КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

ПОБУДОВА ВЕБ-ЗАСТОСУНКУ ДЛЯ АВТОМАТИЧНОГО ОЦІНЮВАННЯ ЗНАНЬ СТУДЕНТІВ З ПРЕДМЕТУ "ОБЧИСЛЮВАЛЬНА ГЕОМЕТРІЯ ТА КОМП'ЮТЕРНА ГРАФІКА"

Виконав студент 4-го курсу
Фісуненко Артем Андрійович

(підпис)

Науковий керівник:
професор, доктор фіз.-мат. наук
Терещенко Василь Миколайович

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики
«____» _____ 2022 р., протокол № ____
Завідувач кафедри
В. М. Терещенко

(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи: 47 сторінок, 8 ілюстрацій, 18 використаних джерел.

Ключові слова: АВТОМАТИЗАЦІЯ НАВЧАЛЬНОГО ПРОЦЕСУ, АВТОМАТИЧНЕ ОЦІНЮВАННЯ, ВЕБ-ЗАСТОСУНОК, ЗАДАЧІ БЛИЗЬКОСТІ, КОМП'ЮТЕРНА ГРАФІКА, ЛОКАЛІЗАЦІЯ ТОЧКИ, ОБЧИСЛЮВАЛЬНА ГЕОМЕТРІЯ, ОПУКЛА ОБОЛОНКА, РЕГІОНАЛЬНИЙ ПОШУК.

Об’єктом роботи є автоматичне оцінювання знань студентів з курсу "Обчислювальна геометрія та комп'ютерна графіка" за допомогою бібліотеки для оцінювання алгоритмічних задач та веб-застосунку, що дає змогу проводити модульні контрольні роботи в режимі онлайн із їх подальшою перевіркою й оцінюванням програмою.

Метою кваліфікаційної роботи є створення й публікація бібліотеки та веб-сайту для проходження й автоматичного оцінювання модульних контрольних робіт з курсу "Обчислювальна геометрія та комп'ютерна графіка" з відкритим вихідним кодом, зручних для користування як студентами, так і викладачами.

Інструментом створення є безкоштовний, вільно поширюваний редактор вихідного коду Visual Studio Code. Мови програмування – Python, JavaScript. Використано фреймворки Django Rest Framework і React та бібліотеки pydantic, CGLib та Ant Design.

Результат роботи: розроблено та опубліковано бібліотеку для оцінювання та прототип сайту під ліцензією GNU General Public License v3.0.

Зв’язок з іншими роботами: у даній кваліфікаційній роботі використано розороблену в ході командної курсової роботи студентів Фісуненка Артема,

Германюка Всеволода, Рудя Максима та Черкашина Михайла алгоритмічну бібліотеку CGLib.

Створені бібліотеку для оцінювання та веб-сайт було розроблено в ході командного проєкту студентами Фісуненком Артемом, Германюком Всеволодом та Геллою Всеволодом.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки. На сьогоднішній день існує декілька готових рішень для автоматизації навчального процесу, наприклад, система Moodle [1], сайти на базі якої використовуються в тому числі викладачами Київського національного університету імені Тараса Шевченка. Moodle дає змогу студентам дистанційно проходити курси, виконувати тести, працювати у групі, завантажувати файли з розв’язками тощо; викладачам – використовувати інструменти для розробки авторських курсів, формувати різні типи тестів, у тому числі автоматично, перевіряти знання в автоматизованому режимі, формувати звіти про успішність здобувачів освіти (електронні журнали) тощо [2].

Актуальність роботи та підстави для її виконання. Незважаючи на наявність у мережі Інтернет згаданих систем, вони, як правило, мають більш загальне призначення, яке не враховує специфіку конкретних задач, і відтак функціонально обмежені. Тому було вирішено створити бібліотеку для оцінювання та веб-сайт, який давав би змогу оцінювати знання студентів з курсу обчислювальної геометрії та, можливо, інших предметів або галузей знань, задачі з яких мають подібний формат (алгоритмічний із покроковим розв’язанням та оцінюванням проміжних результатів).

Мета і завдання роботи. Метою кваліфікаційної роботи є створення й публікація вільно поширюваних кросплатформених бібліотеки та веб-сайту для автоматизованого оцінювання знань студентів.

Можливі сфери застосування. Кінцевий продукт може використовуватися для автоматичного оцінювання контрольних робіт та іспитів з курсу «Обчислювальна геометрія та комп'ютерна графіка». Бібліотеку для оцінювання та веб-сайт також можна використовувати для

оцінювання задач з інших предметів, які мають подібний формат, за умови відповідного корегування вихідного коду та вказання (атрибуції) авторів із посиланням на оригінал на умовах ліцензії GNU General Public License v3.0 [3].

РОЗДІЛ 1. БІБЛІОТЕКА ДЛЯ ОЦІНЮВАННЯ

1.1 Загальні відомості про оцінювання

Курс «Обчислювальна геометрія та комп'ютерна графіка» складається з трьох змістовних модулів, задачі з яких виносяться на контрольні роботи та іспит. У кожній задачі потрібно розв'язати деяку проблему обчислювальної геометрії з використанням відповідного методу (алгоритму):

Модуль 1. Геометричний пошук

1) Локалізація точки методом смуг
2) Локалізація точки методом ланцюгів
3) Локалізація точки методом деталізації триангуляції
4) Регіональний пошук методом *k*-D-дерева
5) Регіональний пошук методом дерева регіонів

Модуль 2. Опуклі оболонки

1) Побудова опуклої оболонки методом Ґрехема
2) Побудова опуклої оболонки методом Швидкобол
3) Побудова опуклої оболонки методом Препарата
4) Побудова опуклої оболонки методом підтримки динамічної опуклої оболонки

Модуль 3. Задачі близькості

1) Пошук найближчої пари методом «розділяй та пануй»
2) Побудова діаграми Вороного методом «розділяй та пануй»

При оцінюванні розв'язку задачі враховується коректність виконання студентом усіх проміжних етапів алгоритму. За замовчуванням це може виглядати як звичайне порівняння очікуваного та отриманого студентом

результатів, але в деяких задачах використовується й більш комплексна логіка – наприклад, додаткові штрафи за систематичність помилки.

Наведемо приклад схеми оцінювання:

Таблиця 1.1 – Схема оцінювання методу Ґрехема (Терещенко В. М.)

	Має бути	**Макс. оцінка**	**Зняття балів**		**Коментарі**
1	Задана множина S із N точок. Знайти внутрішню точку q.	0.25	Невірно знайдено або відсутній крок	-0.25	
2	Використовуючи q як початок координат, побудувати упорядкований за полярним кутом список точок множини S, починаючи із точки «початок» проти год. стрілки.	0.25	Невірно впорядковано або відсутній крок	-0.25	
3	Знайти точку «початок».	0.25	Невірно знайдено або відсутній крок	-0.25	
4	Організувати обхід.	1.25	Невірно виконується або відсутній крок	-1.25	
4.1	Починаючи з точки «початок», розглядаємо трійки сусідніх точок.	0.15	Невірно виконується або відсутній крок	-0.15	
4.2	Перевіряємо внутрішній кут по відношенню до q, який утворює трійка.	0.15	Невірно знайдено або відсутній крок	-0.15	
4.3	Якщо кут менше π – просуваємось вперед по списку на точку (яка перевірялась на крайність).	0.25	Невірно виконується або відсутній крок	-0.25	
4.4	Якщо кут більше або дорівнює π – вилучаємо середню точку трійки і повертаємось на точку назад по списку.	0.6	Невірно виконується або відсутній крок	-0.6 -0.3	Системна помилка Поодинока помилка
4.5	При досягненні точки «початок» третьою точкою трійки перевіряємо кут. Якщо він менше π, зупиняємося, інакше повертаємося на точку назад по списку.	0.1	Невірно виконується (циклиться через точку «початок») або не завершено крок	-0.25	

Як бачимо, процес розв’язання задачі можна умовно розбити на змістовні блоки (етапи), кожен з яких може містити кілька підпунктів (або, тривіально, один підпункт – сам етап). Для кожного з етапів доцільно зробити pydantic-моделі відповідей (результатів виконання) для порівняння й оцінювання за потрібною схемою. У подальших підрозділах розглянемо структуру таких моделей, процес оцінювання та інші складові частини бібліотеки.

1.2 Структура бібліотеки

Бібліотека для оцінювання CGMark складається з наступних модульних блоків: моделі, класи задачі, класи-«будівельники», оцінювання. Розглянемо детально кожну із цих її складових.

1.2.1 Моделі

Для зручного представлення та серіалізації/десеріалізації даних було використано бібліотеку pydantic [4]. Як зазначено у її документації, ця бібліотека також забезпечує валідацію типів під час виконання програми (чого немає у звичайному Python), й у випадку невідповідності типів видає повідомлення про помилку в дружньому для користувача вигляді.

Для отримання доступу до функціоналу pydantic достатньо успадкувати клас від pydantic-класу `BaseModel`. Тепер при зазначенні підказки типу [5] при полях такого класу буде здійснюватися валідація типів під час виконання програми – це гарантуватиме передачу аргументів правильного типу. Самі ж аргументи передаються виключно як keyword arguments – у наступному вигляді:

```
my_model = MyModel(argname1=value1, argname2=value2)
```

Також можна інстанціювати модель, маючи словник із відповідних назв аргументів та їхніх значень (зручно при роботі з JSON у подальшому), використовуючи Python-оператор розпакування словника «**»:

```
d = {“argname1”: value1, “argname2”: value2}
my_model = MyModel(**d)
```

Тепер розглянемо pydantic-реалізацію моделей.

1.2.1.1 Базові моделі

До базових моделей належать:

- точка `Point`;
- список точок `PointList`;
- вершина графу `Vertex`;
- ребро графу `Edge`;
- граф `Graph`;
- вершина бінарного дерева `BinTreeNode`, що містить деякі дані та дані її лівого та правого сина або `None` за їх відсутності;
- бінарне дерево `BinTree`, подане у вигляді списку вершин у порядку рекурсивного обходу дерева зліва направо;
- запитний регіон `Region`, поданий у вигляді *x*- та *y*-проміжків;
- комірка таблиці `TableCell`;
- рядок таблиці `TableRow`, поданий у вигляді списку комірок;
- таблиця `Table`, подана у вигляді списку рядків;
- таблиця із заголовками `HeaderTable`, успадковується від `Table`, додатково містить список заголовків.

Фактично, перші сім моделей є pydantic-відповідниками CGLib-моделей. Відмінність полягає у наявності в pydantic-моделях вбудованого серіалізатора та валідатора полів.

Слід окремо зауважити, що, оскільки координати точок, взагалі кажучи, є довільними дійсними числами, виникає необхідність коректно визначити їхнє порівняння, адже це є поширеною проблемою в арифметиці чисел із плаваючою комою: через обмежену кількість позицій у машинному слові для представлення дійсного числа не всі дійсні значення – елементи континуальної множини – можуть подаватися у точному вигляді, а лише наближено з певною точністю. Тож для уникнення помилок при порівнянні введенних студентом даних та відповіді програми перевизначимо Python-оператор порівняння «==» шляхом перевантаження методу `__eq__(self, other)` класу `Point` наступним чином: перевіряємо, чи аргумент `other` має тип `Point`, і, якщо так, порівнюємо *x*- та *y*-координати даної точки з *x*- та *y*-координатами іншої за допомогою функції `isclose` вбудованої Python-бібліотеки `math`. При цьому додатково слід задати аргумент допустимої абсолютної похибки `abs_tol` цієї функції деяким доволі великим значенням, наприклад, 0.001, щоб числа на кшталт 2.333 та 2.3334 вважалися рівними і студенту не доводилося вводити велику кількість знаків після коми.

1.2.1.2 Моделі умов задач

Незважаючи на доволі велику кількість різноманітних алгоритмів, представлених у курсі «Обчислювальна геометрія та комп’ютерна графіка», в усіх відповідних задачах можна виділити всього три типи умов:

1) **граф та точка** (модуль 1, задачі локалізації точки);

2) **список точок** (модуль 2, задачі знаходження опуклої оболонки; модуль 3, задачі близькості).
3) **список точок та запитний регіон** (модуль 1, задачі геометричного пошуку).

Ці умови представимо за допомогою pydantic-моделей із порожнім базовим класом `Condition`, що мають поля графу та точки (`GraphAndPointCondition`), списку точок (`PointListCondition`), списку точок та запитного регіону відповідно (`PointListAndRegionCondition`). Для уникнення дублювання поля списку точок останній клас успадкуємо від PointListCondition.

Таким чином, ієрархію класів умов можна зобразити у вигляді наступної UML-діаграми:

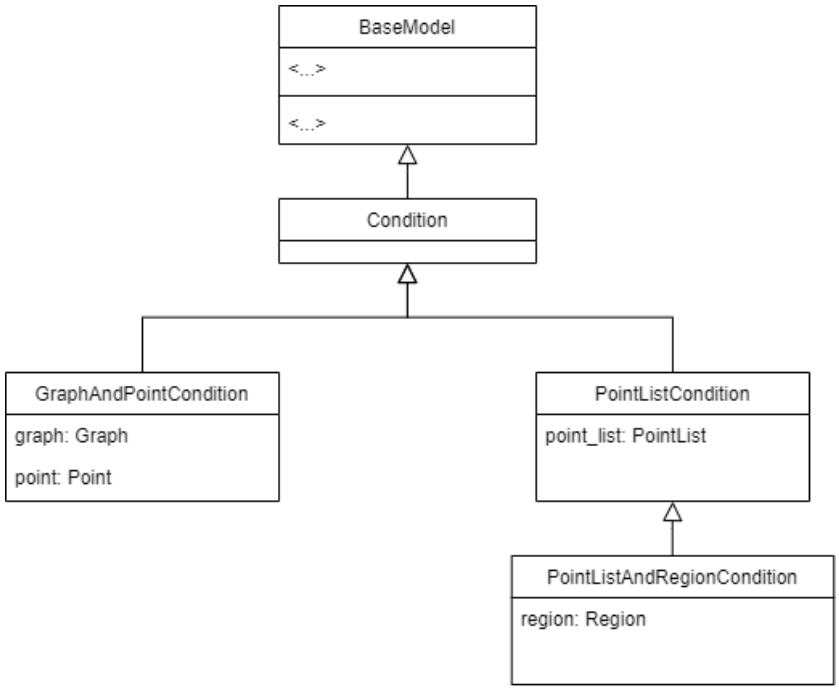


Рис. 1.1 – UML-діаграма класів умов

1.2.2 Класи задачі

Як зазначалося вище, в задачах доцільно виділити етапи та їхні підпункти. З урахуванням цього було спроєктовано наступні класи:

- `Task` (задача);
- `TaskStage` (етап задачі);
- `TaskItem` (підпункт етапу).

1.2.2.1 Задача

Для створення задачі необхідно передати єдиний аргумент – умову (типу `Condition`). Задача має наступні поля та властивості (*курсивом* відмічено поля класу (статичні), підкресленням – властивості):

- `condition` – умова, що є pydantic-моделлю;
- *`description`* – текстовий опис (назва) задачі;
- *`stages`* – список класів етапів (поле класу);
- `stages` – список етапів (поле об'єкта), заповнюється при ініціалізації;
- *`solution_method`* – метод (алгоритм) розв'язання задачі;
- *`item_answer_builder`* – клас «будівельника» проміжних відповідей, детально розглядатиметься далі.
- `items` – список послідовних підпунктів усіх етапів задачі;
- `answers` – список відповідей всіх послідовних підпунктів;
- `unwrapped_condition` – список CGLib-моделей для передачі у метод розв'язання, визначається на основі поля об'єкту задачі `condition` (див. нижче).

При ініціалізації задачі pydantic-умова приводиться до списку CGLib-моделей, що надалі передаються в метод розв'язання у «розпакованому» вигляді (за допомогою Python-оператора «*» список, переданий у метод,

перетворюється на набір його аргументів). Після цього отримаються «сирі» відповіді алгоритму у вигляді генератора [6]. Ці відповіді приводяться до pydantic-вигляду за допомогою «будівельника», після чого розподіляються по етапах та підпунктах задачі.

1.2.2.2 Етап задачі

Етап задачі `TaskStage` має два поля – *`description`* (текстовий опис етапу) та `items` – список підпунктів, який заповнюється при ініціалізації за допомогою об’єкту-будівельника.

1.2.2.3 Підпункт етапу задачі

І, нарешті, підпункт етапу задачі `TaskItem` містить поле *`description`* (текстовий опис підпункту) та `answer` – проміжну відповідь на даний підпункт в алгоритмічній задачі.

Тепер, знаючи повну структуру задачі, розглянемо процес наповнення її відповідями.

1.2.3 Класи-«будівельники»

Для створення pydantic-моделей проміжних відповідей із «сирих» вихідних даних CGLib-алгоритмів та їх подальшого розподілення по етапах та підпунктах задачі було використано дещо розширену варіацію шаблону проєктування «Будівельник» (Builder) із книги «Design Patterns: Elements of Reusable Object-Oriented Software» за авторством «Банди чотирьох» (Gang of Four). Даний шаблон проєктування розв’язує проблему конструювання

складного об’єкту, що має на увазі створення кількох інших об’єктів в процесі, шляхом «розділення процесу конструювання та репрезентації так, що один і той самий процес конструювання може породжувати різні репрезентації» [7]. Якщо створювати складний об’єкт шляхом послідовного інстанціювання таких об’єктів у коді напряму, це негативно позначається на можливості повторного використання. Наприклад, для створення об’єктів з іншими вхідними даними або за іншою схемою доведеться постійно переписувати великі шматки коду зі зміненими значеннями аргументів чи розгалужувати процес створення простіших об’єктів умовними переходами або великою ієрархією успадкування для всеможливих модифікацій об’єктів.

Патерн «Будівельник», навпаки, пропонує зручне рішення: інкапсулювати створення кожного з простіших об’єктів в окремий метод класу-будівельника. Тоді, по-перше, ми звільняємося від необхідності писати повторювані рядки коду, по-друге, розділяємо відповідальність за створення об’єктів на конкретні методи, що дає змогу їх зручно й наочно модифікувати й викликати їх в клієнтському коді – або не викликати деякі етапи створення зовсім: аналогія з реального життя, яка й дала назву шаблону, – власне, будівництво оселі, яке складається з конструювання вікон, дверей, фарбування стін тощо – як правило, кожною із цих задач займається окремий спеціаліст; при цьому ми можемо, наприклад, побудувати дім без вікон – не викликати відповідний метод.

В контексті даної роботи патерн «Будівельник» дає нам змогу компактно розміщувати відповіді в підпунктах етапів для різних вхідних даних, а також не засмічувати метод `__init__` ініціалізації об’єкту задачі викликами конструкторів pydantic-моделей проміжних відповідей, які можуть бути доволі громіздкими через необхідність писати назви аргументів та їхню кількість чи складність. При цьому, умови та схеми конструювання відповідей

є різними, і для кожного нового алгоритму довелося б писати таку ініціалізацію в методі `__init__` наново, що на практиці часто призводить до помилок, особливо при копіюванні, вставці та модифікації позірно схожих фрагментів коду. Також за такого підходу ми фактично не маємо змоги написати в методі `__init__` базового класу задачі загальний для всіх нащадків код ініціалізації без перерахування конкретних нащадків, що є порушенням принципу інверсії залежності (Dependency Inversion у SOLID) [8].

Застосування патерну «Будівельник» вирішує всі згадані проблеми. Тепер, вказавши тип конкретного класу-будівельника у полі класу задачі, ми маємо змогу написати загальний код в методі `__init__` базового класу, що викликається всіма нащадками, і не перевизначати цей метод в жодному з них. Фактично, вся необхідна інформація щодо конкретних нащадків міститься у визначенні їхніх полів, які підставляються в базовий `__init__`.

Розширення ж патерну «Будівельник» у даній роботі виглядає наступним чином. Методи конструювання об’єктів-частин є статичними та викликаються у класовому методі будівельника `build(cls, raw_answers)`. (Відмінність класового методу (декоратор `@classmethod`) від статичного (декоратор `@staticmethod`) полягає в тому, що останній не має інформації про клас і працює лише з аргументами, тоді як класовий метод має інформацію про поля класу, адже його першим аргументом, власне, і є клас [9]).

Таким чином, у методі build викликається допоміжний класовий метод `_build_methods()`, що повертає список перерахованих методів конструювання об’єктів-частин. Це, до речі, є певним покращенням класичного способу використання патерну «Будівельник»: незважаючи на те, що ми явно перераховуємо назви методів у коді будівельника, користувач будівельника позбавляється необхідності робити це самостійно і потенційно

велику кількість разів – достатньо лише один раз викликати метод `build` на одне використання будівельника.

Сам метод `build` повертає генератор відповідей у pydantic-форматі, отриманих шляхом застосування методів конструювання до CGLib-відповідей. Надалі, при наповненні етапів задачі та їхніх підпунктів відповідями, генератор покроково повертає по одній відповіді на кожну ітерацію спискового включення [10]. Ця його властивість дає змогу коректно розподілити відповіді по підпунктах етапів: при кожному новому зверненні до генератора індекс його елементів не збивається на початковий, а продовжує зростати, поки не досягне кінця. Тобто, наприклад, якщо ми маємо відповіді 1, 2, 3, 4, 5, етап 1 із трьома підпунктами та етап 2 із двома підпунктами, то кінцевий розподіл відповідей по етапах буде (і має бути) такий: етап 1 – 1, 2, 3; етап 2 – 4, 5. Якби замість генератора у включенні ми використали список, ми би отримали результат: етап 1 – 1, 2, 3; етап 2 – 1, 2 – адже у списку індексація починається з 0 в кожному новому виклику спискового включення.

Детальніше із кодом будівельника та задач можна ознайомитися за посиланням [11].

1.2.4. Оцінювання

Для оцінювання нам необхідно порівняти списки еталонних відповідей, отриманих за вищевказаною схемою, та відповідей студента. Також, за наявності помилок, потрібно зняти певну кількість балів за визначеною схемою, виставити отриману на кожному етапі оцінку та підрахувати сумарний бал.

1.2.4.1 Моделі помилки

Помилку подамо у вигляді pydantic-моделі `Mistake` з наступними полями:

- `sub` – кількість балів до зняття за хоча б одну помилку;
- `big_sub` – кількість балів до зняття за систематичність помилки (якщо застосовно), за замовчуванням 0;
- `data` – інформація про помилку, наприклад, текстовий опис;
- `systematic` – прапорець застосовності зняття більшої кількості балів за систематичність помилки, за замовчуванням `False`.

Для поетапного оцінювання визначимо pydantic-моделі `ItemMarkData`, що містить поля максимальної оцінки `max_mark` та мінімальної `min_mark` (за замовчуванням 0), та `MarkData`, що містить поле `items` – список моделей `ItemMarkData` для кожного підпункту етапу.

1.2.4.2 Методи знаходження помилок

При підрахунку штрафів за помилки доцільно використати клас-лічильник `Counter` з вбудованої Python-бібліотеки `collections`, оскільки в студента може бути декілька однакових за змістом помилок різних типів. Застосувавши лічильник до списку помилок (для цього вони повинні реалізовувати метод `__hash__`), ми маємо змогу отримати словник з парами вигляду «помилка певного типу – кількість таких помилок». Ми можемо проітерувати по елементах словника з урахуванням того, що в даній роботі штраф за помилки одного типу рахується не як сума штрафів за кожну таку помилку, а як штраф за наявність помилок такого типу загалом, і призначити менший або більший штраф в залежності від потреби знімати більше балів за

систематичність помилки. Тобто при ітерації у парі «помилка певного типу – кількість» ми перевіряємо, чи така потреба є *і* чи помилок дві або більше, і призначаємо більший штраф, якщо так. В інакшому випадку (потреби немає або потреба є і помилка лише одна) призначаємо звичайний штраф.

З міркувань зручності було написано два базові методи знаходження помилок – за замовчуванням `default_grading(correct, answer, sub=0.0)` для порівняння моделей-примітивів, таких як точка, число тощо; та порівняння колекцій `iterable_grading(correct: Iterable, answer: Iterable, sub=0.0, big_sub=0.0)` для пошуку неспівпадінь у спискових, кортежних тощо даних.

`default_grading` здійснює порівняння аргументів `correct` та `answer` за допомогою звичайного Python-оператора «==» та повертає список із єдиною помилкою зі штрафом `sub`.

`iterable_grading`, в свою чергу, здійснює порівняння з урахуванням недостачі чи надлишку елементів в `answer` відносно `correct`. За кожен недостатній чи надлишковий елемент нараховується помилка із відповідними значеннями штрафів, і додається до списку помилок за кожне неспівпадіння елементів списків, що мають однакові індекси, з тими самими значеннями штрафів.

Для пошуку помилок в розв’язку конкретних алгоритмічних задач можуть використовуватися як і ці два, так і більш складні методи, що реалізуються окремо у відповідних файлах.

1.2.4.3 Клас-оцінювач

Клас `Grader` містить поля `markdata` типу `MarkData` (див. вище) та `item_grading_methods` – колекція методів пошуку помилок, за

замовчуванням `cycle([default_grading])` – потенційно необмежений цикл по пошуку помилок за замовчуванням.

`Grader` має єдиний класовий метод оцінювання `grade(cls, correct, answer)`. Процес оцінювання у ньому відбувається наступним чином:

1. Ітерацією по методах оцінювання, застосованих до аргументів `correct` та `answer`, збираємо список помилок.
2. Здійснюємо підрахунок штрафів за знайдені помилки кожного типу для кожного етапу, формуємо список.
3. Виставляємо оцінки за кожний етап, віднімаючи штрафи, з урахуванням мінімально можливого балу з етап.
4. Виставляємо сумарну оцінку як суму оцінок за кожен етап.
5. Повертаємо сумарну оцінку та розподіл оцінок за кожний підпункт етапів.

1.3 Алгоритми, оцінювання яких реалізовано в CGMark

У даній роботі авторами реалізовано оцінювання задач із трьох алгоритмів обчислювальної геометрії: метод Ґрехема, метод Швидкобол та метод *k*-D-дерева.

1.3.1 Оцінювання методу Ґрехема

Схему оцінювання методу див. в Таблиці 1.1.

Моделі, що використовуються при оцінюванні, наступні:

- `GrahamPoint` – точка, успадковується від `Point`;
- `GrahamPointList` – список точок, успадковується від `PointList`;

- `GrahamTrinityCell` – комірка таблиці з кортежем – трійкою точок, успадковується від `TableCell`;
- `PiCompare` – перелік, містить поля `less` та `more` на позначення того, що кут $< \pi$ або $\geq \pi$ відповідно;
- `GrahamPiCompareCell` – комірка таблиці з вмістом – значенням попереднього переліку, успадковується від `TableCell`;
- `GrahamCenterPointCell` – комірка таблиці з точкою, яку потрібно додати чи видалити на черговому кроці виконання обходу списку точок, успадковується від `TableCell`;
- `ToAddGraham` – перелік, містить поля yes та no на позначення додавання чи видалення точки відповідно;
- `GrahamToAddCell` – комірка таблиці з вмістом – значенням попереднього переліку, успадковується від `TableCell`;
- `GrahamTableRow` – рядок таблиці, складається з кортежу всіх попередніх комірок, успадковується від `TableRow`;
- `GrahamTable` – таблиця, містить список рядків та заголовки (за замовчуванням порожні), успадковується від `HeaderTable`.

Методи будівельника `GrahamModelBuilder`, що використовуються при оцінюванні, наступні:

- `_build_internal_point` – повертає внутрішню точку `GrahamPoint` на основі CGLib-точки;
- `_build_ordered` – повертає список точок `GrahamPointList` на основі списку CGLib-точок;
- `_build_origin` – повертає точку «початок» `GrahamPoint` на основі CGLib-точки;

- `_build_steps_table` – повертає таблицю кроків обходу Ґрехема `GrahamTable` на основі CGLib-таблиці.

Структура етапів та їхніх підпунктів наступна:

1) Етап `GrahamStageInternalPoint` та його тривіальний підпункт `GrahamItemInternalPoint` – знаходження центроїду перших трьох неколінеарних точок множини.
2) Етап `GrahamStageOrderedList` та його тривіальний підпункт `GrahamItemOrderedList` – знаходження впорядкованого за полярним кутом списку точок.
3) Етап `GrahamStageOriginPoint` та його тривіальний підпункт `GrahamItemOriginPoint` – знаходження точки «початок».
4) Етап `GrahamStageLookup` та його тривіальний підпункт `GrahamItemLookup` – здійснення обходу Ґрехема. Слід зазначити, що в оригінальній схемі цей етап містить кілька підпунктів, але тут вони для зручності поєднані в один підпункт для повернення й передачі на рівень оцінювання лише таблиці з усією необхідною інформацією.

Відповідна задача `GrahamTask` агрегує в собі ці етапи, в якості будівельника використовує `GrahamBuilder` і при розпакуванні pydantic-умови `PointListCondition` переводить її в список CGLib-точок.

Для зручного оцінювання напишемо дві функції з використанням так званого часткового застосування [12] – механізму функціонального програмування, що дозволяє задати обгортку для функції з наперед ініціалізованими значеннями аргументів:

```
default025 = partial(default_grading, sub=0.25)
iterable025 = partial(iterable_grading, sub=0.25)
```

Такий підхід дозволяє уникнути дублювання ініціалізації аргументів однаковими значеннями в кількох різних викликах функції.

Методи оцінювання по етапах для даного методу наступні:

1) Знаходження центроїду – метод оцінювання `default025`, максимальна оцінка 0.25 бала;
2) Знаходження впорядкованого списку – метод оцінювання `ordered_grading`, що повертає результат `iterable025(correct.points, answer.points)`, де correct та answer мають тип `GrahamPointList`; максимальна оцінка – 0.25 бала;
3) Знаходження точки «початок» – метод оцінювання `default025`, максимальна оцінка 0.25 бала;
4) Здійснення обходу Ґрехема – метод оцінювання `steps_table_grading`, який оцінює всі проміжні підпункти з оригінальної схеми, максимальна оцінка 1.25 бала.

1.3.2 Оцінювання методу Швидкобол

Схема оцінювання методу наступна:

Таблиця 1.2 – Схема оцінювання методу Швидкобол (Терещенко В. М.)

	Має бути	**Макс. оцінка**	**Зняття балів**		**Коментарі**
1	Розбиття. Подати у вигляді дерева.	1.0	Невірно побудоване дерево, або відсутнє	-1	
1.1	Знайти точки min(x), max(x) і розділити множину S із N точок на дві підмножини.	0.25	Невірно знайдено або відсутній крок	-0.25	
1.2	Вказати рекурсивно на кожному кроці точку h, яка	0.25	Невірно вказано або відсутній крок	-0.5 -0.25	Системна помилка Поодинока помилка

	утворює трикутник найбільшої площі.				
1.3	Вилучити рекурсивно на кожному кроці множину точок, які лежать усередині трикутника, або на його сторонах. Передати точки, що лишилися, на наступний рівень рекурсії для обробки.	0.25	Невірно вилучено або відсутній крок	-0.25	
1.4	Процес розбиття завершується, якщо у підмножинах залишилося по дві точки.	0.25	Невірно завершено або відсутній крок	-0.25	
2	Злиття. Рекурсивний підйом, результат – конкатенація списків. Починаючи із листків дерева, рекурсивно конкатенуємо упорядковані за годинниковою стрілкою списки.	1.0	Невірно виконується або відсутній крок	-1	

Моделі, що використовуються при оцінюванні, наступні:

- `QuickhullPoint` – точка, успадковується від `Point`;
- `QuickhullInitialPartition` – початкове розбиття множини, містить поля крайньої лівої точки `min_point`, крайньої правої точки `max_point`, верхньої множини точок `s1` та нижньої множини точок `s2`;
- `QuickhullNodeData` – дані вершини дерева розбиття, містить список точок points, точку `h`, яка утворює трикутник найбільшої площі, та фрагмент опуклої оболонки `hull_piece`;
- `QuickhullTreeNode` – вершина дерева розбиття, містить поля data типу `QuickhullNodeData`, `left` та `right` типу `Optional[QuickhullNodeData]` – тобто може бути або значення типу `QuickhullNodeData`, або `None` (за відсутності

лівого чи правого сина вершини), успадковується від `BinTreeNode`;

- `QuickhullTree` – дерево розбиття, містить список вершин `nodes` та властивість `leaves`, яка повертає листки дерева; успадковується від `BinTree`;
- `QuickhullPartition` – розбиття, складається з початкового розбиття та дерева розбиття.

Методи будівельника `QuickhullModelBuilder`, що використовуються при оцінюванні, наступні:

- `_build_partition` – даний метод повертає розбиття `QuickhullPartition` на основі CGLib-даних крайніх точок, перших двох підмножин та дерева розбиття;
- `_build_tree` – повертає дерево розбиття `QuickhullTree` на основі CGLib-дерева.

Структура етапів та їхніх підпунктів наступна:

1) Етап `QuickhullStagePartition` та його тривіальний підпункт `QuickhullItemPartition` – знаходження розбиття. Слід зазначити, що в оригінальній схемі цей етап містить кілька підпунктів, але тут вони для зручності поєднані в один підпункт для повернення й передачі на рівень оцінювання лише контейнера `QuickhullPartition` з усією необхідною інформацією.
2) Етап `QuickhullStageMerge` та його тривіальний підпункт `QuickhullItemMerge` – злиття фрагментів опуклої оболонки в єдиний список.

Відповідна задача `QuickhullTask` агрегує в собі ці етапи, в якості будівельника використовує `QuickhullBuilder` і при розпакуванні pydantic-умови `PointListCondition` переводить її в список CGLib-точок.

Для зручності визначимо кілька partial-функцій:

```
default025 = partial(default_grading, sub=0.25)
iterable025 = partial(iterable _grading, sub=0.25)
iterable1 = partial(iterable_grading, sub=1.0)
```

Методи оцінювання по етапах для даного методу наступні:

1) Знаходження розбиття – метод оцінювання `grade_partition`, в процесі оцінює коректність початкового розбиття, списку точок, що утворюють трикутник максимальної площі, підмножин точок та завершення розбиття; максимальна оцінка – 1 бал;
2) Проведення злиття – метод оцінювання `grade_merge`, в процесі оцінює коректність визначення злитих фрагментів опуклої оболонки у кожній з вершин дерева, максимальна оцінка – 1 бал.

1.3.3 Оцінювання методу *k*-D-дерева

Схема оцінювання методу наступна:

Таблиця 1.3 – Схема оцінювання методу *k*-D-дерева (Терещенко В. М.)

	Має бути	**Макс. оцінка**	**Зняття балів**		**Коментарі**
1	Попередня обробка	1.0	Відсутній крок або невірно зроблено	-1	
1.1	Відсортовані списки по *x* та *y*	0.25	Відсутній крок або невірно зроблено	-0.25	
1.2	Рекурсивне розбиття площини на прямокутники	0.75	Відсутній крок або невірно зроблено	-0.75	

2	Структура даних – побудова дерева пошуку	1	Відсутній крок або невірно зроблено	-1	
3	Пошук	1	Відсутній крок або невірно зроблено	-1	

Моделі, що використовуються при оцінюванні, наступні:

- `KdTreePoint` – точка, успадковується від `Point`;
- `KdTreeOrderedLists` – впорядковані за *x* та впорядкований за *y* списки;
- `KdTree` – *k*-D-дерево, містить поле регіону типу `Region`; успадковується від `BinTree`;
- `Partition` – перелік, містить поля `vertical` та `horizontal` на позначення вертикальності чи горизонтальності чергової прямої розбиття відповідно;
- `ToAddKdTree` – перелік, містить поля `yes` та `no` на позначення того, додавати чи ні чергову точку в список тих, що потрапляють у регіон, відповідно;
- `Intersection` – перелік, містить поля `yes` та `no` на позначення того, перетинає чи ні регіон пряма розбиття, що проходить через дану точку, відповідно;
- `KdTreePointCell` – комірка таблиці з точкою, успадковується від `TableCell`;
- `KdTreePartitionCell` – комірка таблиці з вмістом – значенням з переліку `Partition`, успадковується від `TableCell`;
- `KdTreeToAddCell` – комірка таблиці з вмістом – значенням з переліку `ToAdd`, успадковується від `TableCell`;

- `KdTreeIntersectionCell` – комірка таблиці з вмістом – значенням з переліку `Intersection`, успадковується від `TableCell`;
- `KdTreePartitionTableRow` – рядок таблиці, складається з кортежу комірок `KdTreePointCell` та `KdTreePartitionCell`, успадковується від `TableRow`;
- `KdTreeSearchTableRow` – рядок таблиці, складається з кортежу комірок `KdTreePointCell`, `KdTreeToAddCell` та `KdTreeIntersectionCell`, успадковується від `TableRow`;
- `KdTreePartitionTable` – таблиця, містить список рядків розбиття та заголовки (за замовчуванням порожні), успадковується від `HeaderTable`.
- `KdTreeSearchTable` – таблиця, містить список рядків пошуку та заголовки (за замовчуванням порожні), успадковується від `HeaderTable`.

Методи будівельника `KdTreeModelBuilder`, що використовуються при оцінюванні, наступні:

- `_build_ordered_lists` – повертає впорядкований по *x* та впорядкований по *y* списки `KdTreeOrderedLists` на основі списків CGLib-точок;
- `_build_partition_table` – повертає таблицю розбиття площини прямокутниками на основі списку кортежів з елементами – точкою та булевим значенням `True` або `False` (вертикальністю або горизонтальністю січної прямої відповідно);
- `_build_tree` – повертає дерево `KdTree` на основі CGLib-дерева;

- `_build_search_table` – повертає таблицю пошуку по дереву на основі CGLib-пошуку.

Структура етапів та їхніх підпунктів наступна:

1) Етап `KdTreeStagePreprocessing` – попередня обробка:
    1) Підпункт `KdTreeItemOrderedLists` – побудова відсортованих за *x* та *y* списків.
    2) Підпункт `KdTreeItemPartition` – рекурсивне розбиття площини на прямокутники прямими, що проходять через точки множини.
2) Етап `KdTreeStageTree` та його тривіальний підпункт `KdTreeItemTree` – побудова дерева пошуку.
3) Етап `KdTreeStageSearch` та його тривіальний підпункт `KdTreeItemSearch` – пошук по дереву.

Відповідна задача `KdTreeTask` агрегує в собі ці етапи, в якості будівельника використовує `KdTreeBuilder` і при розпакуванні pydantic-умови `PointListAndRegionCondition` переводить її в список із трьох аргументів – список CGLib-точок, *x*- та *y*-межі регіону.

Для зручності визначимо кілька partial-функцій:

```
iterable025 = partial(iterable _grading, sub=0.25)
iterable1 = partial(iterable_grading, sub=1.0)
```

Методи оцінювання по підпунктах етапів для даного методу наступні:

1) Знаходження впорядкованих списків – метод оцінювання `grade_ordered_lists`, в процесі оцінює коректність кожного зі списків за допомогою `iterable025`; максимальна оцінка – 0.25 бала;
2) Знаходження розбиття площини прямокутниками – метод оцінювання `grade_partition_table`, в процесі оцінює

відповідність рядків таблиць розбиття (`correct.rows` та `answer.rows`) за допомогою `iterable_grading`; максимальна оцінка – 0.75 бала;

3) Побудова дерева пошуку – метод оцінювання `grade_search_tree`, в процесі оцінює відповідність вершин дерев (`correct.nodes` та `answer.nodes`) за допомогою `iterable1`; максимальна оцінка – 1 бал;
4) Здійснення пошуку по дереву – метод оцінювання `grade_search_table`, в процесі оцінює відповідність рядків таблиць пошуку (`correct.rows` та `answer.rows`) за допомогою `iterable1`; максимальна оцінка – 1 бал.

1.4 Тестування

Для забезпечення коректності роботи бібліотеки та контролю якості було використано підхід Test-Driven Development (TDD): на кожне додавання нової функціональності у бібліотеку пишеться відповідний модульний тест, запускається, і за наявності помилок здійснюється корегування коду. Публікація у віддалений репозиторій на GitHub здійснюється лише після коректного проходження всіх тестів.

В рамках даної бібліотеки тестуванню підлягають дві основні позиції – коректність створення задач та коректність їхнього оцінювання. Відповідно у проєкті присутні тести:

- `test_tasks.py` – у цьому файлі зібрано по одному тесту на кожен алгоритм. При тестуванні ми створюємо задачу з деякою умовою, задаємо вручну всі коректні проміжні відповіді, та

перевіряємо шляхом порівняння, чи коректно вони були розподілені по підпунктах етапів.

- `test_graders.py` – у цьому файлі також зібрано по одному тесту на кожен алгоритм. При тестуванні достатньо задати лише коректні проміжні відповіді, а також деякі інші коректні (або ні) відповіді, передати їх на оцінку у метод `grade` відповідного класу й перевірити, чи збігається задана очікувана оцінка та розподіл поетапних оцінок з виставленими методом.

Для тестування використано вбудований Python-фреймворк unittest. Перевірити правильність тестів можна, запустивши тести напряму з графічного інтерфейсу Visual Studio Code або, альтернативно, із терміналу операційної системи за допомогою команди `python -m unittest`.

1.5 Розширення бібліотеки

Однією з найголовніших переваг бібліотеки CGMark є те, що вона не обмежується можливістю оцінювання лише алгоритмів з предмету «Обчислювальна геометрія та комп’ютерна графіка». Кожен користувач бібліотеки може легко модифікувати чи розширити її під власні потреби за умови, що задачі, оцінювання яких користувач планує автоматизувати, мають подібний (алгоритмічний) формат. Для цього необхідно виконати наступні кроки:

1. Поділити задачу на етапи та їхні підпункти, визначити схему їхнього оцінювання.
2. Реалізувати алгоритм розв’язання задачі з поверненням проміжних відповідей.

3. Надати моделі для проміжних відповідей чи використати для цього представлені у CGMark або стандартні засоби Python.
4. Написати клас-будівельник та його методи конструювання моделей проміжних відповідей.
5. Надати моделі задачі, етапів задачі та підпунктів етапів.
6. Надати методи для оцінювання етапів чи використати представлені у CGMark, написати клас-оцінювач.

Після цього можна створювати та оцінювати задачі, як було описано вище. Обмежень на тематику задач фактично немає, єдиною вимогою залишається алгоритмічність процесу розв’язання (має бути покрокова схема, яку можна формалізувати у вигляді алгоритму з проміжними результатами). У такому вигляді можна реалізувати навіть тестові завдання – для цього достатньо використати метод порівняння за замовчуванням для завдань з одиничним вибором та метод порівняння колекцій для завдань з множинним вибором.

При публікації результатів, отриманих шляхом використання, модифікації чи доповнення CGMark, необхідно виконати наступні умови ліцензії GNU General Public License v3.0:

- включити до результатів, що публікуються, текст ліцензії та копірайт авторів;
- зазначити зміни, внесені до бібліотеки (за наявності);
- вказати джерело (посилання на оригінальний вихідний код);
- результати мають розповсюджуватися за тією самою ліцензією, що й оригінальний проєкт.

Дана ліцензія дозволяє:

- комерційне використання ліцензованого матеріалу та його похідних;

- внесення змін до ліцензованого матеріалу;
- розповсюдження ліцензованого матеріалу;
- патентне використання;
- персональне використання.

Ліцензія GNU General Public License v3.0 має два обмеження – на гарантії щодо використання та відповідальність при використанні іншими людьми ліцензованого матеріалу: матеріал поставляється «як є», і автори не дають жодних гарантій щодо програми та не несуть відповідальності за потенційно некоректну поведінку програми (окрім випадків, передбачених законом) [3].

РОЗДІЛ 2. ВЕБ-САЙТ

Маючи бібліотеку CGMark, ми тепер можемо створити систему, що здійснює автоматичне оцінювання, у вигляді веб-сайту. Окрім власне оцінювання, проєкт веб-сайту також має на увазі:

- авторизацію студентів;
- збереження оцінок за кожну задачу та її етапи у базу даних (електронний журнал);
- обмеження по часу виконання завдань тощо.

Як і при створенні більшості веб-сайтів, в процесі розробляються бекенд- і фронтенд-частини, які ми надалі й розглянемо.

2.1 Бекенд

Наразі найпопулярнішим бекенд-фреймворком на Python є Django [13]. Для зручного створення REST API-ендпоїнтів (адрес для звернення з REST API-запитами у веб-застосунку) часто використовується його структурна частина Django REST Framework (DRF), яку було застосовано також і в даному проєкті.

При генерації Django-проєкту створюється кілька стандартних файлів, серед яких нас цікавитимуть:

- `urls.py` – два файли з такою назвою, один містить посилання всього проєкту, інший – конкретного застосунку;
- `models.py` – файл з моделями;
- `admin.py` – файл з налаштуваннями адміністрування для реєстрації моделей;
- `views.py` – файл із представленнями;

- `db.sqlite3` – база даних SQLite.

Фактично, Django реалізує шаблон проєктування Model-View-Controller (MVC), тільки з дещо іншою термінологією – Model-Template-View (MTV), де преддставлення називаються шаблонами, а контролери – представленнями [14]. Для уникнення розбіжностей зазначимо, що тут і далі в роботі ми дотримуватимемося термінології фреймворку.

Зважаючи на те, що ми створюємо REST API, шаблони (HTML-сторінки) для цього не потрібні. Достатньо забезпечити лише наявність відповідних ендпоїнтів-представлень, до яких в подальшому звертатиметься клієнт на фронтенді.

Спочатку створимо усі необхідні моделі, які надалі мігруємо до бази даних, де створяться відповідні SQL-таблиці.

2.1.1 Моделі

У даній роботі спроєктуємо моделі, пов’язані із процесом оцінювання, та моделі, що використовуватимуться при авторизації студента:

- `UniversityGroup` – група студентів, має поле назви `name`;
- `Student` – користувач-студент, має поле групи `group` та властивість `score`, що повертає сумарну оцінку за всі виконані студентом завдання; успадковується від та розширює функціонал вбудованої Django-моделі користувача `AbstractUser`;
- `Module` – контрольна робота, містить поля назви `name`, опису `description`, часу початку та закінчення `start_time` та `end_time` відповідно, список задач `tasks` та властивість

`max_score`, що повертає суму максимальних оцінок за всі задачі контрольної роботи;

- `Task` – задача, містить поля назви `name`, опису `description`, файлового представлення `pickle_dump` та властивість `max_score`, що повертає суму максимальних оцінок за всі етапи задачі;
- `Stage` – етап задачі, містить поля назви `name`, опису `description`, максимальної оцінки `max_score`, задачі `task` та списку студентів `students`;
- `StudentStage` – модель-зв'язка між моделями `Student` та `Stage` для зберігання оцінок студентів за етапи, містить поля студента `student`, етапу `stage` та оцінки `score`;
- `TaskModule` – модель-зв'язка між моделями `Task` та `Module` для потенційного повторного використання задач у контрольних, містить поля задачі `task` та контрольної `module`.

Структуру відповідних таблиць у БД можна зобразити наступним чином:

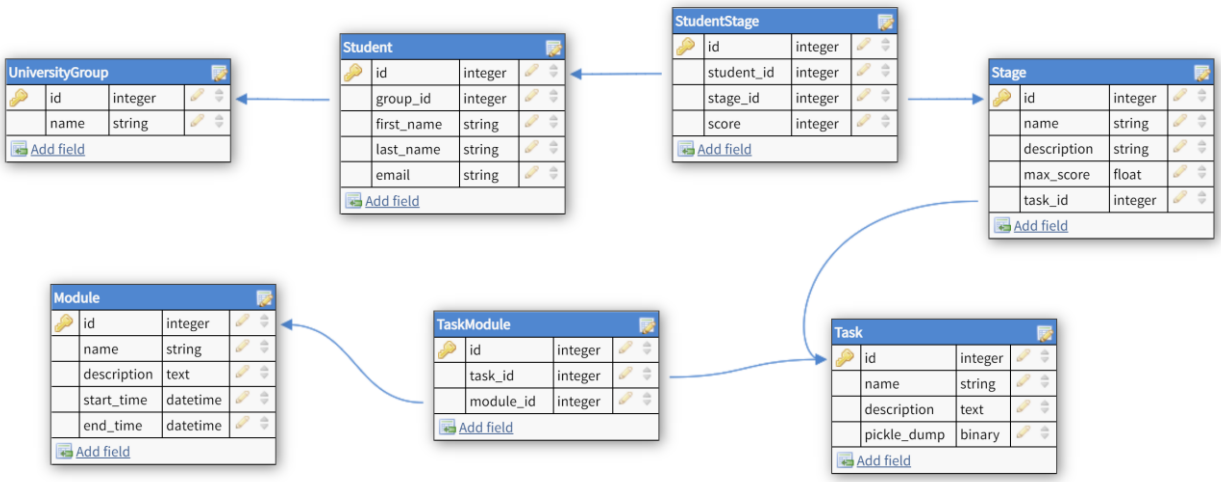


Рисунок 2.1 – Діаграма бази даних

Для занесення цих таблиць у базу даних потрібно виконати дві команди у терміналі: `python manage.py makemigrations` для генерації SQL-коду на основі коду моделей та `python manage.py migrate` для виконання SQL (відповідно, занесення таблиць у БД). Для можливості керування моделями з панелі адміністратора їх необхідно зареєструвати у файлі `admin.py` у вигляді `admin.site.register(Model)`.

Тепер розглянемо формування представлень.

2.1.2 Представлення

Для перегляду та модифікації базових сутностей використано так звані viewsets – представлення, що дають змогу надсилати GET-запити щодо колекції об’єктів або одного об’єкта за його ідентифікатором, а також POST-запит щодо одного об’єкта за його ідентифікатором. При цьому можна вказати дозволи – наприклад, доступ до списку користувачів можуть мати лише адміністратори.

Також, при побудові viewset-класів треба додатково визначити класи-серіалізатори моделей, у метаданих яких (вкладений клас `Meta`) ми вказуємо модель, що підлягає серіалізації, та поля, які включати до відображення. Відповідні шаблони посилань для звернення у веб-застосунку ми реєструємо в `urls.py` застосунку.

Таким чином, було визначено класи `UserViewSet`, `UniversityGroupViewSet`, `TaskViewSet` та `ModuleViewSet`. Приклад звернення за одним із таких ендпоїнтів (`/tasks`), що повертає список задач, наведено нижче:

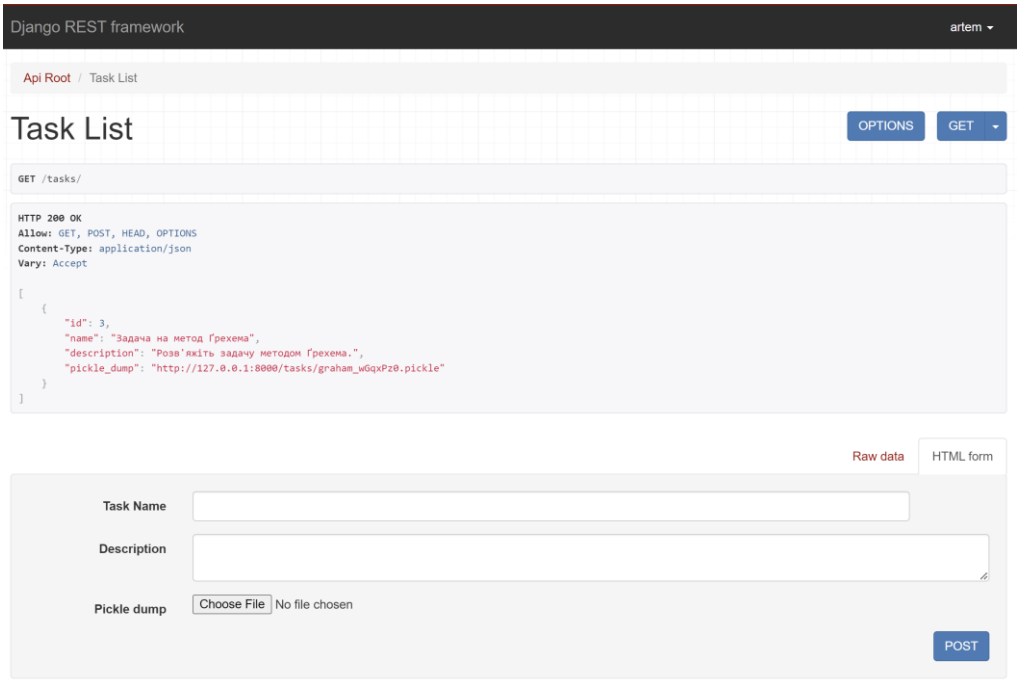


Рисунок 2.2 – Ендпоїнт «Задачі»

Заповнивши та надіславши форму, зображену на рисунку, можна додати до бази даних новий екземпляр задачі. Якщо ж замість ендпоїнту `/tasks` звернутися до `/tasks/3`, отримаємо сторінку задачі з ідентифікатором 3:

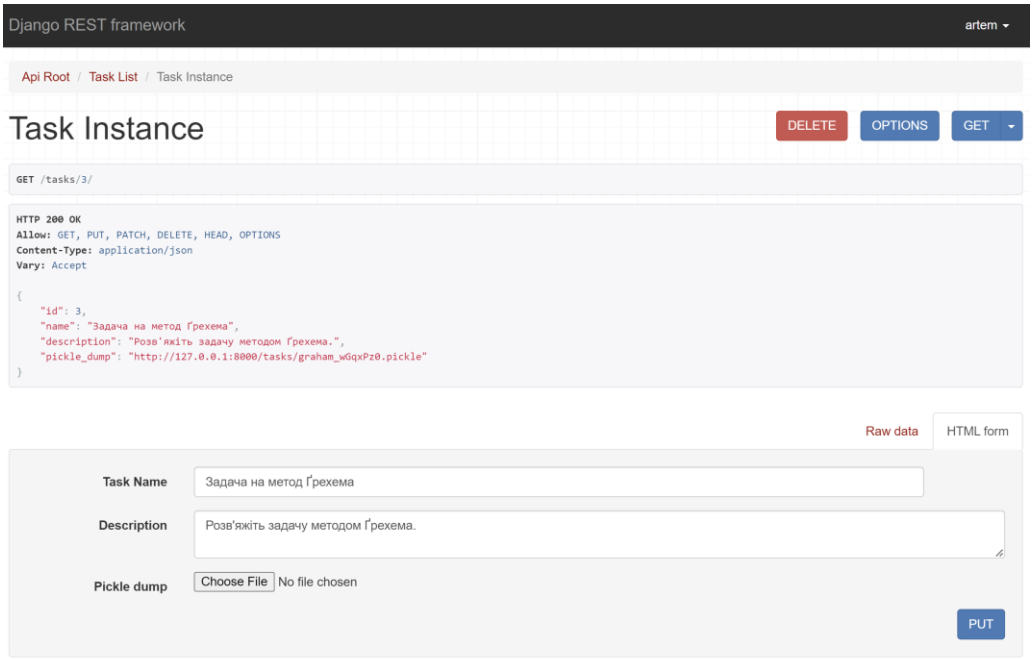


Рисунок 2.3 – Ендпоїнт «Задача»

У формі можна ввести нові значення полів, і при її відправлення у базу даних надійде оновлений екземпляр задачі.

Для задач також визначимо окремо звичайне API-представлення `TaskAPIView`, що успадковується від Django-класу `APIView`. Саме через `TaskAPIView` буде здійснюватися оцінювання. Для цього визначимо методи `get` та `post`. Перший здійснює спробу десеріалізувати задачу з ідентифікатором `id` – аргументом методу `get`. За невдалої спроби (такої задачі в базі не існує) повертається помилка 404 (Not Found), інакше – ми повертаємо відповідь із серіалізованою pydantic-умовою задачі. Другий метод приймає запит `request` із відповідями студента на задачу, аналогічним чином намагається отримати задачу з даним ідентифікатором і, у випадку успіху, повертає сумарну оцінку та розподіл оцінок – для цього необхідно задати поле `grader` класу оцінювання в класі-нащадку TaskAPIView для конкретного типу задач. У даній роботі реалізовано API-представлення для задач на метод Ґрехема – `GrahamTaskAPIView`, відповідно поле `grader` має значення `GrahamGrader`.

Тепер, маючи REST API, ми можемо звертатися до нього з клієнтського боку – фронтенду.

2.2 Фронтенд

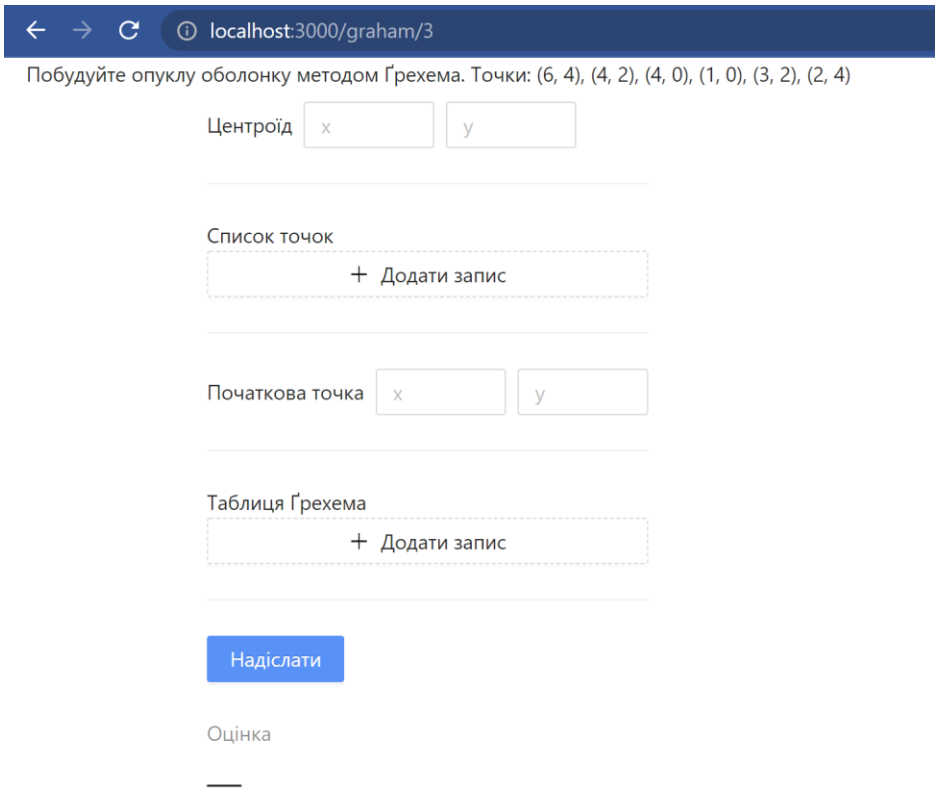
Наразі найпопулярнішим фронтенд-фреймворком на мові JavaScript є React, розроблений компанією Facebook [15, 16], який і було використано в даній роботі. React пропонує формат так званих *функціональних/класових компонентів* – JavaScript-функцій або класів відповідно, що повертають інтерактивні React-елементи веб-сторінки в HTML-подібному форматі.

Усі необхідні компоненти та їхню поведінку можна визначити в папці проєкту `src/components`. Широкий набір базових компонентів доступний у React-бібліотеці Ant Design, яку і було використано в даному проєкті.

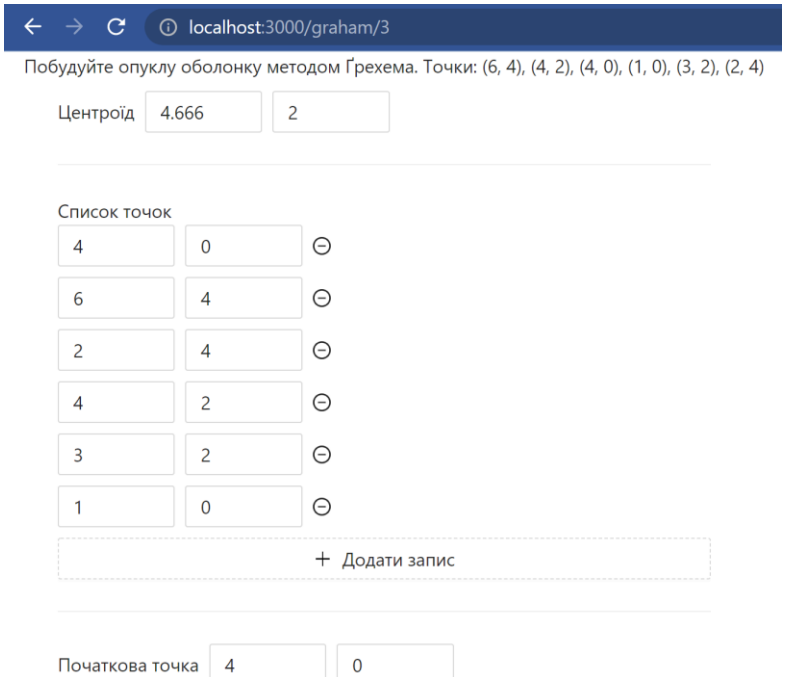
Для отримання/відправки даних з/на бекенд використовується метод `fetch`, параметром якого є посилання (ендпоїнт для звернення) та тип запиту (GET, POST тощо).

Після всіх налаштувань проєкт можна запустити за допомогою команди у терміналі `npm start` (для цього потрібно мати попередньо встановлене середовище виконання Node.js).

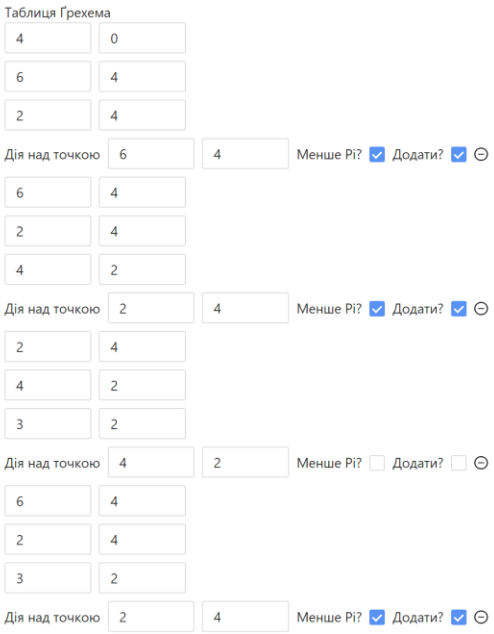
Наведемо приклад розв’язання методу Ґрехема на відповідній сторінці:



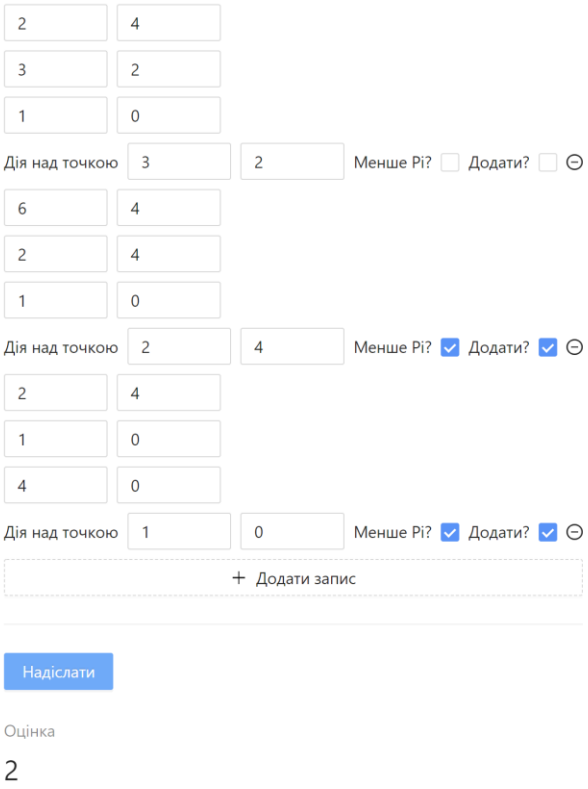
а)


localhost:3000/graham/3
Побудуйте опуклу оболонку методом Ґрехема. Точки: (6, 4), (4, 2), (4, 0), (1, 0), (3, 2), (2, 4)
Центроїд 4.666 2
Список точок
4 0
6 4
2 4
4 2
3 2
1 0
Додати запис
Початкова точка 4 0

б)


Таблиця Ґрехема
4 0
6 4
2 4
Дія над точкою 6 4 Менше Pi? Додати?
6 4
2 4
4 2
Дія над точкою 2 4 Менше Pi? Додати?
2 4
4 2
3 2
Дія над точкою 4 2 Менше Pi? Додати?
6 4
2 4
3 2
Дія над точкою 2 4 Менше Pi? Додати?

в)



г)

Рисунок 2.4 – Розв’язання задачі на метод Ґрехема: а – умова та початковий вигляд форми; б – центроїд, впорядкований список точок та початкова точка; в, г – таблиця обходу Ґрехема та результат – два бали із двох можливих.

Веб-сайт, як і використану в ньому бібліотека для оцінювання, викладено на GitHub під ліцензією GNU General Public License v3.0.

Детальніше з фронтенд-кодом можна ознайомитися за посиланням [17].

ВИСНОВКИ

1) В результаті роботи з використанням сучасних засобів програмування розроблено та викладено на GitHub вільно поширювані бібліотеку для оцінювання CGMark та прототип веб-сайту [18];
2) Надано можливість розширювати бібліотеку для оцінювання та веб-сайт алгоритмічними задачами;
3) Створені бібліотека та веб-сайт відкривають перспективи подальшого доповнення бази алгоритмів іншими методами з курсу «Обчислювальна геометрія та комп’ютерна графіка».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Moodle – Open-source learning platform [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://moodle.org/.
2. Moodle – Вікіпедія, вільна енциклопедія [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Moodle.
3. The GNU General Public License v3.0 - GNU Project - Free Software Foundation [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.gnu.org/licenses/gpl-3.0.en.html.
4. pydantic [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://pydantic-docs.helpmanual.io/.
5. PEP 484 – Type Hints [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://peps.python.org/pep-0484/.
6. Generators – Python Wiki [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://wiki.python.org/moin/Generators.
7. Design Patterns: Elements of Reusable Object-Oriented Software / [Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides]. – Б. : Addison-Wesley, 1994. – 395 с.
8. Dependency inversion principle – Wikipedia, the free encyclopedia [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Dependency_inversion_principle.
9. Python classmethod() [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.programiz.com/python-programming/methods/built-in/classmethod.
10. [] list comprehension — Python Reference (The Right Way) 0.1 documentation [Електронний ресурс]:[Веб-сайт] – Режим доступу

до ресурсу: https://python-reference.readthedocs.io/en/latest/docs/comprehensions/list_comprehension.html.

11. CGMark, CG & CG – Computational Geometry & Computer Graphics [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://github.com/OGKG/CGMark.
12. functools – Higher-order functions and operations on callable objects [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://docs.python.org/3/library/functools.html.
13. Top 13 Python Web Frameworks to Learn in 2020 [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://steelkiwi.com/blog/top-10-python-web-frameworks-to-learn/.
14. MVC Pattern and Django – Django 1.10 Tutorial – OverIQ.com [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://overiq.com/django-1-10/mvc-pattern-and-django/#:~:text=Django%20MTV,and%20controllers%20are%20called%20views.
15. React – A JavaScript library for building user interfaces [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://reactjs.org/.
16. 10 Most Popular Web Frameworks 2022 [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://makeanapplike.com/most-popular-front-end-web-frameworks-for-developments/.
17. CGFrontend, CG & CG – Computational Geometry & Computer Graphics [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://github.com/OGKG/CGFrontend.

18. CG & CG – Computational Geometry & Computer Graphics [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://github.com/OGKG/