КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Спеціальність «Інформатика»

Звіт

З дисципліни «Інформаційні технології»

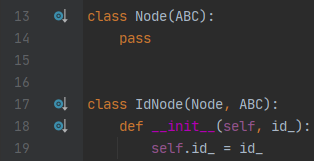
Виконав

студент групи ТТП-4

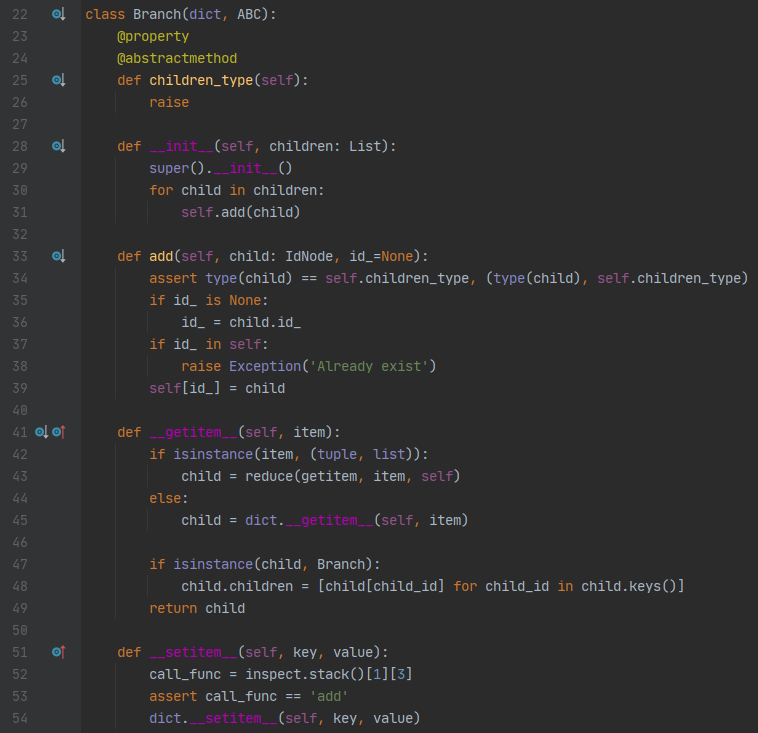
Винник Дмитро

Для створення такої структури були описані базові класи елементів дерева:

1. Вершини з/без ідентифікатора:



1. Гілки:

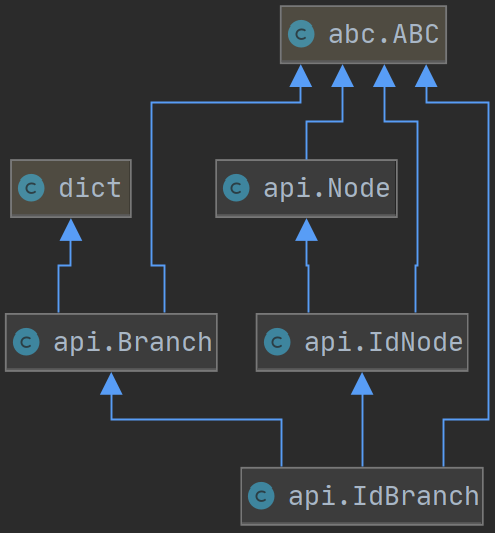


(Рядок 34 перевіряє приналежність об’єкта до типу дітей цієї гілки. Наприклад при виклику add у Base аргумент перевіриться на приналежність до Table.

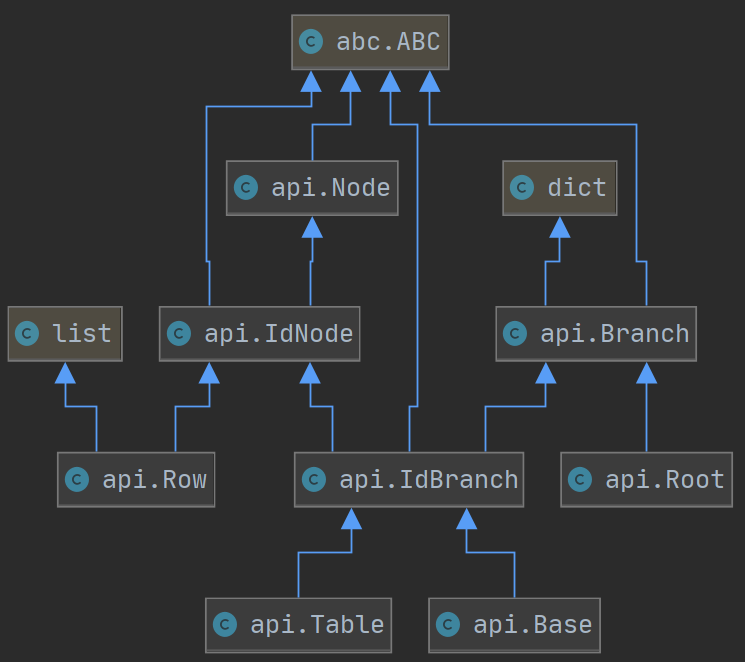
Рядки 42-43 дозволяють індексування через кому, наприклад tree[‘base\_id’, ‘table\_id’]

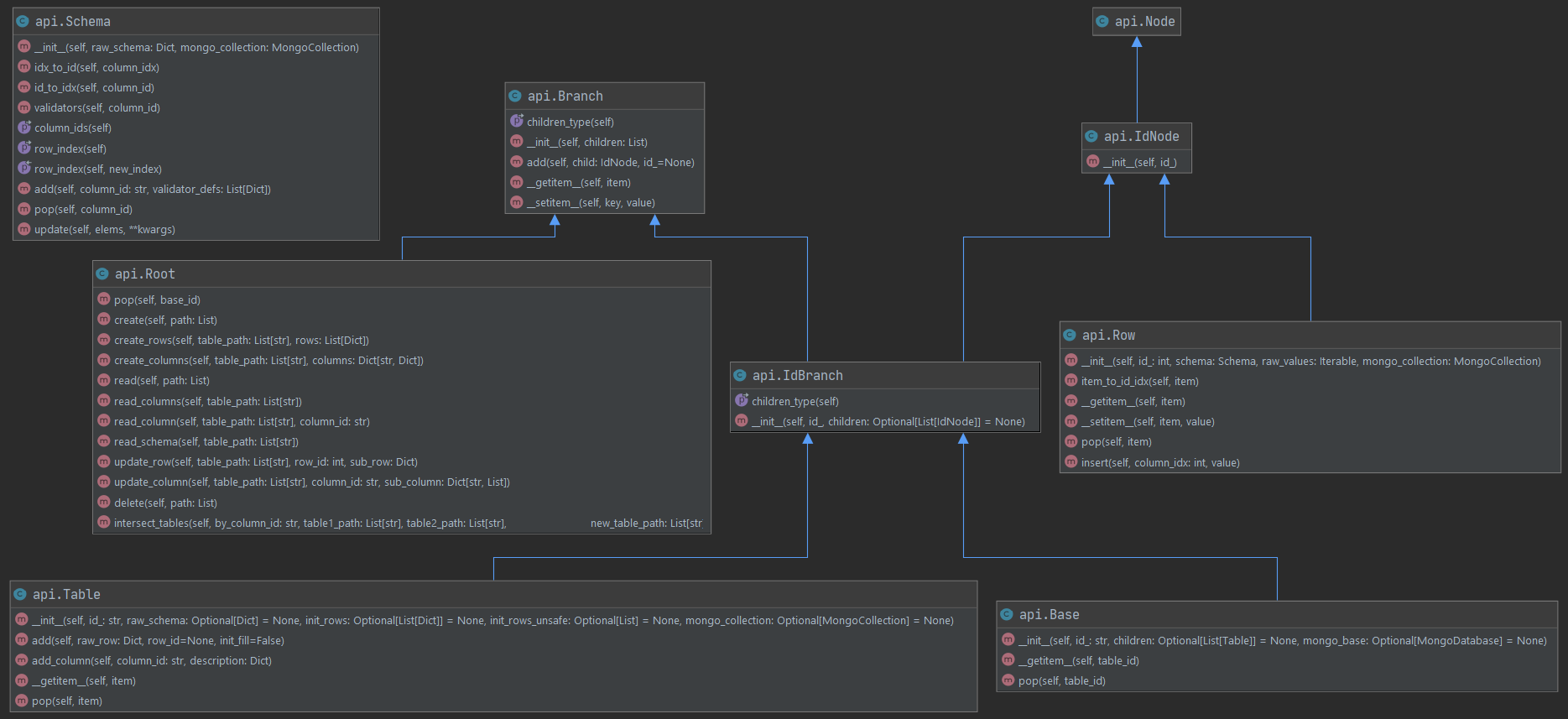
Рядки 47-48 потрібні для примусового виклику перевантаженого гетера у вкладених об’єктах)

Ось так виглядає діаграма наведених вище класів:



На основі цих класів побудовані класи Root, Base, Table, Row:



Класи, їх методи і поля (деякі базові класи прибрано):

Row наслідується від list, через зручність вставляння нового елемента у певну позицію, чого не можна легко зробити у випадку з dict. Тому окремо від рядків в об’єкті таблиці зберігається схема наступного виду:

{‘columns’: {column\_id1: column\_description1, column\_id2: column\_description2, …}, …}

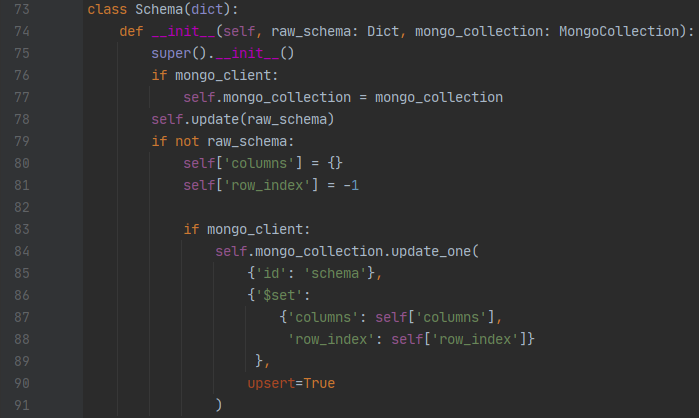
У column\_description міститься опис колонки, наприклад її валідатори.

Через глобальну змінну у модулі *api* можна вмикати і вимикати mongo:

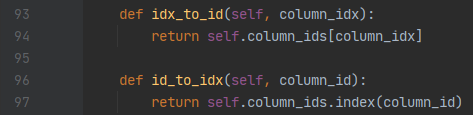


-- Якщо клієнт заданий, то зміни до бази, що в оперативній пам’яті, будуть дублюватися на mongo-базу, що на диску.

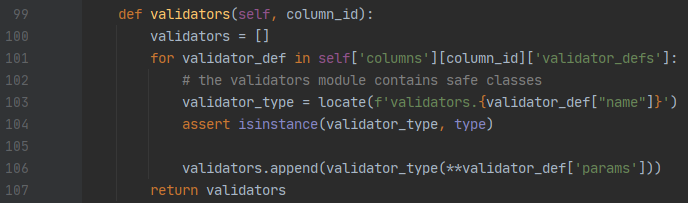
Схема створюється з, і зберігається у звичайному dict:



Перетворення цілочисельного індексу list-рядка на str назву колонки й навпаки:



Створення валідаторів колонки з їх описів (validator\_def):



(у рядку 103, у модулі validators шукається клас з ім’ям validator\_def[“name”])

Приклади опису валідаторів:





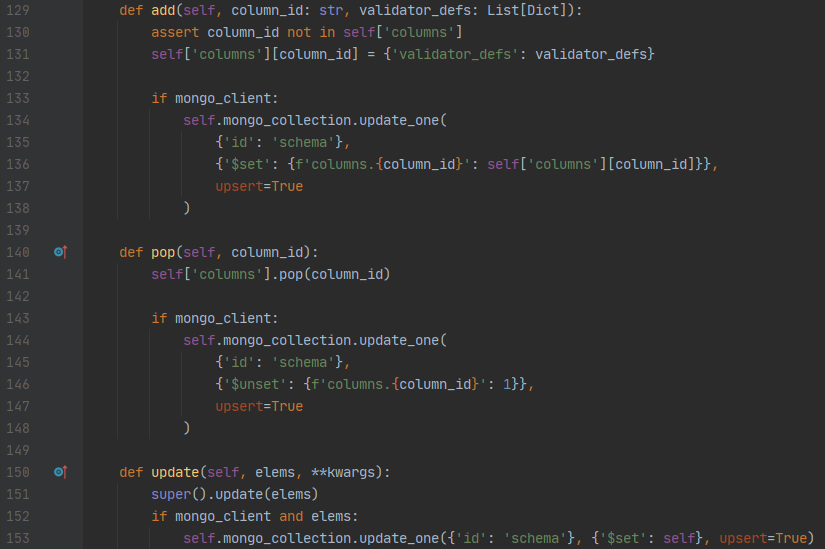
То як виглядають їх класи:



Також схема зберігає номер поточного рядка:



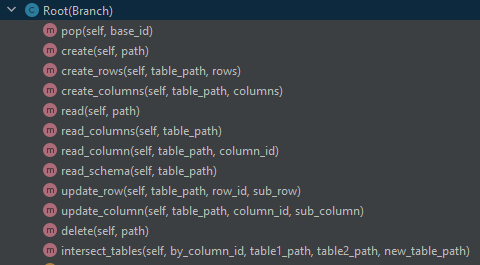
Додавання, видалення, масове додавання колонок у схему:



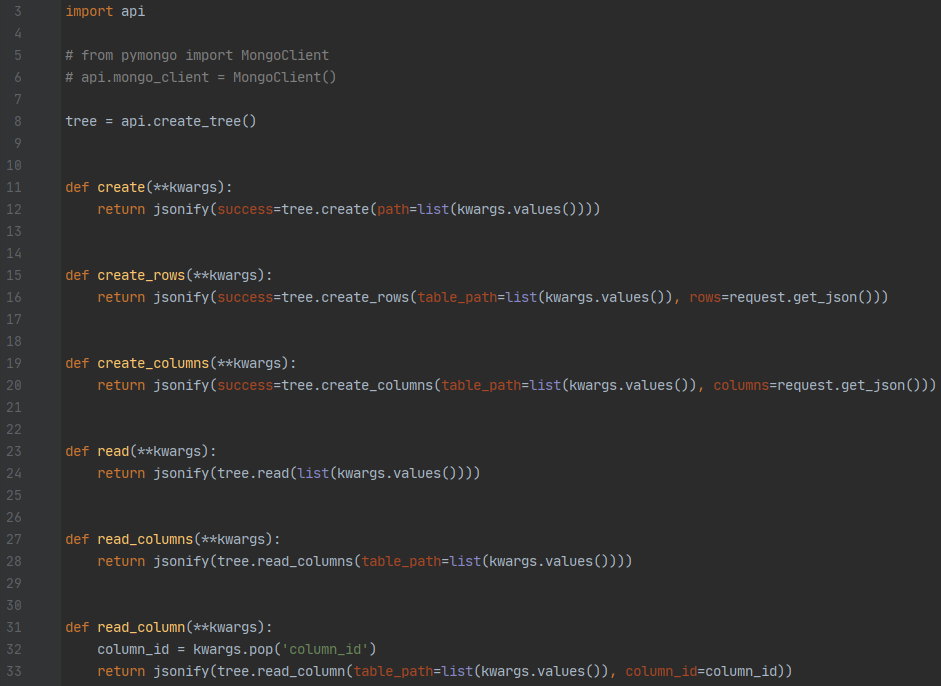
Приклад документів у mongo:



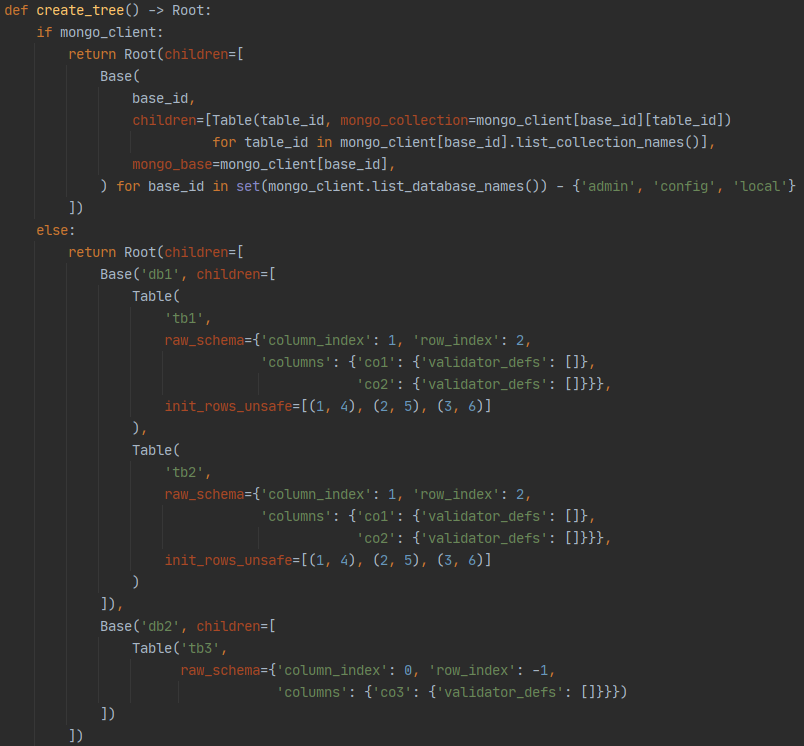
Root є керівним класом, у ньому реалізовані всі методи до яких звертаються gRPC/Flask REST/GraphQL:



Наприклад REST звертається так:



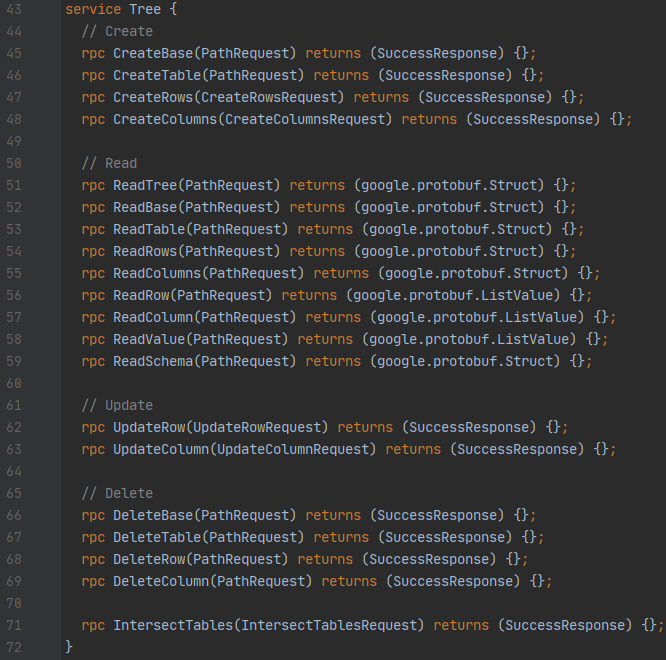
api.create\_tree з 8-го рядка відповідно до встановленого чи відсутнього клієнта завантажує дерево з mongo чи заповнює встановленими вручну значеннями:



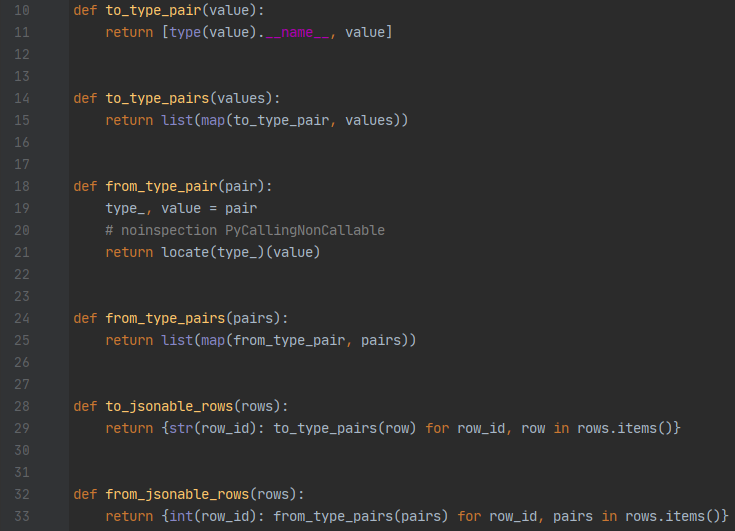
Роутинг Flask:



Опис сервісу у gRPC:



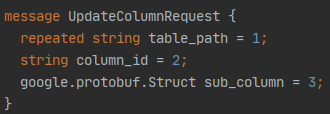
У gRPC повідомлення (message), що відповідають JSON об’єктам (Struct, ListValue, Value), реалізовані відповідно до JSON стандарту [rfc7159](https://tools.ietf.org/html/rfc7159#page-6), тому цілі числа, під час заповнення цих повідомлень перетворюються на double. Тому перед надсиланням дані конвертуються у такі, що зберігають тип значення поряд із самим значенням.

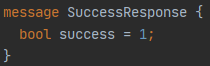


…

Нижче наведений повний шлях процедури gRPC від її опису до реалізації на сервері, клієнті:

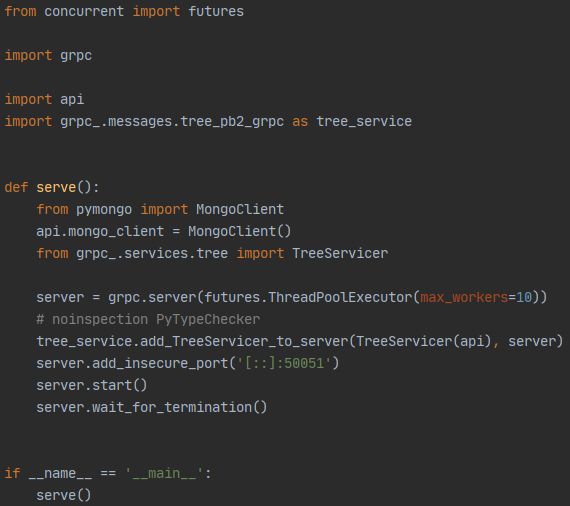
tree.proto:



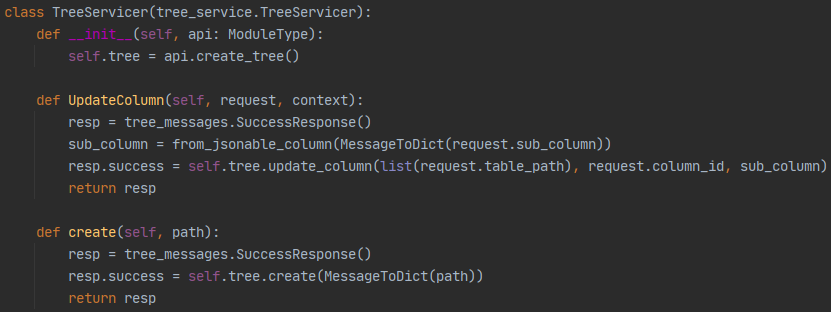


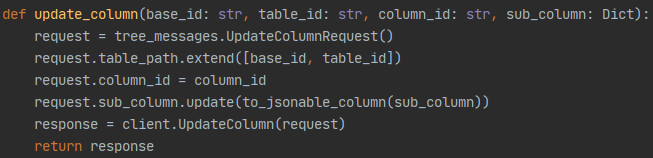


server.py:



grpc\_.services.tree.py:

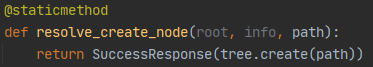


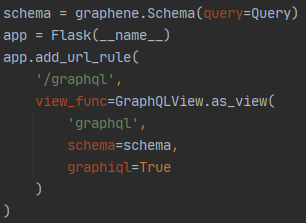
client.py:

Нижче наведений приклад того як працює GraphQL сервер:

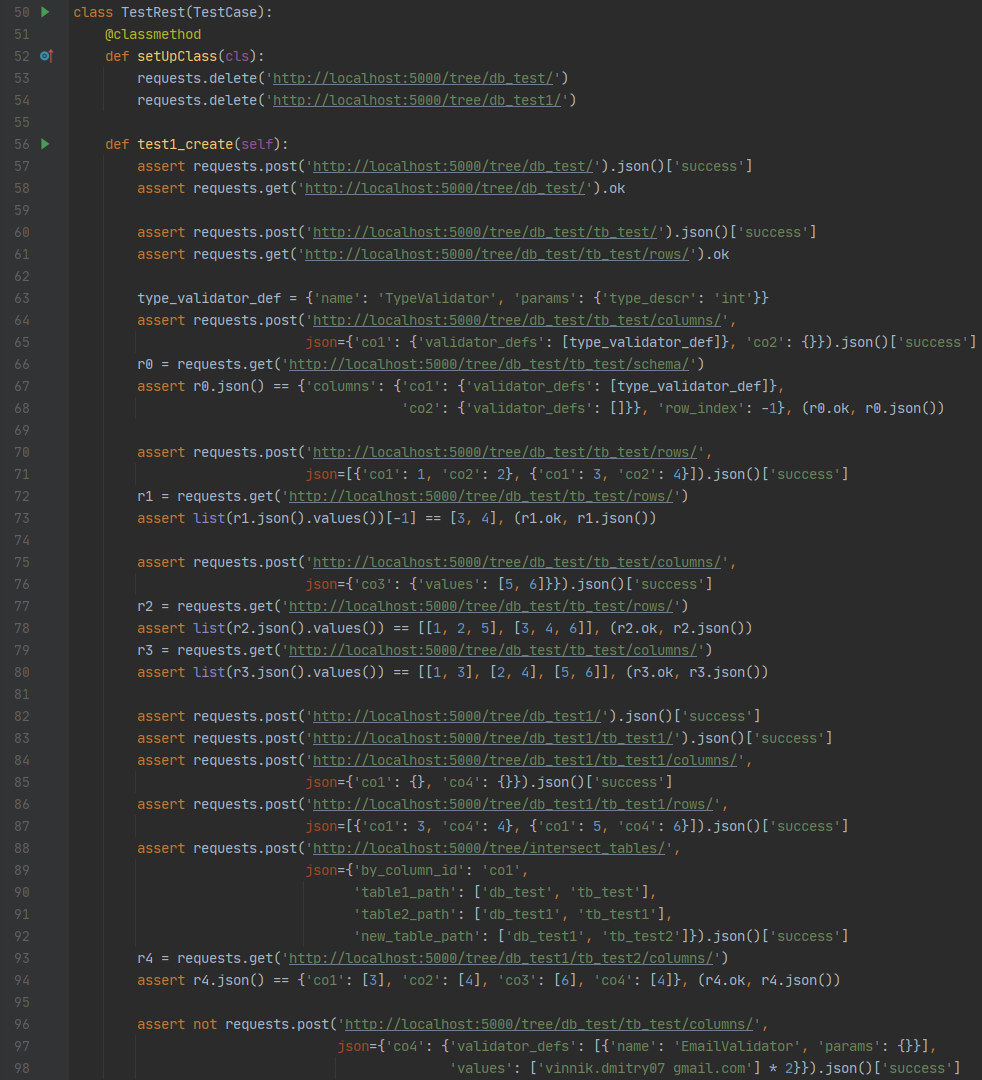


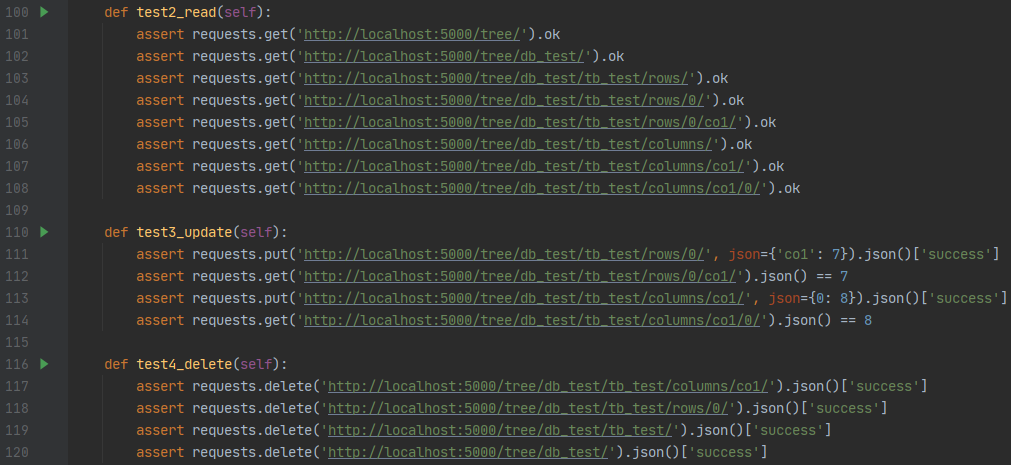






Тести REST:





Такі ж тести gRPC:



