Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.912

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема: Веб-додаток для агрегування і групування нерухомості на основі користувацьких фільтрів з використанням машинного навчання

Спеціальність ─ 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент

ІПЗм-21 _____/Максим НАВРОЦЬКИЙ/

Науковий керівник

к.ф.–м.н., доц._____/Ірина ЮРЧУК/

Консультант

з питань нормоконтролю

к.т.н., асист.____/Анастасія ВЕЧЕРКОВСЬКА /

Допускається до захисту

Завідувач кафедри

д.т.н., проф. _____ _/Олексій БИЧКОВ/

Київ – 2022

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії

професор, доктор техн. наук Андрій БОНДАРЧУК

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____(Олексій БИЧКОВ)
„_____” _____ 2022 р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
СТУДЕНТУ

Навроцькому Максиму Володимировичу
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної магістерської роботи “Веб-додаток для агрегування і групування нерухомості на основі користувацьких фільтрів з використанням машинного навчання”, керівник роботи доцент Юрчук І.А., затверджені на засіданні кафедри програмних системі технологій, протокол №8 від «21» грудня 2021р.

2**.** Строк здачі студентом закінченої роботи _____

3. Вихідні дані до дипломної роботи: фотографії, підручники, навчальніпосібники, статті та тези конференцій вітчизняних і зарубіжних авторів, Інтернет–ресурси з питань розробки рішень на основі нейронних мереж.

4. Зміст пояснювальної записки:

1) Розділ 1: проаналізувати існуючі програмні рішення та засоби класифікації промислових зображень, обґрунтувати доцільність обраної теми дослідження, порівняти з вже існуючими рішеннями на українському та зарубіжному ринку.

2) Розділ 2: спроектувати архітектуру програмного забезпечення для агрегування і групування нерухомості з використанням технологій нейронних мереж, розробити програмне забезпечення, розробити інтерфейс користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1) Розділ 1: 54 рисунки, 3 формули.

2) Розділ 2: 11 рисунків.

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1 розділ	Юрчук І.А.	22.12.2021	22.12.2021
2 розділ	Юрчук І.А.	22.12.2021	22.12.2021

7. Дата видачі завдання _____

Керівник _____ (Ірина ЮРЧУК)

Завдання прийняв до виконання _____ (Максим НАВРОЦЬКИЙ)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Уточнення постановки задачі	29.12.2021–15.01.2022	Виконано
2	Аналіз літератури	23.12.2021–04.02.2022	Виконано
3	Аналіз існуючих методів, концепцій та алгоритмів вирішення завдання	23.12.2021–16.02.2022	Виконано
4	Побудова алгоритмічної моделі основнихпроцесів	28.01.2022–14.02.2022	Виконано
5	Опис розробленого алгоритму	15.02.2022–20.03.2022	Виконано
6	Розроблення програмного забезпечення	21.03.2022–30.04.2022	Виконано
7	Тестування розробленого програмного забезпечення	02.05.2022–15.05.2022	Виконано
8	Оформлення і друк пояснювальної записки	16.05.2022–26.05.2022	Виконано
9	Оформлення презентації	14.05.2022–22.05.2022	Виконано
10	Отримання рецензії		Виконано
11	Затвердження пояснювальної записки роботи завідувачем кафедри		Виконано
12	Захист дипломної роботи	26.05.2022	Виконано

Студент _____ Максим НАВРОЦЬКИЙ

Керівник роботи _____ Ірина ЮРЧУК

АНОТАЦІЯ

Випускна кваліфікаційна магістерська робота: 75 с., 65 рис., 1 табл., 1 додат., 16 джерел.

Тема: Веб-додаток для агрегування і групування нерухомості на основі користувацьких фільтрів з використанням машинного навчання.

Об’єкт дослідження: нейронна мережа з власною архітектурою та моделлю для комерційних потреб.

Мета роботи*:* створення веб додатку та комплексної інтеграції API на базі нейронної мережі на комерційній основі.

Предмет дослідження: архітектура додатку та нейронної мережі і її модель для виконання специфічної задачі.

Результати дослідження: розроблена та коммерційно впроваджена власна модель нейронної мережі, яка окрім базової класифікації зображення спроможна виконувати специфічну задачу у комплексному додатку. Окрім власне розробки повноцінного додатку, була проведена робота по збору датасету і його попердній оброці усередині додатку для подальшого процесу тренування.

Висновок

Повністю розроблений додаток готовий для комерійного корисутвання у сфері нерухомості, впроваджено у роботу API на основі нейронної мережі.

НЕЙРОННА МЕРЕЖА, ЦИФРОВІ ЗОБРАЖЕННЯ, АГРЕГУВАННЯ НЕРУХОМОСТІ

ABSTRACT

Graduation qualification bachelor's work: 75 p., 65 pic., 1 tab., 1 app., 16 sources.

Topic: Web application for aggregating and grouping real estate based on filters using machine learning.

Object of research: development of a complex application using elements of machine learning.

The purpose of the work: automation of the process of finding apartments

Subject of study: neural network for the classification of industrial facilities

Research results:

Investigated image classification methods

Conclusion

Developed and commercially implemented own neural network model, which in addition to the basic image classification is able to perform a specific task in a complex application. In addition to the actual development of a full-fledged application, work was carried out to collect the dataset and its preliminary processing within the application for further training.

NEURAL NETWORK, DIGITAL IMAGES, AGGREGATING OF REAL ESTATE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ORM — Object Related Mapping

DOM — Document Object Model

Фреймворк — платфора для розробки

HTTP — Hypertext Transfering Protocol

HTML — Hypertext Markup Language

API — Application Programming Interface Інтерфейс програмування додатків

Workflow — цикл роботи додатку

Edit — редагування

Timeline — часова шкала

Attributes — атрибути

RDS — Relational Database System

ВСТУП

На сьогодні на ринку України та інших зарубіжних ринках є досить великий попит на сервіси для пошуку нерухомості на оренду/продаж. Проте серед наявних конкурентів на ринку всерівно не вистачає рішень, які мають достатній рівень автоматизації, для того, щоб кінцевий користувач не відчував проблем з користуванням продуктом. Зазвичай навіть просто неправильне зображення для квартири чи будинку, може відштовхнути потенційного покупця від оренди чи покупки нерухомості, що насамперед може привести до втрати доходу компанії, що негативно скажеться на квартальних покзаниках, а отже на продукті в цілому.

Тому за мету було поставлено розробити бізнес продукт, який буде окрім того, що зручний та простий для користування, достатньо автоматизованим для специфічних задач, для того, щоб виключити в першу чергу помилки внаслідок людського фактору та збільшити швидкість взаємодії з кінцевим користувачем.

Для автоматизації процесів, ми використовуватимемо періодичні задачі та API на основі нейронних мереж, що дасть можливість виконувати комлексні та специфічні задачі у короткі сроки.

Створене API буде займатись класифікацією зображень на специфічні класи “на вулиці”/”всередині”, це потрібно для того, щоб коли ми будемо групувати квартири у будинки, то можна було правильно відібрати зображення для нього (на жаль ми не можемо покладатись на зовнішнє API від Google чи схожі, оскільки вони неточно підходять до вибору зображень, а нам необхідно підбирати зображення на основі тих квартир, які безпосередньо входять в множину будинку)

Актуальність теми

Зв'язок роботи з науковими програмами, планами, темами

Зв”язок є насамперед з новітніми розробками та архітектурами в області розробки нейронних мережей.

Мета і задачі дослідження

Мета полягає у створенні веб додатку та комплексної інтеграції API на базі нейронної мережі на комерційній основі.

Об’єкт дослідження: нейронна мережа з власною архітектурою та моделлю для комерційних потреб

Предмет дослідження – архітектура додатку та нейронної мережі і її модель для виконання специфічної задачі

Методи дослідження: для розробки додатку були використані: метод глибокого навчання з наглядом (Deep supervised learning), методи зважених коефіцієнтів, методи перетворення зображення на матрицю розміром 256x256. Для моделювання обробки проміжних даних сегментації було використано бібліотеку OpenCv. Для імплементації програмного інтерфейсу Python Flask.

Новизна одержаних результатів

Розроблена нова модель обробки та класифікації зображень (завдяки специфіці завдання була потреба у розробці додаткових слоїв архітектури), автоматизовано процес відбору зображень для вибіркової множини зображень промислових об”єктів, нерухомості та процес запису результатів на періодичній основі.

Практичне значення одержаних результатів

Практична цінність отриманих результатів дозволила комерційно впровадити API на основі нейронної мережі у додаток і спростити розробку такого рішення у подальшому.

Апробація результатів випускної кваліфікаційної роботи

Тези доповіді «PROGRAM DEVELOPMENT SOLUTION FOR IMAGE CLASSIFICATION WITH NEURAL NETWORK USAGE» на XI Міжнародній науково-практичній конференції «МАТЕМАТИКА. ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ. ОСВІТА», 3-5 червня 2022 р.

Публікації

Стаття на тему «PROGRAM DEVELOPMENT SOLUTION FOR IMAGE CLASSIFICATION WITH NEURAL NETWORK USAGE» у фаховому збірнику праць «МАТЕМАТИКА. ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ. ОСВІТА».

РОЗДІЛ 1
ОСНОВНІ ЕТАПИ РОЗРОБКИ ДОДАТКУ

Перед практичною розробкою додатку потрібно проаналізувати додаток через призму інженерії програмного забезпечення, а конкретно побудувати схеми та діаграми поведінки, відношення, класів, пакетів тощо, задля більш повного сприйняття практичної частини додатку.

Задля зрозумілості викладення матеріалу у практичній частині, вона буде поділена на 4 частини:

1. Інженерний підхід до розробки;
2. Зберігання даних у додатку;
3. Серверна частина;
4. Клієнтська частина.

Кожна з яких буде відповідати за конкретну частину розробки додатку.

1.1. Інженерний підхід до розробки

Починаючи огляд додатку варто зробити першочерговиий фокус на діаграму пакетів, оскільки завдяки їй можна зрозуміти структуру проекту загалом.

Таким чином зі схеми, який надано на Рис.1.1 робимо висновок, що наша архітектура буде мати клієнт-серверну основу + додатково інтегроване API у сервер, для додатку, яке визначає правильність зображень.

Далі, переходячи до більш детального (прикладного) зображення ПЗ, буде використана діаграма компонентів.

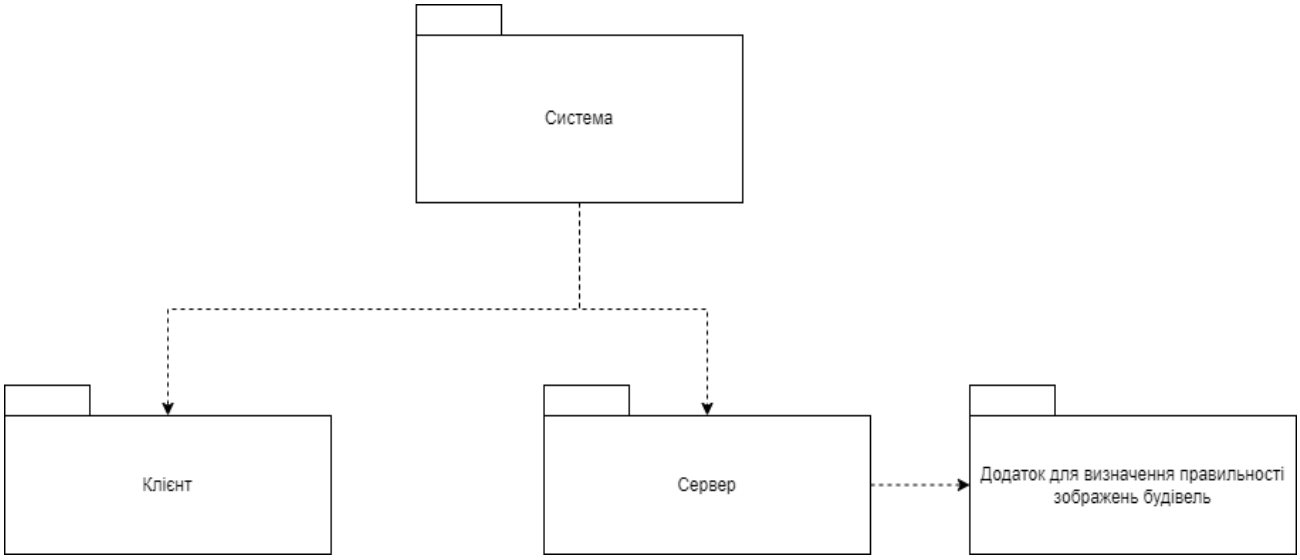


Рис. 1.1. Діаграма пакетів

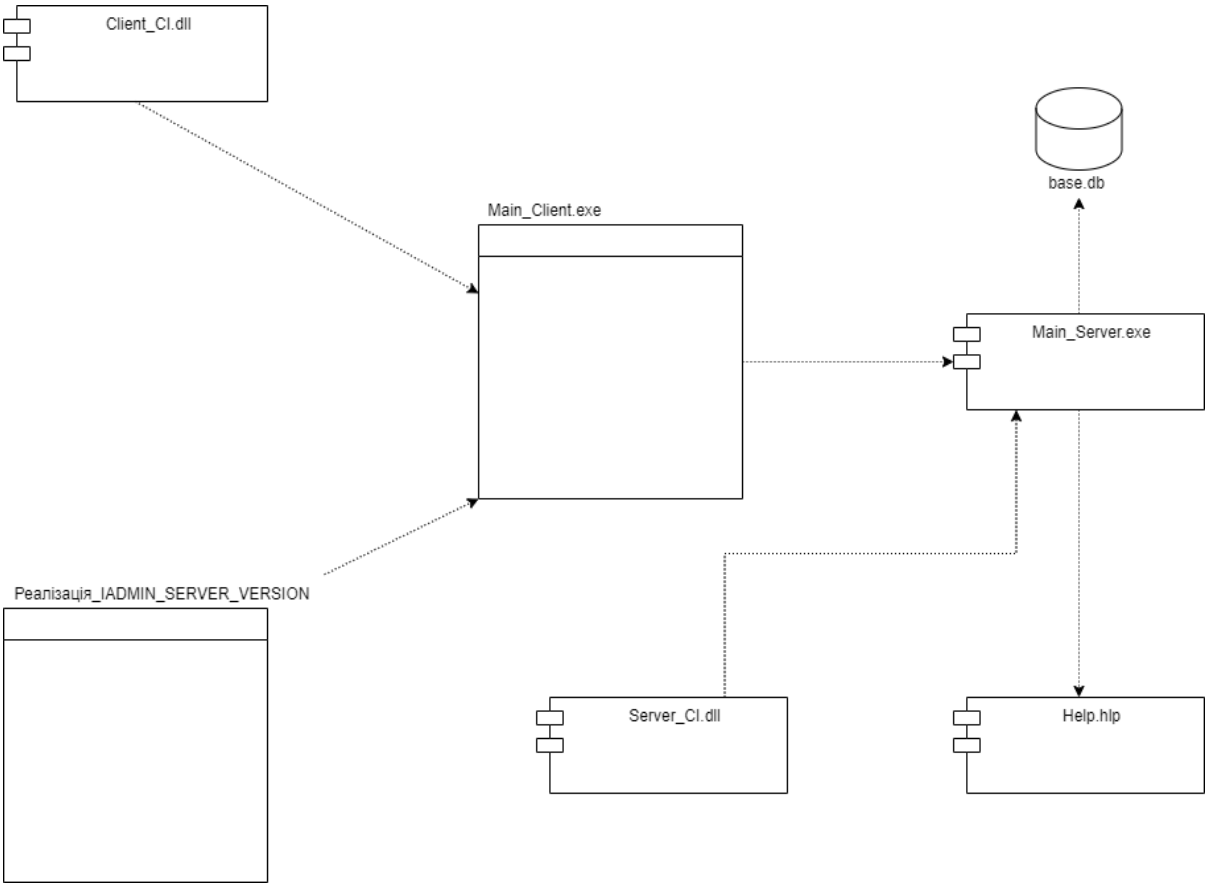


Рис. 1.2. Діаграма компонентів

Таким чином, у нас є основна оболонка для клієнту (скомпонована у вигляді контейнеру) - Main_Client.exe і до неї мають під’єднання серверна частина додатку (Main_Server.exe) і також реалізації інтерфейсів для клієнтського

керування додатком (Client_CI.dll) та інтерфейсу IADMIN_SERVER_VERSION - який дає змогу керувати інтегрованим API для фільтрування зображень. Також до серверної частини додатку під’єднання база даних base.db - для зберігання даних в додатку

Після діаграми компонентів буде доречно розглянути діаграму класів, щоб сформувати ідею взаємодії сутностей у додатку.

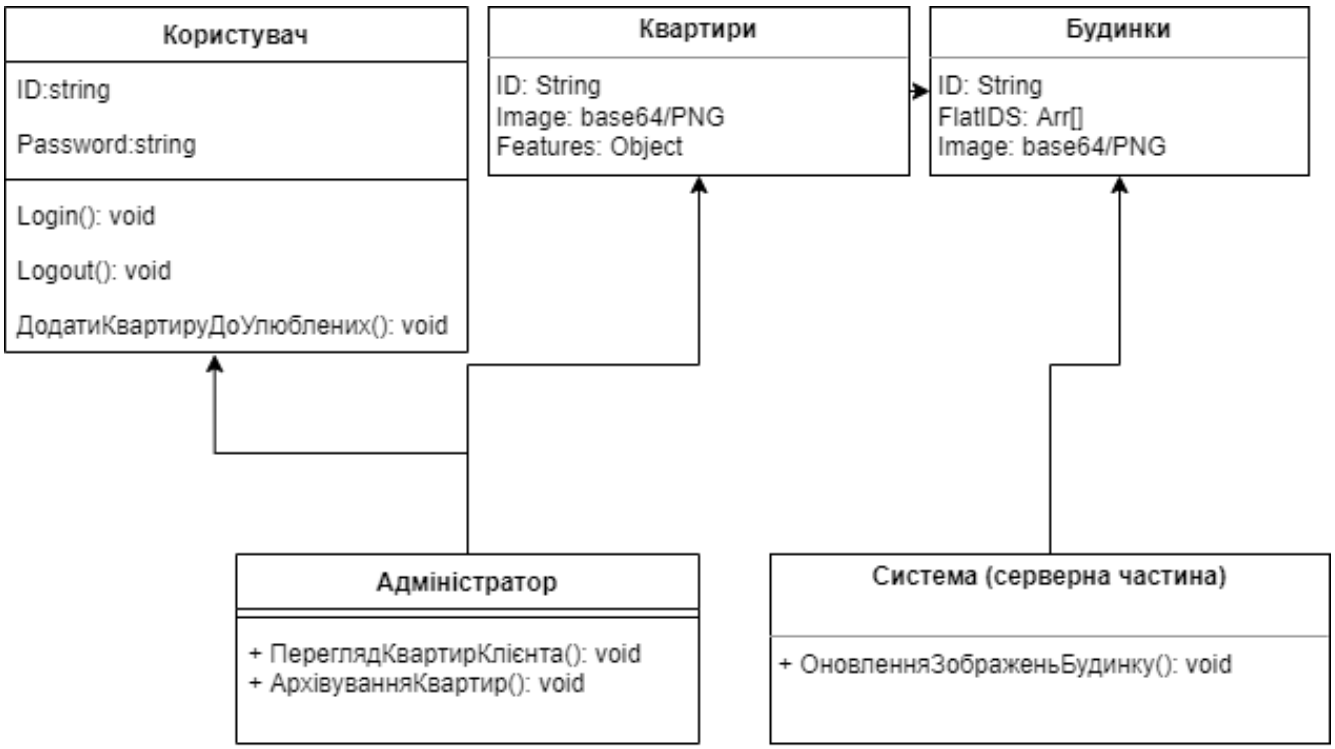


Рис. 1.3. Діаграма класів

Основними сутностями у нас є класи “Користувач” та класи “Адміністратор”, де користувач, має такі дії як логін-вихід з додатку та додавання квартир у список улюблених, для спрощенного перегляду цікавих для користувача квартир у подальшому часі.У той час як адміністратор має змогу переглянути, які квартири переглядав клієнт, та архіувати кварити - для того, щоб не показувати користувачу уже квартири, які здали наприклад. Також для квартир є свій клас з полями ID, Image, Features, які дозволяють в повній змозі описати квартиру для зображення її на клієнтській частині

Варто зазначити, що квартири об”єднуються у сутність Будинки, яка теж має поля ID, FlatIDS, Image. У в FlatIDS входять ідентифікатори квартир, які належать цьому будинку (по адресі та геолокації).

Важливою частиною є клас серверної частини, який завдяки адміністраторам або власноруч через компонувальник задач має можлиівсть регулярно оновлювати інформацію про будинок (завдяки інтрегованому API у серверну частину).

Далі буде корисно дізнатись, яким чином ми будемо розгортати та обслуговувати наш додаток і для цього нам знадобиться діаграма розгортання.

Для бази даних була обрана платформа PostgresSQL.

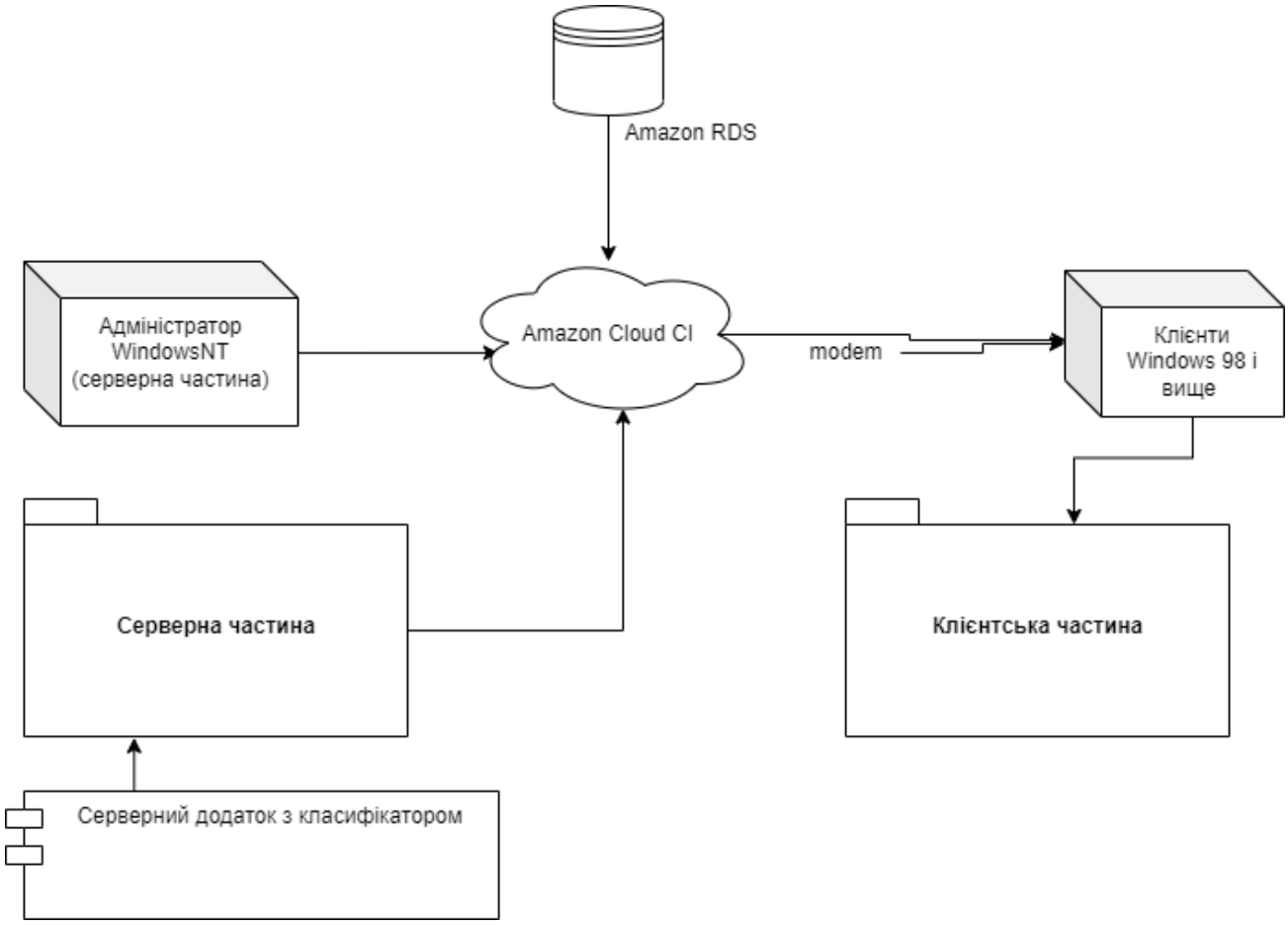


Рис. 1.4. Діаграма розгортання

На діаграмі ми можемо побачити, що для розгортання було обрано хмарну архітектуру, а точніше Amazon Cloud, оскільки завдяки ній у нас немає потреби у фізичних носіях для розгортання серверу і таким чином додаток не залежить від фізичних умов довкілля і має високу стабільність. Також завдяки сервісу Amazon ми можемо налаштувати процес розробки таким чином, що запуск нових версій додатку буде синхронізований з репозиторієм і це зекономить час на розробку, оскільки також не буде необхідності для ручного розгортання додатку.

У нас є серверна і клієнтська частина яка розгорається на цьому сервісі, у серверну частину входить - наше інтегроване API (також розгортається на цьому сервісі, але з обмеженнями - про це далі), до клієнтської частини мають доступ користувачі, в той час як до серверної разом з клієнтською частиною - адміністратори.

Розгортання API з нейронною мережею це достатньо специфічна задача, оскільки вимірювання ціни за користування має інші параметри - зазвичай сервіси орієнтуються на CPU seconds, власне час, який витрачає нейронна мережа для обчислень.

Для бази даних використовується сервіс Amazon RDS, який дає можливість розгортати базу даних завдяки хмарним рішенням, що також є дуже зручно, оскльіки плата за сервіс росте пропорційно до розміру бази даних.

Отже після загального огляду системи варто буде перейти безспосередньо до діаграми прецедентів.

Отже, короткий опис діаграми прецедентів - користувач з полями (ID, password) має можлиівсть проводити додавання власної квартири на продаж, перегляду квартир, перегляду будінків. Система перевіряє правльність заповнення інформації та видає підтвердження або відмову. В той час як адміністратор має можливість супроводжувати систему, через перегляд логів і також має можливість архівування “старих” квартир - ті, які випадково були

пропущенні системою і вже не валідними (продані тощо). Суб’єктами є - користувач та адміністратор.

Передумовою для використання системи є авторизація користувача для входу в БД.

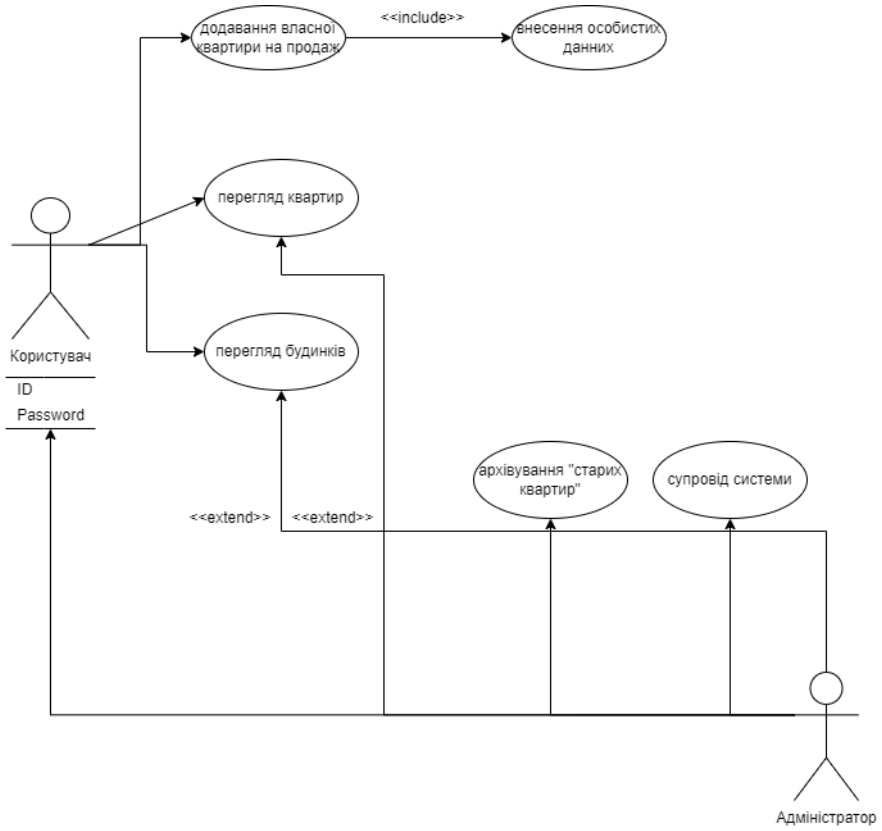


Рис. 1.5 Діаграма прецедентів

Основний потік:

1) Додавання власної квартири на продаж: система відбражує форму введення в бд. Форма вміщує наступне:

1.1) Заголовок для назви будинку;

1.2) Зображення будинку;

1.3) Деталі (надалі поле features - див. Діаграму класів);

2) Якщо всі деталі введені правильно, то після натискання кнопки збережння, дані відправляються на сервер та заносяться до БД.

2.1) Якщо форма з неповною інформацією виконсується А1.

2.1) Якщо сервер не в змозі внести дані до БД, виконується А2.

3) Архівування квартир адміністратором: якщо при перегляді адміністатор натрапляє на “стару” квартиру, то він має змогу заархівувати квартиру і видалити її з результатів пошуку для других користувачів

3.1) Якщо після натискання кнопки “Заархівувати” сервер не змозі оновити статус квартири. Виконується А2.

Альтернативні потоки:

А1) Неповна інформація. Якщо користувач не заповнив усі необхідні поля, система дає можливість їх ввести повторно. Процедент продовжується.

А2) Помилка серверу. Якщо сервер на заносить нову квартиру до БД, то система видає повідомлення “Помилка при зберіганні!” та надає можливість внести дані ще раз.

Постумови:

Якщо внесення, архівування квартир пройшло вдало, то дані записуються до БД або видаються на екран. У проитлежному випадку, стан системи - стан її даних залишаються незмінними.

Після огляду діаграми прецедентів варто перейти до діаграм станів, послідовності та діяльності.

Отже, на першій діаграмі діяльності (для користувача) у початковому стані користувач перебуває на стадії перегляду квартир, далі після того, як він підбирає найбільш підходящу із списку, то наступною дією є ибір та визначення дати перегляду і далі користувач успішно вносить дані до БД про дату перегляду. Якщо усі дані вірні, то користувача регіструють на вказану дату і він має змогу переглянути квартиру (кінцевий стан).

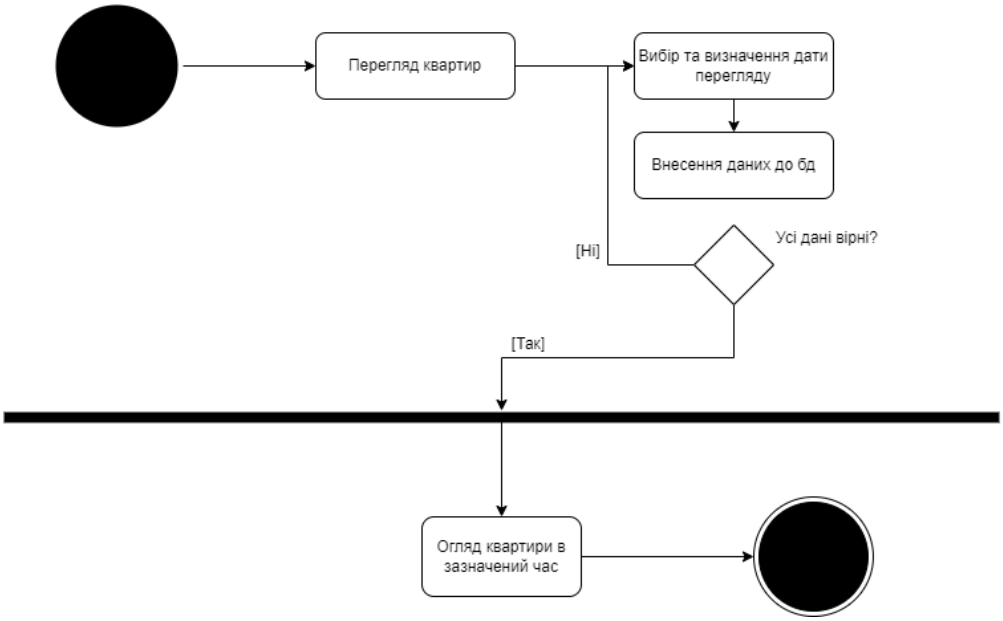


Рис. 1.6. Діаграма діяльності (А)

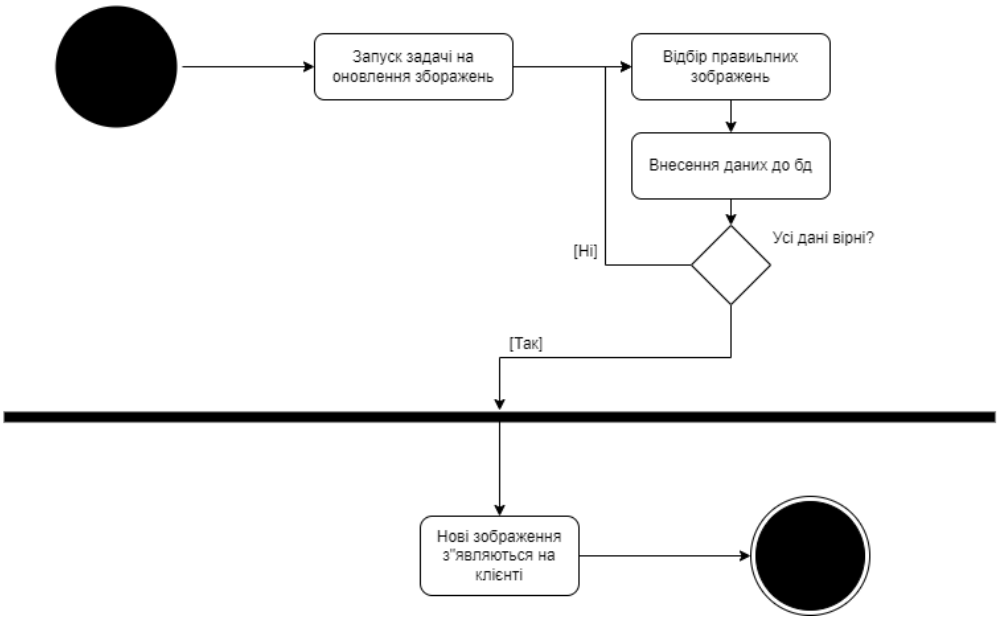


Рис. 1.7. Діаграма діяльності (Б)

Переходячи до другої діаграми діяльності (для серврної частини додатку, яка через компоновальник задач автоматично із заданою періодичністю може обирати коли оновлювати зображення для будинків), початковим станом, є запуск задачі на оновлення зображень (що може зробити і адміністратор за потреби), наступною дією йде відбір правильних зображень і далі оновлення бд, якщо все

пройшло вдало, то кінцевим станом є - поява нових зображень на клієнтській частині додатку.

Після діаграми діяльності для більшої наглядності буде показана діаграма послідовності.

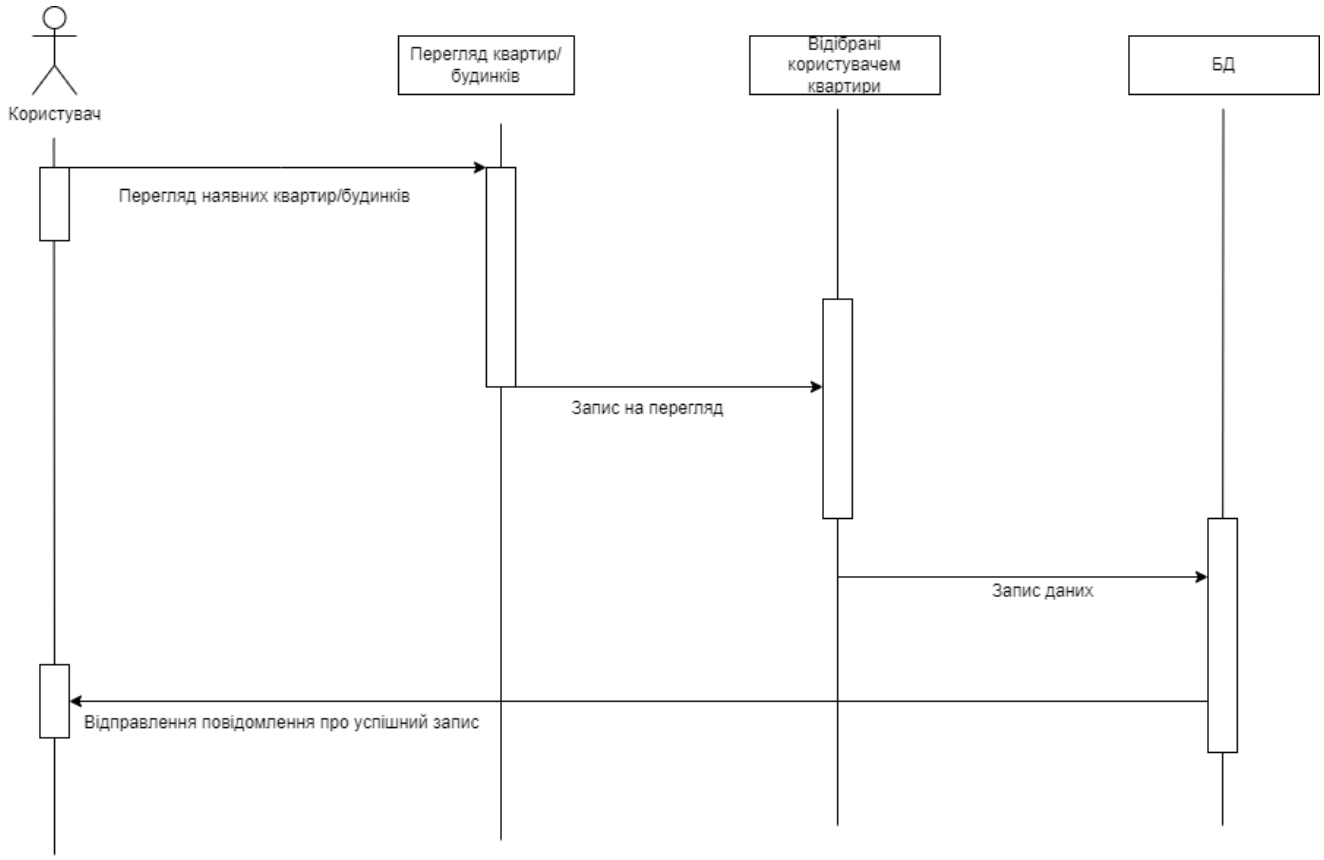


Рис. 1.8 Діаграма послідовності

Отже, на даному прикладі діаграми послідовності, ми будемо розглядати процес запису користувача на перегляд квартири, як можемо побачити усі процеси є синхронними, а отже йдуть одна за одною (для зручності часові рамки не використовувались, оскільки системою можуть користуватись користувачі з різними девайсами і майже все залежить від якості з’єднання). Отже, користувачу для початку потрібно із списку квартир обрати найбільш підходящу, далі запистаись на перегляд. Після запису на перегляд в базу даних через серверну

частину йде відправлення повідомлення про успішний запис із датою на перегляд - на електронну пошту користувача.

Після діаграми послідовності в завершення буде доцільно розібрати діаграму станів для основних дій.

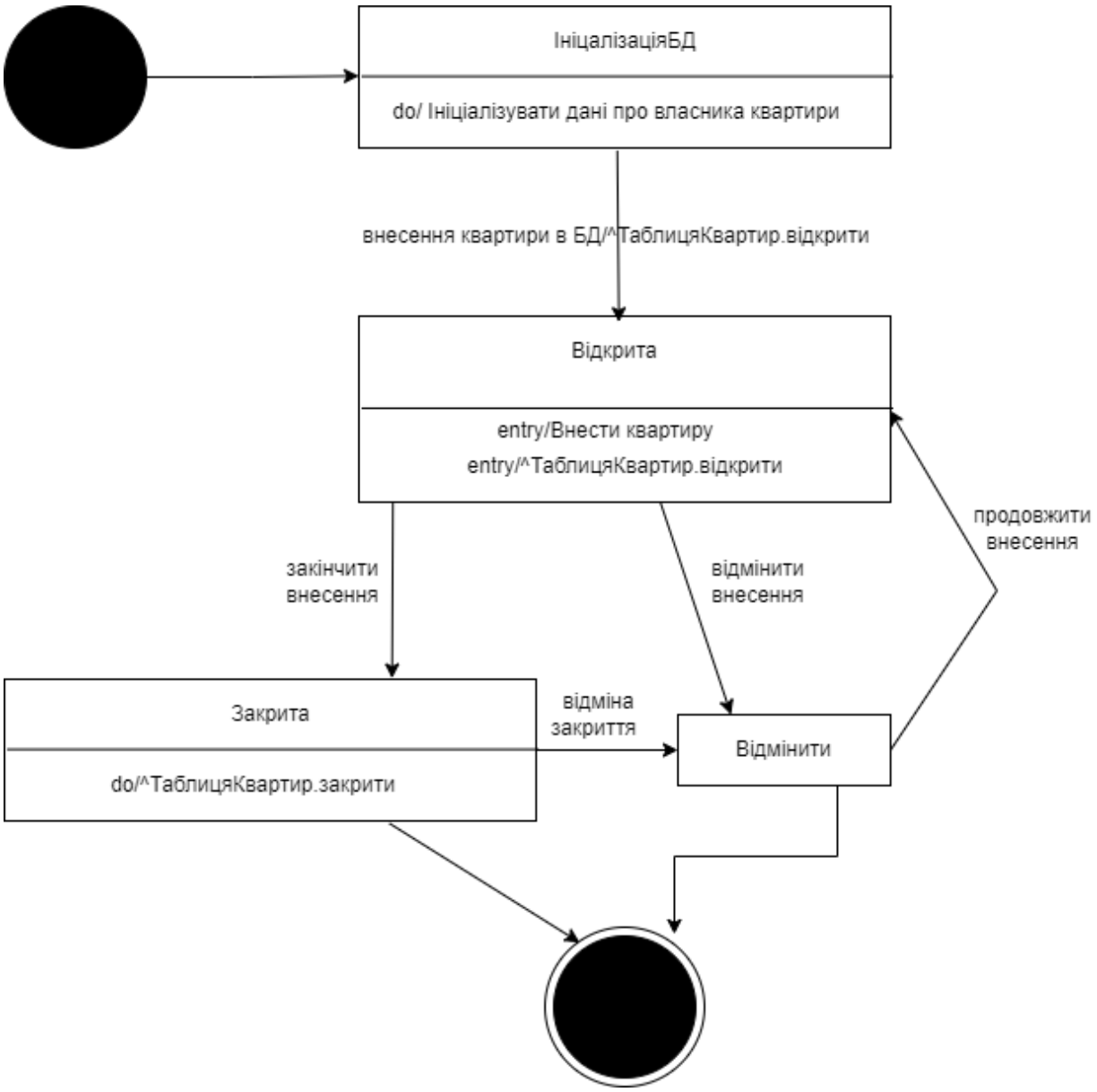


Рис. 1.9 Діаграма станів (А)

Дана діаграма станів описує процес занесення користувачем своєї квартири на продаж у додаток, для того, щоб вона з”являлась у списку актуальних квартир.

Для початку, власнику потрібно авторизуватись, оскільки кожна квартира підв’язана до користувача. Далі, якщо власник успішно заповнив усі форми щодо квартири, то до таблиці Квартири, вноситься запис, якщо це пройшло невдало - то користувач знову зможе спробувати. Якщо запис пройшов успішно - то створюється запис в табличці і квартира з’являється у списку квартир на продаж. З’єднання зачиняється, щоб не забирати лишні потужності у сервера.

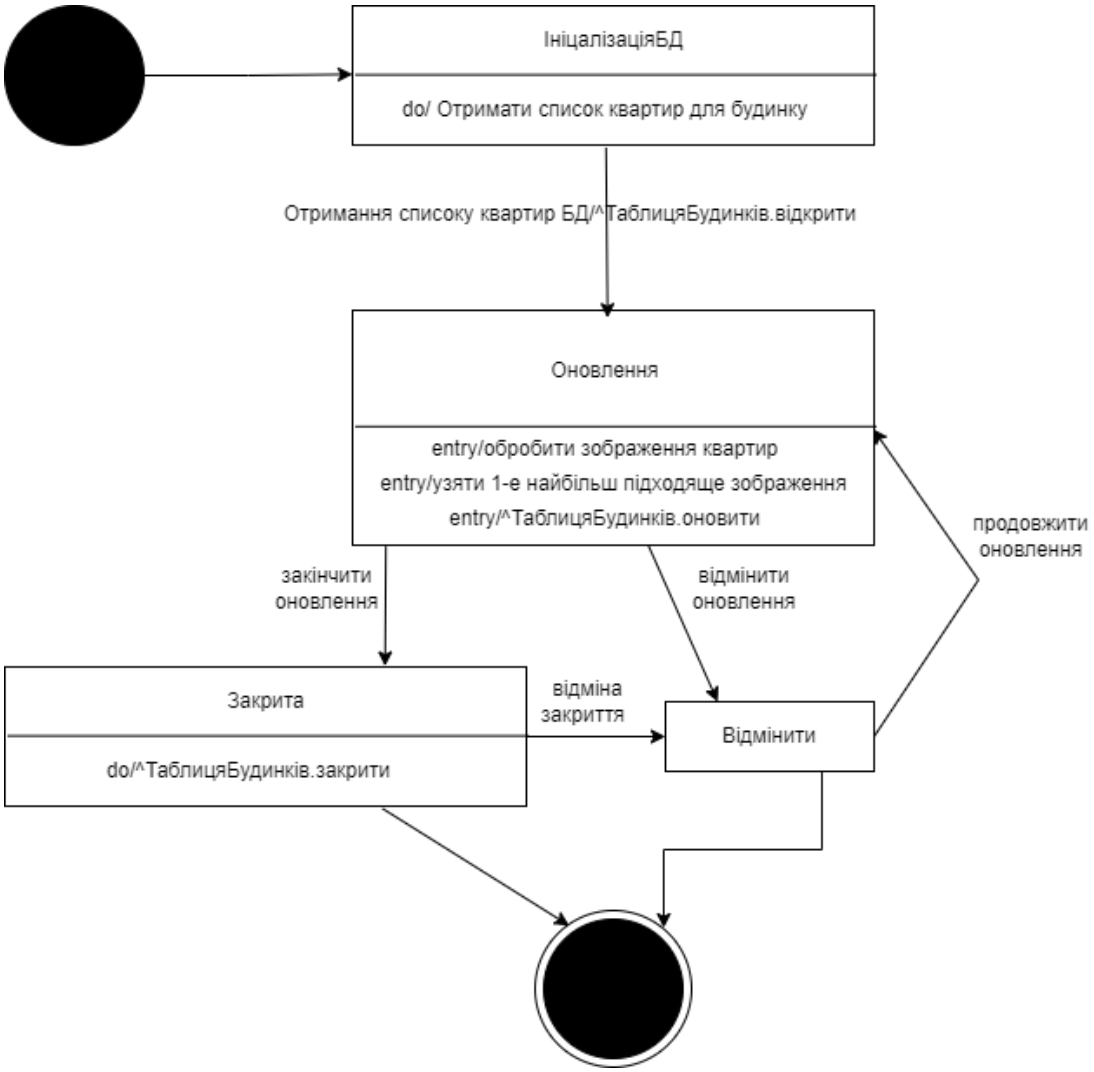


Рис.1.10. Діаграма станів (Б)

Отже, діаграма станів Б - зображає процес оновлення зображення будинку інтегрованим API, який запускається адміністратором або автоматично через компонувальник задач зі сторони серверу із заданою періодичністю. Отже для

початку, потрібно отримати усі квартири для вказаного будинку через список ідентифікаторів. Після цього йде оброблення зображень квартир і після обчислень та виявлення найбільш підходящої квартири (детальніше див. Теоритчна частина) можна оновлювати будинок. Якщо оновлення прошйло успішно - то з’єднання з БД закривається, якщо ні - то далі йде спроба оновити зображення.

1.2. Зберігання даних в додатку

На сьогодні існує два актуальних рішення: реляційні бази даних та NoSQL рішення - які мають динамічну структуру таблиць і зазвичай мають структуру геш таблиць (ключ-значення).

У нашому випадку для традиційної клієнт-серверної архітектури найкращим рішенням буде опиратись пласне на реляційні БД, оскільки:

1) Власне сама мова зберігання SQL - Structured Query Language, більш структрована і чітко огрганізована, основний набір команд який, довзоляє створювати, обновляти, зчитувати, видяляти дані. Важливо зазначити, що їх набір і суть змінюється від мови до мови мінімально або взагалі ніяк. Приклад команди INSERT на діалекті PostgresSQL: INSERT INTO table_name (column1, column2, column3, ...)VALUES (value1, value2, value3, ...); Приклад команди INSERT на діалекті MSSQLINSERT INTO table_name (column1, column2, column3, ...) VALUES (value1, value2, value3, …);

В той час як NoSQL рішення зазвичай не мають чітко визначеної мови запитів, що приводить до додадктових витрат людино-годин при інтеграції нових членівкоманди у проект

2) Простота у використанні, оскільки розробник завжди чітко знає який тип у тої чи іншої колонки у таблиці, тому дуже рідко виникають проблеми у тестуванні;

3) Розширюємість - для того, щоб додати новий об”єкт зі своїми характеристиками буде достатньо визначити лише табличку з колонками і типами даних, проте є і друга сторона, потрібно грамотно визначитись з типами даних, оскільки деякі з них з самого початку забирають більше пам”яті чим інші, більш прості аналоги, наприклад тип VARCHAR (який мже займати від 1-го байту) і TEXT (який може тримати до 65 535 байтів, і при цьому з самого початку потребує не 1, а 2 байти). Таким чином було прийнято рішення вибрати із популярних рішень - PostgresSQL, тому що:

- Платформа є безкоштовоною;
- Краще розширюємою (особливо для складних і високонагружених систем);
- Відкритий вихідний код;
- Безпека - усі транзакції (включаючи наприклад оплата товарів) виконуються по одній за раз, отже не буде непередбачуваних ситуацій, коли один запит виконається раніше чим потрібно, що може привести до неправильних обчислень, а отже втрати коштів.

341 systems in ranking, December 2018

Rank			DBMS	Database Model	Score		
Dec 2018	Nov 2018	Dec 2017			Dec 2018	Nov 2018	Dec 2017
1.	1.	1.	Oracle	Relational DBMS	1283.22	-17.89	-58.32
2.	2.	2.	MySQL	Relational DBMS	1161.25	+1.36	-156.82
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1040.34	-11.21	-132.14
4.	4.	4.	PostgreSQL	Relational DBMS	460.64	+20.39	+75.21
5.	5.	5.	MongoDB	Document store	378.62	+9.14	+47.85
6.	6.	6.	IBM Db2	Relational DBMS	180.75	+0.87	-8.83
7.	7.	↑ 8.	Redis	Key-value store	146.83	+2.66	+23.59
8.	8.	↑ 10.	Elasticsearch	Search engine	144.70	+1.24	+24.92
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	139.51	+1.08	+13.63
10.	10.	↑ 11.	SQLite	Relational DBMS	123.02	+0.31	+7.82

Рис. 1.11. Порівняння актуальності баз даних

Побудова табличок та схем зв’язків між ними

Отже після того, як ми обрали найбільш вдале рішення для зберігання даних у додатку, потрібно визначитсь, які об’єкти буду представляти дані для збереження - такими об’єктами будуть таблички в базі даних.

Для початку варто створити базу даних, цей процес відбувається через простий інтерфейс. Далі потрібно вказати назву бази даних, host - для локального користування “localhost”, ім’я користувача - username та порт (можна зайняти будь-який, який не використовується системою - у нашому випадку 3306 порт підійде якнкайркаще) .

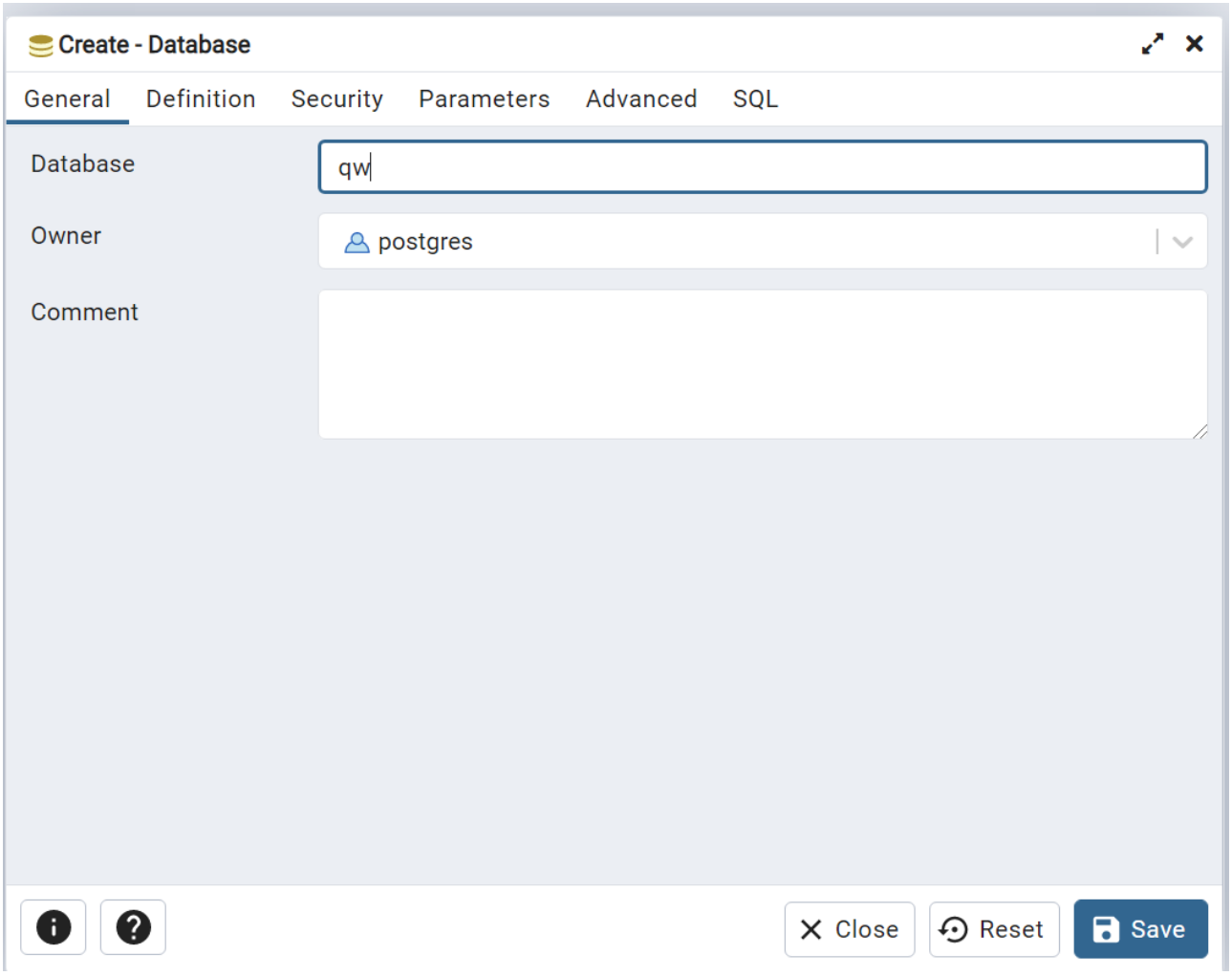


Рис. 1.12 Процес створення бази даних

Після створення бази даних, вона з’явиться у списку доступних баз.

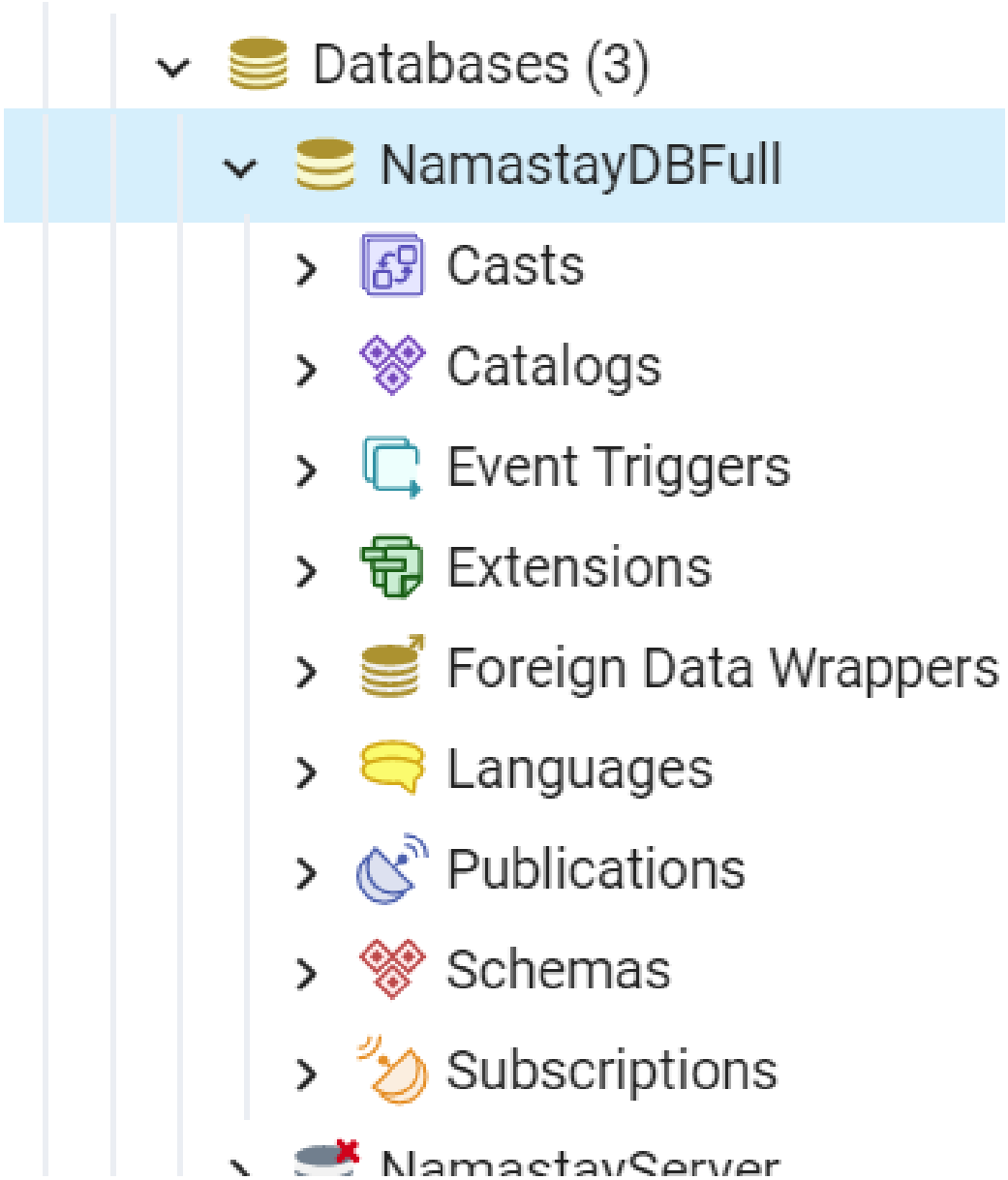


Рис. 1.13 Результат створення баз даних

Далі перейдемо до створення табличок баз даних і побудові зв’язків між ними (список буде посортований по порядку важливості тої чи іншої таблички у порядку спадання):

1) Users - дана табличка містить у собі користувачів чистеми. Містить у собі базові поля ідентифікатор, ім’я, електрону пошту, пароль від системи, тощо.

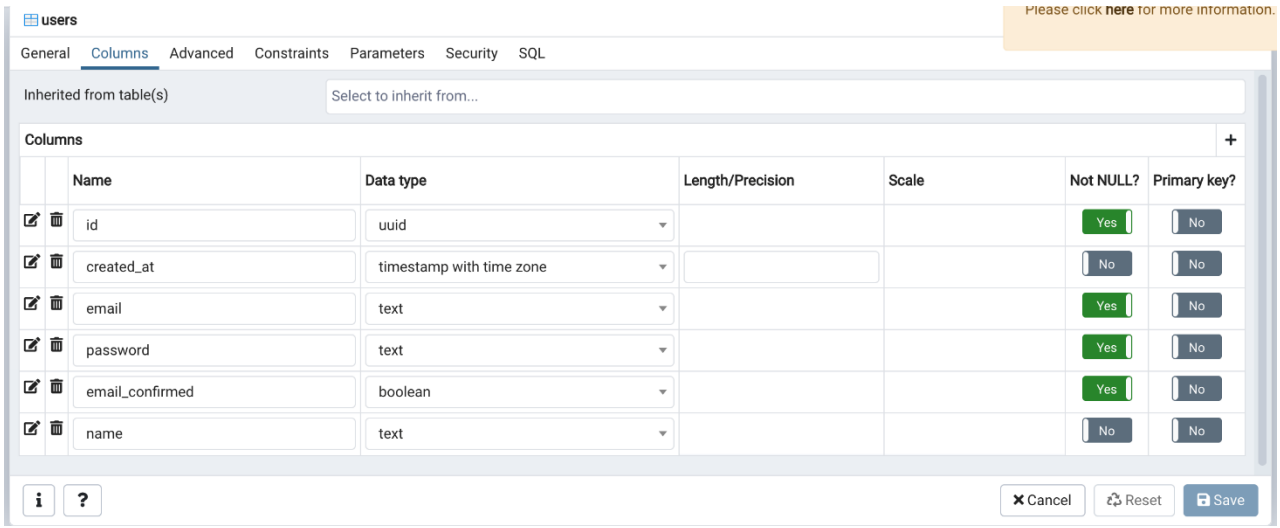


Рис. 1.14 Визначення таблички Users

2) Properties - дана табличка містить у собі інформацію, щодо квартир у системі. Містить у собі множину полів, ідентифікатор, адреси, ціну, рік продажу, чи потрібен логін у системі для перегляду (деякі квартири можуть бути приватними), та інші поля, які ми будемо зображати у вкладці “Amenitites” (зручності)

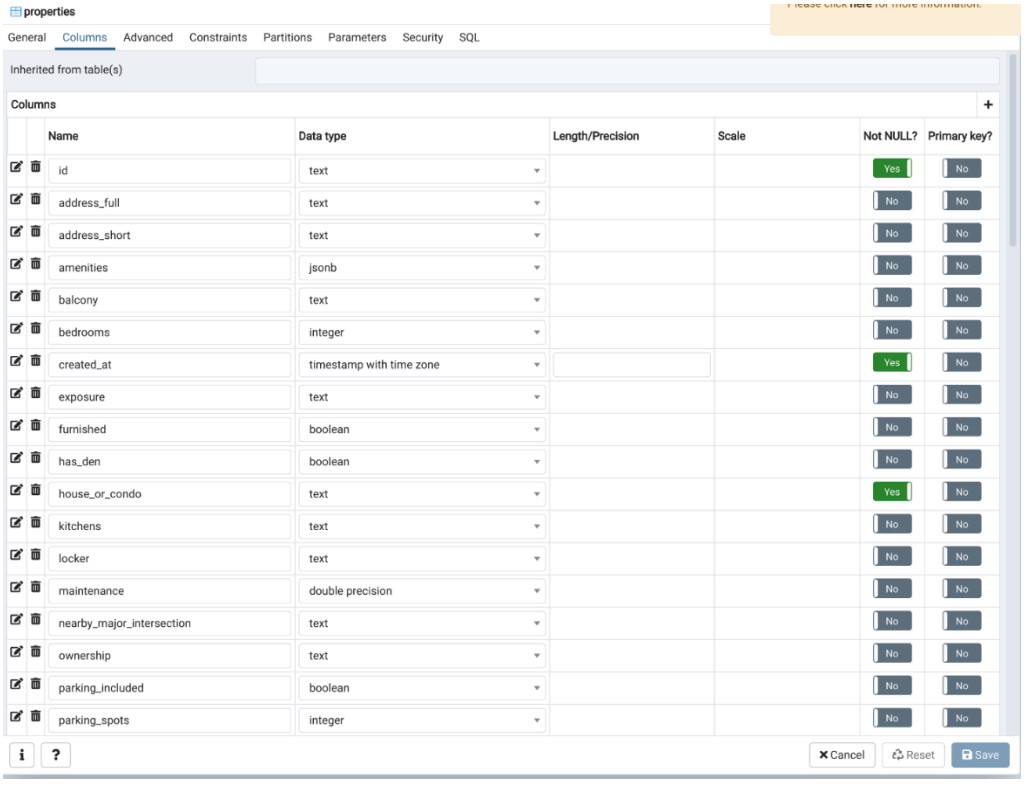


Рис. 1.15 Визначення таблички Properties

3) Favourites - дана табличка містить у собі інформацію, щодо квартир і користувачів, які додали собі їх у список, які їх цікавлять, для того щоб була можливість їх перегянути надалі, а не шукати заново, містить у собі дати, коли створені записи, та посилання на таблиці користувачів та таблиці квартир через зовнішній ключ.

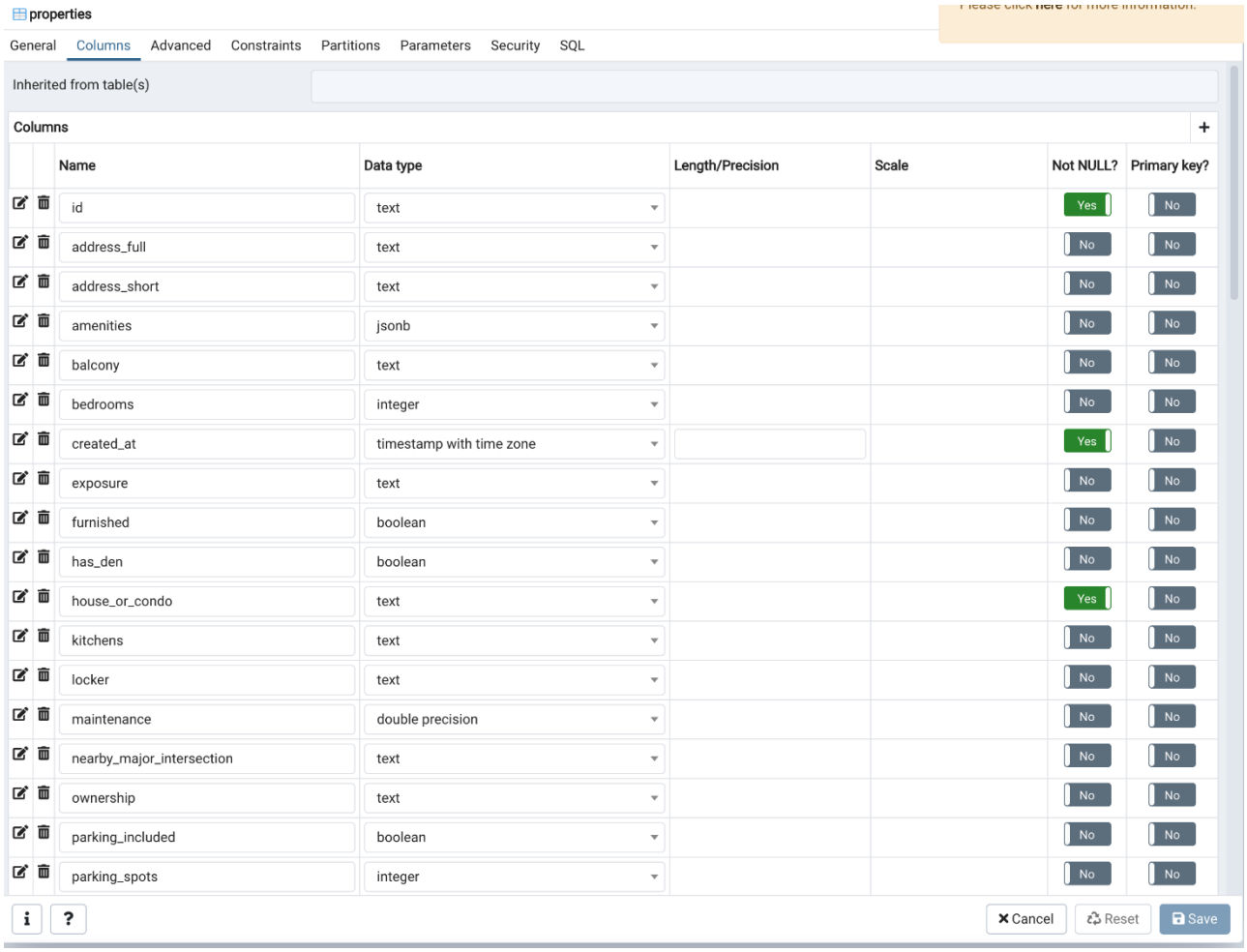


Рис. 1.16 Визначення таблички Favourites

4) Neighborhoods - містить у собі інформацію, щодо районів усередині міст, має поля для географічних координат для того, щоб показувати їх рамки на карті, їх назву, та дату створення;

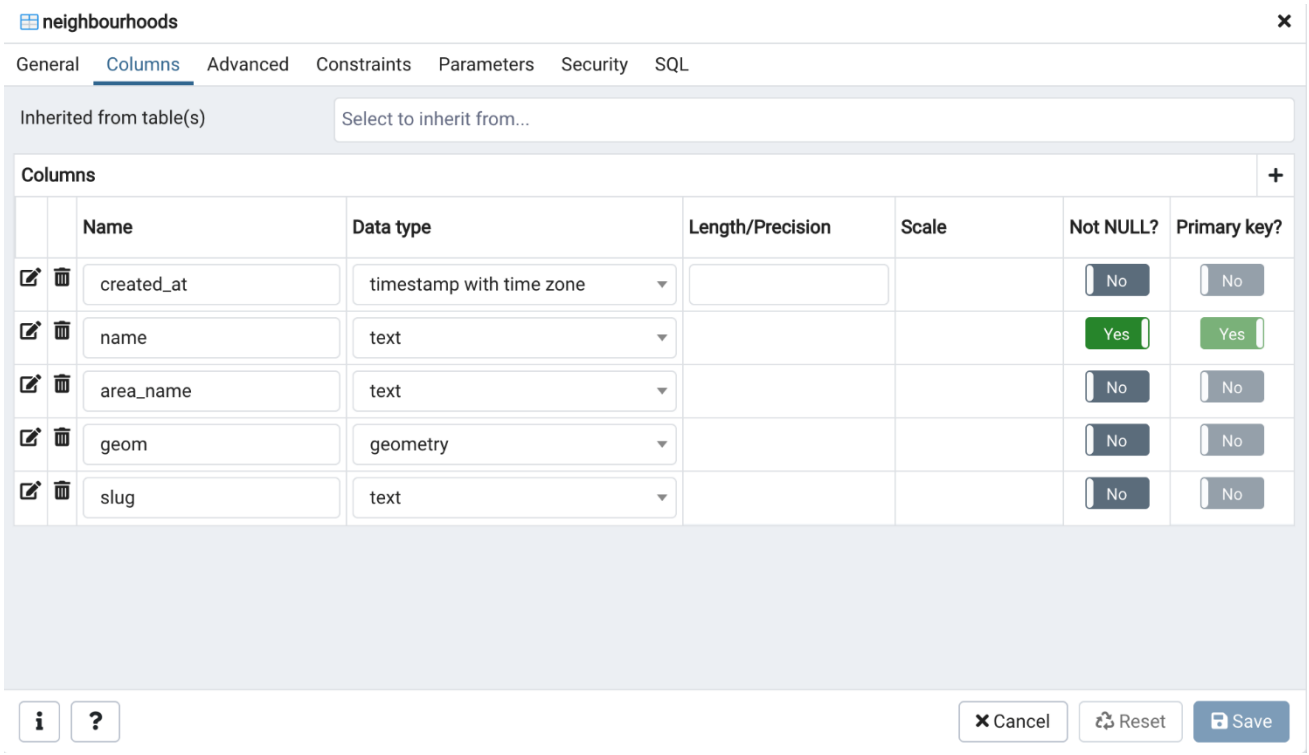


Рис 1.17 Визначення таблички Neighborhoods

5) Municipalities - містить у собі інформацію про власне міста, має схожі поля як і у Neighborhoods;

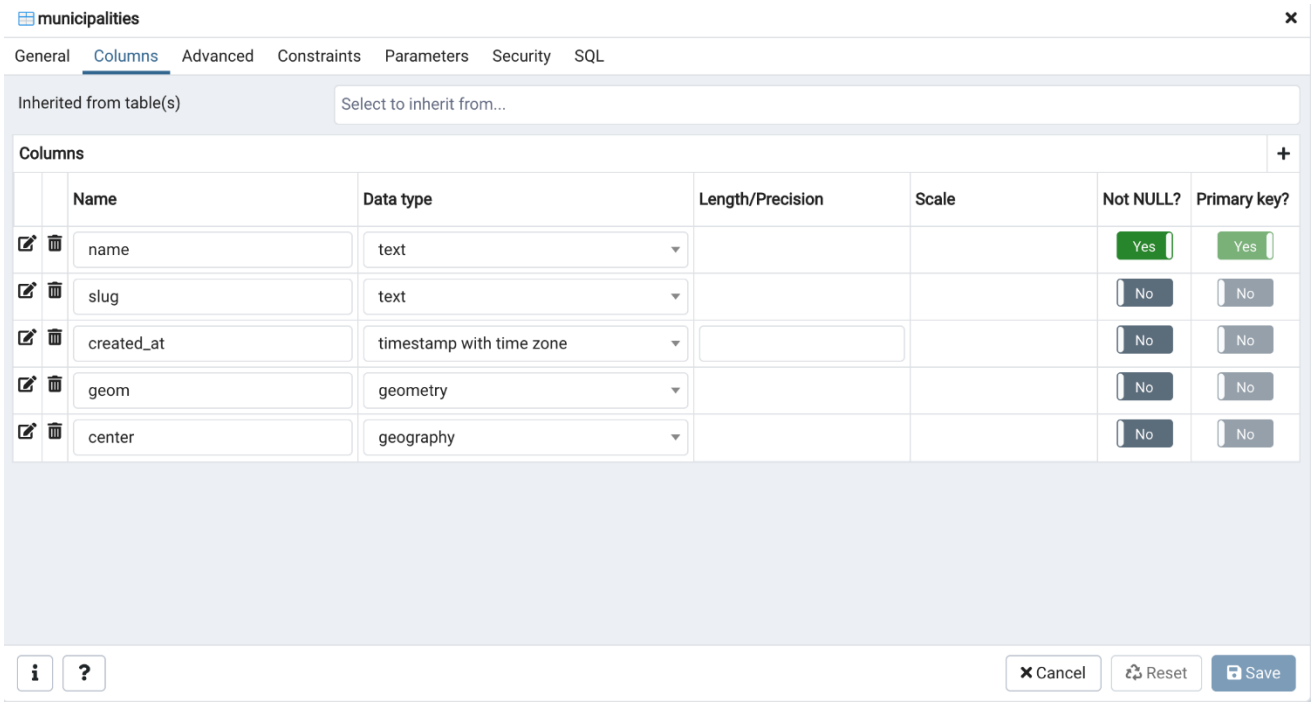


Рис. 1.18 Визначення таблички Municipalities

6) Buildings - містить у собі інформцію щодо будинків, квартири відносяться до будинку через географічні координати (а не через ідентифікатори), що дає можливість більш точно групувати квартири у будинки і відповідно відображати їх на карті. Має достатньо багато полів, такий як кількість гаражів, площу, рік побудови, зображення, тощо;

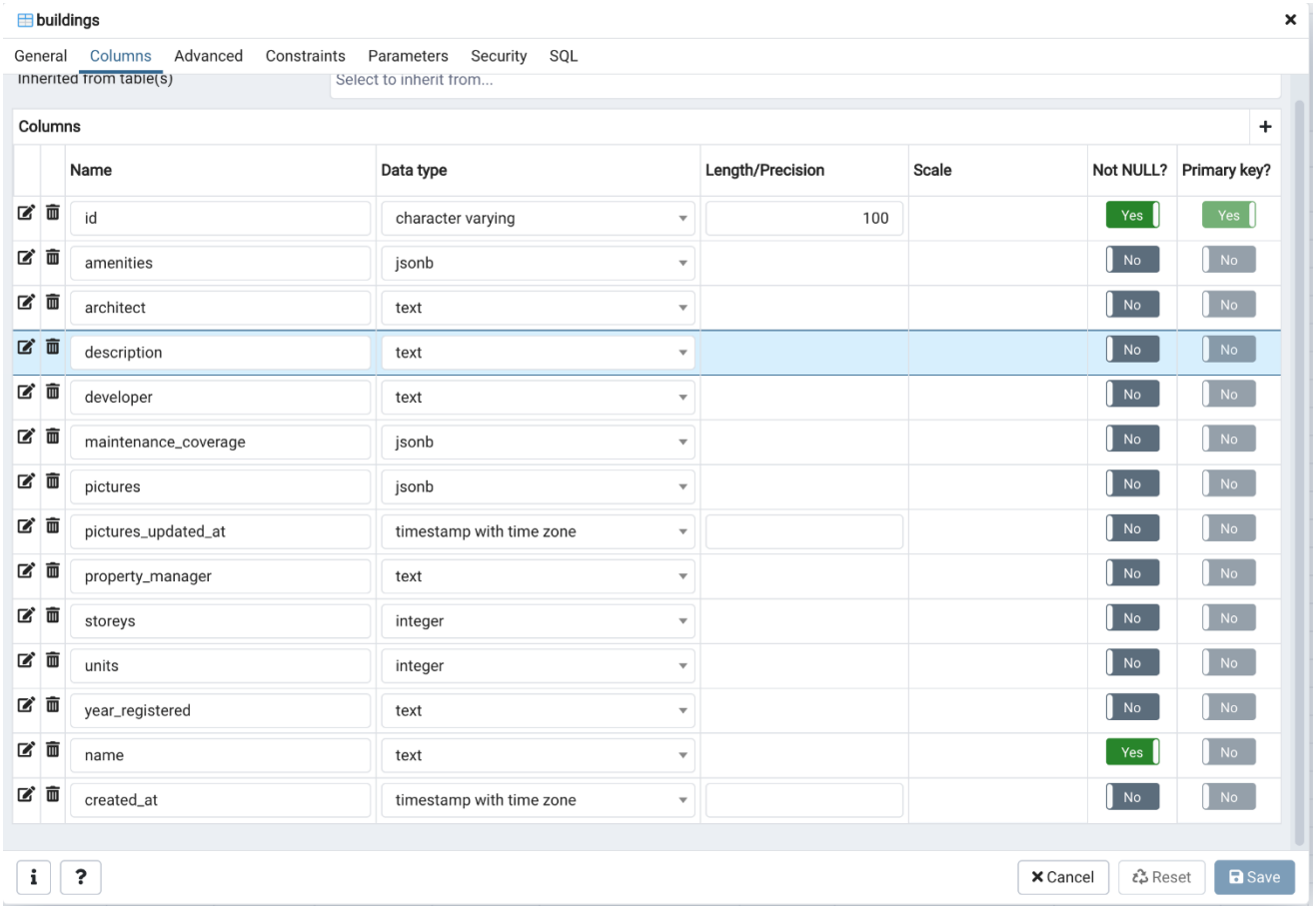


Рис. 1.19 Визначення таблички Buildings

7) Building_addresses - містить у собі географічну інформацію щодо будинків, для їх правильного зображення на карті;

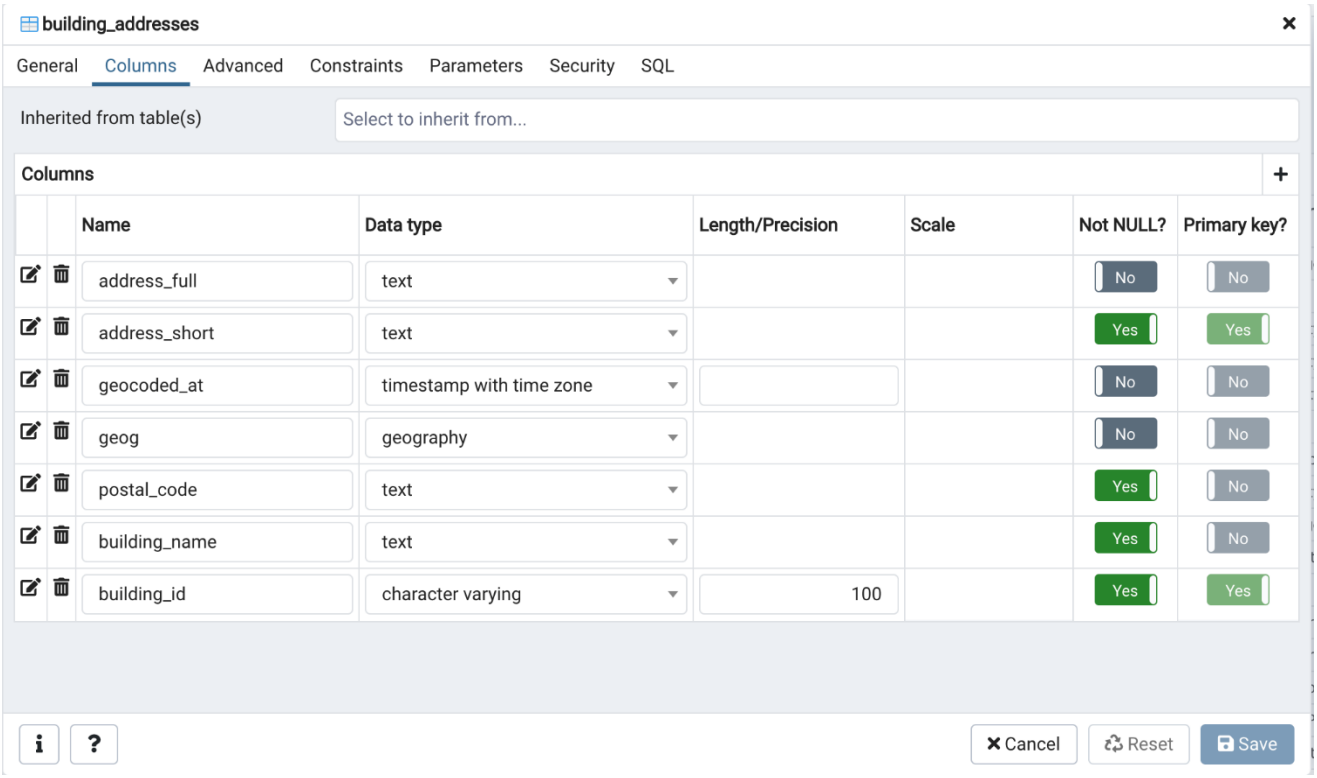


Рис. 1.20 Визначення таблички Building_addresses

8) archived_properties - містить у собі заархівовані квартири, дана табличка потрібна для того, щоб можна було показати “історію” користувачу, у тому випадку, якщо квартира була перепродана, тощо. Містить у собі схожі поля з табличкою Properties, проте також містить і статус квартири (наприклад “Продана”, “Частично продана” тощо)

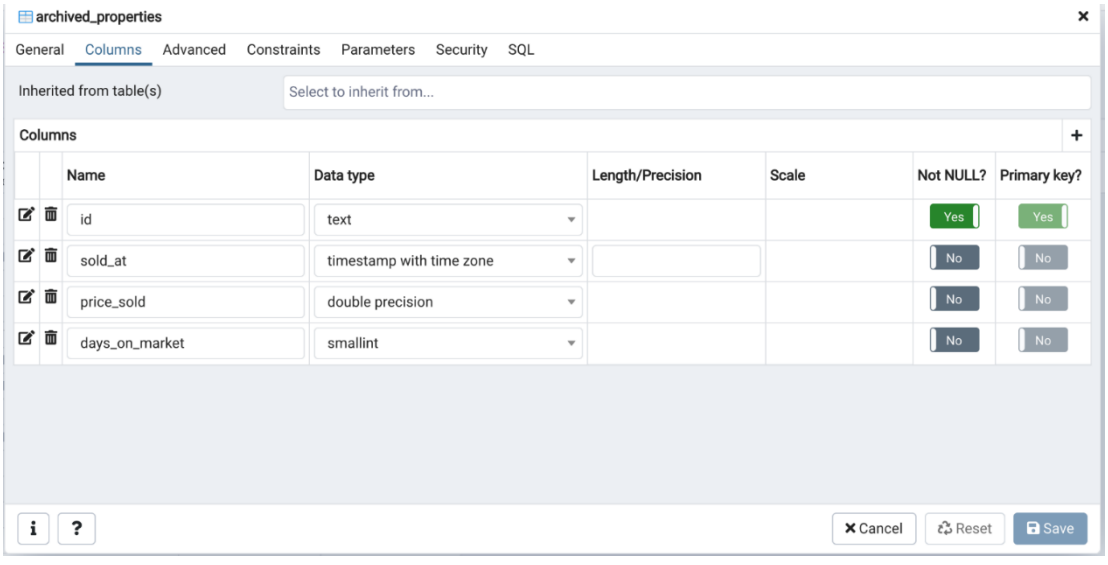
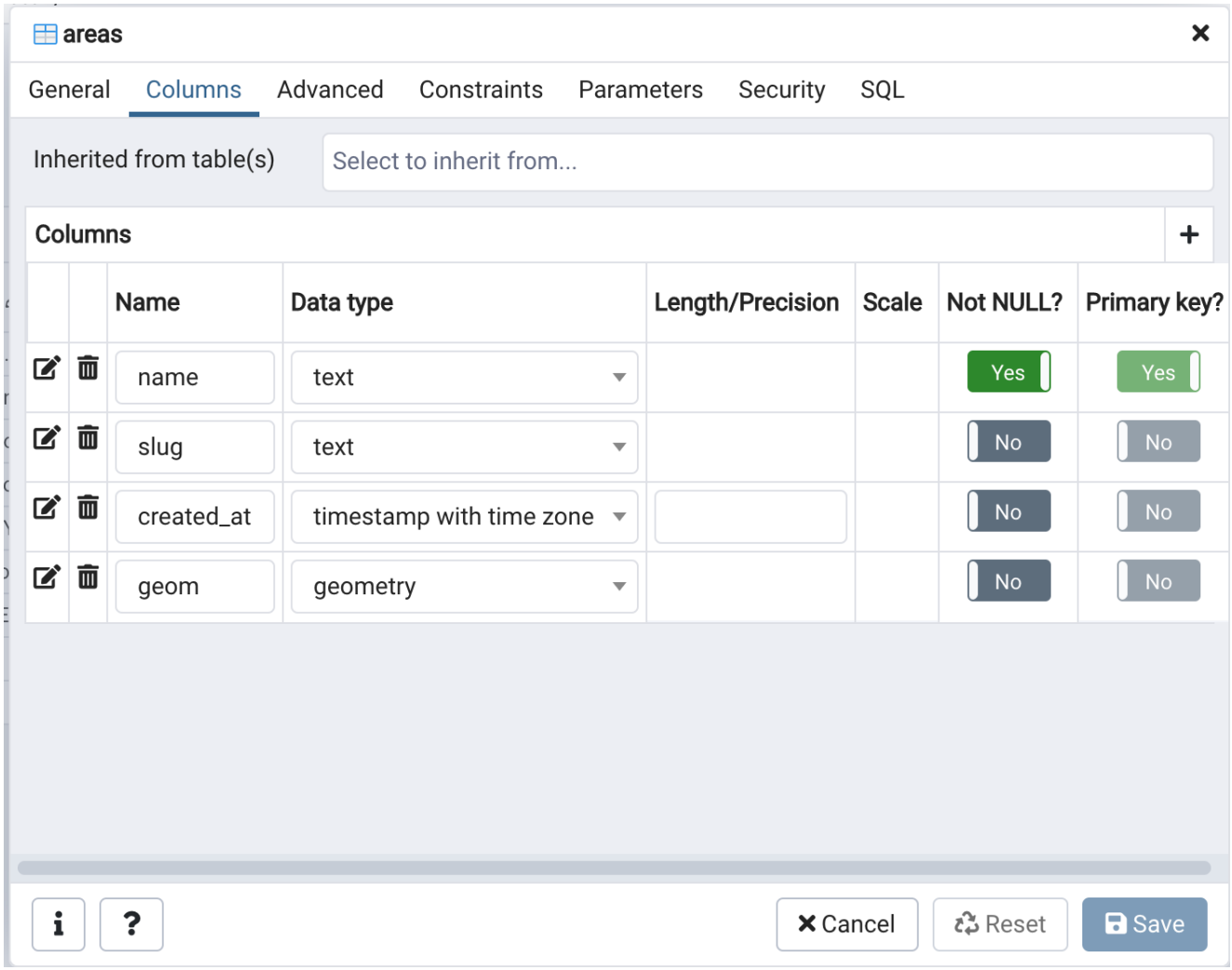


Рис. 1.21 Визначення таблички archived_properties

9) areas - містить у собі інформацію, щодо областей (міста входять в область), також місттять географічні координати, назви тощо



10) offers - містить у собі інформацію, щодо спеціальних пропозицій, які можуть бути цікавими користувачу (наприклад відсоток знижки на якісь святкові дні тощо), містить у собі поля для списку квартир, на які поширюються акції, кінцеві дати, тощо

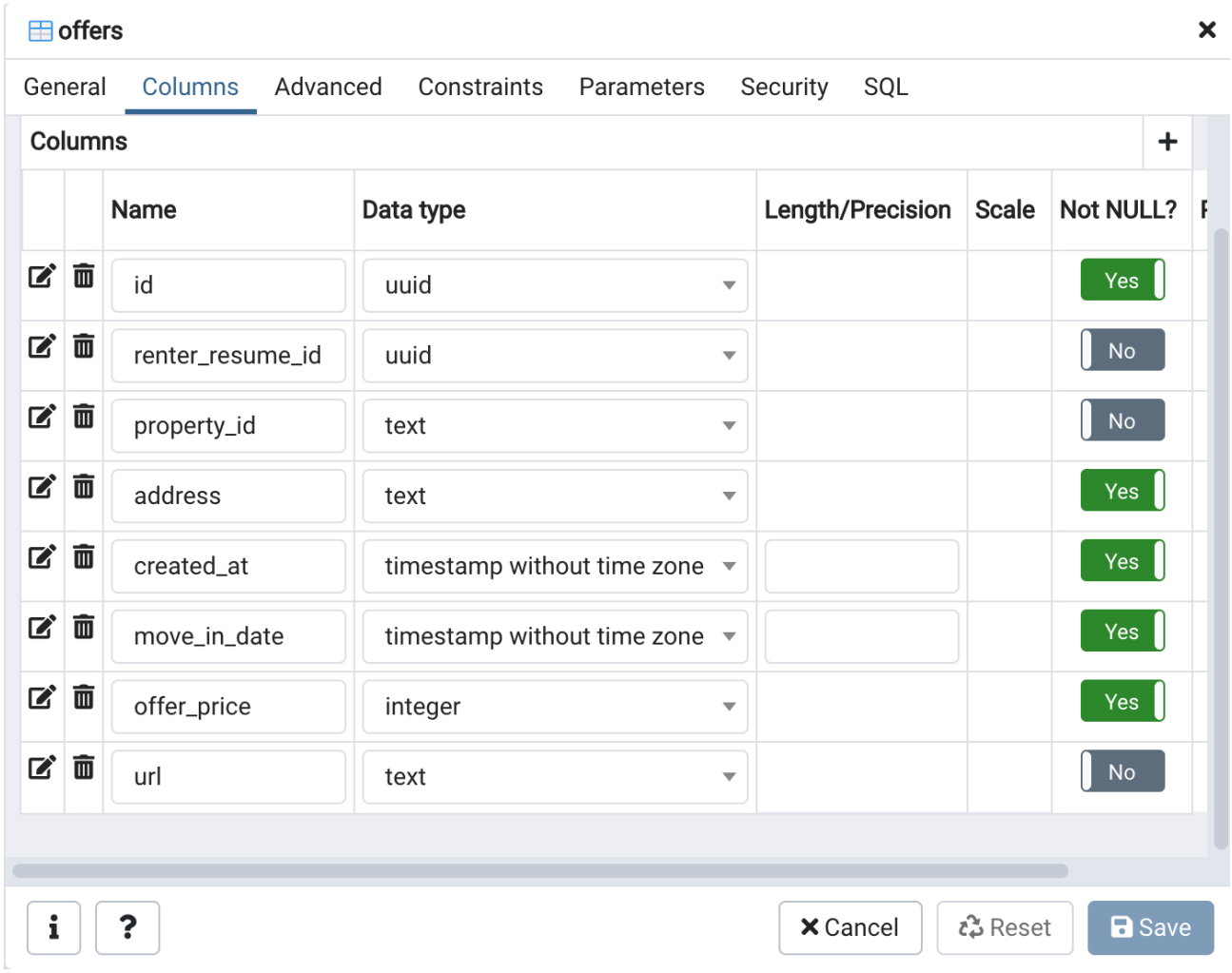


Рис. 1.23 Визначення таблички Offers

Після того, як було побудовано основні сутності та описано їх функцію у додатку, варто прояснити, яким чином вони будуть пов”язані між обою у додатку.

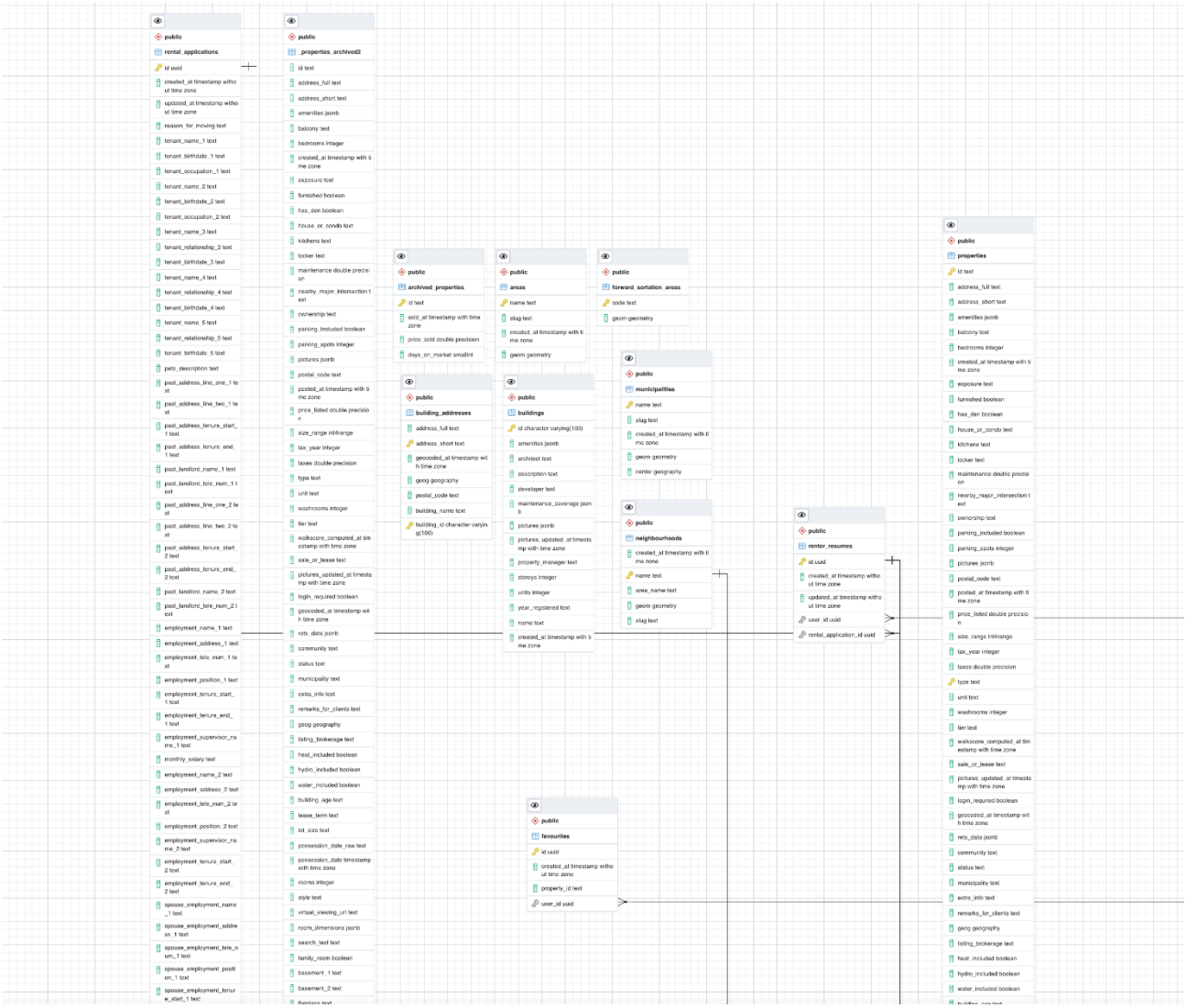


Рис. 1.24 Діграма відношень таблиць

Перейдемо до процесу розгортання (деплойменту) бази даних. Під хостингом (також, даний процес називають деплойментом) розуміють процесс, під час якого, базу даних переносять уже у мережу Інтернет, і таким чином розробник зможе підключатись до неї уже з будь-якої машини, не тількилокальної як у нашому випадку, саме тому - даний пункт є важливим, оскільки упроцессі командної розробки - буде виникати потреба у зміні тих чи інших дани скаладовою потрібно створити локальний проект, також є підтримка хмарних проєктів та локальних серверів.

Для магістерської роботи найбільш підходящим рішенням, буде так зване “хмарне” (cloud) рішення. Була обрана платформа Amazon RDS (relational database service)

Основні її переваги:

1) Простий та зрозумілий процес розгротання - для того, щоб в лічені хвилини запустити готову до роботи базу даних, достатньо лише пару простих дій і команд в консолі управління AWS. Створені бази даних уже відразу настроєні для роботи з вибраним розробником типом сервера. Групи параметрів бази даних надають можливість точної настройки бази даних MySQL і точного управління нею

2) Швидка і надійна плафторма для зберігання даних - Amazon RDS надає два варіанти сховища на SSD-дисках для бази даних MySQL. Універсальне сховище забезпечує економічне зберігання даних для невеликих або середніх робочих нагрузок. Для висконагружених додатків видіяються уже більш значні ресурси, які дозволяють оброблювати до 40 000 операцій вводу-ввиводу в секунду. При необхідності можно виділити додатковий обьект сховища прям в процесі роботи, без простоїв у роботі додатку.

3) Резервне копіювання та відновлення - також, дана платформа надає можливість автоматичного резервного копіювання, яке довзоляє у разі втрати даних відновити останню копію створеної бази даних, в любий момент в межах останніх 35 днів (також можна створити резервну копію вручну). Ці резервні копії баз даних зберігаються в Amazon RDS, поки не будуть видалені вручну.

4) Моніторинг і метрики - дана платформа надає різні метрики для баз дданих, без додактової плати. На даний момент існує понад 50-и різних метрик, серед яких - використана оперативна пам’ять, файлової системи та дискових вводу виводу. Основні операційні метрики, в тому числі використання обчислювальних ресурсів, пам’яті і ресурсів сховища, можна переглянути через консоль управління

Отже, після того як було визначено на якій платформі створювати - можна перейти до створення бази даних на Amazon RDS.

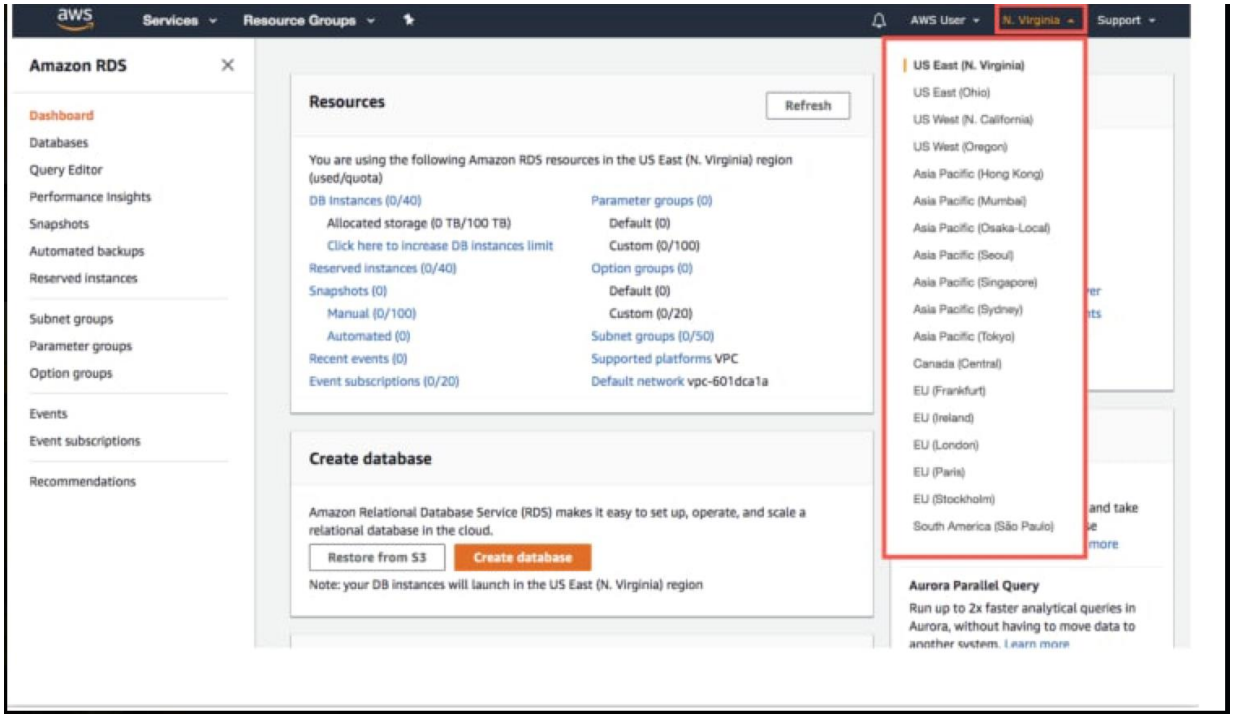


Рис. 1.25 Огляд особистого кабінету

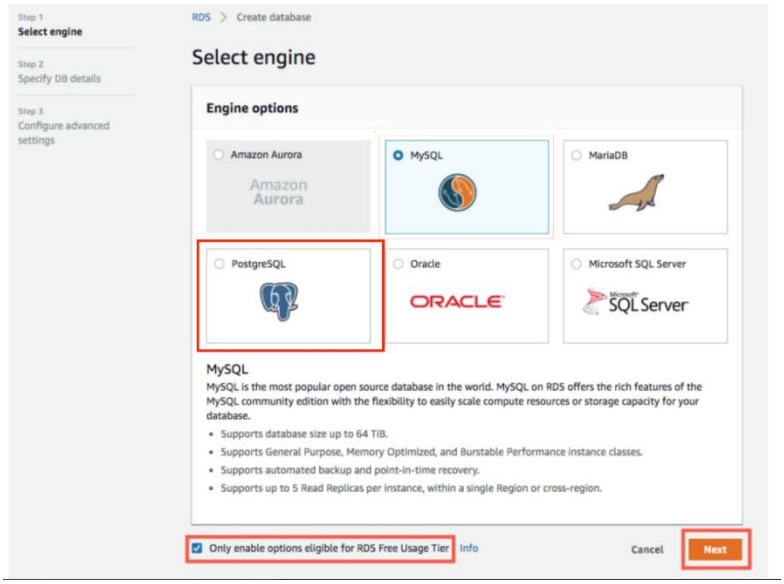


Рис. 1.26 Вибір платформи

Спочатку потрібно обрати платформу для створення бази даних (потрібно обрати PostgresSQL).

Важливо зазначити, що для локальних потреб вистачить і безкоштовних опцій, яких надає сервіс Amazon RDS.

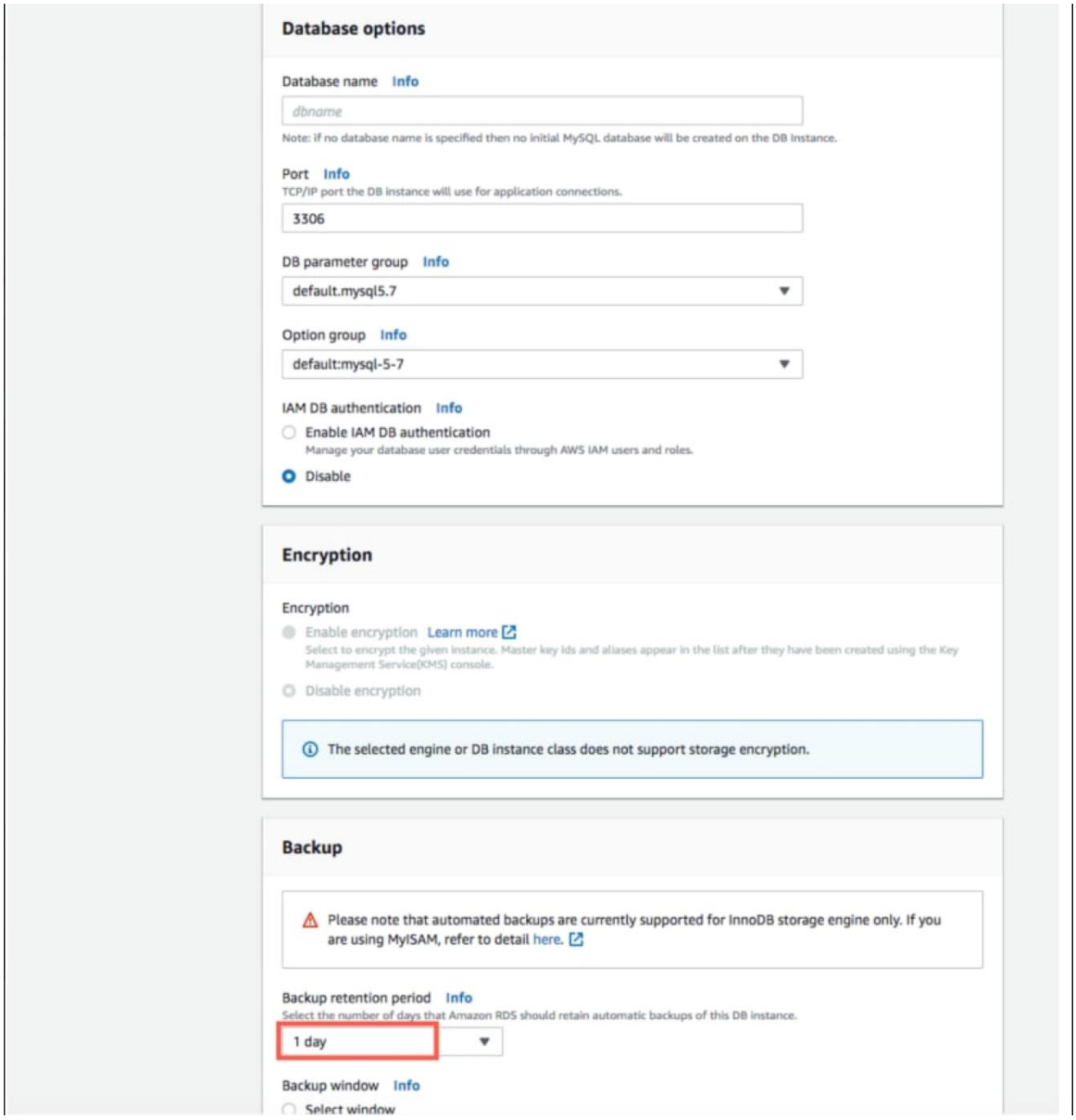


Рис. 1.27 Вибір опцій створення

Після створеня, впевнюємось у активному з’єднанні до бази даних

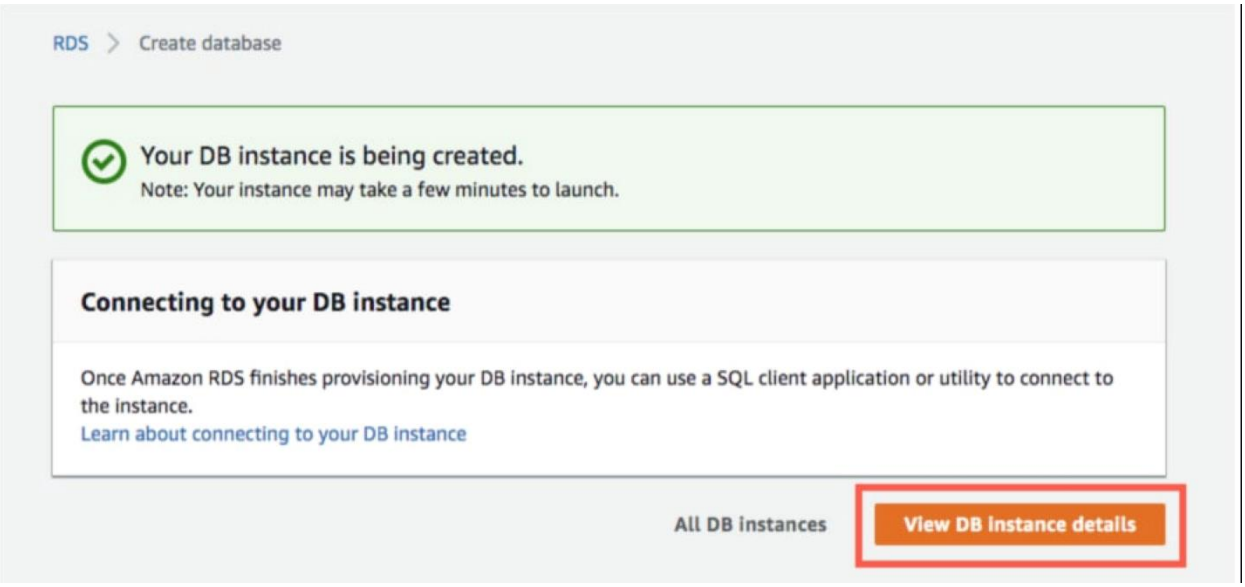


Рис. 1.28 Перевірка створення екземпляру бази даних

Таким чином, був пройдений повний цикл створеня бази даних, аналізу її зв'язків та функціональних потреб та її розгротанні на основі хмарної технології Amazon RDS, тому наступним кропокм буде розробка серверної частини, яка буде взаємодіяти з БД.

1.2.1. Серверна частина

Переходячи до серверної частини, важливо обізначити вибір мови програмування. Отже, зазвичай прийняття рішення щодо вибору мови лежить на розробниках. Хорошою практикою вважається уніфікований підхід - коли усі частини додатку мають однакову мову програмування, що дає можливість розробляти більш "чистий" продукт та уникати побічних ефектів при розробці, оскільки у нашій роботі присутнє API з нейронною мережою, найкращим рішенням буде використовувати мову Python, яка є стандартом для таких комплексних рішень

Для магістерської роботи підійде платформа на основі Python – Flask, оскільки завдяки їй можна розробювати стандартизовані інструменти для взаємодії веб-клієнту з серверною частиною.

Структура файлової системи серверної частини додатку надана на Рис. 1.29

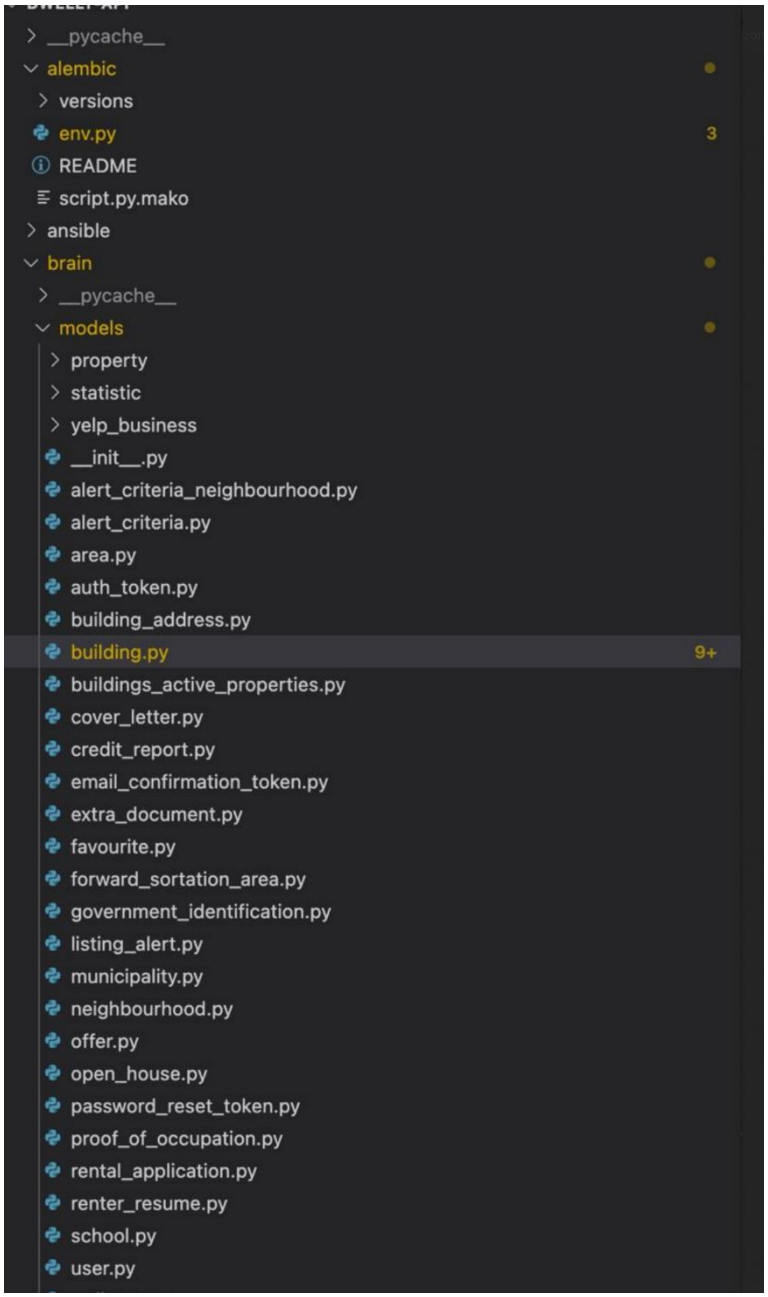


Рис. 1.29 Файлова структура серверної частини додатку

Основні директорії тут:

1) models - знаходяться основні сутності (класи) проекту. Було прийняте рішення використати класичну модель MVC (Model-View-Control).

```python
class Building(db.Model):
    __tablename__ = "buildings"

    amenities = sa.Column(pg.JSONB)
    architect = sa.Column(sa.Text)
    created_at = sa.Column(
        sa.DateTime(timezone=True), default=lambda: datetime.now(tz=pytz.utc)
    )
    description = sa.Column(sa.Text)
    developer = sa.Column(sa.Text)
    maintenance_coverage = sa.Column(pg.JSONB)
    name = sa.Column(sa.Text, nullable=False)
    pictures = sa.Column(pg.JSONB, default=None)
    pictures_updated_at = sa.Column(sa.DateTime(timezone=True))
    property_manager = sa.Column(sa.Text)
    storeys = sa.Column(sa.Integer)
    units = sa.Column(sa.Integer)
    year_registered = sa.Column(sa.Text)
    id=sa.Column(sa.VARCHAR, primary_key=True, nullable=False)
    addresses = sa.orm.relation("BuildingAddress")

    @property
    def properties_for_sale(self):
        return self.properties_by_building("sale")

    @property
    def properties_for_lease(self):
        return self.properties_by_building("lease")

    @property
    def sale_lease_count(self):
        return self.get_sale_lease_count()
```

Рис. 1.30 Приклад моделі для сутності (класу)

2) rest - знаходяться усі необхідні директорії для роботи серверної частини (контролери), відповідає за дії, які надходять на маршрути додатку (наприклад /buildings).

```python
class BuildingCollection(Resource):
    @use_args(
        {
            "page": fields.Integer(missing=1, validate=lambda p: p >= 1),
            "page_size": fields.Integer(missing=25),
            "bbox": fields.Str(
                validate=Regexp(f"{LONG_RE},{LAT_RE},{LONG_RE},{LAT_RE}")
            ),
        }
    )
    def get(self, args):
        predicate = []
        if "bbox" in args:
            coords = [float(coord) for coord in args["bbox"].split(",")]
            predicate.append(models.building_address.within_bbox_predicate(*coords))

        query = models.BuildingsActiveProperties.query.with_entities(
            models.Building.name,
            models.Building.pictures,
            models.BuildingsActiveProperties.addresses,
            models.BuildingsActiveProperties.sale,
            models.BuildingsActiveProperties.lease,
            models.BuildingAddress.geog
        ).join(
            models.Building,
            models.BuildingsActiveProperties.building_name == models.Building.name
        ).join(
            models.BuildingAddress,
            models.BuildingsActiveProperties.building_name == models.BuildingAddress.building_name
        ).filter(
            models.Building.addresses.any(
                sa.and_(models.BuildingAddress.geog != None, *predicate)
            )
        ).distinct(models.Building.name).from_self().order_by(
           models.BuildingsActiveProperties.sale + models.BuildingsActiveProperties.lease.desc()
        )

        buildings, pagination_headers = paginate(query, args["page"], args["page_size"])

        serialized = []
        for building in buildings:
            point = shapely.wkb.loads(building[5].desc, hex=True)
            building = {
                "name": building[0],
                "pictures": building[1],
                "sale_lease_count": {
                    "addresses": building[2],
                    "sale": int(building[3]),
                    "lease": int(building[4])
                },
                "addresses": [
                    {
                        "coords": [
                            point.x, point.y
                        ]
                    }
                ]
```

Рис. 1.31 Приклад файлу з папки rest (контролер)

API побудоване таким чином, що у кожного класу є методи get/post/put/delete (як і у всіх базованих REST API)

3) serializers - дана директорія відповідає за обробку даних, які будуть надходити на клієнт. В таких високонагружених додатках важливо слідкувати за тим які дані відправляються на клієнт, оскільки перенагруженні відповіді з серверу зазвичай призводять, до довгої обробки даних на самому клієнті перед тим як показати їх користувачу.

```
class ComparablePropertySerializer(Serializer):
    id = fields.Method("_id")
    air_conditioning = fields.Method("_air_conditioning")
    address_full = fields.Method("_address_full")
    address_short = fields.Method("_address_short")
    address_street = fields.Method("_address_street")
    bedrooms_display = fields.Method("_bedrooms_display")
    size_range = fields.Method("_size_range")
    washrooms = fields.Method("_washrooms")
    unit = fields.Method("_unit")
    login_required = fields.Method("_login_required")
    price_listed = fields.Method("_price_listed")
    price_sold = fields.Method("_price_sold")
    pictures = fields.Method("_pictures")
    parking_included = fields.Method("_parking_included")
    sold_at = fields.Method("_sold_at")
    sale_or_lease = fields.Method("_sale_or_lease")
    status = fields.Method("_status")
    type = fields.Method("_type")
    url = fields.Method("_url")
    created_at=fields.Method("_created_at")
    pictures_updated_at=fields.Method("_pictures_updated_at")
    class Meta:
        fields = (
            "id",
            "air_conditioning",
            "address_full",
            "address_short",
            "address_street",
            "bedrooms_display",
            "login_required",
            "parking_included",
            "price_listed",
            "price_sold",
            "pictures",
            "sold_at",
            "sale_or_lease",
            "size_range",
            "status",
            "type",
            "unit",
            "url",
            "washrooms",
            "created_at",
            "pictures_updated_at"
        )

    def _id(self, prop):
        return prop.get("mlsNumber")

    def _air_conditioning(self, prop):
        if prop.get("details").get("airConditioning") is None:
            return None
        return prop.get("details").get("airConditioning")

    def _address_full(self, prop):
        streetNumber = prop.get("address").get("streetNumber")
        streetName = prop.get("address").get("streetName")
        streetSuffix = prop.get("address").get("streetSuffix")
```

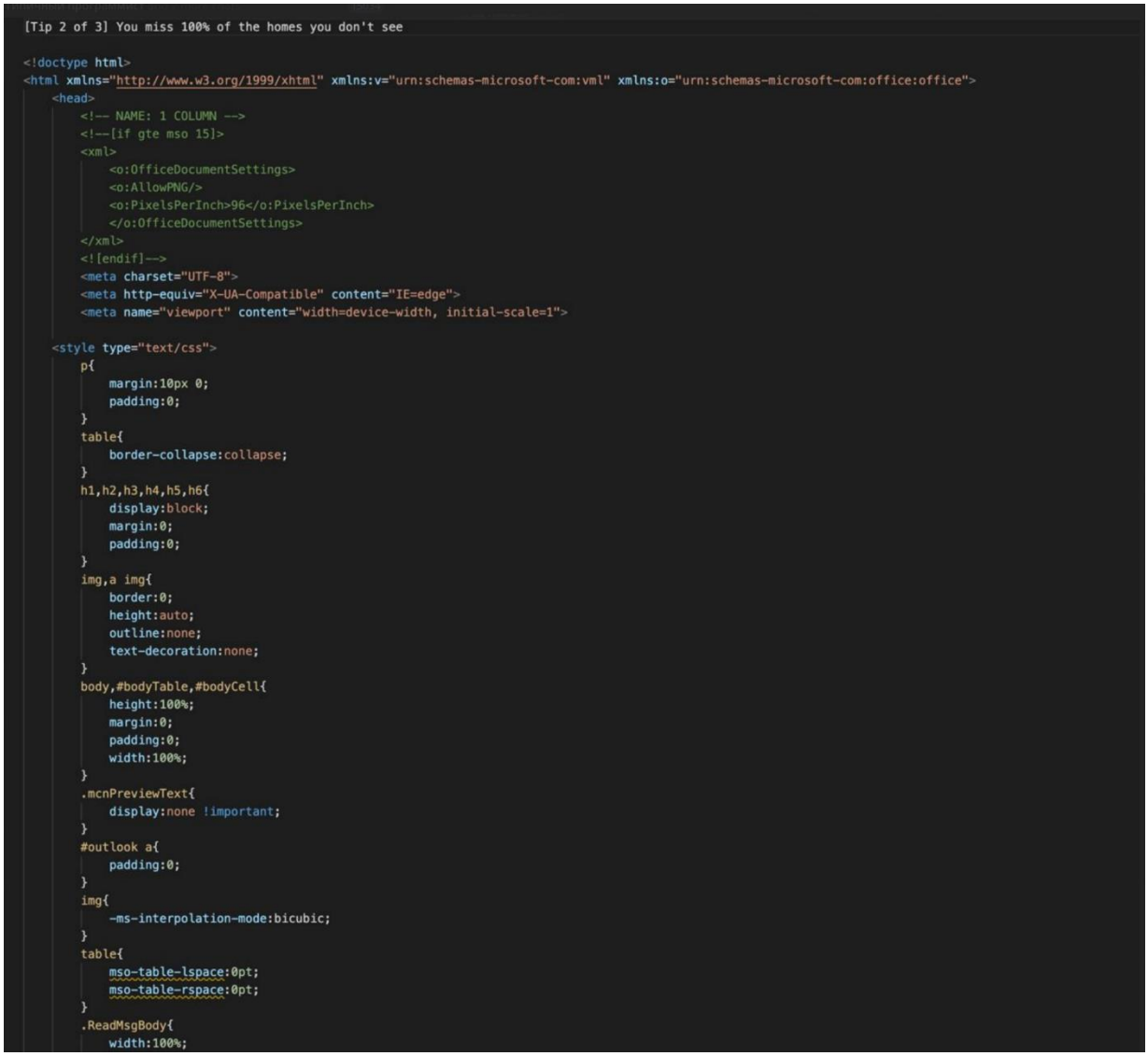
Рис. 1.32 Приклад файлу з директорії serializers

4) services - директорія, яка відповідає за звернення до зовнішніх сервісів, має у собі необхідний функціонал, для того щоб звертатись до сторонніх сервісів (з ініціалізацією ключів від сторонніх сервісів тощо).

```python
from flask import current_app
from brain import models
from datetime import datetime
import requests
from app import config

def get_property(mlsNumber,boardId):
    url = current_app.config["REPLIERS_API_URL"] + f"/listings/{mlsNumber}"
    params={}
    if boardId:
        params={
        "boardId": boardId
    }
    headers = {"REPLIERS-API-KEY": config.REPLIERS_API_KEY}
    resp = requests.get(url, headers=headers, params=params)
    resp.raise_for_status()
    return resp.json()

def get_archived_property(mlsNumber):
    url = current_app.config["REPLIERS_API_URL"] + f"/listings/archived/{mlsNumber}"
    headers = {"REPLIERS-API-KEY": config.REPLIERS_API_KEY}
    resp = requests.get(url, headers=headers)
    resp.raise_for_status()
    return resp.json()

def get_property_history(city, streetName, streetNumber,streetSuffix, unitNumber):
    url = current_app.config["REPLIERS_API_URL"] + f"/listings/history"
    headers = {"REPLIERS-API-KEY": config.REPLIERS_API_KEY}
    params={
        "city": city,
        "streetName":streetName,
        "streetNumber": streetNumber
    }
    if unitNumber:
        params["unitNumber"]=unitNumber
    if streetSuffix:
        params["streetSuffix"]=streetSuffix

    resp = requests.get(url, headers=headers, params=params)
    resp.raise_for_status()
    return resp.json()
```

Рис. 1.33 Приклад файлу з директорії services

5) mails - містить шаблони з електронними листами, які надсилаються користувачам у відповідь на конкретні дії (наприклад покупку квартири, назначення дати перегляду кваритри тощо).



```
[Tip 2 of 3] You miss 100% of the homes you don't see

<!doctype html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:v="urn:schemas-microsoft-com:vml" xmlns:o="urn:schemas-microsoft-com:office:office">
    <head>
        <!-- NAME: 1 COLUMN -->
        <!--[if gte mso 15]>
        <xml>
            <o:OfficeDocumentSettings>
            <o:AllowPNG/>
            <o:PixelsPerInch>96</o:PixelsPerInch>
            </o:OfficeDocumentSettings>
        </xml>
        <![endif]-->
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">

    <style type="text/css">
        p{
            margin:10px 0;
            padding:0;
        }
        table{
            border-collapse:collapse;
        }
        h1,h2,h3,h4,h5,h6{
            display:block;
            margin:0;
            padding:0;
        }
        img,a img{
            border:0;
            height:auto;
            outline:none;
            text-decoration:none;
        }
        body,#bodyTable,#bodyCell{
            height:100%;
            margin:0;
            padding:0;
            width:100%;
        }
        .mcnPreviewText{
            display:none !important;
        }
        #outlook a{
            padding:0;
        }
        img{
            -ms-interpolation-mode:bicubic;
        }
        table{
            mso-table-lspace:0pt;
            mso-table-rspace:0pt;
        }
        .ReadMsgBody{
            width:100%;
```

Рис. 1.34 Приклад файлу з директорії mails

6) util - містить у собі файли з допоміжними функціями, які можуть використовуватись по всьому проекту, тому щоб уникнути дублікацію коду було принято рішення винести найбільш поширені функції у окрему папку, для реалізації принципу перевикористання коду.

```python
import logging
import xml.etree.ElementTree as ET
from typing import Dict, List

import requests

logger = logging.getLogger(__name__)


class ExtractionError(Exception):
    pass


def parse_capability_urls(rets_response_text: str) -> Dict[str, str]:
    capability_names = [
        'Action', 'ChangePassword', 'GetObject', 'Logout', 'Search',
        'GetMetadata', 'Update', 'ServerInformation'
    ]
    # the rets_response_text contains other information that we are not interested in
    # such as MembernName, Broker, etc
    return {
        kv_str.split('=')[0]: kv_str.split('=')[1]
        for kv_str in rets_response_text.split('\n')
        if kv_str.split('=')[0] in capability_names
    }


def extract_xml_data(r: requests.Response) -> ET.Element:
    text = r.text
    if text == '':
        logger.error('Extraction Error: RETS response contains invalid XML data')
        raise ExtractionError(
            'Expected the RETS response to contain valid XML data, received "".'
        )
    tree = ET.fromstring(r.text)

    reply_code = tree.attrib['ReplyCode']
    if reply_code != '0':
        logger.error(f'Received an Error from the RETS server. ReplyCode {reply_code}.')

        raise ExtractionError(
            f'Received an Error from the RETS server. ReplyCode {reply_code}.')
    return tree


def extract_capability_urls(r: requests.Response) -> Dict[str, str]:
    xml_data = extract_xml_data(r)
    rets_reponse_text = xml_data.find('RETS-RESPONSE').text
    return parse_capability_urls(rets_reponse_text)


def extact_search_results(r: requests.Response) -> List[Dict[str, str]]:
    xml_data = extract_xml_data(r)
    listing_els = xml_data.findall('.//Listing')
    results: List[Dict[str, str]] = []

    for listing_el in listing_els:
        attribs: Dict[str, str] = {}
```

Рис. 1.35 Приклад файлу з директорії utils

7) workers - директорія, яка містить у собі файли з функціями, які мають виконуватись на періодичній основі (наприклад раз у неділю), такі функції потрібні, щоб наприклад оновлювати якийсь контент на клієнті тощо.

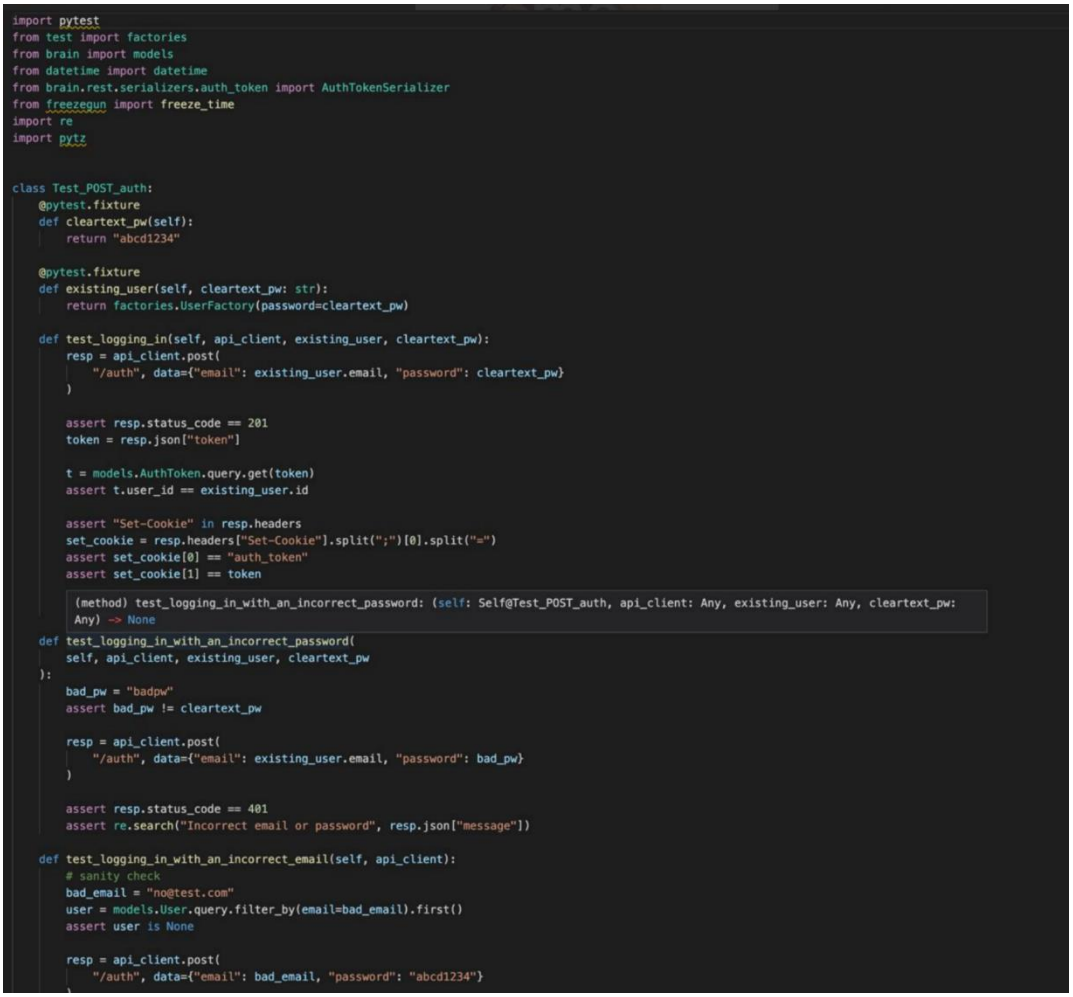
brain > workers > rets_syncer.py > ...

```python
from . import celery
from brain import models
from brain.services.rets import Rets
import brain.services.walkscore as walkscore_srv
import logging

logger = logging.getLogger(__name__)

@celery.task
def sync_properties() -> None:
    rets: Rets = Rets()
    rets.authenticate()

    logger.info(f"LAST SYNCED AT: {rets.properties_synced_at}")
    if rets.properties_synced_at is None:
        rets.sync_properties_full()
    else:
        logger.info("incremental sync started")
        rets.sync_properties_incremental()
        logger.info("incremental sync completed")

    logger.info("sync pictures started")
    rets.sync_pictures()
    logger.info("sync pictures compeleted")
    logger.info("geocoding started")
    models.ActiveProperty.geocode_nongeocoded_properties()
    logger.info("geocoding completed")
    logger.info("walkscore started")
    walkscore_srv.compute_scores()
    logger.info("walkscore completed")

@celery.task
def sync_walkscore() -> None:
    logger.info("walkscore started")
    walkscore_srv.compute_scores()
    logger.info("walkscore completed")
```

Рис. 1.36 Приклад файлу з директорії workers

Важливо зазначити, що ці функції викликаються з-за допомогою фреймворку Seleri для мови Python. Таким чином ми маємо змогу задавати періодичність функціям і сторювати основу для постійної автоматизованої підтримки актуального контенту додатку.
Також є можливість запускати ці функції вручну через термінал, тому гнучкість тут на високому рівні.

8) tests - містить у собі власне тести для компонентів додатку, тести дають можливість упевнитись, що додаток працює стабільно навіть після змін, тому такий підхід є достатньо сучасним у нашому випадку. Можна також виокремити цілий підхід - TDD (Test Driven Development), який базується на першочерговому написанні тестів перед розробкою самих компонентів, що у високонагружених додатків грає велику роль при їх оптимізації.



```python
import pytest
from test import factories
from brain import models
from datetime import datetime
from brain.rest.serializers.auth_token import AuthTokenSerializer
from freezegun import freeze_time
import re
import pytz


class Test_POST_auth:
    @pytest.fixture
    def cleartext_pw(self):
        return "abcd1234"

    @pytest.fixture
    def existing_user(self, cleartext_pw: str):
        return factories.UserFactory(password=cleartext_pw)

    def test_logging_in(self, api_client, existing_user, cleartext_pw):
        resp = api_client.post(
            "/auth", data={"email": existing_user.email, "password": cleartext_pw}
        )

        assert resp.status_code == 201
        token = resp.json["token"]

        t = models.AuthToken.query.get(token)
        assert t.user_id == existing_user.id

        assert "Set-Cookie" in resp.headers
        set_cookie = resp.headers["Set-Cookie"].split(";")[0].split("=")
        assert set_cookie[0] == "auth_token"
        assert set_cookie[1] == token

    def test_logging_in_with_an_incorrect_password(
        self, api_client, existing_user, cleartext_pw
    ):
        bad_pw = "badpw"
        assert bad_pw != cleartext_pw

        resp = api_client.post(
            "/auth", data={"email": existing_user.email, "password": bad_pw}
        )

        assert resp.status_code == 401
        assert re.search("Incorrect email or password", resp.json["message"])

    def test_logging_in_with_an_incorrect_email(self, api_client):
        # sanity check
        bad_email = "no@test.com"
        user = models.User.query.filter_by(email=bad_email).first()
        assert user is None

        resp = api_client.post(
            "/auth", data={"email": bad_email, "password": "abcd1234"}
        )
```

```
(method) test_logging_in_with_an_incorrect_password: (self: Self@Test_POST_auth, api_client: Any, existing_user: Any, cleartext_pw:
Any) -> None
```

Рис. 1.36 Приклад з директорії test

9) config - відповідає за environmental змінні - це такі сутності, які використовуються по усьому додатку і їх можна змінити у цій папці, наприклад для того, змінити роут до запросу, або вказати, використовувати інший набір цих змінних (для безпеки усі ключі помчені червоним кольором, бо це high-sensitive (вразлива) інформація і її не бажано показувати/викладати будь-де, щоб не було загрози втрати даних);

```
class BaseConfig:
    DB_DRIVER = "postgresql+psycopg2"
    DB_HOST = "localhost"
    DB_PORT = 5432

    REDIS_HOST = "localhost"
    REDIS_PORT = "6379"
    REDIS_URL = "redis://localhost:6379"
    LOG_DEST_DEBUG = "/var/log/brain/brain.debug.log"
    LOG_DEST_INFO = "/var/log/brain/brain.info.log"
    LOG_DEST_ERR = "/var/log/brain/brain.err.log"
    UPLOAD_DIR = os.path.join(os.getcwd(), "uploads/")
    PROPERTY_PICS_DIR = os.path.join(os.getcwd(), "static/property_pictures/")
    TEMP_DIR = "/tmp/brain/"
    S3_STATIC_BUCKET = "static.dwelly.ca"
    GA_TRACKING_ID = "UA-116681306-1"
    ELASTIC_SEARCH_URL = 'http://localhost:9200'
    LOGIN_EXPIRATION_DAYS = 90
```

Рис. 1.37 Приклад конфіг файлу для серверної частини.

Організація логіки на стороні серверної частини є важливим етапом розробки веб-застосунку, приклади даниого процесу надані на малюнках нижче.

На Рис 1.38 надано організацію основних маршрутів. Усі маршрути, на які можна робити запити, зазначені у файлі init.py, таким чином ми маємо усі визначення файлів в одному місці, що є зручно для перегляду і подальшої підтримки проекту. Даний файл містить у собі всі типи шляхів для класичного REST додатку (GET/POST/DELETE/PUT).

Тож яким чином працює основна логіка в нашому серверному додатку - для наглядності було побудовано діаграму

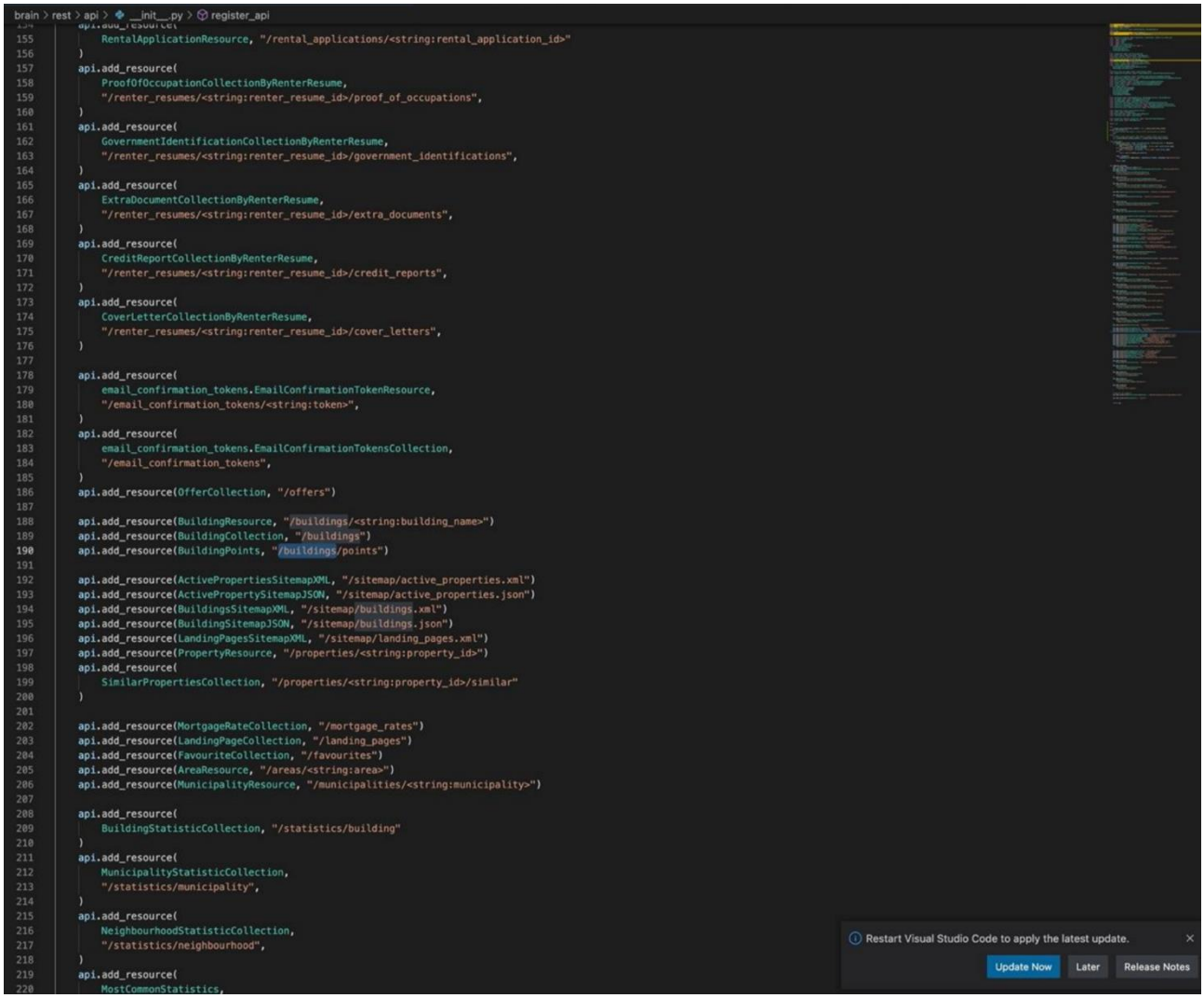


Рис. 1.38 Приклад як організовані основні маршрути

Отже, усі маршрути, на які можна робити запити, зазначені у файлі init.py, таким чином ми маємо усі визначення файлів в одному місці, що є зручно для перегляду і подальшої підтримки проекту. Даний файл містить у собі всі типи шляхів для класичного REST додатку (GET/POST/DELETE/PUT).

Тож яким чином працює основна логіка в нашому серверному додатку - для наглядності було побудовано діаграму.

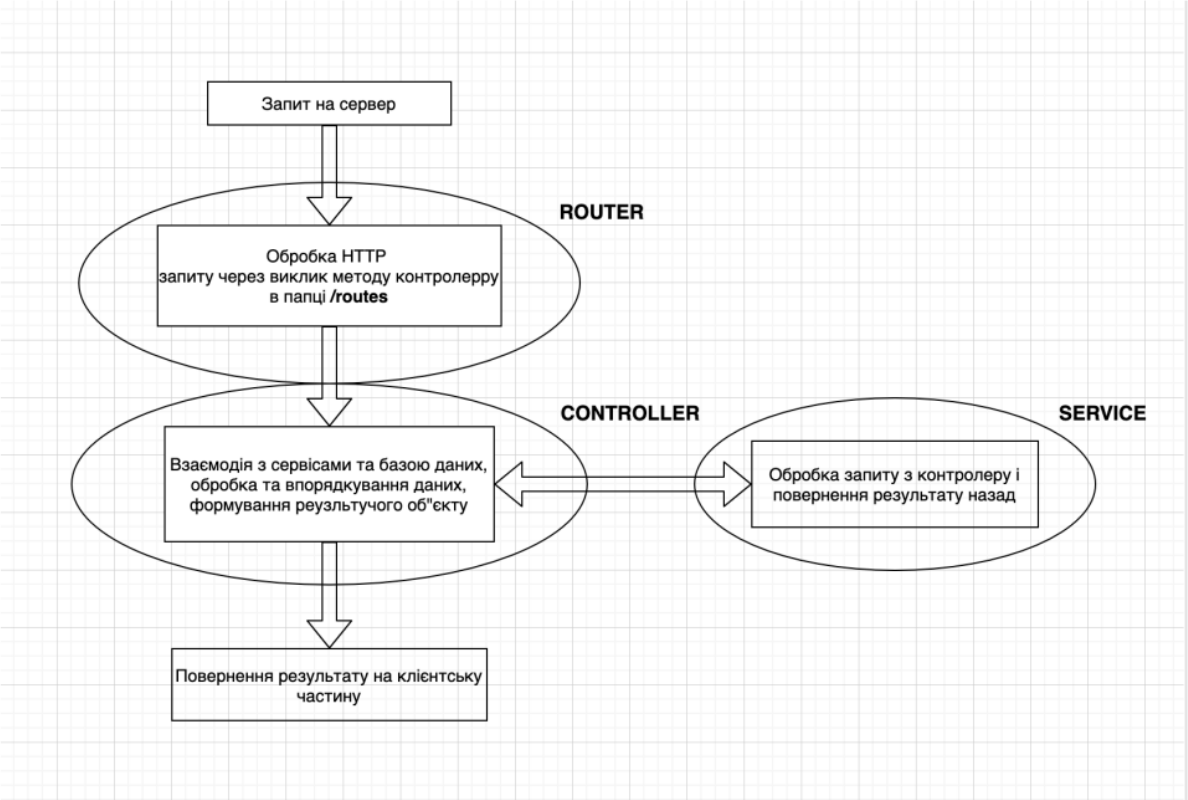


Рис. 1.39 Приклад роботи серверної частини додатку

Важливо зазначити, що з нарощуванням комплексності додатку, також зрозстає кількість різного роду помилок і з перспективи розробника потрібно чітко знати, що стало причиною тої чи іншої помилки, щоб у разі чого можна було завчасно її поправити або хоча би провести дослідження на предмет потенційних проблем. Також кінцевому користувачу непотрібно бачити повну помилку, оскільки для нього вона буде нерепрезантивною і може більше зашкодити чим допомогти, тому також потрібно потурбуватись над “люб’язністю” помилок, щоб уникнути таких проблем. Зазвичай для таких проблем розроблюється окремий слой додатку, який відповідає за перехоплення і подальше оброблення помилок.

```
const common = module.exports = {};
const { GENERIC_ERROR, IRISTEL_503, BELL_API_NOT_WORKING } = require("../errors");
common.errorHandler = function errorHandler(err, req, res, next) {
    console.error(err)

    if (err.predefined) {
        return res.status(err.status).json({
            message: err.message,
            code: err.code,
            errors: []
        })
    }
    if(err.isBell){
        return res.status(BELL_API_NOT_WORKING.status).json({
            message: BELL_API_NOT_WORKING.message,
            code: BELL_API_NOT_WORKING.code,
            errors: []
        })
    }
    if (err.isIristel && err.status === 503) {
        return res.status(IRISTEL_503.status).json({
            message: IRISTEL_503.message,
            code: IRISTEL_503.code,
            errors: []
        })
    }
    return res.status(GENERIC_ERROR.status).json({
        message: GENERIC_ERROR.message,
        code: GENERIC_ERROR.code,
        errors: []
    })
};
```

Рис. 1.40 Приклад обробок помилок на серверній частині додатку

Отже, завжди можна перехватити помилку і обробити її, а також на віртуальних машинах можна виставити додаткові логи, щоб розробники, які мають доступ мали змогу оброблювати ті складні помилки, які кінцевому користувачу не показуються.

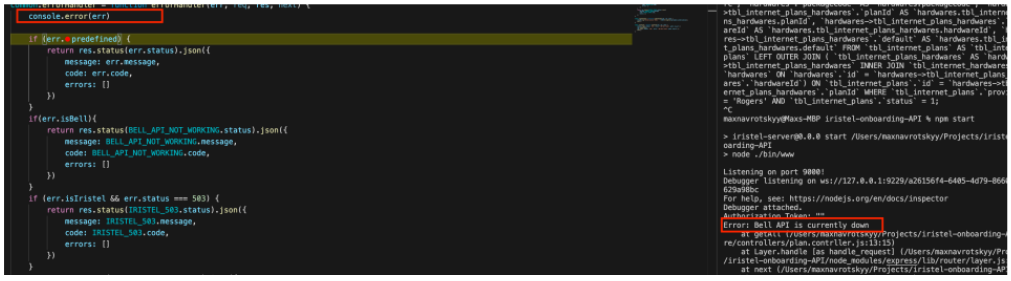


Рис. 1.41 Приклад перехвату помилок розробником на серверній частині додатку

Таким чином, було виявлено, що для ясності повного процесу обробки запиту порібно оновити діаграму логіки серверної частини додатку.

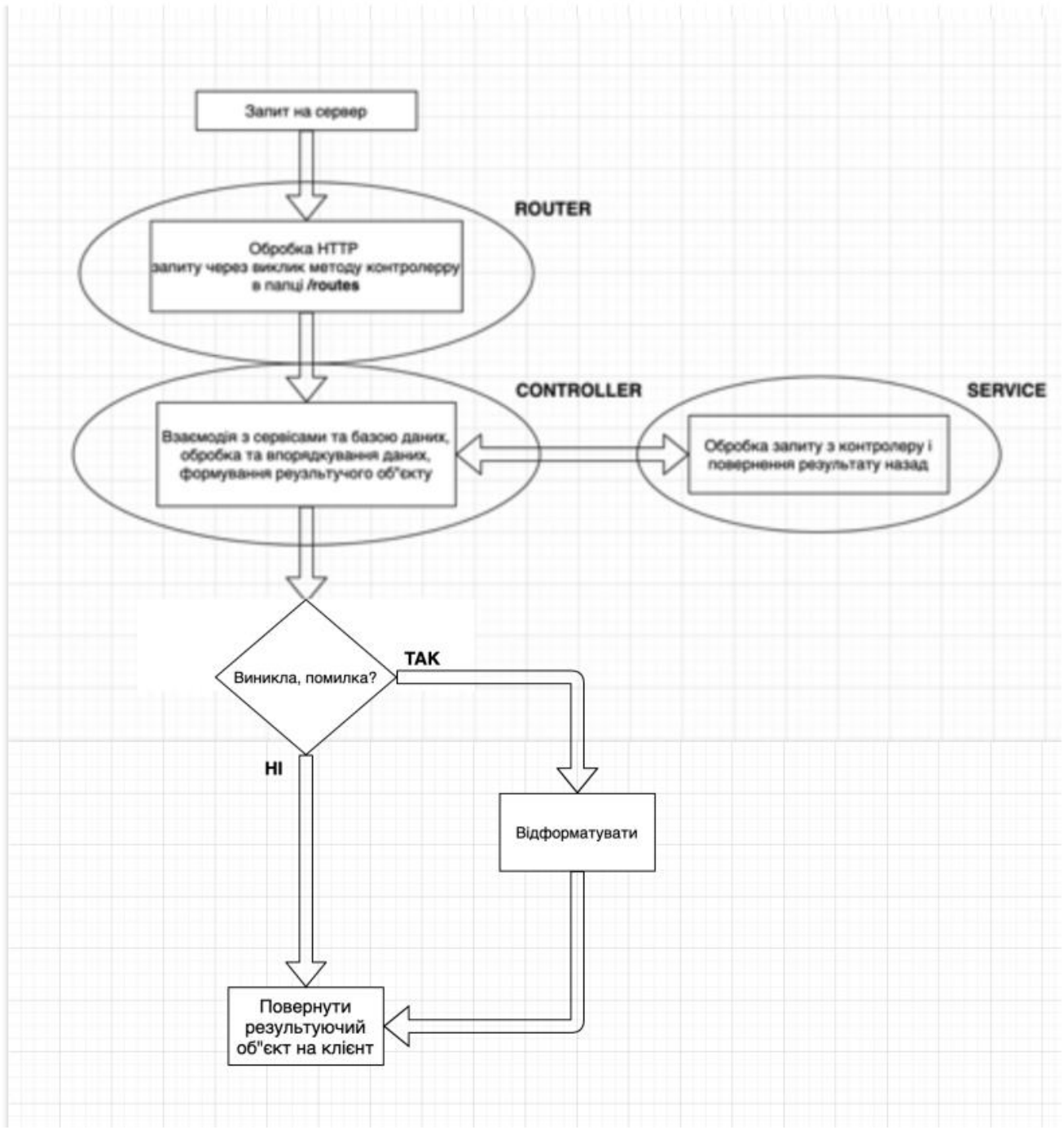


Рис. 1.42 Оновлена схема роботи серверної частини додатку

Після огляду основних принципі роботи серверної частини додатку, потрібно замислитись, яким чином наша клієнтська частина буде взаємодіяти з серверною частиною через мережу інтернет. Таким чином ми можемо впевнитись, що

додаток буде працювати незалежно чи увімкнена локальна машина. Було обрано платформу Heroku, оскільки для базових додатків вона безкоштовна і процес розгортання є простим і зрозмілими розробниками, незнайомих з нею.

Потрібно виконати деіклька кроків:

1) Для розгортання додатку в Heroku необхідно впевнитись, що весь код лежить уже у віддаленому репозиторію, далі необхідно зберегти свій ключ у Heroku, і щоб впевнитись що все працює ввести у терміналі: heroku –version

2) Консоль має вивести:

```
heroku-toolbelt/3.40.11 (x86_64-darwin10.8.0) ruby/1.9.3
```

Рис. 1.43 Приклад версії оточення в терміналі

3) Після цього, як було протестовано інструмент потрібно залогінитись (увійти у
свій аккаунт)

```
heroku login
Enter your Heroku credentials.
Email: joe@example.com
Password: test1234
```

Рис. 1.44 Приклад логіну

4) Далі переходимо на сайт Heroku, та створити додаток через кнопку “create new app”

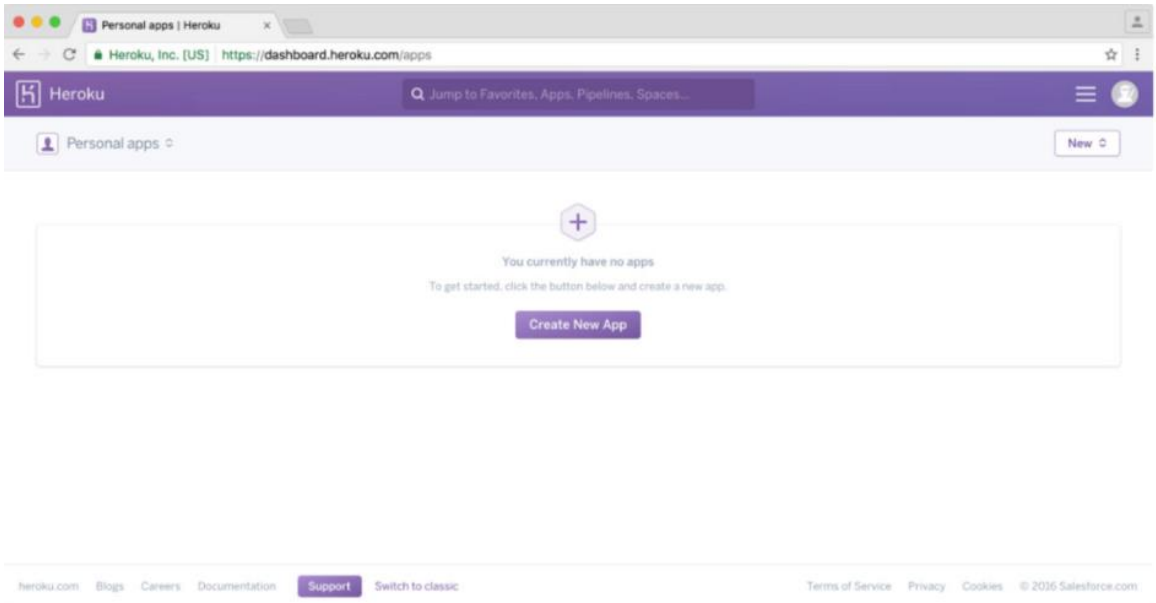


Рис. 1.45. Приклад версії оточення в терміналі

5) Далі, коли під'єднали репозиторій до додатку чекаємо на створення.

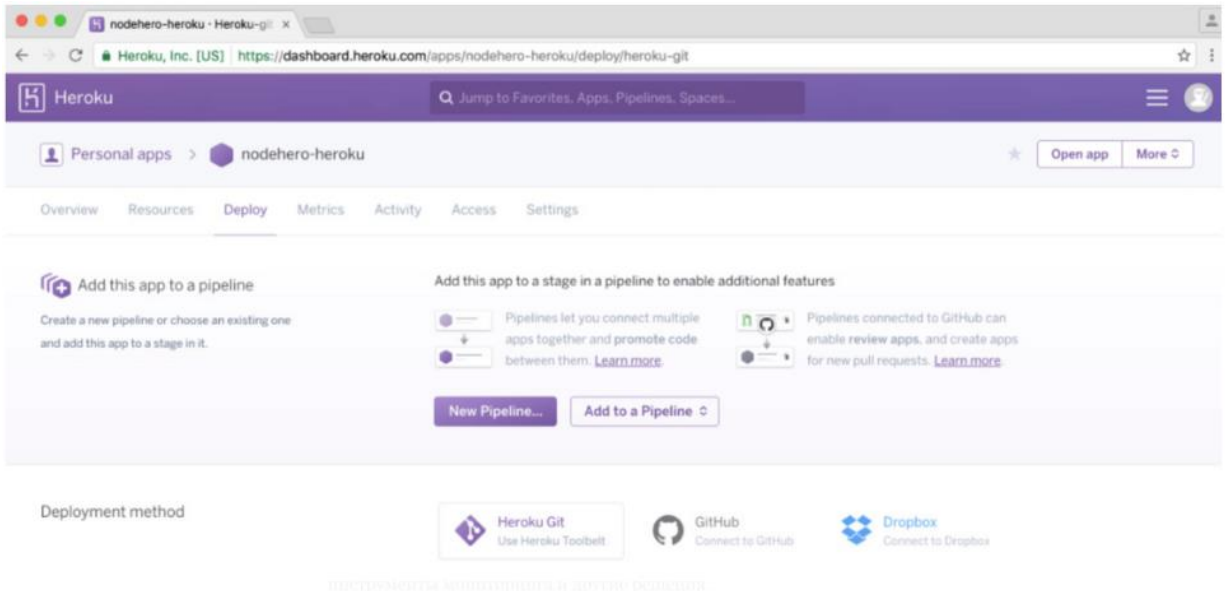


Рис. 1.46 Приклад розгортання додатку

1.2.2. Розробка API з нейронною мережею

Після розробки серверної частини, потрібно приділити увагу власне API, яке ми будемо інтегрувати у серверну частину. Як зазначалось раніше мова програмування для цих двух компонентів додатку буде ідентичною і в цьому випадку - Python.

Для функціонуючого API, ми також використаємо framework Flask, оскільки він достатньо добре підходить як для додатків зі складною логікою, так і для простих рішень. Отже файл server.py буде виглядати ось так:

```python
from file_utils import download_file, delete_file
import os
import requests
from keras import backend as K

#load model
import vgg16_places_365_model

if not os.path.exists('uploads'):
  os.mkdir('uploads')

app = Flask(__name__)

@app.route('/<filename>')
def predict(filename):
  try:
    #Before prediction
    K.clear_session()
    remote_addr = f"http://images.dwelly.ca/{filename}"
    file_path = download_file(remote_addr)
    prediction = vgg16_places_365_model.predict('uploads/' + file_path)
    #After prediction
    K.clear_session()
    delete_file('uploads/' + filename)
    return prediction, 200
  except requests.exceptions.HTTPError:
    return { "err" : "Image not found", "status": 404}, 404
  except Exception as e:
    print("Error", e)
    return { "err" : "Internall error", "status": 500}, 500

if __name__ == '__main__':
  # app.debug = True
  app.run(host='0.0.0.0', port=3002)
```

Рис. 1.47 Приклад основного файлу запуску додатку

Далі, власне для нейронної мережи, потрібно використати бібліотеку Keras і Tensorflow - вони дозволяють визначити, на чому буде можливість запускати роботу нейронної мережі - на процесорі чи на ядрах відеокарти. Якщо є можливість краще усього працювати на ядрах відеокарти - оскільки вони більше всього підходять для обчислень і можуть виділити усю оперативну пам”ять для цього.

Запускаючи нейронні мережі на процессорі (наприклад на ноутбуках тощо), це буде набагато повільніше, оскільки процес буде працювати паралелньо з усіма программами на процессорі, тому це не є оптимальним рішенням, хоча і єдину правильним особливо на девайсах без дискретної відеокарти.

Робота з нейронною мережею

Після того як було розроблено базову структуру проекту, варто перейти безпосередньо до програмної реалізації. Перед тим як користуватись нейронною мережею, потрібно відібрати необхідний датасет (набір вхідних даних). Для цього потрібно зайнятись попереднім маркуванням матеріалу. Процес достатньо трудомісткий, оскільки треба було обробити від 10 000 зображень, але через деякий час усе було готово, процес маркування виглядає так:

Рис. 1.48 Приклад маркувального додатку

Була розроблена невелика утиліта (теж на мові Python), для того щоб можна було вручну промаркувати “правильні” класи картинок.

Рис. 1.49 Приклад маркувального додатку

Далі після ручного маркування усіх зображень, можна перейти до частини тренування моделі.

Тренування мережі

Тренуємо мережу запуском наступної команди:

- python dl_based_parser_train.py

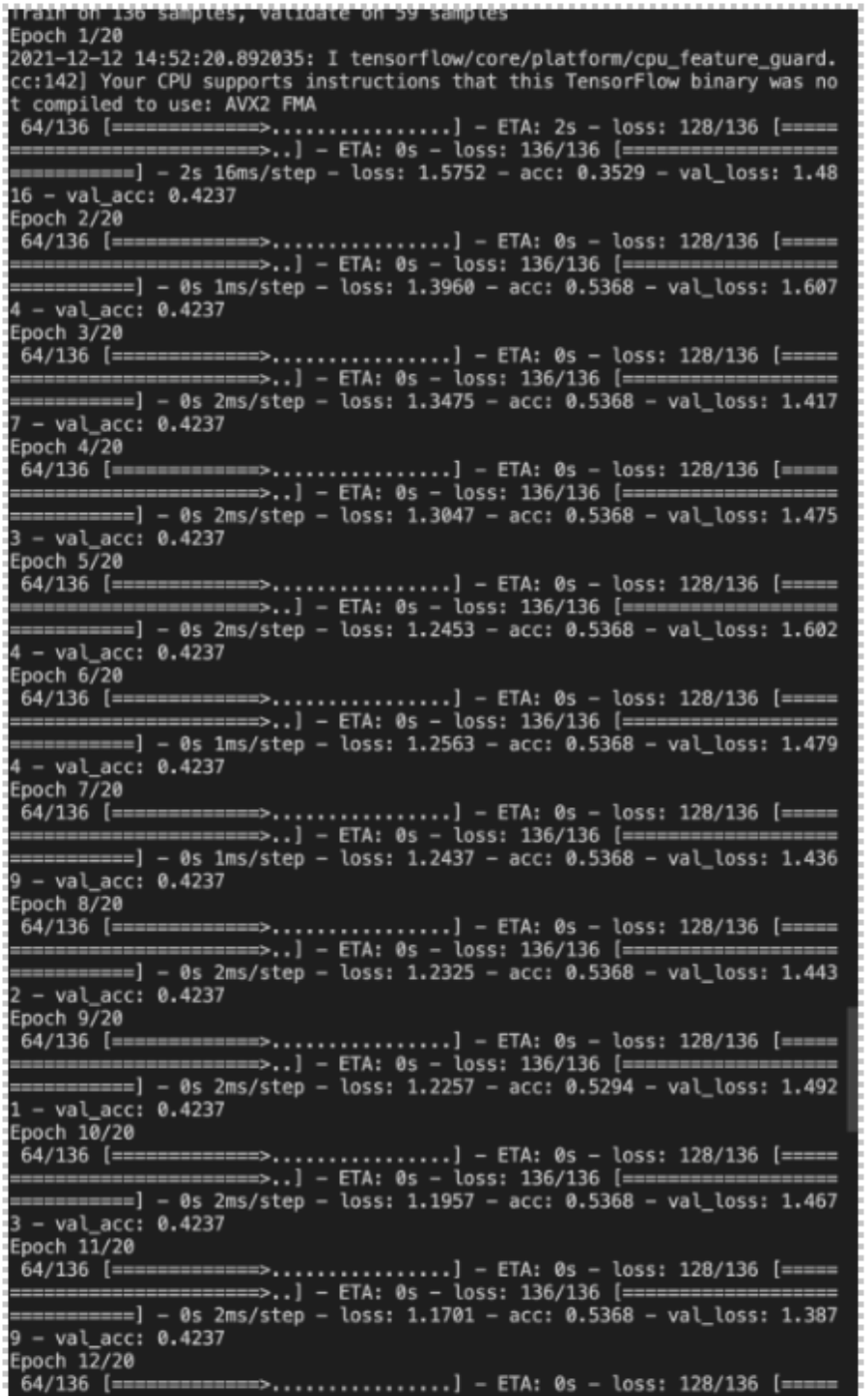


Рис. 1.50 Процес тренування моделі

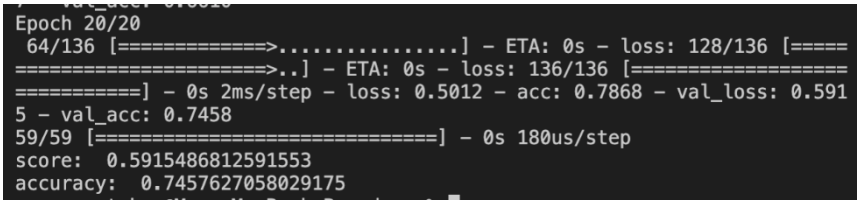


Рис. 1.51 Процес тренування моделі - результат

Для того щоб упевнитись що усе працює, перейдемо до файлу визначення моделі:

```
x = Conv2D(filters=64, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
           activation='relu', name='block1_conv1')(img_input)

x = Conv2D(filters=64, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
           activation='relu', name='block1_conv2')(x)

x = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name="block1_pool", padding='valid')(x)

# Block 2
x = Conv2D(filters=128, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
           activation='relu', name='block2_conv1')(x)

x = Conv2D(filters=128, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
           activation='relu', name='block2_conv2')(x)

x = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name="block2_pool", padding='valid')(x)

# Block 3
x = Conv2D(filters=256, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
           activation='relu', name='block3_conv1')(x)

x = Conv2D(filters=256, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
           activation='relu', name='block3_conv2')(x)

x = Conv2D(filters=256, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
           activation='relu', name='block3_conv3')(x)

x = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name="block3_pool", padding='valid')(x)

# Block 4
x = Conv2D(filters=512, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
           activation='relu', name='block4_conv1')(x)

x = Conv2D(filters=512, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
           activation='relu', name='block4_conv2')(x)

x = Conv2D(filters=512, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
           activation='relu', name='block4_conv3')(x)

x = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name="block4_pool", padding='valid')(x)

# Block 5
x = Conv2D(filters=512, kernel_size=3, strides=(1, 1), padding='same',
           kernel_regularizer=l2(0.0002),
```

Рис 1.52 Приклад моделі

Нами був доданий слой в архітектурі, який дозволяє після класифікації зображення сказати його точний клас (користувацький “ззовні”/”назовні”).

Для того, щоб перевірити роботу моделі і нейронної мережі, необхідно запустити її через термінал:

Робота моделі виконується через команду:

- python dl_based_parser_predict.py

```python
def predict(image_name):
    image = Image.open(image_name)
    image = np.array(image, dtype=np.uint8)
    image = resize(image, (224, 224))
    image = np.expand_dims(image, 0)

    model = VGG16_Places365(weights='places')
    predictions_to_return = 5
    preds = model.predict(image)[0]
    top_preds = np.argsort(preds)[::-1][0:predictions_to_return]

    # load the class label
    file_name = './categories_places365.txt'
    classes = list()
    with open(file_name, 'r') as class_file:
        for line in class_file:
            classes.append(line.strip().split(' ')[0][3:])
    classes = tuple(classes)

    file_name = './IO_places365.txt'
    IO_classes = list()
    with open(file_name, 'r') as class_file:
        for line in class_file:
            IO_classes.append(line.strip().split(' ')[1])
    IO_classes = tuple(IO_classes)
    classes_preds = []
    IO_classes_preds = []
    indoorOccurences = 0
    outdoorOccurences = 0
    place_type = "indoor"
    # output the prediction
    for i in range(0, 5):
        print(classes[top_preds[i]])
        classes_preds.append(classes[top_preds[i]])
        IO_classes_preds.append(int(IO_classes[top_preds[i]]))
        if IO_classes_preds[i] == 1:
            indoorOccurences += 1
        else:
            outdoorOccurences += 1

    if outdoorOccurences > indoorOccurences:
        place_type = "outdoor"

    #After prediction
    K.clear_session()
    return {"url": image_name, "place_classes" : classes_preds, "place_type": place_type}
```

Рис. 1.53 Код для класу класифікації

На виході ми отримуємо класи зображення і чи зображення зроблено на вулиці чи зсередини.

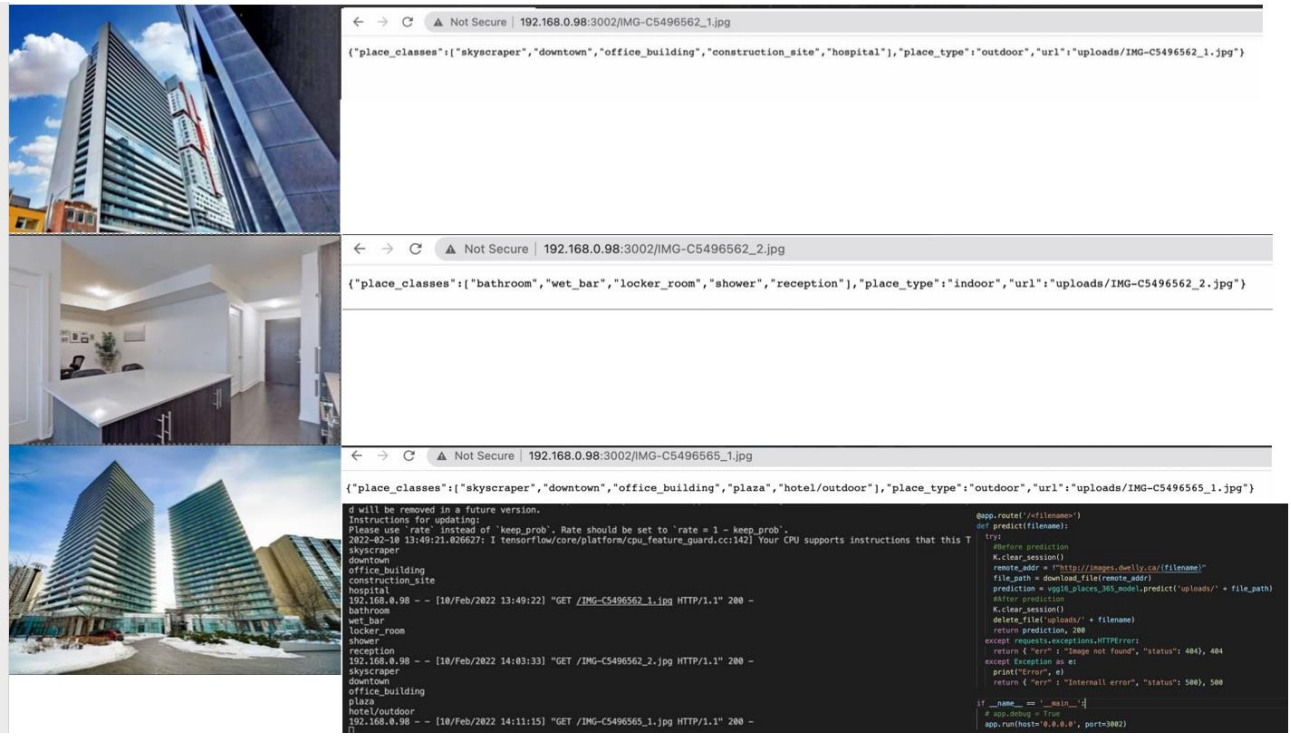


Рис. 1.54 Результат роботи

Дане API можна викликати як в ручну для конкретно цікавого користиувачу зображення, так і для автоматизованих задач.

Висновки до розділу 1

У даному розділі було розглянуто основні принципи роботи серверної частини додатку, взаємодію клієнтської частини з серверною частиною через мережу інтернет. В ході проектування отримано, що додаток буде працювати незалежно чи увімкнена локальна машина. Було обрано платформу Heroku, оскільки для базових додатків вона безкоштовна і процес розгортання є простим і зрозмілими розробниками, незнаймих з неюПісля визначення таблиць та визначення їх ролі у проекті, було побудовано діаграмму відношень. У процес розробки проекту, список може поповнюватись, що не є проблемою для PostgresSQL платформи Був

пройдений повний цикл створення серверної частини, аналізу її основних компонентів та їх логіки, її розгротання на основі платформи Heroku, тому наступним кроком, буде створення клієнтської частини, яка буде взаємодіяти з серверною частиною. А також розроблено API для інтеграції у серверну частину.

РОЗДІЛ 2

РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

Хоча для серверних частин була використана мова Python, для клієнтської найбільшим плюсом буде використання мови Javascript - оскільки браузери і більшості своїй її якраз і виокристовують для користувацьких сценаріїїв.

Для розробки був використаний фреймворк React, на це є декілька причин:

1) Модульний підхід - увесь код поділений на компоненти, а основна суть цих компонентів, це перевикористання коду, що прибирає проблему дублікування коду та його перевикористання

2) Роутинг - у розділі про серверну частину, було розказано як працює роутинг, тут він схожий, лише за одним моментом, що замість структурованої відповіді з серверу, ми повертаємо компонент, елемент який показується користувачу.

Сам процес рендерингу на цій платформі є швидким завдяки технології Virtual DOM. Дана технологія надає можливість створювати та зберігати DOM у пам”яті і таким чином, коли потрібно обновити якусь конкретну частину, іде порівняння між Virtual DOM та оригінальним - і при визначеній зміні компоненту, міяється лише частина, яка змінилась , а не ціла сторінка, що дозволяє її не перезагружати і створювати еффект “реактивності” роботи додатку. Проте незважаючи на всі переваги даної платформи - усі маніпуляції з компонентами йдуть на клієнтській стороні у браузері, при завантаженні сторінки, що є не дуже оптимальним способом, навіть при використовуваннітехнології Vitual DOM. Саме тому клієнтська частина була розроблена у зв’язці React.js + Next.js.Next.js потрібен же для того, щоб “збирати” (білдити) сторінки на стороні вбудованого в платформу серверу - вебсерверу і користувачу уже показується готовий резльутат. Дана технологія має назву SSR (Server Side Rendering) – у контексті мови Javascript, це передзавантаження Javascipt додатків на веб-сервер і відсилання готових HTML

сторінок як відповідь на запит браузера, на той чи інший шлях (роут). Отже коли, було визначено технології з якими була розроблена клієнтська частина, після аналізу архітектури та усіх основних процесів у 3-х основних компонентів бізнес додатку було побудовано загальну схему системи, яка була розроблена протягом написання магістерьскої роботи.

2.1. Проектування клієнтської частини

Таким чином після визначення усіх елементів додатку діаграма побудови та його роботи буде виглядати так:

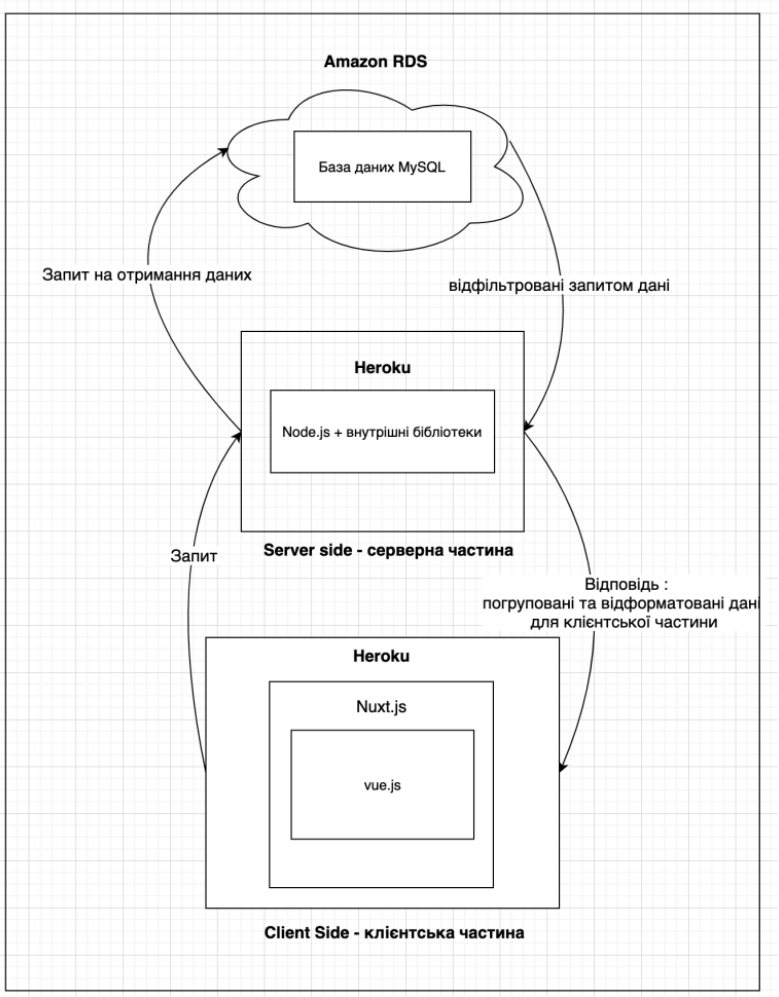


Рис. 2.1 Діаграма роботи додатку

Далі буде доречно розглянути структуру файлової системи клієнтської частини додатку

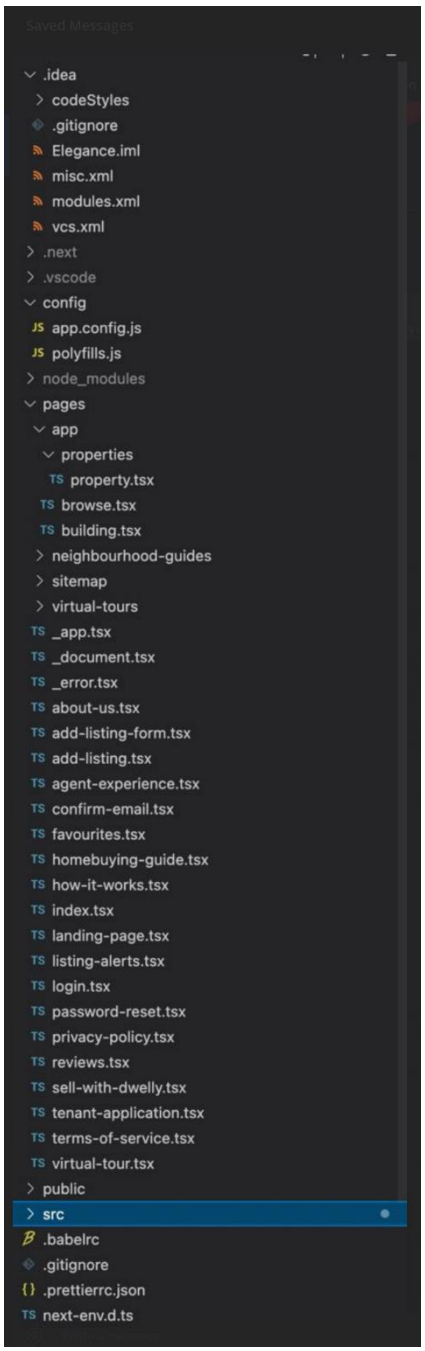


Рис. 2.2 Файлова структура клієнтської частини

Основною робочою директорією є pages - у ній лежать усі необхідні компоненти із розширенням *.tsx -оскільки використовується Typescript для об’єктно-орієнтового підходу до розробки, що дозволяє структуризувати та стандартиризувати підхід до розробки.

Сам веб-сервер, які відповідає за обробку сторінок на серверній стороні

Рис. 2.3 Структура веб-серверу для повернення сторінок на клієнт.

Отже, перейдемо до того, як побудовані самі компоненти

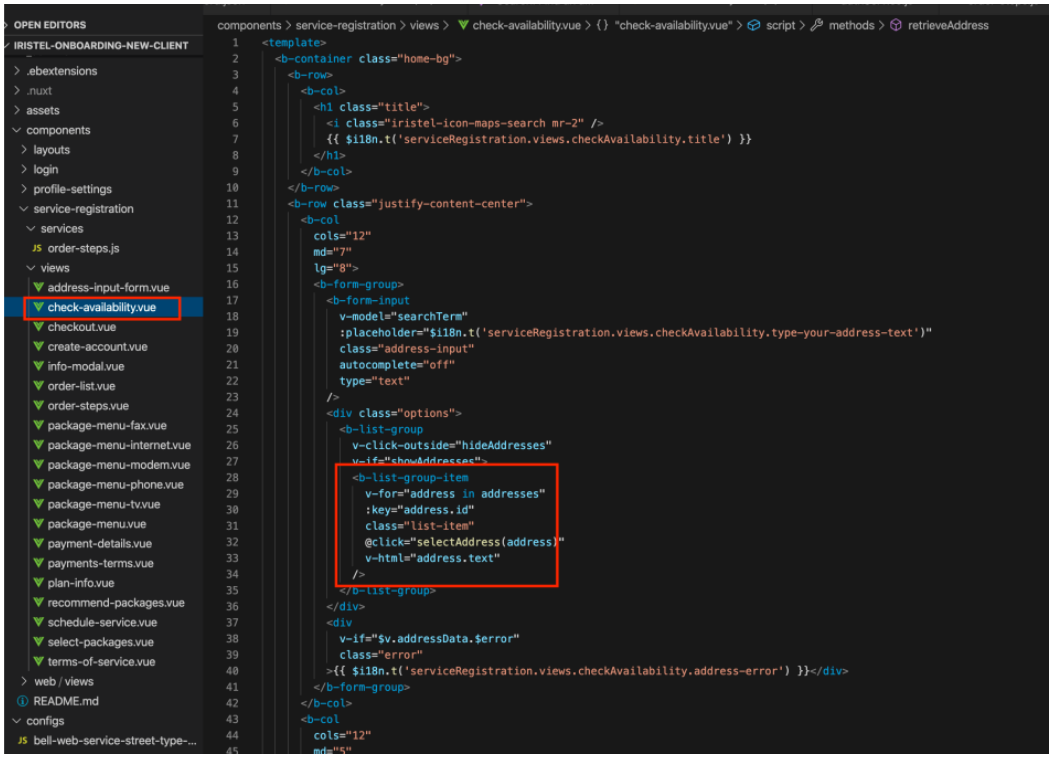


Рис. 2.4 Розмітка клієнтської частини клієнту

```
<script>
import { appLogger } from '@/_common/helpers/appLogger'
import { required, minLength, sameAs } from 'vuelidate/lib/validators'
import service from '@/components/service-registration/services/order-steps'
import addressInput from '@/components/service-registration/views/address-input-form'
import _ from 'lodash'
import ClickOutside from 'vue-click-outside'
import canadianPostAPIStreetTypes from '@/configs/canadian-post-street-type-map'
import bellAPIStreetTypes from '@/configs/bell-web-service-street-type-map'
import * as errorCodes from '@/_common/errorCodes'

export default {
  name: 'CheckAvailability',
  directives: {
    ClickOutside
  },
  components: {
    addressInput
  },
  data() {
    return {
      province: [],
      provinceTaxData: [],
      addressData: '',
      searchTerm: '',
      dataIsLoading: false,
      addresses: [],
      showAddresses: false,
      doSearch: true
    }
  },
  watch: {
    searchTerm: _.debounce(function(addr) {
      if (addr == '') {
        this.addressData = ''
      }
      this.getAddresses(addr)
    }, 500)
  },
  validations: {
    addressData: {
      required
    }
  },
  methods: {
    selectAddress: function(item) {
      if (item.next == 'Find') {
        this.getAddresses(item.text, item.id)
```

Рис. 2.5 Вигляд програмної частини клієнту з обчисленнями

Після того як проведена робота по побудові усіх сторінок перейдемо до огляду логіки роботи клієнту і його запитів до серверної частини

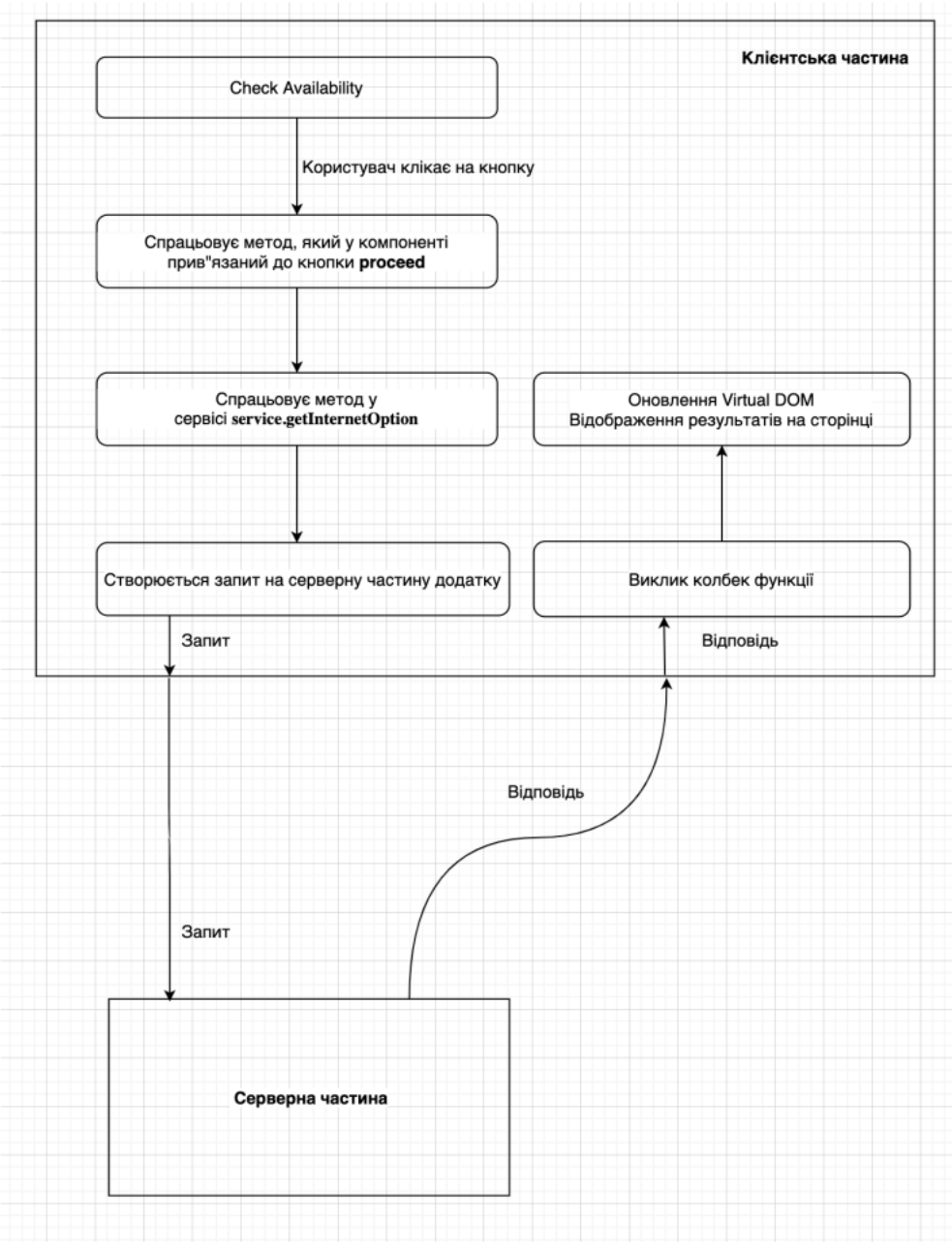


Рис. 2.6 Вигляд робочого циклу клієнтської частини

2.2. Графічні форми інтерфейсу

Дана схема буде актуальна для усіх подальших кроків робочого цилу, тому можна продовжити огляд функціоналу додатку. Таким чином після огляду усіх основних частин, варто ознайомитись, як будуть виглядати усі компоненти візуально

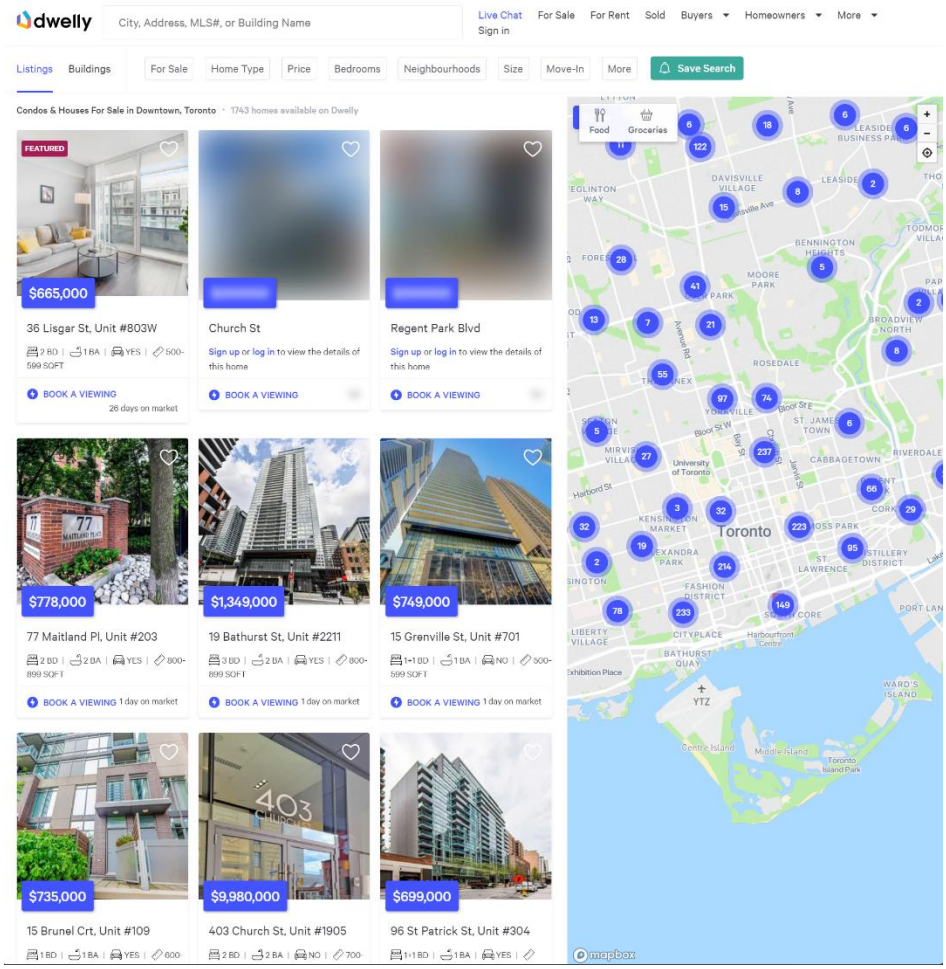


Рис. 2.7 Вигляд інтерактивної карти із квартирами.

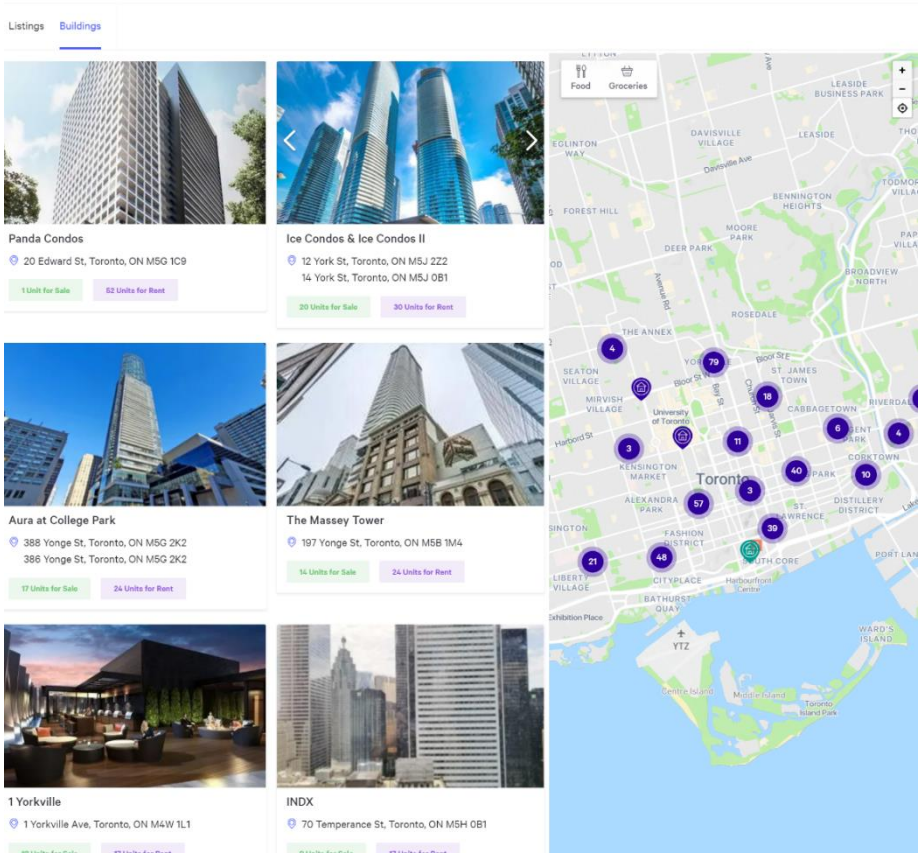


Рис. 2.8 Вигляд інтерактивної карти із будинками

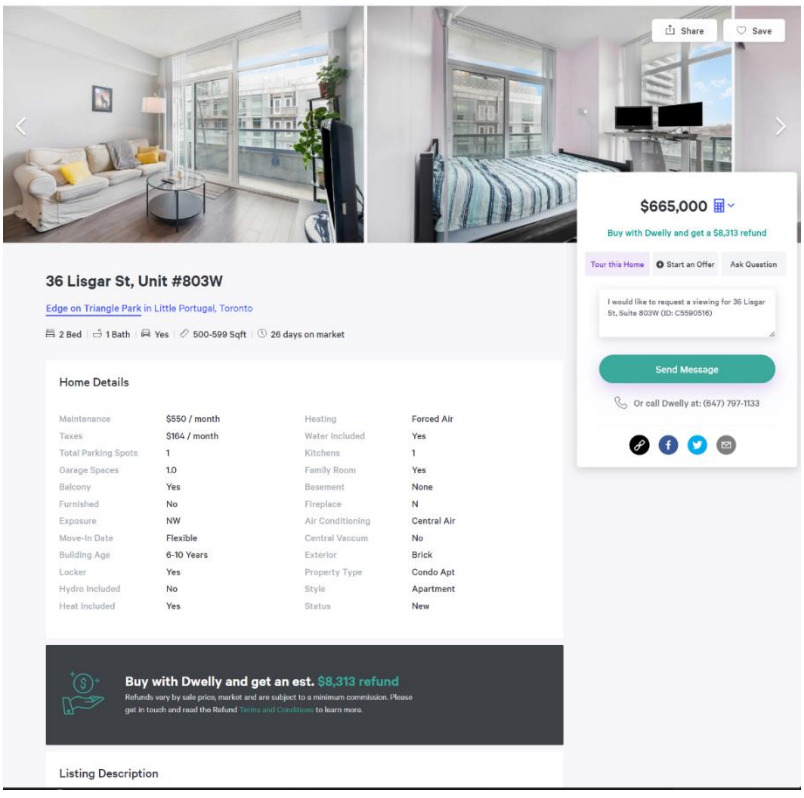


Рис. 2.9 Вигляд сторінки для відображення деталей квартири

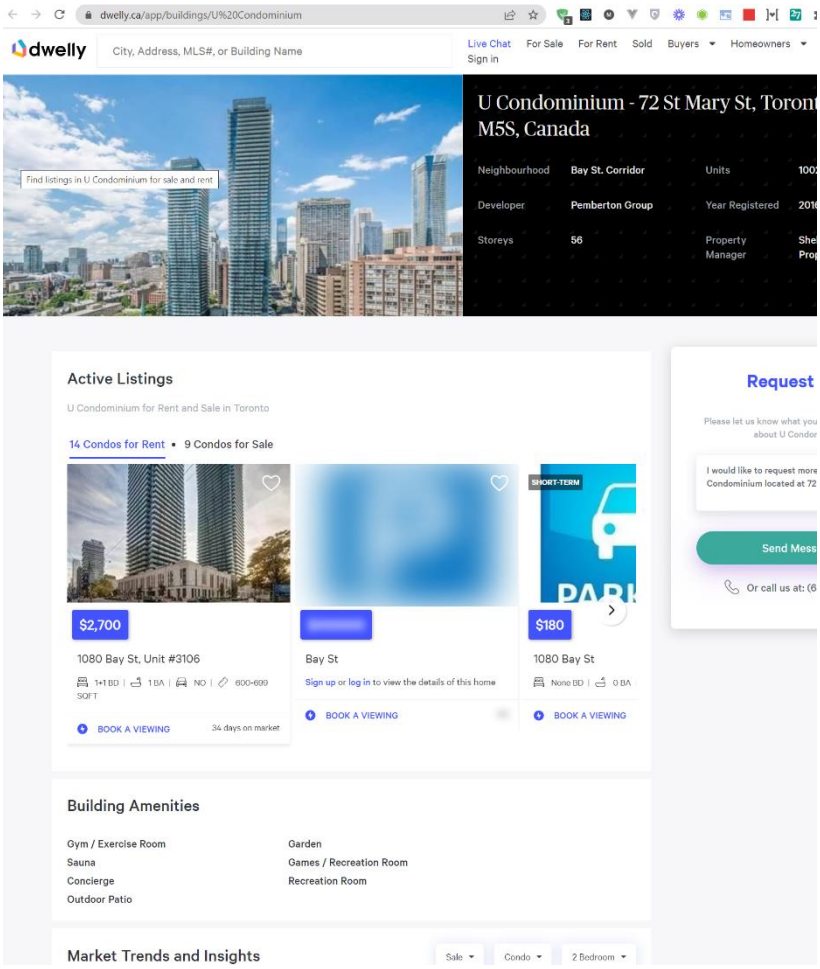


Рис. 2.10 Вигляд сторінки з деталями для будинку

Як пояснювалось у Розділі 1 - будинки це сутності у яких знаходяться квартири, тому на сторінці ми можемо ознайомитись, які з цих квартир належать до будинку. Зображення для будинку ми генеруємо автоматично на основі API, яке було інтегровано в серверну частину.

Також є можливість на основі даних по будинку за рік побудувати приємну для користувача статистику, щоб він міг більш наглядно зрозуміти цінність квартир у даному будинку тощо.

Процес розгортання клієнтської частини додатку є схожим до процесу розгортання серверної частини.

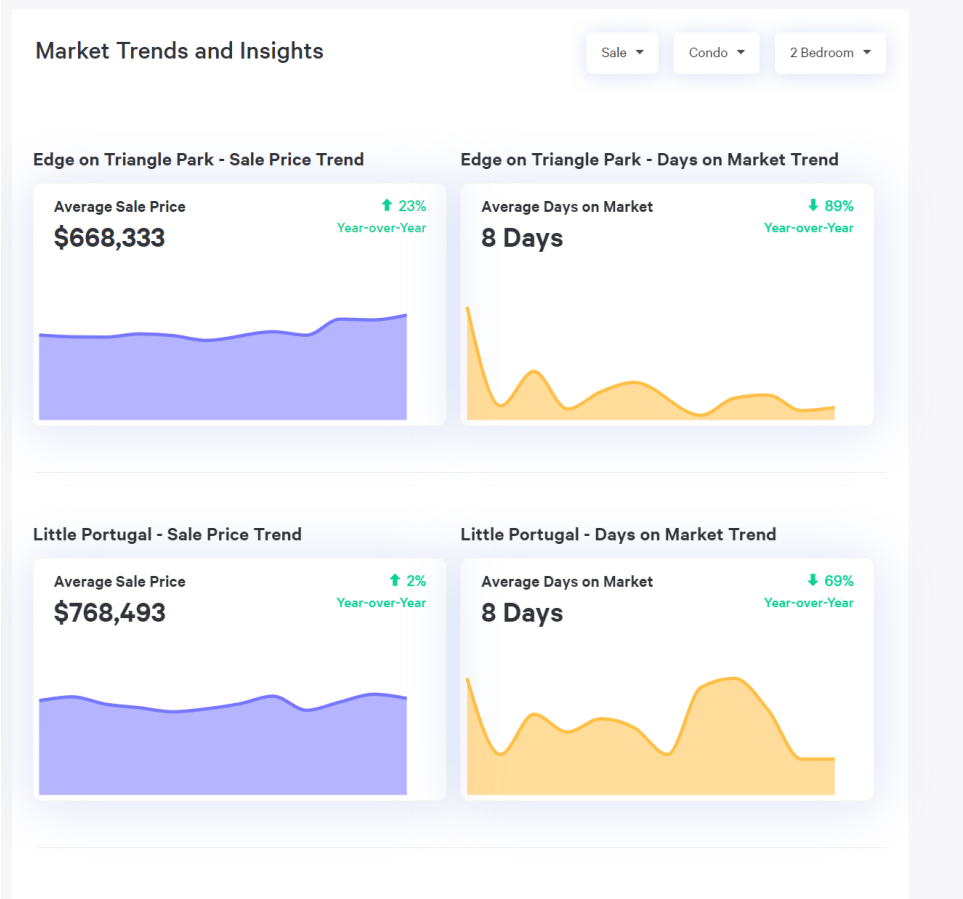


Рис. 2.11 Статистика по будинку.

Єдина відмінність, що до розгорнутого додатку у користувача є публічний доступ.

Висновки до розділу 2

У даному розділі було спроектовано клієнтську частину застосунку, надано приклади типових інтерфейсів, проаналізовано основні характеристики та продемонстровано приклади роботи.

ВИСНОВКИ

Розробка комплексних бізнес рішень є достатньо складним завданням, оскільки навіть для досвіченного розробника є важливим побудвати таку архітектуру, яка буде легко розширюємою і водночас може піддаватись високій нагрузці. Це все потрібно для того, щоб у разі потреби додавання нового контенту у додаток не займало багато часу.

Також була проведена робота по роботі з сучасними технологіями і розробкою власної архітектури та моделі для нейронної мережі і вдалось її обгорнути у абстракційний слой у вигляді API, щоб розробинкам, які будуть працювати з продуктом надалі було достатньо просто його розширювати та оновлювати.

Виконана робота показує перспективність автоматизації систем агрегаторів нерухмості, оскільки таким чином кінцевий користувач має виконати лише декілька кроків, щоб обрати нерухомість, що у наш насичений час є надзвичайною перевагою серед конкурентів.

Підсумовуючи, при розробці був проведений глубокий аналіз сучасний технологій та можливостей і це дало змогу обрати найбільш зручні та зрозумілі інструменти для розробки, що також може бути використано у практичних цілях

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming. San Francisco, 2019. – 474 p.
2. Learning Vue.js 2 by Lt Dornan3. MySQL Cookbook, 3rd Edition by Paul DuBoiis. Режим доступу: https://madewithvuejs.com/books5.
3. Getting Started with MySQL. – [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/mysql-getting-started/en/

4. Amazon Relational Database Service Guide. – [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/ru/rds/7.

5. CMA script 141 reference. – [Електронний ресурс]. – Режим доступу: https://javascript.info/8.

6. Advanced Guide to Vue.js – [Електронний ресурс]. – Режим доступу: https://ru.vuejs.org/v2/guide/

7. Developing Relational Databases. – [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/questions/3855127/dbs-mysql

8. Class composition in Node js – [Електронний ресурс]. – Режим доступу: https://alligator.io/js/class-composition/

9. Artificial Neural Networks as Models of Neural Information Processing // Frontiers Research Topic. Retrieved 2018-02-20.

10. Warren S.M., Pitts W. A Logical Calculus of Ideas Immanent in Nervous Activity // Bulletin of Mathematical Biophysics. 5 (4),1990. – P. 115–133.

11. Голубев Ю. Ф. Нейросетевые методы в мехатронике. — М.: Изд-во Моск. унта, 2007. — 157 с.

12. Петров А. П. О возможностях персептрона // Известия АН СССР, Техническая кибернетика. — 1964. — № 6.

13. Rumelhart D.E., Hinton G.E., Williams R.J., Learning Internal Representations by Error Propagation. In: Parallel Distributed Processing, vol. 1, pp. 318—362. Cambridge, MA, MIT Press. 1986.

14. Інтернет ресурс: http://itcm.comp-sc.if.ua/2017/Sineglazov.pdf

15. Інтернет ресурс: http://jmlda.org/papers/doc/2011/no1/Rudoy2011Selection.pdf

16. Рудой Г. И. Выбор функции активации при прогнозировании нейронными сетями Г. И. Рудой Московский физико-технический институт, ФУПМ, каф. «Интеллектуальные системы».

ДОДАТОК 1 SOFTWARE ARCHITECTURE DOCUMENT

Taras Shevchenko National University of Kyiv

Interior/Exterior

Image recognition

Software Architecture Document (SAD) **CONTENT OWNER: Navrotskyi Maksym**

DOCUMENT NUMBER: RELEASE/REVISION: RELEASE/REVISION DATE: • 1 • v.1 • 20 April, 2022 • • •

• • •

• • •

• • •

• • •

Table of Contents

List of Figures

List of Tables

1.Documentation Roadmap

The Documentation Roadmap should be the first place a new reader of the SAD begins. But for new and returning readers, it is intended to describe how the SAD is organized so that a reader with specific interests who does not wish to read the SAD cover-to-cover can find desired information quickly and directly.

Sub-sections of Section 1 include the following.

● Section 1.1 (“Document Management and Configuration Control Information”) explains revision history. This tells you if you’re looking at the correct version of the SAD.

● Section 1.2 (“Purpose and Scope of the SAD”) explains the purpose and scope of the SAD, and indicates what information is and is not included. This tells you if the information you’re seeking is likely to be in this document.

● Section 1.3 (“How the SAD Is Organized”) explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.

● Section 1.4 (“Stakeholder Representation”) explains the stakeholders for which the SAD has been particularly aimed. This tells you how you might use the SAD to do your job.

● Section 1.5 (“Viewpoint Definitions”) explains the *viewpoints* (as defined by IEEE Standard 1471-2000) used in this SAD. For each viewpoint defined in Section 1.5, there is a corresponding view defined in Section 3 (“Views”). This tells you how the architectural information has been partitioned, and what views are most likely to contain the information you seek.

● Section 1.6 (“How a View is Documented”) explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1.1. Document Management and Configuration Control Information

● Revision Number: 0.1

● Revision Release Date: 19.04.2022

● Purpose of Revision: Initial project release

● Scope of Revision: Whole project

1.2. Purpose and Scope of the SAD

This SAD specifies the software architecture**.** All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

What is software architecture? The software architecture for a system[1]is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible” properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault

[1] Here, a system may refer to a system of systems.

handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain

information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 3.

Behavior. Although software architecture tends to focus on structural information, *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which

is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3. Viewpoint Definitions

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

As described in Section 1.2, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	**Viewpoint(s) that apply to that class of stakeholder's concerns**
Navrotskyy Maksym	Masters program student

1.3.1. Masters program student Viewpoint Definition

1.3.1.1. Abstract

This viewpoint describes the view of a student in masters program who's grade is dependant on the result of the project

1.3.1.2. Stakeholders and Their Concerns Addressed

Main concern of a student is to get their project up to university's standard and to get best possible grade

1.3.1.3. Elements, Relations, Properties, and Constraints

The software consists 2 main script that incorporates multiple modules for interior/exterior image recognition and analysis, numpy and pyplot for visual representation, and fpdf for report creation

1.3.1.4. Language(s) to Model/Represent Conforming Views

Software was made on python using C++ external modules with compiler

2.Architecture Background

2.1. Problem Background

The project has a n image processing system that specifies whether the image is exterior or interior, so the user has a proper image of the building itself.

2.1.1. System Overview

This software takes a image and predicts is class (whether interior or exterior)

2.1.2. Goals and Context

The goal is to create software that helps to recognise whether image is interior or exterior, for proper use cases

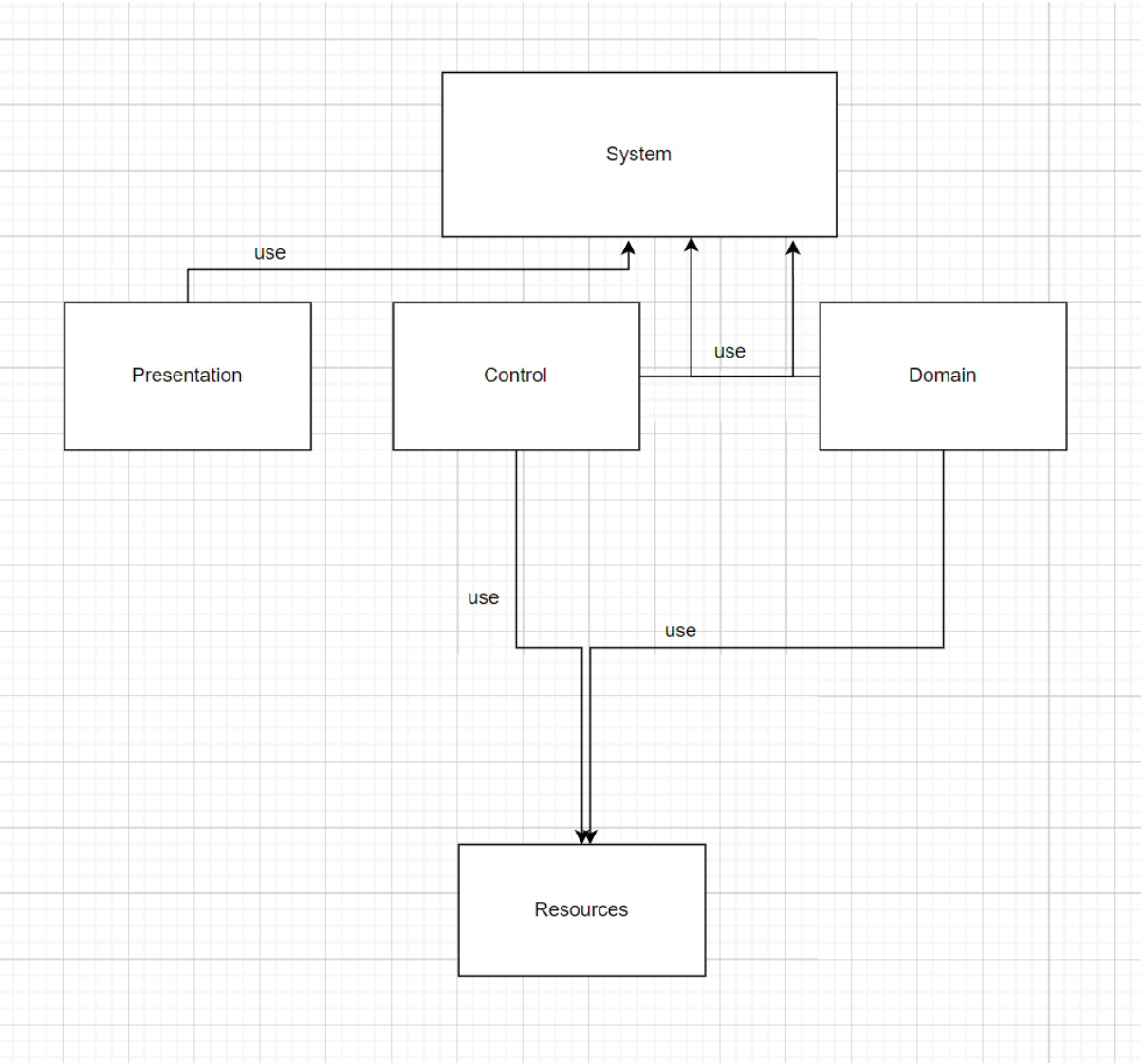


Figure 2: Software's layered architecture diagram

2.1.3 Elements catalog

Presentation Layer. Deals with presentation logic and the pages rendering. Control Layer. Manages the access to the domain layer.

Domain Layer. Related to the business logic and manages the accesses to the resources layer.

Resources Layer. Responsible for the access to the enterprise information system (database, data warehouse, storages etc.)

Shared Layer. Gathers the common objects reused through all the layers.

2.1.4. Significant Driving Requirements

- Implementation of multiple different modules
- Single script does all (AIO)
- Data aggregation

2.2. Solution Background

2.2.1. Architectural Approaches

The AIO architecture was chosen due to relatively small amount of data that needs to be processed and the analytical nature of the project

2.2.2. Analysis Results

After development in testing this architecture proved to be sufficient for the set task

2.2.3. Requirements Coverage

- Python interpreter
- C++ development kit
- PDF reader

3.Referenced Materials

Barbacci 2003	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*, Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.ht ml>.
Bass 2003	Bass, Clements, Kazman, *Software Architecture in Practice*, second edition, Addison Wesley Longman, 2003.
Clements 2001	Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison Wesley Longman, 2001.
Clements 2002	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002.

IEEE 1471 Signal processing methods for music transcription, chapter 5. Springer Science & Business Media.	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000. Klapura, A., and Deivi, M. (Ed.). (2007).

4.Directory

4.1. Index

4.2. Glossary

Term Definition
software architecture The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.

view A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint. view packet The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets. viewpoint A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

4.3. Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common object request broker architecture
COTS	Commercial-Off-The-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
OO	Object Oriented
ORB	Object Request Broker
OS QAW	Operating System Quality Attribute Workshop

RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE SEI	Software Engineering Environment Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG SLOC	Software Engineering Process Group Source Lines of Code
SW-CMM	Capability Maturity Model for Software
CMMI-SW	Capability Maturity Model Integrated - includes Software Engineering
UML	Unified Modeling Language

5. Sample Figures & Tables