Полтавський університет економіки і торгівлі

Навчально-науковий інститут заочнодистанційного навчання

Форма навчання заочна

Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«__»_____202_ р.

КВАЛІФІКАЦІЙНА РОБОТА
на тему
«ПРОГРАМНА РЕАЛІЗАЦІЯ САЙТУ ЗІ СТАТИСТИКОЮ ІГРОВИХ ПРОФІЛІВ»

зі спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавр

Виконавець роботи Кряж Олександр Володимирович

_____«__»_____202_р.

(підпис)

Науковий керівник к.ф.-м.н.,доц., Черненко О.О

_____«__»_____202_р.

(підпис)

Рецензент

ПОЛТАВА 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, символів, скорочень, термінів
Destiny 2	Комп’ютерна гра в жанрі action-adventure з елементами RPG.
Bungie	Ігрова студія, розробник комп’ютерної гри «Destiny 2».
Екзотичне спорядження	Рідкісний предмет (екіпірування чи зброя) з унікальними особливостями в грі «Destiny 2».
Підклас	Набір унікальних здібностей у грі «Destiny 2».
Хеш	Відображення даних у вигляді унікального, короткого набору символів.
API	Application Programming Interface. Програмний інтерфейс, зазвичай входить в опис інтернет-протоколу, програмного каркасу, або стандарту викликів функцій операційної системи.
UI	User Interface. Користувацький інтерфейс
Bungie ID	Ідентифікатор гравця на сайті Bungie і в грі «Destiny 2», який складається з обраного гравцем імені й набором із чотирьох цифр, які об’єднані хештегом (Приклад: Player#7777).
DLC	Downloadable content. Доповнення/додатковий контент, зазвичай покупний.
Фронтенд	Публічна частина вебдодатків, з якою користувач може взаємодіяти.
Бекенд	Серверна частина сайтів/вебдодатків, яка відповідає за оброку даних, взаємодію з базами даних та ін.

ВСТУП

Протягом останніх десяти років у сфері комп'ютерних розваг розвивається новий напрям торгівлі цифровими продуктами. Об'єктом продажу можуть бути внутрішньоігрові предмети, допомога від досвідчених гравців або ігрові акаунти. У такій грі, як «Destiny 2», збір інформації, яка повинна зацікавити потенційних покупців ускладнює процес торгівлі для продавців. Створення оголошення для одного акаунту може займати багато часу. Відсутність інструменту для швидкого перегляду необхідної інформації може сповільнити роботу і знижує зацікавленість покупців.

У такій ситуації зручний вебзастосунок із динамічним оновленням інформації покращить якість процесу для всіх його учасників, підвищить швидкість угод і загальну якість торгівлі, а також стане зручним інструментом для не комерційного застосування звичайними користувачами для перегляду власної статистики.

Мета кваліфікаційної роботи – спроєктувати та реалізувати сайту з використанням програмного прикладного інтерфейсу для відстежування статистики ігрових профілів «Destiny 2».

Для досягнення мети були поставлені такі завдання:

1) Ознайомитися з вебзастосунками, які працюють з Bungie API для отримання інформації профілю. Визначити їхні переваги та недоліки.
2) Ознайомитися з документацією Bungie і принципами роботи з API.
3) Розробити алгоритм роботи вебзастосунку.
4) Розробити алгоритм роботи користувацького інтерфейсу.
5) Реалізувати алгоритм обробки запитів до Bungie API.
6) Реалізувати користувацький інтерфейс.
7) Запустити вебзастосунок на хостинг-сервісі.

Об'єктом розробки є процес пошуку користувачами інформації щодо конкретних ігрових профілів гри «Destiny 2». Зокрема, під час торгівлі ігровими акаунтами.

Предметом розробки є проєктування і програмна реалізація сайту, що шукає та виводить статистику обраного профілю за допомогою Bungie API.

Під час реалізації кваліфікаційної роботи було використано середовище візуальної розробки програм Microsoft Visual Studio, об'єктно-орієнтована мова програмування Python, об'єктно-орієнтована мова програмування JavaScript, мова розмітки HTML, мова таблиць та стилів CSS, Bungie API та середовище PythonAnywhere.

Практична цінність розробки підтверджена її використанням у реальних умовах - у сфері малої торгівлі на інтернет-форумі EpicNPC. За результатами тестування було виявлено значне зменшення часу, витраченого на створення оголошень до акаунтів.

Вебзастосунок має ступінь готовності MVP (minimum viable product), задовольняє необхідні потреби й має потенціал розширення через збільшення предметів статистики, що відстежується, порівняння профілів між собою прямо у вебзастосунку за допомогою паралельних запитів.

Обсяг пояснювальної записки: 94 стор., у т.ч. основна частина 45 стор., додатки – 35 стор., джерел - 15 назв.

1 ПОСТАНОВКА ЗАДАЧІ

Задачею кваліфікаційної роботи є проєктування та програмна реалізація сайту для відстежування статистики ігрових профілів для оптимізації часу на створення оголошень під час торгівлі ігровими акаунтами. Для початку виконання роботи потрібно скласти алгоритм роботи вебзастосунку. План роботи наступний:

1) переглянути приклади подібних робіт;
2) розглянути теоретичний матеріал із розробки сайтів із використанням відкритих API;
3) розробити алгоритм запиту й отримання інформації від Bungie API;
4) розробити алгоритм підстановки отриманої інформації у вебзастосунок;
5) реалізувати процес запиту й отримання інформації від Bungie API;
6) на основі аналізу подібних робіт розробити дизайн вебзастосунку;
7) реалізувати процес підстановки отриманої інформації у вебзастосунок;
8) реалізувати запуск вебзастосунку на онлайн хостингу.

Серверна частина вебзастосунку повинна знаходити в списку визначень необхідну інформацію і виводити лише предмети, які вже отримані на акаунті. Предметами пошуку будуть наступні елементи:

1) Bungie ID акаунту;
2) придбані доповнення (DLC);
3) екзотичне спорядження;
4) екзотична зброя;
5) розблоковані підкласи.

2 ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Класифікація вебзастосунків зі збором статистики

Розробка сайтів для збору та перегляд різної ігрової статистики завжди була ініціативою спільноти навколо гри, а не її розробників, тому саме гравці створюють сайти, які використовують API гри для взаємодії з нею, для особистих цілей, або для покращення життя усієї громади навколо гри.

У залежності від поставлених задач, можна виділити кілька типів сайтів та їхні особливості:

а) сайти для персонального менеджменту:
   1) потребують вхід в акаунт;
   2) можуть взаємодіяти з ігровими предметами (переносити їх зі сховища на персонажів, розбирати та ін.).

б) сайти для перегляду статистики:
   1) не потребують вхід в акаунт для перегляду статистики гравця;
   2) інколи потребують дозвіл на перегляд неекіпірованих предметів;
   3) мають різноманітні тематики (загальна статистика, статистика для особливих завдань чи ігрових режимів).

в) сайти для аналізу предметів:
   1) мають спільну базу предметів гри;
   2) допомагають переглядати різноманітні комбінації особливостей предметів і аналізувати їхню взаємодію.

2.2 Огляд схожих вебзастосунків

Перед розробкою вебзастосунку необхідно переглянути схожі сайти, які відображають глобальну статистику, або створені для вирішення локальних проблем, з якими можуть зіткнутися користувачі. Аналіз роботи обраних сайтів допоможе краще визначити актуальність і цінність проєкту, а також виявити шляхи для його покращення.

Для аналізу було обрано дві схожих за функціоналом роботи. Перший сайт - Time Wasted on Destiny, створений Франсуа Аллардом у 2023 році [1].

На рис. 2.1 продемонстровано, як під час входу на сайт користувач бачить мінімалістичний і зрозумілий дизайн, рядок пошуку, який є основною частиною і головним об’єктом сайту.

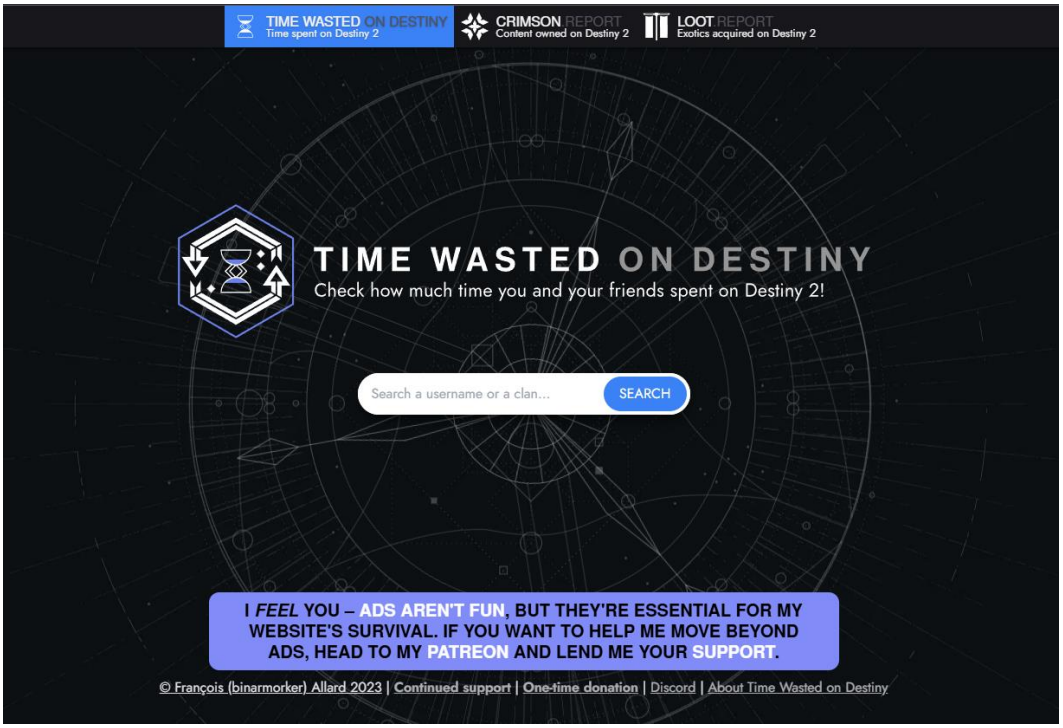

Рисунок 2.1 – Головна сторінка вебзастосунку №1

Вебзастосунок дає можливість шукати профіль як за повним Bungie ID, так і використовуючи лише ім’я профілю, надаючи список гравців із даним іменем і різними числовими тегам. Після завантаження профілю гравця сайт відображає статистику проведеного часу загалом, на видалених персонажах, а також в окремих режимах, як показано на рис. 2.2.

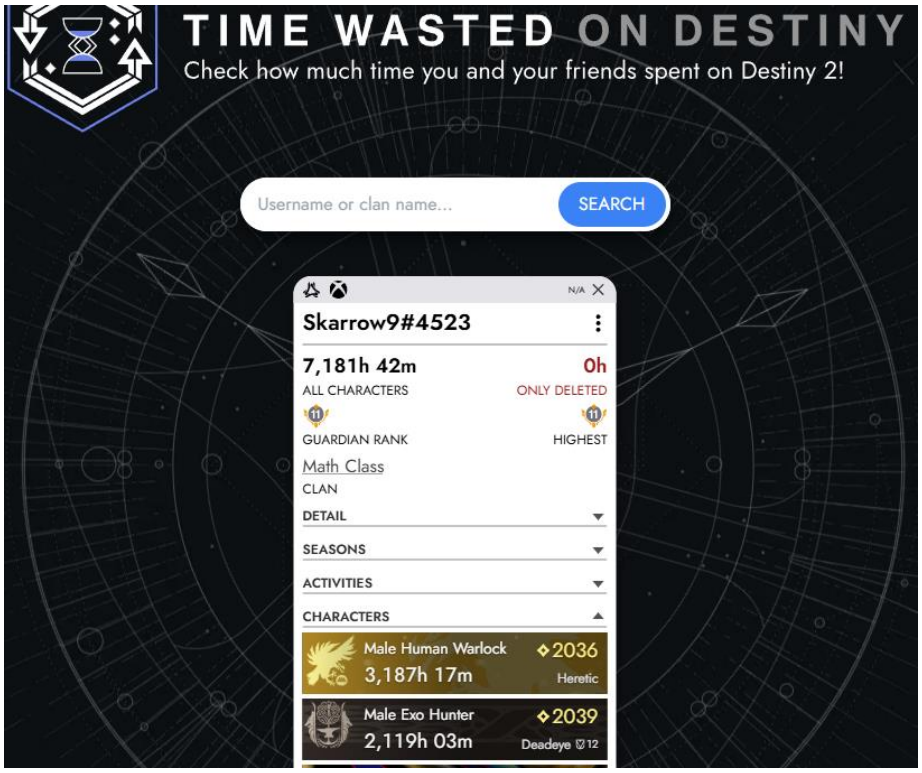

Рисунок 2.2 – Демонстрація статистики

Сайт є дуже простим у використанні, зрозумілим і не перевантаженим зайвою інформацією, користувач отримує те, що він потребує. Недоліком даного проєкту є відсутність повідомлення про помилку в разі пошуку неіснуючого імені.

Далі буде розглянутий більш спеціалізований і простий вебзастосунок - Destiny 2 Reward Checker by Mijago [2].

Одним з ігрових предметів є емблеми, які можна отримати за виконання завдань, або введені унікального чи універсального коду, який необхідно ввести на сайті розробника, щоб зарахувати його на свій акаунт. Destiny 2 Reward Checker має повний список універсальних кодів, а також допомагає відстежити, які з них уже активовані на вашому акаунті.

На рис. 2.3 показано, як під час відкриття сайту одразу можна побачити рядок пошуку, інструкцію до використання сайту, а також список усіх емблем, якщо користувач уже знає, що саме він хоче знайти й активувати. Після пошуку гравця сайт відображає вже зараховані емблеми зеленим кольором.

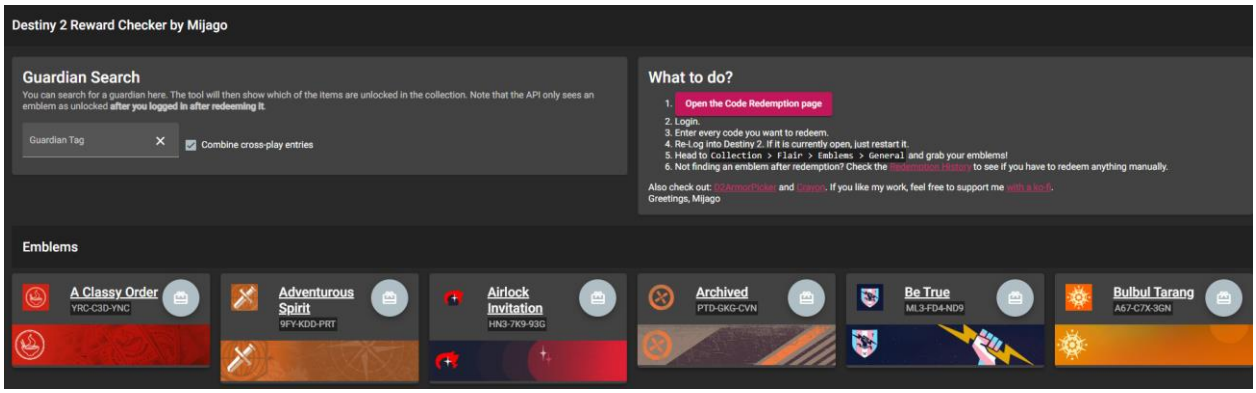

Рисунок 2.3 – Головна сторінка вебзастосунку №2

Дизайн сайту виглядає перенавантаженим, через непримітний рядок пошуку і велику кількість елементів знизу, розмір яких можна зменшити без втрати інформативності.

На прикладі цих двох вебзастосунків можна зробити висновок, що кожен із них має свою особливу тему й мету зробити отримання інформації щодо цієї теми найбільш доступною.

2.3 Вимоги до вебзастосунків зі збором статистики

На основі проаналізованих вебзастосунків, сформульовано наступні вимоги:

1) Гнучкий пошук користувача. Вебзастосунок повинен мати можливість здійснювати пошук профілю за повним Bungie ID, а також лише за іменем, надаючи повний список гравців, чиї Bungie ID відрізняються лише числовим тегом.
2) Обробка помилок. У разі введення некоректного ігрового тегу, або виникненню проблем на стороні API, вебзастосунок повинен повертати повідомлення про поточну помилку.
3) Лаконічний інтерфейс. Інтерфейс сайту має бути мінімалістичним та інтуїтивно зрозумілим. Відображати лише ту статистику, яка стосується теми застосунку.

4) Наявність посилань на зовнішні ресурси. Враховуючи тему проєкту, вебзастосунок повинен мати посилання на зовнішні ресурси з більш специфічними темами для подальшого вивчення інформації профілю.

3 ТЕОРЕТИЧНА ЧАСТИНА

3.1 Мова програмування серверної частини вебзастосунку

Для створення елементів серверної частини вебзастосунку було використано сучасну мову програмування Python.

Python – це універсальна, інтерпретована, динамічна мова програмування. Вона була створена з метою надання доступу до програмування широкому колу користувачів. Основними причинами вибору Python як мови для створення серверної частини вебзастосунку були:

1) Велика стандартна бібліотека – за словами Марка Лутца, автора книги «Learning Python», Python вже під час першого завантаження має велику кількість стандартних бібліотек роботи з файлами, вебсервісами і базами даних, що робить мову гнучкою дає їй можливість розширюватися в будь-якому напрямку [3].
2) Простота та читабельність – простий і зрозумілий код, видалення зайвих елементів у синтаксисі і використовування відступів замість дужок дозволяє зосередитися на логіці, а не прописуванні правил.
3) Кросплатформенність – оскільки Python є інтерпретованою мовою, його можна використовувати без попередньої компіляції, а його робота через інтерпретатор забезпечує коректу роботу на будь-якій платформі.
4) Динамічна типізація – відсутність необхідності оголошувати заздалегідь тип змінної пришвидшує написання коду та тестування.

Основними поняттями Python, як об’єктно-орієнтованої мови є:

Класи – це конструкції, які можуть об’єднати ряд змінних різних типів в одне ціле. Класи можна розділити на: власні дані, підпрограми, блоки та внутрішні класи.

Об’єкти – це конкретний екземпляр класу. Об’єкти мають тип, який ідентифікується їхнім класом. На основі одного класу може бути створено безліч об’єктів, також в об’єктах можна виділити такі поняття:

а) поведінку – які методи застосувати до об’єкта;

б) ідентичність – відмінність його від інших об’єктів;

в) стан – зміни об’єкту, коли застосовуються його методи [4].

Для реалізації серверної частини вебзастосунку було обрано мікрофреймворк Flask. Він широко використовується для створення вебзастосунків на мові програмування Python. Вибір Flask було зумовлено його гнучкістю, простотою у використанні та зручною роботою з API [5].

Задачею серверної частини вебзастосунку для пошуку гравців і відображення їхньої статистики буде звернення до API, пошук необхідного профілю, пошук у цьому профілі визначень, які зв’язані з відповідними предметами, підкласами та ін.

3.2 Основні поняття про Bungie API

Bungie API – це набір інструментів, який дає можливість взаємодіяти з вебзастосунками й іграми студії Bungie. Через нього можна отримати інформацію про профілі гравців, інвентар, завдання, клани й багато іншого. Відповіді на запити повертаються у форматі JavaScript Object Notation (JSON) [6]. Це текстовий формат для зберігання і обміну даними, його особливістю є легка читабельність об’єктів. Такий формат часто використовується для обміну інформацією між сервером і клієнтом, налаштування додатків і зберігання баз даних. До його переваг також входять:

Легкість – відсутність тегів робить JSON легшим, ніж його аналог XML.

Кросплатформенність – цей формат підтримується майже будь-якою мовою програмування.

Універсальність – у файлах типу JSON можна зберігати від простих рядків і чисел до вкладених структур і інших складних видів даних.

Для роботи з Bungie API необхідно отримати унікальний API ключ на сайті bungie.net у розділі «Ваші застосунки». Він зможе надати доступ до

використання інформації акаунту, а також зможе запитувати авторизацію, якщо вона необхідна для роботи сайту.

Bungie API розроблена з метою дати звичайним користувачам взаємодіяти з ним, тому вона має зрозумілу структуру, яка складається з таких елементів:

Кінцеві точки (Endpoints) – це адреси, через які відправляються запити до API. Ці точки необхідні для отримання різних типів даних [6]. Повний список груп кінцевих точок знаходиться в таблиці 3.1.

Таблиця 3.1 Список груп кінцевих точок Bungie API

Назва	Призначення
APP	Робота з додатками, зареєстрованими в системі Bungie
USER	Інформація про користувача
CONTENT	Робота з наповненням сайту Bungie.net
FORUM	Робота з форумом сайту Bungie.net
GROUPV2	Робота з групами і кланами
TOKENS	Робота з різними винагородами (Twitch Drops, Bungie reward)
DESTINY 2	Робота з грою Destiny 2
COMMUNITY CONTENT	Інформація про роботи суспільства сайту
TRENDING	Інформація про популярні матеріали
FIRETEAM –	Інформація про активні команди Destiny 2
FIRETEAMFINDER	Робота з системою пошуку команди
SOCIAL	Робота зі списком друзів

Компоненти (Components) – логічні частини даних, які можна запитати для відповідних кінцевих точок. Вони допомагають структурувати дані для більш точних запитів. Вони дають можливість вказувати, де саме шукати

необхідні дані і створені для прискорення роботи API. Кожен компонент має свій ідентифікатор у вигляді числа. Повний їх список представлений у таблиці 3.2.

Таблиця 3.2 - Список основних компонентів Bungie API:

Назва	Ідентифікатор	Інформація, що повертає
Profiles	100	Профіль
Characters	200	Сумарна інформація про всіх персонажів профілю
ItemInstances	300	Основна інформацію про екземпляри предметів
Vendors	400	Інформація про вендорів
Kiosks	500	Інформація про спеціальний «Монумент згаслого світла»
CurrencyLookups	600	Хеш і кількість предметів, які обраний персонаж може витратити на покупку, якщо дія має ціну
PresentationNodes	700	Інформація про вузли презентації, які використовує інтерфейс для навігації
Collectibles	800	Інформація про колекцію (список предметів, який відображає, які предмети можуть бути відкриті, або вже відкриті на акаунті)

Продовження таблиці 3.2

Records	900	Інформація про тріумфи (досягнення)
Transitory	1000	інформація про дані, які можуть змінюватися занадто часто (наприклад відстежувачі статистики)

Визначення (Definitions) – це статичні дані об’єктів, які не змінюються під час гри (статичні характеристики предмета, список можливих комбінацій особливостей та ін.). Вони зберігаються у Manifest, це можна дізнатись, переглянувши неофіційний сайт, створений суспільством програмістів, які працюють з Bungie API [7].

Manifest – це енциклопедія статичних даних, яка оновлюється кожні декілька місяців із додаванням у гру нових предметів. Її можна завантажити для використання визначень локально, для зниження кількості запитів зі сторони сайту.

Сутності (Entities) – це динамічні дані, які мають інформацію про ігрові об’єкти і їхній статус на даних момент. Ці дані можуть змінюватися від дій гравця, або ігрових подій.

Повний список сутностей та кінцевих точок можна переглянути на офіційному сайті, створеному Bungie для полегшення роботи з API [8]. Повний список визначень можна переглянути на неофіційному сайті Data Explorer [9], він оснащений зручним пошуком та описом визначень, як показано на рис. 3.1.

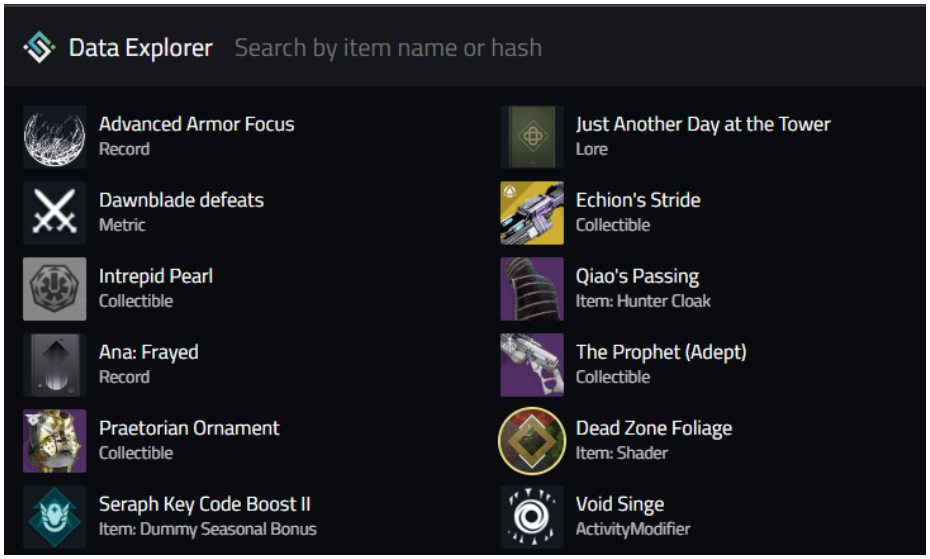

Рисунок 3.1 – Рядок пошуку і список визначень сайту Data Explorer

Підсумовуючи, основним завданням вебзастосунку буде відправлення запитів до деяких кінцевих точок API для пошуку там необхідних даних і повертання отриманої інформації до користувача.

3.3 Методи обробки даних

Різні типи даних потребують різні методи для їх отримання і перевірки. У цьому підрозділі буде розглянуто, якими методами можна отримати необхідні дані.

3.3.1 Пошук гравця

Існує два типи пошуку гравців: точний і глобальний. Для точного пошуку необхідно написати ім’я профілю разом із його унікальним числовим ID. Для глобального достатньо написати лише ім’я, щоб отримати список усіх профілів із цим іменем. Використання обох цих методів гарантує зручність користування вебзастосунком. Логіка використання обох методів одразу реалізується таким чином:

1) Код перевіряє наявність символу «#», який розділяє ім’я від коду.
2) Якщо символ знайдено використовується точний пошук

3) У всіх інших випадках використовується глобальний пошук.

3.3.2 Отримання даних профілю

Для отримання даних профілю необхідно відправити запит на його перегляд. Для цього використовується кінцева точка Destiny2.GetProfile. Для пошуку профілю необхідно знати 2 ідентифікатори:

Membership type – ідентифікатор платформи, з якої грає користувач.

- Xbox – 1;
- PSN – 2;
- Steam – 3;
- Blizzard – 4;
- Stadia – 5;

Membership id – ідентифікатор профілю, він унікальний і виглядає як набір з 19 цифр (наприклад 4611686018450092220).

Цю інформацію вебзастосунок буде отримувати після вибору користувачем профілю в рядку пошуку.

3.3.3 Отримання придбаних доповнень

Список активних доповнень зберігається у вигляді бітової маски в полі «versionsOwned», де комбінація бітів відповідає за список доповнень. Значення бітів бітової маски винесено в таблицю 3.3.

Таблиця 3.3 – Бітова маска поля «versionsOwned»

Бітова позиція	Значення	Назва доповнення
0	1	Destiny 2
1	2	DLC1 (Curse of Osiris)
2	4	DLC2 (Warmind)

Продовження таблиці 3.3

3	8	Forsaken
4	16	YearTwoAnnualPass
5	32	Shadowkeep
6	64	Beyond Light
7	128	Anniversary 30th
8	256	The Witch Queen
9	512	Lightfall
10	1024	The Final Shape

Тобто, якщо акаунт має всі доповнення, значення буде дорівнювати 2047. Але ці дані не є актуальними і потребують редагування. Бітові позиції 0,1,2,4 не будуть розглядатися, тому що: «Destiny 2» - це оригінальна гра, яку не потрібно купляти, також із 2018 року в безкоштовній версії знаходяться перші два доповнення. «YearTwoAnnualPass» активується з покупкою доповнення «Forsaken».

Для виведення актуальних придбаних доповнень необхідно ввести їхній список і значення, а потім перевірити чи встановлений відповідний біт у полі «versionsOwned».

3.3.4 Отримання відкритих підкласів

Підкласи в API – це предмети в інвентарі персонажів, а також завжди один в екіпіруванні. Усі вони зберігаються в спеціальному відділі типу «DestinyInventoryBucketDefinition», який грає роль сховища для предметів різного типу. Для пошуку необхідного сховища (Bucket), буде використовуватися його хеш (3284755031), який можна отримати через інформаційну панель визначення на сайті Data Explorer [9]. Інформація отримана через сайт продемонстрована на рис. 3.2.

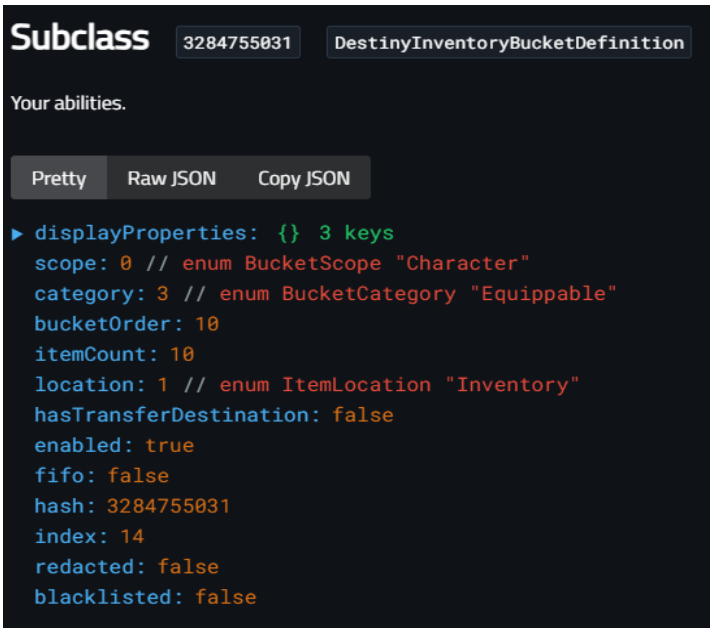

Рисунок 3.2 – Інформація про «Subclass inventory bucket»

Щоб ідентифікувати підкласи і віднести їх до різних персонажів, необхідно ввести 3 словника для кожного персонажу з їхньою назвою і хешем предмету. Значення словників можна переглянути в таблиці 3.4

Таблиця 3.4 - Підкласи персонажів і їхні хеши

Назва	Хеш
Titan	
Sunbreaker	2550323932
Striker	2932390016
Sentinel	2842471112
Behemoth	613647804
Berserker	242419885
Prismatic Titan	161634684
Hunter	
Gunslinger	2240888816
Arcstrider	2328211300
Nightstalker	2453351420
Revenant	873720784
Threadrunner	3785442599

Продовження таблиці 3.4

Prismatic Hunter	4282591831
Warlock	
Dawnblade	3941205951
Stormcaller	3168997075
Voidwalker	2849050827
Shadebinder	3291545503
Broodweaver	4204413574
Prismatic Warlock	3893112950

Далі проводиться пошук предметів в «characterInventories», значення «bucketHash» яких співпадає зі значенням «subclass_bucket_hash». А потім визначається, до яких персонажів відносяться ці предмети. Теж саме робиться для «characterEquipment», щоб отримати по одному підкласу, які екіпіровані на даний момент на персонажі.

Слід зазначити, що для пошуку предметів в інвентарі, володар акаунту повинен дати згоду не лише на перегляд його профілю, а ще й на перегляд не екіпірованих предметів у параметрах конфіденційності на сайті bungie.net.

3.3.5 Отримання відкритого екзотичного спорядження і зброї

Спорядження, яке відкрито на акаунті і спорядження в інвентарях персонажів/у сховищі прив’язані до різних визначень. Предмети можна розібрати, деякі з них можна повторно отримати через колекцію, а чи було це спорядження отримано хоча б один раз на акаунт відображає колекція. Тому кількість відкритих екзотичних предметів отримується через «DestinyCollectibleDefinition».

Для позначення, статусу предмету в колекції, як і для відображення отриманих DLC, використовуються бітові маски поля «State», значення яких винесені в таблицю 3.5.

Таблиця 3.5 – Бітова маска поля «State»

Бітова позиція	Значення	Статус
0	0	None (Немає)
1	1	NotAcquired (ще не розблоковано в колекції)
2	2	Obscured (треба використовувати інший хеш предмету)
3	4	Invisible (предмет не відображається в колекції гравця)
4	8	CannotAffordMaterialRequirements (недостатньо матеріалів для покупки)
5	16	InventorySpaceUnavailable (недостатньо місця в інвентарі)
6	32	UniquenessViolation (цей предмет не можна тримати в декількох екземплярах)
7	64	PurchaseDisabled (не можна купити через колекцію)

Позиції, які необхідно відстежувати – «NotAcquired» та «Invisible», саме вони відповідають за те, чи є той чи інший предмет у колекції гравця. Для того, щоб отримати потрібний статус, достатньо щоб ці два параметри були вимкнені (=0).

Шукається екзотичне спорядження через вузли колекцій. Кожен розділ колекції має своє визначення вузла, яке зберігає інформацію про те, що в ньому знаходиться. В JSON файлі ця інформація позначається як «children», як показано на рис. 3.3.

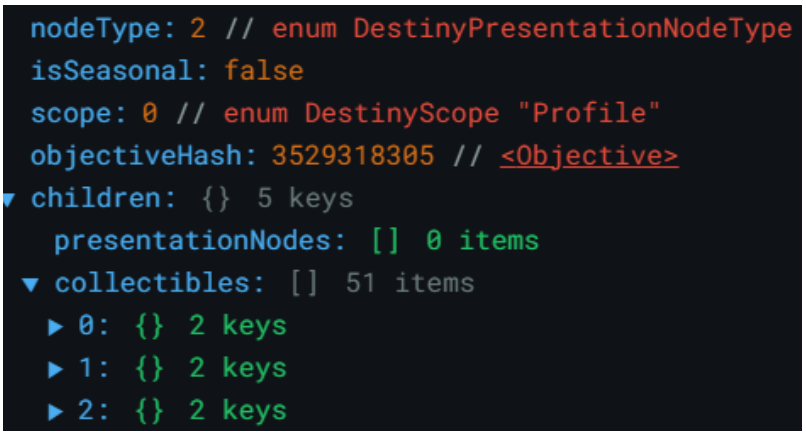

Рисунок 3.3 – Інформація про вміст вузла

Список вузлів для екзотичного спорядження різних персонажів:

- «Titan»: 2598675734;
- «Hunter»: 2765771634;
- «Warlock»: 1573256543;

Список вузлів для екзотичної зброї різних типів:

- «Kinetic»: 2969886327;
- «Energy»: 3919988882;
- «Power»: 1139971093;

За допомогою цієї інформації можна отримати список відкритих екзотичних предметів у відповідних вузлах. Для пошуку всіх предметів із вузлів слід локально завантажити «DestinyPresentationNodeDefinition».

3.4 Алгоритм роботи серверної частини вебзастосунку

Крок 0. Імпортування модулів, створення Flask-додатку

Крок 1. Встановлення API ключа, визначення словників із необхідними хешами.

Крок 2. Локальне завантаження необхідних визначенн.

Крок 3. Створення запиту на отримання даних профілю з необхідними компонентами.

Крок 4. Створення запиту на отримання куплених доповнень на основі бітового прапору.

Крок 5. Створення запиту на отримання даних персонажів і їхнього екіпірування.

Крок 6. Пошук підкласів в екіпіруванні й інвентарях персонажів.

Крок 7. Отримання екзотичних предметів із вузлів

Крок 8. Розрахунок відкритих екзотичних предметів

Крок 9. Обробка маршруту, перевірка необхідних даних для знаходження профілю.

Крок 10. Відправка запиту на отримання даних профілю з необхідними компонентами.

Крок 11. Отримання даних користувача, об’єднання глобального імені і коду користувача, отримання куплених доповнень, отримання списку відкритих підкласів, отримання відкритих екзотичних предметів.

Крок 12. Формування відповіді.

3.5 Принципи побудови UI

Користувацький інтерфейс має дуже велику роль у вебзастосунку. Він має бути функціональним і зручним для використання. Під час розробки дизайну інтерфейсу необхідно врахувати такі вимоги:

1) Мінімалізм і інформативність. Вебзастосунок не повинен бути перевантажений зайвими елементами. Усі елементи статистики мають чітко розділятися на різні блоки із заголовком.
2) Візуальне виділення. Рядок пошуку має бути розміщений зверху сайту, для того щоб звертати на себе увагу користувача. Отримані елементи статистики мають виділятися різними кольорами і контрастувати з основною гамою сайту.

3) Візуальні підказки. Назви класів, DLC та інші елементи сайту повинні мати власну символіку для можливості швидкого сприймання інформації без читання тексту.
4) Інтерактивність об’єктів. Додавання анімацій під час взаємодії з різними об’єктами на сайті допоможе знайти користувачу кнопки, текст із посиланням, а також виділить отриманий елемент статистики від заблокованого.

3.6 Мова програмування фронтенду

Для створення публічної частини вебзастосунку, з якою користувач буде взаємодіяти було використано мову програмування JavaScript.

JavaScript – динамічна, об’єктно-орієнтована прототипна мова програмування. Її часто використовують для написання логіки вебдодатків, створення стаціонарних і мобільних застосунків, а також для написання сценаріїв у прикладних програмах. Перевагами JavaScript, як мови програмування для фронтенду проєкту є:

1) Підтримка всіма інтернет-браузерами без необхідності встановлення додаткових плагінів.
2) Простота використання під час створення вебдодатків.
3) Активна спільнота, що може допомогти під час вирішення проблем у написанні коду.

Окрім логіки вебдодатку, необхідно реалізувати структуру вебсторінки за допомогою HTML (HyperText Markup Language). Для стилізування вебдодатку використовується CSS (Cascading Style Sheets). Девід Фленаган, автор книги «Javascript: The Definitive Guide», назвав ці три технології «тріадою технологій, яку веброзробник зобов’язаний вивчити» [10].

3.7 Логіка роботи фронтенду

У цьому підрозділі будуть розглянуті основні принципи роботи вебсайту на стороні користувача.

3.7.1 Рядок і кнопка пошуку

Рядок пошуку це перший елемент, з яким взаємодіє користувач. Його використання повинно бути не перевантаженим і легким. Для виконання даних вимог необхідно додати наступні особливості:

1 Кнопка пошуку змінює колір у разі наведення на неї.
2 Пошук гравця активується у разі натискання на кнопку пошуку, а також у разі натискання на клавішу Enter, якщо рядок пошуку активний.
3 Якщо серверна частина не знайшла профіль із вказаним іменем, повертає відповідне повідомлення.
4 У разі оновленні сторінки або переході на іншу адресу текст у рядку пошуку видаляється.
5 Список гравців можна закрити шляхом натискання на будь-який елемент за його кордонами.
6 Під час завантаження профілю символ пошуку змінюється на анімацію завантаження.
7 Рядок пошуку залишається зверху сторінки, навіть під час прокручуванні сторінки.

3.7.2 Блок посилань на інші ресурси

Блок із посиланнями має назви інших ресурсів із більш специфічною інформацією та відправляє користувача до інших вебзастосунків саме на сторінку обраного профілю. Даний блок не є головним елементом сайту, тому

він не повинен перенавантажувати сайт і звертати на себе багато уваги. Для виконання даних вимог необхідно додати наступні особливості:

1. Блок повинен з’являтися лише після завантаження профілю.
2. Усі тексти з посиланням мають змінювати колір у разі наведення на них.

3.7.3 Інформативні блоки

Блоки зі статистикою профіля це головна частина сайту. Вони повинні бути помітними, займати більшу частину сторінки, а елементи статистики мають виділятися на їхньому фоні. Для виконання даних вимог необхідно додати наступні особливості:

1. Блок з іменем профілю повинен бути вище інших і мати простір для відображення імен різної довжини.
2. Усі блоки статистики повинні мати заголовок із назвою предмету, що відображається на них.
3. Колір заблокованих доповнень і підкласів повинен відповідати кольоровій гамі сайту. Розблоковані доповнення і підкласи отримують кольори і стають інтерактивними (у разі наведенні картинка збільшується і стає яскравішою).
4. У разі отриманні всього можливого екзотичного спорядження свого типу, лічильник отримує ефект світіння.

3.8 Середовище розгортання вебзастосунку

Для розгортання вебзастосунку було обрано середовище «PythonAnywhere». Головними причинами виступили:

1. Підтримка більшості вебфреймворків Python.
2. Швидкий процес розгортання.
3. Активна служба підтримки.

3.9 Алгоритм роботи фронтенду вебзастосунку

Крок 0. Завантаження стилів і бекграунду.

Крок 1. Відправлення запиту на сервер для пошуку введеного користувачем імені.

Крок 2. Повертання відповіді сервера у вигляді списку імен, або повідомлення про помилку.

Крок 3. Після вибору користувачем профілю, видалення тексту з рядка пошуку, заміна символа на кнопці пошуку анімацією.

Крок 4. Завантаження даних профілю.

Крок 5. Заміна анімації завантаження на символ пошуку.

Крок 6. Заповнення блоків отриманими даними.

3.10 Діаграма послідовності

Для структурування принципу роботи сайту, було розроблено діаграму. До її елементів входять: користувач, який активує процес, фронтенд, який є мостом між користувачем і сервером, сервер, які формулює запити і відправляє їх, а також Bungie API, яка отримує запити і відповідає на них. На рисунку 3.4 зображена діаграма послідовності вебзастосунку.

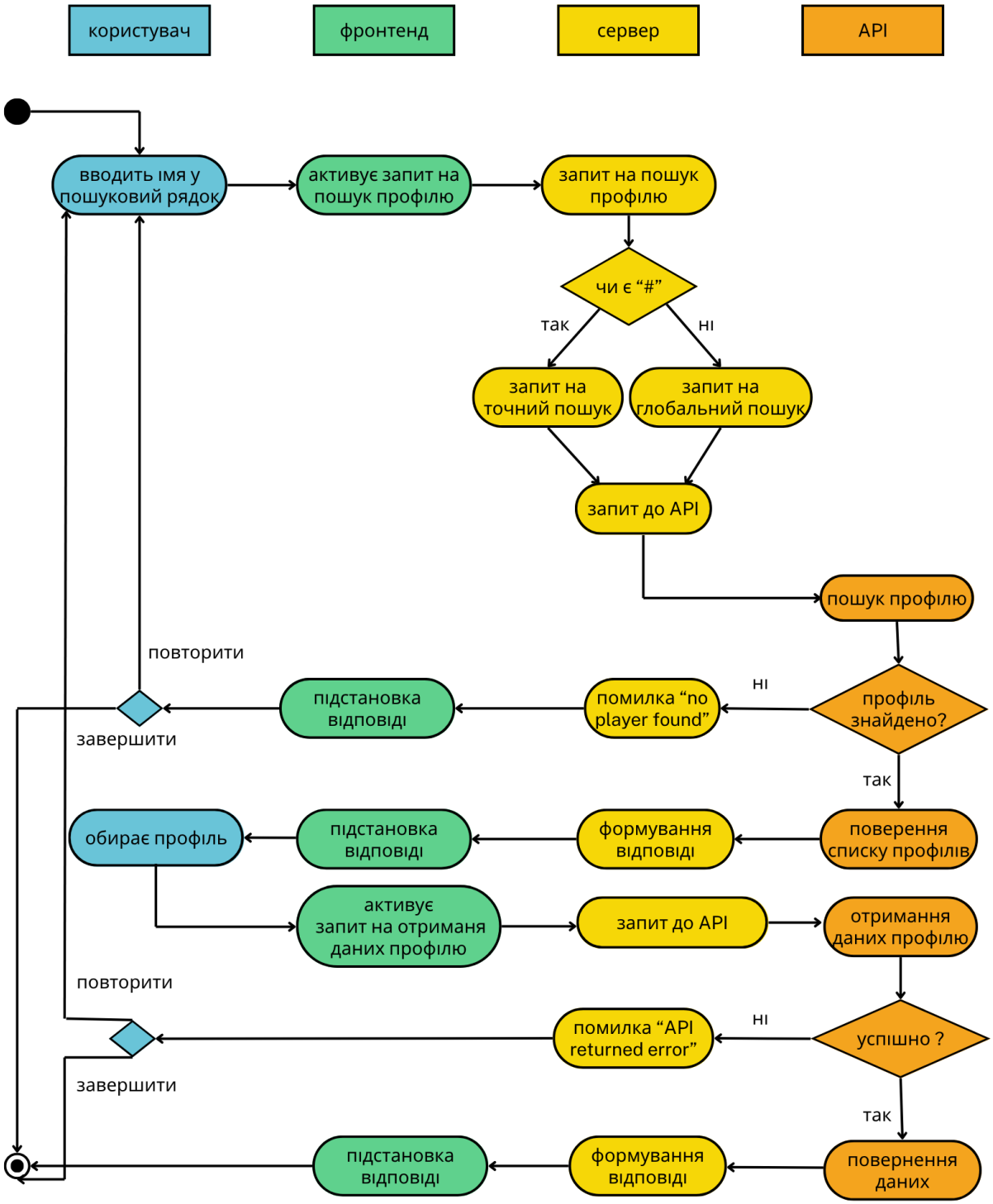

Рисунок 3.4 – Діаграма послідовності вебзастосунку

Ця діаграма демонструє послідовність дій і взаємодію користувача, вебінтерфейсу, серверу й бази даних Bungie API.

4 ПРАКТИЧНА ЧАСТИНА

4.1 Алгоритм створення нового проєкту Python в Visual Studio 2022

Для того щоб розпочати писати код, потрібно створити новий проєкт у Visual Studio. Його створення включає наступні етапи:

1. Переконатися, що пакети Python для Visual Studio встановлені. Для цього треба запустити Visual Studio Installer і натиснути на «change» у вікні доступних оновлень. Біля пункту «Розробка на Python» повинна стояти пташка.
2. Запустити Microsoft Visual Studio 2022 і обирати «Create a new project».
3. У новому вікні натиснути на «All languages» і обрати «Python». У списку обираємо шаблон «Python Application», як на рис. 4.1 і натиснути «next».

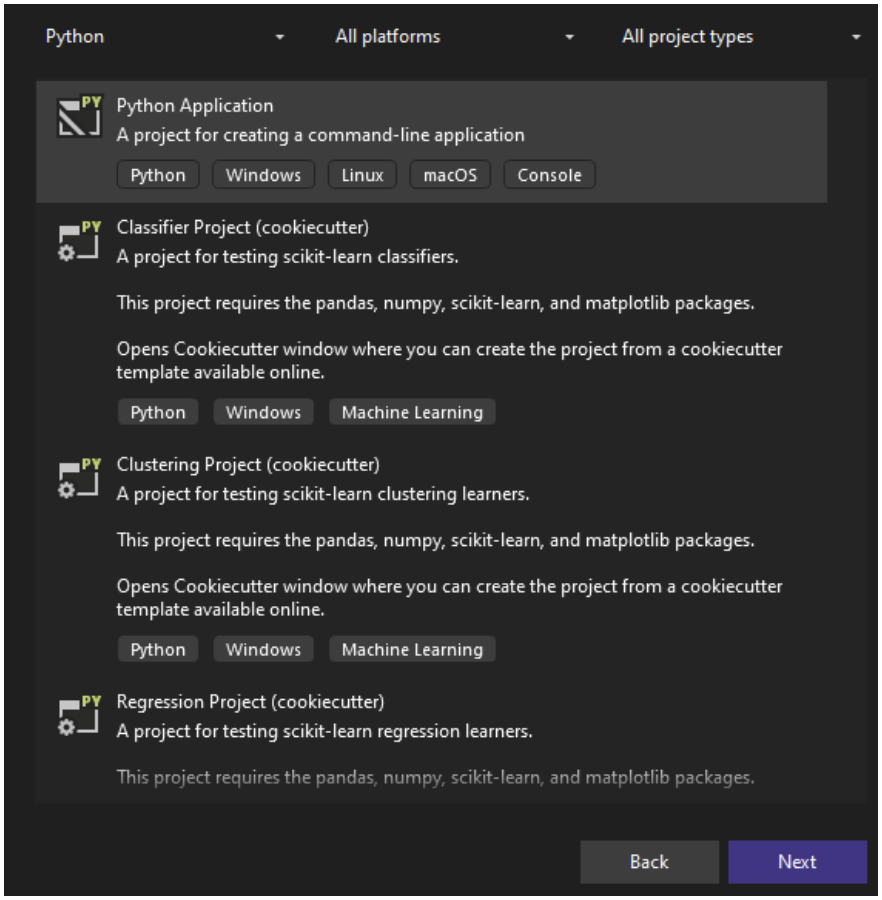

Рисунок 4.1. – Вибір шаблону

4. У новому вікні ввести ім’я проєкту і його розташування. Пташка біля «Place solution and project in the same directory» повинна бути активована. Натиснути «Create».

Середовище створено і готово для написання коду на Python.

4.2 Програмна реалізація серверної частини вебзастосунку

4.2.1 Завантаження визначень з Manifest

Для початку роботи з Bungie API треба імпортувати 2 бібліотеки, які показані на рис. 4.2 – json і requests. Json потрібен для роботи з визначеннями, а requests дає можливість відправляти HTTP-запити, щоб звернутися до Bungie API. Також потрібно встановити фреймворк для створення вебзастосунків, а також модулі і функції для роботи з HTTP-запитами, форматуванню у JSON і генерації HTML сторінок. Прописується API-ключ для доступу до Bungie API і головна адреса, з якої починаються всі запити. Ключ вписується до заголовку, які відправляються разом із запитами.

```
import json
import requests
from flask import Flask, jsonify, request, render_template
app = Flask(__name__)

API_KEY = "2d43ec6af8654c35a68d3d46d48c2e70"
BASE_URL = "https://www.bungie.net/Platform"
```

Рисунок 4.2 – Імпортовані бібліотеки і API-ключ

Наступним кроком буде завантаження визначень «DestinyPresentationNodeDefinition» з Manifest. Він знадобиться для знаходження екзотичних предметів. Для цього формується адреса для запиту даних з manifest. Перевіряється код відповіді й результат пошуку даних, на випадок помилок. Код для реалізації завантаження визначень продемонстрований на рис. 4.3.

```
def download_manifest_data():

    manifest_url = f"{BASE_URL}/Destiny2/Manifest/"

    try:

        response = requests.get(manifest_url, headers=HEADERS)
        if response.status_code != 200:
            print(f"Failed to fetch manifest: {response.status_code}")
            return

        manifest_data = response.json().get("Response", {})
        if not manifest_data:
            print("No manifest data found.")
            return
```

Рисунок 4.3 – адреса для запиту з manifest і перевірка роботи запиту

Далі отримуються шляхи до «PresentationNode» і створюється URL-адреси для завантаження даних [7].

Тепер можна завантажити дані. Для цього отримується адреса, а потім і визначення, відкривається JSON файл із кодуванням UTF-8. Зберігається відповідь і створюються відступи на випадок необхідності переглядати файл вручну. Якщо файл успішно збережено, виводиться «DestinyPresentationNodeDefenition downloaded successfully». Якщо відповідь запиту відрізняється від потрібної, виводиться код помилки разом з «Failed to download DestinyPresentationNodeDefenition», як на рис. 4.4.

```
print("Downloading DestinyPresentationNodeDefinition...")
presentation_node_response = requests.get(presentation_node_url)
if presentation_node_response.status_code == 200:
    with open("DestinyPresentationNodeDefinition.json", "w", encoding="utf-8") as f:
        json.dump(presentation_node_response.json(), f, ensure_ascii=False, indent=4)
    print("DestinyPresentationNodeDefinition.json downloaded successfully.")
else:
    print(f"Failed to download DestinyPresentationNodeDefinition: {presentation_node_response.status_code}")
```

Рисунок 4.4 – завантаження DestinyPresentationNodeDefenition

Прописується запуск функції. Для запуску треба натиснути клавішу F5, або кнопку Start на верхній панелі застосунку. Якщо завантаження пройшло успішно відображається повідомлення про це у вікні консолі, як на рис. 4.5.

```
Downloading DestinyPresentationNodeDefinition...
DestinyPresentationNodeDefinition.json downloaded successfully.
Press any key to continue . . .
```

Рисунок 4.5 – Повідомлення про успішне завантаження визначень

Тепер увесь код для завантаження даних з Manifest можна замінити на завантаження DestinyPresentationNodeDefenition локально. Для цього використовується json.load для перетворення JSON даних у Python-об’єкт, що показано на рис 4.6.

```python
def load_local_data(filename):
    with open(filename, "r", encoding="utf-8") as f:
        return json.load(f)

presentation_nodes_data = load_local_data("DestinyPresentationNodeDefinition.json")
```

Рисунок 4.6 – Локальне завантаження визначень

Після цих маніпуляцій визначення отримуються через локальний файл і не потребують чекати їх завантаження під час роботи застосунку.

4.2.2 Пошук профілю

Пошук профілю поділений на дві функції: точний і глобальний. Точний пошук отримує ім’я користувача (display_name) і його числовий тег (display_name_code). Тип платформи (membership_type) має значення -1, що означає будь-який тип платформи. На рис. 4.7 продемонстрована функція, яка формує URL-адресу з кінцевою точкою «SearchDestinyPlayerByBungieName». Створюється тіло запиту «payload» для POST-запиту з іменем і тегом гравця. Далі виконується запит із заданими параметрами (URL-адреса, тіло запиту «payload», заголовок, який включає в себе API ключ).

Відповідь повертається і аналогічно з локальними даними перетворюється з JSON файлу у Python-словник.

```python
def search_exact_bungie_name(display_name, display_name_code, membership_type=-1):
    url = f"{BASE_URL}/Destiny2/SearchDestinyPlayerByBungieName/{membership_type}/"
    payload = {
        "displayName": display_name,
        "displayNameCode": int(display_name_code)
    }
    response = requests.post(url, json=payload, headers=headers)

    return response.json()
```

Рисунок 4.7 – Точний пошук профілю

Глобальний пошук працює по схожому алгоритму. Змінюється кінцева точка для повернення списку профілів з однаковим іменем – (SearchByGlobalNamePost), а тіло запиту має лише одну змінну – ім’я профілю.

4.2.3 Отримання даних профілю

На рисунку 4.8 показана функція, в якій створюється URL-адреса для отримання профілю і задаються необхідні компоненти. Для реалізації даного проєкту потрібні наступні компоненти: Profile для виведення BungieID і доповнень, Characters, CharactersInventories і CharactersEquipment для підкласів і Collectibles для екзотичного спорядження.

```
def get_profile(membership_type, destiny_membership_id):
    url = f"{BASE_URL}/Destiny2/{membership_type}/Profile/{destiny_membership_id}/"
    params = {"components": "100,200,201,205,800"}
    response = requests.get(url, headers=headers, params=params)
    if response.status_code == 200:
        return response.json()
    else:
        return None
```

Рисунок 4.8 – Отримання даних профілю з компонентами

4.2.4 Отримання куплених доповнень

Для виведення актуальних куплених доповнень необхідно, як показано на рис. 4.9, ввести список доповнень і іхні значення, а потім за допомогою побітового оператора «&» перевірити чи встановлений відповідний біт у полі versionsOwned.

```
def parse_dlc_versions(versions_owned):
    dlc_flags = {
        8: "Forsaken",
        32: "Shadowkeep",
        64: "Beyond Light",
        128: "Anniversary 30th",
        256: "The Witch Queen",
        512: "Lightfall",
        1024: "The Final Shape"
    }
    owned_dlc = [name for bit, name in dlc_flags.items() if versions_owned & bit]
    return owned_dlc
```

Рисунок 4.9 – Отримання куплених доповнень

4.2.5 Отримання відкритих підкласів

Створюються списки підкласів для кожного персонажу (див. таблицю 3.4), отримуються дані персонажів, їхніх інвентарів і екіпірування, вводиться значення хешу сховища для підкласів і кінцеві списки. На рис. 4.10 перевіряється, чи є в інвентарі персонажу предмет, значення «bucketHash» якого дорівнює необхідному значенню, якщо так, перевіряється, до якого списку відноситься його хеш і він відноситься до відповідного списку персонажів. Теж саме проводиться для екіпірування персонажів, для того щоб знайти підклас, який використовується на даний момент.

```
#finding in characters inventories
for char_id, inventory_data in inventories.items():
    items = inventory_data.get("items", [])
    for item in items:
        if item.get("bucketHash") == subclass_bucket_hash:
            item_hash = item.get("itemHash")
            #Checking class for subclass
            if item_hash in TITAN_SUBCLASSES.values():
                subclass_name = [name for name, hash in TITAN_SUBCLASSES.items() if hash == item_hash][0]
                unlocked_subclasses["Titan"].append(subclass_name)
            elif item_hash in HUNTER_SUBCLASSES.values():
                subclass_name = [name for name, hash in HUNTER_SUBCLASSES.items() if hash == item_hash][0]
                unlocked_subclasses["Hunter"].append(subclass_name)
            elif item_hash in WARLOCK_SUBCLASSES.values():
                subclass_name = [name for name, hash in WARLOCK_SUBCLASSES.items() if hash == item_hash][0]
                unlocked_subclasses["Warlock"].append(subclass_name)
```

Рисунок 4.10 – Пошук в інвентарях персонажів

4.2.6 Отримання відкритого екзотичного спорядження і зброї

Отримується список колекційних предметів із локально завантаженого «DestinyPresentationNodeDefenition», рахуються відкриті колекційні предмети, використовуючи їхню бітову маску «state» (див. таблицю 3.5), якщо значення not acquired і invisible дорівнюють 0, предмет вважається відкритим. Логіка роботи пошуку зброї продемонстрована на рис. 4.11 і 4.12.

```
def count_opened_collectibles(profile_data, collectible_hashes):
    #bits for finding obtained exotics
    NOT_ACQUIRED = 1
    INVISIBLE = 4

    profile_collectibles = profile_data.get("Response", {}).get("profileCollectibles", {}).get("data", {}).get("collectibles", {})
    character_collectibles = profile_data.get("Response", {}).get("characterCollectibles", {}).get("data", {})

    count = 0
    opened_collectibles = []
```

Рисунок 4.11 – Отримання колекцій профілю і персонажів, позначення необхідних бітових значень.

```
for collectible_hash in collectible_hashes:
    state = None
    #finding in profile
    collectible_data = profile_collectibles.get(str(collectible_hash))
    if collectible_data:
        state = collectible_data.get("state", 0)
        if state & NOT_ACQUIRED == 0 and state & INVISIBLE == 0:
            count += 1
            opened_collectibles.append(collectible_hash)
            continue
```

Рисунок 4.12 – Пошук відкритих екзотичних предметів у профілі

Для пошуку екіпірування використовується аналогічний алгоритм, але пошук проводиться в кожному персонажі.

4.2.7 Головний обробник запиту

Реєструється маршрут «/» для головної сторінки сайту. Він повертає HTML-шаблон index.html, який буде створено для реалізації фронтенду. Створюється шлях для пошуку гравця, де буде отримано інформацію по ключу «bungie_name» через рядок пошуку. Створюється масив для профілів, якщо ім’я отримано. Якщо ім’я не отримане, повертається помилка. Далі відповідь перевіряється на наявність символу «#». Якщо символ присутній, ім’я відокремлюється від тегу і запускається точний пошук, після цього відповідь збирається в масив «players», як показано на рис. 4.13. В іншому випадку запускається глобальний пошук, який показаний на рис 4.14, створюється повідомлення про помилку, якщо API повернув порожній результат.

Отриманий список формується в зрозумілий для користувача формат імен і повертається користувачу.

```
#exact profile search
if "#" in bungie_name:
    display_name, display_name_code = bungie_name.split("#")
    result = search_exact_bungie_name(display_name, display_name_code)

    if result.get("Response"):
        search_results = result["Response"]

        for account in search_results:
            players.append({
                "bungie_id": account["displayName"],
                "membership_id": account["membershipId"],
                "membership_type": account["membershipType"],
                "is_exact_search": True
            })

        return jsonify({"players": players})
```

Рисунок 4.13 – Реалізація точного пошуку

```
#global search
result = search_global_bungie_name(bungie_name)

if not result.get("Response") or not result["Response"].get("searchResults"):
    return jsonify({"error": "No player found"}), 404

for user in result["Response"]["searchResults"]:
    bungie_id = f"{user.get('bungieGlobalDisplayName', 'Unknown')}#{user.get('bungieGlobalDisplayNameCode', '0000')}"
    membership_id = user.get("bungieNetMembershipId", "Unknown")
    destiny_accounts = user.get("destinyMemberships", [])

    for acc in destiny_accounts:
        players.append({
            "bungie_id": bungie_id,
            "membership_id": acc["membershipId"],
            "membership_type": acc["membershipType"],
            "is_exact_search": False
        })

return jsonify({"players": players})
```

Рисунок 4.14 – Реалізація глобального пошуку та формування відповіді

Наступним етапом буде отримання даних профілю. Прописуються вилучення параметрів запиту, які будуть отримані з обраного користувачем гравця, що показано на рис. 4.15, за допомогою request.args.get, якщо якийсь із параметрів не був отриманий, повертається помилка 400 (bad request).

```python
@app.route('/profile', methods=['GET'])
def profile():
    membership_type = request.args.get('membership_type')
    destiny_membership_id = request.args.get('destiny_membership_id')

    if not membership_type or not destiny_membership_id:
        return jsonify({"error": "Missing membership_type or destiny_membership_id parameter"}), 400
```

Рисунок 4.15 – Вилучення параметрів запиту

Відправляється запит на профіль, використовуючи параметри запиту, які представлені нижче. Якщо відповідь не отримується, повертається помилка 404 (not found).

Відправляється запит на глобальне ім’я і код користувача, використовуючи кінцеву точку «User.UserInfoCard». Вони об’єднуються у вигляд, який зазвичай бачить гравець. Далі запитуються значення отриманих доповнень, проводиться їх аналіз за допомогою функції «parse_dlc_version» (див. рис. 4.9).

Списки відкритих підкласів отримуються за допомогою функції «get_unlocked_subclasses(profile_data)» (див. рис. 4.10). Далі, для розрахунку екзотичного екіпірування в колекції, створюються змінні для підрахунку відкритого екіпірування, та її загальної кількості. Для кожного списку в «ARMOR_NODE_HASHES» вилучається список хешей, а для підрахунку відкритого екіпірування використовується функція «count_opened_collectibles» (див. рис. 4.11). Дані записуються в «armor_counts» для кожного персонажа. Приклад коду для описаних запитів продемонстровано на рис. 4.16

```python
#name and tag
user_info = profile_info.get("userInfo", {})
name_and_tag = f"{user_info.get('bungieGlobalDisplayName')}#{user_info.get('bungieGlobalDisplayNameCode')}"

#DLC
versions_owned = profile_info.get("versionsOwned", 0)
owned_dlc = parse_dlc_versions(versions_owned)

#unlocked subclasses
unlocked_subclasses = get_unlocked_subclasses(profile_data)

#exotic armor
armor_counts = {}
total_armor_unlocked = 0
total_armor_available = 0

for char_class, node_hash in ARMOR_NODE_HASHES.items():
    collectible_hashes = get_collectibles_from_node(node_hash, presentation_nodes_data)
    unlocked, _ = count_opened_collectibles(profile_data, collectible_hashes)
    available = len(collectible_hashes)

    armor_counts[char_class] = {
        "unlocked": unlocked,
        "available": available
    }

    total_armor_unlocked += unlocked
    total_armor_available += available
```

Рисунок 4.16 – Запити до Bungie API

Для отримання даних екзотичної зброї виконується алгоритм аналогічний із пошуком екіпірування.

Формується відповідь, яку сервер буде повертати, вона показана на рис. 4.17.

```python
return jsonify({
    "bungieID": name_and_tag,
    "dlc": owned_dlc,
    "unlocked subclasses": unlocked_subclasses,
    "exotic armor": {
     "total": {
       "unlocked": total_armor_unlocked,
       "available": total_armor_available
        },
     "class": armor_counts
    },
    "exotic weapons": {
     "total": {
       "unlocked": total_weapons_unlocked,
       "available": total_weapons_available
         },
     "slots": weapon_counts
     }
     })
```

Рисунок 4.17 – Формування відповіді

Тепер, коли відбувається запуск програми, створюється тестовий сервер з адресою «http://127.0.0.1:5000». Для того щоби перевірити роботу програми, у браузері треба перейти за адресою:

«http://127.0.0.1:5000/profile?membership_type=*&destiny_membership_id=*», замінивши зірочки на ідентифікатори профілю, наприклад, «1» і «4611686018450092220». Тестування показує, що результат задовільний і сервер повертає інформацію про BungieID користувача, список куплених доповнень, без тих, що втратили актуальність, загальну кількість і кількість відкритого екзотичного спорядження для кожного персонажу й в сумі, кількість відкритої екзотичної зброї для кожного типу, загальну його кількість і в сумі, а також список відритих підкласів для кожного персонажу. Код зберігається в папці проєкту файлом типу «Python source file». Повний код серверної частини вебзастосунку винесено в додаток А.

4.3 Створення файлів структури та стилів проєкту

У папці проєкту створюються папки «templates» (для структури) і «static» (для стилів і зображень). У папці «static» створюються ще дві папки: «css», «img». До «img» завантажуються всі зображення, картинки й символи, які будуть використовуватися у вебзастосунку (окрім зображення фону, воно зберігається в папці «css»).

У поточному проєкті застосунку «Visual Studio» створюється файл «index.html» де будуть зберігатися структура сторінки, а також логіка взаємодії сайту з користувачем і динамічна зміна інтерфейсу. Необхідно обрати пункт «файл»-«новий»-«файл»-«HTML page», як на рис. 4.18, зберігається файл під назвою «index» у папці «templates». По аналогічному шляху створюється файл «Style Sheet» для стилів сайту і зберігається у «static»-«css».

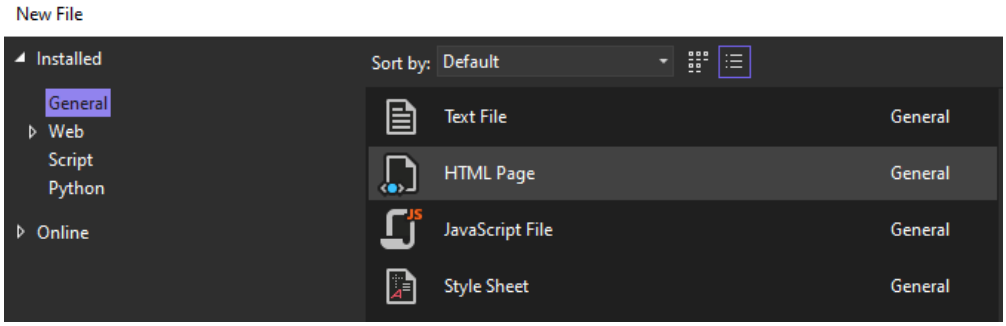

Рисунок 4.18 – Створення нових файлів у поточному проєкті

Після цього можна почати створювати структуру проєкту за допомогою HTML і CSS.

4.4 Реалізація структури фронтенду

На рис. 4.19 показано, як у файлі «index» прописується технічний заголовок сайту. Мова вмісту сторінки, кодування, назва сторінки і значок, які будуть відображатися в браузері. Також підключаються файл для стилізації сторінки і шрифти, які будуть використовуватися для стилізації.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DeDeI</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
    <link rel="icon" type="image/png" href="{{ url_for('static', filename='img/ghost white 1.png') }}">
    <link href="https://fonts.googleapis.com/css2?family=Monda:wght@400;600;700&display=swap" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css2?family=Krub:wght@400&display=swap" rel="stylesheet">
</head>
```

Рисунок 4.19 – Технічний заголовок сайту

На початку тега «body» прописується заголовок сторінки, як показано на рис. 4.20. У ньому створюється форма пошуку з текстовим полем і кнопкою, випадаючий список для гравців, який буде схований поки користувач не почне пошук профілю. Нижче створюється блок із логотипом і назвою сайту.

```
<body>
    <header class="site-header">
        <form id="search-form" class="search-section">
            <div class="search-box">
                <input type="text" id="bungie_name" placeholder="Search a username" class="search-bar" autocomplete="off">
                <button type="submit" class="search-btn">
                    <div id="search-icon-wrapper">
                        <img src="{{ url_for('static', filename='img/search.svg') }}" alt="Search" id="search-icon">
                    </div>
                </button>
            </div>
            <ul id="players-dropdown" class="hidden"></ul>
        </form>
        <div class="logo-container">
            <div class="site-title">
                <img src="{{ url_for('static', filename='img/ghost white.svg') }}" alt="logo">
                <div class="site-text">
                    <h1>DeDeI</h1>
                    <p>
                        Default Destiny Info<br>
                        <span class="author">by Defolt</span>
                    </p>
                </div>
            </div>
        </div>
    </header>
```

Рисунок 4.20 – Заголовок сторінки

На рис. 4.21 продемонстровано, як паралельно з прописуванням структури у файлі «index», у файлі «style» налаштовується стиль і положення для вже створених об’єктів сторінки.

```
.search-box {
    display: flex;
    align-items: center;
    background: rgba(64, 64, 64, 1);
    border-radius: 14px;
    padding: 5px 14px;
    width: 774px;
    height: 48px;
    box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.2);
    z-index: 1001;
}

.search-bar {
    flex: 1;
    border: none;
    background: transparent;
    color: white;
    font-size: 16px;
    padding: 10px;
    outline: none;
}
```

Рисунок 4.21 – стилізація рядку пошуку

Далі створюються блоки для відображення інформації ігрової статистики. Кожен унікальний елемент має «id» для відображення доступності предмета шляхом зміни значка з білого на кольоровий, або його підсвічування. Для екзотичного екіпірування створюються лічильники, які будуть

підставляти значення отримані від серверу. Кожен блок показаний на рис. 4.22, рис. 4.23, рис. 4.24 і рис. 4.25.

```
<div id="player-name-wrapper">
    <div id="player-name-container">
        <span id="player-name">BungieName#7777</span>
        </div>
    </div>

    <div class="dlc-block">
        <div class="dlc-header">DLC</div>
        <div class="dlc-list">

            <div class="dlc-item" id="dlc-forsaken">
                <img src="{{ url_for('static', filename='img/dlc-white/forsaken.svg') }}" alt="Forsaken">
                <span>Forsaken</span>
            </div>
```

Рисунок 4.22 – Блок імені й доповнень

```
<div class="exotics-block">
    <div class="exotics-header">Exotics</div>
    <div class="exotics-columns">
        <div class="exotics-armor">
            <div class="exotic-section">
                <img src="{{ url_for('static', filename='img/helmet.svg') }}" alt="Exotic Armor Icon" class="icon-armor">
                <span class="section-title">Exotic Armor</span>
            </div>

            <div class="armor-content">
                <div class="exotic-subsection">
                    <img src="{{ url_for('static', filename='img/hunter.svg') }}" alt="Hunter Icon" class="icon-hunter">
                    <span>Hunter:</span>
                    <span id="exotic-armor-hunter" class="exotic-value">0 / 0</span>
                </div>
```

Рисунок 4.23 – Блок екзотичного спорядження

```
<div class="weapon-content">
    <div class="exotic-subsection">
        <span>Kinetic weapons:</span>
        <span id="exotic-weapons-kinetic" class="exotic-value">0 / 0</span>
    </div>
```

Рисунок 4.24 – Блок екзотичної зброї

```
<div class="subclass-icon-wrapper">
    <img src="{{ url_for('static', filename='img/solar w.svg') }}" alt="Gunslinger" class="subclass-icon w">
    <img src="{{ url_for('static', filename='img/solar c.svg') }}" alt="Gunslinger" class="subclass-icon c">
</div>
```

Рисунок 4.25 – Блок підкласів

Для блоку з посиланнями на сторонні ресурси створюються посилання з порожніми змінними, які будуть замінюватися на інформацію завантаженого профілю, що показано на рис. 4.26

```
    <div id="more-info">
        <h2>More specific info here:</h2>
        <div id="info-links">
            <a href="https://raid.report/[platform]/[id]" target="_blank" data-replace="platform-id">Raid Report</a>
            <a href="https://dungeon.report/[platform]/[id]" target="_blank" data-replace="platform-id">Dungeon Report</a>
            <a href="https://destinytrialsreport.com/report/[type]/[id]" target="_blank" data-replace="type-id">Trials Report</a>
            <a href="https://crucible.report/report/[type]/[id]" target="_blank" data-replace="type-id">Crucible Report</a>
            <a href="https://emblem.report/p/[type]/[id]" target="_blank" data-replace="type-id">Emblem Report</a>
        </div>
    </div>
</div>
```

Рисунок 4.26 – блок із посиланнями

Далі, за допомогою JavaScript, під ці змінні будуть автоматично підставлятись дані профілю для створення робочих посилань.

4.5 Реалізація логіки фронтенду

4.5.1 Змінні

На рис. 4.27 показано, як створюється тег «script» для відтворення логіки роботи вебзастосунку. Визначається прапор для визначення процесу вибору профілю, статичне посилання на зображення для платформи й замінює «membershipType» на відповідне зображення в списку гравців, що буде отримувати користувач.

```
<script>
let isSelectingPlayer = false;

const STATIC_URL = "{{ url_for('static', filename='') }}";
function getPlatformIcon(membershipType) {
    switch (membershipType) {
        case 1: return STATIC_URL + "img/xbox.svg";
        case 2: return STATIC_URL + "img/psn.svg";
        case 3: return STATIC_URL + "img/steam.svg";
        case 6: return STATIC_URL + "img/epic1.svg";
        default: return STATIC_URL + "img/steam.svg";
    }
}
```

Рисунок 4.27 – початок тегу «script»

4.5.2 Обробка форми пошуку

Прописується подія запуску обробника у разі натискання користувачем кнопки пошуку або клавіши Enter. Звичайна відправка форми зупиняється, для того щоб весь процес проходив на одній сторінці. Отримується ім’я користувача з текстового поля і відправляється запит пошуку на сервер. Список гравців зачищається перед його активацією, ця дія необхідна для вирішення багу, коли випадаючий список відкривався з минулими результатами. Список показується, якщо прапор процесу вибору профілю не активний і кожен елемент списку перетворюється в кнопку, натискання якої активує функцію «selectPlayer», що показано на рис. 4.28

```
fetch("/search_user", {
    method: "POST",
    headers: { "Content-Type": "application/x-www-form-urlencoded" },
    body: "bungie_name=" + encodeURIComponent(bungieName)
})
    .then(response => response.json())
    .then(data => {
        const playersDropdown = document.getElementById("players-dropdown");
        playersDropdown.innerHTML = "";

        if (!isSelectingPlayer) {
            playersDropdown.style.display = "block";
        }

        if (data.error) {
            let messageItem = document.createElement("li");
            messageItem.textContent = data.error;
            messageItem.classList.add("dropdown-message");
            playersDropdown.appendChild(messageItem);
            return;
        }

        data.players.forEach(player => {
            const listItem = document.createElement("li");

            const button = document.createElement("button");
            button.className = "player-btn";
            button.innerHTML = `
```

Рисунок 4.28 – Обробник форми пошуку

4.5.3 Завантаження профілю

Після отримання даних профілю рядок пошуку зачищається, а список гравців зникає, адреса сторінки змінюється підставляючи дані профілю, а профіль завантажується через «fetchProfile».

«fetchProfile» у свою чергу виконує основні зміни на сайті. Він активує анімацію завантаження, відправляє запит на сервер з отриманими «membershipType» і «membershipId». Після отримання відповіді починається підстановка інформації, яку відправив сервер до необхідних блоків. Якщо у відповіді присутня назва доповнення, його символ отримує світіння і стає інтерактивним, реалізація цього процесу в CSS показана на рис. 4.29. Назви підкласу, які присутня у відповіді передаються до функції «updateSubclassIcons» де наявні доповнення отримують статус «unlocked», що видно на рис. 4.30, після чого CSS змінює їхній значок з білого на кольоровий і робить інтерактивним. Лічильник відкритого екзотичного спорядження підставляє дані з відповіді й аналогічним із доповненнями чином виділяє значення, якщо отримана кількість дорівнює загальній кількості в колекції.

На рис. 4.31 показано як відображається додатковий блок із посиланнями на сторонні ресурсу, де в адреси підставляються значення «membershipType» і «membershipId» у різних формах для коректної роботи посилань. Лише після всіх цих дій анімація завантаження закінчується і профіль можна вважати завантаженим на сайт.

```
.dlc-item.unlocked img {
    opacity: 0.6;
    transition: transform 0.2s ease, opacity 0.2s ease;
    transform-origin: center;
}

.dlc-item.unlocked:hover img {
    transform: scale(1.1);
    opacity: 1;
}


.dlc-item.unlocked#dlc-forsaken img {
    filter: drop-shadow(0px 0px 3px rgba(107, 62, 0, 1)) drop-shadow(0px 0px 16px rgba(167, 98, 0, 0.44));
}
```

Рисунок 4.29 – Реалізація світіння значку доповнення у CSS

```
function updateSubclassIcons(subclassData) {
    const columns = document.querySelectorAll('.subclass-column');

    columns.forEach(column => {
        const className = column.querySelector('.subclass-title').textContent.trim();
        const wrappers = column.querySelectorAll('.subclass-icon-wrapper');

        wrappers.forEach(wrapper => {
            const bwIcon = wrapper.querySelector('.w');
            if (!bwIcon) return;

            const subclassName = bwIcon.alt;

            wrapper.classList.remove("unlocked");

            if (subclassData[className] && subclassData[className].includes(subclassName)) {

                wrapper.classList.add("unlocked");
            }
        });
    });
}
```

Рисунок 4.30 – Функція оновлення підкласів

```
const membershipTypeNum = parseInt(membershipType);
const destinyMembershipIdNum = destinyMembershipId;

const platformNames = {
    1: "xb",
    2: "ps",
    3: "pc",
    6: "epic"
};

const infoLinks = document.querySelectorAll('#info-links a');
infoLinks.forEach(link => {
    let href = link.getAttribute('href-original');
    if (!href) {
        href = link.getAttribute('href');
        link.setAttribute('href-original', href);
    }

    let updatedHref = href.replace('[type]', membershipTypeNum)
        .replace('[platform]', platformNames[membershipTypeNum] || "Unknown")
        .replace('[id]', destinyMembershipIdNum);

    link.setAttribute('href', updatedHref);
```

Рисунок 4.31 – Підстановка даних для посилань

4.5.4 Оновлення адреси й рядку пошуку

На рис. 4.32 додаються функції автоматичного відправлення запиту у разі оновленні адреси сторінки шляхом ручного редактування, або функціями браузеру за допомогою «DOMContentLoaded»[11]. У будь-якому випадку зміни адреси, рядок пошуку зачищається. Додається функція закриття випадаючого списку гравців після натискання на область сайту для комфорту користувача, як показано на рис. 4.33.

```
window.addEventListener("DOMContentLoaded", function () {
    const pathParts = window.location.pathname.split('/');
    if (pathParts[1] === "profile" && pathParts.length === 4) {
        const membershipType = pathParts[2];
        const destinyMembershipId = pathParts[3];
        fetchProfile(membershipType, destinyMembershipId);
    }
});
window.addEventListener("popstate", function () {
    const pathParts = window.location.pathname.split('/');

    if (pathParts[1] === "profile" && pathParts.length === 4) {
        const membershipType = pathParts[2];
        const destinyMembershipId = pathParts[3];
        fetchProfile(membershipType, destinyMembershipId);
    } else {
        window.location.href = "/";
    }
});
```

Рисунок 4.32 – Оновлення адреси

```
    document.getElementById("bungie_name").value = "";
    const dropdown = document.getElementById("players-dropdown");
    if (dropdown) dropdown.innerHTML = "";

document.addEventListener("click", function (e) {
    const searchBox = document.querySelector(".search-box");
    const dropdown = document.getElementById("players-dropdown");

    if (!searchBox.contains(e.target) && !dropdown.contains(e.target)) {
        dropdown.style.display = "none";
    }
});
```

Рисунок 4.33 – Зачищення рядку пошуку і функція закриття списку профілів

Повний код структури й логіки, а також стилізації сайту винесено в додатки Б і В

4.6 Реалізація розгортання вебзастосунку

Створюється новий обліковий запис на PythonAnywhere.com і обирається план. Види підписок відрізняються кількістю доступних вебзастосунків з унікальною адресою із потужністю серверу, їхні особливості показані на рис. 4.34. Існує безкоштовний план з мінімальною потужністю і без можливості видавати застосункам унікальні адреси.

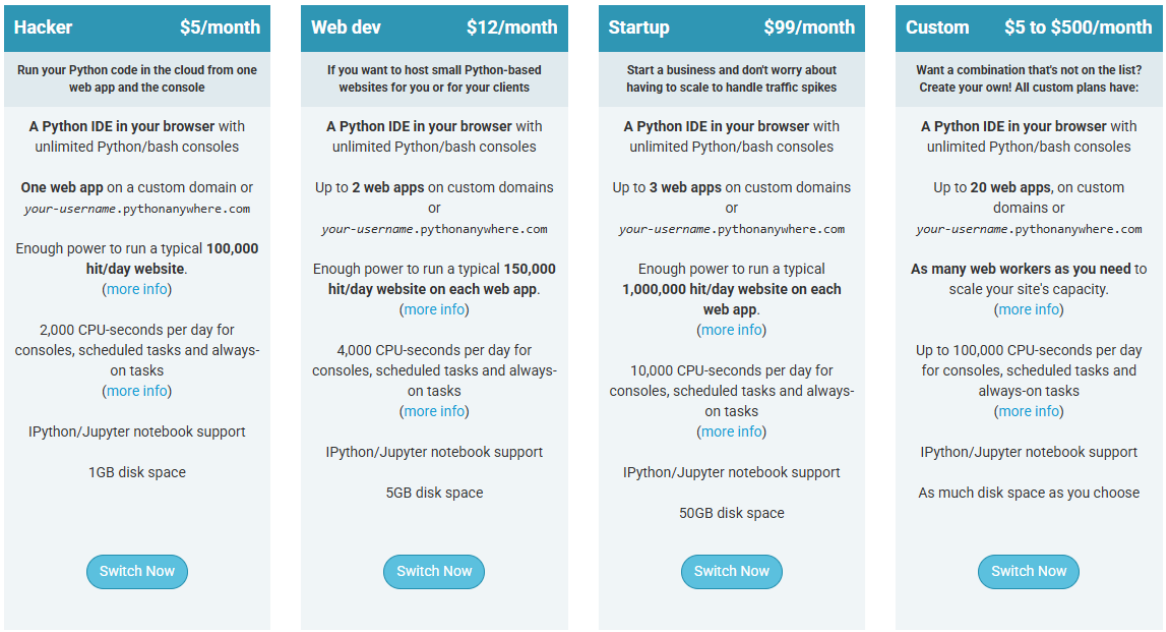

Рисунок 4.34 – Варіанти підписки PythonAnywhere

Після створення аккаунту з’являється можливість створити новий вебзастосунок, перейшовши до вкладки «web» і обравши «Add a new web app», що показано на рис. 4.35

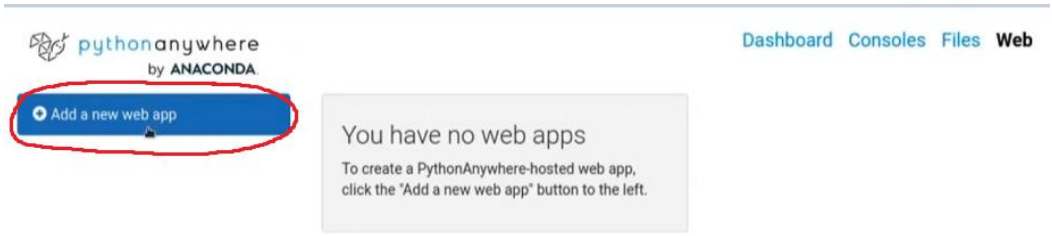

Рисунок 4.35 – Створення нового вебзастосунку

Під час створення нового вебзастосунку обирається Python-фреймворк і версія Python, а також шлях, який буде мати Python файл.

Після створення вебзастосунку необхідно перейти до вкладки «files» і змінити згенерований сервісом файл «app.py» на файл із серверною частиною вебзастосунку. Також необхідно створити аналогічні папки й завантажити всі файли, які використовуються під час роботи вебзастосунку (зображення, «index.html», «style.css», «DestinyPresentationNodeDefinition.json»). Після цього треба вказати шлях до статичних файлів (папки «static») на сторінці вебзастосунку.

Останнім кроком реалізації вебзастосунку на «PythonAnywhere» є налаштування WSGI (Web Server Gateway Interface). На вкладці «web» обирається для редактування «WSGI configuration file» і його вміст змінюється, як показано на рис. 4.36, а саме змінюється шлях до файлу «app.py» і підключається фреймворк «Flask».

/var/www/defoltr9_pythonanywhere_com_wsgi.py

```
1  import sys
2  import os
3
4  path = '/home/DefoltR9'
5  if path not in sys.path:
6      sys.path.insert(0, path)
7
8  from app import app as application
```

Рисунок 4.36 – вміст WSGI файлу

На вкладці «web» вебзастосунок перезавантажується і тепер він повністю готовий до роботи. На рисунках 4.37 і 4.38 продемонстрована домашня сторінка вебзастосунку і завантажений профіль.

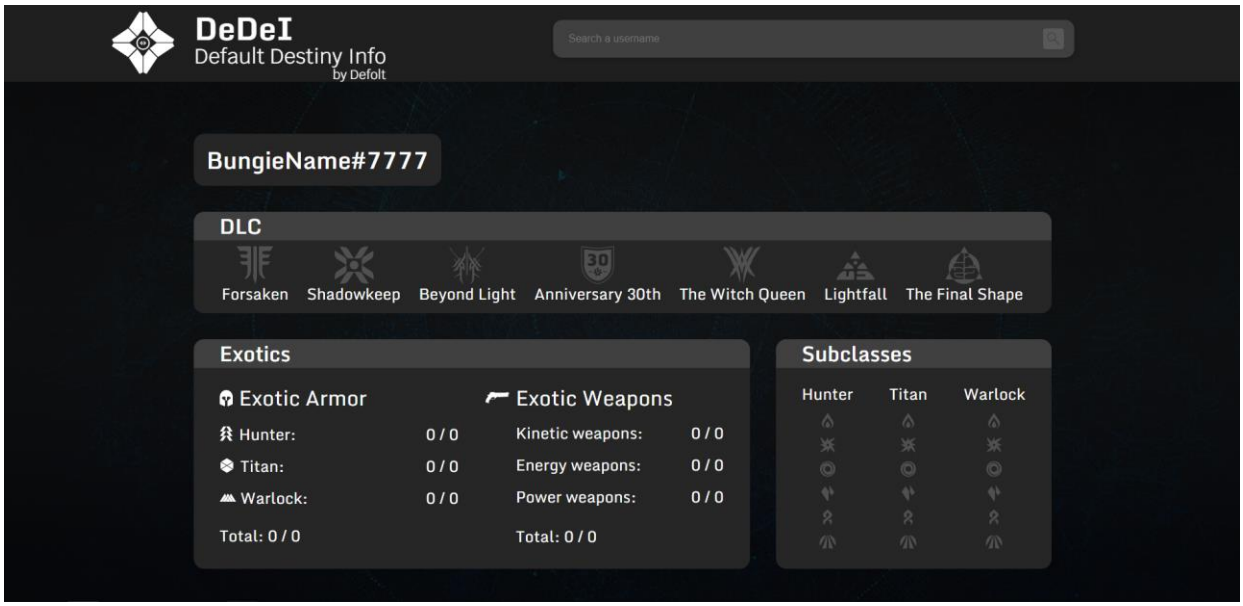

Рисунок 4.37 – Домашня сторінка вебзастосунку

Рисунок 4.38 – Завантажений профіль користувача

Таким чином було з нуля створено вебзастосунок для відображення статистики ігрових профілів, який повністю функціонує, має свою унікальну адресу в інтернеті і може використовуватися для торгівлі ігровими акаунтами, або звичайними користувачами для відстеження своєї статистики.

ВИСНОВКИ

У даній кваліфікаційній роботі було створено повноцінний вебзастосунок, який автоматизує отримання статистики ігрових профілів із відкритого API гри «Destiny 2». Розглянута проблема оптимізації процесу створення оголошень для торгівлі ігровими акаунтами.

Протягом роботи було розглянуто вебзастосунки для отримання різних даних ігрових профілів. Оцінено їхні позитивні і негативні сторони, опрацьовано матеріал з теми. Розроблено серверну частину вебзастосунку, який використовує фреймворк для отримання інформації з відкритого API. Розроблено користувацький інтерфейс і логіка взаємодії користувача і веб-застосунку. Вебдодаток надає користувачу можливість шукати профіль за його іменем, а також надає інформацію по обраному профілю, а саме: список придбаних на акаунт доповнень, кількість розблокованого екзотичного екіпірування та зброї, список розблокованих підкласів, а також посилання на сторонні ресурси для більш детального розгляду. Результати подаються в структурованому, зручному для перегляду вигляді.

Проєкт має потенціал для розширення у вигляді збільшення предметів статистки, що відстежується та порівняння профілів між собою за допомогою паралельних запитів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Time Wasted On Destiny. URL: https://wastedondestiny.com/ (дата звернення: 03.03.2025).
2. Destiny 2 Reward Checker by Mijago. URL: https://rewards.mijago.net/#/ (дата звернення: 03.03.2025)
3. Lutz, M. (2013). *Learning Python*. O'Reilly Media. 1640 p.
4. Bader D. (2017). *Python Tricks: A Buffet of Awesome Python Features*. DB Publication. 301 p.
5. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media. 312 p.
6. BungieNetPlatform. A community run wiki for the Bungie.net Platform APIs. URL: https://destinydevs.github.io/BungieNetPlatform/ (дата звернення: 04.03.2025).
7. Paracausal Science. A guide to the Destiny 2 API. URL: https://paracausal.science/guide/ (дата звернення: 03.03.2025).
8. Bungie Inc. (2025). Bungie API Documentation. Version 2.20.2. URL: https://bungie-net.github.io/multi/index.html (дата звернення: 06.03.2025).
9. Destiny Data Explorer. URL: https://data.destinysets.com/ (дата звернення: 06.03.2025).
10. Flanagan, D. (2011). *JavaScript: The Definitive Guide*. O'Reilly Media. 1093 p.
11. Marquis, M. (2016). *JavaScript for Web Designers*. A Book Apart. 135 p.
12. Duckett, J. (2011). *HTML and CSS: Design and Build Websites*. John Wiley & Sons. 490 p.
13. Richardson, L., Amundsen, M., Ruby, S. (2013). *RESTful Web APIs: Services for a Changing World*. O'Reilly Media. 406 p.
14. Ponelat, J. S., Rosenstock, L. L. (2022). *Designing APIs with Swagger and OpenAPI*. Manning Publications. 424 p.

15. Viafore, P. (2021). *Robust Python: Write Clean and Maintainable Code*. O'Reilly Media. 378 p.