КВАЛІФІКАЦІЙНА РОБОТА

Група ІПЗс-2019

Фараджев Т.Р.

2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук
Кафедра інформаційних технологій

на правах рукопису

Фараджев Тимур Рамізович

УДК 004.378

Розробка системи управління і використання корпоративних програм лояльності засобами Ruby, Ruby on Rails, PWA

Спеціальність 121 – «Інженерія програмного забезпечення»
Кваліфікаційна робота на здобуття кваліфікації бакалавра

Нормоконтроль	Студент
Стисло О.В.	Фараджев Т.Р.
(підпис, дата, розшифровка підпису)	(підпис, дата, розшифровка підпису)
Допускається до захисту Завідувач кафедри	Керівник дипломного проєкту
к.т.н.,доц. Пашкевич О.П.	к.ф-м.н., доц. Бойчук А.М.
(підпис, дата, розшифровка підпису)	(підпис, дата, розшифровка підпису)

Івано-Франківськ – 2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА
Факультет суспільних та прикладних наук
Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»
Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ:
Завідувач кафедри Пашкевич О.П._____
_____
„_____”_____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Фараджев Тимур Рамізович
(прізвище, ім’я, по батькові студента)

1. Тема кваліфікаційної роботи_Розробка системи управління і використання__ корпоративних програм лояльності засобами Ruby, Ruby on Rails, PWA_____
керівник роботи:
Бойчук Андрій Михайлович, кандидат фізико-математичних наук_____
затверджена наказом закладу вищої освіти від „11” листопада 2022 року_____ № 155/1НВ
2. Термін здачі студентом закінченого проєкту_14 червня 2023 року_____
3. Вихідні дані роботи_____існуючі рішення по використанню корпоративних програм лояльності, технології програмування Ruby та RubyOnRails, технології Progressive Web Application, матеріали та результати, отримані під час_____ проходження переддипломної практики._____
4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
_1) Аналіз сучасного стану проблеми та постановка завдання_____
_2) Вибір компонентів та технологій реалізації, проєктування архітектури_____
_3) Розробка прототипу системи _____
_4) Економічне обґрунтування_____
5. Дата видачі завдання__10 жовтня 2022 року_____

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області та постановка задачі	15.02.2023	Виконано
2	Вибір технологій для реалізації завдання. Архітектура та алгоритми	14.03.2023	Виконано
3	Імплементація програмного забезпечення	07.04.2023	Виконано
4	Створення бізнес-плану	28.04.2023	Виконано
5	Оформлення пояснювальної записки	19.05.2023	Виконано
6	Оформлення графічного матеріалу та підготовка до захисту роботи	03.06.2023	Виконано

Студент Фараджев Т.Р.
(підпис) (розшифровка підпису)

Керівник роботи Бойчук А.М.
(підпис) (розшифровка підпису)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Сторінка	Опис графічного матеріалу	Сторінка	Опис графічного матеріалу
14	Скріншоти додатку Discontimo	50	Скріншот сторінки керування Organisation
15	Блок-схема функцій Discontimo для ролі клієнта	51	Скріншот сторінки керування ServiceCompany
16	Скріншоти мобільного додатку IT Club Loyalty	52	Скріншот сторінки керування користувачем
18	Скріншоти мобільного додатку IT Loyalty	54	Надісланий електронний лист із запрошенням
23	Приклад UML-діаграми для опису програми	56	Перегляд профілю бізнесу
25	Структура моделі C4 Model	57	Перелік спеціальних пропозицій
27	Діаграма Context 1 рівня C4 Model	59	Перелік бізнесів і їх базових знижок

АНОТАЦІЯ

Метою проєкту є розробка програмного забезпечення для управління і використання корпоративних програм лояльності зі знижками та спеціальними промозиціями.

Визначено технології і спроєктовано архітектуру реалізації, проаналізовано актуальні аналоги, їх переваги і недоліки.

Розроблена програма дозволить бізнесам поширювати знижки та спеціальні пропозиції, а зацікавленим компаніям — ітнегрувати їх як частину соціального пакету для своїх працівників.

КЛЮЧОВІ СЛОВА: ПРОГРАМИ ЛОЯЛЬНОСТІ, ЗНИЖКИ, БІЗНЕС, КОМПАНІЇ, СОЦІАЛЬНІ ПАКЕТИ, ПРОГРЕСИВНІ ВЕБ-ЗАСТОСУНКИ, ВЕБСАЙТ, МОБІЛЬНИЙ ДОДАТОК, RUBY, ROR, POSTGRESQL.

SUMMARY

The purpose of the project is to develop a software for management and usage of corporate loyalty programs containing discounts and special offers.

Detected the technologies and projected architecture of the realization. Analyzed analogues, their pros and cons.

The developed multi-platform app will allow businesses to share discounts & special offers and companies to integrate them as part of social packages for their employees.

KEYWORDS: LOYALTY PROGRAMS, DISCOUNTS, BUSINESS, COMPANIES, SOCIAL PACKAGES, PROGRESSIVE WEB APP, WEBSITE, MOBILE APP, RUBY, ROR, POSTGRESQL.

ЗМІСТ

ВСТУП

В Україні встановлена ринкова економіка і, відповідно до неї, з кожним роком зростає конкуренція — за клієнтів та працівників.

Незмінно актуальним способом привернення клієнтської уваги до бізнесу є фінансова вигідність його продукції чи послуг. Нерідко бізнеси вдаються до впровадження так званих «програм лояльності», які дозволяють створити умови, за яких персоналізовані чи обмежені в доступі знижки «притягують» нових та клієнтів, а також повертають колишніх.

З іншого боку, компанії, які турбуються про свою репутацію і якість наданих їх працівниками послуг, стараються наймати найкращих та створювати привабливі умови роботи. Це випливає у корпоративні соціальні пакети — набір додаткових програм і можливостей для працівників, що є бажанням компанії, поверх обов’язкових законом заробітної плати і оплачуваних відпусток, лікарняних тощо.

Проблемою одних сучасних бізнесів є вихід їх програм лояльності на широку аудиторію, а інших — розширення своїх соцпакетів. Закриття цих проблем кожним бізнесом потребує вкладень чималих фінансів та часу, аби дослідити ринок, обрати стратегію, налаштувати процеси, впровадити ідею в реальність і перевірити її працездатність. Як альтернативний варіант вирішення проблем обидвох напрямків пропонується застосувати програмне забезпечення, де одні компанії пропонують програми лояльності, а інші — відкривають до них доступ для своїх працівників.

Метою роботи є спрощення і здешевлення впровадження простих програм лояльності та корпоративних соцпакетів для малих і середніх за виручкою та кількістю працівників компаній.

Завданням роботи є розробка прототипу застосунку для внесення бізнесів до бази даних, надання їм доступу для встановлення своїх знижок і пропозицій, а також внесення компаній і їх працівників для відкриття їм можливості скористатись пропозиціями будь-якого бізнесу в системі.

Методи роботи. Ми будуємо новий продукт, точна концепція якого ще не була використана і випробувана в аналізованих програмах. Тому, перш ніж інвестувати час в розробку програми з розширеним функціоналом, варто перевірити працездатність її концепції. Отже програма має бути розроблена з урахуванням вимоги щодо зменшення несуттєвого функціоналу і часу на його розробку і подальшу підтримку.

Результати роботи. Застосунок, що дозволить швидко і без додаткових витрат розгортати програми лояльності на широку аудиторію. А також швидко та дешево розширяти корпоративні соціальні пакети і надавати доступ до привабливих знижок з будь-якого девайсу з браузером і доступом до мережі інтернет. В результаті, працівник однієї компанії стає клієнтом іншої.

Структура роботи. Розділи – 4. Загальний обсяг основної частини – 61 сторінка. Список використаних джерел – 48. Додатки – 17.

РОЗДІЛ 1. ПОНЯТТЯ ПРОГРАМ ЛОЯЛЬНОСТІ І СОЦПАКЕТІВ, ПОСТАНОВКА ЗАДАЧІ

1.1 Поняття програми лояльності

Програма лояльності — комплекс маркетингових дій бізнесу для заохочення покупців [1]. Найпростіша форма програми лояльності працює таким чином, що під час покупки клієнт оплачує суму з вирахуванням наданої постійної чи накопиченої знижки.

Програми лояльності не є новинкою, але вони все ще є актуальними і приносять брендам позитивні результати в рекламі, утриманні клієнтів і прибутках. Все більш популярними серед людей стають програми лояльності на послуги і сервіс у сфері обслуговування, аніж на фізичні продукти, таким чином у 2020 році 72% клієнтів готові рекомендувати бізнеси з хорошими програмами лояльності а в середньому 78% існуючих програм мотивують клієнтів залишатись з бізнесом [2].

1.2 Різновиди програм лояльності

Існує безліч варіантів влаштування програми лояльності, з кожним роком бізнеси вдаються до все більш креативних рішень. Наприклад, всесвітня мережа кав’ярень Starbucks дає безлімітних доступ до безкоштовних доданків до кави тим клієнтам, які своїми попередніми витратами накопичили більше 25 «зірок» у їх мобільному додатку [3].

Розглянемо найбільш поширені формати.

1.2.1 Знижки

Найпростішим елементом програми лояльності є знижка на товари чи послуги. Знижка — це поправка до базової ціни під час реалізації продукції [4].

Економісти розділяють знижки на різноманітні види, з них основними є: касові, кількісні, спеціальні, функціональні, сезонні, фінальні, карткові [5]. Останні — це знижки покупцям, які мають знижкову картку відповідного бізнесу. Відсоток такої знижки зазвичай коливається в межах від 10 до 15 [5].

1.2.2 Спецпропозиції

Переважна більшість інших видів знижок надається за спеціальними умовами та / або протягом певного періоду. Наприклад, 10% знижки на покупку зимових речей у березні. Або 20% знижки на всі позиції при замовленні продуктів на самовиніс.

Для спрощення розгляду цього набору категорій знижок, узагальнимо їх терміном «спеціальні пропозиції».

1.3 Корпоративні соцпакети

Хороші компанії зацікавлені у позитивній репутації серед клієнтів та працівників. Для надання якісного обслуговування, працівники мають бути задоволені своєю роботою. І рівень зарплати сьогодні не найважливішою та єдиною перевагою в конкуренції роботодавців [7].

Один з методів створення і підтримки позитивного сприйняття компанії як роботодавця — соціальний пакет. Соцпакет компанії надає працівникам додаткові бенефіти, поза звичайним відшкодуванням за пророблену роботу. Туди може входити як розширене медичне страхування, надання особистої високоякісної техніки, та і програм лояльності у компаніях-партнерів, наприклад підвищення до квитків бізнес-класу у певного авіаперевізника. Наша програма

надаватиме можливість компаніям отримувати доступ до знижок і пропозицій від безлічі бізнесів за час і ціну, неспівмірних до власних витрат на особистий пошук і встановлення партнерства з бізнесами.

Мабуть, будь-який роботодавець вже зіткнувся з тим, що називається ринком кандидатів і точно розуміє, що зусиль і коштів на залучення і утримання потрібних бізнесу фахівців доводиться з кожним роком витрачати все більше і більше. Якщо ми хочемо утримати співробітника надовго, то повинні пропонувати йому щось якісно відмінне від набору «зарплата, гарантована відпустка і лікарняні». Тому у великих компаніях, які працюють і розвиваються в умовах дефіциту персоналу, боротьба за співробітників йде через соціальні пакети зокрема. Чай/кава печиво/корпоративи — цим уже нікого не здивуєш. Безкоштовне навчання або компенсація витрат співробітника на розвиток — теж уже стало досить поширеним рішенням.

Щоб співробітники мали бажання залишатися і розвиватися разом з компанією, треба постійно вивчати їхні очікування і підлаштовувати рішення в сфері компенсацій і бенефітів.

1.4 Аналоги рішень через масове програмне забезпечення

Популярні програми лояльності в Україні мають як фізичні, так і електронні портативні реалізації:

- невеликі пластикові або паперові картки (напр., Watson’s [8]);
- брелоки зі штрихкодом (напр., WINETIME [9]);
- мобільні додатки (Сільпо [10]);
- смс (SALAD [11]).

Виготовлення фізичних карток і брелоків є важким процесом, яких тягне за собою ускладнення багатьох інших — їх менеджмент серед компаній і працівників, інтеграція бізнесу з новою системою тощо. Більше того, функціональні вимоги щодо спеціальних пропозицій не можуть бути покриті

даним чином. Тому, такі варіанти залишаються останніми у переліку пріоритетних портативних рішень для нашої розробки.

Витрати на розробку СМС-інтерфейсу, його відносна незручність для використання і лімітовані можливості, ставить його другим можливим.

Мобільний додаток, в порівнянні з іншими варіантами, є найкращою опцією для можливості портативного використання системи кінцевими користувачами. Саме тому ми обираємо мобільний додаток в якості однієї з технічних вимог до розробки.

Розглянемо існуючі аналоги мобільних додатків, в яких реалізовано функції масових програм лояльності зі знижками і спеціальними пропозиціями від багатьох бізнесів.

1.4.1 Публічний застосунок Discontimo

Перші згадки про презентацію українського стартапу Discontimo датуються 2015 роком [12].

З моменту свого запуску, цей мобільний додаток був поширений на ринки міст Івано-Франківськ, Чернівці, Вінниця та інші. Проте вже у 2017 він перестав отримувати оновлення на Android [13], згодом додаток зовсім зник з офіційного Google Play Market [14], а соцмережі стали неактивними [15]. Розглянемо як цей застосунок працював та що стало ймовірною причиною його закриття.

Використовувати Discontimo могли бізнеси і клієнти.

Після встановлення клієнтського додатку на смартфон, користувач має пройти авторизацію. Далі відкривається спектр можливостей переглянути доступні постійні чи спеціальні пропозиції переліком, на мапі тощо (рис. 1.1).

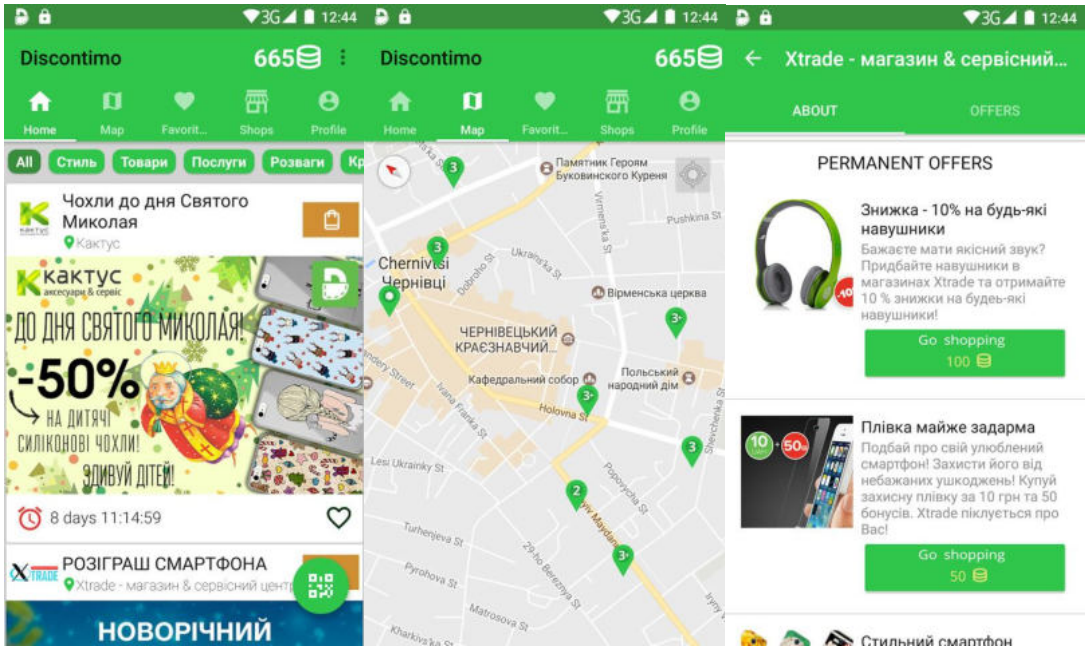


Рисунок 1.1 – Скріншоти додатку Discontimo

Скористатись цими пропозиціями можна обравши позицію і надавши код для її застосування представникові бізнесу, який її активізує, в обмін на бали Discontimo. Сто балів додаток надавав при першій реєстрації. Надалі аби накопичити бали користувач мав реєструвати свій код для накопичення у представника бізнесу в момент оплати. Кількість бонусів залежала від витраченої покупцем суми.

З іншого боку, бізнес отримує доступ до бізнесової версії додатку після підписку на оплату послуг додатку. Далі бізнес може публікувати свої пропозиції для клієнтів, встановлювати на них ціну в балах, переглядати статистику, а також активовувати коди на пропозиції і накопичення балів клієнтові.

Переваги такого підходу: кожен може стати користувачем, безкоштовно, додаток простий у використанні і має багато пропозицій.

Є й ряд недоліків. По-перше, клієнти обов’язково мають витрачати і реєструвати витрати через додаток, аби мати можливість пізніше скористатись пропозиціями (рис. 1.2). Тому, можемо припустити, що незбалансованість вимог і можливостей для клієнта, могла їх відштовхувати, замість того, аби притягувати. По-друге бізнес зобов’язаний постійно сплачувати внески за кожен перегляд свого профілю та своїх пропозицій клієнтами [16], інакше бізнес

втрачатиме доступу до даного додатку. Цей фактор міг відштовхувати вже сторону бізнесу від співпраці з додатком.

Рисунок 1.2 – Блок-схема функцій Discontimo для ролі клієнта

Робимо висновок, що зазначені недоліки могли бути причиною занепаду і врешті припинення існування стартапу Discontimo. Отже, застосунок має містити якомога менше перешкод до можливості користуватись функціями.

1.4.2 Корпоративний застосунок IT Club Loyalty

«IT Club Loyalty» — ще один мобільний додаток з функціоналом програми лояльності. Запущений організацією «Львівський IT Кластер» в 2014 році у Львові, розширений на Київ і Одесу у 2018 [17], і надалі активно розвивається.

Основними ролями користувачів додатку є: бізнес, клієнт і IT-компанія. Працює це наступним чином: бізнес безкоштовно отримує доступ до можливості публікувати свої пропозиції. IT-компанія платить єдиний щомісячний внесок за доступ до програми для своїх працівників (клієнтів для бізнесів).

Клієнт авторизовується у додатку за наданим роботодавцем доступом. Далі клієнт може переглянути пропозиції (рис. 1.3), і аби ними скористатись, має показати свій ідентифікаційний номер з додатку представникові бізнесу. Зі свого боку, бізнес може перевірити актуальність номеру через бізнес-версію додатку. Через цей же додаток бізнес наповнює системи своїми пропозиціями. А також

може користуватись додатковими маркетинговими інструментами для рекламування і відображення в топі, за додаткову плату.

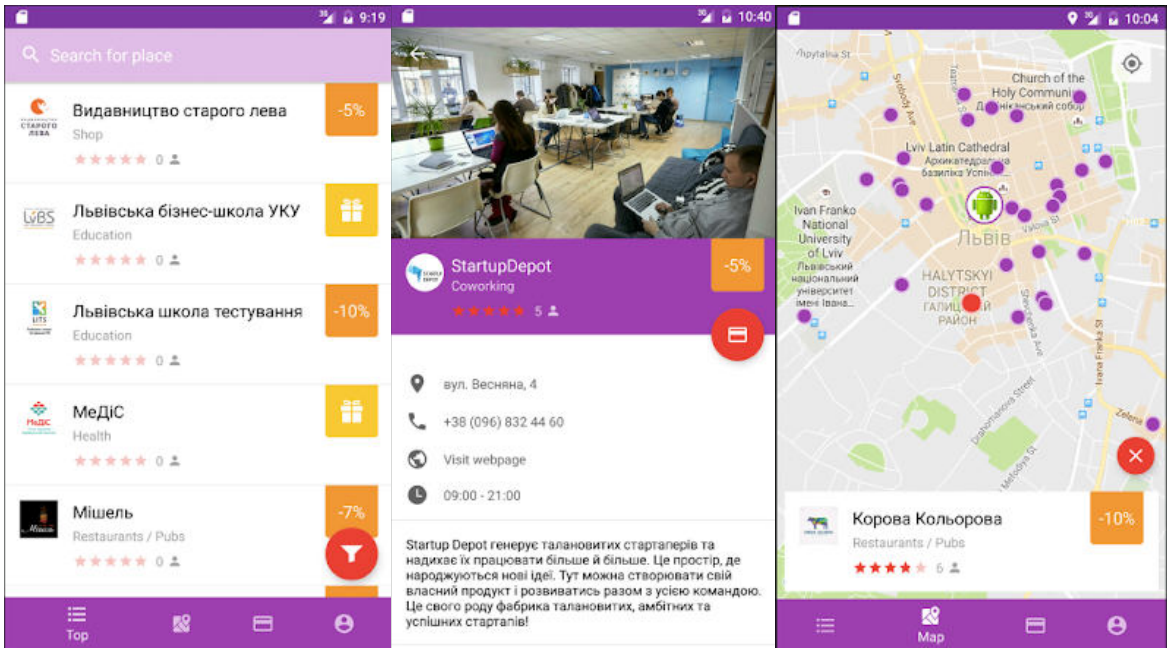


Рисунок 1.3 — Скріншоти мобільного додатку IT Club Loyalty

IT Club Loyalty має очевидні переваги для усіх сторін: бізнес отримує безкоштовну можливість поширювати пропозиції з таргетом на чітку аудиторію IT-сфери, IT-компанія отримує доповлення до свого соцпакету за символічну ціну [18], а працівники – ексклюзивні знижки і пропозиції.

Незважаючи на перелік переваг, все ж і цей додаток має два обмеження, які можна віднести до недоліків: підписатись на додаток можуть лише компанії IT-сфери та й лише трьох міст України (Львів, Київ і Одеса).

Підсумувавши складові IT Club Loyalty та кількість активних встановлень (як мінімум, більше 10000 у маркетплейсі Google Play [19]), робимо висновок, що його реалізація є успішною.

1.4.3 Корпоративний застосунок IT Loyalty

Додаток IT Loyalty з’явився 2018 року виключно для компаній-членів Харківського IT кластеру [20].

Основні функції застосунку такі ж, як і в IT Club Loyalty: ролі бізнес, клієнт і IT-компанія; містить купони на знижки у бізнесів, якими можуть користуватись клієнти (вони ж, працівники IT-компанії), є розділ новин (рис. 1.4).

Особливістю цього додатку є те, що доступ до нього обмежений виключно для IT-компаній-учасників Харківського IT кластеру. Відповідно, це обмеження поширюється і на бізнес-партнерів: вони мають надавати свої послуги у Харкові.

Отже, IT Loyalty так само продовжує успішно розвиватись на ринку Харкова, проте має такі ж недоліки, як і IT Club Loyalty, що стосуються обмежень по напрямку роботи компаній, їх розміщення та розміщення бізнесу, який надає пропозиції знижок і купонів через додаток.

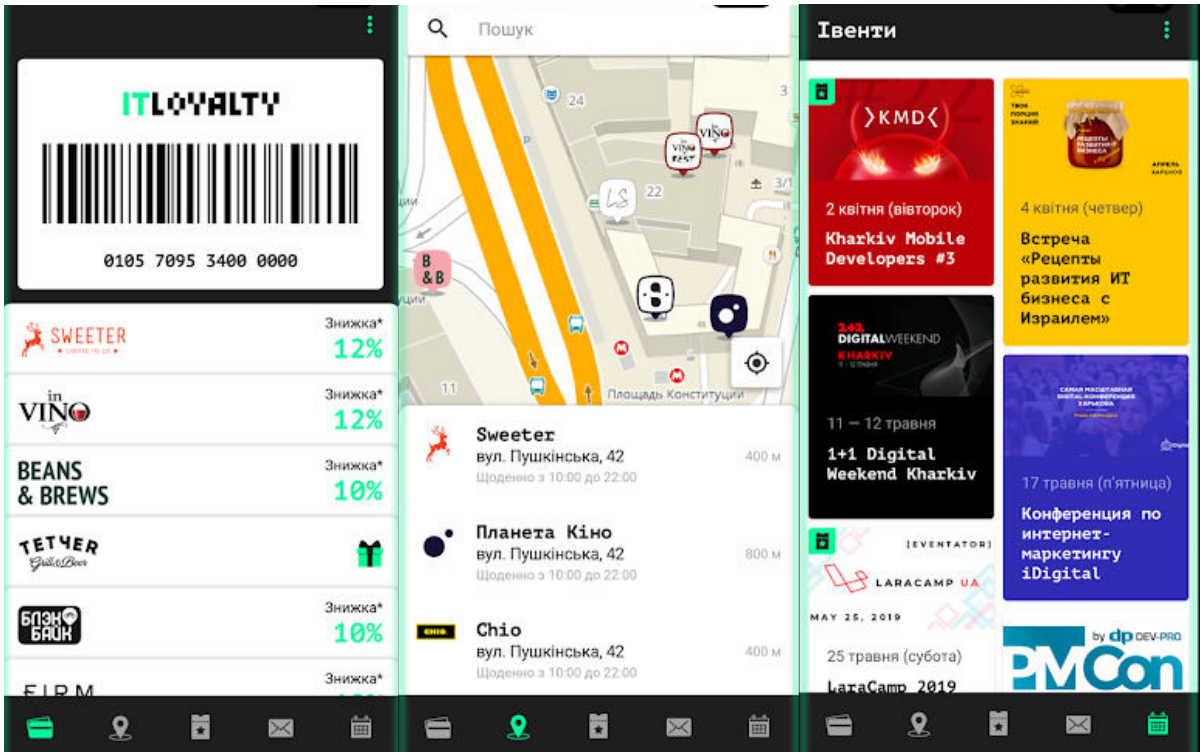


Рисунок 1.4 — Скріншоти мобільного додатку IT Loyalty

1.5 Постановка задачі

На основі зробленого аналізу додатків для звичайних і корпоративних користувачів, взявши до уваги їх маркетингові і бізнес стратегії, а також ідентифіковані проблеми, підготуємо вимоги до розроблюваного застосунку.

1.5.1 Функціональні вимоги

А. Автентифікація і ролі

Використання будь-якої закритої програми не можливе без авторизації.

Користувачі матимуть змогу увійти в застосунок за допомогою своєї електронної пошти та паролю.

В системі буде 4 ролі акаунтів користувачів:

- суперадмін;
- бізнес;
- компанія;
- працівник компанії.

Усіх користувачів додають, змінюють та видаляють суперадміни. Відкритої форми реєстрації немає. Компанія може керувати акаунтами працівників, асоційованих до себе.

Б. Знижки

Основне мета застосунку — надавати кінцевим користувачам можливість скористатись знижкою, яку надає бізнес. Отже, це і ключовий функціонал, з якого ми почнемо.

По-перше, бізнес має мати можливість встановлювати базову знижку на свої товари та послуги для компаній. Базова знижка — відсоткове зменшення ціни в чеку, без умов до її застосування (будь-яка сума, будь-який день, будь-яка категорія товарів і послуг).

Бізнес може не мати базової знижки. Відтак, його відображення в каталозі бізнесів можливе лише за наявності спеціальних пропозицій.

В. Спеціальні пропозиції

Бізнес може публікувати спеціальні пропозиції — тимчасові або безстрокові акції з особливими умовами до їх доступу.

У такій пропозиції бізнес додає обов’язковий коментар про обмеження, які стосуються цієї знижки (напр. яка група товарів підпадає під дію знижки, або яка

мінімальна сума чеку необхідна для її застосування, і т.п.), дата від і дата до, а також зображення за бажанням.

Г. Каталог

Усі працівники усіх компаній матимуть доступ до усіх знижок і спецпропозицій усіх бізнесів.

Для спрощення пошуку бізнесів, працівники компаній матимуть вкладку каталогу. Каталог складається з рядку пошуку і результатів пошуку бізнесів. Каталог посортований за назвою бізнесу. Додаткові варіанти сортування: найбільш популярні, найбільший відсоток знижки. Працівники можуть шукати бізнеси за категоріями, містами та назвами.

З результатів пошуку можна відкрити сторінку бізнесу і переглянути інформацію про бізнес: назву, логотип, міста, контактні номер телефону і електронна пошта; а також: базову знижку і спеціальні пропозиції.

Бізнес може редагувати свою інформацію зі спеціальної сторінки.

Ґ. Картка працівника

Кожен працівник має унікальний шестизначний цілочисельний номер в програмі. Він присвоюється при додаванні працівника в систему. Унікальний номер змінити неможливо. Переглянути його можна на особистій сторінці картки працівника. Ця сторінка оформлена у вигляді картки з трьома елементами: фото працівника, унікальний номер і логотип компанії працівника.

Картка працівника слугує валідатором для бізнесу. Коли працівник має намір скористатись знижкою бізнесу, представник бізнесу має право попросити показати картку працівника в застосунку. Представник бізнесу має інструмент у застосунку для перевірки справжності і статусу картки працівника за допомогою наступного процесу:

- представник бізнесу вводить номер працівника на вкладці валідації у застосунку для бізнесу;
- застосунок перевіряє чи існує працівник з таким номером у застосунку і чи його компанія оплатила поточний місяць доступу до застосунку;

- представник бізнесу отримує результат: підтвердження актуальності даних (відображається та сама картка працівника з фото, номером і логотипом);
- представник може додатково візуально перевірити відповідність фото на картці з реальною людиною перед собою.

За допомогою цього процесу надаються наступні гарантії:

- бізнесу — що доступ до унікальних знижок залишається лише в межах компаній, що мають доступ до програми;
- працівникам компаній — що їх особисті дані (ім’я і прізвище, електронна пошта, номер телефону, тощо) недоступні стороннім особам.

1.5.2 Нефункціональні вимоги

Д. Мобільний додаток

Користувачам необхідне портативне рішення доступу до знижок. Оскільки частина з них матиме наміри скористатись товарами або послугами бізнесів, знаходячись безпосередньо в їх місці розташування. Час на застосування системи знижок має бути мінімізований — будь-які затримки є перешкодою, яка відштовхує користувачів.

Е. Веб-інтерфейс

Такі ролі, як суперадмін, бізнес і компанія потребують доступу до масових операцій (напр., імпорт/експорт працівників) і стаціонарного доступу (як от робочий комп’ютер керівника відділу кадрів).

Зважаючи на це, клієнтську частину застосунку варто реалізувати через веб-інтерфейс, використовуючи сучасні браузери.

Висновки до розділу 1

Дуже багато бізнесів вже мають масові пропозиції (напр. «Кожна шоста кава в подарунок»), які спонукають клієнтів витрачати більше і повертатись

знову. А з персоналізованими програмами лояльності клієнти залишаються з бізнесом в 5.4 рази довше і витрачають в 6.4 рази більше, аніж з масовими [6].

Місцеві кав’ярні і міжнародні медцентри, фінустанови та прокат велосипедів, готелі і кінотеатри, юридичні консультанти та школи іноземних мов тощо — всім необхідні платоспроможні клієнти і можливість тарґету на них.

Саме це і є можливістю принести більшу цінність власникам бізнесів. А особливо тим, які не готові вкладати великі гроші у розробку власної системи лояльності і супроводжуючого програмного забезпечення.

Існуючі мобільні додатки, які ставили за мету опанувати цю можливість на ринку, не досягли значних результатів через негативний користувацький досвід та суттєві обмеження, як от доступність за містами або обнулення кількості балів на персональному рахунку.

Зважаючи на це, ми ставимо за мету створити додаток – майданчик для співпраці одного бізнесу з іншим бізнесом. Де обидві сторони отримують постійну цінність. В одних – у вигляді тарґетованої авдиторії, а в інших – елемент соціального пакету працівника.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ТА ВИБІР ТЕХНОЛОГІЙ

2.1 Проєктування архітектури за допомогою C4 Model

2.1.1 Обґрунтування вибору моделі

Описати архітектуру програмного забезпечення таким чином, аби її однаково розуміли — є дійсно складною задачею. Існують стандарти, як от UML або ж ArchiMate, але вони містять обмеження, особливо коли треба подивитись на програму з різних перспектив. Навіть якщо всі діаграми вчасно оновлюються та створюються неперевершеними експертами, що в реальному житті є рідкістю.

Розглянемо варіант UML (Unified Modeling Language). UML є найбільш популярним рішенням для візуалізації діаграм і була затверджена міжнародним стандартом ISO у 1997 році [21].

Це універсальна візуальна мова і застосовується для описання конкретної проблеми будь-якого масштабу. UML вимагає деталізації на низькому рівні, тобто до найменших деталей (див. Додаток А).

В результаті UML діаграми є найбільш доцільними коли використовується водоспадна методологія розробки [22]. Наприклад, у виробничій сфері, як от автомобільна [23] – де тенденція до змін в процесі виготовлення є мінімальною.

Архітектура програмного забезпечення настільки складна, що можна не знати найкращого рішення, поки не почати процес розробки. Це означає, що проблему неможливо адекватно визначити, поки її вже не вирішено. Для цього є термін: злісна проблема (англ. wicked problem) [24].

Наша стратегія — поступово вдосконалювати архітектуру шляхом регулярного рефакторингу. Тому обрано гнучку методологію розробки (Agile) [25]. Це дозволить програмному забезпеченню ефективніше адаптуватися до непередбачуваних змін.

Отже, UML наразі не є підходящим варіантом для нашої програми.

Для цієї стратегії існує краща альтернатива. Розглянемо метод C4 Model [26]. Він базується на декомпозиції системи на контейнери і компоненти та побудові зв’язків між ними [27].

З переваг C4 виділяють:

- легка у підтримці;
- може бути корисною нетехнічному персоналу (менеджери, дизайнери тощо);
- уникає неоднозначних позначень;
- деякі деталі можуть бути сформовані автоматично (залежно від використовуваного інструменту побудови).

До недоліків відносятся:

- статична візуалізація. З діаграми неможливо визначити коли і як часто трапляється зв’язок між елементами;
- абстракції низького рівня (наслідуючи принцип UML) доволі крихкі і потребують регулярного оновлення.

Перший недолік не є критичним, адже його можна мінімізувати описуючи періодичність зв’язків у примітках до елементів, або бізнес-документації. Другий — також наслідуваний з того, що низькорівневі абстракції найчастіше використовують стиль UML.

Виходячи з цього, використаємо метод C4 Model для проєктування та візуалізації архітектури розроблюваного застосунку.

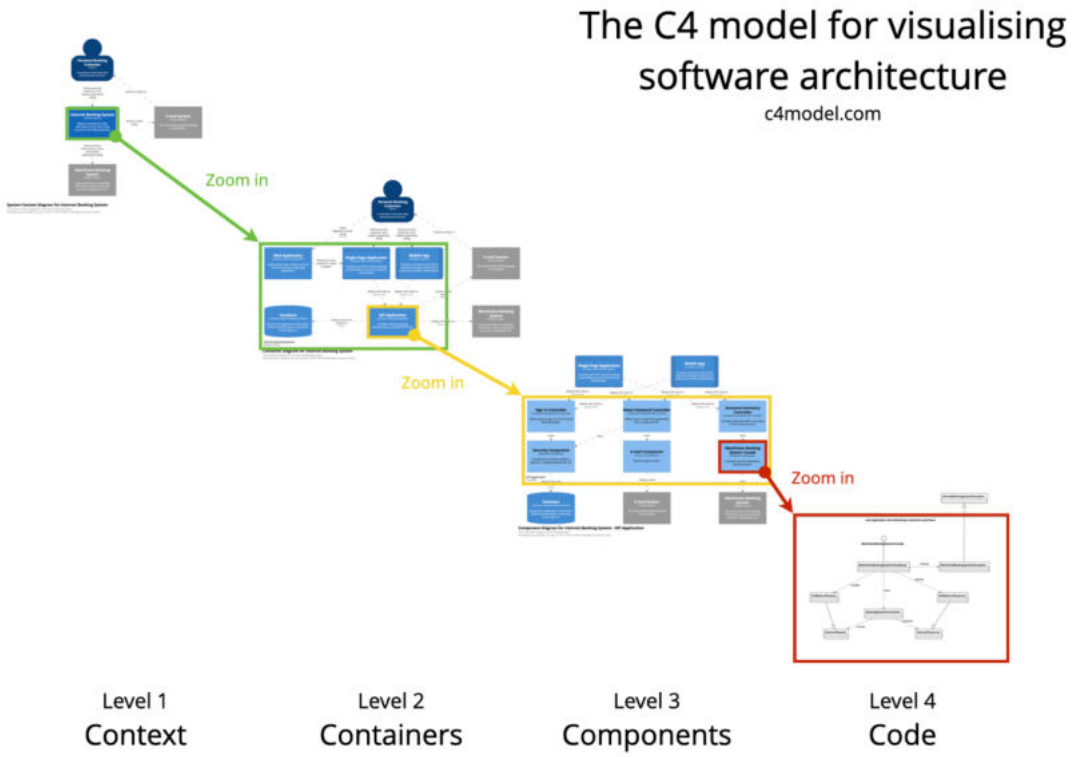


Рисунок 2.1 — Структура моделі C4 Model

C4 Model має чотири основні рівні (рис. 2.1):

- 1 рівень: Context — показує систему у масштабі її взаємодії з користувачами та іншими системами;
- 2 рівень: Container — розбиває систему на взаємопов'язані контейнери (виконувані підсистеми або програми);
- 3 рівень: Component — розділяють контейнери на взаємопов'язані компоненти і відображають зв'язки компонентів між собою, або іншими (наприклад, зовнішніми) системами;
- 4 рівень: Code — надає додаткову інформацію про дизайн архітектурних елементів, які можна порівняти з програмним кодом. Цей рівень спирається на інші нотації, як от UML або ERD.

C4 — це динамічна діаграма, відображення якої можна збільшувати та зменшувати відповідно до того рівня, який нас цікавить. Так само, як на Google Maps. Чим більше масштабування, тим видно більше деталей.

Цей сет можливостей забезпечує швидке документування, гнучкість до оновлень та переосмислення компонентів використовуючи дизайн-мислення.

2.1.2 Рівень 1: Context

Починаємо з найвищого рівня абстракцій. На першому рівні C4 Model технічні деталі не є важливими. Весь фокус — на системи, ролі користувачів та їх взаємодії (рис. 2.2).

Система — це елемент нашого додатку, який приносить цінність користувачам. Це самостійне рішення, яке можна використовувати окремо.

Ролі користувачів:

- ServiceCompany —бізнес, який надає знижки і спецпропозиції;
- Manager — супервайзер або власник компанії, працівникам якої надано доступ до коритування знижками і спеціальними пропозиціями;
- Employee — працівник компанії з доступом до корпоративних знижок і спецпропозицій.

Користувачі не мають жодних прямих зв’язків між собою, в контексті роботи з системою.

Залучені системи:

- Corporate Loyalty Program — розроблювана в рамках даної бакалаврської роботи, система управління і використання корпоративних програм лояльності. Є основною системою в даній роботі;
- Email System — існуюча стороння система відправки повідомлень на електронну пошту.

Email System інтегрована з Corporate Loyalty Program таким чином, що остання надсилає API-запити [28] до першої для надсилання необхідних повідомлень на електронні пошти користувачів.

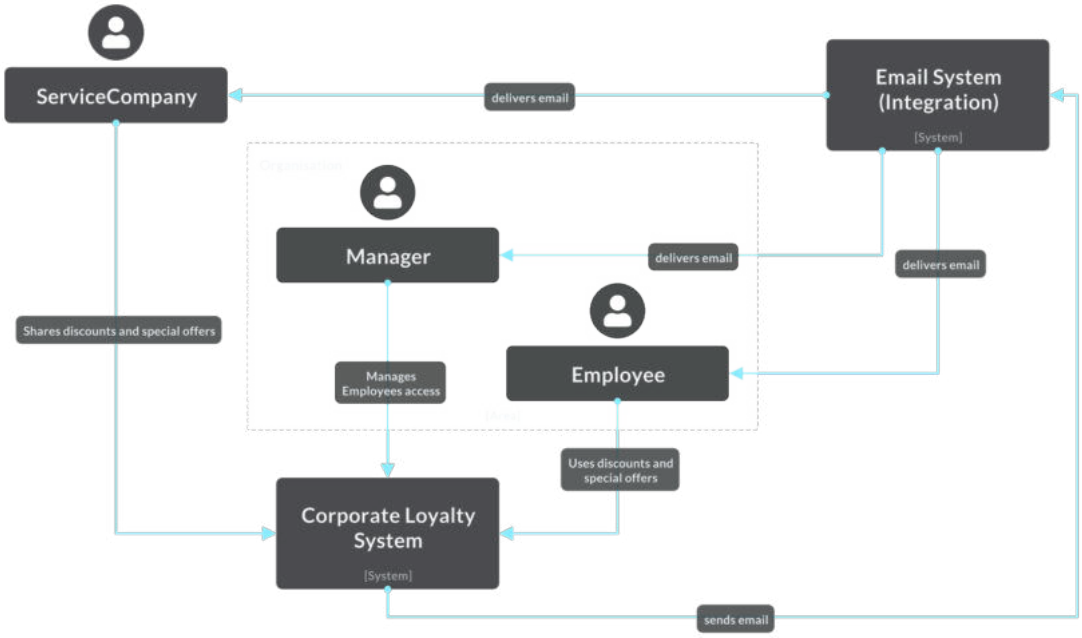


Рисунок 2.2 — Діаграма Context 1 рівня C4 Model

Зв’язки між системою та користувачами влаштовані наступним чином:

- ServiceCompany поширює свої знижки і персональні пропозиції;
- Manager керує налаштуваннями, доступом своїх працівників до системи в якості користувачів ролі Employee;
- Employee переглядає і користується доступними знижками та персональними пропозиціями.

Тепер легко побачити, що є нашою основною системою та хто є нашими ключовими користувачами. Тепер перейдемо до більш детальних елементів основної системи.

2.1.3 Рівень 2: Containers

Наступний рівень моделі C4 деталізовує основну систему за допомогою контейнерів і зв’язків між ними (рис. 2.3).

Контейнери — це більші частини програми, які потенційно можуть бути розгорнуті окремо. В нашому випадку це застосунки та бази даних. Вони спілкуються між собою за допомогою API (наприклад, SOAP, gRPC, REST) і не

залежать безпосередньо одна від одної. На цьому рівні нас також цікавлять протоколи та технології, які використовує кожен контейнер.

Застосунки системи:

- монолітний застосунок Progressive Web Application, що дозволяє уникнути необхідності розробки двох окремих застосунків для веб і мобільного додатку. Використовувана технологія PWA дозволить користувачам смартфонів встановлювати на нього, веб-орієнтований додаток у вигляді мобільного додатку [29]. Тому користувачі можуть працювати з програмою незалежно від їх девайсу.
- Sidekiq — сторонній застосунок для виконання запланованих задач [30]. Його задача — запускати на сервері необхідні команди у встановлений час.

Існує один зв’язок між цими застосунками, коли Sidekiq може виконувати команди, які виконуються за допомогою застосунку Progressive Web Application.

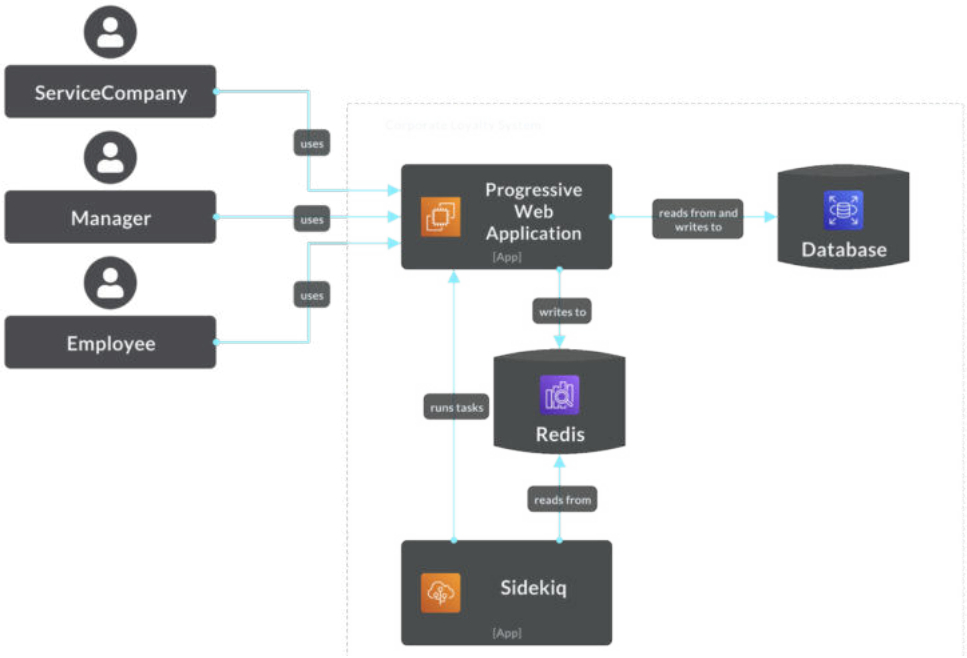


Рисунок 2.3 — Діаграма Containers 2 рівня C4 Model

Бази даних системи:

- Database — основна база даних, що містить усю інформацію про користувачів, їх ролі і доступи, зареєстровані організації та бізнеси, а також їхні знижки і спецпропозиції;
- Redis — мультиплатформна база даних швидкої пам’яті [31]. Містить інформацію про заплановані команди, які має виконати Sidekiq.

Прямого зв’язку між цими базами даних немає.

Зв’язки між застосунками і базами даних влаштовані так, що:

- Sidekiq зчитує Redis на наявність запланованих задач, перевіряє чи є ті, що треба виконати зараз і виконує такі команди. Виконані задачі Sidekiq стирає з бази Redis;
- Progressive Web Application записує у Redis інформацію про ті задачі, які треба виконати і коли;
- усі інші дані у системі Progressive Web Application зчитує з і записує в основну базу даних Database.

2.1.4 Рівень 3: Components

На третьому рівні C4 розкриваються компоненти застосунку та їх взаємодія. Додатково можуть бути зазначені зв’язки зі сторонніми системами, застосунками і базами даних.

Компонент — це набір класів або функцій, що працюють разом. Абстракція такого низького рівня є найбільш важливою для розуміння нюансів та взаємозалежності компонентів. На цьому рівні використовувані рішення та засоби зв'язку між кожним компонентом стають більш технічними.

Наш основний сервіс — це модульний моноліт. За такого сценарію компонентом може бути один модуль або набір цілісних класів.

Рисунок 2.4 — Діаграма Components 3 рівня C4 Model

Компоненти застосунку Progressive Web Application системи Corporate Loyalty System (рис. 2.4):

- Views — компоненти візуальної репрезентації статичних і динамічних даних застосунку:
  - Edit Organisation — форма перегляду і редагування даних організації;
  - View ServiceCompany — для перегляду знижок і контактної інформації бізнесу;
    - View Special Offers — для перегляду спеціальних пропозицій;

- Edit ServiceCompany — форма редагування знижки і контактної інформації про бізнес;
  - Edit Special Offers — форма редагування спеціальних пропозицій;
- Accept Invite — сторінка прийняття запрошення в систему;
- Sign In — форма авторизації;
- Edit Employee — форма перегляду і редагування переліку та даних користувачів ролі Employee;

- Controllers — алгоритми обробки і маніпуляції динамічних даних:
  - OgranizaionsController — відповідає за дані організації, такі як: назва, ідентифікатор, логотип;
  - ServiceCompanyController — відповідає за усі дані бізнесу, включаючи контактну інформацію, логотип, відсоток базової знижки тощо;
  - UsersController — відповідає за дані усіх користувачів;
  - ApplicationController — відповідає за обробку авторизації;
  - ResourcesController — відповідає за користувацьку конфігурацію;
  - InvitationsController — відповідає за обробку запрошень нових користувачів у систему;
  - ServiceWorkerController — відповідає за виконання некористувацьких запланованих команд.

Компоненти застосунку тісно пов’язані між собою і працюють за наступними принципами:

- будь-який користувач може отримати запрошення в систему, що відбувається за допомогою InvitationsController, Sidekiq, Redis, ServiceWorkerController і Email System. Далі користувач може підтвердити своє запрошення, що опрацьовується компонентів ApplicationsController, Accept Invite, а також увійти в систему, що опрацьовується Sign In View, ApplicationsController & UsersController;

- користувач ролі ServiceCompany може бачити і редагувати інформацію про свій бізнес за допомогою групи компонентів View ServiceCompany, Edit ServiceCompany, View Special Offers, Edit Special Offers, ServiceCompanyController;
- користувач ролі Manager може редагувати інформацію про свою організацію, що зачіпає Edit Organisation View і OrganisationsController. Також керувати користувачами Employee, що мають асоціацію до організації – це обробляють Edit Employee і UsersController.
- користувач ролі Employee може переглядати всі бізнеси, їх знижки і спецпропозиції, за допомогою View ServiceCompany, View SpecialOffers, ServiceCompanyController.

Майже у всіх випадках використовується база даних Database.

2.1.5 Рівень 4: Code

Четвертий рівень проєктування архітектури системи за схемою C4 Model є найбільш наближеним до вихідного коду програми. Код може бути класом або методом (або функцією).

Автори C4 не рекомендують [27] включати його в модель, особливо при старті проєкту. Такі конструкції дуже крихкі і з часом сильно змінюються.

В рамках цього рівня деталізації, спроєктовано схему бази даних і візуалізовано діаграмою типу «сутність-зв’язок» (див. Додаток Б).

Оскільки в програмі є пов’язані між собою точки, використаємо реляційну модель бази даних.

2.2 Бекенд

Бекенд – програмно-апаратна частина сайту, яка відповідає за логіку керування сайтом та за роботу з базою даних. Для створення цієї частини було обрано мову Ruby на фреймворку Ruby on Rails.

У 2020 році мова програмування Ruby посідає 14 сходинку у рейтингу найбільш популярних мов програмування серед професіоналів за версією вебсайту Stack Overflow (рис. 2.5) [32].

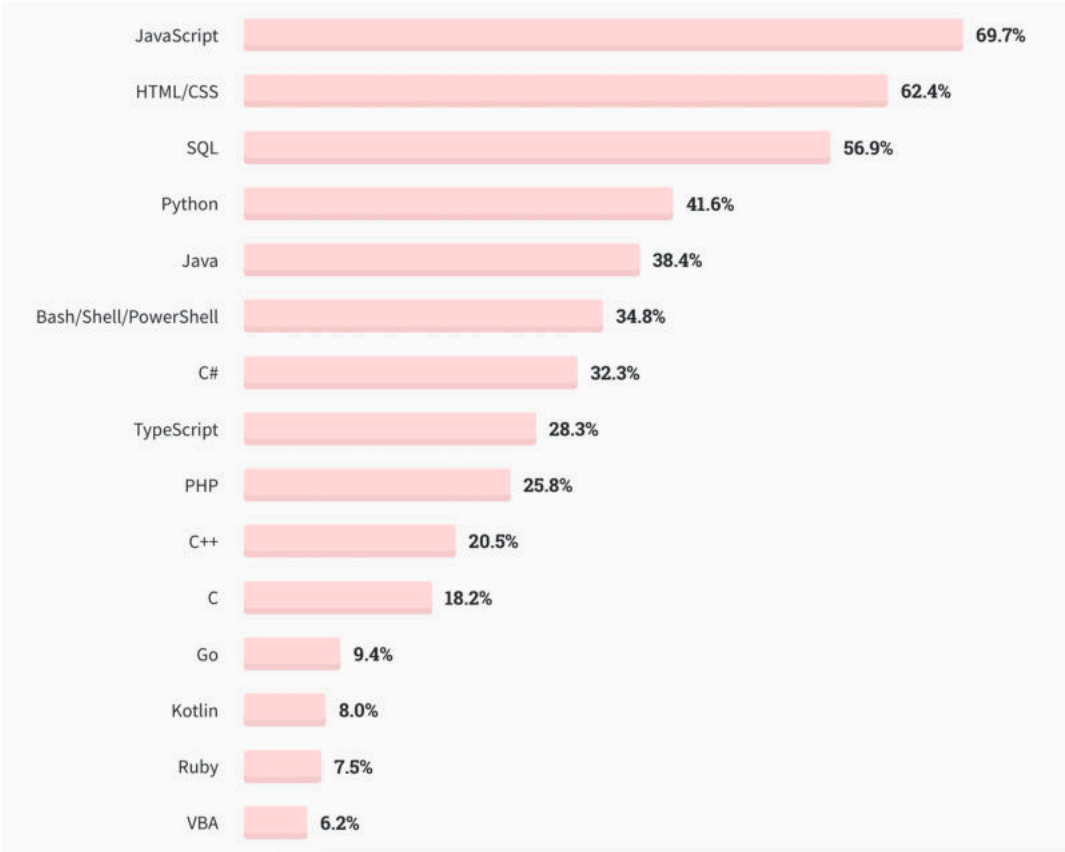


Рисунок 2.5 — Топ-15 найпопулярніших скриптових технологій 2020 року [32]

Ruby — одна з високорівневих мов програмування. В Ruby фактично немає процесу компіляції, тобто програма аналізується і виконується покомандно. Суттєвою особливістю відзначають відсутність множинного наслідування (як от є в мовах програмування C++, PHP, Python чи Delphi) [33].

Основні переваги:

- простий зрозумілий синтаксис;
- виключення стилю Java і Python;
- повністю об’єктно-орієнтована;
- автоматичний garbage collection для всіх об’єктів;
- цілі змінні автоматично конвертуються між типами, що дозволяє виконувати точні цілочисельні математичні обрахунки;

- змінні не вимагають попереднього оголошення;
- просте визначення області видимості;
- опція динамічного завантажування розширень;
- швидке розгортання програми.

Недоліки Ruby:

- некерованість низькорівневих структур даних і процесів;
- відсутність компіляції і, відповідно, оптимізації програми;
- відкрий коду готової програми.

Ruby — одна з найпопулярніших мов програмування серед веб-стартапів. Цю репутацію Ruby і його найпопулярніший фреймворк Ruby on Rails отримали через свою ефективність, масштабованість і економність [34].

2.3 Фронтенд

Фронтенд — клієнтська частина програми, що надає користувачу інтерфейс для її управління. Для його створення обрано фреймворк Bootstrap.

Bootstrap (рис. 2.6) — це сет опен сорс інструментів для створення вебсайтів. Він має шаблони CSS та HTML верстки типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також розширення JavaScript [35].

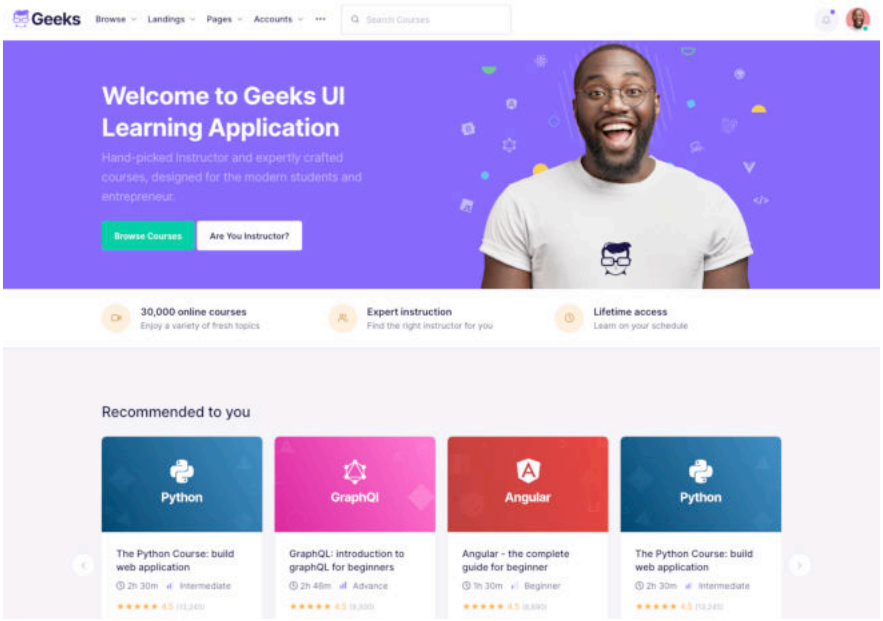


Рисунок 2.6 – Приклад реалізації сайту за допомогою Bootstrap

Основні інструменти Bootstrap:

- grid – сітка з 12 заданих колонок контейнера веб-сторінки;
- typography – типографічні набори різнорівневих заголовків, основного тексту, переліків тощо;
- table – оформлення таблиць, з додатковими можливостями з JS;
- form – оформлення форм і їх елементів;
- nav – оформлення панелей навігації, меню;
- alert – оформлення діалогових вікон, підказок і спливаючих вікон;
- icon font – набір іконок у вигляді шрифту.

2.4 Система управління базою даних

Нам потрібне масштабоване рішення для бази даних — планується розширяти додаток відповідно до користувацького запиту. Відповідно витрати на підняття додаткової бази мають бути мінімальними, а сервер має витримувати обробку великих об’ємів даних.

Rank			DBMS	Database Model	Score		
May 2021	Apr 2021	May 2020			May 2021	Apr 2021	May 2020
1.	1.	1.	Oracle	Relational, Multi-model	1269.94	-4.98	-75.50
2.	2.	2.	MySQL	Relational, Multi-model	1236.38	+15.69	-46.26
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	992.66	-15.30	-85.64
4.	4.	4.	PostgreSQL	Relational, Multi-model	559.25	+5.73	+44.45
5.	5.	5.	MongoDB	Document, Multi-model	481.01	+11.04	+42.02

Рисунок 2.7 — Топ-5 СУБД у 2021 році [36]

З найпопулярніших СУБД у світі [36] (рис. 2.7):

- Oracle — комерційна, реляційна, масштабована;
- MySQL — опен-сорс, реляційна, немасштабована;
- Microsoft SQL Server — комерційна, реляційна, масштабована;
- PostgreSQL — опен-сорс, реляційна, масштабована;

- MongoDB — опен-сорс, документо-орієнтована, масштабована.

Отже, оберемо для нашого застосунку СУБД PostgreSQL [37].

Сервер для PostgreSQL написаний мовою програмування C. Зазвичай розповсюджується у вигляді набору файлів з вихідним кодом. Для встановлення достатньо зкомпілювати файли та перенести програму в потрібний каталог.

Для Ruby існує розширення «pg» для роботи з PostgreSQL. Він забезпечує надійну, швидку та ефективну взаємодію.

2.5 Веб-сервер

Веб-сервер застосовується як для розгортання тестового середовища розробника програмного забезпечення, так і для управління хмарними сервісами, що пропонують гостинг програм.

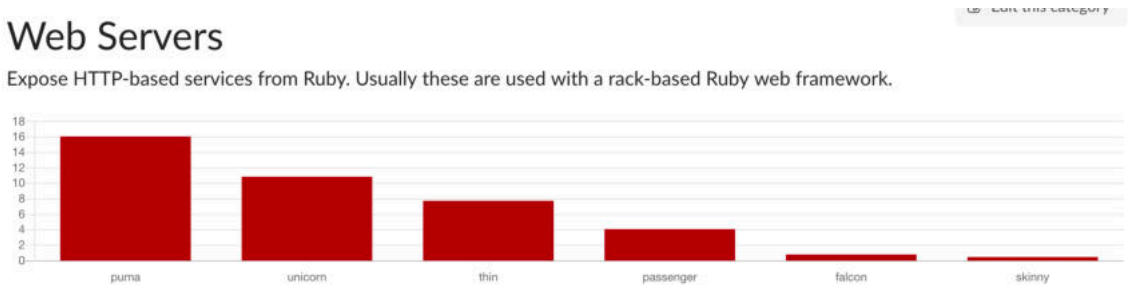


Рисунок 2.8 — Топ веб-серверів для Ruby

Оберемо найпопулярніший сервер (рис. 2.8) для Ruby програм — Puma [38]. Він працює через модульний інтерфейс Rack [39].

2.6 Інфраструктура

По-перше, необхідно використати систему для спрощення керування версіями і спільної роботи. Оберемо Git — надійну і найпопулярнішу [40]. І хостинг для коду GitHub [41], який її підтримує.

По-друге, оскільки вже було вирішено вище, що планується масштабувати програму відповідно до попиту на неї, перед інфраструктурою постають наступні задачі:

- тиражованість — мінімальний час на підняття копії інфраструктури, для тестового середовища, або відокремлення програми для специфічного ринку;
- перевикористання — готовність до підняття майже точної копії програми для розробки з іншими інтеграціями, компонентами тощо;
- портативність — інфраструктура має бути готовою до цілковитої передачі програми потенційному інвестору зі своєю командою розробки.

Існує кілька варіантів вирішення цих проблем:

- встановлюваний скрипт (напр., на Bash). Можна написати скрипт, який буде встановлювати все що треба на потрібні сервери. Недолік такого підходу — крихкість і нестійкість до помилок. Помилки трапляються регулярно, а отже є велика ймовірність що скрипт щось пошкодить. І цю дію не можна буде скасувати і повернути попередню версію.
- хмарні сервіси (напр., Amazon Machine Image). Можна налаштувати один сервер і зробити його образ, який можна копіювати скільки завгодно разів. Найбільший недолік — прив’язаність до сервісу, який це робить. Копію не можна буде підняти де-небуть поза сервісом, в якому було зроблено образ.
- віртуальні машини (напр., Bitnami VM). Налаштовується на будь-якій програмній машини і після цього може переноситись на будь-яку іншу. Головгі недоліки: за потреби будь-якої зміни, її треба зробити на всіх машинах окремо, а також фізичні машини мають ділити оперативну пам’ять з віртуальними машинами.
- контейнеризація (напр., Docker). Контрольований стек як у віртуальних машин, ефективне використання ресурсів сервера і можливість використовувати з будь-яким сервісом робять цей варіант найкращим.

Отже, в процесі розробки додатку використаємо контейнеризацію інфраструктури програми (рис. 2.9). Для цього оберемо одну з найпопулярніших платформ — Docker [42].

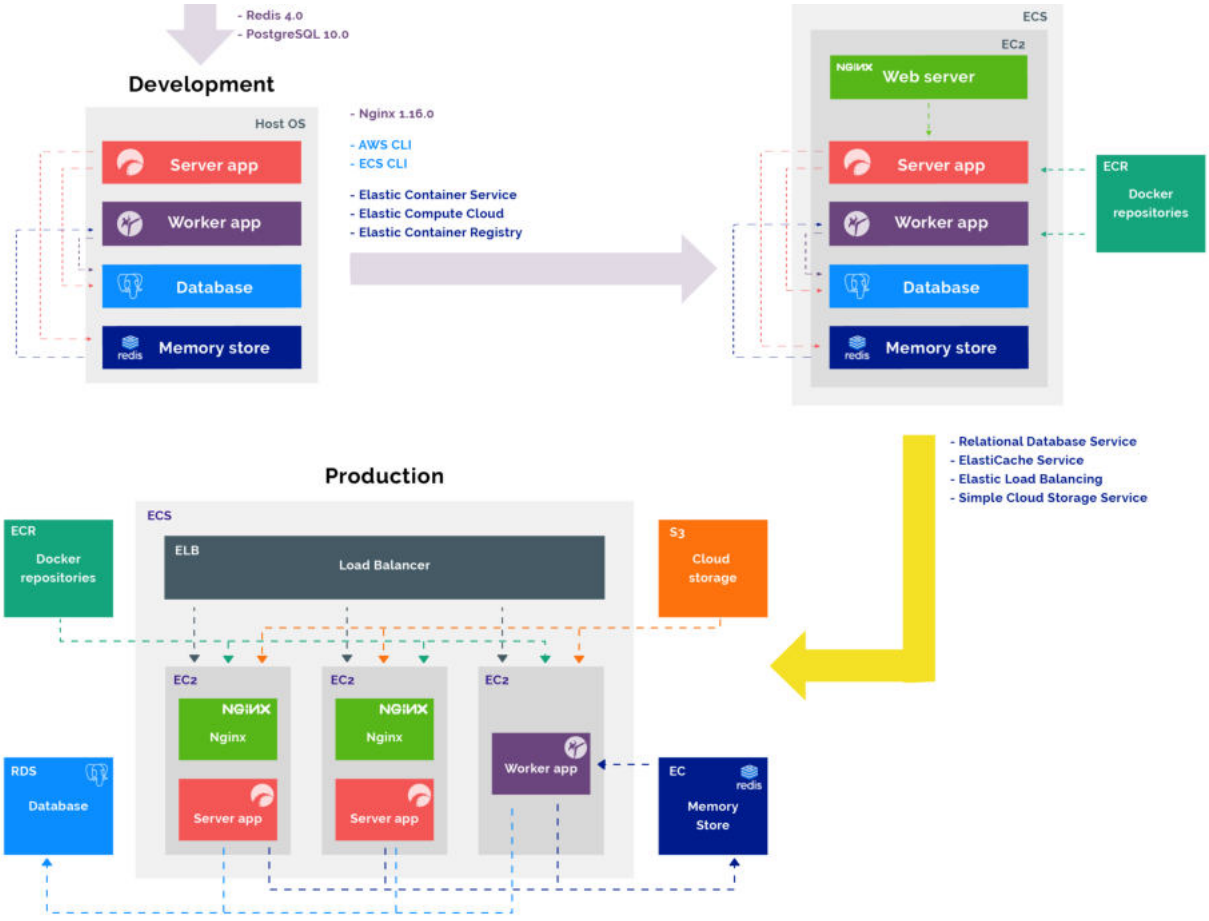


Рисунок 2.9 — Приклад організації інфраструктури з Docker і AWS

Висновки до розділу 2

Для поставленої задачі було обрано оптимальні технології – Ruby, Ruby on Rails, JavaScrips, Progressive Web Apps. Які з одного боку забезпечують високий рівень функціональності і відносну простоту запуску і подальшої підтримки та розвитку платформи.

Розроблено модель архітектури формату C4 Model, яка фіксує схему системи і за потреби дозволить прозоро передати технічну інформацію наступним розробникам системи (наприклад, у випадку її продажу).

Для інфраструктури обрано інструменти GitHub, Docker та Amazon Web Services, які забезпечать швидкодії програми, з мінімізацією фінансового навантаження для утримання самої інфраструктури.

РОЗДІЛ 3. РОЗРОБКА ПРОТОТИПУ

3.1 Розгортання середовища розробки

Використовувана операційна система на локальній машині: macOS Big Sur. Налаштуємо її для підтримки обраного для проєкту стеку технологій: мова програмування Ruby, її фреймворк Ruby on Rails і СУБД PostgreSQL.

3.1.1 Ruby

Ruby стандартно передвстановлена заводом-виробником на усі машини з операційною системою macOS. Тому на цьому кроці достатньо перевірити її наявність та версію за допомогою bash команди «`ruby -version`» в терміналі. Версія має бути вищою за 2.5.0, оскільки цього вимагає для коректної роботи фреймворк Ruby on Rails [43].

На локальній машині встановлено Ruby версії 2.6.3 (рис. 3.1).

```
MacBook-Pro:~ tymur$ ruby --version
ruby 2.6.3p62 (2019-04-16 revision 67580) [-darwin20]
```

Рисунок 3.1 — Перевірка встановленої версії Ruby

3.1.2 Ruby on Rails

Передумови до встановлення Ruby on Rails вимагають попередню наявність на машині Ruby, Node.JS і Yarn [43]. Необхідність пояснюється залежністю програм одна від іншої, коли RoR перевикористовує їх існуючі функції. Наприклад, Node.JS (як Runtime JavaScript) використовується для маніпуляцій з CoffeeScript, JavaScript і CSS [44]. Ruby підтримуваної версії на локальній машині вже є.

Пакет для встановлення Node.JS завантажуємо з офіційного вебсайту [45] і виконуємо. Це також можна було зробити за допомогою bash команди і програмою керування пакетами для macOS Homebrew: «`brew install node`».

Yarn також встановлюється за допомогою bash команди і програми керування пакетами NPM: «`npm install -g yarn`».

Тепер можемо перейти до встановлення безпосередньо фреймворку Ruby on Rails. Це відбувається простою bash командою «`gem install rails`», яка використовує менеджер пакетів RubyGems для мови програмування Ruby. Після успішного виконання команди, перевіряємо версію встановленого фреймворку (рис. 3.2) командою «`rails --version`».

Рисунок 3.2 — Перевірка встановленої версії Ruby on Rails

3.1.3 PostgreSQL

Для коректної роботи RoR з Postgres виконуємо наступні дії:

- виконуємо команду «`brew install postgresql`» для встановлення системи управління базою даних Postgres;
- перевіримо командою «`postgres --version`» (рис. 3.3);
- встановлюємо Ruby гем pg: «`gem install pg`»;
- перемикаємось на суперюзера Postgres «`su - postgres`» і запускаємо систему управління базою даних «`psql`»;
- створюємо користувача в Postgres для інтеграції з проєктом RoR: «`create role dizzcount with createdb login password 'dizz_buzz'`»;

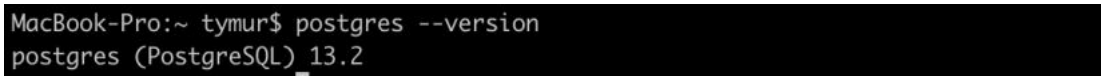


Рисунок 3.3 — Перевірка встановленої версії Postgres

3.1.4 Git

Система управління пакетами Git передвстановлена заводом-виробником на macOS. Перевіряємо версію «`git --version`».

```
MacBook-Pro:~ tymur$ git --version
git version 2.24.3 (Apple Git-128)
```

Рисунок 3.4 — Встановлена версія Git

3.1.5 Ініціація проєкту

Ініціюємо проєкт Ruby on Rails: «`rails new dizzcount --database=postgresql`» (рис. 3.5). Це створило теку «dizzcount», яка містить програму з такою ж назвою.

```
MacBook-Pro:github tymur$ rails new dizzcount --database=postgresql
      create
      create  README.md
      create  Rakefile
      create  .ruby-version
      create  config.ru
      create  .gitignore
      create  Gemfile
         run  git init from "."
Initialized empty Git repository in /Users/tymur/github/dizzcount/.git/
      create  package.json
```

Рисунок 3.5 — Ініціація проєкту RoR

Rails стандартно очікує що ім’я користувача СУБД відповідає назві програми і автоматично генерує файл з конфігураціями (рис. 3.6) для роботи з базою даних «database.yml», розташований у теці «config». Відповідно залишається виконати команду «`rake db:setup`» в теміналі, що сворить бази даних для розробки і тестування, встановить вказаного користувача їх власником і створить «schema_migrations» таблиці в кожній. Ці таблиці використовуються для реєстрації міграцій схем і даних.

database.yml

```
default: &default
  adapter: postgresql
  encoding: unicode
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>

development:
  <<: *default
  database: dizzcount_development
  password: dizz_buzz
  #host: localhost

test:
  <<: *default
  database: dizzcount_test

production:
  <<: *default
  database: dizzcount_production
  username: dizzcount
  password: <%= ENV['DIZZCOUNT_DATABASE_PASSWORD'] %>
```

Рисунок 3.6 — Файл конфігурації для роботи з базою даних

Робимо пробний запуск Ruby on Rails проєкту на сервері Puma. Відповідає команда «`rails s`». Програма успішно запустилась (рис. 3.7) і її веб інтерфейс доступний за HTTP адресою http://127.0.0.1:3000/ (рис. 3.8).

```
MacBook-Pro:dizzcount tymur$ rails s
=> Booting Puma
=> Rails 6.0.3.7 application starting in development
=> Run `rails server --help` for more startup options
Puma starting in single mode...
* Version 4.3.8 (ruby 2.6.3-p62), codename: Mysterious Traveller
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://127.0.0.1:3000
* Listening on tcp://[::1]:3000
Use Ctrl-C to stop
```

Рисунок 3.7 — Успішний запуск програми на сервері

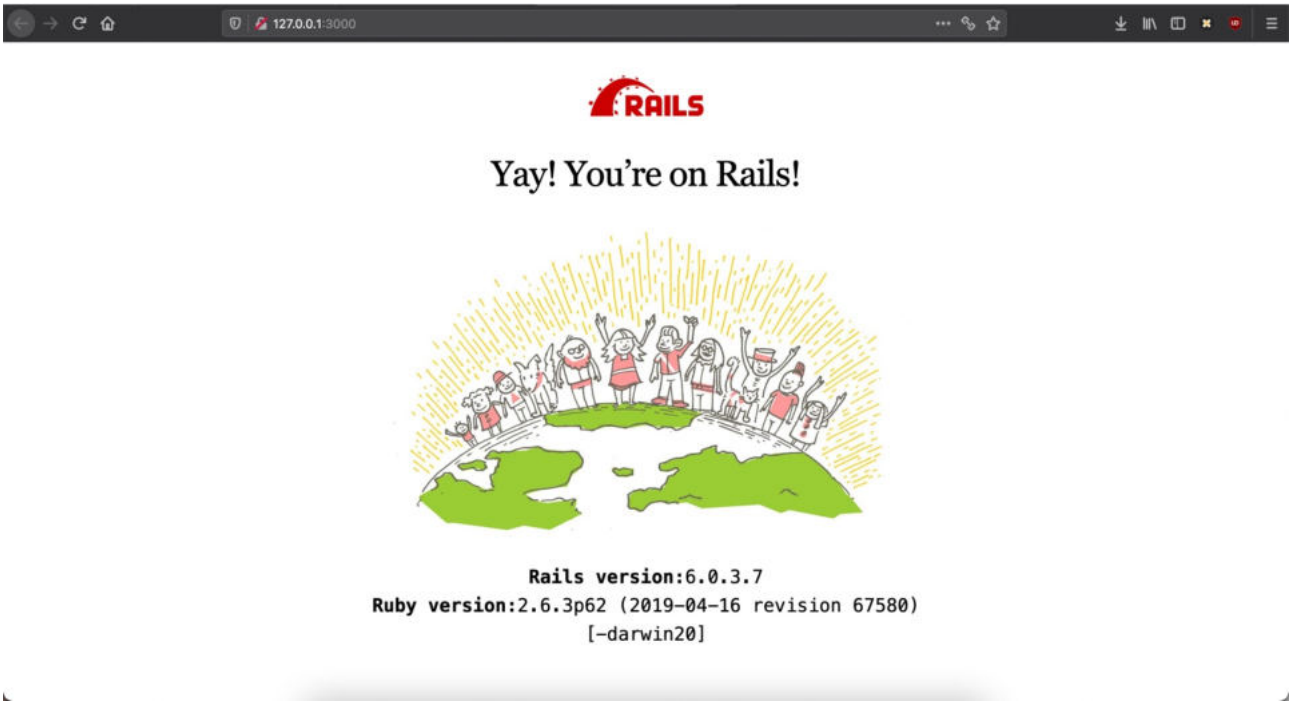


Рисунок 3.8 — Успішний запуск стандартного smoke-тесту працездатності запущеної програми Ruby on Rails

3.2 Розробка інтерфейсу авторизації

Перша точка дотику користувача з системою — це авторизація. Адже лише після неї користувач будь-якої ролі може отримати доступ до персоналізованих функцій і даних в системі.

Для реалізації функціоналу авторизації виконаємо наступні дії:

- прописуємо рядок «`gem 'devise', '~> 4.7'`» в Gemfile. Devise – це Ruby гем, який вже має всі необхідні інструменти для авторизації;
- прописуємо рядки «`gem 'pundit'`» і «`gem 'passpartu'`» в Gemfile. Вони допоможуть розподілити права доступу різних рівнів користувачів до різного функціоналу, залежно від їх ролей;
- прописуємо рядок «`gem 'mailjet'`» в Gemfile. Гем «mailjet» інтегрує програму з MailJet — сторонньою системою для надсилання електронних листів. Вони знадобляться пізніше для відправки листів-запрошень у систему.

- прописуємо рядки «`gem 'sidekiq'`» і «`gem 'redis'`» в Gemfile. Ці два геми знадобляться для запланування задач (наприклад, те ж надсилання системних електронних листів);
- виконуємо команду в терміналі для встановлення гемів на локальну машину «`bundle install`»;
- далі наступну, що встановлює взаємодію гему Devise з RoR програмою: «`rails generate devise:install`».

Пропишемо модель User, попередньо згенеровану в процесі встановлення гему Devise (програмний код зазначено в Додатку А до дипломної роботи). Основні елементи:

- кожен об’єкт обов’язково матиме одну з ролей:
  - admin — власник системи, керує доступом до неї;
  - organization_manager — менеджер компанії, яка купила підписку на систему. Може керувати доступом співробітників до системи;
  - organization_member — працівник компанії, має доступ до знижок і пропозицій бізнесів;
  - service_company_coordinator — представник (координатор) бізнесу, який встановлює рівень знижки і публікує спеціальні пропозиції від бізнесу;
- необов’язкові асоціації з:
  - Organization (компанія) — для менеджерів і працівників компанії;
  - ServiceCompany (бізнес) — для представників бізнесу;
- статус (активний або неактивний);
- методи:
  - Стандартні методи Devise для менеджменту авторизації;
  - Перевірка користувача за ідентифікатором або імейлом (використовуватиметься координатором бізнесу для валідації клієнта – працівника компанії).

Конфігурація рівнів доступу кожної з ролей користувача до можливостей в системі відбуватиметься протягом подальшої розробки.

Видаляємо сторінку, посилання і контролер публічної реєстрації, оновлюємо стилі. Змінюємо форми входу app/views/devise/sessions/new.html.slim і відновлення паролю за допомогою фронтенд елементів Bootstrap (рис. 3.9).

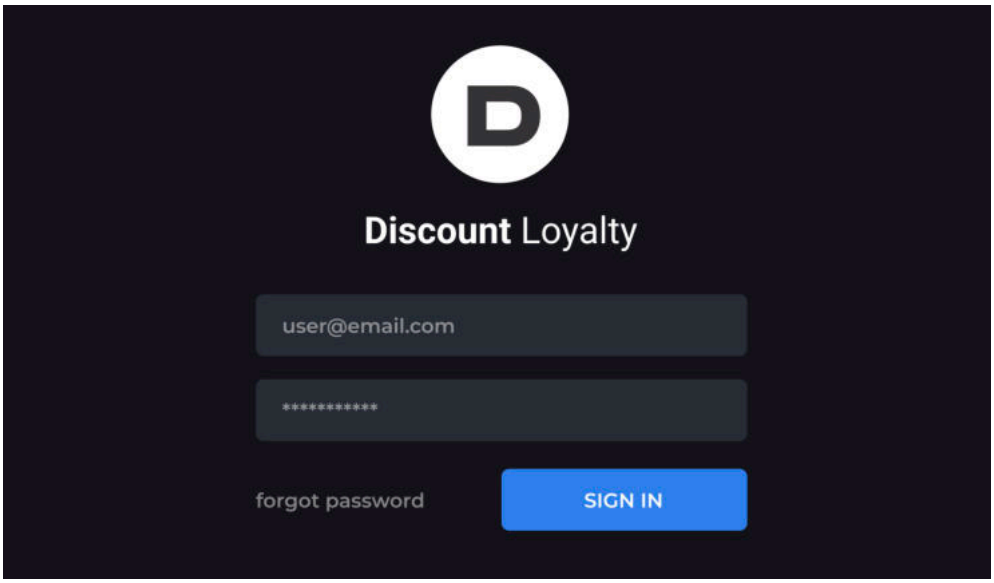


Рисунок 3.9 — Скріншот форми входу

3.3 Розробка інтерфейсу адміністратора

Оскільки в системі існуватимуть сутності, якими треба буде керувати, ці обов’язки будуть покладені на роль адміністратора.

Спершу створимо сутності Organization (компанія) та ServiceCompany (бізнес). Organization матиме багато асоційованих користувачів і деякі з них зможуть керувати даною сутністю. ServiceCompany матиме єдиного асоційованого користувача, який же і буде нею керувати.

Опишемо контролер для керування адмінітратором сутності Organization. Основні методи:

- organization_invitation — для створення нового об’єкту Organization;
- invite_organization — для запрошення компанії у систему;
- update — для оновлення даних про компанію;
- index — для повернення переліку компаній на необхідну сторінку.

Створимо сторінку для відображення об’єктів Organisation (рис. 3.10):

```
      .organizations__list
        = render @records
      .organizations__container_nav
        = render partial: 'pagy/bootstrap_nav', locals: {pagy: @pagy} if
@pagy.pages > 1
        = link_to 'Add', organization_invitation_admin_organizations_path, class:
"submit_btn"
```

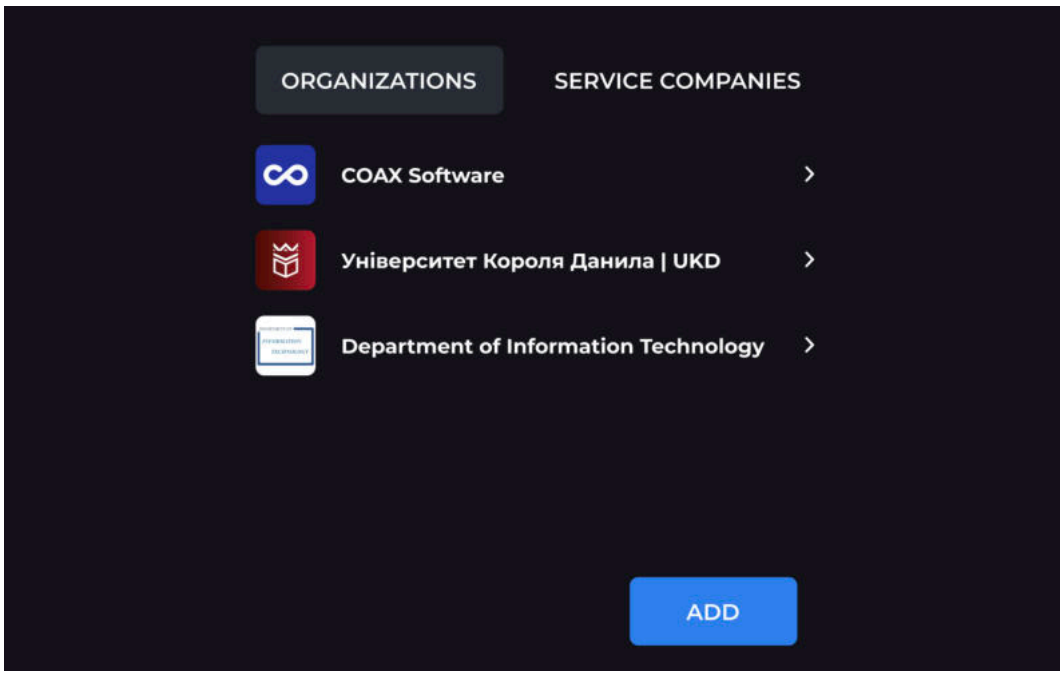


Рисунок 3.10 — Скріншот сторінки адміністрування об’єктів Organisation

А також сторінку їх створення і редагування (див. Додаток Г).

Сторінка включає такі елементи (рис. 3.11):

- аватар — квадратна картинка логотипу компанії. Відображається на картках ідентифікаторів працівників;
- назва — коротка офіційна назва компанії;
- статус — активний / неактивний;
- кнопки:
  - збереження;
  - видалення;
  - перехід на сторінку керування користувачами компанії.

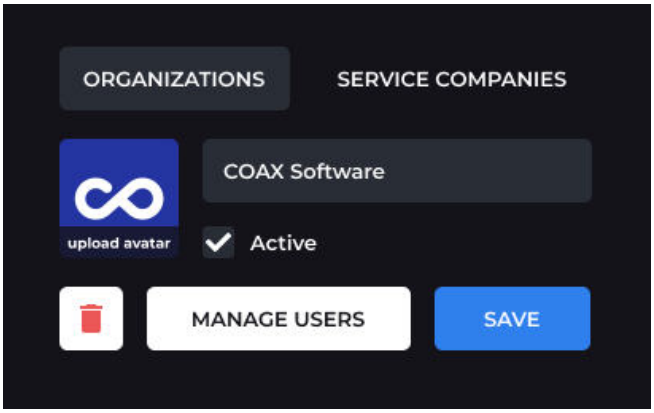


Рисунок 3.11 — Скріншот сторінки керування Organisation

ServiceCompany має ідентичні до Organization контролер та сторінку перегляду списку. Відмінною є сторінка створення (запрошення) та редагування (див. Додаток Ґ).

Набір елементів на даній сторінці наступний (рис. 3.12):

- аватар — квадратна картинка логотипу бізнесу. Відображається в переліку і профілі бізнесу, а також як стандартна картинка спеціальних пропозицій (якщо не зазначено інше);
- назва — офіційна коротка назва бізнесу;
- статус — активний / неактивний;
- email — єдиний ідентифікатор для авторизації в систему, а також контакту з клієнтами;
- телефон — структура введення вільна, номерів може бути декілька;
- місто — одне або декілька міст Україні, в яких бізнес має своє фізичне представництво (офіс);
- вебсайт — одна адреса сайту, поле необов’язкове;
- кнопки:
  - збереження;
  - видалення.

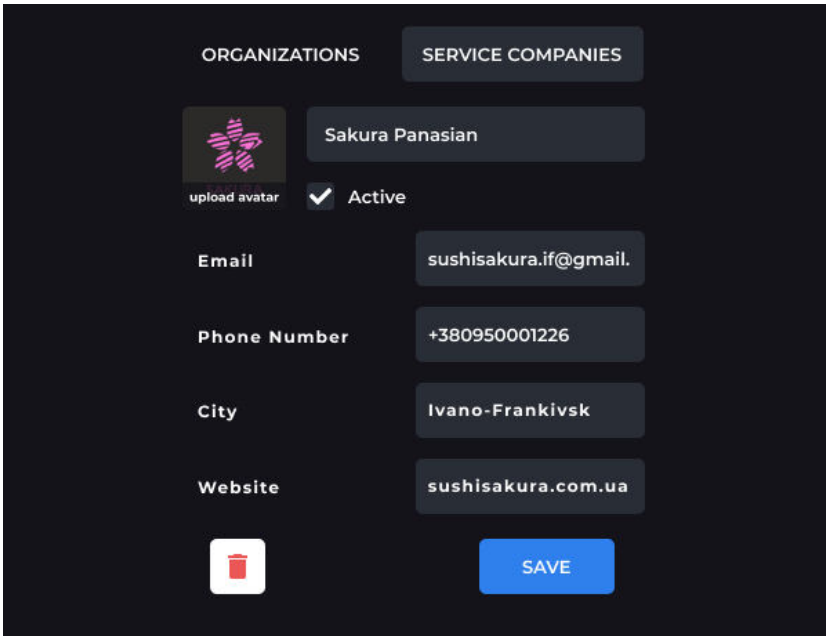


Рисунок 3.12 — Скріншот сторінки керування ServiceCompany

Сторінка керування Organization має кнопку переходу на управління користувачами однієї Organization. Для реалізації цього функціоналу, додамо форму запрошення і редагування користувачів (див. Додаток Д).

Форма організована у вигляді картки, візуально схожої до банківської і включає у себе (рис. 3.13):

- аватар — реальне якісне фото працівника компанії, що дозволить координатору бізнесу перевірити доступ клієнта до програми лояльності;
- email — адреса електронної пошти працівника для авторизації;
- чекбокс присвоєння ролі Organization Manager — надає користувачу додаткові права керування асоційованою організацією;
- код ідентифікатора — автоматично генерується системою, є одним зі способів валідації клієнта представником бізнесу;
- кнопки:
  - збереження;
  - видалення;
  - повернення до сторінки управління компанією.

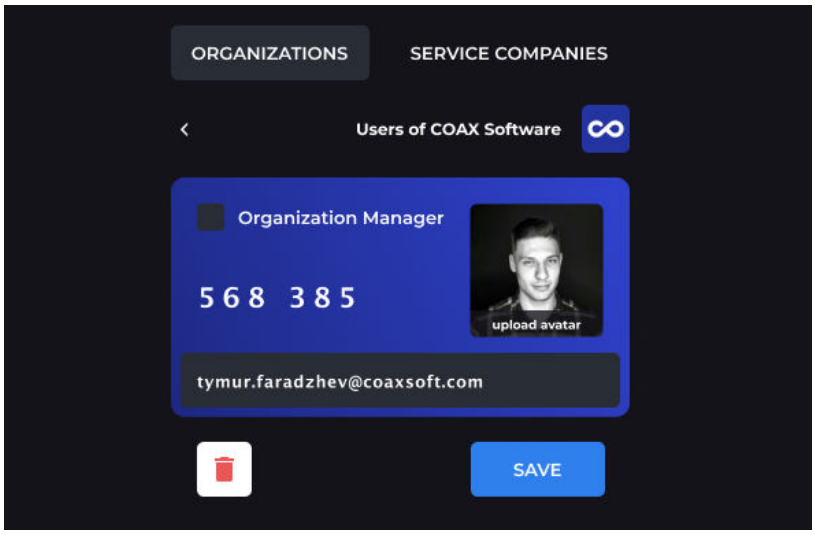


Рисунок 3.13 — Скріншот сторінки керування користувачем

Оскільки ми хочемо надсилати користувачам електронне повідомлення при додаванні їх у систему, доповнимо для відповідного функціоналу наступними рядками програмного коду контролер сутності User:

```
  def invite_params
    record_params.merge!(organization_id: params[:organization_id], service_company_id: params[:service_company_id])
  end
```

Також опишемо розподілення різних роутів між ролями користувачів асоційованих до компаній та бізнесів:

```
  def choose_redirect
    if record.organization_id
      admin_organization_users_path(record.organization_id)
    else
      admin_service_company_users_path(record.service_company_id)
    end
  end
end
```

І створимо окремий простий контролер для обробки запрошень користувачів електронною поштою:

```
class Users::InvitationsController < Devise::InvitationsController
  private

  def update_resource_params
    params.require(:user).permit(:password, :password_confirmation, :invitation_token, :first_name, :avatar)
```

```
  end
end
```

Перевизначимо спрямування на цей контролер необхідні роути в config/routes.rb:

```
devise_for :users, controllers: { invitations: 'users/invitations' }
```

Підготуємо шаблон для електронних повідомлень про запрошення користувача до системи (рис. 3.14):

```
p= t("devise.mailer.invitation_instructions.hello", email: @resource.email)
p= t("devise.mailer.invitation_instructions.someone_invited_you", url: root_url)
p= link_to t("devise.mailer.invitation_instructions.accept"), accept_invitation_url(@resource, invitation_token: @token)
- if @resource.invitation_due_at
  p= t("devise.mailer.invitation_instructions.accept_until", due_date: l(@resource.invitation_due_at, format: :'devise.mailer.invitation_instructions.accept_until_format'))
p= t("devise.mailer.invitation_instructions.ignore")
```

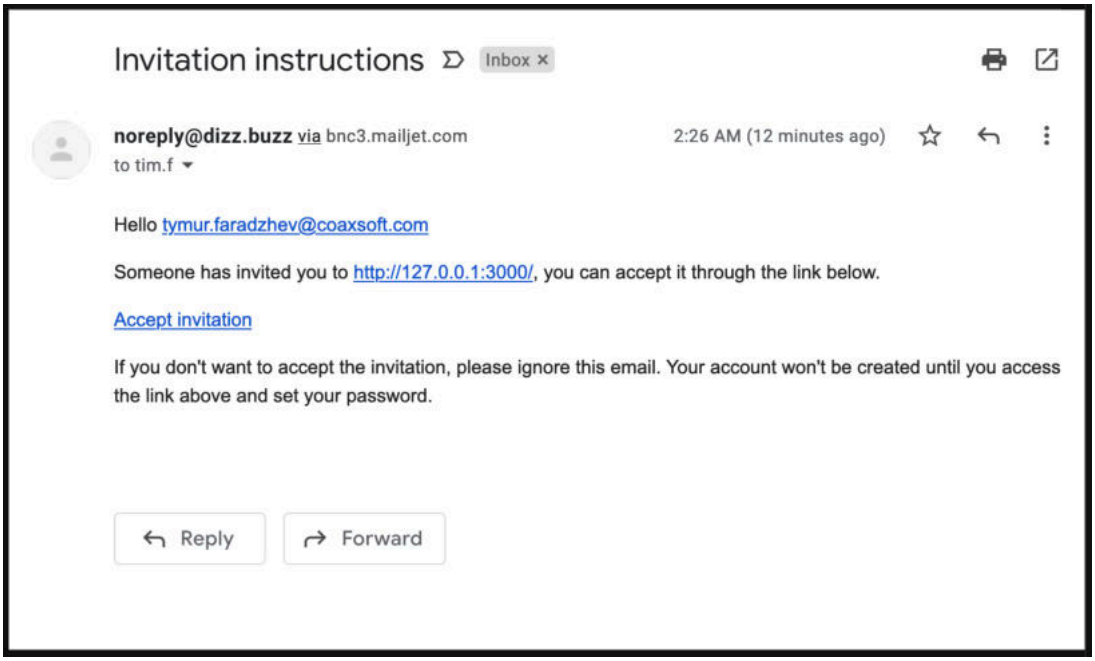


Рисунок 3.14 — Надісланий електронний лист із запрошенням

3.4 Розробка інтерфейсу керування компанією

Після того, як адміністратор надіслав запрошення на електронну адресу новоствореній в системі Organisation (компанії), цей користувач може

підтвердити запрошення і авторизуватись у системі для подальшого керування профілем і користувачами, асоційованими до цієї компанії.

Користувач ролі Organisation може використовувати ті ж самі контролери і сторінки для керування компанією, що і адміністратор. Відповідно до цього, зробимо новий файл для керування правами доступу config/passpartu.yml, і внесемо в нього записи про права адміністратора і компанії:

```
admin:
  organization:
    index: true
    invite_organization: true
    show: true
    edit: true
    update: true
    invite_member: true
  user:
    index: true
    show: true
    create: true
    update: true
    switch_active_status: true
    destroy: true
  service_company:
    index: true
    invite_service_company: true
    my_service_company: true
    info: true
    edit: true
    update: true
organization_manager:
  organization:
    show: true
    edit: true
    my_organization: true
    update: true
    invite_member: true
  user:
    show: true
    create: false
    update: false
    switch_active_status: true
    destroy: false
```

Гем Passpartu дозволяє впускати значення false у налаштуванні параметрів прав доступу у файлі його конфігурацій. Адже він працює за принципом: доступ надається лише до елементів зі значенням true. Це дозволяє суттєво спростити управління правами та зменшити об’єм файлу конфігурацій.

Отже, ми надали права користувачу-менеджеру Organisation на управління власним профілем та асоційованими користувачами.

3.5 Розробка інтерфейсу керування бізнесом, його знижками і спецпослугами

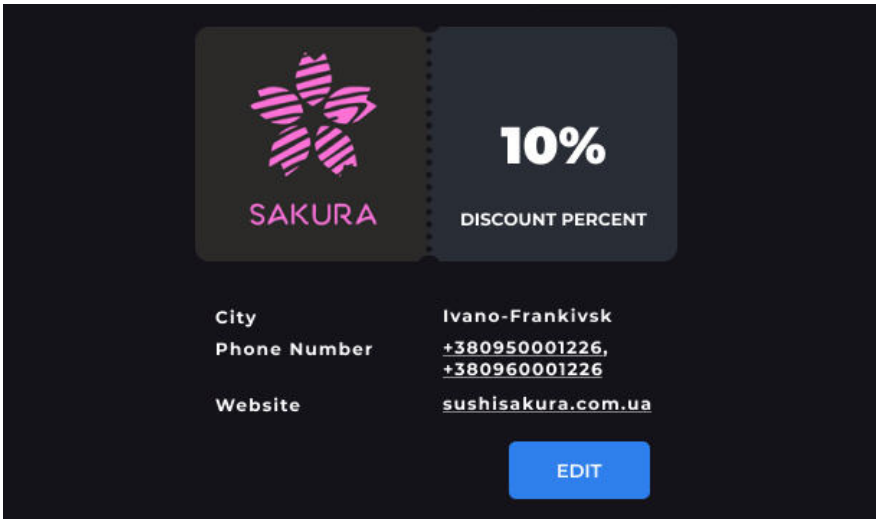


Рисунок 3.15 — Перегляд профілю бізнесу

Аналогічно до Organisation, ServiceCompany має право переглядати та редагувати власну інформацію (рис. 3.15). Тому додамо для цього додатковий запис до config/passpartu.yml:

```
service_company_coordinator:
  service_company:
    index: true
    invite_service_company: false
    my_service_company: true
    info: true
    verify_user: true
    edit: true
    update: true
```

Тепер, розробимо сутність спеціальної пропозиції — SpecialOffer. Редагувати її зможе лише ServiceCompany, а переглядати — лише Organisation (Manager і Member).

Створюємо модель і контролер SpecialOffer (див. Додаток Ж). Сутність містить в собі:

- опис — заголовок з чітким описом умов спеціальної пропозиції, до ста двадцяти символів;
- картинку — необов'язковий елемент, якщо не встановлено – стандартно заповнюється аватаром бізнесу;
- тривалість — період часу скільки дана спеціальна пропозиція є активною, відповідно до чого буде відображатись для кінцевих користувачів;
- асоціацію до об'єкту ServiceCompany.

Будуємо сторінку з формою створення і редагування об'єктів SpecialOffer (код. app/views/sc/special_offers/edit.html.slim міститься у Додатку А).

І завершуємо складанням сторінки переліку спеціальних пропозицій, посекційно розбитих на актуальні та ті, що наближаються (рис. 3.16):

```
= render @special_offers.current
- if @special_offers.upcoming.present?
  span.offer__soon_text soon
  = render @special_offers.upcoming
.offer__btn_container
  = link_to 'Add', new_special_offer_path, class: 'submit_btn'
```

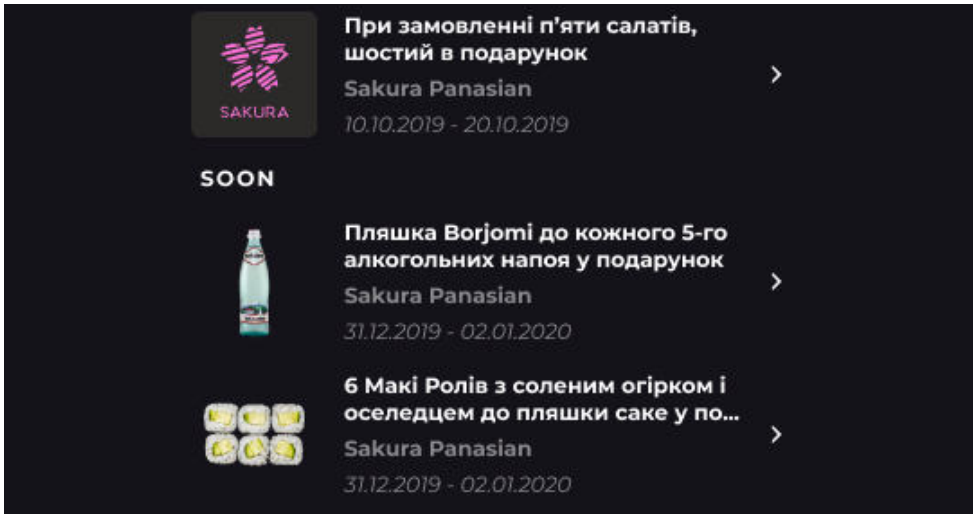


Рисунок 3.16 — Перелік спеціальних пропозицій

Додаємо ServiceCompany права перегляду і редагування об’єктів SpecialOffer через Passpartu:

```
special_offer:
  index: true
  my_special_offers: true
  show: true
  create: true
  new: true
  update: true
  edit: true
  destroy: true
```

3.6 Розробка інтерфейсу використання знижок і спецпропозицій

Адміністратор запросив у систему користувача ServiceCompany, він додав свою контактну інформацію, зазначив свої знижки та наповнив систему спеціальними пропозиціями. Користувачі з асоціацією до Organisation теж отримали запрошення від адміністратора або менеджера компанії і вже готові приступити до використання функціоналу знижок і спецпропозицій.

Спершу, дамо асоційованим до Organisation користувачам роль organization_member і присвоїмо можливість переглядати бізнеси (ServiceCompany) і їх спеціальні пропозиції (SpecialOffer):

```
organization_member:
  service_company:
    index: true
    show: true
  special_offer:
    index: true
    show: true
```

Оскільки цей тип користувачів орієнтується на використання програми через смартфони, підготуємо адаптивне навігаційне меню (код зазначено в Додатку А), що більше схоже на меню нативного мобільного додатку (рис. 3.17).

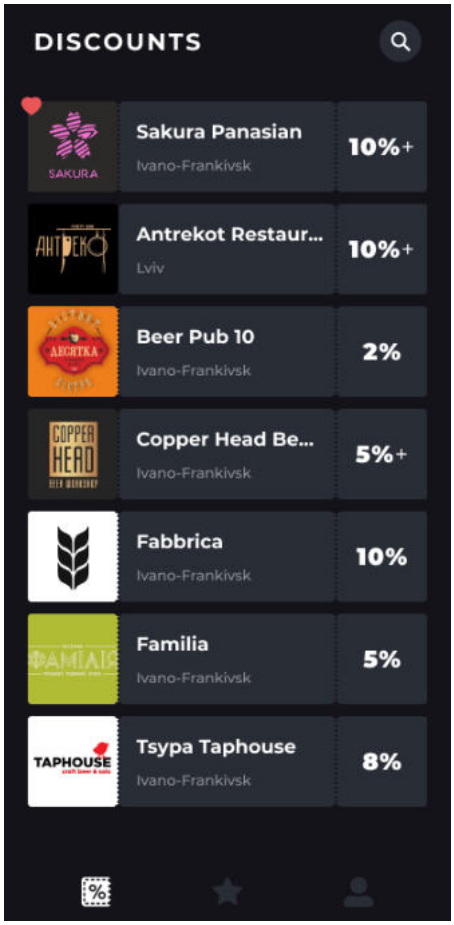


Рисунок 3.17 — Перелік бізнесів і їх базових знижок

Тепер виведимо перелік усіх об’єктів ServiceCompany (див. Додаток Ї). Сторінка містить перелік активних бізнесів і інформацію про кожного (рис. 3.17):

- логотип бізнесу;
- назва бізнесу;
- місто або міста;
- відсоток базової знижки;
- активність спеціальних пропозицій — при їх наявності вказується символ «+» біля відсотку знижки.

Додамо сторінку перегляду одного бізнесу, його інформації та асоційованих спеціальних пропозицій (див. Додаток Й). Сторінка кожного бізнесу має такі елементи (рис. 3.18):

- картка працівника з ідентифікатором;
- логотип бізнесу;

- відсоток базової знижки;
- контактна інформація:
  - місто (або міта);
  - імейл;
  - номер(и) телефону;
- перелік спеціальних пропозицій.

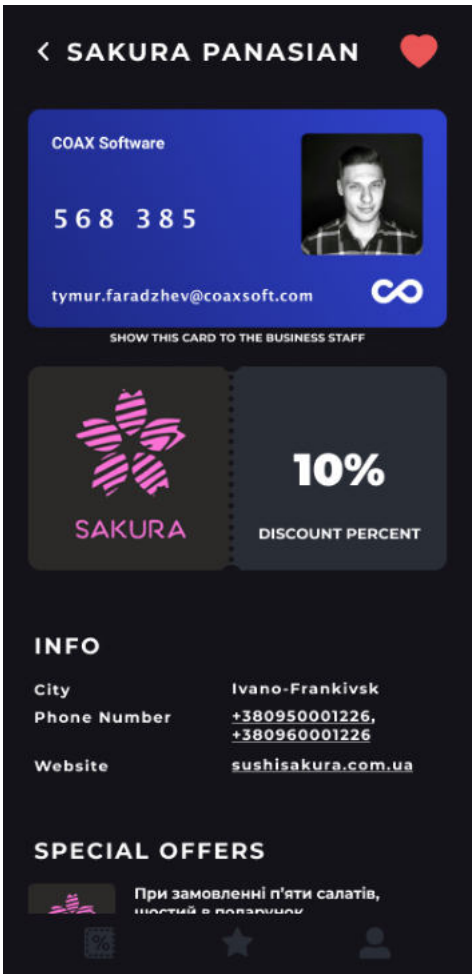


Рисунок 3.18 — Вигляд сторінки бізнесу для працівників компанії

Додатково, для працівника компанії створена сторінка з переліком усіх спеціальних пропозицій усіх бізнесів (рис. 3.19):

```
  spam.title.discount-js special offers
= render @special_offers.current
- if @special_offers.upcoming.present?
  .title soon
  = render @special_offers.upcoming
```

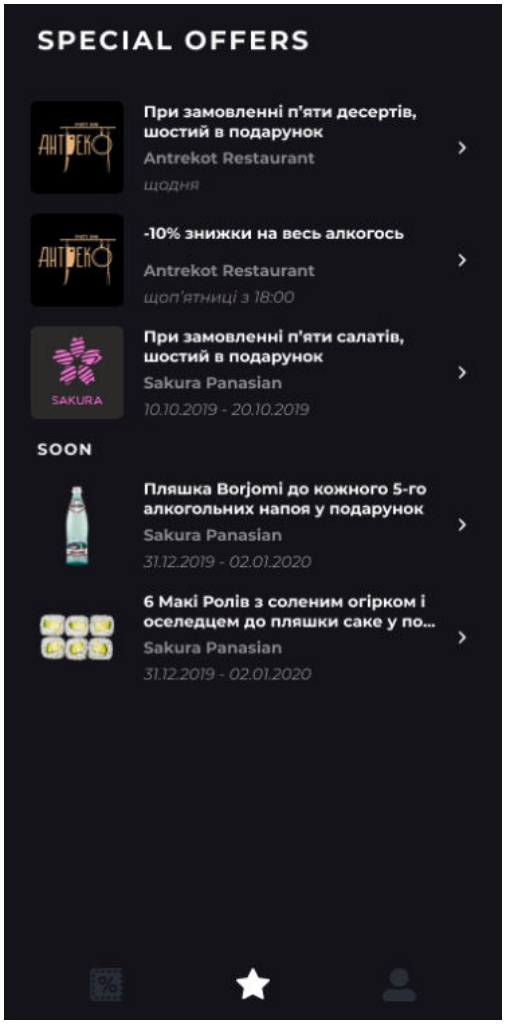


Рисунок 3.19 — Перелік усіх спецпропозицій

Врешті, коли працівник компанії переглянув доступні пропозиції і обрав ту, якою хоче скористатись, для її отримання він має показати її представникові бізнесу. Для того, аби представник бізнесу міг підтвердити справжність і актуальність доступності цієї людини до знижки і спеціальних пропозицій, клієнт має особистий ідентифікатор всередині системи.

Цей ідентифікатор не розкриває персональної інформації про працівника компанії (він же є клієнтом для бізнесу), але дає можливість бізнесу перевірити віртуальну картку лояльності клієнта.

З одного боку, клієнт має спеціальні місця у програмі зі своєю карточкою з цілочисельним ідентифікатором (рис. 3.18 та 3.20). Програмний код вигляду картки на будь-якій з сторінок програми включено у Додаток К.

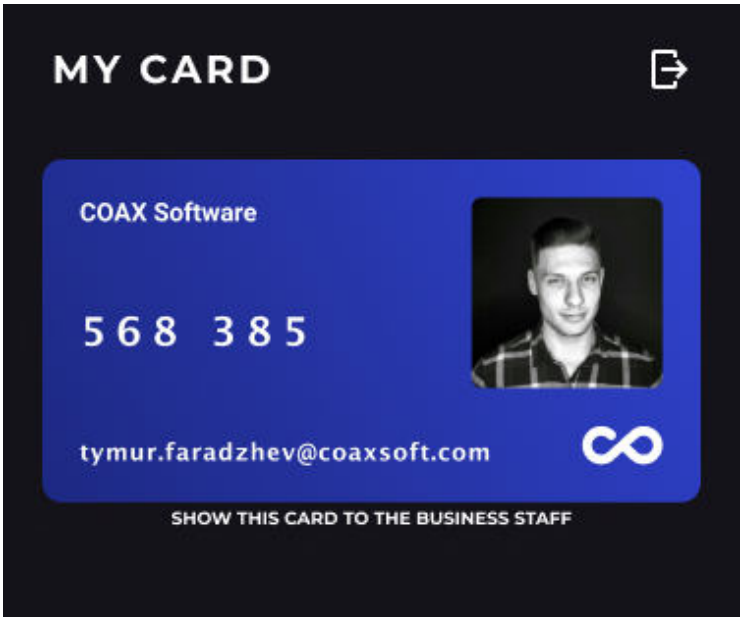


Рисунок 3.20 — Картка працівника компанії з особистим ідентифікатором

З іншого боку, представник бізнесу, що має доступ до акаунту ServiceCompany, має сторінку з формою для перевірки своїх клієнтів двома способами: цілочисельним ідентифікатором, або адресою електронної пошти (рис. 3.21, 3.22, 3.23). Код форми перевірки влючено у Додаток Л.

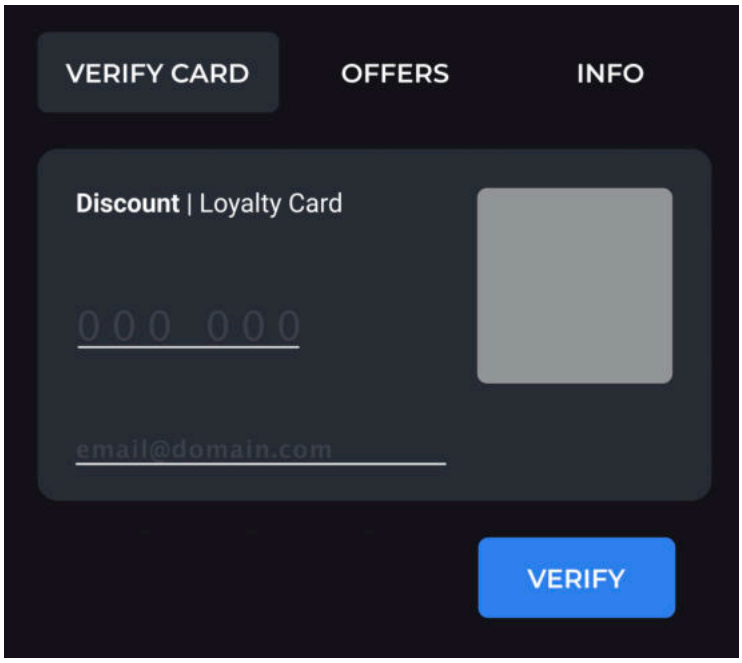


Рисунок 3.21 — Форма перевірки ідентифікатора клієнта

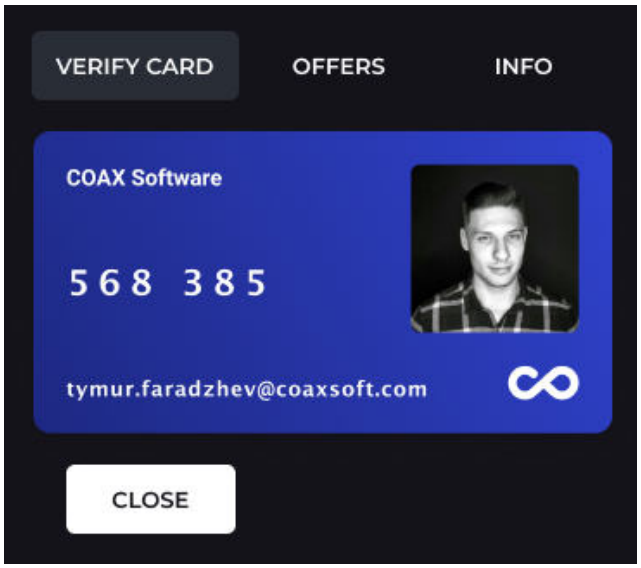


Рисунок 3.22 — Результат успішної валідації клієнта

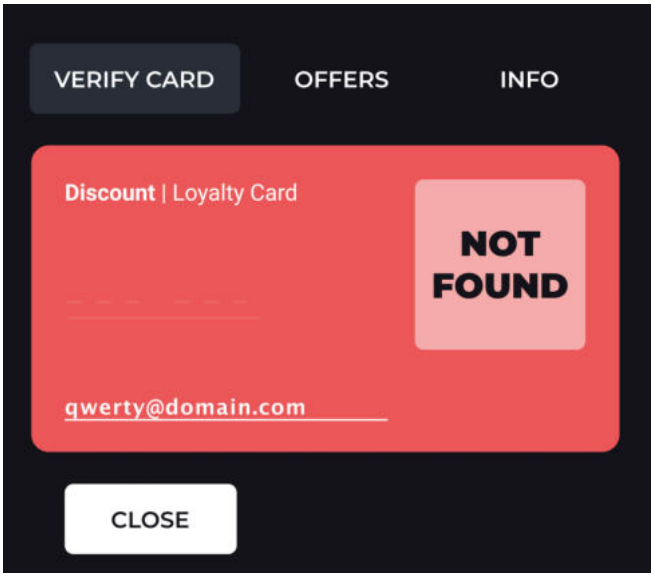


Рисунок 3.23 — Помилка валідації клієнта

3.7 Впровадження технології Progressive Web Application

Одна з нефункціональних вимог до програми — можливість роботи на смартфонах з найбільш близьким до нативного додатку функціоналом. В рамках цієї норми, інтегровано технологію Progressive Web Application [29].

Визначено «manifest.json» файл, в який входять:

- назва програми;
- коротка назва програми;

- коренева тека;
- набір картинок для всіх операційних систем смартфонів;
- колір теми;
- колір фону;
- тип відображення;
- орієнтація екрану.

Покликання на цей файл вбудовано у HTML каркас усіх сторінок:

```
link href="/manifest.json" rel="manifest"
```

Оскільки Webpacker — один із сервісів Ruby on Rails, не повністю підтримує технологію PWA, треба встановити додатковий NPM пакет, що вирішує цю проблему «webpacker-pwa». Виконуємо bash команду «`yarn add webpacker-pwa`», додаємо рядок «`service_workers_entry_path: service_workers`» у файл «webpacker.json» і змінюємо файл «config/webpack/environment.js» в RoR проєкті наступним чином:

```
const { resolve } = require('path');
const { config, environment, Environment } = require('@rails/webpacker');
const WebpackerPwa = require('webpacker-pwa');
new WebpackerPwa(config, environment);
module.exports = environment;
```

Далі, реєструємо ServiceWorkers — створюємо у теці «app/javascript/service_workers» файл «service-worker.js» з кодом:

```
if (navigator.serviceWorker) {
  navigator.serviceWorker.register('/serviceworker.js', { scope: '/pwa_o/'
})
    .then(function(reg) {
      console.log('[Companion]', 'Service worker registered!');
    });
}
```

І врешті реєструємо його в «application.js», що буде виконуватись браузером при завантаженні програми на клієнтській стороні:

```
navigator.serviceWorker.register('/service-worker.js').then(registration
=> {
    console.log('ServiceWorker registered: ', registration);
    var serviceWorker;
    if (registration.installing) {
      serviceWorker = registration.installing;
```

```
      console.log('Service worker installing.');
    } else if (registration.waiting) {
      serviceWorker = registration.waiting;
      console.log('Service worker installed & waiting.');
    } else if (registration.active) {
      serviceWorker = registration.active;
      console.log('Service worker active.');
    }
  }).catch(registrationError => {
    console.log('Service worker registration failed: ', registrationError);
  });
```

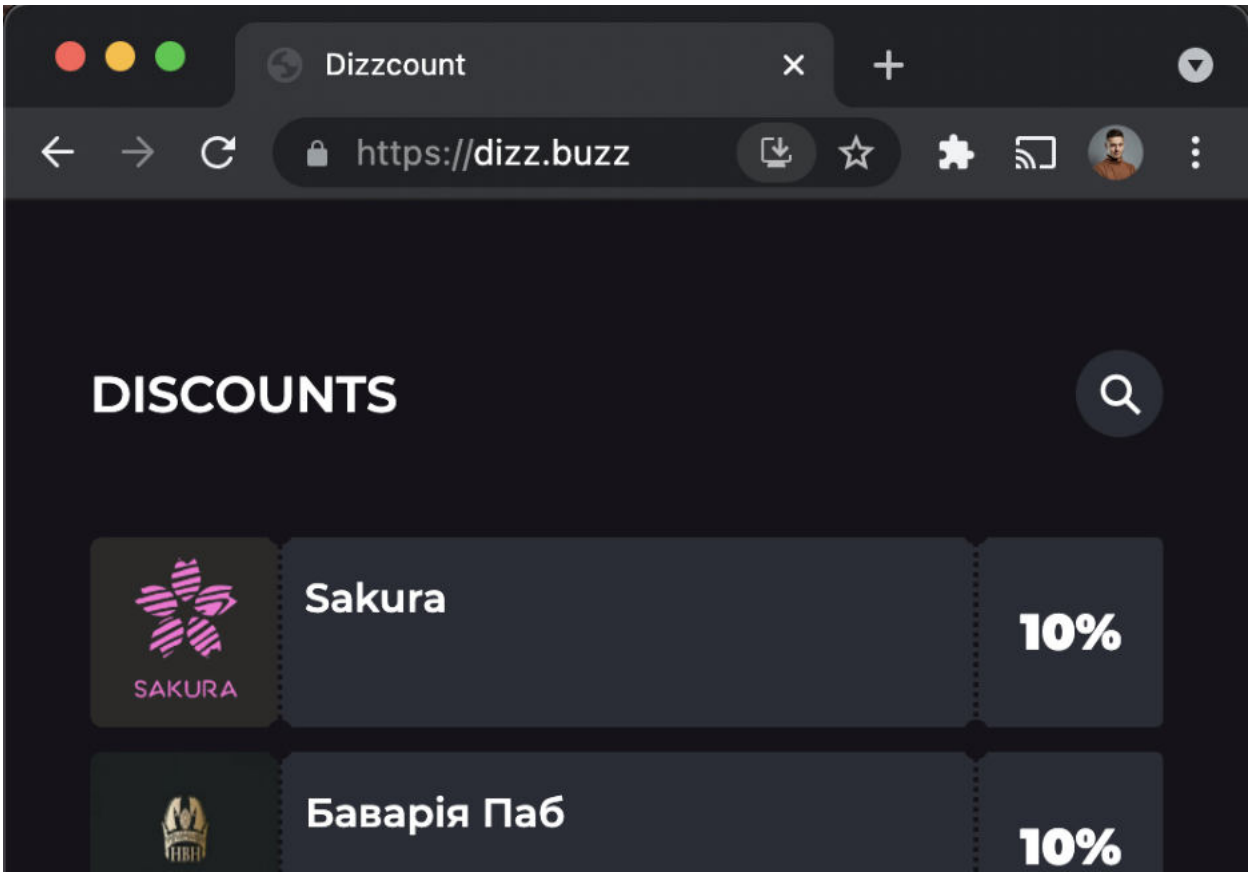


Рисунок 3.24 — Браузер Chrome на macOS пропонує встановити програму

Таким чином, при старті проєкту, Webpacker збирає і зберігає в теці public публічні JavaScript файли, які сумісні з технологією PWA. У свою чергу, manifest.json дає знати браузеру, що програма може бути встановлена на смартфон або навіть комп'ютер (рис. 3.24).

Одна з передумов доступності використання PWA — робота вебсайту через HTTP 2.0 (безпечний протокол HTTPS).

3.8 Деплой

За проєктом інфраструктури системи, використано контейнеризатор програм Docker. Встановлюємо його за допомогою саморозгортаючого пакету з офіційного вебсайту [45].

Перш ніж запустити Docker, зробимо текстовий файл «Dockerfile.rails» в якому визначимо ключові команди (див. Додаток М):

- FROM — вказує на базовий образ, з якого починаються вся процеси;
- RUN — центральні теки для запуску Docker;
- ENV — змінні середовища виконання;
- USER — користувач, від імені якого запускатиметься контейнер;
- WORKDIR — позначить теку для виконання команди з CMD;
- CMD — виконає конкретну команду всередині контейнера.

Підготуємо конфігурацію сервера Puma («config/puma.rb»):

```
max_threads_count = ENV.fetch("RAILS_MAX_THREADS") { 5 }
min_threads_count = ENV.fetch("RAILS_MIN_THREADS") { max_threads_count }
threads min_threads_count, max_threads_count
port        ENV.fetch("PORT") { 3000 }
environment ENV.fetch("RAILS_ENV") { "development" }
pidfile ENV.fetch("PIDFILE") { "tmp/pids/server.pid" }
plugin :tmp_restart
```

Те саме з конфігурацією для Sidekiq («config/initializers/sidekiq.rb»):

```
sidekiq_config = { url: ENV['JOB_WORKER_URL'] }

Sidekiq.configure_server do |config|
  config.redis = sidekiq_config
end
```

Додаємо Docker до списку довірених джерел, адже Rails має вбудовані системи захисту, які стандартно блокують доступ з невідомих джерел. В «config/environment/development.rb» прописуємо рядок «`config.hosts << "drkiq"`».

Підготуємо новий конфігураційний файл для Nginx:

```
Server {
```

```
        listen 8020;
        server_name example.org;
        location / {
            proxy_pass http://dizz:8010;
            proxy_set_header Host $host;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
    }
```

І будуємо новий образ для Nginx, створивши новий файл «Dockerfile.nginx» в кореневій теці проєкту:

```
FROM nginx:latest
COPY reverse-proxy.conf /etc/nginx/conf.d/reverse-proxy.conf

EXPOSE 8020

STOPSIGNAL SIGTERM
CMD ["nginx", "-g", "daemon off;"]
```

Готуємо Docker Compose файл, задля того аби мати можливість запускати кілька Docker образів одночасно і синхронізувати між машинами розробників (див. Додаток Н для вихідного коду docker-compose.yml).

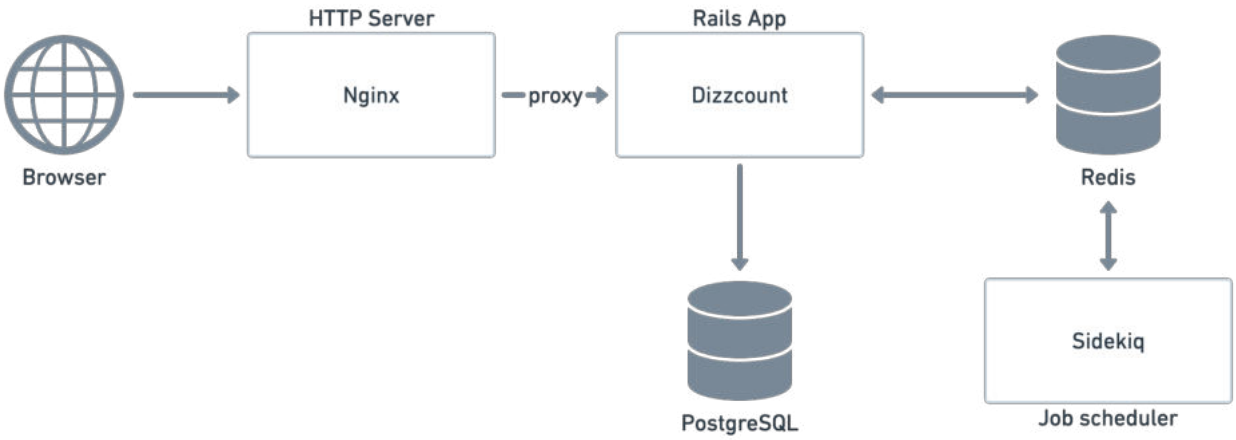


Рисунок 3.25 — Ключові складові системи для Docker Compose

Docker Compose в спрощеному вигляді (рис. 3.25) працює так:

- Postgres і Redis використовують томи Docker для управління стійкістю даної системи;
- Postgres, Redis та Dizzcount виставляють порти;

- Dizzcount і Sidekiq мають зв’язки з Postgres і Redis;
- Dizzcount і Sidekiq читають змінні середовища з .env;
- Sidekiq перезаписує команду для запуску Sidekiq замість Puma.

Створюємо томи для Postgres командою «`docker volume create --name dizzcount-postgres`», для Redis — «`docker volume create --name dizzcount-redis`» і запускаємо, bash команда «`docker-compose up`».

Тепер ініціюємо бази даних: «`docker--compose run dizzcount rake db:reset`», «`docker--compose run dizzcount rake db:migrate`» і знову запускаємо «`docker-compose up`».

Тепер програма готова до деплою на хмарні сервери і спільної роботи між командою кількох розробників.

3.9 Тестування

Для контролю за стилем написання коду програми, встановимо гем Rubocop. Він статично аналізує систему на відповідність коду загальноприйнятим нормам.

Для написання юніт та інтеграційних тестів, підключимо гем Rspec. Він дозволяє описувати очікувану поведінку програми (Behaviour Driven Development) і тестувати її. Приклад організації тестів для контролера сутності User описано в Додатку Б.

Найбільш важливим перед кожним релізом є ручне тестування. Інженер з забезпечення якості має вручну перевірити справність додатку у всіх сучасних браузерах на смартфонах та комп’ютерах.

Коли програму почнуть використовувати реальні користувачі, необхідно буде налаштувати процес регулярного збору відгуків. А впровадження змін, що базуються на таких відгуках — взяті за одним з головних пріоритетів.

Висновки до розділу 3

Виконано повний цикл розробки програмного забезпечення обраними технологіями, налаштовано відповідну інфраструктуру та забезпечено базовий контроль за якістю.

Описано користувацький інтерфейс і обробку операцій налаштування сторінки бізнесу, користувачів, використання віртуальних карток лояльності тощо – через даний інтерфейс.

РОЗДІЛ 4. БІЗНЕС-ПЛАН РОЗВИТКУ СИСТЕМИ

4.1 Резюме

Система для управління та використання корпоративних програм лояльності надасть одним компаніям майданчик для маркетингу шляхом розповсюдження комерційних пропозицій, а іншим — просту у впровадженні можливість розширення корпоративного соціального пакету.

Цільовою аудиторією програми є компанії малого і середнього бізнесів з кількістю працівників від 5 до 500.

На створення системи необхідно залучити від $3500 до $5000 інвестицій. Кошти розраховані на рекламу і залучення одного розробника, одного проєктного менеджера, одного тестувальника і одного DevOps інженера.

Мінімальна місячна виручка з підписок на доступ до програми очікується на рівні $200 починаючи з шостого місяця від дати запуску.

4.2 Бізнес модель

Застосунок має приносити гроші його власникам для утримання і подальшого розвитку. Для цього оберемо оптимальну бізнес модель — стратегію отримання сервісного збору та прибутку.

Перш ніж автоматизувати збір коштів, треба переконатися у правильному виборі моделі. Для початку, платежі стягуватимуться в особистому порядку. Власник програми вручну валідуватиме надходження коштів і контролюватиме доступ до програми.

Кожна компанія щомісяця платитиме фіксовану суму за доступ до програми для всіх своїх працівників, без обмежень в їх кількості.

Бізнес безкоштовно отримує доступ до програми для публікації своїх знижок і спеціальних пропозицій.

У подальших версіях застосунку, можливе розширення функціоналу для бізнесу, коли за додаткову плату бізнес матиме змогу бути в топі всього переліку бізнесів, або певної категорії бізнесу, надсилати персоналізовані сповіщення працівникам компаній тощо.

4.3 Маркетинг

4.3.1 Попит

За даними Google Трендів [46] по Україні програми лояльності, попри свою не нову історію, є популярними на відносно тому ж рівні, що і 10 років тому. Те саме джерело даних вказує, що українці все більше цікавляться темою соціальних пакетів (рис. 4.1).

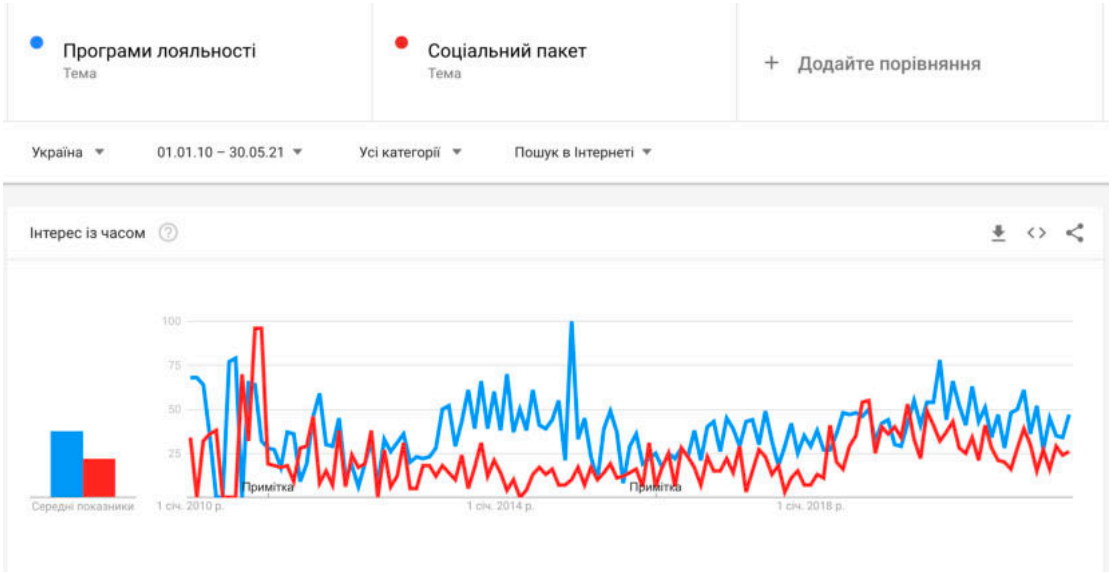


Рисунок 4.1 — Тренди Google за темами програм лояльності і соціального пакету

Додатково, дані Statcounter [47] свідчать про все більше поширення мобільних телефонів, смартфонів (рис. 4.2). Це може означати, що скоро

смартфони стануть популярніші за комп’ютери і світ має орієнтуватись спершу на портативні компактні пристрої.

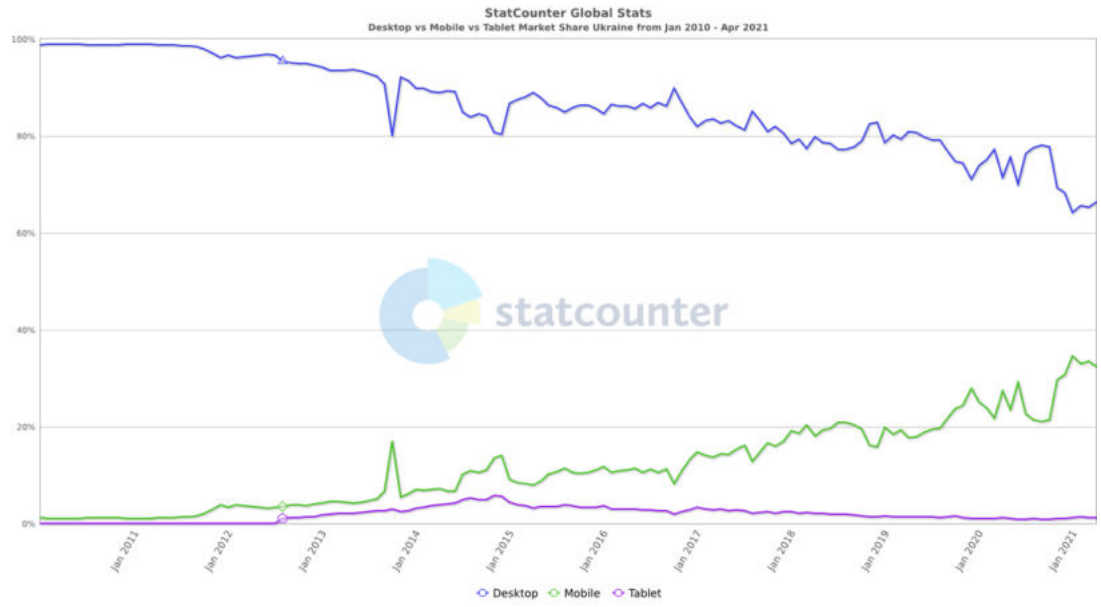


Рисунок 4.2 — Популярність комп’ютерів, телефонів і планшетів в Україні

Кількість мобільних додатків для програм лояльності на Android і iOS [48] та їх завантаження підкріплюють фактом тезу, що ця ніша має неабиякий попит серед населення України.

4.3.2 Конкуренти

IT Club Loyalty та IT Loyalty розглянути як аналоги в першому розділі даної дипломної роботи — прямі конкуренти розробленої системи. Додатки міцно закріпились на високих позиціях серед аудиторії IT-працівників. Проте, цим же вони відрізали себе з решти ринку, що надає можливості розробленій системі посісти це вільне місце.

Непрямими конкурентами системи на українському ринку є електронні рішення, які кожен бізнес встановлює самостійно. Це може ускладнити інтеграцію бізнесу з даною системою, оскільки менеджмент двох систем для програм лояльності вимагає більше операційних (часових) витрат від бізнесу.

4.3.3 План просування

Оскільки принцип розробленої програми дозволяє бізнесам приєднуватись безкоштовно, очікується що вони будуть найбільш зацікавленими до співпраці. Тому перші контакти планується встановити саме з власниками бізнесів, які можуть надати знижки і спеціальні пропозиції.

Короткостроковий план щодо компаній: залучити три організації до безкоштовного тестування системи протягом одного місяця. Зібрати фідбек і впровадити найнеобхідніші зміни.

Середньостроковий план: продати підписку на програму 10+ компаніям загальним об’ємом більше 400 співробітників протягом перших пів року.

Довгостроковий план: збирати аналітику, залучати інвестиції, динамічно розвивати систему в залежності від потреб ринку.

4.3.4 Економічна складова

Вартість послуг для компанії становитиме $0.5 на місяць за кожного працівника. Відповідно очікується приблизно $200 виручки щомісяця з 6 місяця після запуску, тоді окупність системи — близько 22 місяців.

4.3.5 Сценарії розвитку

- оптимістичний: 20 компаній на платформі через рік після запуску.
- реалістичний: 10 компаній користуються платформою рік після запуску даного проєкту.
- песимістичний: 1 компанія приєдналась за рік від дати запуску.

4.4 Кошторис

Обрахунок виробничих витрат наведено на таблиці №1. Обрахунок витрат на сторонні сервіси наведено на таблиці №2.

Таблиця 1 – Виробничі витрати

У доларах США

	Вартість
Розробка	$2600
Тестування	$550
Налаштування серверів	$320
Проєктний менеджмент	$430
СУМА	$3900

Таблиця 2 – Вартість сторонніх сервісів

У доларах США

	Вартість
MailJet (1 рік)	104.28
Amazon Web Services (1 рік Free Tier)	0
СУМА	104.28

4.4.1 Джерела фінансування

Перше джерело фінансування: IT-компанії з працівниками, що простоюють без роботи. Таких компаній і людей багато, тому є можливість укласти угоду з однією компанію про безоплатне надання послуг розробки в обмін на безкоштовну підписку сервісу, на об’єм прямо пропорційний вкладеному зі сторони компанії.

Друге джерело фінансування: зацікавлені приватні інвестори.

4.5 Нормативно-правові нюанси

Правова форма організації проєкту — може бути власністю фізичної-особи підприємця, будь-якої комерційної юридичної особи, або структурним підрозділом такої особи тощо.

4.6 Оцінка можливих ризиків

4.6.1 Технічні ризики

- зупинка інфраструктури – малоймовірна за коректного налаштування і постійної підтримки;
- баги в програмі – уникаються шляхом передбачуваного планування і релізів, написання юніт та інтеграційних тестів, а також ручним тестуванням. Не є суттєвими у випадку їх виникнення на ранніх стадіях використання, з малою кількістю залучених користувачів;
- несправність чи припинення підтримки технології Progressiwe Web Apps сучасними браузерами – важко визначити рівень ризику, розвиток технології наразі непередбачуваний. Треба готувати альтернативні рішення. Впливає лише на мобільну версію.

Технічні ризики не становлять суттєвої небезпеки для проєкту, принаймі на найближчі 12 місяців.

4.6.2 Нетехнічні ризики

- відсутність інвесторів;
- невдача в знаходженні зацікавлених компаній і / та бізнесів;
- неможливість виходу на ринок через поглиблення пандемії;
- поява нового конкурента чи потужнішого аналога.

Нетехнічні ризики є високоймовірними та суттєвими, що може призвести до закриття даного проєкту.

Висновки до розділу 4

Складено попередній план просування програмного забезпечення серед компаній, виставлено очікувані показники успішності. Основна мета – спробувати додаток у реальних ринкових відносинах, зібрати результати і впровадити зміни, необхідні для подальшого розвитку.

Визначено, що нетехнічні ризики є більш вагомими, ніж технічні. Отже, як і в будь-якому стартапі, аби випробувати концепцію системи, мають бути прикладені мінімальні зусилля для досягнення максимального результату за найкоротший можливий час.

ВИСНОВКИ

В рамках цієї роботи досліджено ринок програм лояльності в Україні і розроблено систему для управління і використання корпоративних програм лояльності зі знижками і спецпропозиціями.

У процесі розробки використано сучасну мову програмування Ruby і її фреймворк Ruby on Rails, надійні рекомендовані інструменти як от СУБД Postgres, сторонній сервіс відправки імейлів MailJet і зовсім нова технологія Progressive Web Apps, що ще не встигла набрати уваги серед широкої авдиторії. Засвоєно методики контейнеризації за допомогою Docker, для збільшення надійності і портативності інфраструктури проєкту.

Ця розробка може допомогти малим бізнесам вийти на новий рівень залучення клієнтів, а компаніям порадувати своїх працівників новими можливостями всередині команди. Програма дозволяє значно зекономити ресурси на впровадження подібних ідей самотужки.

Переваги системи над аналогами:

- безоплатний майданчик для реклами бізнесів;
- економне впровадження для компаній;
- доступ, що не обмежений локацією чи сферою роботи компанії.

Також спроєктовано бізнес-план подальшого розвитку застосунку. Пораховано витрати на розробку і утримання, виокремлено стратегію залучення ресурсів і визначено максимальний термін для окупності у 22 місяці. Ідентифіковано потенційні ризики і визначено стратегію уникнення чи зменшення їх ймовірності.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Програми лояльності [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Програми_лояльності (дата звернення: 07.10.2022).

2. The Loyalty Report. LAUNCH EDITION USA EXECUTIVE SUMMARY // Bond Brand Loyalty Inc. – 2020. – С. 4.

3. Starbucks® Rewards Terms of Use [Електронний ресурс] – Режим доступу до ресурсу: https://www.starbucks.com/rewards/terms (дата звернення: 10.10.2022).

4. Лошенюк І. Р., Гуменюк А. М., Чаплінський Ю. Б. Маркетингова цінова політика : навч. посібник. // Дакор, КНТ – 2008. – С. 101.

5. Вішліна Р.І. ЦІНОВІ ЗНИЖКИ ЯК МЕТОД СТИМУЛЮВАННЯ ЗБУТУ / Наукові записки Міжнародного гуманітарного університету. / Фенікс. – Одеса, 2013. – №18. – С. 148-150.

6. Програми лояльності та якість життя – ShiStrategies [Електронний ресурс] – Режим доступу до ресурсу: https://strategi.com.ua/prohramy-loial-nosti-sposib-polipshyty-iakist-zhyttia/ (дата звернення: 10.10.2022).

7. Світлана С. Що мотивує співробітників і які соціальні пакети їм пропонувати [Електронний ресурс] / Скородумова Світлана – Режим доступу до ресурсу: https://biz.nv.ua/ukr/experts/groshi-ne-vtrimayut-spivrobitnika-yaki-socpaketi-proponuvati-pracivnikam-novini-ukrajini-50118190.html (дата звернення: 23.10.2022).

8. ПРАВИЛА ПРОГРАМИ ЛОЯЛЬНОСТІ «WATSONS CLUB» [Електронний ресурс] – Режим доступу до ресурсу: https://www.watsons.ua/uk/official_rules (дата звернення: 01.11.2022).

9. ПРОГРАМА ЛОЯЛЬНОСТІ «WINETIME» [Електронний ресурс] – Режим доступу до ресурсу: https://winetime.ua/about/programa_loyalnosti/ (дата звернення: 01.11.2022).

10. Мобільний додаток «Сільпо» [Електронний ресурс] – Режим доступу до ресурсу: https://silpo.ua/about/additional/mobileapp (дата звернення: 01.11.2022).

11. Програма лояльності - Інтернет-магазин «Salad» [Електронний ресурс] – Режим доступу до ресурсу: https://salad.com.ua/programma-loyalnosti/ (дата звернення: 01.11.2022).

12. Джуровець А.І. СТАН РОЗВИТКУ МАЛОГО ПІДПРИЄМНИЦТВА, ПРОБЛЕМИ ЙОГО ФУНКЦІОНУВАННЯ В ЧЕРНІВЕЦЬКІЙ ОБЛАСТІ / Джуровець А.І., Катана А.В. / Науковий журнал «Молодий вчений». – 2017. – №12. – С. 607.

13. Discontimo for Android - APK Download [Електронний ресурс] – Режим доступу до ресурсу: https://apkpure.com/discontimo/ua.com.discontimo.discontimo (дата звернення: 17.11.2022).

14. Discontimo - Android Apps on Google Play [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=ua.com.discontimo.discontimo (дата звернення: 17.11.2022).

15. Discontimo Ivano-Frankivsk | Facebook – Facebook [Електронний ресурс] – Режим доступу до ресурсу: https://www.facebook.com/discontimo.if (дата звернення: 17.11.2022).

16. discontimo.com [Електронний ресурс] – Режим доступу до ресурсу: http://discontimo.com/ (дата звернення: 17.11.2022).

17. IT Club Loyalty - Львівський IT Кластер [Електронний ресурс] – Режим доступу до ресурсу: https://itcluster.lviv.ua/projects/it-club/ (дата звернення: 18.11.2022).

18. IT CLUB LOYALTY | ДЛЯ IT КОМПАНІЙ [Електронний ресурс] – Режим доступу до ресурсу: https://itclubloyalty.com/dlya-it-companiy (дата звернення: 18.11.2022).

19. IT Club Loyalty - Android Apps on Google Play [Електронний ресурс] – Режим доступу до ресурсу:

https://play.google.com/store/apps/details?id=com.devabit.itclub (дата звернення: 18.11.2022).

20. IT LOYALTY | Kharkiv IT Cluster Play [Електронний ресурс] – Режим доступу до ресурсу: https://it-kharkiv.com/projects/it-loyalty/(дата звернення: 18.11.2022).

21. Information technology - Object Management Group Unified Modeling Language (OMG UML), Infrastructure. // ISO/IEC 19505-1. – 2012. – С. 8.

22. Petersen K. The Waterfall Model in Large-Scale Development / K. Petersen, C. Wohlin, D. Baca // Product-Focused Software Process Improvement, LNBIP 32 / K. Petersen, C. Wohlin, D. Baca. – Berlin, 2009. – (Springer-Verlag). – С. 386–400.

23. Richard F. B. Modeling AUTOSAR systems with a UML/SysML profile. / Boldt Richard. – Somers, NY, 2009. – (IBM Corporation)

24. WICKED PROBLEMS IN PROJECT DEFINITION. // Proceedings of the International Group for Lean Construction 10th Annual Conference / – Brazil, 2002.

25. Катерина К. Agile-філософія як інструмент використання гнучких підходів в публічному управлінні. / Катерина Комарова / Науковий журнал Аспекти публічного управління. – 2020. – С 68–71.

26. Vázquez-Ingelmo A. C4 model in a Software Engineering subject to ease the comprehension of UML and the software development process / A. Vázquez-Ingelmo, A. García-Holgado, F. J. García-Peñalvo. – Porto: IEEE, 2020. – (2020 IEEE Global Engineering Education Conference (EDUCON)). – С. 919-924.

27. The C4 model for visualising software architecture [Електронний ресурс] – Режим доступу до ресурсу: https://c4model.com/ (дата звернення: 20.03.2023).

28. 100+ Найпоширеніших запитань API Testing, REST, SOAP, Web-services [Електронний ресурс] – Режим доступу до ресурсу: https://www.quality-assurance-group.com/100-najposhyrenishyh-zapytan-z-testuvannya-api-rest-ta-soap-stylej-veb-sluzhb-na-spivbesidah-ta-tehnichnyh-interv-yu/ (дата звернення: 26.03.2023).

29. Progressive web apps (PWAs) | MDN [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (дата звернення: 26.03.2023).

30. Sidekiq [Електронний ресурс] – Режим доступу до ресурсу: https://sidekiq.org/ (дата звернення: 28.03.2023).

31. Redis [Електронний ресурс] – Режим доступу до ресурсу: https://redis.io/ (дата звернення: 28.03.2023).

32. Stack Overflow Developer Survey 2020 [Електронний ресурс] – Режим доступу до ресурсу: https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages (дата звернення: 28.03.2023).

33. Тези доповідей II Міжнародної науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення – 2017». // Житомирський державний технологічний університет. – 2017. – С. 66–67.

34. 8 Startups That Owe Their Success to Ruby on Rails – Nopio [Електронний ресурс] – Режим доступу до ресурсу: https://www.nopio.com/blog/ruby-on-rails-startups/ (дата звернення: 28.03.2023).

35. Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/ (дата звернення: 28.03.2023).

36. DB-Engines Ranking [Електронний ресурс] – Режим доступу до ресурсу: https://db-engines.com/en/ranking (дата звернення: 28.03.2023).

37. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/ (дата звернення: 28.03.2023).

38. Category: Web Servers - The Ruby Toolbox [Електронний ресурс] – Режим доступу до ресурсу: https://www.ruby-toolbox.com/categories/web_servers (дата звернення: 28.03.2023).

39. puma/puma: A Ruby Web Server Built For Concurrency [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/puma/puma (дата звернення: 28.03.2023).

40. RhodeCode | Version Control Systems Popularity in 2016 Concurrency [Електронний ресурс] – Режим доступу до ресурсу: https://rhodecode.com/insights/version-control-systems-2016 (дата звернення: 28.03.2023).

41. About GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/about (дата звернення: 29.03.2023).

42. Top 10 Best Container Software in 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/container-software/ (дата звернення: 29.03.2023).

43. Getting Started with Rails — Ruby on Rails Guides [Електронний ресурс] – Режим доступу до ресурсу: https://guides.rubyonrails.org/getting_started.html (дата звернення: 29.03.2023).

44. The Asset Pipeline — Ruby on Rails Guides [Електронний ресурс] – Режим доступу до ресурсу: https://guides.rubyonrails.org/asset_pipeline.html (дата звернення: 29.03.2023).

45. Docker Desktop for Mac and Windows | Docker [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/products/docker-desktop (дата звернення: 29.03.2023).

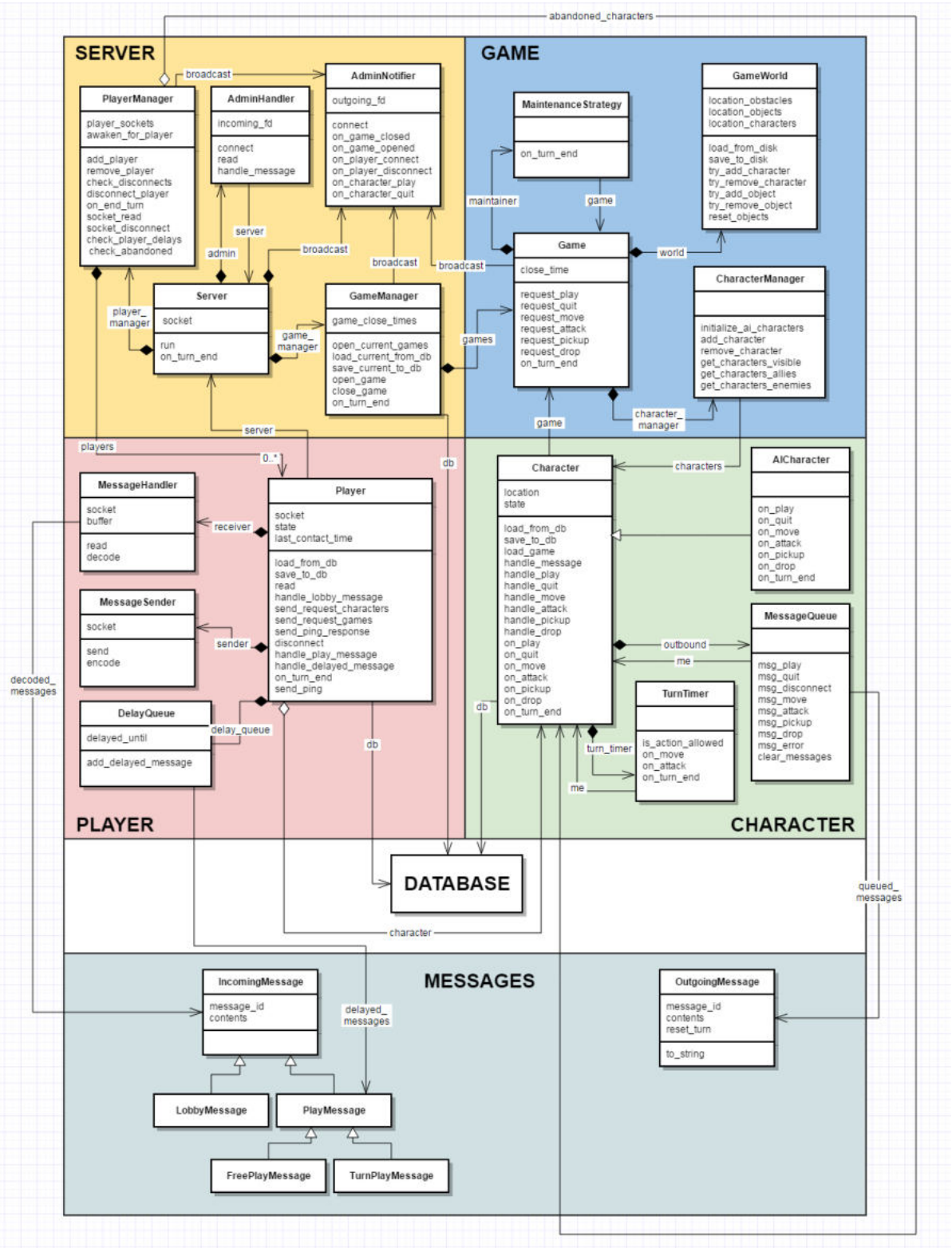
46. Програми лояльності, Соціальний пакет – Google Тренди [Електронний ресурс] – Режим доступу до ресурсу: https://trends.google.com/trends/explore?date=2010-01-01%202021-05-30&geo=UA&q=%2Fm%2F04qf0,%2Fm%2F05j7x6 (дата звернення: 15.04.2023).

47. Desktop vs Mobile vs Tablet Market Share Ukraine [Електронний ресурс] – Режим доступу до ресурсу: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/ukraine/#monthly-201001-202104 (дата звернення: 15.04.2023).

48. Discontimo Apps | App Store [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/developer/discontimo/id1028852548#see-all/i-phonei-pad-apps (дата звернення: 17.04.2023).

ДОДАТКИ

Додаток А. Приклад UML-діаграми для опису програми



Додаток Б. Діаграма «сутність-зв'язок» бази даних

Додаток В. Код контролера app/controllers/admin/organizations_controller.rb

```
class Admin::OrganizationsController < Admin::BaseController
  before_action :verify_class, only: %i[index organization_invitation
invite_organization]
```

```
  before_action :verify_record, except: %i[index organization_invitation
invite_organization]
  def index
    @pagy, @records = pagy(record_class.all, items: 5, size: [0, 0, 0, 0])
  end
  def organization_invitation
    @record = record_class.new
  end
  def invite_organization
    if (res = Organizations::Operations::InviteOrganization.call(record_params:
record_params)).success?
      flash[:success] = MessageHelper.action(record_class.model_name.human,
'invite organization')
      redirect_to :admin_organizations
    else
      flash[:alert] = humanize_errors(res.errors)
      redirect_to :organization_invitation_admin_organizations
    end
  end

  def edit; end

  def update
    if (res = Organizations::Operations::Update.call(record: record,
record_params: record_params)).success?
      flash[:success] = MessageHelper.updated(record_class.model_name.human)
    else
      flash[:alert] = humanize_errors(res.errors)
    end
    redirect_to :edit_admin_organization
  end

  private

  def record_params
    params.require(:organization).permit!
  end

  def record_class
    Organization
  end
end
```

Додаток Г. Код форми app/views/admin/organizations/edit.html.slim

```
.row
  .col-12
    = link_to admin_organizations_path, class: 'organization__item' do
      = image_pack_tag 'prev.png', class: 'arrow_icon'
      span.organization__item_name = 'Back'
.row
  .col-12
    = form_for @record, url: admin_organization_path, method: :put do |f|
      .organization__invitation
        .organization__invitation_content
          .organization__invitation_avatar
            - if @record.logo.present?
              = image_tag @record.logo, id: 'preview', class:
'organization__invitation_img'
            - else
              = image_pack_tag 'no_avatar.png', id: 'preview', class:
'organization__invitation_img'
            .organization__invitation_upload
            label.organization__invitation_btn_upload for="organization_logo"
upload avatar
            = f.file_field :logo, class: 'hidden-image-uploader-js
organization__invitation_input_upload'
          .organization__invitation_inputs
            .organization__invitation_input_container
              = f.label :name, class: "organization__invitation_label"
              = f.text_field :name, class: "form_input"
            .organization__invitation_inputs
              label.container.organization__invitation_check_box_label
                | Active
                = f.check_box :active, class: 'm-1 checkbox'
                span.checkmark
        .organization__invitation_btns_container
          = link_to 'manage users', admin_organization_users_path(@record), class:
"submit_btn organization__invitation_manage_users_btn"
          = f.submit 'save', class: "submit_btn"
```

Додаток Ґ. Код форми app/views/admin/servicecompanies/edit.html.slim

```
.row
  .col-12
    = link_to admin_service_companies_path, class: 'organization__item' do
      = image_pack_tag 'prev.png', class: 'arrow_icon'
      span.organization__item_name = 'Back'
.row
  .col-12
    = form_for @record, url: admin_service_company_path, method: :put do |f|
      .organization__invitation
        .organization__invitation_content
          .organization__invitation_avatar
            - if @record.image.present?
              = image_tag @record.image, id: 'preview', class:
'organization__invitation_img'
            - else
              = image_pack_tag 'no_avatar.png', id: 'preview', class:
'organization__invitation_img'
            .organization__invitation_upload
            label.organization__invitation_btn_upload for="service_company_image"
upload avatar
            = f.file_field :image, class: 'organization__invitation_input_upload
hidden-image-uploader-js'
          .organization__invitation_inputs
            .organization__invitation_input_container
              = f.label :name, class: 'organization__invitation_label'
              = f.text_field :name, class: 'form_input'
            .organization__invitation_inputs
              label.container.organization__invitation_check_box_label
                | Active
                = f.check_box :active, class: 'm-1 checkbox'
                span.checkmark
        div style='align-items: baseline;'
          = f.fields_for :contact_infos, @record.contact_infos do |c_f|
            .sc_info_container
              = c_f.label :city, class: 'organization__invitation_label'
              = c_f.select :city, cities_list(:ua), { include_blank: true }, {
class: 'select2 wide_input' }
            .sc_info_container
```

```
              = c_f.label :phone_number, 'Phone', class:
'organization__invitation_label '
              = c_f.text_field :phone_number, class: 'form_input wide_input'
        div style='align-items: baseline;'
          .sc_info_container
            = f.label :website, class: 'organization__invitation_label'
            = f.text_field :website, class: 'form_input wide_input'
        .organization__invitation_btns_container
          = f.submit 'save', class: 'btn-block submit_btn'
        .organization__invitation_btns_container
          = link_to 'manage users', admin_service_company_users_path(@record),
class: 'btn-block text-center submit_btn
organization__invitation_manage_users_btn'
        .organization__invitation_btns_container
          = link_to 'special offers',
admin_service_company_special_offers_path(@record), class: 'btn-block text-center
submit_btn organization__invitation_manage_users_btn'
```

Додаток Д. Код форми app/views/admin/users/edit.html.slim

```
- if @record.organization.present?
  = link_to admin_organization_users_path(@record.organization), class:
'organization__item' do
    = image_pack_tag 'prev.png', class: 'arrow_icon'
    div
      span.organization__item_name = 'Users of ' + @record.organization.name
      = image_tag @record.organization.logo, class: 'organization__item_image' if
@record.organization.logo.present?
- elsif @record.service_company.present?
  = link_to admin_service_company_users_path(@record.service_company), class:
'organization__item' do
    = image_pack_tag 'prev.png', class: 'arrow_icon'
      Div
        span.organization__item_name = 'Users of ' + @record.service_company.name
        = image_tag @record.service_company.image, class:
'organization__item_image'
= form_for @record, url: admin_user_path, method: :patch do |f|
  .organization__user_card
    .organization__user_card_content
      Div
        - if @record.organization.present?
```

```
          label.container.organization__invitation_check_box_label style='display:
inline-block;'
            | Organization manager
            = f.check_box :role, { class: 'm-1 checkbox', checked:
@record.organization_manager? }, 'organization_manager', 'organization_member'
            span.checkmark
        - else
          = f.hidden_field :role, value: 'service_company_coordinator'
        - if @record.code.present?
          span.organization__user_card_code.loyalty_card__code
            .card-code = number_with_delimiter(@record.code, delimiter: ' ')
      .organization___user_card_avatar
        - if @record.avatar.attachment.present?
          = image_tag @record.avatar.variant(resize_to_fill: [100, 100]), class:
'organization__user_card_img'
          label.organization__user_card_btn_upload for="user_avatar" upload avatar
          = f.file_field :avatar, class: 'organization__invitation_input_upload
hidden-image-uploader-js'
        - else
          = image_pack_tag 'no_avatar.png', id: 'preview', class:
'organization__user_card_img'
          label.organization__user_card_btn_upload for="user_avatar" upload avatar
          = f.file_field :avatar, class: 'organization__invitation_input_upload
hidden-image-uploader-js'
    = f.text_field :email, class: 'organization__user_card_input'
  .row.m-2
    .col-8.delete_button
      = link_to admin_user_path(@record), method: :delete, class: 'btn btn-light'
do
        i.fas.fa-trash-alt.red
    .col-4
      = f.submit 'save', class: 'submit_btn'
```

Додаток Ж. Код моделі app/models/special_offer.rb

```
class SpecialOffer < ApplicationRecord
  include Filterable

  belongs_to :service_company
  has_one_attached :image
```

```
  scope :current, (lambda do
    where(SpecialOffer.arel_table[:start_at].lteq(Time.current).and(
      SpecialOffer.arel_table[:end_at].gteq(Time.current.beginning_of_day)
    ).or(
      SpecialOffer.arel_table[:start_at].eq(nil)
    ))
  end)
  scope :upcoming, -> { where(SpecialOffer.arel_table[:start_at].gt(Time.current))
}
  scope :by_service_company, ->(service_company) { where(service_company:
service_company) }
end
```

Додаток З. Код контролера app/controllers/sc/special_offers_controller.rb

```
class SpecialOffersController < BaseController
  def index
    @special_offers = record_class.filter_collection(by_service_company:
params[:service_company_id]).includes(:service_company)
  end
  def show; end

  private

  def record_class
    SpecialOffer
  end
end
```

Додаток И. Код форми app/views/sc/special_offers/edit.html.slim

```
.row.form-group
  .col-4
    - if f.object.image.attachment.present?
      = image_tag f.object.image.variant(resize_to_fit: [100, 100]), id:
'preview', class: 'img-thumbnail p-0 no-border'
    - elsif f.object.service_company&.image.present?
      = image_tag f.object.service_company.image.variant(resize_to_fit: [100,
100]), id: 'preview', class: 'img-thumbnail p-0 no-border'
    - else
```

```
      = image_pack_tag 'no_avatar.png', id: 'preview', class: 'img-thumbnail p-0
no-border no-background'
    button.avatar-uploader.text-light.rounded-bottom upload avatar
    = f.file_field :image, class: 'hidden-avatar-uploader hidden-image-uploader-
js'
  .col-8
    = f.label :title, class: 'organization__invitation_label'
    = f.text_field :title, rows: 2, class: 'w-100 form_input'
.row
  .col-4
    = f.label :description, class: 'organization__invitation_label'
  .col-8
    = f.text_area :description, rows: 3, class: 'w-100 form_input', style:
'height: 60px;'
.row
  .col-4
    = f.label :time_notice, class: 'organization__invitation_label'
  .col-8
    = f.text_field :time_notice, class: 'w-100 form_input'
.row
  .col-4
    = f.label :start_at, class: 'organization__invitation_label'
  .col-8
    = f.text_field :start_at, value: f.object&.start_at&.strftime("%d.%m.%Y"),
class: 'w-100 form_input datepicker'
.row
  .col-4
    = f.label :end_at, class: 'organization__invitation_label'
  .col-8
    = f.text_field :end_at, value: f.object&.end_at&.strftime("%d.%m.%Y"), class:
'w-100 form_input datepicker'
```

Додаток І. Код мобільного меню app/views/layouts/o/_footer.html.slim

```
div.tabs_navigation
  = link_to service_companies_path do
    svg width="25" height="25" viewBox="0 0 25 25"
xmlns="http://www.w3.org/2000/svg" fill="#{current_page?(service_companies_path) ?
"#FFF" : "#292D36" }"
  = link_to special_offers_path do
```

```
    svg width="25" height="24" viewBox="0 0 25 24"
xmlns="http://www.w3.org/2000/svg" fill="#{current_page?(special_offers_path) ?
"#FFF" : "#292D36"}"
  = link_to pwa_o_profile_path do
    svg width="25" height="25" viewBox="0 0 25 25"
xmlns="http://www.w3.org/2000/svg" fill="#{current_page?(profile_path) ? "#FFF" :
"#292D36"}"
```

Додаток Ї. Код переліку бізнесів app/views/o/servicecompanies/index.html.slim

```
= link_to service_company_path(service_company) do
  .service_company_item
    - if service_company.liked?(current_user)
      svg.service_company_liked width="18" height="16" viewBox="0 0 18 16"
fill="none" xmlns="http://www.w3.org/2000/svg"
        path d="M17.0273 3.53906C16.7655 2.93288 …" fill="#EB5757"
    div
      - if service_company.image.present?
        = image_tag service_company.image, class: 'service_company_img'
      - else
        = image_pack_tag 'image.jpg', class: 'service_company_img'
    .service_company_item_desc_container
      .service_company_item_separate
      span.service_company_item_name = service_company.name
      span.service_company_item_city = service_company.decorate.dc_cities
    .service_company_item_percent_container
      spam.service_company_item_percent =
service_company.discount_percent.to_i.to_s + '%'
```

Додаток Й. Код сторінки бізнесу app/views/o/servicecompanies/show.html.slim

```
    span.title
      = @record.name
  = link_to toggle_like_service_company_path(@record) do
    - if @record.liked?(current_user)
      svg width="29" height="26" viewBox="0 0 18 16" fill="none"
xmlns="http://www.w3.org/2000/svg"
```

```
        path d="M17.0273 3.53906C16.7655 …" fill="#EB5757"
    - else
      i.fas.fa-heart-broken.fa.red
= render 'users/user_card', record: current_user
.discout_ticket_container
  .discout_ticket_container_img
    - if @record.image.present?
      = image_tag @record.image.variant(resize_to_fit: [400, 400]), class:
'discout_ticket_img'
    - else
      = image_pack_tag "no_avatar.png", class: 'discout_ticket_img'
  .discout_ticket_percent
    .discout_ticket_separate
    span.discout_ticket_percent_count = @record.discount_percent.to_i.to_s + '%'
= render partial: 'service_companies/info', locals: { record: @record }
- if @record.special_offers.present?
  .row.mt-2
    .title style='margin-bottom: 15px;'
      | special offers
= render @record.special_offers.current
- if @record.special_offers.upcoming.present?
  .row.mt-2
    .title soon
= render @record.special_offers.upcoming
```

Додаток К. Код картки працівника app/views/o/users/_card.html.slim

```
.loyalty_card_container
  .loyalty_card__content
    .loyalty_card__title_containr
      span.loyalty_card__title record.organization.title
    span.loyalty_card__code.verified
      - if record.code.present?
        div.card-code = number_with_delimiter(record.code, delimiter: ' ')
    span.loyalty_card__code_email.card-email = record.email
  .loyalty_card__content.w-90px style='align-items: flex-end;'
    - if record.avatar.attachment.present?
      = image_tag record.avatar.variant(resize_to_fill: [250, 250]), class:
'loyalty_card__content_img w-100'
    - else
      = image_pack_tag 'no_avatar.png', class: 'special_offer_img'
```

```
    - if record.organization&.logo&.attachment.present?
      = image_tag record.organization.logo.variant(resize_to_fill: [75, 75]),
class: 'loyalty_card__content_img_small_org'
```

Додаток Л. Код вигляду форми перевірки ідентифікатора працівника app/views/sc/users/validate.html.slim

```
.loyalty_card_container
  .loyalty_card__content
    .loyalty_card__title_containr
      span.loyalty_card__title Dizzcount
      span.loyalty_card__subtitle Loyalty Card
    span.loyalty_card__code.verified
      - if @user.code.present?
        = @user.code
    span.loyalty_card__code_email = @user.email
  .loyalty_card__content.w-90px
    - if @user.avatar.attachment.present?
      = image_tag @user.avatar.variant(resize_to_fill: [250, 250]), class:
'loyalty_card__content_img w-100'
    - else
      .loyalty_card__content_img.w-100
        span.loyalty_card__not_img Not Found
    - if @user.organization&.logo&.attachment.present?
      = image_tag @user.organization.logo.variant(resize_to_limit: [75, 75]),
class: 'loyalty_card__content_img_small_org'
    - else
      svg width="51" height="24" viewBox="0 0 51 24"
        path fill-rule="evenodd" clip-rule="evenodd" d="M9.796 0.158387C7.31032 …"
fill="white"

= link_to 'Close', user_verification_ service_companies_path, class:
'loyalty_card__close_btn'
```

Додаток М. Код текстового файлу Dockerfile.rails

```
FROM ruby:2.6.3 AS rails-toolbox

ARG USER_ID
```

```
ARG GROUP_ID

RUN addgroup --gid $GROUP_ID user
RUN adduser --disabled-password --gecos '' --uid $USER_ID --gid $GROUP_ID user

ENV INSTALL_PATH /opt/app
RUN mkdir -p $INSTALL_PATH

RUN gem install rails bundler
RUN chown -R user:user $INSTALL_PATH
WORKDIR $INSTALL_PATH

USER $USER_ID
CMD ["/bin/sh"]
```

Додаток Н. Код конфігураційного файлу docker-compose.yml

```
version: "3.7"
 services:
 postgres:
     image: postgres:13.2
     environment:
       POSTGRES_USER: dizzcount
       POSTGRES_PASSWORD: dizz_buzz
     ports:
       - '5432:5432'
     volumes:
       - dizzcount-postgres:/var/lib/postgresql/data
 redis:
     image: redis:5.0.7
     ports:
       - '6379:6379'
     volumes:
       - dizzcount-redis:/var/lib/redis/data
 drkiq:
     build:
       context: .
       args:
         USER_ID: "${USER_ID:-1000}"
         GROUP_ID: "${GROUP_ID:-1000}"
     volumes:
```

```
      - ./dizzcount:/opt/app
    links:
      - postgres
      - redis
    ports:
      - '8010:8010'
    env_file:
      - .env
 sidekiq:
    build:
      context: .
      args:
        USER_ID: "${USER_ID:-1000}"
        GROUP_ID: "${GROUP_ID:-1000}"
    command: bundle exec sidekiq
    links:
      - postgres
      - redis
    env_file:
      - .env
 nginx:
    build:
      context: .
      dockerfile: ./Dockerfile.nginx
    links:
      - dizzcount
    ports:
      - '8020:8020'
 volumes:
   dizzcount-postgres:
   dizzcount-redis:
```

Додаток О. Код юніт тестів для контролера User

spec/controllers/admin/users_controller_spec.rb

```
require 'rails_helper'

describe Admin::UsersController, type: :controller do
  describe '#index' do
    subject { get :index, params: params }
```

```
    login_user(:admin)
    before do
      subject
    end

    context 'organization users' do
      let!(:organisation) { create(:organization) }
      let!(:params) { { organisation_id: organisation.id } }

      it 'should show all users' do
        expect(response).to have_http_status(200)
      end
    end

    context 'service company users' do
      let!(:service_company) { create(:service_company) }
      let!(:params) { { organisation_id: service_company.id } }

      it 'should show all users' do
        expect(response).to have_http_status(200)
      end
    end
  end

  describe '#create' do
    subject { post :create, params: params }
    login_user(:admin)

    context 'with valid params' do
      let!(:success_message) { 'User successfully invited' }
      context 'organization' do
        let!(:organization) { create(:organization) }
        let!(:params) do
          { user: { role: 'organization_manager',
                    email: 'new_test@mail.com' }, organization_id: organization.id
}
        end

        it 'should create user' do
          expect { subject }.to change { User.count }.from(1).to(2)
          expect(response).to have_http_status(302)
        end
```

```
        it 'should redirect to admin_user_path and show flash' do
          expect(subject).to redirect_to admin_user_path(User.last.id)
          expect(response.request.flash['success']).to eq success_message
          expect(response.request.flash['alert']).to eq nil
        end
      end

      context 'service company' do
        let!(:service_company) { create(:service_company) }
        let!(:params) do
          { user: { role: 'service_company_coordinator',
                    email: 'new_test@mail.com' }, service_company_id:
service_company.id }
        end

        it 'should create user' do
          expect { subject }.to change { User.count }.from(1).to(2)
          expect(response).to have_http_status(302)
        end

        it 'should redirect to admin_user_path and show flash' do
          expect(subject).to redirect_to admin_user_path(User.last.id)
          expect(response.request.flash['success']).to eq success_message
          expect(response.request.flash['alert']).to eq nil
        end
      end
    end

    context 'with invalid params' do
      let!(:error) { "Email: is invalid; can't be blank" }

      context 'organization' do
        let!(:organization) { create(:organization) }
        let!(:params) do
          { user: { role: 'organization_manager',
                    email: '' }, organization_id: organization.id }
        end

        it 'should not create user' do
          expect { subject }.not_to change { User.count }.from(1)
          expect(flash[:alert]).to eq error
```

```
      end

      it 'should redirect back to create form and show flash' do
        expect(subject).to redirect_to new_admin_organization_user_url
        expect(response.request.flash['success']).to eq nil
        expect(response.request.flash['alert']).to eq error
      end
    end

    context 'service company' do
      let!(:service_company) { create(:service_company) }
      let!(:params) do
        { user: { role: 'service_company_coordinator',
                  email: '' }, service_company_id: service_company.id }
      end

      it 'should not create user' do
        expect { subject }.not_to change { User.count }.from(1)
        expect(flash[:alert]).to eq error
      end

      it 'should redirect back to create form and show flash' do
        expect(subject).to redirect_to new_admin_service_company_user_url
        expect(response.request.flash['success']).to eq nil
        expect(response.request.flash['alert']).to eq error
      end
    end
  end
end
describe '#update' do
  subject { patch :update, params: params }
  login_user(:admin)
  context 'with valid params' do
    let!(:success_message) { 'User successfully updated' }

    context 'organization' do
      let!(:organization_manager) { create(:organization_manager) }
      let!(:params) do
        { user: { role: 'organization_member',
                  email: 'new_test@mail.com' }, id: organization_manager.id }
      end
```

```
        it 'should update user' do
          subject
          expect(organization_manager.reload.email).to eq params[:user][:email]
          expect(organization_manager.reload.role).to eq params[:user][:role]
        end

        it 'should redirect to edit_admin_user_path and show flash' do
          subject
          expect(subject).to redirect_to
edit_admin_user_path(organization_manager.id)
          expect(response.request.flash['success']).to eq success_message
          expect(response.request.flash['alert']).to eq nil
        end
      end

      context 'service company' do
        let!(:service_company_coordinator) { create(:service_company_coordinator)
}
        let!(:params) do
          { user: { email: 'new_test@mail.com', role:
'service_company_coordinator' }, id: service_company_coordinator.id }
        end

        it 'should update user' do
          subject
          expect(service_company_coordinator.reload.email).to eq
params[:user][:email]
        end
        it 'should redirect to edit_admin_user_path and show flash' do
          subject
          expect(subject).to redirect_to
edit_admin_user_path(service_company_coordinator.id)
          expect(response.request.flash['success']).to eq success_message
          expect(response.request.flash['alert']).to eq nil
        end
      end
    end

    context 'with invalid params' do
      let!(:error) { "Email: can't be blank" }
      context 'organization' do
        let!(:organization_manager) { create(:organization_manager) }
```

```
        let!(:params) do
          { user: { role: 'organization_member',
                    email: '' }, id: organization_manager.id }
        end

        it 'should redirect back to update form and show flash' do
          subject
          expect(subject).to redirect_to edit_admin_user_path
organization_manager.id
          expect(response.request.flash['success']).to eq nil
          expect(response.request.flash['alert']).to eq error
        end
      end

      context 'service company' do
        let!(:service_company_coordinator) { create(:service_company_coordinator)
}
        let!(:params) do
          { user: { role: 'service_company_coordinator', email: '' }, id:
service_company_coordinator.id }
        end

        it 'should redirect back to update form and show flash' do
          subject
          expect(subject).to redirect_to edit_admin_user_path
service_company_coordinator.id
          expect(response.request.flash['success']).to eq nil
          expect(response.request.flash['alert']).to eq error
        end
      end
    end
  end

  describe '#destroy' do
    subject { delete :destroy, params: params }
    login_user(:admin)
    let!(:success_message) { 'User successfully deleted' }

    context 'with valid params' do
      let!(:organization_manager) { create(:organization_manager) }
      let!(:params) { { id: organization_manager.id } }
```

```
      it 'should delete a user from db' do
        subject
        expect(response).to have_http_status(302)
        expect(response.request.flash['success']).to eq success_message
        expect(response.request.flash['alert']).to eq nil
        expect(User.find_by(id: params[:id])).to eq nil
      end
    end
  end
end
```