Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема Розробка веб додатку для продажу нерухомості

Спеціальність 121 “Інженерія програмного забезпечення”
код і назва спеціальності

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДП.ІПЗ-25.ПЗ
(позначення)

Рецензент	Студент
ст. викладач Савка І.Я.	ІПЗ-41 Сокіл А.С.
(посада) (підпис) (дата) (розшифровка підпису)	(шифр групи) (підпис) (дата) (розшифровка підпису)
Нормоконтролер	Керівник дипломного проекту
ст. викладач Савка І.Я.	доц. Лазарович І.М.
(посада) (підпис) (дата) (розшифровка підпису)	(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

2020

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко М.І.
_____
"_____"_____ 20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Соколу Андрію Святославовичу
(прізвище, ім'я, по батькові студента)

1. Тема проекту Розробка веб додатку для продажу нерухомості.

Затверджена розпорядженням факультету математики та інформатики від 25 жовтня 2019р., №7

2. Термін здачі студентом закінченого проекту 22 травня 2020 р.

3. Вихідні дані до дипломного проекту існуючі електронні ресурси з продажу нерухомості, теорія та інструментальні засоби моделювання, розробки та тестування ПЗ, методи аналізу економічної доцільності проекту.

4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
1. Постановка задачі. Огляд існуючих аналогів
2. Розробка структури проекту
3. Розробка програмного забезпечення
4. Бізнес план

5. Перелік графічного матеріалу (з точним забезпеченням обов'язкових креслень) "Актуальність роботи", "Мета та завдання для проекту", "Складність у пошуку необхідної нерухомості", "Приклад частково схожого функціоналу", "Структура бази даних", "Вигляд адмін частини", "Вигляд форми", "Успішне відправлення форми", "Загальна інформація про проект"

6. Дата видачі завдання 11.09.2019

Керівник _____ Лазарович І.М.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Сокіл А.С.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
1. Аналіз області розробки та постановка задачі	02.12.2019	Виконав
2. Огляд технологій та проектування продукту	15.02.2020	Виконав
3. Розробка демонстраційного програмного забезпечення	17.04.2020	Виконав
4. Бізнес план розробки додатку	11.05.2020	Виконав
5. Оформлення пояснювальної записки	18.05.2020	Виконав

Студент _____ Сокіл А.С.
(підпис) (розшифровка підпису)

Керівник проекту _____ Лазарович І.М.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 80 сторінок (без додатків), 49 рисунків, 2 таблиці, 16 джерел, 1 додаток на 50 сторінках.

Ключові слова: ВЕБ ДОДАТОК, CRUD, FRONTEND, BACKEND, НЕРУХОМІСТЬ, PHP, LARAVEL, VUE.JS, ФОРМА.

Об`єктом дослідження є удосконалення процесу продажі нерухомості.

Мета роботи: розробка веб додатку, при якому продавці і орендодавці будуть мати змогу збільшити кількість клієнтів, а користувачі - потратити мінімально часу і отримати бажаний результат.

Стислий опис тексту пояснювальної записки:

Описано поетапне створення веб додатку, його плюси і мінуси, для кого він буде корисний, а також розроблено бізнес план.

Для розробки було обрано такий інструментарій розробки:

- Laravel – розробка backend частини.
- Vue.js – розробка frontend частини.
- MySQL – СУБД.
- PHPStorm – редактор коду.

ABSTRACT

Explanatory note: 80 pages (without appendix), 49 figures, 2 tables, 16 references, 1 appendix on 50 pages.

Key words: WEB APPLICATION, CRUD, FRONTEND, BACKEND, REAL ESTATE, PHP, LARAVEL, VUE.JS, FORM.

Object of study: improve the process of selling real estate.

Purpose: to develop a web application in which sellers and landlords will be able to increase the number of customers, and for the user it is a great opportunity to spend a minimum of time and get the desired result.

Brief description of the text of the explanatory note:

Describes the gradual creation of a web application, its pros and cons, for whom it will be useful as well as a business plan developed.

The following development tools were selected for development:

- Laravel – develop backend side.
- Vue.js – develop frontend side.
- MySQL – DBMS.
- PHPStorm – code editor.

ЗМІСТ

					ДП.ІПЗ-25.ПЗ		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		Сокіл А. С.			Розробка веб додатку для продажу нерухомості	*Літ.*	*Аркуш* 7 / *Аркуші* 130
Перев.		Лазарович І.М.					
Рецензент		Савка І.Я.				ПНУ ІПЗ-41	
Н. контр.		Савка І.Я.					
Затверд.		Козленко М.І.					

ВСТУП

Ринок нерухомості – це поки що один зі стабільних секторів економіки, який показує позитивну динаміку попиту. За даними Державної служби статистики, протягом 12 місяців 2019 року обсяг будівництва нежитлових будівель у країні зріс більше ніж на 31% порівняно з аналогічним періодом минулого року. Також на 20,8% зріс і обсяг виробленої в Україні будівельної продукції за той самий період. Попит на нове житло поступово наздоганяє пропозицію цьому сприяє по-перше населення, яке прагне покращити свої умови життя і по-друге високий інвестиційний попит, передусім - на ринку новобудов яке можна поділити на дві складові:

- спекуляції – це купівля житла на етапі будівництва і продажі його після його закінчення за значно вищою ціною;
- довгострокові інвестиції з метою здачі житла в оренду або майбутньої перепродажі.

Продаж нерухомості і його купівля, зазвичай ця процедура є не самою простою і для пошуку покупця, або відповідної нерухомості може знадобитися чимало часу. На це впливає багато факторів:

- власні вподобання;
- фінансові можливості;
- географічне розміщення;
- багато інших другорядних факторів.

Для того щоб цей процес відбувався швидше і зручніше існує чимало інтернет сервісів для продажі і купівлі нерухомості, але у більшості випадків це проста консультація або інтернет магазин з багатьма сторінками, де пошук відповідної нерухомості може займати багато часу і проблема в тому, що немає жодних гарантій, що ви знайдете щось належне. Для вирішення цієї проблеми в даному дипломному проекті буде розроблений веб додаток, який буде

покращувати і пришвидшувати цей процес. Основним завданням веб додатку є звести продавця і покупця, а відбуватися цей процес буде додаванням користувачів в адмін частині з набором певних критеріїв продажу і формою для клієнтів, які заповнивши її, відправляють свою форму на оброку і алгоритм автоматично підбере їй необхідного продавця з нерухомістю. Це є дуже актуально, оскільки завжди хочеться за мінімальний час отримати максимальну вигоду і щоб процес був зручний та приносив задоволення під час користування.

1 ПОСТАНОВКА ЗАДАЧІ. ОГЛЯД ІСНУЮЧИХ АНАЛОГІВ

1.1 Аналіз і дослідження проекту

Сьогодні нерухомість, та все що з нею пов’язано, займає найбільшу частину світового “портфеля” багатства, та є одним з найпопулярніших фінансових активів, вартість і зацікавленість інвесторів постійно зростає [1].

Ринок нерухомості - це сукупність організаційно-економічних відносин, засіб перерозподілу земельних ділянок, будинків та іншого майна між власниками і користувачами на основі попиту і пропозиції. За допомогою державного регулювання він забезпечує:

- передачу прав на нерухомість від однієї особи до іншої;
- встановлення рівних цін на нерухомість в регіонах і місцевостях;
- зв'язок між власниками і покупцями;
- розподіл простору між конкуруючими варіантами використання земель та суб'єктами ринку.

Основу частину ринку нерухомості складають вже існуючі ділянки, новостворювані, реконструюються, розвиваються підприємства, будівлі і споруди різного призначення, а також гроші чи фінансовий капітал. Нестача інвестицій стримує розвиток підприємств та пришвидшення вітчизняного виробництва товарів, що в свою чергу зменшує інвестиційні можливості. Незавершені роботи інвестицій знаходяться у власності учасників цього процесу до моменту приймання та оплати інвестором виконаних робіт і послуг. Якщо замовник відмовиться від подальшого інвестування він зобов'язаний компенсувати витрати іншим його учасникам, якщо ця умова передбачена в договорі.

Ринок нерухомості складається з семи частин: попит, пропозиція, ціна, менеджмент, маркетинг, інфраструктура та ділові процедури.

Попит - це кількість земельних ділянок, майнових комплексів та прав на них, які покупці готові купити. При рівних умовах попит на нерухомість змінюється в протилежності від ціни. Формується ж попит на нерухомсіть, наприклад на земельні ділянки, під впливом якихось чинників: соціальних, економічних, природно-кліматичних, демографічних та ін.

Пропозиція - це кількість нерухомості (метрів квадратних, кубічних метрів, соток гектарів, квартир тощо), яке власники готові продати за якусь договірну ціну та за певний проміжок часу. Пропозиція нерухомих об'єктів, особливо землі є дуже нееластична. Хоча за останні декілька років на кожного покупця квартир або офісів на ринку було декілька продавців, а також їх ціна практично не знижувалася.

Ціна - це кількість грошей, яка сплачується по домовленості за якусь одиницю нерухомості.

Вартість - це грошовий еквівалент власності. Це найвища ціна, яку принесе продаж нерухомості на ринку, коли покупець і продавець діють розумно і на операцію не впливають сторонні стимули (ринкова вартість).

Найважливіший елемент і умова існування будь-якого ринку це - інформаційна інфраструктура, яка повинна містити відомості трьох видів:

- існуючі норми і правила роботи на ринку нерухомості;
- структуру попиту і пропозицій на різних об'єктах;
- динаміку і рівень цін нерухомості по районах, мікрорайонах і іншим сегментам ринку.

Основні елементи формування бази даних:

- заявки потенційних продавців і покупців, орендодавців і орендарів, користувачів і власників, які бажають зробити обмін нерухомості;
- реклама в журналах, газетах та інших виданнях;
- спеціальні опитування та дослідження.

В практиці застосовують інформаційні системи, які формуються кожною компанією для власного використання, а також регіональні бази даних.

Оскільки нерухомість включає в себе різні поєднання юридичних прав та інтересів, то в деяких випадках може виходити оренда, в інших - заставні зобов'язання, а в третіх - об'єкт повній власності, який є вільний від будь-яких вимог.

Ринок нерухомості крім специфіки обертання на ньому товарів є ще ряд загальних особливостей, які необхідно враховувати при здійсненні операцій на ньому:

- локалізація ринку, оскільки його ціна залежить від навколишнього середовища де знаходиться дана нерухомість;
- інформація про стан ринку зазвичай є конфіденційною, отже зазвичай вона є неповною і не завжди достовірною, так як операції з нерухомістю часто носять ексклюзивний характер;
- попит зазвичай визначається розташуванням, а не тільки і не стільки споживчими якостями самих об'єктів;
- еластичність пропозиції, оскільки неможливо відразу побудувати в багатьох нових квартирах, тому самі будівлі і споруди довговічні, а земля – вічна;
- товари - об'єкти нерухомості можуть бути обтяжені мораллю третіх осіб (сервітути та інші) в різних комбінаціях;
- висока ступінь державного регулювання законодавчими нормами ринку нерухомості і зонуванням території;
- нерухомість служить не тільки засобом задоволення власних потреб, але і об'єктом інвестиційної діяльності;
- відносно мала кількість учасників ринку і кількість здійснюваних операцій на ньому.
- велика змінність попиту по регіонах, районах і мікрорайонах [2].

Продаж, покупка, здача в оренду та пошук підходящої нерухомості для орендування зазвичай займає багато часу та енергії. В інтересах продавця або орендодавця це продати або ж здати нерухомість за максимально можливу ціну

і за мінімально можливий час в свою чергу для покупця або орендатора найти підходящу нерухомість, яка буде відповідати його побажанням і певним вимогам і домовитися за максимально можливу мінімальну ціну. Проте не завжди продавець з клієнтом доходять спільної згоди є чимало факторів які впливають на ціну нерухомості:

- район та розміщення нерухомості. Престижність району та його особливості, чи є зупинки громадського транспорту, наявність поблизу шкіл, дитсадків, промислових підприємст, парків та наявність чудового виду є одними найважливіших чинників встановлення ціни;
- який тип та матеріал будівлі, його планування, чи є можливість добудови та перепланування. Зазвичай покупці віддають перевагу цегляним будинкам, які є більш тепліші в холодні пори року, а також володіють високою шумоізоляцією, аніж панельні, а також покупці віддають перевагу швидше новішим будовам з сучасним плануванням ніж старим проектам;
- стан ремонту. Більшість покупців чи орендатрів шукають нерухомість зразу з ремонтами, тому ремонт часто відіграє важливу роль;
- коливання ринку нерухомості. У зв’язку з сезонними коливаннями, економічною та інвестиційною погодою ціни можуть змінюватися як догори так і в низ, зазвичай такі коливання можуть бути до 20% за місяць.

При встановленні ціни на нерухомість завжди беруться до уваги вищевказані фактори, а та обмеження у часі продажу, тобто наскільки терміново потрібно власнику продати нерухомість, чим швидше потрібно продати тим ціна буде меньшою, а якщо продавець готовий почекати шанс продати нерухомісіть за більш вигідну ціну збільшується.

Після того як було оцінено нерухомість, встановлено її ціну, то завжди залишається можливість покупцю торгуватися з власником, щоб спробувати купити нерухомість за меньшу ціну на ринку нерухомості як загальноприйняте незначне опускання ціни в разі їх зацікавлення та інтересу до конкретної

квартири. Хоча буває і таке, що при деяких обставинах і розумінню ринку, ціна може бути зниженою до 10%, що є досить хорошим результатом для покупця.

1.2 Порівняння аналогічних програмних продуктів

Зазвичай популярність сайтів розраховується за тим яка є активність на даному веб сайті, а вона залежить від наповненості ресурсу об'єктами нерухомості, зручності пошуку, а також інших факторах які роблять зручність користування веб сайтом максильно приємною. Найбільше популярними веб сайтами є ті які мають безкоштовну можливість виставлення нерухомості.
Розглянемо основні популярні веб сайти з продажі нерухомості:

- DOM.ria.com – веб сайт, який дозволяє швидко і легко розмістити оголошення або придбати нерухомість, а також є можливість знайти житло в новобудові або здати будинок в оренду. Веб сайт надає інформацію про агенції нерухомості, перевірених ріелторів, елітне житло;
- ЛУН.ua – спеціальний сайт – пошукова система для покупки або продажу нерухомості на території України. На сьогодні, напевно, найпопулярніший портал нерухомості в Україні. Житло можна орендувати на тривалий період або подобово. До сторінки прикріплена діюча карта з новобудовами. Дуже зручний, інформативний портал;
- Besplatka.ua – безкоштовна дошка оголошень, де можна шукати інформацію за тематикою або регіоном. Багатофункціональний ресурс, де є рубрики з нерухомістю, транспортом, ремонтом, електронікою та багато іншим;
- Domik.ua – сайт зі зручним каталогом і реальними відгуками про будинки. Користувачі мають можливість придбати акційну нерухомість від забудовників, в новобудовах. Доступна послуга купівлі земельної ділянки, комерційної нерухомості, елітного житла;

- OLX-нерухомість, цей сайт надає більше 200 тис. оголошень з нерухомістю вторинного ринку. Тут можна купити або продати житло, землю, офіс;
- Rieltor.ua – родзинкою сайту з продажу та купівлі нерухомості є список затребуваних ріелторів з відмінним послужним списком і високим рейтингом. В окремій рубриці розміщений список рекомендованих агентств з нерухомості;
- Mesto.ua – сервіс з пошуку нерухомості має зручний фільтр. Тут можна продати, купити або зняти житло в оренду;
- Fn.ua – сайт з фото нерухомістю, його робота ґрунтується на гаслі: «Краще один раз побачити!». База даних має більше 36 тис. доступних оголошень з гарячими пропозиціями і новинними зведеннями;
- Est.ua – надає можливість покупки житлової та комерційної нерухомості, не тільки квартиру, будинок, кімнату або дачу, але і санаторій, готель, торговий центр, заправку;
- Country.ua – молодий сайт, який подає великі надії. Нерухомість з ТОП-списку дозволить швидко знайти будинок мрії;
- Krysha.ua – відмінна пошукова система, що дозволяє швидко знаходити необхідне житло на сайтах-посередниках;
- Address.ua – доступний портал з нерухомістю, популярними оглядами і статистичними даними по ціновій політиці на квартири м.Києва;
- Meget – на цьому порталі можна ознайомитися з нерухомістю вторинного ринку і новобудовами. Для користувачів надані аналітичні відомості, основні закони і послуга купівлі житла в кредит;
- Кварторг – сайт з нерухомістю від господарів без посередників, де можна придбати будинок в оренду або купити квартиру;
- Realt.ua – незалежний український ресурс з новобудовами Києва та України. На головній сторінці можна ознайомитися з існуючими знижками і акціями на нове житло від різних забудовників;

- Metry.ua – на цьому порталі можна зняти, купити, продати житлоплощу, а вподобане оголошення додати «в обране», щоб стежити за зниженням цін;
- BN.ua – сайт зі зручною системою пошуку житла в м.Київ та області;
- Prostodom.ua – сайт порадує здобувача новою формою подачі оголошення, преміум-розміщенням. Дошка оголошень є корисною як для професійного ринку, так і для приватних клієнтів;
- Budynok.com.ua – Новий свіжий портал, є спеціальним інтернет майданчиком з широкими можливостями. На сайті можна переглядати і викладати свої оголошення, додавати новобудови і забудовників, агентства і ріелторів. Вся житлова і комерційна нерухомість України. На даний момент повністю безкоштовний;
- АГЕНТ.ua – інформаційна система з пошуку нерухомості, є можливість розмістити свою візитку з послугами в сфері нерухомості [4].

Кожен з цих сайтів має як плюси так і мінуси. Більшість даних інтернет магазинів на рину вже не перший рік і завоювали популярність серед користувачів. З подібним функціоналом який, ми будем розробляти володіє інтернет магазин АГЕНТ.ua. На веб сайті є можливість розмістити свою візитку з послугами в сфері нерухомості, проте мінусом даного функціоналу є те, що не має можливості для клієнтів, які шукають заповнити якусь форму і щоб їм автоматично підібрало необхідного продавця чи орендодавця. На інших сайтах відсутня така логіка, яку ми будемо розробляти. Отже, наш веб додаток буде мати унікальність на ринку продажі нерухомості в інтернеті.

1.3 Постановка задачі

1.3.1 Загальний опис

Загальний погляд на продукт. Веб додаток, призначений для інтернет магазинів з продажі нерухомості, аналогів на просторах інтернету не було знайдено.

Особливості продукту. Веб додаток, складається з 2 основних частин – це адмін частина та форма для інтегрування в інтернет магазин. Клієнти заповнюють форму, відправляють її і алгоритм автоматично підбере підходящого продавця з необхідною нерухомістю. Після чого продавець отримає email з необхідною інформацією для подальшої комунікації з клієнтом.

Класи і характеристики користувачів. В даному веб додатку існує 2 класи користувачі. Адміністратори, які працюють з адмін частиною (додають продавців чи орендодавців, управляють налаштуваннями форми, відслідковують аналітику і статистику форм). Користувачі, які заповняють форму в інтернет магазині для підбору необхідної нерухомості.

Операційне середовище. Даний програмний продукт це є веб-додаток. Працюватиме в будь якому сучасному браузері, а також є адаптивним. Форма буде мати коректне відображатися на різних розширеннях монітора, планшетах, а також мобільних пристроях.

Обмеження дизайну і реалізації. Веб додаток реалізований за допомогою наступних технологій:

- laravel – фреймворк для реалізації backend частини;
- СУБД MySql;
- vue.js – фреймворк для реалізації frontend частини.

Дизайн форми в інтернет магазині є дуже гнучкий, так як ця форма вставляється на сайт в iframe і може бути розміщена в будь якому зручному місці,

сама форма може бути модифікована по дизайну відповідно до якогось інтернет магазину.

Документація для користувачів. Документація для даного проекту відсутня, так як додаток є дуже простий у використанні, як для клієнтів, так і для адміністраторів.

Припущення і залежності. Основним припущенням може бути відсутність необхідності даного веб додатку, якщо в інтернет магазині існує велика кість фільтрів, яка дозволить дуже зручно і швидко підібрати необхідний товар, який буде максимально відповідати очікуванням клієнта, але мінусом є те, що потрібно самому зв'язуватися з продавцем чи орендодавцем, а у випадку використання нашої форми з клієнтами будуть зв'язуватися самі продавці чи орендодаці.

1.3.2 Функції системи

Опис і пріоритети. Веб додаток складається з 2 частин. Перша частина - це адмін панель, яка має в своєму функціоналі:

- створення продавців або орендодавців з відповідною інформацією для подальшого підбору їм клієнтів, які будуть заповняти форму;
- базові налаштування форми, такі як створення регіонів, районів, налаштування діапазону вартості, вибір дати і тд.;
- відслідковування прогресу відправлених форм, тобто чи було підібрано алгоритмом необхідного продавця чи орендодавця, можливість адміністратором додати клієнту користувача, можливість переглянути форми, які були незавершені і не відправленні, а також переглянути всю інформацію яка була заповнена у формі;
- переглядати аналітику, стосовно того, яких продавець чи орендодавець отримав клієнтів і інформацію про них.

Друга частина – це форма яка інтегрується в інтернет магазин, основною метою якою є після її заповнення підібрати необхідну і бажану нерухомість. Також форма повинна бути на двох мовах на англійській та французькій.

Послідовності «вплив-реакція». Перш за все потрібно адміністраторам в адмін частині додати продавців та орендодавців, після чого провести налаштування форми під особливості інтернет магазину. Наступним кроком інтегрувати форму в інтернет магазин. Коли клієнт заповняє і відправляє форму, алгоритм відповідно до його вподобань підбере необхідного продавця чи орендодавця. У випадку якщо не було нічого підібрано алгоритмом, адміністратор може власноруч додати клієнту користувача.

Після чого кожного ранку скрипт буде відправляти email всім продавцям і орендодавцям інформацію по клієнтах і вони зможуть зв’язатися для подальшої комунікації.

Функціональні вимоги. Функціональні вимоги для адмін частини:

- можливість створювати адміністраторів, продавців, орендодавців;
- можливість настройки полів форми (ціновий діапазон, дата);
- можливість переглядати відправленні форми;
- додавати або видаляти клієнтів для форми;
- можливість відслідковувати аналітику відправлених emails клієнтам.

Функціональні вимоги для форми:

- автоматично відправляти незакінчені форми після заповнення name, email, phone;
- підбір продавця або орендодавця після відправки форми.

Сповіщення і emails:

- адміністратори повинні отримувати email з даними нових форм і мати посилання на саму форму;
- раз в тиждень відправляти всім адміністраторам всі незаповнені форми;
- раз в день відправляти emails для продавців і орендодавців з клієнтами, які були для них підібрані.

Вимоги до зовнішніх інтерфейсів. Вимоги для інтерфейсу зв’язку відсутні. Все на вільний розсуд розробника.

1.3.3 Нефункціональні вимоги

Вимоги до продуктивності. Вимоги до продуктивності відсутні. Проте загальні вимоги можна побачити у стандарті ISO/IEC 9126-4.

Вимоги до збереженості даних. Вимоги до продуктивності відсутні. Проте загальні вимоги можна побачити у стандарті ISO/IEC 9126-4.

Критерії якості програмного забезпечення. Критерій якості програмного забезпечення відсутні. Проте загальні критерії можна побачити у стандарті ISO/IEC 9126-4.

Вимоги до безпеки системи. Вимог до безпеки системи відсутні. Проте загальні вимоги можна побачити у стандарті ISO/IEC 9126-4.

2 РОЗРОБКА СТРУКТУРИ ПРОЕКТУ

2.1 Архітектура проекту

Архітектура програмного забезпечення – це структура програми, яка включає в себе програмні компоненти де є видимими зовні властивості цих компонентів, а також їхні відносини. Цей термін також відноситься до документування архітектури програмного забезпечення. Документування архітектури програмного забезпечення зазвичай спрощує процес комунікації, а також дозволяє зафіксувати прийняті рішення на ранніх етапах проектування високорівневих дизайнів системи, а також надає можливість бути посторно використаними в інших модулях або проектах.

Область комп'ютерних наук завжди зтикалася з проблемами, пов'язаними зі складністю програмних систем та зі складністю їхньої підтримки. Раніше ці проблеми вирішувалися правильною структурою даних та розробки алгоритмі. Хоча даний термін є досить новим для індустрії програмного забезпечення, принципи цієї області невпорядковано застосовувалися розробниками програмного забезпечення починаючи з середини 1980-х. Під час перших спрод охарактеризувати що таке програмна архітектура і для чого вона потрібна була повна неточностей і страждали від нестачі організованості, часто це була просто діаграма з блоків, сполучених лініями. У наступні роки були спроби визначити і систематизувати основні аспекти даної дисципліни. Початковий набір шаблонів проектування, стилів дизайну, деякої логіки та інших деталей були розроблені протягом цього часу.

Ідея зниження складності системи шляхом абстракції і розмежування повноважень це є основна ідеєю дисципліни програмної архітектури. До сьогоднішнього дня так і немає узгодженого терміна щодо терміну "архітектура програмного забезпечення".

Будучи в сьогоденні свого розвитку дисципліною без чітких правил про "правильний" шлях створення системи, проектування архітектури ПЗ все ще є сумішшю науки і мистецтва. Аспект "мистецтва" полягає в тому, що будь-яка комерційна система має на увазі наявність застосування або місії. Те, які ключові цілі має система, описується за допомогою сценаріїв як нефункціональні вимоги до системи, також відомі як атрибути якості, що визначають, як буде вести себе система. Атрибути якості системи включають в себе відмово стійкість, збереження зворотної сумісності, розширюваність, надійність, придатність до сервісного обслуговування (maintainability), доступність, безпека, зручність використання, а також інші якості. З точки зору користувача програмної архітектури, програмна архітектура дає напрям для руху і вирішення завдань, пов'язаних зі спеціальністю кожного такого користувача, наприклад, зацікавленої особи, розробника ПЗ, групи підтримки ПЗ, фахівця із супроводу ПЗ, фахівця з розгортання ПЗ, тестера, а також кінцевих користувачів. У цьому сенсі архітектура програмного забезпечення насправді об'єднує різні точки зору на систему. Той факт, що ці декілька різних точок зору можуть бути об'єднані в архітектурі програмного забезпечення є аргументом на захист необхідності та доцільності створення архітектури ПЗ ще до етапу розробки ПЗ [3].

Від архітектури програмного забезпечення залежить ціна на підтримку і розробку нового функціоналу, трудовитрати на побудову цілої системи з використанням даної архітектури. Тобто формально від архітектури залежить найважливіший параметр розробки – собівартість, а також можливість повторного використання коду, а разом з ним і зменшення трудовитрат на кожну подальшу розробку. У контексті PHP7 - це ієрархія класів, інтерфейси та схеми їх взаємодії. Постановка задачі і її складність впливає на вибір або створення архітектури. Наприклад, наскільки універсальним планується додаток, які модулі повинні бути присутніми, яка запланована навантаження на ресурс. Про важливість чистої архітектури, «О ні, вони не сплять, ні. Багато сучасних розробників працюють як прокляті. Але частина їхнього мозку дійсно спить - та

частина, яка знає, що хороший, чистий, добре пропрацьований код відіграє важливу роль. Ці розробники вірять у відому брехню: Ми зможемо навести порядок потім, нам би тільки вийти на ринок!» [6].

2.1.1 MODEL-VIEWER-CONTROLLER

Стандартна схема архітектури «Модель-Вид-Контролер» зображена на (Рис. 2.1).

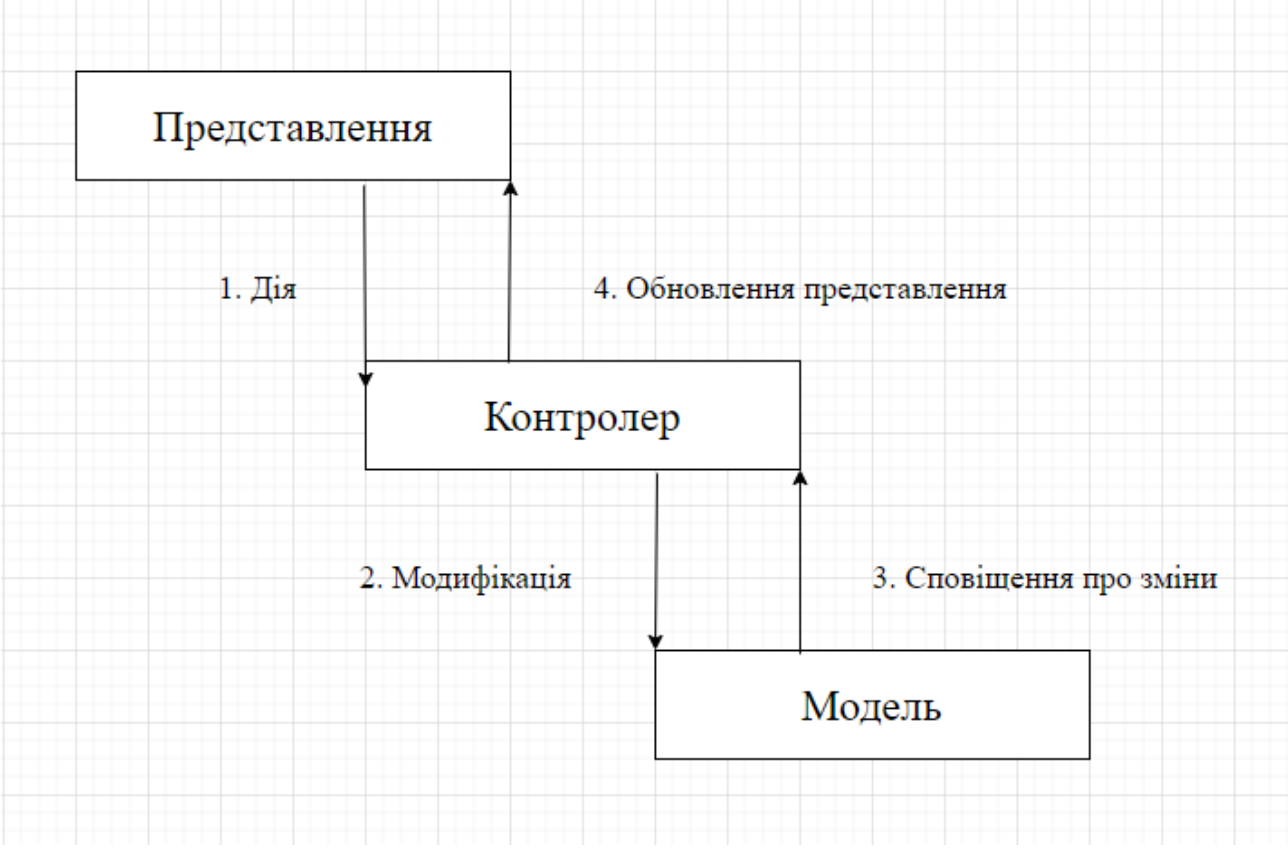


Рисунок 2.1 - Схема архітектури «Модель-Вид-Контролер»

Розберемо по пунктах дану схему.

Як можна здогадатися з назви, є три основних компонента в даній архітектурі:

- модель;

- представлення;
- контролер.

Представлення це те що бачить людина, коли користується даним продуктом, тобто якась інформація надходить з системи і користувач бачить результат роботи.

Модель це є сутність, яка відповідає за роботу алгоритмів, розрахунки, зв’язок з базою даних і тд.

Контролер це свого роду сполучна ланка, за допомогою нього і є можливість розділити модель і представлення. До контролера надходять дані від користувача, він передає їх моделі, модель обробляє дані, повертає результат роботи контролеру, а він в свою чергу відправляє дані для представлення.

Стосовно інтернет-додатків існує думка, що частини контролера і представлення об'єднані, тому що за відображення і одночасно за введення інформації відповідає браузер. Отже, представлення це є frontend частина, це може бути якийсь шаблонізатор, фреймворк, тобто все те що подає інформацію у вигляді HTML на основі даних, які йому приходять.

Контролер - це модуль управління введенням і виведенням даних. Він повинен стежити за переданими в систему даними (через форму, рядок запиту, cookie або будь-яким іншим способом) і на основі введених даних вирішити:

- чи передавати їх у модель;
- чи відправити повідомлення про помилку після чого повинне обновитися представлення з даними помилками.

Також, контролер визначає тип даних, отриманих від моделі (чи є це готовий результат, відсутність повідомлення про помилку) і передавати інформацію в представлення. До мінусів можна віднести:

- збільшення обсягу коду;
- необхідність дотримання заздалегідь заданого інтерфейсу.

До плюсів віднесемо наступне:

- код є більш гнучким;

- при необхідності, розширення додатку буде максимально легким;
- можливість повторно використовувати кожну з трьох складових [7,8].

2.1.2 Backend

Розробка Backend частини буде відбуватися за допомогою фреймворка Laravel мови програмування PHP. В її основі лежить архітектура програмування MVC. Так як даний фреймворк написаний на PHP він є дуже простим і зручним для розробки. Як працює сам Laravel можна зрозуміти з наступного малюнка (Рис. 2.2) [9].

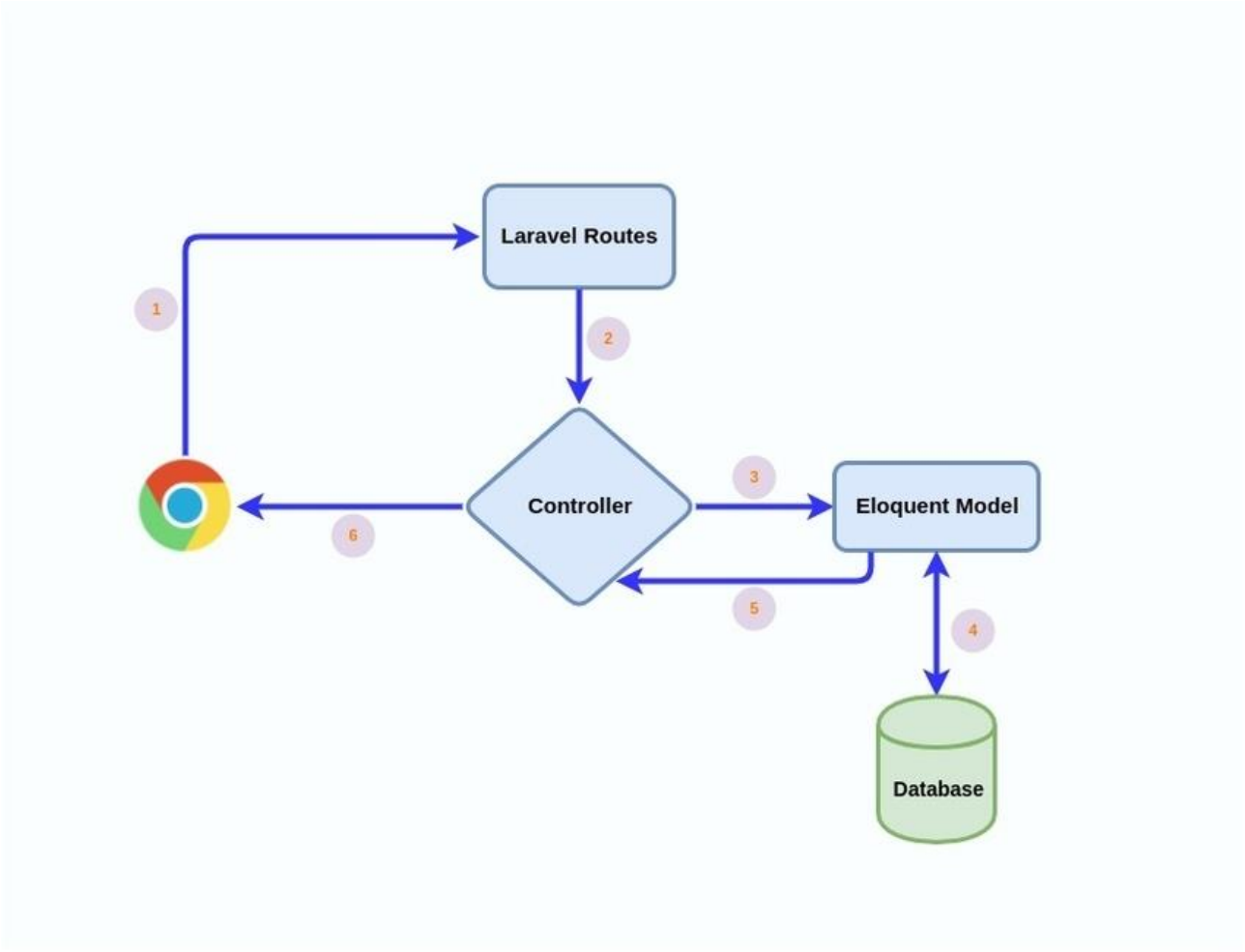


Рисунок 2.2 – Алгоритм роботи laravel

Отже перш за все нам потрібно створити моделі. В нашому випадку моделі це будуть таблиці, які будуть в базі даних. А також створимо в моделях методи, які ймовірно будуть нам потрібні. Вигляд моделей буде наступним (Рис. 2.3).

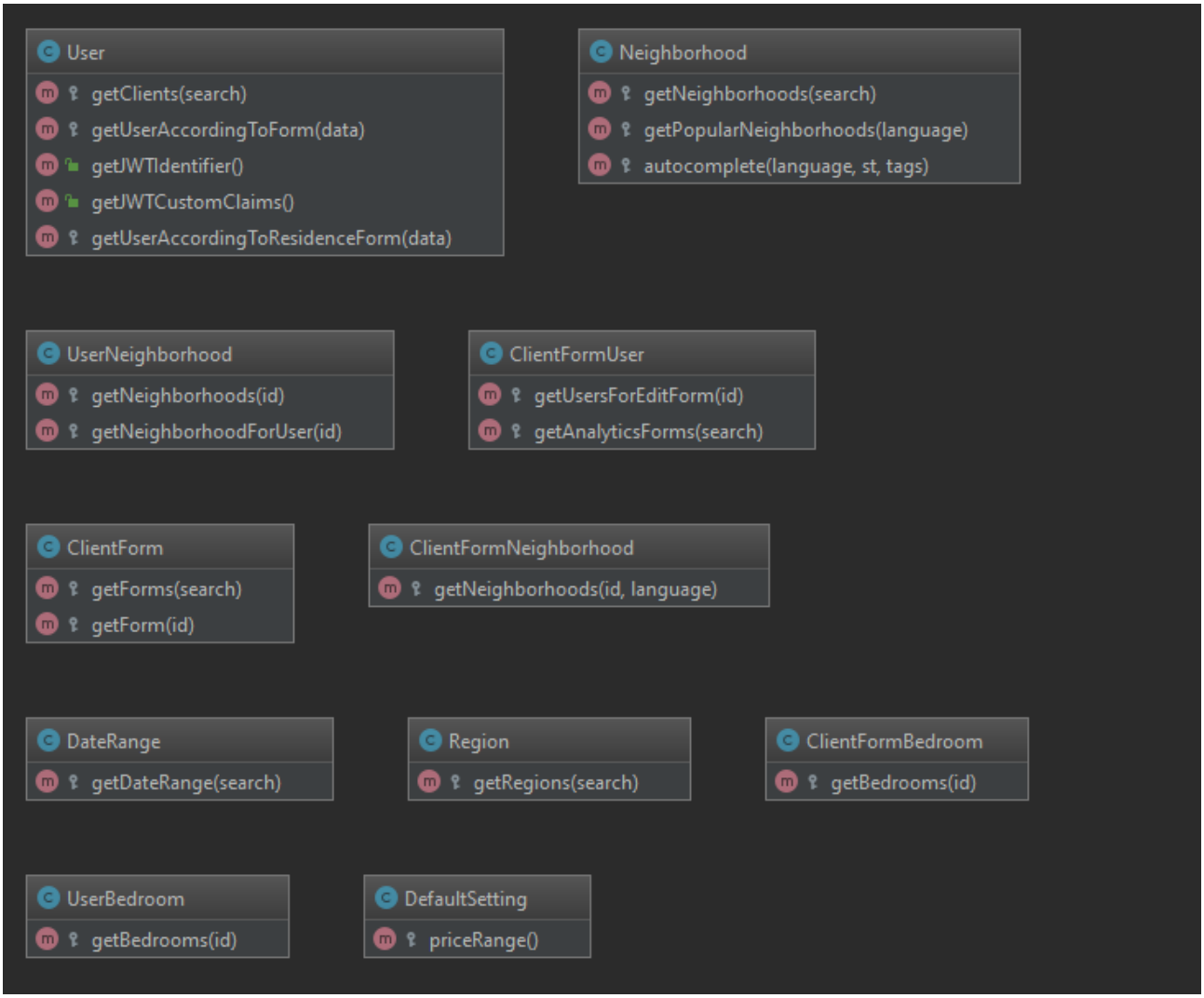


Рисунок 2.3 – Моделі веб додатку

Ми створили модель Users з методами:

- getClients – для отримання клієнтів;
- getUserAccordingToForm – це буде метод з нашим алгоритмом для підбору продавця чи орендодавця.

  Інші методи вони будуть необхідні нам для реєстрації.

  Модель Neighborhood буде мати методи:
- getNeighborhoods – для отримання всіх районів;

- getPopularNeighborhoods – для отримання самих популярних районів (тобто ті райони які найчастіше вибирають користувачі);
- autocomplete – для пошуку району по назві;

Наступна модель це UserNeighborHood з методами:

- getNeighborhoods – даний метод знадобиться нам для отримання всіх районів користувача в одному рядку через кому, для наших emails;
- getNeighborhoodForUser – потрібно, щоб отримати райони якогось конкретного користувача.

Модель ClientForm з методами:

- getForms – для отримання форм;
- getForm – отримання одної конкретної форми зі всіма деталями.

Наступна наша модель це ClientFormUser з методами:

- getUserForEditForm – даний метод буде потрібний, щоб отримати продавців чи орендодавців, які отримали якусь форму;
- getAnalyticsForms - цей метод дозволить отримати нам аналітику форми за якись період, тобто скільки користувачів отримали дану форму і зв’язалися з тим хто заповнював форму.

Модель ClientFormNeighborhood буде мати лише 1 метод getNeighborhoods для отримання всіх районів якоїсь конкретної форми.

Region ця модель має метод getRegions для отримання регіонів. Наступна модель це DateRange з методом getDataRange для отримання всіх дат.

Дві моделі пов’язані з кількістю кімнат, перша модель це UserBedroom з методом getBedrooms для отримання кімнат якогось користувача. Друга модель це ClientFormBedroom з методом getBedrooms також для отримання обраною кількості кімнат, але вже якоїсь конкретної форми. І остання модель це DefaultSetting, в якій будуть знаходитися методи з настройками форми або адмін частини.

Отже, після створення моделей, наступним нашим кроком буде створення контролерів, для зручності ми розділимо їх на Admin та Client. Ми будем чітко

бачити, які контролери відносяться до адмін частини, а які до форми. Перш за все створимо контролери для адмін частини (Рис. 2.4)

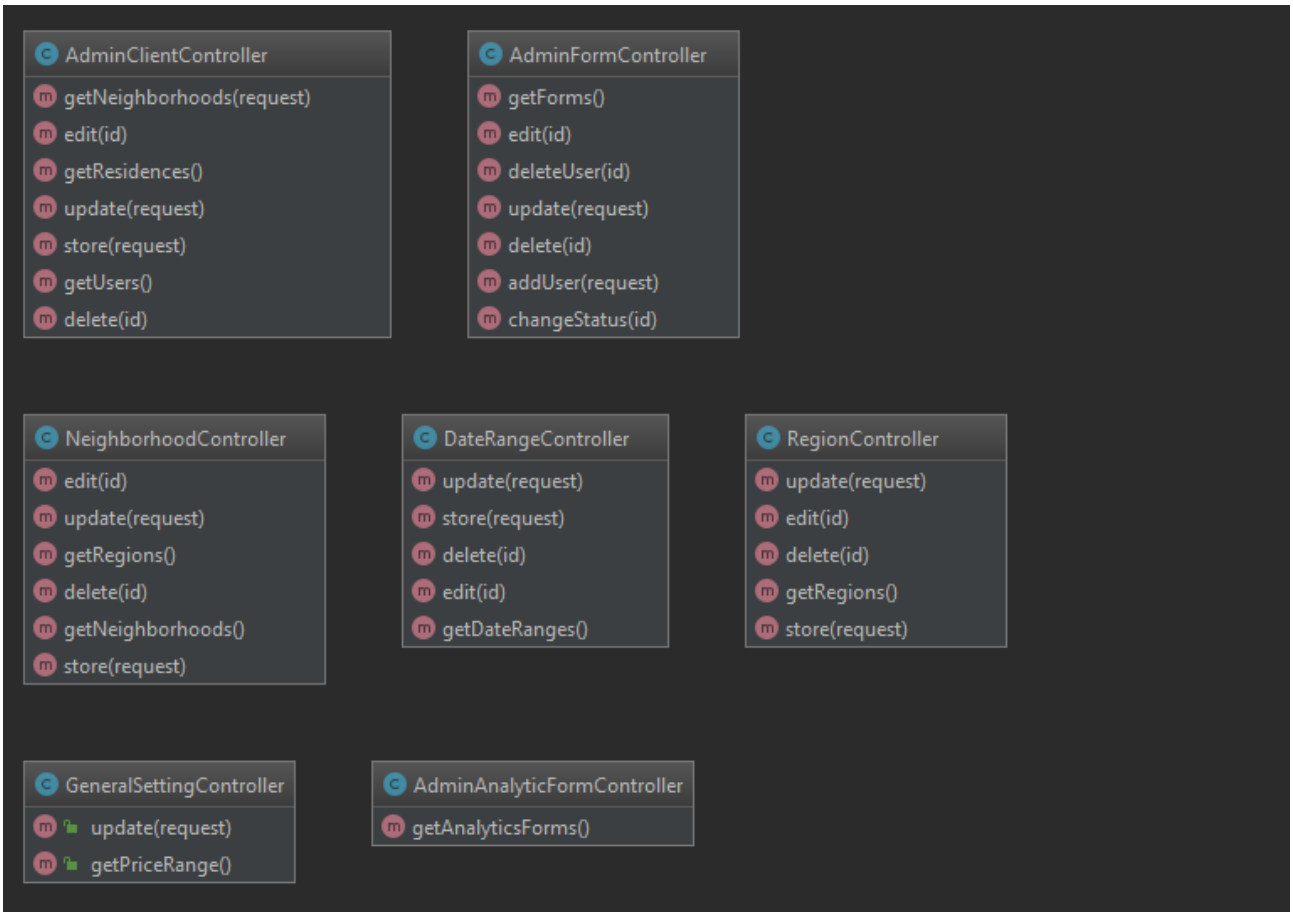


Рисунок 2.4 – Контролери адмін частини

Почнемо створення наших контролерів з AdminClientController даний контролер буде одним з основних для створення, редагування, видалення адміністраторів, продавців та орендодавців він буде мати наступні методи:

- getUsers(GET) – для отримання користувачів, з можливістю фільтрування їх по імені, email, та по ролі;
- getNeighborhoods(GET) – отримання регіонів для селекту;
- store(POST) – створення адміністратора, продавця, орендодавця;
- edit(GET) – отримання інформації якогось user для редагування;
- update(POST) – редагування user;
- delete(DELETE) – видалення user.

Наступні два контролера це котнролери пов'язані з локацією, а саме RegionController та NeighborhoodController.

Для RegionController ми маємо стандартні CRUD методи, а саме:

- getRegions(GET) – отримання регіонів;
- store(POST) – створення регіону;
- edit(GET) – отримання якогось регіону для редагування;
- update(POST) – редагування регіону;
- delete(DELETE) – видалення регіону.

NeighborhoodController буде мати такі методи:

- getNeighborhoods(GET) – отримання району;
- getRegions(GET) – отримання регіонів для селекту;
- store(POST) – створення району;
- edit(GET) – отримання якогось району для редагування;
- update(POST) – редагування району;
- delete(DELETE) – видалення району.

Наступним контролером буде DateRangeController з методами:

- getDateRanges(GET) – отримання дат;
- store(POST) – створення дати;
- edit(GET) – отримання дати для редагування;
- update(POST) – редагування дати;
- delete(DELETE) – видалення дати.

GeneralSettingController буде з двома методами:

- getPriceRange(GET) – отримання цінового діапазону;
- update(POST) – оновлення якоїсь настройки.

В даному контролері є лише update тому що наші настройки будуть в базі як key і value і не буде можливості створювати щось нове. Детальніша реалізація

буде в 3 розділі. І останній наш контролер це AdminAnalyticFormController з одним методом getAnalyticsForms для отримання форм з аналітикою.

З контролерами для адмін частини ми завершили, наступний контролер і лише один єдиний ClientFormController для форми (Рис. 2.5).

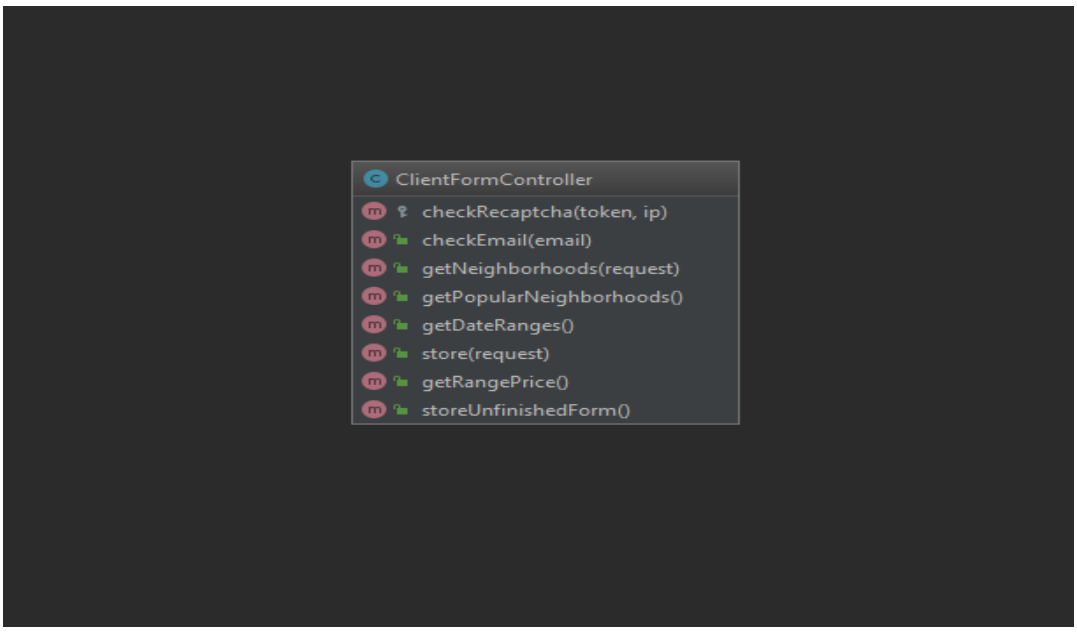


Рисунок 2.5 – Контролери форми

В даному контролері будуть наступні методи:

- CheckRecaptcha – на нашій формі буде recaptcha від google і даний метод буде слугувати перевіркою правильності каптчі;
- checkEmail(GET) – при зміні поля email буде відправлятися запит, для того щоб перевірити чи існує вже форма з таким email, і якщо існує зразу буде показувати помилку;
- getNeighborhoods(GET) – отримання районів для селекту;
- getPopularNeighborhoods(GET) – отримання самих популярних районів;
- getDateRange(GET) – отримання дат у випадку, якщо це включено з адмін частини;
- getRangePrice(GET) – отримання діапазону ціни;
- storenUnfinishedForm(POST) – після того як у формі буде введено name, email та phone number, то на даному етапі вже створиться форма і при

подальшому заповнені вона буде обновлятися і в подальшому адміністратори зможуть переглядати ці незавершені форми і бачити на якому етапі зупинився клієнт;

- store(POST) – цей метод вже буде зберігати кінцеву форму, а також запускати алгоритм підбору продавця або орендодавця і буде відправляти email адміністраторам.

2.1.3 Frontend

Для розробки frontend частини ми будемо використовувати фреймворк Vue.js. Даний фреймворк включає в себе широку функціональність для рівня представлення і чудово підходить для SPA (single page application). Функції Vue.js:

- реактивність інтерфейсів;
- декларативний рендеринг;
- зв'язування даних;
- директиви (всі директиви мають префікс «V-». В директиву передається значення стану, а в якості аргументів використовуються html атрибути або Vue JS події);
- логіка шаблонів;
- компоненти;
- обробка подій;
- переходи і анімації CSS;
- фільтри [10].

Отже, нам потрібно створити компоненти, ми їх розділимо на 2 частини аналогічно як і для Backend. Admin Компоненти для адмін частини, Client компоненти для форми. Для адмін частини для зручності ми їх також по

групуємо ще по папках для кращого розуміння. Отже перші компоненти будуть в папці layouts (Рис. 2.6)

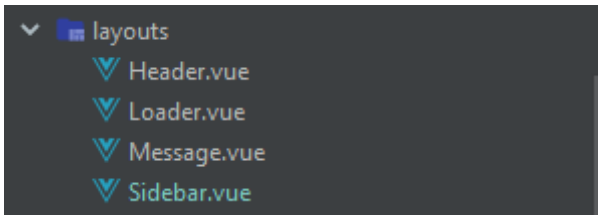


Рисунок 2.6 – Layouts компоненти

З компонентами Header, Loader, Message, Sidebar. Дані компоненти є загальними і однаковими для всіх сторінок. Наступні папки і компоненти є одинакові майже для всіх наших папок. Візьмемо за приклад clients в даній папці будуть компоненти Client це сторінка з таблицею всіх користувачів, ClientAdd сторінка створення клієнта, та ClientEdit сторінка для редагування. Так як структура в більшості папках схожа, то всю структуру компонентів можна побачити на наступному рисунку (Рис. 2.7)

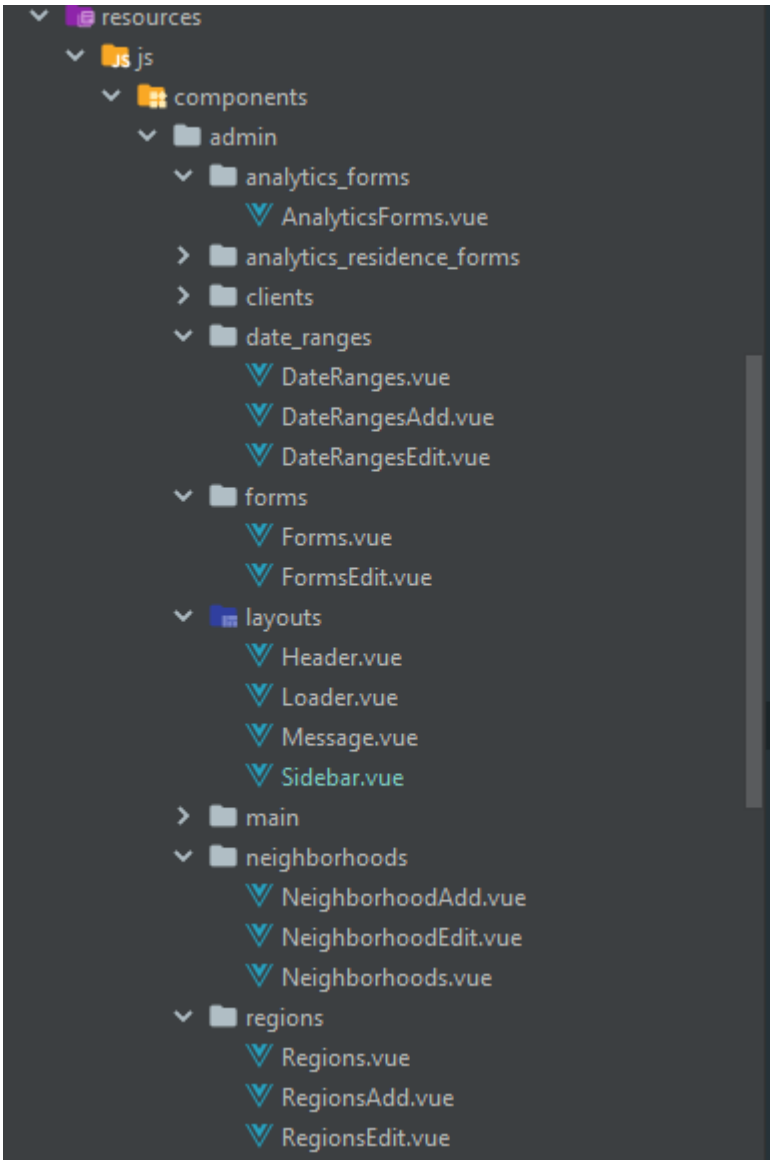


Рисунок 2.7 – Структура компонентів

Для клієнт частини, тобто форми потрібен лише компонент ClientForm, де буде знаходитись наша форма.

2.2 Структура бази даних

База даних (англ. database) – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування. В загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім саме даних, містить їх опис та може містити засоби для їх обробки.

В загальному випадку базою даних можна вважати будь-який впорядкований набір даних. Наприклад, паперову картотеку з формулярами про працівників підприємства у відділі кадрів. На даний час застосунки для роботи з базами даних є одними з найпоширеніших прикладних програм [15].

Структура бази даних буде реалізована в онлайн редакторі dbdesigner. Він є дуже зручним і простим у використанні. Першим кроком буде створення таблиці users, яка нам знадобиться для створення адміністраторів, продавців та орендодавці. Таблиця users буде мати наступні поля:

- id – первинний ключ;
- company_name – назва компанії;
- email – email користувача;
- page_url – посилання на нерухомість в інтернет магазині;
- description - додаткова інформація про користувача або його нерухомість;
- password – пароль користувача;
- role – роль користувача (адміністратор чи клієнт);
- buy – чи продає він нерухомість;
- rent – чи здає в оренду нерухомість;
- price_min – мінімальна ціна;
- price_max – максимальна ціна;
- counter – кількість отриманих клієнтів від форми;

- max_forms – максимально можлива кількість клієнтів з форм отриманих за місяць;
- offline – можливість деактивувати клієнта;
- created_at – дата і час створення користувача;
- updated_at – дата і час обновлення користувача.

Це мінімальний набір полів які необхідні нам для нашого функціоналу (Рис 2.8). В подальших таблицях такі поля як id, created_at та updated_at будуть опускатися, оскільки є одинакові для всіх таблиць.

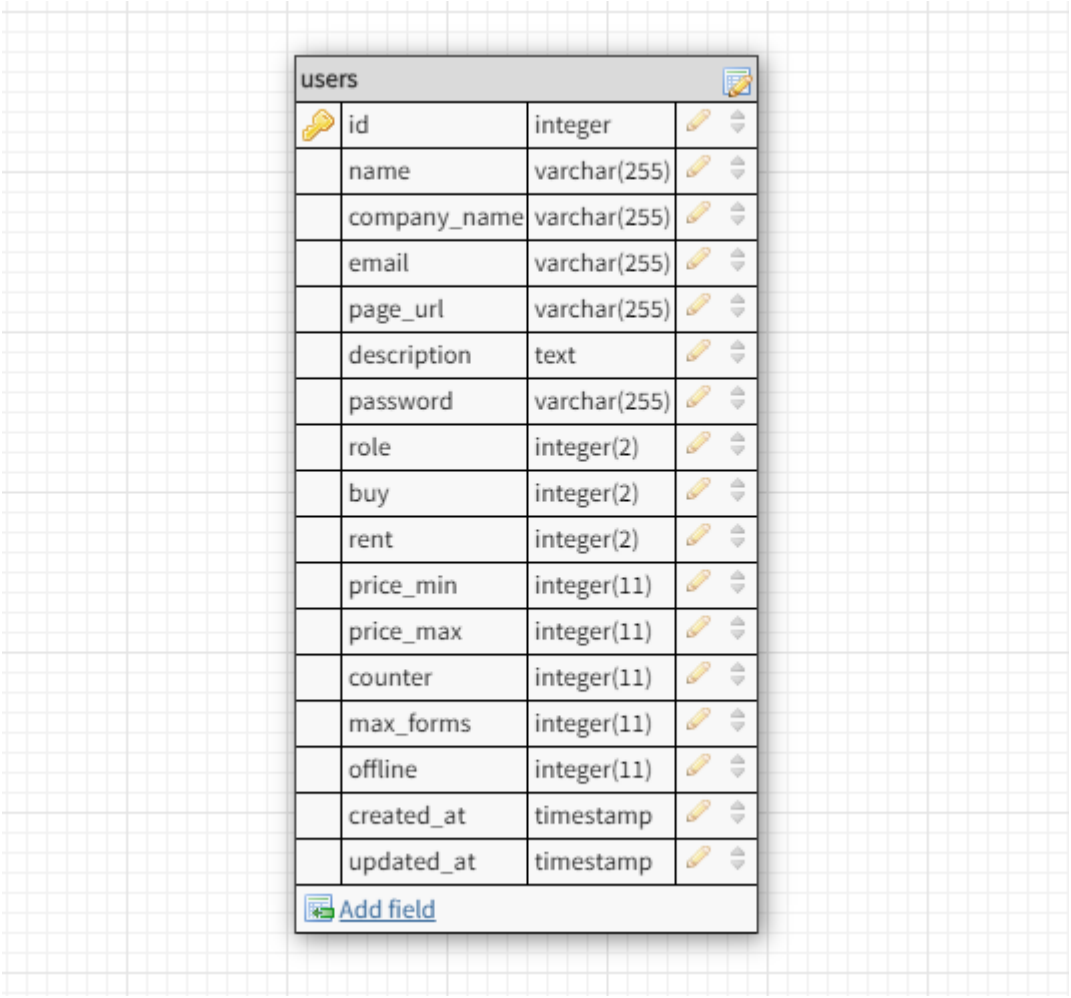


Рисунок 2.8 – Таблиця users

Наступним кроком буде створення таблиць regions та neighborhoods. Ці таблиці потрібні для того, щоб при створенні користувачів адміністратори додавали в якому регіоні і в якому районі знаходиться їхня нерухомість, а також

це нам знадобиться для форми, коли клієнти будуть обирати, в якому місці вони шукають нерухомість. Отже таблиця regions буде з наступними полями:

- name_en – назва регіону на англійській;
- name_fr – назва регіону на французькій.

А таблиця neighborhoods буде в себе включати поля:

- name_en – назва регіону на англійській;
- name_fr – назва регіону на французькій;
- region_id – зв’язок з таблицею regions, тобто до якого регіону належить даний район.

Оскільки зв'язок між users і neighborhoods many to many, то нам знадобиться проміжна таблиця users_neighborhood. Також для користувачів, нам необхідно створити таблицю users_bedroom, для того щоб знати скільки кімнат він може запропонувати клієнтам. В даній таблиці будуть поля:

- user_id – зв’язок з таблицею users;
- bedroom – кількість кімнат.

Це основні таблиці які необхідні при створенні користувача. Після створення даних таблиць ми будемо мати архітектуру такого вигляду (Рис 2.9)

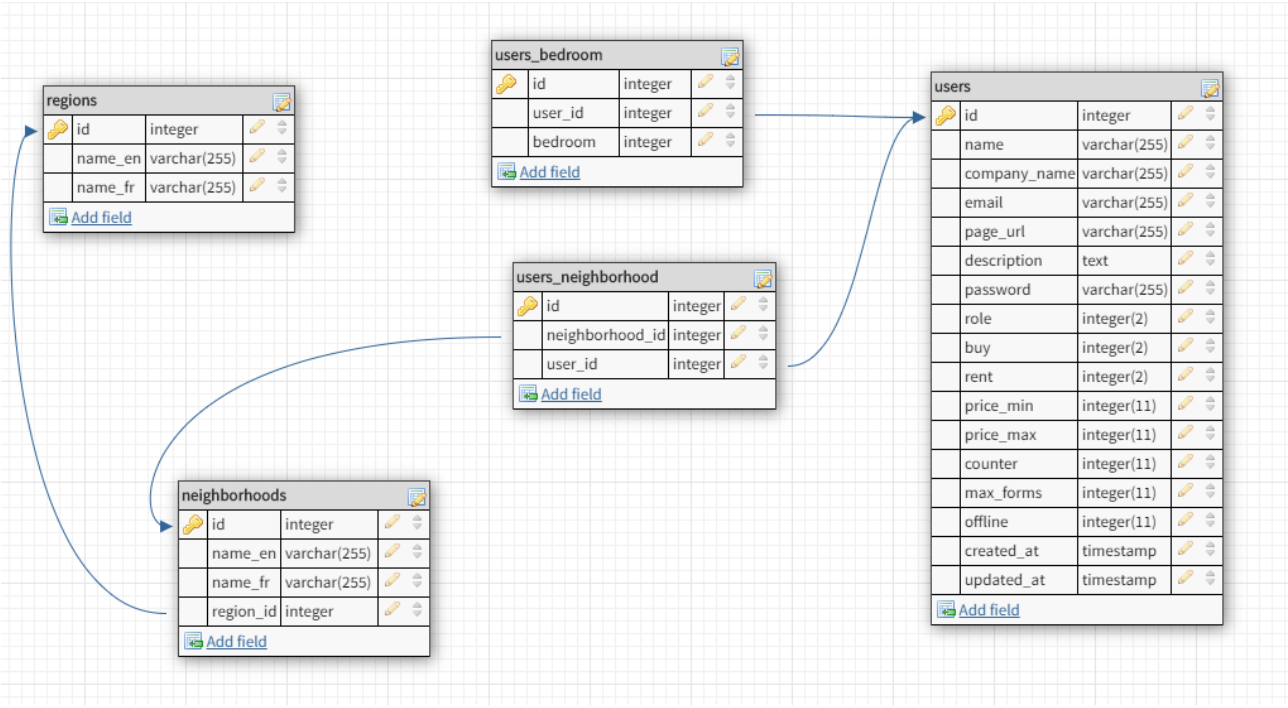


Рисунок 2.9 – Архітектура бази даних

Наступним нашим кроком буде створення таблиць, які будуть нам необхідні для форми. Основну таблицю для форми ми назвемо client_forms і вона буде в себе включати такі поля як:

- name – ім’я користувача;
- email – email користувача;
- phone_number – мобільний телефон користувача;
- buy – чи хоче користувач купити нерухомість;
- rent – чи хоче користувач орендувати нерухомість;
- sale – чи має користувач якусь свою нерухомість для продажу (для варіанту з обмінами);
- price_min – мінімальна ціна;
- price_max – максимальна ціна;
- date_from, date_to – період з і до;
- date_range_id – посилання на дату, яка буде створюватися в адмін частині, по функціоналу буде можливість обрати, який вид дати буде показуватися на формі чи date range select чи select з якимись name з адмін частини.

- additional_info – додаткова інформація;
- status – статус на якому етапі буде знаходитися форма;
- language – на якій мові була відправлена форма;
- is_check – чи ця форма була вже переглянута адміністратором чи ні.

Як було сказано в описі до поля date_range_id в нас повинна бути можливість створювати якийсь період дати в адмін частині. Для цього ми створимо таблицю date_ranges з полями:

- name_en – назва на англійській мові;
- name_fr – назва на французькій мові.

А також нам потрібна таблиця client_forms_bedroom так як у формі буде можливість обрати деяку кулькість кімнат. В цій таблиці будуть поля:

- client_form_id – зв’язок з таблицю client_forms;
- bedroom – кількість кімнат.

І останній наш крок це буде створення таблиці client_form_user для зв'язку many to many форми з продавцями чи орендодавцями. Кінцевий вигляд нашої структури бази даних(Рис. 2.10).

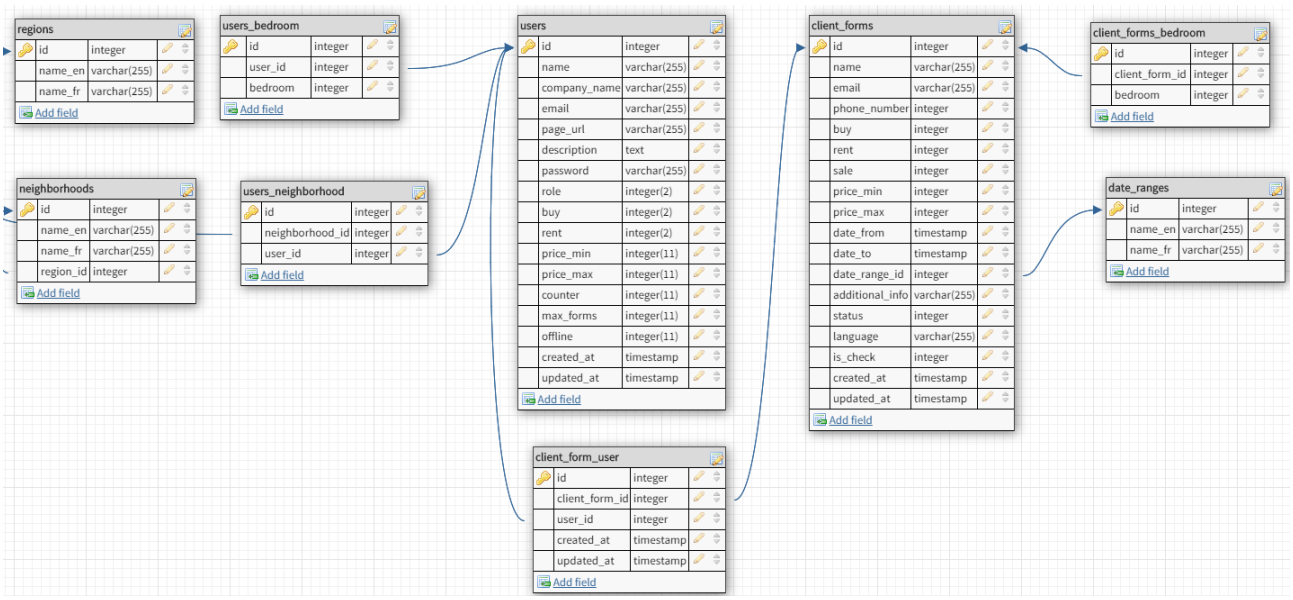


Рисунок 2.10 – Архітектура бази даних

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В попередньому розділі ми розробили структуру нашого проекту, а також створили моделі, контролери та компоненти, які нам будуть необхідні при реалізації проекту. Ціллю даного розділу буде розробка вже безпосередньо нашого веб додатку. Конкретно дана форма буде розроблятися для канадського інтернет магазину Montreal Guide Condo [5].

3.1 Розробка адмін частини

Перш за все нам потрібно розробити сторінку входу в адмін частину, це буде звичайна форма з логіном, паролем і кнопкою login (Рис 3.1).

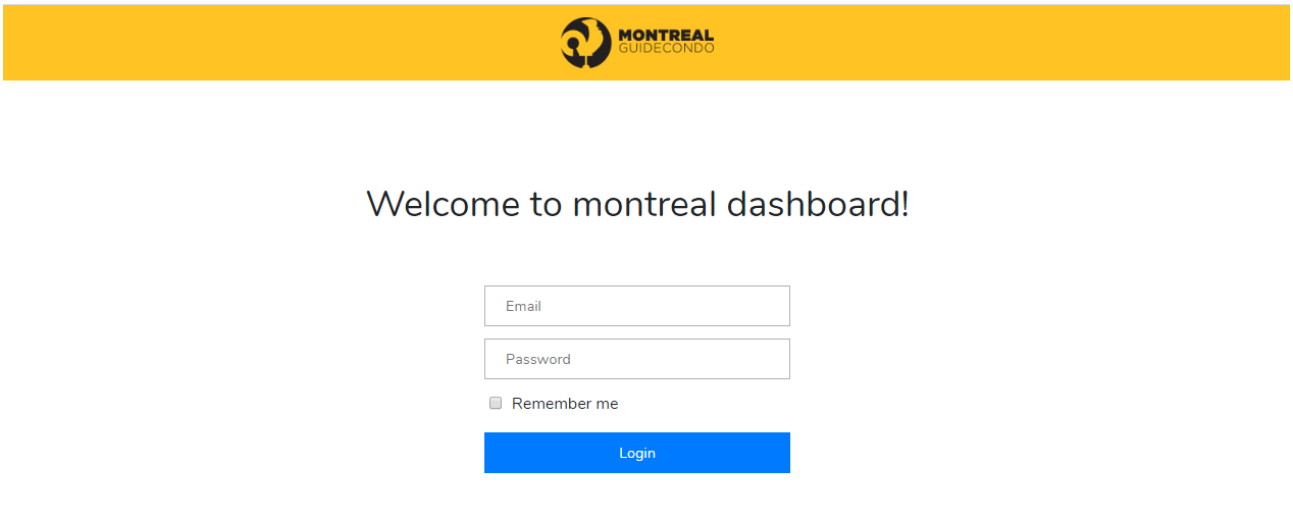


Рисунок 3.1 – Сторінка входу в адмін частину

Для реалізація входу використовується бібліотека jwt-auth для backend частини та vue-auth для frontend. Алгоритм входу відбувається наступним чином, користувач вводить логін і пароль і жме кнопку login. З фронтенда

відправляється запит на url /login, в методі login ми логінемо користувача, тобто створюється token, який відправляється у відповідь на frontend і зберігається в localstorage. Роботу метода login на frontend можна побачити на (Рис. 3.2)

```
login() {
    this.$auth.login({
        params: this.user,
        rememberMe: true,
        success: response => {
            let user = response.data;
            this.localStorage();
            localStorage.setItem('user', JSON.stringify(user));
            this.$router.push({
                name: 'forms'
            });
        },
        error: error => {
            let allErrors = error.response.data.errors;
            this.message.outputMessage = allErrors[
                Object.keys(allErrors)[0]
                ][0];
            this.message.type = 'error';
            this.message.active = true;
            setTimeout(() => {
                this.message = {active: false};
            }, 3000)
        }
    })
},
```

Рисунок 3.2 – Метод логінення на frontend

А роботу методу login на backend можна побачити на (Рис 3.3)

```php
public function login(RequestLogin $request)
{
  $credentials = [
    'email' => request('email'),
    'password' => request('password')
  ];

  $user = User::where('email', $credentials['email'])->first();
  if(@$user->role === 2) {
    if ($token = $this->guard()->attempt($credentials)) {
      $user = User::where('email', $credentials['email'])->first();

      return response()
        ->json($user, 200)
        ->header('Authorization', $token);
    }
  }
    return response()->json(['errors'
        => ['message' => ['Incorrect email or password. Please try again!']]], 401);
}
```

Рисунок 3.3 – Метод логінення на backend

Після цього, нам потрібно розробити sidebar та header для нашої адмін частини, так як у нас вимог до дизайну немає, то будем робити все мінімалістично, але щоб було приємне для сприйняття. Також в sidebar створемо меню для нашого функціоналу. В результаті отримали вигляд на (Рис 3.4)

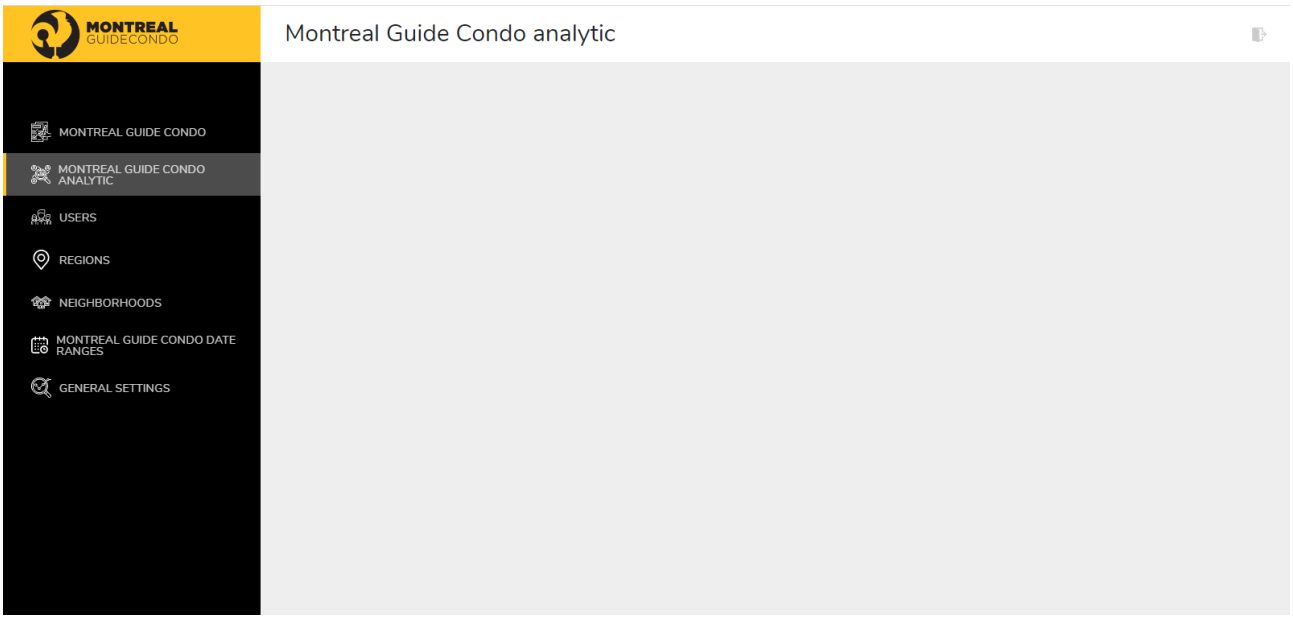


Рисунок 3.4 – Вигляд адмін панелі

3.1.1 CRUD для users

Так як ми вже створили меню для адмін частини, наступним кроком буде створення нових адміністраторів, продавців, орендодавців в меню це Users. Сторінка для створення user буде мати такий вигляд при створенні адміністратора (Рис 3.5).

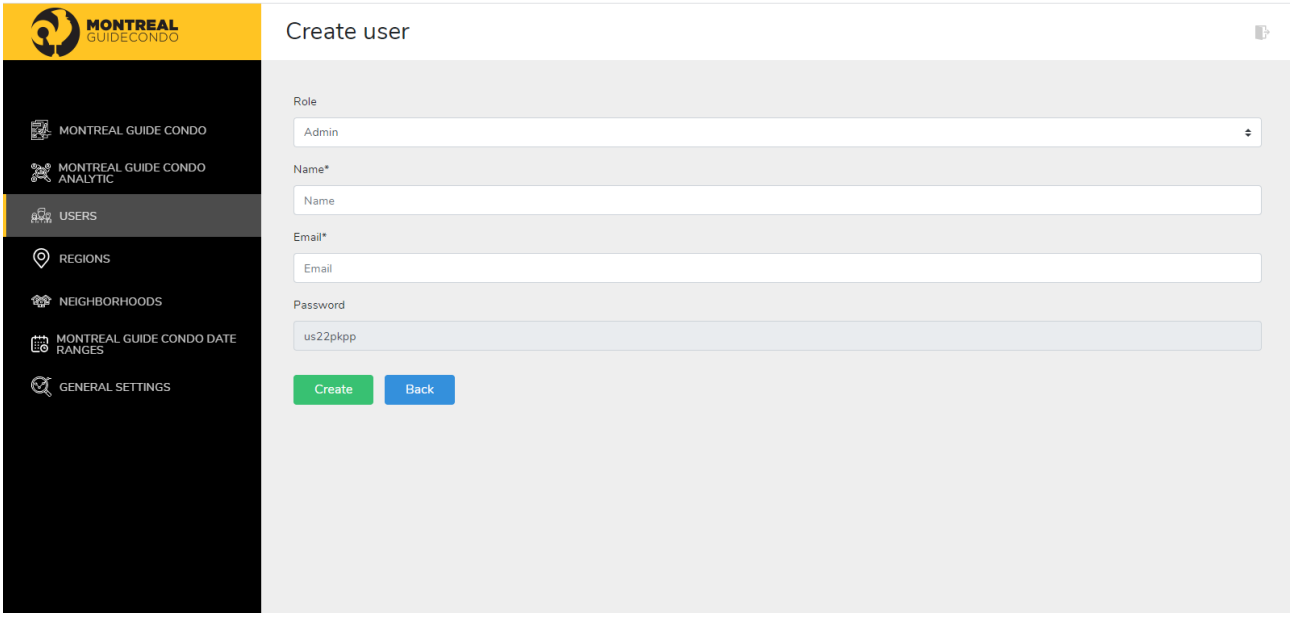


Рисунок 3.5 – Форма для створення адміністратора

А при зміні селекту role на client форма буде мати вигляд (Рис 3.6)

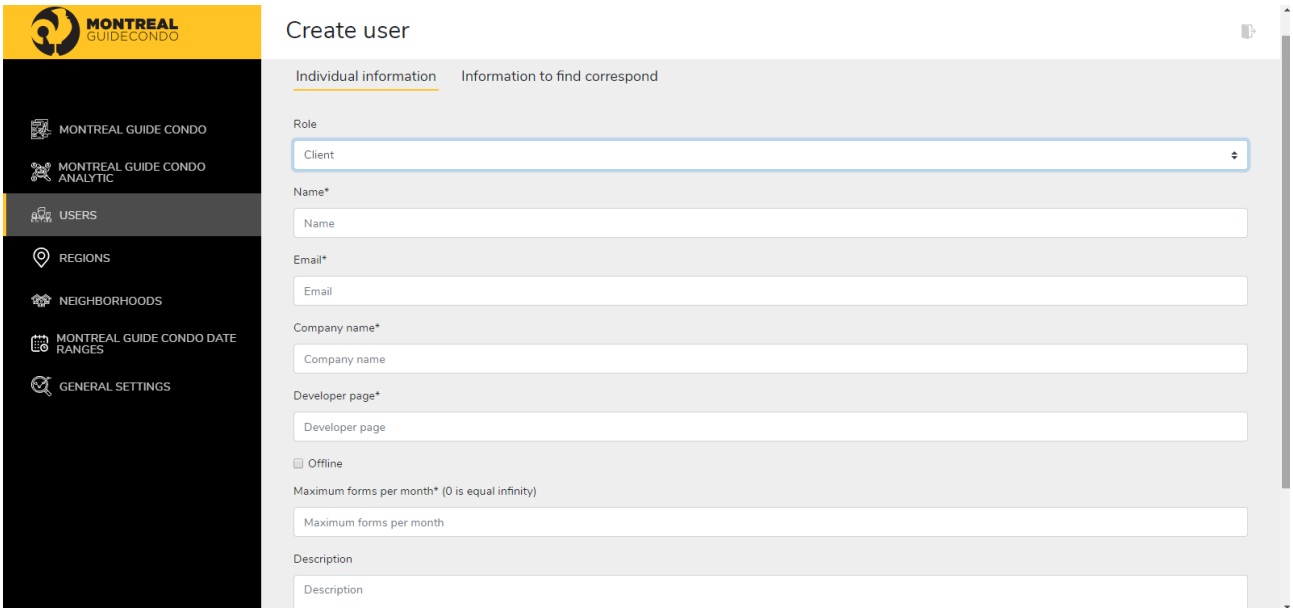


Рисунок 3.6 – Форма для створення користувача

Поля ми створили відповідно до структури нашої бази даних, також можна помітити, що зверху є друга вкладка information to find correspond (Рис 3.7)

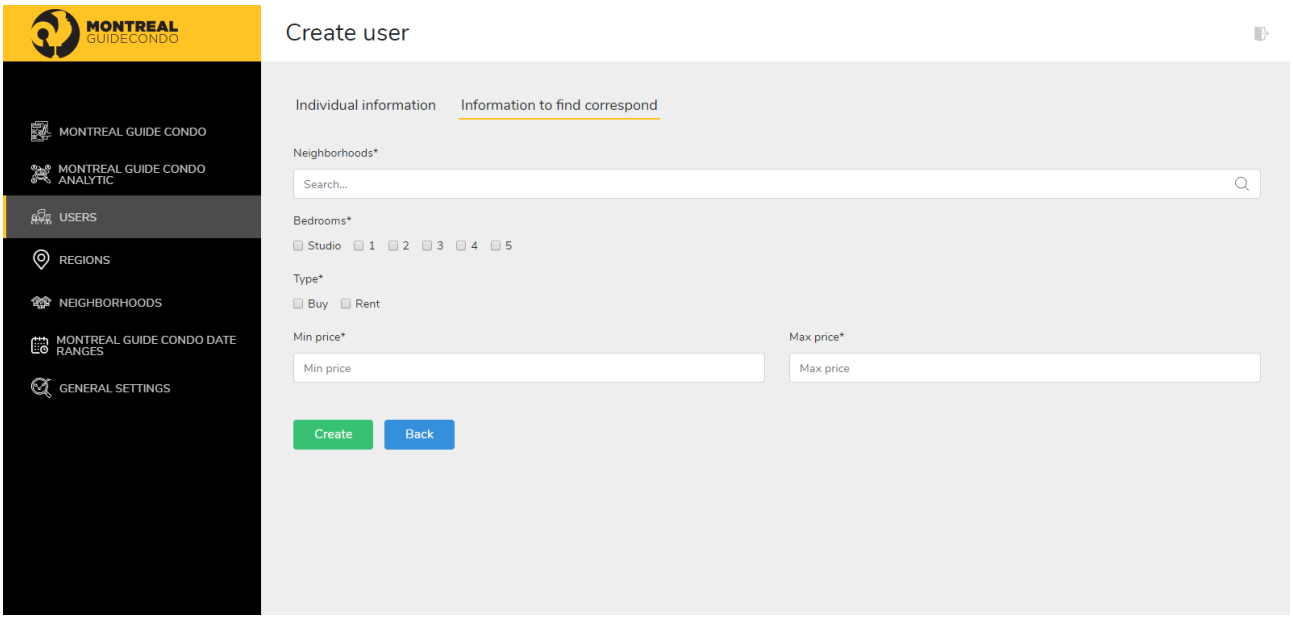


Рисунок 3.7 – Вкладка information to find correspond

Як можна побачити з рисунка на даній вкладці є такі поля як:

- neighborhoods;
- bedrooms;
- type;
- min price;
- max price.

Ці поля необхідні для створення продавців і орендодавців і по цих полях алгоритм буде підбирати їм клієнтів. Після заповнення цих полів ми відправляємо request по url adminpanel/clients/create. У випадку якщо ми введемо неправильні дані ми отримаємо помилку перевірки і покажемо дані помилки знизу наших fields (Рис. 3.8)

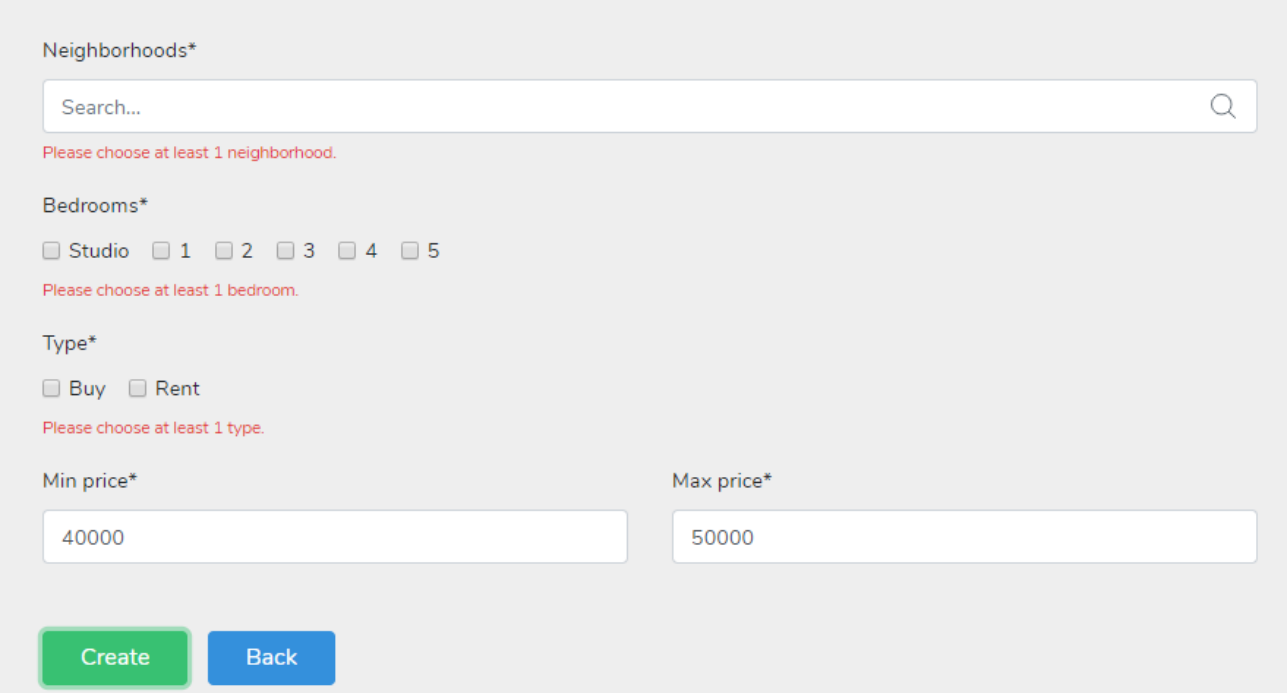


Рисунок 3.8 – Приклад перевірки

Якщо перевірка пройшла успішно ми повинні створити нашого користувача в Laravel. Створення є дуже простим нам всього лиш потрібно в моделі user викликати один з методів (create, insert, insertGetId) і передати масив зі значеннями ключ (поле в базі даних) і значення приклад (рис 3.9)

```php
$user_id = User::insertGetId([
    'name' => $userData->name,
    'company_name' => $userData->company_name,
    'email' => $userData->email,
    'page_url' => $userData->page_url,
    'description' => $userData->description,
    'role' => $userData->role,
    'rent' => isset($userData->rent) ? 1 : 0,
    'buy' => isset($userData->buy) ? 1 : 0,
    'price_min' => $userData->price_min,
    'price_max' => $userData->price_max,
    'max_forms' => $userData->max_forms,
    'offline' => (@$userData->offline ? 1 : 0),
    'created_at' => Carbon::now(),
    'updated_at' => Carbon::now()
]);
```

Рисунок 3.9 – Приклад створення user

Після створення наших користувачів нам потрібно їх вивести. Виведення ми зробимо в таблиці з можливостю пошуку користувачів по полях name, email та role, також добавимо кнопку, яка буде посилатися на сторінку створення користувача і збоку добавимо кнопку з посиланням на сторінку редагування і кнопку для видалення користувача з бази даних. Вигляд таблиці буде мати наступний вигляд (Рис 3.10).

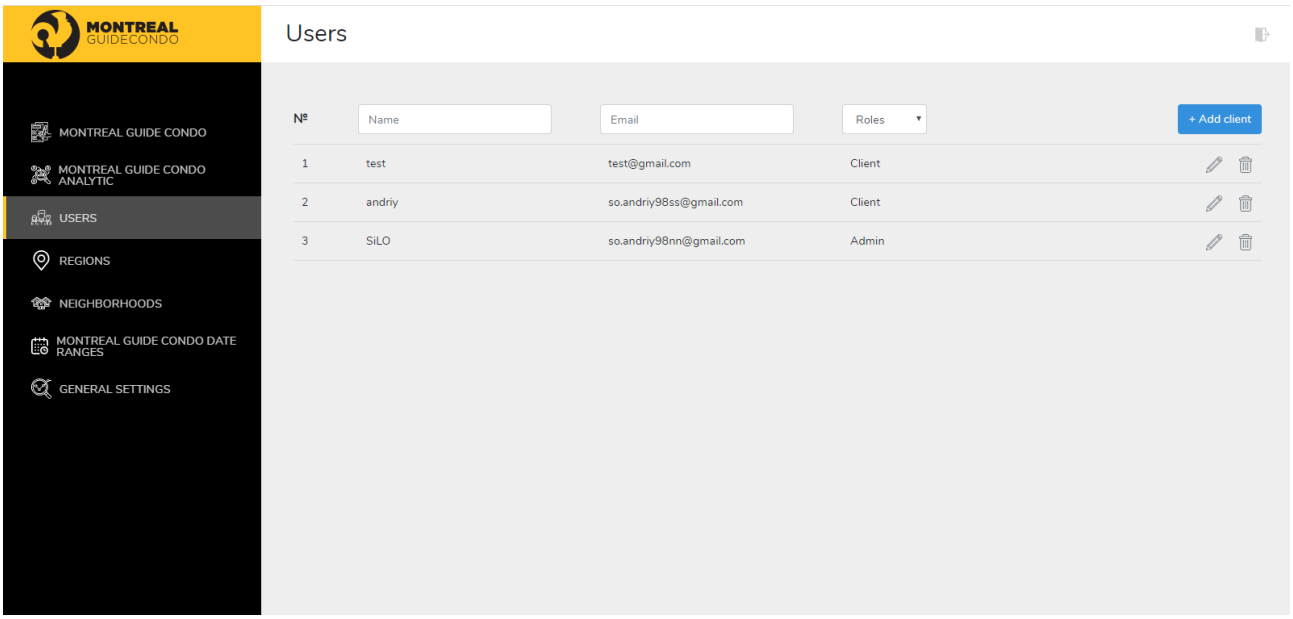


Рисунок 3.10 – Сторінка виводу користувачів

Для отримання користувачів відправляється request по url adminpanel/clients/ з такими query параметрами:

- page – номер сторінки;
- searchName – текст з поля name;
- searchEmail – текст з поля email;
- role – вибрана роль з селекту.

Метод getUser буде мати наступний вигляд (Рис. 3.11)

```
getUsers(page = 1) {
    this.loader = true;
    let url = 'adminpanel/clients/?page='
        + page
        + '&searchName=' + this.searchName
        + '&searchEmail=' + this.searchEmail
        + '&role=' + this.role;

    axios.get(url)
        .then(response => {
            this.users = response.data.users;
            this.pagination = response.data.pagination;
            this.loader = false;
        });
},
```

Рисунок 3.11 – Запит на отримання користувачів

Щоб отримати наших користувачів з бази даних потрібно написати SQL запит. Laravel пропонує свої методи для того, щоб зробити розробку максимально простою і зручною. Отже, наш запит буде мати вигляд (Рис 3.12)

```php
protected function getClients($search)
{
    $name = $search->searchName;
    $email = $search->searchEmail;
    $role = $search->role;

    $users = $this
        ::select(
            'users.*'
        )
        ->when($name, function ($query) use ($name) {
            $query->where('name', 'LIKE', "%$name%");
        })
        ->when($email, function ($query) use ($email) {
            $query->where('email', 'LIKE', "%$email%");
        })
        ->when($role, function ($query) use ($role) {
            $query->where('role', $role);
        })
        ->orderBy('id', 'desc')
        ->where('name', '!=' , 's-admin')
        ->paginate(10);
```

Рисунок 3.12 – Запит для отримання users.

Тепер нам залишилося реалізувати редагування та видалення користувачів. Сторінка редагування буде виглядати аналогічно як сторінка для створення. Різниця буде лише в тому, що спочатку нам потрібно отримати дані конкретного користувача, після чого заповнити ними поля і після того, коли поля будуть змінені відправити request на update даного користувача.

Для того щоб отримати конкретного користувача нам потрібна його id, яке ми отримуємо в нашій таблиці. Після відправлення request на url /adminpanel/clients/edit/{id} нам потрібно отримати дані користувача (Рис. 3.13)

```php
public function edit($id)
{
    $user = User::find($id);
    $neighborhoods = UserNeighborhood::getNeighborhoodForUser($id);
    $typeOfResidences = UserTypeOfResidence::getTypeOfResidenceForUser($id);
    $bedrooms = UserBedroom::where('user_id', $id)->get();
    $decrypt_pass = base64_decode($user->encrypt_password);

    return response()->json([
        'user' => $user,
        'neighborhoods' => $neighborhoods,
        'decryptPass' => $decrypt_pass,
        'typeOfResidences' => $typeOfResidences,
        'bedrooms' => $bedrooms
    ]);
}
```

Рисунок 3.13 – Отримання користувача по id

В Laravel, щоб отримати користувача по id достатньо викликати метод find і передати туди id. Також ми дістаємо інформації з інших таблиць, а саме вибрані регіони і кількість кімнат даного користувача. Після того як користувач натисне кнопку save буде відправлений request на url /adminpanel/clients/update з новими даними і метод update в контролері збереже наші нові дані (Рис 3.14).

```
User
    ::find($userData->id)
    ->update([
        'name' => $userData->name,
        'company_name' => $userData->company_name,
        'email' => $userData->email,
        'page_url' => $userData->page_url,
        'description' => $userData->description,
        'role' => $userData->role,
        'rent' => $userData->rent ? 1 : 0,
        'buy' => $userData->buy ? 1 : 0,
        'price_min' => $userData->price_min,
        'price_max' => $userData->price_max,
        'max_forms' => $userData->max_forms,
        'offline' => (@$userData->offline ? 1 : 0),
        'updated_at' => Carbon::now()
    ]);
```

Рисунок 3.14 – Оновлення даних для користувача

Останнє, що нам залишилося реалізувати - це видалення користувача. Видалення буде відбуватися при кліку на корзину в таблиці users, щоб уникнути випадкового кліку і видалення користувача ми добавимо alert з текстом “are you sure?” і якщо адміністратор вибере “yes”, то тоді буде відправлений request на url /adminpanel/clients/{id}, в нашому методі delete буде все максимально просто, так як видалення в laravel відбувається викликанням 1 метода delete (Рис 3.15)

```
public function delete($id)
{
    User::find($id)->delete();

    return response(200);
}
```

Рисунок 3.15 – Метод видалення користувача

3.1.2 CRUD для regions, neighborhoods та date ranges

Після того як ми розробили створення користувачів, то тепер створення regions, neighborhoods та date ranges буде максимально простим, оскільки в них буде мало полів. Почнемо зі створення регіонів, нам тільки потрібно поле з назвою регіону на 2 мовах (Рис 3.16)

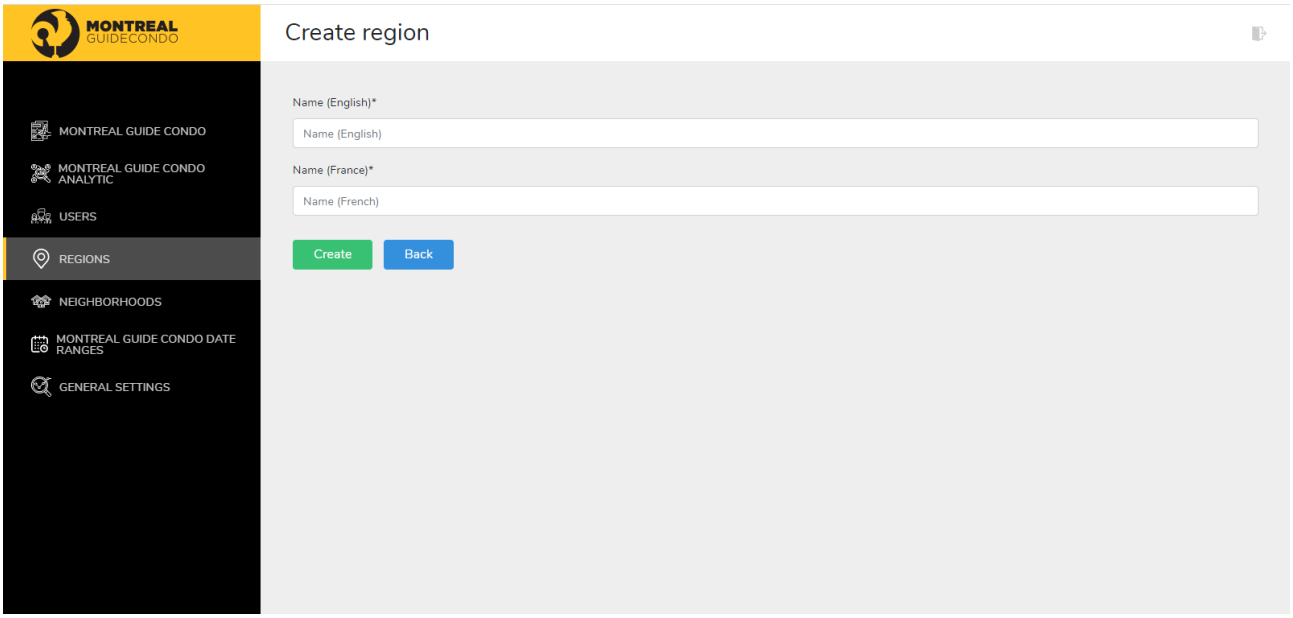


Рисунок 3.16 – Створення регіону

Після натискання на кнопку create буде відправлятися request на url /adminpanel/regions/create з даними які були введені після чого на backend частині відбудеться створення нового регіону (Рис 3.17)

```php
public function store(Request $request)
{
    $region = json_decode($request->regionData);

    Region::create([
        'name_en' => $region->name_en,
        'name_fr' => $region->name_fr
    ]);

    return response()->json([
        'success' => 'Region was successfully create',
    ]);
}
```

Рисунок 3.17 – Створення регіону в базі даних

Коли ми створили декілька регіонів, нам потрібно їх вивести, використаємо як шаблон ту саму таблицю, яку ми розробили для виведення користувачів. Вигляд даної таблиці можна побачити на (Рис 3.18)

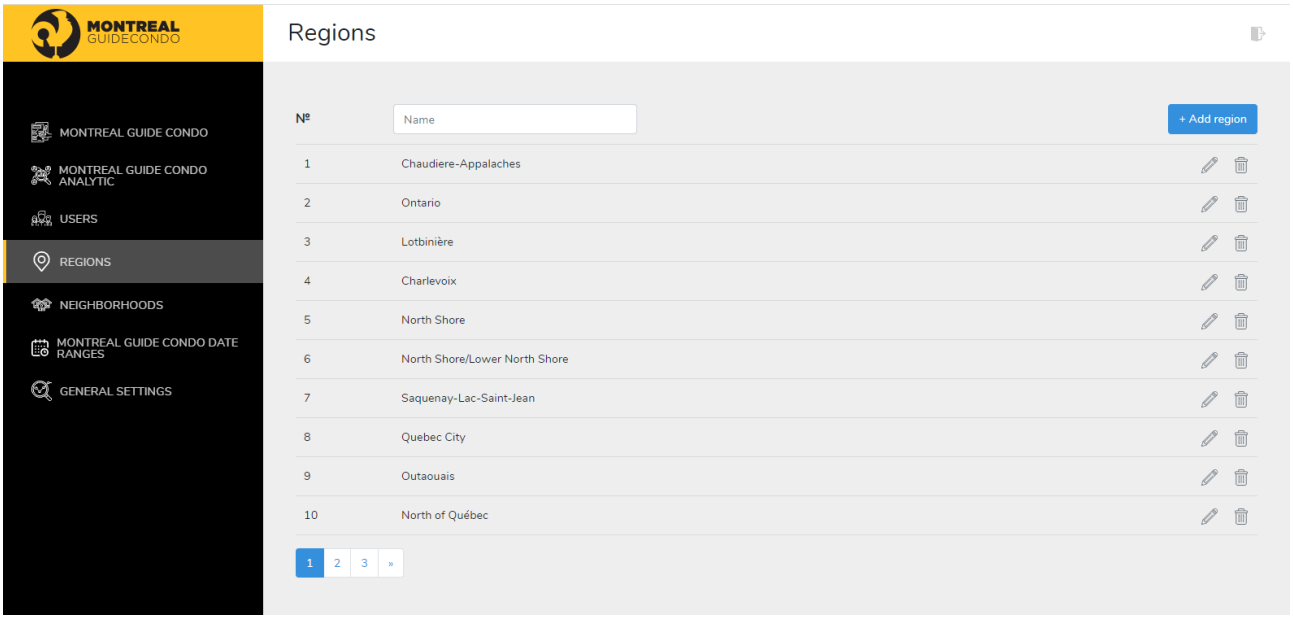


Рисунок 3.18 – Таблиця з регіонами

Наступний крок це редагування, по суті за аналогією як ми це робили в користувача, тому тут не буде нічого нового. По кліку на олівець ми робим перехід на сторінку з нашими полями, беремо назви регіону по id, дані заносяться в поля, після чого користувач може їх редагувати і після натискання кнопки save відправиться запит на збереження нових даних. Видалення має також аналогічну логіку як в користувача, при кліку на корзину в таблиці regions буде показуватися alert з повідомленням “Are you sure?” і якщо буде натиснута “Yes”, то відправиться запит на видалення і регіон буде видалено.

При створення date ranges в нас зміниться лише назва нашого поля name на date range, а алгоритм створення, виводу, редагування і видалення є аналогічним як для створення регіону, тому ми ці моменти опустимо. На наступному рисунку буде видна маленька різниця (Рис. 3.19).

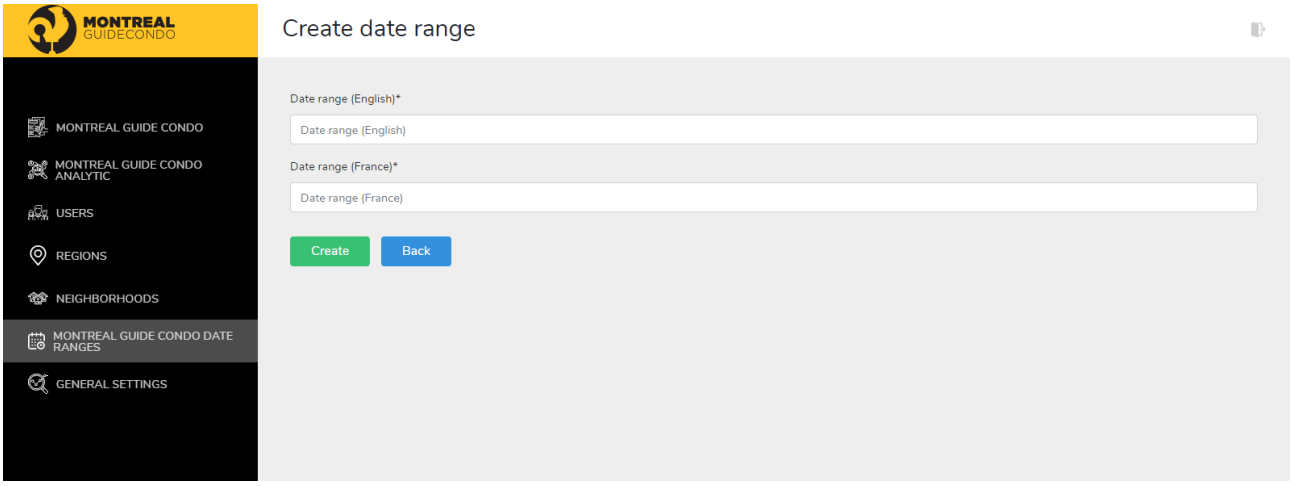


Рисунок 3.19 – Форма створення date range

Створення району, частково схоже до двох попередніх, за одним винятком, що район в нас буде прив’язаний до якогось регіону, тобто нам потрібно додати вибір, де буде можливість обрати до якого регіону належить даний район (Рис. 3.20)

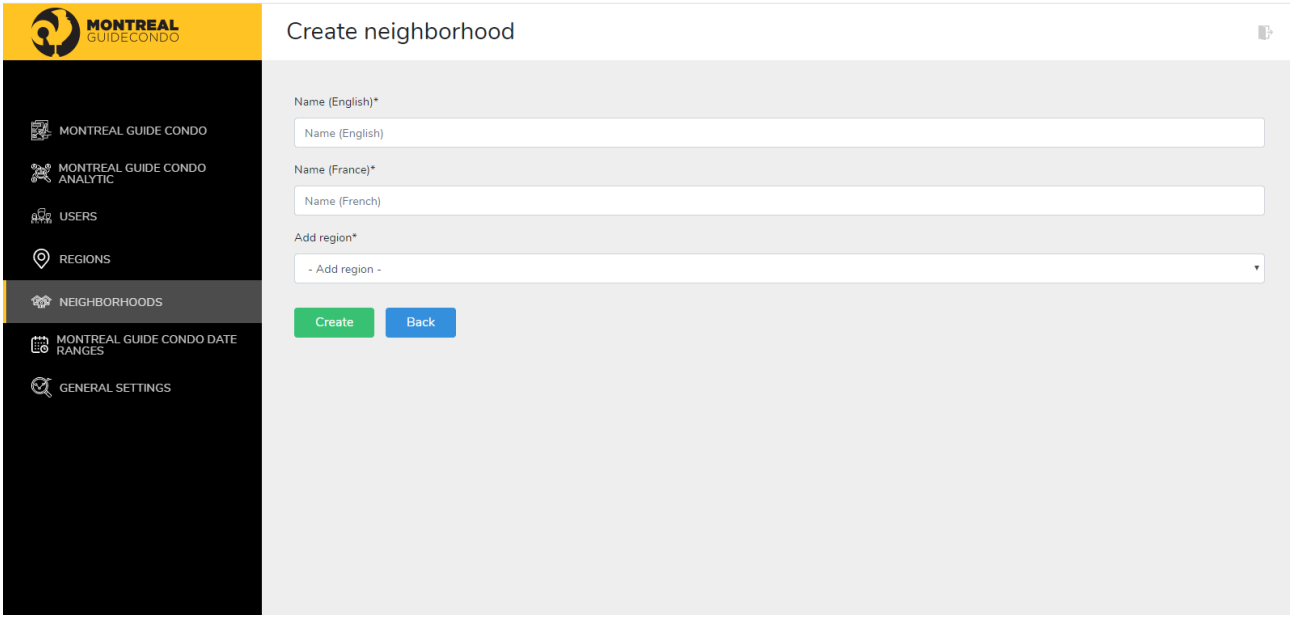


Рисунок 3.20 – Форма створення району

А також ми добавимо дане поле в нашу таблицю, для того щоб можна було обрати якийсь регіон і побачити всі райони які до нього відносяться (Рис. 3.21)

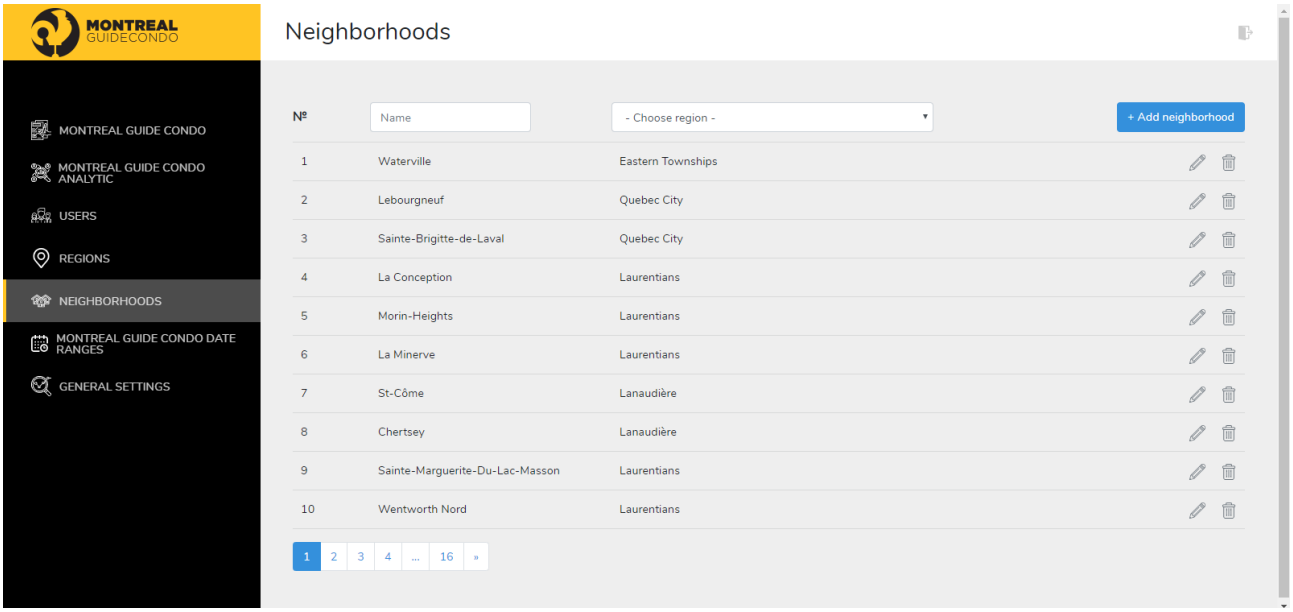


Рисунок 3.21 – Вивід районів

3.1.3 Вивід форм, та її аналітики

Коли користувачі будуть відправляти форму, нам потрібно буде їх показувати, відмічати якимось чином які нові і ще не перевірені, показувати статус форми, повинна бути можливість додати або видалити форму користувачу, а також десь показувати яким користувача яка форма була відправлена.

Почнемо ми з виводу форм, за основу візьмемо таблицю, яка ми розробили раніше для виводу user. Добавимо до неї такі стовбці як name, email, status, date. Коли ми будем переходити на дану сторінку буде відправлятися запит на url /adminpanel/forms/ для отримання наших форм. Запит буде мати вигляд (Рис 3.22)

```
$forms = $this
    ::select(
        'client_forms.id',
        'client_forms.name',
        'client_forms.email',
        'client_forms.status',
        'client_forms.is_check',
        'client_forms.created_at'
    )
    ->when($name, function ($query) use ($name) {
        $query->where('name', 'LIKE', "%$name%");
    })
    ->when($email, function ($query) use ($email) {
        $query->where('email', 'LIKE', "%$email%");
    })
    ->when($status, function ($query) use ($status) {
        $query->where('status', $status);
    })
    ->orderBy('is_check')
    ->orderBy('created_at', 'desc')
    ->paginate(10);
```

Рисунок 3.22 – Запит на отримання форм

Коли ми отримаємо наші форми ми їх виводимо в нашу таблицю

(Рис. 3.23)

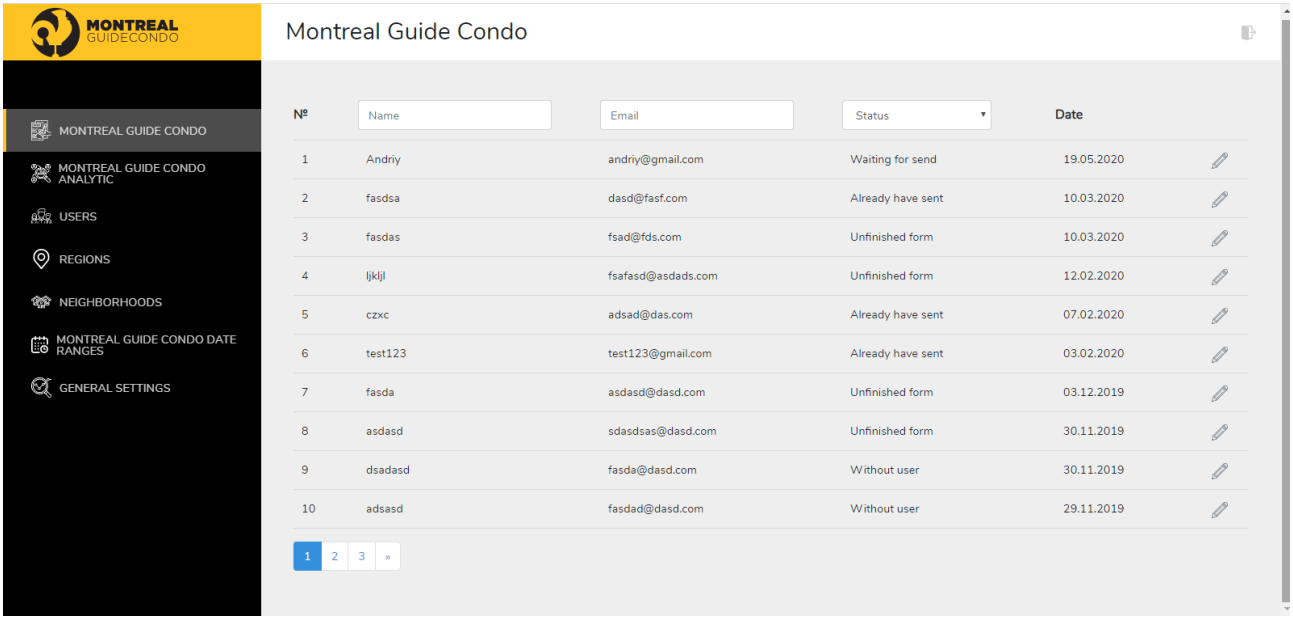


Рисунок 3.23 – Вивід форм

Форма може мати такі статуси:

- waiting for send – очікується відправка email з формою для користувача;
- already have sent – email з формою вже був відправлений;
- without user – даній формі не було підібрано користувача;
- unfinished – форма яка була незавершеною;

Наступним кроком буде створення сторінки, де буде виводитися вся основна інформація про форму. Ми вже знаємо, що можемо отримати дані форми за допомогою метода find, приклад можна подивитися при створенні користувачів в розділі 3.1.1. Після чого ми виведемо ці дані в поля і заберемо можливість їх редагувати за допомогою атрибута “readonly” (Рис 3.24)

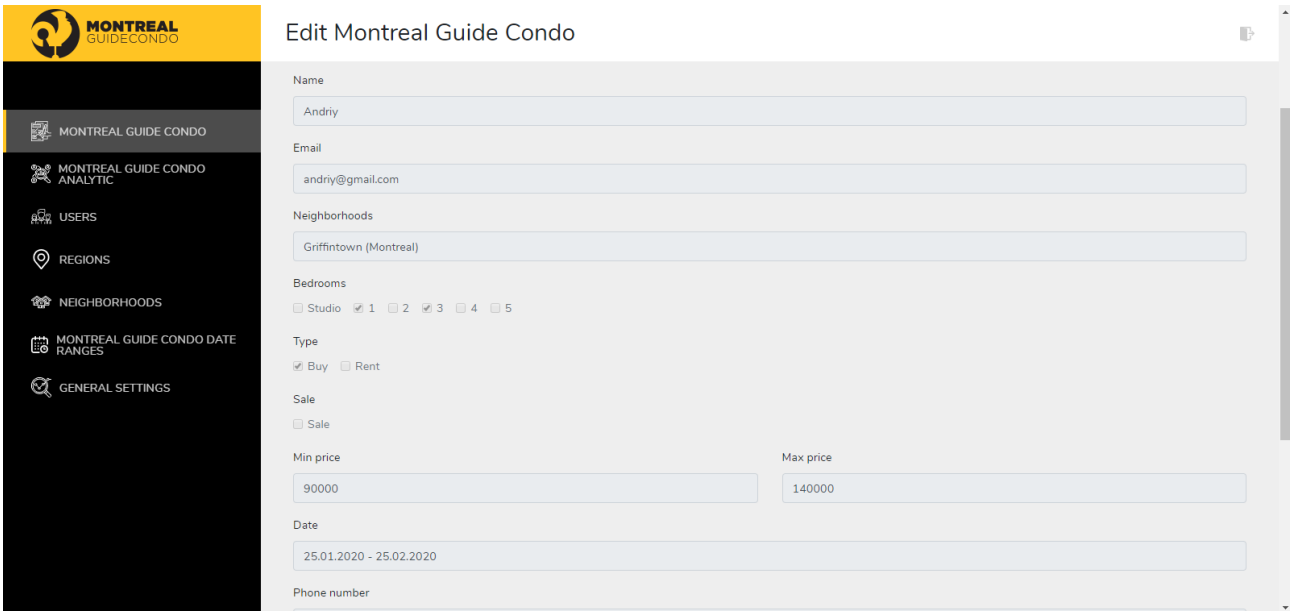


Рисунок 3.24 – Вивід детальної інформації про форму

На даній сторінці ми зразу добавимо можливість додавання і видалення користувача для даної форми. Отже, спочатку нам потрібно відправити запит для отримання користувачі, щоб вивести їх в селект і коли адміністратор буде вибирати нового користувача, то буде відправлятися запит з userId та formId (Рис 3.25). А на backend частині ми всього лиш добавляємо новий запис в нашу таблицю client_form_user.

```
addUser() {
    this.loader = true;
    let url = '/adminpanel/forms/add-user';
    let formData = new FormData();

    formData.append('userId', this.user);
    formData.append('formId', this.formData.id);

    axios
        .post(url, formData)
        .then(response => {
            this.message.outputMessage = response.data.success;
            this.message.type = 'success';
            this.message.active = true;
            this.user = '';
            this.getForm();
            setTimeout(() => {
                this.message = {active: false};
            }, 3000);
        })
},
```

Рисунок 3.25 – Метод відправки запиту для додавання нового користувача для форми

А користувачі, які вже отримали дану форму, ми добавимо в табличку знизу. Результат роботи (Рис. 3.26)

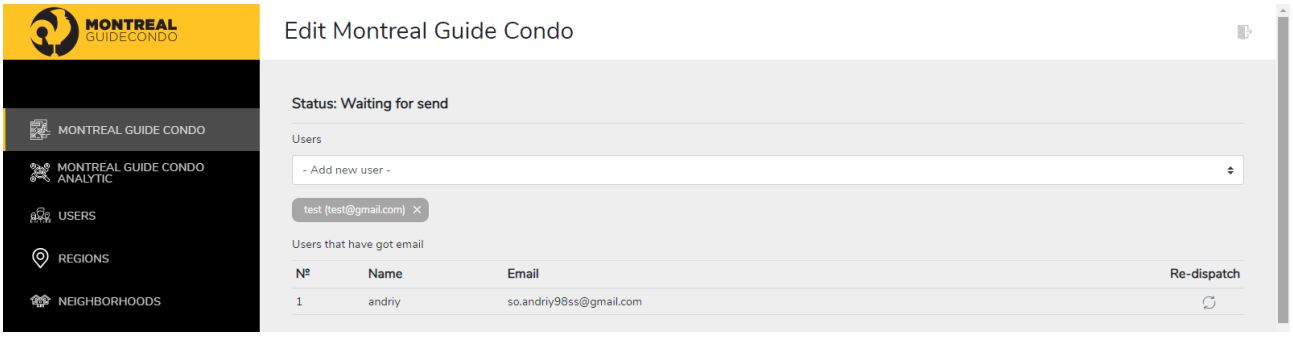


Рисунок 3.26 – Додавання та видалення користувача для форми

І останнім кроком це нам потрібно вивести всіх користувачів і форми, які вони отримали. Це ми зробимо у вкладці analytics, це буде та сама таблиця з іменем користувача, та датою відправки форми. Запит для отримання аналітики (Рис. 3.27)

```
$analyticsForms = $this
    ->select(
        'users.name',
        'client_form_user.client_form_id',
        'client_form_user.created_at'
    )
    ->join(
        'users',
        'client_form_user.user_id',
        'users.id'
    )
    ->when($name, function ($query) use ($name) {
        $query->where('users.name', 'LIKE', "%$name%");
    })
    ->when($date, function ($query) use ($date) {
        $query->whereBetween('client_form_user.created_at', [$date->start, $date->end]);
    })
    ->orderBy('client_form_user.id', 'desc')
    ->paginate(10);
```

Рисунок 3.27 – Запит отримання аналітики

А результат виводу в таблиці можна побачити на наступному рисунку (Рис. 3.28)

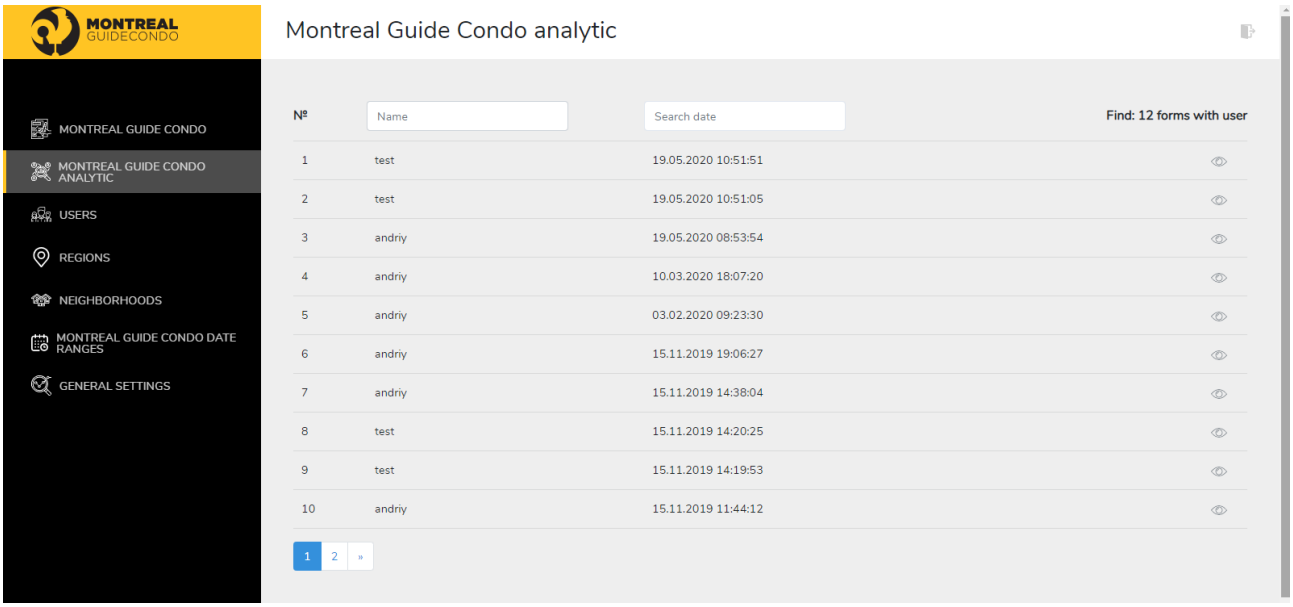


Рисунок 3.28 – Вивід аналітики

3.2 Розробка форми

Розробка форми є одним з найважливіших і основних етапів нашого проекту. Отже першим нашим кроком потрібно створити поля, селекти і підключити range picker, щоб можна було зручно вибирати мінімальну і максимальну ціну. Строгі вимоги до дизайну відсутні, тому все буде на наш розсуд. Результат створення форми буде мати вигляд (Рис 3.29)

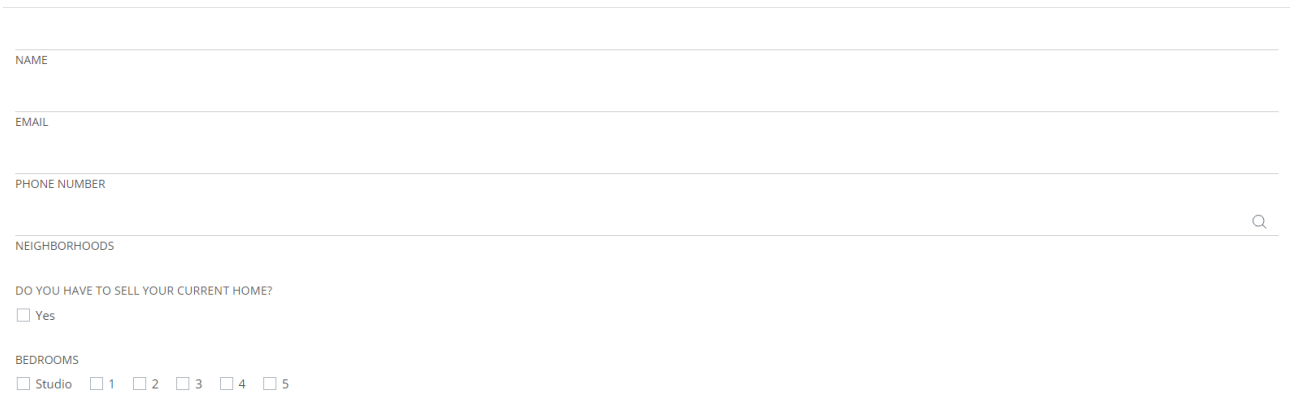


Рисунок 3.29 – Форма для заповнення, аркуш 1

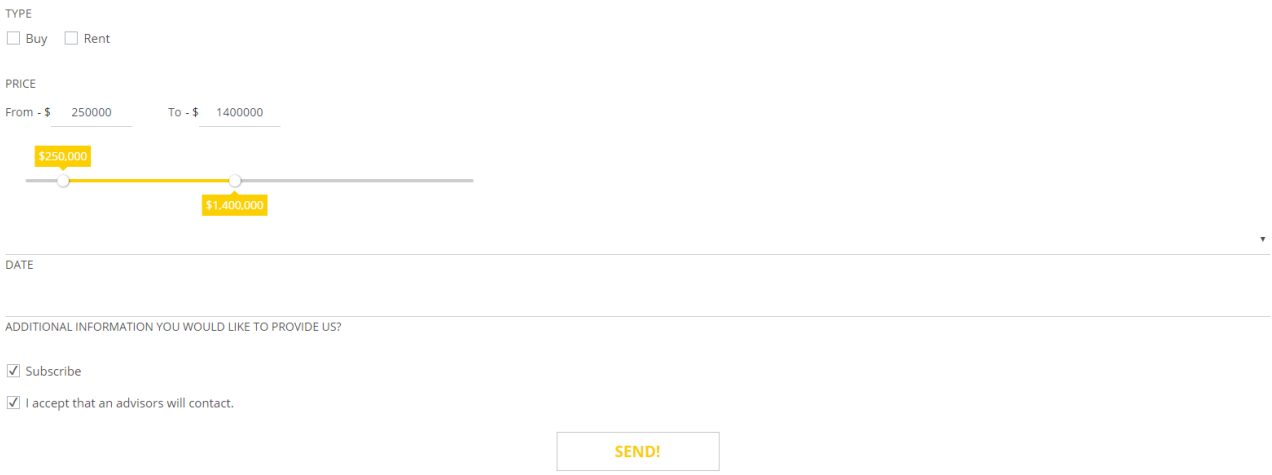
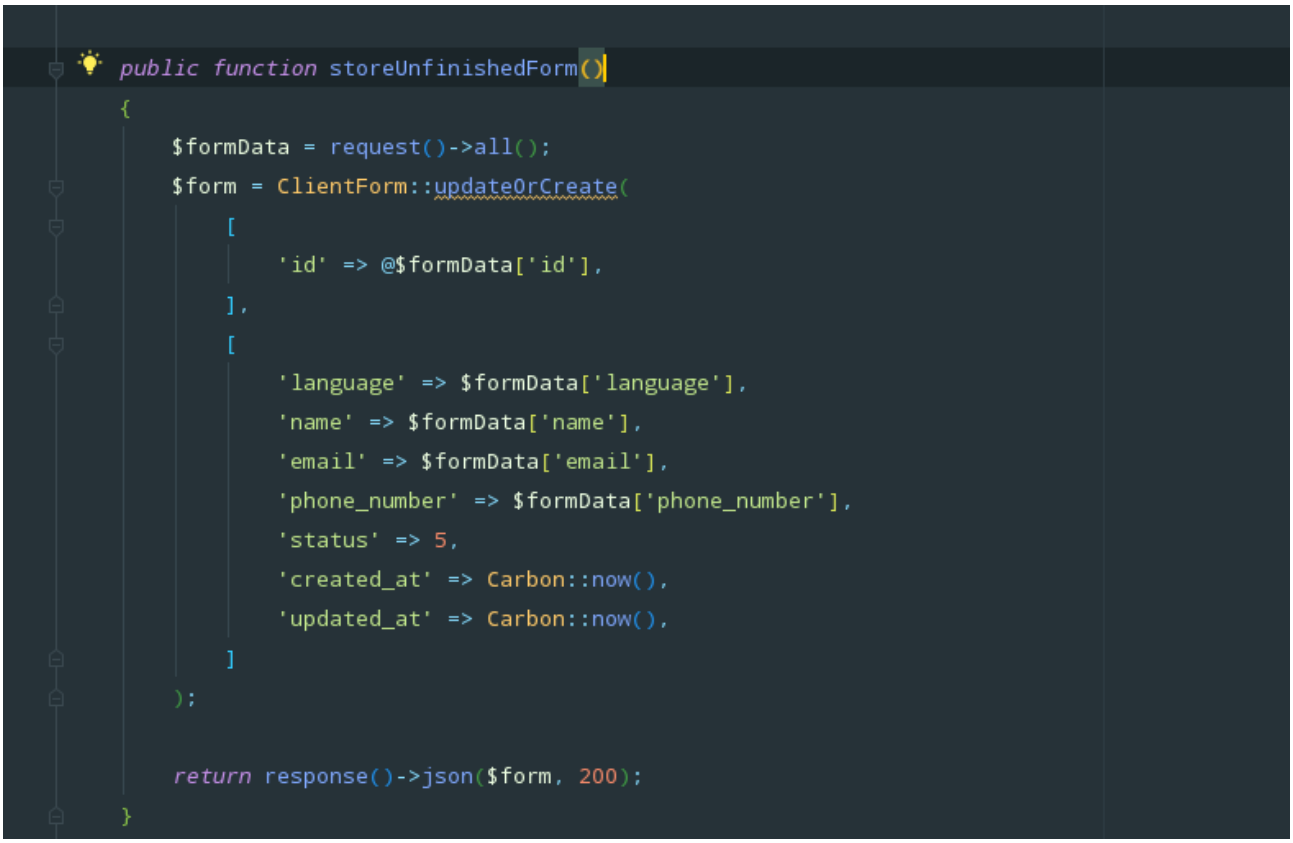


Рисунок 3.29 – Форма для заповнення, аркуш 2

Тепер, коли ми створили frontend частину нам потрібно відправляти форми, які будуть не до кінця заповненні. Перш за все нам потрібно на кожне поле створити watcher, який буде відслідковувати стан кожного поля в нашій формі і якщо користувач заповнить поля name, email та phone number, то тепер при кожній зміні любого поля ми будем відправляти request на url /client-panel/send-unfinished-form з даними які зараз є на формі (Рис. 3.30)

Рисунок 3.30 – Відправлення незавершених форм

На backend частині в методі буде створення або оновлення запису в базі даних (Рис 3.31)



```php
public function storeUnfinishedForm()
{
    $formData = request()->all();
    $form = ClientForm::updateOrCreate(
        [
            'id' => @$formData['id'],
        ],
        [
            'language' => $formData['language'],
            'name' => $formData['name'],
            'email' => $formData['email'],
            'phone_number' => $formData['phone_number'],
            'status' => 5,
            'created_at' => Carbon::now(),
            'updated_at' => Carbon::now(),
        ]
    );

    return response()->json($form, 200);
}
```

Рисунок 3.31 – Створення або оновлення незавершеної форми

Метод updateOrCreate 1 параметром приймає масив з ключами, які є поля в таблиці і їх значеннями, по яких буде перевірка чи існує вже такий запис в базі даних і якщо запис відсутній, то метод створить новий, а якщо вже існує, то він лише його обновить. Наступним нашим кроком буде написання методу перевірки email. Тому що є можливість відправляти одну форму протягом 5 днів. Метод буде мати вигляд на (Рис 3.32)

```php
public function checkEmail($email)
{
    if ($email) {
        $date = Carbon::now()->subDays(5);
        $user_email = ClientForm
            ::where('email', $email)
            ->where('status', '!=', 5)
            ->whereDate('created_at', '>', $date)
            ->first();
        if ($user_email) {
            return response()->json(0);
        }

        return response()->json(1);
    }

    return response()->json(1);
}
```

Рисунок 3.32 – Метод перевірки email

В даному методі ми беремо сьогоднішню дату і віднімаємо від неї 5 днів і після чого провіряємо чи є в базі за останні 5 днів форма з таким email, якщо є то ми повертаємо 0, а якщо ні то 1. Тепер при кожній зміні поля email буде надсилатися запит на унікальність email. Тепер все що нам залишилося - це відправити форму, зберегти її і підібрати продавця чи орендодавця.

Після натискання кнопки “send” ми будемо відправляти request на url /client-panel/send-form. На backend частині, так як в нас вже є створення форма, тому що вона створилася на етапі відправки незавершеної форми, ми обновляємо форму кінцевими даними (Рис 3.33)

```php
$form = ClientForm::updateOrCreate(
    [
        'id' => @$formData->id,
    ],
    [
        'language' => $formData->language,
        'name' => $formData->name,
        'email' => $formData->email,
        'rent' => isset($formData->rent) ? 1 : 0,
        'buy' => isset($formData->buy) ? 1 : 0,
        'sale' => isset($formData->sale) ? 1 : 0,
        'price_min' => $formData->price[0],
        'price_max' => $formData->price[1],
        'date_from' => $date_from,
        'date_to' => $date_to,
        'date_range_id' => $formData->date_range ? $formData->date_range : null,
        'additional_info' => @$formData->additional ? $formData->additional : null,
        'phone_number' => $formData->phone_number,
        'expert_refer' => $formData->refer ? 1 : 0,
        'subscribe' => $formData->subscribe ? 1 : 0,
        'status' => $formData->refer ? 1 : 4,
        'created_at' => Carbon::now(),
        'updated_at' => Carbon::now(),
    ]
```

Рисунок 3.33 – Оновлення форми кінцевими даними

Після чого викликаємо метод getUserAccordingToForm, в якому ми опишемо логіко підбору продавця чи орендодавця (Рис 3.34)

Рисунок 3.34 – Частина запиту з логікою підбора

Ми звертаємся до таблиці user і в нашому запиті додаємо умови підбору, а саме buy and rent, провіряємо чи підходить діапазон ціни у форм до якогось продавця чи орендодавця, перевіряємо кількість кімнат, відповідний район. У випадку, якщо було підібрано декілька продавців чи орендодавців, то ми сортуємо їх по кількості отриманих форм і user який отримав найменшу кількість форм, буде обраний для форми. А у випадку, якщо система сама не підбере продавця чи орендодавця адміністратори зможуть автоматично додати користувача у адмін частині.

3.3 Тестування

Тестування це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, а також пошук помилок під час роботи продукту і сумісність з вимогами середовища в якому він має використовуватись. Техніка тестування також включає як процес випробування програмних складових з метою оцінки [11].

Тестування є дуже важливою частиною розробки веб додатку тому що:

- тестування дозволить перевірити, чи правильно реалізовано усі вимоги до програмного забезпечення, що розроблялось;
- тестування допоможе виявити помилки та проблеми у веб додатку ще до етапу запуску і надасть можливість все виправити;
- тестування зменьшує наслідки та ризики втрат якщо програмний продукт все ж випустили з помилками або неправильними вимогами. Вимоги в такому випадку будуть частково змінені або переоцінені;
- тестування також демонструє, що створене програмне забезпечення відповідає всім вимогам продуктивності;
- тестування допомагає перевірити взаємодію програми з навколишнім середовищем і відсутність неочікуваної поведінки.

Зазвичай люди схильні до помилок, вони можуть призводити до порушення нормальної роботи програмного забезпечення як на стадії розробки продукту, так і на стадії продажі і наслідки цього можуть бути найрізноманітнішими від незначних до катастрофічних. Для програмного забезпечення будь-якої складності неможливо вилучити всі проблеми. Зазвичай дещо неможливо помітити людським оком, проте тестування допомагає виявити більшість помилок, з якими може зіткнутися користувач, й потенційно зменшує ризик виникнення проблем з програмою у майбутньому [13].

Види тестування програмного забезпечення, поділяються на групи які залежуть від переслідуваних цілей. Їх можна умовно розділити на наступні групи:

- функціональні (Functional testing);
- нефункціональні (Non-functional testing);
- пов’язані зі змінами (Regression testing).

Функціональні тести зусереджені на особливостях і функціях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component / Unit testing), інтеграційному (Integration testing), системному (System testing) і приймальному (Acceptance testing). Функціональні види тестування перевіряють зовнішню поведінку системи. Найпоширеніших видів функціональних тестів:

- функціональне тестування (Functional testing)
- тестування безпеки (Security and Access Control Testing)
- тестування взаємодії (Interoperability Testing)

Нефункціональне тестування необхідні щоб визначити характеристику програмного забезпечення, тобто це таке тестування, щоб визначити як в цілому працює система.

Після того як були вирішенні баги чи інші проблеми які були виявленні під час тестування, потрібно перетестувати систем ще один раз, для того щоб бути впевненим, що все було виправленно коректно [14].

Отже першим кроком створимо користувача з декількома районами, виберемо 1 і 2 кімнати, тип виберемо продаж, та встановимо ціновий діапазон від 80000$ до 250000$ (Рис 3.35)

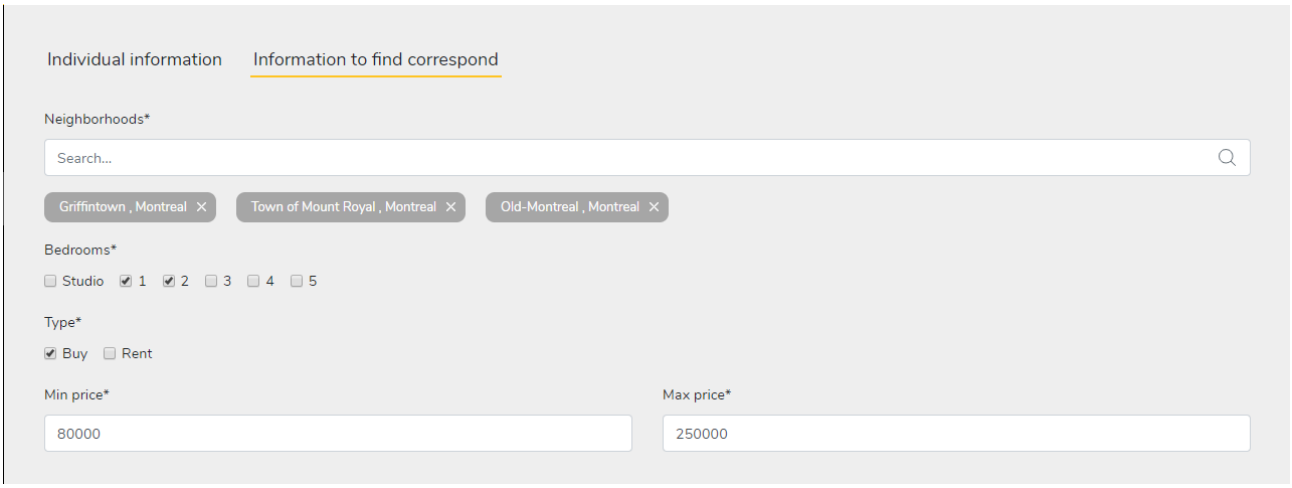


Рисунок 3.35 – Заповнення інформації

Подивимся чи коректо користувачі відображають в таблиці user (Рис 3.36)

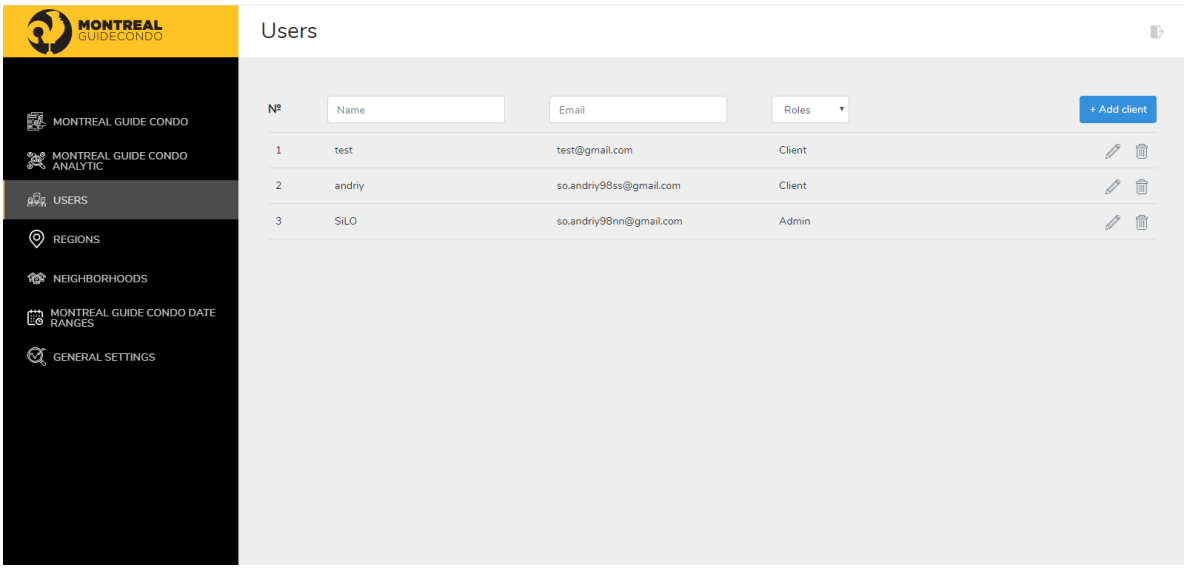


Рисунок 3.36 – Вивід створеного користувача

Тепер перейдемо до нашої форми і заповнимо перших 3 поля і перевіримо чи буде надіслана наша незавершена форма (Рис. 3.37)

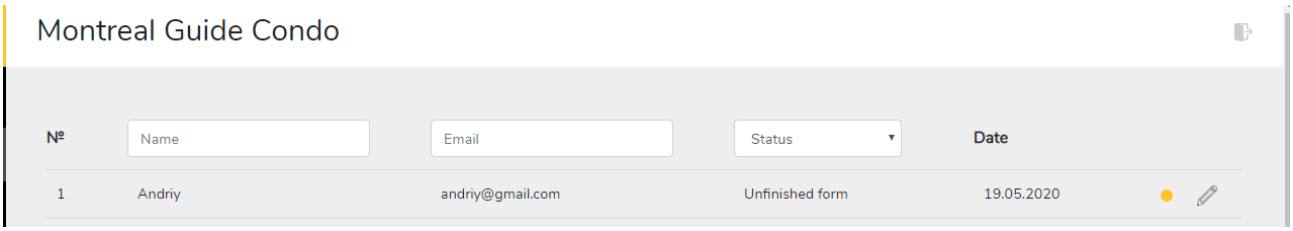


Рисунок 3.37 – Вивід форми

Працює все чудово, тепер заповнимо нашу форму так, що система підібрала нам того користувача, якого ми створили. Після відправлення форми покажеться сповіщення про успішність відправки (Рис. 3.38)

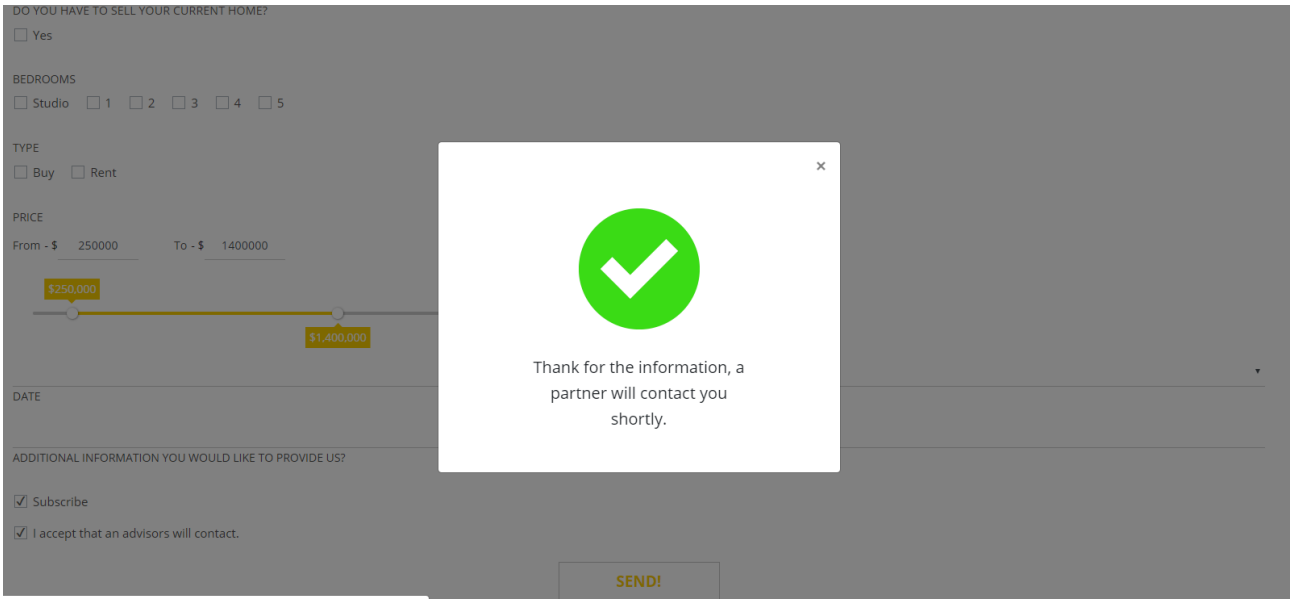


Рисунок 3.38 – Сповіщення успішно відправленої форми

Отже, форма успішно відправилася, і в адмін частині коректно відображається, а система підібрала нам користувача. Дивись результат роботи (Рис. 3.39)

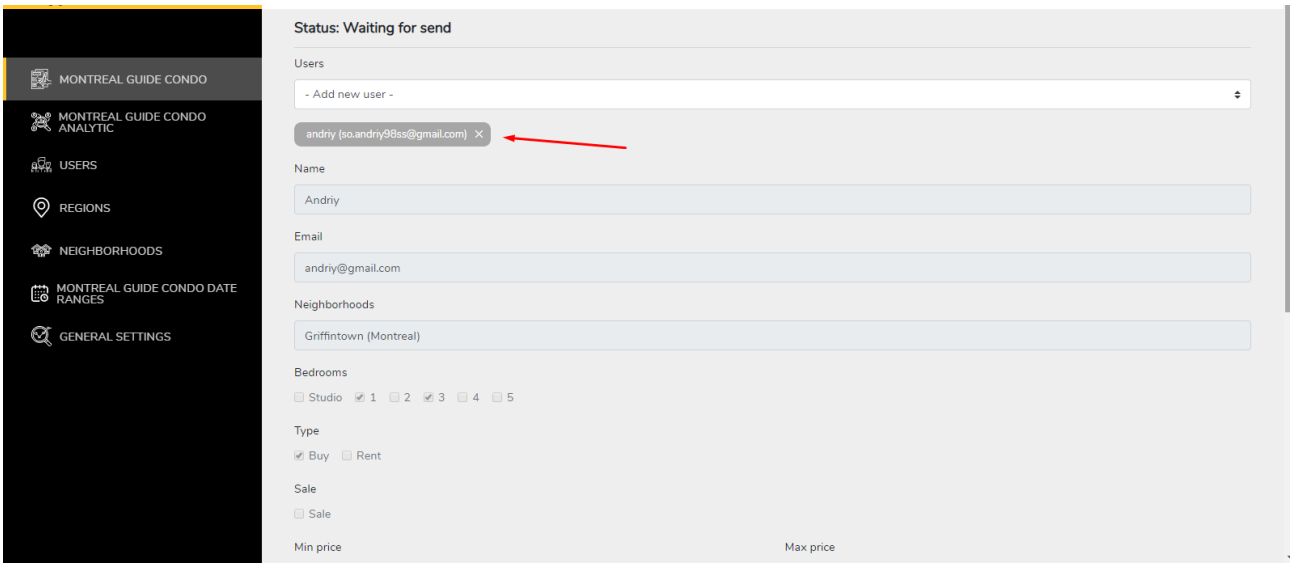


Рисунок 3.39 – Результат підбору користувача

4 БІЗНЕС ПЛАН

4.1 Огляд проекту

Веб додаток для інтернет магазинів з продажі нерухомості. Буде можливість додати продавців або орендодавців в систему за якусь відповідно визначену суму і форма, яка буде інтегрована на сайті, буде приносити додаткових покупців чи орендаторів для клієнтів. В таблиці побачимо всю загальну інформацію про проект:

Таблиця 4.1

№	Назва	Опис
1.	Найменування	“Form for real estate”
2.	Організаційна форма	Індивідуальний підприємець
3.	Види ліцензії на додаток	• 1 місяць • 3 місяці • 6 місяців • 1 рік
4.	Режим роботи	Цілодобовий
5.	Персонал підтримки проекту	Fronted – 1 чоловік Backed – 1 чоловік. Адміністратори – по бажанню залежно від к-сть форм, які відправляються протягом дня
6.	Необхідний стартовий капітал	20 000 грн.
7.	Місячні витрати	10 000 грн.
8.	Конкуренція	Відсутня
9.	Дохід від додатку.	Залежить від к-сть проданих ліцензій.

Так як конкуренція відсутня даний веб додаток може стати чудовим доповненням для інтернет магазинів і залучить додаткову аудиторію і збільшись кількість клієнтів, а для великих інтернет магазинів є можливість заробити на продавцях і орендодавцях і частково або цілком відробити ліцензію даного продукту.

4.2 Послуги

Ми надаємо можливість інтернет магазинам обрати одну з чотирьох видів ліценції:

- 1 місяць – ціна 150$, а також 9% від кількості проданих місць продавцям і орендодавцям (суму місця визначає інтернет магазин). У випадку якщо інтернет магазин компанії з продажі нерухомості, то ціна ліцензії складає 300$;
- 3 місяця – ціна 400$, а також 8% від кількості проданих місць продавцям і орендодавцям (суму місця визначає інтернет магазин). У випадку якщо інтернет магазин компанії з продажі нерухомості, то ціна ліцензії складає 800$;
- 6 місяців - ціна 750$, а також 7% від кількості проданих місць продавцям і орендодавцям (суму місця визначає інтернет магазин). У випадку якщо інтернет магазин компанії з продажі нерухомості, то ціна ліцензії складає 1500$;
- 1 рік - ціна 1350$, а також 6% від кількості проданих місць продавцям і орендодавцям (суму місця визначає інтернет магазин). У випадку якщо інтернет магазин компанії з продажі нерухомості, то ціна ліцензії складає 2800$.

Крім цього ми надаємо послуги для створення власного дизайну форми і адмін частини (ціна залежить від складності). Розробка додаткового функціоналу (ціна залежить від складності реалізації).

4.3 Стратегія маркетингу

Реклама – важливий засіб диференціації продукту, брендингу та формування позиції на ринку. Ви практично необмежені у засобах самовираження, що дозволяє створити креативну рекламу, яку аудиторія буде поширювати. Звичайно, це забезпечить увагу до вашого бренду/продукту, а крім того покращить його сприйняття серед споживачів, що безперечно відобразиться на фінансовому результаті [16].

Реклама є дуже важливою частиною, тому що який би якісний і коричний продукт ви не створили, якщо про нього ніхто не дізнається, то відповідно результату не буде, кошти в розробку були вкладені, а отримати заробіток з продукту не виходить.

Найбільш популярні digital-канали залучення покупців:

- платна реклама в пошукових системах — Search Engine Marketing (SEM або контекстна реклама);
- безкоштовний пошуковий трафік — Search Engine Optimization (SEO);
- прайс-агрегатори і маркетплейси;
- продажі за допомогою поштових розсилок - email-маркетинг (emailing);
- висвітлення інформації в інтернет-виданнях, PR і написання гостьових статей (контент-маркетинг);
- соціальні мережі - Social Media Marketing (SMM);
- різні види контент-маркетингу (реклама у блогерів, на Youtube, спецпроекти і т. д.).

Ми обрали стратегію рекламування нашого продукту в інтернет ресурсах, так як це є найбільш актуально і інтернет магазини зможуть помітити наш продукт.

До старту нашого продукту планується залучити покупців знижкою на наші ліцензії. Це дасть хорошу можливість отримати перших клієнтів і також хороша можливість інтернет магазинам зекономити на покупці.

4.4 Фінансовий план

У табл. 4.2 представлено результати прогнозованих оцінок доходів, витрат і фінансового результату до кінця 2020 р., тобто за першу чверть року функціонування веб додатку.

За прогнозом до кінця року роботи продукту загальний обсяг продажу становитиме 330000,00 грн. Якщо ми віднімемо собівартість даного продукту, то обсяг валового прибутку буде дорівнювати 306000,00 грн. Операційні витрати до кінця року складатимуть 50300,00 грн., зокрема:

- витрати на оренду приміщення – 16000,00 грн.;
- витрати на комунальні послуги – 6300,00 грн.;
- витрати на рекламу – 28000,00 грн.

Отже, найбільшу частку у витратах належатиме витратам на рекламу, так як це є найбільш важливим вкладом коштів.

З урахування доходів і витрат плановий обсяг чистого прибутку буде складати 255700,00 грн.

Таблиця 4.2 - План доходів, витрат і фінансового результату до кінця 2020 р., грн.

Показник	Період				
	Вересень	Жовтень	Листопад	Січень	Протягом 4 місяців
1. Загальний обсяг продажу продукту	50000,00	70000,00	90000,00	120000,00	330000,00
2. Собівартість продукту	6000,00	6000,00	6000,00	6000,00	24000,00
3. Валовий прибуток (рядок 1 – рядок 2)	44000,00	64000,00	84000,00	114000,00	306000,00
4. Операційні витрати – усього	13760,00	11440,00	13500,00	11600,00	50300,00
оренда приміщення	4000,00	4000,00	4000,00	4000,00	16000,00
комунальні послуги	1760,00	1440,00	1500,00	1600,00	6300,00
реклама	8000,00	6000,00	8000,00	6000,00	28000,00
5. Чистий прибуток (р.3 – р.4)	30240,00	52560,00	70500,00	102400,00	255700,00

ВИСНОВКИ

У результаті бакалаврської роботи було розроблено веб додаток для інтернет магазинів з продажі нерухомості. Даний веб додаток буде залучати більшу кількість клієнтів для продавців та орендодавців, а клієнтам буде зручніше підбирати нерухомість.

Під час роботи було описано предметну область, розглянуті сайти конкуренти, а також було створено специфікацію вимог.

Відштовхуючись від поставлених вимог, було обрано інструментарій розробки, створена архітектура проекту (frontend та backend частини) та спроектована структура бази даних.

Реалізація програмного забезпечення відбувалося в 2 етапи. Спочатку була розроблена частина адміністратора, а після цього була реалізована форма та підбір продавця чи орендодавця для клієнта.

Для тестування було обрано ручний метод тестування, було пройдено всі основні етапи в системі, критичних помилок не було виявлено.

Було розроблено бізнес план з обрахунками і з можливим прибутком до кінця 2020 року.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. Економічні, правові, інформаційні та гуманітарні проблеми розвитку України в пост стабілізаційний період. Матеріали наукової конференції професорсько-викладацького складу (Тернопіль, 16 квітня 2008 року). – Тернопіль: Тернопільський національний економічний університет, 2008. 168 с.
2. Основні елементи ринку нерухомості. URL: https://studme.com.ua/1003071214169/ekonomika/osnovnye_elementy_rynka_nedvizhimosti.htm (дата звернення: 21.04.2020).
3. Сайти нерухомості. URL: https://marketer.ua/ua/property-sites-top-20-review/ (дата звернення: 29.04.2020).
4. Архітектура програмного забезпечення. URL: https://uk.wikipedia.org/wiki/Архітектура_програмного_забезпечення (дата звернення: 29.04.2020).
5. Інтернет магазин. URL: https://www.montrealguidecondo.ca/en/ (дата звернення: 02.05.2020).
6. Роберт Мар. Чиста архітектура: мистецтво розробки програмного забезпечення: Фабула, 2019. 416 с.
7. Мэтт Зандстра. PHP: Объекты, шаблоны и методики программирования - 4-е издание: Диалектика, 2015. 576 с.
8. Робин Никсон. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 4-е издание: Питер, 2017. 768 с.
9. Kelt Dockins. Design Patterns in PHP and Laravel: Apress, 2016. 264 с.
10. Никольский А.П. JavaScript на примерах: Наука и техника, 2017. 274 с.

11. Тестування – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Тестування_програмного_забезпечення (дата звернення: 19.05.2020)
12. Стандарт кафедри Інформаційних Технологій ПНУ “Вимоги до змісту та оформлення”, згідно вимог
13. Роль тестування. URL: https://www.quality-assurance-group.com/shho-take-testuvannya-programnogo-zabezpechennya-ta-yake-jogo-znachennya/ (дата звернення: 19.05.2020)
14. Види тестування ПО. URL: http://qalearning.com.ua/theory/lectures/material/testing-types-functional/ (дата звернення: 20.05.2020)
15. База даних. URL: https://uk.wikipedia.org/wiki/База_даних (дата звернення: 20.05.2020)
16. Значення реклама. URL: https://blog.tracklam.com/5-prychyn-chomu-reklama-neobhidna-dlya-roz/ (дата звернення: 20.05.2020)

ДОДАТОК А

Код проекту

Model ClientForm:

```
class ClientForm extends Model
{
    protected $fillable = [
        'language',
        'name',
        'email',
        'buy',
        'rent',
        'sale',
        'price_min',
        'price_max',
        'date_from',
        'date_to',
        'additional_info',
        'phone_number',
        'expert_refer',
        'subscribe',
        'date_range_id',
        'status',
        'is_check'
    ];

    protected function getForms($search)
    {
        $name = $search->searchName;
        $email = $search->searchEmail;
        $status = $search->status;

        $forms = $this
            ::select(
                'client_forms.id',
                'client_forms.name',
                'client_forms.email',
                'client_forms.status',
                'client_forms.is_check',
                'client_forms.created_at'
            )
            ->when($name, function ($query) use ($name) {
                $query->where('name', 'LIKE', "%$name%");
            })
            ->when($email, function ($query) use ($email) {
                $query->where('email', 'LIKE', "%$email%");
            })
            ->when($status, function ($query) use ($status) {
                $query->where('status', $status);
            })
            ->orderBy('is_check')
            ->orderBy('created_at', 'desc')
            ->paginate(10);

        $pagination = $this
```

```
            ->all()
            ->count() > 10
            ? true
            : false;

        $data = ['forms' => $forms, 'pagination' => $pagination];

        return $data;
    }

    protected function getForm($id)
    {
        return $this
            ::select(
                'client_forms.*',
                'date_ranges.name_en',
                'date_ranges.name_fr'
            )
            ->leftJoin(
                'date_ranges',
                'date_ranges.id',
                'client_forms.date_range_id'
            )
            ->where('client_forms.id', $id)
            ->first();
    }
}
```

Model User:

```
class User extends Authenticatable implements JWTSubject
{
    use Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'company_name',
        'email',
        'page_url',
        'description',
        'password',
        'encrypt_password',
        'role',
        'buy',
        'rent',
        'type_of_residence_id',
        'price_min',
        'price_max',
        'counter',
        'max_forms',
        'offline'
    ];

    /**
     * The attributes that should be hidden for arrays.
```

```
 *
 * @var array
 */
protected $hidden = [
    'password', 'remember_token',
];

/**
 * The attributes that should be cast to native types.
 *
 * @var array
 */
protected $casts = [
];

public function getJWTIdentifier()
{
    return $this->getKey();
}

public function getJWTCustomClaims()
{
    return [];
}

protected function getClients($search)
{
    $name = $search->searchName;
    $email = $search->searchEmail;
    $role = $search->role;

    $users = $this
        ::select(
            'users.*'
        )
        ->when($name, function ($query) use ($name) {
            $query->where('name', 'LIKE', "%$name%");
        })
        ->when($email, function ($query) use ($email) {
            $query->where('email', 'LIKE', "%$email%");
        })
        ->when($role, function ($query) use ($role) {
            $query->where('role', $role);
        })
        ->orderBy('id', 'desc')
        ->where('name', '!=' , 's-admin')
        ->paginate(10);

    $pagination = $this
        ->all()
        ->count() > 10
        ? true
        : false;

    $data = ['users' => $users, 'pagination' => $pagination];

    return $data;
}
```

```
    protected function getUserAccordingToForm($data){
        $formData = $data->formData;
        $neighborhoodsId = $data->neighborhoodsId;
        $buy = isset($formData->buy) ? 1 : 0;
        $rent = isset($formData->rent) ? 1 : 0;
        foreach ($neighborhoodsId as $neighborhoodId){
            $neighborhood = Neighborhood::where('id', $neighborhoodId)->get();
            if(isset($neighborhood[0]->region_id) == null){
                $regionId = Region
                    ::select('id')
                    ->where('name_en', $neighborhood[0]->name_en)
                    ->get();
                $newNeighborhoodsId = Neighborhood
                    ::select('id')
                    ->where('region_id', $regionId[0]->id)
                    ->get();
                foreach ($newNeighborhoodsId as $newNeighborhoodId){
                    array_push($neighborhoodsId, $newNeighborhoodId->id);
                }
            }else{
                $region = Region::where('id', $neighborhood[0]->region_id)->get();
                $neighborhId = Neighborhood::where('name_en', $region[0]->name_en)-
>get();
                array_push($neighborhoodsId, $neighborhId[0]->id);
            }
        }

        $user = $this
            ->select(
                'users.*',
                DB::raw("
                (
                    SELECT created_at
                    FROM client_form_user
                    ORDER BY created_at DESC
                    LIMIT 1
                ) as last_date
                ")
            )
            ->leftJoin(
                'users_bedroom',
                'users_bedroom.user_id',
                'users.id'
            )
            ->leftJoin(
                'users_neighborhood',
                'users_neighborhood.user_id',
                'users.id'
            )
            ->when($buy, function ($query) use ($buy) {
                $query->where('buy', $buy);
            })
            ->when($rent, function ($query) use ($rent) {
                $query->where('rent', $rent);
            })
            ->where('users.role', 1)
            ->where('users.price_min', '<=' , $formData->price[0])
            ->where('users.price_max', '>=', $formData->price[1])
            ->where('users.offline', '!=', 1)
```

```
            ->where(function ($query) {
                $query->where(
                    'users.max_forms',
                    '>',
                    DB::raw("
                    (
                        SELECT COUNT(*)
                        FROM client_form_user
                        WHERE client_form_user.user_id = users.id and
                        client_form_user.created_at >= DATE_SUB(NOW(), INTERVAL 1
MONTH)
                        GROUP BY user_id
                        LIMIT 1
                    )
                ")
                )
                ->orWhere('users.max_forms', '=', 0);
            })
            ->whereIn('users_bedroom.bedroom', $formData->bedroom)
            ->whereIn('users_neighborhood.neighborhood_id', $neighborhoodsId)
            ->orderBy('last_date', 'asc')
            ->first();

        return $user;
    }

    protected function getUserAccordingToResidenceForm($data){
        $formData = $data->formData;
        $neighborhoodsId = $data->neighborhoodsId;
        foreach ($neighborhoodsId as $neighborhoodId){
            $neighborhood = Neighborhood::where('id', $neighborhoodId)->get();
            if(isset($neighborhood[0]->region_id) == null){
                $regionId = Region
                    ::select('id')
                    ->where('name_en', $neighborhood[0]->name_en)
                    ->get();
                $newNeighborhoodsId = Neighborhood
                    ::select('id')
                    ->where('region_id', $regionId[0]->id)
                    ->get();
                foreach ($newNeighborhoodsId as $newNeighborhoodId){
                    array_push($neighborhoodsId, $newNeighborhoodId->id);
                }
            }else{
                $region = Region::where('id', $neighborhood[0]->region_id)->get();
                $neighborhId = Neighborhood::where('name_en', $region[0]->name_en)-
>get();
                array_push($neighborhoodsId, $neighborhId[0]->id);
            }
        }

        $user = $this
            ->select(
                'users.*',
                DB::raw("
                (
                    SELECT created_at
                    FROM residence_client_forms_user
                    ORDER BY created_at DESC
```

```
                    LIMIT 1
                ) as last_date
                ")
            )
            ->leftJoin(
                'users_neighborhood',
                'users_neighborhood.user_id',
                'users.id'
            )
            ->leftJoin(
                'users_type_of_residences',
                'users_type_of_residences.user_id',
                'users.id'
            )
            ->where('users.role', 3)
            ->where('users_type_of_residences.type_of_residence_id', $formData-
>residence)
            ->where('users.price_min', '<=' , $formData->price[0])
            ->where('users.price_max', '>=', $formData->price[1])
            ->where(function ($query) {
                $query->where(
                    'users.max_forms',
                    '>',
                    DB::raw("
                    (
                        SELECT COUNT(*)
                        FROM residence_client_forms_user
                        WHERE residence_client_forms_user.user_id = users.id and
                        residence_client_forms_user.created_at >= DATE_SUB(NOW(),
INTERVAL 1 MONTH)
                        GROUP BY user_id
                        LIMIT 1
                    )
                ")
                )
                ->orWhere('users.max_forms', '=', 0);
            })
            ->whereIn('users_neighborhood.neighborhood_id', $neighborhoodsId)
            ->orderBy('last_date', 'asc')
            ->first();

        return $user;
    }
}
```

Model Region:

```
class Region extends Model
{
    protected $fillable = [
        'name_en', 'name_fr'
    ];

    public $timestamps = false;

    protected function getRegions($search){

        $name = $search->name;

        $users = $this
```

```
            ::select(
                'regions.*'
            )
            ->when($name, function ($query) use ($name) {
                $query->where('name_en', 'LIKE', "%$name%");
            })
            ->orderBy('id', 'desc')
            ->paginate(10);

        $pagination = $this->all()
            ->count() > 10
            ? true
            : false;

        $data = ['regions' => $users, 'pagination' => $pagination];

        return $data;
    }
}
```

Model Neighborhood:

```
class Neighborhood extends Model
{
    protected $fillable = [
        'name_en', 'name_fr', 'region_id', 'counter'
    ];

    public $timestamps = false;

    protected function getNeighborhoods($search){
        $name = $search->searchName;
        $regionSearchId = $search->regionSearchId;

        $users = $this
            ::select(
                'neighborhoods.*',
                'regions.name_en as region_name'
            )
            ->leftJoin(
                'regions',
                'neighborhoods.region_id',
                'regions.id'
            )
            ->when($name, function ($query) use ($name) {
                $query->where('neighborhoods.name_en', 'LIKE', "%$name%");
            })
            ->when($regionSearchId, function ($query) use ($regionSearchId) {
                $query->where('region_id', "$regionSearchId");
            })
            ->where('region_id', '!=', 'null')
            ->orderBy('id', 'desc')
            ->paginate(10);

        $pagination = $this
            ->all()
            ->count() > 10
            ? true
            : false;

        $data = ['neighborhoods' => $users, 'pagination' => $pagination];
```

```
        return $data;
    }
    protected function autocomplete($language, $st, $tags){
        return $this
            ::selectRaw("
            neighborhoods.id as neighborhood_id,
            neighborhoods.name_$language as neighborhood_name,
            regions.id as region_id,
            regions.name_$language as region_name,
            CONCAT(neighborhoods.name_$language, \", \", regions.name_$language) as
full_name
            ")
            ->leftJoin(
                'regions',
                'neighborhoods.region_id',
                'regions.id'
            )
            ->when($st, function ($query) use ($st, $language) {
                $query->where("neighborhoods.name_$language", 'LIKE', "%$st%");
            })
            ->when($tags, function ($query) use ($tags, $language) {
                $query->whereNotIn("neighborhoods.name_$language", $tags);
            })
            ->orderBy('region_id')
            ->get();
    }

    protected function getPopularNeighborhoods($language){
        return $this
            ::selectRaw("
            neighborhoods.id as neighborhood_id,
            neighborhoods.name_$language as neighborhood_name,
            regions.id as region_id,
            regions.name_$language as region_name,
            CONCAT(neighborhoods.name_$language, \", \", regions.name_$language) as
full_name
            ")
            ->leftJoin(
                'regions',
                'neighborhoods.region_id',
                'regions.id'
            )
            ->orderBy('counter', 'desc')
            ->paginate(5);
    }
}
```

Model DateRange:

```
class DateRange extends Model {
   public $timestamps = FALSE;
   protected $guarded = [];

   protected function getDateRange($search)
   {
      $name = $search->name;

      $date_ranges = $this
         ::select(
```

```
            'date_ranges.*'
        )
        ->when(
            $name,
            function ($query) use ($name) {
                $query->where('name_en', 'LIKE', "%$name%");
            }
        )
        ->orderBy('id', 'desc')
        ->paginate(10);

    $pagination = $this
        ->all()
        ->count() > 10 ? TRUE : FALSE;

    $data = ['date_ranges' => $date_ranges, 'pagination' => $pagination];

    return $data;
  }
}
```

Model DefaultSetting

```
class DefaultSetting extends Model
{
    protected $fillable = [
        'key', 'value'
    ];

    public $timestamps = false;

    protected function priceRange(){
        return $this
            ->get()
            ->pluck('value', 'key');

    }
}
```

Controller AdminClientController:

```
class AdminClientController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function getUsers()
    {
        $search = (object)[
            'searchName' => request('searchName'),
            'searchEmail' => request('searchEmail'),
            'role' => request('role')
        ];

        $users = User::getClients($search);

        return response()->json($users);
    }
```

```
    public function getResidences(){
        $residenceKey = explode(',', request('residenceKeys'));
        return response()->json(TypeOfResidence
            ::select(
                'type_of_residences.*'
            )
            ->when($residenceKey, function ($query) use ($residenceKey) {
                $query->whereNotIn("id", $residenceKey);
            })
            ->get());
    }

    public function store(Request $request)
    {
        $userData = json_decode($request->userData);
        if(User::where('email', $userData->email)->first()){
            return response()->json(['error' => 'User with this email already exist'],
422);
        }
        if($userData->role != 2) {
            $neighborhoods = json_decode($request->tagsPlace);
            $typeOfResidences = json_decode($request->tagsResidence);
            $data = [
                'name' => $userData->name,
                'company_name' => $userData->company_name,
                'email' => $userData->email,
                'page_url' => $userData->page_url,
                'description' => $userData->description,
                'role' => $userData->role,
                'rent' => isset($userData->rent) ? 1 : 0,
                'buy' => isset($userData->buy) ? 1 : 0,
                'price_min' => $userData->price_min,
                'price_max' => $userData->price_max,
                'max_forms' => $userData->max_forms,
                'offline' => (@$userData->offline ? 1 : 0),
                'created_at' => Carbon::now(),
                'updated_at' => Carbon::now()
            ];
            $message = 'Client was successfully created';
        }else{
            $data = [
                'name' => $userData->name,
                'email' => $userData->email,
                'role' => $userData->role,
                'password' => Hash::make($userData->password),
                'encrypt_password' => base64_encode($userData->password),
                'created_at' => Carbon::now(),
                'updated_at' => Carbon::now()
            ];
            $message = 'Admin was successfully created';
        }

        $user_id = User::insertGetId($data);

        if(isset($neighborhoods)) {
            foreach ($neighborhoods as $neighborhood) {
                UserNeighborhood::create([
                    'user_id' => $user_id,
```

```
                    'neighborhood_id' => $neighborhood->neighborhood_id
                ]);
            }
        }

        if(isset($typeOfResidences)) {
            foreach ($typeOfResidences as $typeOfResidence) {
                UserTypeOfResidence::create([
                    'user_id' => $user_id,
                    'type_of_residence_id' => $typeOfResidence->id
                ]);
            }
        }

        for ($i = 0; $i < count($userData->bedroom); $i++) {
            UserBedroom::create([
                'user_id' => $user_id,
                'bedroom' => $userData->bedroom[$i]
            ]);
        }

        return response()->json(['success' => $message]);
    }

    public function edit($id)
    {
        $user = User::find($id);
        $neighborhoods = UserNeighborhood::getNeighborhoodForUser($id);
        $typeOfResidences = UserTypeOfResidence::getTypeOfResidenceForUser($id);
        $bedrooms = UserBedroom::where('user_id', $id)->get();
        $decrypt_pass = base64_decode($user->encrypt_password);

        return response()->json([
            'user' => $user,
            'neighborhoods' => $neighborhoods,
            'decryptPass' => $decrypt_pass,
            'typeOfResidences' => $typeOfResidences,
            'bedrooms' => $bedrooms
        ]);
    }

    public function update(Request $request)
    {
        $userData = json_decode($request->userData);
        $neighborhoodData = json_decode($request->neighborhoodData);
        $typeOfResidenceData = json_decode($request->typeOfResidenceData);
        $bedroomData = json_decode($request->bedroomData);
        if(User::where('email', $userData->email)->where('id', '!=', $userData->id)-
>first()){
            return response()->json(['error' => 'User with this email already exist'],
422);
        }
        if($userData->role != 2){
            $data = [
                'name' => $userData->name,
                'company_name' => $userData->company_name,
                'email' => $userData->email,
                'page_url' => $userData->page_url,
                'description' => $userData->description,
```

```
        'role' => $userData->role,
        'rent' => $userData->rent ? 1 : 0,
        'buy' => $userData->buy ? 1 : 0,
        'price_min' => $userData->price_min,
        'price_max' => $userData->price_max,
        'max_forms' => $userData->max_forms,
        'offline' => (@$userData->offline ? 1 : 0),
        'updated_at' => Carbon::now()
    ];
    $message = 'Client was successfully updated';
}else{
    $data = [
        'name' => $userData->name,
        'email' => $userData->email,
        'password' => Hash::make($userData->password),
        'encrypt_password' => base64_encode($userData->password),
        'role' => $userData->role,
        'updated_at' => Carbon::now()
    ];
    $message = 'Admin was successfully updated';
}
User
    ::find($userData->id)
    ->update($data);

foreach ($typeOfResidenceData as $typeOfResidence) {
    if($typeOfResidence->name_en == '') {
        UserTypeOfResidence::find($typeOfResidence->id)->delete();
    }else if(!isset($typeOfResidence->type_of_residence_id)) {
        UserTypeOfResidence::create([
            'user_id' => $userData->id,
            'type_of_residence_id' => $typeOfResidence->id
        ]);
    }
}

foreach ($neighborhoodData as $neighborhood) {
    if($neighborhood->neighborhood_name == '') {
        UserNeighborhood::find($neighborhood->id)->delete();
    }else if($neighborhood->id === 0) {
        UserNeighborhood::create([
            'user_id' => $userData->id,
            'neighborhood_id' => $neighborhood->neighborhood_id
        ]);
    }
}

UserBedroom::where('user_id', $userData->id)->delete();
if(count($bedroomData)) {
    for ($i = 0; $i < count($bedroomData); $i++) {
        UserBedroom::create([
            'user_id' => $userData->id,
            'bedroom' => $bedroomData[$i]
        ]);
    }
}


return response()->json(['success' => $message]);
```

```
    }

    public function delete($id)
    {
        User::find($id)->delete();

        return response(200);
    }

    public function getNeighborhoods(Request $request)
    {
        $tags = json_decode($request->tags);
        $tagsArr = [];

        foreach ($tags as $tag){
            array_push($tagsArr, $tag->neighborhood_name);
        }

        $neighborhoods = Neighborhood
            ::autocomplete(
                $request->language,
                $request->search,
                $tagsArr
            );

        return response()->json($neighborhoods);
    }
}
```

Controller AdminFormController:

```
class AdminFormController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function getForms()
    {
        $search = (object)[
            'searchName' => request('searchName'),
            'searchEmail' => request('searchEmail'),
            'status' => request('status')
        ];

        $forms = ClientForm::getForms($search);

        return response()->json($forms);
    }

    public function edit($id)
    {
        ClientForm::find($id)->update(['is_check' => 2]);

        $formData = ClientForm::getForm($id);

        if($formData->status == 5)
            return response()->json([
                'formData' => $formData
```

```
        ]);

    $neighborhoods = ClientFormNeighborhood::getNeighborhoods($formData->id);
    $bedrooms = explode(', ', ClientFormBedroom::getBedrooms($formData->id));
    $pickedUpUsers = ClientFormUser::getUsersForEditForm($formData->id);
    $pickedUpUsersId = [];

    if($formData->date_from && $formData->date_to) {
        $formData->date_from = explode(' ', $formData->date_from)[0];
        $formData->date_to = explode(' ', $formData->date_to)[0];
    }

    foreach ($pickedUpUsers as $pickedUpUser){
        array_push($pickedUpUsersId, $pickedUpUser->user_id);
    }

    $allUsers = User
        ::where('role', 1)
        ->whereNotIn('id', $pickedUpUsersId)
        ->get();

    return response()->json([
        'formData' => $formData,
        'neighborhoods' => $neighborhoods,
        'bedrooms' => $bedrooms,
        'pickedUpUsers' => $pickedUpUsers,
        'allUsers' => $allUsers
    ]);
}

public function update(Request $request)
{
    $newUsers = json_decode($request->newUsers);
    $formId = $request->formId;

    foreach ($newUsers as $user){
        ClientFormUser::create([
            'client_form_id' => $formId,
            'user_id' => $user->id,
        ]);
    }

    return response()->json(['success' => 'Form was successfully updated']);
}

public function addUser(Request $request){
    ClientFormUser::create([
        'client_form_id' => $request->formId,
        'user_id' => $request->userId,
    ]);

    ClientForm::find($request->formId)->update(['status' => 1]);
    return response()->json(['success' => 'User was successfully added']);
}

public function changeStatus($id){
    $clientFormUser = ClientFormUser::find($id);
    $clientFormUser->update(['status' => 1]);
```

```
        ClientForm::find($clientFormUser->client_form_id)->update(['status' => 1]);

        return response()->json(['success' => 'User was successfully sent to re-
dispatch']);
    }

    public function deleteUser($id){
        $clientFormUser = ClientFormUser::find($id);
        $clientFormUser->delete();
        $clientFormUsers = ClientFormUser
            ::where('client_form_id', $clientFormUser->client_form_id)
            ->where('status', 1)
            ->get();
        $clientFormUsersSent= ClientFormUser
            ::where('client_form_id', $clientFormUser->client_form_id)
            ->where('status', 2)
            ->get();
        if(count($clientFormUsers) == 0){
            ClientForm::find($clientFormUser->client_form_id)->update(['status' => 3]);
            if(count($clientFormUsersSent) !== 0){
                ClientForm::find($clientFormUser->client_form_id)->update(['status' =>
2]);
            }
        }

        return response()->json(['success' => 'User was successfully deleted']);
    }

    public function delete($id)
    {
        ClientForm::find($id)->delete();

        return response(200);
    }

}
```

Controller AdminAnalyticFormController:

```
class AdminAnalyticFormController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function getAnalyticsForms()
    {
        $search = (object)[
            'name' => request('searchName'),
            'date' => json_decode(request('date')),
        ];

        $analyticsForms = ClientFormUser::getAnalyticsForms($search);

        return response()->json($analyticsForms);
    }
}
```

Controller NeighborhoodController:

```
class NeighborhoodController extends Controller
{
    public function getNeighborhoods()
    {
        $search = (object)[
            'searchName' => request('searchName'),
            'regionSearchId' => request('regionSearchId'),
        ];

        $neighborhoods = Neighborhood::getNeighborhoods($search);

        $regions = Region::all();

        return response()->json([
            'neighborhoods' => $neighborhoods,
            'regions' => $regions,
        ]);
    }

    public function getRegions(){
        $regions = Region::all();

        return response()->json($regions);
    }

    public function store(Request $request)
    {
        $neighborhood = json_decode($request->neighborhoodData);

        Neighborhood::create([
            'name_en' => $neighborhood->name_en,
            'name_fr' => $neighborhood->name_fr,
            'region_id' => $neighborhood->region_id
        ]);

        return response()->json([
            'success' => 'Neighborhood was successfully create',
        ]);
    }

    public function edit($id)
    {
        $neighborhood = Neighborhood
            ::select(
                'neighborhoods.*',
                'regions.name_en as region_name_en',
                'regions.name_fr as region_name_fr'
            )
            ->leftJoin(
                'regions',
                'neighborhoods.region_id',
                'regions.id'
            )
            ->where('neighborhoods.id', $id)
            ->first();

        $regions = Region::all();
```

```
        return response()->json([
            'neighborhood' => $neighborhood,
            'regions' => $regions
        ]);
    }

    public function update(Request $request)
    {
        $neighborhood = json_decode($request->neighborhoodData);

        Neighborhood
            ::find($neighborhood->id)
            ->update([
                'name_en' => $neighborhood->name_en,
                'name_fr' => $neighborhood->name_fr,
                'region_id' => $neighborhood->region_id
            ]);

        return response()->json([
            'success' => 'Neighborhood was successfully update'
        ]);
    }

    public function delete($id)
    {
        Neighborhood::find($id)->delete();

        return response(200);
    }
}
```

Controller RegionController:

```
class RegionController extends Controller
{
    public function getRegions()
    {
        $search = (object)[
            'name' => request('searchName'),
        ];

        $regions = Region::getRegions($search);

        return response()->json($regions);
    }

    public function store(Request $request)
    {
        $region = json_decode($request->regionData);

        Region::create([
            'name_en' => $region->name_en,
            'name_fr' => $region->name_fr
        ]);

        Neighborhood::create([
            'name_en' => $region->name_en,
            'name_fr' => $region->name_fr,
            'region_id' => null
        ]);
```

```
        return response()->json([
            'success' => 'Region was successfully create',
        ]);
    }

    public function edit($id)
    {
        $region = Region::find($id);

        return response()->json($region);
    }

    public function update(Request $request)
    {
        $region = json_decode($request->regionData);

        Region::find($region->id)
            ->update([
                'name_en' => $region->name_en,
                'name_fr' => $region->name_fr
            ]);

        return response()->json([
            'success' => 'Region was successfully update'
        ]);
    }

    public function delete($id)
    {
        Region::find($id)->delete();

        return response(200);
    }
}
```

Controller DateRangeController:

```
class DateRangeController extends Controller
{
   public function getDateRanges()
   {
      $search = (object)[
         'name' => request('searchName'),
      ];

      $dateRanges = DateRange::getDateRange($search);

      return response()->json($dateRanges);
   }

   public function store(Request $request)
   {
      $dateRange = json_decode($request->dateRange);

      DateRange::create([
         'name_en' => $dateRange->name_en,
         'name_fr' => $dateRange->name_fr
      ]);
```

```
        return response()->json([
            'success' => 'Date range was successfully create',
        ]);
    }

    public function edit($id)
    {
        $dateRange = DateRange::find($id);

        return response()->json($dateRange);
    }

    public function update(Request $request)
    {
        $dateRange = json_decode($request->dateRange);

        DateRange::find($dateRange->id)
              ->update([
                    'name_en' => $dateRange->name_en,
                    'name_fr' => $dateRange->name_fr
              ]);

        return response()->json([
            'success' => 'Date range was successfully update'
        ]);
    }

    public function delete($id)
    {
        DateRange::find($id)->delete();

        return response(200);
    }
}
```

Contrller GeneralSettingController:

```
class GeneralSettingController extends Controller
{
    public function getPriceRange(){
        $priceRange = DefaultSetting::priceRange();

        return response()->json($priceRange);
    }

    public function update(Request $request){
        $priceRange = json_decode($request->priceRange, true);

        foreach ($priceRange as $key => $val){
            DefaultSetting
                ::where('key', $key)
                ->update(['value' => $val]);
        }

        return response()->json(['success' => 'Settings was successfully updated']);
    }
}
```

Component Clients:

```
<template>
    <section>
        <header-layout></header-layout>
        <sidebar-layout></sidebar-layout>
        <div class="wrap">
            <message
                v-if="message.active"
                :type="message.type"
                :message="message.outputMessage">
            </message>
            <div class="ready">
                <table>
                    <thead>
                    <tr>
                        <th style="width: 80px;">№</th>
                        <th width="300px">
                            <div class="form-group form-group_width_medium">
                                <input
                                    type="text"
                                    class="form-control"
                                    placeholder="Name"
                                    v-model="searchName">
                            </div>
                        </th>
                        <th width="300px">
                            <div class="form-group form-group_width_medium">
                                <input
                                    type="text"
                                    class="form-control"
                                    placeholder="Email"
                                    v-model="searchEmail">
                            </div>
                        </th>
                        <th width="150px">
                            <div class="form-group form-group_width_small">
                                <select
                                    class="form-control"
                                    id="role"
                                    v-model="role"
                                    required>
                                    <option value="">Roles</option>
                                    <option value="1">Clients</option>
                                    <option value="2">Admins</option>
                                    <option value="3">Residence clients</option>
                                </select>
                            </div>
                        </th>
                        <th>
                            <div class="form-group form-group_width_small btn_add">
                                <router-link
                                    :to="{name: 'clients.create'}"
                                    tag="button"
                                    class="btn btn-primary">
                                    + Add client
                                </router-link>
                            </div>
                        </th>
                    </tr>
                    </thead>
```

```
                    <tr style="border-bottom: none" v-if="loader">
                        <td colspan="6"><loader></loader></td>
                    </tr>
                    <tbody v-else>
                    <tr
                        v-for="(user, key) in users.data"
                        :key="user.id">
                        <td>{{ key + 1 }}</td>
                        <td>{{ user.name }}</td>
                        <td>{{ user.email }}</td>
                        <td>
                            {{ setRole(user.role) }}
                        </td>
                        <td class="center-align">
                            <router-link
                                :to="{ name: 'clients.edit',
                                 params: {id: user.id}}"
                                tag="span"
                                class="edit">
                                <span v-icons-svg="'edit'"></span>
                            </router-link>
                            <span
                                class="delete"
                                @click="deleteUser(user.id)">
                                <span v-icons-svg="'delete'"></span>
                            </span>
                        </td>
                    </tr>
                    </tbody>
                </table>
                <div v-if="!users.total && loader === false" class="text-center mt-5">
                    <h4>User list is empty</h4>
                </div>
            </div>
                <pagination
                    :data="users"
                    v-if="pagination"
                    :limit="3"
                    class="mt-3"
                    @pagination-change-page="getUsers">
                </pagination>
        </div>
    </section>
</template>

<script>
    import Header from '../layouts/Header.vue';
    import Sidebar from '../layouts/Sidebar.vue';
    import Loader from '../layouts/Loader';
    import Pagination from 'laravel-vue-pagination';
    import axios from 'axios';
    import Message from '../layouts/Message';

    export default {
        components: {
            'header-layout': Header,
            'sidebar-layout': Sidebar,
            'loader': Loader,
            'pagination': Pagination,
```

```
    'message': Message,
},

data: function () {
    return {
        message: {active: false},
        loader: false,
        users: {},
        searchName: '',
        searchEmail: '',
        role: '',
        pagination: false,
    };

},
created() {
    this.getUsers();
    this.debouncedGetUsers = _.debounce(this.getUsers, 700);
},
watch: {
    searchName(){
        this.debouncedGetUsers();
    },
    searchEmail(){
        this.debouncedGetUsers();
    },
    role(){
        this.getUsers();
    }
},
methods: {
    setRole(role){
        switch (role) {
            case 1:
                return 'Client';
            case 2:
                return 'Admin';
            case 3:
                return 'Residence client'
        }
    },
    getUsers(page = 1) {
        this.loader = true;
        let url = 'adminpanel/clients/?page='
            + page
            + '&searchName=' + this.searchName
            + '&searchEmail=' + this.searchEmail
            + '&role=' + this.role;

        axios.get(url)
            .then(response => {
                this.users = response.data.users;
                this.pagination = response.data.pagination;
                this.loader = false;
            });
    },
    deleteUser(id) {
        if(confirm('Are you sure?')){
            axios
```

```
                        .delete('adminpanel/clients/' + id)
                        .then(response => {
                            this.getUsers();
                        })
                }
            }
        },
    }

</script>

<style lang="sass" scoped>
    @import "~admin"

</style>
```

Component Forms:

```
<template>
 <section>
  <header-layout></header-layout>
  <sidebar-layout></sidebar-layout>
  <div class="wrap">
   <message
    v-if="message.active"
    :type="message.type"
    :message="message.outputMessage">
   </message>
   <div class="ready">
    <table>
     <thead>
     <tr>
      <th style="width: 80px;">№</th>
      <th width="300px">
       <div class="form-group form-group_width_medium">
        <input
         type="text"
         class="form-control"
         placeholder="Name"
         v-model="searchName">
       </div>
      </th>
      <th width="300px">
       <div class="form-group form-group_width_medium">
        <input
         type="text"
         class="form-control"
         placeholder="Email"
         v-model="searchEmail">
       </div>
      </th>
      <th>
       <div class="form-group form-group_width_small">
        <select
         class="form-control"
         id="role"
         v-model="status"
         required>
         <option value="">Status</option>
         <option value="1">Waiting for send</option>
```

```
      <option value="2">Already have sent</option>
      <option value="3">Without user</option>
      <option value="4">Refused from user</option>
      <option value="5">Unfinished</option>
     </select>
    </div>
   </th>
   <th>Date</th>
   <th>
   </th>
  </tr>
  </thead>
  <tr style="border-bottom: none" v-if="loader">
   <td colspan="6"><loader></loader></td>
  </tr>
  <tbody v-else>
  <tr
   v-for="(form, key) in forms.data"
   :key="form.id">
   <td>{{ key + 1 }}</td>
   <td>{{ form.name }}</td>
   <td>{{ form.email }}</td>
   <td>
    {{ form.status }}
   </td>
   <td>{{ dateFormat(form.created_at) }}</td>
   <td style="white-space: nowrap" class="center-align">
    <i
     v-if="form.is_check === 1"
     class="dot"
     v-icons-svg="'dot'">
    </i>
    <router-link
     :to="{ name: 'forms.edit',
                               params: {id: form.id}}"
     tag="span"
     class="edit">
     <span v-icons-svg="'edit'"></span>
    </router-link>
    <!--<span-->
     <!--class="delete"-->
     <!--@click="deleteForm(form.id)">-->
     <!--<span v-icons-svg="'delete'"></span>-->
                          <!--</span>-->
   </td>
  </tr>
  </tbody>
 </table>
 <div v-if="!forms.total && loader === false" class="text-center mt-5">
  <h4>Form list is empty</h4>
 </div>
</div>
<pagination
 :data="forms"
 v-if="pagination"
 :limit="3"
 class="mt-3"
 @pagination-change-page="getForms">
</pagination>
```

```
  </div>
 </section>
</template>

<script>
    import Header from '../layouts/Header.vue';
    import Sidebar from '../layouts/Sidebar.vue';
    import Loader from '../layouts/Loader';
    import Pagination from 'laravel-vue-pagination';
    import axios from 'axios';
    import Message from '../layouts/Message';
    import moment from 'moment';

    export default {

        components: {
            'header-layout': Header,
            'sidebar-layout': Sidebar,
            'loader': Loader,
            'pagination': Pagination,
            'message': Message,
        },

        data: function () {
            return {
                message: {active: false},
                loader: false,
                forms: {},
                searchName: '',
                searchEmail: '',
                status: '',
                pagination: false,
            };

        },
        created() {
            this.getForms();
            this.debouncedGetUsers = _.debounce(this.getForms, 700);
        },
        watch: {
            searchName(){
                this.debouncedGetUsers();
            },
            searchEmail(){
                this.debouncedGetUsers();
            },
            status(){
                this.getForms();
            }
        },
        methods: {
            getForms(page = 1) {
                this.loader = true;
                let url = 'adminpanel/forms/?page='
                    + page
                    + '&searchName=' + this.searchName
                    + '&searchEmail=' + this.searchEmail
                    + '&status=' + this.status;
```

```
                axios.get(url)
                    .then(response => {
                        this.forms = response.data.forms;
                        for(let key in this.forms.data){
                            if(this.forms.data[key].status === 1){
                                this.forms.data[key].status = 'Waiting for send';
                            }else if(this.forms.data[key].status === 2){
                                this.forms.data[key].status = 'Already have sent';
                            }else if(this.forms.data[key].status === 3){
                                this.forms.data[key].status = 'Without user';
                            }else if(this.forms.data[key].status === 4){
                                this.forms.data[key].status = 'Refused from user';
       }else{
        this.forms.data[key].status = 'Unfinished form';
       }
      }
                        this.pagination = response.data.pagination;
                        this.loader = false;
                    });
            },
            deleteForm(id) {
                if(confirm('Are you sure?')){
                    axios
                        .delete('adminpanel/forms/' + id)
                        .then(response => {
                            this.getForms();
                        })
                }
            },
          dateFormat(date){
                return moment(date).format('DD.MM.YYYY');
          }
        },
    }

</script>

<style lang="sass" scoped>
 @import "~admin"
 .dot
  display: inline-block
  margin-right: 20px
  width: 12px


</style>
```

Component AnalyticsForms:

```
<template>
 <section>
  <header-layout></header-layout>
  <sidebar-layout></sidebar-layout>
  <div class="wrap">
   <message
     v-if="message.active"
     :type="message.type"
     :message="message.outputMessage">
   </message>
   <div class="ready">
```

```
<table>
 <thead>
 <tr>
  <th style="width: 80px;">№</th>
  <th width="300px">
   <div class="form-group form-group_width_medium">
    <input
      type="text"
      class="form-control"
      placeholder="Name"
      v-model="searchName">
   </div>
  </th>
  <th width="300px">
   <div class="form-group form-group_width_medium">
    <v-date-picker
      mode='range'
      color="yellow"
      :locale="$i18n.locale"
      :masks="{L: 'DD.MM.YYYY'}"
      :popover="{ placement: 'bottom', visibility: 'click'}"
      v-model='searchDate'>
     <input
       style="border-width: 1px !important;"
       placeholder="Search date"
       autocomplete="off"
       id="date"
       slot-scope="{ inputProps, inputEvents, isDragging }"
       class="form-control"
       v-bind="inputProps"
       v-on="inputEvents">
    </v-date-picker>
   </div>
  </th>
  <th width="350px">
   <div v-if="!loader" class="center-align">
    Find:
    {{ total > 1 ? total + ' forms' : total + ' form' }}
    with user
   </div>
  </th>
 </tr>
 </thead>
 <tr style="border-bottom: none" v-if="loader">
  <td colspan="6"><loader></loader></td>
 </tr>
 <tbody v-else>
 <tr
   v-for="(val, key) in analyticsForms.data"
   :key="val.id">
  <td>{{ key + 1 }}</td>
  <td>{{ val.name }}</td>
  <td>{{ val.created_at }}</td>
  <td class="center-align">
   <router-link
     :to="{ name: 'forms.edit',
                              params: {id: val.client_form_id}}"
     tag="span"
     class="edit">
```

```
        <span v-icons-svg="'eye'"></span>
       </router-link>
      </td>
     </tr>
     </tbody>
    </table>
    <div v-if="!analyticsForms.total && loader === false" class="text-center mt-5">
     <h4>List is empty</h4>
    </div>
   </div>
   <pagination
     :data="analyticsForms"
     v-if="pagination"
     :limit="3"
     class="mt-3"
     @pagination-change-page="getAnalyticsForms">
   </pagination>
  </div>
 </section>
</template>

<script>
    import Header from '../layouts/Header.vue';
    import Sidebar from '../layouts/Sidebar.vue';
    import Loader from '../layouts/Loader';
    import Pagination from 'laravel-vue-pagination';
    import axios from 'axios';
    import Message from '../layouts/Message';
    import moment from 'moment';

    export default {
        components: {
            'header-layout': Header,
            'sidebar-layout': Sidebar,
            'loader': Loader,
            'pagination': Pagination,
            'message': Message,
        },

        data: function () {
            return {
                message: {active: false},
                loader: false,
                analyticsForms: {},
                searchName: '',
                searchDate: {
                    start: '',
                  end: ''
                },
                pagination: false,
              total: '',
            };

        },
        created() {
            this.getAnalyticsForms();
            this.debouncedGetAnalyticsForms = _.debounce(this.getAnalyticsForms, 700);
        },
        watch: {
```

```
            searchName(){
                this.debouncedGetAnalyticsForms();
            },
            searchDate(){
                this.debouncedGetAnalyticsForms();
            },
        },
        methods: {
            getAnalyticsForms(page = 1) {
                this.loader = true;
                let date = {
                    start: '',
                  end: ''
                };
                if(this.searchDate.start){
                    date.start = moment(this.searchDate.start).format('YYYY-MM-DD
h:mm:ss');
                    date.end = moment(this.searchDate.end).format('YYYY-MM-DD
h:mm:ss');
                }
                let url = 'adminpanel/analytics-forms/?page='
                    + page
                    + '&searchName=' + this.searchName
                    + '&date=' + (date.start ? JSON.stringify(date) : '');

                axios.get(url)
                    .then(response => {
                        this.analyticsForms = response.data.analyticsForms;
                        this.total = response.data.total;
                        for(let key in this.analyticsForms.data){
                          let time = this.analyticsForms.data[key].created_at.split('
')[1];
                          let date = this.analyticsForms.data[key].created_at.split('
')[0].split('-');
                          this.analyticsForms.data[key].created_at = date[2] + '.' +
date[1] + '.' + date[0] + ' ' + time
                        }
                        this.pagination = response.data.pagination;
                        this.loader = false;
                    });
            },
        },
    }

</script>

<style lang="sass" scoped>
 @import "~admin"

</style>
```

Component Neighborhoods:

```
<template>
                                                <section>
                                                 <header-layout></header-layout>
                                                 <sidebar-layout></sidebar-layout>
                                                 <div class="wrap">
                                                        <message
                                                               v-if="message.active"
```

Продовження Додатку А

```
                        :type="message.type"
                        :message="message.outputMessage">
                    </message>
                    <div class="ready">
                        <table>
                            <thead>
                            <tr>
                                <th style="min-width: 40px;">№</th>
                                <th width="300px">
                                    <div class="form-group form-group_width_small">
                                        <input type="text" class="form-control" placeholder="Name" v-model="searchName">
                                    </div>
                                </th>
                                <th width="400px">
                                    <select class="form-control" id="project" v-model="regionSearchId" required>
                                        <option value="" selected>- Choose region -</option>
                                        <option v-for="region in regions" :key="region.id" :value="region.id">{{ region.name_en }}</option>
                                    </select>
                                </th>
                                <th>
```

```
                                                                <div
class="form-group form-group_width_small btn_add">
                                                    <router-link
                                                    :to="{ name: 'neighborhoods.create' }"
                                                    tag="button"
                                                    class="btn btn-primary">
                                                    + Add neighborhood
                                                    </router-link>
                                                                </div>
                                                            </th>
                                                        </tr>
                                                        </thead>
                                                        <tr style="border-
bottom: none" v-if="loader">
                                                            <td
colspan="4"><loader></loader></td>
                                                        </tr>
                                                        <tbody v-else>
                                                        <tr
                                                            v-
for="(neighborhood, key) in neighborhoods.data"
                                                    :key="neighborhood.id">
                                                            <td>{{ key + 1
}}</td>
                                                            <td>{{
neighborhood.name_en }}</td>
                                                            <td>
                                                                {{
neighborhood.region_name }}
                                                            </td>
                                                            <td
class="center-align">
                                                                <router-
link
                                                    :to="{ name: 'neighborhoods.edit',
                                    params: {id: neighborhood.id}}"
                                                    tag="span"
                                                    class="edit">
                                                    <span v-icons-svg="'edit'"></span>
                                                                </router-
link>
                                                                <span
                                                    class="delete"
                                                    @click="deleteNeighborhood(neighborhood.id)
">
```

```
                                        <span v-icons-svg="'delete'"></span>
                                </span>
                                                                        </td>
                                                                </tr>
                                                                </tbody>
                                                        </table>
                                                        <div v-
if="!neighborhoods.total && loader === false" class="text-center mt-5">
                                                                <h4>Neighborhood list
is empty</h4>
                                                        </div>
                                                </div>
                                                <pagination
                                                        :data="neighborhoods"
                                                        v-if="pagination"
                                                        :limit="3"
                                                        class="mt-3"
                                                        @pagination-change-
page="getNeighborhoods">
                                                </pagination>
                                        </div>
                                </section>
</template>

<script>
    import Header from '../layouts/Header.vue';
    import Sidebar from '../layouts/Sidebar.vue';
    import Message from '../layouts/Message';
    import Loader from '../layouts/Loader';
    import axios from 'axios';
    import Pagination from 'laravel-vue-pagination';

    export default {
        components: {
            'header-layout': Header,
            'sidebar-layout': Sidebar,
            'message': Message,
            'loader': Loader,
            'pagination': Pagination,
        },
        data() {
            return {
                loader: false,
                message: {active: false},
                neighborhoods: {},
                                                        regions: {},
                pagination: false,
                searchName: '',
                regionSearchId: '',
            }
        },
        created() {
            this.getNeighborhoods();
            this.debouncedGetRegion = _.debounce(this.getNeighborhoods, 700);
        },
        watch: {
            searchName(){
                this.debouncedGetRegion();
```

```
            },
            regionSearchId(){
                this.getNeighborhoods();
            },
        },
        methods: {
            backToPreviousPage(){
                window.history.back();
            },
            getNeighborhoods(page = 1){
                this.loader = true;
                let url = 'adminpanel/neighborhoods/?page='
                    + page
                    + '&searchName=' + this.searchName
                    + '&regionSearchId=' + this.regionSearchId;
                axios
                    .get(url)
                    .then(response => {
                        this.neighborhoods = response.data.neighborhoods.neighborhoods;
                        this.pagination = response.data.neighborhoods.pagination;
                        this.regions = response.data.regions;
                        this.loader = false;
                    })
            },
            deleteNeighborhood($id){
                if(confirm('Are you sure?')) {
                    this.loader = true;
                    axios
                        .delete('adminpanel/delete-neighborhood/' + $id)
                        .then(response => {
                            this.getNeighborhoods();
                        })
                }
            }
        },
    }

</script>

<style lang="sass" scoped>
                                            @import "~admin"

</style>
```

Component Regions:

```
<template>
 <section>
  <header-layout></header-layout>
  <sidebar-layout></sidebar-layout>
  <div class="wrap">
   <message
    v-if="message.active"
    :type="message.type"
    :message="message.outputMessage">
   </message>
   <div class="ready">
    <table>
     <thead>
     <tr>
```

```
    <th style="width: 120px">№</th>
    <th>
     <div class="form-group form-group_width_small">
      <input
       style="max-width: 300px;"
       type="text"
       class="form-control"
       placeholder="Name"
       v-model="searchName">
     </div>
    </th>
    <th>
     <div class="form-group form-group_width_small btn_add">
      <router-link
       :to="{name: 'regions.create'}"
       tag="button"
       class="btn btn-primary">
       + Add region
      </router-link>
     </div>
    </th>
   </tr>
   </thead>
   <tr style="border-bottom: none" v-if="loader">
    <td colspan="3"><loader></loader></td>
   </tr>
   <tbody v-else>
   <tr
    v-for="(region, key) in regions.data"
    :key="region.id">
    <td>{{ key + 1 }}</td>
    <td>{{ region.name_en }}</td>
    <td class="center-align">
     <router-link
      :to="{ name: 'regions.edit',
                                params: {id: region.id}}"
      tag="span"
      class="edit">
      <span v-icons-svg="'edit'"></span>
     </router-link>
     <span
      class="delete"
      @click="deleteRegion(region.id)">
      <span v-icons-svg="'delete'"></span>
                           </span>
    </td>
   </tr>
   </tbody>
  </table>
  <div v-if="!regions.total && loader === false" class="text-center mt-5">
   <h4>Region list is empty</h4>
  </div>
 </div>
 <pagination
  :data="regions"
  v-if="pagination"
  :limit="3"
  class="mt-3"
  @pagination-change-page="getRegions">
```

```
   </pagination>
  </div>
 </section>
</template>

<script>
    import Header from '../layouts/Header.vue';
    import Sidebar from '../layouts/Sidebar.vue';
    import Message from '../layouts/Message';
    import Loader from '../layouts/Loader';
    import axios from 'axios';
    import Pagination from 'laravel-vue-pagination';

    export default {
        components: {
            'header-layout': Header,
            'sidebar-layout': Sidebar,
            'message': Message,
            'loader': Loader,
            'pagination': Pagination,
        },
        data() {
            return {
                loader: false,
                message: {active: false},
    regions: {},
    pagination: false,
    searchName: '',

            }
        },
        created() {
   this.getRegions();
            this.debouncedGetRegion = _.debounce(this.getRegions, 700);
        },
        watch: {
            searchName(){
                this.debouncedGetRegion();
            },
        },
        methods: {
            backToPreviousPage(){
                window.history.back();
            },
   getRegions(page = 1){
    this.loader = true;
                let url = 'adminpanel/regions/?page='
                    + page
                    + '&searchName=' + this.searchName;

    axios
     .get(url)
     .then(response => {
         this.regions = response.data.regions;
                        this.pagination = response.data.pagination;
         this.loader = false;
     })
   },
   deleteRegion($id){
```

```
                if(confirm('Are you sure?')) {
                    this.loader = true;

                    axios
                        .delete('adminpanel/delete-region/' + $id)
                        .then(response => {
                            this.getRegions();
                        })
                }
    }
         },
     }

</script>

<style lang="sass" scoped>
 @import "~admin"

</style>
```

Component ClientForm:

```
<template>
  <div class="client_form">
    <transition name="fade"
                class="fade">
      <message
          v-if="message.active"
          :type="message.type"
          :message="message.outputMessage">
      </message>
      <div
          v-if="loader"
          key="loader"
          class="pre-loader">
      </div>
      <div v-else
           class="contact_form">
        <form
            @keypress.enter.prevent
            @submit.prevent="validate">
          <div class="form-group">
            <input
                type="text"
                class="form-control"
                id="name"
                name="name"
                v-model="formData.name"
                required>
            <label for="name">{{ $t('name') }}</label>
          </div>
          <div class="form-group">
            <input
                type="email"
                class="form-control"
                id="email"
                name="email"
                v-model="formData.email"
                required>
            <label for="email">{{ $t('email') }}</label>
```

```
            <div
                v-if="!exist_email"
                class="invalid">
              {{ $t('validate.email') }}
            </div>
          </div>
          <div class="form-group">
            <the-mask
                required
                autocomplete="off"
                type="tel"
                class="form-control"
                id="phone_num"
                :mask="['(###) ###-####']"
                pattern="\([0-9]{3}\)\s[0-9]{3}-[0-9]{4}"
                v-model="formData.phone_number">
            </the-mask>
            <label for="phone_num">{{ $t('phone') }}</label>
            <div
                v-show="invalid.phone"
                class="invalid">
              {{ $t('validate.phone') }}
            </div>
          </div>
          <div class="location">
            <div
                v-if="tagsPlace.length"
                class="location_tags">
              <div
                  class="location_tags_item"
                  v-for="(tag, key) in tagsPlace">
                <div class="location_tags_item_name">

                                              <span>

                                                {{ tag.neighborhood_name }} {{
tag.region_name ? ', ' + tag.region_name : '' }}

                                              </span>
                </div>
                <div class="location_tags_item_delete">

                                              <span
                      @click="deleteNeighborhood(key)"
                      v-icons-svg="'close-icon'">

                                              </span>
                </div>
              </div>
            </div>
            <div class="location_place">
              <div
                  style="position: relative; margin-bottom: 0;"
                  class="form-group">
                <input
                    autocomplete="off"
                    id="search"
                    v-model="searchNeighborhood"
                    class="form-control"
```

```
                @click="setPopular"
                type="text">
              <span class="search-icon"
                    v-icons-svg="'search'"></span>
              <label>{{$t('neighborhoods')}}</label>
              <div
                  v-if="neighborhoods.length"
                  class="searched">
                <div
                    v-for="neighborhood in neighborhoods"
                    @click="addTagPlace(

                                                neighborhood.region_id,

                                                neighborhood.neighborhood_id,

                                                neighborhood.region_name,

                                                neighborhood.neighborhood_name)"
                    class="searched_item">

                                            <span>

                                                {{ neighborhood.region_id

                                                  ? neighborhood.full_name

                                                  : neighborhood.neighborhood_name}}

                                            </span>
                </div>
              </div>
              <div
                  v-show="invalid.neighborhoods"
                  class="invalid">
                {{ $t('validate.neighborhood') }}
              </div>
            </div>
          </div>
        </div>
        <div
            style="margin-bottom: 2rem"
            class="form-group">
                                                                    <span
          style="text-transform: uppercase; color: #665a56; margin-bottom:
0.5rem; display: inline-block">
                                                                        {{
$t('saleTitle') }}
                                                                    </span>
          <div class="raw">
            <input
                v-model="formData.sale"
                class="checkbox"
                type="checkbox"
                id="sale"
                value="1">
            <label class="label_style_font"
                   for="sale">
              {{ $t('sale') }}
```

```
    </label>
  </div>
</div>
<div
    style="margin-bottom: 2rem"
    class="checking form-group">
  <span style="margin-bottom: 0.5rem;">{{ $t('bedrooms') }}</span>
  <div
      class="raw responsive_bedroom">
    <input
        v-model="formData.bedroom"
        class="checkbox"
        type="checkbox"
        id="bedroom0"
        value="0">
    <label style="font-size: 15px;"
           class="mr-3 label_style_font"
           for="bedroom0">
      {{ $t('studio') }}
    </label>
    <input
        v-model="formData.bedroom"
        class="checkbox"
        type="checkbox"
        id="bedroom1"
        value="1">
    <label class="label_style_font mr-3"
           for="bedroom1">1</label>
    <input
        v-model="formData.bedroom"
        class="checkbox"
        type="checkbox"
        id="bedroom2"
        value="2">
    <label class="label_style_font mr-3"
           for="bedroom2">2</label>
    <input
        v-model="formData.bedroom"
        class="checkbox"
        type="checkbox"
        id="bedroom3"
        value="3">
    <label class="label_style_font mr-3"
           for="bedroom3">3</label>
    <input
        v-model="formData.bedroom"
        class="checkbox"
        type="checkbox"
        id="bedroom4"
        value="4">
    <label class="label_style_font mr-3"
           for="bedroom4">4</label>
    <input
        v-model="formData.bedroom"
        class="checkbox"
        type="checkbox"
        id="bedroom5"
        value="5">
    <label class="label_style_font mr-3"
```

```
                for="bedroom5">5</label>
          </div>
          <div
              v-show="invalid.bedrooms"
              class="invalid">
            {{ $t('validate.bedroom') }}
          </div>
        </div>
        <div
            style="margin-bottom: 2rem"
            class="form-group">
                                                                    <span
            style="text-transform: uppercase; color: #665a56; margin-bottom:
0.5rem; display: inline-block">
                                                                          {{
$t('type') }}
                                                                    </span>
          <div class="raw">
            <input
                v-model="formData.buy"
                class="checkbox"
                type="checkbox"
                id="buy"
                value="1">
            <label class="mr-3 label_style_font"
                   for="buy">
              {{ $t('buy') }}
            </label>
            <input
                v-model="formData.rent"
                class="checkbox"
                type="checkbox"
                id="rent"
                value="1">
            <label class="mr-3 label_style_font"
                   for="rent">
              {{ $t('rent') }}
            </label>
          </div>
          <div
              v-show="invalid.type"
              class="invalid">
            {{ $t('validate.type') }}
          </div>
        </div>
        <div
            class="form-group price-group"
            style="margin-bottom: 2rem"
            v-bind:class="{ 'english' : this.$route.params.language === 'en' }">
          <label class="responsive_margin"
                 for="price_range">
            {{ $t('price') }}
            {{ onlyRentPrice ? '(' + $t('per_month') + ')' : '' }}
          </label>
          <div class="row responsive">
            <div
                class="col responsive_margin">
              <label
```

```
                    style="margin-right: 5px; text-transform: capitalize; white-space:
nowrap"
                    for="date-from">
                  {{ $t('from') }}
                </label>
                <span>- {{ this.$route.params.language === 'en' ? '$ ' : '' }}</span>
                <input
                    @keypress.enter="updatePrice(1)"
                    @change="updatePrice(1)"
                    :min="rangePrice.min"
                    :max="rangePrice.max"
                    id="date-from"
                    step="10"
                    v-model="inputPrice[0]"
                    type="number"
                    style="background-color: rgba(0,0,0,0)"
                    class="form-control">
                {{ this.$route.params.language === 'fr' ? '$ ' : '' }}
              </div>
              <div class="col responsive_margin">
                <label
                    style="margin-right: 5px; text-transform: capitalize; white-space:
nowrap"
                    for="date-to">{{ $t('to') }}
                </label>
                <span>- {{ this.$route.params.language === 'en' ? '$ ' : '' }}</span>
                <input
                    @keypress.enter="updatePrice(2)"
                    @change="updatePrice(2)"
                    :min="formData.price[0]"
                    :max="rangePrice.max"
                    id="date-to"
                    step="10"
                    v-model="inputPrice[1]"
                    type="number"
                    style="background-color: rgba(0,0,0,0)"
                    class="form-control">
                {{ this.$route.params.language === 'fr' ? '$ ' : '' }}
              </div>
            </div>
            <div class="range">
              <vue-slider
                  id="price_range"
                  :enableCross="false"
                  :min="rangePrice.min"
                  :max="rangePrice.max"
                  :silent="true"
                  :interval="10"
                  :use-keyboard="true"
                  :tooltip-formatter="$i18n.locale === 'en' ? formatter_en :
formatter_fr"
                  :tooltip-placement="['top', 'bottom']"
                  v-model="formData.price"
                  :process-style="{ backgroundColor: '#fbcf00' }"
                  :tooltip-style="{ backgroundColor: '#fbcf00', borderColor: '#fbcf00'
}"
                  :tooltip="'always'">
              </vue-slider>
            </div>
```

Продовження Додатку А

```
</div>
<div v-if="dateRangeMode" style="position: relative;"
     class="form-group">
  <select id="date_range" class="form-control"
          v-model="formData.date_range">
    <option v-for="val in dateRanges" :value="val.id">
      {{ val['name_' + $i18n.locale] }}
    </option>
  </select>
  <label for="date_range">{{ $t('date') }}</label>
</div>
<div v-else style="position: relative;"
     class="form-group">
  <v-date-picker
      mode='range'
      color="yellow"
      :locale="$i18n.locale"
      :masks="{L: 'DD.MM.YYYY'}"
      :popover="{ placement: 'bottom', visibility: 'click'}"
      v-model='formData.date'>
    <input
        style="border-width: 1px !important;"
        autocomplete="off"
        id="date"
        slot-scope="{ inputProps, inputEvents, isDragging }"
        class="form-control"
        v-bind="inputProps"
        v-on="inputEvents">
  </v-date-picker>
  <label for="date">{{ $t('date') }}</label>
</div>
<div style="margin-bottom: 2rem;"
     class="form-group">
  <input
      type="text"
      class="form-control"
      id="additional"
      name="additional"
      v-model="formData.additional">
  <label for="additional">{{ $t('additional') }}</label>
</div>
<div class="form-group">
  <div class="raw">
    <input
        v-model="formData.subscribe"
        class="checkbox"
        type="checkbox"
        id="subscribe"
        :value="+1">
    <label class="label_style_font"
           for="subscribe">{{
      $t('subscribe')
      }}</label>
  </div>
</div>
<div
    style="margin-bottom: 0.5rem"
    class="form-group">
  <div class="raw">
```

```
        <input
            v-model="formData.refer"
            class="checkbox"
            type="checkbox"
            id="refer"
            :value="+1">
        <label class="label_style_font"
               for="refer">{{ $t('refer') }}</label>
      </div>
    </div>
    <div class="form-group row">
      <div class="col-md-6 offset-md-4">
        <vue-recaptcha
            style="opacity: 0"
            ref="recaptcha"
            size="invisible"
            :sitekey="sitekey"
            @verify="sendForm"
            @expired="onCaptchaExpired"
        />
      </div>
    </div>
    <div class="actions-button">
      <button
          :disabled="loaderBtn"
          class="btn-send"
          v-html="btnContent">
      </button>
    </div>
  </form>
  <div class="modal fade"
       id="modal"
       tabindex="-1"
       role="dialog"
       aria-labelledby="exampleModalScrollableTitle"
       aria-hidden="true">
    <div class="modal-dialog modal-dialog-centered"
         role="document">
      <div class="modal-content">
        <div class="modal-header">
          <button type="button"
                  class="close"
                  data-dismiss="modal"
                  aria-label="Close">
            <span aria-hidden="true">&times;</span>
          </button>
        </div>
        <div class="modal-body">
          <img src="/icons/success.png"
               alt="success"
               class="modal-icon">
          <span>{{ $t('modalMessage') }}</span>
        </div>
      </div>
    </div>
  </div>
  <div v-if="loaderBtn"
       class="block_content"></div>
</div>
```

```
    </transition>
  </div>
</template>

<script>
    import axios from 'axios'
    import VueSlider from 'vue-slider-component'
    import 'vue-slider-component/theme/default.css'
    import VueRecaptcha from 'vue-recaptcha'
    import Message from '../admin/layouts/Message'

    export default {
        components: {
            VueSlider,
            VueRecaptcha,
            Message
        },
        data() {
            return {
                loader: false,
                unfinishedForm: true,
                message: {active: false},
                forReset: true,
                btnContent: '',
                loaderBtn: false,
                sitekey: '6LfGCagUAAAAAD0uQ-L43NeIstEQ-MLYWkICdK9w',
                invalid: {
                    phone: false,
                    neighborhoods: false,
                    bedrooms: false,
                    type: false,
                },
                defaultPrice: [],
                rangePrice: {
                    min: 0,
                    max: 0
                },
                inputPrice: ['', ''],
                searchNeighborhood: '',
                neighborhoods: {},
                tagsPlace: [],
                formData: {
                    id: 0,
                    language: '',
                    date: {
                        start: null,
                        end: null
                    },
                    bedroom: [],
                    refer: 1,
                    subscribe: 1,
                    price: [0, 0],
                    date_range: '',
                },
                onlyRentPrice: false,
                formatter_en: v => `$${('' + v).replace(/\B(?=(\d{3})+(?!\d))/g,
',')}`,
                formatter_fr: v => `${('' + v).replace(/\B(?=(\d{3})+(?!\d))/g, ' ')}
$`,
```

```
        popularNeighborhoods: {},
        deletedPopularNeighborhoods: {},
        exist_email: 1,
        dateRanges: {},
        dateRangeMode: 0,
    }
},
watch: {
    searchNeighborhood() {
        this.debouncedGetNeighborhoods()
    },
    'formData.price': function (newValue, oldValue) {
        if (this.forReset) {
            if (this.inputPrice[0] || oldValue[0] !== newValue[0]) {
                this.inputPrice.shift()
                this.inputPrice.unshift(this.formData.price[0])
            }
            if (this.inputPrice[1] || oldValue[1] !== newValue[1]) {
                this.inputPrice.pop()
                this.inputPrice.push(this.formData.price[1])
            }
        } else {
            this.forReset = true
        }
    },
    'formData.bedroom': function () {
        this.invalid.bedrooms = false
    },
    'formData.rent': function () {
        this.invalid.type = false
        this.checkPriceRange()
    },
    'formData.buy': function () {
        this.invalid.type = false
        this.checkPriceRange()
    },
    'formData.name': function () {
        this.debouncedUnfinishedForm()
    },
    'formData.email': function () {
        this.debouncedCheckEmail()
        this.debouncedUnfinishedForm()
    },
    'formData.phone_number': function () {
        this.invalid.phone = false
        this.debouncedUnfinishedForm()
    }
},
mounted() {
    $(document).click(e => {
        let search = $('#search')
        let searched = $('.searched')

        if (!search.is(e.target) && search.has(e.target).length === 0
            && !searched.is(e.target) && searched.has(e.target).length === 0) {
            this.searchNeighborhood = ''
            this.neighborhoods = {}
        }
    })
```

```
},
created() {
    try {
        if (this.$route.params.language === 'fr') {
            this.$i18n.locale = this.$route.params.language
        }
    } catch (e) {
    }
    this.formData.language = this.$i18n.locale
    this.$nextTick(() => {
        this.btnContent = this.$t('send')
    })
    this.debouncedGetNeighborhoods = _.debounce(this.getNeighborhoods, 700)
    this.debouncedCheckEmail = _.debounce(this.checkEmail, 500)
    this.debouncedUnfinishedForm = _.debounce(this.checkUnfinishedForm, 1000)
    this.getDateRanges()
    this.getRangePrice()
    this.getPopularNeighborhoods()
},

methods: {
    checkEmail() {
        let url = 'client-panel/check-form-email/' + this.formData.email

        axios
            .get(url)
            .then(response => {
                this.exist_email = response.data
            })
            .catch(error => {
                this.exist_email = 1
            })
    },
    setPopular() {
        if (!Object.keys(this.neighborhoods).length) {
            this.neighborhoods = this.popularNeighborhoods.data
        }
    },
    getPopularNeighborhoods() {
        let url = 'client-panel/get-popular-neighborhoods/' + this.$i18n.locale

        axios
            .get(url)
            .then(response => {
                this.popularNeighborhoods = response.data
            })
    },
    getDateRanges() {
        let url = 'client-panel/get-date-ranges';

        axios
            .get(url)
            .then(response => {
                this.dateRanges = response.data
            })
    },
    checkPriceRange() {
        if (
            !this.formData.rent && !this.formData.buy
```

```
                || !this.formData.rent && this.formData.buy
                || this.formData.rent && this.formData.buy
            ) {
                this.rangePrice.min = this.min_price
                this.rangePrice.max = this.max_price

                this.onlyRentPrice = false
                this.formData.price = [
                    this.default_min_price,
                    this.default_max_price
                ]

            } else {
                this.formData.price = [
                    this.min_rent_price,
                    this.max_rent_price
                ]
                this.rangePrice.min = this.min_default_rent_price
                this.rangePrice.max = this.max_default_rent_price
                this.onlyRentPrice = true
            }

            this.forReset = false
        },

        checkUnfinishedForm() {
            var check = 1

            if (!this.formData.name || this.formData.name.length == 0)
                check = 0

            if (!this.formData.email || this.formData.email.indexOf('@') == -1)
                check = 0

            if (!this.formData.phone_number || this.formData.phone_number.length <
10)
                check = 0

            if (check == 1)
                this.sendUnfinishedForm()
        },

        sendUnfinishedForm() {
            if (this.unfinishedForm || this.formData.id) {
                this.unfinishedForm = false
                var formData = {
                    'name': this.formData.name,
                    'email': this.formData.email,
                    'phone_number': this.formData.phone_number,
                    'language': this.$i18n.locale
                }

                if (this.formData.id > 0)
                    formData['id'] = this.formData.id

                axios
                    .post('client-panel/send-unfinished-form', formData)
                    .then(response => {
                        this.unfinishedForm = true
```

```
                        this.formData.id = response.data.id
                    })
            }
        },

        getRangePrice() {
            this.loader = true
            axios
                .get('/client-panel/get-range-price')
                .then(response => {
                    this.rangePrice.min = response.data.min
                    this.rangePrice.max = response.data.max
                    this.min_rent_price = response.data.min_rent_price
                    this.max_rent_price = response.data.max_rent_price
                    this.min_default_rent_price =
response.data.min_default_rent_price
                    this.max_default_rent_price =
response.data.max_default_rent_price
                    this.dateRangeMode = response.data.date_range

                    if (response.data.default_min >= response.data.min) {
                        this.defaultPrice[0] = response.data.default_min
                        this.formData.price[0] = response.data.default_min
                        this.inputPrice[0] = response.data.default_min
                    } else {
                        this.defaultPrice[0] = response.data.min
                        this.formData.price[0] = response.data.min
                        this.inputPrice[0] = response.data.min
                    }
                    if (response.data.default_max <= response.data.max) {
                        this.defaultPrice[1] = response.data.default_max
                        this.formData.price[1] = response.data.default_max
                        this.inputPrice[1] = response.data.default_max
                    } else {
                        this.defaultPrice[1] = response.data.max
                        this.formData.price[1] = response.data.max
                        this.inputPrice[1] = response.data.max
                    }
                    this.loader = false

                    this.default_min_price = this.defaultPrice[0]
                    this.default_max_price = this.defaultPrice[1]
                    this.min_price = this.rangePrice.min
                    this.max_price = this.rangePrice.max
                })
        },
        getNeighborhoods() {
            if (this.searchNeighborhood !== '') {
                let url = '/client-panel/get-neighborhoods'
                let formData = new FormData()

                formData.append('language', this.$i18n.locale)
                formData.append('search', this.searchNeighborhood)
                formData.append('tags', JSON.stringify(this.tagsPlace))

                axios
                    .post(url, formData)
                    .then(response => {
                        this.neighborhoods = response.data
```

```
                    })
                }
            },
            sendForm(recaptchaToken) {
                if (!this.exist_email) {
                    return
                }
                if (this.validation()) {
                    this.loaderBtn = true
                    this.btnContent = `<div
                                                                    class="spinner-
border text-warning"
                                                                    style="width:
20px; height: 20px;"
                                                                    role="status">
                                                                    <span class="sr-
only">Loading...</span>
                                                                </div>`
                    let formData = new FormData()

                    //                if(this.formData.rent && !this.formData.buy){
                    //                    this.formData.price = [0,0];
                    // }

                    formData.append('formData', JSON.stringify(this.formData))
                    formData.append('tagsPlace', JSON.stringify(this.tagsPlace))
                    formData.append('recaptcha_token', recaptchaToken)

                    axios
                        .post('client-panel/send-form', formData)
                        .then(response => {
                            setTimeout(() => {
                                this.formData = {
                                    language: this.$i18n.locale,
                                    date: {
                                        start: null,
                                        end: null
                                    },
                                    bedroom: [],
                                    refer: 1,
                                    subscribe: 1,
                                    price: [this.defaultPrice[0],
this.defaultPrice[1]],
                                    date_range: ''
                                }
                                this.getPopularNeighborhoods()
                                this.deletedPopularNeighborhoods = {}
                                this.forReset = false
                                this.inputPrice = [this.defaultPrice[0],
this.defaultPrice[1]]
                                this.tagsPlace = []
                                this.searchNeighborhood = ''
                                this.neighborhoods = {}
                                this.loaderBtn = false
                                this.btnContent = this.$t('send')
                                this.$refs.recaptcha.reset()
                                $('#modal').modal('show')
                                setTimeout(() => {
                                    $('#modal').modal('hide')
```

```
                            }, 5500)
                        }, 500)
                    })
                    .catch(error => {
                        this.message.outputMessage = error.response.data.error
                        this.message.type = 'error'
                        this.message.active = true
                        this.loaderBtn = false
                        this.btnContent = this.$t('send')
                        setTimeout(() => {
                            this.message = {active: false}
                        }, 3000)
                    })
            }
        },
        addTagPlace(regionId, neighborhoodId, regionName, neighborhoodName) {
            let data = {
                'region_id': regionId,
                'neighborhood_id': neighborhoodId,
                'region_name': regionName,
                'neighborhood_name': neighborhoodName
            }

            for (let key in this.popularNeighborhoods.data) {
                if (this.popularNeighborhoods.data[key].neighborhood_id ==
neighborhoodId) {
                    this.deletedPopularNeighborhoods[neighborhoodId] =
this.popularNeighborhoods.data[key]
                    this.popularNeighborhoods.data.splice(key, 1)
                    break
                }
            }

            this.tagsPlace.push(data)

            this.invalid.neighborhoods = false
            this.searchNeighborhood = ''
            this.neighborhoods = {}
        },
        deleteNeighborhood(key) {
            for (let k in this.deletedPopularNeighborhoods) {
                if (this.tagsPlace[key].neighborhood_id ==
this.deletedPopularNeighborhoods[k].neighborhood_id) {

this.popularNeighborhoods.data.push(this.deletedPopularNeighborhoods[k])
                    delete this.deletedPopularNeighborhoods[k]
                    break
                }
            }

            this.tagsPlace.splice(key, 1)
        },
        validation() {
            let is_valid = true
            if (!this.formData.bedroom.length) {
                this.invalid.bedrooms = true
                is_valid = false
            }
            if (!this.tagsPlace.length) {
```

```
                this.invalid.neighborhoods = true
                is_valid = false
            }
            if (!this.formData.rent && !this.formData.buy) {
                this.invalid.type = true
                is_valid = false
            }
            if (!this.formData.phone_number.length) {
                this.invalid.phone = true
                is_valid = false
            }
            this.$refs.recaptcha.reset()
            return is_valid
        },
        updatePrice(changed) {
            if (changed === 1) {
                if (!this.inputPrice[0]) {
                    this.inputPrice.shift()
                    this.inputPrice.unshift(this.rangePrice.min)
                }
                if (+this.inputPrice[0] > +this.formData.price[1]) {
                    this.inputPrice.shift()
                    this.inputPrice.unshift(+this.formData.price[1])
                }
                if (+this.inputPrice[0] < this.rangePrice.min) {
                    this.inputPrice.shift()
                    this.inputPrice.unshift(this.rangePrice.min)
                }
            } else {
                if (!this.inputPrice[1]) {
                    this.inputPrice.pop()
                    this.inputPrice.push(this.inputPrice[0])
                }
                if (+this.inputPrice[1] < +this.formData.price[0]) {
                    this.inputPrice.pop()
                    this.inputPrice.push(this.formData.price[0])
                }
                if (+this.inputPrice[1] > +this.rangePrice.max) {
                    this.inputPrice.pop()
                    this.inputPrice.push(this.rangePrice.max)
                }
            }
            if (this.inputPrice[0]) {
                this.formData.price.shift()
                this.formData.price.unshift(this.inputPrice[0])
            }
            if (this.inputPrice[1]) {
                this.formData.price.pop()
                this.formData.price.push(this.inputPrice[1])
            }
        },
        onCaptchaExpired() {
            this.$refs.recaptcha.reset()
        },
        validate() {
            this.$refs.recaptcha.execute()
        }
    }
</script>
```