МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО

"____"_____2021 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: " Візуалізація статистики web-сервісу пошуку роботи для користувачів "

Виконавець: Резаєв Ярослав Олегович

Керівник: к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер: _____ Ігор РАЙЧЕВ

Київ - 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра Комп'ютерних інформаційних технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Аліна САВЧЕНКО
«____»_____2021р.

ЗАВДАННЯ

на виконання дипломної роботи студента

Резаєва Ярослава Олеговича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Візуалізація статистики web-сервісу пошуку роботи для користувачів» затверджена наказом ректора від 12.10.2021 за № 2228/ст.

2. Термін виконання роботи: з 12.10.2021 по 31.12.2021.

3. Вихідні дані до роботи: мови програмування – PHP, JavaScript, back end фреймворк – Laravel, технології front end частини – HTML 5, CSS 3, Blade, JavaScript, jQuery, БД – Amazon DynamoDB, сховище – Amazon S3, CDN – Amazon CloudFront, веб-сервер – NGINX, середовище розробки – Visual Studio Code.

4. Зміст пояснювальної записки: вступ, аналітичний огляд і постановка завдання, архітектура, структура та особливості реалізації сервісу, аналіз показників та характеристик отриманого сервісу, висновки.

5. Перелік обов'язкового ілюстративного матеріалу: схеми середовища розгортання сервісу та архітектури для версії production release, робота сервісу на різних пристроях, порівняння часу повного завантаження сторінки Dashboard.

6. Календарний план-графік

№ п/п	***Завдання***	***Термін виконання***	***Підпис керівника***
1.	Проаналізувати літературу та джерела за темою дипломної роботи	12.10.21 – 14.10.21р.	
2.	Розроблення та затвердження плану дипломної роботи	15.10.21 – 17.10.21р.	
3.	Провести консультації з науковим керівником щодо створення першого розділу	18.10.21 – 19.10.21р.	
4.	Розробка розділу 1	20.10.21 – 29.10.21р.	
5.	Розробка розділу 2	30.10.21 – 10.11.21р.	
6.	Розробка розділу 3	11.11.21 – 21.11.21р.	
7.	Висновки та оформлення пояснювальної записки дипломної роботи	28.11.21 – 10.12.21р.	
8.	Підписання необхідних документів у встановленому порядку	11.12.21 –19.12.21р.	
9.	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломної роботи	20.12.21 – 21.12.21р.	

7. Дата видачі завдання: 12.10.2021р.

Керівник дипломної роботи _____ Тетяна ХОЛЯВКІНА
(підпис керівника)

Завдання прийняв до виконання _____ Ярослав РЕЗАЄВ
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Візуалізація статистики web-сервісу пошуку роботи для користувачів» викладена на 112 сторінках, містить 40 рисунків в основній частині та 2 рис. в додатках, 2 додатки, 40 літературних джерел.

Мета роботи: реалізувати систему візуалізації статистики сервісу пошуку роботи для користувачів, яке матиме достатньо гнучку архітектуру, можливість масштабування, зручний користувацький інтерфейс для отримання позитивного користувацького досвіду, а також відповідатиме вимогам продуктивності та надійності.

Предмет дослідження: візуалізація статистики web-сервісу пошуку роботи для користувачів.

Об'єкт дослідження: програмна система візуалізації статистики web-сервісу.

Ключові слова: ПРОГРАМНА СИСТЕМА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВІЗУАЛІЗАЦІЯ СТАТИСТИКИ, СЕРВІС ПОШУКУ РОБОТИ, WEB-СЕРВІС, WEB-ДОДАТОК, LARAVEL, PHP, BOOTSTRAP, UI, UX, КОМБІНОВАНА АРХІТЕКТУРА, ХМАРНІ СЕРВІСИ, МАСШТАБОВАНІСТЬ, АДАПТИВНІСТЬ, ЕФЕКТИВНІСТЬ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ 7
ВСТУП 8
РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД І ПОСТАНОВКА ЗАВДАННЯ 11
1.1. Дослідження предметної області 11
1.2. Поняття статистики 13
1.3. Візуалізація як ефективний спосіб представлення інформації 14
1.4. Типи програмного забезпечення за характером або областю виконання 17
1.5. Переваги та недоліки web-сервісів та web-додатків як типу реалізації програмного забезпечення у порівнянні з нативними додатками 18
1.6. Популярні мови програмування та back-end фреймворки для реалізації програмного забезпечення у вигляді web-сервісів 20
1.7. Front-end фреймворки 24
1.7.1. Materialize 25
1.7.2. Bootstrap 25
1.7.3. Material UI 26
1.7.4. React 27
1.7.5. Vue.js 28
1.7.6. Angular 28
Висновок до розділу 1 30
РОЗДІЛ 2 АРХІТЕКТУРА, СТРУКТУРА ТА ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ 31
2.1. Значення архітектури програмної системи та її складових 31
2.2. Загальна архітектура програмної системи та її складові 32
2.3. Laravel та патерн MVC 35
2.4. База даних 39
2.4.1. Робота з DynamoDB в Laravel 40
2.4.2. Таблиця програмної системи 41
2.5. Сховище 44
2.6. Мережа доставки контенту 46
2.6.1. Послідовність кроків роботи Amazon CloudFront при доставці вмісту 48

2.6.2. Invalidations 50
2.7. Балансувальник навантаження 51
2.8. Користувацький інтерфейс програмної системи 55
2.8.1. Використання фреймворку Bootstrap 57
2.8.2. Використання інструменту для побудови графіків Chart.js 59
2.8.3. Використання бібліотеки для побудови графіків ApexCharts.js 60
2.8.4. Використання інструменту для побудови інфографіки Peity.js 62
2.8.5. Сторінка Dashboard 63
2.8.6. Сторінка Statistics 65
Висновок до розділу 2 68
РОЗДІЛ 3 АНАЛІЗ ПОКАЗНИКІВ ТА ХАРАКТЕРИСТИК ОТРИМАНОЇ ПРОГРАМНОЇ СИСТЕМИ 69
3.1. Аналіз безпеки програмної системи 69
3.2. Аналіз ефективності архітектурних рішень 71
3.3. Аналіз ефективності та масштабованості програмної системи 72
3.4. Аналіз ефективності використання та масштабованості NoSQL БД Amazon DynamoDB 73
3.6. Аналіз юзабіліті користувацького інтерфейсу 75
3.7. Аналіз користувацького досвіду 80
3.8. Порівняння з аналогічними сервісами 81
Висновок до розділу 3 88
ВИСНОВКИ 89
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ 91
ДОДАТКИ 96
Додаток А 96
Додаток Б 111

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

HTTP	протокол передачі гіпер-текстових документів;
HTTPS	розширення протоколу HTTP для підтримки шифрування;
CDN	мережа доставки контенту;
ПК	персональний комп’ютер;
PWA	прогресивний веб-додаток;
UI	користувацький інтерфейс;
UX	користувацький досвід;
MVP	мінімально життєздатний продукт;
AWS	Amazon Web Services;
S3	Simple Storage Service;
GSI	глобальний вторинний індекс;
ORM	об'єктно-реляційна проекція;
EC2	Elastic Compute Cloud;
KPI	Key Performance Indicator (ключовий показник ефективності);
HR	Human Resources (людські ресурси).

ВСТУП

Статистика відіграє важливу роль в житті кожної людини. Галузь статистики використовує науку про вивчення даних в якості основи. Знання статистики допомагають використовувати належні методи для збору даних, застосовувати правильний аналіз та ефективно представляти результати. Статистика є вирішальним процесом у тому, як ми робимо відкриття в науці, приймаємо рішення на основі даних та робимо прогнози. Статистика дозволяє набагато глибше зрозуміти тему. Повертаючись до ефективного представлення результатів, варто згадати поняття візуалізації.

Візуалізація — це будь-яка техніка для створення зображень, діаграм або анімації для передачі повідомлення.

Загальна ідея використання візуалізації для подання інформації не є новою, візуалізація використовувалась на картах, наукових кресленнях та графіках представлення даних більше тисячі років. Візуалізація за допомогою зорових образів була ефективним способом передачі як абстрактних, так і конкретних ідей ще на зорі людства. Прикладами з історії є наскельний живопис, єгипетські ієрогліфи, грецька геометрія, революційні методи технічного малювання Леонардо да Вінчі для технічних і наукових цілей, «Географія Птолемея» (II ст.), карта Китаю (1137 р.), карта Мінарда (1861 р.) та ін. Основна частина концепцій, визначених при розробці цих зображень прямо переноситься на візуалізацію за допомогою комп’ютерної техніки.

На сьогодні сфера застосування візуалізації постійно розширюється в освіті, науці, техніці, медицині, інтерактивних мультимедіа тощо. Область комп’ютерної графіки є типовим прикладом такого застосування. Сам винахід комп’ютерної графіки (у тому числі і комп’ютерної 3D-графіки) може бути найважливішим кроком у розвитку візуалізації з моменту винаходу центральної перспективи в епоху Відродження. Свій внесок зробив і розвиток анімації.

Однак, варто зазначити що на ранніх етапах, відсутність достатньої потужності графічних систем часто обмежувала її корисність.

Якщо зображення краще тисячі слів, то інтерактивний інтерфейс – мільйона (або тисячі зображень). Адже користувацький інтерфейс є не тільки відображенням певного способу взаємодії людини з машиною, він є основою користувацького досвіду.

Поява онлайн-платформ для пошуку роботи та найму персоналу змінила ландшафт сфери найму та працевлаштування. Так, люди все ще передають фізичні копії своїх резюме потенційним роботодавцям, але більшість можливостей зараз вже розміщується в мережі. Завдяки появі LinkedIn, таких веб-сайтів з працевлаштування, як Indeed, Monster, AngelList та рекрутерських сервісів на зразок Pitch, існують різні способи того, як люди зараз можуть шукати та знаходити роботу.

Безліч варіантів, доступних для кандидатів, означає, що вони можуть практично порівнювати роботу та зважувати свою ідеальну роль на основі таких змінних, як заробітна плата, місцезнаходження, виплати та прогрес – інформація, яка частіше доступна в інтернеті. За день пошуку роботи кандидат може претендувати на десятки ролей, оцінюючи та порівнюючи найкращі варіанти натисканням кнопки. А візуалізація статистики користувача дає змогу наочно відобразити результативність певних дій за визначені проміжки часу, ефективно підбивати підсумки та зручно виділяти ключові показники та важливі дані, що дає значні переваги при оцінці тих чи інших факторів.

Те, що технології змінюють набір персоналу – вже встановлений факт. Ці технології просто створили інноваційну та нову динаміку не лише в процесі найму та відбору, але й у процесі пошуку роботи. Компанії отримують доступ до більшого, диверсифікованого та висококваліфікованого фонду талантів, який може допомогти їм досягти своїх організаційних цілей, тоді як шукачі роботи отримують користь від більш ефективного процесу найму.

Популярність та затребуваність сервісів пошуку роботи стабільно зростають на фоні постійного збільшення потреби у працевлаштуванні населення планети, а ситуація з карантинними заходами у всьому світі, введеними у зв’язку з пандемією COVID-19 спричинила стрімкий ріст рівня безробіття та відповідно попиту на

сервіси пошуку роботи і рекордні темпи зростання кількості користувачів таких сервісів.

Використання візуалізованих даних сприяє більшому залученню користувачів та дозволяє довше утримувати їх увагу. А утримання уваги клієнтів є однією з головних складових управління стимулами купівельної активності і в підсумку – доходами бізнесу. В результаті різних досліджень було встановлено що додавання інфографіки та візуального вмісту дозволяє збільшити відвідуваність web-сайтів та забезпечити зростання числа користувачів. Таким чином, реалізація візуалізації статистики web-сервісу пошуку роботи для користувачів дозволяє не лише покращити користувацький досвід та створити більш інноваційний, зручніший для клієнта продукт, а й збільшити прибутки та отримати конкурентну перевагу бізнесу.

Враховуючи всі вищезазначені факти, можна дійти висновку, що обрана тема має великий потенціал та є актуальною.

Відповідно, метою дипломної роботи є реалізація візуалізації статистики web-сервісу пошуку роботи для користувачів, яке матиме достатньо гнучку архітектуру, можливість масштабування, зручний користувацький інтерфейс для отримання позитивного користувацького досвіду, а також відповідатиме вимогам продуктивності та надійності.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД І ПОСТАНОВКА ЗАВДАННЯ

1.1. Дослідження предметної області

Тема дипломної роботи «Візуалізація статистики web-сервісу пошуку роботи для користувачів» включає в себе ряд таких понять: програмне забезпечення, статистика, візуалізація, сервіс пошуку роботи. Тому для того щоб перейти до описання архітектури, структури та особливостей реалізації програмної системи візуалізації статистики, спершу потрібно визначити та дослідити ці поняття.

Сервіс пошуку роботи в даному випадку реалізований як веб-сервіс і представляє собою сайт з пошуку роботи з відповідною інфраструктурою. Мета сервісу – допомагати пошукачам та роботодавцям знаходити один одного якомога швидше, простіше та з найменшим стресом.

Для пошукачів — це можливість швидко знайти бажану роботу. Типовий алгоритм дій пошукача при використанні сервісу зображено на рис. 1.1.

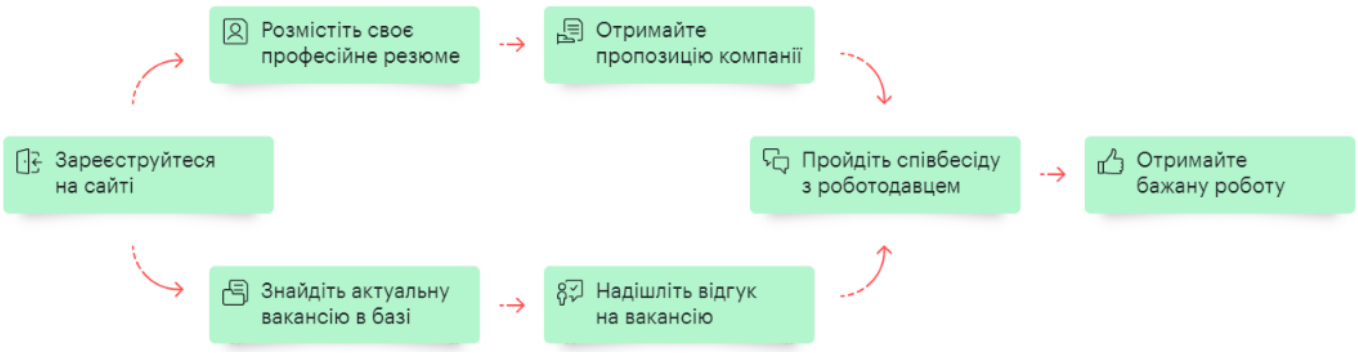

Рис. 1.1. Алгоритм дій пошукача при використанні сервісу пошуку роботи

Для роботодавців — це ефективний спосіб знайти потрібного кандидата. Типовий алгоритм дій роботодавця при використанні сервісу зображено на рис. 1.2.

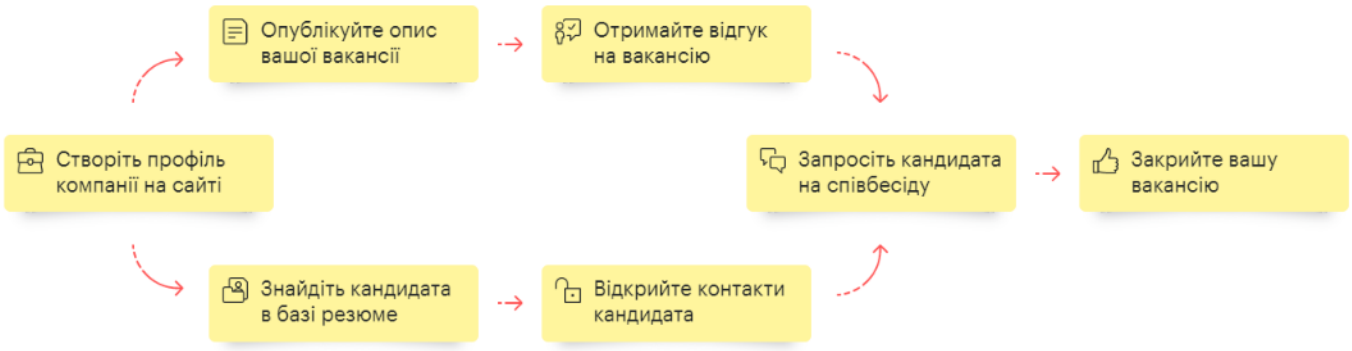

Рис. 1.2. Алгоритм дій роботодавця при використанні сервісу пошуку роботи

Відповідно для роботодавців деякими з основних функцій сервісу є:

- Пошук кандидата – розширений пошук роботи по базі резюме з можливістю поставити необхідні критерії відбору – категорію, досвід, навички, місто, бажаний рівень заробітної плати, тощо;
- Публікація опису вакансії – складання опису, розміщення опису на сайті, сервіси з управління описами вакансій;
- Підписка на розсилку нових резюме – оперативна інформація щодо нових резюме згідно з вимогами роботодавця до кандидата.

А для кандидата (пошукача) деякі з основних функцій представлені так:

- Пошук вакансій – розширений пошук роботи по базі вакансій з можливістю поставити необхідні критерії відбору – компанію-роботодавця, місто, категорію, рівень заробітної плати, тощо;
- Публікація резюме – складання резюме, розміщення резюме на сайті, сервіси з управління резюме пошукача;
- Підписка на розсилку вакансій – оперативна інформація щодо нових вакансій згідно з вимогами пошукача до бажаної посади.

В наступних підрозділах розглянемо такі поняття як статистика, візуалізація, типи програмного забезпечення за характером або областю виконання, переваги та недоліки веб-сервісів та веб-додатків як типу реалізації програмного забезпечення у порівнянні з нативними додатками, популярні мови програмування та back-end фреймворки для реалізації програмного забезпечення у вигляді веб-сервісів та front-end фреймворки.

1.2. Поняття статистики

Статистика (від лат. «status» – положення, стан явищ) – це дисципліна та технологія екстрагування, інтерпретації, осмислення та усвідомлення інформації на основі даних, часто в умовах невизначеності. Цю інформацію потім можна використовувати для прийняття рішень та дій, починаючи від медичної діагностики, закінчуючи корпоративним або державним плануванням, фінансуванням та ін.

Статистика включає збір, опис, аналіз і виведення висновків з кількісних даних. Математичні теорії статистики значною мірою спираються на диференціальне та інтегральне числення, лінійну алгебру та теорію ймовірностей. Статисти, люди, які займаються статистикою, зацікавлені визначенням того, як зробити надійні висновки про великі групи та загальні події з поведінки та інших спостережуваних характеристик малих вибірок. Оскільки в багатьох випадках збір вичерпних даних про всю сукупність є занадто дорогим, важким або зовсім неможливим, статистика починається з вибірки, яку можна зручно дослідити за прийнятною ціною. Ці невеликі зразки представляють частину великої групи або обмежену кількість випадків загального явища.

Дві основні області статистики відомі як описова статистика, яка описує властивості даних вибірки та сукупності, та індуктивна статистика, яка використовує ці властивості для перевірки гіпотез і висновків.

Описова статистика використовується для опису або підсумовування характеристик вибірки або набору даних, таких як середнє значення змінної, стандартне відхилення або частота. В індуктивній статистиці, на відміну від

описової, використовується будь-яка кількість методів, щоб зв’язати змінні в наборі даних одна з одною, наприклад, використовуючи кореляційний або регресійний аналіз. Потім їх можна використовувати для оцінки прогнозів або формування висновку про причинно-наслідкові зв’язки.

Деякі поширені статистичні інструменти та процедури включають наступне:

- описова статистика;
- індуктивна статистика;
- середнє значення;
- дисперсія;
- скіс;
- куртоз;
- лінійний регресійний аналіз;
- дисперсійний аналіз;
- моделі Logit/Probit;
- перевірка нульової гіпотези.

Статистика використовується практично в усіх наукових дисциплінах, таких як фізичні та соціальні науки, а також у бізнесі, гуманітарних науках, державному управлінні та виробництві.

1.3. Візуалізація як ефективний спосіб представлення інформації

Візуалізація – це загальна назва прийомів представлення числової інформації або фізичного явища у вигляді, зручному для зорового спостереження та аналізу, також процес побудови графічного образу даних, що допомагає у процесі загального аналізу даних вбачати аномалії, структури.

Використання візуалізованої інформації зросло на 680% у літературі з 1990 року [1], на 9900% в Інтернеті з 2007 року [2] та на 142% у газетах в проміжок між 1985 та 1994 роками [3].

Популярність візуального представлення інформації пояснюється декількома чинниками, серед яких чи не найважливішими є фізіологічні особливості людини:

- приблизно 70% усіх сенсорних рецепторів знаходяться в очах [4];
- майже 50% людського мозку задіяно у обробці візуальної інформації [4];
- людина може визначити суть візуальної сцени менш ніж за 1/10 секунди [5].

Потрібно лише 150 мс для обробки символу та ще 100 мс, щоб надати йому значення [6, 7]. Розглянемо наочний приклад використання візуалізації у дорожніх знаках (рис. 1.3).

Рис. 1.3. Візуалізація дорожнього знаку «Низьколітаючі літаки»

Візуалізація допомагає вирішувати такі глобальні питання як інформаційне перевантаження, оскільки людина на сьогодні:

1. Отримує в 5 разів більше інформації, ніж в 1986 році [8];
2. Споживає 34 Гб інформації (еквівалент 100500 слів) поза роботою за середньостатистичний день [9];
3. Читає лише 28% слів на відвідуваних веб-сайтах [10].

Інфографіка дозволяє боротися з інформаційним перевантаженням завдяки більшому залученню. В дослідженні корпорації Xerox виявили що кольорові зображення збільшують коло потенційних читачів на 80% [11]. В іншому дослідженні сприйняття етикеток лікарських засобів було встановлено що для

етикеток лише з текстовою інформацією рівень сприйняття становив 70%, в той час як для етикеток з текстом, що супроводжувався ілюстраціями – 95% [12].

В цілому, люди, які дотримуються вказівок із текстом та ілюстраціями справляються із поставленими завданнями на 323% краще ніж люди, які дотримуються вказівок без ілюстрацій [13].

Візуалізована інформація виглядає більш переконливо – дослідження, проведене в Вортонській бізнес-школі, показало, що:

- 50% аудиторії було переконано суто словесною презентацією;
- 67% аудиторії було переконано словесною презентацією, що мала супровідний візуальний матеріал.

Додавання зображень сканування мозку та згадування когнітивної нейробіології роблять людей більш схильними вірити в те, що вони читають [14].

Візуалізовану інформацію також легше запам’ятати – людина пам’ятає 10% того що чує, 20% того, що читає та 80% того що бачить та робить [15].

Серед найпоширеніших способів візуалізації даних можна виділити такі:

- графіки;
- діаграми;
- інфографіка;
- схеми;
- інтерактивний сторітеллінг;
- бізнес-аналітика;
- карти і картограми;
- тощо.

Унаочнення є доречним як для презентацій, так і під час аналізу даних, адже воно полегшує сприйняття тих матеріалів, з якими йде безпосередня робота. Візуалізація часто використовується у таких галузях та формах:

- статистика та звіти – найчастіше для одночасної демонстрації даних за певний період;

- довідкова інформація – як додаток до основного тексту, що наглядно демонструє те, про що йдеться в тексті;
- ілюстрації – дублюють або доповнюють те, про що йдеться в тексті.

1.4. Типи програмного забезпечення за характером або областю виконання

Практично на всіх комп'ютерних платформах програмне забезпечення можна згрупувати за кількома широкими категоріями. Однією з таких категорій і є характер або область виконання. В цій категорії розрізняють такі типи програмного забезпечення, як

- програми для настільних ПК;
- web-сервіси (або web-додатки);
- плагіни та розширення;
- вбудоване програмне забезпечення;
- мікрокод.

До категорії програми для настільних ПК входить таке програмне забезпечення як веб-браузери та Microsoft Office, а також програми для смартфонів та планшетів (так звані «додатки»). (У деяких частинах індустрії програмного забезпечення є певна комбінація програм для настільних ПК з мобільними додатками. Певною мірою Windows 8 та пізніше Ubuntu Touch намагалися дозволити користуватися однаковим стилем користувацького інтерфейсу на настільних ПК, ноутбуках та мобільних телефонах).

Web-сервіси (або web-додатки) зазвичай працюють на web-сервері і виводять динамічно згенеровані web-сторінки у web-браузери, використовуючи, наприклад, PHP, Java, ASP.NET або навіть JavaScript, що працює на сервері. У даний час вони зазвичай включають також деякий об'єм JavaScript-коду, який також запускається у web-браузері, і в такому випадку вони зазвичай працюють частково на сервері, частково у web-браузері.

Плагіни та розширення – це програмне забезпечення, яке розширює або змінює функціональність іншого програмного забезпечення і вимагає використання розширюваного програмного забезпечення для його функціонування;

Вбудоване програмне забезпечення зберігається як «прошивка» у вбудованих системах, пристроях, призначених для одноцільового використання, або використання в декількох цілях, таких як автомобілі та телевізори (хоча деякі вбудовані пристрої, такі як чіпсети бездротового зв’язку, можуть самі бути частиною звичайної невбудованої комп'ютерної системи, такої як ПК або смартфон). У контексті вбудованої системи іноді немає чіткого розмежування системного програмного забезпечення та прикладного програмного забезпечення. Однак деякі вбудовані системи працюють із вбудованими операційними системами, і ці системи зберігають відмінність між системним програмним забезпеченням та прикладним програмним забезпеченням (хоча, як правило, буде лише один, фіксований додаток, який завжди запускається).

Мікрокод - це особливий, відносно непомітний тип вбудованого програмного забезпечення, який повідомляє самому процесору, як виконувати машинний код, тому насправді він навіть нижче рівнем за машинний код. Зазвичай він є власністю виробника процесора, і всі необхідні оновлення програмного забезпечення для виправлення мікрокоду надаються ним користувачам (що набагато дешевше, ніж доставка замінного процесорного апаратного забезпечення).

1.5. Переваги та недоліки web-сервісів та web-додатків як типу реалізації програмного забезпечення у порівнянні з нативними додатками

Основні підходи до реалізації програмного забезпечення (особливо додатків для мобільних пристроїв) включають гібридну модель, нативні додатки та прогресивні web-додатки (спеціально адаптовані web-сервіси) (рис.1.4).

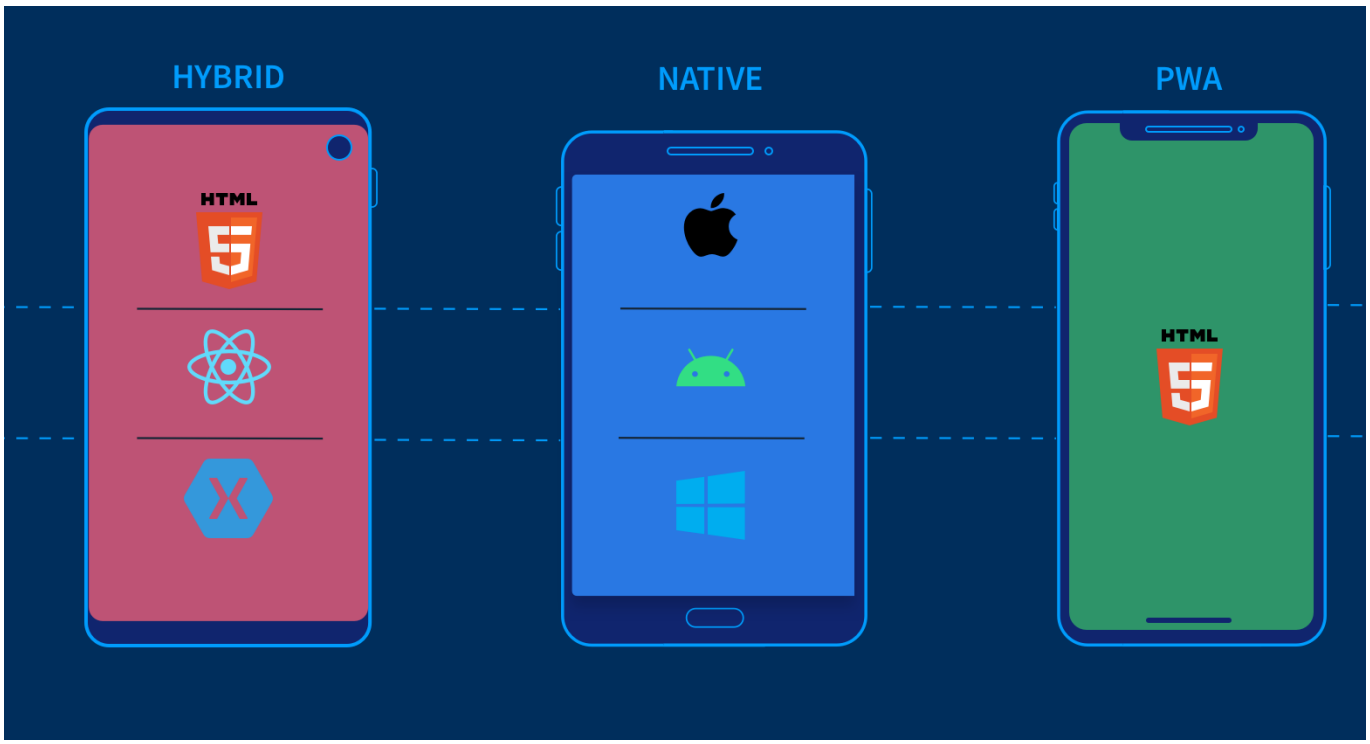

Рис. 1.4. Основні підходи реалізації мобільних додатків

Оскільки сам сервіс пошуку роботи реалізовано як web-сервіс, оптимальним типом реалізації ПЗ візуалізації статистики сервісу пошуку роботи для користувачів за областю виконання буде саме web-сервіс. Окрім схожості структури та архітектури з іншими частинами сервісу пошуку роботи, в цьому конкретному випадку, реалізація у вигляді web-сервісу (а за необхідності і прогресивного web-додатку – PWA) має ряд переваг над нативними додатками, а саме:

- Адаптивність. Сервіс є кросплатформенним та пропонує однаковий UI / UX для всіх пристроїв та розмірів екрану;
- Оптимізована продуктивність. Сервіс працює швидко на практично будь-якому пристрої (швидкість роботи браузера для різних пристроїв не враховується) та підключенні до Інтернету з мінімально допустимою пропускною здатністю;
- Додаткове встановлення не потрібне. Користувач може використовувати сервіс або PWA без встановлення. Тому багато користувачів можуть навіть не помічати, що вони використовують web-сервіс;

- Немає обов’язкових оновлень. Користувач завжди взаємодіє з останньою версією;
- Загальнодоступний. Користувач може поділитися сервісом використовуючи лише URL-адресу;
- Захищений. Оскільки протокол HTTPS є обов’язковим, програмне забезпечення запобігає розкриттю або зміненню інформації.

З іншого боку, є кілька недоліків:

- Обмеження специфічного функціоналу на iOS;
- Швидше розряджання акумулятора. PWA вимагають більше часу центрального процесора, і якщо додаток побудований з фреймворків - він швидше розряджає акумулятор;
- Обмежений доступ до апаратних компонентів та даних.

Враховуючи те, що: була необхідність створити крос-платформну програмну систему для охоплення великої аудиторії, обмежений бюджет та умови розробки й оновлення системи, відсутність необхідності в тісній інтеграції з апаратним забезпеченням пристрою, а також той факт, що push-повідомлення для користувачів iOS не є невід’ємною частиною проекту, очевидним вибором є реалізація програмної системи візуалізації статистики web-сервісу пошуку роботи для користувачів також у вигляді web-сервісу, оскільки переваги такого рішення нівелюють всі його недоліки.

1.6. Популярні мови програмування та back-end фреймворки для реалізації програмного забезпечення у вигляді web-сервісів

Коли ведеться мова про спеціалізовані рішення для веб-розробки, необхідно виділити найбільш популярні та домінуючі мови програмування - це PHP та Ruby. Дебати між розробниками, які є прихильниками цих двох мов програмування, ведуться постійно і не вщухають.

Розглянемо детальніше особливості обох. PHP – це скриптова мова програмування, що ідеально підходить для веб-розробки, в той час як прихильники

мови Ruby наголошують, що їх мова програмування має значну перевагу – простоту конструкцій. Завдяки мові Ruby можна писати найпростіший код за допомогою звичайного англійського синтаксису. Проте це не означає, що саму мову програмування легко вивчити.

Переваги PHP. Завдяки наявності значної спільноти розробників, PHP має ряд плюсів, серед яких варто виділити пакети, готові бібліотеки, наявність підтримки та фреймворків. Використання PHP дає можливість вирішити проблему, використовуючи декілька підходів у написанні коду. Область PHP складається з Eclipse, NetBeans, PhpStorm, Aptana та багатьох, багатьох інших IDE (інтегрованих середовищ розробки), які можуть знадобитися залежно від складності проекту.

Мова PHP є набагато швидшою ніж Ruby. Вона залишається популярним вибором навіть для програмістів-юніорів, оскільки вона досить легко вчиться. Вона також має ряд інструментів та редакторів у порівнянні з Ruby.

Розглянемо нижче найпопулярніші фреймворки мови PHP, що пришвидшують роботу програмістів, які задіяні у розробці веб-сайтів:

- Laravel – це найбільш популярний та найбільш продуктивний фреймворк PHP, Laravel дозволяє використовувати MVC паттерн;
- CodeIgniter – відзначається своєю швидкістю та продуктивністю порівняно з іншими фреймворками PHP, є відносно простим та елегантним фреймворком для PHP;
- Symfony – це фреймворк з відкритим кодом для прискорення обслуговування та створення web-додатків;
- CakePHP – заснований на концепціях та принципах Ruby on Rails; цей фреймворк був написаний для прискореної розробки.

Якщо зробити порівняльний аналіз між фреймворками PHP та Ruby, тоді Laravel неодмінно перемагає Ruby on Rails. Це тому, що зазначений фремйворк PHP забезпечує більш високу ефективність обробки коду. Особливо відчутною є різниця при використанні web-додатків, таких як Facebook, Twitter тощо, де відбувається швидка обробка декількох запитів одночасно. Ефективність коду має велике значення у веб-розробці [16].

Швидкість розробки за допомогою фреймворку Laravel є дуже високою, проте фреймворк Rails орієнтований на швидшу реалізацію проекту. Тому Rails використовується для стартап-проектів та MVP. Популярність PHP залишається стабільно високою і тому кількість нових ресурсів у спільноті програмістів постійно збільшується.

На базі Laravel побудовано найбільше сайтів серед усіх php-фреймворків і цей фреймворк має найнижче використання пам’яті (рис. 1.5).

Рис. 1.5. Інфографіка використання пам’яті Laravel, Symfony та Yii та сайтів, побудованих на базі них

Також Laravel пропонує найкращі опції безпеки та є найбільш «дружелюбним до розробників» (рис. 1.6).

Рис. 1.6. Інфографіка безпеки та «дружелюбності до розробників» Laravel, Symfony та Yii

80% веб-сайтів створені за допомогою PHP. Чи не це є показником успіху? Саме мова програмування диктує те, як будуть розроблятися та розвиватися веб-програми в майбутньому. Будь-яка мова, яка широко використовується для програмування, є також більш сфокусованою на нових конструкціях, кращій обробці даних та нових способах удосконалення якості кінцевого продукту.

Найпопулярніші веб-сайти, створені на PHP:

- Facebook;
- Yahoo;
- Wikipedia;
- Flickr;
- WordPress.

Недоліки PHP. Обробка помилок знаходиться на невисокому рівні, і оскільки кодування є простішим у виконанні; мова програмування є слабкою з точки зору проектування конструкцій (designing constructs). Часом PHP називають мовою, яка виробляє «код спагетті», оскільки код є заплутаним в результаті того, що розробник

вносить кілька суперечливих правок і декілька разів переписує код, щоб зробити його працюючим та отримати бажані результати.

Переваги Ruby. З чітким і елегантним синтаксисом, Ruby є зручнішим у використанні. За допомогою метапрограмування це дозволяє досягти більшого за рахунок меншої кількості коду. Оскільки Ruby використовується в галузі розробки програмного забезпечення більше 20 років, ви можете знайти значну кількість ресурсів експертного рівня на ринку. Також ви отримаєте більш високу підтримуваність (maintainability) створеного коду.

Недоліки Ruby. Якщо говорити про мінуси, то Ruby є повільнішою ніж версія PHP 7. Незважаючи на чіткий синтаксис, вивчення Ruby є значно складнішим. Серед програмістів-юніорів Ruby не є першою мовою для вивчення. Лише досвідчені розробники вивчають Ruby. Підтримка роботи сервера для Ruby / Rails знаходиться на невисокому рівні.

Очевидно що для програмного забезпечення візуалізації статистики сервісу пошуку роботи для користувачів необхідні найвищі показники швидкодії та ефективності, тому для реалізації web-сервісу було обрано мову php та back-end фреймворк Laravel.

1.7. Front-end фреймворки

Front-end фреймворки дозволяють вам одразу приступити до роботи при розробці нового веб-сайту. Завдяки їх популярності на даний доступний широкий спектр front-end фреймворків, і регулярно з’являються нові.

Оскільки варіантів на вибір дуже багато, точно визначити правильний front-end фреймворк може бути складно. Правильно обраний front-end фреймворк спрощує, впорядковує та пришвидшує процес проектування та розробки веб-сайтів, одночасно надаючи гнучкість та функції, необхідні для досягнення виняткових результатів.

Розглянемо найпопулярніші front-end фреймворки.

1.7.1. Materialize

Адаптивний front-end фреймворк Materialize, також реалізує специфікації Google material design і навантажений готовими до використання кнопками, піктограмами, картками, формами та іншими компонентами. Він пропонується як у стандартній версії, так і в тій, що працює на Sass.

Materialize включає зручну функцію сітки колонок IZ, яка може бути використана для макетів веб-сайтів. Він також постачається з CSS, який готовий використовувати з коробки для створення тіней для дизайну матеріалів, типографіки, кольорів та інших функцій.

Додаткові функції включають анімацію ефектів пульсацій, мобільні меню, що розтягуються, та включення Sass.

Переваги:

- Величезний вибір компонентів;
- Адаптивна підтримка гарантує підтримку веб-сайтів на всіх пристроях.

Недоліки:

- Великий розмір файлу робить цей фреймворк громіздким для роботи;
- Немає підтримки моделі Flexbox.

Ідеально підходить для: менш досвідчених розробників, яким потрібні вказівки щодо специфікацій Google material design.

1.7.2. Bootstrap

Цей список не був би повним без включення шалено популярного front-end фреймворку Bootstrap. Створений розробниками Twitter і спочатку випущений в 2011 році, він є найбільш часто використовуваною структурою з відкритим кодом у світі.

Як і будь-який ефективний front-end фреймворк, Bootstrap включає компоненти CSS, HTML та JavaScript. Він дотримується адаптивних стандартів веб-дизайну, що дозволяє розробляти адаптивні сайти будь-якої складності та розміру.

Оскільки він постійно оновлюється, Bootstrap, як правило, містить найновіші та найкращі функції. Наприклад, теми, що відповідають керівним принципам Google material design було додано незабаром після їх публікації, а також фреймворк було оновлено, щоб використовувати Sass як препроцесор CSS.

Переваги:

- Підтримка адаптивного веб-дизайну (за потреби також може бути відключена);
- Велика документація.

Недоліки:

- Розмір готового файлу 276 КБ через надмірну кількість рідко використовуваних стилів, однак це можна зменшити, видаливши невикористаний CSS. З Bootstrap 5 він буде ще меншим, оскільки jQuery буде скинуто як залежність;
- Надмірна кількість класів HTML та елементів DOM може бути безладною та заплутаною.

Ідеально підходить для: початківців та тих, хто віддає перевагу надійному фронтенд-фреймворку.

1.7.3. Material UI

Material UI полегшує дотримання вказівок Google щодо material design. На сьогодні це найдосконаліший механізм реалізації цих вказівок, але є одне застереження: цей фреймворк не призначений для використання як початкова точка для абсолютно нового проекту веб-дизайну.

Навантажений готовим до використання CSS та компонентами, що відповідають material design, Material UI використовує рішення CSS-in-JS. Це відкриває багато чудових функцій, включаючи вкладання тем, динамічні стилі, самопідтримку тощо.

Переваги:

- Найпростіший спосіб відповідати вимогам Google щодо material design при використанні фреймворку;

- Висока настроюваність.

Недоліки:

- Не призначений використовуватись в якості відправної точки для веб-проектів, що починаються з нуля;
- Потрібне гарне розуміння React для ефективного використання.

Ідеально підходить для: розробників, які розуміють і мають досвід роботи з React і яким потрібен простий спосіб дотримуватися вказівок щодо material design.

1.7.4. React

Створений як проект з відкритим кодом і досі використовуваний Facebook, React - це популярний фреймворк JS, який орієнтований на досвід користувачів.

Що робить React унікальним, так це те, що він може бути відтворений як на сервері, так і на стороні клієнта. Залежно від вимог до захисту даних, певні компоненти можуть відображатися на сервері, а інші - на клієнті.

Переваги:

- Багаторазові компоненти React гарантують, що розробникам не доведеться переписувати один і той же код знову і знову;
- Завдяки своїй популярності в мережі Інтернет доступна величезна кількість безкоштовних допомог від однолітків-розробників.

Недоліки:

- Посилений фокус React на розробці інтерфейсу може зробити інші аспекти розвитку складними;
- Крива навчання для цієї основи висока, частково через невідповідну проектну документацію.

Ідеально підходить для: тих, хто має досвід розробки та хоче створити веб-сайт або мобільний додаток із розширеним інтерфейсом.

1.7.5. Vue.js

Vue.js був розроблений як альтернатива Angular та React. Він був створений як мінімалістична версія Angular, але з роками він значно зростав.

Спочатку він використовувався для невеликих проектів, що ведуться розробниками, і зараз став повноцінним фреймворком.

Використовуючи традиційні HTML, CSS та JS, розробники можуть створювати компоненти, як і інші популярні фреймворки, такі як React. Що відрізняє Vue.js – це двостороння підтримка прив'язки даних.

Переваги:

- Відмінна документація і призначена для початківців та досвідчених розробників;
- Широка інструментальна екосистема, яка зросла за ці роки;
- Фреймворк невеликий за розміром і простий у вивченні.

Недоліки:

- Існує ризик надмірної гнучкості, оскільки занадто багато варіантів може призвести до різних підходів до програмування;
- Враховуючи його вік, Vue отримує менше підтримки, ніж конкуренти.

Ідеально підходить для: тих, хто має мінімальний досвід веб-розробки, якому потрібно швидко створювати прототипи.

1.7.6. Angular

Флагманський JS фреймворк від Google, Angular, розробляється вже досить давно. Хоча це не найпростіший фреймворк для вивчення, крута крива навчання може в кінцевому підсумку вартувати затраченого на вивчення часу.

Це чудово підходить для проектів, що потребують команди, які змінюються з часом, оскільки спосіб інкапсуляції компонентів робить його модульним та легким для розуміння новим розробникам.

Переваги:

- Надзвичайно складні веб-програми можна розробити в enterprise-масштабах, і скласти конкуренцію програмам для настільних ПК;
- Оскільки Google гарантує довгострокову підтримку цього проекту з відкритим кодом, розробники можуть бути впевнені, що його не покинуть найближчим часом.

Недоліки:

- Angular дуже складний і має одну з найкрутіших кривих навчання;
- Налагодження може бути проблематичним, оскільки йому бракує інструментів калібру деяких його конкурентів.

Ідеально підходить для: досвідчених розробників та інженерів, що роблять корпоративні програми, які потребують максимальної гнучкості та готові витратити час на навчання.

Для реалізації програмної системи (web-сервісу) візуалізації статистики сервісу пошуку роботи для користувачів було обрано front-end фреймворк Bootstrap як найбільш надійний, ефективний та перевірений часом варіант. Окрім основних переваг Bootstrap (рис. 1.7), варто зазначити що інші частини web-сервісу пошуку роботи також використовують Bootstrap, що забезпечує оптимальну інтеграцію компонентів та зручну підтримку всієї системи.

Рис. 1.7. Інфографіка переваг front-end фреймворку Bootstrap

Висновок до розділу 1

У результатах виконання індивідуального завдання було здійснено аналіз ключових концепцій, пов’язаних з проектуванням та реалізацією програмної системи візуалізації статистики web-сервісу пошуку роботи для користувачів, проведено огляд функціоналу сервісу пошуку роботи, визначено мету функціонування сервісу.

Розкрито поняття статистики, візуалізації. Розглянуто основні типи реалізації програмного забезпечення за характером або областю виконання. Охарактеризовано переваги та недоліки web-сервісів та web-додатків як типу реалізації програмного забезпечення у порівнянні з нативними додатками. Обрано web-сервіс як найбільш доцільний тип реалізації програмного забезпечення для візуалізації статистики сервісу пошуку роботи.

Здійснено огляд популярних мов програмування та фреймворків для реалізації web-сервісів, на основі чого обрано мову php та фреймворк Laravel для подальшої реалізації програмної системи візуалізації статистики сервісу пошуку роботи для користувачів.

Оглянуто рейтинг найпопулярніших front-end фреймворків, проаналізовано переваги та недоліки кожного та враховано численні фактори, що впливають на реалізацію, в результаті чого обрано Bootstrap як найбільш надійний, ефективний та перевірений часом варіант.

РОЗДІЛ 2
АРХІТЕКТУРА, СТРУКТУРА ТА ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

2.1. Значення архітектури програмної системи та її складових

Архітектура програмної системи – це форма, що її надають системі творці. Ця форма утворюється розподілом систем на компоненти, їхньою організацією та визначенням способів взаємодії між ними.

Мета форми – спростити розробку, розгортання і супровід програмної системи, що міститься в ній. Головна стратегія такого спрощення полягає в тому, щоб якомога довше мати якнайбільше варіантів.

Розробка

Програмній системі, яку важко розвивати, навряд чи варто розраховувати на довге і здорове життя. Тому архітектура системи повинна робити її простою у розвитку для тих, хто її розробляє.

Розгортання

Щоб досягти високої ефективності, програмна система повинна легко розгортатись. Що вище вартість розгортання, то нижче ефективність системи. Відповідно, метою архітектури є створення системи, яку можна розгорнути в одну дію.

Стратегія розгортання рідко береться до уваги на початкових етапах розробки. У результаті може вийти архітектура, що забезпечує простоту розробки, але істотно ускладнює розгортання [17]. Щоб запобігти цьому, стратегія розгортання була визначена ще на етапі проектування програмної системи візуалізації статистики сервісу пошуку роботи.

Робота системи

Вплив архітектури системи на її роботу менш драматичний, ніж на розробку, розгортання і супровід, однак не менш важливий.

У випадку програмної системи візуалізації статистики сервісу пошуку роботи для користувачів, спроектованої для забезпечення масштабування та роботи в умовах високого навантаження, однією з головних задач є забезпечення високого рівня ефективності архітектури.

Існує ще одна роль, яку архітектура грає в роботі системи: гарна архітектура виражає оперативні потреби системи.

Можна навіть сказати, що гарна архітектура робить роботу системи зрозумілою розробнику. Архітектура повинна відображати особливості роботи. Архітектура повинна піднімати варіанти використання, особливості та очікування реакції системи до рівня повноправних сутностей, які слугуватимуть видимими орієнтирами для розробника. Це спростить розуміння системи і, відповідно, надасть значну допомогу в розробці та супроводі [17].

2.2. Загальна архітектура програмної системи та її складові

Програмна система візуалізації статистики сервісу пошуку роботи для користувачів створювалася з використанням php-фреймворку Laravel. З міркувань продуктивності та практичності було обрано останню доступну версію 8.61.0. Вибір фреймворку продиктований необхідністю мати гнучку архітектуру (а отже довше залишати більше варіантів розробки, розгортання і супроводу) та підтримки різних інструментів для взаємодії із сторонніми сервісами. Серед інших компонентів – хмарні сервіси AWS, а саме S3, що виконує роль сховища сервісу, NoSQL база даних DynamoDB та CDN Cloudfront. Веб-сервер – NGINX. На рисунку 2.1 наведено загальну схему середовища розгортання програмної системи.

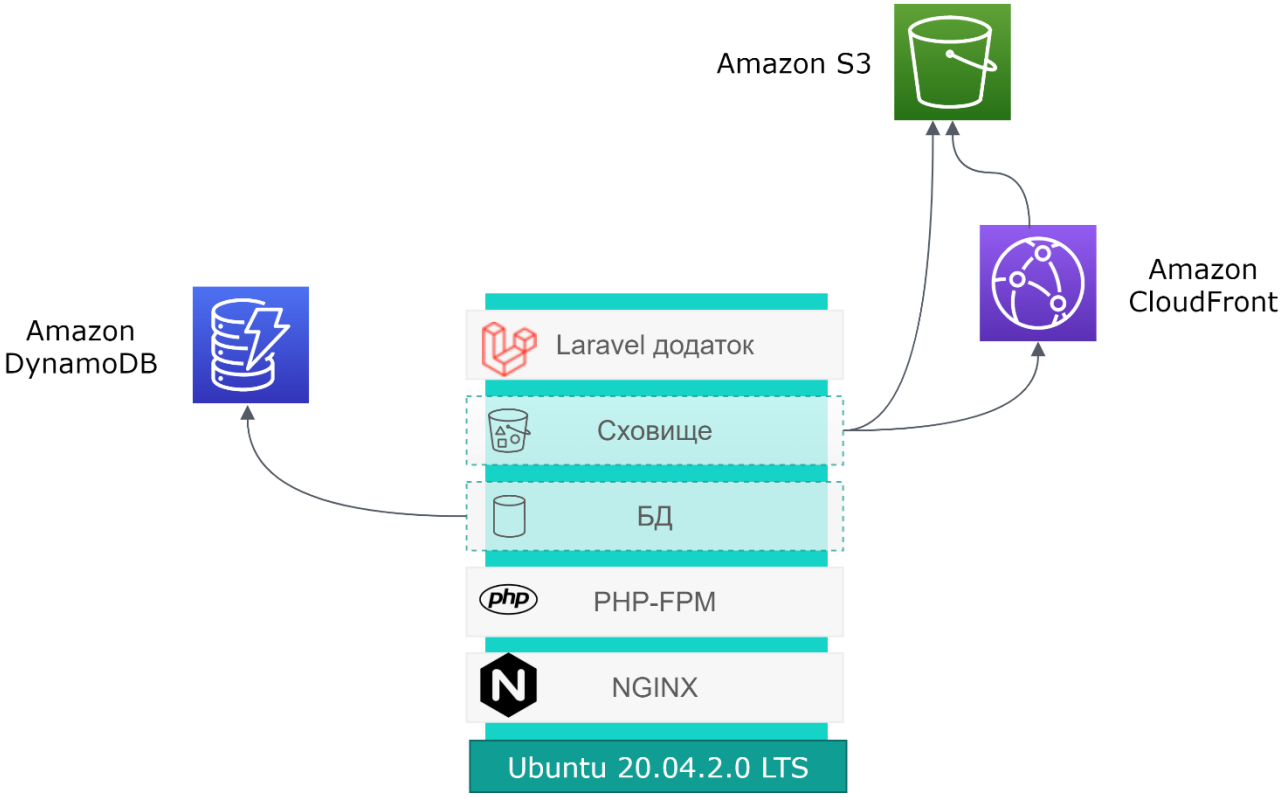

Рис. 2.1. Загальна схема середовища розгортання програмної системи

На рисунку 2.2 наведено загальну спрощену схему архітектури програмної системи для версії production release.

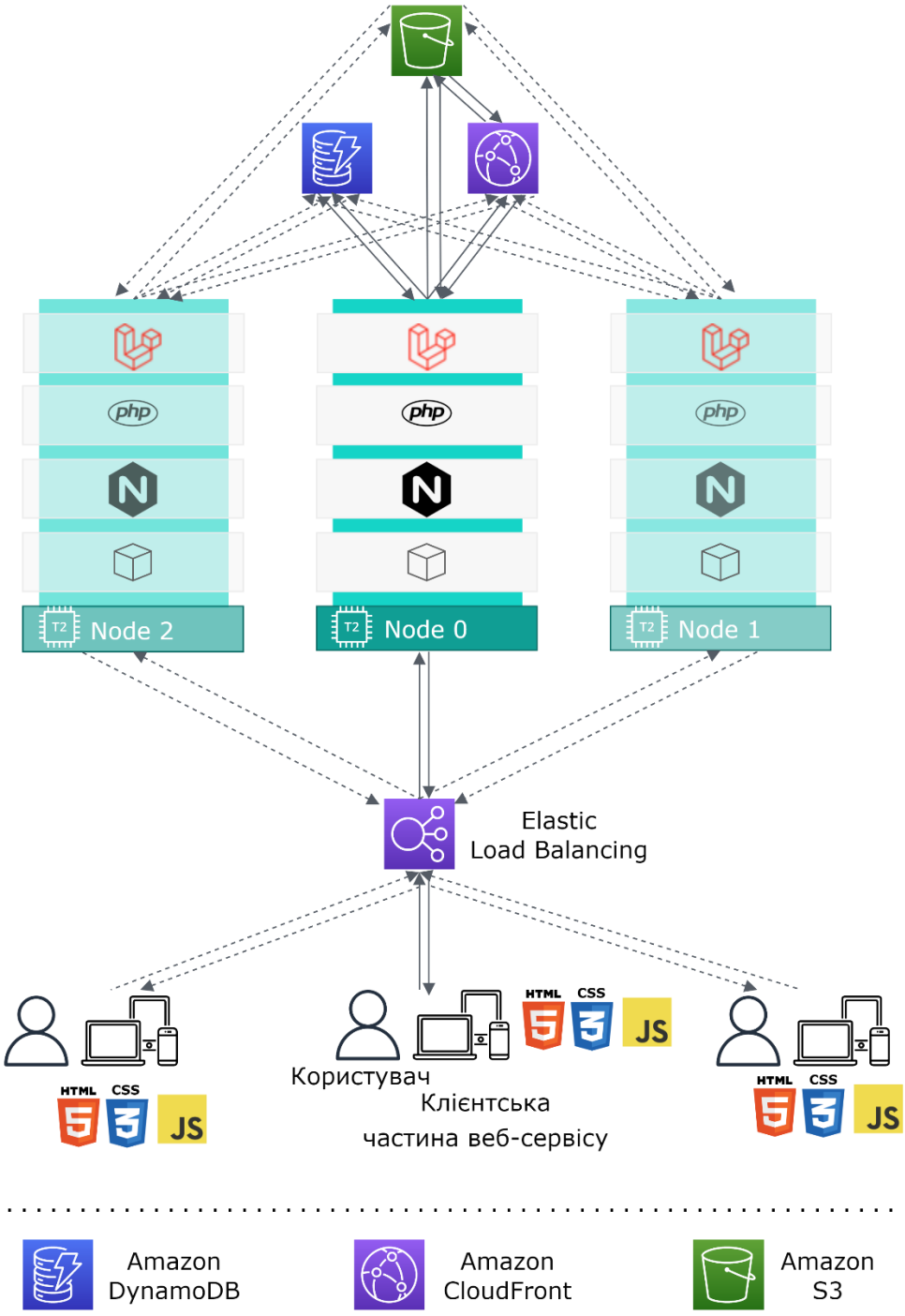

Рис. 2.2. Загальна спрощена схема архітектури програмної системи для версії production release

Загальна архітектура системи комбінує декілька стилів, серед яких клієнт-серверна архітектура, специфічна мікроархітектура Laravel та хмарна архітектура. Одним з основних патернів проектування, що застосовувалися для реалізації сервісу є MVC.

В наступних підрозділах буде розглянуто детальніше вищезгадані компоненти та патерни.

2.3. Laravel та патерн MVC

Laravel дозволяє реалізовувати web-додатки з використанням одного із найпопулярніших шаблону проектування (або патерну, що семантично ближче до оригінального англійського терміна pattern) MVC.

Модель-представлення-контролер (зазвичай відомий під назвою MVC від англ. Model-View-Controller) – це патерн проектування програмного забезпечення, який зазвичай використовується для розробки користувацьких інтерфейсів та розділяє відповідну логіку програми на три взаємопов'язані елементи. [18]

Розглянемо елементи MVC та їх взаємодію (рис. 2.3).

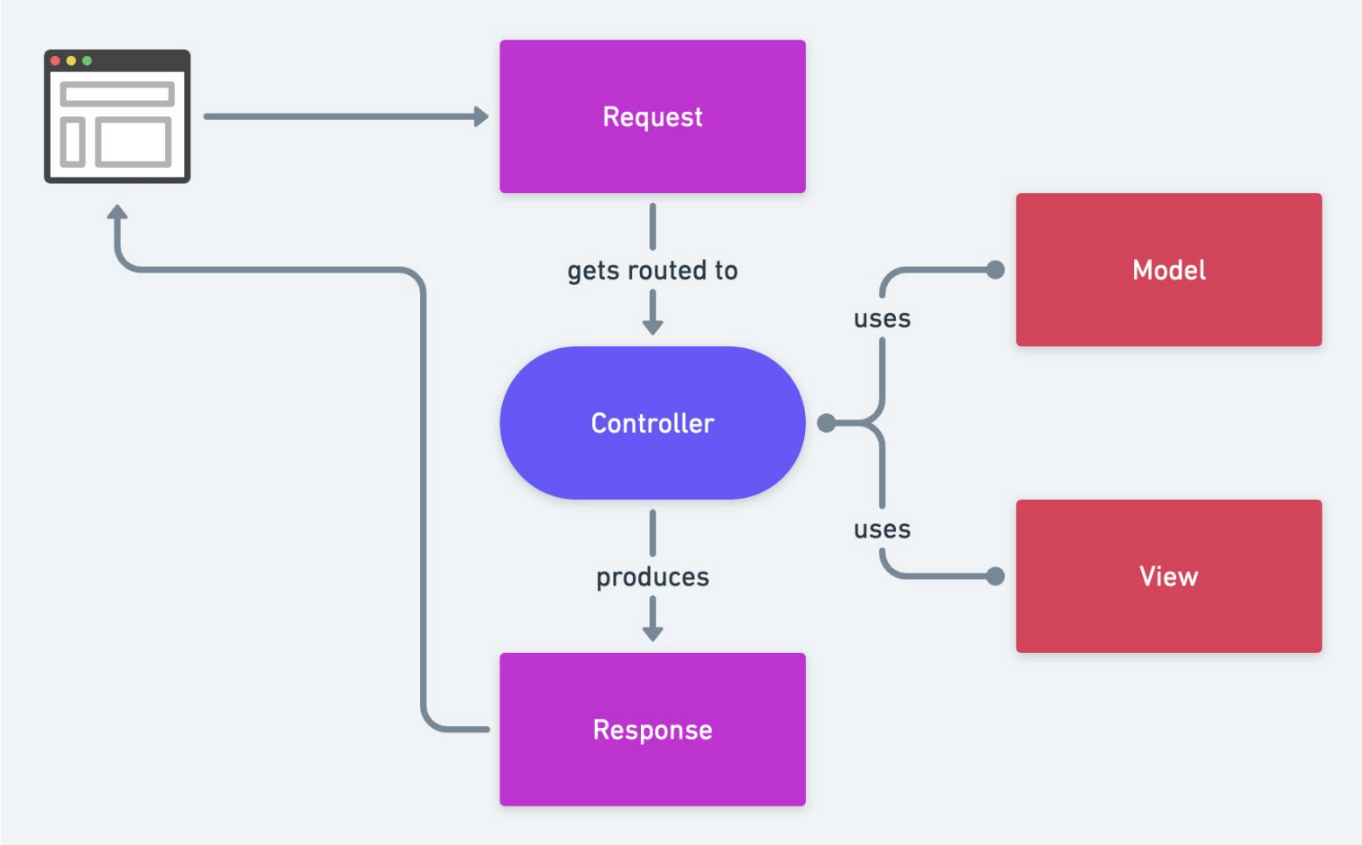

Рис. 2.3. Елементи MVC та діаграма взаємодій у межах патерну

Модель

Центральний компонент патерну. Це динамічна структура даних програми, незалежна від інтерфейсу користувача. Модель безпосередньо керує даними, логікою та правилами програми.

Представлення

Будь-яке представлення інформації, наприклад графіки, діаграми або таблиці. Можливі кілька видів (представлень) однієї й тієї самої інформації, такі як гістограма для менеджменту та табличне представлення для бухгалтерів.

Контролер

Приймає введення та конвертує його в команди для моделі чи представлення.

Важливо зауважити, що у Laravel MVC – це лише «верхівка айсберга». Laravel дозволяє створювати додатки, використовуючи патерн MVC, але фреймворк сам по собі є набагато складнішим. Загальна структура проекту включає не лише MVC (наприклад, контролери є частиною HTTP-шару, що також містить Middleware та опціонально Requests). Тобто можна побачити, що шарів набагато більше і вони не обмежуються лише “M”, “V” і “C”. Як зазначав сам автор Laravel – Taylor Otwell: «Неможливо вмістити всі аспекти робастних (відмовостійких) web-додатків у цих 3 літерах» [19].

Laravel дозволяє створювати власну архітектуру. При цьому, створюваний додаток або сервіс все ж здебільшого обгорнутий навколо патерну MVC, оскільки завдяки широкому та потужному вбудованому інструментарію Laravel більшу частину часу навіть немає потреби щось настроювати чи додавати «під поверхнею».

При розробці програмної системи візуалізації статистики сервісу пошуку роботи для користувачів було створено представлення (views) та планування (layouts) для кожної відповідної сторінки сервісу, моделі, що вміщують бізнес-логіку для кожної необхідної сутності та контролери, які відстежують визначені події, що виникають як наслідок дій користувачів. Події зареєстровані контролером трансформуються в різні запити, які передаються компонентам моделі або об'єктам, відповідальним за відображення даних.

Розглянемо декілька прикладів.

Для операцій з користувачами та пов’язаними об’єктами було створено ряд представлень для сторінок профілю користувача та налаштувань облікового запису, модель User.php та контролери UserController.php, RegisterController.php, LoginController.php, ConfirmPasswordController.php, ForgotPasswordController.php, ResetPasswordController.php, VerificationController.php, RolesController.php, PermissionController.php.

Розглянемо структуру моделі User.php. з основними властивостями (рис. 2.4):

- table – таблиця, пов’язана з моделлю;
- primaryKey – первинний ключ для моделі;
- incrementing – вказує чи ідентифікатори (ID) автоматично збільшуються;
- fillable – атрибути, для яких доступне масове присвоювання;
- dynamoDbIndexKeys – індекси простого або композитного ключа DynamoDB;
- compositeKey – масив композитного ключа DynamoDB;
- dateFormat – формат дати за замовчуванням, відповідний ISO 8601;
- exists – вказує, чи існує модель;
- wasRecentlyCreated – вказує, чи була модель впроваджена протягом поточного життєвого циклу запиту;
- attributes – атрибути моделі;
- original – початкові значення атрибутів моделі;
- timestamps – вказує чи включена автоматична підтримка стовпців або ключів created_at та updated_at;
- rememberTokenName – назва стовпця або ключа для токена “remember me”.

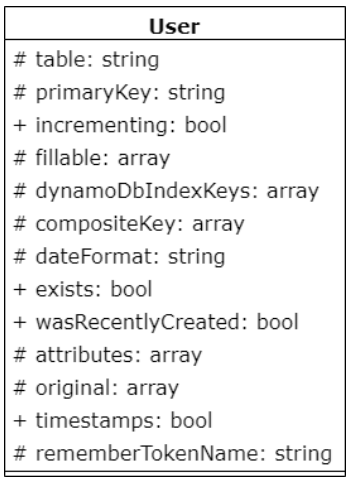

Рис. 2.4. Структура моделі User.php. з основними властивостями

Розглянемо вміст масиву fillable та структуру масиву attributes моделі User.php (рис. 2.5). Масив fillable містить такі значення: “username”, “email”, “password”, “user-info”, “skills”, “favorite-companies”, “preferred-positions”. Масив attributes містить ключі, що представляють значення масиву fillable, а також ключі “id”, “created_at”, “updated_at” та “remember_token”.

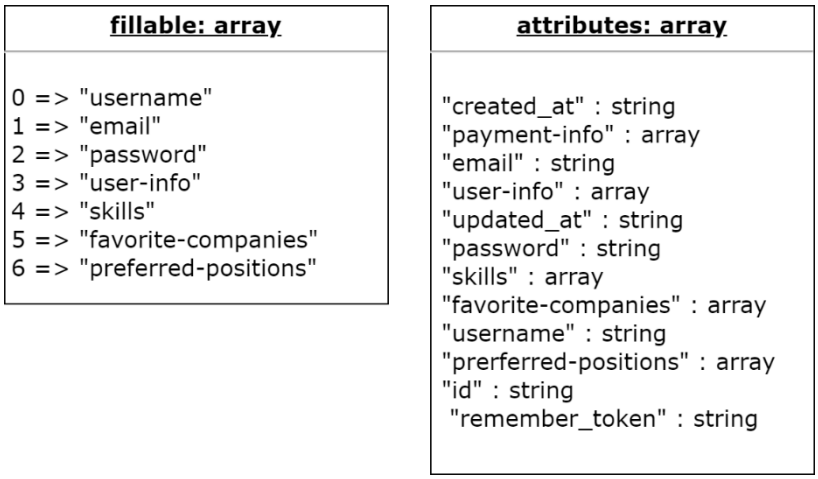

Рис. 2.5. Вміст масиву fillable та структуру масиву attributes моделі User.php.

Фактично, для визначення моделі в коді, необхідно визначити лише властивості “table”, “primaryKey”, “CompositeKey” та “fillable”. Всі інші властивості буде впроваджено в клас Eloquent ORM з використанням значень, заданих в

DynamoDbModel, яка успадковується моделлю User, PHP magic methods та зчитаних метаданих відповідної таблиці БД. Сирцевий код User.php наведено в першому розділі Додатку А.

Для операцій з візуалізації статтистики користувача було створено представлення dashboard/dashboard/index.blade.php та dashboard/statistics/index.blade.php, модель Statistics.php та контролер StatisticsController.php, що успадковує абстрактний клас DashboardController.php. За таким же принципом було створено всі необхідні компоненти MVC для всіх інших необхідних сутностей. Всі вищезгадані сутності мають схожу структуру моделі, представлень та контролерів. Фрагмент сирцевого коду DashboardController.php та сирцевий код UserController.php, RolesController.php наведено в другому, третьому та четвертому розділах Додатку А відповідно.

2.4. База даних

NoSQL, що є абревіатурою від “not only SQL” та розшифровується як «не тільки SQL», – це підхід до розробки БД, який забезпечує гнучкі схеми зберігання та пошуку даних поза рамками традиційних табличних структур, що знаходяться в реляційних базах даних. Найпоширеніші типи баз даних NoSQL – це бази даних пар «ключ-значення», бази даних документів, wide-column store та графові БД.

Бази даних ключ-значення – це відносно простий тип бази даних, де кожен елемент містить ключі та значення. Значення зазвичай може бути отримане лише шляхом звернення по його ключу, тому запитувати певну пару ключ-значення, як правило, просто. Простота цієї моделі робить БД «ключ-значення» швидкими, простими у використанні, масштабованими та гнучкими.

Бази даних документів зберігають дані в документах, подібних до об'єктів JSON (JavaScript Object Notation). Кожен документ містить пари полів і значень. Значення зазвичай можуть бути різних типів, включаючи string, number, boolean, array, або object, і їх структури зазвичай узгоджуються з об'єктами, з якими

розробники працюють в коді. Вони можуть горизонтально масштабуватися для розміщення великих обсягів даних. [20]

Для реалізації БД програмної системи було обрано NoSQL сервіс Amazon DynamoDB.

Amazon DynamoDB – це база даних пар «ключ-значення» та документів, яка забезпечує однознакову мілісекундну продуктивність (тобто затримку менше 10 мілісекунд) при будь-якому масштабі. Це повністю керована, надійна БД для додатків у масштабі всього Інтернету, яка працює в декількох регіонах з кількома провідними серверами і має вбудовані засоби забезпечення безпеки, резервного копіювання та відновлення, а також кешування в пам'яті. Ці фактори є надзвичайно важливими для програмної системи візуалізації статистики сервісу пошуку роботи для користувачів, яка передбачає роботу зі складними та об’ємними наборами даних, що можуть змінюватися. DynamoDB може обробляти більше 10 трлн запитів в день і справлятися з піковими навантаженнями, що перевищують 20 млн запитів на секунду.[21]

2.4.1. Робота з DynamoDB в Laravel

Одним з ключових рішень, інтегрованих у Laravel є Eloquent ORM. ORM (також відоме як O/RM або O/R mapping tool) – об’єктно-реляційне відображення або об’єктно-реляційна проекція – метод програмування для конвертування даних між несумісними системами типів з використанням об’єктно-орієнтованих мов програмування. По суті, створюється «віртуальна об’єктна база даних», яка може бути використана всередині мови програмування.

Eloquent ORM забезпечує водночас просту та прогресивну реалізацію шаблону проектування (або патерну, що семантично ближче до оригінального англійського терміна pattern) ActiveRecord для роботи з базами даних. ActiveRecord передбачає наступне: кожна таблиця бази даних має відповідну «Модель» (“Model”), представлену PHP-класом яка використовується для взаємодії з цією таблицею. Клас відображає таблицю бази даних, об’єкт – запис (рядок, або в даному випадку –

елемент) у таблиці, а методи класу реалізують операції CRUD та інший функціонал. ActiveRecord представляє “M” у MVC, тобто модель – шар системи, відповідальний за представлення бізнес-даних та логіки. Відповідно для операцій CRUD замість методів класу використовуються методи контролера. Active Record полегшує створення та використання бізнес-об’єктів, для даних яких потрібно постійне зберігання в базі даних.

За замовчуванням, Eloquent не підтримує роботу з DynamoDB. Тим не менш, доступна велика кількість сторонніх пакетів, які дозволяють зручно працювати з іншими БД, використовуючи синтаксис Eloquent [16].

Тому для забезпечення комфортної роботи з DynamoDB було використано інструмент baopham/laravel-dynamodb, що дозволило:

- Розширити моделі з BaoPham\DynamoDb\DynamoDbModel, завдяки чому стало можливе використання підтримуваних методів Eloquent. Ідея полягає в тому, що можна в будь-який момент переключитися безпосередньо на Eloquent для роботи з SQL базами даних, не змінюючи запитів. Це забезпечує додаткову гнучкість проекту;
- Використовувати фасад конструктора запитів у разі виникнення потреби створення більш складних запитів;
- Запускати такі запити Eloquent, як збереження, оновлення та видалення асинхронно на DynamoDB. Це можливо завдяки використанню guzzlehttp promises у AWS SDK v3 для PHP для забезпечення асинхронних процесів.

2.4.2. Таблиця програмної системи

DynamoDB є schema-less (тобто не має схеми). Це означає, що при створенні таблиці не визначається жорстка схема таблиці, а вказуються лише атрибути первинного ключа, такі як ключ розділу (partition key) або ключ розділу (partition key) та ключ сортування (sort key). У будь-який момент можна додати атрибут будь-якого типу до будь-якого з елементів таблиці.

При створенні таблиці в DynamoDB Management Console спочатку вказуються назва таблиці, атрибути первинного ключа та їх типи даних. Потім встановлюються значення read/write capacity mode – provisioned або on-demand відповідно. У випадку provisioned capacity, встановлюються значеня Read capacity units (RCU) та Write capacity units (WCU) для таблиці, після чого можна увімкнути опцію автоматичного масштабування Read capacity та Write capacity. Таблиці також можна створювати програмно (наприклад безпосередньо викликаючи скрипт створення таблиці), зазначаючи ті ж дані.

Дизайн таблиці DynamoDB відповідає схемі введення реляційного порядку, яка представлена в найкращих практиках моделювання реляційних даних в DynamoDB [22]. Він слідує патерну проектування Adjacency List, який є загальним способом представлення реляційних структур даних у DynamoDB.

Цей патерн проектування вимагає визначити набір типів сутностей, які зазвичай співвідносяться з різними таблицями в реляційній схемі. Потім елементи сутності додаються до таблиці за допомогою складеного (з ключа розділу та ключа сортування) первинного ключа. Ключ розділу цих елементів сутності є атрибутом, який однозначно ідентифікує елемент і є загальним для всіх елементів як PK. Атрибут ключа сортування містить значення атрибута, яке можна використовувати для інвертованого індексу або глобального вторинного індексу (GSI). Він зазвичай позначається як SK.

Визначаються сутності, які підтримують схему введення в реляційному порядку. Серед багатьох сутностей розглянемо такі:

- HR-співробітник - ПК: EmployeeID, SK: ім'я працівника
- HR-Регіон - PK: RegionID, SK: Назва регіону
- HR-Country - PK: CountryId, SK: Назва країни
- HR-Location - PK: LocationID, SK: Назва країни
- HR-Job - PK: JobID, SK: Назва посади
- HR-відділ - ПК: DepartmentID, SK: DepartmentID
- OE-Клієнт - PK: Ідентифікатор клієнта, SK: Ідентифікатор облікового запису

- OE-заявка - PK ApplicationID, SK: CustomerID

Після додавання цих елементів сутностей до таблиці можна визначити взаємозв'язки між ними, додавши крайні елементи до розділів елементів сутності. Наступна таблиця (рис. 2.6) демонструє цей крок.

HR OE TABLE

Primary Key		Attributes											
PK	SK (GSI-1-PK)	GSI-1-SK											
HR-EMPLOYEE1	EMPLOYEE1	Data (Full Name)	StartDate	EndDate	JobID	JobTitle	PhoneNumber	Email	ManagerID	Country	City	Region	Department
		John Smith											
	QUOTA-2017-Q1	Data (Order Totals USD)	EmployeeName										
		50000											
	HR-CONFIDENTIAL	Data (Hire Date)	EmployeeName	Salary	CommissionPct								
		2015-11-08											
	WA|SEATTLE	Data (Desk Location)	EmployeeName										
		B01|F07|A27|R05	John Smith									
	J-AM3	Data (Job Title)	DepartmentID	StartDate	EndDate	JobID							
		Principal Account Manager											
	JH-AM2	Data (Job Title)	DepartmentID	StartDate	EndDate	JobID							
		Senior Account Manager											
	JH-AM1	Data (Job Title)	DepartmentID	StartDate	EndDate	JobID							
		Account Manager											
HR-REGION1	PNW	Data (Region Name)	RegionName										
		Pacific Northwest Territory											
HR-COUNTRY1	USA	Data (Country Name)	CountryName	RegionID									
		United States											
HR-LOCATION1	WA|SEATTLE	Data (City State)	CityName	PostalCode	StreetAddress	StateProvince	CountryID						
		Seattle, Washington											
HR-JOB1	J-AM3	Data (Job Title)	JobTitle	MinSalary	MaxSalary								
		Principal Account Manager											
HR-DEPARTMENT1	COMMERCIAL	Data (Department Name)	DepartmentName	ManagerID	City	Location							
		Commercial Sales		EMPLOYEE2									
OE-CUSTOMER1	CUSTOMER1	Data (Customer Name)	Address	IncomeLevel	PhoneNumber	NLSLanguage	NLSTerritory	CreditLimit	CustEmail	CustLocatio	DateOfBirth	MaritalStatus	Gender
		ACE Building Supplies											
OE-ORDER1	CUSTOMER1	Data (StatusDate) (GSI-2-SK)	GSI-Bucket (GSI-2-PK)	SalesRepID	AccountManager	OrderMode	OrderTotal	PromotionID					
		OPEN#2018_08_11	RND(0,N)	EMPLOYEE1									
	EMPLOYEE1	Data (StatusDate)	Order Total										
		OPEN#2018_08_11	2500										
	PRODUCT1	Data (StatusDate) (GSI-2-SK)	GSI-Bucket (GSI-2-PK)	OrderQuantity	UnitPrice								
		OPEN#2018_08_11	RND(0,N)										
OE-PRODUCT1	PRODUCT1	Data (Product Name)	ProductDescription	WAREHOUSE1	WAREHOUSE2	CategoryID	WeightClass	WarranytPeri	SupplierID	ProductSta	ListPrice	MinPrice	CatalogURL
		Quickcrete Cement - 50 lb bag		InventoryQty	InventoryQty								
	PNW	Data (Region Name)	TranslatedName	Description									
		Pacific Northwest	TRANSLATED_NAME										
OE-WAREHOUSE1	PNW	Data (Warehouse Type)	WarehouseSpec	Location	WHGeoLocation								
		Building Supplies											

Рис. 2.6. Таблиця для роботи з реляційними даними в DynamoDB

У цьому прикладі розділи Employee та Application у таблиці мають додаткові крайні елементи, які містять вказівники на інші елементи сутності в таблиці. Далі визначаються кілька глобальних вторинних індексів (GSI) для підтримки всіх патернів доступу, визначених раніше. Елементи сутності не всі використовують однаковий тип значення для первинного ключа або атрибута ключа сортування. Все, що потрібно, це наявність первинного ключа та атрибутів ключа сортування для вставки в таблицю.

Той факт, що деякі з цих сутностей використовують власні імена, а інші використовують ідентифікатори інших сутностей як значення ключа сортування, дозволяє одному і тому ж глобальному вторинному індексу підтримувати кілька типів запитів. Цей прийом називається перевантаженням GSI. Він ефективно усуває

обмеження за замовчуванням на 20 глобальних вторинних індексів для таблиць, що містять кілька типів елементів. Це показано на наступній схемі (рис. 2.7) як GSI 1.

GSI #1

GSI 1 Primary Key		Projected Attributes												
GSI-1-PK	GSI-1-SK													
EMPLOYEE1	John Smith	PK	SK	StartDate	EndDate	JobID	JobTitle	PhoneNumbe	Email	ManagerID	Country	City	Region	Department
		HR-EMPLOYEE1	EMPLOYEE1											
	OPEN#2018_08_11	PK	SK	OrderTotal										
		OE-ORDER1	EMPLOYEE1	2500										
PNW	Pacific Northwest Territory	PK	SK	RegionName										
		HR-REGION1	PNW											
	Building Supplies	PK	SK	WarehouseSpec	Location	WHGeoLocation								
		OE-WAREHOUSE1	PNW											
	Pacific Northwest	PK	SK	TranslatedName	Description									
		OE-PRODUCT1	PNW											
CUSTOMER1	ACE Building Supplies	PK	SK	Address	IncomeLevel	PhoneNumber	NLSLanguage	NLSTerritory	CreditLimit	CustEmail	CustLocation	DateOfBirth	MaritalStatus	Gender
		OE-CUSTOMER1	CUSTOMER1											
	OPEN#2018_08_11	PK	SK	GSI-Bucket	SalesRepID	AccountManager	OrderMode	OrderTotal	PromotionID					
		OE-ORDER1	CUSTOMER1	RND(0,N)	EMPLOYEE1									
WA|SEATTLE	B01|F07|A27|R05	PK	SK	EmployeeName										
		HR-EMPLOYEE1	WA|SEATTLE	John Smith										
	Seattle, Washington	PK	SK	CityName	PostalCode	StreetAddress	StateProvince	CountryID						
		HR-LOCATION1	WA|SEATTLE											
J-AM3	Principal Account Manager	PK	SK	DepartmentID	StartDate	EndDate	JobID							
		HR-EMPLOYEE1	J-AM3											
	Principal Account Manager	PK	SK	JobTitle	MinSalary	MaxSalary								
		HR-JOB1	J-AM3											
PRODUCT1	OPEN#2018_08_11	PK	SK	GSI-Bucket	OrderQuantity	UnitPrice								
		OE-ORDER1	PRODUCT1	RND(0,N)										
	Quickcrete Cement - 50 lb bag	PK	SK	ProductDescription	WAREHOUSE1	WAREHOUSE2	CategoryID	WeightClass	WarranytPe	SupplierID	ProductStatus	ListPrice	MinPrice	CatalogURL
		OE-PRODUCT1	PRODUCT1		InventoryQty	InventoryQty								
QUOTA-2017-Q1	50000	PK	SK	EmployeeName										
		HR-EMPLOYEE1	QUOTA-2017-Q1											
HR-CONFIDENTIAL	2015-11-08	PK	SK	EmployeeName	Salary	CommissionPct								
		HR-EMPLOYEE1	HR-CONFIDENTIAL											
JH-AM2	Senior Account Manager	PK	SK	DepartmentID	StartDate	EndDate	JobID							
		HR-EMPLOYEE1	JH-AM2											
JH-AM1	Account Manager	PK	SK	DepartmentID	StartDate	EndDate	JobID							
		HR-EMPLOYEE1	JH-AM1											
USA	United States	PK	SK	CountryName	RegionID									
		HR-COUNTRY1	USA											
COMMERCIAL	Commercial Sales	PK	SK	DepartmentName	ManagerID	City	Location							
		HR-DEPARTMENT1	COMMERCIAL		EMPLOYEE2									

Рис. 2.7. Перевантаження глобального вторинного індексу в таблиці для роботи з реляційними даними в DynamoDB

GSI 2 (другий глобальний вторинний індекс) призначений для підтримки досить поширеного патерну доступу до програми, який полягає в отриманні всіх елементів таблиці, які мають певний стан. Для великої таблиці з нерівномірним розподілом елементів по доступних станах цей шаблон доступу може призвести до «гарячого ключа», якщо елементи не розподілені по кількох логічних розділах, які можна запитувати паралельно. Цей патерн проектування називається шардинг запису.

2.5. Сховище

Достатній об’єм сховища даних є одним з ключових аспектів, важливою та часто просто необхідною деталлю справді масштабованих сервісів. Управління

великомасштабним сховищем для серверів та його підтримка – дійсно непроста задача, адже необхідно розрахувати потрібні об’єми резервування, показники доступності, кількості накопичувачів на один сервер та вирішити, як управляти накопичувачами, щоб у разі виходу накопичувача з ладу про це було відомо заздалегідь завдяки аналітичним даним.

Рішенням є використанням стороннього сервісу зберігання файлів. В такому випадку усі відповідні завдання передаються сервісу, при цьому часто зростає економічна ефективність, оскільки ціна зазвичай є меншою ніж у випадку створення та підтримки власного сховища. Коли мова йде про сервіси зберігання файлів, золотим стандартом є сервіс Amazon S3 [16].

Amazon S3 (або Amazon Simple Storage Service) – це сервіс зберігання об'єктів, що пропонує кращі в галузі продуктивність, масштабованість, доступність та безпеку даних. Amazon S3 пропонує прості у використанні інструменти адміністрування, які дозволяють організувати дані і точно налаштувати обмеження доступу відповідно до конкретних потреб бізнесу та/або законодавчих вимог. Amazon S3 забезпечує надійність 99,999999999% (11 дев'яток) і зберігає дані мільйонів додатків в інтересах компаній з усього світу [23].

Laravel надає потужну абстракцію файлової системи завдяки використанню PHP пакету Flysystem. Дана інтеграція Flysystem у Laravel забезпечує прості у користуванні драйвери для роботи з локальними файловими системами, Amazon S3 та Amazon CloudFront. Це дає можливість легко та зручно перемикатися між даними опціями сховищ, оскільки API залишається одним і тим самим для кожної системи, а отже підвищує загальну гнучкість сервісу.

Для реалізації сховища програмної системи візуалізації статистики сервісу пошуку роботи для користувачів було створено бакет Amazon S3. Бакет (англ. bucket) – контейнер сховища в Amazon S3. Бакети та об’єкти є первинними ресурсами, при чому об’єкти зберігаються в бакетах. Amazon S3 має плоску структуру замість ієрархії, яка зазвичай зустрічається у файловій системі. Однак для простоти організації консоль Amazon S3 підтримує концепцію папок як засіб групування об’єктів. Amazon S3 робить це за допомогою спільного префіксу імені

для об'єктів (тобто об’єкти мають імена, які починаються із загального рядка). Назви об’єктів також називають іменами ключів.

Розглянемо детальніше структуру бакету програмної системи візуалізації статистики сервісу пошуку роботи для користувачів, використовуючи концепцію папок консолі Amazon S3. В бакеті, серед іншого містяться 2 папки (рис. 2.8) – assets, users.

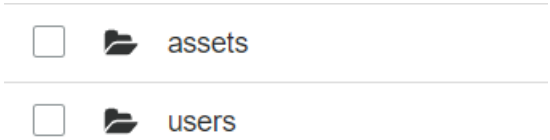

Рис. 2.8. Фрагмент вмісту бакету web-сервісу

В папці assets містяться папки css, icons, images та js. Папка css містить таблиці стилів для різних сторінок та частин web-сервісу. Icons – іконки, які використовуються на сервісі. Images – логотип сервісу, зображення, що використовується на головній сторінці, фонові зображення сторінок реєстрації, зображення-плейсхолдери тощо. Папка js – JavaScript скрипти та бібліотеки.

В папці users містяться підпапки для кожного користувача, в яких, в свою чергу, міститься по дві папки original та square, де знаходяться користувацькі фотографії профілю, original – з роздільною здатністю 500 x 500 пікселів, small – з роздільною здатністю 66 x 66 пікселів.

2.6. Мережа доставки контенту

Мережа доставки контенту або мережа дистрибуції контенту (CDN) – це географічно розподілена мережа серверів контенту, що виконують роль проксі-серверів та відповідних центрів обробки даних. Мета – забезпечити високу доступність та продуктивність, розподіляючи сервіс просторово відносно кінцевих користувачів. Результатом використання мережі доставки контенту контент-

провайдерами є зменшення затримки та прискорення завантаження різних типів цифрового вмісту для кінцевих користувачів.

CDN мають сервери розташовані в багатьох різних фізичних локаціях, часто поряд з специфічними для регіону точками обміну Інтернет-трафіком (IXP) або безпосередньо в них. Ці локації називають “Points of Presence” (PoPs) тобто «Точками Присутності», а сервери в них – “edge servers” тобто «крайовими серверами» або «периферійними серверами» (через це мережі доставки контенту ще часто називають «периферійними або крайовими кешами»). Використання периферійних серверів кешування в низці точок присутності дозволяє CDN не лише забезпечувати масштабованість великим числом серверів, а й надавати сервери, які знаходяться в безпосередній близькості від кінцевого користувача.

Мережа доставки контенту дозволяє перенаправити запити клієнтської частини сервісу, створювані користувачами з URL-адреси файлу у сховищі Amazon S3 на URL-адресу зображення в CDN . Коли на цю адресу вперше приходить запит, у CDN ще немає даних про зображення, але є правила які вказують, як знайти оригінальне зображення у сховищі нашого сервісу. Починаючи з цього моменту, коли мережа доставки контенту вперше отримує зображення, доставка зображень користувачам буде здійснюватися CDN, при цьому запити користувачів не будуть проходити через сервери сервісу [16].

Наприклад, коли хтось у Сан-Франциско відкриває сторінку статистики на веб-сервісі, розміщеному у Франкфурті, він отримує таблиці стилів та скрипти JavaScript із сервера розміщеного в США. Очевидно, це значною мірою покращує користувацький досвід, дозволяючи користувачам швидко отримувати дані від серверів, що знаходяться неподалік, на противагу очікуванню, поки запит перетне океани для отримання даних.

Мережа доставки контенту для програмного забезпечення візуалізації статистики сервісу пошуку роботи для користувачів реалізована сервісом Amazon CloudFront.

Amazon CloudFront - це зручний для розробників сервіс глобальної мережі доставки контенту (CDN), що забезпечує швидку і безпечну передачу даних, медіа,

додатків і API клієнтам по всьому світу з низькими затримками і високою швидкістю.

CloudFront інтегрований з AWS – як фізичними локаціями, безпосередньо підключеними до глобальної інфраструктури AWS, так і іншими сервісами AWS. CloudFront бездоганно працює з сервісами, включаючи AWS Shield для нейтралізації DDoS, Amazon S3, Elastic Load Balancing або Amazon EC2 в якості серверів-джерел для додатків. При цьому, у разі використання таких джерел AWS, як Amazon S3, Amazon EC2 або Elastic Load Balancing, платити за передачу будь-яких даних між цими сервісами і сервісом CloudFront не потрібно [24].

Кешування статичних ресурсів

Amazon CloudFront дозволяє прискорити доставку статичного контенту (наприклад, зображень, таблиць стилів, скриптів JavaScript і т.д.) користувачам по всьому світу. CDN за замовчуванням пропонує багаторівневий кеш з регіональними периферійними серверами кешування, що дозволяють зменшити затримки і знизити навантаження на сервери-джерела, поки об'єкт ще не потрапив в кеш edge location (кеш периферійного розташування). Кешування статичного контенту забезпечує необхідну продуктивність і масштабування, щоб забезпечити швидку і надійну роботу при відвідуванні web-сервісу.

2.6.1. Послідовність кроків роботи Amazon CloudFront при доставці вмісту

Після налаштування CloudFront для доставки вмісту, відбувається наступна послідовність кроків, коли користувачі запитують файли:

1. Користувач здійснює доступ до веб-сервісу та запитує один або декілька файлів, наприклад зображення свого профілю (рис. 2.9);
2. DNS направляє запит до CloudFront POP (периферійного місцеположення), яке може здійснити найкраще обслуговування запиту – як правило, це найближчий POP CloudFront по показнику затримки;
3. У POP, CloudFront перевіряє кеш на потрібні файли. Якщо файли знаходяться в кеші (це називається кеш-влученням (англ. cache hit)),

CloudFront повертає їх користувачеві. Якщо файлів в кеші немає (Це називається кеш-промахом (англ. cache miss)), він робить наступне:

3.1. CloudFront порівнює запит із специфікаціями у distribution та пересилає запит на файли на початковий сервер (сервер-джерело) відповідного типу файлів – наприклад, у бакет Amazon S3 для файлів користувачів, а саме папки зображень профілів та папку профіля конкретного користувача;

3.2. Сервери-джерела відправляють файли назад у периферійне місцеположення;

3.3. Як тільки перший байт надходить з джерела, CloudFront починає пересилати файли користувачеві. CloudFront також додає файли в кеш у периферійній локації для наступного разу, коли хтось запитає ці файли.

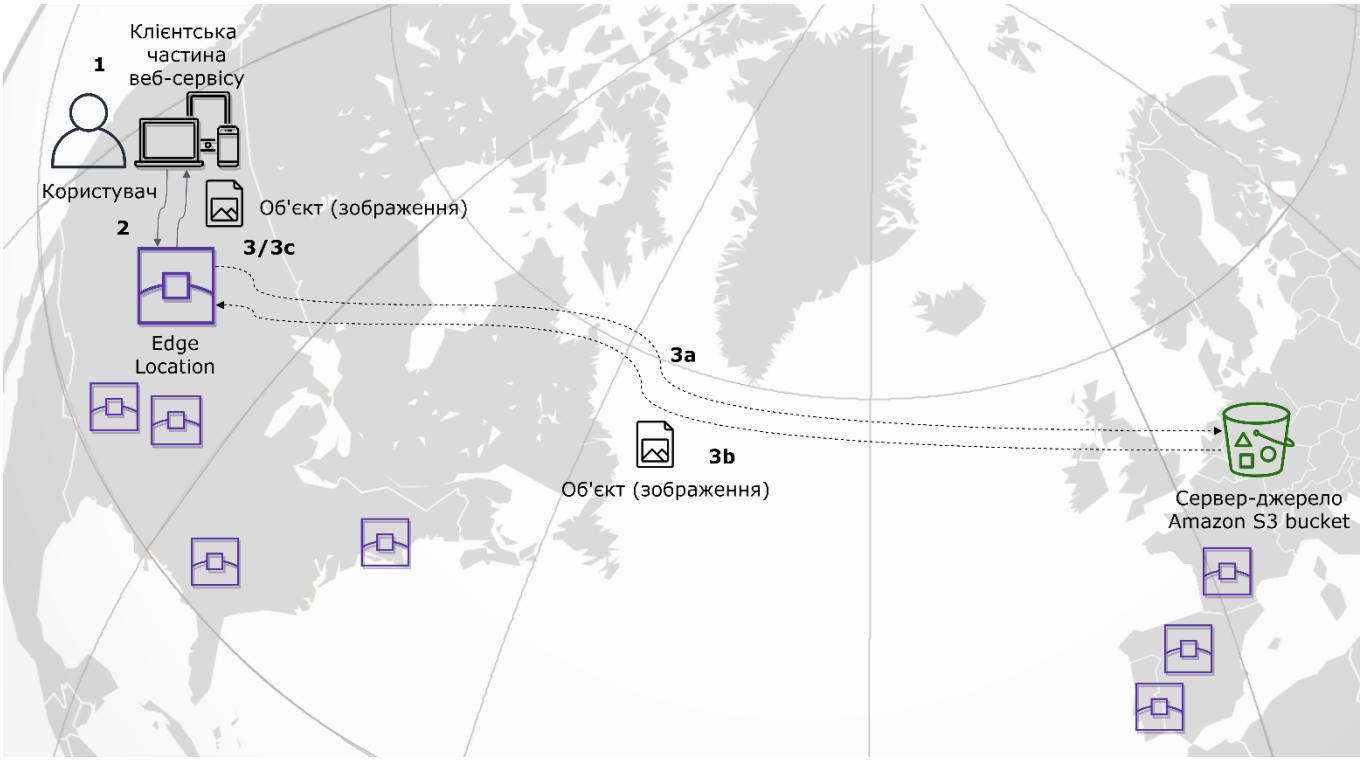

Рис. 2.9. Покрокова схема роботи Amazon Cloudfront

2.6.2. Invalidations

За звичайних умов, об’єкт у бакеті Amazon S3, що є частиною CloudFront distribution, може знаходитися у кеші CloudFront edge location протягом TTL (часу життя) об’єкта. У багатьох ситуаціях можливо заздалегідь визначити прийнятне значення TTL. В інших випадках може бути потреба у використанні переваг кешування CloudFront, але також може виникати необхідність змінювати об'єкт S3 у непередбачувані моменти часу.

Для цього в CloudFront є функція invalidation. Все що необхідно зробити – визначити список об’єктів у CloudFront distribution для вилучення з кешу і об’єкти будуть вилучені з усіх периферійних локацій (edge locations) протягом декількох хвилин. Invalidation відбувається асинхронно і декілька запитів Invalidation можуть знаходитись в очікуванні завершення обробки одночасно [25].

Наприклад може виникнути потреба видалити таблицю стилів, яка була пошкоджена на якомусь з етапів підготовки чи завантаження. Для цього в CloudFront Management Console створити invalidation з указанням шляху вилучення, в даному випадку – це шлях до файлу assets/css/style.css (рис. 2.10).

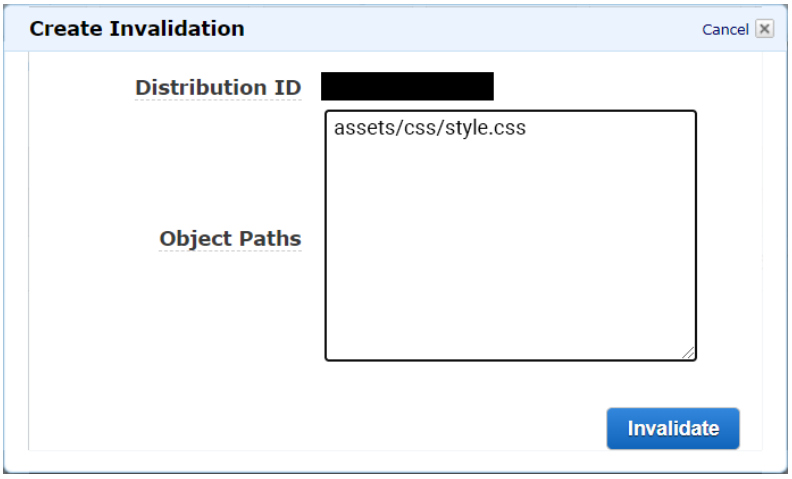

Рис. 2.10. Створення invalidation

Після натиснення кнопки “Invalidate” буде запущено процесс вилучення файлів із усіх периферійних серверів кешування. Наступного разу, коли від кінцевого користувача надійде запит на файл каскадної таблиці стилів, CloudFront порівняє запит із специфікаціями у distribution та перешле запит на файли на початковий сервер (сервер-джерело) відповідного типу файлів, щоб отримати останню версію файлу (рис. 2.11).

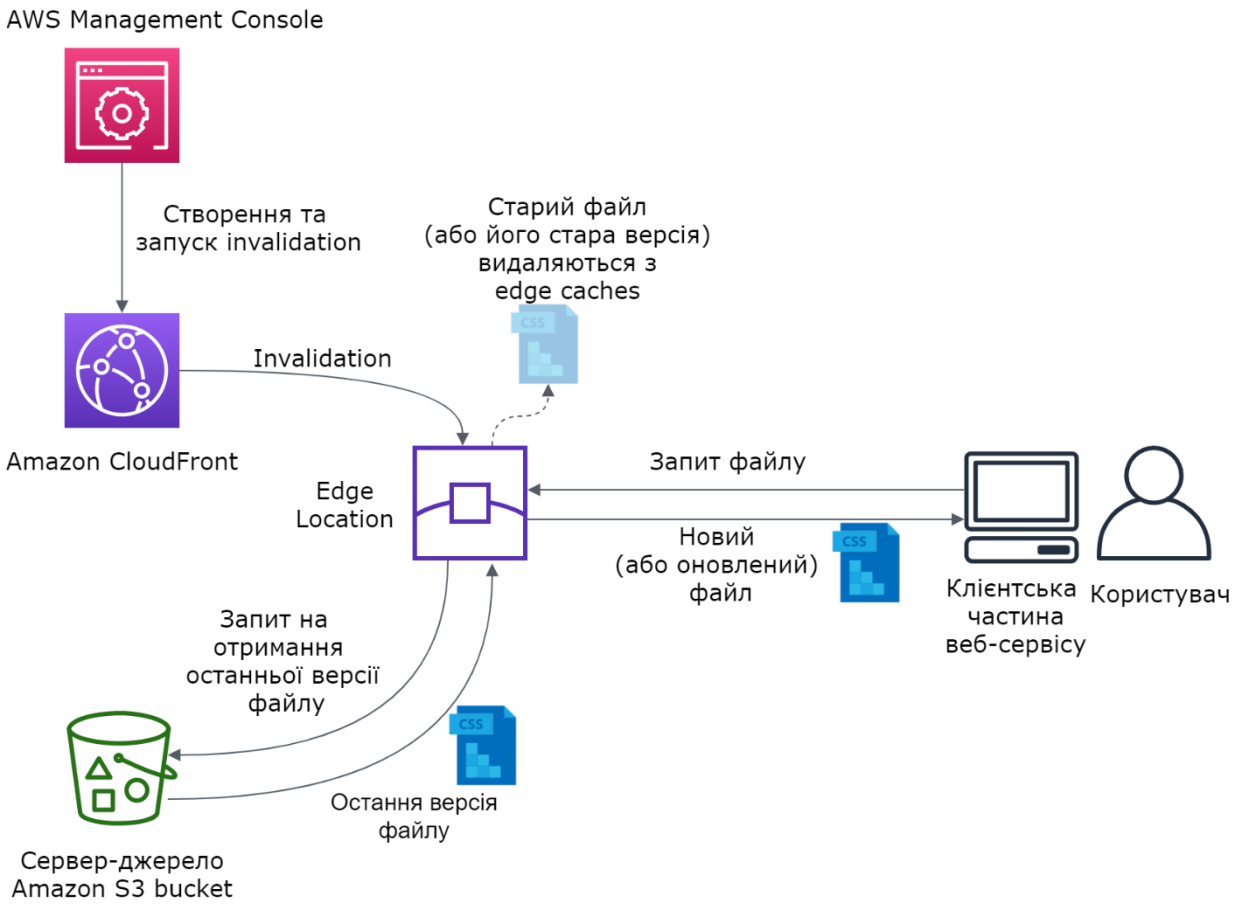

Рис. 2.11. Принцип роботи CloudFront Invalidation

2.7. Балансувальник навантаження

Під балансуванням навантаження фактично розуміється процес розподілу набору завдань за набором ресурсів (обчислювальних юнітів) з метою підвищення ефективності їх загальної обробки. Використання балансування навантаження

дозволяє досягнути як оптимізації часу відгуку обчислювальних юнітів, так і уникнути нерівномірного навантаження та перевантаження обчислювальних вузлів, тоді як інші обчислювальні вузли залишаються неактивними.

Балансування навантаження є предметом дослідження у сфері паралельних обчислень. Існує два основних підходи: статичні алгоритми, які не враховують стан різних машин, і динамічні алгоритми, які, як правило, є більш загальними та ефективнішими, але вимагають обміну інформацією між різними обчислювальними юнітами з ризиком втрат ефективності.

Для реалізації автомасштабування програмної системи було обрано балансувальник навантаження Elastic Load Balancer у складі екосистеми AWS.

Elastic Load Balancer (ELB) автоматично розподіляє вхідний трафік додатків між кількома цілями та віртуальними пристроями в одній або кількох зонах доступності (рис. 2.12) [26].

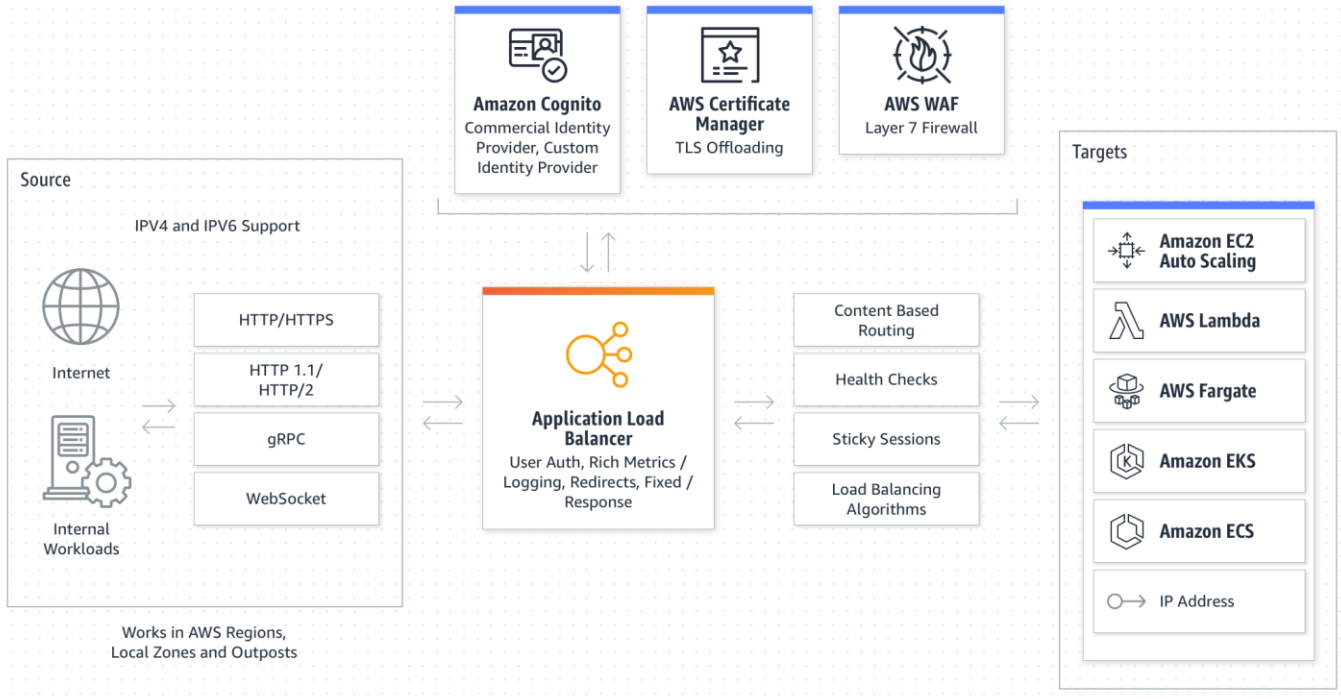

Рис. 2.12. Принцип роботи Elastic Load Balancer

За замовчуванням, балансувальник навантаження додатку направляє кожен запит незалежно до зареєстрованої цілі на основі вибраного алгоритму балансування

навантаження. Однак можна використовувати функцію sticky session (також відому як session affinity), щоб дозволити балансувальнику навантаження прив’язати сеанс користувача до певної цілі. Це гарантує, що всі запити від користувача під час сеансу надсилаються лише до однієї цілі. Ця функція корисна для серверів, які зберігають інформацію про стан, щоб забезпечити безперервний користувацький досвід для клієнтів [27]. Наприклад, користувач сервісу захоче переглянути інформацію про подані заявки, переходячи з віджету на сторінці Dashboard. Коли запит користувача досягне певного інстанс EC2, цей інстанс повинен отримати інформацію про користувача з даних стану, які повинні зберігатися глобально. У інстанса немає можливості кешувати будь-які дані, оскільки є висока ймовірність того, що кількість певних запитів від одного користувача/браузера буде зменшуватися при додаванні інстансів до пулу балансувальника навантаження. За допомогою опції sticky sessions можна дати балансувальнику навантаження інструкцію направляти повторювані запити до того самого інстанса EC2, коли це можливо (рис. 2.13) [28].

Рис. 2.13. Принцип роботи опції sticky sessions

В такому випадку інстанси можуть локально кешувати дані користувача для кращої продуктивності. Серія запитів від користувача буде перенаправлена на той самий інстанс EC2, якщо це можливо. Якщо інстанс було зупинено або він не пройшов нещодавню перевірку працездатності, балансувальник навантаження направить запит до іншого екземпляра. Щоб використовувати sticky sessions, клієнт повинен підтримувати cookie.

Балансувальники навантаження додатків підтримує як duration-based cookies, так і application-based cookies. Ключем до керування sticky sessions є визначення того, як довго балансувальник навантаження повинен послідовно направляти запит

користувача до до одного і того ж інстансу. Elastic Load Balancer створює файл cookie під назвою AWSELB, який використовується для зіставлення сеансу з відповідним інстансом [29].

2.8. Користувацький інтерфейс програмної системи

Для створення front end частини сервісу використовувались HTML5, CSS3, фронтенд-фреймворк Bootstrap версії 4.5.0, JavaScript, jQuery та Blade, бібліотеки Chart.js, ApexCharts.js, а також jQuery плагін Peity.js.

HTML визначає значення та структуру веб-контенту. Каскадні таблиці стилів (CSS) – це простий механізм додавання стилів (наприклад, шрифтів, кольорів, інтервалу) до веб-документів. Bootstrap дозволяє швидко розробляти та кастомізувати адаптивні веб-сайти використовуючи адаптивну сіткову систему, широкий набір вбудованих компонентів та потужні плагіни JavaScript. Динаміку забезпечують JavaScript та jQuery.

Blade – це простий, але потужний рушій обробки шаблонів, забезпечений Laravel. На відміну від інших популярних рушіїв обробки шаблонів PHP, Blade не обмежує розробників щодо використання звичайного PHP-коду у представленнях. Насправді всі представлення Blade компілюються у звичайний PHP-код і зберігаються в кешованому стані, доки вони не будуть змінені, тобто Blade додає фактично нульові накладні витрати до створюваного додатку. Файли представлень Blade використовують розширення файлу .blade.php і зазвичай зберігаються в папці resources/views додатку [30].

Laravel Routing дозволяє призначити контролер на відповідний запит за URL-адресою. Таким чином, коли користувач перейде за певною URL-адресою, буде викликано, встановлений на цю адресу, контролер, який поверне представлення для цієї адреси. Усі маршрути Laravel визначені у файлах route, які знаходяться в папці routes.

Ці файли автоматично завантажуються фреймворком. Файл routes/web.php визначає маршрути, призначені для веб-інтерфейсу. Цим маршрутам призначається

web middleware group, яке забезпечує такі функції, як стан сесії та захист CSRF. Маршрути в routes/api.php є stateless та присвоюються api middleware group[34].

Для більшості додатків все починається з визначення маршрутів у файлі routes/web.php. До маршрутів, визначених у routes/web.php, можна отримати доступ, ввівши визначену URL-адресу маршруту у браузері.

Представлення містять HTML, поданий сервісом, і відокремлюють логіку контролера/програми від логіки представлення. Представлення зберігаються в каталозі resources/views[35].

Наприклад представлення зберігається в resources/views/home.blade.php, таким чином ми можемо повернути його користувачу за допомогою global helpers для представленнь (рис. 2.14).

```
return view( view: 'home', compact( var_name: 'data'));
```

Рис. 2.14. Повернення представлення home.blade.php

Перший аргумент, переданий view helper, відповідає імені файлу представлення в каталозі resources/views. Другий аргумент - це масив даних, який повинен бути доступний для перегляду. У цьому випадку ми передаємо змінну *data*, яка відображається у представленні за допомогою синтаксису Blade.

Двома основними перевагами використання Blade є спадкування шаблонів та секції.

При визначенні дочірнього представлення, використовується директива Blade @extends, щоб вказати, який макет дочірнє представлення повинне «успадкувати». Представлення, які розширюють макет Blade, можуть вводити вміст у розділи макета за допомогою директив @section (рис. 2.15) [30].

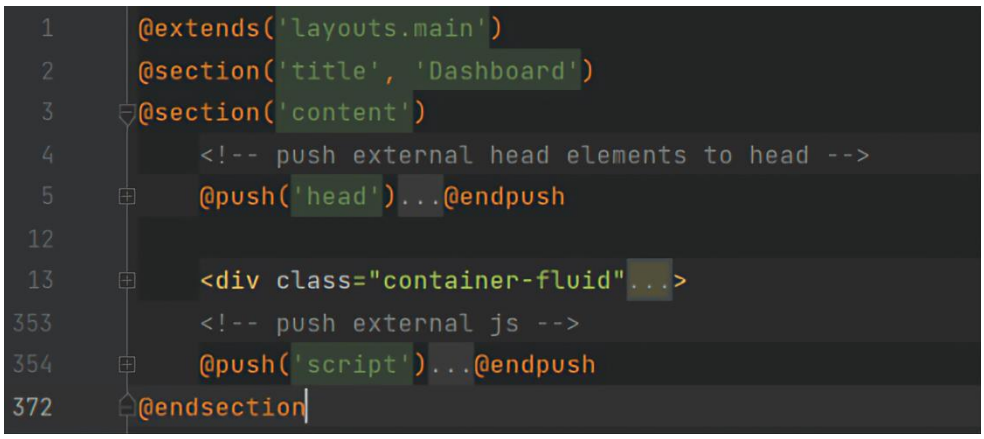

Рис. 2.15. Представлення, оформлене з використанням Blade

2.8.1. Використання фреймворку Bootstrap

Одна з переваг, яку надає Bootstrap – це можливість реалізувати усталений вигляд. Прямо з коробки доступний набір елементів і компонентів інтерфейсу користувача, які охоплюють практично кожен варіант використання. В загальному випадку практично не має значення під яку конкретну систему чи сервіс потрібно імплементувати користувацький інтерфейс – фреймворк дає значну перевагу в дизайні.

Крім того, Bootstrap є розширюваним. Він створений з урахуванням кастомізації, що робить процеси зміни різних компонентів для задоволення тих чи інших вимог досить простим. Також можливе створення власних компонентів.

Цей рівень зручності та гнучкості, безсумнівно, зіграв велику роль у швидкості та ефективності реалізації користувацького інтерфейсу програмної системи візуалізації статистики web-сервісу пошуку роботи для користувачів.

Одна з найзручніших функцій Bootstrap – система сітки. Використання потужної mobile-first сіткової системи (рис. 2.16), базованої на flexbox надає можливості для створення макетів будь-якої форми та розміру завдяки системі дванадцяти стовпців, п’яти адаптивним рівням за замовчуванням, змінним і комбінаціям Sass та десяткам попередньо визначених класів.

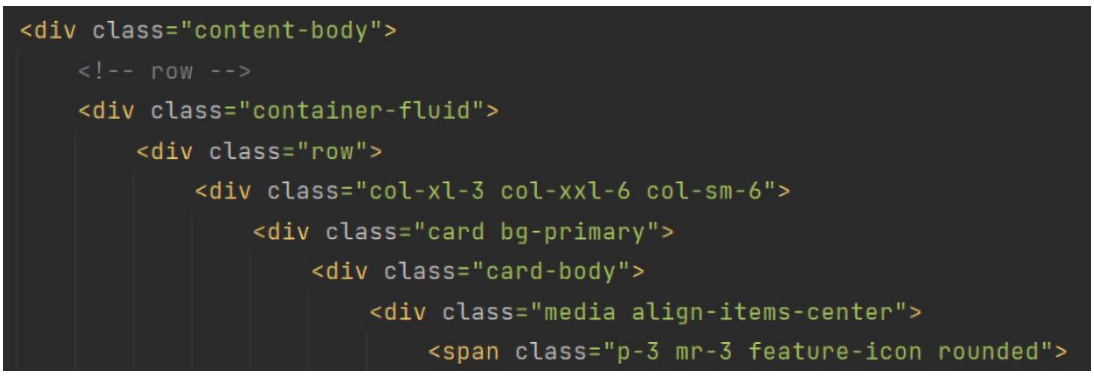

Рис. 2.16. Використання Bootstrap в програмній системі візуалізації статистики сервісу пошуку роботи для користувачів

Прниципп роботи сітки Bootstrap полягає в наступному:

- Контейнери забезпечують можливість центрування та горизонтального розміщення вмісту веб-сайту. Рекомендується використовувати “.container” для адаптивної піксельної ширини або “.container-fluid” для ширини: 100% для всіх viewport та розмірів пристроїв.
- Рядки - це обгортки для колон. Кожен стовпець має horizontal padding (який називається gutter) для контролю простору між ними. Потім цей padding протидіє рядкам з від’ємними margins. Таким чином, весь вміст у стовпцях візуально вирівнюється по лівій стороні.
- У макеті сітки вміст повинен розміщуватися всередині стовпців, і лише стовпці можуть бути безпосередніми нащадками рядків.
- Завдяки flexbox стовпці сітки без заданої ширини автоматично розміщуватимуться як стовпці однакової ширини. Наприклад, чотири екземпляри “.col-sm” автоматично матимуть ширину 25% від малої контрольної точки і вище.
- Класи стовпців вказують кількість стовпців, які використовуватимуться з можливих 12 на рядок. Отже, якщо потрібно зробити три однакові по ширині колонки, можна використовувати “.col-4”.
- Ширина стовпців встановлюється у відсотках, тому вони завжди є «пластичними» та за розміром встановлюються відносно батьківського елемента.

- Стовпці мають horizontal padding для створення gutter між окремими стовпцями, однак можна видалити margin з рядків і padding з стовпців “.no-gutters” на “.row”.
- Щоб зробити сітку гнучкою, існує п’ять контрольних точок сітки, по одній для кожної адаптивної контрольної точки: усі контрольні точки (надмаленькі), малі, середні, великі та надвеликі.
- Контрольні точки сітки базуються на медіа-запитах мінімальної ширини, тобто вони застосовуються до цієї однієї контрольної точки та всіх над нею (наприклад, “.col-sm-4” застосовується до малих, середніх, великих та надвеликих пристроїв, але не до першої контрольної точки ‘xs”).
- Можна використовувати заздалегідь визначені класи сітки (наприклад “.col-4”) або комбінації Sass для більш семантичної розмітки [16].

Важливо в той же час пам’ятати про обмеження та помилки навколо flexbox, як-от неможливість використовувати деякі елементи HTML як flex-контейнери.

2.8.2. Використання інструменту для побудови графіків Chart.js

Chart.js – простий, але гнучкий інструмент для побудови графіків на JavaScript для дизайнерів та розробників. Можна виділити наступні характеристики та функції:

- Змішані типи діаграм – можна змішувати та поєднувати стовпчасті та лінійні діаграми, щоб забезпечити чітке візуальне розрізнення між наборами даних;
- Нові типи осей діаграми дозволяють будувати графіки складних, розріджених наборів даних за часом, логарифмічними або навіть цілком власними масштабами з легкістю (рис. 2.17);
- Анімація всього – одразу готові приголомшливі переходи під час зміни даних, оновлення кольорів та додавання наборів даних;
- Відкритий сирцевий код. Chart.js - проект, який підтримується спільнотою;
- 8 типів діаграм дають можливість візуалізувати дані 8 різними способами; кожен з них анімований та настроюваний;

- HTML5 Canvas – відмінна ефективність рендерингу у всіх сучасних браузерах (IE11+);
- Адаптивність – Chart.js перемальовує діаграми при зміні розміру вікна, щоб забезпечити чудову глибину деталізації в будь-якому масштабі [31].

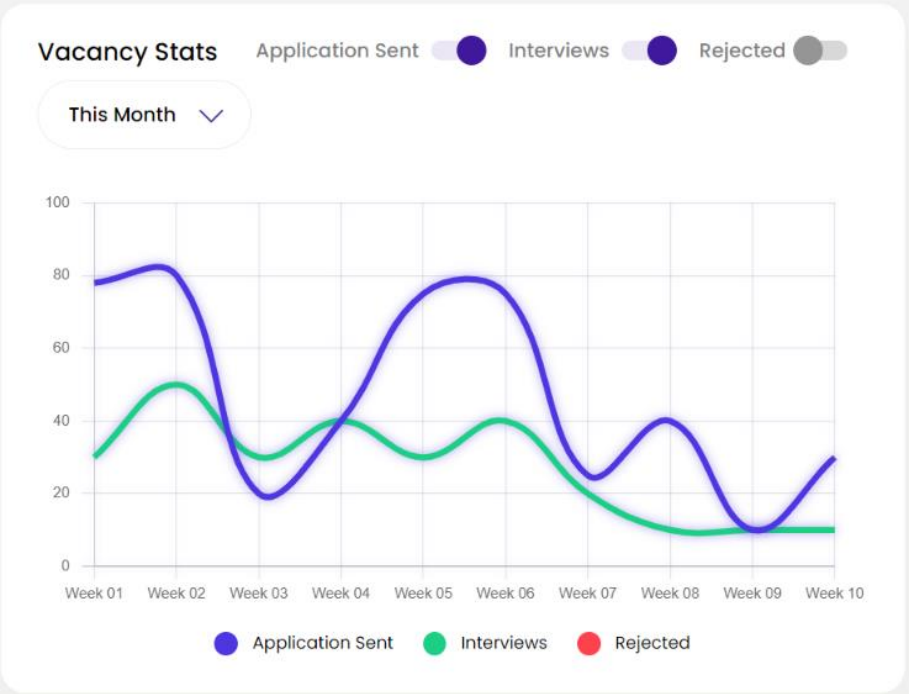

Рис. 2.17. Використання Chart.js в програмній системі візуалізації статистики web-сервісу пошуку роботи для користувачів

2.8.3. Використання бібліотеки для побудови графіків ApexCharts.js

ApexCharts.js – це сучасна бібліотека графіків, яка допомагає розробникам створювати красиві та інтерактивні візуалізації для веб-сторінок. Це проект з відкритим кодом, ліцензований під MIT, і його можна безкоштовно використовувати в комерційних додатках. Завдяки обширній документації API та більш ніж 100 готовим до використання зразкам, інтеграція візуалізації з ApexCharts стає настільки простою, наскільки це можливо.

Серед основних функцій:

- Підтримка декількох типів діаграм – можна обирати з широкого діапазону діаграм або навіть створювати комбінації різних діаграм для забезпечення чіткої різниці між даними;
- Інтерактивність – масштабування, панорамування, прокрутка даних, перемикання видимості наборів даних у декількох серіях, показ інформативних підказок, коли користувач наводить курсор на точки даних. Кожен із цих варіантів допомагає ефективніше передавати дані (рис. 2.18);
- Динамічність – динамічний характер ApexCharts дозволяє завантажувати дані по виділенню та створювати інші діаграми на основі цих виділень. Іншими словами, це той функціонал, який робить дані по-справжньому інтерактивними;
- Повна адаптивність – ApexCharts гнучкі та адаптивні, завдяки чому графіки працюють як на настільних ПК, планшетах, так і на мобільних телефонах;
- Висока настроюваність – налаштування ApexCharts дуже просте. Вичерпна документація допоможе швидко налаштувати діаграми;
- Плавна анімація – ApexCharts забезпечує плавний інтерактивний користувацький досвід при зміні наборів даних, завантаженні динамічних даних та взаємодії з діаграмами;
- Стилізація елементів діаграми – можна додавати градієнти, зображення, візерунки та тіні до елементів діаграми та надавати діаграмам індивідуальний вигляд [32].

Рис. 2.18. Використання бібліотеки ApexCharts.js в програмній системі візуалізації статистики web-сервісу пошуку роботи для користувачів

2.8.4. Використання інструменту для побудови інфографіки Peity.js

Peity.js – це плагін jQuery, який перетворює вміст елемента в <svg> міні-секторну, кільцеву (рис. 2.19) або стовпчасту діаграму та сумісний з будь-яким браузером, що підтримує <svg>: Chrome, Firefox, IE9+, Opera, Safari [33].

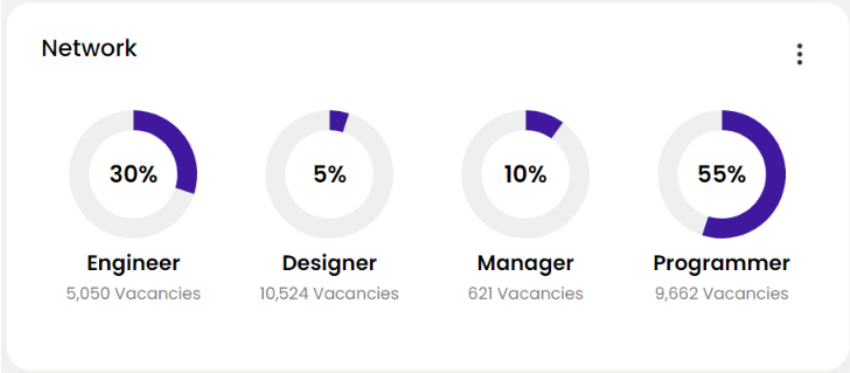

Рис. 2.19. Використання інструменту Peity.js в програмній системі візуалізації статистики web-сервісу пошуку роботи для користувачів

Даний інструмент є достатньо потужним та водночас простим у використанні, дозволяючи створювати різні типи діаграм з використанням зручного синтаксису.

2.8.5. Сторінка Dashboard

Спершу розглянемо функціональне призначення сторінки Dashboard. Dashboard (інформаційна панель) – це візуальне відображення основних даних. Незважаючи на те, що її можна використовувати різними способами, її основна мета полягає в тому, щоб моментально надавати інформацію, як-от KPI.

Інформаційні панелі дозволяють різним професіоналам контролювати ефективність роботи, створювати звіти та встановлювати оцінки та цілі для майбутньої роботи.

Інші переваги включають:

- Візуальне представлення продуктивності, наприклад, за допомогою діаграм і графіків;
- Уміння визначати тенденції;
- Простий спосіб вимірювання ефективності;
- Засіб створення детальних звітів одним кліком;
- Здатність приймати більш обґрунтовані рішення;
- Повна видимість усіх систем, кампаній і дій;
- Швидка ідентифікація викидів даних і кореляцій.

Інформаційна панель розташована на відповідній сторінці та отримує інформацію із пов’язаної бази даних. Dashboard можна налаштувати, що дає користувачам можливість вибирати, які дані вони хочуть бачити, і чи хочуть вони вмикати відображення діаграм чи графіків для візуалізації чисел.

Верхня навігаційна панель дає можливість згортати та розгортати бічну навігаційну панель, здійснювати пошук, переглядати нотатки / попередження / чат по кліку мишею на іконку повідомлень. За наявності не переглянутих попереджень / повідомлень в чаті в правому верхньому кутку над іконкою повідомлення відображається їх кількість. Клік мишею по іконці дзвоника відкриває спадне меню

з сповіщеннями, отриманими користувачем в його обліковому записі. За наявності не переглянутих сповіщень в правому верхньому кутку над іконкою дзвоника відображається їх кількість. Клік мишею по імені або зображенню профілю користувача відкриває спадне меню, завдяки якому користувач може отримати доступ до сторінки Profile, пошти, а також здійснити вихід з облікового запису.

Бічна навігаційна панель надає можливість переходити на головну сторінку Dashboard по кліку мишею по логотипу сервісу та забезпечує опції вибору різних сторінок всього сервісу пошуку роботи — Dashboard, Search Job, Applications, Profile, Companies, Statistics, що відповідно представляють головну сторінку сервісу, сторінку пошуку роботи, сторінку поданих заявок, сторінку профіля користувача, сторінку компаній та сторінку статистики. Також, з бічної навігаційної панелі може бути доступний перехід до підсервісів пошти та календаря в залежності від ролі користувача.

На цій сторінці користувачу презентується його статистика – першим згори розташований блок інфографіки з картками, що відображають кількість запланованих співбесід, надісланих заявок, переглядів його профілю та кількість непрочитаних повідомлень від HR-спеціалістів та роботодавців (рис. 2.20).

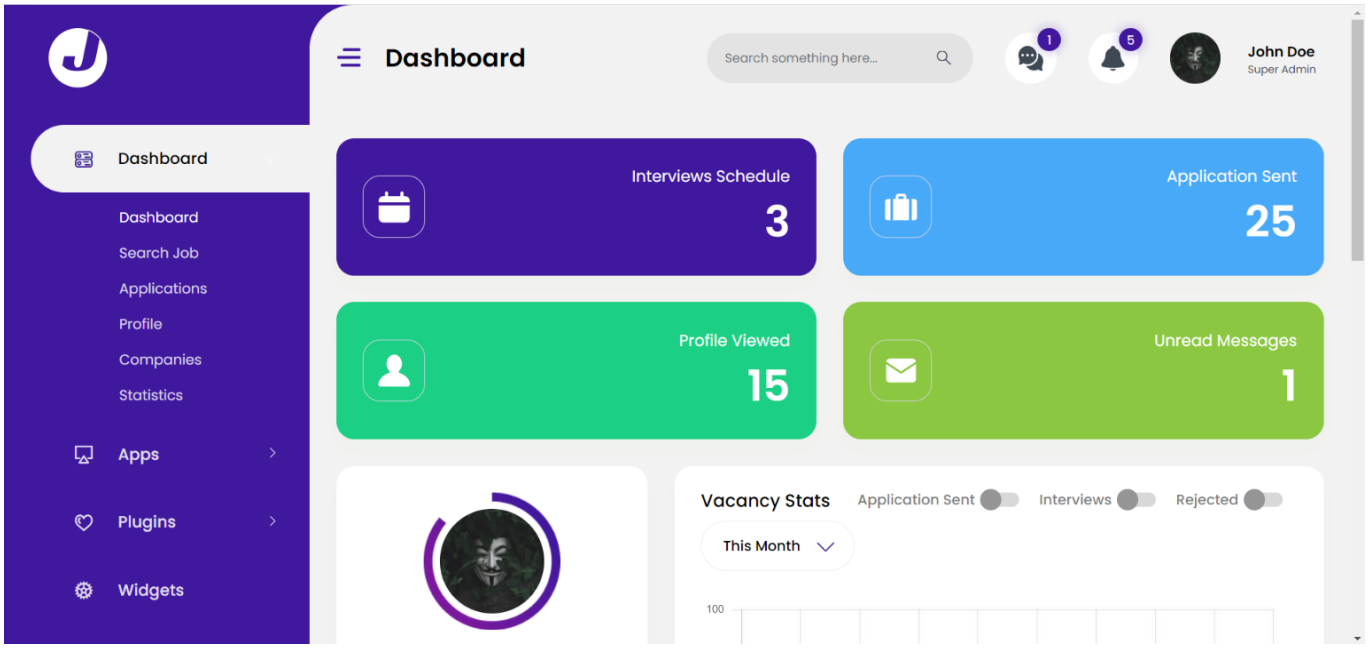

Рис. 2.20. Сторінка Dashboard

При скроллі вниз одразу після блоку інфографіки розташовані:

1. Комбінований блок візуалізованих даних профілю користувача (фотографія профілю, професія та рівні основних професійних навичок) та нещодавньої активності;

2. Блок візуалізованої статистики по вакансіям з інтерактивними графіками, що можуть відображати кількість надісланих заявок, кількість запланованих співбесід та кількість відмов за різні періоди часу;

3. Блок рекомендованих вакансій, що містить картки вакансій з назвою компанії, її логотипом, назвою вакансії, пропонованим рівнем або «вилкою» заробітної плати (діапазоном окладу), коротким описом вакансії, розташуванням місця роботи та ключовими тегами

4. Блок обраних компаній, що містить картки компаній з логотипом компанії, назвою та кількістю вакансій;

5. Футер сторінки, що містить інформацію про авторські права.

Скріншот повної сторінки Dashboard наведено в розділі 1 додатку Б.

2.8.6. Сторінка Statistics

Має ті самі верхню та бічну панелі навігації, що й сторінка Dashboard. Секція основного вмісту розділена на дві колонки (рис. 2.21).

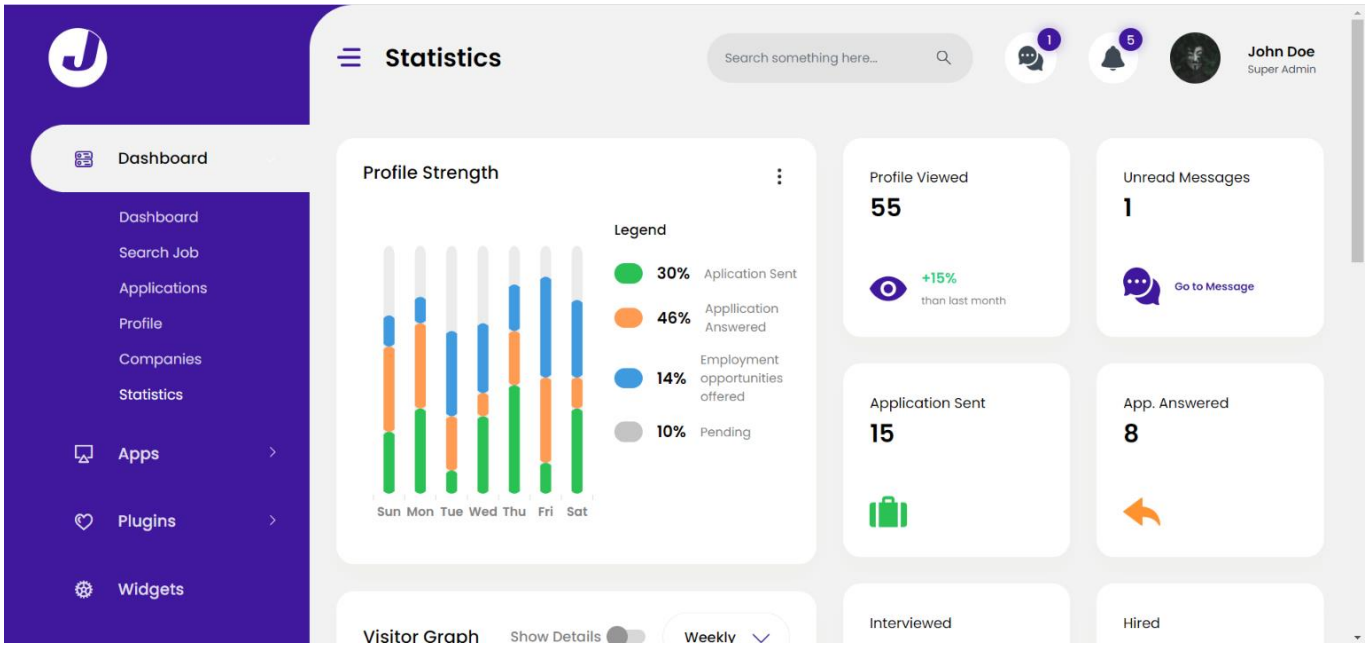

Рис. 2.21. Сторінка Statistics

В лівій частині розташовується колонка з комбінованим блоком візуалізованої статистики «потужності профілю» – стовпчаста діаграма з накопиченням, що може відображати кількості надісланих заявок, заявок на які було отримано відповідь, заявок по яким в результаті було отримано job offer та заявок що очікують на розгляд. Нижче відображається блок з візуалізованою статистикою відвідування – точкова діаграма з плавними лініями, що може відображати кількість переглядів профілю та кількість отриманих пропозицій розглянути вакансію за різні періоди часу.

В правій частині – колонка, вгорі якої відображається блок візуалізованих даних з картками, що відображають кількість переглядів профілю з порівнянням даних за минулий місяць, кількість непрочитаних повідомлень від HR-спеціалістів та роботодавців, кількість надісланих заявок, кількість заявок на які було отримано відповідь, а для профілів користувачів з роллю «HR-спеціаліст» додатково відображаються картки з кількістю проведених співбесід по отриманим заявкам та кількістю найнятих працівників. Нижче відображаються блоки складової професійної соціальної мережі для пошуку і встановлення ділових контактів. Перший блок містить кількість відстежуваних компаній та людей, що відстежують

профіль з відповідною візуалізацією у вигляді кільцевої діаграми. Другий блок відображає розподіл мережі ділових контактів за професіями з відповідною візуалізацією у вигляді кільцевих діаграм та кількість доступних вакансій для кожної професії.

Сторінка також містить футер, де вказана інформація про авторські права (копірайт) та посилання на профілі автора-розробника web-сервісу в соціальних мережах.

Скріншот повної сторінки Statistics наведено в розділі 2 додатку Б.

Висновок до розділу 2

У другому розділі було визначено значення архітектури та її складових для системи. Розглянуто та охарактеризовано архітектуру та структуру системи, особливості її реалізації. Досліджено роботу php-фреймворку Laravel та сервісів AWS – S3, DynamoDB, CloudFront та Elastic Load Balancing у контексті конкретних задач та потреб програмної системи візуалізації статистики сервісу пошуку роботи для користувачів.

Розкрито принципи функціонування компонентів графічного користувацького інтерфейсу у системі. Досліджено використання фреймворку Bootstrap, використання інструменту для побудови графіків Chart.js, бібліотеки для побудови графіків ApexCharts.js та інструменту для побудови інфографіки Peity.js в програмній системі візуалізації статистики web-сервісу пошуку роботи для користувачів. Розглянуто основні сторінки сервісу, їх елементи та вміст, описано доступний функціонал.

РОЗДІЛ 3
АНАЛІЗ ПОКАЗНИКІВ ТА ХАРАКТЕРИСТИК ОТРИМАНОЇ ПРОГРАМНОЇ СИСТЕМИ

Успіх всього програмного забезпечення значною мірою залежить від ряду показників – серед яких показники безпеки, масштабованості, надійності, гнучкості, швидкодії, зручності використання та функціональності.

3.1. Аналіз безпеки програмної системи

Безпека програмного забезпечення є центральним компонентом та одним з головних пріоритетів будь-якого бізнесу, що веде свою діяльність в веб-середовищі. Глобальний характер Інтернету піддає веб-об’єкти атакам з різних місць та різних рівнів масштабу та складності.

Атаки на веб-додатки варіюються від цільової маніпуляції з базами даних до масштабних збоїв у мережі. Серед поширених методів нападу або «векторів», що часто використовуються: Cross site scripting (XSS) – «Міжсайтовий скриптинг», SQL ін'єкція, Denial-of-service (DoS) та distributed denial-of-service (DDoS) атаки, атаки типу Cross-site request forgery (CSRF) – «Міжсайтова підробка запиту».

Laravel має деякі вбудовані механізми та засоби безпеки, які надають захист від типових вразливостей додатків.

Захист від ін'єкції SQL. Laravel використовує PDO binding для запобігання атак ін'єкцій SQL, оскільки жодна змінна не передається до бази даних без валідації.

Захист від CSRF. Laravel дозволяє легко захистити додаток від CSRF-атак. Міжсайтова підробка запиту – це тип зловмисного експлойту, за допомогою якого виконуються несанкціоновані команди від імені автентифікованого користувача.

Laravel автоматично генерує «токен» CSRF для кожного активного сеансу користувача, керованого додатком. Цей токен використовується для підтвердження того, що аутентифікований користувач є тим, хто насправді робить запити до додатку.

Захист від XSS. Laravel запобігає міжсайтовому скриптингу, оскільки його синтаксис "@{{}}" допомагає автоматично перетворювати об'єкти HTML, які можуть бути частиною змінної представлення.

Окрім вищезазначеного, у Laravel вже встановлений надійний процес аутентифікації користувача з відповідним шаблоном кодогенерації. Laravel використовує providers та guards для полегшення процесу аутентифікації. Мета guards – аутентифікація користувачів для кожного запиту, який вони роблять, в той час як providers забезпечують отримання користувачів із бази даних.

У Laravel є два основні криптографічні фасади: Crypt (симетричне шифрування) та Hash (одностороннє криптографічне хешування). Паролі хешуються, а файли cookie (опціонально) шифруються. Таким чином, за замовчуванням, у Laravel є механізми забезпечення безпеки для паролів та файлів cookie [16].

AWS відповідає за захист інфраструктури, на якій працюють сервіси AWS у AWS Cloud. Ефективність безпеки регулярно тестується та верифікується сторонніми аудиторами як частина програм відповідності AWS.

Усі дані, які зберігаються в DynamoDB шифруються за замовчуванням. Рівень безпеки даних зростає завдяки шифруванню з використанням ключів шифрування, що зберігаються в AWS Key Management Service. Шифрування даних, що зберігаються в DynamoDB забезпечує високий рівень безпеки, відповідний жорстким нормативним та законодавчим вимогам в сфері шифрування. [36]

При розробці програмного забезпечення використовувались методи валідації та екранування введених даних, хешування паролів з використанням алгоритму CRYPT_BLOWFISH та шифрування файлів cookie, а також інші вбудовані засоби та механізми Laravel та усіх сервісів AWS для забезпечення максимального рівня безпеки як на програмному рівні, так і на рівні інфраструктури.

Враховуючи всі вищезазначені факти, можна стверджувати що web-сервіс має достатньо високий рівень безпеки.

3.2. Аналіз ефективності архітектурних рішень

Загалом, комбінація декількох архітектурних стилів, серед яких клієнт-серверна архітектура, специфічна мікроархітектура Laravel та хмарна архітектура, як і очікувалося, виявилася вдалим рішенням, оскільки дозволила забезпечити одночасно високу швидкість роботи, продуктивність, потужність, надійність, гнучкість та масштабованість програмної системи у формі web-сервісу. Laravel – це найбільш швидкий, потужний та продуктивний php-фреймворк, тоді як web-сервіси Amazon є «золотим стандартом», коли мова йде про хмарні послуги.

При реалізації web-сервісу використовувався патерн MVC, серед переваг якого можна виділити наступні:

- Висока когезія (англ. high cohesion) – MVC дозволяє логічно групувати пов'язані дії на контролері. Види (представлення) для конкретної моделі також групуються;
- Слабке зчеплення (англ. loose coupling) – сама природа MVC така, що зчеплення між моделями, видами чи контролерами є слабким;
- Простота модифікації – через поділ відповідальності, майбутній розвиток чи модифікація простіші;
- Кілька представлень для моделі – моделі можуть мати декілька представлень;
- Тестовність – за чіткішого поділу обов’язків, кожна частина може бути краще протестована незалежно (наприклад, тестування моделі без необхідності створювати заглушку для представлення).

Примітка: Когезія (англ. cohesion) – це показник зв’язності всередині модуля. Зчеплення - це показник зв’язності між модулями. Правильно спроектоване програмне забезпечення має високу когезію і слабке зчеплення.

Недоліки MVC в цілому можна категоризувати як накладні витрати для програмного забезпечення з неправильною декомпозицією. Інакше кажучи, для додатків, в яких загальне проектування та декомпозиція були здійснені правильно, наведені нижче недоліки можуть проявлятися незначною мірою або не проявлятися зовсім [16]. Загалом, перелік недоліків можна визначити як такий:

- Багато-артефактна узгодженість – декомпозиція функціональних можливостей на три артефакти викликає розсіювання. Це, в свою чергу, вимагає від розробника підтримки узгодженості декількох представлень одночасно;
- Надмірний шаблон кодогенерації – завдяки тому, що обчислення програми та стан зазвичай кластеризовані в одну з 3-х частин, інші частини вироджуються або в частини шаблону кодогенерації, або в супроводжуючий код, що існує лише для задоволення схеми MVC;
- Виражена крива навчання – знання декількох різних технологій стає нормою. Розробники, що використовують MVC, мають бути кваліфікованими у кількох технологіях.

Із недоліків під час реалізації проявився лише перший недолік, при чому лише частково, що дозволяє стверджувати що загальне проектування та декомпозиція була здійснені правильно.

3.3. Аналіз ефективності та масштабованості програмної системи

Питання масштабованості web-сервісу фактично є питанням створення серверної архітектури та горизонтального масштабування. Тестування web-сервісу під час розробки відбувалося на локальному сервері. Все що необхідно зробити для деплойменту production-версії на Amazon EC2 зводиться до певної послідовності кроків.

Спочатку – підключити сервіс балансування навантаження перед інстансами EC2, де розміщені Laravel-додатки web-сервісу. Таким чином можна буде підтримувати будь-яке число Laravel-додатків (тобто копій web-сервісу на різних інстансах). Далі необхідно налаштувати балансувальник для використання Sticky

Sessions, щоб спрямовувати повторні запити до того самого інстансу EC2, коли це можливо. У цьому випадку екземпляри можуть кешувати дані користувачів локально для кращої продуктивності.

Наступним кроком буде направлення доменної адреси на балансувальник навантаження.

В результаті, можна буде додавати нові інстанси EC2 до балансувальника навантаження за потреби обробки більшої кількості трафіку.

Горизонтальне масштабування сервісів AWS здійснюється автоматично.

3.4. Аналіз ефективності використання та масштабованості NoSQL БД Amazon DynamoDB

Спочатку розглянемо ефективність NoSQL рішень**.** Основні відмінності між SQL та NoSQL полягають в продуктивності, доступності, гнучкості та масштабованості з перемогою NoSQL рішень.

Бази даних NoSQL побудовані для специфічних моделей даних та мають гнучкі (динамічні) схеми і дозволяють використовувати «неструктуровані дані». По суті це означає, що можна створити програму без попереднього визначення схеми БД. У реляційній базі даних необхідно визначити схему перед додаванням даних у систему. Відсутність заздалегідь визначеної схеми дозволяє значно легше оновлювати бази даних NoSQL коли дані та вимоги змінюються. Зміна структури схеми в реляційній базі даних може бути надзвичайно дорогим та трудомістким процесом і часто тягне за собою простої або перебої в обслуговуванні.

NoSQL дозволяє швидше та більш гнучко зберігати та обробляти дані, що робить відповідні БД кращим варіантом для сучасних додатків, які мають більш складні, набори даних, що можуть постійно змінюватися, а тому вимагають гнучкої моделі даних без необхідності попереднього визначення. На відміну від традиційних, реляційних баз даних на основі SQL, бази даних NoSQL можуть зберігати та обробляти дані в режимі реального часу.

Ідея NoSQL полягає в спрощенні шаблонів запитів та шаблонів доступу. Як результат – висока ефективність та економічність рішення.

Хоча в базах даних SQL все ще є деякі конкретні випадки використання, NoSQL бази даних мають багато функціональних можливостей, з якими БД SQL не здатні впоратися без величезних витрат і критичних пожертвувань швидкістю, оперативністю тощо.

Реляційні бази даних мають можливість вертикального масштабування, але зазвичай ця опція є досить дорогою. Оскільки їм потрібен єдиний сервер для розміщення всієї бази даних, для здійснення масштабування потрібно купувати більший (з точки зору апаратних ресурсів – обчислювальних потужностей, місткостей сховищ і т.д.), а відповідно і дорожчий сервер. Бази даних NoSQL надають значно вигідніше рішення, оскільки по-перше є можливість горизонтального масштабування, навіть на інфраструктурі, що складається з дешевих типових серверів, а по-друге – можна використати денормалізовану модель даних для забезпечення простих шаблонів доступу до БД [16].

DynamoDB не потребує розподілення будь-яких серверів, встановлення на них фіксів або керування ними. Крім того, відпадає необхідність в установці, обслуговуванні і використанні відповідного програмного забезпечення. DynamoDB автоматично масштабує таблиці, коригуючи обсяг доступних ресурсів і зберігаючи високу продуктивність. Вбудовані механізми забезпечення доступності та відмовостійкості усувають необхідність в проектуванні цих можливостей в рамках програми.

DynamoDB еластично масштабується для обробки будь-яких навантажень і забезпечення низької затримки, що критично важливо для програмної системи візуалізації статистики сервісу пошуку роботи для користувачів.

Використання DynamoDB для вирішення завдань масштабування дає змогу зосередитися на розробці нових можливостей, а не управлінні базами даних.

Враховуючи вищезазначене, можна стверджувати що використання Amazon DynamoDB є ефективним рішенням.

3.6. Аналіз юзабіліті користувацького інтерфейсу

Користувацький інтерфейс створеної програмної системи має чітку ієрархію дизайну, що означає, що інформація чітко організована з точки зору важливості та легко доступна та зрозуміла. Крім того, користувацький інтерфейс втілює візуальний настрій або естетику, яка має значення для людей, які з ним взаємодіють.

Основними принципами дизайну інтерфейсу користувача є ясність, гнучкість, ознайомленість, ефективність і послідовність. З них ясність є метою номер один. Ясність дизайну інтерфейсу користувача дає можливість користувачам мати впевнену взаємодію та легко знаходити потрібну інформацію.

Користувацький інтерфейс є інтуїтивно ясним та простим. Створений з використанням стандартів HTML5 та CSS3, а також найпопулярнішого frontend-фреймворку Bootstrap версії 4.5.0 для створення реактивних, та в першу чергу мобільних сайтів, він забезпечує комфортну роботу на будь-якому пристрої – від настільних ПК до смартфону (рис. 3.1).

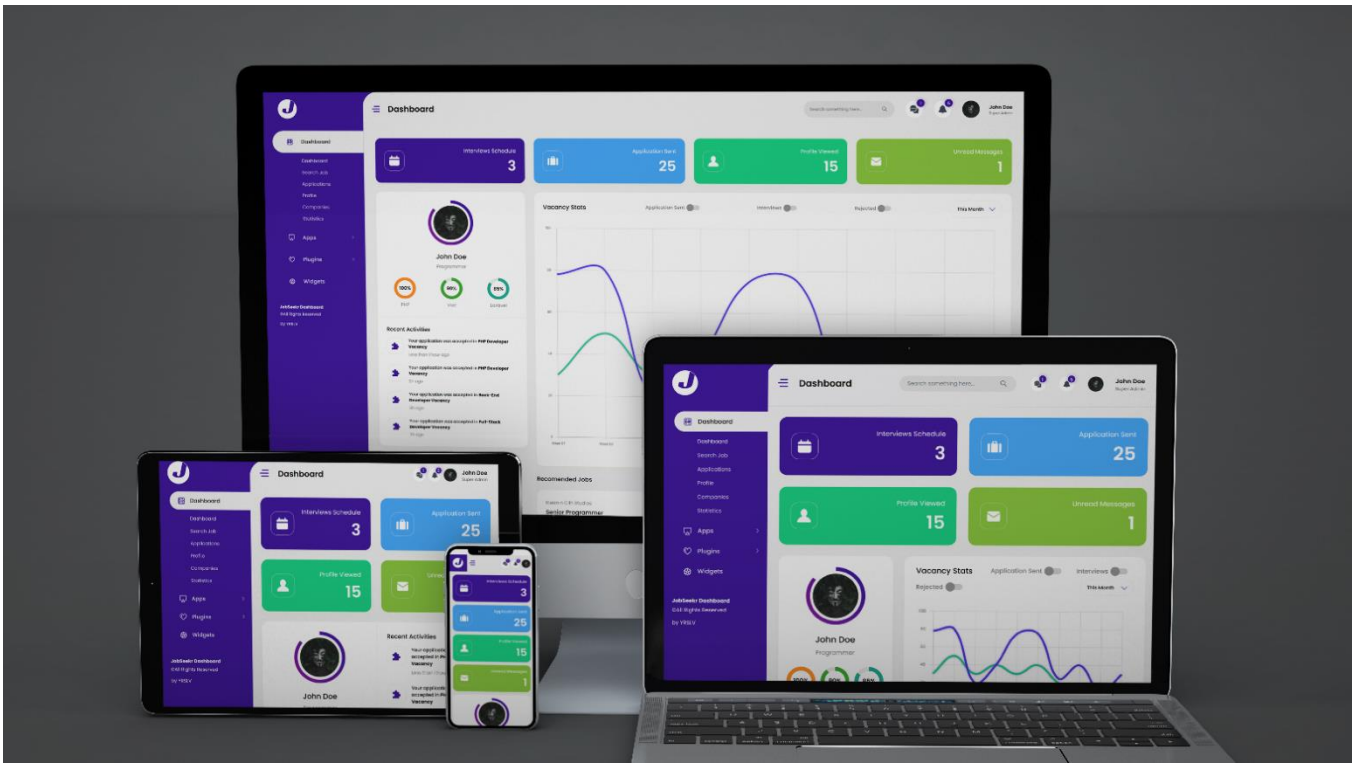

Рис. 3.1. Робота сервісу на різних пристроях

На рисунках 3.2 та 3.3 показано адаптацію сторінки Dashboard під екран смартфону.

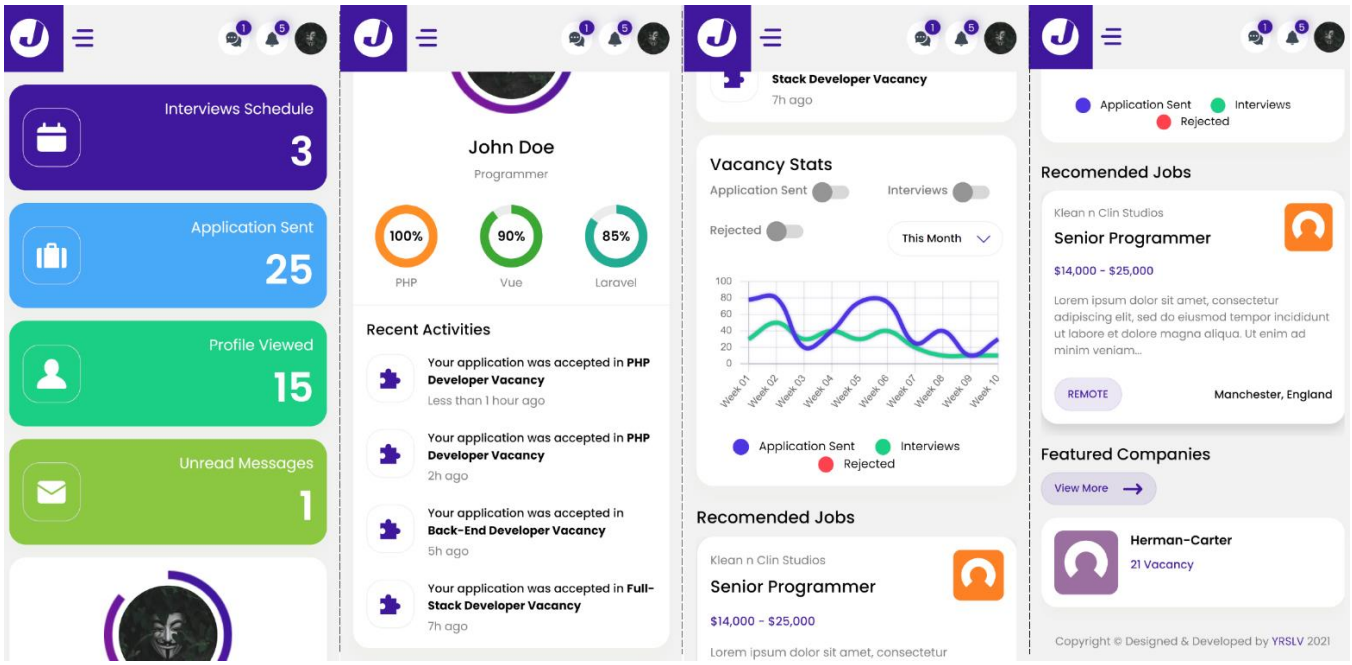

Рис. 3.2. Адаптація чотирьох частин сторінки Dashboard під екран смартфону (сірим пунктиром розділено скріншоти, які зображують різні частини екрану)

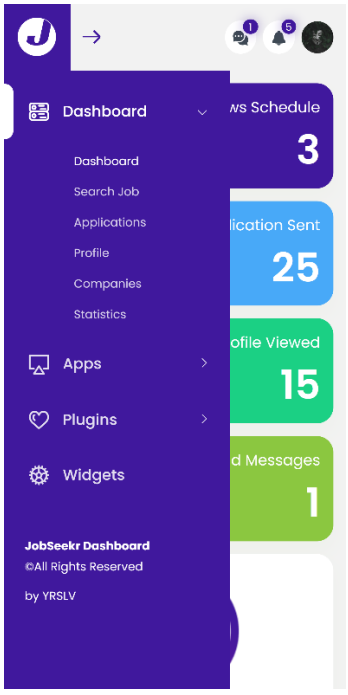

Рис. 3.3. Адаптація меню сторінки Dashboard під екран смартфону

На рисунках 3.4 та 3.5 зображено адаптацію сторінки Statistics під екран смартфону.

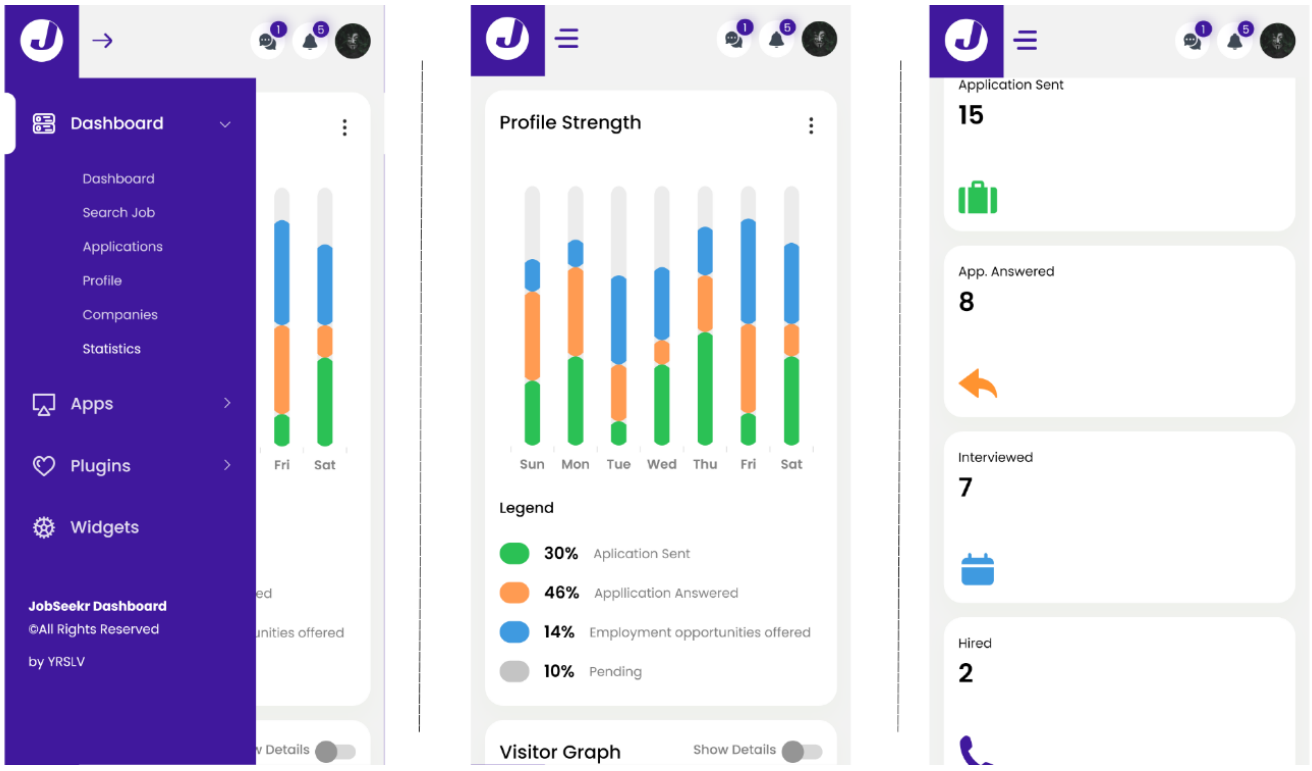

Рис. 3.4. Адаптація перших трьох частин сторінки Statistics під екран смартфону (сірим пунктиром розділено скріншоти, які зображують різні частини екрану)

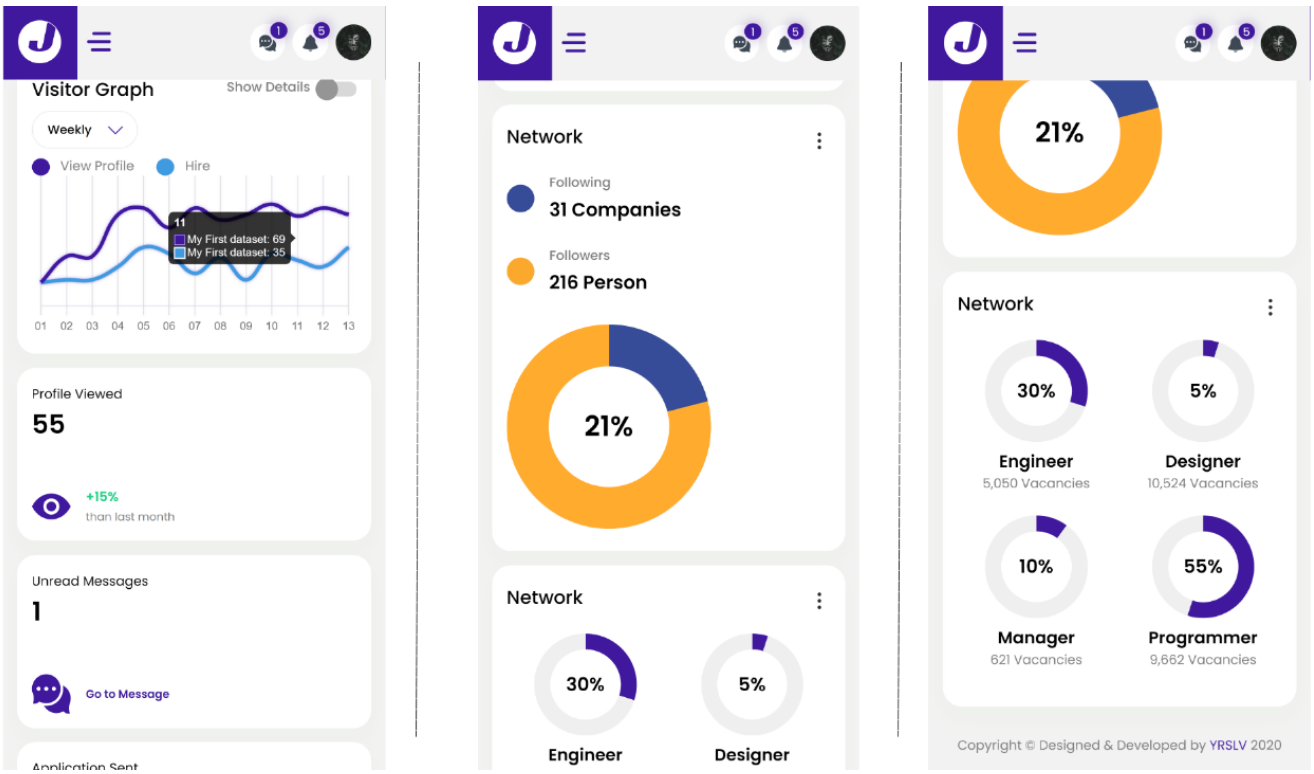

Рис. 3.5. Адаптація наступних трьох частин сторінки Statistics під екран смартфону (сірим пунктиром розділено скріншоти, які зображують різні частини екрану)

На рисунку 3.6 показано адаптацію сторінки Dashboard під екран планшету.

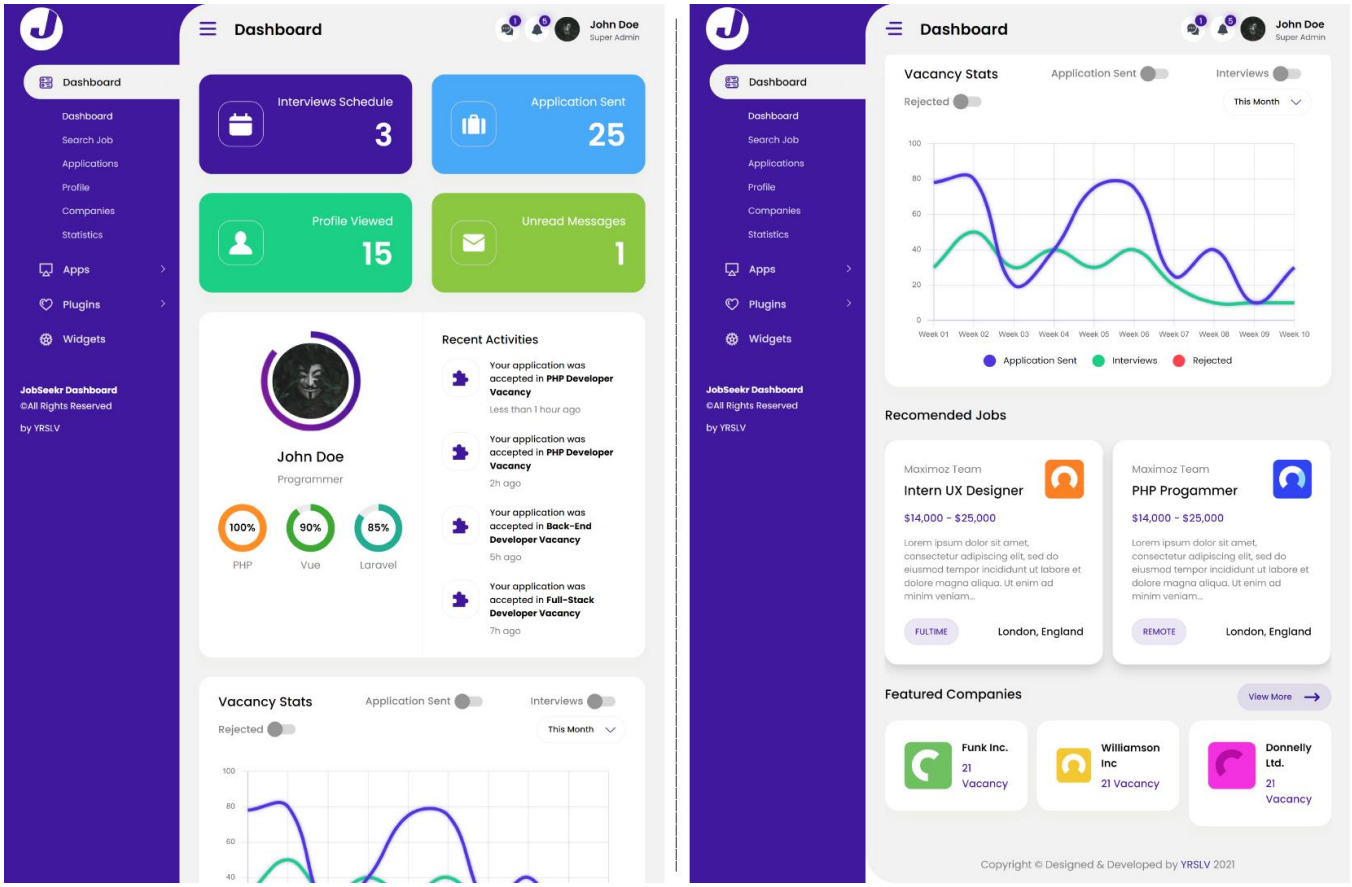

Рис. 3.6. Адаптація сторінки Dashboard під екран планшету (сірим пунктиром розділено скріншоти, які зображують різні частини екрану)

На рисунку 3.7 показано адаптацію сторінки Statistics під екран планшету.

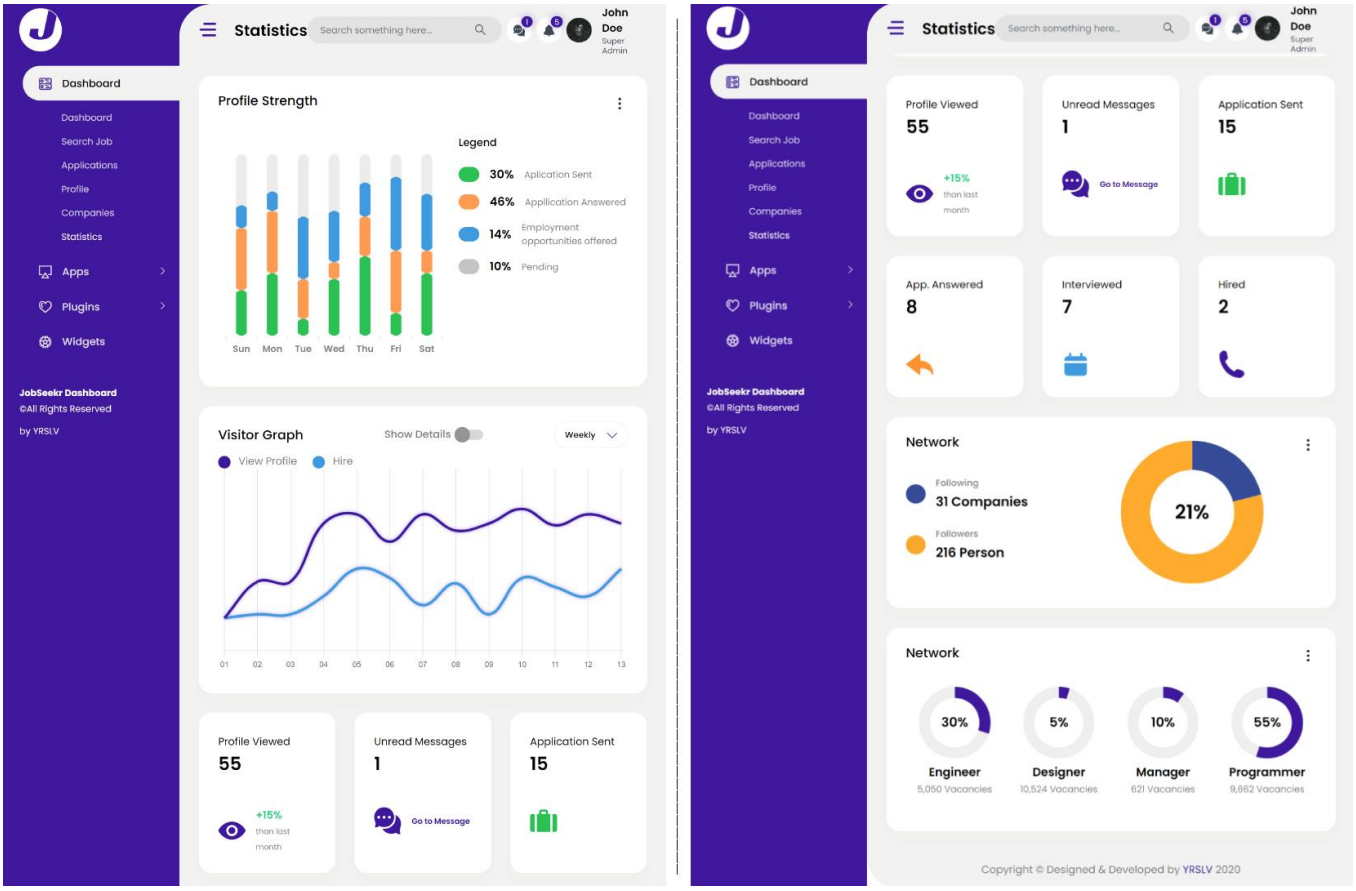

Рис. 3.7. Адаптація сторінки Statistics під екран планшету (сірим пунктиром розділено скріншоти, які зображують різні частини екрану)

Окрім вищезазначеного, користувацький інтерфейс також слідує принципам структурної організації, простоти та мінімалізму, видимості необхідного функціоналу, зворотнього зв'язку, толерування помилок та повторного використання.

3.7. Аналіз користувацького досвіду

Програмна система забезпечує хороший користувацький досвід, оскільки відповідає вимогам корисності, ефективності та зручності використання. Зміст ефективності включає просте маніпулювання (Закон Фіттса), прості кроки (кількість кліків для виконання завдання), чітку навігацію (користувачі завжди знають, де вони знаходяться і не губляться в сторінках сервісу).

Зручність використання підтверджується евристичними оцінками та оцінками продуктивності. Зокрема сервіс відповідає таким евристикам Якоба Нільсена:

- Видимість стану системи;
- Відповідність між системою та реальним світом;
- Контроль та свобода користувача;
- Послідовність та стандарти;
- Попередження помилок;
- Впізнавання, а не пригадування;
- Гнучкість та ефективність використання;
- Естетичність та мінімалістичний дизайн;
- Допомога користувачу у розпізнанні, діагностуванні та виправленні помилок [38].

3.8. Порівняння з аналогічними сервісами

Для порівняння було обрано 2 сервіси – “work.ua”[39] та “rabota.ua”[40]. На рисунку 3.8 представлено скріншот сторінки Dashboard створеного сервісу в поданні для настільних ПК, а на рисунках 3.9 та 3.10 аналогічні сторінки сервісів work.ua та rabota.ua відповідно.

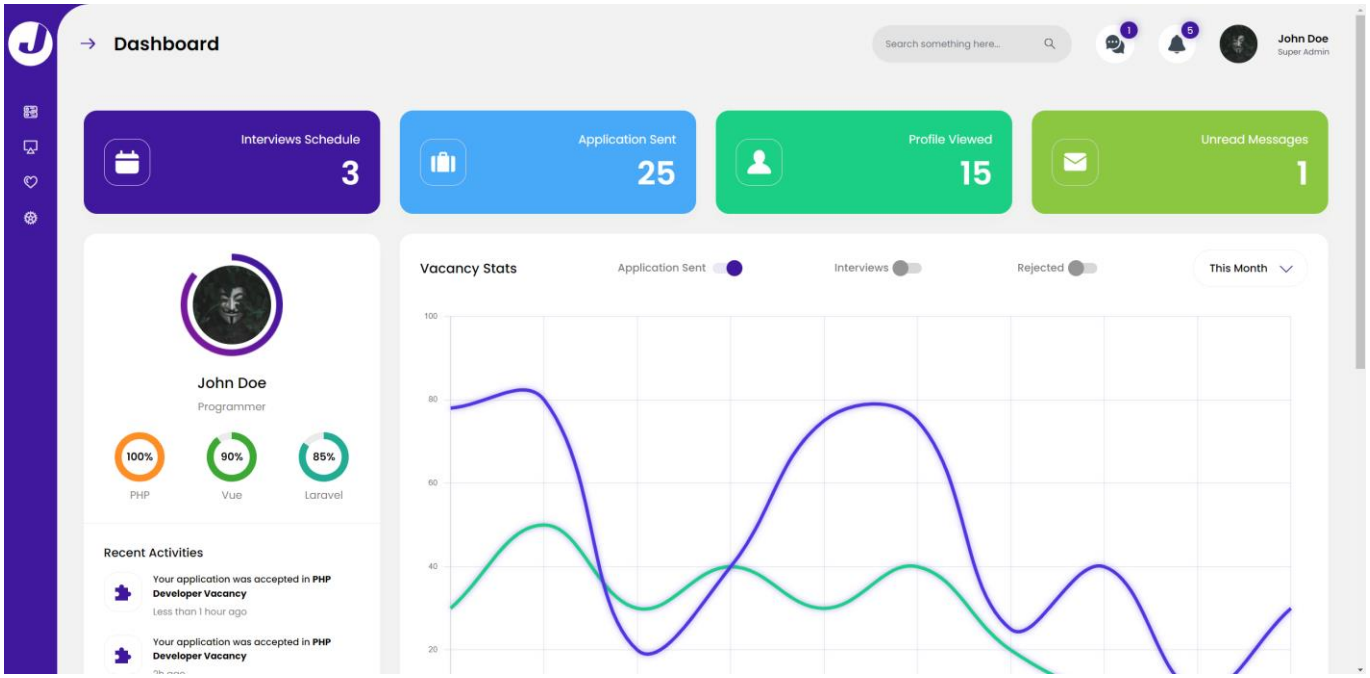

Рис. 3.8. Скріншот сторінки Dashboard створеної програмної системи візуалізації статистики web-сервісу пошуку роботи для користувачів

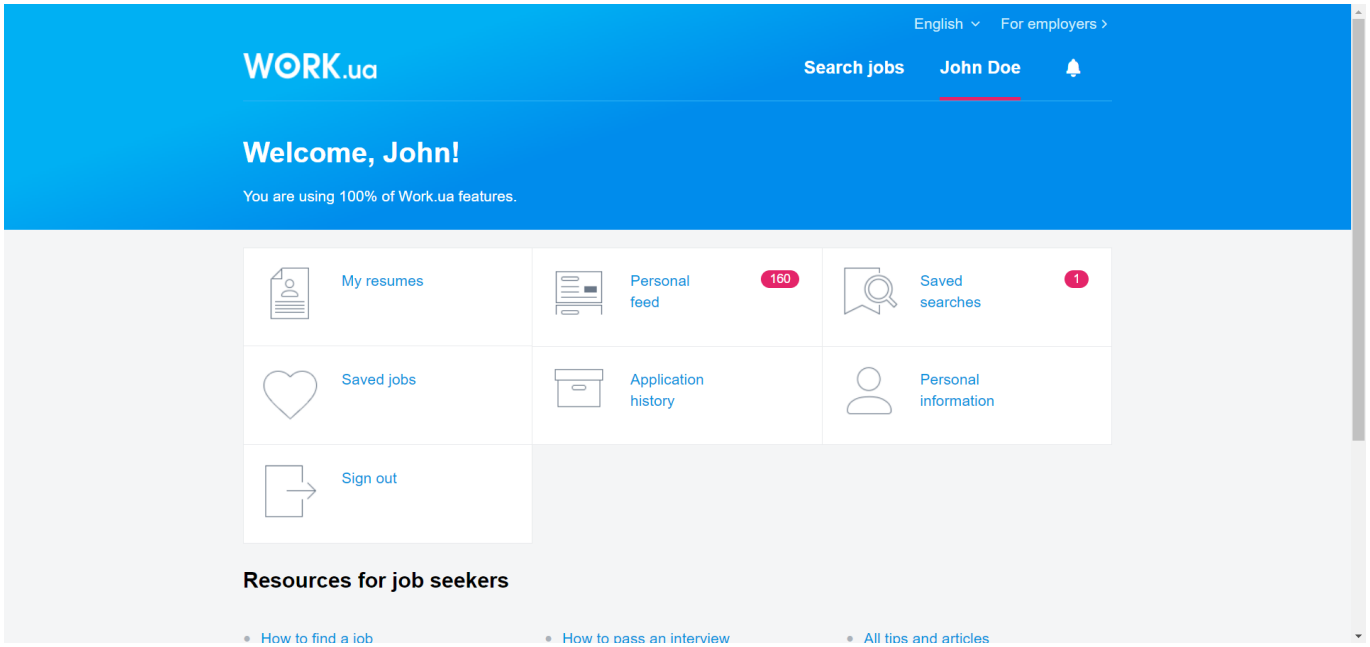

Рис. 3.9. Скріншот сторінки з інформаційною панеллю сервісу work.ua

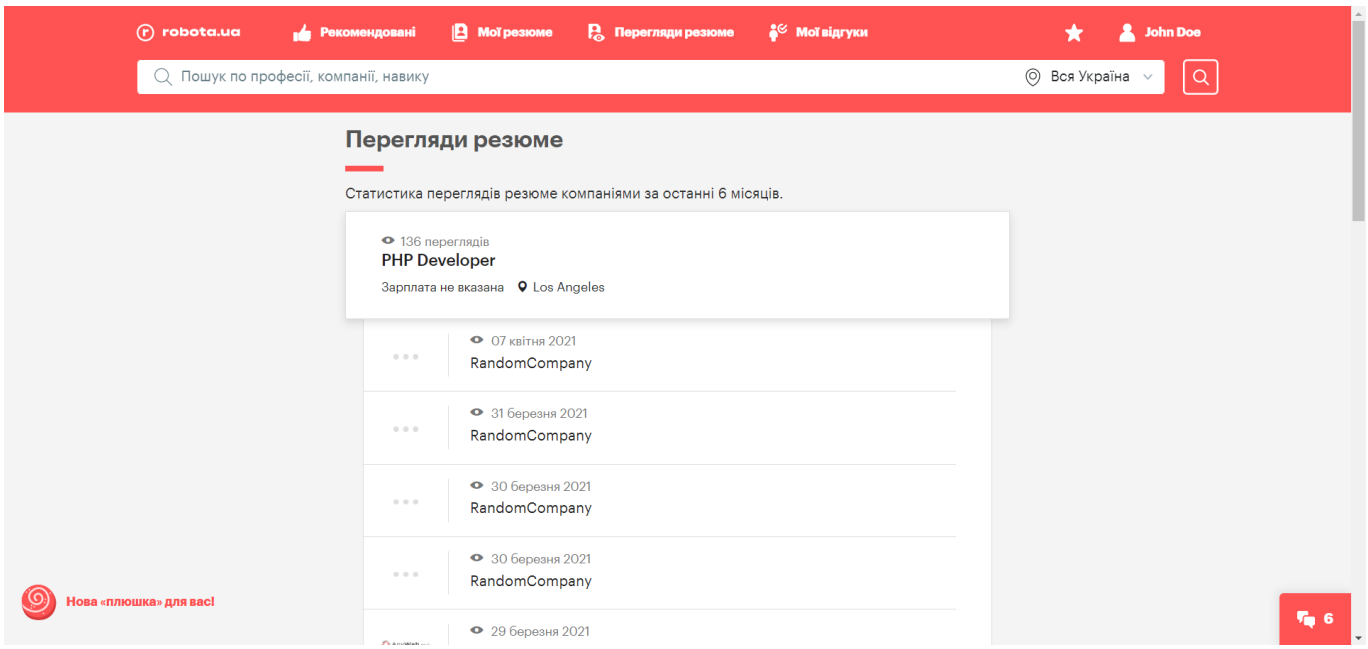

Рис. 3.10. Скріншот сторінки з інформаційною панеллю сервісу rabota.ua

На рисунку 3.11 представлено порівняння скріншотів тих самих сторінок усіх трьох сервісів в поданні для телефонів.

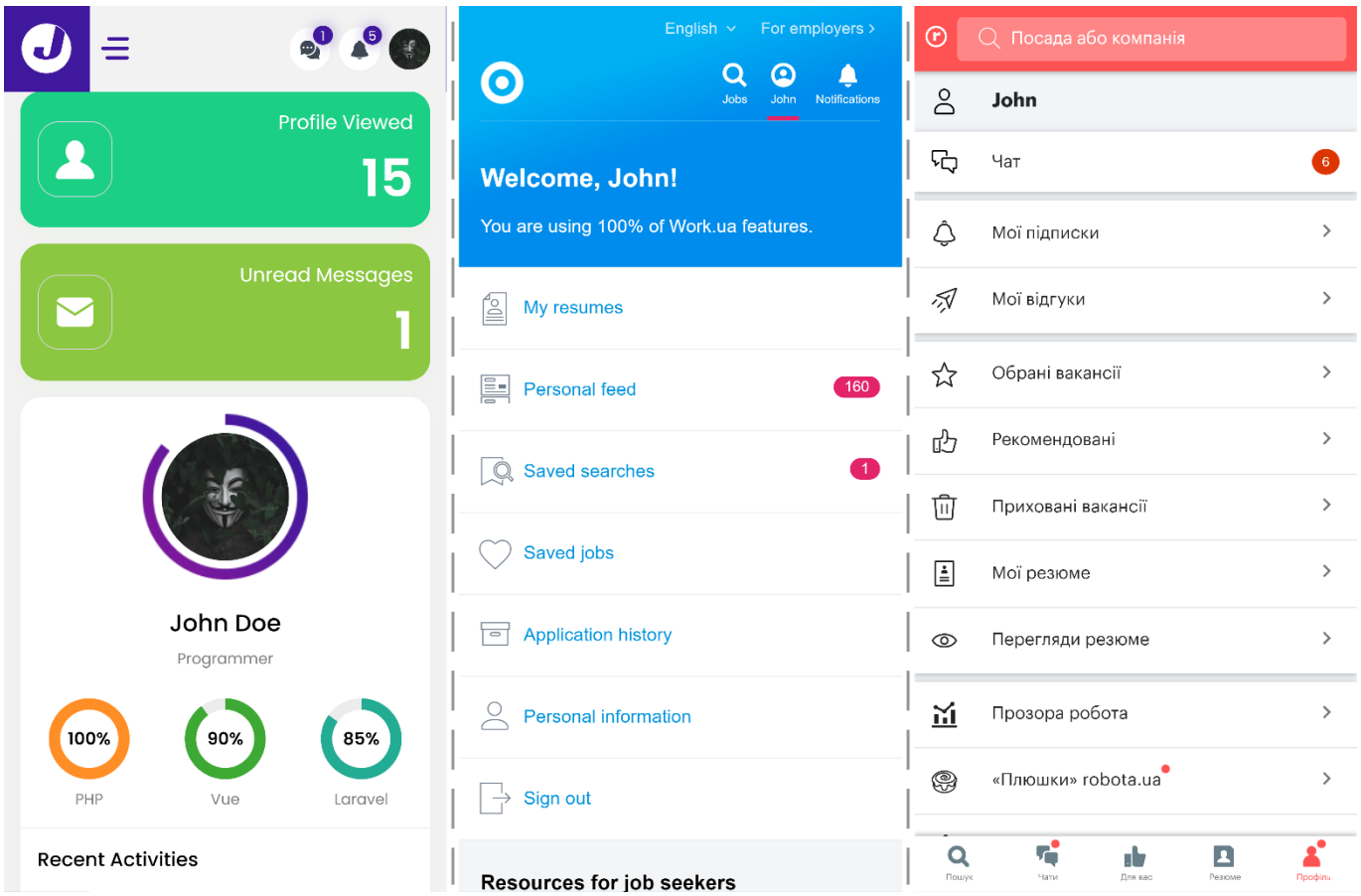

Рис. 3.11. Порівняння скріншотів сторінок усіх трьох сервісів в поданні для телефонів (сірим пунктиром розділено скріншоти, які відповідають різним сервісам)

Функціонал. З погляду загального набору функцій сервісу пошуку роботи, створений сервіс як частина більшого сервісу пошуку роботи, “work.ua” та “rabota.ua” мають схожий функціонал. Кожен з сервісів пропонує пошук вакансій та перегляд. Доступна реєстрація користувачів. Користувачі можуть надсилати заявки (відгукуватися) на вакансії, спілкуватися в чаті з HR-спеціалістами, підписуватися на оновлення компаній тощо. Що істотно відрізняється – це функціонал відображення та візуалізації статистики, власне те, для чого і створювалася програмна система візуалізації статистики сервісу пошуку роботи для користувачів, те, чому і присвячена ця робота.

Work.ua відображає дані по кількості показів резюме, переглядів резюме, кількості рекомендованих вакансій та кількості вакансій у збережених пошуках. Всі дані відображаються або просто чисельними значеннями або простим графічним

елементом типу «бейдж» – заокругленим прямокутником, заповненим кольором та числовим значенням всередині.

Robota.ua представляє лише дані по кількості переглядів резюме роботодавцями та інформацію по відгукам на вакансії у вигляді числових значень на відповідних елементах сторінок та у вигляді бейджів з відповідними числами на вкладках користувацького інтерфейсу. Також відображається кількість непрочитаних повідомлень в чаті в нижньому правому кутку екрану.

Створена програмна система візуалізації статистики сервісу пошуку роботи для користувачів відображає такі дані: блок візуалізованої статистики по вакансіям з інтерактивними графіками, що можуть відображати кількість надісланих заявок, кількість запланованих співбесід та кількість відмов за різні періоди часу на сторінці Dashboard. На сторінці Statistics: комбінований блоком візуалізованої статистики «потужності профілю» – стовпчаста діаграма з накопиченням, що може відображати кількості надісланих заявок, заявок на які було отримано відповідь, заявок по яким в результаті було отримано job offer та заявок що очікують на розгляд, блок з візуалізованою статистикою відвідування – точкова діаграма з плавними лініями, що може відображати кількість переглядів профілю та кількість отриманих пропозицій розглянути вакансію за різні періоди часу, блок візуалізованих даних з картками, що відображають кількість переглядів профілю з порівнянням даних за минулий місяць, кількість непрочитаних повідомлень від HR-спеціалістів та роботодавців, кількість надісланих заявок, кількість заявок на які було отримано відповідь, а для профілів користувачів з роллю «HR-спеціаліст» додатково відображаються картки з кількістю проведених співбесід по отриманим заявкам та кількістю найнятих працівників. Нижче відображаються блоки складової професійної соціальної мережі для пошуку і встановлення ділових контактів. Перший блок містить кількість відстежуваних компаній та людей, що відстежують профіль з відповідною візуалізацією у вигляді кільцевої діаграми. Другий блок відображає розподіл мережі ділових контактів за професіями з відповідною візуалізацією у вигляді кільцевих діаграм та кількість доступних вакансій для кожної професії. При цьому конфігурація блоків інфографіки та візуалізованої статистики можна модифікувати, розширювати та

доповнювати. Всі ці фактори дають значну конкурентну перевагу над двома іншими сервісами.

Швидкодія. Для порівняння часу повного завантаження була обрана головна сторінка профілю користувача з інформаційною панеллю (Dashboard). За значення часу повного завантаження сторінки було взято середнє арифметичне трьох вимірювань, які проводились починаючи з 2 завантаження сторінки. Усі сервіси відкривались у веб-браузері Google Chrome версії 96.0.4664.45 (64-bit). При використанні усіх трьох сервісів в місті Київ час повного завантаження сторінки Dashboard створеного сервісу складає близько 0.8 секунд, work.ua – 1.1 секунди, robota.ua – 1.3 секунди. На рисунку 3.12 наведено час повного завантаження сторінки Dashboard при використанні усіх трьох сервісів в різних точках планети (географічних локаціях), де SF позначає Сан Франциско, а JHB – Йоганнесбург.

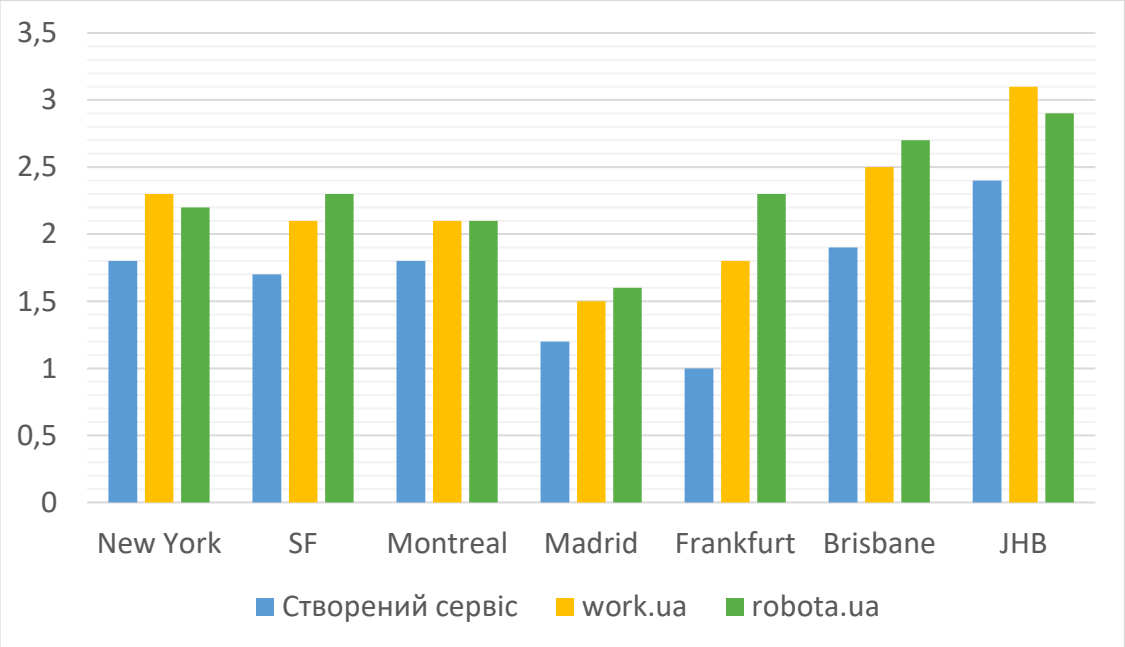

Рис. 3.12. Час повного завантаження головної сторінки профілю користувача з інформаційною панеллю (Dashboard) у секундах

Створений сервіс завантажується найшвидше серед усіх в протестованих географічних локаціях.

Користувацький інтерфейс. Кожен з сервісів має власний унікальний користувацький інтерфейс, який забезпечує різний користувацький досвід. Адаптивність у всіх 3 сервісів також різна. Створений сервіс є повністю адаптивним, і забезпечує комфортне використання на будь-якому форм-факторі пристрою користувача. “WORK.ua” та “robota.ua” також є адаптивними, але на окремих сторінках зустрічаються надто дрібні елементи та деякі анімації є не дуже плавними або взагалі відсутні. Також у сервісу “robota.ua” за час тестування були помічені проблеми з перекладом вмісту та навігаційних меню, на одних сторінках текст міг відображатися однією мовою, а на інших – іншою, при тому що опція зміни мови інтерфейсу не використовувалася в цей час.

Користувацький досвід. Створений сервіс забезпечує хороший користувацький досвід шляхом дотримання відповідності вимогам корисності, ефективності та зручності використання. Для кожного з порівнюваних сервісів користувацький досвід буде різним. Хоча сервісом “ WORK.ua ” в цілому зручно користуватися на мобільних пристроях та пристроях з великими екранами високої роздільної здатності, окремі моменти, такі як дрібні елементи на деяких сторінках та не дуже зручна навігація псують загальне враження. Інтерфейс сервісу “robota.ua” містить баги, деякі його частини є заплутаними, що також негативно впливає на користувацький досвід. Таким чином створений сервіс забезпечує найкращий користувацький досвід з усіх трьох сервісів.

Підсумовуючи вищезазначені деталі, можна зробити висновок, що створений сервіс має ряд істотних переваг над існуючими аналогічними рішеннями.

Висновок до розділу 3

У третьому розділі було проаналізовано усі складові безпеки програмного забезпечення, ефективність архітектурних рішень та патернів проектування. Також здійснено аналіз ефективності та масштабованості програмної системи візуалізації статистики сервісу пошуку роботи для користувачів в цілому і окремих його компонентів. Детально досліджено та описано можливості масштабування php-фреймворку Laravel та сервісів AWS – S3, DynamoDB та CloudFront у контексті потреб програмної системи візуалізації статистики сервісу пошуку роботи для користувачів.

Розглянуто користувацький інтерфейс програмної системи візуалізації статистики сервісу пошуку роботи для користувачів та принципи, які забезпечують його ефективність. Описано користувацький досвід та його відповідність евристикам Якоба Нільсена. Проведено детальне порівняння створеного сервісу з 2 аналогічними продуктами.

ВИСНОВКИ

Дана робота присвячена проектуванню архітектури та подальшої реалізації програмної системи візуалізації статистики web-сервісу пошуку роботи для користувачів.

Після огляду теоретичного матеріалу щодо понять візуалізації статистики та варіантів її реалізації в програмному забезпеченні, було розпочато роботу над визначенням архітектури проекту.

Проектування архітектури та розробка проекту здійснювались з урахуванням того факту, що програмна система візуалізації статистики сервісу пошуку роботи для користувачів може працювати в умовах великих навантажень.

Для реалізації back end частини програмного забезпечення було обрано мову програмування php, парадигму об’єктно-орієнтованого програмування, фреймворк Laravel та комбінацію декількох архітектурних стилів, серед яких клієнт-серверна архітектура, специфічна мікро-архітектура Laravel та хмарна архітектура.

Базу даних було реалізовано з використанням сервісу Amazon DynamoDB, сховище – Amazon S3, мережу доставки контенту – Amazon CloudFront. Вибір кожного компонента системи продиктований рядом вимог, серед яких вимоги швидкодії, надійності, безпеки, гнучкості, масштабування та ін.

Обрані рішення спростили розробку, деплоймент і супровід програмної системи. Таке спрощення дозволило зробити систему максимально гнучкою, а відповідно якомога довше мати якнайбільше варіантів для різних потенційних змін. Було досягнуто маскимально простої процедури деплойменту. Після початкового налаштування інфраструктури всі необхідні дії можна організувати в один скрипт, що дає можливість виконувати деплоймент одним натисненням.

Висока ефективність отриманої системи, відмовостійкість та можливості масштабування є результатами ретельного проектування та розробки.

Одним з основних патернів проектування, що застосовувалися для реалізації сервісу є MVC. Також використовувалися ActiveRecord та Facade.

Інтерфейс програмної системи проектувався та розроблявся з метою забезпечення якнайкращого користувацького досвіду та зручності використання. Дотримання простих принципів дозволило зробити інтерфейс потужним, мінімалістичним та зручним.

Для створення front end частини сервісу використовувались HTML5, CSS3, фронтенд-фреймворк Bootstrap версії 4.5.0, JavaScript, jQuery та Blade, бібліотеки Chart.js, ApexCharts.js, а також jQuery плагін Peity.js. Обрані рішення дозволили створити максимально органічний та інтерактивний користувацький досвід. Реалізація візуалізації статистичних даних вдало поєднує широкі можливості використаних технологій та доволі лаконічний дизайн.

Окрім вищезазначеного, варто окремо підкреслити той факт, що фактично було отримано serverless-ready систему, усі компоненти якої, окрім back end на Laravel в умовах розробки вже працюють за serverless схемою, при цьому Laravel-додаток можна легко розгорнути на Amazon EC2 інстансах.

Подальшим напрямком роботи може бути інтеграція з системою обробки великих даних, що генеруються користувачами сервісу та використання створеної програмної системи візуалізації статистики для відображення певних результатів, закономірностей тощо. Також завдяки гнучкості отриманої системи, вона може використовуватися з іншими сервісами як незалежний модуль візуалізації наборів даних.

Отримана програмна система показала відмінні результати, не тільки не поступаючись за характеристиками комерційним аналогам, а навіть переважаючи їх в ряді випадків, що доводить правильність та ефективність вибраного архітектурного рішення.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Books Ngram Viewer: infographic,data visualization,information graphics [Електронний ресурс] – Режим доступу: https://books.google.com/ngrams/graph?content=Infographic%2Cdata+visualization%2Cinformation+graphics&year_start=1990&year_end=2019&corpus=26&smoothing=0&case_insensitive=true (дата звернення 25.10.2021р) – Назва з екрана.
2. Google Trends: infographic [Електронний ресурс] – Режим доступу: https://trends.google.com/trends/explore?date=all&q=infographic (дата звернення 25.10.2021р) – Назва з екрана.
3. J. Zacks, E. Levy, B. Tversky, D. Schiano: Graphs in Print / M. Anderson, B. Meyer, P. Olivier (eds.) // Diagrammatic Representation and Reasoning – London: Springer-Verlag, 2002. – pp. 187-206.
4. Human Anatomy & Physiology 7th Edition / E. N. Marieb, K. Hoehn – San Francisco: Benjamin Cummings, 2006 – 1296 p.
5. The SAGE Handbook of Political Communication / H. A. Semetko, M. Scammell – Thousand Oaks: SAGE Publications, 2012 – 544 p.
6. Thorpe S., Fize D., Marlot C. Speed of processing in the human visual system / S. Thorpe, D. Fize, C. Marlot // Nature – 1996 – Vol. 381, Issue 6582 – pp. 520-522.
7. Holcomb P., Grainger J. On the Time Course of Visual Word Recognition / P. Holcomb, J. Grainger // Journal of Cognitive Neuroscience – 2006 – Vol. 18, Issue 10 – pp. 1631-1643.
8. R. Alleyne. Welcome to the information age – 174 newspapers a day. The Telegraph [Електронний ресурс] – Режим доступу: http://www.telegraph.co.uk/science/science-news/8316534/Welcome-to-the-information-age-174-newspapers-a-day.html (дата звернення 25.10.2021р) – Назва з екрана.
9. Bohn R., Short J. Measuring Consumer Information / R. Bohn, J. Short // International Journal of Communication – 2012 – Vol. 6. – pp. 980-1000.

10. J. Nielsen. How Little Do Users Read? Nielsen Norman Group3 [Електронний ресурс] – Режим доступу: https://www.nngroup.com/articles/how-little-do-users-read/ (дата звернення 25.10.2021р) – Назва з екрана.
11. Xerox. 20 Ways to Share Color Knowledge [Електронний ресурс] – Режим доступу: https://www.office.xerox.com/latest/COLFS-02UA.PDF (дата звернення 25.10.2021р) – Назва з екрана.
12. Dowse R., Ehlers M. Medicine labels incorporating pictograms: Do they influence understanding and adherence? / R. Dowse, M. Ehlers // Patient Education and Counseling – 2005 – Vol. 58, Issue 1. – pp. 63-70.
13. Levie W. H., Lentz R. Effects of text illustrations: A review of research / W. H. Levie, R. Lentz // Educational Communication and Technology – 1982 – Vol. 30 – pp. 195-232.
14. McCabe D. P., Castel A. D. Seeing is believing: The effect of brain images on judgments of scientific reasoning / D. P. McCabe, A. D. Castel // Cognition – 2008 – Vol. 107, Issue 1 – pp. 343-352.
15. P. M. Lester. Syntactic Theory of Visual Communication. [Електронний ресурс] – Режим доступу: http://paulmartinlester.info/writings/viscomtheory.html (дата звернення 25.10.2021р) – Назва з екрана.
16. Резаєв Я. О. Web-сервіс потокової трансляції відео : дипломний проект бакалавра : захищений 16.06.2020 / Резаєв Ярослав Олегович ; Національний Авіаційний Університет. – Київ, 2020. – 118 с.
17. Чиста Архітектура: Мистецтво розроблення програмного забезпечення / пер. з англ. І. Бондар-Терещенко. – Харків : Вид-во «Ранок» : Фабула, 2019. – 368 с. – Бібліогр.: с. 156-158
18. CS 410/510 - Software Engineering Architectural Design [Електронний ресурс] – Режим доступу: https://cs.ccsu.edu/~stan/classes/CS410/Notes16/06-ArchitecturalDesign.html (дата звернення 07.11.2021р) – Назва з екрана.
19. Por qué Laravel NO es un framework MVC y tú deberías olvidarte de MVC [Електронний ресурс] – Режим доступу: https://styde.net/porque-laravel-no-es-

mvc-y-tu-deberias-olvidarte-de-mvc/ (дата звернення 07.11.2021р) – Назва з екрана.

20. NoSQL Databases Explained [Електронний ресурс] – Режим доступу: https://www.mongodb.com/nosql-explained (дата звернення 07.11.2021р) – Назва з екрана.
21. Amazon DynamoDB [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/dynamodb/ (дата звернення 07.11.2021р) – Назва з екрана.
22. Amazon DynamoDB Developer Guide. Best Practices for Modeling Relational Data in DynamoDB [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/amazondynamodb/ (дата звернення 20.11.2021р) – Назва з екрана.
23. Amazon S3 [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/s3/ (дата звернення 07.11.2021р) – Назва з екрана.
24. Amazon CloudFront [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/cloudfront/ (дата звернення 07.11.2021р р) – Назва з екрана.
25. AWS News Blog. New CloudFront Feature: Invalidation [Електронний ресурс] // AWS News Blog – Режим доступу: https://aws.amazon.com/ (дата звернення 07.11.2021р) – Назва з екрана.
26. AWS Elastic Load Balancing [Електронний ресурс] – Режим доступу: https://aws.amazon.com/elasticloadbalancing/ (дата звернення 20.11.2021р) – Назва з екрана.
27. AWS Documentation. Sticky sessions for your Application Load Balancer [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/elasticloadbalancing/ (дата звернення 20.11.2021р) – Назва з екрана.
28. AWS News Blog. New Elastic Load Balancing Feature: Sticky Sessions [Електронний ресурс] // AWS News Blog – Режим доступу: https://aws.amazon.com/ (дата звернення 20.11.2021р) – Назва з екрана.

29. AWS Documentation. Configure sticky sessions for your Classic Load Balancer [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/elasticloadbalancing/ (дата звернення 20.11.2021р) – Назва з екрана.
30. Laravel Blade Templates [Електронний ресурс] – Режим доступу: https://laravel.com/docs/7.x/blade (дата звернення 07.11.2021р) – Назва з екрана.
31. Chart.js. Simple yet flexible JavaScript charting for designers & developers [Електронний ресурс] – Режим доступу: https://www.chartjs.org/ (дата звернення 07.11.2021р) – Назва з екрана.
32. Apexcharts.js. Modern & Interactive Open-source Charts [Електронний ресурс] – Режим доступу: https://apexcharts.com/ (дата звернення 07.11.2021р) – Назва з екрана.
33. Peity [Електронний ресурс] – Режим доступу: https://benpickles.github.io/peity/ (дата звернення 07.11.2021р) – Назва з екрана.
34. Laravel Routing [Електронний ресурс] – Режим доступу: https://laravel.com/docs/7.x/routing (дата звернення 07.11.2021р) – Назва з екрана.
35. Laravel Views [Електронний ресурс] – Режим доступу: https://laravel.com/docs/7.x/views (дата звернення 07.11.2021р) – Назва з екрана.
36. Возможности Amazon DynamoDB [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/dynamodb/features/?nc1=h_ls (дата звернення 20.11.2021р) – Назва з екрана.
37. Best Practices Design Patterns: Optimizing Amazon S3 Performance [Електронний ресурс] // AWS Documentation – Режим доступу: https://docs.aws.amazon.com/AmazonS3/latest/dev/optimizing-performance.html (дата звернення 20.11.2021р) – Назва з екрана.
38. 10 Usability Heuristics for User Interface Design [Електронний ресурс] – Режим доступу: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення 20.11.2021р) – Назва з екрана.

39. WORK.ua [Електронний ресурс] – Режим доступу: https://www.work.ua/ (дата звернення 20.11.2021р) – Назва з екрана.
40. robota.ua [Електронний ресурс] – Режим доступу: https://rabota.ua/ (дата звернення 20.11.2021р) – Назва з екрана.

ДОДАТКИ

Додаток А

А1. Сирцевий код моделі User.php

```
<?php

namespace App;

use BaoPham\DynamoDb\DynamoDbModel;
use Illuminate\Auth\Authenticatable;
use Illuminate\Contracts\Auth\Authenticatable as IAuthenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;

class User extends DynamoDbModel implements IAuthenticatable
{
    use Authenticatable, HasApiTokens, Notifiable;

    /**
     * The table associated with the model.
     *
     * @var string
     */
    protected $table = 'Users';

    /**
     * The primary key associated with the table.
     *
     * @var string
     */
    protected $primaryKey = 'id';

    /**
```

```
     * Indicates if the IDs are auto-incrementing     *
     * @var bool
     */
    public $incrementing = false;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'username', 'email', 'password','user-info', ' skills', 'favorite-
companies', 'preferred-positions'
    ];
```

А2. Фрагмент сирцевого коду контролеру DashboardController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller
{
    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');
    }
```

```
    /**
     * Show the application dashboard.
     *
     * @return \Illuminate\Contracts\Support\Renderable
     */
    public function index()
    {
        return view('home');
    }
}
```

А3. Сирцевий код контролеру UserController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\Facades\Hash;

use App\User;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;
use DataTables,Auth;

class UserController extends Controller
{
    /**
     * Create a new controller instance.
     *
     * @return void
     */
```

```
public function __construct()
{
    $this->middleware('auth');
}

/**
 * Show the application dashboard.
 *
 * @return \Illuminate\Contracts\Support\Renderable
 */
public function index()
{
    return view('users');
}

public function getUserList(Request $request)
{

    $data  = User::get();

    return Datatables::of($data)
            ->addColumn('roles', function($data){
                $roles = $data->getRoleNames()->toArray();
                $badge = '';
                if($roles){
                    $badge = implode(' , ', $roles);
                }

                return $badge;
            })
            ->addColumn('permissions', function($data){
                $roles = $data->getAllPermissions();
                $badges = '';
                foreach ($roles as $key => $role) {
```

```
                    $badges .= '<span class="badge badge-dark m-
1">'.$role->name.'</span>';
                }

                return $badges;
            })
            ->addColumn('action', function($data){
                if($data->name == 'Super Admin'){
                    return '';
                }
                if (Auth::user()->can('manage_user')){
                    return '<div class="table-actions">
                            <a href="'.url('user/'.$data->id).'" ><i
class="ik ik-edit-2 f-16 mr-15 text-green"></i></a>
                            <a href="'.url('user/delete/'.$data-
>id).'"><i class="ik ik-trash-2 f-16 text-red"></i></a>
                        </div>';
                }else{
                    return '';
                }
            })
            ->rawColumns(['roles','permissions','action'])
            ->make(true);
    }

    public function create()
    {
        try
        {
            $roles = Role::pluck('name','id');
            return view('create-user', compact('roles'));

        }catch (\Exception $e) {
            $bug = $e->getMessage();
```

```
            return redirect()->back()->with('error', $bug);

        }
    }

    public function store(Request $request)
    {
        // create user
        $validator = Validator::make($request->all(), [
            'name'     => 'required | string ',
            'email'    => 'required | email | unique:users',
            'password' => 'required | confirmed',
            'role'     => 'required'
        ]);

        if($validator->fails()) {
            return redirect()->back()->withInput()->with('error', $validator-
>messages()->first());
        }
        try
        {
            // store user information
            $user = User::create([
                        'name'     => $request->name,
                        'email'    => $request->email,
                        'password' => Hash::make($request->password),
                    ]);

            // assign new role to the user
            $user->syncRoles($request->role);

            if($user){
                return    redirect('users')->with('success',    'New    user
created!');
```

```
            }else{
                return redirect('users')->with('error', 'Failed to create new
user! Try again.');
            }
        }catch (\Exception $e) {
            $bug = $e->getMessage();
            return redirect()->back()->with('error', $bug);
        }
    }

    public function edit($id)
    {
        try
        {
            $user  = User::with('roles','permissions')->find($id);

            if($user){
                $user_role = $user->roles->first();
                $roles     = Role::pluck('name','id');

                return                                          view('user-edit',
compact('user','user_role','roles'));
            }else{
                return redirect('404');
            }

        }catch (\Exception $e) {
            $bug = $e->getMessage();
            return redirect()->back()->with('error', $bug);
        }
    }

    public function update(Request $request)
    {
```

```
        // update user info
        $validator = Validator::make($request->all(), [
            'id'       => 'required',
            'name'     => 'required | string ',
            'email'    => 'required | email',
            'role'     => 'required'
        ]);

        // check validation for password match
        if(isset($request->password)){
            $validator = Validator::make($request->all(), [
                'password' => 'required | confirmed'
            ]);
        }

        if ($validator->fails()) {
            return redirect()->back()->withInput()->with('error', $validator-
>messages()->first());
        }

        try{

            $user = User::find($request->id);

            $update = $user->update([
                'name' => $request->name,
                'email' => $request->email,
            ]);

            // update password if user input a new password
            if(isset($request->password)){
                $update = $user->update([
                    'password' => Hash::make($request->password)
```

```
                ]);
            }

            // sync user role
            $user->syncRoles($request->role);

            return redirect()->back()->with('success', 'User information
updated succesfully!');
        }catch (\Exception $e) {
            $bug = $e->getMessage();
            return redirect()->back()->with('error', $bug);

        }
    }


    public function delete($id)
    {
        $user   = User::find($id);
        if($user){
            $user->delete();
            return redirect('users')->with('success', 'User removed!');
        }else{
            return redirect('users')->with('error', 'User not found');
        }
    }
}
```

А4. Сирцевий коду контролеру RolesController.php

```
<?php

namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;
use Illuminate\Support\Facades\Validator;
use DataTables,Auth;

class RolesController extends Controller
{
    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');
    }

    /**
     * Show the roles page
     *
     */
    public function index()
    {
        try{
            $permissions = Permission::pluck('name','id');

            return view('roles', compact('permissions'));
        }catch (\Exception $e) {
            $bug = $e->getMessage();
            return redirect()->back()->with('error', $bug);

        }
    }
```

```
    /**
     * Show the role list with associate permissions.
     * Server side list view using yajra datatables
     */

    public function getRoleList(Request $request)
    {

        $data  = Role::get();

        return Datatables::of($data)
                ->addColumn('permissions', function($data){
                    $roles = $data->permissions()->get();
                    $badges = '';
                    foreach ($roles as $key => $role) {
                        $badges   .=   '<span   class="badge   badge-dark   m-
1">'.$role->name.'</span>';
                    }
                    if($data->name == 'Super Admin'){
                        return  '<span  class="badge  badge-success  m-1">All
permissions</span>';
                    }

                    return $badges;
                })
                ->addColumn('action', function($data){
                    if($data->name == 'Super Admin'){
                        return '';
                    }
                    if (Auth::user()->can('manage_roles')){
                        return '<div class="table-actions">
                                    <a  href="'.url('role/edit/'.$data->id).'"
><i class="ik ik-edit-2 f-16 mr-15 text-green"></i></a>
```

```
                                    <a      href="'.url('role/delete/'.$data-
>id).'"  ><i class="ik ik-trash-2 f-16 text-red"></i></a>
                                </div>';
                    }else{
                        return '';
                    }
                })
                ->rawColumns(['permissions','action'])
                ->make(true);
    }

    /**
     * Store new roles with assigned permission
     * Associate permissions will be stored in table
     */

    public function create(Request $request)
    {
        // laravel default validator
        $validator = Validator::make($request->all(), [
            'role' => 'required'
        ]);

        if ($validator->fails()) {
            return redirect()->back()->withInput()->with('error', $validator-
>messages()->first());
        }
        try{

            $role = Role::create(['name' => $request->role]);
            $role->syncPermissions($request->permissions);

            if($role){
```

```
                return   redirect('roles')->with('success',   'Role   created
succesfully!');
            }else{
                return  redirect('roles')->with('error',  'Failed  to  create
role! Try again.');
            }
        }catch (\Exception $e) {
            $bug = $e->getMessage();
            return redirect()->back()->with('error', $bug);
        }
    }

    public function edit($id)
    {
        $role  = Role::where('id',$id)->first();
        // if role exist
        if($role){
            $role_permission = $role->permissions()
                                    ->pluck('id')
                                    ->toArray();

            $permissions = Permission::pluck('name','id');

            return                                          view('edit-roles',
compact('role','role_permission','permissions'));
        }else{
            return redirect('404');
        }
    }

    public function update(Request $request)
    {
```

```
        // update role
        $validator = Validator::make($request->all(), [
            'role' => 'required',
            'id'   => 'required'
        ]);

        if ($validator->fails()) {
            return redirect()->back()->withInput()->with('error', $validator-
>messages()->first());
        }
        try{

            $role = Role::find($request->id);

            $update = $role->update([
                        'name' => $request->role
                    ]);

            // Sync role permissions
            $role->syncPermissions($request->permissions);

            return  redirect('roles')->with('success',  'Role  info  updated
succesfully!');
        }catch (\Exception $e) {
            $bug = $e->getMessage();
            return redirect()->back()->with('error', $bug);

        }
    }


    public function delete($id)
    {
        $role   = Role::find($id);
```

```
        if($role){
            $delete = $role->delete();
            $perm   = $role->permissions()->delete();

            return redirect('roles')->with('success', 'Role deleted!');
        }else{
            return redirect('404');
        }
    }
}
```

Б1. Скріншот повної сторінки Dashboard

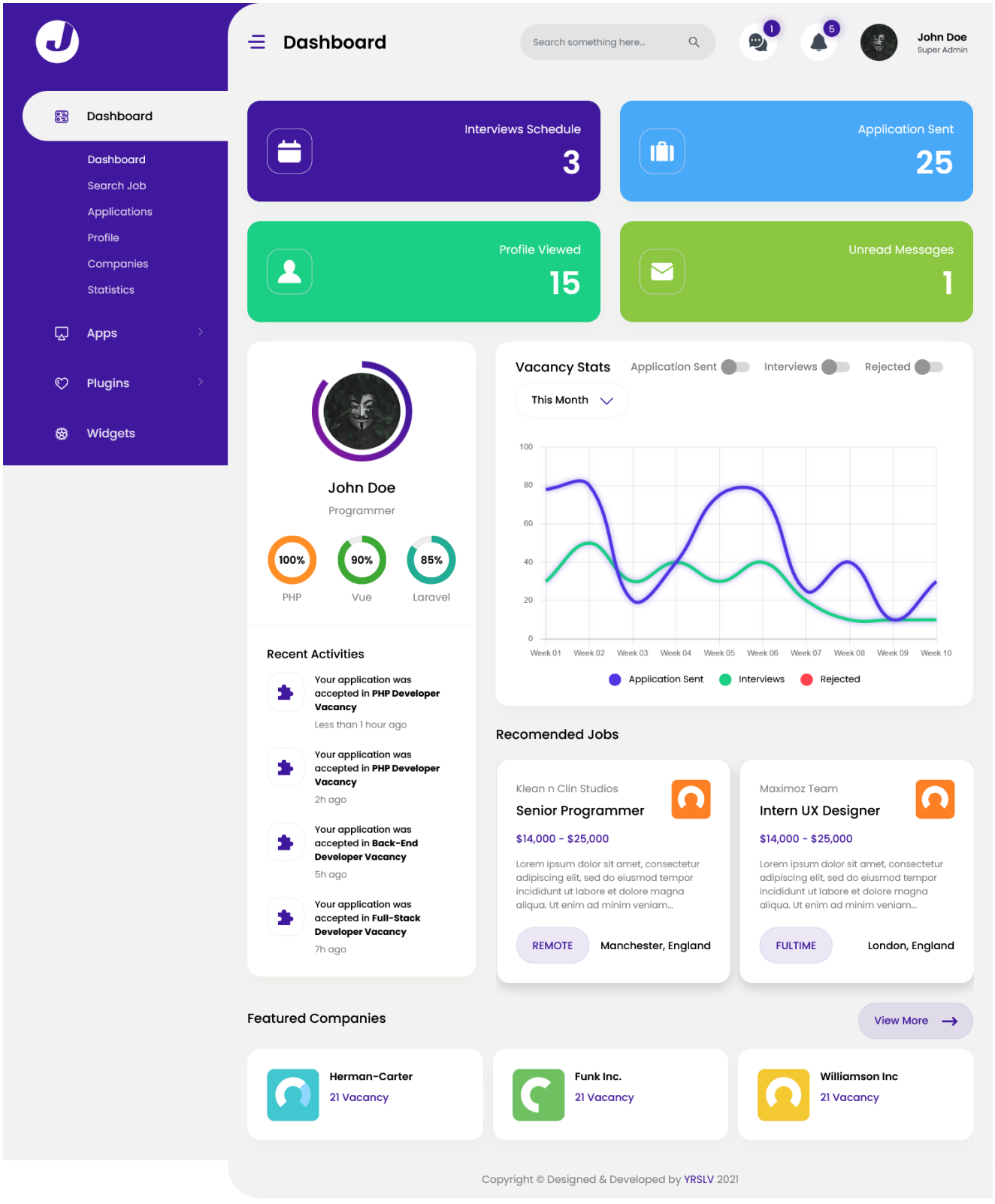

Рис. Б.1. Сторінка Dashboard повністю

Б2. Скріншот повної сторінки Statistics

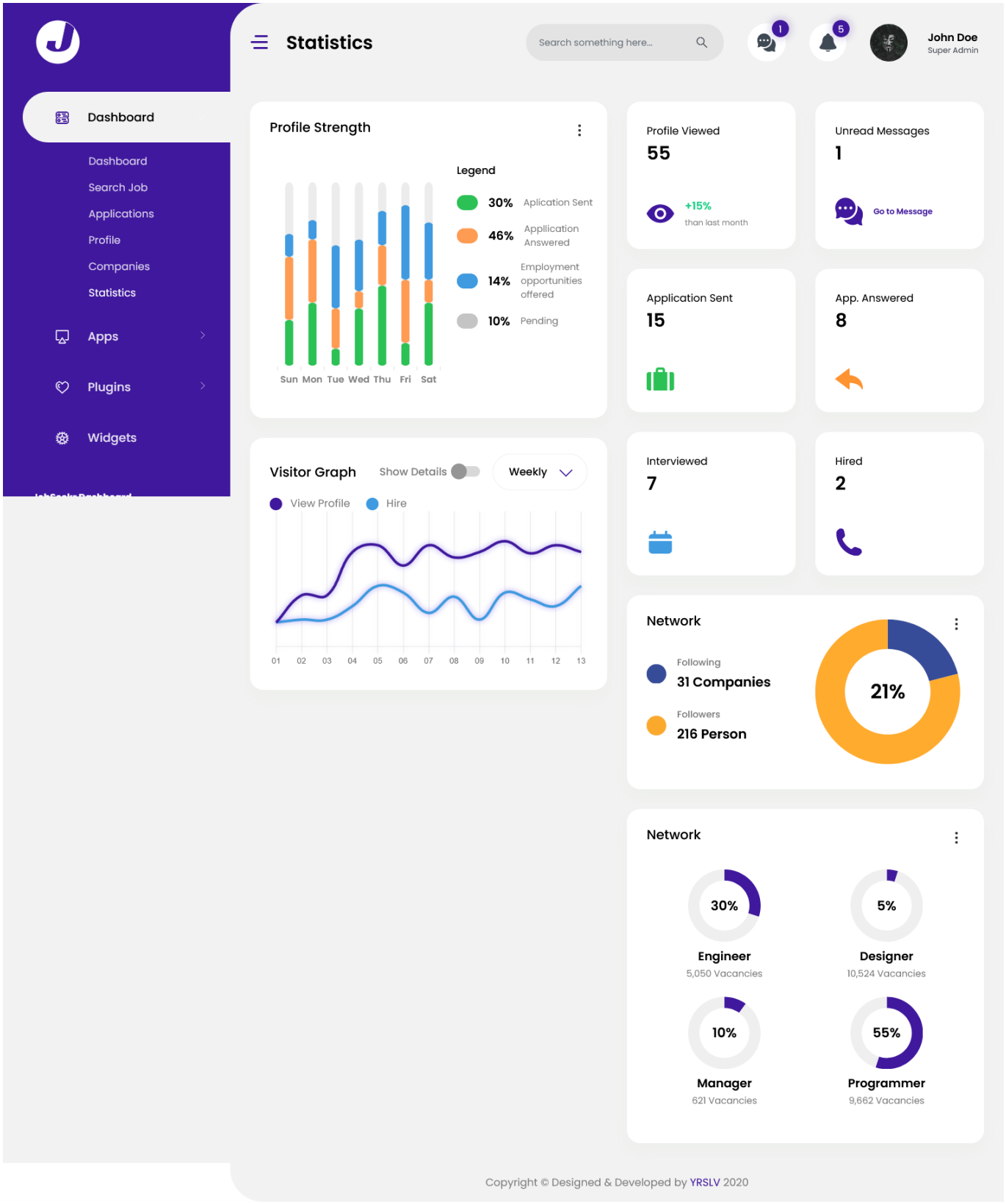

Рис. Б.2. Сторінка Statistics повністю