МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є

"___" _____ 2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Програмний модуль розробки вебдодатків на базі node.js

Виконавець: Попов М.Д.

Керівник: Марченко Н.Б.

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Спеціальність 123 "Комп'ютерна інженерія"
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Литвиненко О. Є.

«_____»_____2021 р.

ЗАВДАННЯ
на виконання дипломного проє56ккту
Попова Михайла Дмитровича
(прізвище, ім'я, по батькові)

1.Тема роботи: «Програмний модуль розробки веб-додатків на базі node.js»

затверджена наказом ректора від "21" грудня 2020 року № 2523 /ст.

2. Термін виконання роботи: з 14 жовтня 2020 р. по 19 лютого 2021 р.

3. Вхідні дані до проєкту: постановка задачі до виконання роботи,

мова програмування: *JavaScript*, платформа *Node.js*, СУБД: *PostgreSQL.*

4. Зміст пояснювальної записки:

1) Аналіз проблем у галузі розробки веб-додатків;

2) Використані технології при розробці модуля;

3) Розробка модуля створення веб-додатків на базі node.js;

4) Використання модуля користувачем.

5. Перелік обов'язкового графічного матеріалу:

1) ;

2) ;

3) ;

3) ;

6. Календарний план-графік

№ пор.	Етапи виконання дипломного проєкту	Термін Виконання	Примітка
1	Провести аналіз літератури за темою дипломної роботи	14.10.2020 – 29.10.2020	
2	Написати розділ «Аналіз проблемної галузі та постановка задачі»	30.10.2020 – 12.11.2020	
3	Провести аналіз необхідних компонентів для реалізації програмного модуля.	13.11.2020 – 26.11.2020	
4	Написати розділ «Інформаційне забезпечення»	27.11.2020 – 03.12.2020	
5	Написати розділ «Програмний модуль розробки веб-додатків на базі node.js»	08.01.2021 – 14.01.2021	
6	Написати розділ «Використання модуля користувачем»	15.01.2021 – 21.01.2021	
7	Оформити пояснювальну записку	22.01.2021 – 27.01.2021	
8	Оформити графічний (ілюстрований) матеріал	28.01.2021 – 30.01.2021	

7. Дата видачі завдання «_14_» жовтня 2020 р.

Керівник дипломного проєкту _____ Марченко Н.Б.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Попов М.Д.
(підпис студента) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту «Програмний модуль розробки вебдодатків на базі node.js», 55 сторінок, 18 рисунків, 2 таблиці, 21 джерело літератури.

МОДУЛЬ, ВЕБ-ДОДАТОК, АРХІТЕКТУРА ДОДАТКІВ, БАГАТОПОТОЧНІСТЬ, АСИНХРОННІСТЬ, ОБРОБКА ДАНИХ, ФРЕЙМВОРК, JAVASCRIPT, NODE.JS, API

Об’єкт дипломного проєкту – розробка веб-додатків.

Предмет дипломного проєкту – модуль створення веб-додатків на базі node.js.

Мета дипломного проєкту – розробити програмний модуль для розробки веб-додатків на базі node.js;

Методи дослідження – мова програмування *JavaScript,* середовище розробки *WebStorm*, програмна платформа *Node.js.*

В дипломній роботі спроектовано та розроблено програмний модуль для розробки веб-додатків на базі node.js з використанням технологій *JavaScript* та *Node.js*, який дозволяє створювати кросплатформні веб-додатки, використовуючи спільну кодову базу. На базі модулю було спроектовано кросплатформений додаток для керування онлайн порталом в навчанні правил дорожнього руху, модуль керуванням усіма інтерфейсами порталу за допомогою розробленого *API.*

Прогнозні припущення щодо розвитку об’єкта дослідження – створення робочого зразка програмного модуля та використання його як базову платформу у розробці веб-додатків.

Результати дипломної роботи рекомендується використовувати при розробці нових веб-додатків на базі node.js.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API (Application programming interface) – інтерфейс для взаємодії клієнта та сервера.

REST (Representational state transfer) – архітектурний стиль взаємодії додатка із сервером.

GUI (Graphical User Interface) – система засобів для взаємодії користувача з комп'ютером, заснована на представленні всіх доступних користувачеві системних об'єктів і функцій у вигляді графічних компонентів екрану (вікон, значків, меню, кнопок, списків).

DNS (Domain Name Service) – система доменних імен.

TCP/IP (Transmission Control Protocol/Internet Protocol) – протокол управління передачею/міжмережевий протокол.

ПЗ – програмне забезпечення.

ВСТУП

Ще на початку 1990 років, веб-розробка почала стрімко зростати в популярності. Вже через три роки число інтернет-хостів перетнуло позначку в два мільйони, а в мережі діяло більш ніж 600 веб-сайтів. Через один рік був створений консорціум, який об’єднав вчених із різних компаній, який почав займатись стандартами в світі Інтернету. Ще через декілька років започаткувалась специфікація HTML 2.0. Із початком розвитку технологій, були започатковані перші архітектурні підходи, нові мови програмування, та багато іншого. Кожен підхід, приносив певні результати, деякі з них використовуються і на теперішній час. Спільнота веб-розробників, забезпечує безперервний розвиток різних систем. Але ідеальних систем не існує, і розглядаючи проблематику в першому розділі дипломної роботи, можемо прийти до певних висновків, та проблем які існують на даний час у світі веб-розробки. За останні 10 років, спільнота перестала створювати архітектурні підходи, вирішувати глобальні проблеми у розробці веб-додатків, натомість тільки розширює світ веб-розробки новими методами, та створює нові фреймворки для полегшення розробки, уникаючи глобального вирішення проблем, які виникають у тих чи інших підходах. Мрія розробників JavaScript спільноти, щодо використання її на фронтенд та бекенд частинах вже більше десяти років працює на ринку. Мова, яка має досить багато плюсів, підходів, започаткована ще в 1995 році, досі вирішує купу проблем, і не втрачає свою актуальність до тепер. Але й вона має певні недоліки.

Вихід з цієї ситуації потрібно шукати в проектуванні нових архітектурних підходів, застосуванні останніх технологій, технологіях кросплатформеної розробки для створення зручного модулю, який дозволить перевикористовувати компоненти та функції, який не буде залежати від купи бібліотек, буде мати легку вагу коду, простоту розуміння, і не тягнути за собою пів-світу компонентів із пакетного менеджеру npm чи yarn.

Досягти ефективної роботи модулю, можливо за допомогою платформи Node.js. Також автоматизувати в цій системі певні процеси. Під час створення

веб-додатків від програміста вимагається виконання певних процесів, починаючи від створення адаптивних інтерфейсів, до оптимізації додатку під певні платформи, слідкуючи за швидкодією програмного модуля.

В даній дипломній роботі розроблено модуль, що реалізує сучасні кросплатформені технології, і вирішує проблеми платформи Node.js. За його допомогою створено програмний додаток для керування усіма інтерфейсами веб-сайту що навчає правилам дорожнього руху, та готує користувачів порталу до державної комісії в отриманні водійського посвідчення. Кількість пристроїв, які можуть підтримуватись системою, досягає десяткам тисяч.

Потрібно розробити архітектуру, алгоритми модуля, які будуть повністю відповідати вимогам і швидко та якісно виконувати поставлені задачі. Обрати середовище розробки, мови програмування, програмні інструменти та інші засоби для створення програмного продукту.

Об’єктом дослідження даної роботи є розробка веб-додатків. Предметом даної роботи є модуль для розробки веб-додатків на базі Node.js

В першому розділі аналізується проблематика в розробці веб-додатків, порівнюються технології які використовуються на ринку. Розглянуто тенденції спільноти розробників, методи, платформи у розробці веб-додатків. На основі аналізу прийняті рішення що до проектування програмного додатку.

В другому розділі «Інформаційне забезпечення» проаналізовано та створено структуру модуля для розробки веб-додатків, обрані методи для створення та сформульовані основні функціональні вимоги.

В третьому розділі «Програмна реалізація» описані системні вимоги до розроблюваного додатку, розроблено та протестовано модуль для розробки веб-додатків.

В четвертому розділі «Керівництво користувача» описано покрокову інсталяцію програмного модуля. Розглянуто інтерфейс користувача та описано інструкцію з експлуатації.

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Веб-додаток у ролі застосунку

Поняття веб-додаток – це будь-яка комп'ютерна програма, яка виконує певну функцію, використовуючи веб-браузер у якості свого клієнта. Додаток може бути таким же простим, як дошка оголошень чи контактна форма, або настільки складний як текстовий редактор Microsoft Word що поданий у формі веб-сайту. Головною функцією браузера виступає - відкриття веб-сторінок. Самі ж сторінки веб-сайту складаються з коду, який власне і отримує браузер з сервера, на якому знаходиться сайт. Такий тип передачі інформації між користувачем, та сервером має визначення «клієнт-сервер», тобто це середовище в якому кілька комп'ютерів обмінюються інформацією, наприклад введенням інформації в базу даних, або спілкуванням в онлайн чаті. У випадку з введенням інформації, "клієнт" – це програма подана користувачу, яка використовується для введення даних, а "сервером" – програма що використовується для зберігання або обробки чи передачі інформації.

Найчастіше веб-додаток являє собою веб-сайт, на якому розміщені сторінки з частково або повністю несформованим вмістом. Остаточний вміст формується тільки після того, як відвідувач сайту запросить сторінку з веб-сервера (Рис. 1.1).

Розглянемо деякі з основних переваг використання веб-додатків. Одним із аспектів визначають звільненням розробника від відповідальності за створення клієнтської чи серверної частин, окрім того істотною перевагою побудови є те, що усі функції виконуються незалежно від операційної системи клієнту. Різна реалізація специфікацій додатку може викликати проблеми при розробці і подальшої підтримки, адже можливість користувача налаштовувати багато параметрів браузера може перешкодити коректній роботі застосунку.

Унаслідок універсальності й відносної простоти у розробці – веб-додатки стали широко популярними на початку 2000-х років [1]. Але ще на початку своєї популярності існував інший підхід із використанням Adobe Flash або Java-аплетів для повної або часткової реалізації користувацького інтерфейсу.

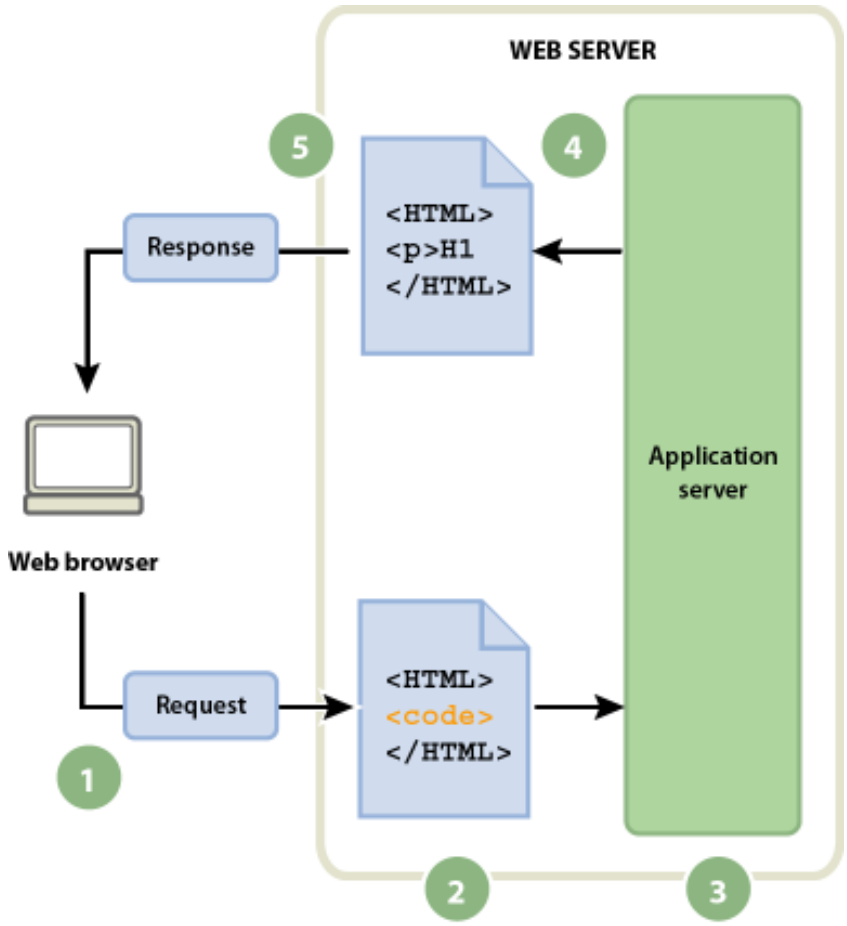

Рис. 1.1 – Зображення принципу роботи веб-додатків

Через архітектурну схожість аплетів та Flash з традиційними клієнт-серверними застосунками, існують суперечки щодо коректності зарахування подібних систем до стандартної концепції веб-додатків. Альтернативним терміном для таких концепцій розробки є «Насичений інтернет додаток» [1]. Хоча подані технології надавали розробнику більший контроль над інтерфейсом і були здатні обходити багато невідповідностей у конфігураціях браузерів, несумісність між Java або Flash реалізаціями клієнта спричиняла певні ускладнення. Через плин часу, та невизнання спільнотою розробників, станом на 2020 рік, Java-аплети та Flash-технологія практично вийшли з ужитку.

Веб-додатки існують з тих пір, поки мережа інтернет не набула популярності. Наприклад, Ларрі Уолл розробив Perl, популярну мову сценаріїв на

стороні серверу, ще у 1987 році. Це було за сім років до того, як мережа Інтернет почала набирати популярність поза академічними та технологічними колами. Перші додатки були відносно простими, але наприкінці 1990 років відбувся поштовх до більш складних.

1.2 JavaScript як серверна мова програмування

У 1995 році компанія Netscape поставила завдання вбудувати мову програмування Scheme чи «якусь схожу» в браузер Netscape. З часом, концепція розроблюваної мови програмування була розширена до можливості використання безпосередньо в HTML-коді сторінки. Під час розробки планувалось створити мову, що могла зв'язати різні частини веб-сайтів: зображення, Java-аплети, об'єктну модель документа. Ця мова повинна була стати зручною для веб-дизайнерів та некваліфікованих програмістів. Робочою назвою нової мови була Mocha, яка була змінена на LiveScript в перших двох бета-версіях браузера Netscape 2.0. А дещо пізніше, користуючись популярністю бренду Java, LiveScript був перейменований на JavaScript і третя бета-версія вже вийшла в світ вже вийшла з сучасною назвою.

Наразі JavaScript, є однією з найпопулярніших мов програмування в мережі інтернет. В перші роки свого існування, більшість професійних програмістів скептично ставилися до нової мови, до цього ще й цільова аудиторія складалася з програмістів-аматорів, але поява AJAX змінила ситуацію та звернула увагу професійної спільноти, а її подальші модифікації внесли багато корисних можливостей, яких не вистачало для ефективного програмування. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки, поширилося використання JavaScript поза браузером.

JavaScript – це динамічна мова зі слабо типізованими, динамічно розширюваними об’єктами, які неформально оголошуються в міру необхідності. Функції в ній є повноцінними об’єктами і зазвичай використовуються у вигляді анонімних замикань, шо робить JavaScript більш потужною мовою, порівняно з деякими іншими, часто вживаними для розробки веб-додатків.

Один з основних недоліків JavaScript – глобальний об’єкт. Всі змінні верхнього рівня «звалюються» до глобального об’єкту, і при використанні одночасно декількох модулів може призвести до некерованого хаосу. Оскільки веб-додатки зазвичай складаються з безлічі об’єктів, можливо, що створювалися різними організаціями, то може виникнути побоювання, ніби програмування на JavaScript те ж саме що ходіння по мінному полю, нашпигованому конфліктуючими між собою глобальними об’єктами. Однак це не так, оскільки в мові використовується система організації модулів CommonJS, а це означає, що локальні змінні деякого модуля так і будуть локальними в ньому, нехай навіть виглядають як глобальні. Таке чітке розмежування між модулями вирішує проблему глобального об’єкту.

Використання єдиної мови програмування на сервері і на клієнтській частині давно вже було мрією розробників. З появою Node, нарешті мрія стала реальністю, та зробила JavaScript мовою, що використовується по обидві сторони вебу. Веб-додатки можуть виконуватись на серверах, які використовують Java 6, та наступних версій, але окрім Java, існує ряд платформ які використовують інтерпретатори для виконання її на серверній частині. Так наприклад JavaScript на стороні серверу виконується на проектах компанії Google. Продукт Google Sites дозволяє підлаштовувати сценарії за допомогою двигуна Rhino, який повністю написаний на JavaScript. Rhino перетворює JavaScript-скрипти в Java-класи, працює як в режимі компіляції, так і декомпіляції. Оскільки головне призначення це використання в server-side додатках, двигун не має вбудованої підтримки об'єктної моделі браузера, які зазвичай асоціюються з JavaScript. З початку набуття популярності мови у вигляді серверної, було розроблено декілька платформ для її виконання. Список платформ для виконання серверних додатків на JavaScript:

1. Jaxer, інтерпретатором є SpiderMonkey;
2. Persevere-framework, інтерпретатором є Rhino;
3. Helma, двигун Rhino;
4. V8cgi, як інтерпретатор використовується V8;
5. Node.js, інтерпретатором є V8;

6. Gopherjs, двигун Go.

Розробники клієнтських додатків, незалежно від того, використовують вони JavaScript чи ні, повинні усвідомлювати, що їх творчість може перебувати під контролем зловмисників. Тому будь-яка перевірка на стороні клієнту, нажаль під загрозою, та може бути обійдена, а код, який зазнавав обфускації:

1. Може стати об’єктом зворотної розробки;
2. Дані форми можуть бути відправлені на сервер, минаючи валідацію;
3. Скрипти частково можуть бути відключені.

Обфускація – це заплутування коду, приведення вихідного тексту або виконуваного коду програми до виду, що зберігає її функціональність, але ускладнює аналіз, розуміння алгоритмів роботи і модифікацію при декомпіляції.

1.3 Вступ до платформи Node.js

Node.js – це програмне забезпечення з відкритим вихідним кодом, яке має можливість працювати з декількома апаратними платформами, або на різних операційних системах для виконання мови JavaScript [2]. Від моменту випуску цієї платформи в 2009 році вона стала надзвичайно популярною та виконує велику роль в області веб-розробки. Важливим моментом після випуску платформи є можливість запустити JavaScript не тільки в браузері, а в якості окремого додатку. Платформа побудована на базі JavaScript двигуна V8 від компанії Google. Основним та найчастішим випадком використання є створення веб-додатків, проте сфера використання на цьому не обмежується. JavaScript тотально домінує на стороні фронтенду та не має явних конкурентів, але на стороні бекенд-розробки, ситуація має інший напрям. Проте, якщо обирати JavaScript для серверного ряду, то проект набуває ряд переваг:

1. Зростання ефективності розробки завдяки використанню однієї мови для фронтенд та бекенд частин, і можливості перевикористання коду;
2. Можливість використовувати найбільший пакетний менеджер npm;
3. Простіший пошук розробників, так як JavaScript входить до списку найпопулярніших мов програмування;

4. Node.js підходить для розробки RTA, веб-додатків, що реагують на дії користувачів в реальному часі;
5. Node.js легко та швидко обробляє велику кількість запитів, та забезпечує швидкодію програми.

Для того, щоб полегшити роботу розробникам, було створено велику кількість бібліотек. Розглянемо список найпопулярніших фреймворків:

1. Express.js – спроектований для створення вебдодатків та API. Де-факто є стандартним каркасом для Node.js.
2. Meteor.js – є просто надбудовою над node.js, головною метою у створенні є простота у використанні.
3. Sails.js – інфраструктура веб-додатків, працює за принципом «модель - представлення – контролер».
4. Koa.js – гнучка інфраструктура веб-додатків на базі Node.js, що надає набір функцій для веб і мобільних додатків.
5. Nest.js – створений для полегшення життя розробника, який використовує правильні архітектурні підходи і диктує свої правила в розробці.

Node.js (Рис. 1.2) як бекенд частину використовують Netflix, Uber, eBay та інші відомі організації та проекти.

Рис. 1.2 – Логотип платформи Node.js

Платформу розробив Ryan Dahl у 2009 році, після двох років експериментування зі створенням серверних веб-компонентів на Ruby та іншими мовами. У ході своїх досліджень він прийшов до висновку, що замість традиційної моделі паралелізму на основі потоків слід звернутися до подієво-орієнтованим системам. Ця модель була обрана за простоту (добре відомо, що

багатопотокові системи важко реалізувати правильно), за низькі накладні витрати, в порівнянні з ідеологією «один потік на кожне з'єднання», і за швидкодію. Мета Node - запропонувати «простий спосіб побудови масштабованих мережевих серверів». При проектуванні за зразок було взято такі системи, як EventMachine (Ruby) і каркас Twisted (Python). Прийнята в Node модель принципово відрізняється від поширених платформ для побудови серверів додатків, в яких масштабованість досягається за рахунок багатопоточності. Стверджується, що завдяки подієво-орієнтованій архітектурі знижується споживання пам'яті, підвищується пропускна здатність і спрощується модель програмування. Зараз платформа Node швидко розвивається, і багато хто вважає її привабливою альтернативою традиційному підходу до розробки веб-додатків - на базі Apache, PHP, Python і т. д. В основі Node лежить автономна віртуальна машина JavaScript з розширеннями, що роблять її придатною для програмування загального призначення з упором на розробку серверів додатків. Платформу Node не має сенсу прямо порівнювати ні з мовами програмування, які зазвичай використовуються для створення веб-додатків, ні з контейнерами, що реалізовують протокол HTTP, наприклад Apache, Tomcat, або Glassfish. У той же час багато хто вважає, що потенційно вона може замінити традиційні стеки веб-додатків.

В основі реалізації лежить цикл обробки подій неблокуючих введень/виведень і бібліотеки файлового і мережевого введень/виведень, причому все це побудовано над двигуном V8 JavaScript запозиченого з веб-браузера Chrome. Бібліотека введення/виводу володіє достатньою спільністю для реалізації будь-якого протоколу на базі TCP або UDP, DNS, HTTP, IRC, FTP та інших. Але хоча вона підтримує розробку серверів і клієнтів довільного протоколу, найчастіше застосовується для створення звичайних веб-сайтів, де замінює Apache з PHP або Rails.

1.4 Дослідження існуючих фреймворків для розробки веб-додатків

Якщо проаналізувати результати опитувань серед розробників, що проводилися у 2020 році на платформі StackOverflow, можемо зробити висновок,

що більша частина розробників на стадії вибору місця роботи вважає що головним аспектом виступає: стек технологій, використання фреймворків, мови програмування, а ось вже на другій сходинці є пункт апаратне забезпечення та культура компанії (Рис. 1.3).

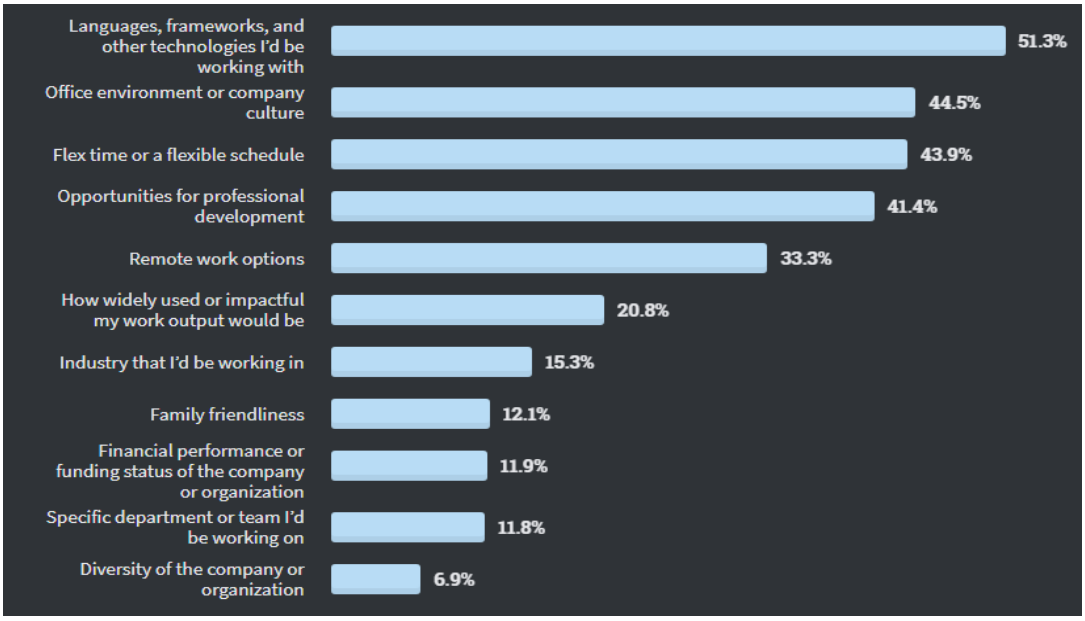

Рис. 1.3 – Фактори, що впливають на вибір місця роботи за опитуванням на ресурсі StackOverflow

Якщо поглянути на результати дослідження StackOverflow, то виявиться, що Express знаходиться на першому місці серед бекенд-фреймворків, які полюбляють розробники (Рис. 1.4). Інший фреймворк, Django, слідує з невеликим відривом. В результаті виявляється, що проект, заснований на перспективній скриптовій мові JavaScript, лідирує, а за ним вже слідує проект Django, в якому використовується одна з мов, що володіє найбільшими можливостями, тобто - Python.

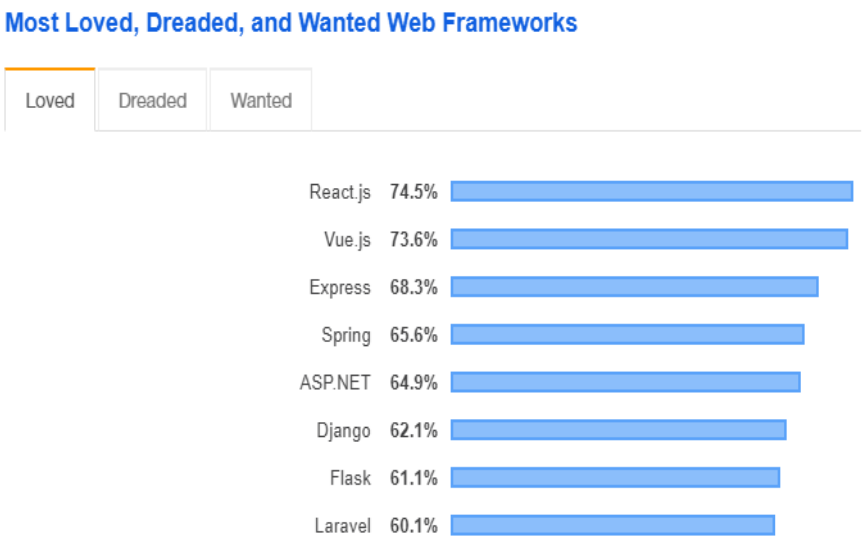

Рис. 1.4 – Популярність веб-фреймворків згідно дослідження Stack Overflow

Розглянемо порівняння фреймворків для платформ Django та Node.js.

Django - це опенсорсний бекенд-фреймворк, написаний на Python. Як відомо, Python – це одна з мов, які користуються найбільшою любов'ю розробників. Але у того, щоб обрати Django в якості фреймворку у 2021 році, є й інші причини. Django дозволяє без особливих складнощів створювати динамічні веб-додатки з використанням Python. Фреймворк підтримує патерн MVC. Це допомагає розробникам в поділі призначеного для користувача інтерфейсу і бізнес-логіки додатку. Django не відноситься до мінімалістичних фреймворків, широко використовуваних для розробки мікросервісів. Він відрізняється потужністю, універсальністю і певною своєрідністю. Автори цього фреймворка серйозно ставляться до безпеки, тому вони надають розробникам що використовують Django, відповідні інструменти. Фрейморк підтримує систему автентифікації користувачів, містить інструменти для захисту від різних атак. Серед них - засоби захисту від SQL-ін'єкцій, від міжсайтового скриптинга, від між-сайтової підробки запитів. Проекти відрізняються компактністю коду. Розробники, що використовують Django, можуть моделювати базові класи. Це означає, що в їх розпорядженні завжди є ORM. Django - це крос-платформний проект який відмінно працює на різних операційних системах. Крім того, підтримує взаємодію з різними базами даних, а його додатки, добре піддаються масштабуванню. Так як Django є великим і монолітним фреймворком, цей фактор дозволяє спільноті розробляти сотні універсальних модулів і додатків, але знижує

швидкість розробки самого Django. Крім того, фреймворк повинен підтримувати зворотну сумісність, тому він розвивається відносно повільно.

Node.js - платформа, яка виводить мову JavaScript за межі браузера і дозволяє використовувати його ще й в серверних додатках. В основі платформи лежить виключно швидкий движок JavaScript, запозичений з браузера Chrome, V8, до якого додана швидка і надійна бібліотека асинхронного мережевого вводу/виводу. Поява Node.js уможливила фуллстак-розробку веб-проектів на JavaScript. В результаті чого в розпорядженні розробників серверних частин додатків виявилися і сильні можливості JavaScript, і напрацювання екосистеми JS, бібліотеки, якими стало реально скористатися в серверному оточенні. Продуктивність JavaScript-коду досить висока у застосуванні його в проектах де важливим аспектом є швидкість роботи коду. Код клієнтських і серверних частин проектів легше підтримувати в узгодженому стані, так я в обох середовищах використовується одна мова. Завдяки існуванню модулів Node.js, які, по суті, являють собою певним чином оформлені фрагменти коду, розробники можуть із зручністю використовувати в своїх проектах чужий код разом з власними напрацюваннями. Платформа Node.js, і, відповідно, засновані на ній фреймворки, відрізняються невибагливістю до ресурсів і масштабованості. У Node.js JavaScript код компілюється в машинний код, що дозволяє отримати набагато вищу продуктивність, ніж при інтерпретації коду. Спільнота JavaScript-розробників бачить постійне поліпшення продуктивності Node.js за рахунок того, що Google постійно працює над вдосконаленням V8. Завдяки тому, що в Node.js є система введення-виведення, що не блокує головний потік, платформа демонструє високу продуктивність. Node.js, в найближчому майбутньому, може стати платформою, яку використовуватимуть для проведення «важких» обчислень, на зразок тих, які застосовуються для вирішення завдань у машинному навчанні. Якщо врахувати сильні сторони проекту, стають зрозумілими причини її величезної популярності.

1.5 Аналіз створення додатків на базі Node.js

JavaScript в Node - це не та мова, з якою усі ми знайомі з досвіду роботи в браузерах. У Node не вбудована ні об'єктна модель документа (DOM), ні будь-які

ще можливості веб-браузера. Саме мова JavaScript у поєднанні з асинхронним уведенням/висновком робить Node потужною платформою для розробки додатків. Але от для чого Node непридатна, так це для розробки персональних додатків з графічним інтерфейсом користувача (GUI). На сьогоднішній день в Node немає вбудованого еквівалента Swing (або SWT). Немає можливості підключення бібліотеки GUI для Node, і впровадити Node в браузер теж не можна. Якби для Node існувала бібліотека GUI, то на цій платформі можна було б будувати і персональні програми. Нещодавно з'явилося кілька проектів по створенню інтерфейсу між Node і GTK, підсумком яких повинна стати крос-платформна бібліотека GUI. До складу двигуна V8, використовуваного в Node, входять API-розширення, що дозволяють додавати до кодової бази написаний на C/C ++ код для розширення JavaScript або інтеграції движка з платформеними бібліотеками. Крім вбудованого вміння виконувати код на JavaScript, включені до складу дистрибутива модулі надають і інші можливості:

1. Утиліти командного рядка (для включення в скрипти оболонки);
2. Засоби написання інтерактивних консольних програм (цикл «читання - виконання - друк»);
3. Чудові функції управління процесами для спостереження за дочірніми процесами;
4. Об'єкт Buffer для роботи з двійковими даними;
5. Механізм для роботи з сокетами TCP і UDP з повним комплектом зворотніх викликів у відповідь на події;
6. Пошук в системі DNS;
7. Засоби для створення серверів і клієнтів протоколів HTTP і HTTPS, побудовані на основі бібліотеки TCP-сокетів;
8. Засоби доступу до файлової системи;
9. Вбудована підтримка автономного тестування за допомогою тверджень.

Мережевий шар Node знаходиться на низькому рівні, але працювати з ним все одно просто. Наприклад, модулі HTTP дозволяють реалізувати HTTP-сервер (або клієнт), написавши всього кілька рядків коду, але, тим не менш, на цьому рівні програміст працює дуже близько до реальних запитів по протоколу і може

точно вказати, які HTTP-заголовки слід включати в відповідь на запит. Якщо програміст на PHP зазвичай не цікавиться заголовками, то для програміста на Node вони істотні. Іншими словами, написати на Node HTTP-сервер дуже просто, але типовому розробнику веб-додатків немає потреби працювати на такому низькому рівні. Наприклад, кодуючи на PHP, програміст припускає, що Apache вже присутня, так що реалізовувати серверну частину стека йому не потрібно. Спільнота, що склалося навколо Node, створила широкий спектр каркасів для розробки веб-додатків, у тому числі Connect, які дозволяють швидко конфігурувати HTTP так, щоб надавалося все, до чого ми звикли, - сеанси, куки, обслуговування статичних файлів, протоколювання, та багато іншого. JavaScript ні в чому не поступається іншим мовам, але при цьому підтримує багато сучасних уявлень про те, якою повинна бути мова програмування. Завдяки широкому поширенню є чимало досвідчених програмістів на JavaScript.

Це динамічна мова зі слабо типізованими, динамічно розширюваними об'єктами, які неформально оголошуються в міру необхідності. Функції в ній є повноцінними об'єктами і зазвичай використовуються у вигляді анонімних замикань. Це робить JavaScript більш потужною мовою, порівняно з деякими іншими, часто вживаними для розробки веб-додатків. Теоретично наявність подібних можливостей має підвищувати продуктивність програмістів. Але суперечки між прихильниками динамічних і статичних мов, а також суворої і слабкою типізації досі не припинилися і навряд чи коли-небудь закінчаться.

Завдяки асинхронної івент-орієнтованій архітектурі Node демонструє таку високу продуктивність що є ще однією з переваг у розробці на платформі. Якщо розглядати традиційну модель сервера в яких паралелізм забезпечується за рахунок використання блокуючого вводу/виводу і декількох потоків, де кожен потік повинен чекати завершення введення/виводу, перед тим як приступити до обробки наступного запиту, то у Node є єдиний потік виконання, без будь-якого контекстного перемикання або очікування вводу/виводу. При будь-якому запиті введення/виведення задаються функції обробки, які згодом викликаються з циклу обробки подій, коли стануть доступні дані або відбудеться ще щось значуще. Модель циклу обробки подій і обробника подій - річ поширена, саме так

виконуються написані на JavaScript скрипти в браузері. Очікується, що програма швидко поверне управління циклу обробки, щоб можна було викликати наступне вартісне в черзі завдання. Автор платформи в презентації «Cincode Node», показав що відбувається при виконанні такого коду:

```
result = query ('SELECT * from db);
```

Виконання програми в цій точці призупиняється на час, поки шар доступу до бази даних відправляє запит базі, яка обчислює результат і повертає дані. Залежно від складності запиту його виконання може зайняти помітний час. Це погано, тому що поки потік простоює, може прийти інший запит, а якщо зайняті всі потоки, то запит буде просто відкинутий. Та й контекстне перемикання обходиться не безкоштовно; чим більше запущено потоків, тим більше часу процесор витрачає на збереження та відновлення їх стану. Крім того, стек кожного потоку займає місце в пам'яті. І просто за рахунок асинхронного подієво-орієнтованого введення/виведення Node усуває більшу частину цих накладних витрат, привносячи зовсім небагато власних.

Часто розповідь про реалізацію паралелізму за допомогою потоків супроводжується застереженнями типу «дорого і загрожує помилками», «ненадійні примітиви синхронізації в Java» або «проектування паралельних програм може виявитися складним, і не виключені помилки» (фрази взято з результатів, виданих пошуковою системою). Причиною цієї складності є доступ до поділюваних змінним і різні стратегії запобігання взаімоблокіровок і змагань між потоками. «Примітиви синхронізації в Java» - один із прикладів такої стратегії, і, очевидно, багато програмістів вважають, що користуватися ними важко. Щоб якось приховати складність, притаманну багатопоточність паралелізму, створюються каркаси типу java.util.concurrent, але все одно деякі вважають, що спроба запроторити складність подалі не чинить проблему проще.прізивает підходити до паралелізму по-іншому. Зворотні дзвінки з циклу обробки подій - набагато простіша модель паралелізму, як для розуміння, так і для реалізації.

Щоб пояснити необхідність асинхронного введення/виводу, Райан Дав нагадує про відносне часу доступу до об'єктів. Доступ до об'єктів в пам'яті

(порядку наносекунд) проводиться швидше, ніж до об'єктів на диску або відвідай (мілісекунди або секунди). Час доступу до зовнішніх об'єктів вимірюється незліченною кількістю тактових циклів і може виявитися вічністю, якщо клієнт, не дочекавшись завантаження сторінки протягом двох секунд, втомиться витріщатися на вікно браузера і відправиться в інше місце.

У Node вищезгаданий запит слід було б записати так:

```
query (SELECT * from db ', function (result) {
    //виконати операції з результатом
 }):
```

Різниця в тому, що тепер результат запиту не повертається в якості значення функції, а передається функції зворотного виклику, яка буде викликана пізніше. Таким чином, повернення в цикл обробки подій відбувається майже відразу, і сервер може перейти до обслуговування інших запитів. Одним з таких запитів буде відповідь на запит, відправлений базі даних, і тоді буде викликана функція зворотного виклику. Така модель швидкого повернення в цикл обробки подій підвищує ступінь використання ресурсів серверів. Це чудово для власника сервера, але ще більший виграш отримує користувач, перед яким швидше відкривається вміст сторінки.

У наші дні веб-сторінки все частіше збирають дані з десятків джерел. Кожному потрібно надіслати запит і дочекатися відповіді на нього. Асинхронні запити дозволяють виконувати ці операції паралельно - відправити відразу всі запити, задавши для кожного власний зворотний виклик, і, не чекаючи відповіді, повернутися в цикл обробки подій. А коли прийде відповідь, буде викликана відповідна йому функція. Завдяки паралельному розподілу дані можна зібрати набагато швидше, ніж, якби запити виконувалися синхронно, один за іншим. І користувач по той бік браузера щасливий, так як сторінка завантажується швидше. Тобто, однією з причин подання подібного додатку, є формалізація даних, до одного стандарту в кодовій базі, отримуючи функціонал з модульної системи. Завдяки даній системі розробник матиме можливість мінімізувати кількість коду, та перевикористовувати модулі, не імпортуючи можливо понад важкі модулі, безпосередньо до кожного сервісу.

1.6 СУБД PostgreSQL

PostgreSQL – це вільно поширювана об'єктно-реляційна система управління базами даних, є найбільш розвиненої з відкритих СУБД в світі і є реальною альтернативою для комерційних баз даних (Рис 1.5) [4].

Рис. 1.5 – Логотип СУБД. PostgreSQL

PostgreSQL створена на основі некомерційної СУБД Postgres, розробленої як open-source проєкт в Каліфорнійському університеті в Берклі. До розробки Postgres, що почалася в 1986 році, мав безпосереднє відношення керівник більш раннього проекту Ingres, Майкл Стоунбрейкер. на той момент вже придбаного компанією Computer Associates. Під час створення Postgres були застосовані багато вже раніше зроблених напрацювання та перейнятих практик з Ingres. Стоунбрейкер і його студенти розробляли нову СУБД протягом восьми років з 1986 по 1994 рік. За цей період в синтаксис були введені процедури, правила, призначені для користувача типи і інші компоненти.

Перелік основних переваг СУБД:

1. Надійність;
2. Продуктивність;
3. Можливість розширення;
4. Підтримка великої кількості стандартів SQL;
5. Підтримка великої кількості типів даних;
6. Наслідування;
7. Підтрика функцій в запитах;
8. Активна розробка;

Існує два підходи при роботі з базою даних:

1. Використання рідної мови запитів баз даних (тобто SQL)
2. Використання об'єктної моделі даних (ODM) або об'єктно-реляційної

моделі (ORM).

ODM або ORM представляють дані веб-сайту як об'єкти JavaScript, які потім відображаються на підтримуючу базу даних. Деякі ORM прив'язані до певної базі даних, тоді як інші не залежать від конкретної бази даних.

Найкращу продуктивність можна отримати за допомогою SQL або іншої мови запитів, підтримуваного базою даних. Об'єктні моделі (ODM) часто повільніше, тому що вимагають переведення об'єктів в формат бази даних, при цьому не обов'язково будуть використані найбільш ефективні запити до бази даних, враховуючи що ODM призначена для різних баз даних і повинна йти на великі компроміси в підтримці тих чи інших функцій бази даних.

Перевага застосування ORM полягає в тому, що розробники можуть зосередитися на об'єктах JavaScript, а не на семантиці бази даних - особливо, якщо потрібно працювати з різними базами даних, на одному чи різних веб-додатках. Окрім того в ORM своє місце займає валідація і перевірка даних.

1.7 Постановка завдання проектування

Проаналізувавши проблематику створення веб-додатків, також дослідивши існуючі платформи для розробки, було сформульовано наступні завдання для виконання в ході дипломного проектування. Запропонувати модуль для розробки веб-додатків на базі Node.js, що має вирішити низку проблем із платформою на початку розробки веб-додатку, запобігти в майбутньому витрати часу розробників, на реалізацію початкових системи у розробці веб-додатку. Задати новий архітектурний підхід до розробки. Спростити доступ до керування важливими модулями, що дозволить не використовувати понад важкі бібліотеки у розробці, методом створення власної системи керування, яка буде надаватись із коробки.

Для реалізації поставленої мети необхідно розв’язати наступні задачі:

1. Створити низку модулів для керування системою;
2. Розробити практичну архітектурну модель проекту;
3. Використовувати наявні функції системи, без підключення бібліотек;
4. Спростити систему станів додатку, та покращити її можливості;

5. Реалізувати підтримку швидкої розробки для скорочення часу;
6. Розробити модуль керування паралельними процесами за допомогою черг у системі процесів та поліпшення модуля роботи з процесами;
7. Протестувати роботу програмного модуля.

1.7 Висновки до розділу 1

Аналіз найпопулярніших фреймворків, що застосовуються для створення веб додатків показав, що платформа Node.js, в особі фреймворку Express, і Django керують на ринку веб-додатків. Можемо вважати, що найцікавішою платформою для розробки серверних частин додатків в наступні декілька років буде Node.js.

Порівнявши Node.js з основними його конкурентом фреймворком Django було виділено ряд переваг Node, а саме: можливість розробки на стороні сервера та клієнта та простота транспортування коду між ними, спільна мова реалізації фронтенд, бекенд, серверної частин. З використанням Node простіше масштабувати серверну частину. При одночасному підключенні до сервера десяти і більше користувачів Node працює асинхронно, тобто ставить пріоритети і розподіляє ресурси більш коректно за своїх конкурентів. Дослідження показує, що Node.js вже показав себе як багатообіцяюча технологія. Отже, остаточний вибір припадає на платформу Node і можемо впевнено стверджувати, що у найближчому майбутньому саме ця платформа й займатиме лідируючу позицію у топі популярності серед розробки веб-додатків.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Функціональні можливості програмного модуля

До функціональних вимог модуля для розробки веб-додатків належать:

1. Автоматична маршрутизація для HTTP (S), WS (S);
2. Перезавантаження серверного коду в реальному часу;
3. Контрольоване завершення роботи та перезавантаження додатку;
4. Багатопоточність для використання в ізоляції усіх процесів;
5. Обслуговування декількох портів в потоках;
6. Обслуговування статичних файлів в кеш пам’яті;
7. Файлова конфігурація модулів та додатку;
8. Модульні тестування, та тестування API;
9. Модуль контролю часу виконання, та черги запитів;
10. Модуль паралельного виконання API;
11. Балансувальник навантажень для масштабування;
12. Обслуговування даних за допомогою бази даних.

За допомогою системи автоматичної маршрутизації модуль переймає найкращу практику в реалізації навігації, адже простішого керування роутингом навіть не уявити, а підтримуючи веб-сокети на рівні з звичайними маршрутами, додаток стає ще більш простіше використовувати, та налагоджувати, адже зникає необхідність кожного разу описувати конфігурацію до кожного з маршрутів, та створювати нові конфігурації для потоку сокетів.

Автоматичним перезавантаженням файлів в системі пропонується керувати власним модулем. Існують модулі які вже мають цей фунціонал, але основною метою додатку, є простота коду, та не використання великих бібліотек для створення простих функцій. Так наприклад бібліотека nodemon що слідкує за файлами в режимі реального часу без наявних причин імпортує в свої залежності бібліотеку, що виконує прості математичні функції, яка займає досить велику кількість місця.

Обслуговуванням бази даних додатку виступатиме PostgreSQL, оскільки ця СУБД зарекомендувала себе, як найкраще розвинену і надійну.

2.2 Багатопоточність та цикл подій у веб-додатку

Припустимо, що користувач має абстрактний web-сервер, який виконує певні дії за маршрутами. Якщо з даного маршруту приходить вхідний запит, платформа читає файл, виводить інформації в консоль. Відповідно, той час, який витрачається на запит і читання файлу, сервер буде заблокований, ніякі інші вхідні запити він обробляти не зможе, виконувати інші операції - теж. Існує два варіанти вирішення проблеми:

1. Багатопоточність
2. Неблокуючий ввід та вивід

Для першого варіанту розглянемо приклад з web-сервером на Apache проти серверу на Nginx (Рис. 2.1).

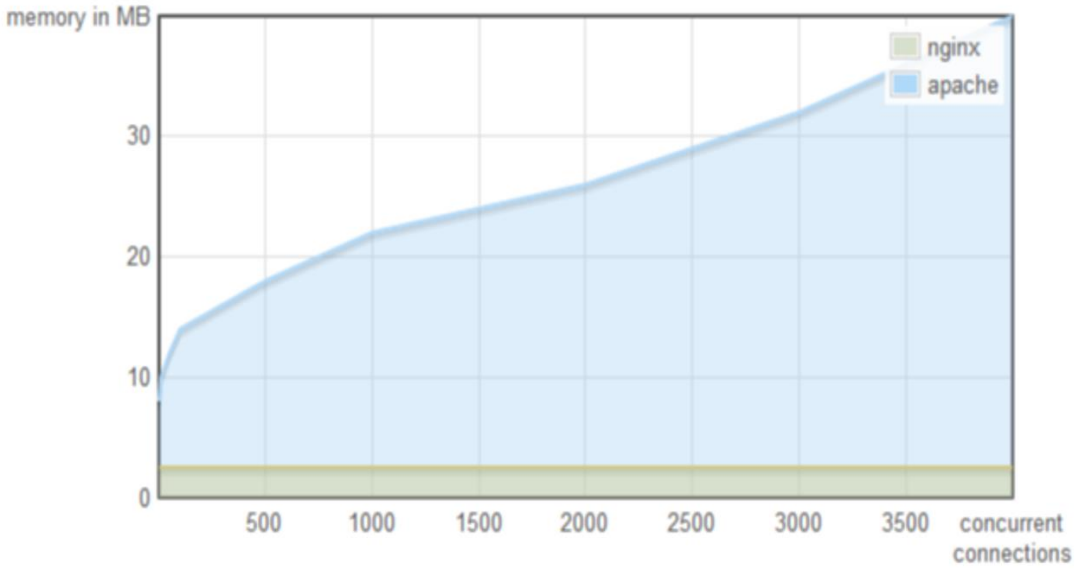

Рис. 2.1. Графік порівняння запитів на веб-серверах Apache проти Nginx

Судячи з графіку, можемо зробити висновок, що зі збільшенням кількості вхідних запитів збільшується обсяг споживаної пам'яті саме у Apache. Основною перевагою платформи Node.js є можливість керувати подіями. Неблокуючий ввід та вивід став можливим завдяки сучасним операційним системам, які надають механізм - демультиплексор подій.

Демультиплексор – це механізм, який приймає від додатка запит, реєструє його і виконує.

У верхній частині схеми (Рис. 2.2) видно, що в додатку виконуються операції, наприклад читання файлів. Для цього робиться запит в демультиплексор

подій,

в потік відправляється посилання на файл, потрібна операція і коллбек. Демультиплексор подій реєструє цей запит і повертає управління безпосередньо з додатком - таким чином, воно не блокується. Потім він виконує операції над файлом, і після цього, коли файл буде прочитаний, коллбек реєструється в черзі на виконання. Потім Event Loop поступово синхронно обробляє кожен коллбек з цієї черги. І, відповідно, повертає результат з додатком, далі якщо необхідно все робиться по колу.

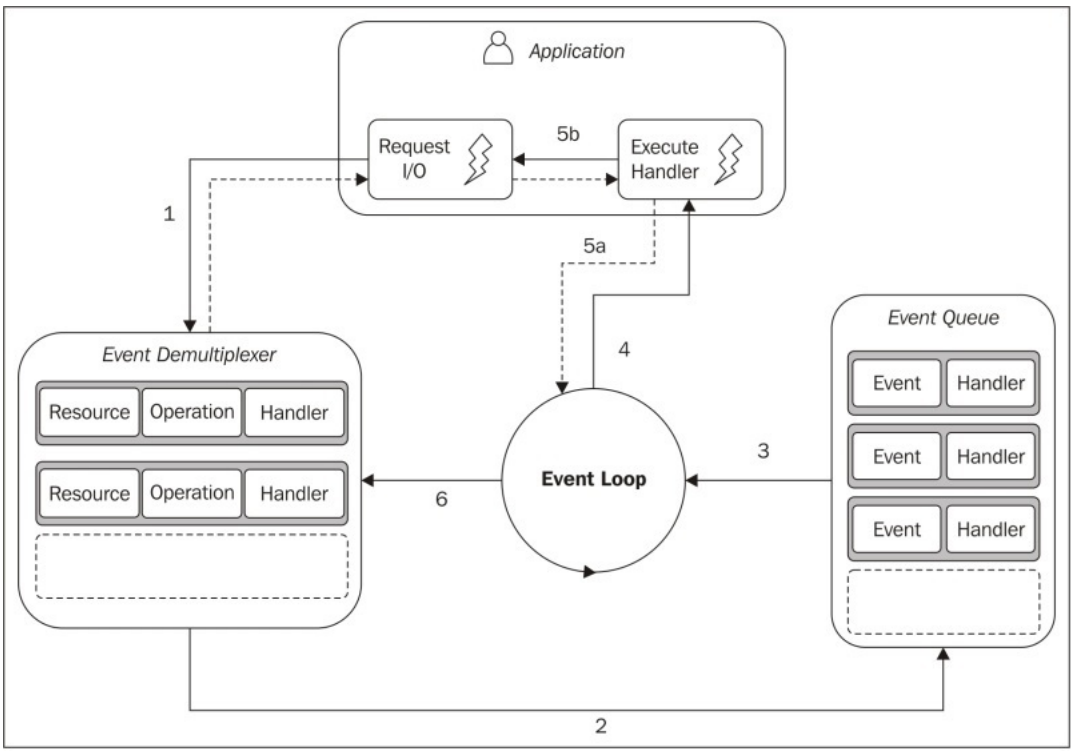

Рис. 2.2 Принцип роботи демультиплексора подій в Node.js

Таким чином, завдяки цьому неблокуючому вводу / виводу Node.js може бути асинхронним. В даному випадку неблокуюча система введення / виведення надає нам саме операційна система. Після того як така можливість з'явилася, Райан Дав, розробник Node.js, був натхненний досвідом Nginx, яка використовувала неблокуючий ввід / вивід, і вирішив створити платформу саме для розробників. Перше, що йому потрібно було зробити, це створити взаємозв’язок із демультиплексором подій, але проблема була в тому, що в кожній операційній системі демультиплексор реалізований по-різному, і йому

довелося написати обгортку, яка згодом стала називатися libuv. Бібліотека, написана на C, що надає єдиний інтерфейс роботи з цими демультиплексор подій.

Якщо розглядати список компонентів з яких складається архітектура платформи Node.js (Рис 2.3), то для компіляції коду JavaScript в машинний код використовується движ демультиплексора ок Google V8, також є Node.js Core library, де зібрані модулі для роботи з мережевими запитами, файлової системою і модуль для логування. Для загальної роботи вищевказаних модулів, були написані Node.js Bindings. Ці чотири компонента складають саму структуру Node.js, тобто сам механізм Event Loop що знаходиться в libuv.

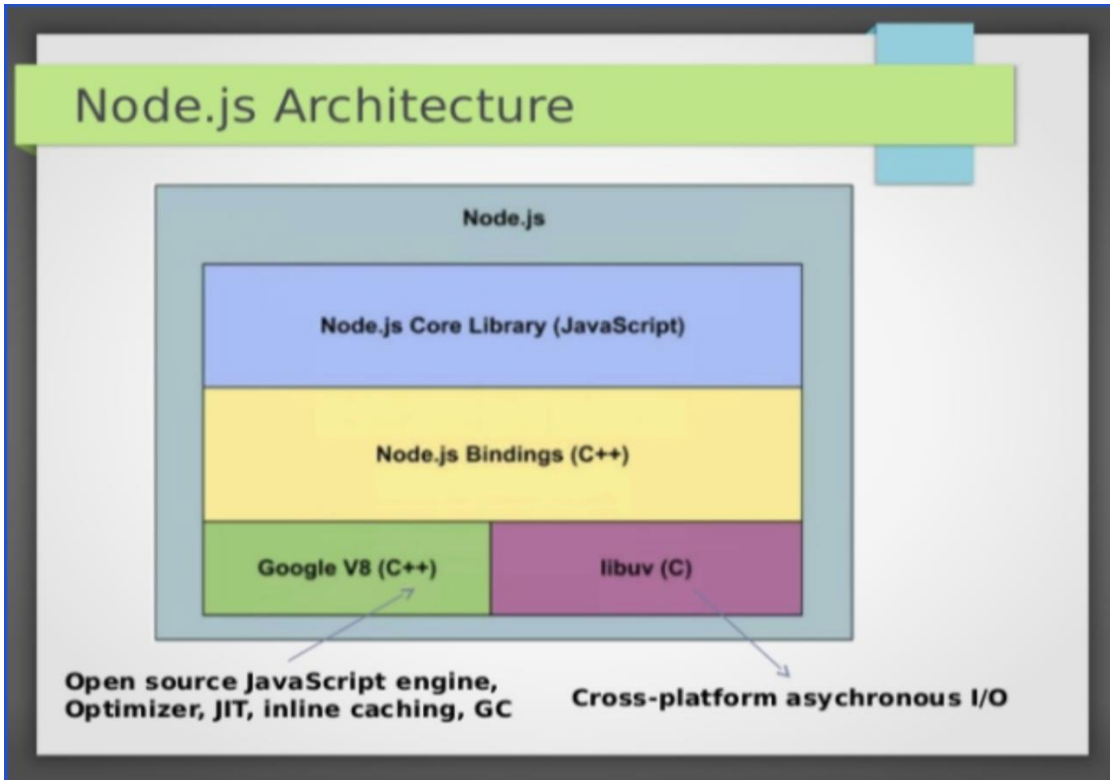

Рис. 2.3. Схема компонентів архітектури платформи Node.js

Цикл подій в Node.js поділяється на шість фаз. Перша фаза це таймери, виконується безпосередньо в Event Loop. В існуючому середовищі, існує певний стек таймерів. Розглянемо приклад роботи першої фази. Спочатку береться таймер з найменшим часом, порівнюється з поточним часу Event Loop , і, якщо настав час для виконання даного таймера, виконується його коллбек. Тут варто відзначити що в Node.js є реалізація setTimeout і setInterval. Для libuv це, одне і те ж, тільки в setInterval ще є прапор repeat. Відповідно, якщо у даного таймера існує прапор repeat, то він знову поміщається в чергу подій і потім знову обробляється.

Друга фаза це I/O-коллбек. Коли демультиплексор подій виконує дію читання будь-якого файлу і ставить виконання коллбеку в чергу, це як раз відповідає другій фазі, тобто етапу I/O-коллбек. В ньому виконуються коллбек для неблокуючого вводу / виводу, тобто ті ж функції, які використовуються після запиту в базу даних або інший ресурс або на читання чи то запис файлу.

Третя фаза – очікування, підготовка. Це внутрішні операції для коллбеків, оскільки користувач не може напряму впливати на цю фазу, існує процес process.nextTick, а його коллбек може ненавмисно бути виконаний на фазі «очікування, підготовка». Process.nextTick може спрацювати абсолютно на будь-якій фазі. Реалізованого інструменту, щоб запустити код на фазі «очікування, підготовка», в Node.js немає.

Четверта фаза – опитування, в якій виконується весь програмний код, що написаний JS. Спочатку всі запити, які робить користувач, потрапляють саме в цю фазу, і саме тут Node.js може бути заблокована. Якщо сюди потрапить якась важка операція по обчисленню, то на цьому етапі наш додаток може просто зависнути і очікувати, поки не виконається дана операція.

П’ята фаза – перевірка. У Node.js є таймер setImmediate, його коллбек виконуються на цій фазі.

Шостою, останньою фазою є коллбек подій close. Наприклад, web-сокету потрібно закрити з'єднання, на цій фазі буде викликаний callback цієї події.

Розглянувши основний механізм роботи Node.js, можемо скласти простий графік принципу його роботи (Рис. 2.4).

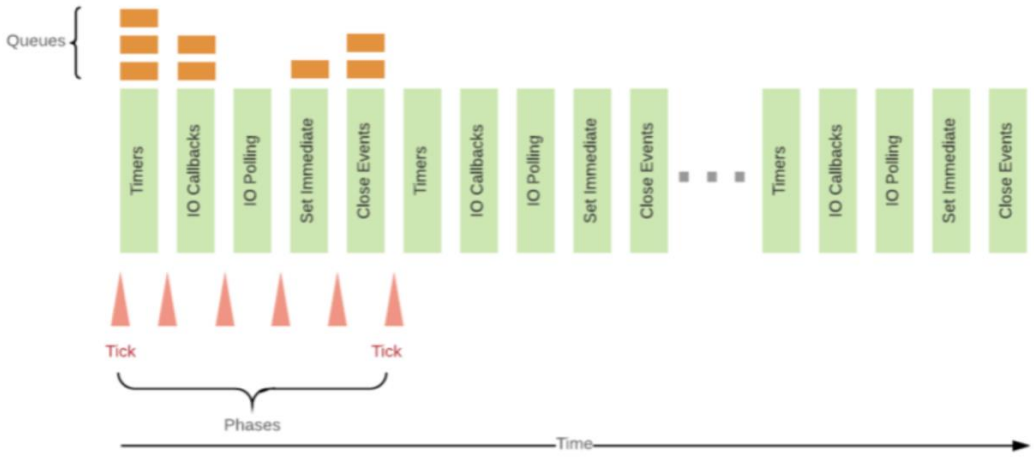

Рис. 2.4 Принцип роботи циклу подій в Node.js

Досягти багатопоточності в Node дозволяє модуль worker_threads - це пакет, який дозволяє створювати повнофункціональні багатопотокові програми.

Потоковий Воркер (thread worker) - фрагмент коду (зазвичай отримують із файлу), створений в окремому потоці. Для використання потоку в воркер потрібно імпортувати модуль worker_threads. Почнемо зі створення функції, яка допоможе створити цей воркер, а також розглянемо їх властивості.

```
type WorkerCallback = (err: any, result?: any) => any;
export function runWorker(path: string, cb: WorkerCallback, workerData: object | null = null) {
 const worker = new Worker(path, { workerData });
 worker.on('message', cb.bind(null, null));
 worker.on('error', cb);
 worker.on('exit', (exitCode) => {
  if (exitCode === 0) { return null; }
  return cb(new Error(`Worker has stopped with code ${exitCode}`));
 });return worker; }
```

Для створення потокового Воркеру необхідно створити екземпляр класу Worker. У першому аргументі вказуємо шлях до файлу, який містить код Воркер; у другому надаємо об'єкт, що містить властивість з ім'ям workerData. Це ті дані, до яких потік буде мати доступ при запуску, якщо того хоче розробник. Незалежно від того, чи використовується JavaScript або щось, що в нього транспілірується (наприклад TypeScript), шлях завжди повинен посилатися на файли з розширеннями .js або .mjs. Варто зазначити, чому використовується callback-функція замість повернення Проміс (promise), який буде передавати результат в resolve при запуску події message. Це пов'язано з можливістю потокових Воркерів відправляти багато подій message, а не тільки одне.

Зв'язок між потоками заснований на подіях. Це означає, що треба налаштувати обробники, які будуть викликатися після відправки потоком даної події. Розглянемо найбільш поширені події.

```
worker.on ( 'error', (error) => {});
```

Подія error генерується, коли всередині Воркер виникає необроблене виняток. Потім потік завершується, а помилка стає першим аргументом в callback.

worker.on ('exit', (exitCode) => {});

Подія exit генерується, коли Воркер закінчує своє виконання. Якщо process.exit () викликається всередині потоку, exitCode буде надано в callback. Якщо потік переривається за допомогою worker.terminate (), код буде 1.

worker.on ('online', () => {});

Online генерується, коли Воркер припиняє парсинг коду JavaScript і починає його виконання. Ця подія використовується нечасто, але в певних випадках воно може бути інформативним.

worker.on ('message', (data) => {});

Message генерується, коли Воркер відправляє дані в батьківський потік.

2.3 WebStorm IDE

JetBrains WebStorm – інтегроване середовище розробки на JavaScript, CSS & HTML від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA. WebStorm забезпечує автодоповнення, аналіз коду в режимі реального часу, навігацію по коду, підказки щодо рефакторингу за останніми стандартами мови, налагодження, і інтеграцію з системами управління версіями. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами, в тому числі, рефакторинг коду JavaScript, що знаходиться в різних файлах і папках проекту, а також вкладеного в HTML. Підтримується множинна вкладеність, тобто коли в документ на основі HTML було вкладено скрипт на Javascript, в який вкладено інший код HTML, всередині якого вкладено Javascript - тобто в таких конструкціях підтримується коректний рефакторинг.

Перелік основних можливостей середовища:

1. Модифікація файлів css, html, js з переглядом результатів в реальному часі;
2. Підтримка HTML5;
3. Підтримка JSDoc;
4. Підтримка Node.js;

5. Можливості Zen Coding і Emmet;
6. Налагодження коду на JavaScript;
7. Віддалене розгортання по протоколах FTP, SFTP, на монтованих мережевих дисках з можливістю автоматичної синхронізації;
8. Інтеграція з системами управління версіями: Subversion, Git, GitHub, Perforce, Mercurial, CVS що підтримуються з коробки з можливістю створення списків змін і відкладених змін;
9. Інтеграція з системами стеження за проблемами Live Edit LiveEdit - нова можливість WebStorm, що з'явилася у версії 5 і дозволяє одночасно редагувати код html, css або javascript і бачити, як результат відображається в браузері;
10. WebStorm підтримує налагодження додатків в node.js. Також підтримується повний набір функцій редагування додатків на javascript - як для виконання на сервері, так і в браузері: автодоповнення, навігація по коду, рефакторинг і перевірка на помилки. Також підтримується виведення повідомлень node.js на окрему вкладку в IDE;
11. Мови стилів LESS, Sass, SCSS і Stylus які розширюють можливості описів стилів в CSS;

2.4 Архітектура програмного модуля

Для вирішення питання з модульністю додатку, виникає потреба в ізоляції слоїв додатку. Оскільки сама архітектура складається з двох важливих речей: слоїв та зв'язків між ними, потрібно розбити додаток на шари, не допустити протікання з одного в інший, правильно організувати ієрархію шарів і зв'язку між ними.

Виникає питання, який тип архітектури обрати, та як правильно розбити додаток на шари. Є класичний трирівневий підхід: дані, логіка, уявлення, але такий підхід вважається застарілим. Проблема в тому, що основними є дані, а значить, додаток проектується в залежності від того, як дані представлені в БД, а не від того, в яких бізнес-процесах вони беруть участь. Більш сучасний підхід передбачає, що в додатку виділено доменний шар, який працює з бізнес-логікою і

є поданням реальних бізнес-процесів в коді. Однак якщо звернутись до класичної праці Еріка Еванса Domain-Driven Design, то виявимо там таку схему шарів програми: UI, рівень логіки додатку, домен, інфраструктура. Але основою додатку, спроектованого по DDD, повинен бути домен - високорівневої політики, найважливіша і найцінніша логіка, тому що під цим шаром лежить вся інфраструктура: шар доступу до даних (DAL), логування, моніторинг, тобто політики набагато більш низького рівня і меншою важливості. І в такому випадку інфраструктура знаходиться в центрі додатку, і банальна заміна модуля логування може привести до перетрушування всієї бізнес-логіки. Такий підхід не зовсім підходить до модульного додатку, тому можемо стверджувати що додаток краще проектувати як конструктор, в якому легко замінити складові частини. Один із прикладів такої реалізації - гексагональна архітектура (архітектура портів і адаптерів). Доменне ядро з усією бізнес-логікою надає порти для спілкування із зовнішнім світом. Все, що потрібно, підключається зовні через адаптери, що дозволить реалізувати модульність у розробці веб-додатку.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Системні вимоги

Для того, щоб забезпечити безперешкодне функціонування додатків на розроблюваному API було виставлено вимоги до апаратного та системного забезпечення, на якому буде використовуватися даний мобільний додаток.

Мінімальні вимоги до апаратного забезпечення користувача:

1. Процесор: Qualcomm Snapdragon 600 series;
2. Графічний процесор: Qualcomm Adreno 350;
3. Дисплей: 1080 x 720, сенсорний;
4. Оперативна пам’ять: 1 ГБ;
5. Операційна система: Android 7.1.1 Nougat;
6. Мережеві можливості: 802.11b/g/n Wi-Fi.

Так, як для роботи додатку, йому необхідно надсилати та отримувати інформацію від сервера ми, також, маємо виставити вимоги до серверного устаткування, щоб забезпечити безперервну та надійну комунікацію сервера з пристроями користувачів.

Мінімальні системні вимоги до сервера:

1. Операційна система: Windows 8\8.1\10;
2. Оперативна пам'ять: 2 ГБ;
3. Наявність програмного забезпечення: Node.js;
4. Наявність бази даних.

Також, варто зазначити, що мобільні додатки, що працюю на данном API, отримують та користується даними, що надходять із мережі, тому обов’язковою умовою їх роботи є наявність інтернет з’єднання.

3.2. Інсталяція PostgreSQL

PosthreSQL є одниз з основних модулів нашого додатку, так як нам необхідно десь зберігати дані.

Завантажити PostgreSQL можна з офіційного сайту .Нижче, на скріншоті буде наведено перелік сумісності версій PostgreSQL з платформою Windows.

PostgreSQL Version	64 Bit Windows Platforms	32 Bit Windows Platforms
13	2019, 2016	
12	2019, 2016, 2012 R2	
11	2019, 2016, 2012 R2	
10	2016, 2012 R2 & R1, 7, 8, 10	2008 R1, 7, 8, 10
9.6	2012 R2 & R1, 2008 R2, 7, 8, 10	2008 R1, 7, 8, 10

Рис. 3.1. Сумісність версій бази даних з операційною системою Windows

PostgreSQL Version	64-bit macOS Platforms
13	10.14 - 11.0
12	10.13 - 10.15
11	10.12 - 10.14
10	10.11 - 10.13
9.6	10.10 - 10.12

Рис. 3.2. Сумісність версій бази даних з операційною системою macOS

Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
13.3	N/A	N/A	Download	Download	N/A
12.7	N/A	N/A	Download	Download	N/A
11.12	N/A	N/A	Download	Download	N/A
10.17	Download	Download	Download	Download	Download
9.6.22	Download	Download	Download	Download	Download
9.5.25 (Not Supported)	Download	Download	Download	Download	Download
9.4.26 (Not Supported)	Download	Download	Download	Download	Download
9.3.25 (Not Supported)	Download	Download	Download	Download	Download

Рис. 3.3. Сторінка завантаження

Далі нам необхідно обрати потрібну нам версію.

Розглянемо встановлення PostgreSQL на прикладі операційної системи Windows.

Для встановлення нам необхідно виконати наступні кроки:

- Запуск завантаженої програми встановлення PostgreSQL

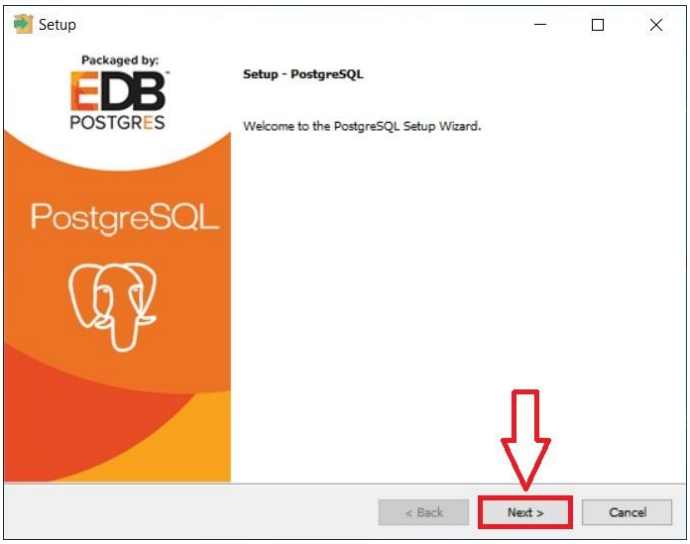

Рис. 3.4. Стартова сторінка

- Вказуємо каталог для установки PostgreSQL

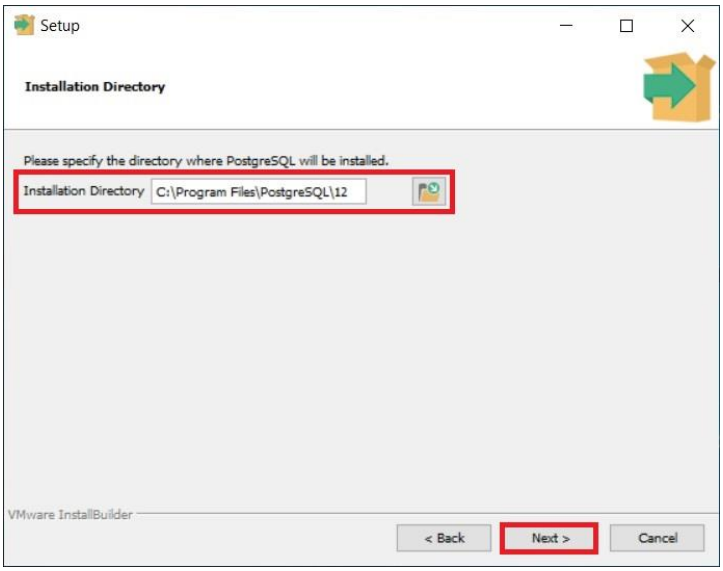

Рис. 3.5. Вікно з вибором цільового каталогу

- Вибираємо компоненти для установки

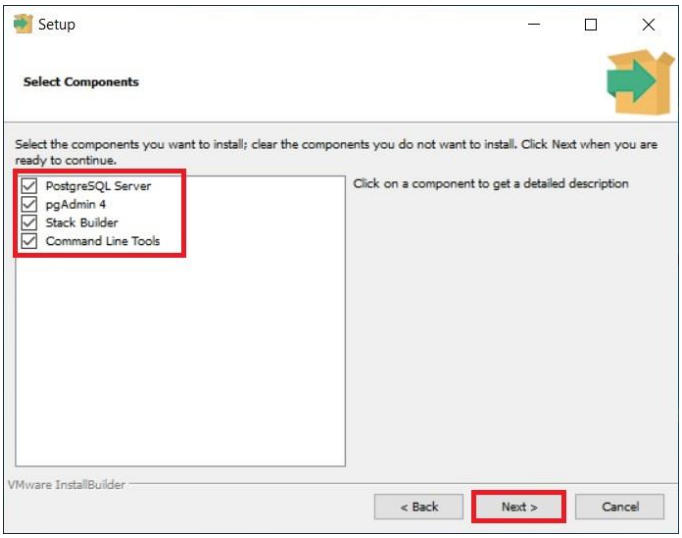

Рис. 3.6. Вікно з вибором необхідних компонентів до встановлення

- Вказуємо каталог для зберігання файлів баз даних

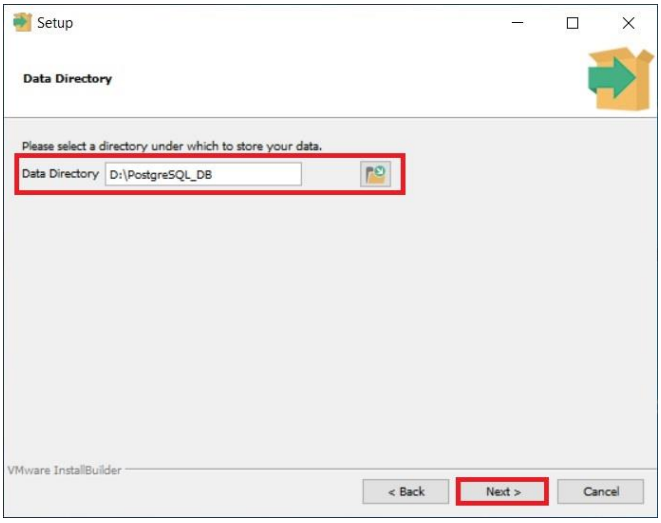

Рис. 3.7. Шлях до каталогу, де за замовчуванням будуть зберігатись файли баз даних

На скріншоті 3.7 вказано запропоновану інсталятором теку, в нашому сервісі файли бази даних зберігаються за іншою адресою.

- Задаємо пароль для системного користувача postgres

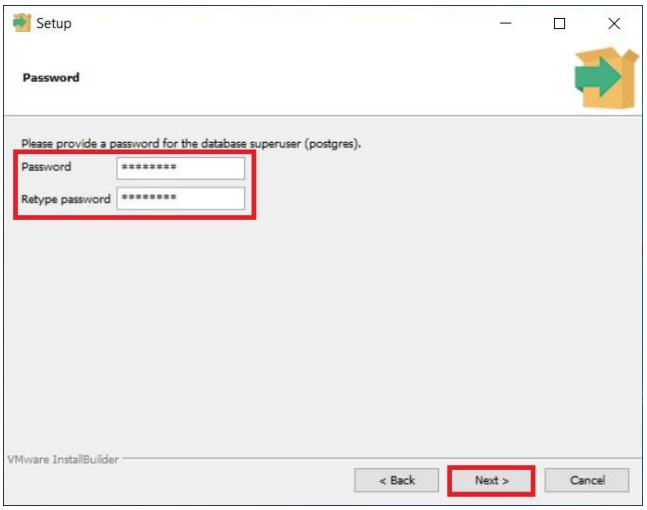

Рис. 3.8. Вікно з полями для встановлення паролю

- Вказуємо порт для сервера PostgreSQL

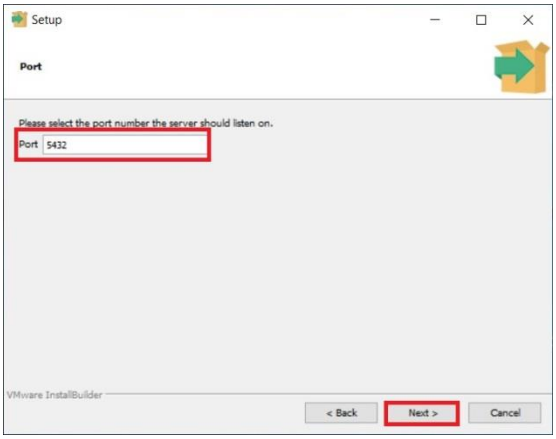

Рис. 3.9. Вікно з полем для задання номеру порту сервера

- Вказуємо кодування даних в базі

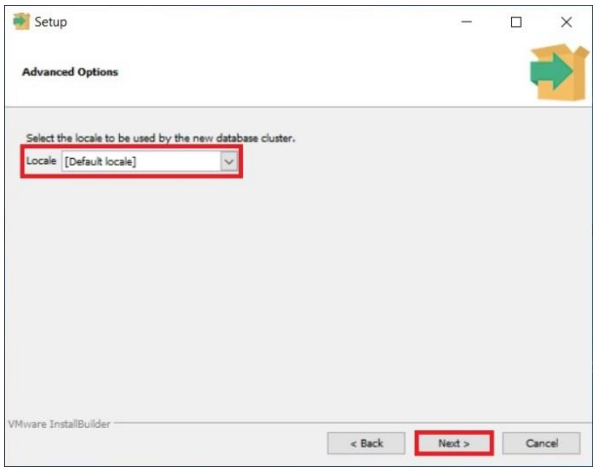

Рис. 3.10. Вікно з вибором кодування бази даних

- Перевірка параметрів установки PostgreSQL

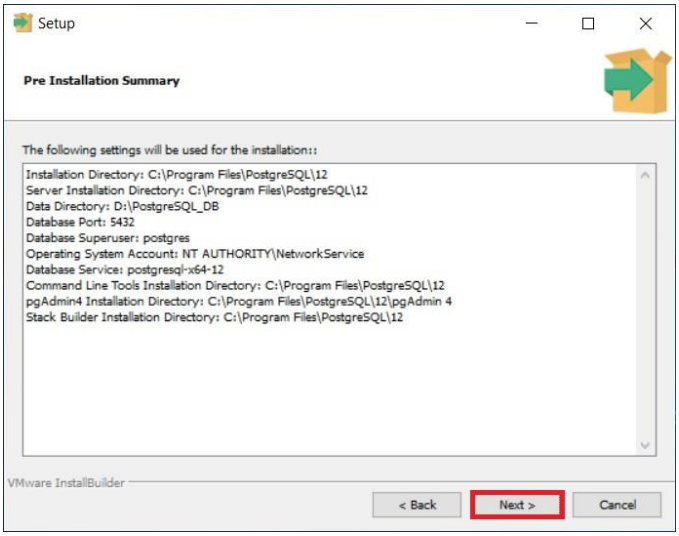

Рис. 3.11. Вікно з заданими параметрами для

- Запуск процесу установки

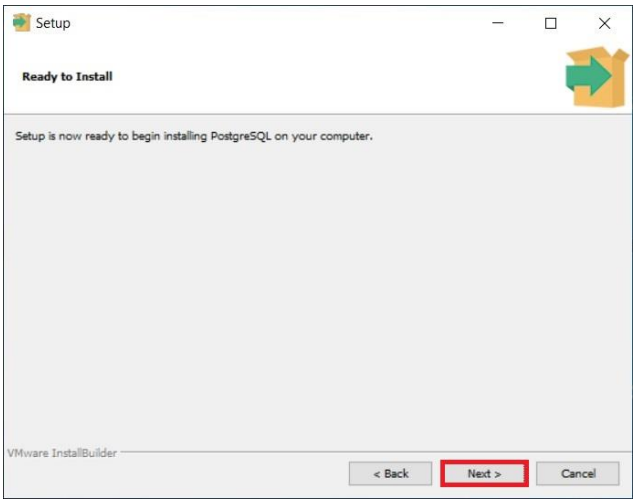

Рис. 3.12. Вікно з вибором цільового каталогу

Процес встановлення може зайняти декілька хвилин, після чого PostgreSQL буде успішно встановлено на вашу систему.

3.3. Реалізація сервіс API

Було розроблено сервіс, що дозволяє одночасно обслуговувати велику кількість подібних додатків (додатків, що використовуватимусь інформацію стосовну отримання водійського посвідчення) та їх користувачів. Для покриття якомога більшої кількості користувачів було прийнято рішення розширити додаток та додати можливість його використання також поза межами України.

Для демонстрації роботи сервісу було розроблено примітивний веб-сайт що надсилає запити до сервісу та отримує дані взамін. Необхідно створити лише базові функції сайту, без надлишкової уваги до дизайну задля економії часу розробки.

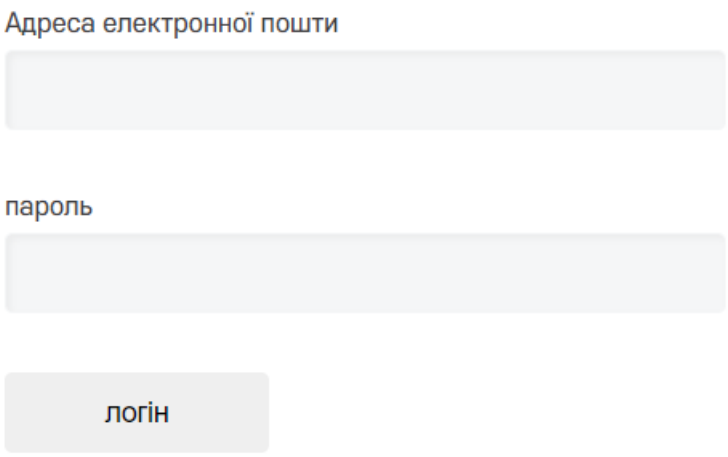

Рис. 3.1. Форма логіну

Форма реєстрації має подібний вигляд, відмінності полягають лише в кнопці логіну та реєстрації відповідно.

Код з файлу *signin.js* що забезпечує можливість користувача залогінитись:

```
({access: 'public',
  method: async ({ login, password }) => {
    const user = await api.auth.provider.getUser(login);
    const hash = user ? user.password : undefined;
    const valid = await metarhia.metautil.validatePassword(password, hash);
    if (!user || !valid) throw new Error('Incorrect login or password');
    console.log(`Logged user: ${login}`);
    const token = await context.client.startSession(user.accountId);
    return { status: 'logged', token };},});
```

Наступним буде продемонстровано код з register.js. З назви файлу ми можемо зрозуміти, що він відповідає за реєстрацію нових користувачів.

```
({access: 'public',
  method: async ({ login, password, fullName }) => {
    const hash = await node_threads.metautil.hashPassword(password);
    await api.auth.provider.registerUser(login, hash, fullName);
    const token = await context.client.startSession();
    return { status: 'success', token };},});
```

Також, наш сервіс передбачає можливість відновлення сесії. Це описано в файлі *restore.js*. Код цього файлу можете переглянути нижче.

```
({access: 'public',
  method: async ({ token }) => {
    const success = await context.client.restoreSession(token);
    const status = success ? 'logged' : 'not logged';
    return { status };},});
```

До наданого коду не додано коментарі з описом їх роботи через інтуїтивно зрозумілі функції, що випливають з назви файлу та методів.

Загальне керуванням авторизації та реєстрації користувача описане у файлі *provider.js*. Він більший за попередні файли, тому буде додано коментарі то його методів.

```
// створення точена за яким користувач отримає можливість надсилати запити та отримувати дані у відповідь
  ({generateToken() {
    const { characters, secret, length } = config.sessions;
    return node_threads.metautil.generateToken(secret, characters, length); },

// збереження сесії кристувача
  saveSession(token, data) {
    domain.db.update('Session', { data: JSON.stringify(data) }, { token });},

// початок сесії користувача
  startSession(token, data, fields = {}) {
    const record = { token, data: JSON.stringify(data), ...fields };
    domain.db.insert('Session', record); },

// відновлення сесії користувача
```

```
  async restoreSession(token) {
    const record = await domain.db.row('Session', ['data'], { token });
    if (record && record.data) return record.data;
    return null;},

// видалення сесії
  deleteSession(token) {
    domain.db.delete('Session', { token }); },
  async registerUser(login, password) {
    return domain.db.insert('Account', { login, password }); },

  async getUser(login) {
    return domain.db.row('Account', { login });},});
```

Приклад інтерфейсу тестового додатку для проходження тесту на знання правил дорожнього руху.

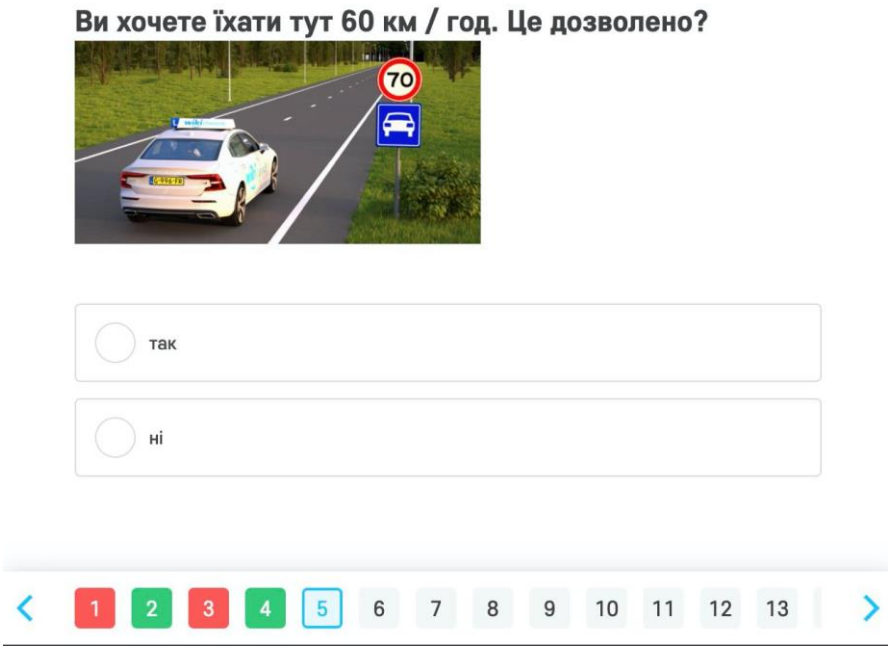

Рис. 3.2. Скріншот тесту на знання правил ПДР

На малюнку 3.2. ми можемо бачити що наш сервіс відпрацьовує коректно, надаючи дані для веб-сайту, що їх виводить на екран.

Сервіс формує відповіді на запити веб-сайту. Далі буде наведено приклади таких відповідей.

При запиті списку питань сервіс повертає їх в форматі json. Приклад відповіді надано нижче.

'''[{"question":3,

"answer":{

"id":10,

"title":"110 км/год",

"ordering":2,

"is_right":true},

"choices":[

{"id":8,

"title":"120 км/год",

"ordering":0,

"is_right":false},

{"id":9,

"title":"90 км/год",

"ordering":1,

"is_right":false},

{"id":10,

"title":"110 км/год",

"ordering":2,

"is_right":true}],

"comment":"На автомобільній дорозі з окремими проїзними частинами, що відокремлені одна від одної розділювальною смугою — не більше 110 км/год.",},]'''

З даного прикладу ми можемо побачити, що сервіс надсилає масив питань, що включають в себе унікальний ідентифікатор питання, відповідь (правильний варіант відповіді для подальшого його порівняння з обраною користувачем відповіддю) а також список можливих відповідей. Також, з сервісу, разом з відповіддю приходить коментар, що описує правило, яке обумовлює обрання

правильної відповіді.

Варіант відповіді в свою чергу включає в себе унікальний ідентифікатор (нумерація продовжується від питання до питання, тобто якщо в першого питання було два варіанти відповіді (ідентифікатори ‘0’ та ‘1’ відповідно, то в другого питання нумерація варіантів відповіді почнеться з ‘2’), назва (текст, що показуватиметься користувачу як відповідь), порядковий номер та мулеве значення що показує чи ця відповідь є правильною.

На наступному скріншоті продемонстровано відповіді сервісу на дії користувача. Було пройдено тест на знання правил дорожнього руху.

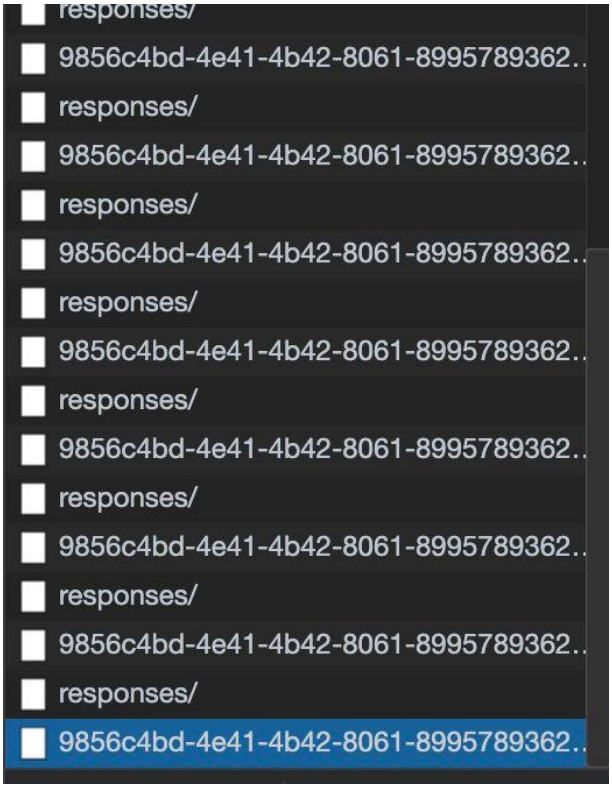

Рис. 3.3. Вкладка Network браузеру

Як ми можемо бачити з малюнку, на кожну відповідь користувача на питання ми отримуємо від сервісу два набори даних. В першому з них (responses/) отримуємо дані подібні до вищеописаних даних, що отримуємо на початку, щодо питань та відповідей на них. Формат даних ідентичний. Єдина відмінність полягає в тому, що приходять дані не по масиву питань, а лише щодо одного, на яке відповідав користувач.

Друга відповідь сервісу повертає дані про загальний стан проходження тесту: кількість правильних та не правильних відповідей.

Приклад цієї відповіді можна побачити на малюнку 3.4.

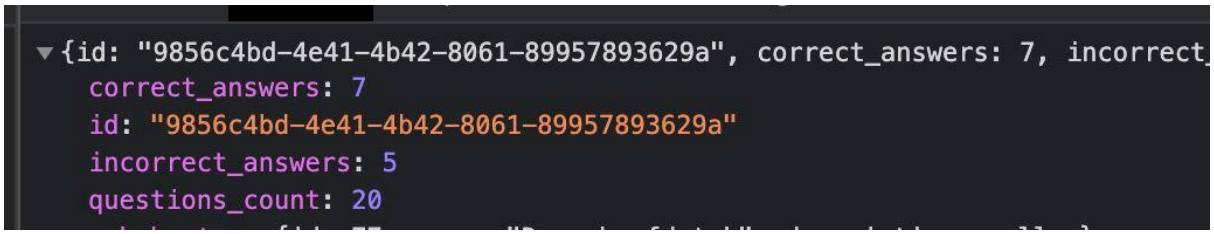

Рис. 3.4. Дані про поточний стан тесту