**Міністерство освіти і науки України**

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних технологій та програмної інженерії

**ЗВІТ**

про виконання розрахунково-графічних робіт

з дисципліни:

«Проектування геоінформаційних систем»

Виконав:

ЗВО ПІ-181 Мамчуровський В. С.

Керівник:

Доцент, к.т.н. Білоус І. В.

Чернігів НУЧП 2021

**Розрахунково-графічна робота №1**

**Створення web-сервісу для обробки геореляційних даних**

**Мета роботи:** створити web-сервіс, що реалізує обробку геореляційних даних; реалізувати виклики методів web-сервісу у web-додатку; протестувати web-сервіс та web-додаток в цілому.

## Хід роботи

У рамках даної розрахунково-графічної роботи для створення web-сервісу, який реалізує обробку геореляційних даних було обрано мову програмування Python, середовище розробки Jupyter Notebook і СУБД PostgreSQL 13.

### Створення бази даних та завантаження даних

Спочатку створили нову базу даних «ukraine\_db», після чого надали їй розширення postgis. Для завантаження даних з [shp-файлів](https://gadm.org/download_country.html) до новоствореної бази даних використали утиліту “PostGIS Shapefile Import/Export Manager”: налаштували з’єднання, додали та імпортували файли (рисунок 1.1).

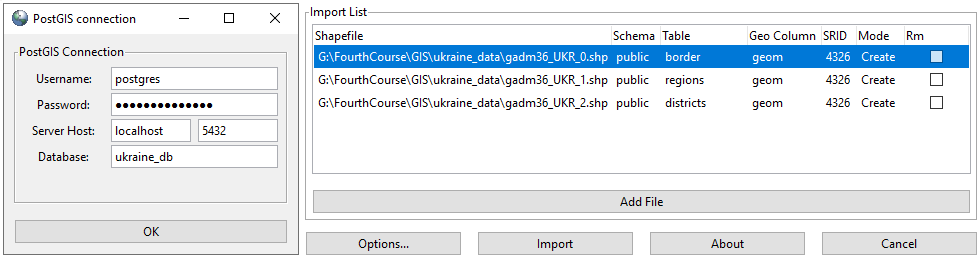


Рисунок 1.1 – Завантаження даних до БД “ ukraine\_db”

### Функція знаходження центру географічних об’єктів

Створили функції обробки геореляційних даних, які були протестовані на попередніх лабораторних роботах. Першою з них стала збережена процедура *find\_centroids*, що виконує розрахунок центру кожного з географічних об’єктів вказаного шару, а також записує отримані результати до таблиці відповідної таблиці. SQL-запит на створення таких таблиць та просторового індексу для атрибута, що зберігає географічні дані об’єкта, наведено нижче у лістингу 1.1, текст функції *find\_centroids* – у лістингу 1.2, а результати тестування – на рисунку 2.1.

Лістинг 1.1 – Створення таблиць *districts\_centroids* і *regions\_centroids*

|  |
| --- |
| CREATE TABLE IF NOT EXISTS public.districts\_centroids (  id\_centroid serial NOT NULL,  centroid\_geom geometry NOT NULL,  centroid\_text text NOT NULL,  id\_district bigint NOT NULL,  CONSTRAINT districts\_centroid\_fk FOREIGN KEY (id\_district)  REFERENCES public.districts (gid) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE  );  CREATE INDEX centroids\_geom\_index ON public.districts\_centroids  USING GIST (centroid\_geom);  CREATE TABLE IF NOT EXISTS public.regions\_centroids (  id\_centroid serial NOT NULL,  centroid\_geom geometry NOT NULL,  centroid\_text text NOT NULL,  id\_region bigint NOT NULL,  CONSTRAINT regions\_centroids\_fk FOREIGN KEY (id\_region)  REFERENCES public.regions (gid) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE  );  CREATE INDEX regions\_centroids\_geometry\_index ON  public.regions\_centroids USING GIST (centroid\_geom); |

Лістинг 1.2 – Процедура розрахунку центру об’єкту вказаного шару

|  |
| --- |
| CREATE OR REPLACE PROCEDURE find\_centroids(layer text)  AS $body$  BEGIN  IF layer = 'districts' THEN  INSERT INTO districts\_centroids(centroid\_geom,  centroid\_text, id\_district)  SELECT ST\_Centroid(geom), ST\_AsText(ST\_Centroid(geom)), gid  FROM districts;  ELSIF layer = 'regions' THEN  INSERT INTO regions\_centroids(centroid\_geom, centroid\_text,  id\_region)  SELECT ST\_Centroid(geom), ST\_AsText(ST\_Centroid(geom)), gid  FROM regions;  END IF;  END;  $body$  LANGUAGE plpgsql;  CALL find\_centroids('districts');  CALL find\_centroids('regions'); |

### Побудова найкоротшого маршруту між об’єктами

Далі перейшли до створення функції *build\_route*, яка будує та повертає найкоротший маршрут через довільну кількість об’єктів, що задаються користувачем. Шлях прокладається між центрами об’єктів, до яких належать задані точки. Для знаходження таких об’єктів і їх географічних центрів створили функцію *get\_polygon\_centre*, до якої у якості параметрів передаються дані про точку та рівень, на якому шукаємо об’єкт.

Текст функції *get\_polygon\_centre* наведено нижче у лістингу 1.3, а функції *build\_route* – у лістингу 1.4.

Лістинг 1.3 – Функція *get\_polygon\_centre*

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_polygon\_centre(p geometry, l text)  RETURNS geometry  AS $body$  BEGIN  IF l = 'districts' THEN  RETURN (SELECT dc.centroid\_geom  FROM districts d, districts\_centroids dc  WHERE d.gid = dc.id\_district  AND ST\_Contains(d.geom, p));  ELSIF l = 'regions' THEN  RETURN (SELECT rc.centroid\_geom  FROM regions r, regions\_centroids rc  WHERE r.gid = rc.id\_district AND  ST\_Contains(r.geom, p));  END IF;  END;  $body$  LANGUAGE plpgsql; |

Лістинг 1.4 – Функція *build\_route*

|  |
| --- |
| CREATE OR REPLACE FUNCTION build\_route(points\_string text, layer text)  RETURNS TABLE(route\_geometry geometry, route\_json text)  AS $body$  DECLARE  centre geometry;  geom\_point geometry;  geom\_points geometry[] := ARRAY(  SELECT ST\_PointN(column1, generate\_series(1, ST\_NPoints(column1)))  FROM (VALUES(ST\_GeomFromText(points\_string, 4326))) AS foo);  centre\_points geometry[];  BEGIN  FOREACH geom\_point IN ARRAY geom\_points LOOP  centre := get\_polygon\_centre(geom\_point, layer); |

Лістинг 1.4 – Функція *build\_route* (продовження)

|  |
| --- |
| IF centre IS NOT NULL THEN  centre\_points := centre\_points || centre;  ELSE  RAISE EXCEPTION 'Точка вне объекта!';  END IF;  END LOOP;  RETURN QUERY (SELECT ST\_MakeLine(centre\_points) AS route\_geometry,  ST\_AsGeoJSON(ST\_MakeLine(centre\_points)) AS route\_json);  END;  $body$  LANGUAGE plpgsql; |

### Створення 2D сітки для визначеного полігону

Створили функцію *create\_grid\_2d*, до якої в якості параметрів передаються просторові межі полігону у текстовому форматі, а також висота та ширина комірки сітки у метрах. Функція повертає єдине значення типу geometry, яке містить в собі всі комірки, які перетинаються з заданим полігоном. Текст функції наведено нижче у лістингу 1.5.

Лістинг 1.5 – Функція *create\_grid\_2d*

|  |
| --- |
| CREATE OR REPLACE FUNCTION create\_grid\_2d(Bound\_polygon\_text text,  Cell\_width integer, Cell\_height integer)  RETURNS TABLE(geom geometry)  AS $body$  DECLARE  Bound\_polygon geometry := ST\_GeomFromText(Bound\_polygon\_text,  4326);  Xmin DOUBLE PRECISION; Xmax DOUBLE PRECISION;  Ymax DOUBLE PRECISION;  X DOUBLE PRECISION; Y DOUBLE PRECISION;  NextX DOUBLE PRECISION; NextY DOUBLE PRECISION;  CPoint geometry; Sectors geometry[]; New\_cell geometry;  i INTEGER; SRID INTEGER;  BEGIN  Xmin := ST\_XMin(Bound\_polygon); Xmax := ST\_XMax(Bound\_polygon);  Ymax := ST\_YMax(Bound\_polygon); SRID := ST\_SRID(Bound\_polygon);  Y := ST\_YMin(Bound\_polygon); -- координата угла текущего сектора  i := -1;  <<yloop>>  LOOP  IF (Y > Ymax) THEN  EXIT;  END IF;  X := Xmin;  <<xloop>>  LOOP |

Лістинг 1.5 – Функція *create\_grid\_2d* (продовження)

|  |
| --- |
| IF (X > Xmax) THEN  EXIT;  END IF;  CPoint := ST\_SetSRID(ST\_MakePoint(X, Y), SRID);  NextX := ST\_X(ST\_Project(CPoint, Cell\_width,  radians(90))::geometry);  NextY := ST\_Y(ST\_Project(CPoint, Cell\_height,  radians(0))::geometry);  New\_cell := ST\_MakeEnvelope(X, Y, NextX, NextY, SRID);  -- Проверка пересечения ячейки и заданного полигона  IF ST\_Intersects(New\_cell, Bound\_polygon) THEN  i := i + 1;  sectors[i] := ST\_MakeEnvelope(X, Y, NextX,  NextY, SRID);  END IF;  X := NextX;  END LOOP xloop;  CPoint := ST\_SetSRID(ST\_MakePoint(X, Y), SRID);  NextY := ST\_Y(ST\_Project(CPoint, Cell\_height,  radians(0))::geometry);  Y := NextY;  END LOOP yloop;  RETURN QUERY (SELECT ST\_Collect(Sectors) AS geom);  END;  $body$  LANGUAGE 'plpgsql'; |

На цьому завершили налаштування таблиць та збережених функцій БД.

### Імпорт необхідних бібліотек

Далі перейшли до середовища розробки Jupyter Notebook. Перш за все виконали імпорт основних бібліотек, класів та функцій, які в подальшому були використані для реалізації сервісу. Для кожного імпорту у коментарі поруч коротко описали їх необхідність (рисунок 1.2).

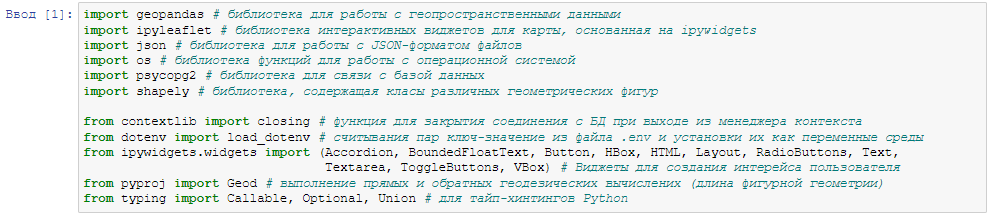


Рисунок 1.2 – Імпорт необхідних бібліотек

### Створення інтерфейсу. Карта

Створили об’єкт інтерактивної мапи: в якості базової мапи визначили open-source мапу OpenStreetMap, встановили значення стартової до показу локації, поточного та максимального зуму. Також додали панель контролю шарів, на якій можна «приховувати» певні шари, можливості відкриття мапи у повноекранному режимі та перегляду поточного масштабу (рисунок 1.3).

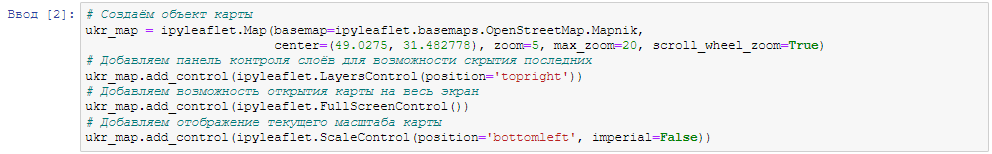


Рисунок 1.3 – Створення об’єкту мапи та додаткових панелей керування

### Створення інтерфейсу. Групи керування

Для зручності користування додатком створили так звані групи керування: побудова маршруту (дві кнопки та поле виводу довжини шляху), створення 2D сітки (два активних числові поля ширини та висоти комірки, дві кнопки), перегляд повідомлень про помилки (текстове поле та кнопка для очищення поля). Текст створення всіх елементів груп керування та їх компонування наведено нижче на рисунку 1.4.



Рисунок 1.4 – Створення груп керування та їх компонування

### Робота з мапою. Завантаження та додавання шарів

У якості користувацьких шарів додали наступні: «Кордон України», «Області України» з даними про кордони областей, «Райони України» з даними про кордони районів й обласних центрів. Окрім цього було визначено назви та такі стильові характеристики шарів як колір, прозорість, колір при наведенні миші тощо. Також для подальшої зручності була створена функція *get\_geodataframe\_data\_from\_db*, у якій відбувається з’єднання з базою даних для отримання всіх записів вказаної у запиті таблиці (рисунки 1.5 – 1.7).

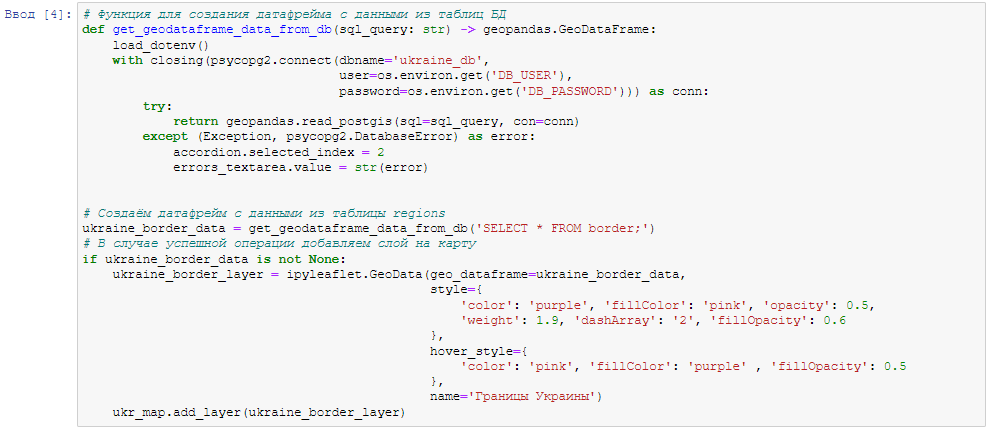


Рисунок 1.5 – Завантаження шару «Кордон України»

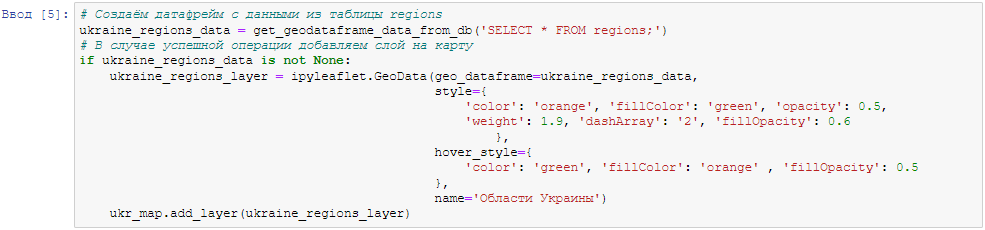


Рисунок 1.6 – Завантаження шару «Області України»

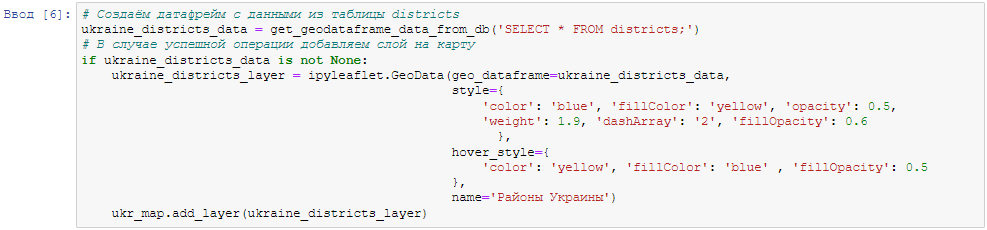


Рисунок 1.7 – Завантаження шару «Райони України»

### Робота з мапою. Створення та налаштування шару маркерів

Для зручності користувача визначати точки для побудови маршрутів між полігонами, створили кластер маркерів. Окрім цього визначили наступні функції: додавання та видалення маркерів, реакції на клік по карті, на переміщення маркера по карті та на подвійний клік по маркеру (рисунок 1.8).



Рисунок 1.8 – Створення та налаштування шару маркерів

### Робота з мапою. Реалізація шару побудованого маршруту

Для того, щоб побудувати маршрут, використали відповідну збережену функцію бази даних, яка повертає маршрут у форматі *geometry* для подальшої обробки. В рамках даного пункту створили функції-реакції при натисканні на кожну з двох кнопок “Build Route!” і “Remove Route!”, а також функції, які генерують параметр для збереженої функції, викликають її та відображають отриманий маршрут на мапі (рисунок 1.9).



Рисунок 1.9 – Реалізація шару побудованого маршруту

### Робота з мапою. Створення 2D сітки

Для того, щоб створити 2D сітку, використали відповідну збережену функцію бази даних, яка повертає полігон у форматі *geometry* для подальшої обробки. В рамках даного пункту створили функції-реакції при натисканні на кожну з двох кнопок “Create 2D Grid!” і “Remove 2D Grid!”, а також функції, які знаходять необхідний полігон відповідно до обраного рівня карти, викликають збережену функцію та відображають отриману сітку на мапі (рисунок 1.10).



Рисунок 1.10 – Створення 2D сітки

### Проведення тестування

Для тестування проекту розгорнули його на локальному сервері. Початковий вигляд інтерактивної мапи наведено на рисунку 1.11. Клієнтська частина додатку повністю наслідує функціонал попередніх лабораторних робіт. Під картою наведено кнопки перемикання, які використовуються для вибору шару, відносно об’єктів якого будуються маршрут та сітка. Далі йдуть описані у попередніх пунктах групи керування

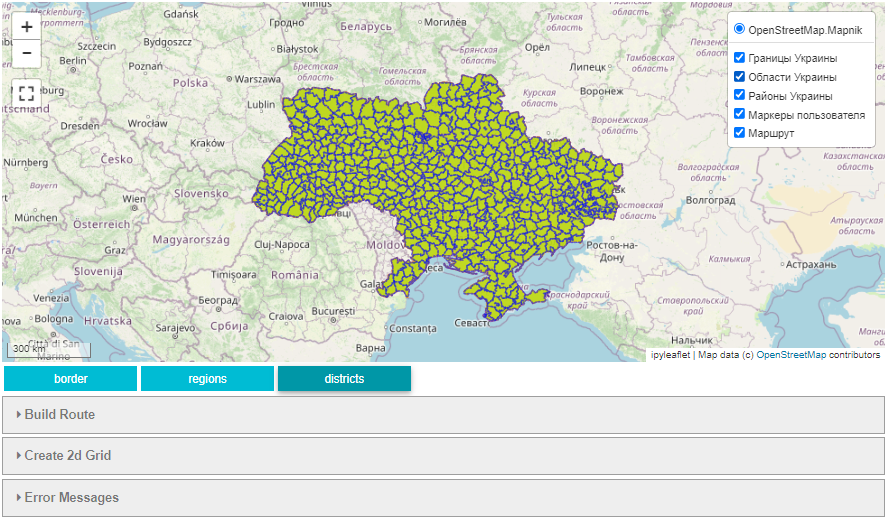


Рисунок 1.11 – Інтерфейс користувача. Початковий вигляд.

Для побудови шляху між центрами областей чи районів достатньо додати декілька маркерів шляхом натискання у будь-якій точці мапи України, а потім натиснути на кнопку “Build Route!” (рисунки 1.12 – 1.13).

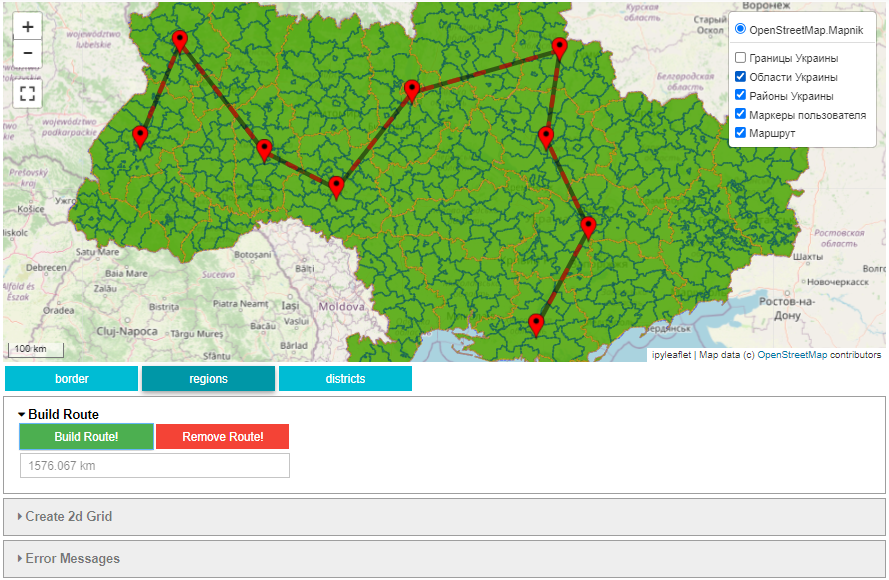
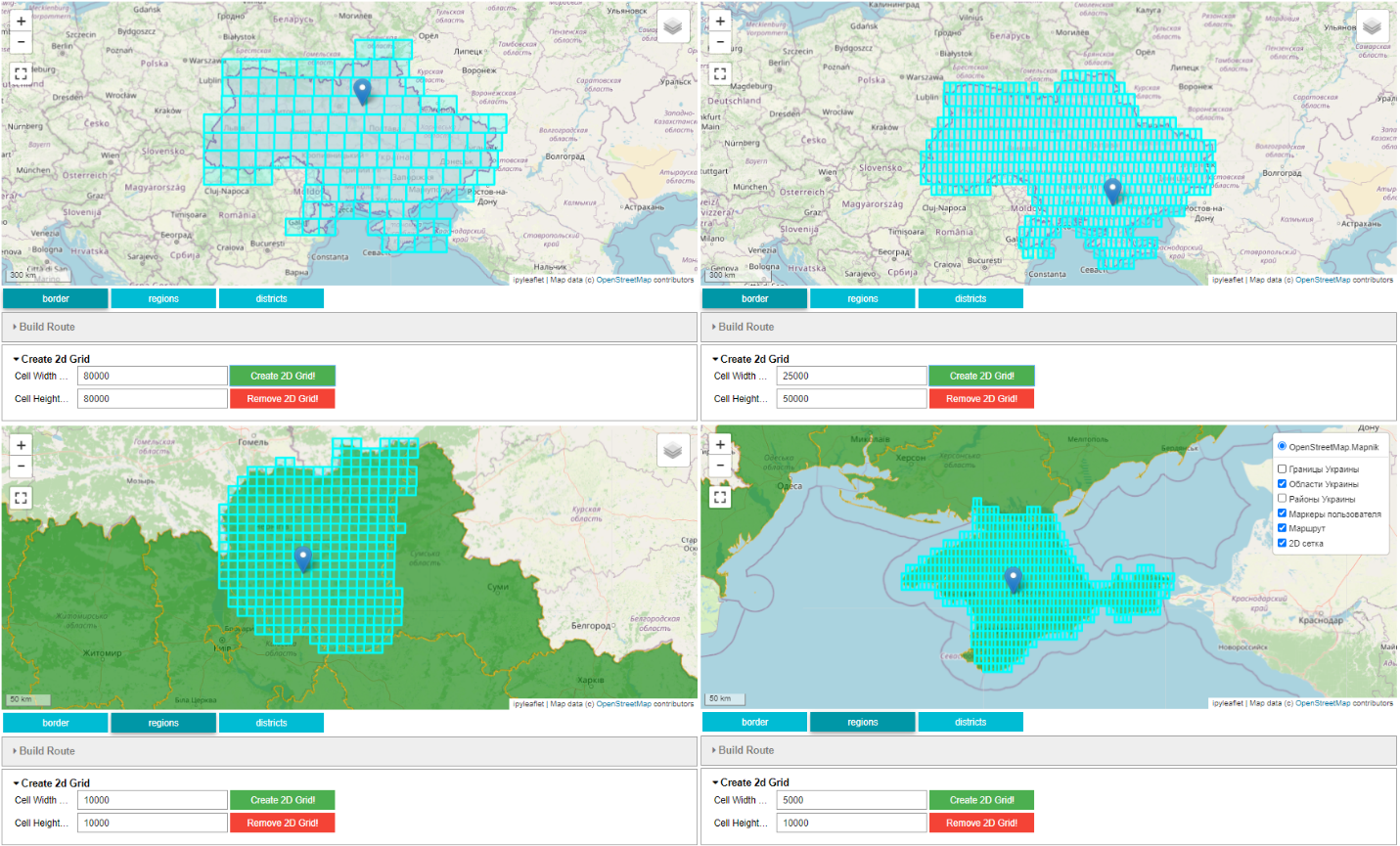


Рисунок 1.12 – Побудова маршруту (рівень «Області України»)



Рисунок 1.13 – Побудова маршруту (рівень «Райони України»)

Для побудови 2D сітки достатньо обрати шар (кордон країни, області чи району), встановити маркер в межах бажаного полігону, вказати ширину та висоту комірки, а потім натиснути на “Create 2D Grid!” (рисунки 1.14 і 1.15).

Рисунок 1.14 – Створення 2D сітки (рівні «Кордон», «Області України»)

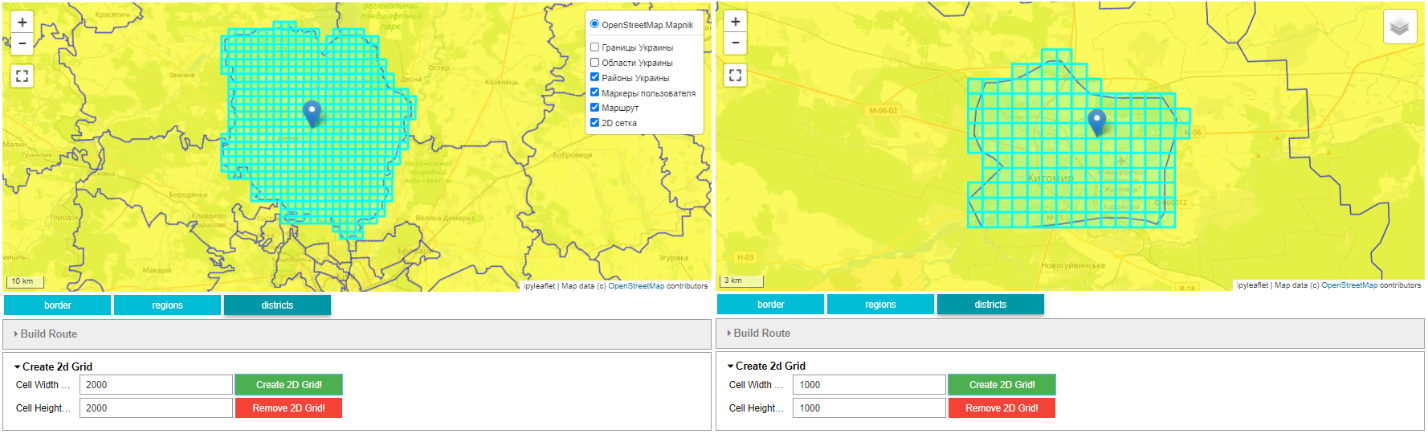


Рисунок 1.14 – Створення 2D сітки (рівень «Райони України»)

## Висновки

У рамках даної розрахунково-графічної роботи було створено та протестовано web-додаток, який реалізує обробку геореляційних даних. Для виконання завдання було використано мову програмування Python, середовище розробки Jupyter Notebook і СУБД PostgreSQL 13.