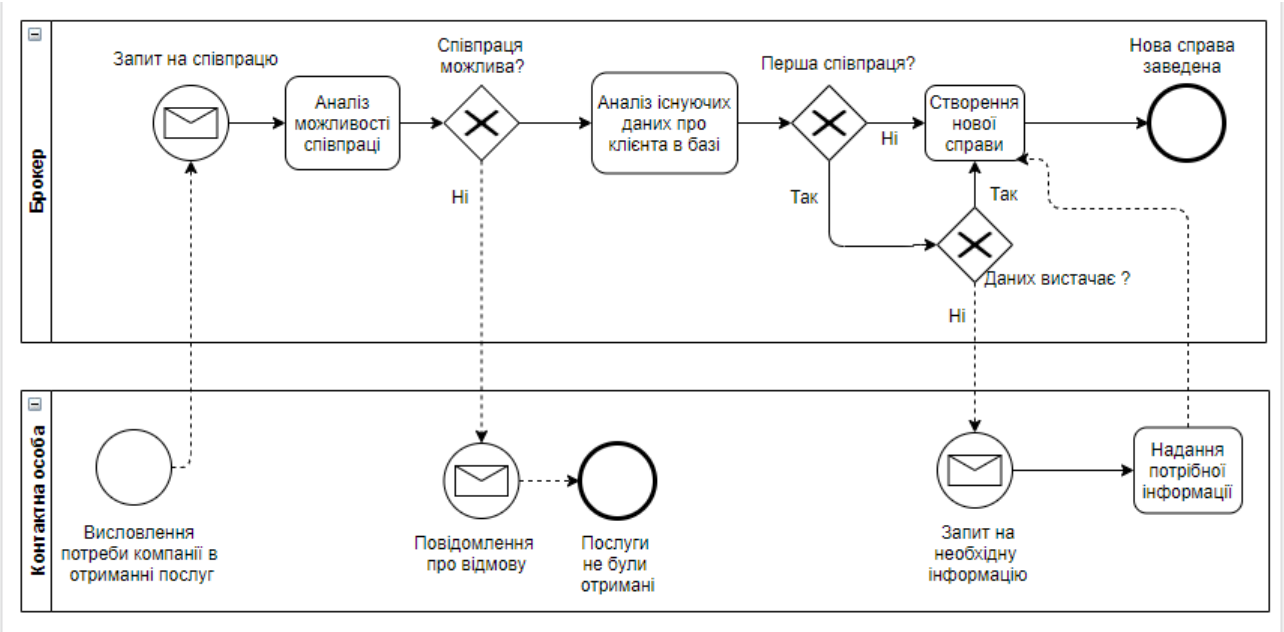


Рис. Створення нової справи

Розділ 3. Огляд технологій

Для реалізації власної системи було обрано Django - веб-фреймворк з відкритим кодом, для роботи з яким використовується мова програмування високого рівня Python. Django створений для розробки веб-застосунків з клієнт-серверною архітектурою, він надає велику кількість інструментів, що значно підвищують ефективність розробника.

3.1. Структура Django проекту

Веб-фреймворк Django впроваджує певну структуру, яку наслідують всі проекти, створені на його основі. Проект обов'язково містить утиліту manage.py для власного командного рядка та декілька конфігураційних файлів, в яких вказуються загальні налаштування для всіх застосунків (apps), що будуть створені у цьому проекті. Проект може містити один або декілька застосунків. Кожен застосунок є окремим логічним модулем. В межах кожного застосунку використовується паттерн Model-Template-View.

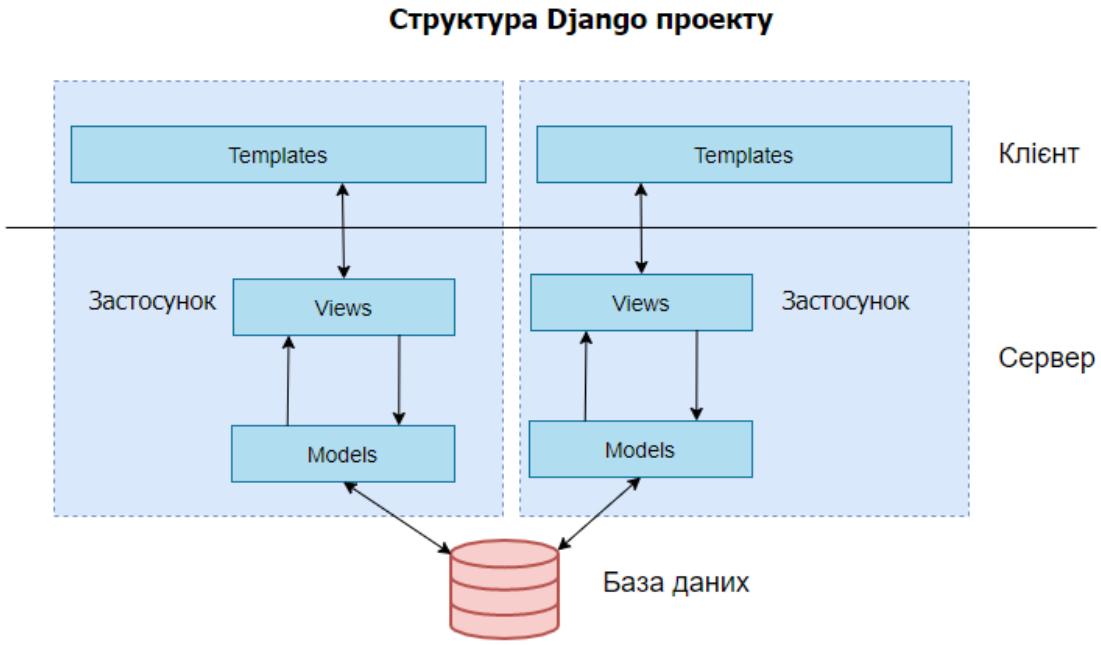


Рис. 3.1.1. Структура Django проекту

Така архітектура забезпечує чітку структуру проекту, навіть при дуже великих масштабах розробки.

3.2 Паттерн Model-Template-View

Веб-фреймворк Django використовує архітектурний шаблон проектування Model-Template-View (MTV).

1. Частина Model описує інформаційні об’єкти та служить інтерфейсом для роботи з даними, що зберігаються в базі даних.
2. Частина View описує те, *які* дані будуть представлені користувачеві.
3. Частина Template описує те, *як* представлення дані будуть представлені користувачеві.

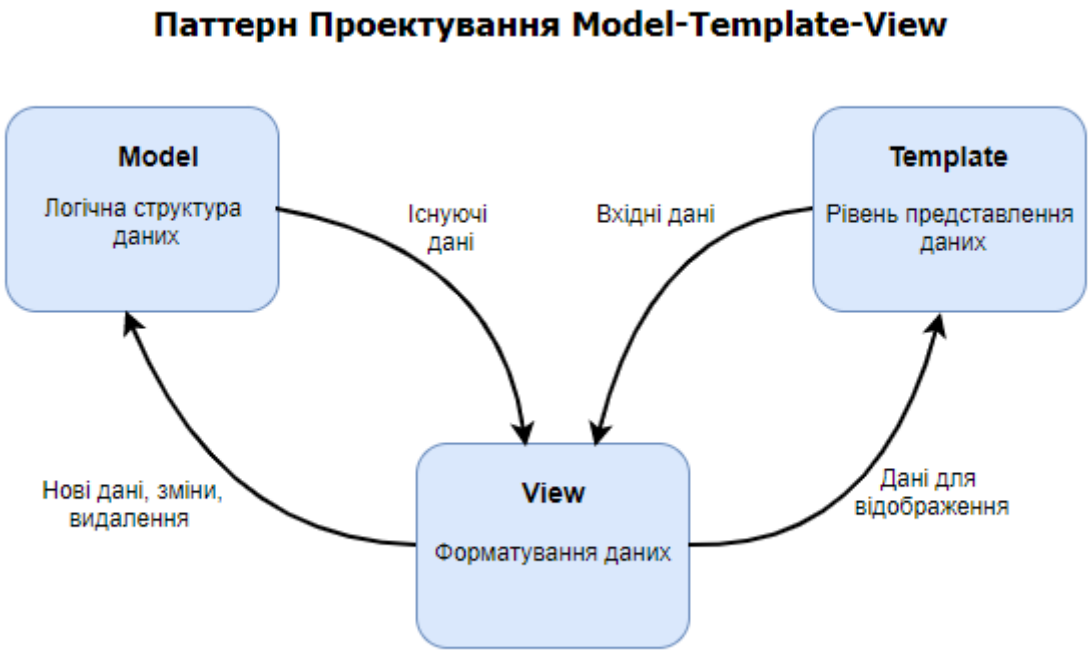


Рис. 3.1.2. Паттерн проектування Model-Template-View

3.3 Взаємодія з базою даних через Django ORM

Веб-фреймворк Django надає вичерпний набір інструментів для роботи з базами даних. Розробник отримує в своє розпорядження Django ORM, що дозволяє виконувати взаємодію з обраною базою даних через API високого рівня абстракції. Щоб визначити створену модель, як модель даних потрібно успадкувати її від класу django.db.models.Model та описати відношення з іншими моделями даних, якщо такі відношення існують. Дана технологія об'єктно-реляційної проекції часто значно пришвидшує процес розробки.

3.4 Робота з URL dispatcher

URL dispatcher – технологія, що забезпечує роботу URL схеми в застосунку, використовуючи регулярні вирази. Для створення URL схеми для застосунку, потрібно створити Python-модуль та дати назву URLconf.py чи urls.py. Цей модуль повинен містити список з назвою urlpatterns, що зберігає результати виконання функцій path(route, view, kwargs, name) з модулю django.urls. об’єкти класу django.urls.path. Функція path приймає URL шаблон та посилання на потрібне view. Веб-фреймворк не накладає обмежень на складність URL шаблону та надає додаткові інструменти для забезпечення чистоти та ненадлишковості схеми.

3.5 Використання Python-модулю smtplib

Мова програмування Python надає можливість роботи з поштою через інтерфейс модуля smtplib. smtplib створює об’єкт сесії SMTP клієнта, що дає можливість надсилати пошту через SMTP та ESMTP протоколи. Веб-фреймворк Django використовує власні обгортки над даним інтерфейсом, для полегшення та прискорення роботи розробника. Для цього використовується Модуль django.core.mail. В файлі проекту settings.py визначаються всі необхідні параметри такі як: обраний SMPT клієнт, порт, користувач, тощо.

Розділ 4. Програмна реалізація системи

4.1 Створення та початкова підготовка проекту

Для розробки системи потрібно встановити в систему мову програмування Python. Веб-фреймворк Django, встановлюється за допомогою системи керування пакетами pip. Всі наступні команди виконуються у командному рядку.

Команда для встановлення Django:

```
…\> pip install django
```

Новий Django-проект створюється за командою:

```
…\> django-admin startproject crm_system
```

Проект повинен містити один або більше застосунків (apps). В проекті crm_system було створено два застосунки: crm та analytics. Перший буде містити головну логіку CRM-системи, другий застосунок – аналіз даних. Були використані команди:

```
…\> py manage.py startapp crm
…\> py manage.py startapp analytics
```

Створені застосунки потрібно зареєструвати у файлі crm_system\settings.py:

```
INSTALLED_APPS = [... , ‘crm’, ‘analytics’,]
```

Для подальшої роботи з застосунками потрібен суперкористувач.

Суперкористувач створюється за командою:

```
…\> py manage.py createsuperuser
```

Після виконання команди вказуються дані для авторизації цього користувача.

4.2. Застосунок crm

Застосунок crm інкапсулює в собі основні бізнес-процеси CRM-системи. На момент створення застосунок має таку структуру:

```
crm/
        __init__.py
        admin.py
        apps.py
        migrations/
                __init__.py
        models.py
        tests.py
        views.py
```

В подальшій розробці, для більшої зручності, було створено директорії models, views, templates. Кожна модель даних буде визначатись в окремому файлі з назвою моделі, та знаходитись у директорії models. Відповідно будуть створюватись всі екземпляри view та template.

Файл admin.py є конфігураційним файлом для панелі адміністратора. В ньому зберігаються налаштування прав доступу груп користувачів CRM-системи та реєструються всі моделі, до яких повинен бути доступ з панелі.

У файлі urls.py були визначені такі URL-шаблони:

```
urlpatterns = [
    path('create-email/<int:case_id>/', staff_member_required(create_email), name='create_email'),
    path('send-email/<int:email_id>/', staff_member_required(send_email), name='send_email'),
    path('convert/<int:lead_id>/', staff_member_required(convert), name='convert'),
]
```

4.2.1 Створення та реєстрація моделей даних у панелі адміністратора

При створенні власні моделі даних були успадковані від класу django.db.models.Model, що надає надають вичерпну систему типів (DateTime, CharField, ForeignKey, тощо), функцій для операцій з ними та зробили можливим зберігати створені моделі у базі даних. Кожен користувач системи є об’єктом класу django.contrib.auth.models.User, що забезпечує систему аутентифікації.
На Рис. 4.2.1.1 зображена UML діаграма класів, що представляє ієрархію моделей даних CRM-системи.

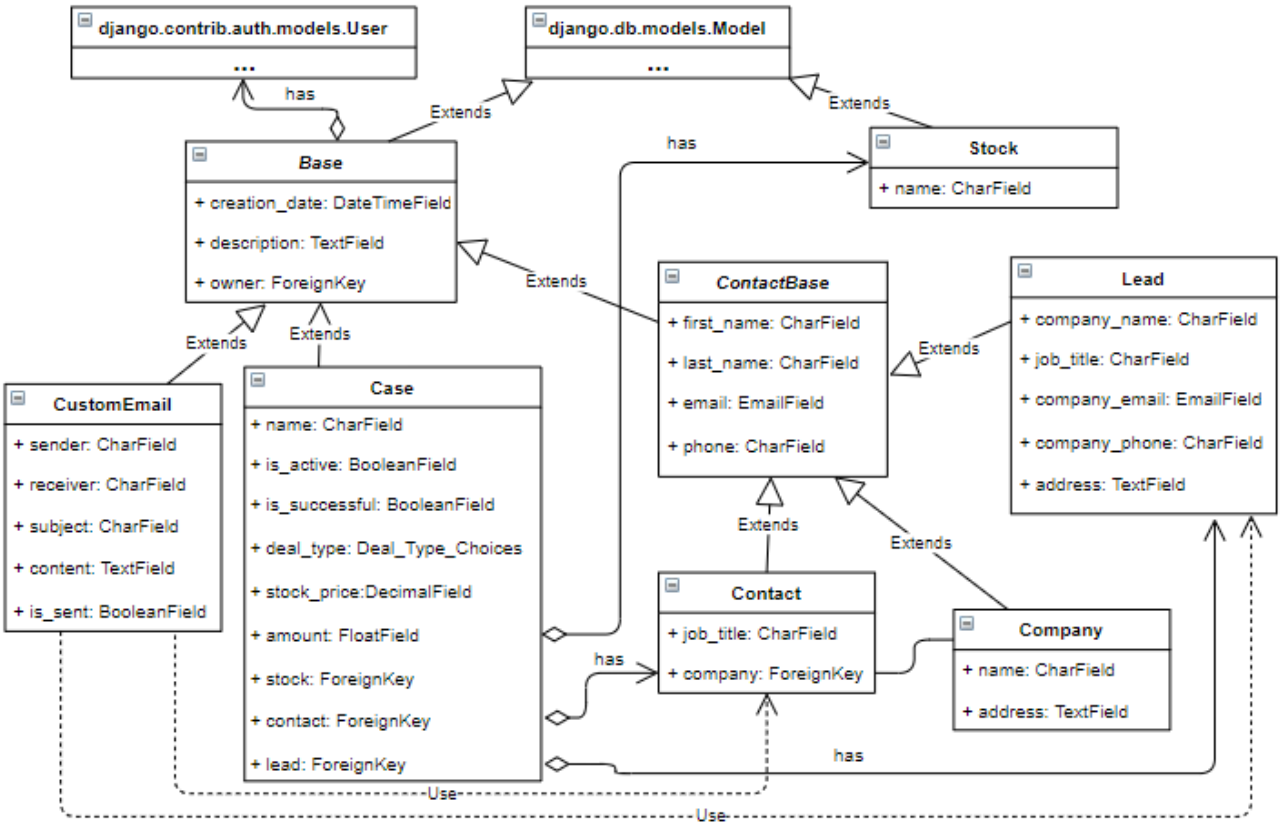


Рис. 4.2.1.1. UML діаграма класів системи

Для забезпечення гнучкості та ненадлишковості ієрархії класів були створені два абстрактних класи: Base та ContactBase, від них успадковується більшість моделей системи.

Реалізація класу Base:

```
class Base(models.Model):
    class Meta:
        abstract = True
    creation_date = models.DateTimeField(auto_now_add=True)
    description = models.TextField(blank=True)
    owner = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
```

Реалізація класу ContactBase:

```
class ContactBase(Base):
    class Meta:
        abstract = True
    first_name = models.CharField(
        verbose_name=_("First Name"),
        help_text=_("Person's first name"),
        max_length=250
    )
    last_name = models.CharField(
        verbose_name=_("Last Name"),
        help_text=_("Person's last name"),
        max_length=250
    )
    email = models.EmailField()
    phone = models.CharField(max_length=100)
```

В конфігураційному файлі admin.py був створений клас MyModelAdmin, що успадковується від django.contrib.admin.ModelAdmin. Клас створений з метою перевизначення методів батьківського класу. Були прибрані права доступу брокерів, що дозволяли змінювати чи видаляти дані, що стосуються іншого брокера. Всі класи, що містяться у admin.py та відповідають за представлення моделей даних у панелі адміністратора успадковані від класу MyModelAdmin. Для реєстрації моделі даних у панелі адміністратора використовується функція admin.site.register(Model, ModelAdmin), де Model – певна модель даних, ModelAdmin - клас успадкований від ModelAdmin.

Реалізація класу MyModelAdmin:

```
class MyModelAdmin(admin.ModelAdmin):

    def has_change_permission(self, request, obj=None):
        return self.get_permission_value(request, obj)

    def has_delete_permission(self, request, obj=None):
        return self.get_permission_value(request, obj)

    def get_permission_value(self, request, obj=None):
        if not request.user.is_superuser:
            if obj and not obj.owner:
                return True
            if obj and obj.owner != request.user:
                return False
        return True
```

4.2.3 Модель даних Company

Реалізація моделі даних Company:

```
class Company(ContactBase):
    class Meta:
        verbose_name_plural = "Companies"

    name = models.CharField(
        verbose_name=_("Name"),
        help_text=_("Company's name"),
        max_length=250
    )
    first_name = None
    last_name = None
    address = models.TextField(blank=True, default="")

    def __str__(self):
        return self.name
```

Реалізація класу CompanyAdmin:

```
class CompanyAdmin(MyModelAdmin):
    fields = ['name', 'description', 'email', 'phone', 'address', 'creation_date', 'owner']
    readonly_fields = ['creation_date']
    list_display = ['name', 'email']
    list_filter = ['owner', 'creation_date']
```

Після реєстрації моделі у панелі адміністратора, користувач може додавати , змінювати, передивлятись історію змін, видаляти, об’єкти класу Company.

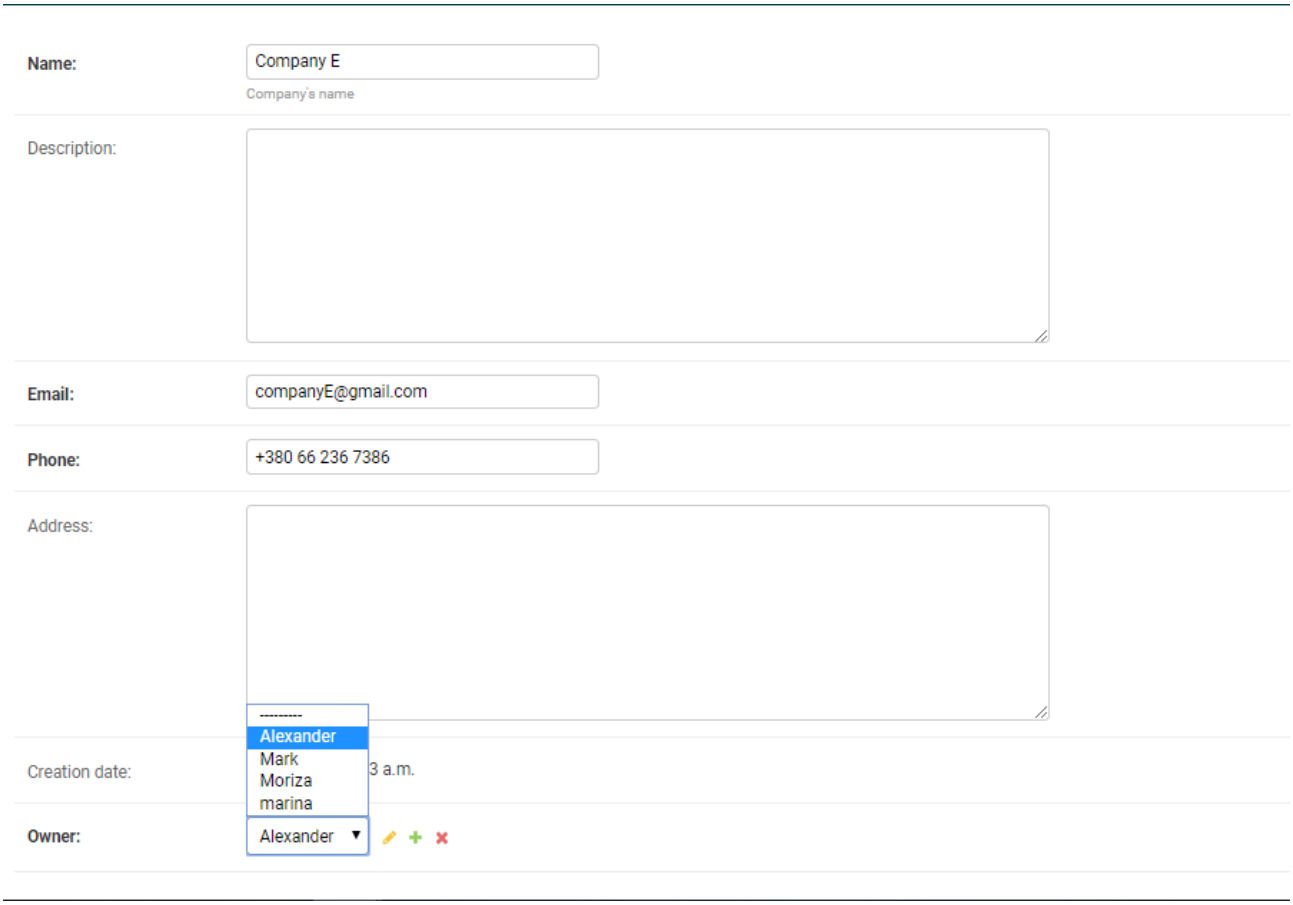


Рис. 4.2.3.1. UML Створення нової компанії

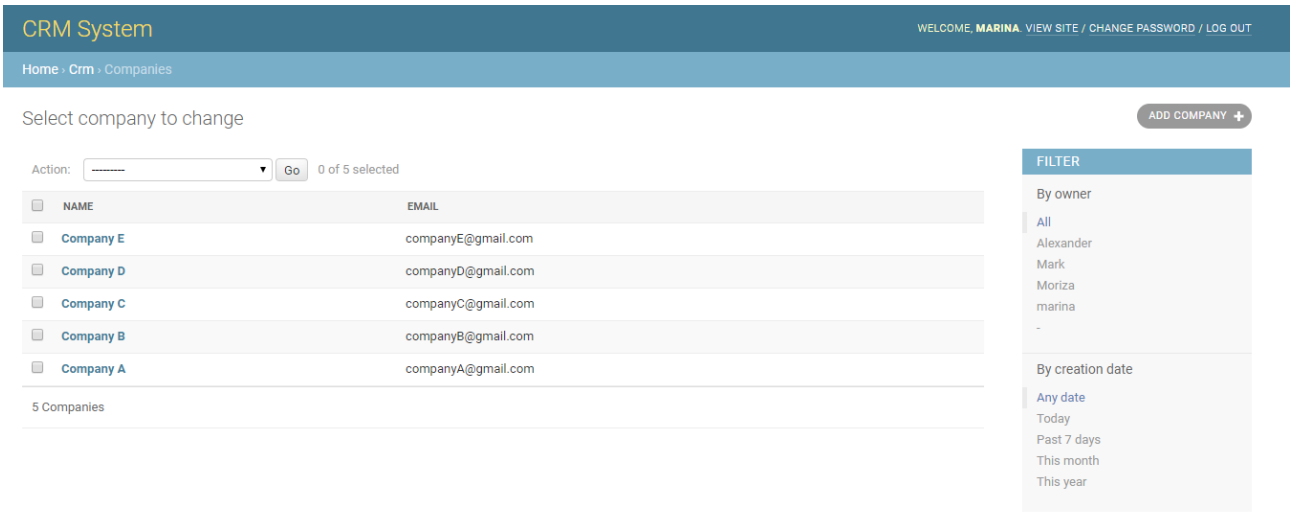


Рис. 4.2.3.2. Список всіх створених компаній

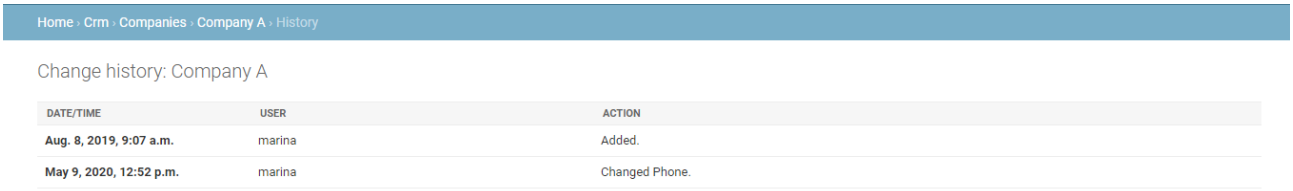


Рис. 4.2.3.3. Перегляд історії змін компанії

4.2.4 Модель даних Contact

Реалізація моделі даних Contact:

```
class Contact(ContactBase):
    company = models.ForeignKey(Company, on_delete=models.CASCADE)
    job_title = models.CharField(blank=True, null=True, max_length=250)
    @property
    def full_name(self):
        return f'{self.first_name} {self.last_name}'

    def __str__(self):
        return f'{self.full_name}, {self.company}'
```

Реалізація класу ContactAdmin:

```
class CompanyAdmin(MyModelAdmin):
    fields = ['name', 'description', 'email', 'phone', 'address', 'creation_date', 'owner']
    readonly_fields = ['creation_date']
    list_display = ['name', 'email']
    list_filter = ['owner', 'creation_date']
```

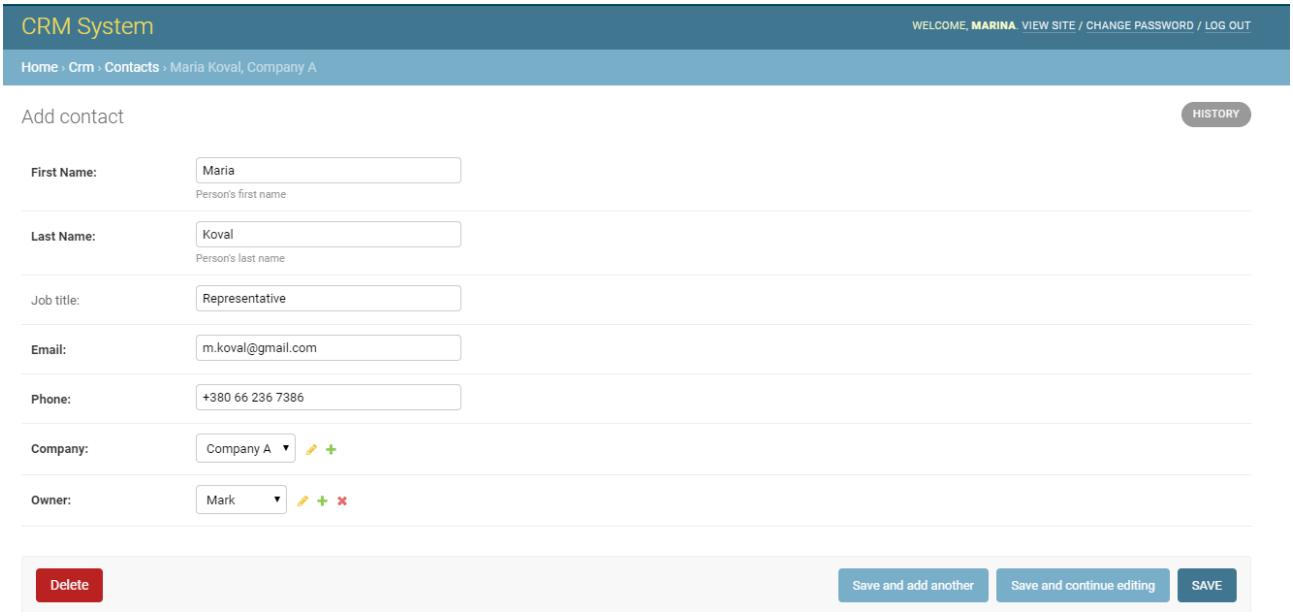


Рис. 4.2.4.1. Створення нового контакту

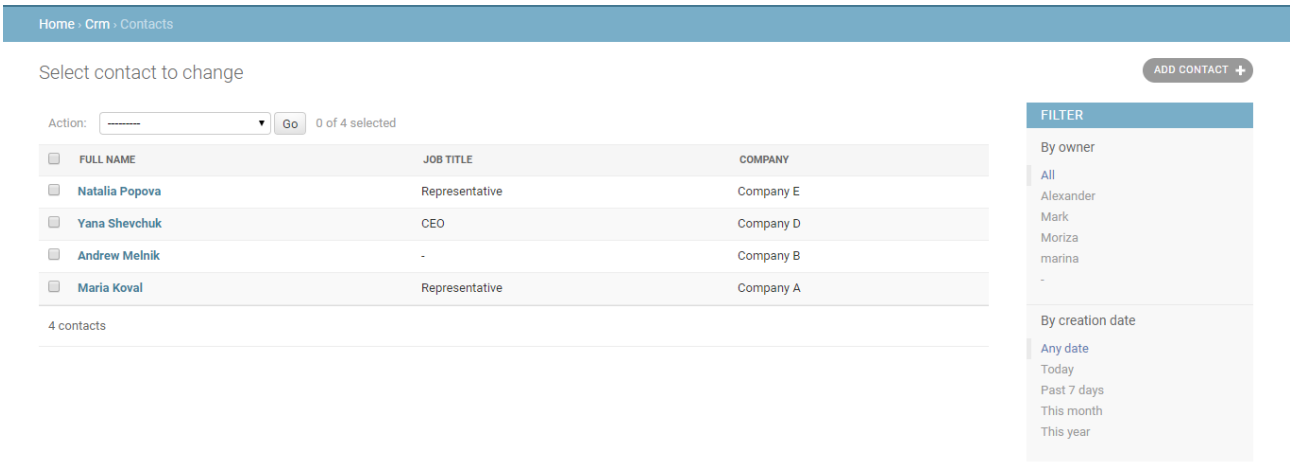


Рис. 4.2.4.2. Перегляд всіх створених контактів

4.2.5 Модель даних Lead

Реалізація моделі даних Lead:

```
class Lead(ContactBase):
    company_name = models.CharField(
        verbose_name=_("Company's name"),
        help_text=_("Company's name"),
        max_length=250,
        blank=True,
        default=""
    )
    job_title = models.CharField(blank=True, null=True, max_length=250)
    company_email = models.EmailField(blank=True, default="")
    company_phone = models.CharField(blank=True, default="", max_length=250)
    address = models.TextField(blank=True, default="")

    @property
    def full_name(self):
        return f'{self.first_name} {self.last_name}'

    def __str__(self):
        return self.full_name
```

Реалізація класу LeadAdmin:

```
class LeadAdmin(MyModelAdmin):
    fields = [
        'first_name', 'last_name', 'job_title', 'email', 'phone', 'company_name',
        'company_email', 'company_phone', 'address', 'owner'
    ]
    list_display = ['full_name', 'email', 'company_name', 'company_email', 'owner']
    list_filter = ['owner', 'creation_date']
```

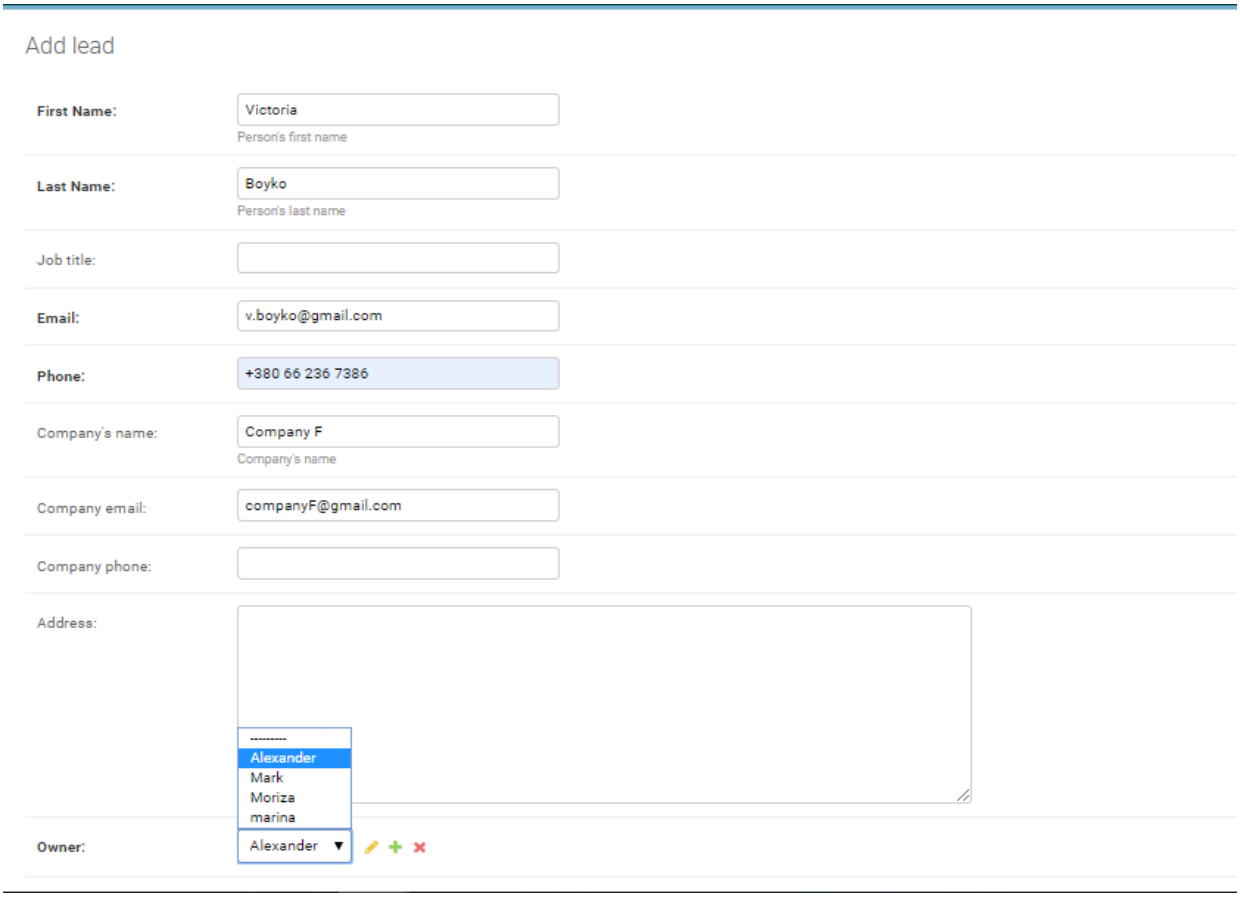


Рис. 4.2.5.1. Створення нового “lead”

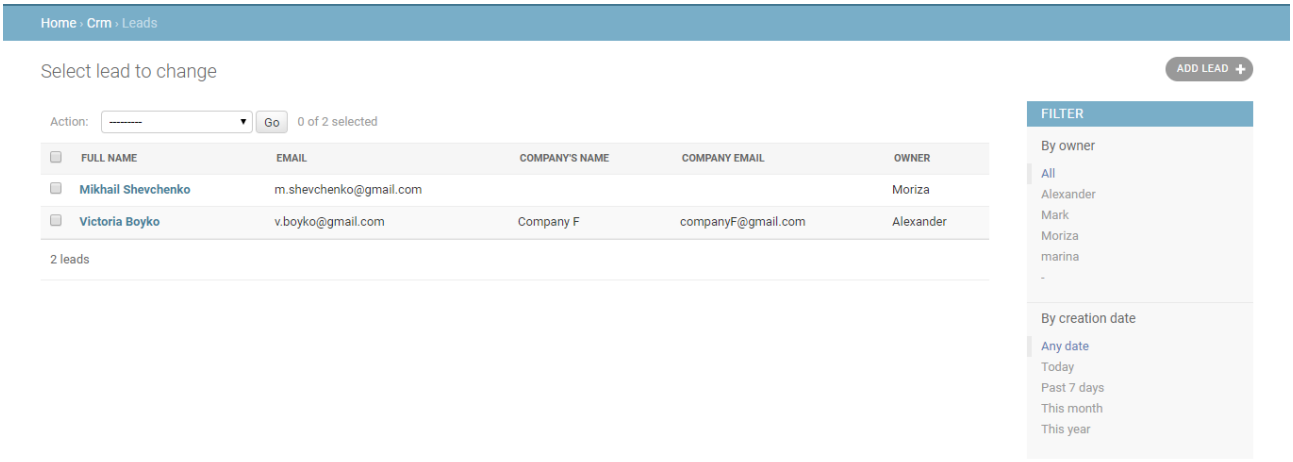


Рис. 4.2.5.2. Перегляд всіх існуючих “leads”

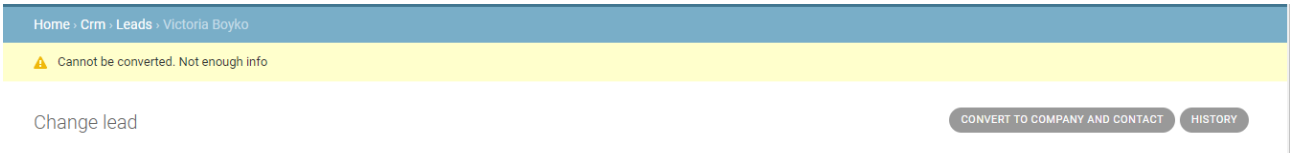


Рис. 4.2.5.3. Невдала при конвертації “lead” у компанію

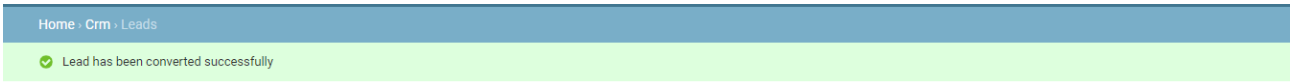


Рис. 4.2.5.4. Вдала при конвертації “lead” у компанію

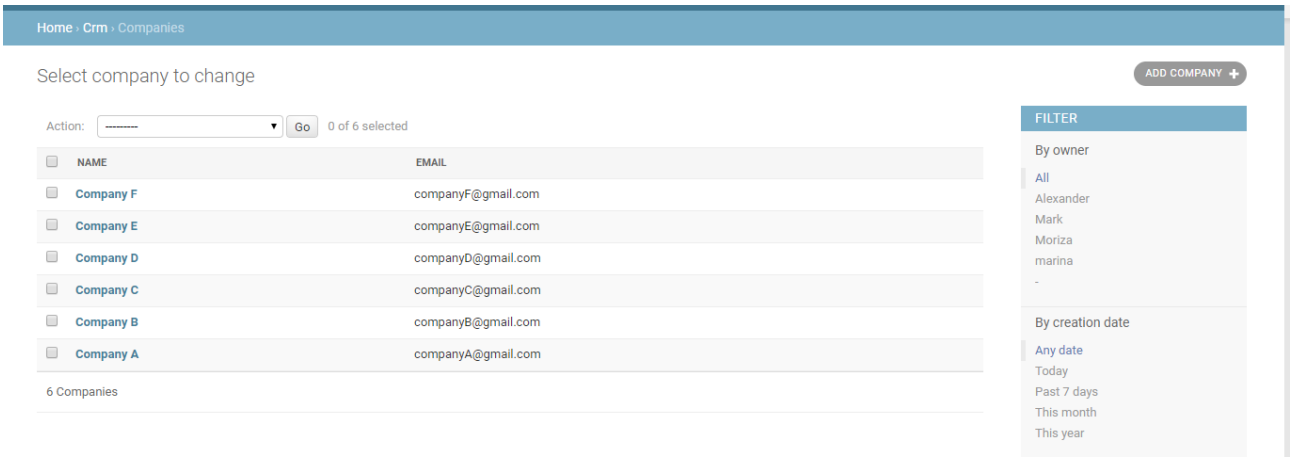


Рис. 4.2.5.4. Приєднання конвертованого “lead” до списку компаній

4.2.6 Модель даних Stock

Реалізація моделі даних Stock:

```
class Stock(models.Model):
    name = models.CharField(max_length=250)

    def __str__(self):
        return self.name
```

Реалізація класу StockAdmin:

```
class StockAdmin(MyModelAdmin):
    fields = ['name']
    list_display = ['name']
```

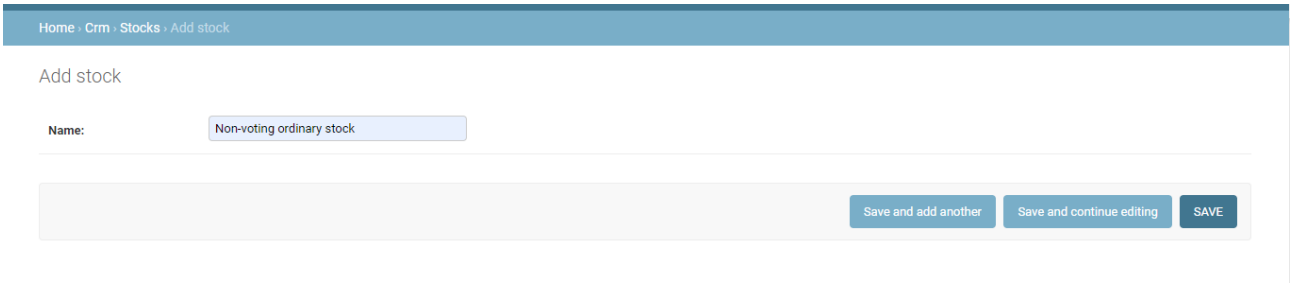


Рис. 4.2.6.2. Створення нового типу акцій

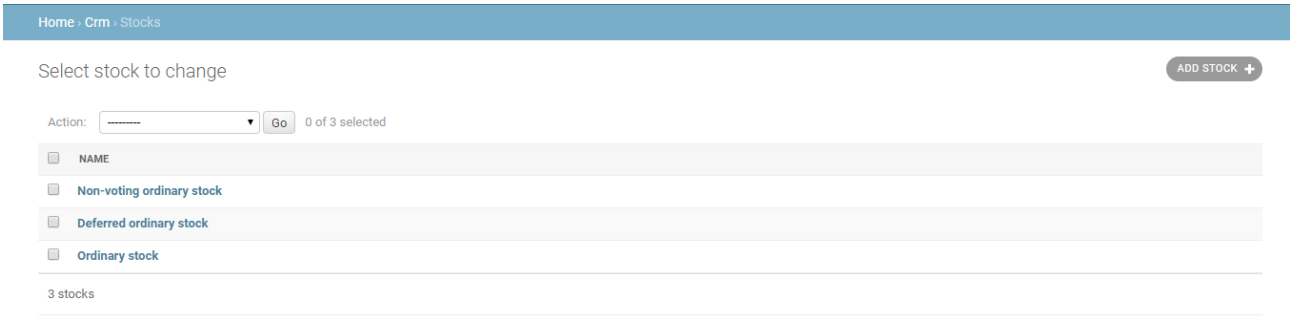


Рис. 4.2.6.2. Список всіх існуючих типів акцій

4.2.7 Модель даних Case

Реалізація моделі даних Case:

```
class Case(Base):
    name = models.CharField(
        verbose_name=_("Name"),
        max_length=250
    )
    contact = models.ForeignKey(Contact, on_delete=models.CASCADE, blank=True, null=True)
    lead = models.ForeignKey(Lead, on_delete=models.CASCADE, blank=True, null=True)
    is_active = models.BooleanField(default=True)
    is_successful = models.BooleanField(default=False)
    stock = models.ForeignKey('Stock', on_delete=models.SET_NULL, blank=True, null=True)
    deal_type = models.CharField(
        max_length=1,
        choices=DEAL_TYPE_CHOICES,
        default=''
    )
    stock_price = models.DecimalField(max_digits=10, decimal_places=2, default=0, blank=True,
null=True)
    amount = models.FloatField(default=0.0, blank=True, null=True)

    def __str__(self):
        return self.name
```

Визначення типів угод:

```
DEAL_TYPE_CHOICES = [
    ('p', 'purchasing'),
    ('s', 'selling'),
]
```

Реалізація класу CaseAdmin:

```
class CaseAdmin(MyModelAdmin):
    readonly_fields = ['creation_date', 'price']
    list_display = ['name', 'deal_type', 'is_active', 'is_successful', 'creation_date', 'owner']
    list_filter = ['owner', 'deal_type', 'is_active', 'is_successful',  'creation_date', 'owner']
```

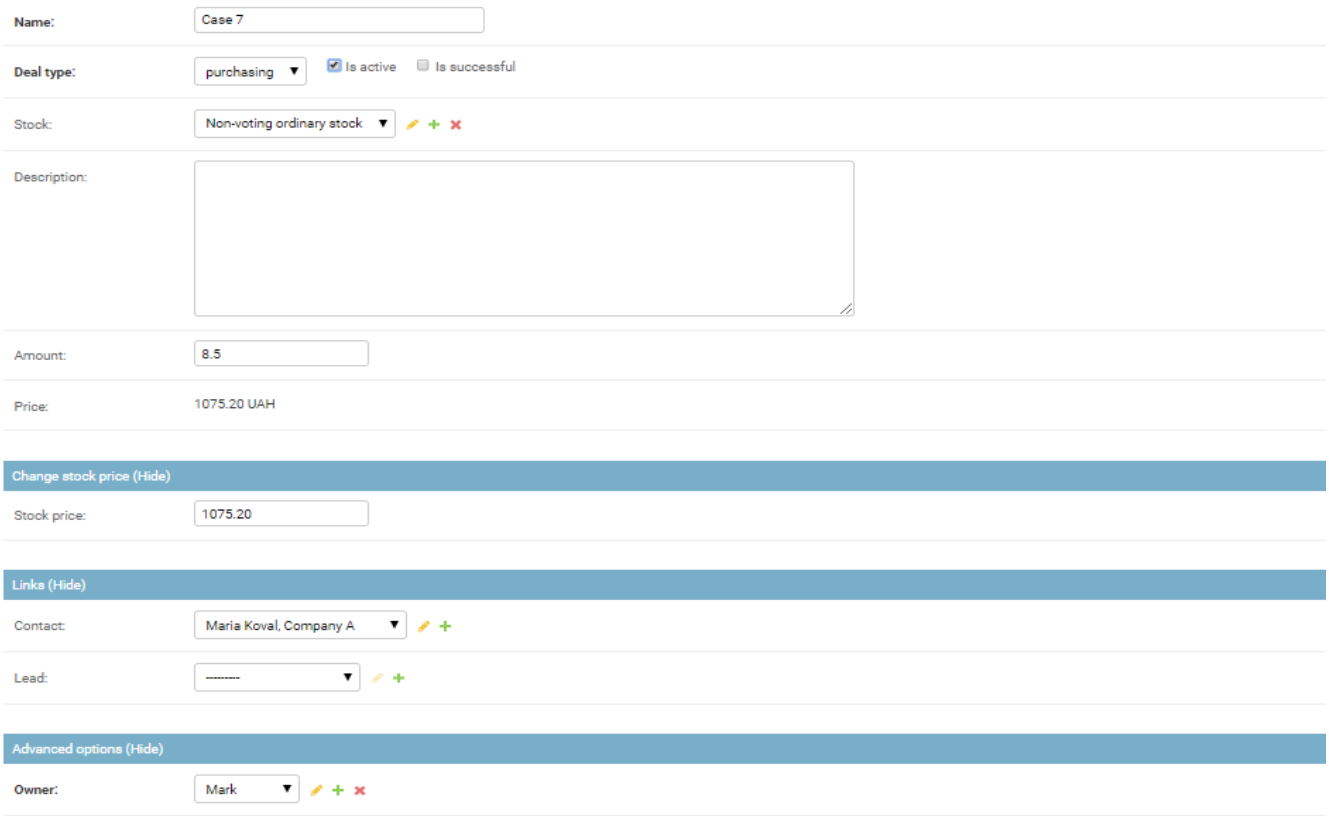


Рис. 4.2.7.1. Створення нової справи

4.2.8 Модель даних CustomEmail

Реалізація моделі даних CustomEmail:

```
class CustomEmail(Base):
    class Meta:
        verbose_name = "Email"
    sender = models.CharField(max_length=250, verbose_name="From")
    receiver = models.CharField(max_length=250, verbose_name="To")
    subject = models.CharField(max_length=250, verbose_name="Subject")
    content = models.TextField()
    description = None
    is_sent = models.BooleanField(default=False)

    def __str__(self):
        return self.subject
```

Реалізація класу CustomEmailAdmin:

```
class EmailAdmin(MyModelAdmin):
    fields = ['sender', 'receiver', 'subject', 'content', 'creation_date', 'owner']
    list_display = ['subject', 'sender', 'receiver', 'creation_date', 'owner', 'is_sent']
    list_filter = ['owner', 'is_sent', 'creation_date']
    readonly_fields = ['creation_date']
```

Зразок конфігурації SMTP сесії в проекті. Данні налаштування вказані у файлі crm_system/settings.py.

```
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_PASSWORD = 'CRMadmin5'
EMAIL_HOST_USER = 'crmcoursework@gmail.com'
EMAIL_PORT = 587
EMAIL_SUBJECT_PREFIX = 'CRM: '
EMAIL_USE_TLS = True
SERVER_EMAIL = 'crmcoursework@gmail.com'
```

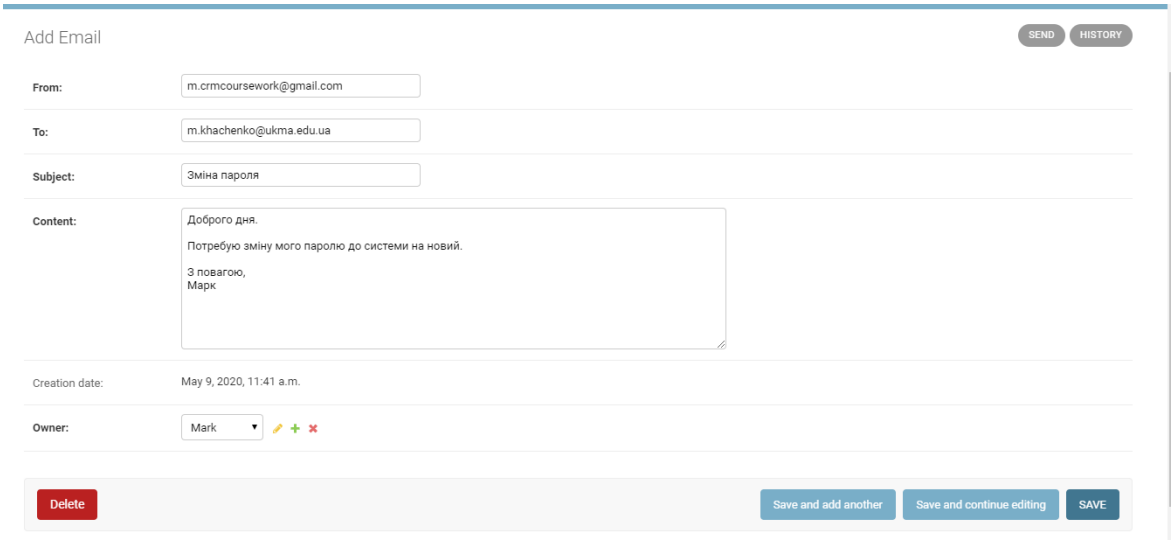


Рис. 4.2.8.1 Створення нового листа

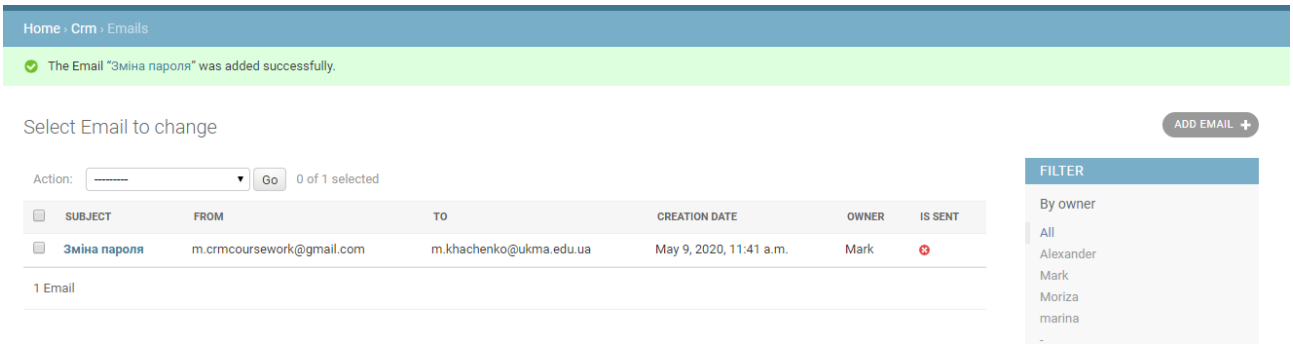


Рис. 4.2.8.2 Лист знаходиться у стані чернетки

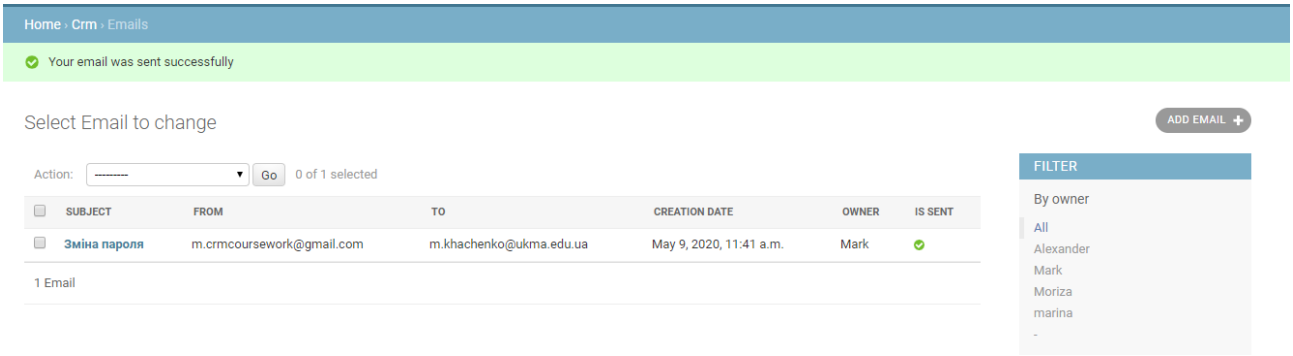


Рис. 4.2.8.2 Лист було надіслано

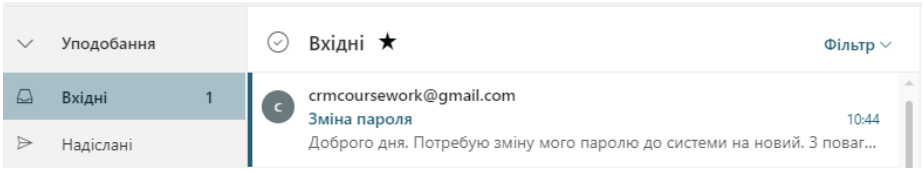


Рис. 4.2.8.2 Лист було отримано

4.2.9 Групи користувачів та дозволи

Система генерує стандартні права доступу до створених моделей даних, що за бажанням можна перевизначити, як було зроблено у підрозділі 4.2.1.

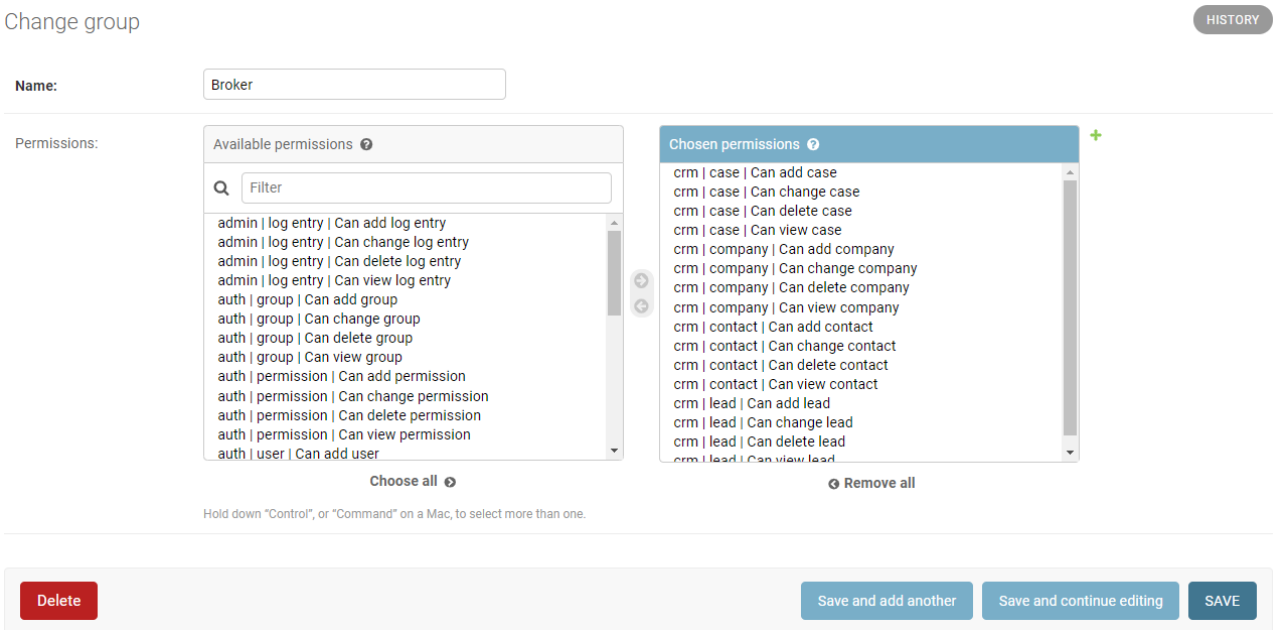


Рис. 4.2.2.1. Визначення прав для групи користувачів

4.2.10 Створення користувача

Нижче представлений процес створення нового користувача.

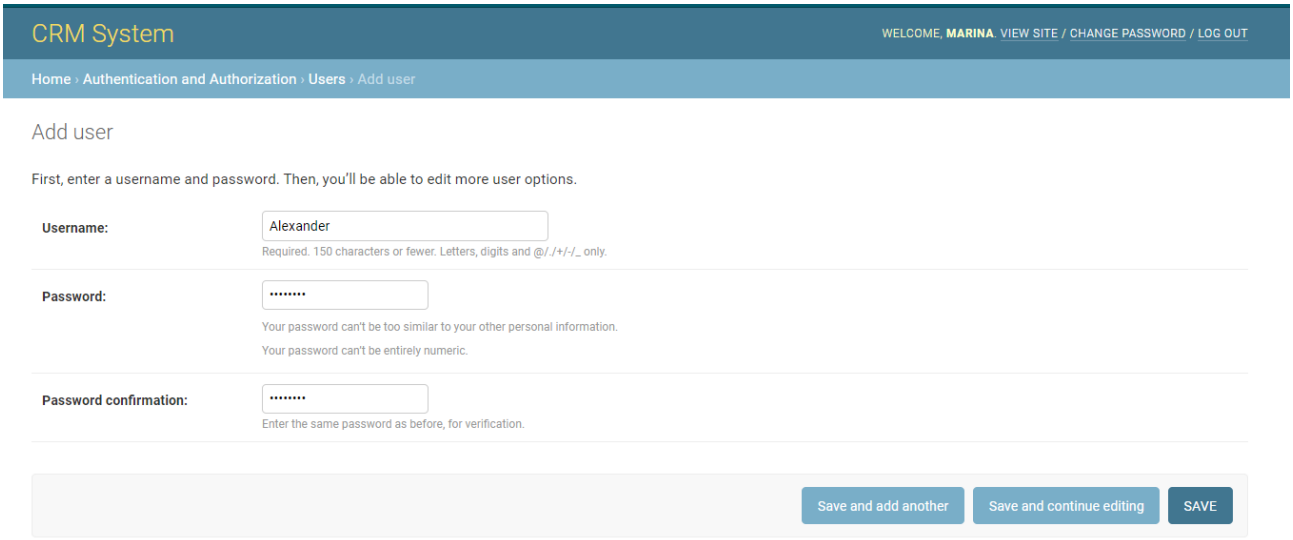


Рис. 4.2.3.1. Створення нового користувача

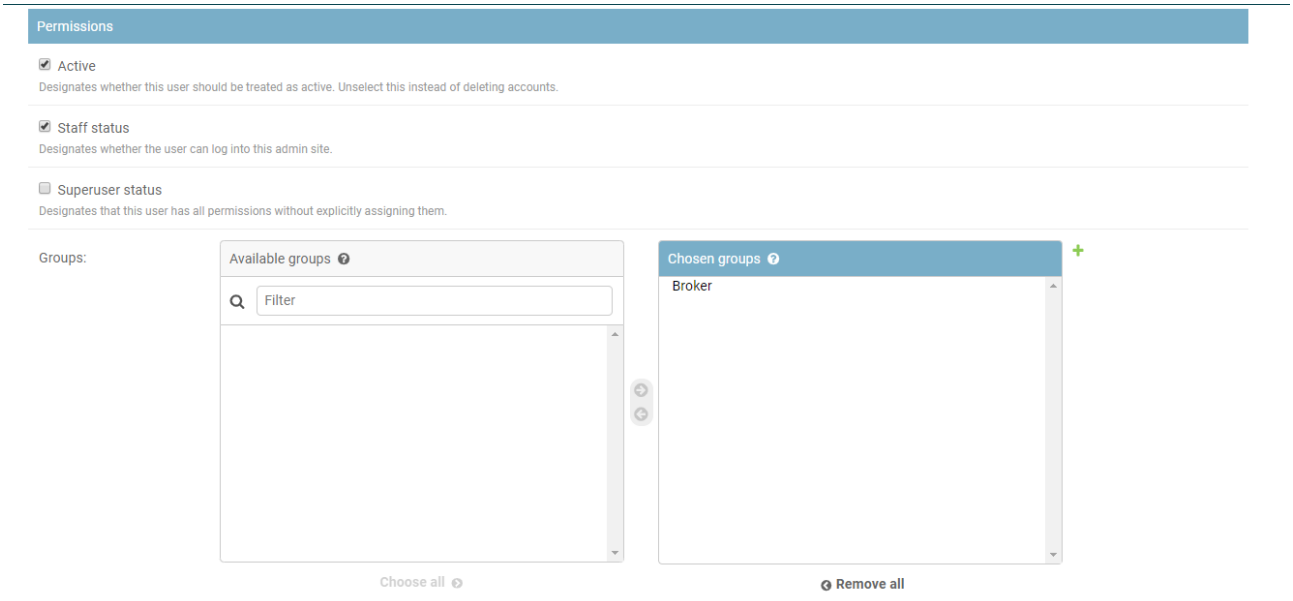


Рис. 4.2.10.2. Додавання нового користувача до існуючої групи

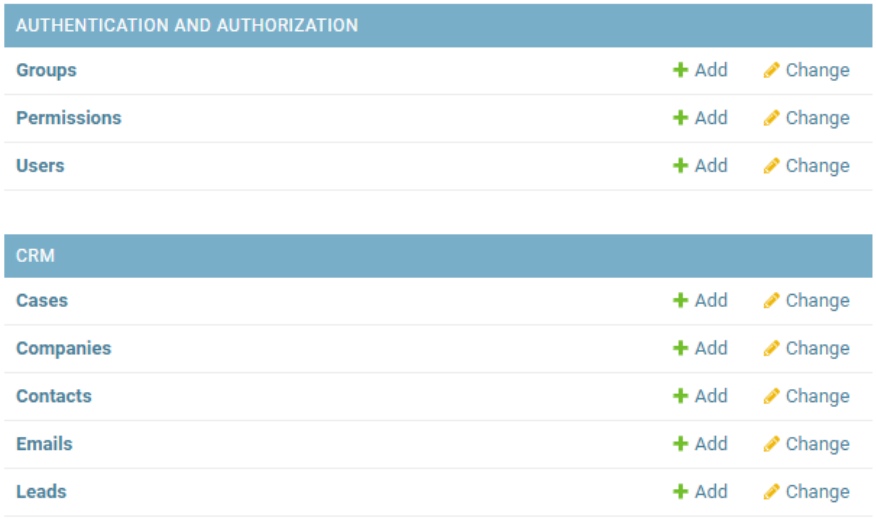


Рис. 4.2.10.3. Загальний вигляд панелі адміністратора

4.3 Застосунок analytics

Застосунок містить логіку підрахунку статистичних даних за останній рік.

У файлі analytics/mymodeladmin.py було створено клас CaseStatAdmin, що відповідає за відображення даних з панелі адміністратора.

Реалізація класу MyModelAdmin:

```
class MyModelAdmin(admin.MyModelAdmin):

    def has_add_permission(self, request, obj=None):
        return False

    def changelist_view(self, request, extra_context=None):
        response = super().changelist_view(
            request,
            extra_context=extra_context,
        )
        if response.status_code == 302:
            return response
        response.context_data['charts'] = list()
        try:
            queryset = response.context_data['cl'].queryset
        except (AttributeError, KeyError):
            return response
```

```
        self.today = timezone.now().date()
        self.year_ago_date = self.today + relativedelta(months=-12, days=+1)
        response = self.create_context_data(request, response, queryset)
        return response

    def create_context_data(self, request, response, queryset):
        """Should be implemented in child class."""
        return response

    def check_time_periods(self, queryset):
        item_list = list(queryset)
        date = self.today.replace(day=1) + relativedelta(months=-11)
        if item_list and item_list[0]['period'] < date:
            item_list.pop(0)
        for i in range(0, 12):
            item = next((
                item for item in item_list if item["period"] == date
            ), None)
            if not item:
                item_list.insert(i, {'period': date, 'total': 0})
            date = date + relativedelta(months=1)
        return item_list

    @staticmethod
    def add_chart_data(response, title, param, max_value):
        aa = {
            'title': title,
            'data': ({
                'period': x['period'],
                'total': x['total'] or 0,
                'pct':
                    (int(round((x['total'] or 0)) / max_value * 100)) or 2
                    if max_value else 2,
            } for x in param)
        }
        response.context_data['charts'].append(aa)

    @staticmethod
    def get_values_over_time(queryset, field):
        values = queryset.annotate(
            period=Trunc(field, 'month')
        ).values('period').annotate(
            total=Count('id')
        ).order_by('period')
        return check_time_periods(values), get_maximum(values)


def check_time_periods(queryset):
    item_list = list(queryset)
    today = timezone.now().date()
    date = today.replace(day=1) + relativedelta(months=-11)
    if item_list and item_list[0]['period'].date() < date:
        item_list.pop(0)
    for i in range(0, 12):
        item = next((
            item for item in item_list if item["period"] == date
        ), None)
        if not item:
            item_list.insert(i, {'period': date, 'total': 0})
        date = date + relativedelta(months=1)
    return item_list
```

У файлі analytics/admin.py було створено клас CaseStatAdmin, що відповідає за відображення даних з панелі адміністратора.

Реалізація класу CaseStatAdmin:

```
class CaseStatAdmin(MyModelAdmin):
    change_list_template = 'admin/analytics/case_stat/case_summary_change_list.html'
    list_filter = ['owner']

    def create_context_data(self, request, response, queryset):
        total_qs = queryset.filter(creation_date__gte=self.year_ago_date)
        successful_qs = total_qs.filter(is_successful=True)
        total_cases = total_qs.count()
        successful_cases = successful_qs.count()

        k = round(successful_cases * 100 / total_cases, 2) if total_cases else 0

        response.context_data['summary'] = {
            _('Total cases'): total_cases,
            _('Successful cases'): successful_cases,
            _('Success Conversion'): f'{k} %',
        }

        title = _('Total cases')
        summary_over_time_total, max_value = MyModelAdmin.get_values_over_time(
            total_qs,
            'creation_date'
        )
        self.add_chart_data(
            response, title, summary_over_time_total, max_value
        )
        title = _('Successful cases')
        summary_over_time_successful, maximum = MyModelAdmin.get_values_over_time(
            successful_qs,
            'creation_date'
        )
        self.add_chart_data(
            response, title, summary_over_time_successful, max_value
        )
        return response
```

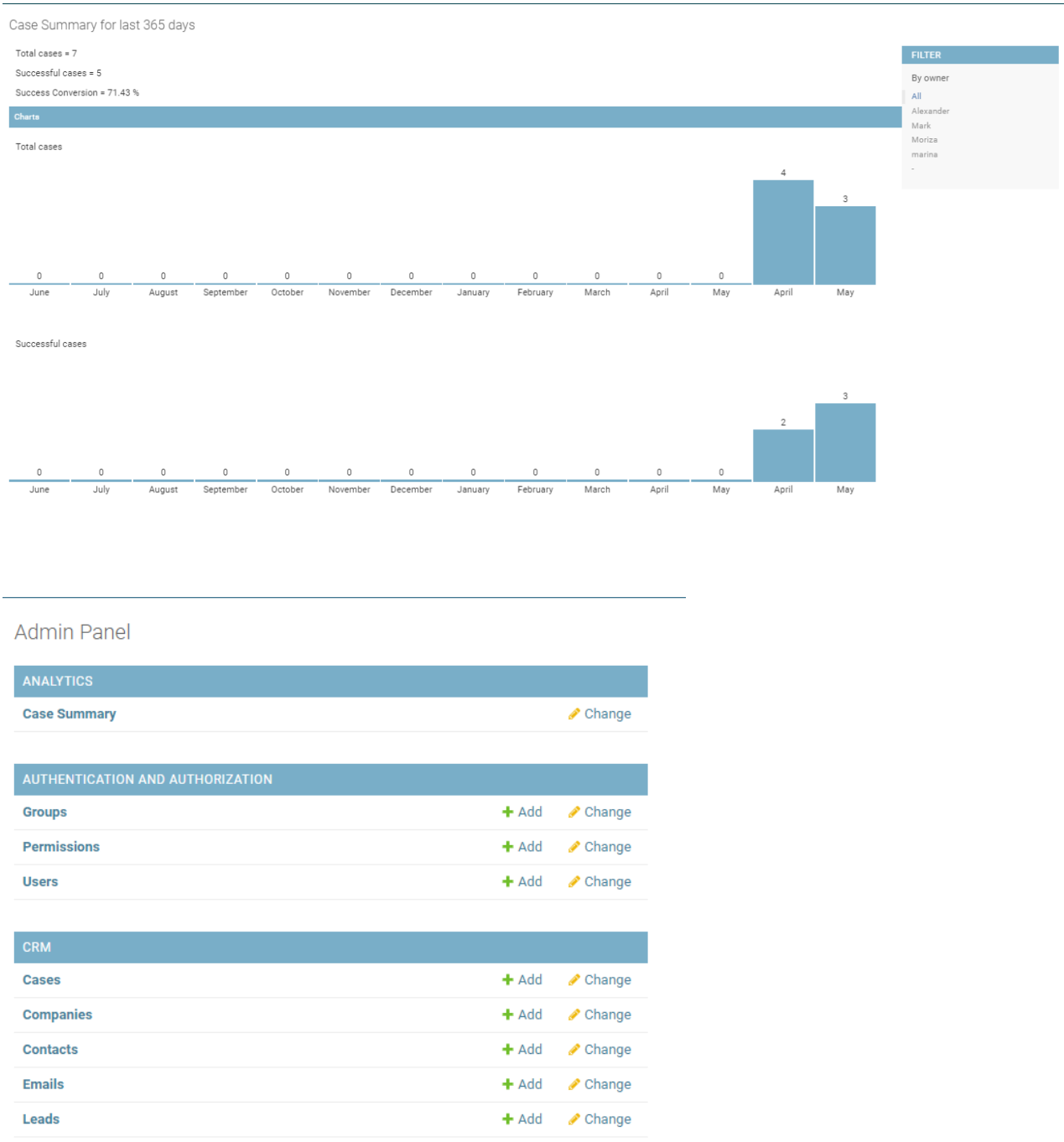


Рис. 4.3.1. Загальний вигляд панелі адміністратора

Висновки

Під час виконання цієї роботи були досліджені особливості роботи, проектування та розробки систем управління відносинами з клієнтами (CRM-системи).

В процесі дослідження представлених на ринку CRM-систем, було відмічено великий рівень складності таких систем, та великий рівень ознайомленості з обраною CRM-системою, який повинен мати розробник, щоб успішно створювати нові системи на основі вже існуючого рішення. Було розглянуто питання чому деякі компанії можуть віддати перевагу створенню CRM-системи з самого початку.

Крім того, під час реалізації власної системи, був розглянутий процес роботи з Веб-фреймворком Django та детально описано своє рішення. Технологія була обрана вдало. Вона добре підходить для створення рішень в області систем управління відносинами з клієнтами.

Список використаної літератури

1. Документація веб-фреймворку Django [Електронний ресурс] - Режим доступу до ресурсу: https://docs.djangoproject.com/en/3.0/
2. Документація BPML нотації [Електронний ресурс] - Режим доступу до ресурсу: http://www.bpmn.org/