МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н.Каразіна
Факультет математики і інформатики
Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

бакалавр

на тему проєкт веб-сервісу розрахунку режиму фрезерування кінцевими фрезами

Виконав: студент 4 курсу, групи МФ-41
спеціальність 122 «Комп'ютерні науки»
освітньо-професійна програма
«Інформатика»

Радченко К. С.
(прізвище та ініціали)
Керівник Фролов В. В.
(прізвище та ініціали)
Рецензент Доцент кафедри теоретичної на прикладної системотехніки,
факультету комп'ютерних наук,
к. т. н. Бакуменко Н.С.
(прізвище та ініціали)

Харків – 2024 року

ЗМІСТ

ВСТУП

Мета роботи:

Запровадити веб-сервіс для розрахунку режиму фрезерування кінцевою фрезою, який дозволить підвищити ефективність розрахунків режиму фрезерування кінцевою фрезою за рахунок використання математичних моделей, на основі інтерполяції, які дозволяють більш гнучко працювати з табличними даними, спрощуючи кінцевому користувачу розрахунок режиму фрезерування. Основними завданнями є:

1. Розробити представлення даних із бази даних у вигляді математичної моделі на основі інтерполяціїї нормалізованих даних.

2. Розробити архітектуру веб-сервісу, яка посприяє нормальній взаємодії користувача із кінцевими даними

3. Реалізувати веб-застосунок мовою Python для розрахунку вхідних даних та видачі їх кінцевому користувачу

4. Розробити клієнтський додаток на операційній системі Android мовою програмування Kotlin, для передачі вхідних даних від кінцевого користувача до веб сервісу, на якому проводяться відповідні розрахунки, з подальшим виведенням даних у зрозумілому вигляді

Актуальність теми:

Проаналізувавши існуючі наразі калькулятори режиму фрезерування для нашого типу фрези, я виявив деякі недоліки через які дані програми мають гіршу ефективність, а саме:

Не всі розрахункові програми мають додатки для різних операційних систем, таких як Windows OS, Mac OS, IOS, Android і т.д, що не дає достатньої мобільності у просторі. Сервіс мусить бути або встановлений на кожний станок (що є неможливим, так як більшість станків можуть бути необладнаними системними можливостями), або має бути присутній комп'ютер біля кожної машини, або має бути оператор, яких розраховує режим для кожної машини самостійно, що є довготривалим процесом для тяжкої промисловості.

Другим і головним недоліком є те, що у всіх сервісів розрахунку розрахунок табличних значень, таких як питома сила різання, відсутній, тобто працівнику фрезерувальної машини необхідно вводити дані вручну, що в свою чергу може привести до помилок розрахунку, спираючись на людський фактор, через що фрезерувальник мусить повторно вводити дані, що в свою чергу значно сповільнює процес свердлення деталей, що, особливо у поточний час воєнного стану в Україні може призвести до серйозних складнощів, так як значна частка часу, яку можна автоматизувати, буде витрачена на постійне корегування інформації.

Виходячи із вище перерахованих фактів випливає актуальність моєї роботи та обраних мною цілей, так як вони торкаються сучасної ситуації воєнного стану, також вона на пряму відноситься до галузі важкої промисловості, яка прямо позначається на темпах розвитку економічної ситуації будь-якої країни.

Методи дослідження:

Для дослідження ми використовували мову програмування Python через те що вона володіє, на мою думку, найкращим набором інструментів для дослідження різних математичних моделей, отримання та візуалізації даних від цих моделей.

Робота складається зі вступу, 4 глав, висновків кожної задачі, загальних висновків, списку використаної літератури, додатків. Зміст роботи висвітлено на 43 сторінках основного тексту.

ОСНОВНА ЧАСТИНА

РОЗДІЛ 1

1.1 Порівняння існуючих технологій

Я вивчив інтерфейси декількох наразі існуючих калькуляторів Walter, Sandvik Coromant та Seco.

- Walter

  Вивчивши даний калькулятор було виявлено, що версія андроід та іос додатків є застаріло та не підтримується на нових девайсах. Нижче на рис 1.1 можна ознайомитись з базовим інтерфейсом цього калькулятора

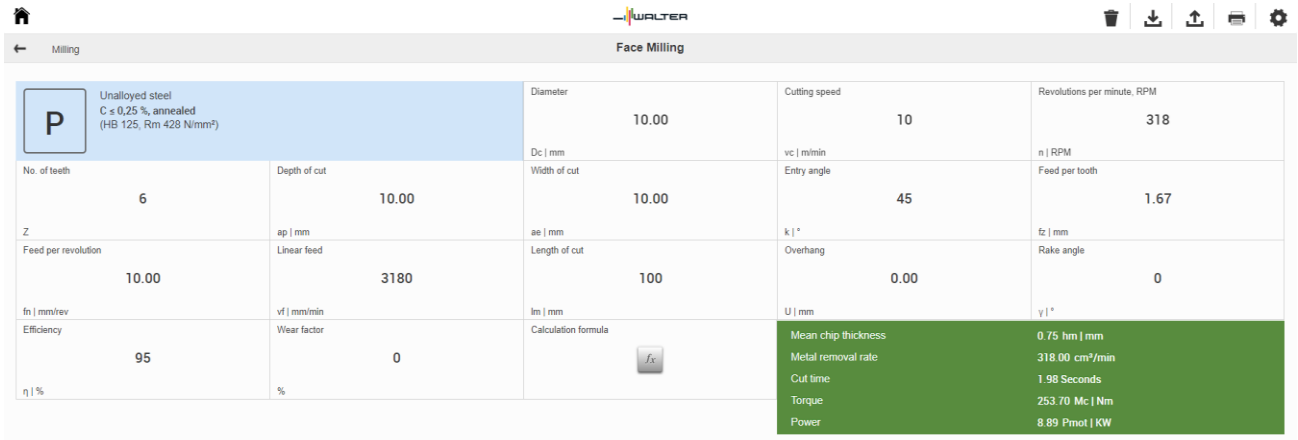


рис. 1.1 інтерфейс калькулятора Walter

- Seco

Ознайомившись з даним додатком можна відмітити наявність функціонуючої андроід аплікації, яка підтримує базові розрахункові операції. Але вона також не розраховує табличні дані самостійно, що не робить дану програму ідеальною. Із базовим інтерфейсом можна ознайомитись на рис 1.2

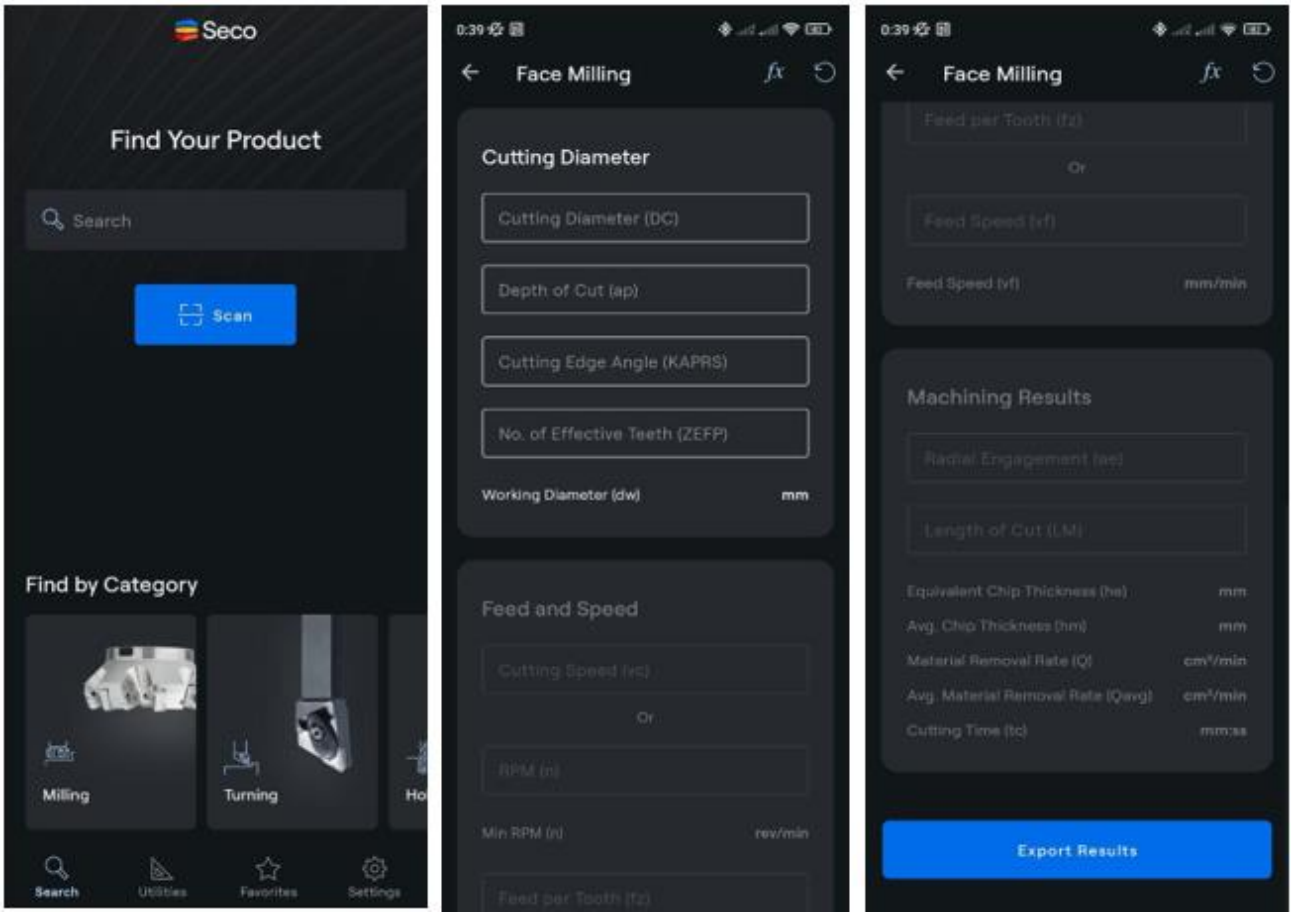


рис. 1.2 інтерфейс калькулятора SECO

- Sandvik Coromant

  Вивчивши інтерфейс та наданий ним функціонал, було зафіксована відсутність будь-яких аплікацій, окрім веб додатка, також, як і всі інші вище перераховані калькулятори він не надає нам результатів розрахунків табличних значень. Інтерфейс програми можна побачити на рис 1.3

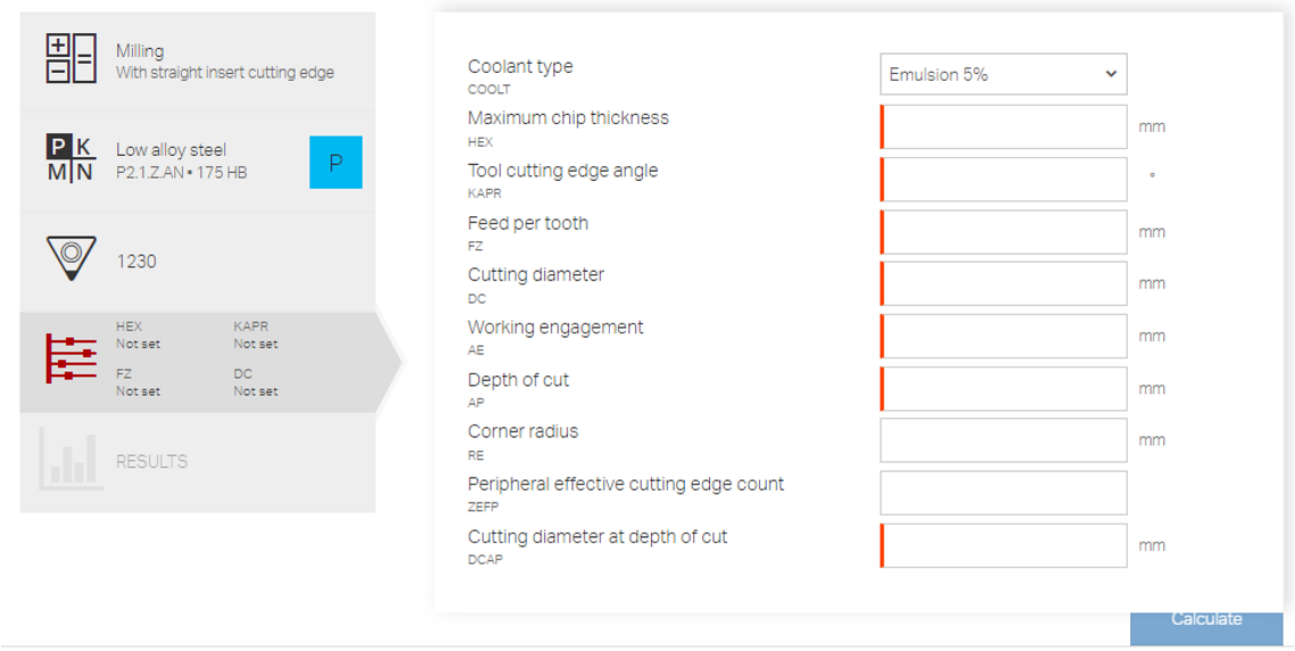


рис. 1.3 інтерфейс калькулятора Sandvik Coromant

Порівнявши всі ці калькулятори, я виніс їх загальну характеристику у дану таблицю:

Калькулятор	Базовий розрахунок	Доступність на різних девайсах	Розрахунок табличних даних
Seco	+	+	-
Sandvik Coromant	+	-	-
Walter	+	-	-

1.2 Формулювання мети роботи, задач та обґрунтування актуальності теми

Метою моєї роботи є розробка ПО, яке надасть розрахунок режиму фрезерування із застосування математичних моделей для обробки табличних даних.

Вивчивши інші програми, я поставив такі головні задачі, які будуть виділяти наш калькулятор на фоні інших.

Задача 1 – Обрати найкращий математичний метод розрахунку табличних даних на основі вхідних даних.

Задача 2 – розробити архітектуру майбутньої аплікації, в якій буде прорахована можливість розробки додатків, працюючих на різних операційних системах.

Задача 3 – розробити клієнтську частину програмного забезпечення для зручного проведення розрахунків.

Актуальність моєї роботи визначається зв’язком із важливими галузями економіки України, також у нижче приведеній таблиці видно, що моя програма матиме більше переваг аніж аналоги

Калькулятор	Базовий розрахунок	Доступність на різних девайсах	Розрахунок табличних даних
Seco	+	+	-
Sandvik Coromant	+	-	-
Walter	+	-	-
Моя програма	+	+	+

РОЗДІЛ 2

2.1 Вибір математичної моделі

Обирав необхідну модель за такими критеріями:

1. Швидкість виконання – нам потрібна швидка модель, яка би не втрачала точність в межах обмеженої вибірки даних, так як табличні дані не можуть виходити за межі наданих меж

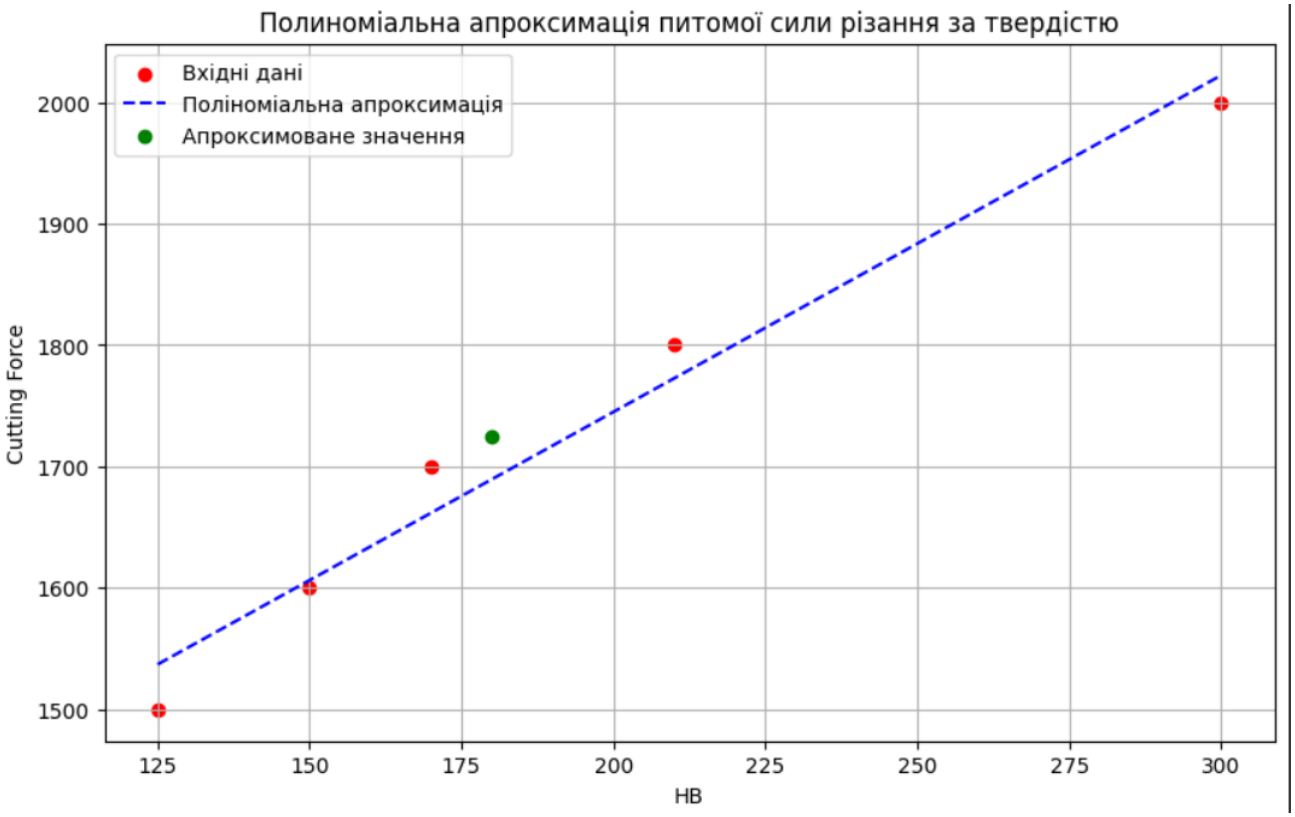
2. Точність підрахунків – модель мусить мати мінімальну похибку при розрахунках на обмеженій вибірці даних

У своїй роботі я розгляну різні моделі для підрахування даних.

Для тестування моделі я візьму тестовими даними такі числа питома сила різання (1500, 1600, 1700, 1800, 2000) та твердість по Брінелю (125, 150, 170, 210, 300) відповідно. Виходячи з таких даних можна побачити, що для твердості в 180 немає прямого табличного значення, тому необхідно використовувати саме математичну модель для підрахунку відповідного значення. Дані для математичних моделей я нормалізував за допомого Python класу MinMaxScaler бібліотеки sklearn.preprocessing до діапазону значень від 0 до 1

Розглянемо поліноміальну апроксимацію поліномів різних ступенів:

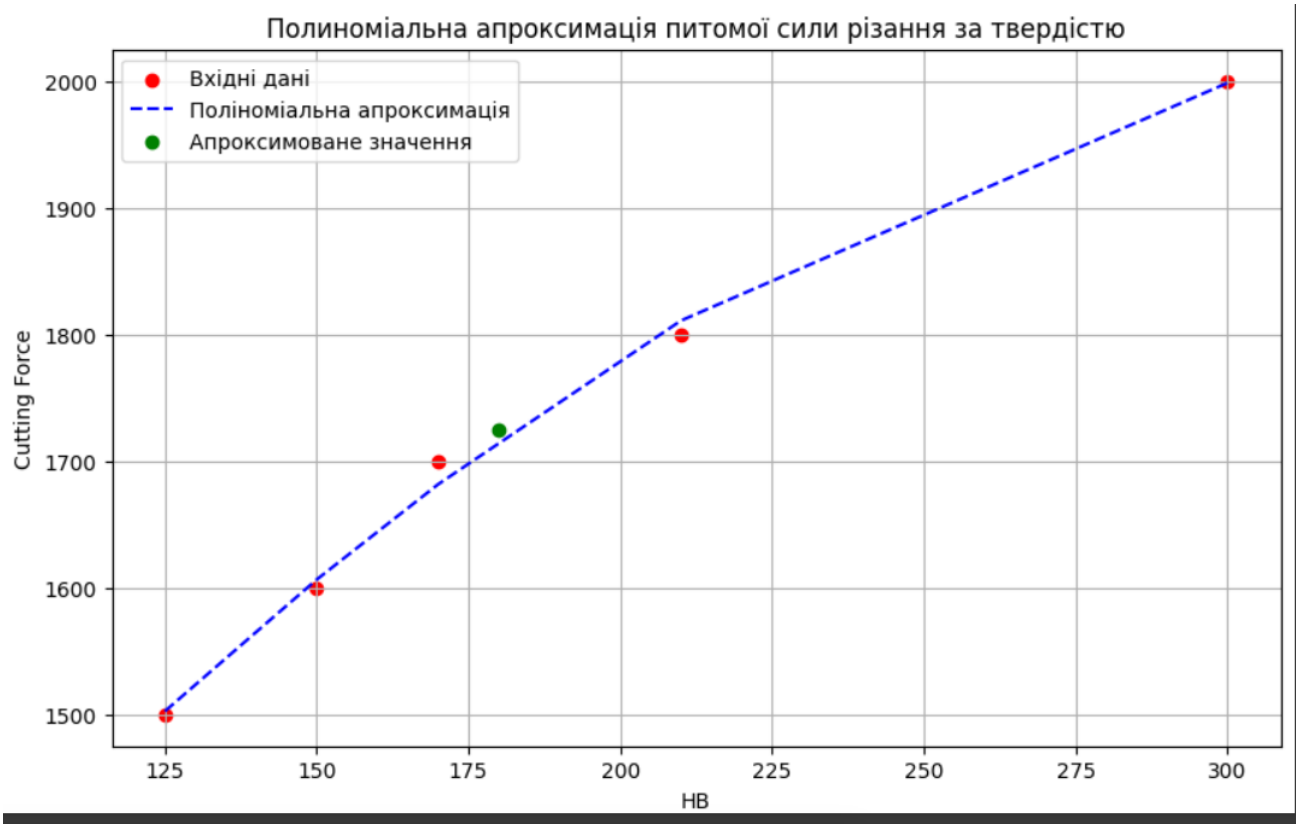
Нижче можна побачити графік розрахунку для поліноміальної апроксимацію для поліному першого ступеня із поліномом 0.9704 x + 0.07404:



Оцінка даного метода RMSE=28.67, SSE=4110.58, R^2=0.97 оцінка часу 0.0005409717559814453.

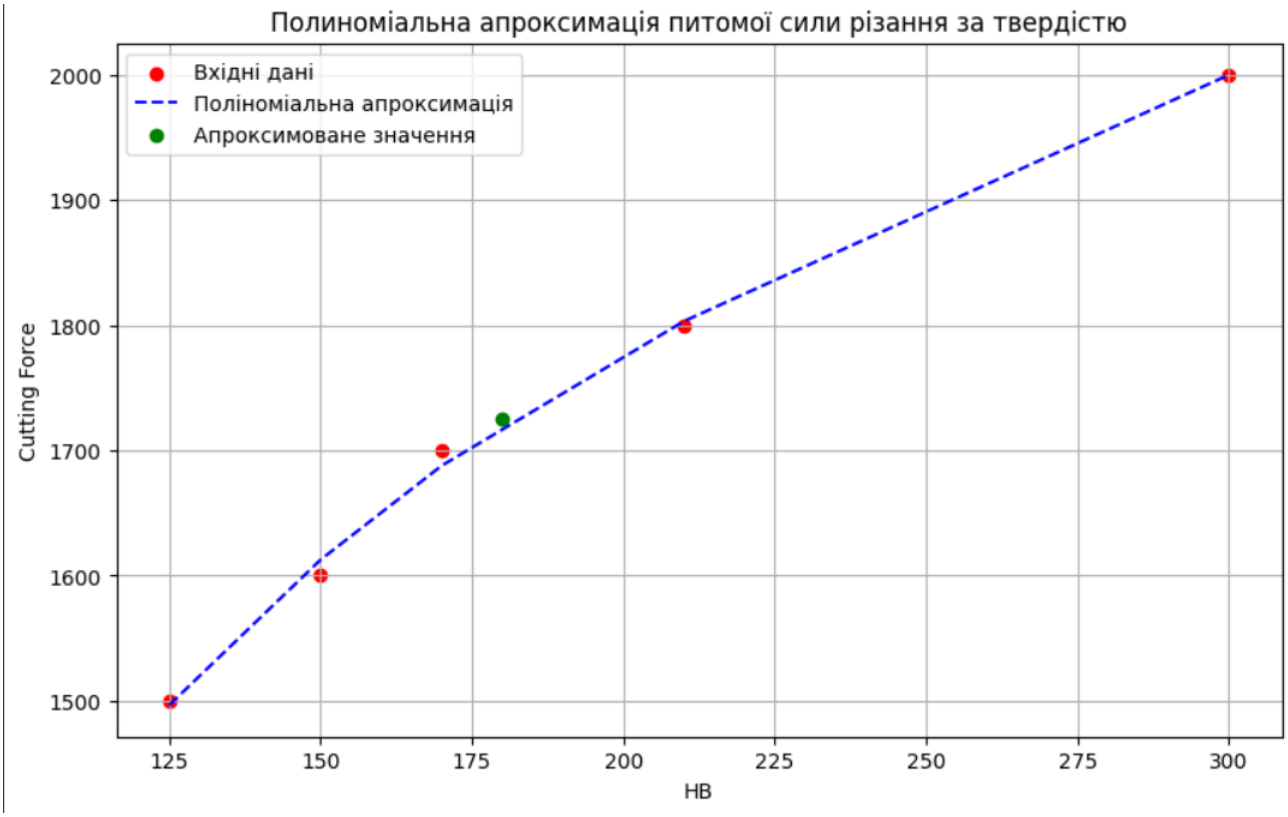
Наступний графік буде для поліному третьої ступені із поліномом

$-0.54x^2 + 1.532x + 0.005042$



Вже на другій степені можна побачити значне покращення точності функції, оцінка для даної моделі : RMSE=10.05, SSE=505.50, R^2=1.00, Time=0.0004248619079589844 середня квадратична помилка знизилась вдвічі, коли сума помилок в 8 разів.

Остання модель поліноміальної апроксимації буде для поліному 3-го ступеня, поліном – $0.651x^3 – 1.472^2x + 1.822x – 0.006825$. Відповідний графік наведено нижче:



Можна побачити, що різниця між 2 та 3 ступенем не велика, також якщо подивитись на оцінку даної моделі RMSE=8.14, SSE=331.25, R^2=1.00, Time=0.0004868507385253906, можна зауважити, що програмувати моделі із більшим значенням поліному немає сенсу, так як коефіцієнт детермінації вже дорівнює одиниці і зменшення середньої квадратичної похибки майже не відбувається.

Вибір даних за методом найменших квадратів нам не підходить, в першу чергу, через те, що цей метод потрібен для оптимізації даних серед яких можна спостерігати шум, моя ж вибірка є лінійно зростаючою.

Щодо сплайн-інтерполяції, незважаючи на її точність і гладкість через те що для побудови вибірки вона використовує умови гладкості похідних, я відмовився через зависоку складність алгоритму для невеликого об’єму даних.

Остання модель яку я розглянув це лінійна інтерполяція даних. Це один із найпростіших видів інтерполяції даних, основна ідея якої полягає у використанні прямих ліній для оцінки значень функції між відомими парами точок. Її перевага полягає в тому, що вона має високу точність в межах існуючих точок, але точність стрімко падає за її межами, але в мене немає потреби в результатах за межами вибірки, томи я ігнорую цей недолік.

Формула для лінійної інтерполяції:

Рівняння прямої яка проходить між двома точками (x_0; y_0) та (x_1; y_1) можна записати у такому вигляді:

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0}$$

Рівняння виводиться із рівняння прямої у відрізковій формі

$$y = y_0 + (y_1 - y_0)\frac{x - x_0}{x_1 - x_0}$$

Покроковий опис функції:

Першим кроком ми обираємо дві найближчі точки, між якими знаходиться точка, для якої ми шукаємо значення, після чого необхідно порахувати нахил прямої за формулою $\frac{y_1 - y_0}{x_1 - x_0}$ і останнім кроком ми за формулою прямої шукаємо необхідне значення.

Ознайомившись із лінійною інтерполяцією, я вивів такі переваги та недоліки:

Переваги:

Вона проста в реалізації, із чого випливає достатньо низький необхідний час для проведення розрахунку, також своєю простотою дана інтерполяція не затребує великої кількості пам’яті машини. Лінійна інтерполяція дуже точно

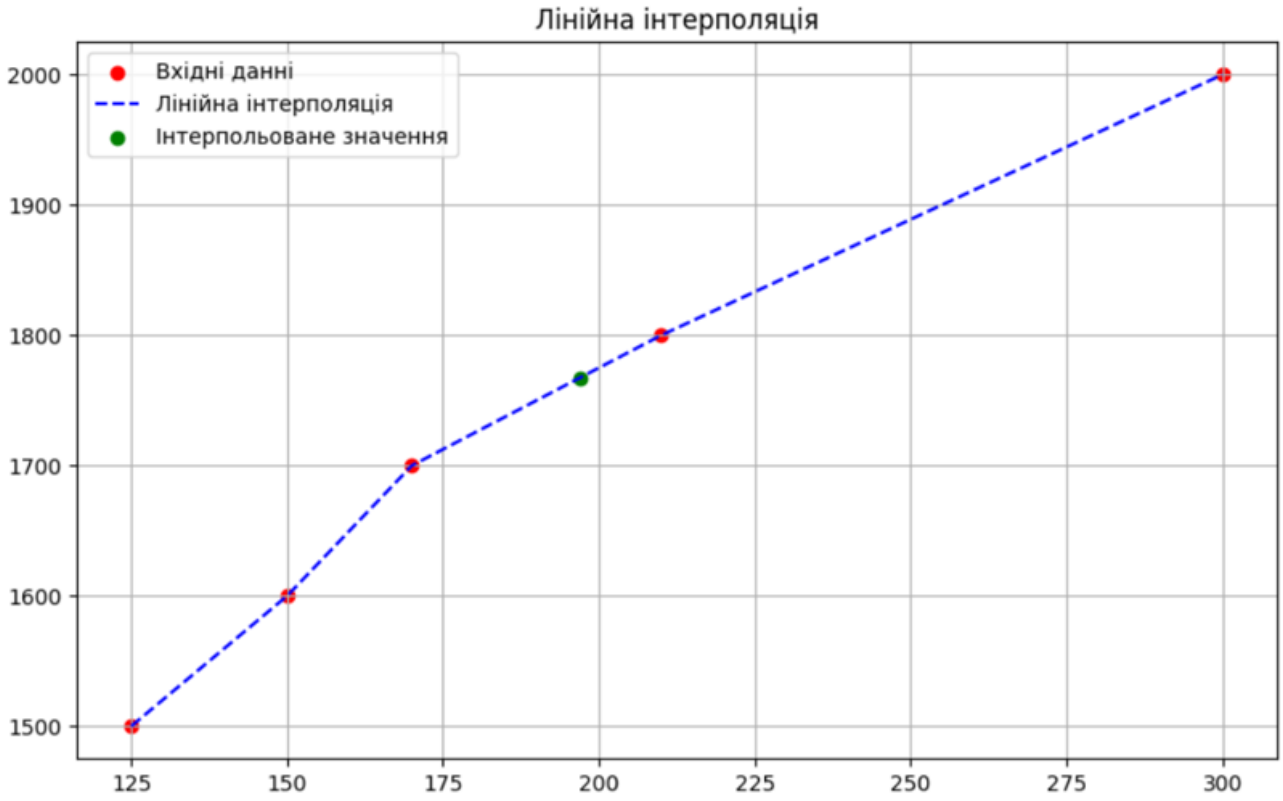
вираховує значення для вибірки даних які походять на лінійну залежні, що є саме моїм випадком.

Недоліки:

Із недоліків можна виділити обмеженість, функція ефективна лише коли зміни між відомими точками є лінійними або наближеними або якщо є різкі зміни між точками.

В моїй роботі всі дані є лінійно залежними, через що даний метод найбільше підходить до вирішення моєї задачі.

Нижче буде наведено графік для лінійної інерполяції:



Також математична оцінка для відповідної інтерполяції RMSE=0.0, SSE=0, R^2=1.0, Time=0.004621982574462891, виходячи із цієї оцінки можна зробити висновок, що для нашої вибірки даних найточнішим та найшвидшим методом є лінійна інтерполяція.

Нижче, я надам програмний код, написаний мовою Python, за допомогою якого я проводив розрахунки:

Код нормалізації даних:

```
hardness = np.array([125, 150, 170, 210, 300]).reshape(-1, 1)

cutting_force = np.array([1500, 1600, 1700, 1800, 2000])

scaler_hardness = MinMaxScaler()

scaler_cutting_force = MinMaxScaler()


normalized_hardness = scaler_hardness.fit_transform(hardness)

normalized_cutting_force =
scaler_cutting_force.fit_transform(cutting_force.reshape(-1, 1)).flatten()
```

Із цими нормалізованими даними мі працювали коли досліджували фукції

Код для лінійної інтерполяції:

```
def get_cutting_force(hardness_value):


    normalized_value =
scaler_hardness.transform(np.array([[hardness_value]])).flatten()



    interpolation_function = interp1d(normalized_hardness.flatten(),
normalized_cutting_force, kind='linear')

    interpolated_cutting_force = interpolation_function(normalized_value)
```

```
    original_cutting_force =
scaler_cutting_force.inverse_transform(interpolated_cutting_force.reshape(-1,
1)).flatten()


    return int(np.round(original_cutting_force[0]))
```

Код для поліноміальної апроксимації:

```
degree = 3  # Ступінь полінома

coefficients = np.polyfit(normalized_hardness, normalized_cutting_force, degree)

polynomial = np.poly1d(coefficients)


def getCuttingForce(hardness_value):

    normalized_value =
scaler_hardness.transform(np.array([[hardness_value]])).flatten()

    approximated_cutting_force = polynomial(normalized_value)

    original_cutting_force =
scaler_cutting_force.inverse_transform(approximated_cutting_force.reshape(-1,
1)).flatten()

    return int(original_cutting_force)
```

Код для розрахунку математичної оцінки функцій:

```
#  Оцінка  якості інтерполяції

start_time = time.time()
```

```
y_test = cutting_force

y_pred_interp = [get_cutting_force(hb) for hb in hardness.flatten()]

interp_time = time.time() - start_time


# Метрики

rmse_interp = np.sqrt(mean_squared_error(y_test, y_pred_interp))

sse_interp = np.sum((y_test - y_pred_interp) ** 2)

r2_interp = r2_score(y_test, y_pred_interp)



print(f"Интерполяция: RMSE={rmse_interp}, SSE={sse_interp}, R^2={r2_interp}, Time={interp_time}")


# Поліноміальна апроксимація

start_time = time.time()

predicted_cutting_force = scaler_cutting_force.inverse_transform(polynomial(normalized_hardness).reshape(-1, 1)).flatten()

interp_time = time.time() - start_time

# Метрики

rmse = np.sqrt(mean_squared_error(cutting_force, predicted_cutting_force))

sse = np.sum((cutting_force - predicted_cutting_force) ** 2)
```

```
r2 = r2_score(cutting_force, predicted_cutting_force)


# Вивід
print(f"Аппроксимация: RMSE={rmse:.2f}, SSE={sse:.2f}, R^2={r2:.2f}, Time={interp_time}")
```

Та код, який я використав для виводу графіків:

```
# Графік інтерполяції
plt.figure()
plt.scatter(hardness.flatten(), cutting_force, color='red', label='Data Points')
plt.plot(hardness.flatten(), y_pred_interp, color='blue', label='Interpolated')
plt.xlabel('HB')
plt.ylabel('Cutting Force')
plt.title('Интерполяция')
plt.legend()
plt.show()


# Графік апроксимації
plt.figure(figsize=(10, 6))
plt.scatter(hardness, cutting_force, color='red', label='Вхідні дані')
```

```
plt.plot(hardness,
scaler_cutting_force.inverse_transform(polynomial(normalized_hardness).reshape(-1,
1)).flatten(), color='blue', linestyle='dashed', label='Поліноміальна апроксимація')

plt.scatter([hardness_value], [cutting_force_value], color='green',
label='Апроксимоване значення')

plt.xlabel('HB')

plt.ylabel('Cutting Force')

plt.title('Полиноміальна апроксимація питомої сили різання за твердістю')

plt.legend()

plt.grid(True)

plt.show()
```

2.2 Висновки

Спираючись на інформацію, отриману із розділу 2.1, я зробив відповідні висновки:

В даному розділи наша задача була обрати підходящу математичну модель для розрахунку підходящих значень із вибірки табличних значень. Виходячи із цього, порівнявши різні модель на точність, швидкість, та простоту моделі я зупинився на лінійній інтерполяції, так як вона є найточнішою та найшвидшою для обмеженої лінійно залежної вибірки даних, яка в мене реалізована, можна затверджувати, що перша поставлена мною задача виконана.

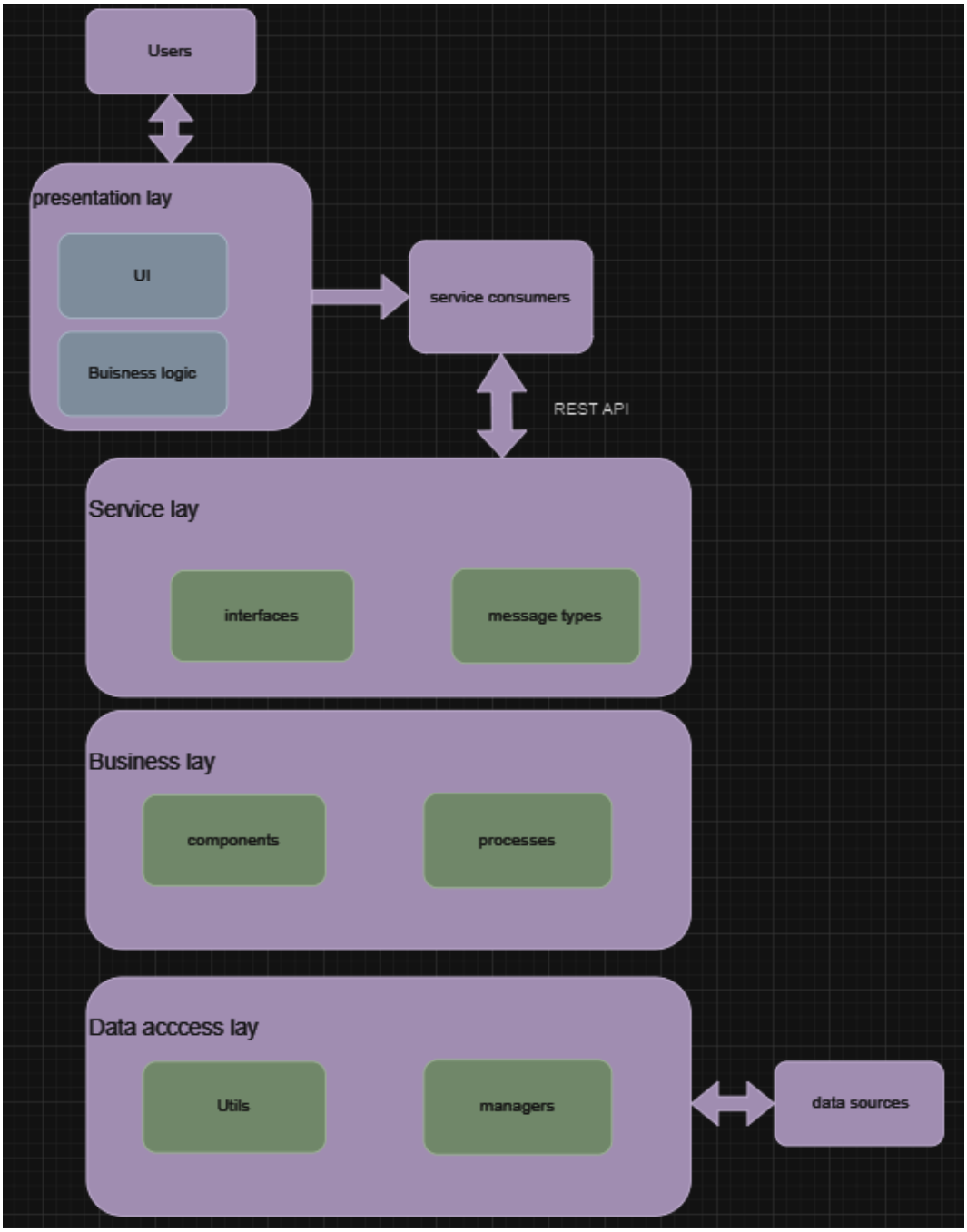
РОЗДІЛ 3

3.1 Архітектура веб сервісу

Архітектуру мого додатка можна розділити на чотири основні шари:

1. Шар доступу до даних – відповідає за отримання вибірки табличних даних та передачі розрахованих значень до наступного шару
2. Шар Бізнес логіки – отримує всі необхідні дані для розрахунку режиму фрезерування та передачі їх до наступного шару
3. Шар сервісу – обгортка над шаром з бізнес логікою, яка займається обміном даними між користувачем та шаром бізнес логіки
4. Шар клієнта – останній шар, який запроваджує зручний інтерфейс, та можливість відправки даних до шару сервісів для кінцевого користувача

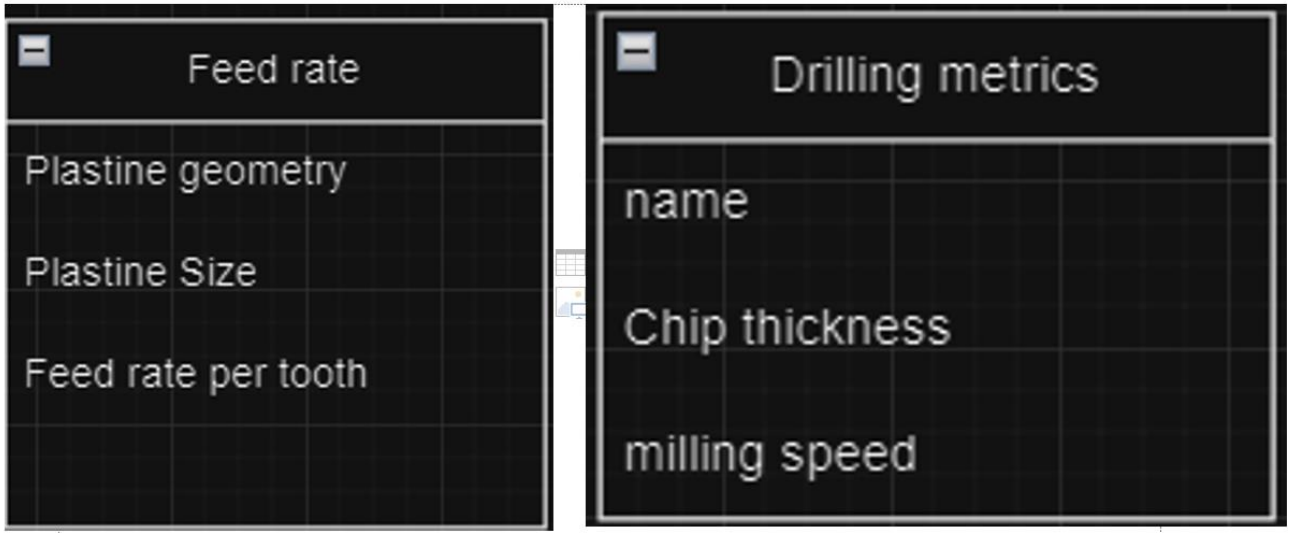
Нижче я наведи стислу схему архітектури моєї аплікації:



3.2 Шар доступу до даних

У своїй програмі для створення бази даних, я використав бібліотеку, яку надає моя середа розробки (IDE) repl.it, назва бібліотеки replit.db, вона надає можливість простою обробки табличних даних

Шар являє собою дві таблиці з даними, нижче продемонструю стислу схему нашої бази даних:



Перша таблиця репрезентує собою відношення геометрії пластини та розміру до відповідної подачі на зуб. Таблиця складається з рядку геометрії пластини у форматі строки, розміру пластини у форматі цілого числа та подачі на зуб у форматі списку флоат, перший елемент якого рекомендоване значення, а два наступні – мінімальне і максимальна відповідно.

Для знаходження необхідного значення, я застосував досить простий алгоритм, спочатку я перевіряв всі дані і знаходив ті значення подачі, які відповідали нашій геометрії пластини, після чого відбирав лише те, яке підходило нам по значенню розміру пластини.

Друга таблиця репрезентує собою необхідні дані для пошуку швидкості різання. Ця таблиця складається із назви матеріалу ріжучою частини у форматі строки, товщини стружки – список із трьох дробових чисел та швидкості різання, списку списків із трьох відповідних швидкостей.

Алгоритм пошуку такий, за ім’ям матеріалу ріжучої частини ми обираємо необхідний стовбець товщин стружки та списків трійок швидкостей, далі за

допомогою обраної мною математичної моделі лінійної інтерполяції, та завчасно знайденої питомої сили різання мі обираємо підходящу трійку швидкостей, після чого у нас залишається два лінійно залежних списків точок – товщин стружки та швидкостей різання та інтерполюємо це за значенням рекомендованої подачі на зуб, так як максимальна товщина стружки буде дорівнювати саме їй.

Нижче буде наведений код для знаходження питомої сили різання:

```
 def getCuttingForce(hardness_value):
  normalized_value = scaler_hardness.transform(np.array([[hardness_value]])).flatten()
  interpolated_cutting_force = np.interp(normalized_value, normalized_hardness.reshape(-1,), normalized_cutting_force)
  original_cutting_force = scaler_cutting_force.inverse_transform(interpolated_cutting_force.reshape(-1, 1)).flatten()
  return int(original_cutting_force)
```

код для знаходження подачі на зуб:

```
def getFeedRatePerTooth(geometry:str, size) -> list[float]:
    rows = []
    for i in db["feed_rate"]:
        if (i[0] == geometry):
            rows.append(i)
```

```
    for i in rows:
        if (i[1] == size):
            return [i[2], i[3][0], i[3][1]] #REQ -> Min -> Max
```

код для знаходження трійки швидкостей методом лінійної інтерполяції:

```
def get_cutting_speeds(cutting_force_value, cutting_speeds):
    normalized_value = scaler_cutting_force.transform(np.array([[cutting_force_value]])).flatten()
    interpolated_speeds = np.zeros(3)
    for i in range(3):
        interpolated_speeds[i] = np.interp(normalized_value, normalized_cutting_force, cutting_speeds[:, i])
    return interpolated_speeds
```

для знаходження оптимальної швидкості:

```
def get_optimal_speed(cutting_force_value, max_chip_thickness, normalized_thickness, speeds_list):
  speeds = get_cutting_speeds(cutting_force_value, speeds_list)
  normalized_thickness_value = scaler_thickness.transform(np.array([[max_chip_thickness]])).flatten()
  optimal_speed = np.interp(normalized_thickness_value, normalized_thickness, speeds)
  return  int(np.round(optimal_speed[0]))
```

3.3 Шар бізнес логіки

В даному шарі відбуваються всі основні розрахунки режиму фрезерування кінцевою фрезою. Цей шар отримує дані від шару сервісів, робить запит на отримання відповідних даних до менеджера бази даних, розраховує все за формулами, після чого повертає сервісам готову відповідь. В результаті розрахунків шар надає такі поля у якості відповіді:

- ширину перекриття – відношення ширини фрезерування до діаметру самої фрези, розраховується за формулою $\frac{a_e}{D_c}$, де a_e – ширина фрезерування, D_c – діаметр фрези
- подачу на зуб, мм/зуб , ми отримуємо від менеджера бази даних = f_z
- швидкість різання v_c, ми отримуємо від менеджера бази даних
- частота обертання шпинделя об/хв, розраховується за формулою $n = 100\frac{V_c}{\pi * D_c}$
- хвилинна подача мм/хв
- швидкість зняття метала $см^3$/хв
- питома міцність різання, ми отримуємо із бази даних = k_{c1}
- максимальна товщина стружки дорівнює рекомендованій подачі на зуб
- середня товщина стружки при зміщеній установці фрези розраховується за даною формулою $h_m = \frac{114.7 * f_z * \sin(Kr * \deg) * \frac{a_e}{D_c}}{90 + \text{asin}(\frac{a_e - \frac{D_c}{2}}{\frac{D_c}{2}})}$
- також ми розраховуємо питому силу різання у Ньютонах за формулою $k_c = k_{c1} * h_m^{-mc} * (1 - \frac{\gamma}{100})$
- міцність різання кВт, розраховується за формулою $P_c = \frac{Q * k_c}{60 * 10^3 * \mu_m}$
- поперечний переріз зрізу мм2, розраховуємо за формулою $A = \frac{a_e * a_p * f_z * z_n}{\pi * D_c}$
- міць різання Н, розраховуємо за формулою $F_c = A * k_c$

- крутний момент Нм, розраховуємо за формулою $M_c = \frac{F_c * \frac{D_c}{2}}{1000}$

також необхідно звірити значення із допустимим параметри максимальна міць різання не має бути більша за міць станка, теж саме і для крутний момент.

По закінченню розрахунків всі дані формуються у Json файл та відправляються до сервісів.

3.4 Шар сервісів

Виконує роль передавача інформації між різними шарами, ми отримуємо запит від користувача на знаходження режиму фрезерування кінцевою фрезою із даними.

Основна ідея реалізації полягає у використанні REST API для зв'язку клієнта із веб частиною додатку. Архітектура рест дуже проста, він складається із HTTP запиту, сам може анотуватися тегами GET, POST, PUT, DELETE і так далі в залежності від цілей поставлених перед конкретним запитом, за допомогою запитів клієнт надсилає дані у форматах JSON, HTML, XML, або іншому необхідному форматі до серверної частини додатка, де в свій час даний запит опрацьовується на віртуальній машині і після опрацювання даних і, при необхідності, внесення змін усередині сервера повертає відповідь у вигляді HTTP коду (базового або кастомного) із Response Body, який також може бути у форматі JSON, HTML, XML, або будь-якому іншому. Також REST API надає змогу на взаємодію із сервером програмам створеним на основі будь-якої операційної системи, дана технологія також має свої недоліки такі як відсутність оптимізації для робити в реальному часі, тобто у даної технології немає можливості налаштувати постійне підключення до сервера.

Виходячи з вище перелічено стисло підведемо характеристику REST API:

Переваги:

- Зручний та простий інтерфейс
- Гнучкий у використанні з різних платформ
- Надає можливість налаштування клієнт-серверної архітектури

Недоліки:

- Безстановість, тобто для того щоб запит був опрацьован та не повернув ніяких помилок, треба надати йому всі необхідні дані, також сервер не зберігає стан кожного запиту
- Обмеженість HTTP, сервіс надає можливість зв’язку у вигляді запит-відповідь
- Не оптимізован для роботи в реальному часі, сервіс не має реалізації постійного підключення до серверу через постійно відкрите з’єднання

Причини вибору:

У своїй роботі я обрав REST API як основний сервіс зв’язку між клієнтом та сервером через те що у мене в програмі немає необхідності постійного підключення до сервера. Мені був потрібен сервіс який легко впроваджується для програм будь-якого виду, та можливе його використання на будь-якій операційній системі.

Структура сервісу:

Для підтримки серверу в активному стані я скористався безкоштовним деплоінгом який є в обраному мною редакторі коду replit. Веб програма запускається також із допомогоюPython бібліотеки Flask. Сам додаток складається із одного POST запиту із ендпоінтом /count_parameters/ у який користувач має передати в якості BODY параметру JSON файл, який має таку структуру:

{

```
"bilet": {
    "geometry": {
        "milling_width":Int,
        "milling_depth":Int
    },
    "material": {
        "mark":String,
        "tensile_strength":Int,
        "brinnell_hardness":Int
    }
},
"machine": {
    "model":String,
    "max_speed":Int,
    "power":Int,
    "torque":Int,
    "tool_store":Int,
    "CPD":Float
},
"tool": {
    "model":String,
```

```
        "catalogue_name":String,
        "diameter":Int,
        "number_of_teeth":Int,
        "plate":String,
        "plate_height":Int,
        "plate_width":Float,
        "max_drilling_depth":Int,
        "pinnacle_radius":Float,
        "main_angle":Int,
        "plastine_geometry":String,
        "material_drilling_part":String,
        "front_angle":Int
    }
}
```

Як можна побачити із наведеного вище JSON файлу можна розділити підготовку даних клієнтом на три етапи.

Після проведення необхідних розрахунків сервіс має повернути HTTP код 200 OK із JSON файлом, який має таку структуру:

```
{
    "avg_chip_thickness_at_offset_cutter_setting": float,
    "cutting_force": float,
```

"cutting_power": float,

"drilling_speed":Int,

"feed_rate_per_tooth": List<float>

"is_cutting_force_ok": bool,

"is_torque_ok": bool,

"max_chip_thickness": float,

"metal_removal_rate":float,

"minute_feed":float,

"shear_cross_section": float,

"slab_width": float,

"specific_cutting_force":float,

"spindle_speed": float,

"torque": float

}

У відповіді сервіс надає всі дані у межах одного об’єкта.

3.5 Шар клієнта

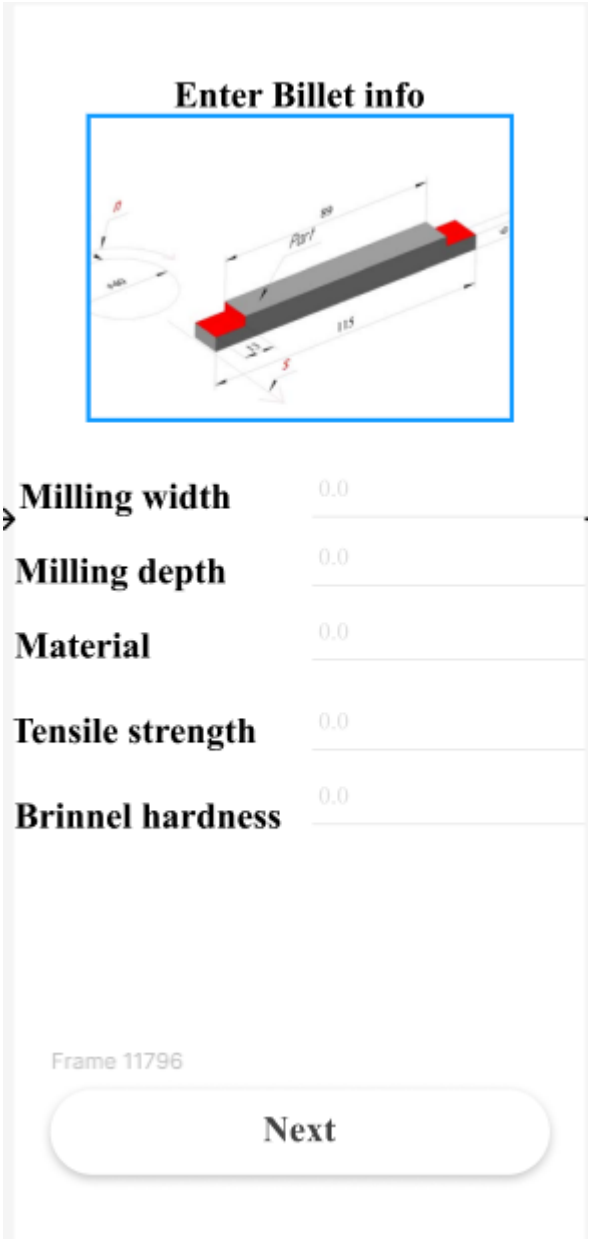
Останній шар мого додатку, може бути реалізованим різним способом, аплікація на систему Android\IOS, програма для Windows\MacOs, або веб додаток, я обрав реалізацію клієнта на базі андроїд аплікації написаною мовою Kotlin.

РОЗДІЛ 4

4.1 Опис клієнта

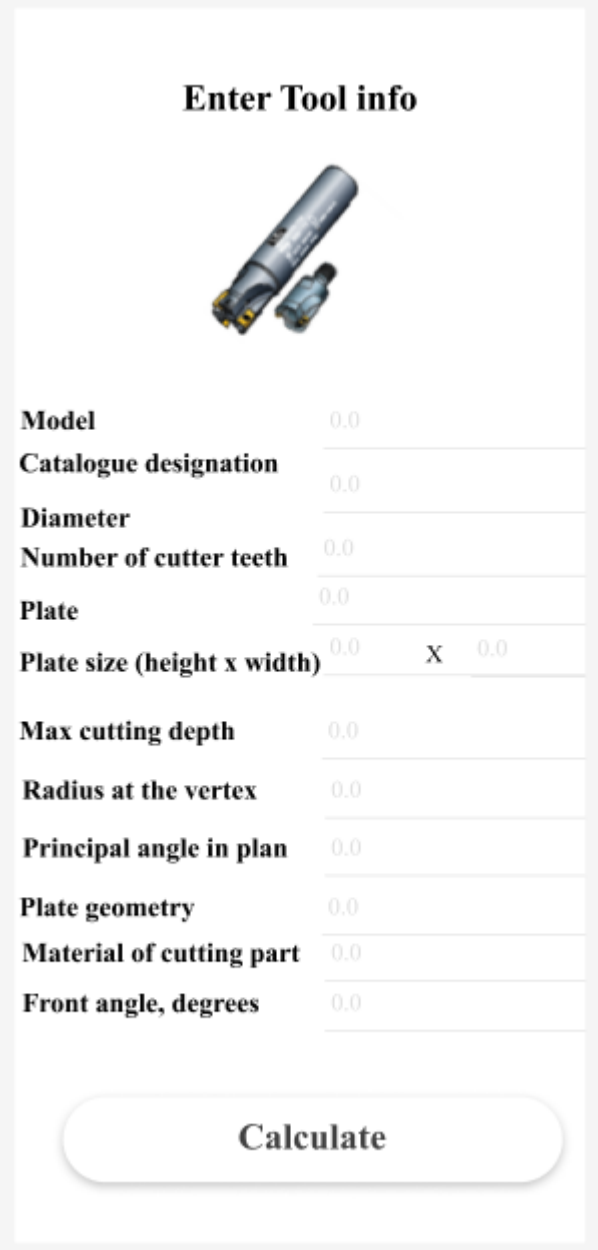
Інтерфейс Android додатка складається із чотирьох основних фрагментів, три для отримання моделей даних для відправки на сервіс, а четвертий для виводу отриманого результату.

Перший екран показано нижче:



Фрагмент не має нічого зайвого, поля для заповнення даних, зображення елементи, для якого ми вводимо дані, та кнопка переходу до наступного фрагменту, показаного нижче:

другий фрагмент збирає інформацію про станок, як можна побачити, що по наповнюваості інтерфейс не має вагомих відмінностей від минулого екрану. Інтерфейс третього фрагмента:



Третій екран збирає дані про інструмент, яким відбувається фрезерування заготовки, єдине, що його відрізняє від двох минулих фрагментів інтерфейсу – це текст кнопки змінився, натякаючи користувачу, що більше даних від нього не потребується.

Останній екран:

На останньому екрані не буде нічого зайвого, тільки інформація про розрахований режим та можливість повернутися на головний екран для проведення нових розрахунків.

Для впровадження зв’язку я використав api service на основі OkHttp бібліотеки Retrofit, так як для андроїд розробки це є актуальною бібліотекою. Код запиту виглядає даним чином:

```
@POST("/count_parameters/")
    fun countDrillingMode(
        @Body body : DataBody
    ) : Single<Answer>


    data class DataBody(
        @SerializedName("bilet")
        val bilet : Bilet,
        @SerializedName("machine")
        val machine : Machine,
        @SerializedName("tool")
        val tool : Tool
    ) {
        data class Bilet(
            @SerializedName("geometry")
            val geometry : Geometry,
            @SerializedName("material")
            val material : Material,
```

```
) {
    data class Geometry(
        @SerializedName("milling_width")
        val width : Int,
        @SerializedName("milling_depth")
        val depth : Int,
    )
    data class Material(
        @SerializedName("mark")
        val mark : String,
        @SerializedName("tensile_strength")
        val strength : Int,
        @SerializedName("brinnell_hardness")
        val hardness : Int,
    )
}
data class Machine(
    @SerializedName("model")
    val model : String,
    @SerializedName("max_speed")
    val maxSpeed : Int,
```

```
    @SerializedName("power")
    val power : Int,
    @SerializedName("torque")
    val torque : Int,
    @SerializedName("tool_store")
    val store : Int,
    @SerializedName("CPD")
    val cpd : Float,
)

data class Tool(
    @SerializedName("model")
    val model : String,
    @SerializedName("catalogue_name")
    val catalogueName : String,
    @SerializedName("diameter")
    val diameter : Int,
    @SerializedName("number_of_teeth")
    val teethQuantity : Int,
    @SerializedName("plate")
    val plate : String,
```

```
        @SerializedName("plate_height")
        val plateHeight : Float,
        @SerializedName("plate_width")
        val plateWidth : Float,
        @SerializedName("max_drilling_depth")
        val maxDrillingDepth : Int,
        @SerializedName("pinnacle_radius")
        val pinnacleRadius : Float,
        @SerializedName("main_angle")
        val mainAngle : Float,
        @SerializedName("plastine_geometry")
        val plastineGeometry : String,
        @SerializedName("material_drilling_part")
        val materialDrillingPart : String,
        @SerializedName("front_angle")
        val frontAngle : Float,
    )
}
```

4.2 Тестування програми

Для тестування програми я взяв тестові дані, для яких вже відомий вихідний результат. Тестовий флоу у нас є таким: спочатку я за допомогою

програми Postman перевірив розрахунок даних та правильний формату відповіді на запит, після чого повторив цей процес але вже із своєї андроїд аплікації.

Дані які я надавав до запиту:


```
{
    "bilet": {
        "geometry": {
            "milling_width": 13,
            "milling_depth": 4
        },
        "material": {
            "mark": "steel 45",
            "tensile_strength": 598,
            "brinnell_hardness": 197
        }
    },
    "machine": {
        "model": "SPINNER VC750",
        "max_speed": 12000,
        "power": 17,
        "torque": 108,
        "tool_store": 37,
        "CPD": 0.95
    },
    "tool": {
        "model": "CoroMill390",
        "catalogue_name": "R390-040A32-11M",
        "diameter": 40,
        "number_of_teeth": 4,
        "plate": "R390-11",
        "plate_height": 11,
        "plate_width": 3.5,
        "max_drilling_depth": 10,
        "pinnacle_radius": 0.8,
        "main_angle": 90,
```


Після відправки запиту до серверу я отримав таку відповідь:

Parameter	Value
Avg Chip Thickness at Offset Cutter Setting	0.03702550252465479
Cutting Force	466.7268867804237
Cutting Power	2.521962826813518
Drilling Speed	308
Feed Rate per Tooth	0.1,0.08,0.15
Is Cutting Force OK	true
Is Torque OK	true
Max Chip Thickness	0.1
Metal Removal Rate	50.980511371195924
Minute Feed	980.3944494460753
Shear Cross Section	0.16552114081557115
Slab Width	0.325
Specific Cutting Force	2819.7418437350275
Spindle Speed	2450.9861236151883
Torque	9.334537735608473

Звіривши кожне значення параметру із правильною відповіддю, я визначив, що всі дані були розраховані правильно.

ВИСНОВКИ

Задачею роботи є визначення математичної моделі для опрацювання табличних даних, та розробка веб-сервісу із можливістю впровадження його у додатки різних операційних систем.

Виходячи із всього, що я зробив протягом свого дослідження, я хочу підвести такі висновки за своєю роботою.

- Протягом своєї роботи я дослідив різні математичні моделі для отримання на предмет швидкості, складності алгоритму та реалізації та на точність розрахунків для обмеженої ділянки даних і обрав для свого проєкту модель лінійної інтерполяції даних через її простоту та точність в зазначених межах вибірки
- Була розроблена архітектура для веб сервісу, яка складається з 4-ьох шарів, кожен з яких виконує свою роль і може обмінюватися даними лише із сусідніми шарами
- Була розроблена серверна частина додатку яка розраховує режим фрезерування для кінцевої фрези із можливістю використовувати апі із будь-якої аплікації, через використання REST технології
- Був розроблений клієнт на базі Android додатку для взаємодії оператора станка або іншого користувача із сервісом та протестовано функціонал на тестових даних

Спираючись на зазначені висновки, можна стверджувати, що цілі дослідження були досягнуті мною та отриман задовільній результат роботи.

Узагальнюючи, дана дипломна робота займається актуальною темою для галузі тяжкої промисловості, у якій різання деталей застосовується з високою частотою, а також результати можуть посприяти пришвидшенню розвитку

відповідної галузі та полегшенню роботи для людей, зв’язаних з фрезеруванням різних деталей.

Подальше використання результатів розробки програми може сприяти розвитку тяжкої промисловості країни, у якій даний сервіс буде використовуватися.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sandvik Coromant Application guide - URL: https://www.sandvik.coromant.com/en-gb/tools/tooling-systems/turning-centres-and-lathes/silent-tools/application-guide-silent-tools (Date: 05.06.2024)
2. Microsoft Application Architecture Guide, 2nd Edition. - URL:https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff650706(v=pandp.10) (Date: 05.06.2024)
3. Sam Newman Building Microservices / Newman Sam. - Boston: O'Reilly Media, Inc., 2023. - 624p.Seco. Catalog and technical guide 2024.1 Milling. - https://www.secotools.com/article/84570?language=en
4. Calculator Walter. - https://www.walter-tools.com/ru-ru/news-and-media/media-library/apps-and-software/walter-machining-calculator?authuser=0&hl=ru
5. Calculator Sandvik Coromant. - https://www.sandvik.coromant.com/en-gb/tools/tooling-systems/turning-centres-and-lathes/silent-tools/application-guide-silent-tools/milling-appl-guide
6. Calculator Seco. - https://www.secotools.com/article/114063?authuser=0&hl=ru&language=en