Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

EXPRESSJS CLI – КОМАНДНИЙ ІНТЕРФЕЙС ГЕНЕРАТОРА ПРОСТИХ БЕКЕНД СТРУКТУР

EXPRESSJS CLI – COMMAND LINE INTERFACE OF GENERATOR FOR EASY BACKEND STRUCTURES

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІс-21
Здробилко Андрій Сергійович

(підпис)

Керівник:
к.т.н., доцент
Бортник Катерина Яківна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« 08 » червня 2024 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
« 10 » 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Здробилко Андрію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *ExpressJS CLI – командний інтерфейс генератора простих бекенд структур*

Керівник роботи *к.т.н., доцент Бортник Катерина Яківна*

затверджені наказом закладу вищої освіти від «30» грудня 2023 року № 459/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 11.06.2024р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Поняття генератора. Процедура роботи генераторів
Аналіз проблематики та проектування майбутнього рішення
Опис програми генератора та тестування на базі реального прикладу
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Використання технологій
Архітектура системи
Інтерфейс системи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Поняття генератора. Процедура роботи генераторів	*Бортник К. Я., доцент*		
Аналіз проблематики та проектування майбутнього рішення	*Бортник К. Я., доцент*		
Опис програми генератора та тестування на базі реального прикладу	*Бортник К. Я., доцент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Лавренчук С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І.,асистент*		

7. Дата видачі завдання 10.01.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Розділ 1. Поняття генератора. Процедура роботи генераторів*	до 15.02.2024 р.	Виконано
2.	*Розділ 2. Аналіз проблематики та проектування майбутнього рішення*	до 15.03.2024 р.	Виконано
3.	*Розділ 3. Опис програми генератора та тестування на базі реального прикладу*	до 04.05.2024 р.	Виконано
4.	*Висновки та пропозиції*	до 07.05.2025 р.	Виконано
5.	*Формування списку використаних джерел*	до 10.05.2024 р.	Виконано
6.	*Формування додатків*	до 15.05.2024 р.	Виконано
7.	*Оформлення ілюстративного матеріалу*	до 20.05.2024 р.	Виконано
8.	*Нормоконтроль*	до 01.06.2024 р.	Виконано
9.	*Інструментальна перевірка на академічний плагіат*	до 04.06.2024 р	Виконано
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	до 11.06.2024 р.	Виконано

Здобувач вищої освіти _____ (підпис) Здробилко А.С. (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) Бортник К.Я. (прізвище, ініціали)

АНОТАЦІЯ

Здробилко А.С. ExpressJS CLI – командний інтерфейс генератора простих бекенд структур. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2024. 75 с.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, додатків.

Перший розділ присвячено огляду предметної олбасті, тут розглядаються основні поняття про генератори, процес генерації та сфери використання. Також в цьому розділі здійснено огляд програми-аналог (NestJS).

В другому розділі здійснено вибір та обгронтування засобів розробки. Обрано засоби: NodeJS, Bash-скрипти, JavaScript (стандарт ES6). Розглянуто питання процедурної парадигми в JavaScript.

Третій розділ присвячено опису розробленої програми візуалізації роботи генератора бекенд структур: генерація бази проєкту; додавання нових компонентів до бекенду.

Ключові слова: генератор, генерація, інтерфейс командного рядка, CLI, командний рядок, bash-скрипти, JavaScript, NodeJS, ExpressJS, NodeJS CLI, інтерфейс командного рядка NodeJS, процедурна парадигма програмування, ES6, ECMAScript.

ANNOTATION

Zdrobylko A.S. ExpressJS CLI – command line interface of generator for easy backend structures. Manuscript.

Qualifying work of a bachelor of EP «Computer Engineering» specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2024. 75 p.

Qualification work consists of an introduction, three sections, conclusions, a references, three appendices.

The first section is devoted to the review of the subject matter, here the basic concepts of generators, generation process and areas of this use are considered. Also in this section, an overview of analog program (NestJS).

In the second section, the selection and substantiation of development tools is made. Selected tools: NodeJS, Bash-scripts, JavaScript (ES6 standart). The issue of procedural paradigm in JavaScript is considered.

The third section is devoted to the description of the developed program of visualization of work of generating simple backend structures: basic project generation; adding new components to exiting project.

Keywords: generator, generating process, command line interface, CLI, command line, bash-scripts, JavaScript, NodeJS, ExpressJS, NodeJS CLI, NodeJS command line interface, procedural paradigm of programming, ES6, ECMAScript.

ЗМІСТ

ВСТУП

Актуальність теми. Генератор можна представляти, як метод створення набору даних та роботи із ними. З системами генерації ми стикаємось доволі часто, особливо в епоху розвитку цифрових технологій, розвитку ігрової індустрії, а також в простих для людей речах. Більшість задач можна спроєктувати із використанням генератора (генерація унікального ID для гравця комп'ютерної гри, генерація чисел для вибора переможця конкурсу, створення певного набору даних чи інформації на базі результатів опитування тощо). Генератори використовують скрізь, де є можливість отримати рандомний набір даних або де є можливість сформувати певний набір інформації згідно вхідних даних. В теперішній час генератори використовуються навіть і у сфері розробки, де потрібно сформувати початкові варіації коду майбутнього програмного продукту.

Метою роботи є створення та запропонування рішення для розробників бекенд застосунків, на базі бібліотеки ExpressJS генератора проєктів, а також системи для пришвидшення та покращення процесів розробки, котрий допоже розробникам в швидкій реалізації потрібних для них речей без витрачення для цього великого обсягу часу.

Об'єкт дослідження – методології створення генераторів та приклади їхньої реалізації

Предмет дослідження – генератор, створений із використанням конструкторів та шаблонів для формування початкових компонентів проєкту.

Завдання, які необхідно виконати:

– Реалізувати програмний продукт, котрий буде формувати початковий вихідний код проєкту на базі бібліотеки ExpressJS із використанням готових для цього шаблонів проєктування.

– Розробити модульний загальний інтерфейс проекту.

– Запропонувати новітнє та просте рішення для розробників бекенду що у своїй роботі використовують бібліотеку NodeJS.

РОЗДІЛ 1
ПОНЯТТЯ ГЕНЕРАТОРА. ПРОЦЕДУРА РОБОТИ ГЕНЕРАТОРІВ

1.1 Поняття генератора. Сфера його застосування

Генератори у сфері інформаційних технологій (ІТ) є ключовими компонентами багатьох систем та додатків, що потребують створення випадкових чи псевдовипадкових послідовностей чисел. Ці послідовності можуть використовуватися у численних галузях, включаючи криптографію, симуляції, ігрову індустрію, статистику та багато іншого.

Генератор – це пристрій або алгоритм, що створює послідовність чисел або символів, яка вважається випадковою або псевдовипадковою [1]. Розрізняють два основні типи генераторів у ІТ: апаратні та програмні.

- Апаратні генератори використовують фізичні явища для створення випадкових чисел. Наприклад, вони можуть використовувати шум електронних компонентів або радіоактивний розпад для генерування істинно випадкових чисел. Перевагою апаратних генераторів є висока ступінь випадковості, але їхня реалізація може бути дорогою та складною.
- Програмні генератори, також відомі як генератори псевдовипадкових чисел (ГПВЧ), використовують алгоритми для створення чисел, що здаються випадковими. Такі генератори починають роботу з початкового значення (seed) і потім використовують математичні формули для генерування послідовності чисел. Хоча ці числа не є істинно випадковими, вони можуть бути достатньо випадковими для багатьох застосувань.

Генератори в інформатиці існують в різних варіаціях залежно від поставленої задачі, проте є ряд основних видів генератора, котрі найчастіше використовуються в сфері ІТ, а саме:

- Генератори істинно випадкових чисел (ГІВЧ).
- Генератори псевдовипадкових чисел (ГПВЧ).
- Криптографічні генератори псевдовипадкових чисел (CSPRNG).

Дані основні види генераторів має доволі глибокий спектр застосування не тільки для розробки, але й в повсякденному житті вцілому. Наприклад, генератори випадкових і псевдовипадкових чисел знаходять застосування в багатьох галузях, таких як:

– Наукове моделювання, особливо у моделюванні складних систем, таких як кліматичні моделі або моделі поведінки ринку, випадкові числа використовуються для імітації випадкових подій та варіацій.

– Криптографія та безпека даних. Генератори випадкових чисел є критично важливими для створення криптографічних ключів, випадкових сольових значень для хешування та інших захищених параметрів.

– Ігрова індустрія. Генератори використовуються для створення випадкових подій у відеоіграх, таких як розподіл луту або результати битв. Вони забезпечують непередбачуваність та підвищують інтерес гравців.

– Статистика та аналітика. Даний вид генераторів використовуються для створення випадкових вибірок з великих наборів даних. Це дозволяє дослідникам отримувати репрезентативні зразки для аналізу.

– Телекомунікації. У телекомунікаційних системах генератори випадкових чисел використовуються для розподілу частот, уникнення колізій та шифрування даних.

У фінансових технологіях ГВЧ застосовуються для моделювання та управління ризиками. Наприклад, при моделюванні фінансових ринків для прогнозування цінових коливань використовуються складні математичні моделі, які базуються на випадкових числах. Це дозволяє створювати сценарії, які допомагають інвесторам та фінансовим установам приймати обґрунтовані рішення.

Генератори також є важливими в індустрії розваг, зокрема в онлайн-казино та ігрових автоматах. Випадковість є ключовим елементом цих ігор, що забезпечує чесність та непередбачуваність результатів. ГВЧ використовуються для визначення результатів спінів, роздачі карт та інших ігрових подій, створюючи відчуття випадковості та інтриги для гравців.

В побуті генератори випадкових чисел можуть бути використані в багатьох різних пристроях і додатках. Наприклад, у сучасних пральних машинах генератори випадкових чисел можуть використовуватися для вибору різних режимів прання та оптимізації процесу відповідно до конкретних умов. У мобільних додатках для здоров'я та фітнесу генератори випадкових чисел можуть використовуватися для створення індивідуальних планів тренувань, які враховують випадковість для запобігання звикання організму до одноманітних вправ.

Криптографічні генератори псевдовипадкових чисел зазвичай використовують у місцях, де важливим аспектом є високий рівень захисту даних. Дані такі генератори досить поширенні в тематиці шифрування даних, простими прикладами цього є метод шифрування SHA-256 та MD5, а також на сьогоднішній час досить поширено використовуються в браузерах, де система пропонує варіації надійного пароля, задля безпеки даних користувача. Даний вид генераторів дозволяє реалізовувати високий рівень безпеки різноманітної інфомрації, що унеможливлює шанс для несанкціонованого доступу до даних.

1.2 Історія розвитку генераторів

Розвиток генераторів псевдовипадкових чисел бере свій початок з моменту розвитку математики, а також з моменту розвитку обчислювальної техніки вцілому, котрі спрямовані на створення числових послідовностей, які відображаться, як випадкові.

Перші спроби створення генераторів випадкових чисел базувалися на фізичних процесах. Один із найвідоміших методів – використання механічних пристроїв, таких як рулетки або кості. Однак ці методи були незручними для широкого використання в наукових дослідженнях.

У 1946 році Джон фон Нейман запропонував середньоквадратичний метод (middle-square method) як перший числовий підхід до генерації випадкових чисел. Метод полягав у піднесенні до квадрату початкового числа та виборі середніх цифр результату як нового числа для наступної ітерації [2]. Незважаючи на

інноваційність, метод мав серйозні недоліки, включаючи короткі цикли та передбачуваність.

У 1950-х роках Дональд Кнут і його колеги розробили лінійний конгруентний генератор (ЛКГ), який став значним кроком вперед у генерації псевдовипадкових чисел. ЛКГ став основою для багатьох сучасних ГПВЧ завдяки своїй простоті та ефективності. Він використовувався в багатьох програмних реалізаціях, включаючи стандартні бібліотеки мов програмування.

У 1997 році Мацумото і Нішимура розробили алгоритм Мерсенна-Твістера, який значно покращив характеристики генерації псевдовипадкових чисел. Назва генератора походить від простого числа Мерсенна, яке використовується в алгоритмі. Головні переваги методу Мерсенна-Твістера включають:

- Довгий період (2^19937-1), що забезпечує відсутність повторень у послідовності чисел на практиці.
- Висока швидкість генерації чисел.
- Відсутність значних статистичних відхилень, що робить його ідеальним для наукових розрахунків та симуляцій.

Метод Мерсенна-Твістера став стандартом у багатьох сучасних програмних середовищах завдяки своїй ефективності та надійності.

З розвитком криптографії виникла потреба в генераторах, які можуть забезпечити високий рівень безпеки. Звичайні ГПВЧ не могли гарантувати непередбачуваність, необхідну для криптографічних застосувань. Це привело до створення криптографічно стійких генераторів псевдовипадкових чисел (CSPRNG).

Одним із перших таких генераторів став алгоритм Yarrow, розроблений Брюсом Шнайером та Джоном Келссом у 1999 році. Yarrow використовує комбінацію хеш-функцій та криптографічних примітивів для створення випадкових чисел, стійких до криптографічного аналізу.

У 2008 році була представлена система Fortuna, яка стала подальшим розвитком ідей, закладених у Yarrow. Fortuna використовує кілька пулів ентропії для зменшення ризику вичерпання випадковості та забезпечує високий рівень безпеки для криптографічних застосувань.

Сучасні ГПВЧ поєднують різні математичні методи та алгоритми для досягнення максимальної випадковості та ефективності. До таких підходів належать:

- Комбіновані методи - використання кількох генераторів одночасно для покращення якості випадковості.
- Квантові генератори - використання квантових ефектів для створення істинно випадкових чисел. Ці генератори забезпечують непередбачуваність на рівні фундаментальних фізичних процесів.
- Гібридні алгоритми - поєднання традиційних математичних методів із сучасними криптографічними техніками для досягнення високої надійності та швидкості генерації чисел.

1.3 Процедура роботи генераторів

Процедура роботи генераторів випадкових чисел (ГВЧ) та генераторів псевдовипадкових чисел (ГПВЧ) базується на різних математичних та фізичних методах [3]. Розглянемо основні принципи їх роботи, деталі алгоритмів, що використовуються, а також специфіку апаратних генераторів.

Генератори випадкових чисел поділяються на дві основні категорії: апаратні та програмні. Апаратні генератори використовують фізичні процеси для створення істинно випадкових чисел, тоді як програмні генератори, або ГПВЧ, застосовують математичні алгоритми для створення чисел, які виглядають випадковими [4].

Апаратні генератори випадкових чисел (АГВЧ) генерують випадкові числа на основі фізичних явищ, таких як термічний шум, фотонні ефекти або квантові флуктуації. Наприклад, генератор може використовувати шум у резисторі або напівпровіднику, який завжди є випадковим і непередбачуваним.

Генератори псевдовипадкових чисел (ГПВЧ) створюють послідовності чисел, які є детермінованими, але виглядають випадковими. Вони починають роботу з певного початкового значення (seed), яке визначає послідовність чисел. Зміна початкового значення дозволяє створювати різні послідовності.

Існує кілька основних алгоритмів, які використовуються для генерації псевдовипадкових чисел:

– Лінійний конгруентний генератор (ЛКГ). Це один з найстаріших і найпростіших алгоритмів. Розрахунок ЛКГ визначається за наступною формулою, котру ми розглядали раніше (1.1):

$$X_{n+1} = (aX_n + c)\ mod\ m, \tag{1.1}$$

де X – послідовність чисел, a, c і m – константи. ЛКГ є досить швидким, але має обмежений період і може мати проблеми з циклічністю та кореляцією між числами.

– Метод Мерсенна-Твістера. Цей алгоритм був розроблений у 1997 році і є одним з найпопулярніших завдяки своєму довгому періоду (2^19937-1) та високій швидкості генерації чисел. Він використовує рекурентні формули для створення чисел, які мають хороші статистичні властивості.

– Алгоритм Лаггед-Фібоначчі. Цей метод базується на модифікованих послідовностях Фібоначчі і використовує формулу (1.2):

$$X_n = \left(X_{n-j}\ +\ X_{n-k}\right) mod\ m, \tag{1.2}$$

Де X – послідовність чисел, j і k – індекси попередніх чисел у послідовності. Цей метод забезпечує більшу ентропію, але потребує більше пам’яті для збереження попередніх значень.

Криптографічні генератори псевдовипадкових чисел (CSPRNG) призначені для забезпечення високої безпеки і стійкості до передбачення. Основні методи включають:

– Алгоритм Yarrow. Розроблений Брюсом Шнайером, цей алгоритм використовує комбінацію криптографічних хеш-функцій та інших криптографічних примітивів для створення надійних випадкових чисел. Він поділяє випадковість на кілька пулів, щоб запобігти вичерпанню ентропії.

– Fortuna. Покращена версія Yarrow, Fortuna використовує множинні пули ентропії і має кращу архітектуру для збирання та обробки випадкових подій. Вона також використовує криптографічні примітиви для забезпечення високого рівня безпеки.

– AES-CTR DRBG - один з алгоритмів, рекомендованих NIST, який використовує симетричний шифр AES у режимі лічильника (CTR). Він забезпечує високу продуктивність і стійкість до атак.

Апаратні генератори використовують фізичні явища для створення істинно випадкових чисел. Основні принципи їх роботи включають:

– Термічний шум. Використання електронного шуму в резисторах або інших компонентах для генерації випадкових чисел. Це один з найпоширеніших методів, оскільки шум завжди є випадковим і непередбачуваним.

– Фотонні ефекти. Використання квантових флуктуацій або фотонних подій для створення випадкових чисел. Наприклад, генератори можуть використовувати ефект Гейзенберга, щоб отримати істинно випадкові значення. Квантові генератори: Використання квантових процесів, таких як суперпозиція і заплутаність, для генерації чисел. Ці методи забезпечують максимальну ентропію і є абсолютно непередбачуваними.

1.4 Генератор як інструмент в сфері розробки

На початку розвитку інтернету та інтернет ресурсів, почали з’являтись потреби у розробці прикладних програм, котрі виконують роботу із базою даних, такі як зберігання, видалення, оновлення, створення, а також роботу з отриманими даними, наприклад, формування кредиту для користувача онлайн банку, чи обрахунок статей на новинному порталі. Саме такі програми отримали свою назву, як бекенд застосунки, пізніше програмісти почали називати дані програми просто – бекендом. Проте популярність та потрібність у такому роді програм була не лише в сфері розробки сайтів, а також у розробці мобільних застосунків, а також й у сфері розробки ігор. Саме така потрібність дала можливість

реалізувати різні варіанти бекенду починаючи від простої програми для збереження новин та повідомлень з форумів, закінчуючи логікою роботи банків, та різноманітних онлайн ігор. Проте на сьогоднішній день бекенд досі має вагоме значення в сфері веб-розробки, адже саме в цій сфері бекенд застосунки, є елементом роботи із онлайн даними та їх поширенням із іншими користувачами чи програмами(напр. Публічні API для різних застосунків). Саме через роботу із онлайн даними та роботою через мережу, а також сферою звідки з'явилось таке поняття бекенд вважають невідокремленною частиною веб-розробки незважаючи на те, що бекенд використовують й в інших сферах розробки, окрім розробки веб-застосунків.

Створення бекенду є клопіткою, а також роботою із великою відповідальністю. На відміну від створення візуалу сайтів(далі Фронтенд), де потрібно лише відобразити вигляд сайту, бекенд вимагає від себе знання в сфері безпеки даних, роботи із великим набором даних та в поняттях оптимізації прикладного програмного забезпечення, котре дозволяє реалізовувати відповідь до фронтенду за лічені мілісекунди часу. Саме ці аспекти і надають важливості поняттю бекенду, адже через дану програму користувач напряму зв'язується із базою даних та працює з отриманими даними.

Проте на момент розвитку даного різновиду розробки прикладних застосунків, почала з'являтися проблематика, щодо доступу до персональних даних користувача, адже паралельно із розвитком бекенду в цілому, почали з'являтися різні засоби отримання даних без згоди користувача, такі як SQL Injection, різноманітні механізми отримати великий обсяг даних несанкціонованим шляхом, саме через це почали повставати питання, щодо захисту особистих даних користувача та різних засобів обмеження доступу до тієї чи іншої інформації. Так з'явилися різноманітні методи шифрування даних та механізмів передавання даних, такі як RPC (віддалений виклик процедур) чи REST 2.0 (захищений варіант передачі даних через посилання та різноманітні конструкції виводу інформації).

На сьогоднішній день було сформовано список правил, котрі кожен розробник повинен дотримуватись, щоби унеможливити несанкціоновані доступи до даних, та захисту від їх отримання навіть доволі складними шляхами, такі як:

– Шифрування паролей та інших особистих даних користувача шляхом SHA-256 та кращими методами шифрування.

– Захист від SQL Injection (методологія SQL Injection полягає у передачі несанкціонованого набору команд через текстовий рядок, використовуючи звичайні символи, котрі для програми означатимуть кінцем текстового рядка, та початком наступної частини команд, що робить ваш бекенд та базу даних вразливи та легкими для зловмисників).

– Встановлення правил доступу до певного набору даних, для розділення і визначення, якій групі людей надати дозвіл до даної інформації, а котрій заборони до неї доступ.

Крім захисту інформації від зловмисників, почало повставати питання щодо оптимізації бекенд застосунків, адже дана проблема тягне за собою низку наслідків, котрі розробники почали помічати з часом. Наприклад, бекенд тепер не лише зберігає новини, а й також виконує певну складну логіку, для обрахунку та реалізації певних дії, котрі потрібні для певного функціоналу в інтерфейсі користувача чи просто велика кількість користувачів(напр. 1 000 000 користувачів) звертається до одного і того самого набору даних одночасно, тоді проблема тепер не у захисті, а у часі відповіді бекенду і в його оптимізації.

Для вирішення даного питання розробники почали знаходити різноманітні засоби задля оптимізації бекенд застосунків, починаючи від розділення на набір викликів від користувачів шляхом правилами обмеження на кількість отриманні запитів, закінчуючи розмежування кожного набору дії на окремі системи, після чого дана технологія отримала свою назву Мікросервісна архітектура бекенду.

Саме ці аспекти в сьогодення дозволили нам побачити роботу бекенду в якісному та надійному форматію. Простим прикладом цього є сервіс Instagram, бекенд якого здатний обробляти до мільярду запитів одночасно з різних куточків світу та захищати їх від несанкціонованого доступу до даних користувачів.

Під час розвитку бекенду в галузі програмування було створено різноманітну кількість технологій, котрі могли вирішувати різноманітну варіацію проблем та рішень. Одним із досить популярних обговореннь проблеми стала проблематика

генерації проекту, а також його компонентів з використанням поширених технологій, а саме програмний модуль NodeJS та бібліотека ExpressJS. Саме дані технології викликають у програмістів безліч питань з точки зору спрощення створення проекту майбутнього бекенду, адже його конкурент фреймворк NestJS з моменту свого створення має внутрішній інтерфейс командного рядка, котрий дозволяє генерувати проект бекенду за лічені секунди та спрощує роботу розробникам в рази.

Були різноматні спроби вирішення даної проблематики, проте дані рішення базувались виключно на вузькій спеціалізації чи відносилась виключно на мале коло користувачів, котрим воно з легкістю вирішувало прості проблематики. Так до прикладу розробники бібліотеки ExpressJS створили першу версію власного генератора бекенд проекту, проте його спеціалізація базувалась виключно на генерації бази проєкту. Але не зважаючи на спеціалізацію, генератору не вдавалось реалізовувати повністю свою роботу через низку багів та проблем, через що дану ідею розробники віднесли до закриття та зупинили усі подальші роботи над ним.

Спираючись на даний аналіз інформації про генератор, можемо зрозуміти, що генератори є досить поширеним елементои в повсякденному житті кожної людини. Їхній спектр використання розширюється доволі стрімко, а також моделі генерації на сьогоднішній день показують доволі великий показник в якості роботи. Якщо розглядати ситуацію, що стосується сучасної проблематики в генерації бекенд структур з використанням бібліотеки ExpressJS, можемо дійти висновку, що дана проблема доволі актуальна і потребує термінового рішення, задля забезпечення розробникам комфортної розробки бізнес логіки застосунків.

РОЗДІЛ 2
АНАЛІЗ ПРОБЛЕМАТИКИ ТА ПРОЕКТУВАННЯ МАЙБУТНЬОГО РІШЕННЯ

2.1 Аналіз проблематики

На сьогоднішній день, як було описано в першому розділі, постає важливе питання для розробників, котрі використовують в своїй постійній роботі бібліотеку ExpressJS – як саме спростити процес розробки та які є строгий стандарт розробки проекту з використанням даної бібліотеки. Станом на 2024 рік, якщо проводити дослідження ресурси коду розробників на платформі GitHub [5] можемо помітити, що кожен розробник формує свій власний стандрат для формування каталогів та папок, та чіткої структури проекту в своїх проектах (рисунки 2.1-2.2).

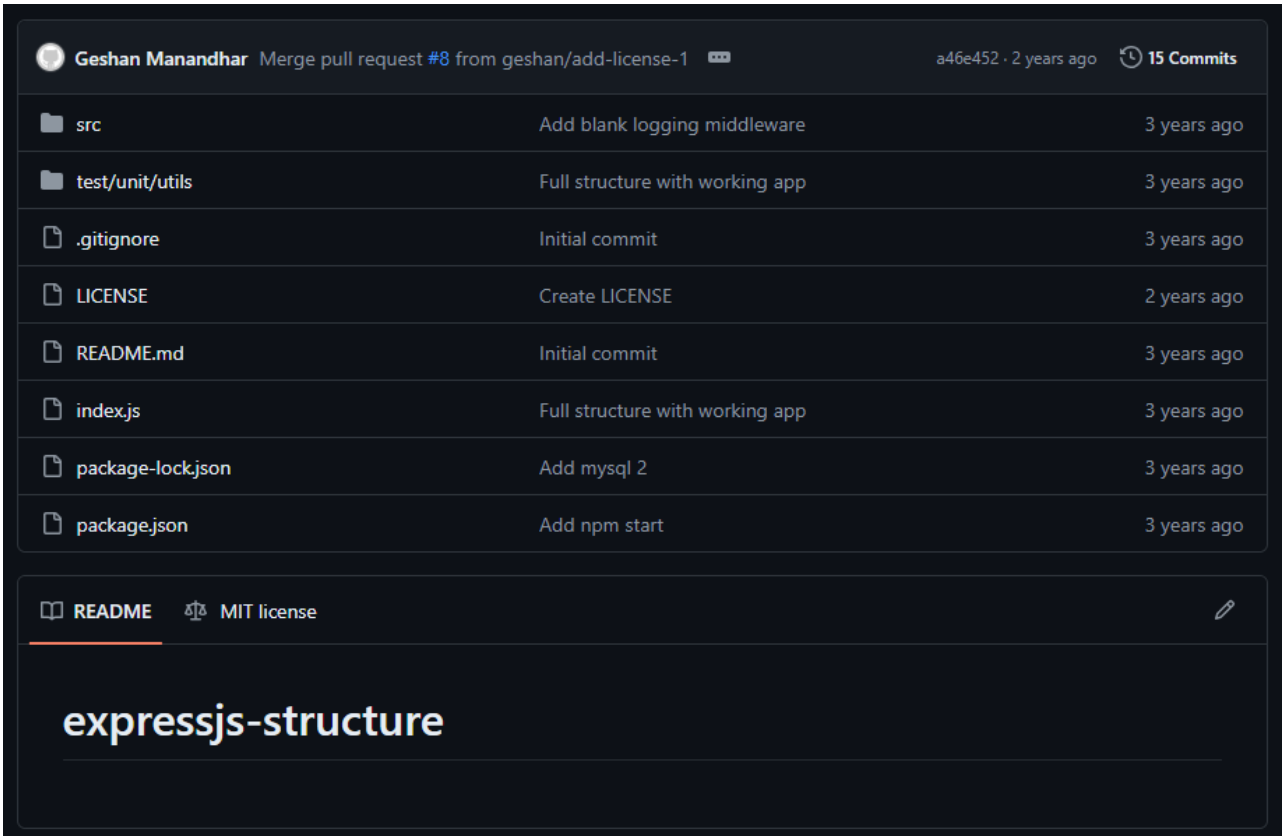

Рисунок 2.1 – Приклад структури проекту одного із розробників на платформі GitHub [6]

Рисунок 2.2 – Приклад структури проекту одного із розробників на платформі GitHub [7]

Окрім відсутності строгої структуризації проекту, розробники часто зустрічають проблематику повязану із швидкою розробкою простих бекендів з використанням даного інструменту, адже бувають випадки, коли потрібно швидко та в декілька кліків мати готову структуру проекту у яку внесуть маленькі корективи та запустять для постійної роботи із нею. Саме дані проблеми часто спонукають розробників створити генератор проектів на базі даної технології, що дозволить не лише формувати кодову частинку бекенду правильно, а також створювати його без проблем та за лічені хвилини.

Саме дані аспекти поставленої проблематики ми будемо вирішувати та будемо розглядати варіанти вирішення даної ситуації на базі реального проекту.

2.2 Підбір ресурсів та технологій для реалізації проекту

Головна ціль генератора бекенд застосунку полягає у тому, що потрібно створювати набір файлів коду, які є початковою версією бекенду, при цьому користувач повинен ситуативно розуміти, що саме потрібно зробити, та яким чином даний застосунок створює код майбутнього бекенду.

Так як, потрібно розробити генератор бекенд застосунку, нам потрібно розглянути кілька варіантів реалізації даного рішення проблематики. В першу чергу розглянемо варіанти вигляд застосунку. У нас є два варіанти реалізації інтерфейсу: інтерфейс командного рядка та інтерфейс прикладного застосунку з користувацьким інтерфейсом. Якщо розглядати перший варіант, то ми не витрачаємо час на моделювання та створення зовнішнього користувацького інтерфейсу, та робимо більше акцент на написання логіки генератора, проте для користувача з точки зору дизайну це вважається поганим користувацьким досвідом(він же поганим UX – ом). З точки зору доступності та актуальності у використанні розробниками саме інтерфейсу командного рядка, ми можемо зробити висновок, щодо того, що саме даний варіант дає нам перевагу у тому, що кожен користувач зможе без проблем встановити та запустити генератор, як пакет (англ. package – набір бібліотек та функцій об'єднаних в загальний набір коду, котрий вирішує певний набір проблем) і тим самим ми робимо наш генератор кросплатформенним прикладним застосунком. Простим прикладом генератора в вигляді інтерфейсу командного рядка є NestJS CLI (рисунок 2.3).

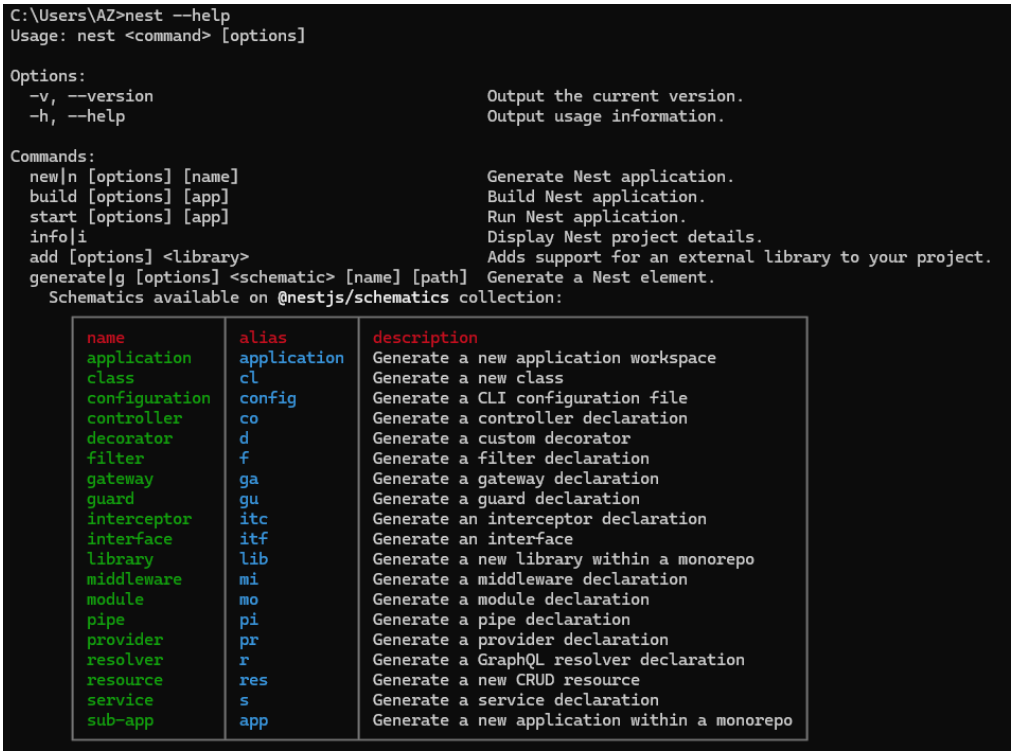

Рисунок 2.3 – Загальний вигляд генератора бекенд структур NestJS CLI у форматі інтрефейсу командного рядка

NestJS CLI – генератор бекенд стурктур для фреймворку NestJS (фреймворк створенний для мови програмування JavaScript), призначений для створення бекенду структур на базі високо-рівневих інструментів. Даний генератор має у собі простий вигляд інтерфейсу та складну логіку, котра дозволяє роботи доволі якісні файли коду за лічені хвилини. NestJS CLI окрім генерації містить у собі, ще великий спектр команд, котрі дозволяють організовувати різноматні операції із бекендом, а також надають розробникам великий набір інструментів для взаємодії із кодовою базою проекта написаного з допомогою фреймворку NestJS.

В другому варіанті ми витрачаємо ресурси на моделювання та створення інтерфейсу користувача, а також робимо менший акцент на написання логіки генератора. З точки доступності у використанні та актуальності серед розробників ми маємо проблеми з питань зібрання коду у застосунок під кожну операційну систему, що порушує цілісну доступність, адже кожна операційна система має свої вимоги щодо побудови застосунку з користувацьким інтерфейсом. Саме даний аспект не дає нам можливість реалізувати наш генератор кросплатформенним застосунком. Проте є й альтернативніший варіант створення користувацького інтерфейсу – веб-застосунок. Веб-застосунки дозволяють нам обійти вирішення проблеми із кросплатформерністю, проте вимагає від користувача постійний доступ до мережі інтернет, що інколи не є доволі гарною ідеєю. Одним із простих прикладів використання веб-застосунку для генерації бекенд структур є Spring Initializr (рисунок 2.4). Spring Initializr – генератор бекенд стурктур для фреймворку Spring та Spring Boot (фреймворки створенні для мови програмування Java), котрі використовуються для написання бекенду для середнього та великого бізнесу (даний напрямок розробки називають ще Enterprise розробкою). Дана веб-сторінка включає у себе набір інструментів для початкового налаштування проекта, а також надає можливість розробникам додати потрібні бібліотеки та обрати із якою саме версією мови програмування та фреймворку вони будуть працювати. Це дає можливість власноруч обрати під ті чи інші задачі налаштування та згенерувати початкову кодову базу проєкта. Такий варіант зовнішнього вигляду надає користувачеві великий спектр умов для налаштування фундаменту проекта, що

неабияк дозволяє адаптовуватись під різні умови поставлених перед розробником задач.

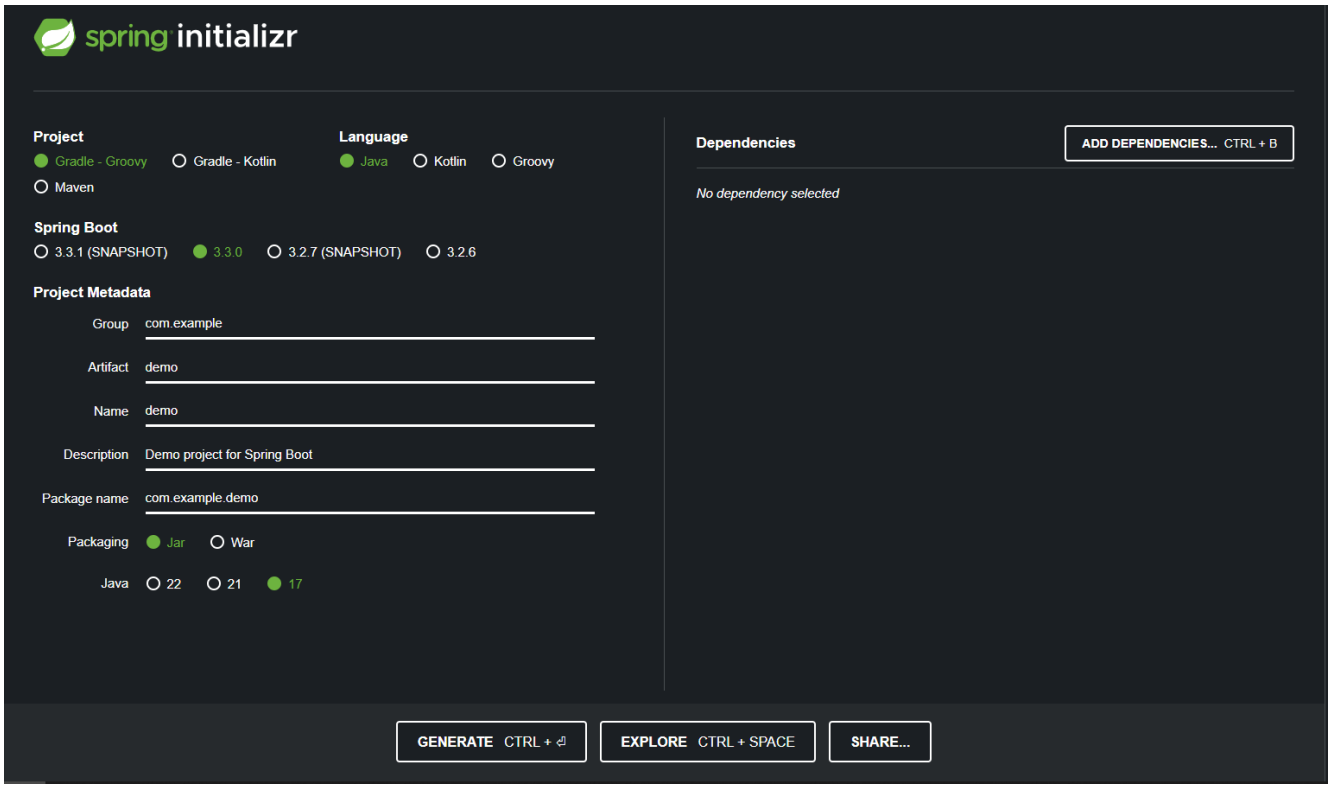

Рисунок 2.4 – Загальний вигляд генератора бекенд структур Spring Initializr у форматі користувацького інтерфейсу веб-застосунка [8]

Наступний момент, котрий будемо оглядати та аналізувати – які саме ресурси буде використано для побудови генератора? В першу чергу ми повинні розуміти, що задля того аби ми мали змогу швидко та якісно генерувати бекенд, ми повинні мати загальні заготовленні шаблони коду, котрі дозволяють нам створювати швидко і лаконічно потрібні файли коду для бекенду. Крім того, ми повинні використати потрібно використати бібліотеки та пакети, котрі дозволяють нам розробити логіку генерації і робити певні дії, щодо активації команд та ініціалізації проекту без участі розробника(як це було зроблено у фреймворку NestJS).

Спираючись на дані порівняння двох варіантів інтерфейсу та огляду на потрібність у певних бібліотеках та пакетах ми отримуємо висновок, що надійним та оптимальним варіантом залишається створення інтерфейсу командного рядка з огляду на доступність, актуальність та кросплатформеність [9], а також

використання певних пакетів, котрі дозволяють нам зробити наш інтерфейс привабливим та швидким у роботі, а саме:

- Logger.
- Commander.
- Chalk.
- Lodash.
- Dirname-Filename-ESM.

Бібліотека Logger призначена для формування логів програми (набір інформації про дії в застосунку), що дає певне розуміння про хід дії программи, а також про помилки, які виникли під час виконання тієї чи іншої задачі.

Бібліотека Commander буде використовуватись, для формування набору команд, якими буде користуватись розробник для виконання тих чи інших операцій, наприклад використовуючи дану бібліотеку ми із легкістю можемо отримати інструкцію по використанню команд викликавши командою “--help” [10].

Бібліотека Chalk [11] та Lodash призначені для стилізації текстової частини інтерфейсу командного рядка. Зазвичай дану бібліотеку використовують для підкреслення тексту в тих місцях де відбулась помилка під час виконання певних дій, або під час некоретного отримання вхідних даних від користувача.

У кожній операційній системі (Windows, Unix, Linux чи MacOS) є власне дерево каталогів та власна файлова система і це для розробників зазвичай є складним завданням, коли мова йде про використання файлової системи відразу в усіх доступних операційних системах. Для таких варіантів було організовано універсальні бібліотеки, котрі дозволяють працювати із файловою системою не зважаючи на властивості операційної системи. Саме для таких випадків простим прикладом є бібліотека Dirname-Filename-ESM. Бібліотека Dirname-Filename-ESM використовується для організації крос-платформерної роботи із файловою системою. Ця бібліотека є важливим елементор для генератора, адже для створення проєкту потрібно вказувати абсолютний шлях до місця куди потрібно згенерувати кодову базу, а також вказувати шляхи для копіювання шаблонів для кожної операційної системи, де буде наш генератор запускатись.

Отже, для того, щоби реалізувати генерування простого бекенд застосунку нам потрібно врахувати низку моментів, а саме:

– Генератор повинен бути у вигляді інтерфейсу командного рядка із зрозумілим та інтерактивним виглядом.

– Підготовка шаблонів коду.

– Моделювання логіки зчитування команд та генерації бекенд застосунку.

Підсумувавши даний аналіз, ми можемо дійти висновку, що для створення прикладного генератора в вигляді командного рядка ми будемо використовувати наступні технології та рішення: програмний модуль NodeJS, спеціалізовані скрипти, котрі дозволяють нам організувати роботу даного застосунку в ролі CLI (інтерфейсу командного рядка). Дана програма буде генерувати код простого бекенду на базі бібліотеки ExpressJS. Дана бібліотека слугує для написання бекенду в динамічній формі, що робить бекенд зручним та швидким у роботі [12].

РОЗДІЛ 3
ОПИС ПРОГРАМИ ГЕНЕРАТОРА ТА ТЕСТУВАННЯ НА БАЗІ РЕАЛЬНОГО ПРИКЛАДУ

3.1 Опис моделі роботи проекту

Головна концепція полягає у створенні файлів коду на базі готових шаблонів враховуючи вибір користувача(напр. чи потребує розробник рушій для відображення HTML-сторінок, якою мовою програмування буде описана логіка бекенду і т.д). Дані аспекти і формують певні варіації вибору генерування проекту, та додає усі потрібні елементи до проекти, котрі необхідні для роботи бекенду. Якщо додатково розглядати процес генерації компонентів до існуючого вже проекту, головною ціллю постає тепер те, шо саме потребує розробник в створенні, буцім-то це Controller(набір процедур, що відповідають за роботу основної логіки бекенду) чи Routes(набір процедур, що відповідають за формування посилань на запити та їх типи). На базі даного вибору потрібно сформувати чітку логіку, оскільки в залежності від вибора користувача, який компонент йому потрібно згенерувати, саме такий файл потрібно сформувати та внести в іморт залежностей (підключити до іншого файлу коду, для доступу до потрібних функцій та вихідних даних).

Окрім формування логіки потрібно врахувати до уваги і загальний вигляд інтерфейсу командного рядка. Для стилізації та простоти в взаємодії користувача із інтерфейсом командного рядка ми використаємо такі бібліотеки, як: Chalk, Lodash. Дані бібліотеки дозволять нам зробити певну стилістику у нашому інтерфейсі командного рядка та зробить його цікавішим(рисунок 3.4). Проте для початку ми повинні налаштувати наш проект таким чином, щоби він міг працювати, як інтрефейс командного рядка, та зуміг запускатись на будь якій операційній системі. Для цього ми повинні налаштувати головний файл код у формат коду, котрий запускається, як bash-скрипт, а також вказати в яких версійних діапазонах runtime модуля NodeJS допустити до запуску (рисунки 3.1-3.3).

```
"engines": {
  "node": "^14.0.0 || >=20.0.0"
},
```

Рисунок 3.1 – Налаштування рушія запуску проекту для перетворення коду в скрипт командного рядка

```
"bin": {
  "eg": "./index.js"
},
```

Рисунок 3.2 – Налаштування шляху до бінарного файлу для подальшого його запуску в консолі/терміналі

```
#!/usr/bin/env node
import runProgram from "./scripts/runProgram.js";
import welcome from './ui/welcome.js'

runProgram(async (program) => {
  if (program.args.length == 0) {
    await welcome();
    return;
  }
});
```

Рисунок 3.3 – Налаштування головного файлу для роботи у вигляді скриптового bash-файлу [13]

Рисунок 3.4 – Загальний вигляд інтерфейсу проекта в командному рядку

Після даних налаштувань далі ми прописуємо певну логіку, яка дозволить прочитати усі можливі команди з командного рядка, та ідентифіковувати їхнє призначення та виконувати потрібні дії для цього. На базі описаних вище моментів в логіці роботи генератора ми можемо побудувати певну модель роботи нашого проекту. Основна логіка буде працювати таким чином, що програмує очікує на вибір користувача, чи він обирає генерацію проекту, чи він будує нові компоненти до свого вже існуючого бекенду. Далі в залежності від вибору відбувається подалі дії генерації. Проте перед самою генерацією генератор повинен прочитати набір команд котрі до нього поступають, адже наш проект формується у вигляді скрипта командного рядка, а отже його першочергове завдання полягає у прочитанні та зрозумінні, що саме відправляє користувач і що саме потрібно виконати. Синтакс простої команди створення проекту виглядатиме наступним чином (рисунок 3.5):

```
C:\Users\AZ>eg new dev-project
```

Рисунок 3.5 – Приклад команди відправленої до генератора

В даному прикладі ми бачемо чітке розділення на деякі ключові слова, котрі генератор повинен розбити та опрацювати. В даному випадку ми задля відкритого доступу до нашого проекту ми використовуємо ключове для нього слово «eg», саме це ключове ідентифікує в операційній системі інформацію про те, що ми звертаємось саме до нашого генератора котрий сформувався в системних файлах у ролі bash-скритпа. Наступний ключовий словом у нас виступає команда «new». В контексті нашого генератора дана команда слугує для генерації проекта з використанням бібліотеки ExpressJS, відповідно для створення проекту ми використовуємо дане ключове слово, при цьому, якщо ми не вказали з самого початку назву майбутнього проекту у нас спрацює інтерактивне меню, де нас система попросить внести назву проекта. В даному прикладі 3 ключове слово є саме назва проекту, якщо ми дане ключове слово не вкажемо, відповідно спрацює 2 варіант генерації проекту котрий ми вказали раніше. Загальний алгорит роботи даної структури буде виглядати наступним чином (рисунки 3.6-3.8):

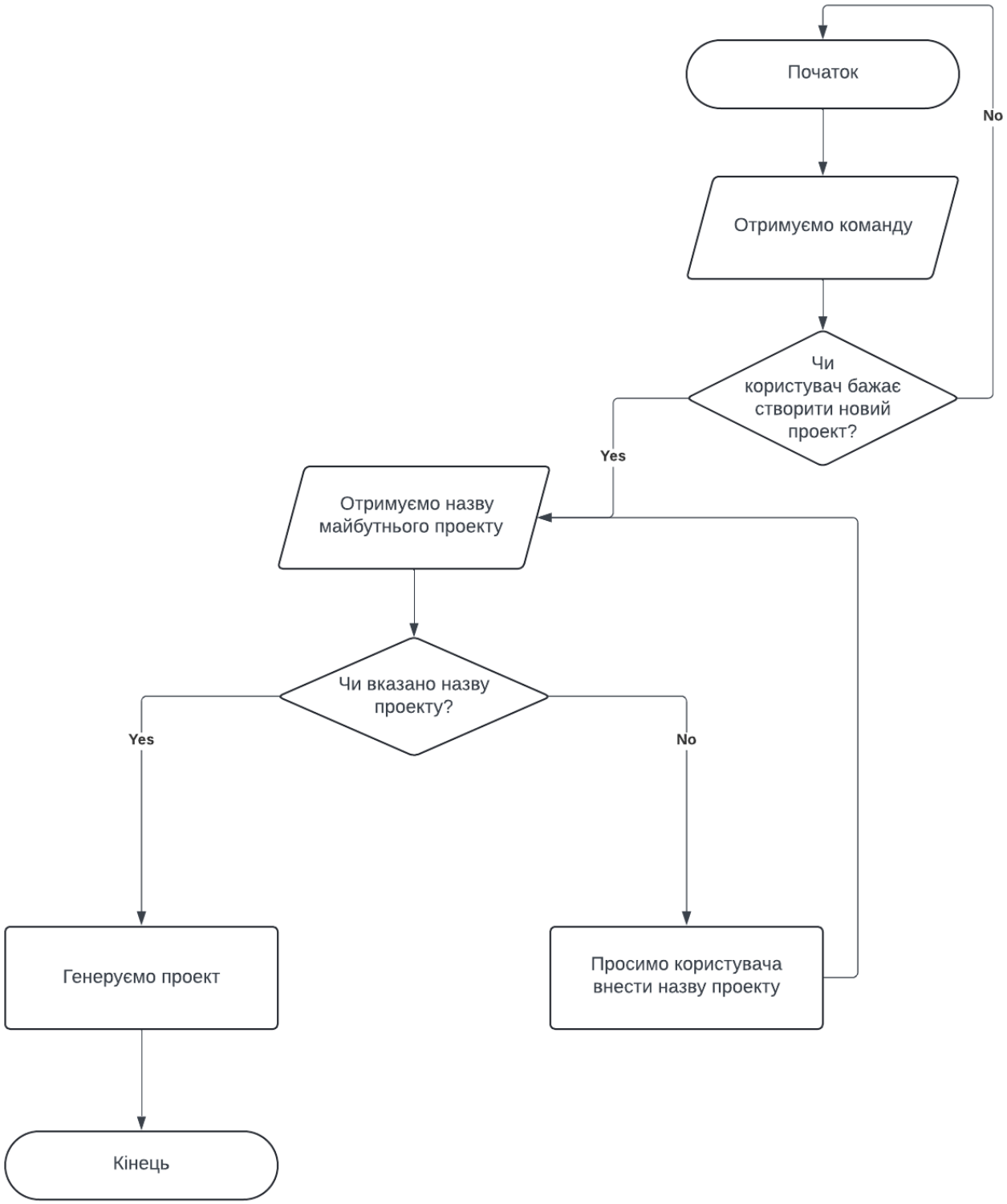

Рисунок 3.6 – Загальний вигляд алгоритму роботи команди створення проекту

```
import {generateProject, generateProjectFull} from '../scripts/generateProject.js';

export function newProject(program) {
  program
    .command("new")
    .description("generate project with name")
    .argument("<project-name>")
    .action(async (projectName) => {
      await generateProject(program, projectName);
    })
}

export function newProjectInteractive(program) {
  program
    .command("interactive")
    .description("generate project in interactive mode")
    .action(async () => {
      await generateProjectFull(program);
    })
}
```

Рисунок 3.7 – Функція зчитування команди створення проекту

```
export async function generateProject(program, name) {
  while (isValidProjectName(name)) {
    console.log(
      `${chalk.yellow(
        ">"
      )} Project must contains only:\n  - at least 5 characters\n  - lower case type of name\n  - project name may contain '_' and '-' characters.\nPlease
    );
    projectName = await askProjectName();
  }
  await header();

  projectName = name || (await askProjectName());

  while (isProjectAlreadyExists(projectName)) {
    console.log(
      `${chalk.yellow(
        ">"
      )} Target directory "${projectName}" is not empty. Please re-enter the project name.`
    );
    projectName = await askProjectName();
  }
  console.log(`${chalk.green('? Project name: ')} ${chalk.blueBright(projectName)}`);

  template = program.opts()["template"] || (await askTemplate());
  viewEngine = program.opts()["view"] || (await askViewEngine());
  packageManager = program.opts()["package"] || (await askPackageManager());
  try {
    await createExpressApp(projectName, template, viewEngine, packageManager);
    showTips(projectName);
  } catch (err) {
    console.log(`Error on creating app: ${chalk.red(err.message.toString)}`);
  }
}
```

Рисунок 3.8 – Функція створення проекту на базі зчитаних команд

Наступним нашим кроком є запит у користувача, які інструменти потрібно додати до проекту, наприклад який рушій для запуску HTML-сторінок чи яку стилістику коду використовувати. Для цього ми формуємо список для вибору даних варіацій, в першу чергу ми запитуємо у користувача, який формат шаблону

буде використовувати користувач для роботи (рисунок 3.9). Даний список буде реалізовано як пункт діалового вікна.

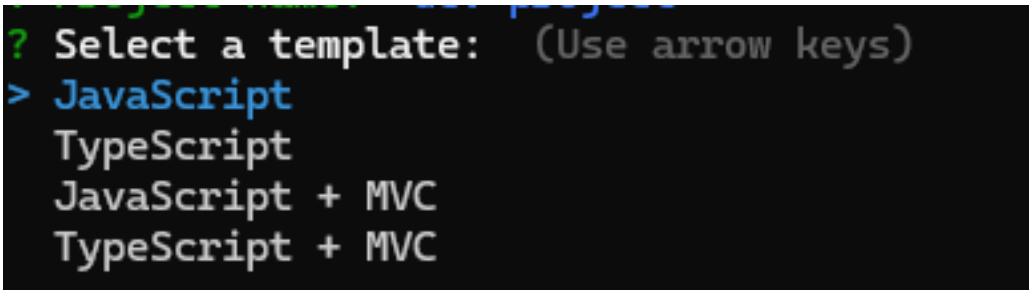

Рисунок 3.9 – Діалогове вікно вибору шаблону стилістики проекту

Наступним нашим кроком є втсановлення перевірки на правильність вказаного ім’я майбутнього проекту та вибір рушія запуску. Дані моменти є важливими компонентами для генерації проекту, адже є чіткі стандарти по створенні проекту, котрі порушувати строго заборонено. Якщо говорити стосовно назви проекту то головними вимогами є: не менше 5 символів, а також заборона використання пробілів(замість них доступно символи «_» та «-»). Дані вимоги є обо`язковими до виконання, адже не врахувавши їх система не дозволить нам завантажувати бібліотеки та вцілому запускати проект. Дані аспекти було додано в якості регулярних виразів (англ. Regular Expressions або RegExp), де ми вказуємо допустимі умови для роботи. В разі, якщо назва виявляється валідною(правильно вказаною), ми дозволяємо користувачеві далі генерувати проект. В інших випадках ми просимо користувача внести коректну назву ще раз. Для генерації рушія, що дозволить нам працювати із веб-сторінками ми використаємо 3 доволі поширених варіанта рушіїв, а саме: pug, hbs, ejs. Саме ці варіанти рушіїв є доволі поширеними у своєму використанні та використовуються досить часто. Після формування меню генерації потрібно підготувати шаблони, завдяки яким ми зможемо генерувати майбутні проекти. Для цього ми згідно кожного пункту у меню створимо окремий файл, який ми будемо постійно клонувати в потрібну директорію де генеруємо проект (рисунок 3.10).

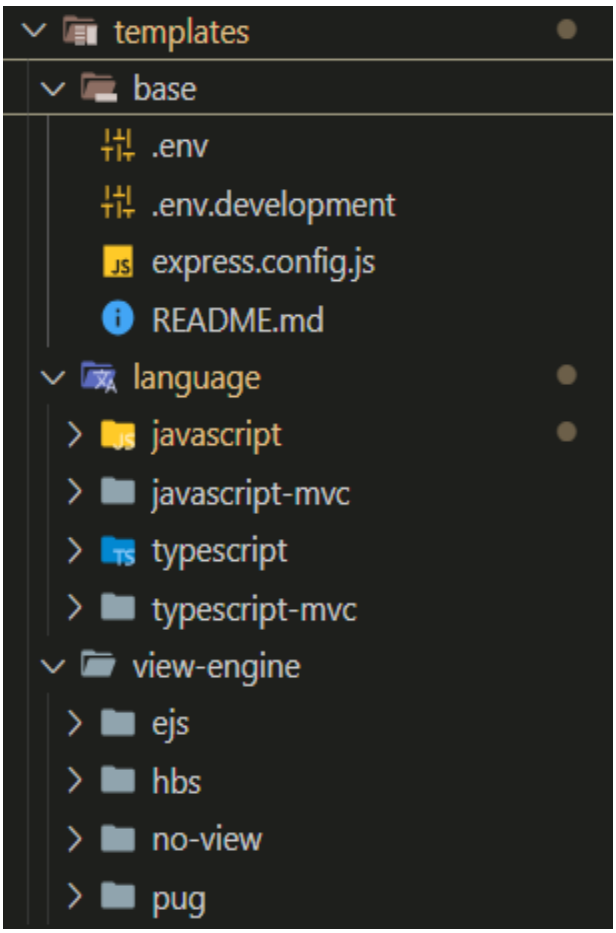

Рисунок 3.10 – Ієрархія файлів та папок шаблонів для майбутніх генерацій проектів

В даній ієрарії відображено основні варіації генерації проекту, в залежності від вибору користувача. Кожна деректорія містить в собі потрібні файли та директорії згідно специфікації.

Директорія «base» містить у собі шаблони, що є логічною частиною проекта вцілому. Іншими словами дані шаблони відповідають за налаштування та організації умов для запуску нашого майбутнього бекенд застосунку. Так, файли «.env» та «.env.development» відповідають за зберіагння спецілізованих змінних (інша назва змінні середовища), котрі будуть використовуватись для підключення до баз даних, для запуску бекенду під певним портом та проксі-налаштуванням. Даний файл також використовують для зберігання обмеженого набору даних котрі вказують на дані для доступу до тих чи інших сервісів. Файл «.express.config.js» відповідає за вказування налаштуваннь для бібліотеки ExpressJS та його складових.

Зазвичай цей файл містить у собі налаштування для самої бібліотеки, аніж для самого бекенду вцілому.

Директорія «language» містить у собі стилі подачі нашого проекту на базі обраної мови: typescript чи javascript, а також є варіанти вибору патерну проектування(стандарт вигляду нашого проекту). Таким чином ми для себе розділяє усі потрібні елементи на логічні сегменти, котрі потім ми зможемо без проблем внести у код генератора.

Директорія “view-engine” містить у собі файли конфігурацій для певного набору варіантів рушіїв для сапуску веб-сторінок. Створено директорії для 3 рушіїв – «ejs», «pug» та «hbs», а також одна директорія «no-view», де містяться конфігурації для бекендів, котрі не міститимуть у собі рушія для веб сторінок вцілому. Кожна із під директорій пістить у собі набір налаштувань для запуску та налаштування рушіїв (у випадку із «no-view» налаштування рушія для запуску веб застосунків не створено), котрі потім буде додано до майбутнього бекенду та його кодові структури вцілому.

Наступним нашим кроком є прописання логіки генерації проекту на базі готових шаблонів, котрі ми маємо заготовленими, для цього ми пропишемо певні головні команди для генерації, а саме отримання директорії звідки викликана команда (для цього використовується команда process.cwd() в JavaScript), адже саме це й буде місцем де ми створимо директорію проекту зі всіма потрібними файлами коду.

В першу чергу ми зберемо усю отриману інформацію і передамо її до функції, де ми будемо робити наступні кроки. Отримавши усю потрібну інформацію ми починаємо процес формування проекту, проте перед цим ми перевірямо чи дана директорія, яку наша система створить, існує. Якщо ця директорія відсутня, тоді ми починаємо генерацію.

Спочатку відбувається генерація конфігураційниї файлів проекту, шляхом копіювання їх в потрібну директорію, після чого на базі вибраного стилю коду нашого майбутнього проекту ми копіюємо інші шаблони з директорії «templates/language». На кінець ми додаємо (чи ігноруємо додавання) рушія для роботи із HTML-сторінками. Після завершення генерації ми запускаємо процес

встановлення потрібніх бібілотек до нашого новостворенного проекту, котрий було створено шляхом нашого генератора (рисунки 3.11-3.13).

```
export async function createExpressApp(projectName, template, viewEngine, packageManager) {
  const spinner = createSpinner("Creating project...")

  try {

    const PROJECT_PATH = path.join(process.cwd(), projectName)

    spinner.start()

    // Creating app structure
    await createStructure(template, viewEngine, projectName, PROJECT_PATH)
    // downloading dependency
    await installDependency[packageManager](projectName)

    spinner.success({ text: "Done." })
    console.log("")
  } catch (err) {

    spinner.error({ text: err })
    console.log("")
    throw new Error(err)
  }

}
```

Рисунок 3.11 – Функція генерації майбутнього проекту

```
const createStructure = async function (template, viewEngine, projectName, projectPath) {

  async function fromBase() {

    // copy directory from templates/base to express-app
    await fsp.cp(path.join(TEMPLATES_PATH, "base"), projectPath, { recursive: true })

  }

  async function fromTemplate() {

    // merge directory from templates/language to express-app
    await fsp.cp(path.join(TEMPLATES_PATH, "language", template), projectPath, { recursive: true })
    await Promise.all([generateDotGitignore(projectPath, projectName)])
  }

  async function fromViewEngine() {

    // merge directory from templates/view-engine to express-app
    await fsp.cp(path.join(TEMPLATES_PATH, "view-engine", viewEngine), projectPath, { recursive: true })

    // delete js, ts file
    if (template === "javascript" || template === "javascript-mvc") {
      await fsp.unlink(path.join(projectPath, "src", "app.ts"))
    } else {
      await fsp.unlink(path.join(projectPath, "src", "app.js"))
    }

  }

  await fromBase()
  await fromTemplate()
  await fromViewEngine()

}
```

Рисунок 3.12 – Функція формування структури майбутнього проекту

```
const installDependency = {
  "npm": (projectName) => {
    return exec(`cd ${projectName}&&npm i&&cd ${process.cwd()}`)
  },
  "yarn": (projectName) => {
    return exec(`cd ${projectName}&&yarn install&&cd ${process.cwd()}`)
  },
  "pnpm": (projectName) => {
    return exec(`cd ${projectName}&&pnpm i&&cd ${process.cwd()}`)
  },
}
```

Рисунок 3.13 – Функція інсталювання потрібних бібліотек до згенерованого проекту

Таким чином, ми отримуємо готовий до роботи генератор проектів на базі біліотеки ExpressJS, котрий може працювати на будь якій операційній системі, де встановлено програмний модуль запуску кода в реальному часі NodeJS, а також встановлено даний проект на персональний комп’ютер.

Для перевірки роботи нашого генератора буде відбуватись тестування нашого генератора на реальному прикладі. Для цього ми будемо генерувати бекенд для звичного замовлення від клієнта, котрому потрібно створити бекенд застосунок для його магазину. Для реалізації даного проекту потрібно створити HTML-сторінки, які будуть слугувати в якості адмін сторінками для власника магазину, а також потрібно створити базовий REST API сервіс(сервіс, котрий слугує для звязку із фронтендом). Для цього ми використаємо наш генератор бекенд структур. Для цього ми вкажемо команду – «eg new kovalbud-api» (рисунок 3.12), та встановимо усі потрібні компоненти для майбутньої роботи.В якості стилістики нашого бекенду ми будемо використовувати JavaScript та патерн проектування MVC (означ. Модель-Вигляд-Контролер, шаблон проектування коду, який слугує для розділення логічної частини від частини представлення вихідних даних, отриманих під час роботи логіки програми), а в якості рушія для веб-сторінок буде використано «hbs engine». «hbs engine» дозволяє робити якісну взаємодію із вибраним стилем коду нашого бекенду, а також має елементи оптимізації для швидкої генерації веб-сторінок.

Рисунок 3.12 – Процес генерації проекту на базі поставленої реальної задачі

Результатом у нас є готовий згенерований проект, з яким можна працювати і не витрачати час на ручне формування структури програмного коду.

Отже, для створення подібного програмного продукту, не потрібно докладати багато зусиль та часу, а також і ресурсів, для створення просто генератора бекенд структур. Для забезпечення розробки подібного продукту достатньо лише 3 основних бібліотек, а також допоміжні бібліотеки, котрі дозволяють розробнику зробити інтерфейс командного рядка більш інтерактивнішим та цікавішим.

ВИСНОВКИ

В ході дослідження було вивчено та проаналізовано проблематику, щодо вирішення ситуації із створенням проектів на основі бібліотеки ExpressJS. Було переглянуто різноматні спроби вирішення проблеми, а також розглянуто можливі версії створення подібного інтрефейсу командного рядка. В ході аналізу та моделюванню шляхів вирішення, було вивчено різноматні нові технології, а також різні варіації написання програмного коду.

Здійснено аналіз проблематики повязаної із генерацією бекенд структур на базі біліотеки ExpressJS, а також було досліджено акуальність проблематики в ході виконання поставленого завдання. В результаті було отримано позитивні результати досліджень і спроектовано варіанти вирішення проблематики одним із найпоширеніших методів його розвязання. Методом вирішення стало створення програми, котра дозволяла генерувати проект бекенду за стандартизованою структурою та в швидкому форматі для розробників.

Реалізовано генератор простих бекенд структур з використанням модуля запуску програмного коду в реальному часі NodeJS, бібліотеку Commander, Chalk, Logger та Loadash. Даний генератор може використовуватись, як допоміжний інструмент для створення бекенд застосунків за лічені хвилини із дотриманням усім вимог до формування проекту.

Розроблено модульний загальний інтерфейс проекту. Даний інтерфейс полегшує взаємодію із усіма доступни функціями генератора, а також вирішує проблему у користувацьому досвіді (від словосполучення User Experience або UX) при роботі з даним інтерфейсом.

Запропоновано використання розробленої системи на постійній основі для створення власних проектів, під час виконання замовлень на фріланс / аутсоурс проектах, створювати проекти рівня Enterprise, а також в якості елемента для вивчення новітніх моделей роботи із runtime модулем NodeJS.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке генератори і як їх використовувати. DevZone. URL: https://devzone.org.ua/qna/shcho-take-heneratory-i-iak-yikh-vykorystovuvaty. (дата звернення: 20.03.2024)

2. Історія створення ГПВЧ запропоноване Джон фон Нейманом. URL: http://surl.li/tzyjh (дата звернення: 20.03.2024)

3. Генератор псевдовипадкових чисел. Стаття з вільної енциклопедії. URL: http://surl.li/tzyjc (дата звернення: 20.03.2024)

4. Галицький Данііл Керований генератор псевдовипадкових двійкових векторів дип. робота. Київ 2020р URL: https://ela.kpi.ua/server/api/core/bitstreams/7790c8dc-1ec8-417e-b8f7-eec6d43598a7/content (дата звернення: 20.03.2024)

5. «ExpressJS Public Repositories». GitHub Community. URL: https://github.com/search?q=expressjs&type=repositories (дата звернення: 05.04.2024)

6. Geshan «expressjs-structure». GitHub. URL: https://github.com/geshan/expressjs-structure (дата звернення: 18.04.2024)

7. Irohanrajput «backendwithJS / 02_ DatabaseModeling». Github. URL: https://github.com/irohanrajput/backendwithJS/tree/main/02_DatabaseModeling (дата звернення: 18.04.2024)

8. Генератор бекенд структур для фреймворку Spring Boot URL: https://start.spring.io/ (дата звернення 28.04.2024)

9. Стаття авторства Манав Шрівастава – Let`s Build a CLI | Command Line Interface by NodeJS URL: https://medium.com/@manavshrivastava/lets-build-a-cli-command-line-interface-with-node-js-d3b5faacc5ea (дата звернення: 18.04.2024)

10. Документація для бібліотеки Commander URL: https://www.npmjs.com/package/commander (дата звернення: 28.04.2024)

11. Документація для бібліотеки Chalk URL: https://www.npmjs.com/package/chalk (дата звернення: 28.04.2024)

12. Документація для бібліотеки ExpressJS URL: https://expressjs.com/en/4x/api.html (дата звернення: 20.04.2024)

13. David Flanagan / JavaScript: The Definitive Guide, 7th Edition. – 2020. – 1080 с