МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.
«_____»_____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
" МАГІСТР "

Тема: Система керування взаємовідносинами з клієнтами засобами фреймворку Django

Виконавець: Лузанов Є. О.

Керівник: к.т.н., доцент Росінська Г. П.

Нормоконтролер: : к.т.н., доцент Росінська Г. П. .

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп’ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Литвиненко О.Є.
«_____»_____ 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи (проєкту)
Лузанова Євгена Олександровича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема кваліфікаційної роботи (проєкту) Система керування взаємовідносинами з клієнтами засобами фреймворку Django
затверджена наказом ректора від « 16 » вересня 2022 р. № 1530/ст.

2. Термін виконання роботи (проєкту): з 05.09.2022 по 30.11.2022

3. Вихідні дані до роботи (проєкту): програмний продукт розробляється на мові програмування *Python* та фреймоврку *Django*

4. Зміст пояснювальної записки:

1) Аналіз предметної області;

2) Проектування структури системи керування взаємовідносинами з клієнтами;

3) Реалізація серверної частини додатку.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) зображення панелі списку усіх угод та лідов;

2) форма створення задачі;

3) панель календарю з задачами;

4) панель карточки працівника;

5) панель центру комунікацій.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомитись з постановкою задачі	16.09.2022	
2	Ознайомитись з літературою за темою роботи	17.09.2022-18.09.2022	
3	Аналіз мови програмування та її бібліотек	19.09.2022-20.09.2022	
4	Написати розділ 1	21.09.2022-31.09.2022	
5	Проектування структури системи	01.10.2022-21.10.2022	
6	Написати розділ 2.	22.10.2022-28.10.2022	
7	Розробка серверного додатку	29.10.2022-01.11.2022	
8	Написати розділ 3.	02.11.2022-05.11.2022	
9	Оформити пояснювальну записку	07.11.2022	
10	Підготувати графічний демонстраційний матеріал та доповідь	12.11.2022	

7. Дата видачі завдання: “ 16 " вересня 2022 р.

Керівник кваліфікаційної роботи (проєкту)_____ Росінська Г.П.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання_____ Лузанов Є. О.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційного проєкту “Система керування взаємовідносинами з клієнтами засобами фреймворку *Django* ”: 92 с., 72 рис., 25 літературних джерел, 1 додаток.

БАЗА ДАНИХ, *SQL, SQLITE, CRM,* СУБД, ВЕБ-ДОДАТОК, *DJANGO*, *DJANGO-REST-FRAMEWORK*.

Об’єкт дослідження – серверна частина клієнту веб-додатку.

Предмет дослідження – система керування взаємовідносинами з клієнтами засобами *Django*.

Мета кваліфікаційної роботи – розробити серверну частину веб-додатку системи керування взаємовідносинами з клієнтами засобами *Django* .

Методи дослідження – застосування фреймворку *Django* для створення серверної частини веб-додатку.

Прогнозні припущення щодо розвитку об’єкта дослідження – розробка працюючої програми та використання її для керування взаємовідносинами з клієнтами.

Результати кваліфікаційної роботи рекомендується використовувати при керуванні взаємовідносинами з клієнтами.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПЗ – програмне забезпечення

СУБД – система управління базами даних

БД – база даних

ООП – об'єктно-орієнтоване програмування

CRM – Customer Relationship Management (управління взаємовідносинами з клієнтами)

SQL – Structured Query Language (структурована мова запитів)

URL – Universal Resource Locator (уніфікований локатор ресурсу)

ЄДРПОУ – Єдиний державний реєстр підприємств та організацій України

MVT – Model View Template(патерн програмування модель, представлення, шаблон)

АТС – автоматична телефона станція

ВСТУП

На сьогоднішній компанії займаються оцифруванням документів для покращення продуктивності усіх процесів всередині та зовні компанії. У процесі, кожен для себе виявив важливі причини для оцифрування:

– Підвищення продуктивності. Цифрові дані добре доступні для пошуку, швидко доступні та легко передаються, що сприяє кращим і швидшим робочим процесам. Залежно від того, на чому зосереджена компанія, стає можливим збільшити обсяг виробництва за допомогою тих самих ресурсів. Наприклад, паперовий документ, який можна знайти в середньому за 12 хвилин, тепер можна отримати за секунди , обробити та доставити протягом 1/10 нецифрового часу. Таким чином оцифрування документів, дозволяє досягти 10-кратного підвищення продуктивності, без скорочення перерви.

– Оптимізація діяльності. Завдяки сучасним технологіям кожен може мати доступ і співпрацювати в реальному часі. Тому, це не лише наявність цифрової версії документа замість фотокопіювання. Основні переваги спрощених оцифрованих операцій походять від миттєвого та практично універсального доступу до інформації, швидкого аналізу, управління проектами та співпраці, відстеження часу, не використовуючи жодного аркуша паперу.

– Зниження витрат. Темною стороною «підвищення продуктивності» є «зниження витрат». Так, можливо зробити більший обсяг роботи з тією ж кількістю ресурсів, а також може знадобитися досягти тих самих результатів з меншими ресурсами. Експертна сторона компанії може виявитися екзистенційно важливою. Оцифрування може допомогти відмовитися від обробки, оновлення та зберігання паперових документів із потенційно значним скороченням витрат на персонал, зберігання та транспортування.

– Підвищення ефективності. Оптимізовані операції з вищою продуктивністю та меншими витратами призведуть до більш гладкої та ефективної роботи компанії. Хоча оцифрування само по собі може призвести до

підвищення ефективності, також можливо використовувати процес оцифрування як унікальний шанс проаналізувати та підвищити ефективність різних функцій, наприклад виробництво та логістика, запити та підтримка клієнтів, продажі та маркетинг тощо.

– Посилена безпека. Цифровий документ можна відстежувати, і контролювати, хто має до нього доступ. Для збереження конфіденційності можна встановити робочі процеси та індивідуальні дозволи. Можна встановити кілька копій, для впевненості, що документи ніколи не будуть втрачені. Та навпаки, інформація, що зберігається в паперових форматах, піддається розкладанню, або через природні причини, або через нещасний випадок.

Одним з найкращих способів оцифрування бізнесу є використання систем керування взаємовідносинами з клієнтами або *Customer Relationship Management(CRM).*

За своєю суттю, управління взаємовідносинами з клієнтами (*CRM*) — це всі дії, стратегії та технології, які компанії використовують для керування взаємодією зі своїми поточними та потенційними клієнтами. *CRM* допомагає компаніям будувати відносини зі своїми клієнтами, які, у свою чергу, створює лояльність і утримання клієнтів. Оскільки лояльність клієнтів і дохід є якостями, які впливають на дохід компанії, *CRM* — це стратегія управління, яка призводить до збільшення прибутку для бізнесу. У своїй основі, інструмент *CRM* створює простий користувальницький інтерфейс для збору даних, який допомагає компаніям розпізнавати клієнтів і спілкуватися з ними масштабованим способом.

Аналітична *CRM* стосується технологій, які агрегують інформацію про клієнтів і забезпечують аналіз даних про клієнтів для покращення прийняття управлінських рішень і дій. Він заснований на таких технологіях, як сховища даних і інтелектуальний аналіз даних. В ідеалі база даних клієнтів повинна бути доступна з усіх відповідних відділів, таких як продажі, обслуговування клієнтів і маркетинг. Аналітичний *CRM* формує основу для планування та оцінки маркетингових кампаній і допомагає функціям крос-продажів і додаткових

продажів. Щоб успішно запровадити *CRM*, фірми повинні поєднувати фізичні ресурси та інформаційні ресурси.

За своєю суттю управління відносинами з клієнтами просте. Однак його можна реалізувати різними способами: веб-сайти, соціальні медіа, телефонні дзвінки, чат, пошта, електронна пошта та різноманітні маркетингові матеріали — все це можна інтегрувати в рішення *CRM*.

Завдяки різноманітності *CRM*, вона приносить користь не лише великим підприємствам – використання та підтримка інструменту *CRM* є основою для масштабованої системи продажів і маркетингу. Будь-яка компанія виграє від ведення запису про те, які розмови, покупки та маркетингові матеріали можуть бути пов’язані з потенційними клієнтами та клієнтами. Одним із плюсів використання *CRM*, є можливість підтримки малих підприємств та фрілансерів.

Основні способи, завдяки яким, *CRM* дозволяє досягти поставленої задачі:

– Навчання. *CRM* допомагає компаніям дізнаватися про своїх клієнтів, зокрема про те, хто вони і чому купують ваші продукти, а також про тенденції в історії покупок клієнтів. Це дозволяє підприємствам краще передбачати потреби своїх клієнтів і, як наслідок, задовольняти їх. Ефективне використання управління відносинами з клієнтами також може забезпечити стратегічну перевагу. Добре організовані дані клієнтів допомагають компаніям вибирати правильних одержувачів для рекламних акцій і нових продуктів.

– Організація. *CRM* дозволяє підприємствам підвищити ефективність шляхом організації та автоматизації певних аспектів бізнесу. Від процесів продажів до маркетингових кампаній і бізнес-аналітики, а також даних про клієнтів, *CRM* автоматизує та оптимізує ці процеси для компаній. Це дозволяє підприємствам простіше організувати ці процеси, легше зрозуміти дані.

– Оптимізація. Програмне забезпечення *CRM* дозволяє підприємствам оптимізувати взаємодію з клієнтами. Спрощуючи та оптимізуючи багато складніших процесів взаємодії з клієнтами, *CRM* підвищує задоволеність клієнтів.

Значною перевагою може стати розвиток кращих відносин з існуючими клієнтами, що може призвести до:

– збільшення продажів за рахунок кращого часу завдяки передбаченню потреб на основі історичних тенденцій;

– більш ефективне визначення потреб шляхом розуміння конкретних вимог клієнтів;

– збільшення продажів інших продуктів шляхом виділення та пропозиції альтернатив або покращень;

– визначення того, хто з клієнтів є прибутковим, а хто ні.

Це може привести до кращого маркетингу продуктів або послуг, зосередившись на:

– ефективності цільових маркетингових комунікаціях, спрямованих конкретно на потреби клієнтів;

– більш особистий підхід і розробка нових або вдосконалених продуктів і послуг для залучення більшого бізнесу в майбутньому.

Зрештою це може призвести до:

– підвищення задоволеності та утримання клієнтів, забезпечуючи подальше зростання доброї репутації на ринку;

– збільшення цінності від існуючих клієнтів і зниження витрат, пов’язаних з їх підтримкою та обслуговуванням, підвищення загальної ефективності та зниження загальної вартості продажів;

– підвищення прибутковості завдяки зосередженню на найбільш прибуткових клієнтах і роботі з збитками більш економічно ефективними способами;

Коли компанія почне ефективно піклуватися про своїх існуючих клієнтів, зусилля можна зосередити на пошуку нових клієнтів і розширенні ринку. Але, знову ж таки, чим більше інформації про існуючих клієнтів, тим легше виявити нових потенційних клієнтів і збільшити клієнтську базу.

Успішне впровадження CRM можна визначити як те, що відбувається, коли система CRM допомагає компанії вигідно надавати ринкові пропозиції

клієнтам, які забезпечують цінність для клієнтів — можливо, за нижчу ціну, забезпечують більшу цінність за ту саму відносну вартість, або забезпечують більшу цінність за нижчу вартість.

Об’єкт дослідження – серверна частина клієнту веб-додатку.

Предмет дослідження – система керування взаємовідносинами з клієнтами засобами *Django*.

Мета кваліфікаційної роботи – розробити серверну частину веб-додатку системи керування взаємовідносинами з клієнтами засобами.

Відповідно до даної мети необхідно вирішити наступні завдання:

- Спроектувати повноцінну базу даних додатку.
- Розробити серверний додаток на мові *Python*.
- Зв’язати роботу бази даних із серверним додатком.
- Впровадити сповіщення за електронною поштою.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1.Огляд існуючих розробок

1.1.1. Zoho

Zoho Corporation — це індійська багатонаціональна технологічна компанія, яка створює веб-інструменти для бізнесу. Він найбільш відомий завдяки офісному онлайн-пакету *Zoho Office Suite*.

Компанію було засновано в 1996 році Шрідхаром Вембу та Тоні Томасом і представлено в семи місцях із глобальною штаб-квартирою в Ченнаї, Таміл Наду, Індія, і корпоративна штаб-квартира за межами Остіна в Дель Валле, Техас.

З 1996 по 2009 рік компанія була відома як *AdventNet* і спочатку постачала програмне забезпечення для керування мережею.

У 2001 році *AdventNet* розширила свою діяльність у Японії та зосередилася на малому та середньому бізнесі.

Zoho CRM був випущений у 2005 році разом із *Zoho Writer*, першим продуктом офісного пакету компанії. Проекти *Zoho, Creator, Sheet* і *Show* були випущені в 2006 році.

Zoho розширила простір для співпраці з випуском *Zoho Docs* і *Zoho Meeting* у 2007 році.

У 2008 році компанія додала програми для виставлення рахунків і пошти, охопивши мільйон користувачів до серпня того ж року.

У 2009 році компанія була перейменована в *Zoho Corporation* на честь її онлайн-офісного пакету. Компанія залишається приватною власністю. У 2017 році *Zoho* запустила *Zoho One*, повний набір із понад сорока програм.

Станом на жовтень 2021 року *Zoho One* було розширено до 50 програм.

У січні 2020 року Zoho охопив понад 50 мільйонів клієнтів. У липні 2022 року компанія оголосила, що має понад 80 мільйонів користувачів.

Zoho призначений для програмного забезпечення для бізнесу та продуктивності. *Zoho* — величезна компанія, яка в основному розташована в Америці, Індії та на Близькому Сході.

Компанія співпрацює із усіма малими та великими підприємствами, щоб підвищити їх ефективність за допомогою програмного забезпечення для інвентаризації, програмного забезпечення для кадрів, програмного забезпечення для бухгалтерського обліку, програмного забезпечення на основі хмарних технологій тощо. Як тільки повідомляється про інфраструктуру свого бізнесу, налаштовується програмне забезпечення відповідно до потреб.

1.1.2. *Bitrix24*

Програмне забезпечення має величезний набор основних інструментів для п’яти ключових сфер бізнесу, включаючи комунікації, завдання та проекти, *CRM*, контакт-центр і конструктор веб-сайтів. Цей комплексний підхід сприяє ефективній та продуктивній співпраці всієї команди, незалежно від його розміру. *Bitrix24* CRM ставить клієнта в центр бізнесу. Він показує всю історію кожного клієнта, від першого контакту до продажу та далі, дозволяючи розвивати та підтримувати стосунки з клієнтом. З *Bitrix24*, можливо отримати повну картину роботи співробітників та коли справа доходить до взаємодії з клієнтами, діяльності з продажу та керування потенційними клієнтами. *CRM* містить багато інструментів, що необхідні для автоматизації маркетингу та продажів, що полегшує ведення бізнесу та підвищує дохід.

CRM пропонує такі ключові функції:

– Автоматизація маркетингу — за допомогою таких інструментів, як *SMS* та маркетинг електронною поштою, можливо розвивати потенційних клієнтів і швидко перевести їх від потенційних клієнтів до угод. Можливо створювати правила та тригери для навігації клієнтів через воронки продажів і між ними.

– Багатоканальний зв'язок — *Bitrix24 CRM* містить усі необхідні інструменти для ефективної комунікації з клієнтами, включаючи електронну пошту, дзвінки, текстові повідомлення, чат, соціальні мережі та месенджери. Цей підхід означає, що завжди є можливість зустрітися з клієнтами там, де вони є, і вони зможуть зв’язатися з компанією найкращим для них способом. Це також дозволяє вам запускати майже будь-який тип маркетингової кампанії, від маркетингу електронною поштою до голосового мовлення, залучаючи якомога більше потенційних клієнтів.

– Контактний центр клієнтів — підтримка клієнтів цілодобово за допомогою контакт-центру, який має всі канали зв’язку, згадані вище.

– Аналітика продажів — надається докладні звіти для маркетингових кампаній і каналів. Треба визначити, яке джерело трафіку є найуспішнішим із рентабельністю інвестицій у рекламу, і переглянути коефіцієнти конверсії продажів у звіті про трафік. Також є можливість отримати доступ до прогресу кожного менеджера за допомогою зручної для читання таблиці, це означає, що можливо визначити, де команда може відчувати труднощі, і переконатися, що вони отримають необхідну підтримку.

– Звіти та продажі — відстеження цілей продажів і інформація, скільки угод має кожен етап конвеєра.

– Рахунки в *CRM* — є можливість обробляти платежі в CRM і генерувати пропозиції та рахунки-фактури з угод. *Bitrix24* підтримує всі основні платіжні системи, включаючи кредитні та дебетові картки, *PayPal*, *Stripe* і *Braintree*.

– Відстеження потенційних клієнтів і веб-форми — додавання веб-форми на свій веб-сайт і можливість отримати доступ до інформації про потенційних клієнтів із *CRM*.

1.1.3. *Worksection*

Worksection стає основним вибором завдяки своїй гнучкості та потужній функціональності, таким чином сприяючи будь-якій сучасній методології *Agile*. Простий інтерфейс, таймер і пристрій відстеження часу, дошки *Kanban*, діаграма Ганта та гнучке управління проектами є основними цінностями для 1000 компаній, які використовують інструменти *Worksection*, включаючи, зокрема, *BBDO, Havas Digital*, маркетингове агентство *Volta One*, студія цифрового виробництва *COXO* та інші.

Навіть працівник з невеликим досвідом роботи з подібними сервісами легко освоїть *Worksection*. Сервіс допомагає досвідченим *IT*-спеціалістам, бухгалтерам і менеджерам.

Є можливість створити унікальний статус для кожного проекту. Наприклад, відеопродукція може використовувати статус для позначення етапів підготовки, зйомки, постпродакшну та публікації.

Має систему тегів. Тег - це набір маркерів для проектів і завдань, які вказують на тип, складність і обсяг проекту або результат роботи. Теги допоможуть упорядкувати список проектів за відповідними параметрами.

Можливість створювати пріоритети для завдань, щоб спланувати щотижневі спринти та спростити спільну роботу команд над проектами на основі *Worksection*.

Будь-яка дія співробітника в системі буде повідомлена сповіщенням. Кожен користувач індивідуально налаштовує призначення таких сповіщень: поштову скриньку, вхідну скриньку, мобільний телефон або робочий стіл.

Усе спілкування, пов’язане із завданням, відбувається в межах завдання. Це груповий чат, який містить всю історію оновлення завдань, включаючи

призначення відповідального керівника, причину зсуву дедлайну, те, що написав клієнт, і відповідь менеджера.

Kanban візуалізує пов'язані з проектом завдання у вигляді дошки з наклейками та допомагає співробітникам просувати завдання поетапно. Після завершення етапу *Worksection* автоматично перенаправить завдання новому та повідомить відповідальних працівників.

Worksection має ведику кількість можливостей для покращення роботи в команді. Фактичний план показує кількість запланованих завдань і кількість завдань, виконаних протягом одного спринту. Усі ці цифри відображатимуться як розрізані за людьми, проектами або людьми та проектами за вибраний період.

Робота з клієнтами з США, Європи та Азії. Сервіс доступний англійською, німецькою, іспанською, китайською, польською, українською та російською мовами. Наприклад, в рамках одного проекту можливо легко працювати з одним клієнтом англійською мовою, а з іншим – іспанською мовою інтерфейсу.

Здатність керувати завданнями через додаток *Worksection* для *iOS* та *Android*.

Можливість інтеграції з іншими бізнес-додатками, такими як системи *CRM*, *GoogleDocs*, *Slack* і *Telegram*. Відкритий *API*, який дозволяє завантажувати та завантажувати дані з *Worksection*.

1.1.4. *NetHunt*

NetHunt CRM – це хмарний інструмент керування взаємовідносинами з клієнтами, який інтегрується з *Gmail* і дозволяє компаніям керувати взаємодією та записами безпосередньо з папки «Вхідні». Він генерує потенційних клієнтів через електронну пошту, повідомлення в чаті та соціальні мережі. Він забезпечує всі оновлення *CRM* через ці платформи.

Проста автоматизація для зайнятих людей. *NetHunt* автоматично переміщує інформацію між вашими веб-програмами, щоб ви могли зосередитися на найважливішій роботі. *NetHunt CRM* підключається до низки інструментів, щоб спростити пошук потенційних клієнтів: *LinkedIn, Web Forms, Facebook, Gmail, Google Contacts* тощо. Можливість додати нових потенційних клієнтів автоматично або одним натисканням кнопки, очищуйте дані про потенційних клієнтів і негайно вставляйте їх у свою систему *CRM*, готову до догляду.

1.2. Значення системи управління взаємовідносинами з клієнтами

Термін «управління взаємовідносинами з клієнтами» (*CRM*) з'явився у 1980-х роках, але процес управління даними про клієнтів почався задовго до цього. Перші дні *CRM* передбачали просто подання профілів клієнтів на папері. Однак із величезним діловим і технологічним прогресом *CRM* стали потужним бізнес-програмним забезпеченням, що робить його ключовим інструментом для команд будь-якого розміру.

Ця історія *CRM* почалася з однієї з найперших форм зберігання контактної інформації клієнтів — *Rolodex*. Інструмент, що обертається, створений датським інженером Хільдауром Нейлсеном у 1956 році, містить колекцію покажчиків і візитних карток, які люди можуть обертати та гортати. У 1950-х роках багато компаній використовували *Rolodex* для додавання, оновлення, і зберігати інформацію своїх клієнтів.

Мейнфрейми — це надійні комп'ютерні системи, які використовуються для зберігання та обробки даних. Хоча першу мейнфрейм-систему винайшов між 1930-ми і 1940-ми роками дослідник з Гарварду Говард Ейкен, вона стала доступною для підприємств лише приблизно в 1960-х роках. Компанії які швидко розвивалися перейшли від ручного *Rolodex* до використання мейнфреймів для оцифрування інформації про клієнтів, такої як імена, адреси та контактні дані.

На початку 1980-х років підприємства знайшли спосіб використовувати інформацію про клієнтів, яку вони збирали через маркетинг баз даних. Маркетинг бази даних — це процес звернення до клієнтів у їхній базі даних і пропонування їм продуктів або послуг. Зазвичай це робилося через пряму поштову розсилку, де брошури та каталоги продуктів надсилалися клієнтам з метою переконати їх зробити покупку.

Ближче до другої половини десятиліття компанії почали практикувати більш проактивний підхід до продажів, проводячи вихідні дзвінки з продажу. У цей час комп'ютери також стали більш доступними для бізнесу, що уможливило подання та керування інформацією про клієнтів у цифровому форматі.

Роки після того, як було представлено маркетинг баз даних і перше програмне забезпечення для керування контактами, продовжували розроблятися нові технології для кращого керування даними клієнтів, що постійно зростали. Це проклало шлях до автоматизації відділу продажів, піонером якої був бізнесмен і технолог Том Сібель.

У 1993 році Сібель заснував компанію *Siebel Systems*, яка спочатку пропонувала інструменти автоматизації продажів, а згодом розширилася до розробки програм автоматизації маркетингу та обслуговування клієнтів. *Siebel Systems* стала лідером ринку та найпопулярнішим постачальником систем автоматизації продажів на той час.

У 1999 році з'явилася портативна *CRM*, коли *Siebel Systems* запустила першу в історії мобільну *CRM* під назвою *Siebel Sales Handheld*. Пристрій, що працює на *ОС Windows CE*, дозволяв користувачам обмінюватися та синхронізувати інформацію про клієнтів через дані *Siebel Sales Enterprise*.

Це дало змогу членам відділу продажів бути в курсі всіх взаємодій з клієнтами. *Oracle, SAP* і *PeopleSoft* наслідували їхній приклад і також випустили власні мобільні версії.

Salesforce, популярне рішення *CRM*, також запущене в 1999 році. Воно було розроблено для доставки програмного забезпечення за новою моделлю під

назвою *Software-as-a-Service* у той час, коли Інтернет набув широкого поширення. Метою компанії було усунення дорогих початкових витрат і витрат на технічне обслуговування, а також тривалого впровадження системи. Згодом *Salesforce* стане власником цього простору та стане найбільшим у світі постачальником *CRM*.

Запровадження Інтернет-програмного забезпечення *Salesforce* було дуже успішним і домінувало в 2000-х роках. Це вважалося проривом, оскільки дозволяло користувачам отримувати доступ до даних про продажі та клієнтів через будь-який підключений пристрій.

У 2004 році було створено безкоштовну *CRM*-програму з відкритим кодом, коли технологи Клінт Орам, Джон Робертс, і Джейкоб Тейлор працювали над проектом під назвою *Sugar Open Source*. Використовуючи свій попередній досвід роботи в *IBM* і *Hewlett-Packard*, вони створили та розмістили програмне забезпечення на *SourceForge*, сховищі безкоштовного програмного забезпечення з відкритим кодом. Зрештою це змусило їх заснувати *SugarCRM*, зробивши *Sugar Open Source* версії 1.0 загальнодоступною.

У 2009 році компанія *Salesforce* досягла ще однієї віхи в історії обслуговування клієнтів, представивши *Service* Cloud. Використовуючи модель *SaaS*, *Service Cloud* пропонує функції автоматизації обслуговування клієнтів і підтримки. У тому ж році з'явився перший додаток, побудований на *Force.com* для *iPhone* від *Apple* було запущено та стало першим корпоративним партнером у *App Store* від *Apple*.

Кінець 2000-х, підприємства почали використовувати канали соціальних мереж, такі як *Facebook* і *Twitter*, як спосіб налагодити стосунки з клієнтами. Компанії почали використовувати соціальні мережі у своїй стратегії *CRM*, переходячи від транзакційного до інтерактивного управління взаємовідносинами з клієнтами. Ближче до кінця десятиліття, були засновані деякі постачальники *CRM*, які ми знаємо сьогодні, наприклад *Insightly*, *Pipedrive* і *Freshsales*.

За останнє десятиліття відбувся різкий прогрес у технологіях і тому, як бізнес покладається на них. У 2010 *CRM*-системи та технології загалом стали справді мобільними завдяки збільшенню кількості смартфонів, планшетів та інших розумних пристроїв. Індустрія перейшла від простого використання традиційних комп'ютерів або ноутбуків для доступу до Інтернету , багатоканальне підключення.

З цими розробками прийшов приплив нових типів програмного забезпечення *CRM* з власними унікальними торговими пропозиціями. Окрім простого пропонування універсальних варіантів *CRM*, ці постачальники спеціалізуються на задоволенні певних бізнес-потреб або обслуговуванні певних галузей, як-от:

– соціальні *CRM* працюють з різними платформами соціальних мереж, які дозволяють користувачам відстежувати взаємодію, а також слухати свою аудиторію та спілкуватися з нею. Це також дає змогу компаніям оптимізувати взаємодію з клієнтами на всіх етапах життєвого циклу та надає повніші контактні дані;

– *CRM* для професіоналів з бухгалтерського обліку розроблено, щоб допомогти бухгалтерам, бухгалтерам і фінансовому персоналу керувати виставленням рахунків і виставленням рахунків на централізованій платформі. Він пропонує пов'язані з фінансами функції, такі як обробка платежів і обчислення податків;

– *CRM* для професіоналів з нерухомості надають інструменти для керування операціями з нерухомістю, генерації потенційних клієнтів щодо купівлі житла та моніторингу списків. Він також пропонує функції для відстеження потенційних клієнтів і маркетингу для них через електронну пошту, телефон, онлайн-рекламу та соціальні мережі;

– *CRM* для будівельних підприємств пропонують інструменти для керування контрактами, пропозиціями та поточними проектами. До них належать генеральні підрядники та спеціальні роботи, як-от теслярство, знесення, сантехніка, електрика, реставрація та зварювання;

– *CRM* для професіоналів у сфері страхування, включно з незалежними страховими агентами та брокерами, допомагають користувачам шукати та розвивати потенційних клієнтів. Він також відстежує нові угоди та цілі, керує політиками та планами, а також реєструє такі дії, як електронні листи та зустрічі.

Більш складні вдосконалення в різних функціональних можливостях *CRM*, таких як аналітика, звітність і аналітика даних під час обробки інформації про клієнтів, неминучі. Очікується, що *CRM* продовжуватимуть адаптувати більш надійну все-в-одному, ще більше зменшуючи потребу в інтеграції сторонніх розробників.

Деякі з найбільш передових і передових-мислячі постачальники *CRM* почали інтегрувати штучний інтелект у своє програмне забезпечення. Машинне навчання обробляє величезні обсяги даних для визначення потенційних клієнтів, автоматизації нудних процесів і надання користувачам найповнішої картини кожного з їхніх потенційних клієнтів.

Через різноманітні досягнення в *CRM* цілком природно, що її інтегрують у загальну стратегію обслуговування клієнтів. Крім того, він може точно передбачити, як клієнти взаємодіятимуть із їхнім брендом на основі їхнього минулого досвіду. Підприємства можуть бачити, як поводяться клієнти, починаючи з етапів обізнаності та закінчуючи етапами оцінки та покупки.

Це зрештою дає їм можливість персоналізувати та налаштовувати свої пропозиції на детальному рівні.

1.3. Висновки до розділу

На даний момент існує велика кількість CRM, які дозволяють покрити 90% потреб клієнтів. Але створення власного продукту дозволяє розробити інтерфейс, який користувачу більш підходить під його можливості. Також не кожна CRM система має гнучку систему інтеграції з зовнішніми сервісами, що змушує користувача використовувати декілька додатків.

Актуальність теми полягає в замовленні клієнта розробити власний продукт. Дана CRM-система має бути повністю ізольована та універсальна, пов’язана з інтернет магазином клієнта, мати використовувати зазначенні клієнтом технології та інтерфейс.

У якості мови програмування при розробці обрано мову програмування Python. Оскільки на даний момент набирає популярність використовувати веб-додатки, було прийнято рішення розробити CRM-систему на фреймворці Django.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СТРУКТУРИ СИСТЕМИ КЕРУВАННЯ ВЗАЄМОВІДНОСИНАМИ З КЛІЄНТАМИ

2.1. Проектування загального функціонального прототипу системи CRM

2.1.1. Функціональний прототип панель відображення даних лідів для обраної CRM воронки

Лід – у сенсі систем керування взаємовідносинами з клієнтами, це потенційний клієнт, який перейшов за посиланням до ресурсу, та якось з ним взаємодіяв. Панель відображення лідів, яка зазначена на рис. 2.1, потрібна для користувача, щоб відстежувати усіх потенційних клієнтів сортувавши їх за станом, який зазначено у фільтрах(рис. 2.2).

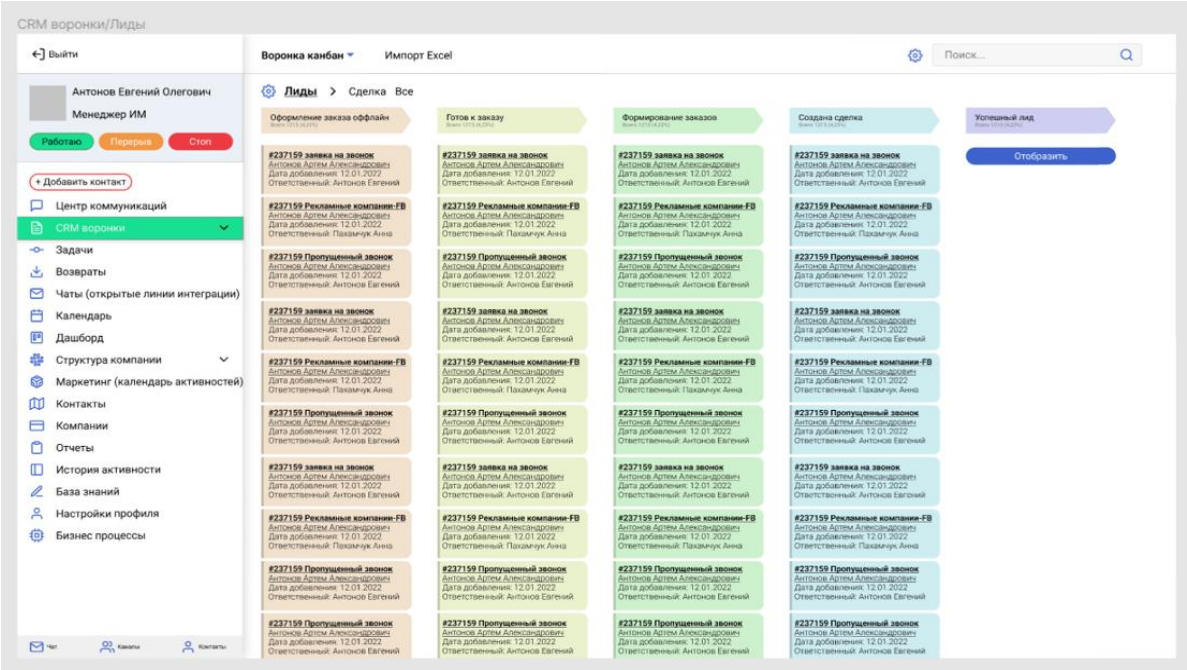


Рис. 2.1. Функціональний прототип панель відображення даних лідів

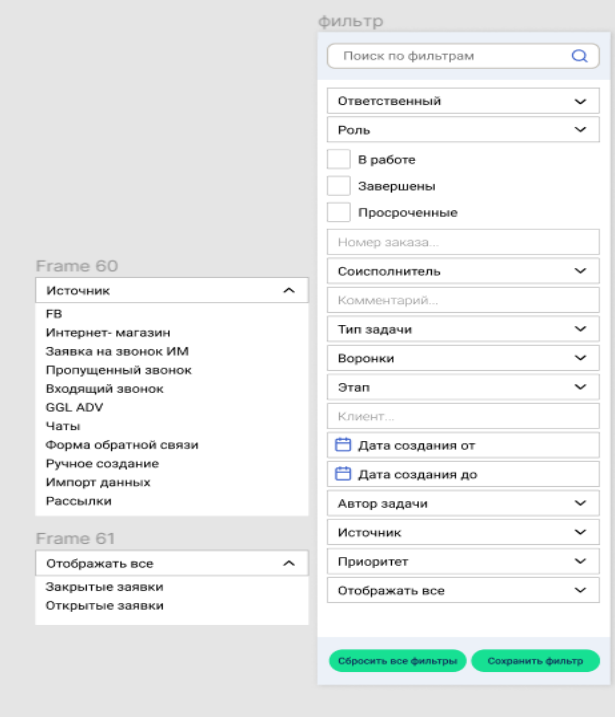


Рис. 2.2. Функціональний прототип панелі фільтрів для панелі лідів

2.1.2. Функціональний прототип панелі відображення даних про обрану CRM воронку та її угоди

Угода – у сенсі системи керування взаємовідносинами, це клієнти, які вже здійснили покупку товарів, та мають статус активного користувача. Панель для відображення угод, яка зазначена на рис. 2.3, потрібна користувачу, щоб фільтрувати клієнтів за статусами їх замовлень.

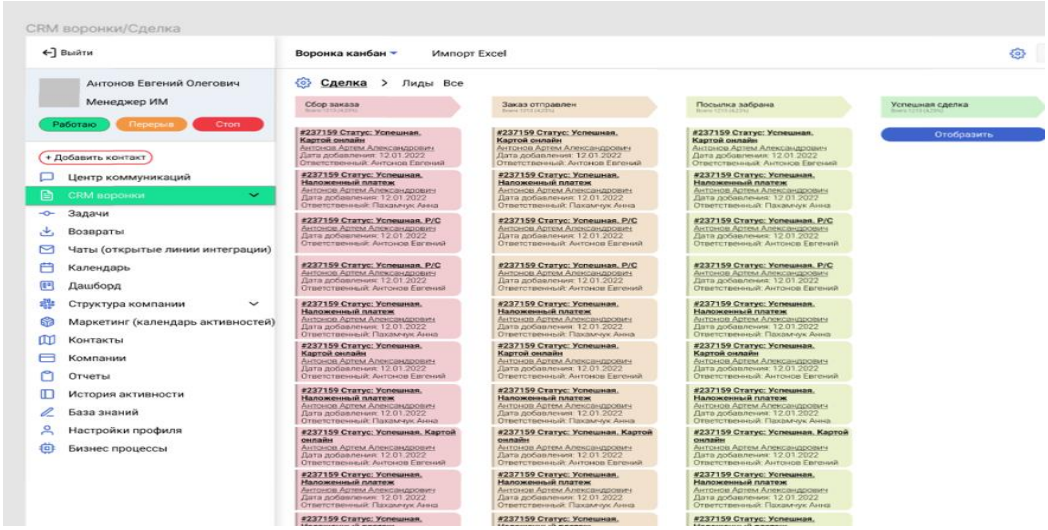


Рис. 2.3. Функціональний прототип панелі воронки та угод

2.1.3. Функціональний прототип панелі загального списку усіх лідов та угод

Дана панель потрібна користувачу бачити загальне його навантаження, потенційних та активних клієнтів (рис. 2.4).

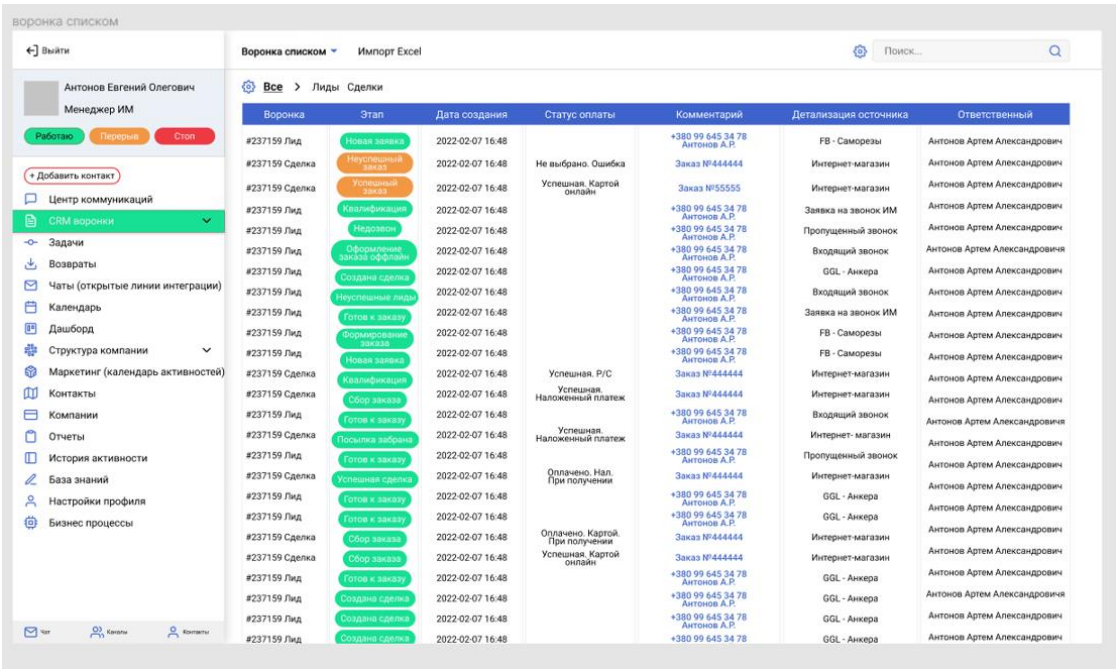


Рис. 2.4. Функціональний прототип загального списку усіх лідов та угод

2.1.4. Функціональний прототип відображення акцій у вигляді календаря з акціями

Дана панель відображає, у вигляді календарю, коли в компанії діють акції (рис. 2.5).

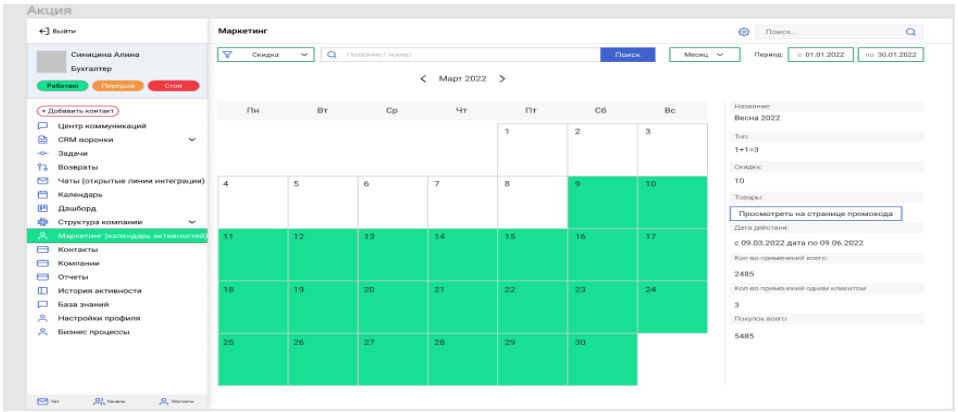


Рис. 2.5. Функціональний прототип календаря з акціями

2.1.5. Функціональний прототип знижок у вигляді календаря зі знижками

Дана панель відображає, у вигляді календарю, коли в компанії діють знижки (рис. 2.6).

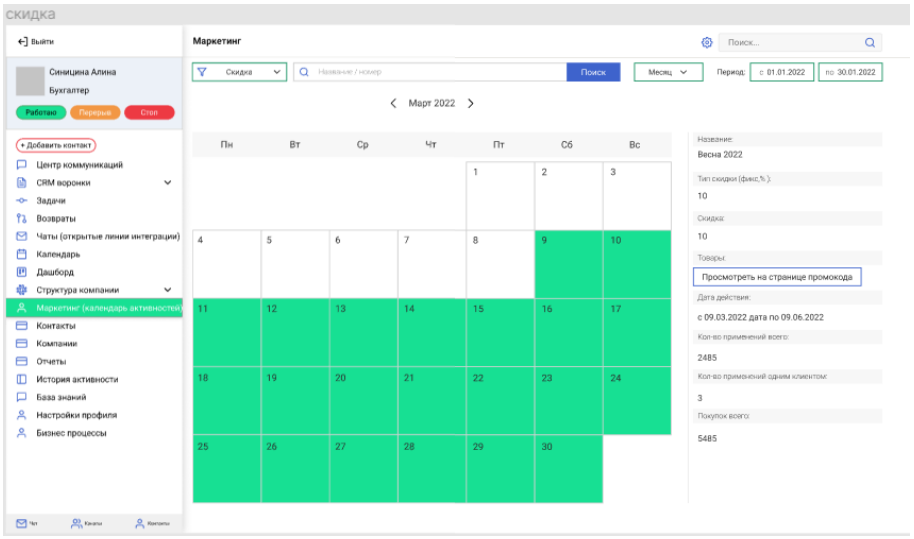


Рис. 2.6. Функціональний прототип календаря з знижками

2.1.6. Функціональний прототип промокодів у вигляді календаря

Дана панель відображає, у вигляді календарю, коли в компанії діють промокоди (рис. 2.7).

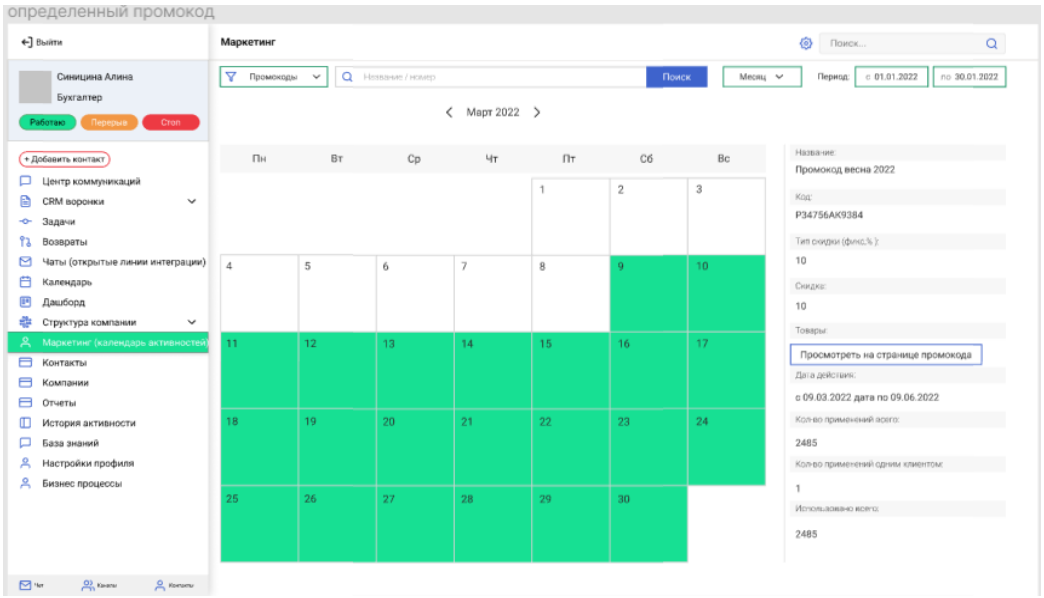


Рис. 2.7. Функціональний прототип усіх промокодів за обраний період

Також при необхідності є можливість обрати період за який потрібно отримати інформацію стосовно промокодів (рис. 2.8).

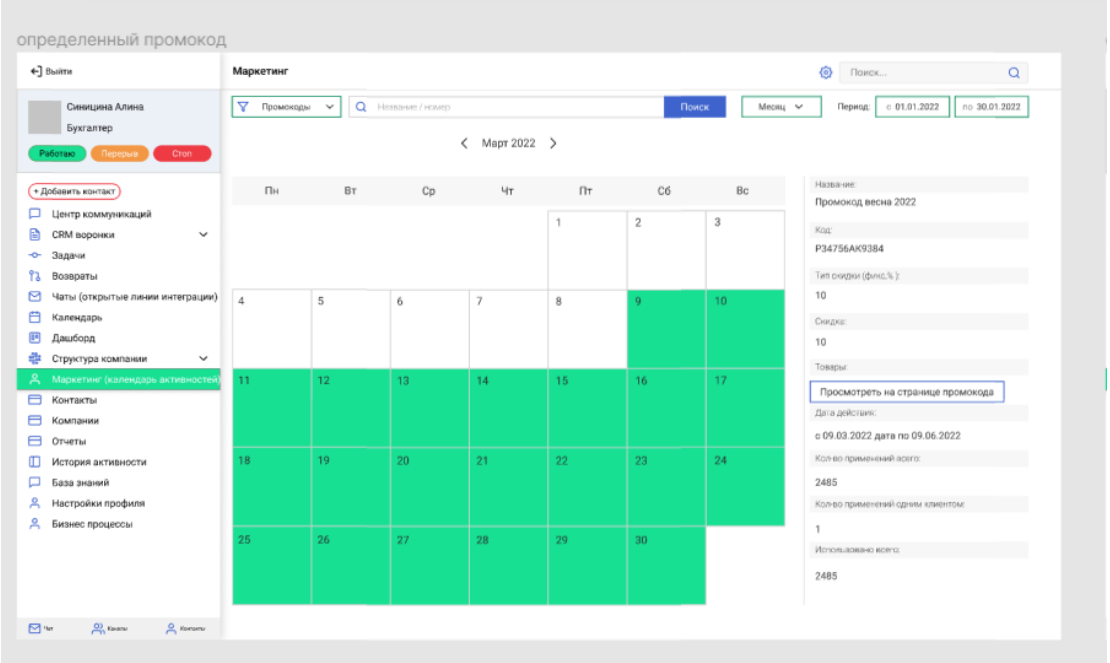


Рис. 2.8. Функціональний прототип інформації про окремий промокод

2.1.7. Функціональний прототип про усі види акцій, промокодів та знижок

Панель для відображення усіх видів акцій, дозволяє вивести список акцій у вигляді календарю за обраний період, наприклад за тиждень як на рис. 2.9 або за місяць як на рис. 2.10 Також можливо приховати окремий вид акцій(рис. 2.11).

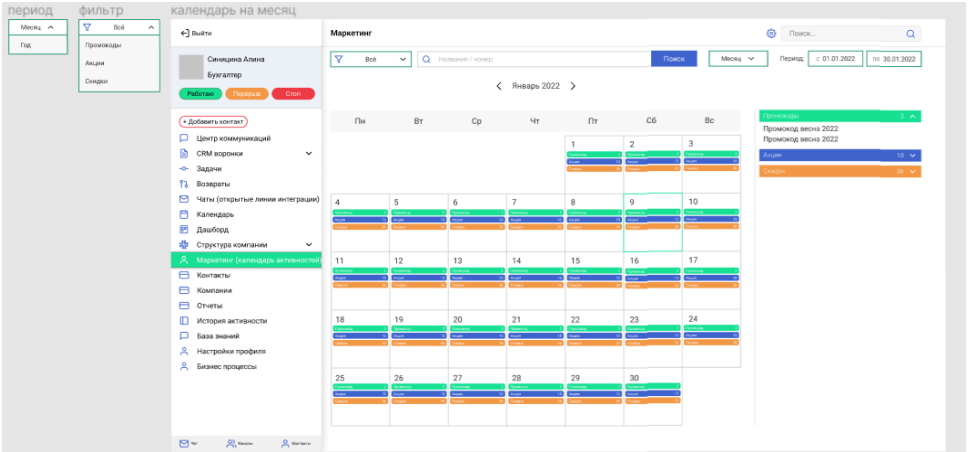


Рис. 2.9. Функціональний прототип про усі види акцій за місяць

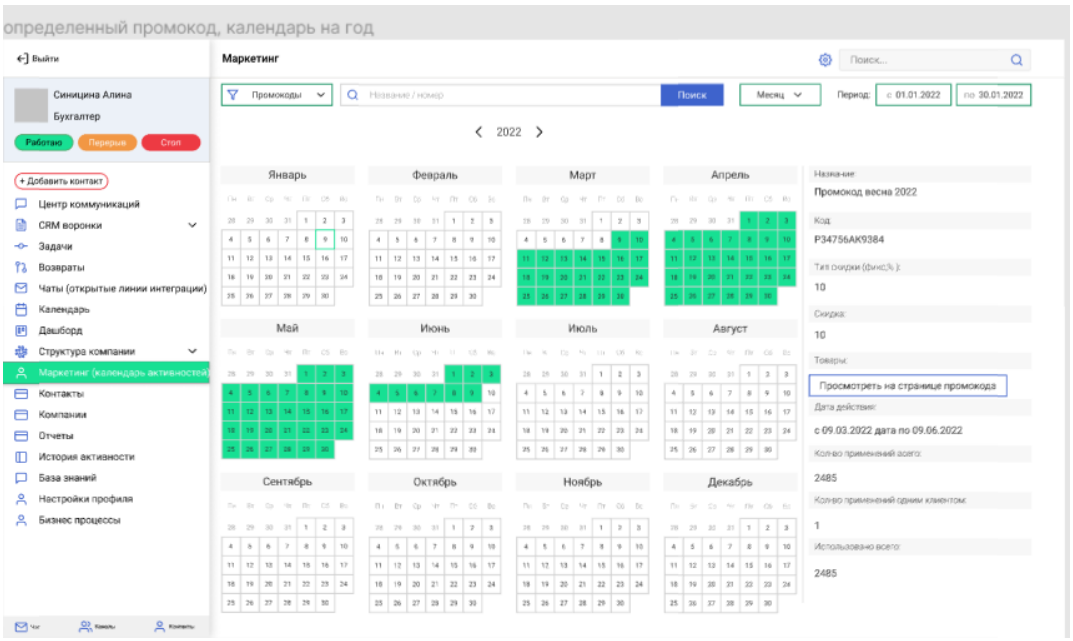


Рис. 2.10. Функціональний прототип про усі види акцій за рік

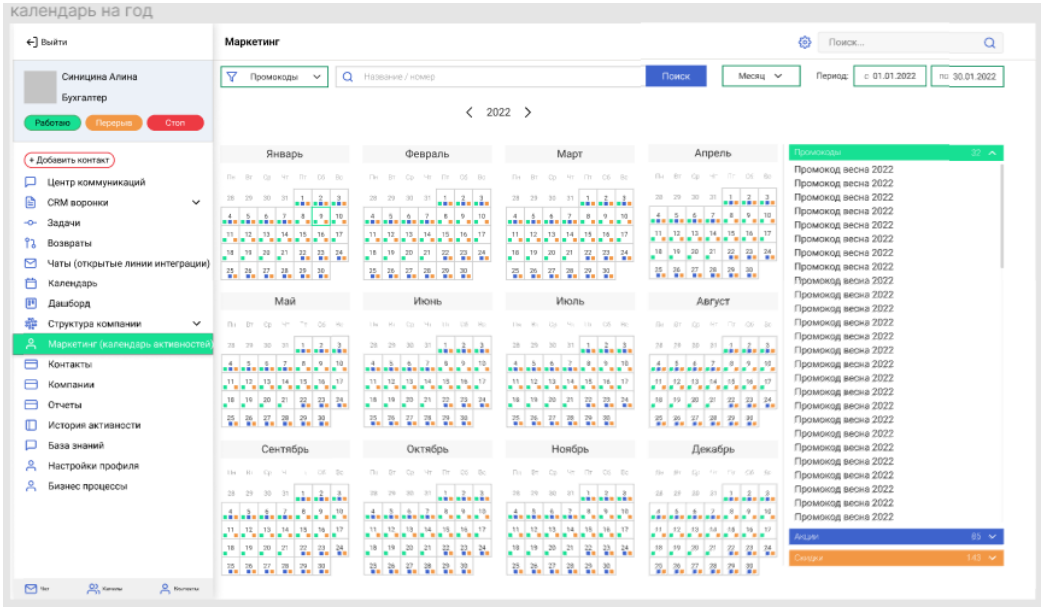


Рис. 2.11. Функціональний прототип про окремий промокод за рік

2.1.8. Функціональний прототип центру комунікацій

Центр комунікацій – це панель для відображення повідомлень користувача у системі (рис. 2.12). Для цього використовується запис повідомлень на кожну дію системи. Також панель повинна містить інформацію про усі ліди та угоди користувача, про заплановані задачі, список повернення

товарів, причини звернення клієнтів та список останніх дзвінків. Зверху праворуч панелі міститься три кнопки:

– створення задачі;

– відправка *SMS;*

– відправки електронного листа.

Ліворуч розташовано кнопка створення заказу. В низу панелі розташовані три кнопки:

– зберегти зміни;

– невдалий клієнт;

– змінити статус.

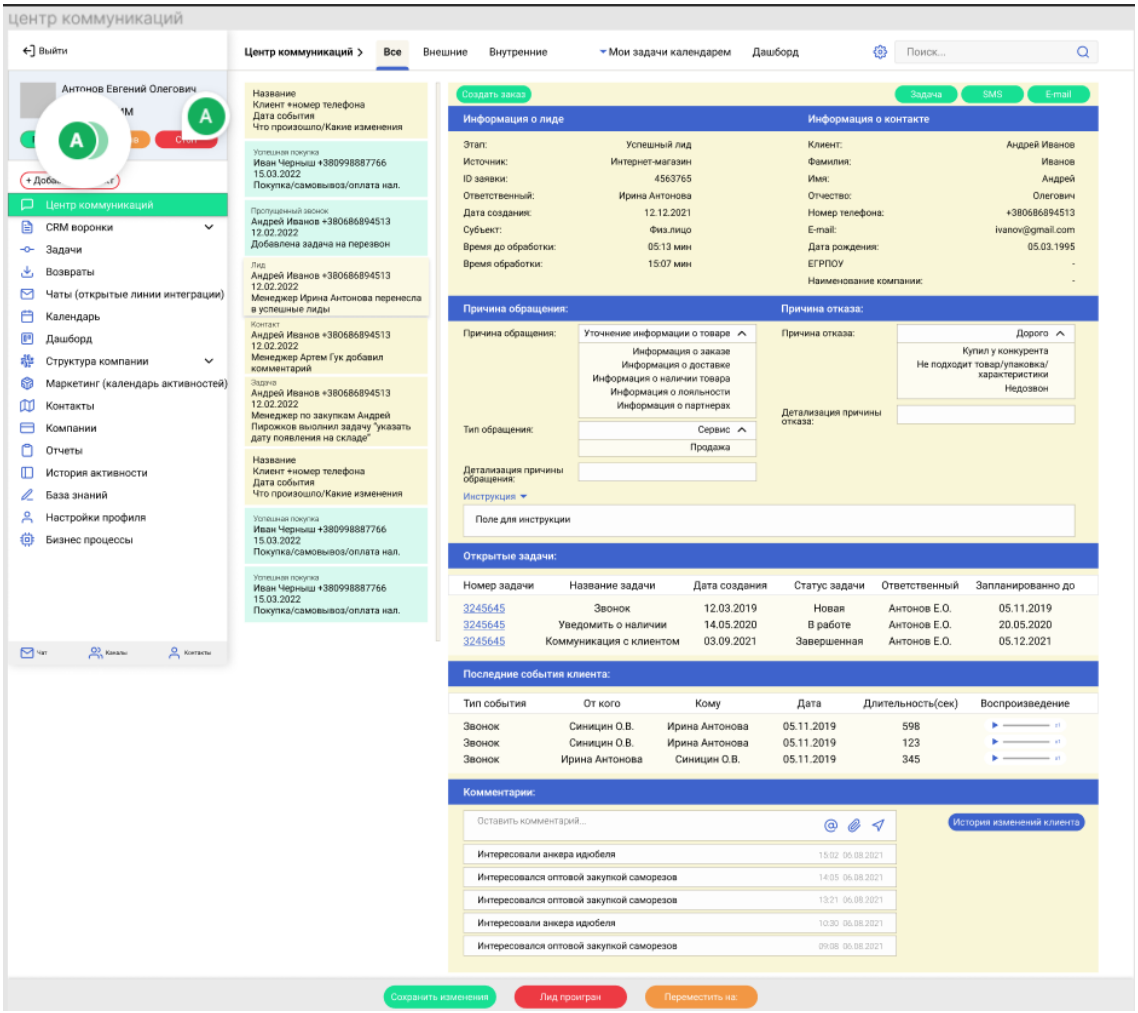


Рис. 2.12. Функціональний прототип центру комунікацій

2.1.9.Функціональний прототип бокового меню

Бокове меню, яке зазначене на рис. 2.13, потрібно для швидкої зміни статусу роботи користувача. Перехід для найбільш популярних посилань системи.

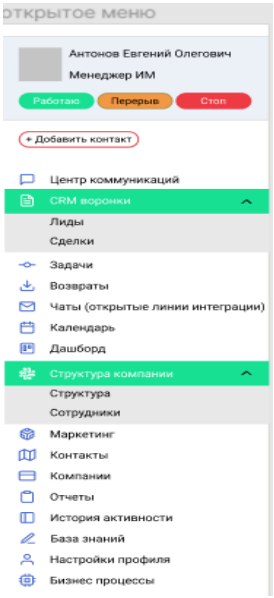


Рис. 2.13. Функціональний прототип бокового меню

2.1.10. Функціональний прототип бокової панелі чату

Бокова панель чату дозволяє швидко перейти до останніх чатів з іншими користувачами (рис. 2.14).

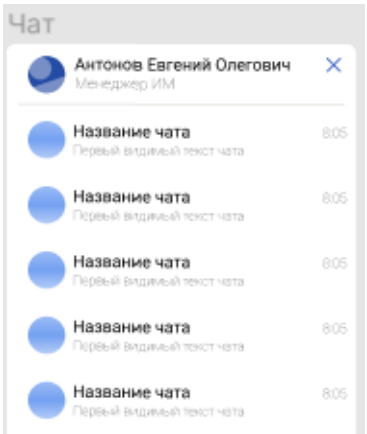


Рис. 2.14. Функціональний прототип бокової панелі чату

2.1.11. Функціональний прототип панелі окремої угоди

Для відображення інформації про кожну угоду потрібна відповідна панель (рис. 2.15).

Панель містить інформацію про угоду, про замовлення, про товари що входять до замовлення, про останні дзвінки стосовно угоди та останні дії клієнта. Також у розділі звернення та повернення товару є інформація про статус товару та можливість відмінити доставку товару. В самому низу панелі розташовано три кнопки:

– зберегти зміни;

– угода провалена;

– змінити етап угоди.

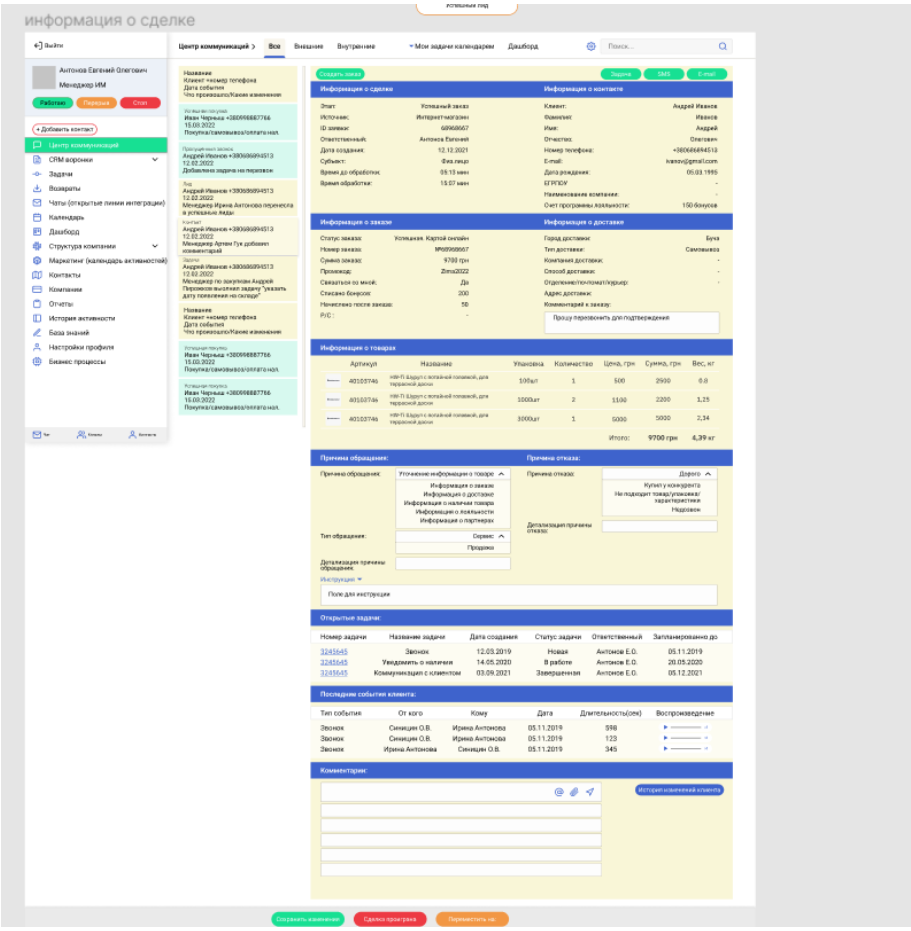


Рис. 2.15. Функціональний прототип панелі окремої угод

2.1.12. Функціональний прототип панелі створення задачі

Нові задачі потрібно створювати у спеціальній панелі (рис. 2.16). Вона повинна містить поля для заповнення, а також спеціальні кнопки для задачі:

– відмітити користувача;
– додати файл до задачі;
– встановити теги;
– додати до списку задач.

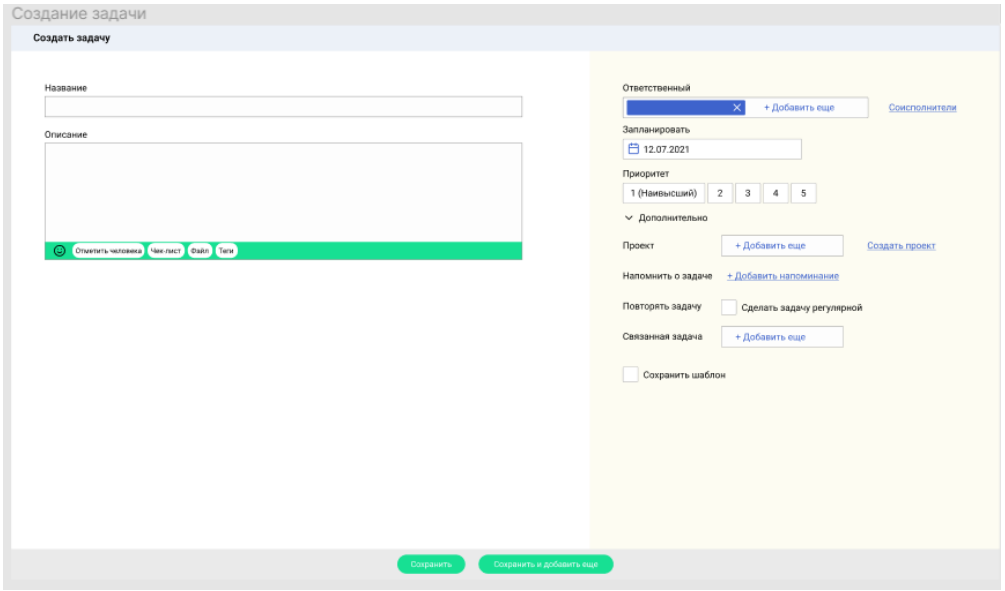


Рис. 2.16. Функціональний прототип панелі створення задачі

2.1.13. Функціональний прототип панелі історії зміни клієнта

Кожен крок клієнта повинен фіксуватися та записуватися у базі даних. Для відображення даних про дії клієнта повинна бути панель історії зміни клієнта (рис. 2.17).

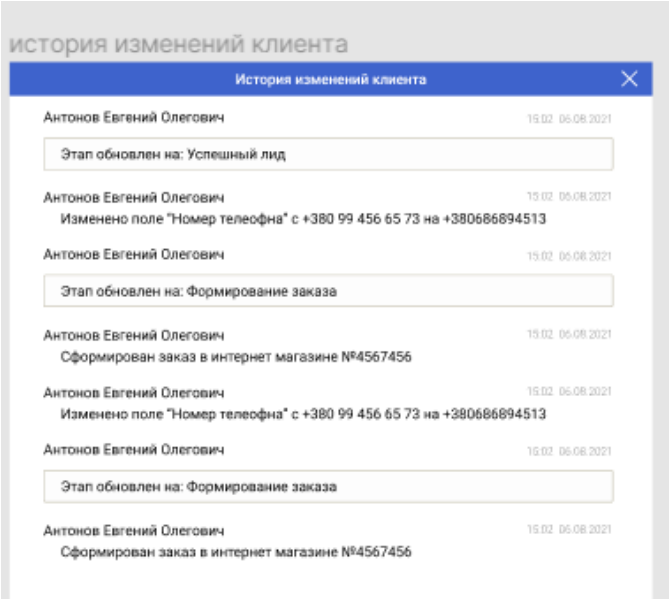


Рис. 2.17. Функціональний прототип панелі історії зміни клієнта

2.1.14. Функціональний прототип календаря користувача з задачами

Один з варіантів відображення списку задача користувача – календар із задачами (рис. 2.18). Даний метод відображення задач дозволяє користувачу розподілити свій час рівномірно та бачити прогрес виконання задач протягом тижня.

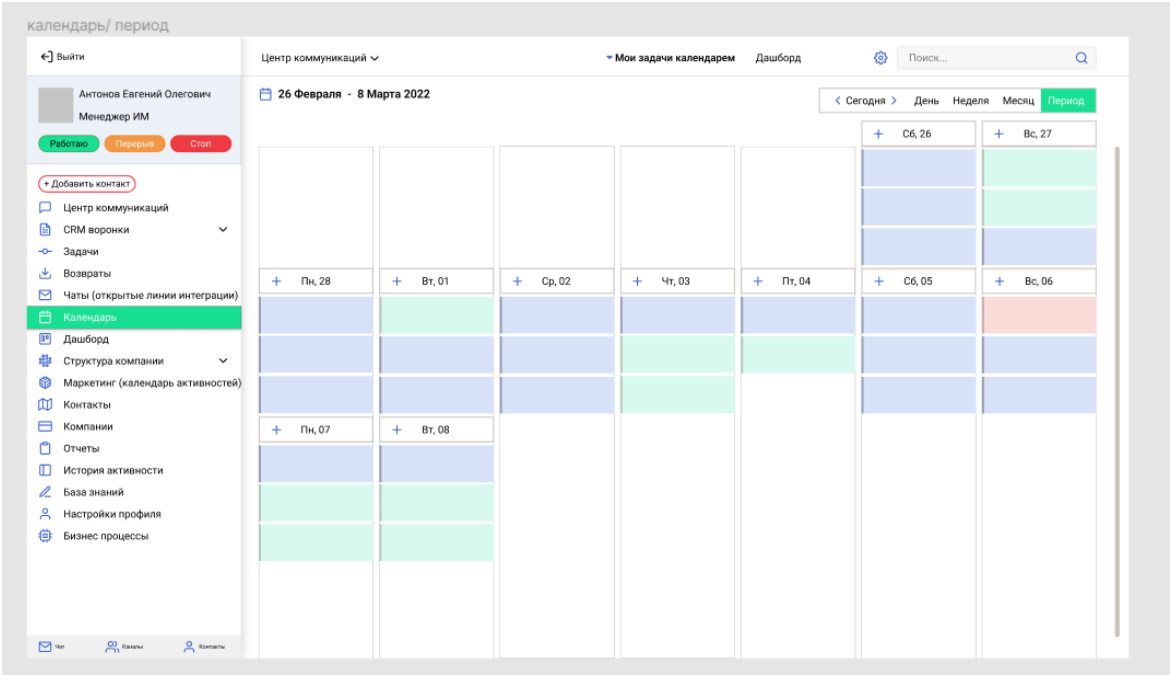


Рис. 2.18. Функціональний прототип календаря користувача з задачами

2.1.15. Функціональний прототип панелі історії користувача у системі

Для контролю співробітників, потрібна панель для відображення їх історії у системі (рис. 2.19). Історія містить дані про зміни у лідах та угодах співробітника, усі дзвінки користувача відфільтровані по даті та часу, а також змогу коментувати кожну дію користувача.

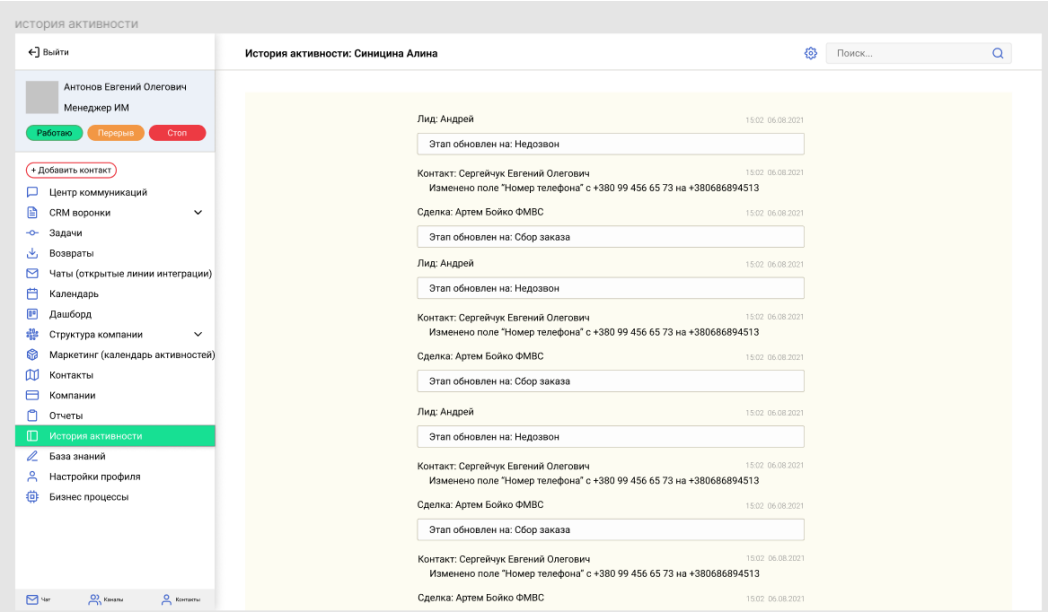


Рис. 2.19. Функціональний прототип панелі історії користувача у системі

2.1.16. Функціональний прототип панелі повернення товару

Панель повернення товару, яка зображена на рис. 2.20, містить інформацію про повернення товарів клієнтів користувача. Також праворуч від кожного повернення розміщені кнопки підтвердження або відхилення повернення товару клієнту.

Для кращого розуміння, які товари є у системі, нижче є спеціальна таблиця з інформацією про артикул, назву, упаковку та кількість товарів.

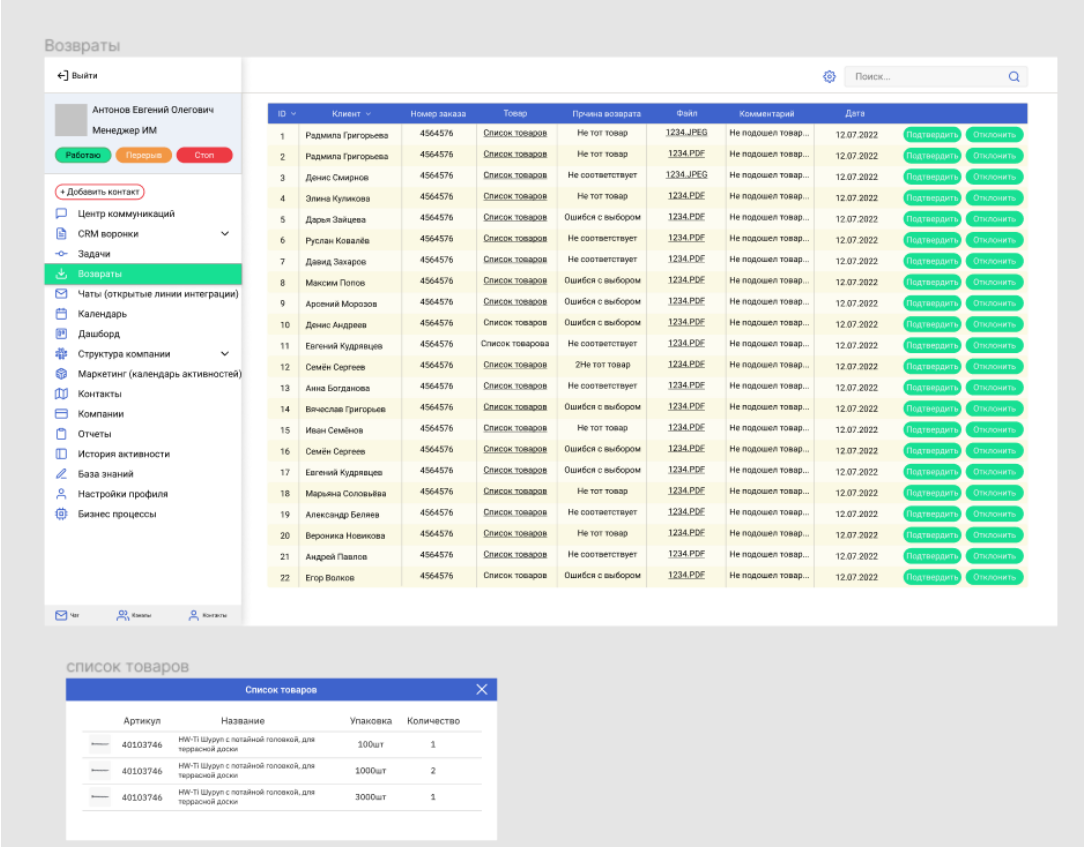


Рис. 2.20. Функціональний прототип панелі повернення товару

2.1.17. Функціональний прототип панелі структури компанії

Усі користувачі мають відношення до компаній та відділів у яких вони працюють. Для загального уявлення структури компанії є панель відображення ієрархії усіх користувачів та відділів(рис. 2.21).

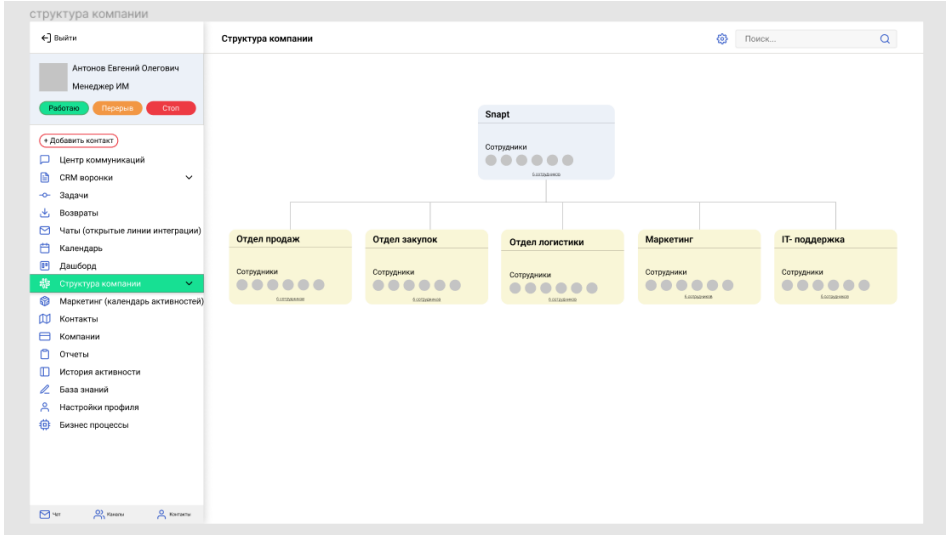


Рис. 2.21. Функціональний прототип панелі структури компанії

2.1.18. Функціональний прототип панелі списку співробітників у компанії

Панель відображення списку працівників, що зображена на рис. 2.22, дозволяє побачити усіх працівників компанії та додати нових до списку, якщо користувач власник.

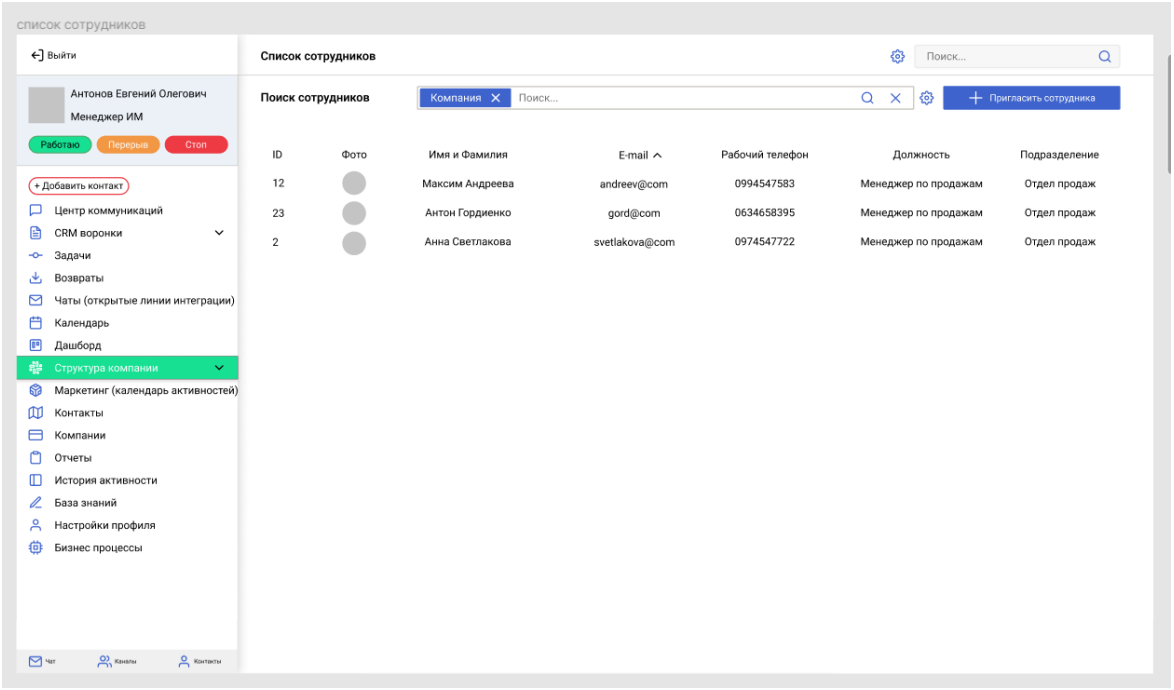


Рис. 2.22. Функціональний прототип списку співробітників у компанії

2.1.19. Функціональний прототип панелі глобального пошуку

Дана панель містить поле вводу інформації для пошуку у системі (рис.2.22). Результати пошуку групуються за типом об’єкта пошуку та виділяються різними цвітами в залежності від наявного статусу та виводяться на нову панель(рис.2.23).

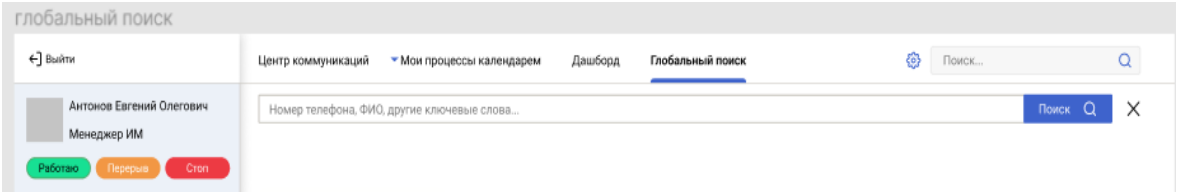


Рис. 2.23. Функціональний прототип панелі глобального пошуку

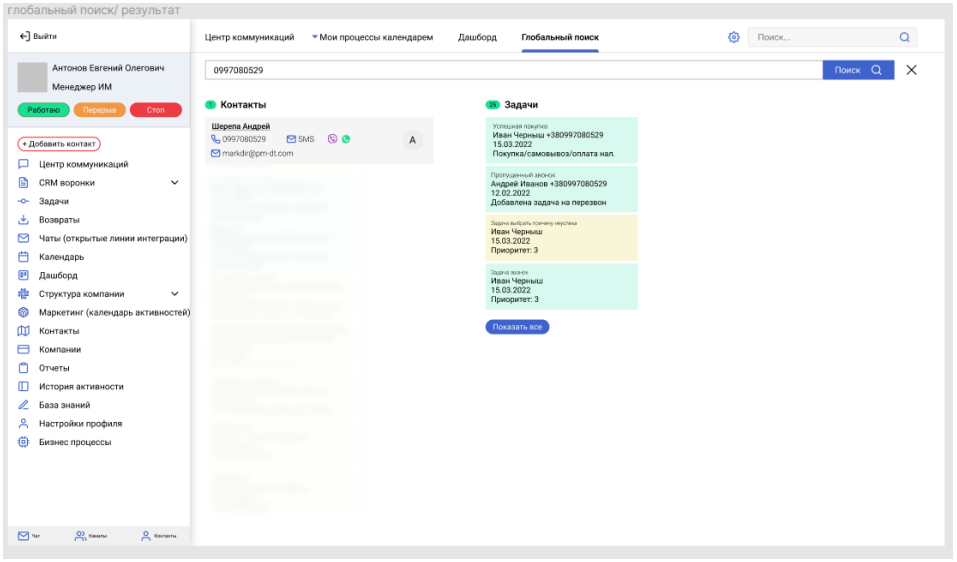


Рис. 2.24. Функціональний прототип панелі з результатом пошуку

2.1.20. Функціональний прототип панелі співробітника з аналітичними даними

Кожний співробітник може відкрити сторінку своїх підлеглих та подивитися статистичні дані про їх роботу на панелі профілю працівника (рис.2.25).

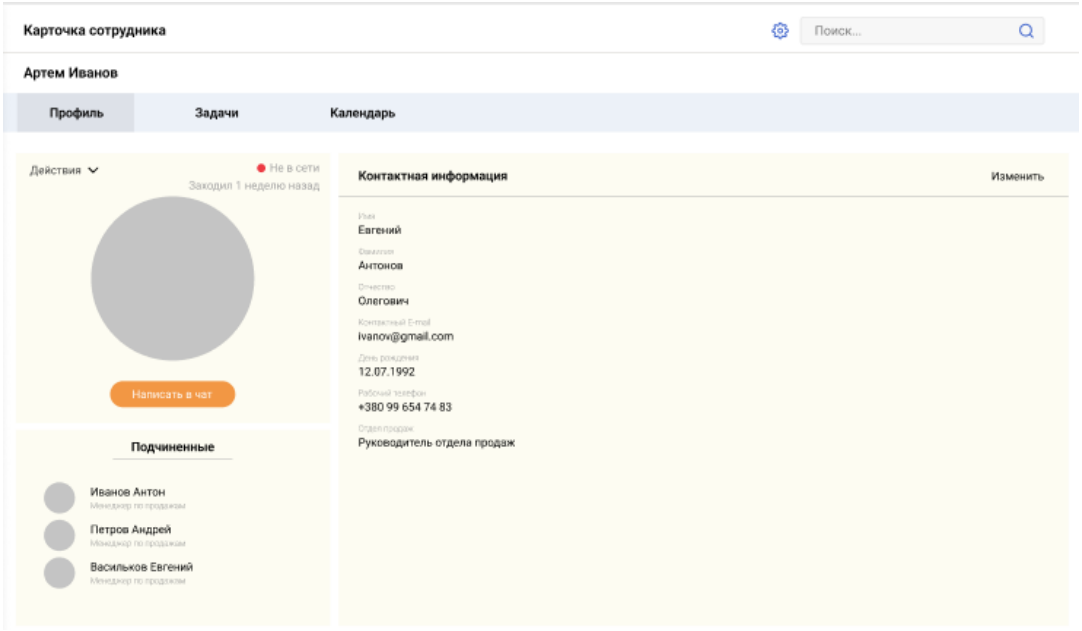


Рис. 2.25. Функціональний прототип аналітики співробітника

2.1.21. Функціональний прототип панелі контакту та його історія замовлень

Для перегляду даних про клієнта необхідно відкрити панель контакту, де знаходяться історія його замовлень, а також уся особиста інформація (рис.2.26). Також на панелі є три кнопки для взаємодії з клієнтом:

– створити задачу та пов’язати з цим клієнтом;
– зв’язатися через електрону пошту;
– відправити *SMS*.

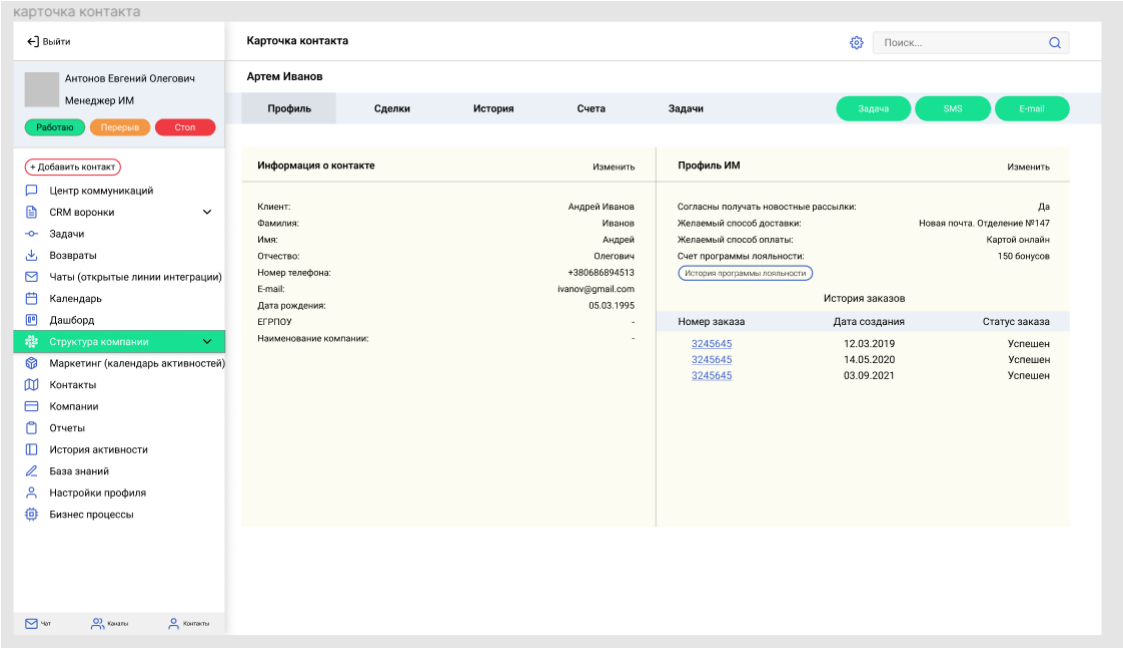


Рис. 2.26. Функціональний прототип контакту та його історія замовлень

2.1.22. Функціональний прототип панелі карточки компанії

Кожна компанія має особисту карточку з особистою інформацією (рис.2.27).

Також на панелі є три кнопки для взаємодії з компанією:

– створити задачу та пов’язати з цією компанією;
– зв’язатися через електрону пошту;
– відправити *SMS*.

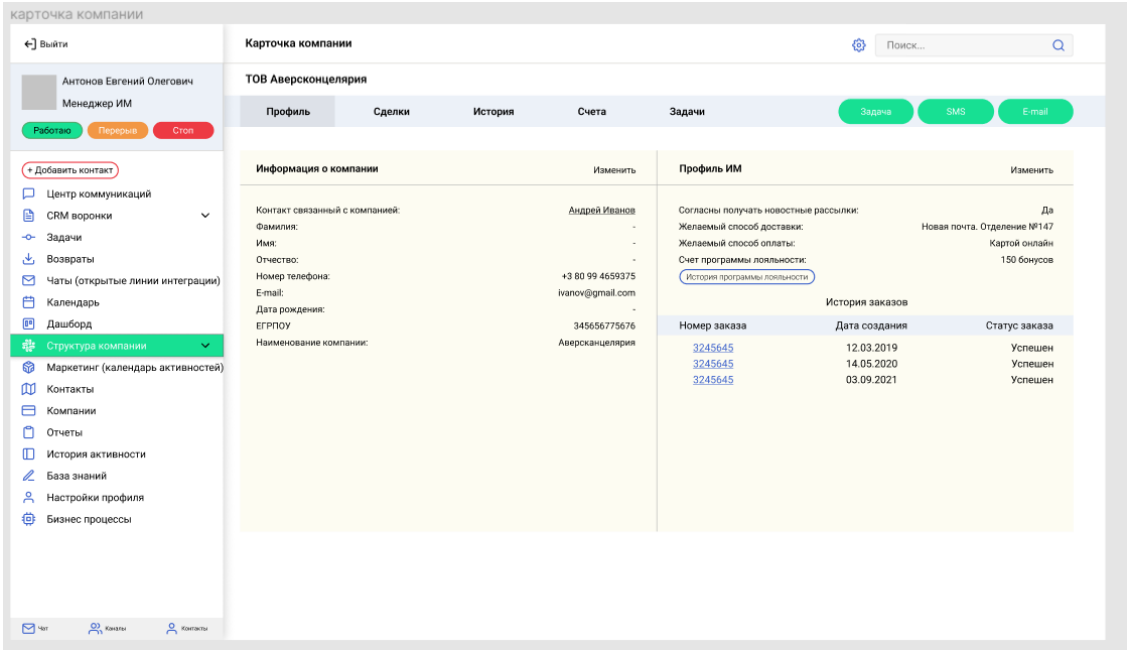


Рис. 2.27. Функціональний прототип панелі карточки компанії

2.1.23. Функціональний прототип панелі історії програми лояльності клієнтів

Для кожного контакту є своя історія програми лояльності. Вона містить усі бонуси які контакт використав або поповнив.

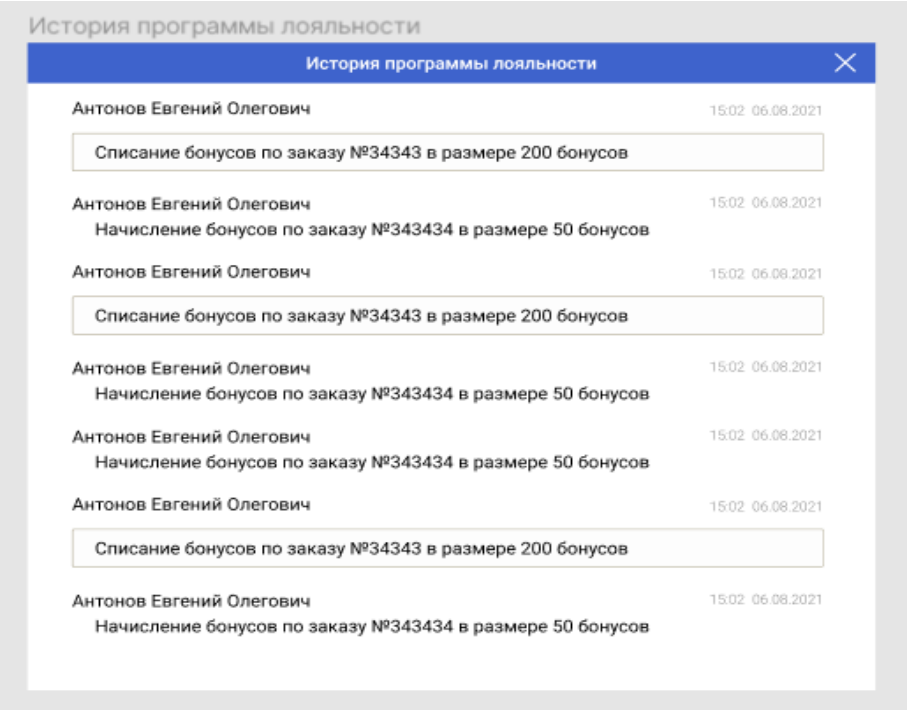


Рис. 2.28. Функціональний прототип історії програми лояльності клієнтів

2.2. Проектування бази даних системи керування взаємовідносинами з клієнтами

Оскільки система використовує технології відображення даних в реальному часі, для більшої швидкодії, було прийнято рішення, що окремі таблиці з даними, які використовуються у цих технологіях, мають зберігатися в окремих базах даних, щоб зменшити складність запитів *SQL*.

Окремі бази даних спроектовані так, щоб дані які в них зберігалися збігалися за смисловим значення.

2.2.1. Структура бази даних пов’язана з користувачем

База даних (рис. 2.29) має наступні таблиці:

- групи користувачів;
- атрибути користувачів;
- користувач;
- спеціальне ім’я користувача для чатів;
- токен запрошення користувача до системи;
- ролі користувачів;

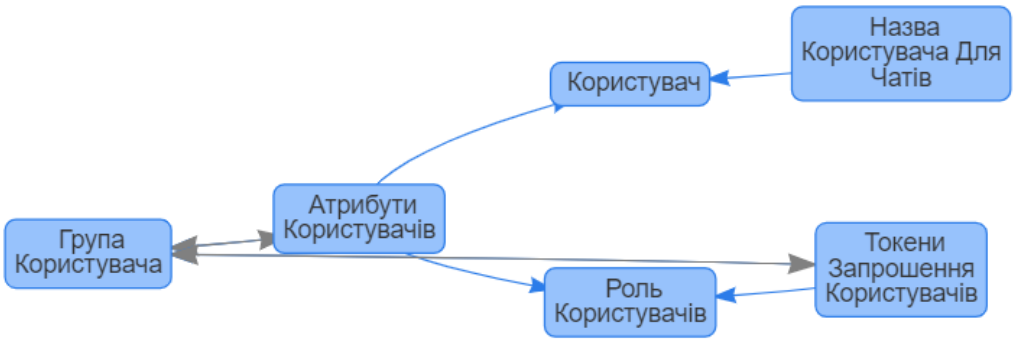
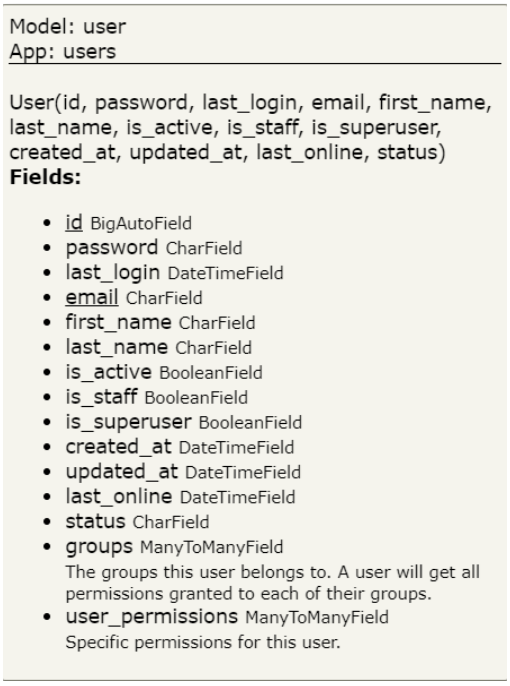


Рис. 2.29. Структурна схема бази даних користувач

Таблиця користувач(рис. 2.30) має наступні поля:

– унікальний номер користувача;
– пароль для авторизації;
– дата та час останнього входу до системи;
– ім’я;
– прізвище;
– дата створення акаунта ;
– дата оновлення акаунта;
– дата та час останньої присутності у системі;
– статус;
– приналежність до групи користувачів;
– рівні доступу в системі;


Model: user
App: users

User(id, password, last_login, email, first_name, last_name, is_active, is_staff, is_superuser, created_at, updated_at, last_online, status)
Fields:

- id BigAutoField
- password CharField
- last_login DateTimeField
- email CharField
- first_name CharField
- last_name CharField
- is_active BooleanField
- is_staff BooleanField
- is_superuser BooleanField
- created_at DateTimeField
- updated_at DateTimeField
- last_online DateTimeField
- status CharField
- groups ManyToManyField
  The groups this user belongs to. A user will get all permissions granted to each of their groups.
- user_permissions ManyToManyField
  Specific permissions for this user.

Рис. 2.30. Структура таблиці користувач

Таблиця атрибути користувача(рис. 2.31) має наступні поля:

– унікальний номер атрибутів користувачі;
– зв’язок з користувачем;
– по батькові;
– фото;
– дата народження;

– особистий телефон;
– робочий телефон;
– роль;
– групу;
– спеціальна група.

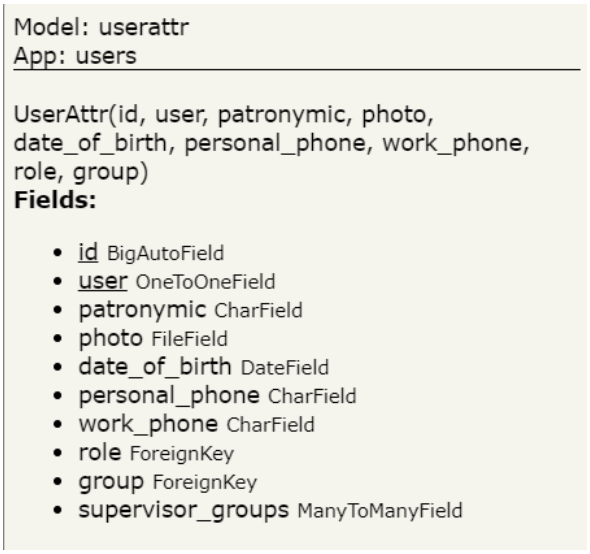


Рис. 2.31. Структура таблиці атрибути користувача

Таблиця токен запрошення користувача (рис. 2.32) має наступні поля:

– унікальний номер токену;
– електрона пошта;
– прізвище;
– ім’я;
– по батькові;
– дата народження;
– персональний номер телефону;
– робочий номер телефону;
– роль;
– група.

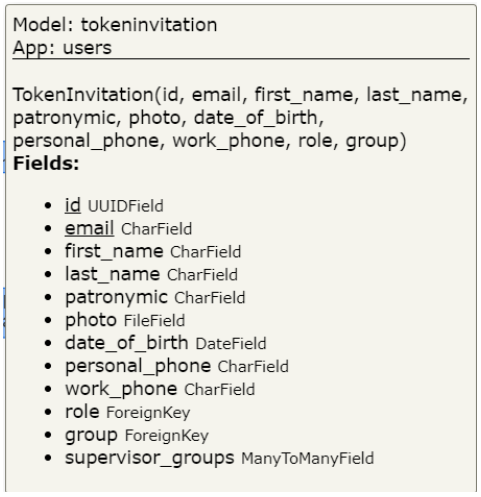


Рис. 2.32. Структура таблиці токен запрошення користувача

Таблиця токен група користувачів (рис. 2.33) має наступні поля:

– унікальний номер групи;

– назва групи.

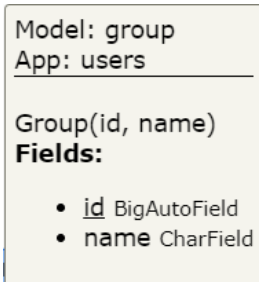


Рис. 2.33. Структура таблиці група користувачів

Таблиця ролі користувачів (рис. 2.34) має наступні поля:

– унікальний номер ролі;

– назва ролі.

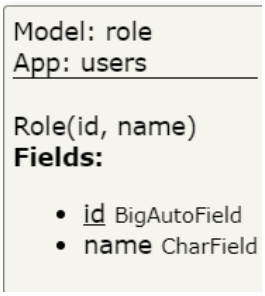


Рис. 2.34. Структура таблиці ролі користувачів

Таблиця ім’я користувача для чатів(рис. 2.35) має наступні поля:

– унікальний номер ролі;

– спеціальне ім’я користувача для чату;

– зв’язок з користувачем.

Рис. 2.35. Структура таблиці ім’я користувача для чатів

Структура таблиці активність користувача(рис. 2.36) має наступні поля:

– унікальний номер активності;

– назва активності;

– дата та час виконання активності;

– зв’язок з користувачем;

– зв’язок з таблицею якою користувач користувався.

Рис. 2.36. Структура таблиці активність користувача

2.2.2. Структура бази даних пов’язана з трекером часу

База даних трекеру часу (рис. 2.37) має наступні таблиці:

– загальний період часу;

– окремий період часу.

Рис. 2.37. Структурна схема бази даних таймеру часу

Таблиця загального період часу (рис. 2.38) має наступні поля:

– унікальний номер періоду часу;
– зв’язок з користувачем;
– ознака закінченості.

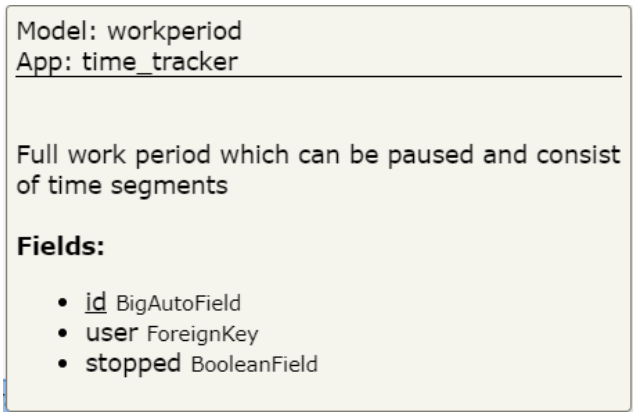


Рис. 2.38. Структура таблиці загального період часу

Таблиця окремого період часу (рис. 2.39) має наступні поля:

– унікальний номер періоду часу;
– зв’язок із загальним періодом часу;
– дата та час початку роботи;
– дата та час кінця роботи.

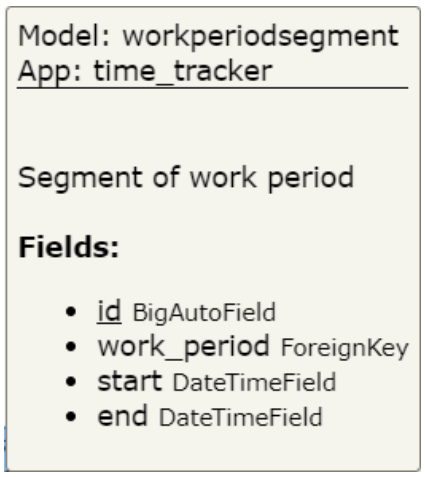


Рис. 2.39. Структура таблиці загального період часу

2.2.3. Структура бази даних повернених товарів

База даних повернених товарів (рис. 2.40) має наступні таблиці:

– контакт;

– повернення товару.

Рис. 2.40. Структурна схема бази даних повернення товару

Таблиця контакту (рис. 2.41) має наступні поля:

– унікальний номер контакту;

– ім’я;

– прізвище;

– по батькові;

– телефон;

– електрона пошта;

– дата народження;

– назва компанії;

– код ЄДРПОУ.

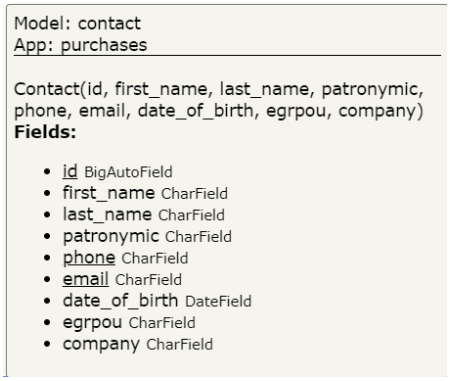


Рис. 2.41. Структура таблиці контакту

Таблиця повернення товару (рис. 2.42) має наступні поля:

– унікальний номер повернення товару;
– зв’язок з контактом;
– причина повернення;
– файл;
– коментар;
– дата;
– статус.

Рис. 2.42. Структура таблиці повернення товару

2.2.3. Структура бази даних угоди

База даних угоди (рис. 2.43) має наступні таблиці:

– сторінка угоди;
– стандартна угода;

– нестандартна угода.

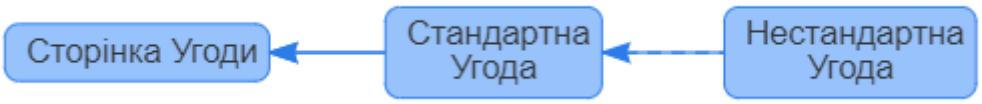


Рис. 2.43. Структурна схема бази даних угоди

Таблиця сторінка угоди (рис. 2.44) має наступні поля:

– унікальний номер сторінки угоди;

– зв’язок з базовою сторінкою;

– титульна назва.

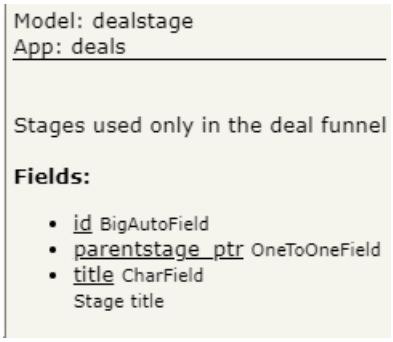


Рис. 2.44. Структура таблиці сторінка угоди

Таблиця стандартна угода (рис. 2.45) має наступні поля:

– унікальний номер стандартної угоди;

– зв’язок з клієнтом;

– зв’язок з батьківською сторінкою;

– унікальне значення з магазину;

– статус оплати ;

– тип оплати;

– загальний статус;

– назва компанії.

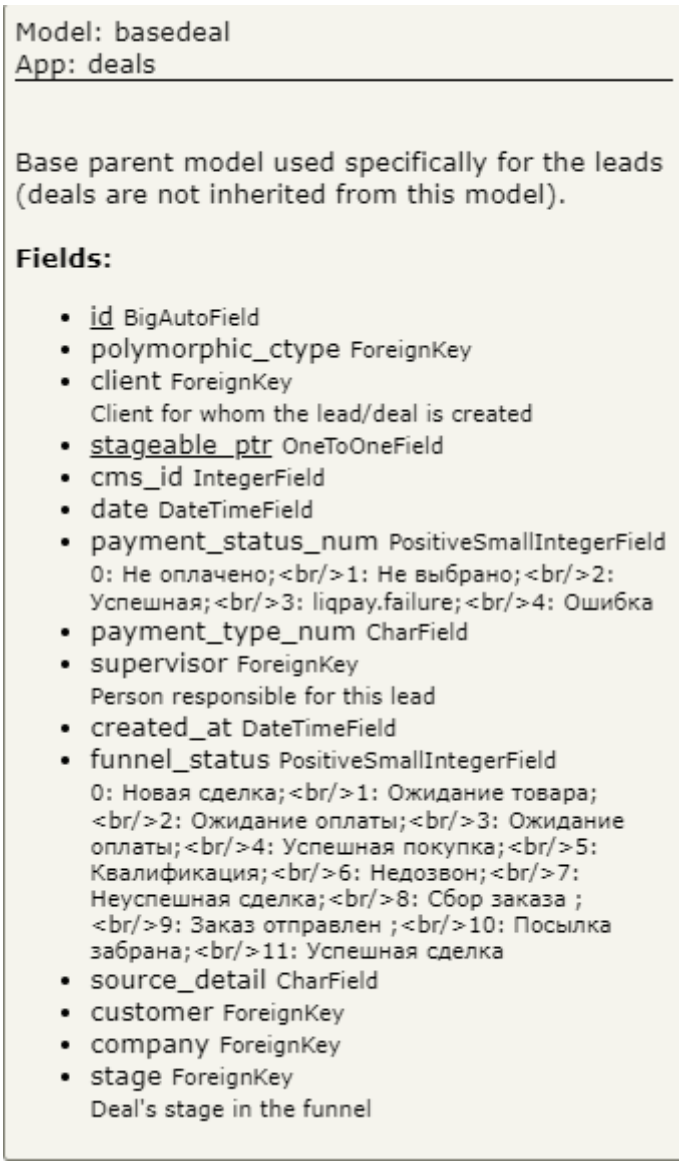


Рис. 2.45. Структура таблиці стандартна угода

Таблиця нестандартна угода (рис. 2.46) має наступні поля:

– унікальний номер нестандартної угоди;
– зв’язок з клієнтом;
– зв’язок з батьківською сторінкою;
– унікальне значення з магазину;
– статус оплати;
– тип оплати;
– загальний статус;
– назва компанії;
– опис нестандартної угоди.

Model: customdeal
App: deals

Lead with custom title created by manager

Fields:

- id BigAutoField
- polymorphic_ctype ForeignKey
- client ForeignKey
  Client for whom the lead/deal is created
- stageable_ptr OneToOneField
- cms_id IntegerField
- date DateTimeField
- payment_status_num PositiveSmallIntegerField
  0: Не оплачено;
1: Не выбрано;
2: Успешная;
3: liqpay.failure;
4: Ошибка
- payment_type_num CharField
- supervisor ForeignKey
  Person responsible for this lead
- created_at DateTimeField
- funnel_status PositiveSmallIntegerField
  0: Новая сделка;
1: Ожидание товара;
2: Ожидание оплаты;
3: Ожидание оплаты;
4: Успешная покупка;
5: Квалификация;
6: Недозвон;
7: Неуспешная сделка;
8: Сбор заказа ;
9: Заказ отправлен ;
10: Посылка забрана;
11: Успешная сделка
- source_detail CharField
- customer ForeignKey
- company ForeignKey
- stage ForeignKey
  Deal's stage in the funnel
- basedeal_ptr OneToOneField
- custom_title CharField
  Custom text

Рис. 2.46. Структура таблиці нестандартна угода

2.2.4. Структура бази даних компаній

База даних компаній (рис. 2.47) має наступні таблиці:

– компанія;

– папка;

– база знань;

– файли компанії.

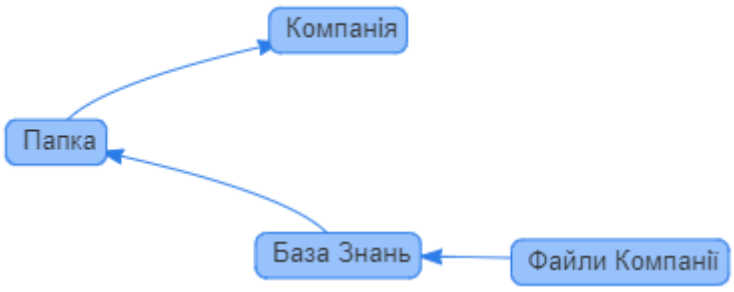


Рис. 2.47. Структурна схема бази даних компанії

Таблиця компанія (рис. 2.48) має наступні поля:

– унікальний номер компанії;
– контакт компанії;
– назва компанії;
– код ЄДРПОУ;
– співробітники.

Рис. 2.48. Структура таблиці компанія

Таблиця папка (рис. 2.49) має наступні поля:

– унікальний номер папки;
– назва папки;
– зв’язок з компанією.

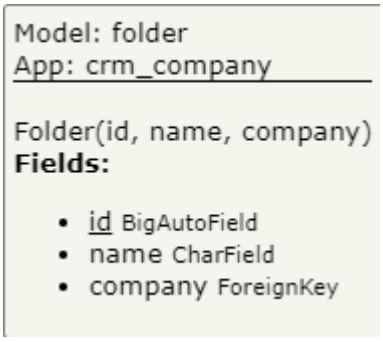


Рис. 2.49. Структура таблиці папка

Таблиця база знань (рис. 2.50) має наступні поля:

– унікальний номер бази знань;
– назва бази знань;
– опис;
– дата та час створення;

– зв’язок з папками;

– загальна кількість перегляду.

Рис. 2.50. Структура таблиці база знань

Таблиця файли компанії (рис. 2.51) має наступні поля:

– унікальний номер файлу;

– зв’язок з базою знань;

– назва;

– дата створення;

– посилання на файл.

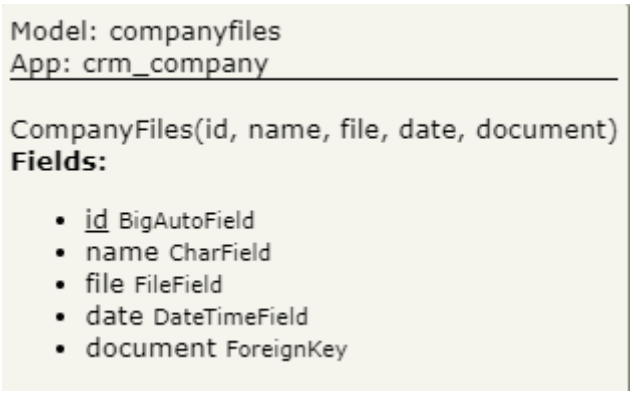


Рис. 2.51. Структура таблиці файли компанії

2.2.5. Структура бази даних компаній

База даних контактів (рис. 2.52) має наступні таблиці:

– контакт;
– телефон контакту;
– дзвінки;
– подія контакту;
– активність контакту;
– письмо контакту.

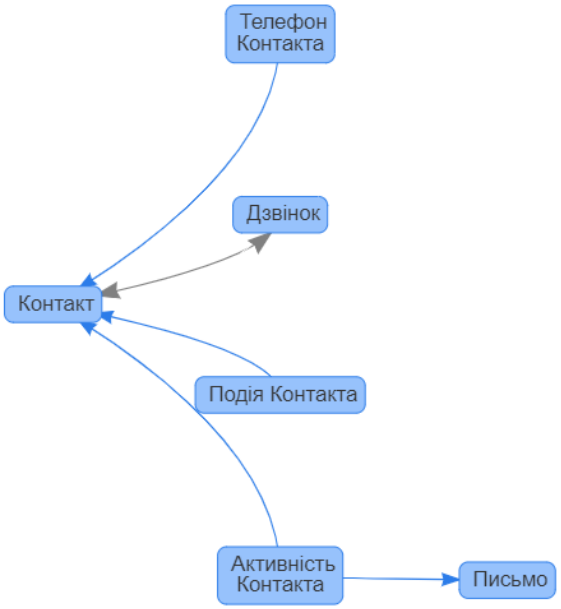


Рис. 2.52. Структурна схема бази даних контакту

Таблиця контакти (рис. 2.53) має наступні поля:

– унікальний номер контакту;
– ім’я;
– прізвище;
– по батькові;
– дата народження;
– електрона пошта;
– метод оплати;
– метод доставки;
– компанія доставки;
– ознака отримання розсилки новин.

Model: contact
App: contacts

Contact(id, first_name, last_name, patronymic, email, date_of_birth, receive_news, payment_method, delivery_method, delivery_company)
Fields:

- id BigAutoField
- first_name CharField
- last_name CharField
- patronymic CharField
- email CharField
- date_of_birth DateField
- receive_news BooleanField
- payment_method CharField
- delivery_method IntegerField
- delivery_company IntegerField

Рис. 2.53. Структура таблиці контакти

Таблиця дзвінок (рис. 2.54) має наступні поля:

– унікальне значення номеру дзвінка;

– тип дзвінка;

– номер телефону оператору;

– файл;

– тривалість дзвінку;

– номер телефону клієнту;

– зв’язок з контактом.

Model: call
App: contacts

Call(id, call_id, type, phone_number, file, duration, user_phone)
Fields:

- id BigAutoField
- call_id IntegerField
- type CharField
- phone_number CharField
- file FileField
- duration TimeField
- user_phone CharField
- contact ManyToManyField

Рис. 2.54. Структура таблиці дзвінки

Таблиця події контакту (рис. 2.55) має наступні поля:

– унікальне значення номеру події;
– тип події;
– дата та час події;
– зв’язок з клієнтом;
– номер дзвінка у системі.


Model: contactevents
App: contacts

ContactEvents(id, type, date, client, call_id)
Fields:

- id BigAutoField
- type CharField
- date DateTimeField
- client ForeignKey
- call_id IntegerField

Рис. 2.55. Структура таблиці події контакту

Таблиця активність контакту (рис. 2.56) має наступні поля:

– унікальне значення номеру активності;
– назва активності;
– тип активності;
– зв’язок з відповідальним за активність користувачем;
– дата та час активності;
– зв’язок з клієнтом;
– зв’язок з повідомленням.


Model: contactactivity
App: contacts

ContactActivity(id, event, type, responsible, date, client, message)
Fields:

- id BigAutoField
- event CharField
- type CharField
- responsible ForeignKey
- date DateTimeField
- client ForeignKey
- message ForeignKey

Рис. 2.56. Структура таблиці активності контакту

2.2.6. Структура бази даних чатів

База даних чатів (рис. 2.57) має наступні таблиці:

– повідомлення;
– файл повідомлення;
– назва чату.

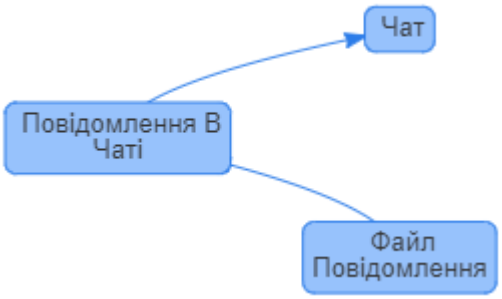


Рис. 2.57. Структурна схема бази даних чатів

Таблиця повідомлення (рис. 2.58) має наступні поля:

– унікальне значення номеру повідомлення;
– текст повідомлення;
– статус повідомлення;
– зв’язок з чатом за необхідності;
– зв’язок з користувачем;
– дата та час створення.

Model: message
App: chat

Message(id, text, status, chat, user, created_at)
Fields:

- id BigAutoField
- text TextField
- status CharField
- chat ForeignKey
- user ForeignKey
- created_at DateTimeField

Рис. 2.58. Структура таблиці повідомлення

Таблиця файлів повідомлення (рис. 2.59) має наступні поля:

– унікальне значення файлу повідомлення;
– зв’язок з повідомленням;
– файл.

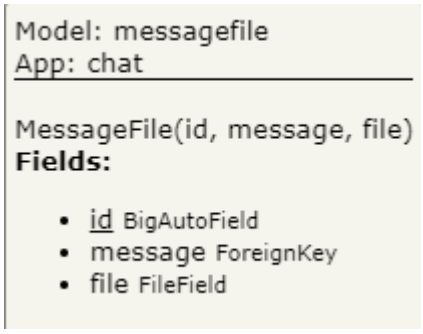


Рис. 2.59. Структура таблиці файл повідомлення

Таблиця файлів повідомлення (рис. 2.60) має наступні поля:

– унікальне значення номеру чату;
– спеціальна назва чату;
– загальна назва чату;
– зв’язок з користувачем який створив чат;
– зв’язок з адміністраторами ;
– зв’язок з користувачами.

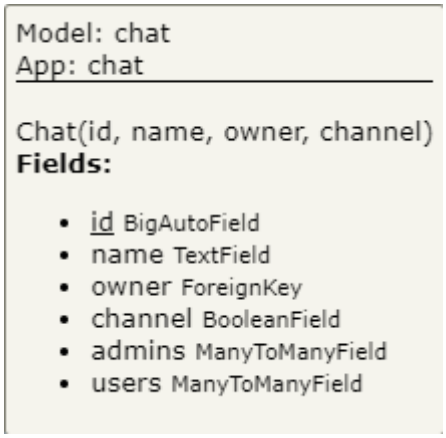


Рис. 2.60. Структура таблиці чат

2.2.6. Структура основної бази даних

Загальна база дана включає в себе наступні таблиці:

– повідомлення;
– коментарі;
– дані аналітики.

Таблиця повідомлення (рис. 2.61) має наступні поля:
– унікальне значення номеру повідомлення;
– назва повідомлення;
– дата та час;
– ім’я користувача;
– прізвище користувача;
– ознака перегляду;;
– опис
– зв’язок з користувачем.

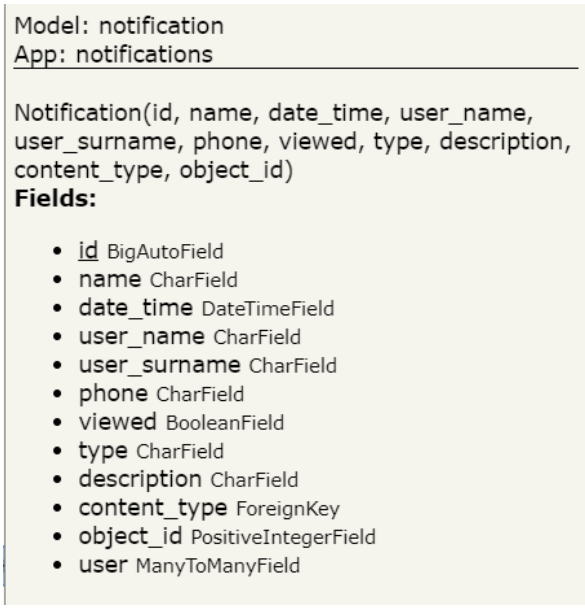


Рис. 2.61. Структура таблиці повідомлення

Таблиця коментарі (рис. 2.62) має наступні поля:
– унікальне значення номеру коментаря;
– опис;
– зв’язок з користувачем;
– файл;

– дата та час створення.

Model: comment
App: comments

Comment(id, description, user, date, file, content_type, object_id)
Fields:

- id BigAutoField
- description TextField
- user ForeignKey
- date DateTimeField
- file FileField
- content_type ForeignKey
- object_id PositiveIntegerField

Рис. 2.62. Структура таблиці коментарі

Таблиця дані аналітики (рис. 2.63) має наступні поля:

– дані аналітики.

Model: metrics
App: dashboard

Metrics(id, request, total_count_users, count_users, total_count_session, consersion_purchases, consersion_products, roi, abandoned_carts, revenue_per_transaction)
Fields:

- id BigAutoField
- request JSONField
- total_count_users IntegerField
- count_users IntegerField
- total_count_session IntegerField
- consersion_purchases FloatField
- consersion_products FloatField
- roi FloatField
- abandoned_carts FloatField
- revenue_per_transaction FloatField

Рис. 2.63. Структура таблиці коментарі

2.2.7. Загальна структурна схема баз даних системи керування взаємовідносинами з клієнтами

В результаті проектування окремих баз даних, була спроектована система усіх баз даних системи керування взаємодії з клієнтами, яка представлена на рис. 2.64. В даній структурній схемі описуються таблиці та їх зв’язки з іншими таблицями. Окремим кольором визначена кожна база даних та її зв’язки.

Загальна кількість таблиць даних складає 52, загальна кількість баз даних 13. Для реалізації такої системи використовуються різні види бази даних:

– реляційні;
– нереляційні.

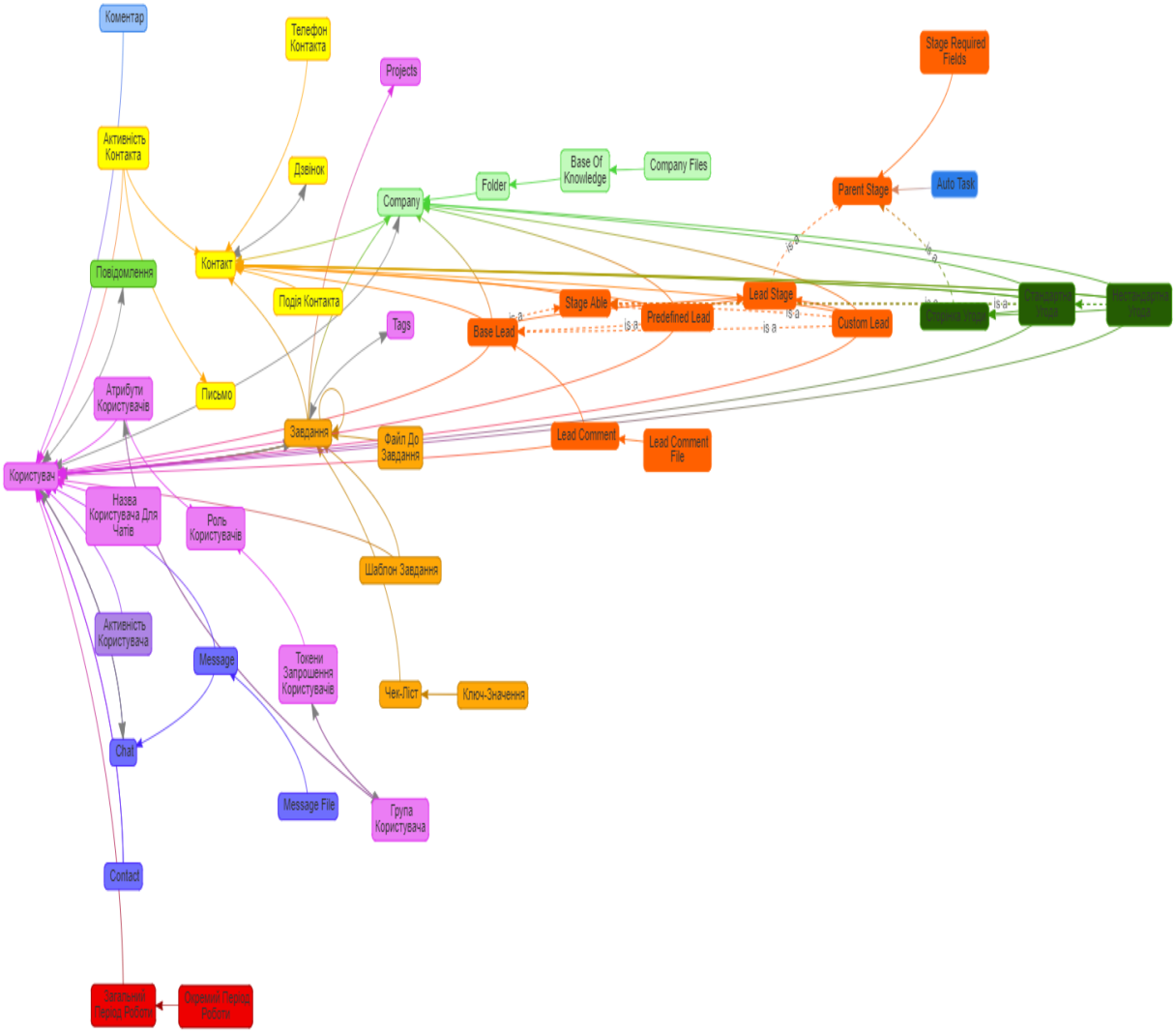


Рис. 2.64. Загальна структурна схема баз даних

2.3. Висновки до розділу

Для проектування структури системи керування взаємовідносинами з клієнтами було обрано два методи опису структури:

- функціональний прототип кожної складової системи;
- структурний опис кожної моделі бази даних системи.

Функціональний прототип — це найпростіший робочий прототип, створений для перевірки дизайну продукту. Крім того, це невід'ємна частина процесу складання, тестування, пілотування та дослідження ринку, який оцінює дизайн, матеріали, міцність, допуски, збірку, робочі механізми, технологічність. Функціональні прототипи, виготовлені за допомогою кількох методів створення прототипів і матеріалів інженерного рівня, є альтернативою готовим продуктам, що дозволяє перевіряти та тестувати форму, придатність і функціональність у екстремальних умовах роботи для покращення дизайну.

Структурний опис даних — це спеціалізований формат для опису, обробки, пошуку та зберігання даних. Структурний опис даних дозволяє виділити потрібні данні і процеси роботи з ними відповідним чином. Головне, структурний опис даних створює структуру інформації, щоб машини та люди могли її краще зрозуміти. Кожний структурний опис даних містить інформацію про значення даних, зв'язки між даними та, у деяких випадках, функції, які можна застосувати до даних.

В даному розділі описані усі процеси проектування функціонального прототипу та структурного опису баз даних.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ ДОДАТКУ

3.1. Створення проекту

Для контролю версій бібліотек проекту використовується віртуальне оточення *pipenv*.

Список зовнішніх залежностей міститься у файлу *Pipfile* та *Pipfile.lock*. *Pipfile* містить інформацію про назву бібліотеки та її версію, а *Pipfile.lock* містить сценарій завантаження бібліотек на пристрій. Вміст файлу *Pipfile* наведений на рис. 3.1.

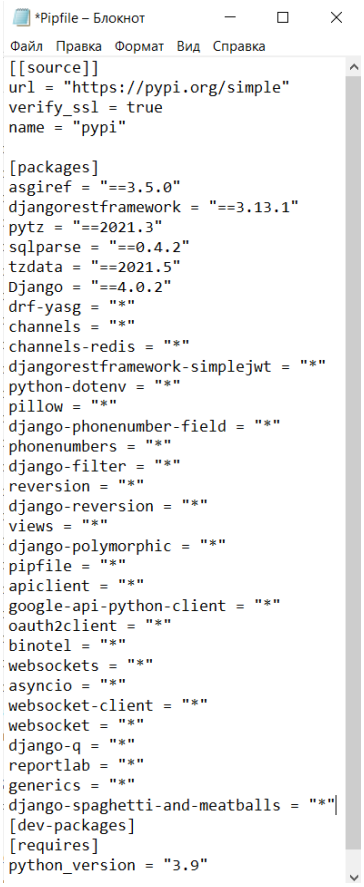


Рис. 3.1. Перелік необхідних бібліотек для роботи додатку

Запуск додатку починається з команди :

python manage.py runserver

Також використовується система контролю версій *Git*, для цього створюється файл *.gitignore* у який заносяться типи файлів що не будуть відслідковуватися, для забезпечення конфіденційності та безпеки додатку. Вміст файлу зображений на рис. 3.2.

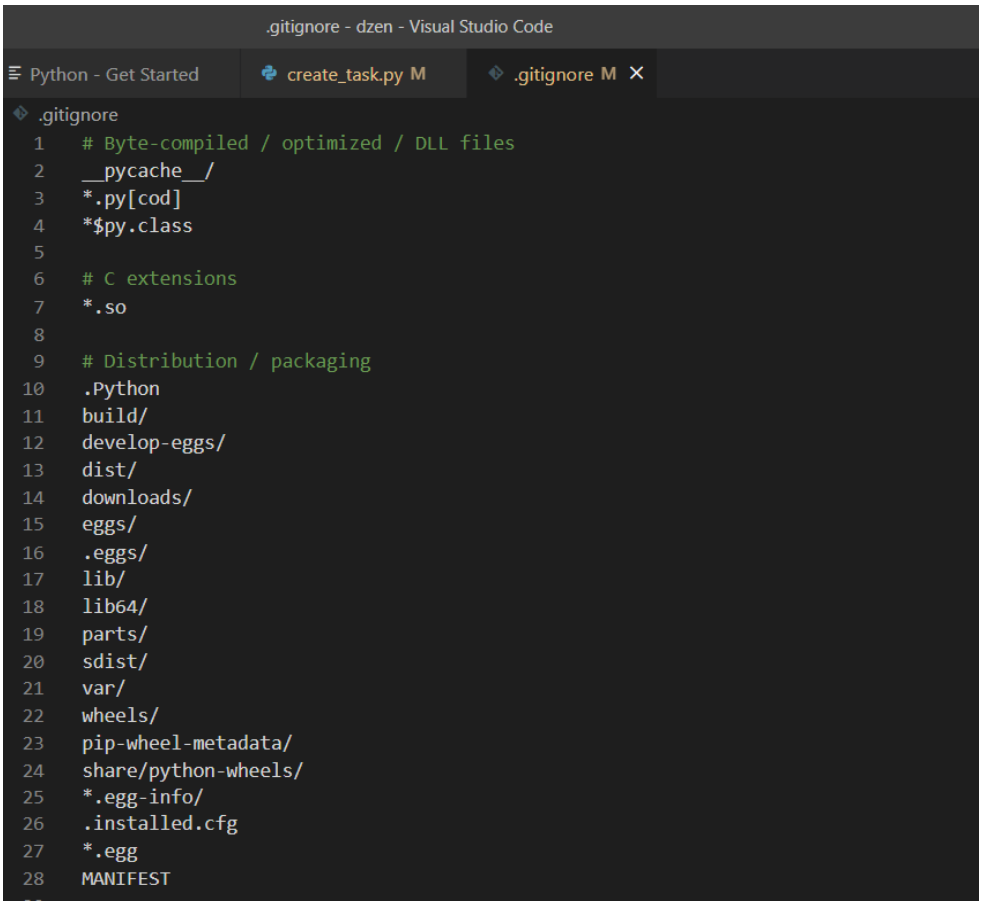


Рис. 3.2. Вміст файлу *.gitignore*

При реалізації додатку використовується модульну систему. Увесь проект поділено на програмні сутності за смисловим значенням:

– задачі;
– дзвінки;
– календарі;
– чати;
– коментарі;
– контакти;

– база знань компаній;
– дошка аналітики;
– угоди;
– ліди;
– повідомлення;
– акції;
– користувачі;
– активність користувача;
– вимірювач робочого часу.

В якості фреймворку при розробці проекту використовується *Django*. *Django* — це бібліотека *Python*, яка полегшує створення веб-сайтів за допомогою *Python*. *Django* дотримується шаблону проектування *MVT* (*Model View Template*).

Модель – дані, які ви хочете представити, зазвичай дані з бази даних.

Перегляд – обробник запитів, який повертає відповідний шаблон і вміст на основі запиту користувача.

Шаблон – текстовий файл ,наприклад, файл *HTML*, що містить макет веб-сторінки, з логікою того, як відображати дані.

REST — це архітектурний стиль написання веб-додатків, який визначає набір правил, які використовуватимуться для створення веб-служб. «Веб-служби, які відповідають архітектурному стилю *REST*, відомі як веб-служби *RESTful*[2]».

Це дозволяє запитуючим системам отримувати доступ до веб-ресурсів і маніпулювати ними за допомогою єдиного попередньо визначеного набору правил. Взаємодія в системах на основі *REST* відбувається через Інтернет-протокол передачі гіпертексту.

REST архітектуру додатку дозволяє реалізувати одне з *Django, Django Rest Framework.*

Одним з основних інструментів, який *Django Rest Framework* використовує для побудови *API,* є серіалізатор. «Серіалізатори дозволяють

складним даним, таким як набори запитів і екземпляри моделей, перетворюватися на рідні типи даних *Python*, які потім можна легко відобразити в *JSON, XML* або інші типи вмісту[1]». Серіалізатори також забезпечують десеріалізацію, дозволяючи аналізованим даним перетворюватися назад у складні типи після попередньої перевірки вхідних даних.

3.2 Реалізація користувачів

Django вже містить базове визначення моделі користувача з основним переліком атрибутів. Але ця модель зв’язана з функцією авторизації через ім’я користувача та його пароль. Оскільки в системі першочергове створення користувачів вимагає лише пошти, треба перевизначити абстрактну модель користувача. Щоб нестандартна модель користувача працювала треба обов’язково визначити два методи:

- метод створення користувача;
- метод створення користувача адміністратора.

Метод створення користувача повинен отримувати на вхід тільки пошту користувача, оскільки після запису пошти в базі даних, буде відправлена повідомлення з посиланням на анкету, яку потрібно заповнити та після цього отримати доступ до системи. Загальний алгоритм створення користувача у системі представлена на рис .3.3. Цей алгоритм забезпечує ізольованість системи від сторонніх користувачів. Значення пошти отримує адміністратор компанії який вже є у системі, або після розмови зі службою підтримки, дані вносяться особисто адміністратором системи. Також в базі даних є доповнення інформації о користувачі.

Автентифікація використовує *jwt-token*. *JWT* або *JSON token* найчастіше використовуються для ідентифікації автентифікованого користувача. Вони видаються сервером автентифікації та споживаються клієнт-сервером (для захисту його *API*).

Рис. 3.3. Алгоритм створення користувача у системі

3.3. Реалізація активності користувачів

Активність користувача – це фіксація дій користувача у системі. Для цього використовуються сигнали *Django*. Сигнал *Django* – це, умовна точка входу іншої функції у тілі іншої. У даному випадку сигнал розміщується в усіх запитах що робить користувач.

У тілі сигналу є функція, що робить запис в базу даних запис. В даному випадку використовується технологія виведення даних в режимі реального часу за допомогою *Websockets.*

WebSocket є двонаправленим, повнодуплексним протоколом, який використовується в сценарії зв'язку клієнт-сервер, на відміну від *HTTP*, він починається з *ws://* або *wss://.* Це протокол із збереженням стану, що означає, що з'єднання між клієнтом і сервером буде підтримуватися, доки його не буде розірвано однією зі сторін. Після закриття з'єднання клієнтом і сервером з'єднання припиняється з обох сторін.

Коли з'єднання встановлено та живе, зв'язок відбувається за тим самим каналом з'єднання, доки його не буде розірвано.

Ось як після встановлення зв'язку клієнт-сервер клієнт-сервер вирішить нове з'єднання, щоб зберегти його, це нове з'єднання буде відоме як WebSocket. Після встановлення каналу зв'язку та відкриття з'єднання обмін повідомленнями відбуватиметься в двонаправленому режимі, доки не буде збережено з'єднання між клієнтом і сервером. Якщо або клієнт, або сервер вимикається або вирішує закрити з'єднання, закривається обома сторонами. Спосіб роботи *socket* дещо відрізняється від того, як працює *HTTP*, код статусу 101 позначає протокол комутації в *WebSocket*.

З'єднання *Websockets* використовується для виведення інформації про активність користувача. Загальний алгоритм роботи активності користувача наведено на рис. 3.4.

Виведення інформації працює в двох напрямках:

- виведення самої інформації на панель активності користувача;
- відправлення сигналу повідомлення про наявність нової інформації.

Рис. 3.4. Алгоритм роботи активності користувача

3.4. Реалізація панелі дзвінків

Для фіксації усіх дзвінків компанії використовується додаткова служба *Binotel Calls*. *Binotel* – провідний оператор ділового зв'язку в Україні. 8,2 тис. компаній використовують продукти *Binotel* у щоденній роботі, 1,5 млн людей щодня спілкуються через послуги *Binotel*, 32 млн телефонних номерів підключено через АТС *Binotel.*

Binotel має інтеграцію клієнтів у свою мережу. У обмежених випадках, коли *Binotel* діє як контролер, він може використовувати певний мінімум персональних даних для таких цілей:

– реєстрація та адміністрування облікового запису;

– усунення несправностей і вдосконалення програми.

Більшість операцій з обробки відбувається через використання додатку, *websockets* та *api* клієнтами.

Binotel може збирати наступні категорії персональних даних про клієнтів:

– біографічні дані, такі як ім’я, адреса електронної пошти, номер телефону;

– інформація, пов’язана з дзвінками, наприклад ім’я або номер телефону абонента, ім’я або номер телефону абонента, час очікування, тривалість дзвінка;

– час дзвінка та його запис;

– будь-які персональні дані, які можуть міститися в записах дзвінків.

Binotel захищає персональні дані від втрати, неправильного використання або несанкціонованого доступу. Використовується різноманітні засоби захисту, такі як шифрування, контроль цифрового та фізичного доступу, угоди про нерозголошення та інші технічні та організаційні заходи для захисту персональних даних.

Після створення угоди про співпрацю з *Binotel*, вони відправляють адресу *websockets*, ключ та секретний пароль. Далі за допомогою бібліотек для відправки запитів *requests* та бібліотеки асинхронного програмування *asyncio*, створюється програма, яка постійно відправляє запити до каналу *Binotel* та отримує дані. Для цього використовується наступний код:

```
message = json.dumps({
    'task': 'authLikeService',
     'key': 'da4125-e036da5',
     'secret': '2c27e8-e0f81c-7de53c-89e232-e3ad4e79'
})
async def main():
   async with websockets.connect('wss://ws.binotel.com:9002') as client
      await client.send(message)
```

```
data = json.loads(await client.recv())
print(data)
await client.send(message)
data = json.loads(await client.recv())
print(data)
while True:
    msg = await client.recv()
    print(msg)
    try:
        parse_call(msg)
    except SynchronousOnlyOperation:
        pass
```

Отримані дані записуються в базу даних та за допомогою *websockets* відображаються на панелі. Загальний алгоритм роботи панелі дзвінків представлено на рис. 3.5.

Рис. 3.5. Алгоритм роботи панелі дзвінків

3.5. Реалізація панелі календарів

Для виведення інформації у вигляді календаря, необхідно зв’язати дані з датою та часом цих даних та відфільтрувати їх за ними. Для фільтрації даних використовується *django_filters.rest_framework* . Це модуль бібліотеки, який дозволяє фільтрувати дані. Також цей модуль вже має створений фільтр за часом, він має наступні параметри:

- сьогодні;
- учора;
- поточний тиждень;
- поточний місяць;
- поточний рік.

Отож загальний алгоритм роботи генерації календарю для даних представлений на рис. 3.6. У разі відсутності в даних дати та часу, повертає повідомлення з неможливістю.

Рис. 3.6. Алгоритм роботи генерації календарю

3.6. Реалізація чатів

Система чатів реалізується за допомогою бібліотеки *Django-channels*. Ця бібліотека дозволяє створювати власні *websockets.* Для цього необхідно виділити окрему нереляційну базу даних, в даному випадку *Redis*.

Redis часто називають сервером структур даних. Це означає, що *Redis* надає доступ до змінних структур даних за допомогою набору команд, які надсилаються за допомогою моделі сервер-клієнт із *TCP*-сокетами та простим протоколом. Таким чином, різні процеси можуть запитувати та змінювати ті самі структури даних спільним способом.

Структури даних, реалізовані в *Redis*, мають кілька спеціальних властивостей:

– *Redis* піклується про те, щоб зберігати їх на диску, навіть якщо вони завжди обслуговуються та змінюються в пам’яті сервера. Це означає, що Redis швидкий, але він також є енергонезалежним.

– Реалізація структур даних підкреслює ефективність пам'яті, тому структури даних у *Redis*, ймовірно, використовуватимуть менше пам’яті порівняно з тією самою структурою даних, змодельованою за допомогою мови програмування високого рівня.

– *Redis* пропонує низку функцій, які природно знайти в базі даних, як-от реплікація, регульовані рівні довговічності, кластеризація та висока доступність.

Іншим хорошим прикладом є уявлення про *Redis* як про більш складну версію *memcached*, де операції — це не просто *SET* і *GET*, а операції, які працюють зі складними типами даних, такими як списки, набори, упорядковані структури даних тощо.

Після підключення бібліотеки та бази даних *Redis*, залишилося створити динамічний приклад каналу зв’язку. Для цього треба необхідно визначити декілька обов’язкових методів:

– функція підключення;

– функція обробки повідомлення;
– функція відключення;

Також створено додаткові функції:

– функція відправлення повідомлення
– функція відображення повідомлення
– зміна формату повідомлення до *json.*

Кожен чат має своє особисте ім’я, яке зберігається в базі даних. Іноді ім’я чату складається з назви користувача та унікального ключу, іноді з комбінації назв користувачів, що приєднуються до нього.

Для того щоб приєднатися то чату треба або перейти за посиланням до нього, або перейти на панель входу до чату (рис. 3.7).

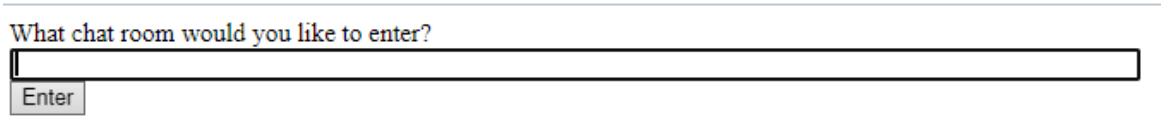


Рис. 3.7. Панель входу до чату

3.7. Реалізація коментарів

При створенні коментарів використовувалося принцип динамічного зв’язку з таблицями інших даних. Оскільки коментарі можливо залишати в більшості об’єктів, а створювати окремі таблиці для кожного об’єкту нераціонально, було прийнято рішення використовувати динамічний зв’язок *GenericForeignKey.*

GenericForeignKey дозволяє створити таблицю, яка в полях має значення номеру таблиці бази даних, та номер об’єкту в цієї базі даних. Тим сам було реалізовано процес зв’язку однієї таблиці коментарів з усіма можливими таблицями в інших базах даних.

Також при реалізації серіалізаторів для коментарів було створено нестандартний серіалізатор, який виключає доступ до інформації стосовно номерів баз даних. Код серіалізатору наведений нижче:

class CommentSerializer(serializers.ModelSerializer):

```
class Meta:
    model = Comment
    read_only_fields=('content_type', 'object_id')
    fields = '__all__'
```

3.8. Реалізація контактів

Кожний клієнт компанії це контакт. У системі можна знайти особисту інформацію контакту, а також його останні дії та замовлення. Відстеження дій контактів аналогічний процес відстеженню активності користувачів. А процес відстеження замовлень клієнтів, це запит до магазину компанії, стосовно замовлень, та їх відображення. З сервісу магазина компанії приходить файл зі списком замовлень за певний період часу, далі йде аналіз файлу, в якому дані розбиваються за значенням та записуються до бази даних.

Також реалізовані декілька методів представлення для контактів:

- вивід усіх контактів
- вивід активності контактів
- вивід задач контактів

Методи представлення в *Django rest framework* використовуються стандартний алгоритм обробки даних: на вхід функція приймає аргумент *request,* цей аргумент визначає сукупність усіх параметрів, що користувач відправляє до серверу, далі визначається запит до бази даних та серіалізатор який буде оброблювати та коригувати цей запит. У результаті функція представлення повертає набір даних у форматі *json* або *xml*.

Для прикладу представлено код функції виведення усіх контактів у системі:

```
class ContactViewSet(ModelViewSet):
    queryset = Contact.objects.all()
    serializer_class = ContactSerializer
    permission_classes = [permissions.IsAuthenticated]
```

Так в представленні вказано запрограмовані рівні доступу до ресурсу, в даному випадку якщо користувач автентифікован він може виконати запит або перейшовши за посиланням або знаючи спеціальний *url.*

Django також дозволяє створити особисті *url* та запрограмувати на них представлення. В даному проекті використовуються *rest_framework.routers.* Роути це один з видів представлення *url* у *Django,* вони приймають адресу *url*, назву підпрограми та функцію представлення. Код програмування *url* контактів представлено нижче:

```
from django.urls import include, path
from rest_framework.routers import DefaultRouter
from . import views
router = DefaultRouter()
router.register('contact', views.ContactViewSet, basename='contact')
router.register('company', views.CompanyView, basename='company')
router.register('contact_tasks',views.ContactTasksViewSet,
basename='contact_activity')
router.register('activity',views.ContactActivityViewSet,
basename='contact_activity')
app_name = "contacts"
urlpatterns = [
   path("", include(router.urls))
]
```

Даний код демонструє 4 *url* з відповідними представленнями. Також оскільки увесь проект поділений на програмні сутності-додатки, для цього додатку визначений свій *url – contacts*.

3.9 Реалізація дошки аналітики

У цьому розділі розроблено панель аналітики компанії. Аналітичні дані беруться за допомогою інтеграції сервісу *Google Analytics Report API.* В основі

Google Analytics є система збору даних. *Reporting API* версії 4 дозволяє отримати доступ до цієї інформації. Нижче наведено можливості цього інтерфейсу:

– вирази для показників;

– *API* дозволяє отримати як вбудовані показники, так й поєднання показників, виражені з допомогою математичних операцій;

– діапазони дат;

– *API* дозволяє за допомогою одного запиту отримати дані за два діапазони дат;

– когорти та загальну цінність;

– *API* забезпечує широкі можливості створення звітів щодо когортів та загальної цінності;

– підтримка кількох сегментів;

– *API* дозволяє за допомогою одного запиту отримати декілька сегментів.

Для використання цього сервісу потрібно мати профіль *Google*. Після цього створити свій проект у розширенні *Google Analytics.* Далі сервіс надасть код, який треба додати до шаблону додатку в якому цікавить аналітика, та файл з паролями та ключами для запиту до *API.*

Було створено функції асинхронного запиту до ресурсу та обробки наданих даних:

```
def get_report(start_date, end_date):
    print(start_date, end_date)
    metrics_names = []
    credentials = ServiceAccountCredentials.from_json_keyfile_name(location, ['https://www.googleapis.com/auth/analytics.readonly'])
    body = {'reportRequests': [{'viewId':VIEW_ID,
                                'dateRanges': [{'startDate': start_date, 'endDate': end_date}],
                                'metrics': [{'expression': 'ga:users'},
```

```
                    {"expression": "ga:sessionsPerUser"},
                    {"expression": "ga:totalValue"},
                    {"expression": "ga:transactions"},
                    {"expression": "ga:transactionsPerUser"},
                    {"expression": "ga:productRevenuePerPurchase"},
                    {"expression": "ga:adCost"},
                    {"expression": "ga:transactionRevenue"},
                    {"expression": "ga:productAddsToCart"},
                    {"expression": "ga:revenuePerTransaction"},
                    ],
                'dimensions': [{'name': 'ga:yearMonth'}],
            }]}
    http = credentials.authorize(httplib2.Http())
    service = build('analyticsreporting', 'v4', credentials=credentials)
    response = service.reports().batchGet(body=body).execute()
```

У даному випадку цікавлять окремі показники:

- кількість клієнтів;
- кількість сесій кожного клієнту;
- загальна кількість сесій;
- загальна кількість транзакцій;
- кількість транзакцій на кожного користувача;
- суми покупки товарів;
- кількість додання до корзини.

Після отримання показників, також розраховується коефіцієнт рентабельності інвестицій, коефіцієнт невдалих покупок та кількість транзакцій на кожного користувача.

Усі показники записуються у базу даних та виводяться по запиту користувача. Для успішного запиту, треба в його тілі вказати період часу за який треба отримати показники, наприклад:

http://127.0.0.1:8000/dashboard/metrics/?start_date=2022-03-23&end_date=2022-03-23

3.10. Реалізація угод

Інформацію стосовно усіх угод та їх статусу надає магазин компанії в якому цей функціонал інтегровано. За допомогою *API* магазину, створюється запит та функція обробки. Дана функція створює запис у базі даних та застосовує фільтри.

Також реалізована *Kanban* система. *Kanban* — це популярний метод економічного керування робочим процесом для визначення, управління та вдосконалення послуг, які забезпечують роботу інформації. Це допомагає візуалізувати роботу, підвищити ефективність і постійно вдосконалюватися. Робота представлена на дошках *Kanban*, дозволяє оптимізувати виконання роботи кількома командами та обробляти навіть найскладніші проекти в єдиному середовищі.

Kanban пропонує гнучкість використання методу поверх існуючих робочих процесів, систем і процесів, не порушуючи те, що вже є. Метод визнає, що існуючі процеси, ролі, обов’язки та звання мають цінність і, як правило, варті збереження. Він висвітлює проблеми, які необхідно вирішити, і допоможе оцінити та спланувати зміни, щоб їх впровадження було якомога менш руйнівним. У центрі кожної організації має стояти надання цінності клієнту. Розуміння потреб і очікувань клієнтів привертає увагу до якості наданих послуг і вартості, яку вони створюють.

Для успішного впровадження необхідно мати шість основних принципів:

- візуалізація робочого процесу;
- обмеження незакінченого процесу;
- керування потоками даних;
- чіткі правила обробки процесу;
- цикли зворотного зв’язку;

– можливість співпраці над єдиним процесом.

Починаючи реалізацію, створено функцію запиту до *API* за інформацією:

import requests

from requests.auth import HTTPBasicAuth

basic = HTTPBasicAuth('admin@pm-dt.com', 'admin')

result = requests.get(f'https://snapt.pmapplications.com/api/order/all/', auth=basic)

result1= result.json()

orders_list = result1['results']

Після цього усі угоди діляться за статусом та відображаються групами. Також створено фільтри, що обрати окремі угоди за статусом або датою та часом.

3.11. Реалізація контролю часу роботи

Для кожного працівника створюється за допомогою *websockets* спеціальний таймер. Він потрібен щоб користувач вмикав його під час роботи. Таймер поділений на сектора, щоб працівник мав можливість ставити на паузу. Під кінець робочого усі сектора складаються та записуються до бази даних.

Інформацію про працівника та його активність можливо отримати за запитом. Також створено фільтри для відображення графіків роботи працівника на різних часових проміжках.

За роботою таймеру проведено unit-тестування. Створено наступні функції перевірки працездатності:

– функція зупинки таймеру;
– функція старту таймеру;
– функція паузи;
– функція створення таймеру;
– функція створення сегменту часу.

Усіх функції тестування успішно пройшли перевірку.

3.12. Реалізація задач

При розробці бази даних використовується метод написання запитів для об’єктно-реляційних моделей. Цей метод використовує опис таблиці бази даних у вигляді класу з аргументами та функціями. У результаті ці класи перекладаються на мову *SQl.*

Приклад створення моделі таблиці бази даних наведено нижче:

```
class Task(models.Model):
    name = models.CharField(max_length=250, verbose_name='Название')
    description = models.TextField(verbose_name='Описание', blank=True)
    responsible = models.ManyToManyField(User, related_name='responsible')
    date = models.DateField(verbose_name='Запланировано')
    create_date = models.DateTimeField(auto_now_add=True)
    priority = models.IntegerField(verbose_name='Приоритет')
    project = models.ForeignKey(Projects, blank=True, null=True, on_delete=models.SET_NULL)
    remind = models.BooleanField(default=False)
    remind_description = models.CharField(max_length=250, null=True, blank=True)
    repeated_task = models.BooleanField(default=False)
    repeated_task_days = models.PositiveIntegerField(blank=True, null=True)
    related_task = models.ForeignKey('self', null=True, blank=True, on_delete=models.SET_NULL)
    tags = models.ManyToManyField(Tags)
    related_people = models.ManyToManyField(User, related_name='related_people', blank=True)
```

created_by = models.ForeignKey(User, on_delete=models.CASCADE, related_name='created_by')

updated_by = models.ForeignKey(User, blank=True, null=True, on_delete=models.CASCADE, related_name='updated_by')

files = models.FileField(verbose_name='Файл', blank=True, null=True)

notification = GenericRelation(Notification)

finished = models.BooleanField(default=False)

last_update = models.DateTimeField(auto_now=True)

company = models.ForeignKey(Company, on_delete=models.CASCADE, null=True, blank=True, related_name='company_tasks')

contact = models.ForeignKey(Contact, on_delete=models.CASCADE, null=True, blank=True, related_name='contact_tasks')

def __str__(self):

return self.name

Також в атрибутах задачі вказано, що можна поділитеся нею з іншими користувачами. Цей метод реалізовано завдяки сигналам. Під час створення задачі та вказівки користувачів, серед яких треба поширити цю задачу, функція сигналу отримує унікальні значення працівників. Приклад створення сигналу наведено нижче:

from django.dispatch import Signal

related_people_signal = Signal()

Далі створюється повідомлення та відправляється до каналу зв’язку *websockets* кожного користувача у списку. Алгоритм роботи відправки задачі користувачам зображено на рис. 3.8.

Для того щоб задача закріпилася за користувачем, якого додав інший користувач, він має у особистих повідомленнях перейти за посиланням до задачі та підтвердити свою причетність до неї. Якщо користувач скасує задачу, повідомлення про цю дію прийде до творця задачі.

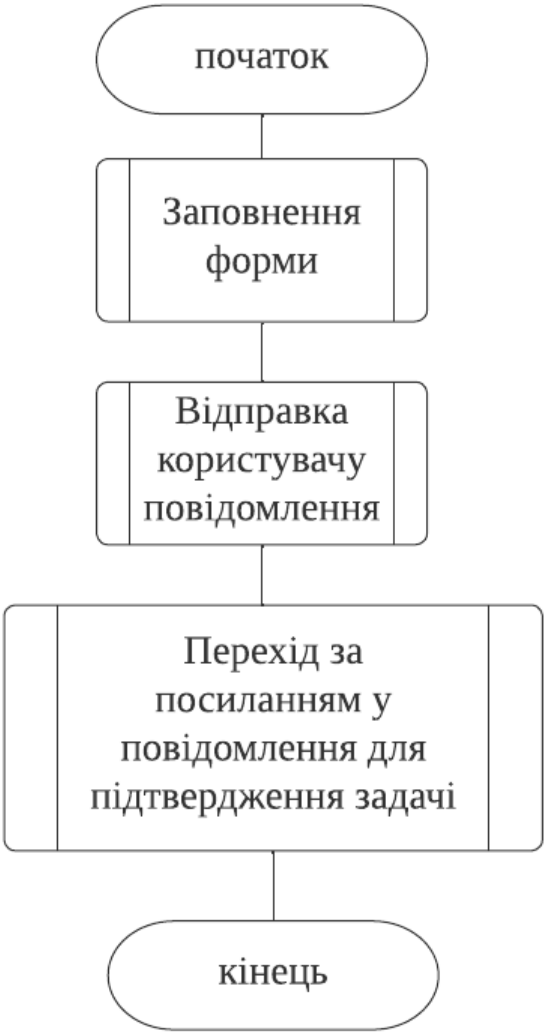


Рис. 3.8. Алгоритм роботи відправки задачі користувачу

Задача користувача може містити прикріплений файл. Для цього створено окреме місце у базі даних, щоб файли зберігалися в хмаровому сховище користувача та валідатор, який перевіряє правильність формату файлу та його ім’я. Код валідатору наведено нижче:

```
def validate_file_extension(value):
    ext = os.path.splitext(value.name)[1]
    valid_extensions = ['.jpg', '.png', '.doc', '.docx', '.pdf', '.xls', '.xlsx']
    if not ext.lower() in valid_extensions:
        raise ValidationError('Unsupported file extension.')
```

Користувач може отримати список декількох задач, відфільтрувати його, а також, якщо має доступ, отримати інформацію про задачі інших працівників.

3.13. Реалізація акцій, промокодів та знижок

Уся інформація про акції, промокоди та знижки береться з *API* магазину. Для цього реалізовано функцію, яка робить запит та оброблює отримані дані. Після обробки функція повертає дані у вигляді *json*:

```
class PromotionsListAPIView(generics.ListAPIView):

    def list(self, request, *args, **kwargs):
        page = request.query_params.get('page')
        page_size = request.query_params.get('page_size')
        if page is not None and page_size is not None:
            result= requests.get(f'{CMS_ROOT}/api/promotions/?page={page}&page_size={page_size}')
        elif page is not None:
            result = requests.get(f'{CMS_ROOT}/api/promotions/?page={page}')
        elif page_size is not None:
            result= requests.get(f'{CMS_ROOT}/api/promotions/?page_size={page_size}')
        else:
            result = requests.get(f'{CMS_ROOT}/api/promotions/')
        response_result = result.json()
        response_result['next']= str(response_result['next']).replace(CMS_ROOT, request.get_host())
        response_result['previous']= str(response_result['next']).replace(CMS_ROOT, request.get_host())
        return Response(response_result, status.HTTP_200_OK)
```

Отримані дані можна сформувати у вигляді календарю, відфільтрувати та сформувати звіт.

3.14. Реалізація додаткового функціоналу

У системі реалізовано статуси користувача:

- в мережі;
- відійшов;
- не турбувати;
- працює;
- не в мережі.

Під час реалізації було додаткове проміжне програмне забезпечення, яке робило запит до системи, отримувало інформацію про останні дії користувача і в залежності від часу повертала статус користувача. Приклад програми наведено нижче:

```
class OnlineNowMiddleware(object):
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        user = request.user
        response = self.get_response(request)
        if user.is_authenticated:
            User.update_user_activity(user.pk)
            return response
        else:
            return response
```

3.15. Висновки до розділу

В даному розділі було описано процес проектування системи для взаємодії з клієнтами. Було описано технології і процеси розробки кожної підсистеми. В якості баз даних було використано *PostgreSQL, Redis, SQLite3.*

Реалізовано функціонал та модель користувача, систему коментарів, систему відстеження дій користувачів та клієнтів, інтеграцію з автоматичною телефонною станцією *Binotel*, інтеграцію з аналітичним сервісом *Google Analytics Report API*, розраховані показники для компаній стосовно клієнтів, реалізовано інтеграцію з магазином в розділі акцій, замовлень та клієнтів, розроблено таймер робочого часу, а також розроблено систему чатів.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було детально досліджено існуючі аналогові розробки. З основних систем керувань взаємовідносинами виділено та досліджено 4:

- *zoho*;
- *bitrix24*;
- *Worksection;*
- *NetHunt.*

Усі ці системи є у відкритому доступі. В усіх системах реалізовано базовий та додатковий функціонал. Але усі вони нав’язують свій інтерфейс користувачу, що зв’язує його та примушує опускати деякі моменти в бізнес процесах компанії. Не кожна дозволяє реалізувати інтеграцію з стороннім ресурсом, популярні системи керування взаємовідносинами з клієнтами орієнтуються на великі міжнародні сервіси, що доволі незручно локальним користувачам. Тож створення власного продукту, який дозволяє зробити усі процеси більш динамічними та гнучким є актуальною темою сьогодення.

Кожна нова система *CRM* система, яка розробляється під компанію, має задовольняти цим потребам:

- швидка комунікація з клієнтом;
- ізольованість даних;
- повний набір інструментів для роботи компанії;
- можливість інтеграції зі сторонніми сервісами;
- можливість контролювати та оптимізувати роботу персоналу;
- швидка комунікація серед персоналу.

Для цього необхідно використовувати модульний підхід. Кожна частина системи має працювати самостійно, незалежно від роботи іншої частини. Тим самим процес інтеграції проходить мінімізуючи помилки при роботі системи.

Використовуючи такий підхід було виділено наступні програмні сутності – частини системи:

- задачі;
- дзвінки;
- календарі;
- чати;
- коментарі;
- контакти;
- база знань компаній;
- дошка аналітики;
- угоди;
- ліди;
- повідомлення;
- акції;
- користувачі;
- активність користувача;
- вимірювач робочого часу.

Наступним кроком було проектування *CRM*. Було обрано два методи проектування:

- створення функціонального прототипу кожної складової системи;
- структурний опис кожної моделі бази даних системи.

Під час розробки функціонального прототипу було створено за допомогою редактору *Figma* макети усіх сторінок додатку. Це дозволило покращити розуміння структури розроблюваного додатку та мати чітке розуміння очікуваного результату. Створення прототипів є важливою частиною будь-якого виробництва, незалежно від того, розробляється апаратне чи програмне забезпечення. Функціональний прототип може приймати різні форми: намальований від руки та змодельований за допомогою паперу, візуалізований у каркасах або надрукований на *3D*.

Структурний опис моделей баз даних відображає організацію системи з точки зору компонентів, які складають цю систему, та їхніх взаємозв’язків. Структурні моделі можуть бути статичними моделями, які показують структуру проекту системи, або динамічними моделями, які показують організацію системи під час її виконання.

Характеристики, представлені ознакою, можуть бути екземплярами класифікатора, що розглядаються окремо або самого класифікатора. Одна й та сама функція не може бути статичною в одному контексті та нестатичною в іншому. Атрибути класу представлені екземплярами властивості, якими володіє клас. Об’єкти класу повинні містити значення для кожного атрибута, який є членом цього класу, відповідно до характеристик атрибута, наприклад його типу та кратності.

Програмування системи вироблялося на мові *Python.* Вона дозволяє швидко та ефективно розробляти серверні додатки. В якості головного інструменту було обрано *Django* та його розширення *Django Rest Framework.*

На початку було створено основу проекту. В ній зазначено підключення до бази даних, налаштування серверу, підключення окремих додатків, налаштована автентифікація, підключене додаткове програмне забезпечення, визначено шаблони веб-сторінок, створені канали зв’язку *websockets*, налаштована електрона пошта для розсилок та підключена інтеграція з магазином клієнта.

У якості баз даних було використано *PostgreSQL, Redis* та *SQLite. PostgreSQL* використовуються для великих об’ємів даних. *Redis* використовуються для системи виводу даних в режимі реально часу. *SQLite* використовуються для менших об’ємів даних.

Далі розроблювалися окремі додатки визначених програмних сутностей. Взагалі було зроблено наступне:

- реалізовано користувачів;
- реалізовано відстеження активності користувачів;
- реалізовано панелі дзвінків;

– реалізовано панелі календарів;
– реалізовано чати;
– реалізовано коментарі;
– реалізовано контакти;
– реалізовано дошки аналітики;
– реалізовано угоди;
– реалізовано таймери контролю часу роботи;
– реалізовано задачі;
– реалізовано акції, промокоди та знижки;
– реалізовано додатковий функціонал.

Одержані результати дають підстави вважати, що завдання реалізоване, мета досягнута.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Основи розробки веб-додатків. — СПб.: Пітер, 2015. — 272 с.: іл. — (Серія «Бестселіри *O'Reilly*»).
2. Вивчаєм *HTML, XHTML* и *CSS*. 2-е вида. — СПб.: Пітер, 2014. — 720 с.: іл. — (Серія «*Head First O'Reilly*»).
3. Вивчаєм *Python*, том 1, 5-е вида.: Пер. з англ. — СПб.: ООО "Діалектика", 2019. — 832 с. : іл. — Парад, тит. англ.
4. Програмування на *Python* 3. – Пер. з англ. – СПб.: СимволПлюс, 2009. – 608 с., іл.
5. Бейдер Д. Чистий *Python*. Тонкості програмування для профі. — СПб.: Пітер, 2018. — 288 с.: іл. — (Серія «Бібліотека програміста»).
6. Молинаро Э. SQL. Сборник рецептов. – Пер. с англ. – СПб: Символ!Плюс, 2009. – 672 с., ил.
7. *Python*. Карманный справочник, 5-е изд. : Пер. с англ. – М. : ООО "И.Д. Вильямс", 2015. – 320 с. : ил. – Парал. тит. англ.
8. ГрантКит *CSS* для профи. — СПб. : Питер , 2019.—496 с. : ил . — (Серия «Библиотека программиста»).
9. Харрисон Мэтт Как устроен *Python*. Гид для разработчиков, программистов и интересующихся. — СПб.: Питер, 2019. — 272 с.: ил. — (Серия «Библиотека программиста»).
10. Прохорецок Н. А. *Python* 3 и PyQt. Разработка приложений. - : - СПб.: БХВ-Петербург, 2012.-704 с.: ил.
11. Новая большая книга *CSS*. — СПб.: Питер, 2016. — 720 с.: ил. — (Серия «Бест-селлеры *O'Reilly*»).
12. Слаткив Бретт. Секреты *Python*: 59 рекомендаций по написанию эффективного кода.: Пер. с англ. -М.: ООО ии.д. Вильяме", 2016. - 272 с.: ил. -Парал. тит. англ.

13. Бизли Д., Джонс Б. К.Б59 *Python*. Книга рецептов / пер. с анг. Б. В. Уварова. – М.: ДМК Пресс, 2019. – 648 с.: ил.
14. Дронов В. *HTML5*, *CSS3* и *Web* 2.0. Разработка современных *Web*-сайтов. — СПб.: БХВ-Петербург, 2011. —416с.: ил. — (Профессиональное программирование).
15. Гринберг М. Рзработка веб-приложений с использованием *Django* на языке *Python* / пер. с англ. А.Н. Киселева. – М.: ДМК Пресс, 2014. – 272 с.: ил.
16. ДСТУ 3008-95 Документація. Звіти у сфері наук і техніці Структура і правила оформлення.
17. Положення про дипломні роботи(проекти) випускників Національного авіаційного университету СМЯ НАУ П 03.01(10) – 02 – 2017.
18. Сузи Р.А. *Python* –СПб.: БХВ-Петербург, 2002. – 768 с.: ил.
19. Свергайт Эл. *Python*. Чистый код для продолжающих. — СПб.: Питер, 2022.—384 с. : ил . — (Серия «Библиотека программиста»).
20. Чан Джейми. *Python*. Быстрый старт. — СПб.: Питер, 2021.—224 с. : ил . — (Серия «Библиотека программиста»).
21. Тихоненко В. *Pythonissam* В. Тихоненко — «Издательские решения»
22. Ален Б. Дауни. Think DSP. Цифровая обработка сигналов на Python / пер. с англ. Брядинский А.Э. – М.: ДМК Пресс,2017. – 160 с.: ил.
23. Лутц М. Программирование на *Python*, том I, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с., ил.
24. Златопольский Д.М. Основы программирования на языке *Python*. – М. ДМК Пресс, 2017. – 284 с.: ил.
25. Форсье Дж., Биссекс П., Чан У. *Django*. Разработка веб-приложений на *Python*. – Пер. с англ. – СПб.: Символ-Плюс,2009. – 456 с., ил.

ДОДАТОК А

Лістинг коду основного програмного модулю

Файл *chat.views.py*

```
from django.shortcuts import render, get_object_or_404
from rest_framework.viewsets import ModelViewSet
from chat.serializers import ChatSerializer, ContactSerializer, MessageSerializer
from chat.models import Chat, Contact, Message
from rest_framework import viewsets, status, permissions, generics, filters
def get_last_10_messages(chat_id):
    chat = get_object_or_404(Chat, id=chat_id)
    messages = Message.objects.filter(chat=chat)
    result_list = messages.order_by('-created_at').all()[:10]
    for msg in result_list:
        if msg.status != "Read":
            msg.status = "Read"
            msg.save()
    return result_list
def index(request):
    return render(request, 'chat/index.html')
def room(request, room_name):
    return render(request, 'chat/room.html', {
        'room_name': room_name
    })
class ChatViewSet(ModelViewSet):
    serializer_class = ChatSerializer
    queryset = Chat.objects.all()
    permission_classes = [permissions.IsAuthenticated]
```

```
    def get_queryset(self):
        user = self.request.user
        return Chat.objects.filter(users__in=[user])
class ContactViewSet(ModelViewSet):
    serializer_class = ContactSerializer
    queryset = Contact.objects.all()
    permission_classes = [permissions.IsAuthenticated]
    def get_queryset(self):
        user = self.request.user
        return Contact.objects.filter(owner=user)
class MessageAPIView(generics.RetrieveAPIView):
    search_fields = ['text']
    filter_backends = (filters.SearchFilter,)
    serializer_class = MessageSerializer
    permission_classes = [permissions.IsAuthenticated]
    def get_queryset(self):
        user = self.request.user
        chats = Chat.objects.filter(user__in=[user])
        return Message.objects.filter(chat__in=[chats])
    def load_user(user_id):
        return User.query.get(user_id)
    return app
```

Файл *chat.serializers.py*

```
from rest_framework import serializers
from chat.models import Chat, Contact, Message
from users.serializers import UserSerializer
from users.models import User
from rest_framework.exceptions import PermissionDenied
from rest_framework import filters
```

```
class ChatSerializer(serializers.ModelSerializer):
    owner = UserSerializer(required=False)
    users = serializers.ListSerializer(child=UserSerializer(), read_only=True)
    class Meta:
        model = Chat
        fields = '__all__'
    def create(self, validated_data):
        user = self.context['request'].user
        validated_data['owner'] = user
        chat = Chat.objects.create(**validated_data)
        chat.users.add(user)
        return chat
    def to_representation(self, instance):
        response = super().to_representation(instance)
        msg = Message.objects.filter(chat=instance)[-1]
        response['last_message'] = msg.text
        return response
    def update(self, instance, validated_data):
        user = self.context['request'].user
        if instance.owner == user:
            instance = super().update(instance, validated_data)
            return instance
        else:
            raise PermissionDenied('Permission denied')
class ContactSerializer(serializers.ModelSerializer):
    contact = serializers.SlugRelatedField(slug_field='email',
queryset=User.objects.all())
    class Meta:
        model = Contact
```

```
        fields = ('contact', )
    def create(self, validated_data):
        user = self.context['request'].user
        validated_data['owner'] = user
        contact = Contact.objects.create(**validated_data)
        return contact
    def to_representation(self, instance):
        user = self.context['request'].user
        if instance.owner == user:
            response = super().to_representation(instance)
            return response
        else:
            raise PermissionDenied('Permission denied')
    def update(self, instance, validated_data):
        user = self.context['request'].user
        if instance.owner == user:
            instance = super().update(instance, validated_data)
            return instance
        else:
            raise PermissionDenied('Permission denied')
class MessageSerializer(serializers.ModelSerializer):
    class Meta:
        model = Message
        fields = "__all__"class Status(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, nullable=False)
    task = db.Column(db.Integer, db.ForeignKey("task.id"))
    def __repr__(self):
        return f"<Status {self.name}>"
class Comment(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
    text = db.Column(db.Text, nullable=False)
    date = db.Column(db.DateTime, nullable=False)
    attach = db.Column(db.LargeBinary, nullable=True)
    task = db.Column(db.Integer, db.ForeignKey("task.id"))
    author = db.Column(db.Integer, db.ForeignKey("user.id"))
    def __repr__(self):
        return f"<Comment {self.text}>"
class Tag(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, nullable=False)
    def __repr__(self):
        return f"<Tag {self.name}>"
import json
from asgiref.sync import async_to_sync
from channels.generic.websocket import WebsocketConsumer
from chat.views import get_last_10_messages
from chat.models import Chat, Message
from notifications.models import Notification
from django.contrib.contenttypes.models import ContentType
from users.models import UserAttr
```

Файл *chat.consumers.py*

```
class ChatConsumer(WebsocketConsumer):
    def fetch_messages(self, data):
        messages = get_last_10_messages(data['chatId'])
        content = {
            'command': 'messages',
            'messages': self.messages_to_json(messages)
```

```
            }
            self.send_message(content)
        def messages_to_json(self, messages):
            result = []
            for message in messages:
                result.append(self.message_to_json(message))
            return result
        @staticmethod
        def message_to_json(message):
            return {
                'id': message.id,
                'author': str(message.user.first_name) + " " +
str(message.user.last_name),
                'text': message.text,
                'created_at': str(message.created_at)
            }
        def connect(self):
            self.room_name = self.scope['url_route']['kwargs']['room_name']
            self.room_group_name = 'chat_%s' % self.room_name
            self.user = self.scope['user']
            try:
                self.chat = Chat.objects.get(id=self.room_name)
                if self.user in self.chat.users.all():

                    # Join room group
                    async_to_sync(self.channel_layer.group_add)(
                        self.room_group_name,
                        self.channel_name
                    )
                    self.accept()
```

```
        else:
            async_to_sync(self.channel_layer.group_add)(
                self.room_group_name,
                self.channel_name
            )
            self.accept()
            self.send_message(dict(
                action='connect',
                error="Permission Denied",
                status=404
            ))
            async_to_sync(self.channel_layer.group_discard)(
                self.room_group_name,
                self.channel_name
            )

    except Chat.DoesNotExist:
        async_to_sync(self.channel_layer.group_add)(
            self.room_group_name,
            self.channel_name
        )
        self.accept()
        self.send_message(dict(
            action='connect',
            error="The chat isn't found",
            status=404
        ))
        async_to_sync(self.channel_layer.group_discard)(
            self.room_group_name,
```

```
                self.channel_name
            )
    def disconnect(self, close_code):
        async_to_sync(self.channel_layer.group_discard)(
            self.room_group_name,
            self.channel_name
        )
    # Receive message from WebSocket
    def receive(self, text_data):
        text_data_json = json.loads(text_data)
        message_text = text_data_json['message']
        message = Message.objects.create(text=message_text, chat=self.chat,
user=self.user, status='Sent')
        chat_type = ContentType.objects.get(app_label='chat', model='chat')
        for u in self.chat.users.all():
            if u != self.user:
                user_attr = UserAttr.objects.get(user=u)
                Notification.objects.create(name=f'Новое сообщение в чате
{self.chat.name}',
                                            user_name=u.first_name,
                                            user_surname=u.last_name,
                                            phone=user_attr.personal_phone,
                                            type="IN",
                                            description="Пришло сообщение",
                                            content_type=chat_type,
                                            object_id=1
                                            )
    sync_to_sync(self.channel_layer.group_send)(
            self.room_group_name,
            {
```

```
                'type': 'chat_message',
                'id': message.id,
                'author': str(message.user.first_name) + " " +
str(message.user.last_name),
                'text': message.text,
                'created_at': str(message.created_at)
            }
        )

    # Receive message from room group
    def chat_message(self, event):
        msg_id = event['id']
        message = Message.objects.get(id=msg_id)
        message.status = "Read"
        message.save()
        if self.chat.channel is False or (self.user == self.chat.owner or self.user in
self.chat.admins.all()):
            # Send message to WebSocket
            self.send(text_data=json.dumps({
                'id': message.id,
                'author': str(message.user.first_name) + " " +
str(message.user.last_name),
                'message': message.text,
                'created_at': str(message.created_at)
            }))

    def send_message(self, message):
        self.send(text_data=json.dumps(message))
```