Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка фреймворку для створення веб- інтерфейсів для візуалізації даних і моделей машинного навчання

Виконав: студент IV курсу, групи СП-43

спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Шинкарук В.П.
(підпис) (прізвище та ініціали)

Керівник _____ Гладьо Ю.Б.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Матійчук Л.П.
(підпис) (прізвище та ініціали)

Тернопіль - 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
«__» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Студенту Шинкаруку Владиславу Петровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка фреймворка для створення веб- інтерфейсів для візуалізації даних і моделей машинного навчання

Керівник роботи Гладьо Юрій Богданович, к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «18» 04 2025 року № 4/7-335

2. Термін подання студентом завершеної роботи 21.06.2025р.

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст роботи (перелік питань, які потрібно розробити)

Вступ

1. Огляд предметної галузі.

2. Проектна частина

3. Практична частина

4. Безпека життєдіяльності, основи охорони праці

Висновки

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

1. Титулка. 2. Актуальність дослідження. 3. Мета, задачі дослідження. 4. Порівняння аналогів 5. Архітектура програмного рішення. 6. Технологічний стек 7. Розбиття API фреймворка на групи. 8. Основні елементи ієрархії класів віджетів. 9. Фрагменти програмного коду функцій. 10. Програмні форми для відображення інформації 11. Swagger UI, створений на базі згенерованої специфікації. 12. Висновки. Основні результати проведеного дослідження.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Лазарюк В.В., доц.каф. МТ		

7. Дата видачі завдання 18.04.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	18.04 – 19.04.25	*Виконано*
2.	Підбір джерел про фреймворка для створення веб-		
	інтерфейсів для візуалізації даних і моделей		
	машинного навчання	20.04 – 22.04.25	*Виконано*
3.	Опрацювання джерел про такі фреймворки	23.04 – 29.04.25	*Виконано*
4.	Проведення дослідження щодо генерації створення веб-		
	інтерфейсів для візуалізації даних і моделей		
	машинного навчання	30.04 – 04.05.25	*Виконано*
5	Розроблення програмного коду	05.05 – 12.05.25	*Виконано*
6.	Оформлення розділу «Огляд предметної галузі»	13.05 – 18.05.25	*Виконано*
7.	Оформлення розділу «Проектна частина»		*Виконано*
8.	Оформлення розділу «Практична частина»	19.05 – 24.05.25	*Виконано*
9.	Виконання завдання до підрозділу «Безпека		
	життєдіяльності, основи хорони праці»	25.05 – 31.05.25	*Виконано*
10.	Оформлення кваліфікаційної роботи		*Виконано*
11.	Нормоконтроль	01.06 – 05.06.25	*Виконано*
12.	Перевірка на плагіат	06.06 – 09.06.25	*Виконано*
13.	Попередній захист кваліфікаційної роботи	10.06 – 14.06.25	*Виконано*
14.	Захист кваліфікаційної роботи	28.06.25	

Студент _____ Шинкарук В.П.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Гладьо Ю.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025, Сторінок 53, рисунків 33, джерел 32.

Ключові слова: віджет, візуалізація, машинне навчання, фреймворк, javascript, pytnon

У кваліфікаційній роботі бакалавра спроектовано та розроблено програмне рішення для створення веб-застосунків при відсутності специфічних для розробки знань.

Проаналізовано альтернативні програмні засоби, визначено їх слабкі ти сильні сторони. Визначено основні критерії для їх порівняння.

Спроектована програмна архітектура фреймворку. При проектуванні фреймворку проведено огляд API, описано процесу взаємодії з API з метою побудови веб-інтерфейсів та взаємодії з API для створення HTTP API і технічним API.

Розроблено модулі, що відповідають за виконання коду користувача та який реагує на зміни в користувальницькому коді. Розроблено JavaScript модуль із забезпеченням маршрутизація запитів до обробників, різних віджетів та подій (оновлення екрану та різних типів повідомлень).

Розроблено модуль, котрий відповідає за генерацію HTTP API і OpenAPI специфікації на основі користувальницького коду. Забезпечено генерацію HTTP API та обробка запитів користувача, а також генерацію OpenAPI специфікації.

Створене програмний інструмент наблизить моделі машинного навчання до кінцевого користувача і спростить інтеграцію таких моделей зі сторонніми системами.

ABSTRACT

Bachelor's thesis. Ivan Puluj Ternopil National Technical University, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2025, Pages 53, figures 33, sources 32.

WIDGET, VISUALIZATION, MACHINE LEARNING, FRAMEWORK, JAVASCRIPT, PYTHON

In the bachelor's thesis, a software solution for creating web applications in the absence of specific knowledge for development was designed and developed.

Alternative software tools were analyzed, their strengths and weaknesses were identified. The main criteria for their comparison were determined.

The software architecture of the framework was designed. When designing the framework, an API review was conducted, the process of interacting with the API was described in order to build web interfaces and interact with the API to create HTTP API and technical APIs.

Modules were developed that are responsible for executing user code and reacting to changes in user code. A JavaScript module was developed to provide routing of requests to handlers, various widgets and events (screen updates and various types of messages).

A module was developed that is responsible for generating HTTP API and OpenAPI specifications based on user code. HTTP API generation and user request processing, as well as OpenAPI specification generation, are provided.

The created software tool will bring machine learning models closer to the end user and simplify the integration of such models with third-party systems.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Pprogramming Interface) – інтерфейс програмування застосунків (прикладний програмний інтерфейс).

HTTP (англ. Hypertext Transfer Protocol) – протокол передачі гіпертекстових документів.

Markdown – полегшена мова розмітки даних, яку створено з ухилом на читабельність та зручність у публікації.

ML (Machine Learning) – машинне навчання.

Віджет – елемент взаємодії (кнопка або смуга).

ПЗ – програмне забезпечення.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. ML – це вид програмної інженерії, котрий стрімко розвивається, що надає можливості створення складних систем передбачення. Воно може бути застосоване там, де є залежність від даних і немає можливості побудувати детермінований алгоритм будь-якою мовою програмування.

Створення, валідація та застосування ML -моделей вимагають особливого набору знань та навичок, для їх розробки та впровадження компанії наймають спеціально навчених людей – ML -інженерів. Для тих, хто не має спеціалізованих знань, розуміння ML -моделей та взаємодія з ними викликає різноманітні складнощі, створюючи бар'єри для зворотного зв'язку з розвитку ML -моделей, а також для встановлення довіри до ML загалом.

Наприклад, відсутність простого та зрозумілого способу взаємодії з ML -моделлю може ускладнити для експертів у галузі, під яку вона розробляється, валідацію правильності її роботи, змушуючи менеджмент орієнтуватися на метрики, зібрані інженерами на основі тестових даних. Через це на реальних даних, які часто включають шуми, артефакти або особливості, які відсутні в стандартних тренувальних даних, модель буде показувати менш точні результати. Надавши експертам спосіб взаємодії з ML -моделлю, цього можна було б уникнути, оскільки протестувавши модель на різних реальних даних з відхиленнями, вони змогли повідомити інженерів про неточності в її роботі.

Також, відсутність розуміння використаних даних та можливості взаємодії з ML -моделлю, може перешкодити встановленню довіри до моделі менеджментом, у зв'язку з чим подальше фінансування та розвиток проекту, в рамках якого було виконано завдання створення та навчання ML -моделі, може бути поставлене під питання. Тому важливо забезпечити прозорість та доступність моделі для експертів та зацікавлених сторін, щоб забезпечити успішне впровадження та використання ML- моделі в рамках проекту.

Додаткові складності під час впровадження ML -моделей становлять інтеграції з іншими системами. Самі собою ML -моделі не надають зручних способів взаємодії із зовнішніми застосунками, тому, найчастіше, одних ML -інженерів недостатньо. Компаніям доводиться витрачати ресурси команди розробки, відділу контролю якості, системних аналітиків. Якщо після впровадження результати роботи ML -моделі в реальному середовищі виявляться гіршими за очікувані, зазначені вище ресурси будуть витрачені дарма. Наявність інструменту, що дозволяє створити точки для інтеграції без необхідності наявності специфічних для серверної розробки знань, дозволило б суттєво знизити витрати на тестове впровадження та інтеграцію з ML -моделями.

Відмінним способом вирішення всіх зазначених вище складнощів може бути розробка веб-застосунку на основі ML -моделі, яка надаватиме користувачам доступ до даних, використаних при створенні ML -моделі, можливість взаємодії з ML -моделлю за допомогою елементів керування, представлених на веб-сторінках, а також можливість інтеграції з іншими системами через надання HTTP API до методів взаємодії з моделлю. Однак такий підхід все одно вимагає наявності специфічних знань, які можуть бути відсутніми в ML –інженерів.

Таким чином можна сформулювати наступну проблему - на ринку присутня потреба в інструменті, що дозволяє створювати веб-застосунки без наявності специфічних для серверної розробки та веб-розробки знань.

Метою роботи є проектування та розробка ПЗ для створення веб-інтерфейсів для візуалізації даних і ML -моделей.

Для досягнення мети потрібно вирішити наступні завдання розробки:

– API фреймворку;

– модуля, що відповідає за виконання користувальницького коду;

– модуля, що реагує на зміни в користувальницькому коді;

– JavaScript модуля, що відповідає за створення веб-інтерфейсів на основі коду користувача;

– модуля, що відповідає за генерацію HTTP API і OpenAPI специфікації на основі коду користувача.

1 ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ

Для визначення ступеня розробленості проблеми було проведено аналіз предметної галузі та виділено критерії для порівняння альтернативних рішень.

1.1 Аналіз альтернативних програмних засобів

Dash [4] - фреймворк для побудови програми для роботи з даними на мові Python (рис. 1.1). За його допомогою розробник може швидко створити веб-програми для візуалізації даних. Його розробка почалася ще в 2015 році, і з того часу він отримав широку підтримку python спільноти. Тим не менш його не можна назвати ідеальним - він підтримує тільки роботу з бібліотекою для візуалізації даних plotly, а для створення веб-інтерфейсів з його допомогою необхідно знати елементи мови розмітки HTML, так як API, що їм надається, сильно перегукується з тегами, визначеними у вищевказаній мові розмітки.

Рисунок 1.1 – Логотип фреймворка Dash

Streamlit [5] - фреймворк з відкритим вихідним кодом для мови Python, здебільшого, націлений на відображення даних та роботу з ними (рис. 1.2). Підтримує роботу з різними бібліотеками для візуалізації даних, також дозволяє створювати елементи керування для взаємодії з ML -моделлю та не вимагає знання та розуміння мови розмітки HTML. До його мінусів можна віднести наступне:

– при кожній взаємодії з елементами веб-інтерфейсу відбувається повторне виконання коду користувача;

– не надає HTTP API для роботи з моделями машинного навчання;

– має дуже обмежений функціонал в частині авторизації користувачів веб-застосунку.

Рисунок 1.2 – Логотип фреймворка Streamlit

Gradio [6] - на відміну попереднього аналога, Gradio (рис. 1.3) орієнтується на створення інтерфейсів для роботи з ML-моделями [7], функціонал візуалізації даних перестав бути для його розробників пріоритетною завданням, хоча можливості для цього представлені в API фреймворка. До мінусів Gradio можна віднести:

– досить скромне API для візуалізації даних;

– відсутність можливості генерації HTTP API та документації до нього у поширеному форматі OpenAPI [8].

Рисунок 1.3 – Логотип фреймворка Gradio

Panel [9] - даний фреймворк для мови програмування Python спрямований на створення веб-додатків для візуалізації даних і побудови аналітичних панелей (рис. 1.4). Можливості створення інтерфейсів для роботи зML-моделями в даному фреймворку представлені не так широко як у деяких з описаних вище аналогів. Мінуси Panel включають відносну складність налаштування та використання в порівнянні з більш простими фреймворками, що може створювати труднощі для

новачків. Крім того, через свою гнучкість і великі можливості, Panel може мати більш високі вимоги до ресурсів і продуктивності при створенні складних додатків.

Рисунок 1.4 – Логотип фреймворка Panel

1.2 Визначення критеріїв порівняння аналогів

Ґрунтуючись на розглянутих рішеннях, можна сформулювати перелік критеріїв, яким має відповідати розроблюване рішення, щоб перевершити альтернативні рішення в предметній галузі [1-3].

1. Підтримка авторизації клієнтів програми - це важливо для задоволення потреб безпеки та захисту персональної інформації.
2. Можливість створення веб-інтерфейсу для взаємодії з ML -моделлю – це дозволяє користувачам взаємодіяти з моделлю машинного навчання та отримувати результати в режимі реального часу. Це спрощує використання машинного навчання для користувачів без глибоких знань програмування та сприяє ширшому застосуванню ML -технологій у бізнесі та науці.
3. Підтримка роботи з різними фреймворками візуалізації даних – цей критерій забезпечує гнучкість у виборі інструментів для відображення даних. Це важливо, оскільки різні завдання можуть вимагати використання різних бібліотек візуалізації, таких як Matplotlib, Plotly та інші, для створення найбільш ефективних та зрозумілих графіків та діаграм.
4. Можливість створення HTTP API для взаємодії з ML- моделлю з автоматичною генерацією документації OpenAPI – наявність даного функціонала спрощує інтеграцію моделі в інші системи та програми. Це важливо для

забезпечення стандартизації, покращення сумісності та спрощення процесу розробки та підтримки API.

5. Оновлення змінених елементів без необхідності перезавантаження всієї сторінки - наявність даного функціоналу покращує користувальницький досвід за рахунок швидкої та плавної взаємодії із застосунком. Це знижує навантаження на сервер та заощаджує трафік, що особливо важливо для додатків з високою інтерактивністю та частими змінами даних.

У табл. 1.1 наведено результати порівняльного аналізу описаних раніше альтернативних рішень та виділених критеріїв.

Таблиця 1.1 – Порівняння альтернативних рішень за описаними критеріями

Номер критерію	Dash	Streamlit	Gradio	Panel
1	-	+	+	-
2	+	+	+	+
3	-	+	+	+
4	-	-	-	-
5	-	-	-	-

Таким чином, на момент проведення огляду предметної області жодне з рішень повністю не задовольняє зазначеним критеріям.

2 ПРОЕКТНА ЧАСТИНА

2.1 Архітектура програмного рішення

Задля реалізації поставленої мети, а також задоволення усім зазначеним вище критеріям була спроектована така архітектура (рис. 2.1).

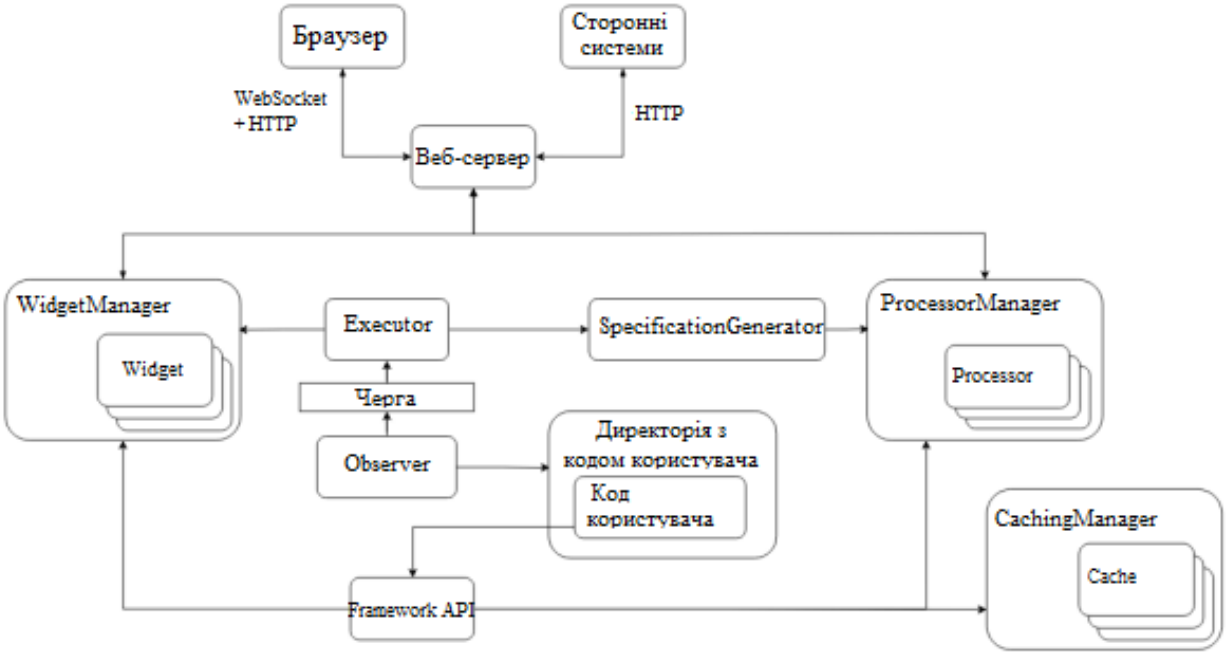


Рисунок 2.1 – Архітектура програмного рішення

Розглянемо докладніше кожен компонент наведеної архітектури.

Веб-сервер - служить для обробки запитів і запитів від сторонніх систем. Для вибору веб-сервера був проведений порівняльний аналіз чотирьох популярних фреймворків для розробки веб-застосунків із застосуванням мови Python, як то Django, Flask, Tornado та FastAPI. Зазначені фреймворки порівнювалися за такими критеріями:

– наявність підтримки WebSocket технології;
– продуктивність;
– можливість низькорівневого налаштування;
– рівень зрілості.

Під усі перелічені критерії найкраще підійшов фреймворк Tornado. Саме тому він був обраний для виконання завдань обслуговування клієнтських запитів, а також запитів від сторонніх систем.

Менеджер віджетів (WidgetManager) - відповідає за всі взаємодії з віджетами. Віджети - сутності фреймворку, що відповідають за те, що буде відображено на екрані користувача.

Менеджер процесорів (ProcessorManager) - відповідає за зберігання процесорів - сутностей, на основі яких генерується HTTP API, вони інкапсулюють в собі функцію користувача і логіку пов'язану з її викликом.

Менеджер кешів (CachingManager) – відповідає за кешування даних.

Спостерігач (Observer) - сутність, що відповідає за спостереження за змінами в коді користувача і модулях від якого він залежить.

Виконавець (Executor) - сутність, що відповідає за виконання коду користувача і подальшої обробки результатів. Він виконується в окремому потоці та взаємодія з ним відбувається через потокобезпечні черги.

Програмний інтерфейс фреймворку (Framework API) - набір функцій та декораторів [10], які відповідають за створення процесорів та віджетів.

Генератор специфікацій (SpecificationGenerator) - сутність, що відповідає за генерацію специфікації у форматі OpenAPI.

2.2 Проектування фреймворку

2.2.1 Огляд API фреймворку

API Фреймворка – це набір інтерфейсів, що надаються фреймворком, який дозволяє розробникам, у нашому випадку – ML-інженерам взаємодіяти з його функціоналом. Тобто, надаючи API фреймворк дозволяє інженерам використовувати готові компоненти та функції без необхідності писати їх самостійно.

Загальний зміст API відображено на рис. 2.2.

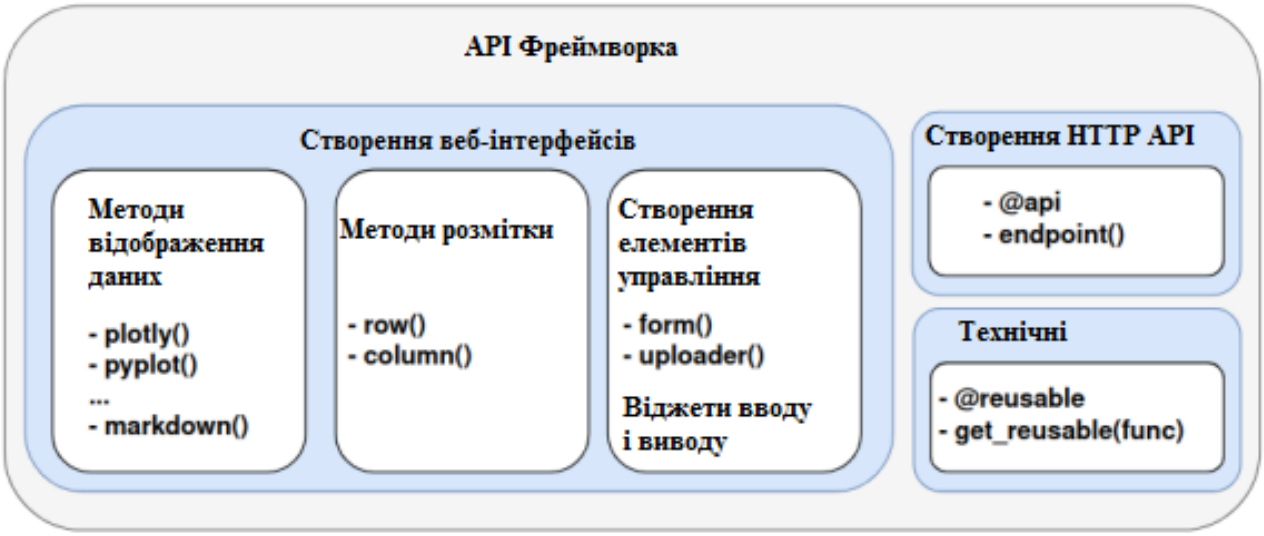


Рисунок 2.2 – Демонстрація розбиття API фреймворку на групи

Під час підготовки до реалізації API фреймворку було виділено 3 основні групи інтерфейсів, які будуть надані користувачеві:

– API для створення веб-інтерфейсів;

– API для створення кінцевих точок для взаємодії зі сторонніми системами;

– технічне API. Воно не впливає на відображення та взаємодію зі сторонніми системами безпосередньо, але використання інтерфейсів, що належать до цієї групи, може дозволити оптимізувати процес виконання коду користувача.

При цьому в процесі реалізації в рамках 1 групи було виділено 3 додаткові підгрупи:

– інтерфейси для відображення даних. Всі методи та декоратори, що належать до цієї підгрупи, дозволяють користувачеві відобразити на екрані будь-яку інформацію: текст, картинку, графік;

– інтерфейси, які дають змогу контролювати розмітку сторінки. Сюди відносяться методи для розташування елементів у рядок або стовпець. При цьому їх можна комбінувати, наприклад, в рамках одного рядка може бути кілька стовпців і так далі;

– інтерфейси для створення елементів керування, зокрема кнопки, поля вводу, різні форми.

2.2.2 Опис процесу взаємодії з API для створення веб-інтерфейсів

Як було зазначено раніше, у рамках цієї групи присутні 3 підгрупи інтерфейсів. Взаємодія з будь-яким інтерфейсом цієї групи призводить до реєстрації віджету в менеджері віджетів.

На рис. 2.3 представлений алгоритм створення віджету, який виконується для будь-якого методу, що відноситься до групи, що описується.

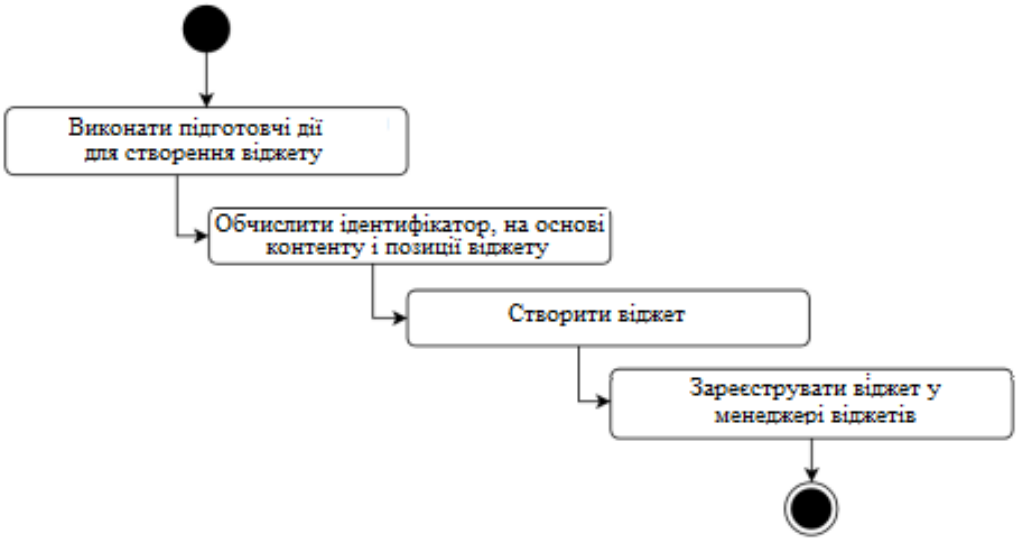


Рисунок 2.3 – Загальний алгоритм створення віджету під час взаємодії з API фреймворку

Спершу виконуються підготовчі дії. Вони варіюються в залежності від того, який віджет ми створюємо. Наприклад, при виклику методу plotly (fig) буде виконано збереження графіка як картинки з розширенням PNG. А при виклику методу form(inp, out , func) буде здійснено валідацію віджетів, переданих у вигляді аргументу inp і створено об'єкт віджет форми.

Потім буде виконано обчислення ідентифікатора для віджету, що створюється. Воно виконується на основі його контенту та позиції на екрані та складається з 2 етапів:

– обчислення хеш значення контенту віджету. Залежно від типу віджету, хеш обчислюється по-різному. Наприклад, для віджету, що є текстом у форматі Markdown, буде обчислено MD5 хеш [11] від його контенту;

– обчислення позиції віджету. У випадку, якщо віджет унікальний, його ідентифікатор збігатиметься з обчисленим першому етапі хешем. Якщо ж у додатку вже існують дублікати даного віджету, то до хеша конкатенується порядковий номер цього віджету щодо його дублікатів.

Третій крок - створення об'єкта, який буде збережено у менеджері віджетів. У рамках реалізації фреймворку було створено ієрархію, що складається з класів віджетів. Це дозволило перевикористовувати спільні поля, без необхідності їх повторного оголошення в кожному віджеті, а також визначити загальну поведінку.

Частина ієрархії класів представлена на рис. 2.4. На ньому відбито основні віджети і підгрупи, такі як ресурси, контейнери тощо.

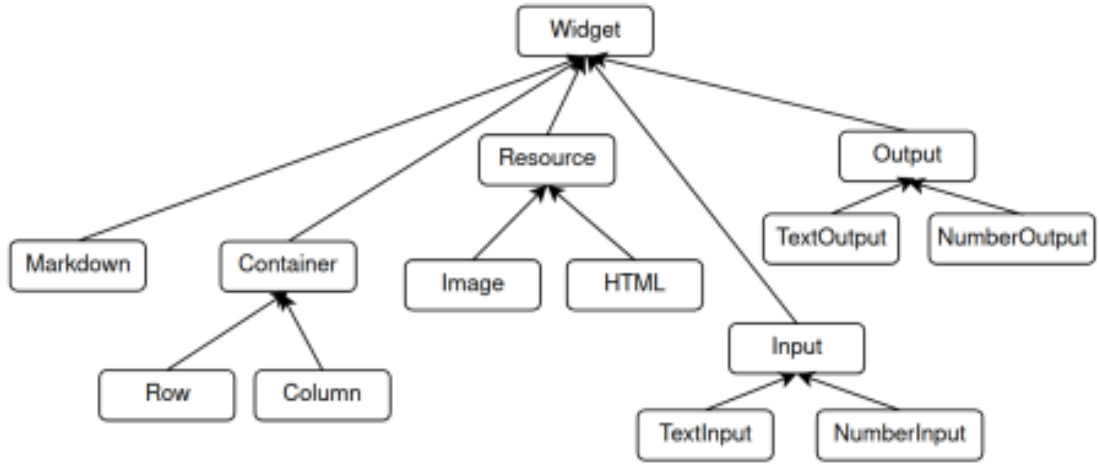


Рисунок 2.4 – Основні елементи ієрархії класів віджетів

Після того, як було створено об'єкт віджету, відбувається його реєстрація в менеджері віджетів. Як було сказано раніше, менеджер віджетів відповідає за всі взаємодії з віджетами. Це включає реєстрацію та видалення віджетів, надання

способу ітерації по віджетах і отримання віджету по ідентифікатору. При реєстрації ми зберігаємо ідентифікатор віджету в структуру даних множин, представлену в мові програмування Python як клас set [12], а сам віджет в структуру даних список, представлену в мові програмування Python як клас list [12], оскільки вона зберігає порядок, в якому елементи були внесені.

Після того як віджет був зареєстрований, метод, що надається користувачем, що надається API фреймворку, закінчує своє виконання.

Лістинг коду методу API для створення віджету Markdown наведено на рис. 2.5. Так як він може бути створений без виконання підготовчих дій – цей крок пропускається. Як було сказано раніше, ідентифікатор елемента обчислюється у 2 етапи:

– отримуємо MD5 хеш від тексту у форматі Markdown;

– звертаємося в менеджер віджетів для отримання зміненого ідентифікатора, якщо елемент з ідентичним текстом вже зареєстрований.

Після цього створюємо сам віджет та реєструємо його в менеджері віджетів.

```
def markdown(mdText):
      elementHash =
widgetManager.get_id(hashlib.md5(mdText.encode()).hexdigest())
      widget = Markdown(elementHash, mdText)
      widgetManager.register(widget.hash, widget)
```

Рисунок 2.5 – Метод API для створення віджету Markdown

2.2.3 Опис процесу взаємодії з API для створення HTTP API

Розроблений у рамках кваліфікаційної роботи фреймворк надає інженеру 2 інтерфейси для створення HTTP API на базі описаного ним коду:

– декоратор API;

– метод endpoint.

Вказані вище інтерфейси еквівалентні один одному, тому що при взаємодії з ними відбувається звернення до менеджера процесорів. Єдина для обох інтерфейсів послідовність дій зображена на рис. 2.6.

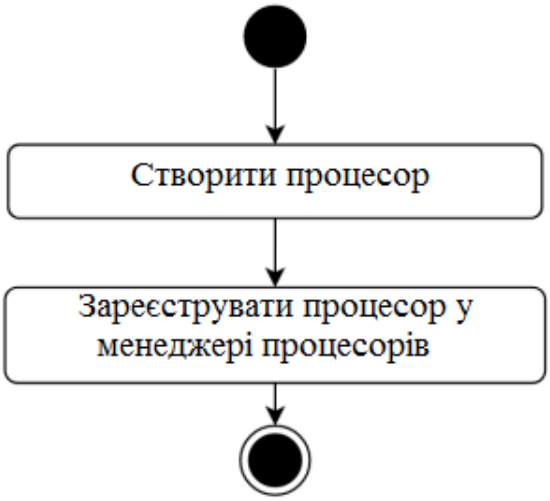


Рисунок 2.6 – Послідовність дій, що виконується під час реєстрації процесора

Процесор - сутність, яка зберігає в собі інформацію про адресу кінцевої точки, за яким повинна викликатися користувальницька функція, посилання на функцію користувача, а також тип значення, що повертається. Адреса і тип значення, що повертається, необхідні для генерації специфікації у форматі OpenAPI. Також під час створення процесора, з урахуванням параметрів функції, генерується модель об'єкта з бібліотеки Pydantic [13]. Вона буде використана як для створення специфікації, так і для валідації та приведення типів параметрів, переданих у HTTP запиті.

Реєстрація процесора в менеджері процесорів виконується за допомогою виклику методу registerEndpoint об'єкта менеджера процесорів, представленого в застосунку в єдиному екземплярі. У вигляді вхідних параметрів метод приймає адресу кінцевої точки, за якою повинна викликатися функція користувача, і, безпосередньо, об'єкт процесора. Ці параметри зберігаються в структурі даних словника, представленого в мові програмування Python як клас dict [12], ключем є адреса, а значення - об'єкт процесора.

Лістинг коду прикладу взаємодії з інтерфейсом api представлений на рис. 2.7. В результаті для користувача буде згенеровано кінцеву точку, запити на яку будуть перенаправлятися в даний метод.

```
@drw.api(endpoint="/greet", outputType=str)
def greet_user(username: str):
    return f"Hello {username}!"
```

Рисунок 2.7 – Приклад взаємодії з інтерфейсом API

Приклад взаємодії зі створеною кінцевою точкою продемонстровано на рис. 2.8. За допомогою командної утиліти curl [14] відправляється запит до сервера, у тілі запиту передається ім'я користувача. У відповіді сервера міститься результат виконання функції.

```
(.venv) emdavl t480s:~/dev/diploma/demo [1211]% curl \
-X POST \
-sd "{\"username\": \"Emil\"}" \
localhost:8888/api/greet | jq
{
  "result": "Hello Vlad!"
}
```

Запит

Відповідь

Рисунок 2.8 – Взаємодія з API користувача

2.2.4 Опис процесу взаємодії з технічним API

Фреймворк надає 2 інтерфейси, що дозволяють інженеру вказати, що результат виконання функції повинен бути використаний повторно між запусками коду користувача додатком:

– декоратор reusable;
– метод get reusable.

Перший інтерфейс - декоратор reusable - декорує певну користувачем

функцію, отже кожного наступного звернення до функції буде виконано перевірка: чи існує цієї функції значення у кеші, відповідне переданим у функцію параметрам. Якщо в результаті перевірки значення знайдено, то виклик функції пропускається, відразу повертається знайдене значення. Якщо значення знайдено не було, відбувається виконання функції, збереження результату в кеші та повернення значення. Всі наступні дзвінки функції, що збігаються з набором параметрів, повертатимуть закешоване значення.

Другий інтерфейс - метод get_reusabie - має ідентичну поведінку. У вигляді вхідних параметрів він приймає функцію і набір аргументів так, як вони мають бути передані в неї. Потім відбувається перевірка наявності значення кеші для цієї функції. Якщо значення знайдено - воно буде повернено. Інакше програма виконає функцію з переданими параметрами та збереже результат виконання.

Загальний механізм роботи вищеописаних інтерфейсів представлений на рис. 2.9.

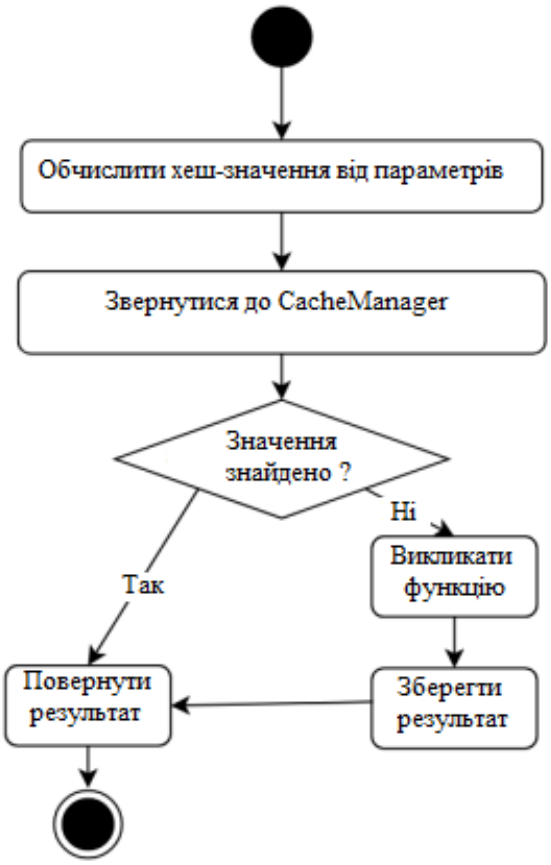


Рисунок 2.9 – Механізм роботи декоратора reusable та методу get_reusable

На рис. 2.10 наведено лістинг прикладу взаємодії з даним інтерфейсом в коді користувача. Функція “print_and_sum” друкує текст у стандартний висновок та повертає суму переданих значень. Для кожного набору аргументів, виконана вона буде лише один раз, тому що на кожен повторний виклик відповідь буде повернена з кеша.

```
@drw.reusable
def print_and_sum(first: int, second: int):
    print('We are in print_and_sum now!')
    return first + second
```

Рисунок 2.10 – Прикладу взаємодії з інтерфейсом в коді користувача

2.3 Розробка модуля, що відповідає за виконання коду користувача

Запуск програми виконується за допомогою запуску через модуль фреймворку. При старті відбувається ініціалізація всіх загальних змінних, різних параметрів конфігурації та об'єктів, які повинні бути представлені в застосунку в єдиному екземплярі. Перед тим, як запустити веб-сервер, необхідно заповнити менеджери віджетів, процесорів та кешів. Для цього відбувається виконання коду користувача. Воно ж відбувається при виявленні змін у коді або модулях, від яких він залежить. За цей процес відповідає описуваний у цьому підрозділ модуль (далі Виконавець). Також Виконавець відповідає за виконання дій з обробки результатів виконання коду користувача.

Оскільки основний потік програми зайнятий веб-сервером, виконання в ньому блокуючих операцій, що займають процесорний час, призведе до зменшення чутливості програми. У зв'язку з цим було ухвалено рішення винести Виконавця в окремий потік. Взаємодія з Виконавцем з інших потоків виконується через 2 потокобезпечні черги. Одна на надсилання повідомлень з Виконавця, інша на надсилання повідомлень до Виконавця. Це, до того ж, дозволяє уникнути такої

поширеної помилки багатопотокового середовища, як взаємне блокування.

Виконання коду відбувається при отриманні Виконавцем будь-якої з двох подій:

– START - означає, що програма запускається і необхідно виконати перший запуск коду. Після першого виконання Виконавець відправляє у вихідну чергу повідомлення з результатом виконання - успішно воно чи ні. Ця подія надсилається при запуску всієї програми;

– RERUN - означає, що було виявлено зміну коду користувача і його потрібно виконати повторно. Ця подія відправляється спостерігачем.

Схема отримання Виконавцем повідомлень наведена на рис. 2.11.

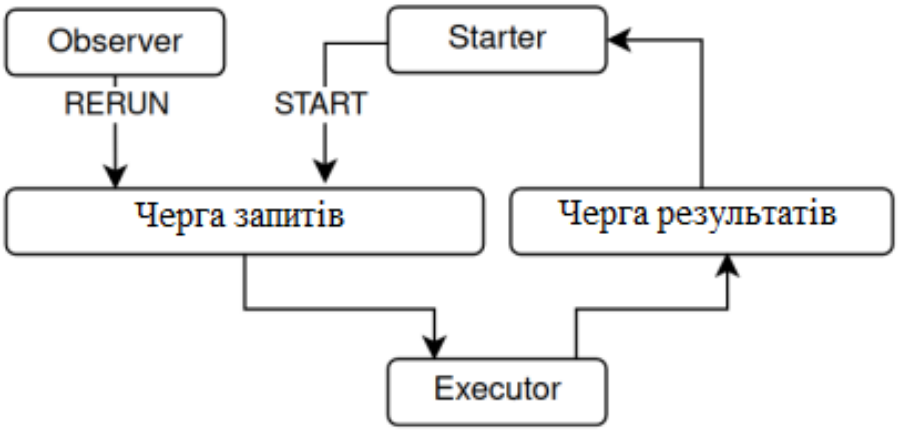


Рисунок 2.11 – Схема отримання Виконавцем повідомлення

2.3.1 Дії, які виконуються при отриманні події START

При отриманні події START Виконавець виконує код користувача, використовуючи вбудовану функцію exec [15]. Логіка виконання коду винесена в окремий метод - це дозволяє використовувати її повторно, уникаючи дублювання коду.

Потім Виконавець ініціює оновлення спостерігача. Останнім кроком є ініціювання складання сміття за допомогою виклику функції collect модуля gc [16].

Якщо всі кроки виконані успішно, Виконавець пише у вихідну чергу подію STARTED, інакше FAILED.

Описана вище послідовність кроків відображена на рис. 2.12.

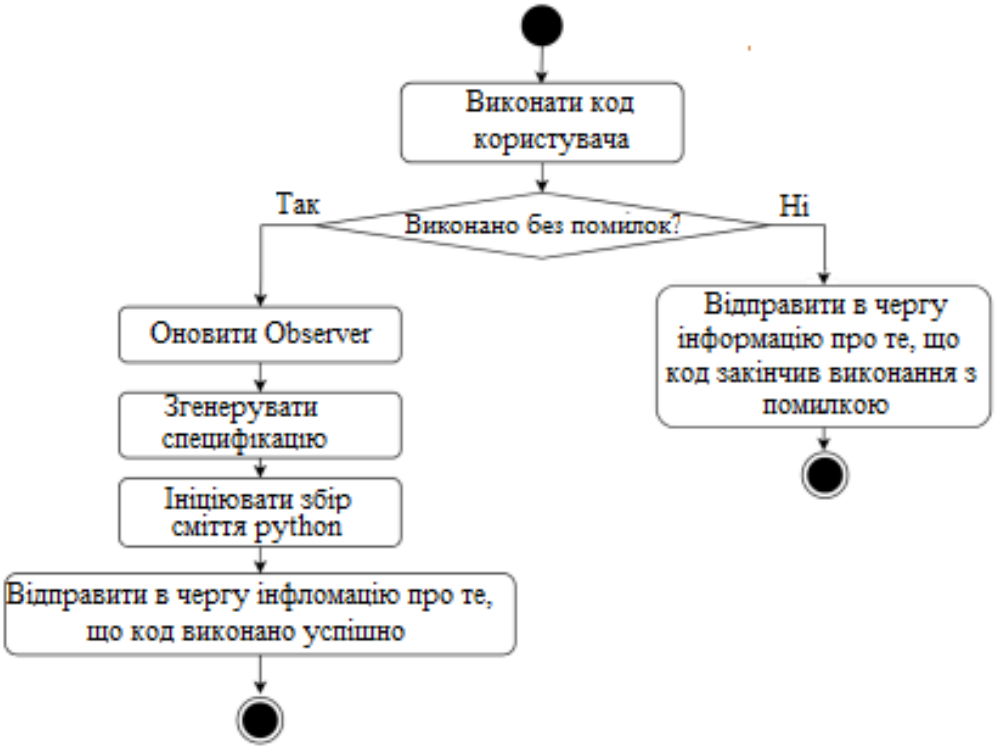


Рисунок 2.12 – Послідовність кроків, що виконується при отриманні події START

2.3.2 Дії, які виконуються при отриманні події RERUN

Кількість кроків, які виконуються при отриманні події RERUN більше, ніж при отриманні події START. Так як крім виконання коду користувача, нам необхідно видалити неактуальні віджети, процесори і кеші. А також надіслати оновлення користувачеві та оновити специфікацію. Послідовність кроків, які виконуються при отриманні події RERUN, наевдена на рис. 2.13.

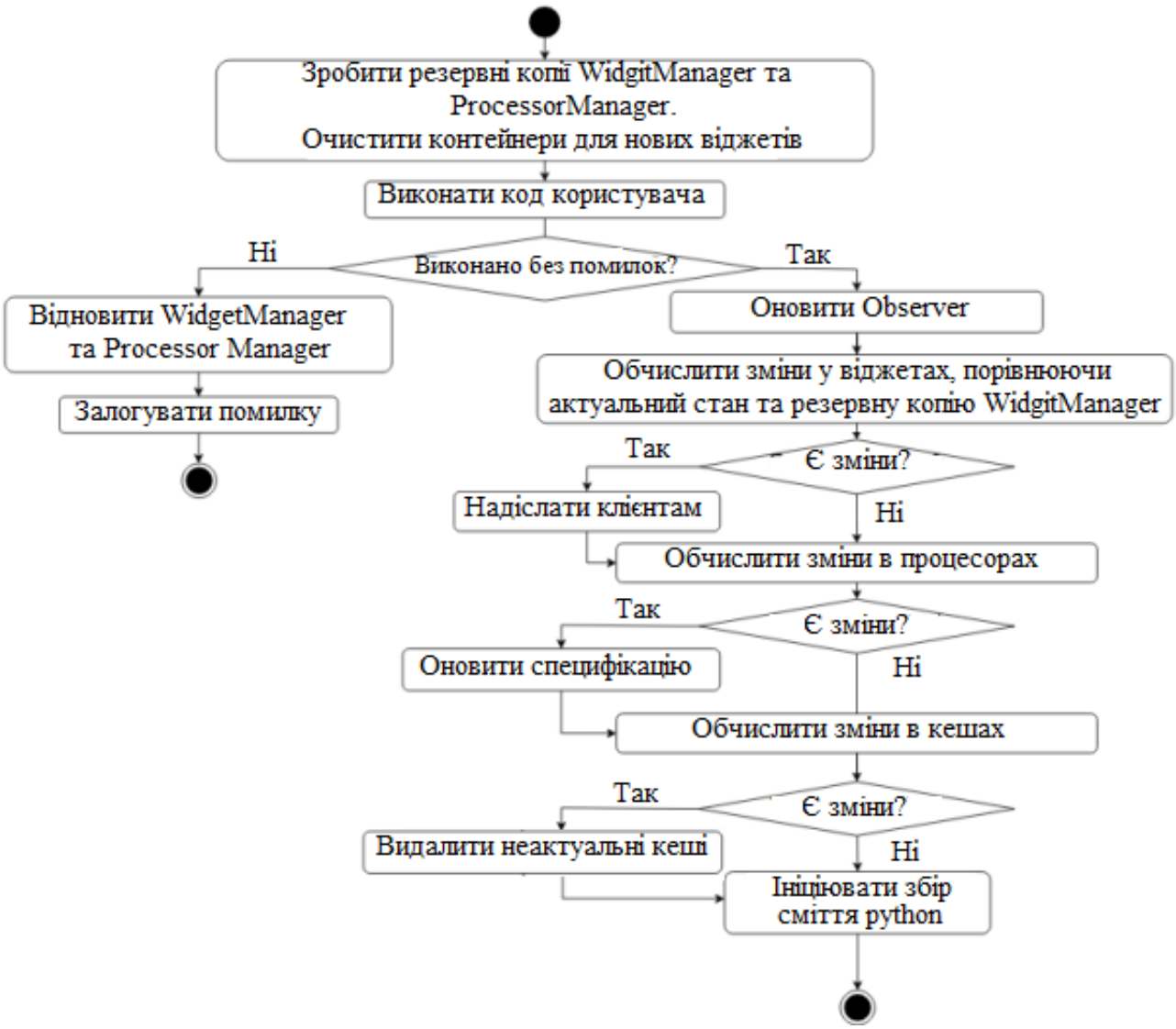


Рисунок 2.13 – Послідовність кроків, що виконуються при отриманні події RERUN

Насамперед необхідно створити резервні копії менеджера віджетів та процесорів. Для цього до даних менеджерів був доданий метод snapshot.

При виклику вказаного методу у менеджера віджетів виконуються дії, наведені нижче:

– створюються копії множин ідентифікаторів та списку віджетів;

– встановлюється прапор, що вказує на те, що програма ініціювала процес повторного виконання коду. Доки цей прапор не буде знятий, нові користувачі будуть отримувати дані зі зроблених на 1 кроці копій множини ідентифікаторів і списку віджетів;

– очищаються множина ідентифікаторів та список віджетів. Вони будуть заповнені при повторному виконанні коду користувача.

При виклику вказаного методу у менеджера процесорів виконуються такі действия:

– створюється копія поточного словника зіставлення адрес кінцевих точок і процесорів, відповідальних за обробку запитів, отриманих за відповідною адресою;

– встановлюється прапор, що вказує на те, що застосунок ініціював процес повторного виконання коду. Доки цей прапор не буде знято, нові запити будуть перенаправлятися в процесори, що зберігаються в копії словник, створеної на 1 кроці;

– словник очищується. Він буде заповнений при повторному виконанні коду користувача.

Наступний крок - виконання коду користувача. Він ідентичний кроку, який виконується при отриманні події START.

Якщо код користувача закінчив своє виконання з помилкою, то відбувається відновлення менеджера віджетів і процесорів. Прапори, що вказують на те, що програма ініціювала процес повторного виконання коду користувача, знімаються, помилка логується.

У разі успішного виконання коду користувача здійснюються дії з обробки результатів взаємодії з сутностями програми.

Здійснюється оновлення спостерігача. Цей процес не відрізняється від того, що виконується при обробці події START.

Відбувається обчислення змін у віджетах, що зберігаються у менеджері віджетів. Цей крок потрібний для успішного оновлення екрана користувача. За наявності будь-яких змін усі користувачі отримують інформацію про те, які віджети їм потрібно видалити, які пересунути, а які створити. Збережені копії множини ідентифікаторів та списку віджетів видаляються.

Потім виконується обчислення змін у процесорах. За наявності таких відбувається оновлення специфікації. Збережена копія словника зіставлення адрес

кінцевих точок і процесорів, відповідальних за обробку запитів, отриманих за відповідною адресою, видаляється.

Далі виконується обчислення змін у кешах. При необхідності неактуальні кеші видаляються.

Останнім кроком є ініціалізація збирання сміття. Цей процес не відрізняється від того, що виконується при обробці події START.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Розробка модуля, що реагує на зміни в користувальницькому коді

Як вже було сказано в п. 2.3, подія RERUN генерується Спостерігачем, при виявленні модифікації файлу з кодом користувача або файлів модулів, від яких код користувача залежить. У цьому підрозділі йтиметься про те, як відбувається вибірка файлів для спостереження, також будуть описані подробиці реалізації Спостерігача.

На рис. 3.1 наведено алгоритм пошуку файлів для спостереження.

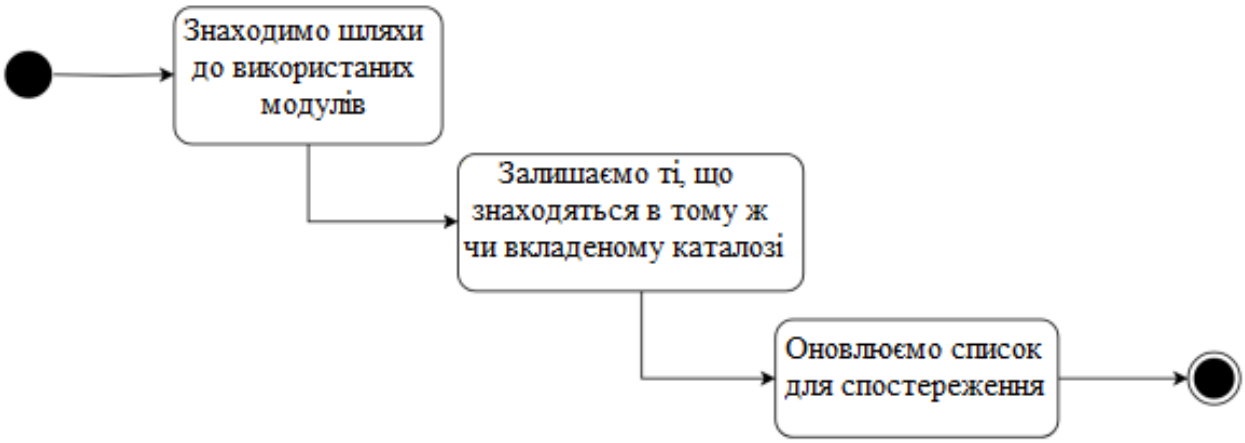


Рисунок 3.1 – Алгоритм пошуку файлів спостереження

Розглянемо його докладніше. Оскільки код користувача виконується в одному контексті з додатком, програма має доступ до всіх імпортованих модулів. Було визначено 3 способи вилучення шляху з модуля.

1. Отримання значення параметра _file_ [17] - це дозволяє отримувати шляхи до файлу з якого було завантажено модуль, за умови, що модуль завантажено з файлу.

2. Отримання значення параметра _spec_ [18] - у даного параметра ми отримуємо значення атрибуту origin. Атрибут _spec_ встановлюється відповідно до специфікації модуля, використаної при його імпорті. Винятком є модуль _main_, де в деяких випадках _spec_ може бути встановлений у значення None. Під

"походженням" у контексті імпорту розуміється система (або ресурс усередині системи), з якої походить модуль. Інтерпретація та використання походження модуля залишаються на розсуд завантажувача.

3. Отримання значення параметра _path_ [19] - цей спосіб актуальний для модулів, імпортованих із пакетів-просторів імен. У таких модулів атрибут _path_ належить до типу _NamespacePath_, в якому є поле _path_, що містить різні шляхи пакету.

Зазначені кроки застосовуються до кожного відомого модуля. В результаті виходить набір шляхів до кожного використаного файлу скриптом. Дані шляхи відфільтровуються - залишаються тільки ті, що знаходяться в тій же або у вкладених директоріях із зазначеним при старті програми файлом, що містить код користувача.

Потім виконується створення об'єкта оброблювача - ScriptModificationHandler. Йому буде делеговано повідомлення про зміну будь-якого зі спостережуваних файлів. Єдине його завдання - при отриманні події зміни відправити повідомлення типу RERUN в чергу взаємодії з Виконавцем.

Після того, як об'єкт обробник був створений, відбувається запуск механізму спостереження за користувальницькою директорією.

При реалізації Спостерігача використовувалася бібліотека watchdog [20]. При запуску програми ми створюємо об'єкт Спостерігача, який представлений у застосунку в єдиному екземплярі. Він запускає в окремому потоці observer, що надається бібліотекою watchdog. Подальша реєстрація нових обробників відбуватиметься з використанням цього об'єкта observer.

3.2 Розробка JavaScript модуля

3.2.1 Маршрутизація запитів до обробників

В рамках цього завдання було необхідно реалізувати механізм маршрутизації

повідомлень, які приходять з сервера по WebSocket [21] з'єднанню, у відповідний тип повідомлення обробник.

На відміну від традиційного HTTP-протоколу, який є одностороннім (клієнт відправляє запит, сервер відповідає), веб-сокет дає змогу встановити безперебійне з'єднання між клієнтом та сервером, що властиво дає змогу обидвом сторонам відсилати дані один одному в реалі.

Під обробниками тут і далі маються на увазі класи або функції, реалізовані з використанням мови JavaScript [22], які при отриманні повідомлення виконують певну послідовність дій націлених на обробку цього повідомлення та відображення будь-якого результату користувачеві.

Під типом повідомлення в контракті взаємодії між сервером і браузером визначено окреме поле.

Контракт запиту визначає правила та формат повідомлень, котрі можуть бути відправлені та отримані між клієнтом та сервером завдяки з'єднанню WebSocket. Контракт запиту включає інформацію про:

– типи повідомлень;
– структуру даних;
– протокол обміну повідомленнями;
– очікувані відповіді.

Цей контракт запиту важливий для обох сторін - клієнта і сервера - щоб забезпечити правильну взаємодію та розуміння даних, що передаються. Він допомагає стандартизувати обмін повідомленнями, покращити надійність та ефективність з'єднання WebSocket, а також забезпечити узгодженість взаємодії між клієнтом та сервером.

Лістинг коду контракту запиту між браузером і сервером, реалізований у застосунку, який використовує фреймворк, наведено на рис. 3.2.

```
{
  "type": "TYPE IN UPPER CASE",
  "data": {
    // Корисний зміст повідомлення
  }

}
```

Рисунок 3.2 – Контракт запиту між браузером і сервером

Якщо з сервера надійде тип, який невідомий браузеру - буде відображено спливаюче вікно, що сповіщає користувача про помилку. Таке може статися тільки при помилці розробника фреймворку - на сервер тип події додано, а обробник на клієнтській частині з якоїсь причини ще немає.

Механізм визначення необхідного обробника зводиться до отримання посилання на обробник із структури даних Map [22]. Ця структура даних дозволяє зберігати пари ключ-значення. При цьому в середньому отримати значення ключа можна з тимчасовою складністю рівною O(1). Заповнення зазначеної структури даних виконується під час першого завантаження сторінки.

3.2.2 Розробка обробників віджетів

Після того як був реалізований основний маршрутизатор запитів, що передає запит на створення віджету потрібному обробнику, виникла потреба у тому, щоб реалізувати логіку маршрутизації запитів до методів, які створюють конкретні віджети. Тут був застосований аналогічний механізм - створюється структура даних Map, яка заповнюється за таким правилом: ключ - тип віджету, значення - функція, яка вміє перетворювати віджет в HTML елемент.

Тип віджету передається разом з іншою інформацією поле data, описаного вище контракту взаємодії. На рис. 3.3 наведено лістинг коду прикладу повідомлення, яке передається при створенні віджету с Markdown [23] текстом, а

на рис.3.4 представлено те, на що даний віджет перетворюється JavaScript модулем. Для прикладу, на тому ж рисунку відображені заголовки 1 і 3 рівнів.

```
{
        "type": "WIDGET",
        "data": {
            "id": "5ff549b8-dd66-4ff9-99d4-3ecefe4fd26c",
            "hash": "7f8fbfaf56d2c8a098a6cac2982103c2",
            "type": "markdown",
            "markdown": "## Це заголовок 2 рівня"
        }
}
```

Рисунок 3.3 – Повідомлення, яке передається при створенні віджету с Markdown

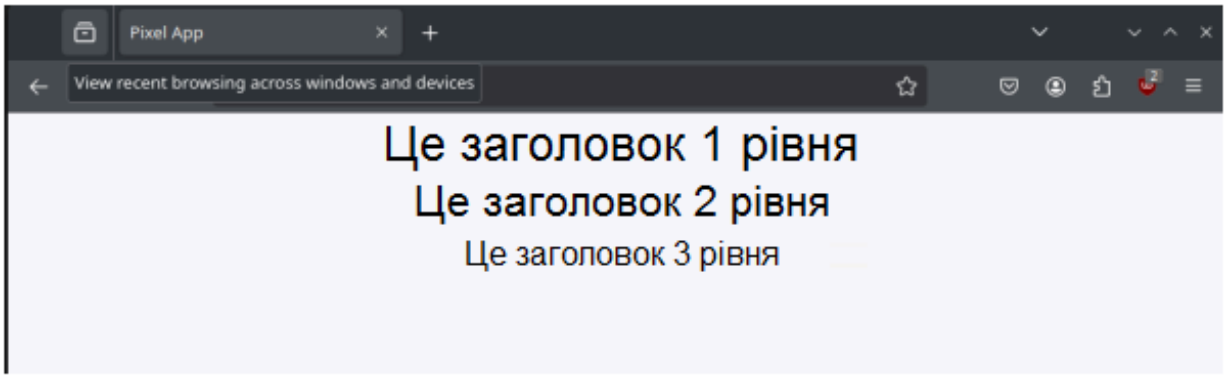


Рисунок 3.4 – Відображення заголовків 1, 2 і 3 рівнів, створених при допомоги віджету Markdown

Отримавши повідомлення, обробник приступає до його перетворення. Незалежно від типу створюваного віджету, спочатку створюється контейнер типу div [24]. Це необхідно для більш гнучкого налаштування розміру елемента, що відображається, а також для того, щоб ми могли позиціонувати весь віджет, незалежно від кількості тегів, використаних для його відображення. Потім виконуються дії, необхідні для перетворення віджету на відповідний тег або набір тегів. Наприклад, для багаторядкового markdown тексту з заголовком віджет перетворюється на кілька тегів - h1 [24] для заголовка першого рівня та p [24] для

тексту. Для графіка pyplot використовується тег img [24]. Приклад відображення візуалізації заголовка та графіка pyplot наведено на рис. 3.5.

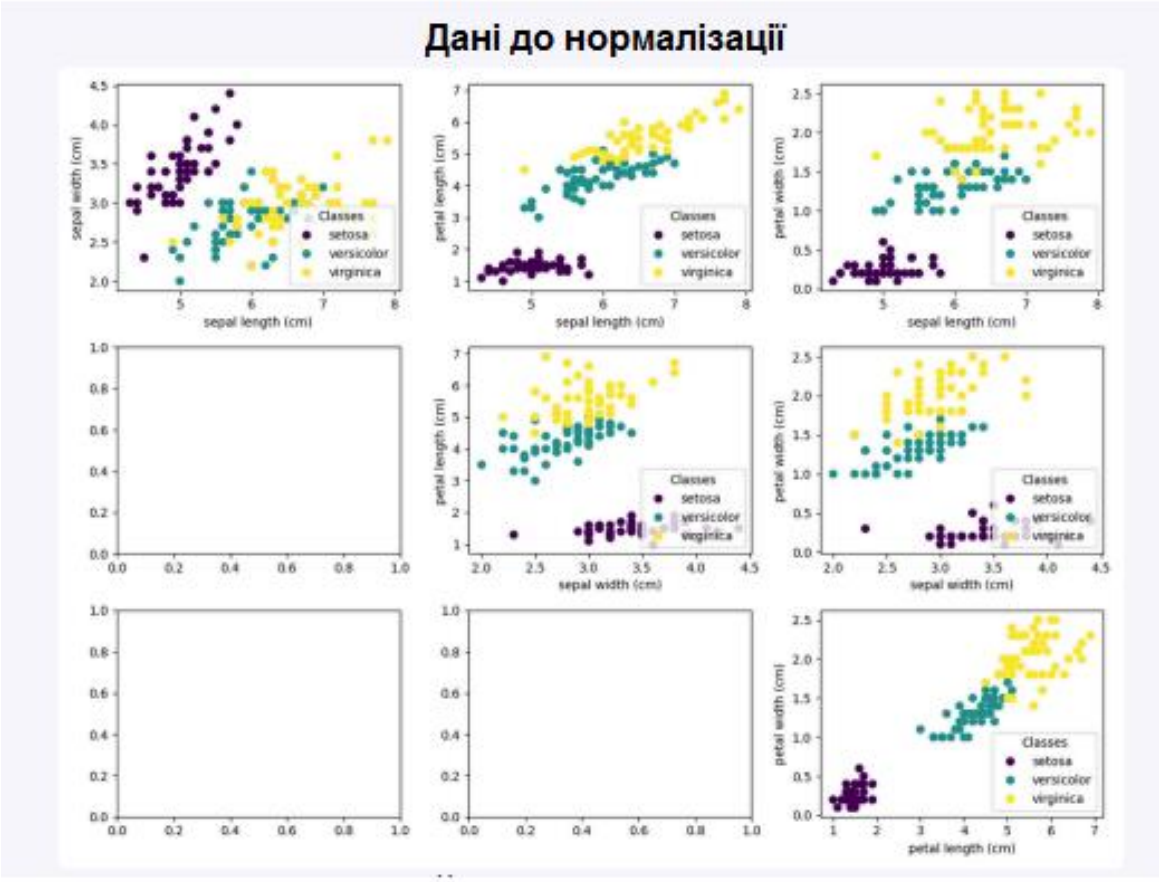


Рисунок 3.5 – Приклад відображення тексту і pyplot графіка

Після того як перетворення завершено, елемент додається в кінець сторінки, його хеш додається в кінець списку елементів, що зберігає послідовність елементів, хеш і посилання на div зберігаються в контейнері widgets типу Map як ключ і значення відповідно.

Алгоритм перетворення повідомлення отриманого від сервера елемент на екрані користувача представлений на рис. 3.6.

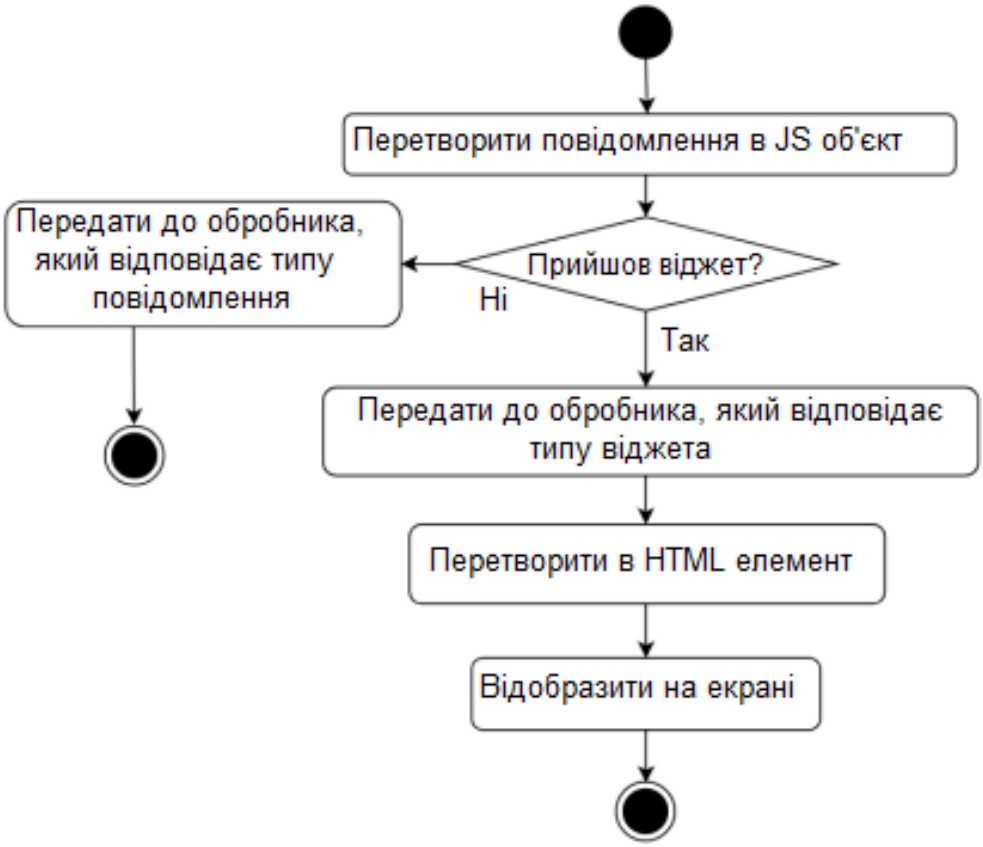


Рисунок 3.6 – Алгоритм перетворення повідомлення на HTML елемент тексту

3.2.3 Розробка обробника події оновлення екрану

При зміні коду користувача сервер обчислює різницю з тим, що відображено на екрані в даний момент, і відправляє її всім клієнтам. Лістинг коду прикладу такого повідомлення наведено на рис. 3.7.

При отриманні запиту на оновлення екрана маршрутизатор перенаправляє його відповідному обробнику. Його завдання застосувати ці зміни так, щоб користувачеві відображався актуальний стан застосунку.

```
    {
      "type": "UPDATE",
      "data": {
          "toDelete": [
                "dea2e59422463e206a3a91d5c9569ea6"
          ],
          "toMove": [{
                "elementHash": "06e338bfac94eb5dc52ac0957dfd593c",
                "previousElementHash": "",
                "nextElementHash": ""
          }],
          "toCreate": [{
                "widget": {
                "id": "c71200b7-b3b1-4ab5-bc88-b8996b69c3f6",
                "hash": "7a5d69dd5db12ff9a308c36f42cd6d81",
                "type": "markdown",
                "markdown": "### Third level header"
          },
          "previousElementHash": "06e338bfac94eb5dc52ac0957dfd593c",
          "nextElementHash": ""
    }]
  }
}
```


Рисунок 3.7 – Код, який обчислює різницю з тим, що відображено на екрані в даний момент, і відправляє її всім клієнтам

Алгоритм застосування змін зображено на рис. 3.8. Для видалення віджетів ми перебираємо рядкові значення, надіслані сервером у полі toDelete. Для кожного значення отримуємо відповідний div з контейнера widgets і виконуємо видалення використовуючи вбудовані функції мови JavaScript.

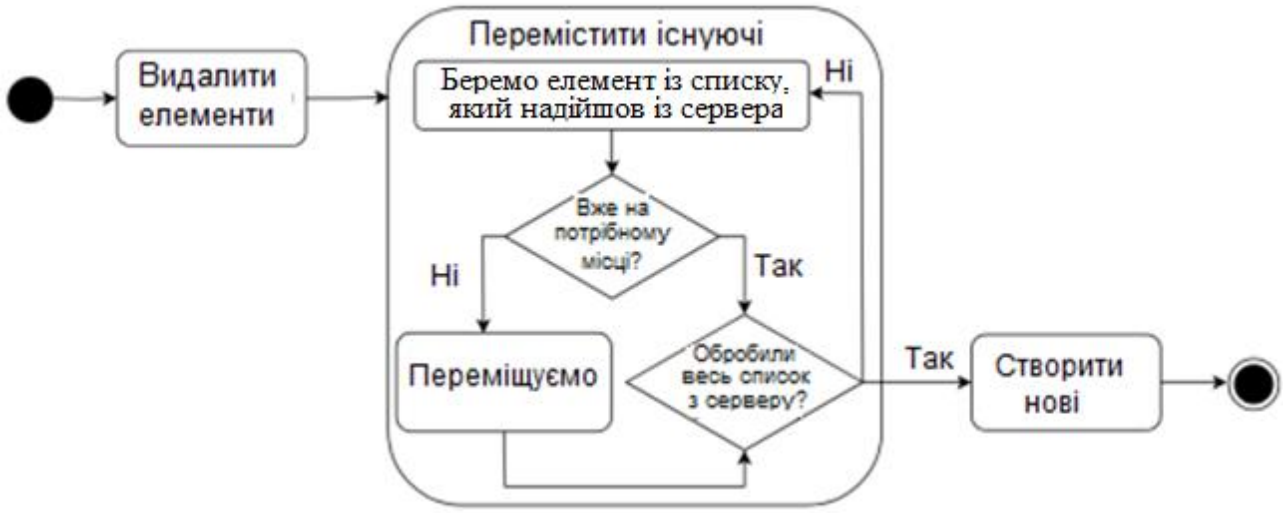
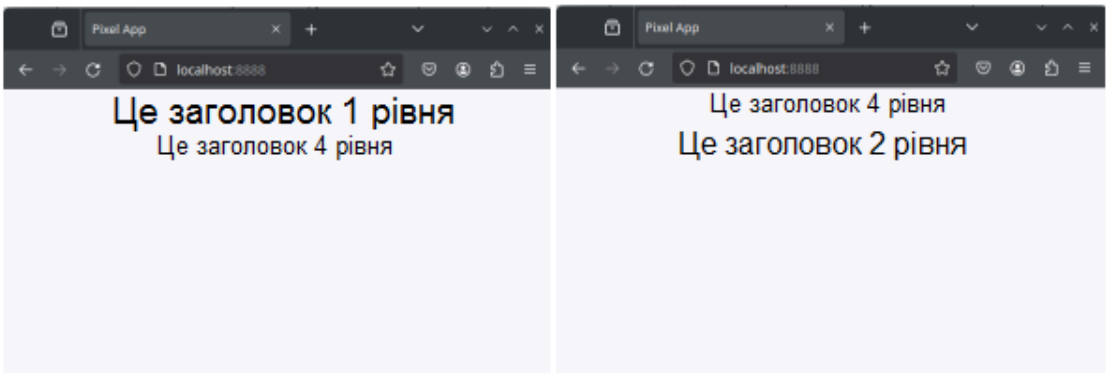


Рисунок 3.8 – Алгоритм оновлення екрана

Для переміщення існуючих віджетів ми використовуємо список toMove . У ньому відображаються всі віджети, представлені на сторінці, в тому порядку, в якому вони повинні бути відповідно до змін, виконаних у коді користувача. Для того щоб привести їх до такого порядку потрібно поелементно порівнювати списки, який відображає поточне відображення, і отриманий з сервера, за наявності відмінностей переміщаємо елемент на визначене користувачем в коді місце.

Фінальним кроком є створення нових елементів. Список toCreate із повідомлення містить інформацію про віджет, який нам потрібно створити, а також ідентифікатори його сусідів. При цьому елементи в цьому списку розташовані в такому порядку, що для кожного з них на момент обробки на екрані буде як мінімум 1 із сусідів, виключенням може бути елемент під індексом 0, так як до моменту його створення екран може бути порожній.

На рис. 3.9 представлено стан екрану до зміни в коді користувача (а) та після зміни (б). Зміна була виконана відповідно до кроків, описаних вище.



а) б)

Рисунок 3.9 – Стан екрану

3.2.4 Розробка обробників інших типів повідомлень

Крім подій створення віджету та оновлення екрана сервер може відправити подію відповіді на взаємодію з елементами керування та подію помилки, пов'язаної з обробкою запиту користувача.

При взаємодії з елементами керування на екрані, на сервер надсилається запит, що містить ідентифікатор елемента керування. Сервер виконує визначені користувачем дії та повертає результат. Лістинг коду прикладу результату наведено на рис. 3.10. Те, як це повідомлення буде відображено, представлено на рис. 3.11 - у текстовому полі з синім тлом відображено назву типу ірису, який найбільше підходить під введені параметри.

```
{
  "type": "FORM_RESPONSE",
  "data": {
      "outputType": "TEXT_OUTPUT",
      "formId": "50c800a5-ea60-4efb-9973-611dfec30f55",
      "text": "setosa"
  }
}
```

Рисунок 3.10 – Код функції результату

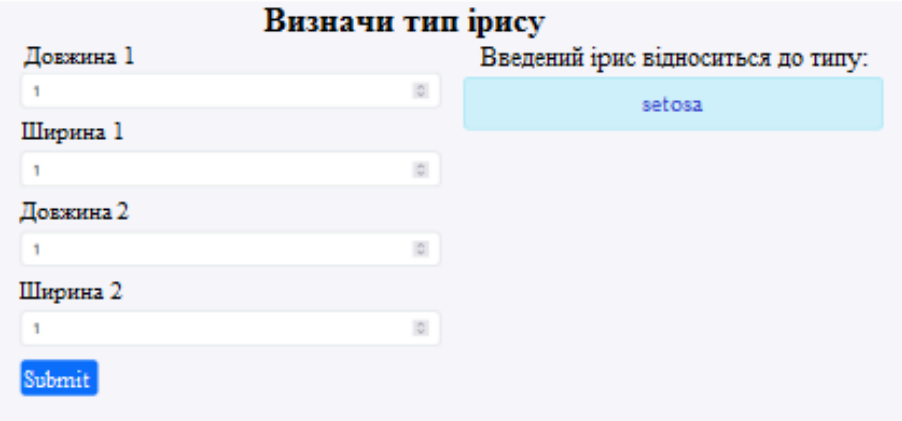


Рисунок 3.11 – Відображення результату взаємодії з формою

Помилки, пов'язані з обробкою запиту користувача, відображаються з використанням виклику функції alert [25]. У тілі alert вказано причину помилки. Приклад зображено на рис. 3.12.

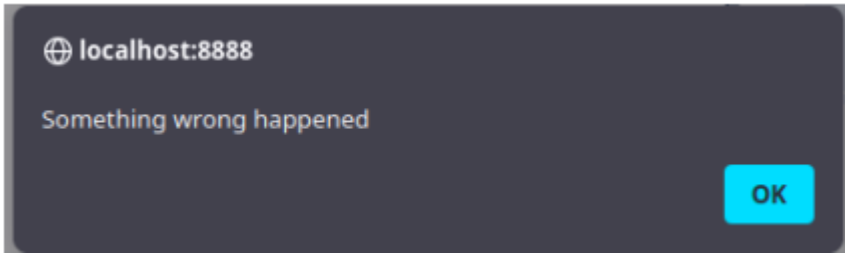


Рисунок 3.13 – Приклад відображення повідомлення про помилку

3.3 Розробка модуля, що відповідає за генерацію HTTP API і OpenAPI специфікації на основі коду користувача

Як було сказано в п. 2.3, для генерації HTTP API і OpenAPI специфікації користувач повинен використовувати декоратор api або метод endpoint. При цьому буде створено відповідний запис у менеджері процесорів, що зіставляє адресу та, відповідальний за обробку запиту, процесор.

3.3.1 Генерація HTTP API та обробка запитів користувача

На рис. 3.14 зображена схема обробки запиту користувача. Для того, щоб обслуговувати запити користувача в веб-сервер був доданий обробник всіх запитів, адреса запиту яких має префікс "/api" а використаний HTTP метод - POST [26] З кожного отриманого запиту він витягує тіло запиту, також він обрізає шлях запиту, виключаючи з нього префікс "/api". Потім обробник делегує запит менеджеру процесорів, викликаючи у того метод process.

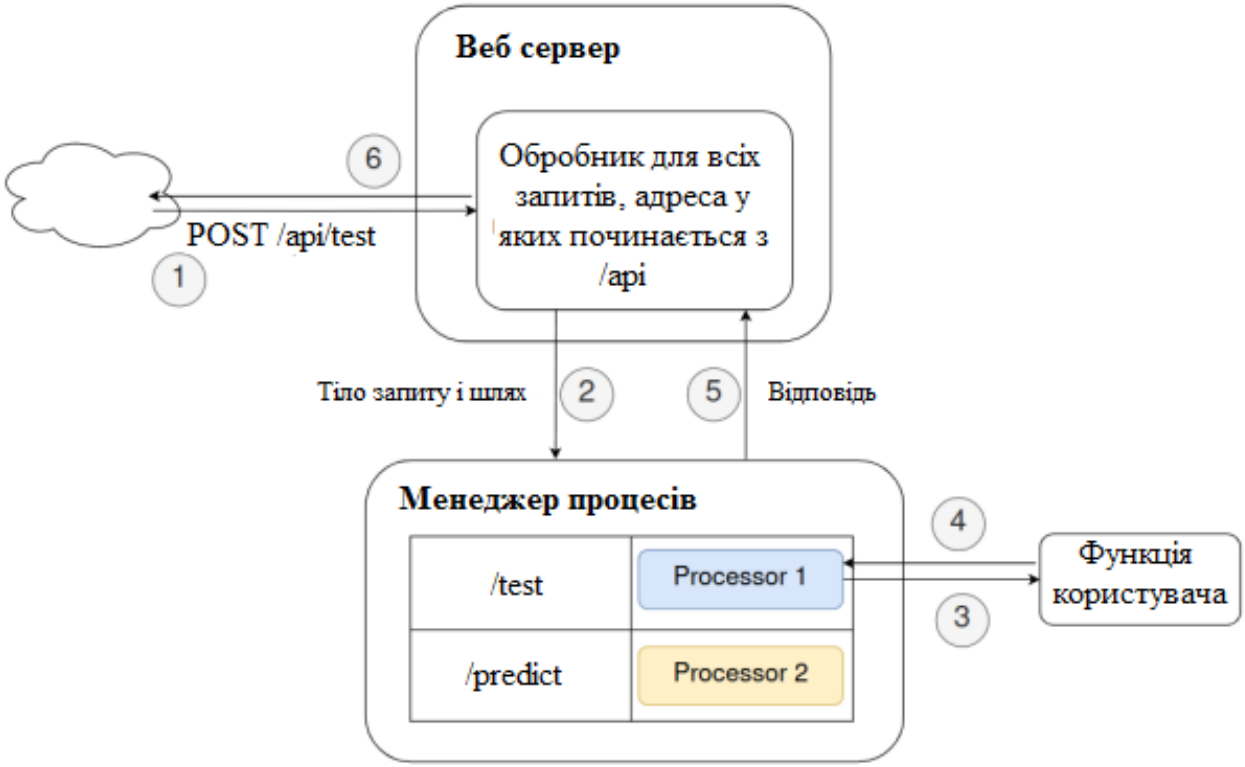


Рисунок 3.14 – Схема обробки запиту користувача

Отримавши запит, менеджер процесорів робить спробу знайти процесор, який відповідає заданому шляху. У разі неуспіху, метод припиняє своє виконання та користувачу повертається статусний код 404 Not Found [26]. Якщо процесор знайдено, запит делегується у нього.

При отриманні запиту здійснюється спроба зіставлення JSON -об'єкта з

параметрами функції, а також приведення типів. У разі неуспішної спроби зіставлення генерується помилка і користувачу повертається відповідь з кодом 400 Bad Request [26]. Інакше отримані параметри передаються в функцію користувача. Результат виконання функції повертається по ланцюжку вгору.

3.3.2 Генерація OpenAPI специфікації

За генерацію специфікації у форматі OpenAPI відповідає об'єкт класу specificationGenerator (далі Генератор специфікації).

На рис. 3.15 зображено послідовність дій, що виконуються Генератором специфікації. Для ініціалізації процесу генерації специфікації має бути викликаний метод generate. Після цього генератор специфікацій викликає метод iterator менеджера процесорів, щоб отримати ітератор [27] по множині процесорів.

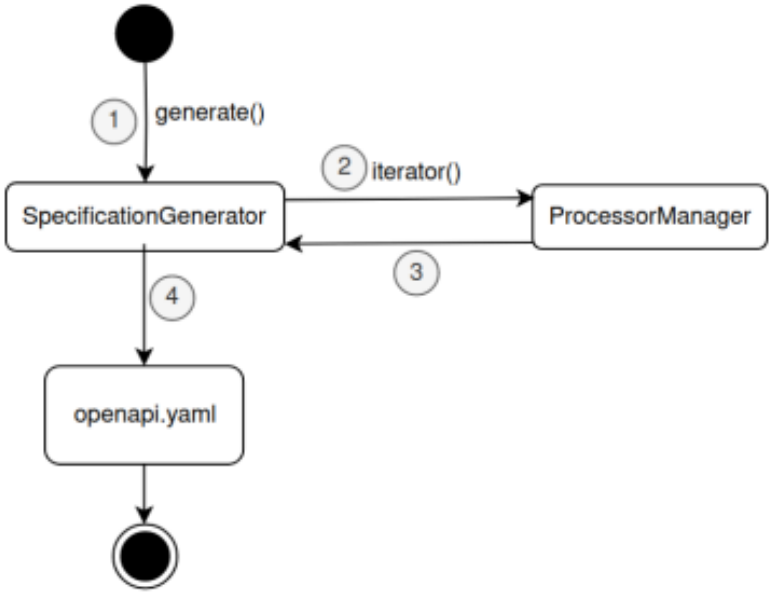


Рисунок 3.15 – Послідовність дій, що виконуються при генерації специфікації

Процесор містить у собі метод toOperation, який повертає опис даного процесора, що відповідає вимогам до опису операції [28]. Для кожного процесора,

Генератор специфікацій викликає зазначений метод і додає отриманий опис до специфікації, що генерується в даний момент.

Після того, як кожен процесор був описаний, специфікація зберігається у файл "openapi.yaml" Надалі він може бути використаний як для генерації клієнтів для систем-споживачів даного API, так і для відображення в Swagger UI [29]. Приклад сторінки Swagger UI, для створеної специфікації зображено на рис. 3.16. На ньому представлений створений користувачем метод, вказаний приклад тіла запиту, і навіть приклад відповіді.

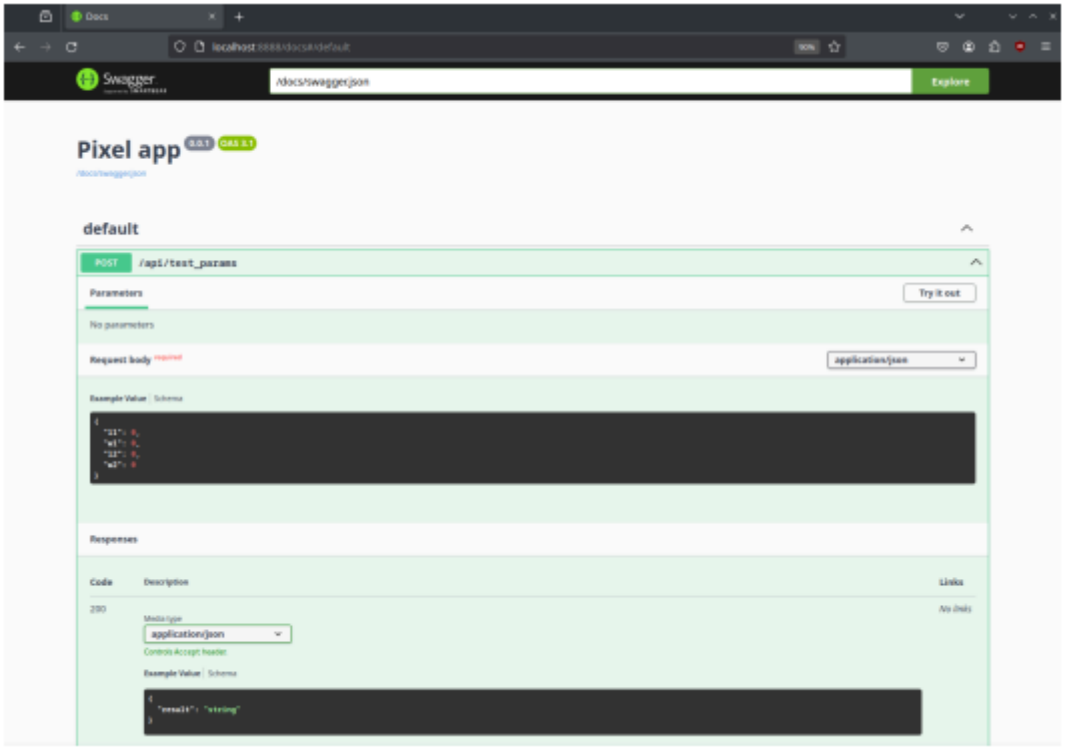


Рисунок 3.16 – Swagger UI, створений на базі згенерованої специфікації

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при ураженні електричним струмом

Удар електричним струмом є поширеною травмою і часто може закінчитися летальним випадком. Ураження електричним струмом відбувається підчас контакту тіла людини з джерелом електричної енергії під напругою. Це реакція організму людини на проходження електричного струму через тіло. Вона проявляється по різному, від легких потрясінь до небезпечних здоров'ю травм, які можуть вплинути на тканини в організмі.

Шкода завдана електричним струмом залежать від кількох факторів: наскільки висока була напруга, яка область тіла була вражена і від виду струму. Фізичні наслідки для людини можуть коливатися від опіків частин тіла до серйозних вражень внутрішніх органів [30].

Часто люди стикаються з підвищеним ризиком ураження високою напругою. Низька напруги не наносить серйозних травм для люди, а з іншої сторони висока напруги, яка має більше 500 В, може призвести до серйозного пошкодження тканин. У дітей або підлітків при враженні електричним струмом в діапазоні від 110 до 200 В можуть вникнути значні травми. Зазвичай це трапляється підчас порушення техніки безпеки при роботі з електричними приладами, які є в побуті. Це можуть бити електричні шнури, подовжувачі, несправні розетки та багато чого іншого.

Виділяють чотири основні фактори від яких залежить вплив електричного струму на організм: величина струму, що протікає через тіло; органи, через які проходить струм; час, протягом якого струм вражає тіло; частота струму.

Фізичні наслідки на пряму залежать від величини струму, яка вражає тіло людини. Струм менший за 1 мА не завдає жодної шкоди людина фізичного ефекту. При 1 мА можуть відчуватися слабкі поколювання, а при 5 мА – легкий поштовх. Однак при струмі ведичною від 6 до 25 мА людина може відчувати больовий шок і деяку втрату контролю над м'язами.

При ураженні електричним струмом від 50 до 150 мА може спричинити у людини сильний біль, м’язове скорочення і навіть призвести до зупинку дихання. У певних випадках можливий летальний результат для людини. Струм від 1000 до 4300 мА призводить до ймовірної смерть, тому що дана напруга спричиняє пошкодження нервових зав’язків, м’язових скорочення та порушення ритму серця. До 10 000 мА електричний струм спричиняє важкі опіки шкіри людини та зупинку серця. Висока ймовірність смерті.

Найпоширеніші ознаки та симптоми враження електричним струмом включають:

– втрата свідомості;
– ускладнення або зупинка дихання;
– опіки, які виникають там, де струм входить і виходить з тіла;
– зупинка серця;
– слабкий і непостійний пульс або його припинення.

Перш за все, що потрібно зробити при першій допомозі, це відключити джерело живлення. Вимкнути електропостачання, від’єднати електроприлад від джерела електричного струму або вимкнути блок запобіжників, якщо він знаходиться неподалік. Не потрібно намагатися підходити близько до жертви, якщо не переконані, що це безпечно і живлення вимкнено.

Потрібно бути обережним у вологих місцях, тому що вода є електричним провідником і рятівник може стати теж жертвою. Якщо людина не впевнена щодо вологості поверхні, потрібно відключити основне електропостачання будинку. У випадку коли це неможливо, використати підручний предмет, який не є провідником і відокремити людину від джерела струму. Це може бути дерев’яна або пластмасова річ.

Після того як постраждалого відокремили від джерела електрики, потрібно викликати швидку і надати першу медичну допомогу. Далі потрібно визначити стан жертви. Перевірити, чи людина у свідомості і дихає. У складних випадках у жертви може бути слабкий пульс або його відсутність. Можливо, що дихання зупинилося. Якщо людина втратила свідомість і перестала дихати, потрібно почати

серцево-легеневу реанімацію. Руки розташовуємо в центрі грудної, одна на іншу. Сильно і швидко натиснути 30 раз приблизно до третини діаметра грудної клітки. Після кожного натискання на грудну клітку робиться два рятувальні вдихи. Потрібно відкинути голову потерпілого назад і підняти підборіддя. Затиснути ніс і створити повне ущільнення. Далі подути потерпілому в рот і подивитися, чи підніметься грудна клітка. Потрібно продовжувати робити натискань на грудну клітку та вдих, поки не прибуде медична допомога або людина не почне сама дихати. Якщо потерпілий живий, перемістити його у зручне йому положення подальше від небезпеки. Можна запобігти шоку, поклавши людину рівно на землю, з головою трохи нижче тіла [31].

Якщо людина при свідомості, нормально дихає і на тілі є опіки, потрібно накрити їх звичайною харчовою плівкою або іншою неклейкою пов’язкою, але без мазі чи лосьйону. Якщо кровотеча у потерпілого, може знадобитися компресія та джгут.

При роботі з соціальною мережею користувач повинен знати і вміти як правильно поводитися з ПК, тому що людина перебуває у непосередньому контакті з джерелом напруги. Удар електричним струмом є потенційно смертельною травмою. Негайна медична допомога важлива, щоб запобігти серйозним травмам і смерті. Для запобігання уражень електричним струмом при роботі за ПК слід встановити додаткові захисні пристрої, що забезпечують недоступність токопровідних частин для дотику. Для зменшення небезпеки використовувати розділовий трансформатор для розв'язки з основною мережею.

4.2 Вимоги ергономіки до організації робочого місця оператора ПК

Робоче місце – це ділянка простору, яка облаштована необхідним обладнанням, відповідно до трудової діяльності, для виконання поставлених завдань.

Правильно побудоване робоче місце повинне забезпечувати:

– найкраще розміщення обладнання і предметів праці;

– не допускати дискомфорту;

– підвищувати продуктивність праці;

– зменшувати втому працівника.

Розмір робочого місця повинен бути таким, щоб людина не виконувала лишніх рухів і не відчувала дискомфорту під час виконання роботи. Також для працівника важливо мати змогу змінити робочу позу, наприклад, положення тулуба, рук або ніг. Потрібно мінімізувати або звести до нуля всі незручності положення тіла [31].

Різні дослідження заявляють, що при правильному проектуванні робочого місця продуктивність людини може зрости від 15-25%. Такі фактори як рівень освітлення, вологість повітря, температура, шум, вібрація, токсичність, мають значний вплив на умови життєдіяльності і працездатності людини.

Антропометричні вимоги визначають відповідність робочого місця до фізіологічних параметрів тіла людини як зріст і розміри тіла. Індикатором цього є правильна робоча поза, відсутність дискомфорту, оптимальні зони досягнення, раціональні рухи. Психофізіологічні та фізіологічні вимоги формулюють відповідність обладнання і робочого місця можливостям співробітника щодо розуміння, обробки даних, пошук і реалізації рішень.

Організація робочого місця передбачає наступні пункти:

– раціональне положення робочого місця у приміщенні;

– вибір робочих меблів відповідно до фізіологічних характеристик працівника;

– правильне компонування і розміщення обладнання на робочих місцях;

– урахування особливостей та характеру професійної діяльності.

До загальних принципів організації робочого місця відносять:

– робоче місце повинне містити тільки ті предмети, які беруть учать у робочому процесі, але не заважати йому;

– предмети, які часто використовуються у роботі, розміщуються ближче,

ніж ті речі, якими користуються рідше;

– предмети, які беруться лівою рукою, повинні розміщуватися зліва, а предмети, які використовуються правою рукою — справа;

– якщо при роботі з предметом працівник використовує дві руки, то він розміщується з урахуванням зручності захоплювання його двома руками;

– робоче місце не повинно бути засмічене;

– необхідна оглядовість повинна бути забезпечена при правильній організації робочого місця.

Робоча поза – це найбільш тривале положення тіла працівника протягом робочого дня. При зручній робочій позі забезпечується стійкість положення тулуба, ніг, рук, голови і витрачається мінімальний запас енергії та максимальну продуктивність [31].

Сидячи і стоячи – дві найпопулярніших пози у робочому процесі. При проектуванні робочого місця потрібно враховувати, що з фізичним навантаженням бажана поза стоячи, а при малих зусиллях – сидячи. При роботі стоячи, людина стомлюється більше ніж сидячи. У відсотковому еквіваленті це на 10% більше енергії. При додатковому навантаженні підвищується артеріальний і венозний тиск крові, розширення вен, пошкоджуються ступені та викривляється хребет. У свою чергу при сидячій роботі нижня частина тіла розслаблена, а основне навантаження спрямоване на м'язи шиї, спини, таза, стегон. При неправильній сидячій позі розвивається застій крові у ногах, а якщо пальці виконують багато роботи можливе запалення суглобів.

Організація робочого місця при використанні ПК повинна відповідати усім ергономічним вимогам. Ключові ергономічні вимоги до проектування робочого місця оператора ПК зображені на рисунку 4.1.

При роботі з персональним комп’ютером потрібно:

– зменшувати кількість статичних напружень;

– розподіляти кількість і час статичних напружень;

– змінювати робочі пози під професійної діяльності.

Саме вибір правильної робочої пози визначається від впливу багатьох факторів. Одні з найважливіших це – кількість зусиль яка прикладається, велична робочої зони, відношення висоти робочої поверхні і ростом працівника.

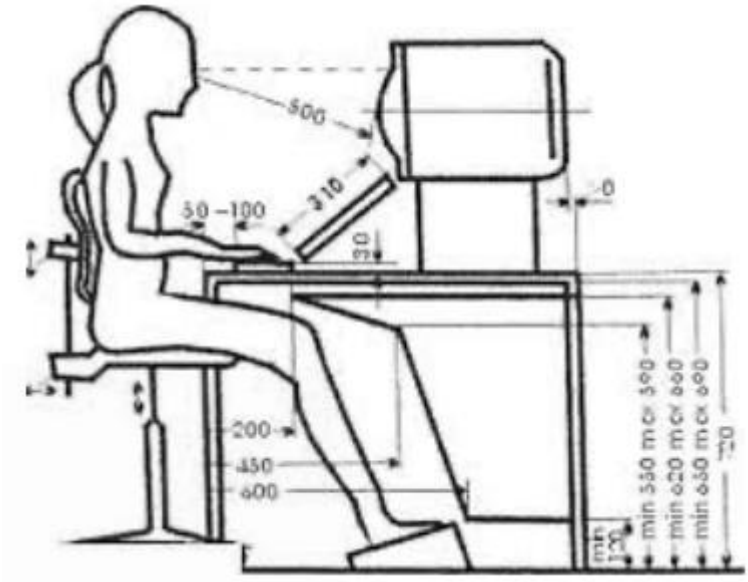


Рисунок 4.1 – Робочий стіл і розміщення користувача ПК

При використанні розробки користувач повинен дотримуватися вище зазначених правил, які будуть сприяти комфортній і продуктивній роботі. Важливо регулярно робити короткі перерви. Часта зміна заняття – кращий спосіб уникнути можливих неприємностей.

ВИСНОВОК

В результаті виконання кваліфікаційної роботи було успішно розроблено фреймворк для створення веб-інтерфейсів для візуалізації даних та ML-моделей.

Було виконано такі завдання:

– розроблено API фреймворку;

– розроблено модуль, котрий відповідає за виконання коду користувача;

– розроблено модуль, що реагує на зміни в користувальницькому коді;

– розроблено JavaScript модуль, що відповідає за створення веб-інтерфейсів на основі коду користувача;

– розроблено модуль, що відповідає за генерацію HTTP API і OpenAPI специфікації на основі коду користувача.

Створене в рамках даної роботи рішення дозволить створювати веб-застосунків без наявності специфічних для серверної розробки та веб-розробки знань. Що, у свою чергу, допоможе збільшити довіру до ML-моделей, наблизить моделі до користувача та спростить інтеграцію моделей зі сторонніми системами.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The research landscape on the artificial intelligence: a bibliometric analysis of recent 20 years / Hui Gao & Xiuhao Ding // Springer. 2022. P. 1-13.

2. Can Deep Clinical Models Handle Real-World Domain Shifts? / Jayaraman J. Thiagarajan, Deepta Rajan, Prasanna Sattigeri // arXiv preprint. 2018. P. 6-7.

3. Challenges and trends. // Data Science Salon – State of ai in the enterprise 2023 report. [Електронний ресурс] - Режим доступу до ресурсу: https://www.datascience.salon/state-of-ai-in-the-enterprise-2023-report/ (дата звертання: 10.04.2025).

4. Dash Python User Guide // Dash Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://dash.plotly.com/ (дата звертання: 10.04.2025).

5. Streamlit // Streamlit Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://streamlit.io/ (дата звертання: 10.04.2025).

6. Gradio // Gradio Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://www.gradio.app/ (дата звертання: 10.04.2025).

7. Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild / Abubakar Abid, Ali Abdalla, Ali Abid, Dawood Khan, Abdulrahman Alfozan, James Zou // arXiv preprint. 2019. P. 1

8. OpenAPI Specification // OpenAPI Initiative. [Електронний ресурс] - Режим доступу до ресурсу: https://spec.openapis.org/oas/latest.htm l (дата звертання: 13.04.2025).

9. Panel // Panel Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://panel.holoviz.org/ (дата звертання: 13.04.2025).

10. PEP 318 – Decorators for Functions and Methods // Python Enhancement Proposals. [Електронний ресурс] - Режим доступу до ресурсу: https://peps.python.org/pep-0318/ (дата звертання 14.04.2025).

11. The MD5 Message-Digest Algorithm // RFC. [Електронний ресурс] - Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc1321 (дата звертання 16.04.2025).

12. Data Structures // Python Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.python.org/3/tutorial/datastructures.html (дата звертання: 17.04.2025).

13. Command line tool and library for transferring data with URLs // Curl. [Електронний ресурс] - Режим доступу до ресурсу: https://curl.se/ (дата звертання 18.04.2025).

14. Models // Pydantic Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.pydantic.dev/latest/concepts/models/ (дата звертання: 18.04.2025).

15. Exec // Python Documentation - Built-it Functions. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.python.org/3/library/functions.html#exec (дата звертання: 20.04.2025).

16. Garbage Collector interface // Python Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.python.org/3/library/gc.html (дата звертання 20.04.2025).

17. Datamodel // Python Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.python.org/3/reference/datamodel.html (дата звертання 21.04.2025).

18. Import // Python Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.python.org/3/reference/import.html#__spec__ (дата звертання 22.04.2025).

19. PEP 420 – Implicit Namespace Packages // Python Enhancement Proposals. [Електронний ресурс] - Режим доступу до ресурсу: https://www.python.org/dev/peps/pep-0420/ (дата звертання 22.04.2025).

20. Watchdog // Watchdog Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://python-watchdog.readthedocs.io/en/stable/index.html (дата звертання 23.04.2025).

21. The WebSocket Protocol // RFC. [Електронний ресурс] - Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc6455 (дата звертання: 24.04.2025).

22. JavaScript // MDN. [Електронний ресурс] - Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звертання: 24.04.2025).

23. Markdown // Daring Fireball. [Електронний ресурс] - Режим доступу до ресурсу: https://daringfireball.net/projects/ markdown/ (дата звертання: 24.04.2025).

24. HTML5 // W3C. [Електронний ресурс] - Режим доступу до ресурсу: https://www.w3.org/TR/2014/WD-html5-20140617/ (дата звертання: 24.04.2025).

25. Window: alert() method // MDN. [Електронний ресурс] - Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/Window/alert (дата звертання 26.04.2025).

26. Hypertext Transfer Protocol // RFC. [Електронний ресурс] - Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc2616 (дата звертання 28.04.2025).

27. Iterator // Python Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.python.org/3.10/ glossary.html#term-iterator (дата звертання: 29.04.2025).

28. Operation Object // OpenAPI-Specification GitHub repository. [Електронний ресурс] - Режим доступу до ресурсу: https://github.com/OAI/ OpenAPI-Specification/blob/main/versions/3.1.0.md#operation-object (дата звертання: 02.05.2025).

29. Swagger UI // Swagge. [Електронний ресурс] - Режим доступу до ресурсу: https://swagger.io/tools/swagger-ui/ (дата звертання: 02.05.2025).

30. Яремко З. М. Безпека життєдіяльності: Навч. посіб. Львів., 2005. 301с .

31. Желібо Є. П. Заверуха Н.М., Зацарний В.В. Безпека життєдіяльності. Навчальний посібник. К.: Каравела, 2004. 328 с.

32. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: https://dl.tntu.edu.ua/bounce.php?course=5329.

ДОДАТКИ

Додаток А. Диск з кваліфікаційною роботою бакалавра